



Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich



Institut für  
Technische Informatik und  
Kommunikationsnetze

# **VSHMI Experimentation System**

**Annual Report of the SNSF project no. 200021 130224/1**

## **Vision-Supported Speech-Based Human Machine Interface**

Tofigh Naghibi

Speech Processing Group

TIK, ETH Zurich

March 2011

# 1 Abstract

The following progress report on the Vision-Supported Speech-Based Human Machine Interaction (VSHMI) project provides the summary of the activity to date on the project and the remaining steps needed to be accomplished. This document mainly talks about the first phase of the project, its accomplishments and technical issues and only briefly gives an overview of the VSHMI and remaining work to be done.

## 2 Project Overview

The VSHMI project was proposed to improve the human machine interaction (HMI) algorithms. Current audio based human-computer interaction algorithms are not robust under realistic conditions. In a real environment, audio-based systems are highly error prone because of the high background noise, human voice interference, speech signal discontinuity, etc. To overcome these difficulties, the use of both audio and video modality was proposed in this project. Multi modal architectures have been previously studied for meeting room and cocktail party scenarios and there are sufficiently large databases available for further investigation such as [3]. However, unlike the meeting room scenario, there is no multi-modal, multi-channel database available for dialog-based interface applications such as machines providing a publicly accessible service (like ticket machines). In this kind of applications, dialog module plays the central role. It continuously considers all information from the various system components and makes the optimal decision on the next action to be executed. In contrast to meeting room scenario, it requires compact sensors, namely more restriction on sensor positions, and real-time processing. Therefore, algorithms for such a task have to cope with more challenges than meeting room scenarios. One of the objectives of the VSHMI project is to provide the suitable database for audio-visual multi-channel HMIs. This database will be used in the next step to develop the new algorithms for HMIs that can handle the aforementioned challenges.

Schedules and timing details of the project can be found in the VSHMI description. However, in summary, VSHMI project can be broken down into three phases (not necessarily with the same workload):

1. Build up an appropriate hardware and software infrastructure to set up test scenarios.
2. Record and preparation of a suitable multi-channel multi-modal database for HMI applications.
3. Develop and improve the HMI algorithms based on the new database.

This report provides results from the first phase.

## 3 Infrastructure Implementation

For the development and test of the new algorithms we need appropriate audio-visual data from typical user interactions. To this end, we built up an experimentation system that can be used to set up test scenarios, i.e. applications like games, an information kiosk etc. For the initial system the currently available audio and video components (speech recognizer, face tracker etc.) have been used. However, the system is flexible enough to easily replace these components by improved ones. The interactions of the trial users with the system and all relevant information of the system and its components is recorded. The amount and the precision of the recorded data is sufficient to rerun the trial experiments with the recorded audio and video inputs and to reproduce the original results (reproducibility). Later, it will be possible to apply the improved algorithms for exactly the same input data and thus the improvement can be measured.

## 4 System Architecture

In order to provide the required flexibility, the architecture of the experimentation system is as follows: There are three main parts, the video server, the audio server and the dialog center as shown in Figure 1. The video and audio

servers contain both hardware, i.e. cameras and microphone array, and software components and provide functions such as user detection/tracking and identification (by means of face and/or voice features), speech recognition and utterance detection. The dialog center is the heart of the system and is responsible for controlling the audio and video servers, fusing their information and making decision for the next dialog steps. A set of audio and video server commands is needed to control the servers, to communicate with them and collect their information.

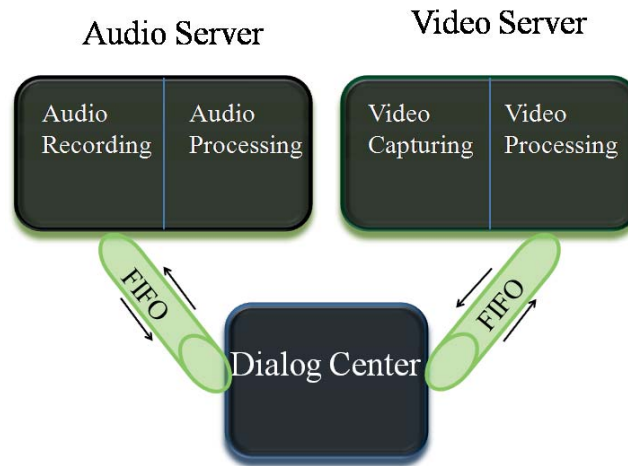


Figure 1: System Architecture: three main components: Audio and video servers communicate with dialog center via FIFOs

This architecture allows us to use different programming languages for the servers and the dialog center. Furthermore, the three system components can be easily run on different computers. In the currently implemented system, the dialog center communicates via FIFOs with the servers. More precisely, the dialog center writes commands to a first FIFO, the server fetches them, starts or performs the required action and writes a confirmation (and possibly a result) to the second FIFO.

As depicted in Figure 1, audio and video servers, consist of four modules: audio recording, audio processing, video capturing and video processing. The audio capture is achieved through audio streaming using the microphone array. Thanks to the multi-channel audio interface device which is discussed in details later, audio stream can be captured by various sampling rates. Here, for our database, audio data is recorded with DVD quality, namely, 48 kHz and saved with the wav extension in the database. The audio processing module also takes this audio stream as an input to perform utterance detection and speech recognition. The recognizer then sends the recognized words to the dialog center.

Video data is captured by means of a normal USB, RGB webcam. As we will point it out, this low quality webcam is being replaced with Microsoft's Kinect in the next phase. However, for the initial system, this RGB camera provides us with 320x240, 10 fps, 16-bit raw video data. This video stream is then sent to the video processing module where its main objectives are to recognize faces (up to two faces) and to give the location estimate of these faces. Since it is a normal RGB camera, it cannot provide us with the depth estimate and thus dialog center is just approximately aware of the speaker location. To improve the source location and orientation estimate, audio information has to be taken into account. To this end, a function for delay estimation has been implemented in the audio processing module. This function uses a cross-correlation based approach to calculate the channel latency differences. Both audio and video location and orientation estimates are then used to perform optimal beam-forming.

In our initial system, the dialog center interface consists of several HTML pages and dialog messages. Each HTML page contains at least one hypertext which is read by speakers and recognized by speech recognizer and a few lines of instruction. These instructions guide the trial speakers on their gestures and poses. For instance, one page may ask speaker to look at the left bottom of the monitor. Thus audio and video data can be recorded in various speaker's gestures. This enables us to evaluate our algorithms for different speaker's head directions. As it was mentioned earlier, there is a hypertext in each page which is read by speakers and recognized by the speech recognizer. The better the beam-forming algorithm is, the higher the chance of recognition would be and as we already know, the beam-forming algorithm relays on both audio and video information at least for the location estimate. Based on the speech recognizer output and video server information, dialog center makes the decision for the next action.

This is a good example of how components have to cooperatively and effectively work to accomplish the task. These components are discussed in more details in the following sections.

## 5 Audio Server

Audio server is a software/hardware system serving several acoustic functionalities for the client (dialog center). Microphone array and data acquisition device constitute its hardware and are responsible for capturing the audio signal. Audio data is then processed and several features are extracted. These features are used to provide functions such as speech recognition and utterance detection for the client.

### 5.1 Microphone array

As it has been planned in VSHMI project, in order to extract desired signal in multiple-speaker noisy environment, a microphone array is used. It provides us with multi-channel audio data and enables us to employ spatial signal processing techniques, namely beam-forming, to improve the signal to noise ratio.

Building the microphone array is a time-consuming/expensive task and still is required because of its potential to suppress the background noise and/or eliminate interferences. The main difficulty in building a multi-channel audio device is to synchronize the audio channels. It means all A/D channels have to work with the same clock. The second technical difficulty is to efficiently transmit this huge data (collected from several channels) to the processor. If a PC is used as a processor, the best way to convey the data is to use the USB or FireWire1394 port.

The two common ways to build a microphone array are to assemble the analog circuit consisted of several A/D synchronized with the same clock and a USB port, which is a time-consuming approach or to use a multi-channel data acquisition device which is available in the market. However, this device has to be chosen carefully to meet the project requirements. Here, the second approach has been taken to construct the microphone array.

The array described in this report was built with an inexpensive data acquisition device plus cheap microphones. Eight microphones, Monacor ECM-2005, connected to an eight-channel data acquisition device, Focusrite Saffire PRO 40 as shown in Figure 2 (see also [1]), were mounted in a specially designed bracket in the configuration of the linear nested array. The assembled device is shown in Figure 3.

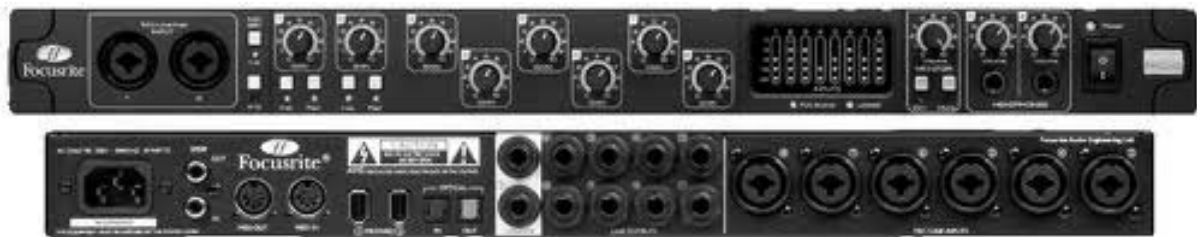


Figure 2: Front view (top) and back view (bottom) of the 8-channel audio acquisition device Saffire PRO 40

Monacor ECM-2005 microphones are one of the common lavalier microphones available in market and thus are easily replaceable in case of damage. Although they have some drawbacks such as their poor calibration in magnitude and phase, they are still suitable for this project since our goal is to resemble the low-cost microphone arrays that can be used in public self-service machines and to compensate for their disadvantages by using appropriate digital signal processing techniques. It then guarantees that the improved results are achievable in public self-service machines.

Focusrite Saffire PRO 40 audio interface, which is referred to as the data acquisition device here, is originally used by musicians in studios to record audio tracks of various musical instruments and microphones simultaneously. Apart from its reasonable price, it also provides us with so many options and features such as different sampling rates (44.1, 48, 88.2 and 96 kHz), supporting different operating systems and so on. More details about this device can be found in [1].



Figure 3: Microphone array consisting of eight microphones mounted on a bracket. The black boxes under the bracket are the microphone preamplifiers.

The data acquisition device has been set to work at 48 kHz sampling rate, based on the fact that it is straightforward to down-sample from 48 kHz to 8 kHz which is needed for some audio server functions such as speech recognizer. Since it is a FireWire device, a FireWire based audio device driver has to be used to collect audio data from this device. Thanks to the FFADO driver, it is now available in Linux. As they stated in their site, “The FFADO project aims to provide a generic, open-source solution for the support of FireWire based audio devices for the Linux platform”. FFADO can be connected to the well-known open-source audio connection kit in Linux, JACK, which is a system for handling real-time, low latency audio. JACK gives the full control over this device to the programmer and thus everything from the sampling rate to the internal buffer size can be set by the software.

After choosing the appropriate acquisition device, the second step is to design the optimal configuration for the microphones. As it can be seen in Figure 3, the microphones are not equally spaced. Actually the microphone array consists of 3 linear arrays each equipped with 4 microphones, as shown in Figure 4. That is because of the well-known beam-pattern frequency dependence effect.

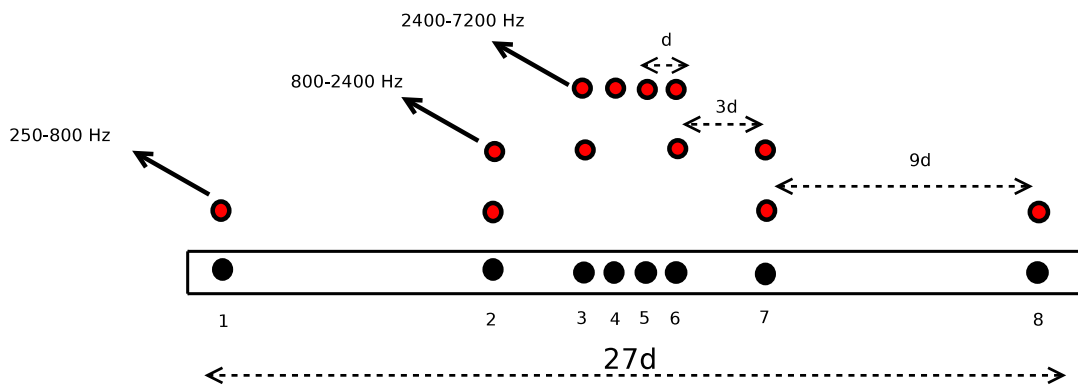


Figure 4: Non-uniform linear microphone array consisting of three nested uniform linear microphone arrays: the distances of the microphones in these three linear arrays are  $d$ ,  $3d$  and  $9d$ , resp., where  $d = 3$  cm.

The spatial angular response of arrays is usually referred to beam-pattern in array literature and plays an important role in microphone array architecture design and beam-forming algorithms. A Beam-pattern is intrinsically frequency dependent. The dependency on the operating frequency means that the response characteristics (beam-width, sidelobe level) will only remain constant for narrowband signals. To cover broadband signals such as speech, response-invariant broadband beam-forming techniques have to be applied. There are two common ways to design such a broadband beam-former.

The first one is to perform a subband decomposition and design the narrow-band beam-formers at each frequency such that all have the same response characteristics (beam-width, sidelobe level). The underlying idea of this approach is to split the broadband signal to several narrow-band signals, apply the narrow-band beam-forming to them and then add them up together. This is equivalent to applying a temporal filter to the array outputs, which is widely known as the filter-and-sum structure.

The second approach is to mechanically decompose the beam-pattern to the narrow band beam-patterns by implementing the array as a series of sub-arrays, where each sub-array corresponds to the specific frequency band (octave). Sub-arrays share the microphones to reduce the required microphones. Therefore microphones may belong to more than one sub-array. This approach is known as harmonic nesting (see Figure 4).

To attain invariance within each octave, here we combine harmonic nesting and filter-and-sum beam-forming together, whereby each element of a sub-array is followed by an FIR filter whose response is determined by the desired beam-pattern. Namely, array consists of 3 sub-arrays which are themselves linear arrays with uniform

spacing and filter-and-sum beam-forming as shown in Figure 5.

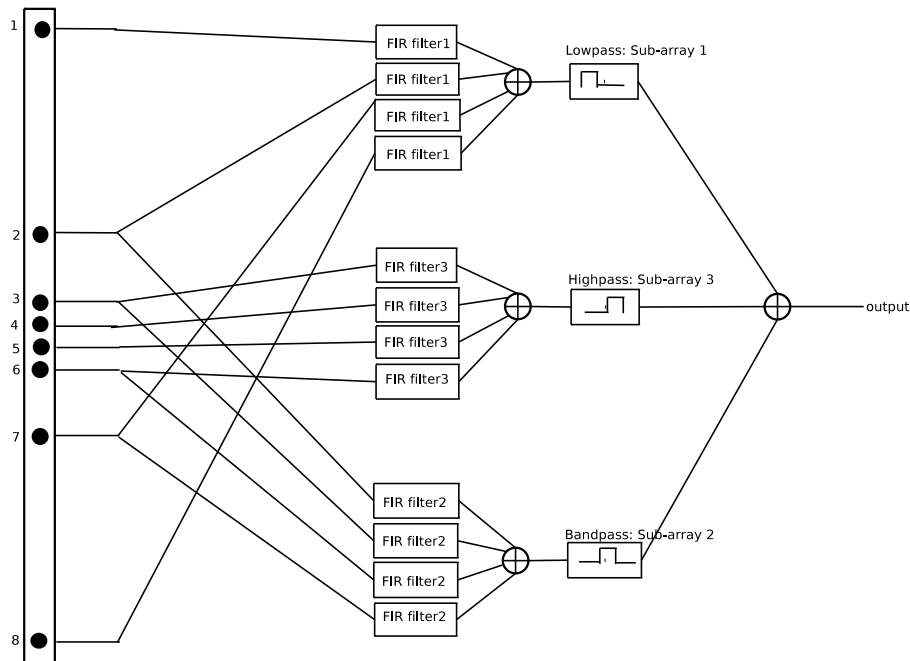


Figure 5: Nested linear microphone arrays with filter and sum beam-forming structure; note that some of the microphones are used by two arrays

As it has been depicted in figure 4, three sub-arrays have been used here:

- Sub-array 1 with microphones: 1,2,7,8 and working in [250-800] Hz
- Sub-array 2 with microphones: 2,3,6,7 and working in [800-2400] Hz
- Sub-array 3 with microphones: 3,4,5,6 and working in [2400-7200] Hz

Sub-array 1 has the largest inter-element space and therefore the ability to sense the phase variation of low frequency components while, sub-array 3 has the smallest inter-element distance and therefore used for high frequencies.

Each sub-array employs its own FIR-filters to achieve the desired beam-pattern in its operating frequency band. It means FIR filter design procedure has to be executed separately for each sub-array. Therefore, it has to be computationally efficient. FIR filters perform the desired beam-forming in this architecture. For instance, if we replace the FIR filters with delay compensation blocks, the simplest FIR filter  $\delta(n - n_0)$ , the simple delay and sum beam-former can be achieved. Here we use the more complicated beam-former to exploit the full potential of the multi-dimensional signal processing. The Main requirements that have to be met by the FIR filters are:

1. Attenuate the background noise as much as possible.
2. Their corresponding beam-pattern remains almost constant over the frequency band.
3. Pass the desired signal without distortion.
4. Force nulls in interference directions.

Basically, the filter design procedure is an optimization problem with an objective and several constraints.

The first proposition introduces the objective of the FIR filter design optimization problem. Here, the objective is to design FIR filters minimizing the diffuse background noise subjecting to the constraints defined by the next requirements. It has been shown that highly reverberant spaces, like office rooms, closely obey the diffuse noise model.

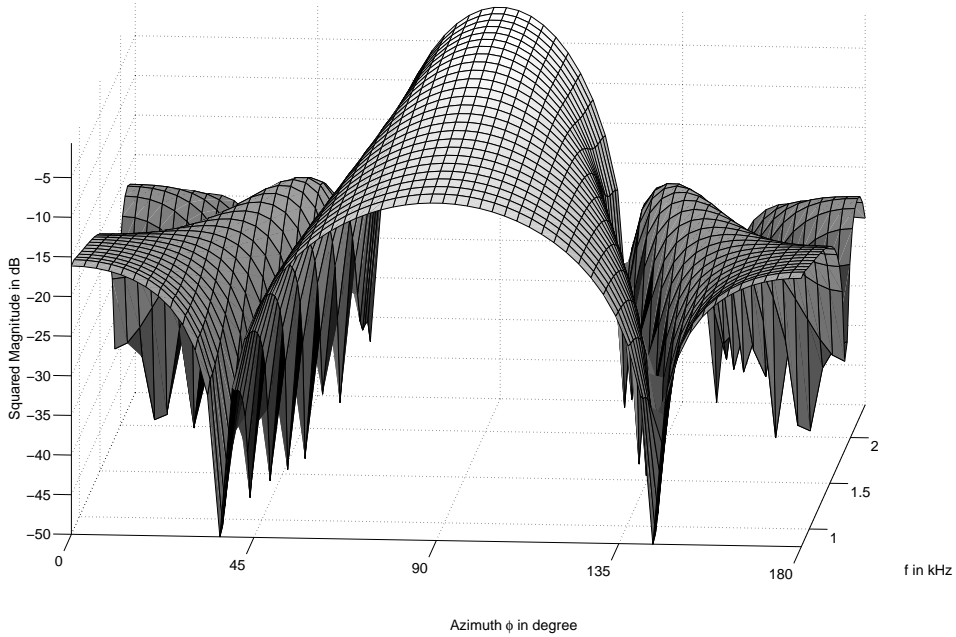


Figure 6: Beam-pattern for sub-array 2 with 4 microphones and  $d=9\text{cm}$

The second proposition introduces a set of constraints that have to be satisfied by beam-pattern. That is, beam-pattern has to be constant over a set of frequency points  $f_1, f_2, \dots, f_n$  where  $f_i \in [f_{l_j}, f_{u_j}]$ .  $f_{l_j}$  and  $f_{u_j}$  are the cut-off frequencies of the sub-array filter  $j$ ,  $j = 1, 2, 3$ , i.e. we try to have the same beam-pattern at these frequency points. It is worthwhile to mention that since our constraints are imposed at some points along the frequency axis, the final performance of the design procedure heavily depends on the number and places of these points.

The third and fourth requirements can be met by imposing the distortionless response and null constraints to our FIR filter design procedure.

This filter design procedure results in a broadband, frequency-invariant beam-pattern, Figure 6.

## 5.2 Software implementation of audio server

The Audio server software has been implemented in Java and as shown in Figure 7, it consists of five main components: Beamformer, JJackSystem, JJackReader, UtteranceDetector and HMMRecognizer.

Each component includes several methods providing various functions for other components or client. Since the client cannot directly access the audio server components, they use two unidirectional FIFO files to communicate. Commands written by the client into the first FIFO, are fetched by the server and their acknowledgments of receipt sent back in the second FIFO. The received commands are then executed and results are written into the second FIFO which is read by the client. Details of the implementations and function descriptions can be found in [4]. However, as an example, utteranceDetector component is briefly discussed here.

### 5.2.1 UtteranceDetector

UtteranceDetector is responsible for detecting utterances. It has several parameters that can be tuned such that it adapts to new environments and scenarios. Some of its parameters are as follows:

- \* winLen                    window length in samples
- \* winShift                window shift in samples
- \* maxClickSec            maximum duration of click sounds (in seconds)

## AUDIO SERVER

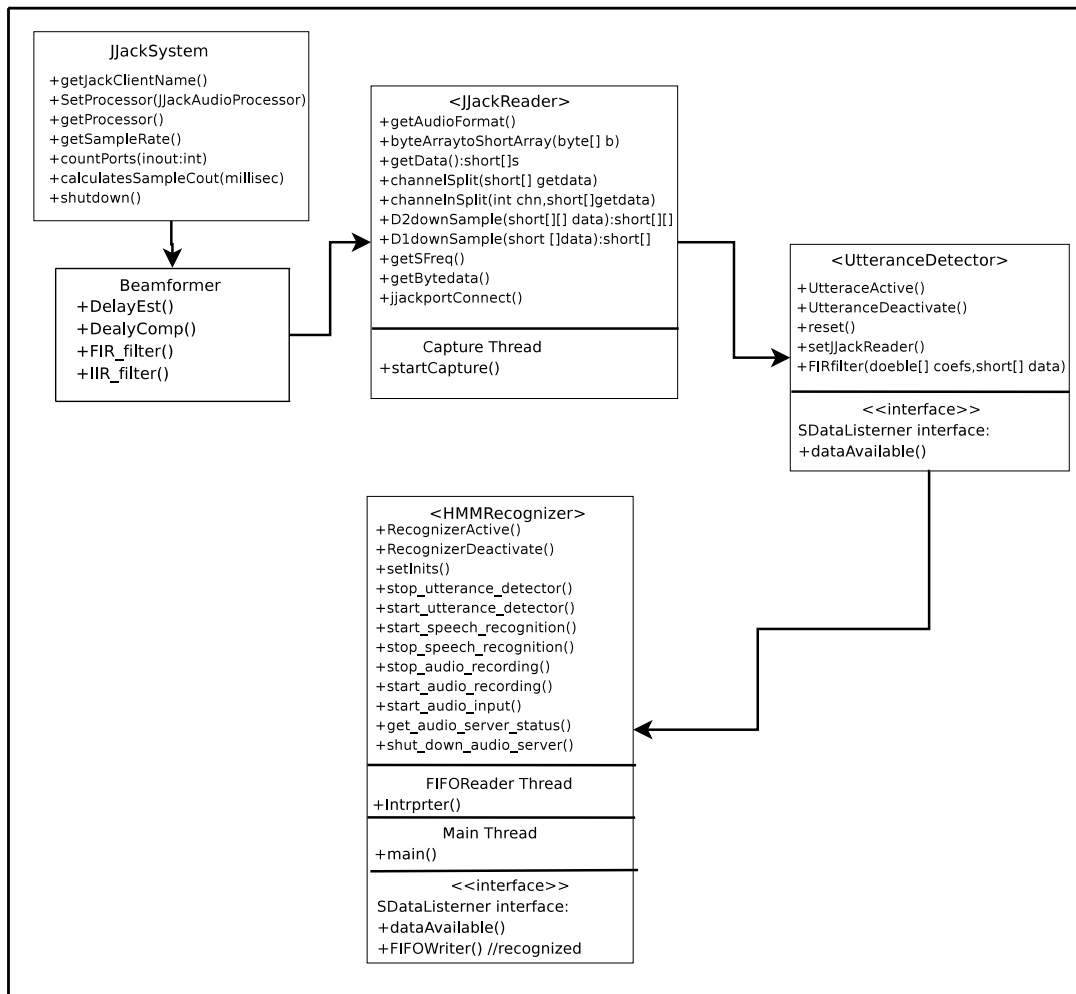


Figure 7: Audio Server structure

- \* maxPauseSec            maximum duration of a speech pause (in seconds)
- \* maxWaitSec            maximum time to wait for speech (in seconds)
- \* maxSpeechSec        maximum duration of an utterance (in seconds)
- \* minSpeechSec        minimum duration of an utterance (in seconds)

For instance, the last four parameters can be set such that the utterance detector works as a speech verifier for a given length of speech. More details can be found in [4]. In addition, some of the functions implemented to allow the dialog center control this component, are as follows:

UtteranceActive(): It activates the utteranceDetector.

UtteranceDeactivate(): It deactivates the utteranceDetector.

reset(): It resets the utteranceDetector to the initial settings.

FIRfilter(double coefs, short[] data): It returns the filtered version of the data by accepting the coefficients of the desired filter as an argument.

## 6 Video Server

This section gives a brief overview of the video server from dialog-center perspective. More details of this component can be found in the vision group report [2]. The video server has been implemented in C++ based on the open source library, OpenCV. In the initial system, video server has provided two functions for dialog center, face detection and face localization. Face localization is just in two dimension and cannot exploit the depth information. However, the video server functions will be increased when the currently available webcam is being replaced by Microsoft Kinect camera. Face localization information is used by dialog center to improve the audio based location estimate. This improved estimate is then passed as an input to the beam-forming module. Face detection information is employed by the dialog center to decide on the next dialog step. For instance, if there is no user present in the camera view based on the face detection information, regardless of the audio-based utterance detector output, dialog center will not start the communication and audio-based detected utterances, if any, will be considered as the background noise. Some video server commands that can be used by the client are as follows:

```
get_video_server_status() --> (cc,nch)
    Test if the server is active and get some status information
```

```
Output: cc completion codes:
    0: server is alive and waiting
    1: video input active: video data is analyzed but
       not recorded on the disk
    2: server is analyzing the video stream(s)
       and saving it to disk
    3: recording automatically stopped because max
       duration or storage limit has been encountered
    nch number of available video channels, i.e, cameras
```

```
set_facetracker_params(max_no_faces, min_face_size, algo, freq, log) --> (cc)
    Set parameters for the face tracker.
```

```
Inputs: max_no_faces    maximum number of faces to be tracked
        min_face_size   minimum expected face size (as a ratio to
                           the image size)this basically tells the system
                           how far the faces can be from the camera
        algo            algorithm to be used for tracking
        freq           frequency with which the results are provided
```

## 7 Technical Issue

There is a technical issue that has to be addressed.

**Audio-video data synchronization:** The video and audio capture are performed using two different sensors, a camera and a microphone. These sensors are not connected, and so the captured signals are totally independent of one another. Since the proposed system is such that the video processing and audio processing are performed separately, dialog center has to find a way to synchronize the extracted audio and video information. To this end, exact time stamps have to be assigned to all stored data packages and extracted information. In other words, audio and video server have to assign the time stamp to each packet. Nevertheless, since audio and video data are sensed by separate devices with different internal buffers and processed with various algorithms with different level of complexity, they may report different time stamps for the same event. The difference between time stamps can be seen as a time stamp jitter. As long as this time stamp jitter is in the order of several milliseconds, it is tolerable. Otherwise, it has to be compensated. One potential solution is to report the time stamps not only at the start of the events but also during their existence, i.e. audio and video servers periodically report the time stamps during the event lifetime. Hence, any clock drift would be observed.

## 8 Project Outputs

This section summarizes the achievements of the first phase of the project. An appropriate infrastructure is needed to collect a suitable multi-channel audio-video database. This infrastructure has been built and implemented in the first phase. It consists of both hardware and software parts. Microphone array and cameras constitute the hardware components. Microphone array has been built by several cheap microphones and multi-channel audio input-output device. Software components are audio and video servers which have been implemented in Java and C++ respectively. They provide several audio and video-based functions which can be used by any client. Here, the dialog center is the client and has been implemented in Java. Microphone array and its corresponding signal processing issues were studied and the missing knowledge has been acquired.

## 9 Targets for the Next phase

The next natural step is to record audio and video data by means of the built up experimentation system. Then the new algorithms will be developed and their performance improvement be measured based on this database. A very important issue of the audio-visual approaches is to optimally combine the audio and video information in order to take the best decision at each moment. The final goal of the project is to find this optimal fusion algorithm. At the end, it has to be mentioned that instead of using expensive ToF camera, Microsoft's Kinect which works much better than ToF and is much cheaper, is employed in the next phases.

## References

- [1] Focusrite Company. *Saffire PRO 40 Audio Interfaces*.
- [2] G. Fanelli. *Vision Component of the VSHMI Experimentation System*. Annual Report of the SNSF project no. 200021 130224/1. Computer Vision Laboratory, ETH Zurich, April 2011.
- [3] I. Vepa M. Lincoln, I. McCowan and H. K. Maganti. The multichannel wall street journal audio visual corpus (mc-wsj-av): Specification and initial experiments. *in Proc. ASRU*, pages 357–362, 2005.
- [4] B. Pfister and T. Naghibi. *Concept of the VSHMI Experimentation System*. Report of the SNSF project no. 200021 130224/1. TIK, ETH Zurich, June 2010.