



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich



Concept of the VSHMI Experimentation System

Report of the SNSF project no. 200021 130224/1

Vision-Supported Speech-Based Human Machine Interface

Beat Pfister and Tofigh Naghibi

Speech Processing Group

TIK, ETH Zurich

June, 2010

1 Introduction

The main aim of the SNF project VSHMI is to improve the human-machine interaction algorithms. For the development and test of such algorithms we need appropriate audio-visual data from typical user interactions. Therefore we have to build 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.) will be used. However, the system has to be 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 have to be recorded. The amount and the precision of the recorded data should be 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 improved algorithms for exactly the same input data and thus the improvement can be measured.

2 System architecture

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

The video and audio servers provide functions such as user detection/tracking and identification (by means of face and/or voice features), speech recognition and recording of audio and video input signals. The dialog center is the heart of the system and is responsible for controlling the audio and video servers, fusing their information and making decisions for next dialog steps. Therefore, a set of audio and video server commands is needed to control the servers, to communicate with them and collect their information.

This architecture allows to use different programming languages for the servers and the dialog center. Furthermore, the three system components can easily be run on different computers.

In the first version of the system, the dialog center will communicate via FIFOs with the servers. More precisely the dialog center will write commands to a first FIFO, the server will fetch them, start or perform the required action and write a confirmation (and possibly a result) to the second FIFO. Note that the servers have to send the confirmation back as quickly as possible. It should not happen that a command is sent to a server and the corresponding acknowledgment arrives after several seconds only. Thus the servers have to be designed in a way that they are always able to immediately respond to incoming commands, in particular also to stop commands.

The names of the input and output FIFOs for a server will be defined by means of arguments of the server start command that has the following form: `audio_server_xy <input FIFO> <output FIFO>`
For flexibility reasons, all other information for the server will be transferred via these FIFOs.

For server actions that produce a more or less continuous stream of output results (such as the face tracker), an additional FIFO is used to transfer the server results to the dialog center. Note that the specification of this FIFO and of all other input and output channels will be provided by means of commands that the server receives via the above mentioned first FIFO.

3 Audio and video server coordination

The idea is to start with a rather simple scenario: The trial users have to interact with the system that performs a sequence of dialog steps. The audio and video data and the results of the audio and video processing components have to be stored in a way that reproducibility is guaranteed and the improvement of future algorithms can be measured (see Section 1).

The reproducibility can be attained by logging the actions of the servers and of the dialog center. The logged information should be detailed enough. In particular it has to show exactly from which audio and/or video data which algorithms (including all settings of parameters and options) have produced the logged results.

For future multimodal methods, e.g. audio-visual speech recognition, it is also necessary to know which audio and video data belong to the same time points. In order to accurately synchronize audio and video data, exact time stamps have to be assigned to all stored data packages, to the logged commands and communication events. Detailed requirements for this time stamping have to be elaborated.

4 Command summary

Unless otherwise specified, all the below specified commands respond with a confirmation even if the result of the required action is not yet available. If e.g. the audio server has to listen for a spoken input from the user at a particular time instant of a dialog, the dialog center sends a `start_speech_recognition` command and the server immediately has to start the utterance detection and recognition process and then it has to answer back instantly, if the start was successful or not. The recognition result is fetched later from the server by polling.

Note that the servers answer for all types of invalid commands with the completion code -99 (for the sake of compactness this code is not mentioned in the below command specifications).

The format of the commands is `<command string> -> <reply string>`, where `<command string>` designates the command (character string) that the dialog center sends to the server, `<reply string>` is the answer or confirmation of the server and `->` is a delimiter between these two items that is only used for the sake of clarity of the commands specified in the following sections.

4.1 Audio server commands

The following commands are used to select, activate and deactivate the audio inputs, start and stop the recording of audio data and to initiate the various audio server functions. Note that if an argument includes more than one value (e.g. the audio channels to be activated) it has to be specified as `{arg1, arg2, arg3, ...}`.

```
get_audio_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: audio input is active: data from audio input
             device is stored in the server-internal buffer
             only; from there it can be read for further
             transfer or processing
          2: server is recording input audio data (data of
             the server-internal buffer is continuously
             written to file)
          3: recording automatically stopped because max
             duration or storage limit has been encountered
          -1: server-internal buffer overflow occurred
          nch    number of available audio channels
```

```
init_audio_server(fs1,fs2,ch,buf,fifo,log) --> (cc)
```

Initialize the audio server and set some parameters for audio input.

```
Inputs: fs1      sampling frequency of audio acquisition device
          (set of possible values depends on device)
          fs2     sampling frequency of audio signals to be used
          (8 or 16 kHz)
          ch      audio channel(s) to be activated as inputs
          buf     internal buffer size
          fifo    output FIFO name (for recognition results etc.)
          log     log file name (existing log file will be overwritten)
```

```
Output: cc      completion codes:
          0: server successfully initialized
          -1: invalid audio device sampling frequency
          -2: invalid speech signal sampling frequency
          -3: invalid channel specs
```

- 4: internal buffer not available
- 5: output FIFO not available
- 6: cannot open log file

start_audio_input() --> (cc)

Start analog-to-digital conversion of the audio input device and initiate the transfer of the data from the selected channel(s) (see command `init_audio_server`) to the server-internal buffer. Old buffer content is flushed.

Output: cc completion codes:
0: audio input to server-internal buffer started
-1: audio server not initialized

set_audio_data_storage(dir,fil,alloc) --> (cc)

Define disk space for audio data storage

Inputs: dir name of directory for audio data storage
fil name of file for audio data storage
alloc disk space in MByte to be allocated (audio data recording will automatically be stopped if the allocated disk space gets full)

Output: cc completion codes:
0: storage successfully allocated
-1: cannot create directory
-2: cannot create file
-3: cannot allocate required disk space

start_audio_recording(maxdur) --> (cc)

Start transfer of the audio input data from the server-internal buffer to the directory/file set with command `set_audio_data_storage`. Note that the data storage will be stopped as soon as the allocated disk space is used or the maximum recording duration is reached.

Input: maxdur maximum recording duration in seconds
(0 means no duration limit)

Output: cc completion codes:
0: audio recording successfully started

stop_audio_recording() --> (cc)

Audio data transfer from the server-internal buffer to file is stopped and the recording file is closed.

Note that the transfer of audio data from the audio device to the server-internal buffer is continued!

Output: cc completion codes:
0: audio input stopped and file closed
-1: audio input is not active
-2: audio recording not active

-3: cannot close recorded audio data file

stop_audio_input() --> (cc)

Data transfer from the audio input device to the server-internal buffer is stopped.

Output: cc completion codes:
 0: audio input stopped
 -1: audio input is not active

set_detection_params(nlevel,pause,focus) --> (cc)

Set utterance detection parameters (this command will be extended to handle additional parameters)

Inputs: nlevel adaptive noise level is initialized to this value
 pause maximum allowed pause length within an utterance
 focus microphone array focus direction (angle)

Output: cc completion codes:
 0: initialization successfully completed
 -1: active utterance detection or recognition encountered; request denied
 -2: invalid parameter value

set_recognizer_vocabulary(voc,opt) --> (cc,nt)

Load, add or remove speech recognizer vocabulary items

Inputs: voc character string of semicolon-separated vocabulary items of the following form (see also recognizer dictionary): (<item1>; <item2>; ...)
 opt command options:
 0: supersede previously loaded vocabulary items
 1: add the new items to previously loaded ones
 2: remove the specified items from the loaded vocabulary

Outputs: cc completion code:
 0: command successfully completed
 -1: speech recognizer is active; request denied
 -2: invalid vocabulary item encountered
 -3: item to be removed not in loaded vocabulary
 tn number of vocabulary items loaded

start_speech_recognition(ch,dtime,wtime,num) --> (cc)

Start the speech recognizer; after the maximum number of utterances have been detected, recognized and the results written to the appropriate FIFO (see command init_audio_server) the recognizer will automatically stop.

The format of the recognition results are: {word,rcode}

The elements of a result mean:

word graphemic representation of the recognized vocabulary item

```
rcode utterance detection and recognition code:
  0:  recognition successfully completed
 -1:  out-of-vocabulary word recognized
 -2:  noise detected/recognized
 -3:  head of utterance truncated
 -4:  tail of utterance truncated
 -5:  maximum waiting time exceeded
```

```
Inputs: ch      audio input channel (one single!) to used for
          recognition
        dtime   relative utterance detection stating time
          (example: if dtime = -2 then the utterance detection
          starts two seconds in the past, i.e. 2*fs samples
          before the end of the server-internal buffer)
        wtime   maximum waiting time in seconds
        num     maximum number of utterances to be detected and
          recognized
```

```
Outputs: cc     completion code:
              0:  recognition successfully started
             -1:  analog input not active
             -2:  invalid command argument encountered
             -3:  unknown error
```

```
stop_speech_recognition() --> (cc)
```

The speech recognizer is immediately stopped and all partial recognition results are discarded.

```
Outputs: cc     completion code:
              0:  command successfully completed
             -1:  no ongoing recognition
```

4.2 Video server commands

The following commands are used to select, activate and deactivate the video inputs, set its parameters, start and stop video capturing and call video server functions.

The following commands are just ideas; they have to be completed, maybe renamed and in particular specified more precisely:

```
ping_video_server:          (test if server is active)

set_video_parameters:      frame rate
                           resolution
                           internal buffer size
                           output FIFO buffer

set_video_storage:        directory
                           filename
                           disk space allocation
                           log filename

start_video_recording:     max recording duration

stop_video_recording:

set_facetracker_params:    tracker initialization params
                           frequency of results

start_face_tracking:

stop_face_tracking:
```