



UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

ESTREAM

Gabriela-Carmen Dinică ¹ Robert Niemiec ²
Asan Urazimanov ³

SUPERVISOR:

Prof. Dr. Denis Lalanne

28 MAY 2019

MULTIMODAL USER INTERFACES - REPORT

¹gabriela-carmen.dinica@students.unibe.ch, University of Bern

²robert.niemiec@students.unibe.ch, University of Bern

³asan.urazimanov@students.unibe.ch, University of Bern

Abstract

This paper provides a solution for improving the current version of the Logitech Capture streaming application. The approach is based on two input modalities: voice and movement. Based on these inputs, different commands are triggered in the Logitech Capture application.

Keywords: Seminar report, Stream, Multimodal, Gesture recognition, Voice recognition, Controlled evaluation, T test

Contents

1	Introduction	1
2	Main Idea	1
3	CASE CARE	2
3.1	CASE	2
3.2	CARE	2
4	Technologies	3
4.1	Hardware	3
4.2	Software	3
4.2.1	Haar Cascade	3
4.2.2	Speech Recognition	4
5	Implementation	5
5.1	Haar Cascades	5
5.2	Speech Recognition	5
5.3	Screenshots	6
5.4	Logging	6
5.5	Gesture Recognition	7
6	Evaluation	8
6.1	Controlled evaluation	8
6.2	Results	8
7	Conclusion	11
	Appendices	15
A	Form used for controlled evaluation	15

1 Introduction

With the increasing popularity of online content streaming services, an increasing amount of people consider the job of a content creator on streaming platforms instead of creating static content (e.g. YouTube). Following this trend, many ideas are considered to improve the content creation process of streamers, i.e. simplifying streaming, as well as making the user-viewer interaction more dynamic and diversified. One of the potential approaches is multimodality.

This project focuses on creating an interface which utilizes both voice and gesture commands to provide more freedom and ease of use for the streamer.

2 Main Idea

The main idea of the project is to integrate voice recognition and movement detection for increasing the efficiency of Logitech Capture, a streaming application. The current problem is that a streamer needs to be next to a keyboard in order to issue any commands to the application. For example, in order to start the recording, the user needs to press “alt + R”. It has the possibility to change the letter combinations but it still has to give input from the keyboard in order to do any action in the application.

Our solution consists of identifying different commands from the user based on speech and gesture recognition. Initially, the streamer has to raise one hand above the head in order to trigger the microphone. Once the microphone is triggered, it listens for the voice command. Depending on what the person is saying, the program will interpret it and will execute the command.

We covered all the basic commands that the applications offers: starting and stopping the recording, muting and unmuting the system sound, taking a screenshot, switching the sources and cycling the sources. In addition, we added voice commands for increasing and decreasing the system’s volume.

All these changes were designed to increase user satisfaction. Usually a streamer is not always next to a computer or does not have easily access to keyboard. For this case, we decided that the best solution would be integrating voice and gesture commands. The hardware technologies that we used are camera and microphone so that everyone can have easy access to these resources.

3 CASE CARE

In this section the project's modalities are positioned according to the CASE/CARE models.

3.1 CASE

Use of modalities: all of the implemented multimodal interactions start with a hand gesture (hold hand above head), followed by a voice command, so the use of modalities is sequential.

Fusion: In order for the speech to be registered, the hand gesture has to be performed first. This means that the speech system uses a coreference to the gesture system, meaning fusion is required.

Levels of abstraction: The gestures require to hold the hand above the head. In that sense, the feature level is examined (height of the hand relative to the head). It is worth noting that the source switch command also checks the x-axis position of the hand.

Conclusion: The implemented modalities are alternate in the CASE model (A task with temporal alternation of modalities, using coreferences)

3.2 CARE

Complementarity: The interactions are complementary, since no modality taken individually is sufficient to reach the state.

Assignment: The requirement for reaching this property is that only one modality can be used to reach a given state. The implemented interactions utilize both voice and gestures, so assignment is not achieved here.

Redundancy: Both modalities have to be used in order to accomplish the desired interaction, neither of the modalities have the same expressive power. Therefore, Redundancy is not achieved.

Equivalence: There is only one choice for achieving the end states, it is insufficient to use only one of the available modalities. Also, there is a temporal constraint between activating the gesture and uttering the voice command. Therefore, Equivalence is not achieved.

4 Technologies

4.1 Hardware

Since the aim of the project is to enhance the streaming experience for the creator, it stands to reason that the two most commonly used streaming devices - cameras and microphones - should be used for this solution. The video and audio streams are collected and processed by their corresponding libraries.

4.2 Software

4.2.1 Haar Cascade

Haar cascade is a machine learning algorithm used to detect visual objects. The algorithm was introduced by Paul Viola and Michael Jones in their paper titled “Rapid Object Detection using a Boosted Cascade of Simple Features”. The algorithm offers an extremely fast and high precision detection, making it ideal not only for object recognition in images, but also in videos.

The algorithm is composed of three stages

- Haar Feature Selection using Integral Images
- Training
- Cascading Classifier.

The first step is collecting the Haar Features. A Haar feature consists of placing one of the masks in Figure 1 at a location in the image and then adding all pixel intensities in the white region and subtracting all pixel intensities of the black regions. Therefore, the Haar feature is actually a single value. However, an image could have quite a large number of features. For example, as stated in the OpenCV documentation for Haar Cascades [1], even a 24x24 image window results in over 160000 features.

Because of the large number of features present in an image, Integral Images is used

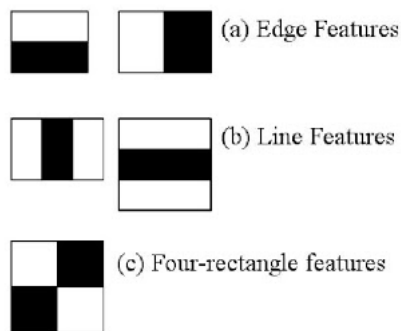


Figure 1: OpenCV example for Haar features[1]

to determine the mask efficiently. First, we observe that each of the masks present in Figure 1 is reduced to an addition or subtraction of 2, 3 or 4 sub-matrices of the original matrix – for example, the first mask can be reduced to the sum of the top white region minus the sum of the bottom black region.

Now, some of the Haar features are more important than the others. In fact, most of them are irrelevant and can be dropped. The process of training consists of selecting a subset of all Haar features of an image in such a way that they still correctly represent the desired object. This is done using the **Adaboost** algorithm. The algorithm not only determines an ordered list of the most important features, but also a threshold for each of them which should be used by a classifier.

The last step is the classification. Assume that we start with an image in which we would like to detect the object. We start by placing the most important Haar mask at each location in the image, and compare the response with the respective threshold. All the remaining candidates are then moved to the second mask and so on until all masks are used - in a sort of cascade ordering.

4.2.2 Speech Recognition

For the speech recognition, we used the publicly available Google Speech Recognition algorithm. The algorithm works fairly fast, though not as fast as the image recognition algorithm described earlier. More implementation details can be found in the section 5.2.

5 Implementation

In this section, we will describe the various implementation details that were used in the project. All of the implementation was done in Python 3, which allowed the components to directly interact between each other.

5.1 Haar Cascades

As shown in Section 4.2.1, Haar Cascades could be used to detect objects in real-time with a fairly high accuracy. In the project we used them to detect the hands and the face of the user almost immediately as the webcam records the video.

The recommended library for image and video manipulation is *OpenCV* [5]. It has complete integration with *Python*, making it ideal for our use case. It offers implementation for Haar Cascades, in the form of `cv2.CascadeClassifier`.

In order to construct the classifier, an *XML* file containing the pre-trained Haar features and their thresholds must be passed to it. *OpenCV* offers these *XML* files for a set of popular objects – including classifiers for hands and faces, found in the GitHub repository [2]. The classifiers have been initialized as follows:

```
hand_cascade = cv2.CascadeClassifier('Hand.Cascade.1.xml')
face_cascade =
    cv2.CascadeClassifier('haarcascade_frontalface_alt2.xml')
```

These classifiers were trained on Gray-scale images, hence the normal feed that we get of the webcam would not work. Fortunately, *OpenCV* provides a library of converting a frame of the video into Gray-scale:

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

Detection was then done by calling the classifiers on the Gray-scale frame:

```
hands = hand_cascade.detectMultiScale(gray, 1.1, 10)
faces = face_cascade.detectMultiScale(gray, 1.1, 5)
```

The functions return a list of rectangles where the individual objects were found – like a bounding box. During development, it was observed that the hands recognition was slightly worse in terms of precision than the face one. Therefore, we asked the algorithm to be more restrictive in matching, by setting the third parameter to 10 as opposed to 5. This filters out false positives by using a neighborhood approach — as in, filter a result out if it does not have at least 10 other matches in its close vicinity.

5.2 Speech Recognition

The Google Speech recognition can be called via the SpeechRecognition Python library [6]. In fact, the library contains a larger set of recognizers — such as Google Cloud Speech API, Microsoft Bing Voice Recognition and IBM Speech to Text — though Google Speech Recognition is the most popular one.

Additionally, the library also provides support for recording the voice from the microphone. This is done as follows:

```
audio = r.listen(source, timeout=1, phrase_time_limit=5)
```

Lastly, the audio is converted to text as follows:

```
voice_cmd = r.recognize_google(audio)
```

5.3 Screenshots

One of the issues encountered was integrating the components with the Logitech Capture application [4], since it did not have any publicly available API. In particular, the webcam is not directly accessible from two different applications – the Logitech Capture and our program.

We solved this issue by changing the data source of the video from the webcam to taking continuous screenshots of the Desktop – which in turn contained the webcam feed from the Logitech Capture. Then, the image recognition ran the same way as previously described. This works well as a proof of concept, though if given the opportunity, a direct integration into the Logitech Capture webcam feed would be preferable.

Screenshots were taken using the *MSS* [3] library of Python. The image was also scaled down by a factor of 0.6 to compensate for the large difference in resolution between the regular webcam feed and the monitor.

```
img = np.array(sct.grab(monitors))  
frame = cv2.resize(img, (w_resize, h_resize))
```

5.4 Logging

In order to keep track of various possible errors, logging was added for the speech recognition. Any such logging information is appended to a common output file. Each message that needs to go into the file gets prepended with *ISO-formatted* current time. The following is an example of the logging information available in the file:

```
2019-05-23T20:56:10.393434: Error in recognition  
2019-05-23T20:56:51.376196: volume up  
2019-05-23T20:57:46.240635: toggle recording  
2019-05-23T21:00:21.700145: volume down
```

5.5 Gesture Recognition

There are several methods to detect the hand gestures. The skin color detection is one of the most popular methods. This method is simple and depends on skin color that can be white, black, or other colors, and the environment light conditions, as well as the background. Another method does not use the color hand; it uses the convexity detection of OpenCV.

The aim was to device a program that is able to detect out hands, track them in realtime and perform some guesture recognition. It is do be done with signal processing performed on images obtained from a regular laptop web-camera.

Given the feed from the camera, the 1st thing to do is to remove the background. We use running average over a sequence of images to get the average image which will be the background too. We use an inbuilt background subtractor based on a Gaussian Mixture-based Background/Foreground Segmentation Algorithm. Background subtraction involves calculating a reference image, subtracting each new frame from this image and thresholding the result which results is a binary segmentation of the image which highlights regions of non-stationary objects . We then use erosion and dilation to enhance the changes to make it more prominent. We convert input RGB image into gray scale image(3d to 2d),

```
cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

blur the gray scale image using GousianBlur function,

```
cv2.GaussianBlur(gray,(5,5),0)
```

then we threshold it using.

```
cv2.threshold(blur,70,255,cv2.THRESH_BINARY_INV+cv2.THRESHOTSU)
```

It should be noted, mask detection with upper and lower HSV ranges.

```
inRange(converted,lower,upper)
```

```
cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(7,7))
```

```
cv2.morphologyEx(skinMask5,cv2.MORPH_CLOSE,kernelClose)
```

```
cv2.dilate(skinMask,kernel,iterations=4)
```

```
cv2.GaussianBlur(skinMask,(7,7),0)
```

```
cv2.bitwise_and(img5,img5,mask=skinMask5)
```

Contours to hand regions.

```
cv2.findContours(skinMask,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
```

Next steps are finding out convexity defects.

For Virtual Mouse we setup global variables, as screen resolution, captured image resolution, implement the Open Gesture operation, based on mathematical calculations, and implement Close Gesture/ Clicking. The tuning operations have been provided at the end.

6 Evaluation

The feature that we wanted to test was switching the cameras. We implemented two methods for this case. The first one is triggering the microphone by raising the hand above the head and then issuing the voice command. In this case the command is a longer one “switch to first source” or “switch to second source”. The second case is triggering the microphone by raising one hand above the head and giving the command “switch source”. Depending on which side the hand was risen, it will either switch to source one or two.

6.1 Controlled evaluation

We chose to conduct a within-group evaluation with six people. The first three people started with method one and followed with method two. The last three people started with method two and followed with method one.

Null-Hypothesis

There is no difference in user experience (time and satisfaction) between: switching the camera depending on the side on which the hand is and switching the camera using only a voice command.

Independent variables

- voice command
- gesture command

Dependent variables

- time to switch the sources
- satisfaction scores

For the first task users were asked to perform the following instruction: “Trigger the voice command by raising one hand above the head and give the following command: switch to first source. Repeat the process and switch to second source.”. The second task was the following: “Switch to source one by raising the right hand above the head and giving the command: switch source. Switch to source two by raising the left hand above the head and give the following command: switch source”.

6.2 Results

We chose to have both qualitative and quantitative evaluations. The qualitative one was based on the user satisfaction. The figures 2 and 3 below illustrate the satisfaction of the users that tested both methods. As we can see, the second method was better rated.

How much did you like the method 1?

6 responses

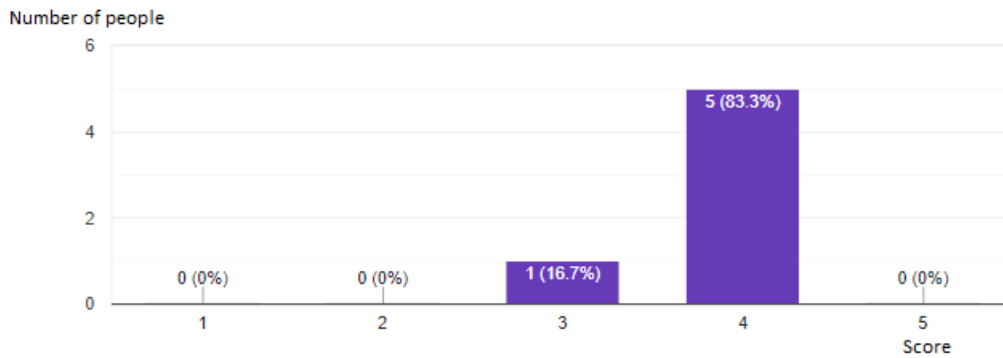


Figure 2: Scores given to method 1

How much did you like the method 2?

6 responses

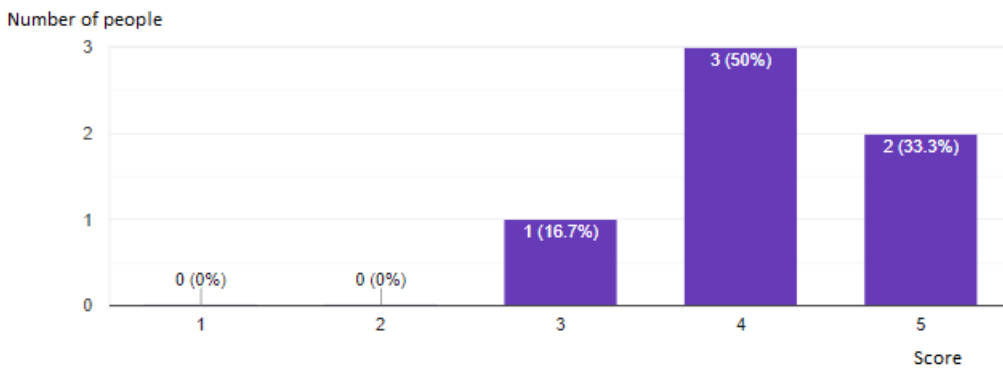


Figure 3: Scores given to method 2

In the table below we can see that the second method was slightly faster than the first method. The average time for the first method is 18.91 while the average time for the second method is 16.63 seconds. Despite this result the T test has a value of 0.1788. Therefore we failed to reject the Null Hypothesis i.e., there is no difference between the means.

Method 1 (sec)	Method 2(sec)
16.28	12.87
21.37	17.23
19.64	15.65
22.67	20.87
16.89	18.02
16.62	15.16

Even if the second method is faster, some people stated that it can be confusing which hand to raise depending the camera that they want to change. We concluded that every method had its advantages and disadvantages so we chose to let both implementations available for the user. Therefore, a streamer can choose what method he wants to use.

7 Conclusion

We developed a voice and gesture recognition for the Logitech Capture application. Our goal was to help streamers be time efficient when it comes to regular tasks like changing the camera or starting and stopping the recording. They no longer need to be next to a keyboard to perform these actions.

The project uses Haar Cascades – a real-time and high accuracy object detection algorithm – to detect the faces and hands in the webcam feed. If at any point a hand is raised above the head for a short period of time then the algorithm listens via the microphone. If the user pronounces one of the predefined commands, then an action is triggered on the Logitech Capture application via a keybinding. Speech-to-text is handled via the Google Speech Recognition library.

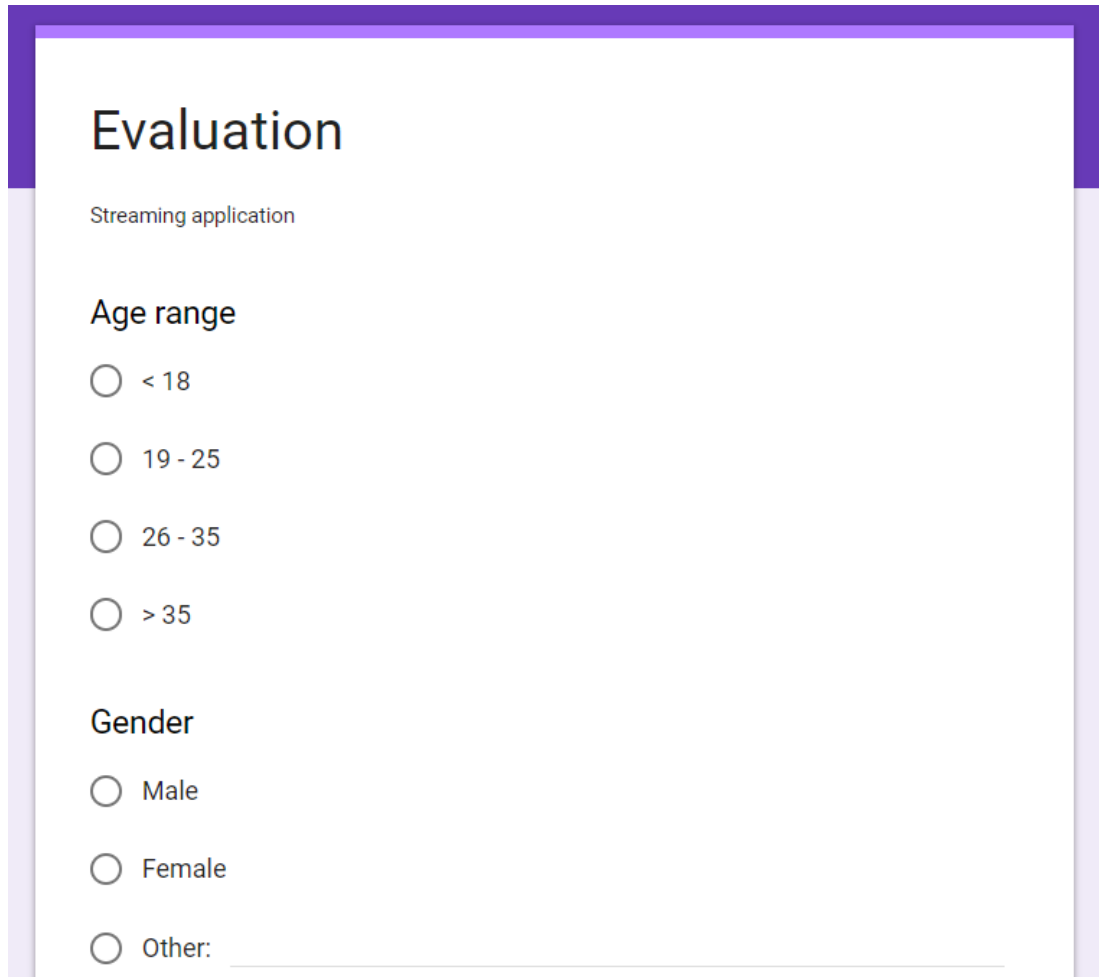
There is much room for improvement of the algorithm, which has not been considered due to time constraints. Firstly, the text that comes from the translation is matched against a predefined set of commands. This could be improved by adding some understanding layer to capture the users desired action better. Secondly, users don't normally move the webcam position too much. Under this assumption, we could do some image processing to eliminate the background in the video, thus reducing some of the over-triggering of the image recognition algorithm.

References

- [1] Face detection using haar cascades. https://docs.opencv.org/3.4/d7/d8b/tutorial_py_face_detection.html. Accessed: 2019-05-22.
- [2] Github for the haar cascades pre-trained xml files. <https://github.com/Balaje/OpenCV/tree/master/haarcascades>. Accessed: 2019-05-22.
- [3] Mss library in python. <https://pypi.org/project/mss/>. Accessed: 2019-05-22.
- [4] Official website for the logitech capture streaming application. <https://www.logitech.com/en-ch/product/capture>. Accessed: 2019-05-22.
- [5] Opencv library in python. <https://pypi.org/project/opencv-python/>. Accessed: 2019-05-22.
- [6] Speechrecognition library in python. <https://pypi.org/project/SpeechRecognition/>. Accessed: 2019-05-22.

Appendices

A Form used for controlled evaluation



The image shows a digital evaluation form titled "Evaluation" for a "Streaming application". The form is presented within a purple-bordered frame. It includes three sections: "Age range" with four radio button options (< 18, 19 - 25, 26 - 35, > 35), "Gender" with three radio button options (Male, Female, Other: _____), and a final "Other:" option with a text input field.

Evaluation

Streaming application

Age range

☐ < 18

☐ 19 - 25

☐ 26 - 35

☐ > 35

Gender

☐ Male

☐ Female

☐ Other: _____

Profession

Your answer

Method 1 - Time

Your answer

Method 2 - Time

Your answer

How much did you like the method 1?

	1	2	3	4	5	
Not at all	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very much

How much did you like the method 2?

	1	2	3	4	5	
Not at all	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very much

Comments

Your answer

SUBMIT