GROUP PROJECT REPORT

JHA

Handy Music Assistant

Multimodal user interface
University of Fribourg
May 2019

Hichem Belhocine
Julia Eigenmann
Aliaksei Syrel

Index

1. Introduction

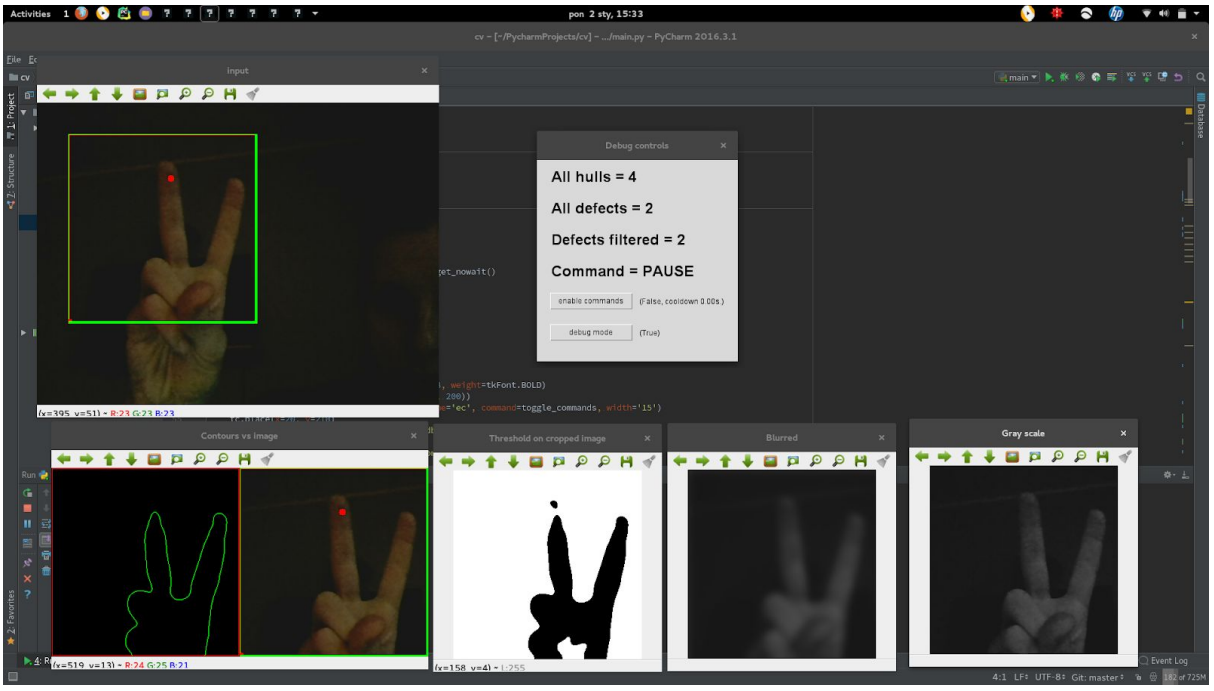
In the following report we present our user assistant application *Handy Music Assistant* that facilitates people to play music. In the first part of this report, we briefly present the hand detection with libraries that work with it. In the second part of the report, we briefly present the *SpeechRecognition*. In the last part, we present how we integrate *SpeechRecognition* with hand detection for our application Handy Music Assistant and make a conclusion.

2. Real Time Hand Detection with TensorFlow and OpenCV

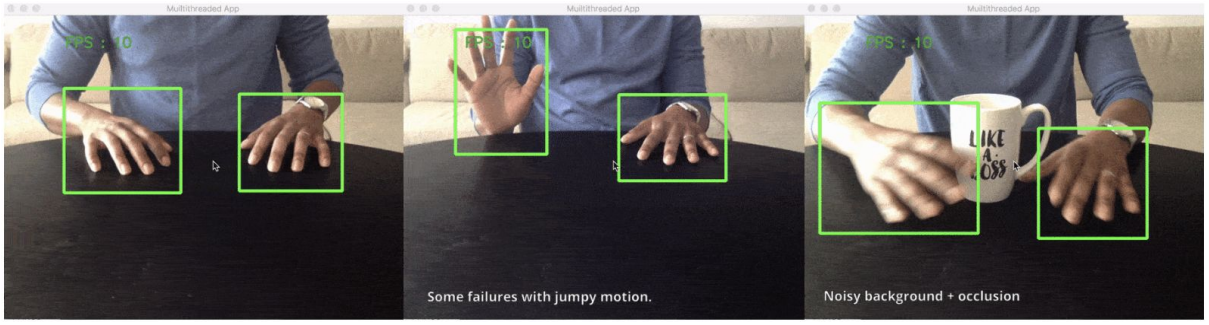
In this chapter we describe different approaches to hand detection and present our solution.

2. 1 Related work

GestureMusicPlayer[1] is an experimental hand detection project with an aim to control a music player using hand gestures. The algorithm relies on a difference in lighting between hand and the background behind it to perform a thresholding on a grayscale version of an original image frame captured from a WebCam. The presented usage example assumes that the background is darker than a hand. Moreover hand is the only presented object within a frame.



handtracking[2] is a Real-time Hand-Detection using Neural Networks (SSD) on Tensorflow. Its goal is to detect and track hands in a video stream. The result of the hand detection is a rectangle that represents an approximate area on a picture with a hand.



[1] https://github.com/antnieszka/GestureMusicPlayer
[2] https://github.com/victordibia/handtracking

2. 2 Problem

The idea behind our *Handy Music Assistant* project is to let musician streamers to control music or sound within their stream using hand gestures combined with voice commands. One of the requirements for this kind of assistant is to be able to detect hand gestures in a live video stream that has a complex multi-colored and non-uniformly lit up background as also a composite background that potentially contains other objects rather than a hand.

2. 3 Process

At first, to detect a hand in a picture we tried to rely solely on OpenCV[3] and its image processing capabilities. An approach similar to GestureMusicPlayer seemed to be a good starting point. The algorithm can be split in the following steps:

- ❏ Get image from camera
- ❏ Convert to grayscale
- ❏ Blur the image to reduce noise (details of the image are not important)
- ❏ Use threshold (low pass filter) to get black and white image
- ❏ Analyze the received shape to find convex hull/convexity defects

We selected Pharo[4] as an implementation language and environment of choice. Pharo is a pure object-oriented programming language and a powerful live, immersive environment, focused on simplicity and immediate feedback.

Getting an image frame

The first step is to capture a frame from a live camera stream. For the purpose of this report we took a non-modified static frame from an iSight webcam integrated into a MacBook Pro.

Once frame is captured and stored in the memory we can start an actual hand detection. In the following sections we will walk step by step through the algorithm and present intermediate results.

[3] https://opencv.org
[4] https://pharo.org

Grayscale with smoothing

Let's convert an input frame to a grayscale image in which the value of each pixel is a single sample representing only an amount of light, that is, it carries only intensity information:

To eliminate noise and unnecessary artifacts we apply a gaussian blur on a grayscale version of the original image:

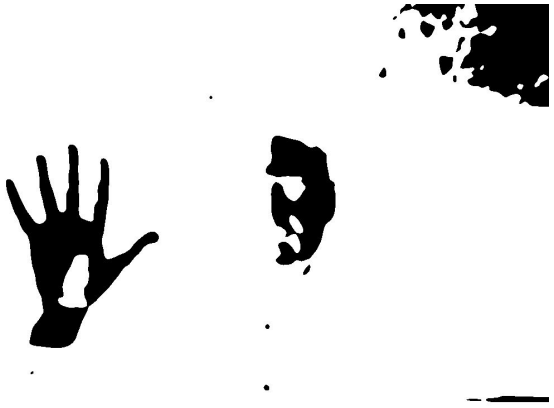
Thresholding

To detect contours in an image we should first convert it to a binary (black and white) image based on some threshold value that defines the magnitude or intensity of light that must be exceeded by a pixel for it to become white, otherwise it turns black.

Global thresholding

The simplest approach would be to manually and statically define a threshold value for the whole image. This method is called Global thresholding. In the GestureMusicPlayer the value is set to 127. Using that value for our picture results in the following black and white version:

As one can see, the hand is not perfectly distinct from the background and we still have noise to the right of it. Manually adjusting the threshold value to 170 gives us a much better result:



However, it means that we should adjust the value each time the environment or hand position changes as it is very difficult to predict the lighting conditions. Another problem of using a *Global Thresholding* method is that it relies only on one parameter, which is light intensity. If background or some other part of the picture is lit up similarly to the hand we will not be able to find a threshold value that would make it possible to separate hand from other objects or background.

Adaptive thresholding

Instead of defining a threshold globally for the whole image we can use individual values for each segment of the picture. This method is called Adaptive thresholding. It gives better results for images with varying illumination, which is a case in real life scenarios[5]:

However, it does not help us as the details in the foreground and background are still present and it is not realistic to extract a hand's shape from such image.

Otsu's method

Another way to solve the problem of a statically defined threshold is to use more advanced adaptive threshold methods, in particular Otsu's method[6]. The algorithm assumes that the image contains two classes of pixels following bi-modal histogram (foreground pixels and background pixels), it then calculates the optimum threshold separating the two classes so that their combined spread (intra-class variance) is minimal, or equivalently (because the sum of pairwise squared distances is constant), so that their inter-class variance is maximal.

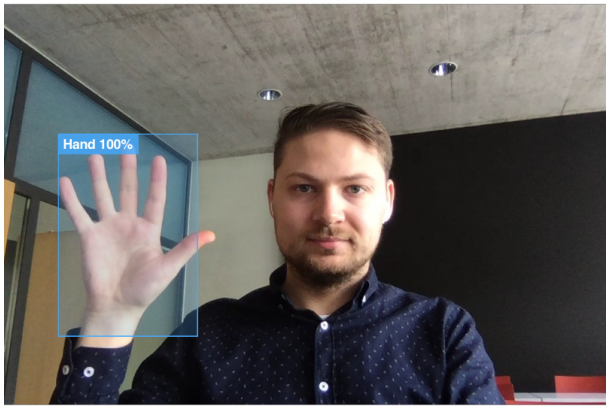
Unfortunately, the composition of the original image is quite complex, hence the Otsu's method can not properly differentiate foreground from the background. It also comes from the fact that there are a lot of other details in the image and the hand contour can not be reliably extracted from the image. Therefore we should find a way to crop the frame so that only hand remains. To accomplish this we can use object detection and classification capabilities of neural networks.

[5] https://docs.opencv.org/3.4.0/d7/d4d/tutorial_py_thresholding.html
[6] https://en.wikipedia.org/wiki/Otsu%27s_method

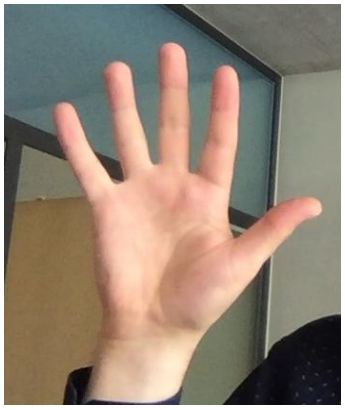
2. 4 Neural Networks (SSD) on TensorFlow.

TensorFlow[7] is a powerful tool and framework for object detection and recognition. Hand tracking is also one of its applications, for example the *handtracking* project focuses on hand detection in an image. It can be used to find an area with a hand so that we can crop the image accordingly. The neural network model that we use is provided by *victordibia*[8].

Let's use it to detect hands on the original image frame:



The neural network successfully detected a hand which is highlighted with a cyan rectangle. In this particular case the detection score is 100% even though the background behind the hand has different colors. However, the area detection is not absolutely precise, which means we can not simply crop the image based on that and we should use a slightly wider rectangle which is computed out of the area given by the neural network:



One of the limitations of this neural network model is inability to tell the shape of the hand, for example how many fingers user shows at the given time. Therefore we have to go back to additionally using Computer Vision to detect an actual shape of the hand. Compared to analysing the whole image frame it is much more efficient and robust to process a cropped image where hand occupies most of the space and is in fact the largest object in the scene. However, we still have to face the problems related to thresholding and contour detection as previously described.

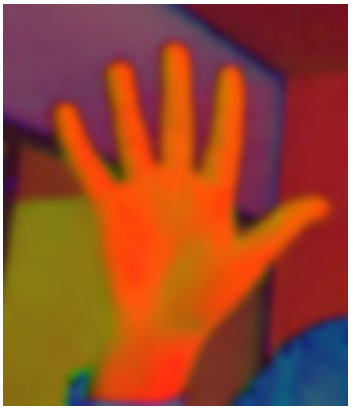
---

[7] https://www.tensorflow.org
[8] https://github.com/victordibia/

HSV color space and skin color

Since we are able to crop a part of the image with a hand on it we can improve thresholding by utilizing the following hand properties. Hand is located in the middle of the image, hand occupies most of the space and it is the largest object and skin color is same across the whole hand. HSV (hue, saturation, value) color space can be used to threshold an image based on the color rather than the lightness. To do so we first convert an image to HSV:

and smooth it out with the help of a Gaussian Blur:



To detect a skin color we select a rectangle of size (image extent / 4) and compute the mean color of all pixels within that rectangle. In our case the color is as follows:

Extracting the mean

Once we know the skin color of the hand in a frame we can transform an image into a binary version be making all pixels within a skin color range white and the rest leave black.

However, because contour detection algorithm detects black shapes on a white background we should invert the image by swapping black and white colors:

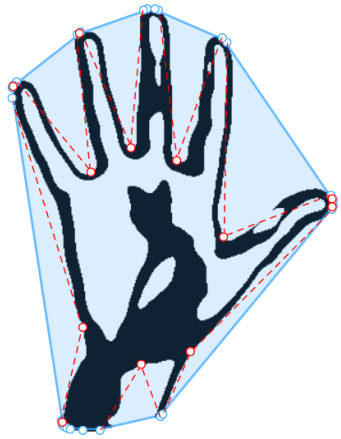
Contour detection

At this point we are ready to run a contour finding algorithm which gives us a list of detected contours that are marked with a cyan color.

Note that there can be multiple contours inside of the hand, therefore we should select the one that goes around the whole hand. Since we know that the hand is the largest object in the cropped image we find the largest contour based on the area:

To count how many fingers a user shows we should analyse the shape of the contour and count how many cavities there are in an object. Such cavities are called convexity defects. To do so we first build a convex hull of the largest contour:

and then use that convex hull to find convexity defects, marked with red dashed lines:



As soon as we got the first collection of convexity defects it became clear that it is not enough to say that the amount of fingers is equal to amount of defects plus one, because there are some defects that are not related to fingers and come from the rest of the hand or from actual form defects due to the image thresholding. However, human hand and fingers have some unique properties that we can rely on to select only interesting defects:

- ❏ The angle between fingers is < 90 degrees
- ❏ Fingers point up
- ❏ Fingers are long

Let's first select defects with an angle less than 90 degrees

And then choose only those defects whose cavity is below the anchor points

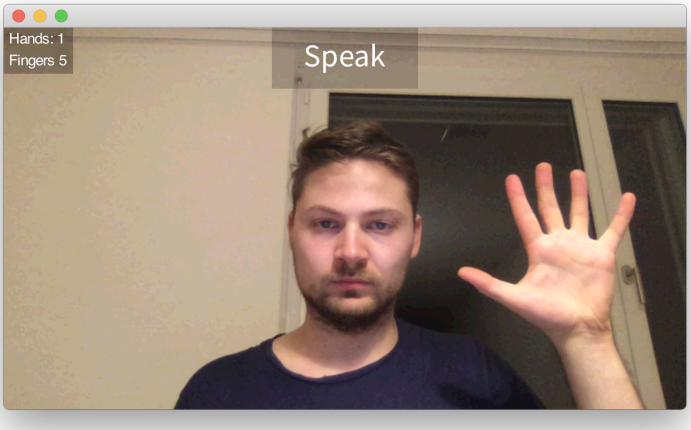
As the last but not least step we count the amount of convexity defects that match finger properties and after adding one we get the amount of fingers shown by the user.

In this example it is 5.

3. Speech recognition with Google Cloud Speech API

Speech recognition from a developer perspective is a more straightforward process. There exist a wide body of libraries providing fluent and easy to use API to transform speech into text. One of these is a python library SpeechRecognition[9]. It collects various speech recognition services under one umbrella and provides a uniform API to works with them. For our project we decided to rely on Google Cloud Speech API[10].

To control a device with voice commands there should be a way to indicate to a voice recognition system when user is about to start saying commands, for example by pressing an activation button or using special activation words such as “Alexa”, “Hey, Siri” or “Okey, Google”. For our project we decided to involve multiple modalities to perform voice commands, for instance we use hand gestures to activate speech recognition. Showing one hand with at least four fingers activates speech recognition and the application asks user to speak a command:



When voice command is recognized the application shows the command name in green. For example saying “Play” results in start playing music command while “Stop” stops playing the music.

The use of hand gestures together with speech to perform commands is an example of a multi-modal user interface that involves two modalities.

[9] https://github.com/Uberi/speech_recognition
[10] https://cloud.google.com/speech-to-text/

To further extend capabilities of the Music Assistant we introduced a two hand gesture to toggle the music. For that a user should show two hands with any amount of fingers:

Having that, we can now start and stop the music by utilising hand gestures (one hand) together with speech or only by using hands (two hands). This gives users without an ability to speak or users in loud environments a chance to use the Handy Music Assistant.

When voice command is not understood a *Try again* error message is shown to the user:

More voice commands can be added in the future.

4. Evaluation

4. 1 Hypothesis

Based on this approach to stop and start music, we state the following null hypothesis.

> ***Hypothesis*** *There is no difference in user performance (time and error rate) when using hand with voice or using two hands to start or stop the music assistant.*

4.2 Variables Definition

4.2.1 Independent Variable

Our hypothesis includes the independent variable that we will manipulate during the experiment independently from the subject's behavior. The independent variable being the gesture and voice. For that purpose, we have prepared an application that uses both gesture and voice. This allows us to easily test both independent variables with the users.

4.2.2 Dependent Variable

We state several dependent variables that we want to measure during the experiment. To ensure exclusive dependency on our independent variable, we state the following performance factors:

- time that JHA takes to detect the hand.
- number of voices (start/stop) required to start or stop the music
- number of errors (JHA can not detect the hand)
- satisfaction score

4.3 Subject Selection & Assignment

For our experiment we chosen the within-group design. So all subjects are assigned to all cases. It helps us isolating the effect of the individual differences and allows the users to better compare the two approaches with the satisfaction score. As our subjects, we selected six people from different age groups, two participants between 20-29 and two participants between 30-39 and two participant between 40-49. For the gender, we have three men and three women. We formed three groups with two participants for each group. This selection and grouping of users helps to eliminate any biases that might stem from subject variability.

4.4 Controlling bias

For each group, we use a different interviewer. Hence, we are reducing the bias stemming from the perception of the interviewer. Furthermore, we prepared a script with the task to be read to the user and review it before the experiment. Therefore, all the interviewers give the exact same instructions - word by word. Questions by the users are not answered during the experiment to prevent the different responses from the different interviewers to effect the results.

4.5 Statistical Analysis

Since we used the within-group design, we utilized the two-tailed paired-samples t-test to evaluate our results on a 0.5 significance level. The statistical analysis is conducted for each dependent variable first across all participants but we also calculate it separately for each group.

5. Conclusion

In this report we presented our *Handy Music Assistant* which with the help of the multi-modal interface involving two modalities helps users to control a music player. Hand and finger detection was a challenging problem from both technical and conceptual perspectives. Even by relying on unique hand and finger properties we were not able to detect the amount of fingers shown by the user with high enough precision to assign different commands to different amount of fingers. Instead we were able to recognize that user shows few (<= 3) or many fingers (>=4). This gave us a chance to utilize hand and finger detection to activate speech recognition. Having an ability to recognize the amount of hands shown at the moment we also assigned a command to the two hand gesture. This gave us a chance to compare two ways of controlling the music player: by using one hand with voice and by using two hands. The use of multiple modalities at the same time may at first be slightly confusing to the inexperienced user, however it gives a better flexibility to the set of available commands and proposed features. Multiple modalities allow developers to provide a wide range of commands, since adding a new modality increases the amount of possible combinations. In addition to that, relying on multiple modalities such as one hand and voice appeared to be more robust and less error prone compared to just using two hands.