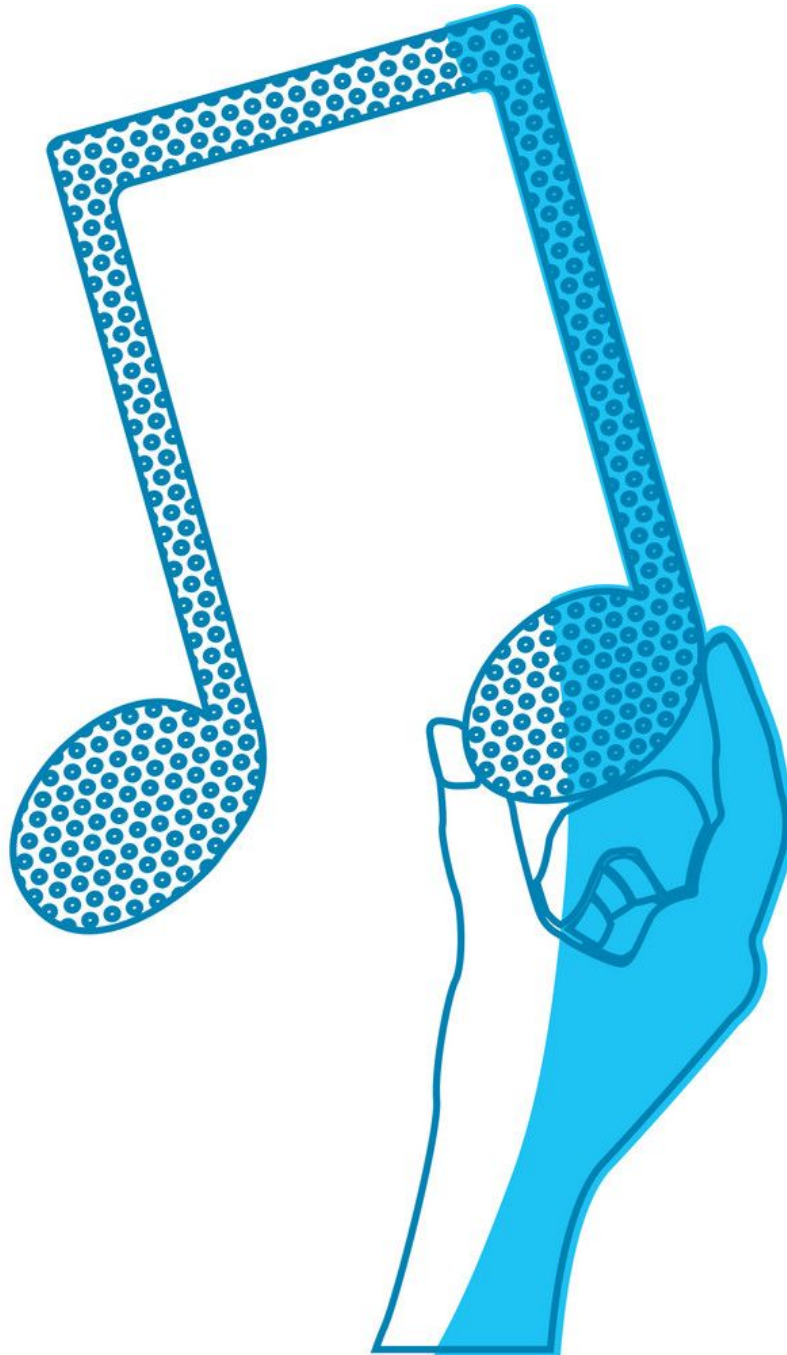


Gestify - The hand gesture music controller

The ideal music controller that swipes all your controlling problems away



[1]

Table of contents

Introduction	1
Literature Study	2
Concept of Operations	3
System Requirements	5
Use Case Specifications	7
System Design	9
Components	9
Features	9
Classifier	10
Program overview.	10
Code Overview	12
The music player.	12
Getting the right coordinates	14
Image processing.	14
Classifier	16
Project Plan	18
Testing	19
Evaluation	21
Appendix	22
References	28

Introduction

Music plays an integral and essential role in our life. Some of us can not live without it and we have come to a point where many people are used to streaming and listening to music during the study time while having dinner and many other moments. However, we still come across many practical issues when we want to control our music. For example, when we want to skip to the next song in our playlist, we have to grab either our phone, remote control, or press buttons on the device that's currently playing our music. This might not be a problem with the expensive headphones that we currently have, but there are tons of devices (e.g. soundbars, hi-fi speakers, etc.) for which performing these tasks can be very annoying. Additionally, our current systems do not allow visually impaired people to simply control their music. Is it really necessary to search for the remote control and awkwardly read the braille on each button until he/she finds the correct one? This is definitely not the most efficient way to control your music.

The idea of creating a new way to control music that is more efficient and accessible for a wider range of people was our motivation to start working on this specific project.

The first concept that probably comes to mind is *voice recognition*. Voice recognition has been an innovative and revolutionary idea that has advanced exponentially over the past few years. However, dedicating our project to a concept that already exists and has been researched thoroughly did not sound like a great idea to us. Part of the goal of our project is to create an innovative system that could be further developed and applied in real life. To us, using voice recognition sounded more like walking in the steps of someone else. Additionally, the voice recognition we develop with our current knowledge of (pre-)processing and classifying sound might recognize the music as voice commands. For example, the music may unintentionally pause when a musician says 'hold up' or 'stop' during a song. Furthermore, of course, mute people would not be able to use our voice recognition.

So, we thought of another innovative method that we could develop to control music, that is available for a broader spectrum of people and achievable with our current physical computing skills: *gesture recognition*. We wanted to develop a system that's able to recognize and interpret movements of hands in order to interact with and control music players without direct physical contact. This way, users can simply control their music without having the mentioned problems that voice recognition has.

We do not have the intention to replace our current system of controlling music. Our controller, in combination with some extra hardware, could be used as an additional feature and could live in harmony with many of our modern music players. Users would simply have to turn our 'feature' on or off and do not have to make any other changes. This way, the system brings only benefits when used. Our idea could easily be implemented in IoT (Internet of Things) or Smart Home systems to assist in everyday life.

Literature Study

Just like any other project, ours began with literature research. The knowledge of processing, classification, and actuation that we gained during the Lab sessions contributed a lot to our project, but, unfortunately, this project required more skills. We were all relatively new to the subject of gesture classification and realized that it was necessary to study literature in order to learn about the ways how other groups with similar projects had tackled their problems. Before we actually started searching for these similar projects, we first listed the following critical questions that needed to be answered in order to successfully execute our project:

- Would it be easier to classify distinct hand movements or the shape of hand gestures with the knowledge we currently have?
- Which features should we use to classify our images? And what are their limitations?
- Which type of classifier works best for classifying hand gestures? And what limitations does it have?
- What is the most efficient way to distinguish between the background and the hands during repeated image acquisition, where the background might slightly change?
- How do we create a simple music player to test our software on?
- Which tools (sensors) should we use for gesture recognition?
- Which hand gestures should we use?

The first book we read, “Human Computer Interaction Using Hand Gestures” by Prashan Premaratne (from the VU library), provided us some useful insights on how to successfully classify hand gestures by using many techniques similar to the ones we have already learned throughout this course [2]. However, this book summarizes the writer’s sustained research in this area for the past 10 years, which is noticeable in the degree of complexity in each sequence of steps of his program (segmentation, classifying, etc.). Nevertheless, this book still answered most of our questions and helped a lot during the development of our project. It gave a comprehensive in-depth explanation of different ways to record the hand gesture by either using regular cameras, depth-aware/stereo/Radar cameras or data gloves such as the MIT Data Glove and the CyberGlove series.

The writer put a great effort into explaining how skin can be optimally detected and how to preprocess and extract the features from the acquired images. Last but not least, the writer described numerous types of classifiers (linear and nonlinear) that he used to recognize the gestures.

Despite the large amount of information from our first source of knowledge, we still wanted to read from one more source in order to obtain a broader understanding of the problem that we are facing with gesture recognition. The next paper we read, written by students from Atharva College of Engineering (2016), gave us this extra insight [3]. This paper introduced a completely different way of classifying the gestures. Instead of using complex machine learning techniques, they used a rule-based classifier that looks at features like orientation, the center of mass (centroid), the status of fingers, thumb in terms of raised or folded fingers of the hand and their respective location in the image. They found a way to identify and count the number of fingers visible in the hand gesture, which made it possible to work with a different set of gestures than the previous paper we read.

Now, there was only one more challenge left to look into: creating our own music player. For that reason, we read several articles on MathWorks, a website that documents and explains all built-in features in MatLab [4]. For our case specifically, we looked into the built-in audio player of MatLab.

Concept of Operations

As we have already explained in the introduction, our goal and objectives are to build a system that's able to recognize and interpret movements of hands in order to interact with and control music players without direct physical contact. The controller will recognize gestures that correspond to the following four most-used operations: playing the next song, playing the previous song, starting & pausing the music, and exiting the program. This way, we solve the problems many music listeners have with the current way we control our music, possibly because they're visually impaired or mute, or because they do not like the system in which they have to press buttons or grab their phones for very simple tasks. These people will form the biggest group of stakeholders involved in our project, followed by the clients, sponsors, all responsible technical persons, and managers. Since voice recognition has become a popular and affordable alternative, our stakeholders would require cost-effectiveness, low manufacturing costs, and high reliability.

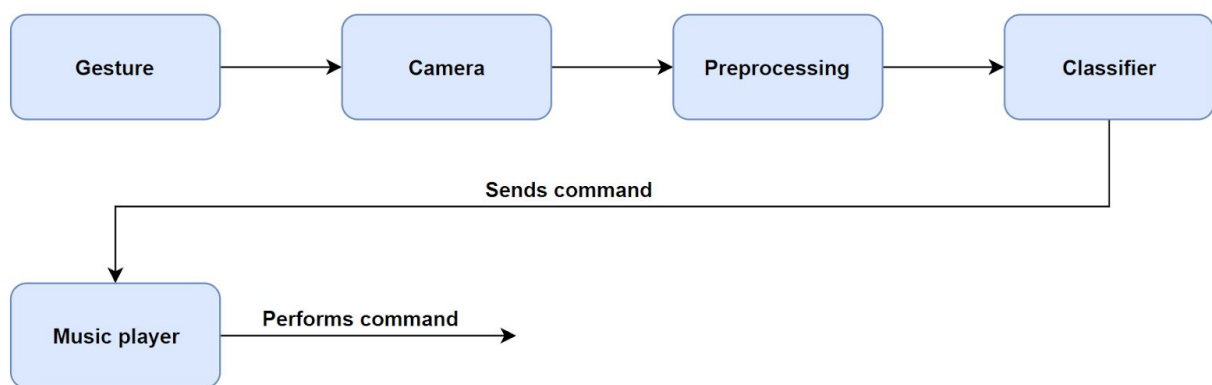


Figure 1: *The block diagram of the project, showing the connections between the different components.*

The following examples give a very good illustration of situations where both visually impaired and normal people benefit from using our system.

Normal person #1

It is a late Sunday night and a student is studying very hard for his exams next week. In order to stay focussed and amused, the student has some lo-fi hip-hop playing in the background. While the student is trying to focus on solving example questions, one of the songs breaks his concentration and he wants to skip this song. But, because he is studying, he left his phone in one of the other rooms to prevent himself from wasting time on social media. So, instead of taking the risk of waking his parents up by walking around in the house this late or wasting time on his phone, he decides to turn Gestify, our music controller, on, which is connected to the device that is currently playing the music. Now, he only has to make the right hand gesture and the next song starts playing, saving him from a lot of work and potential problems.

Normal person #2

It is a Sunday night and a father has a nice family dinner with his wife and kids. Entertaining background music is quietly playing on the TV in the background, but suddenly a really inappropriate or undesired song starts playing. While having dinner, it is not appropriate to use phones and the father does not want to stand up, walk all the way to the television and press the 'next' button on the remote. Instead, he decides to turn Gestify on, which is connected to the television, possibly through a smart home network. He could also have turned on Gestify before they started dinner. Now, he only has to raise his hand, make the right hand gesture, and the next song starts playing, saving him from a lot of work and potential problems.

Visually impaired person

Imagine, a visually impaired person who really enjoys music plays music very frequent at home. Because the person is visually impaired, the only way to control his music is by pressing buttons. If they find their remote, which is already a great difficulty, they still face the problem of having to remember the location and meaning of each button. Or, they have to feel the braille-text on each button until they find the right one. This is really inefficient and could easily be solved if he turns Gestify on, or have it always turned on while playing music for convenience. This way, the person only has to remember the correct hand gesture for each operation and all he has to do is perform the right gesture to control his music.

We truly think that our idea will help a lot of people out and encourage many individuals to listen to music by introducing a fun way to play with it.

System Requirements

In our project, we have prioritized our requirements and constraints by using the MoSCoW approach. Although our system does not have any influence on factors such as privacy or safety, we still felt like this is an important factor for our project development. Out of all software development methods we learned, we chose to follow the Agile approach. This made it very easy to constantly finish a new version of working software in each sprint and correct errors throughout the process. Many people complain that the disadvantage of the Agile approach is that there is no long-term plan for developing and testing the product, and the documentation is not extensively maintained. That's why we chose to create a flexible plan at the beginning of the process. We agreed that we were not required to follow this plan strictly: it was more like a guideline. After each weekly sprint, we made sure that we documented all new developments to prevent an insufficiently maintained documentation. Our goals for each sprint can be found in the Project Plan section.

The use cases are graphically represented in UML using a use case diagram (figure 2), in addition to the state transition diagram (figure 3). In this diagram, the primary actor is the user of Gestify, and the camera is the secondary actor.

Must have	Should have	Could have	Will not have
<p>The system must be able to execute the following commands by recognizing hand gestures:</p> <ol style="list-style-type: none"> 1. Start/pause a song 2. Play the next song 3. Play the previous song 4. Exit the program <p>The system must recognize a hand gesture with a camera with an accuracy of at least 80%.</p> <p>The system must not store any personal information about system users like their names and the songs they've played.</p> <p>The system must not stop running if the user did not tell the system to do so. This means that no gesture may be misclassified as a program exit gesture.</p> <p>The system must wait a couple of seconds after executing the operation indicated by the user's gesture to make sure that the operation does not execute more times than intended.</p>	<p>The system should also recognize hand gestures that have not been specified and execute the 'none' operation, which does not do anything, as a result.</p> <p>The system should be able to process pictures with background noise.</p> <p>The production costs of the system should be under \$10.</p> <p>When one of the gestures is misqualified as 'none', no change will occur in the music player. Because this is much more preferable than executing a wrong command, in case of misclassification, there should be a chance of at least 60% that the gesture is classified as 'none'.</p>	<p>The system could be improved by also implementing volume control operations.</p>	<p>Our system will not have additional components like ultrasound sensors or microphones.</p> <p>Our system will not have features that recognize who the hand belongs to.</p>

Table 1: *the system requirements following the MoSCoW approach*

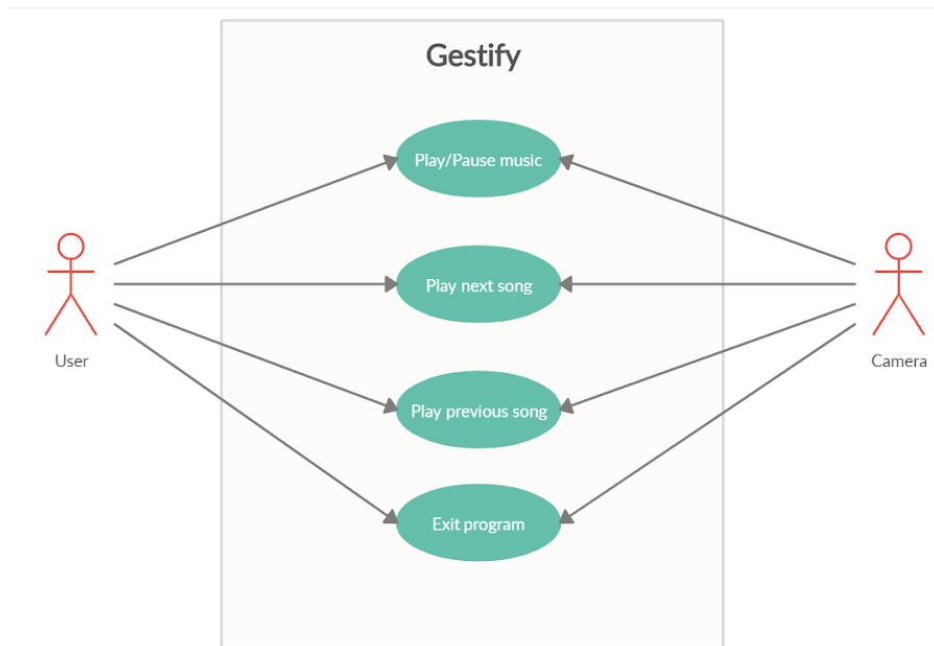


Figure 2: *the use-case diagram*

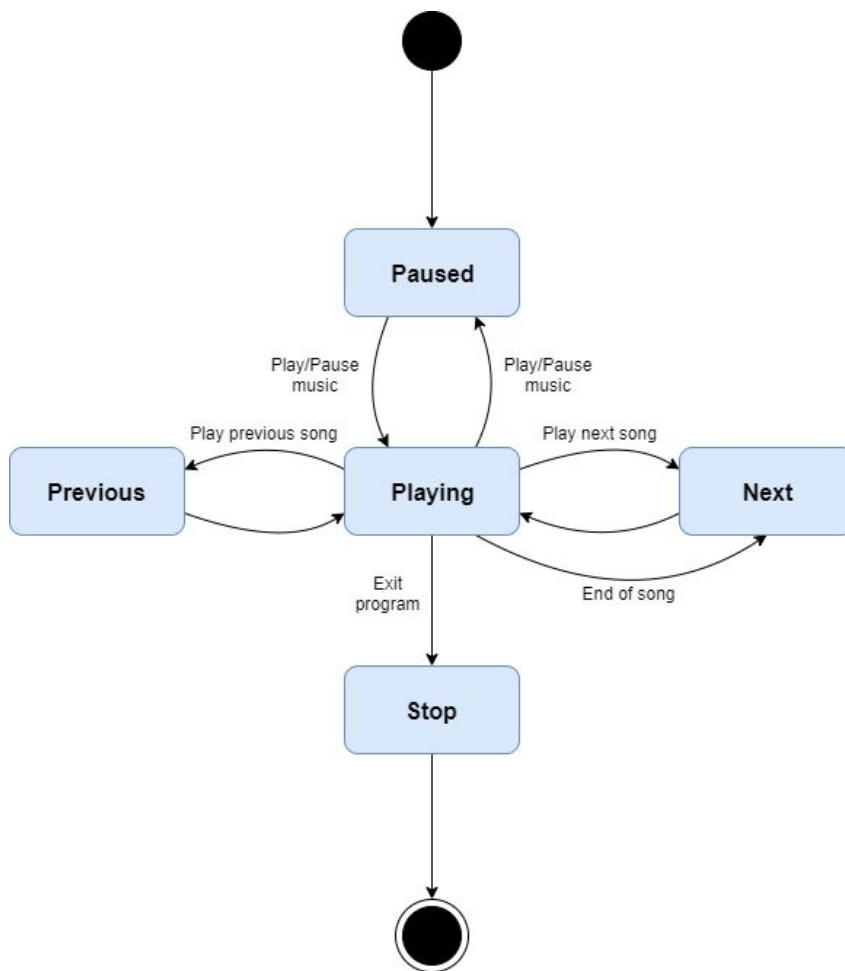


Figure 3: *the state transition diagram*

Use Case Specifications

Name:	Play/Pause music
Short description:	The music player will start or pause the song
Precondition:	The camera is connected and functional, and the user performs the right gesture
Postcondition:	The music player pauses in case music was already playing and starts in case the music was paused. Afterward, it is ready to receive a new command
Error situation:	The classifier misclassifies the 'play/pause' gesture
System state in the event of an error:	The music player receives the incorrect command and executes it
Standard process:	<ol style="list-style-type: none"> 1. The user uses the correct hand gesture to play/pause the music 2. The camera takes a picture 3. The picture gets preprocessed 4. The classifier successfully classifies the gesture and sends the matching command to the music player 5. The music player executes the command
Alternative process:	<ol style="list-style-type: none"> 4. The classifier misclassifies the gesture and sends the wrong command to the music player 5. The music player executes the wrong command

Name:	Play next song
Short description:	The music player will play the next song
Precondition:	The camera is connected and functional, and the user performs the right gesture
Postcondition:	The music player will stop the song that's currently playing and start playing the next song. Afterward, it is ready to receive a new command
Error situation :	The classifier misclassifies the 'next' gesture
System state in the event of an error:	The music player receives the incorrect command and executes it
Standard process:	<ol style="list-style-type: none"> 1. The user uses the correct hand gesture to skip to the next song 2. The camera takes a picture 3. The picture gets preprocessed 4. The classifier successfully classifies the gesture and sends the matching command to the music player 5. The music player executes the command
Alternative process:	<ol style="list-style-type: none"> 4. The classifier misclassifies the gesture and sends the wrong command to the music player 5. The music player executes the wrong command

Name:	Play previous song
Short description:	The music player will play the next song
Precondition:	The camera is connected and functional, and the user performs the right gesture
Postcondition:	The music player will stop the song that's currently playing and start playing the previous song. Afterward, it is ready to receive a new command
Error situation:	The classifier misclassifies the 'previous' gesture
System state in the event of an error:	The music player receives the incorrect command and executes it
Standard process:	<ol style="list-style-type: none"> 1. The user uses the correct hand gesture to play the previous song 2. The camera takes a picture 3. The picture gets preprocessed 4. The classifier successfully classifies the gesture and sends the matching command to the music player 5. The music player executes the command
Alternative process:	<ol style="list-style-type: none"> 4. The classifier misclassifies the gesture and sends the wrong command to the music player 5. The music player executes the wrong command

Name:	Exit program
Short description:	The program will terminate
Precondition:	The camera is connected and functional, and the user performs the right gesture
Postcondition:	The music player will stop the song that's currently playing and the program will terminate. It is not ready to receive a new command but can be turned on manually again.
Error situation:	The classifier misclassifies the 'exit' gesture
System state in the event of an error:	The music player receives the incorrect command and executes it
Standard process:	<ol style="list-style-type: none"> 6. The user uses the correct hand gesture to stop the program 7. The camera takes a picture 8. The picture gets preprocessed 9. The classifier successfully classifies the gesture and sends the matching command to the music player 10. The music player executes the command
Alternative process:	<ol style="list-style-type: none"> 4. The classifier misclassifies the gesture and sends the wrong command to the music player 5. The music player executes the wrong command

System Design

At this time it should be clear what our music controller should be able to do, but how are we going to accomplish this task? In this section, we will discuss how our system is built in terms of components, algorithms, features and what difficulties we encountered.

Components

Our system does not require a lot of extra components to be implemented, because it relies mostly on our algorithms and computing power of the PC and we use only a few external devices for data acquisition and actuation. The following list shows all the components that we used to build our system:

- A computer that runs our program (on MatLab)
- A Microsoft LifeCam camera
- Speakers / Headphones

These are the components we chose to work with, but that does not mean that the future product in which this project could later result in has to be limited with these exact components. Each component is replaceable, which makes this project truly unique. The future user should, for example, not be restricted from using a tiny microcontroller board with a different camera attached to it, which would be much more practical, instead of using a personal computer and the LifeCam camera.

In our system's setup, we are also using a black square made out of tape to indicate the location where the users should place their hands.

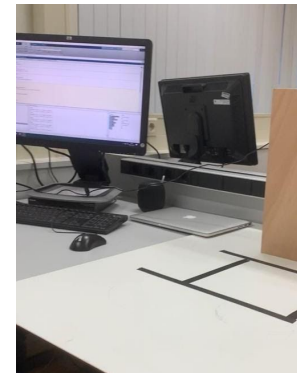


Figure 4: *system setup*

Features

To be able to successfully classify and distinguish between our four hand gestures, we extract three features from the BLOB's we obtained after processing our images. These features are the form factor, solidity, and the extent.

Because we have five fingers on each hand, it is really easy to change the form of our hands. Even lifting one finger in a gesture makes a noticeable change in the form factor. Therefore, it was very easy to differentiate between gestures based on the increase/decrease of the form factor. Other features (like area, perimeter, etc.) were too similar for each gesture. Moreover, if a person holds the camera closer or further away from his/her hands, a classifier based on the area could wrongly interpret the gesture. The same argument can apply to people with different hand sizes. So, just by using the form factor, we were already able to distinguish the gestures from each other.

Additionally, we found that features like solidity and extent helped us improve the classification and get more accurate results. The solidity is defined as the ratio of the hand area and the convex hull area¹. The extent is considered as the ratio of the hand area and the area of the bounding box. By using these two additional features, we were also able to differentiate between a 'known' gesture and an 'unknown' one. This means that we could recognize gestures that weren't originally assigned to one of the music player operations and do nothing when we see these.

¹ [Convex hull](#) area can be viewed as the area of a polygon where the BLOB is enclosed.

Classifier

Because we are only using three features in our classification, it is not necessary to implement complex machine learning algorithms to classify our hand gestures. In the literature we read, using Neural Networks was a popular choice, but we experienced that it was not essential to make our program work. Therefore, we chose to adopt a rule-based classifier. The classifier divides the hand images into five classes - one for each gesture and one for the not-valid gestures or the gestures our program could not classify referenced as the “none” class. As mentioned above, the form factor had a major role during the classification process. Extern and solidity had a minor role for classifying the four gestures but by including them into our classifier we managed to discard not valid gestures.

Beyond our expectations, the classifier actually performed well enough to get a score of 88% on the testing confusion matrix (see Testing).

Hand gestures

Based on the distinct features of each hand gesture, we chose to use the following hand gestures to control the music player.





Use-case	Play/Pause music	Play next song	Play previous song	Exit program
Hand gesture				

Table 2: The hand gestures assigned to each music player command [5]

Program overview

The workflow of our application is graphically displayed on the next page (figure 5) and can be summarized into the following steps:

1. Start the video feed of the camera
2. Load the music files into the music player
3. Get the coordinates of the box area defined by the black tape after taking a snapshot of the background
4. A Loop where the program:
 - 4.1. Gets an image (snapshot) from the camera every 0.3 seconds
 - 4.2. Crops the image with the coordinates from step 3
 - 4.3. Processes the acquired image
 - 4.4. Classifies the acquired image
 - 4.5. Send a command to the music player if a valid gesture is found within the image.
 - 4.6. After sending the command wait for 2 seconds for the user to remove his/her hand.

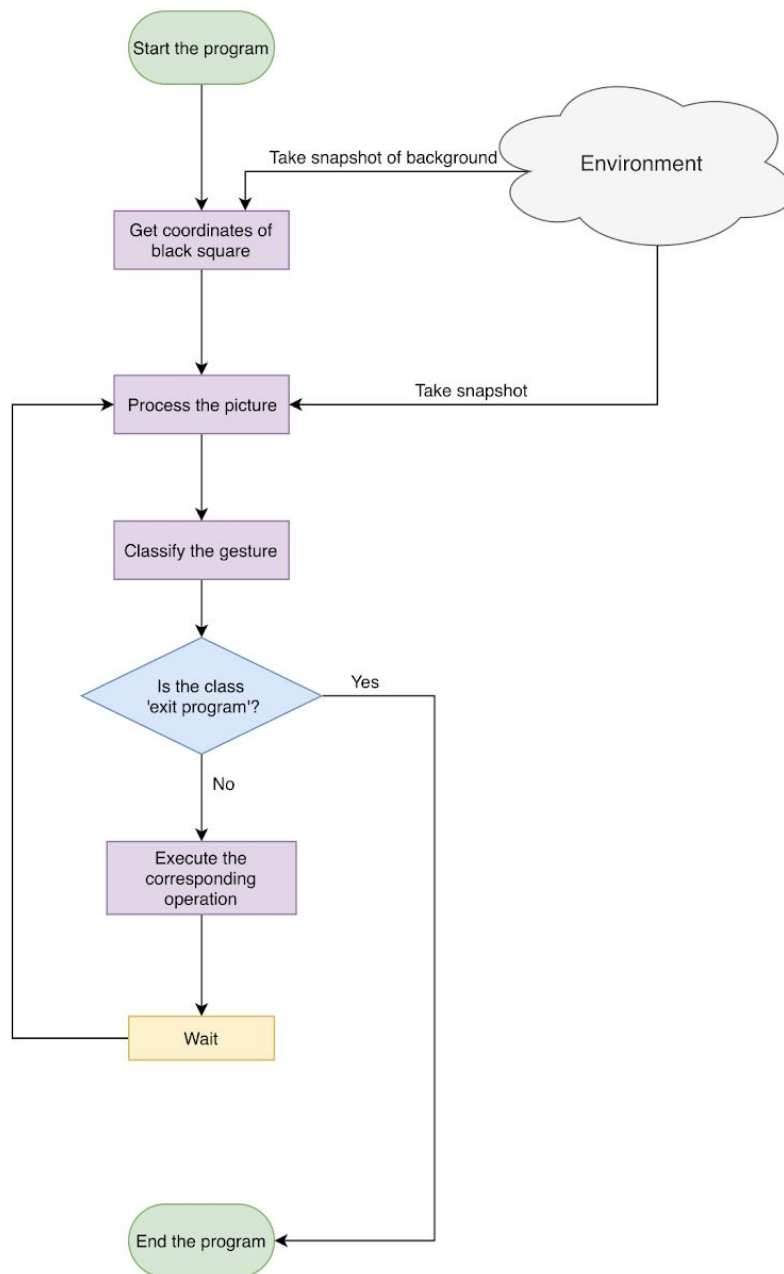


Figure 5: *Flowchart of our basic system design*

Code Overview

In this section, the overall code and each function will be explained in-depth. The code can be found in the appendix. Before proceeding to the explanation of each function, it is essential to understand how each component of our software interacts with each other.

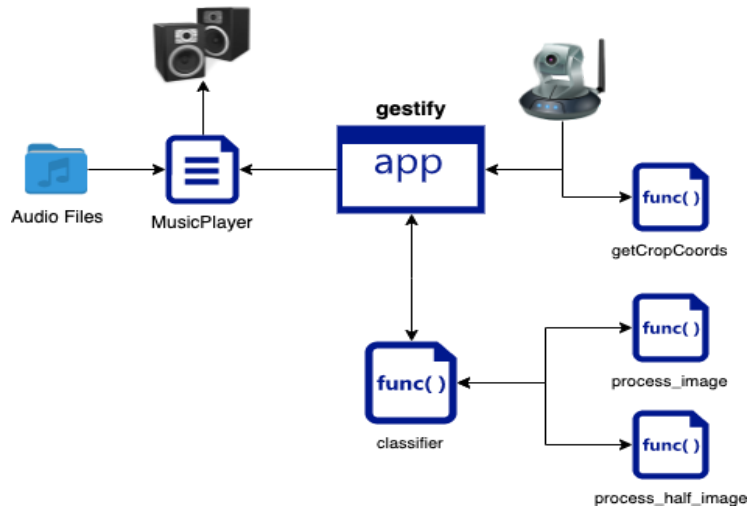


Figure 6: Hardware/software overview

First of all, the file *gestify.m* serves as the main function of our application and it is the coordinator between the classification process and the music player as explained at the program overview. When we start our program, *gestify.m* starts running. It creates the *MusicPlayer* object, a video object to get a constant video feed, and also calls the *getCropCoords.m* function to get the coordinates of the black box area defined by the black tape.

Each time after the system takes a new snapshot from the video feed, it calls the *classifier.m* function with the image acquired from the camera feed as an argument. The function returns the class the image belongs to. Finally, the main function sends the appropriate signal to the *MusicPlayer* object, which is the one interacting with the speakers/headphones.

The *classifier.m* function is the one that interacts with the image processing functions. First, it calls the *process_image.m* function, which tries to find a single BLOB (the hand) in the image. Then, it extracts the features of the hand and classifies the gesture in the image by using a rule-based classifier. The *process_half_image.m* function, which we will explain later in this section, is only called by the *classifier* if the BLOB from the *process_image* function cannot be classified in a correct way and requires further processing to determine the hand gesture.

The music player - *MusicPlayer*

The *MusicPlayer* class is a relatively simple class that serves as a wrapper for MatLab [audioplayer](#). It implements the four basic features of our music player: play, pause, play the next song, play the previous song. For these features, the following public functions are used: start, stop, nextSong, previousSong.

MusicPlayer
<ul style="list-style-type: none"> - player: audioplayer object - musicDB: String Array - currentSong: Integer - paused: Boolean
<ul style="list-style-type: none"> + MusicPlayer() + start() + stop() + nextSong() + previousSong() + isPlaying(): Boolean - loadMusicDB() - randomSong() - stopCallback(obj,~,~)

UML Diagram

The private properties of the class are:

- player - Audioplayer object
- musicDB - Array with the path of songs that are included in the music directory
- currentSong - The index of the current song in the MusicDB
- paused = true; - The state of the player as boolean - Default: True

Below there is a brief explanation of each method:

Constructor

Creates the MusicPlayer object and calls the loadmusicDB and randomSong methods.

start

It starts or resumes the music player and sets the state of the player as *paused = false*;

stop

It pauses the music player and sets the state of the player as *paused = true*;

nextSong

The player loads the next song and plays it. This is achieved by accessing the next element of the musicDB array. If the song before the nextSong command is executed was the last one in the musicDB array then the player goes to the first song of the array, this functionality is achieved by using the mod operation.

previousSong

The player loads the previous song and plays it. This is achieved by accessing the previous element of the musicDB array. If the song that was playing before the nextSong command was executed was the first song in the musicDB array, then the player goes to the last song of the array. This functionality is achieved by using the mod operation.

isPlaying

Returns the state of the player as boolean.

loadMusicDB

Loads the mp3 files from the music directory into the musicDB array as strings indicating the path where the mp3 files are.

randomSong

Loads a random song into the player object with the use of the MatLab *randi* function.

stopCallback

This is a callback function from the MatLab [audioplayer](#). It is triggered every time the player stops for any reason. By comparing the state of our MusicPlayer, we can determine if the callback call was caused by an action of the user or because the song ended. In that case, we need to load and start the next song.

Getting the right coordinates - *getCropCoords.m*

As we already explained in the system design, the first, but essential, step of our application is to locate the area within the black box made out of tape. The *getCropCoords.m* function does exactly this. It takes an image of the background (without the hand of the user) and finds the coordinates of the left-top pixel of the black square, as well as the width and height of the box. Then it returns this information to the main function (*gestify.m*) for later use. The method for finding this pixel is easy. First, we have to convert the image from RGB to grayscale and then the grayscale image to a binary image (black & white), by using the global thresholding technique we learned in Lab 3 of the physical computing course.

The pixel we are looking for has the following properties: (1) It has to be white (2) The pixel next to it has to be black (3) The pixel above it also has to be black. Our algorithm simply checks every pixel within the first one-third of the image with these properties.

To calculate the width of the white box after finding the coordinates of the pixel which indicates the top-left point of the white box, we simply implemented a loop that checks every pixel in the same row as the top-left pixel. In case if a black pixel is found, it means that the border of the black tape is found. The width is the number of iterations it took for the loop to finish. By applying the same logic, we check every pixel in the same column as the top-left pixel until we find a black one.

In the end, the function *getCropCoords.m* returns the values of the coordinates of the top-left pixel, as well as the height and the width, so it can be used to crop the images taken from the video feed with the MatLab function *imcrop*.

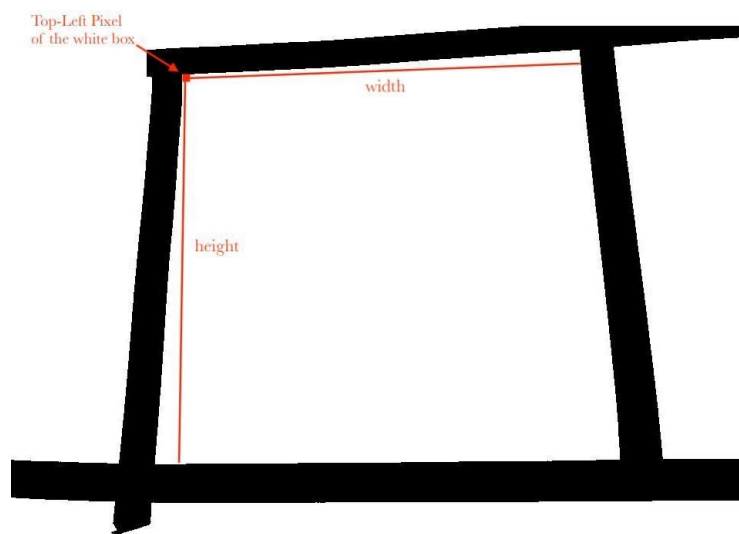


Figure 7: Overview of the black square-detection process

Image processing - *process_image.m*

Image processing is an important operation of our application. The success of gesture recognition depends on how well we process the image. If the image is processed correctly, then the classifier should be able to classify each gesture correctly. The function *process_image* tries to find a single BLOB inside the image. If it succeeds, it extracts the features of the hand, as described in the “Features” section.

If the algorithm does not detect a single BLOB in the image, all features are set to 0. The main problems we faced were the shadows created by the hand itself and the changing illumination the room had. To tackle these problems, we tried to use local adaptive thresholding instead of global thresholding.

Adaptive Thresholding²

Just like in global thresholding, we first need to convert the acquired RGB image to grayscale. In comparison to global thresholding, where one value is picked by an algorithm as a threshold value for the whole image, in local adaptive thresholding, each pixel has its own (threshold) value based on the range of intensity values of its neighborhood pixels. To determine this threshold value for each neighborhood, we use the mean of the minimum intensity value and the maximum intensity value. In order to succeed with this technique, the neighborhood size needs to be large enough to cover both background and foreground pixels.

In our algorithm, we try to use different neighborhood sizes (referenced as windowSizes in the code) by using a loop to find a suitable neighborhood size where the hand is considered a single BLOB. In each image resulted from local adaptive thresholding, we also apply a mild morphological correction. After each iteration, the window size (so also the neighborhood size) increases. If a single BLOB is found in the new window, we stop the loop and proceed to calculate the features of the hand gesture.

In figure 8, the difference between the local adaptive thresholding and the global thresholding in our images is shown. When we use adaptive thresholding, the shadows are separated from the hand, while global thresholding still recognizes shadows as being part of the hand.

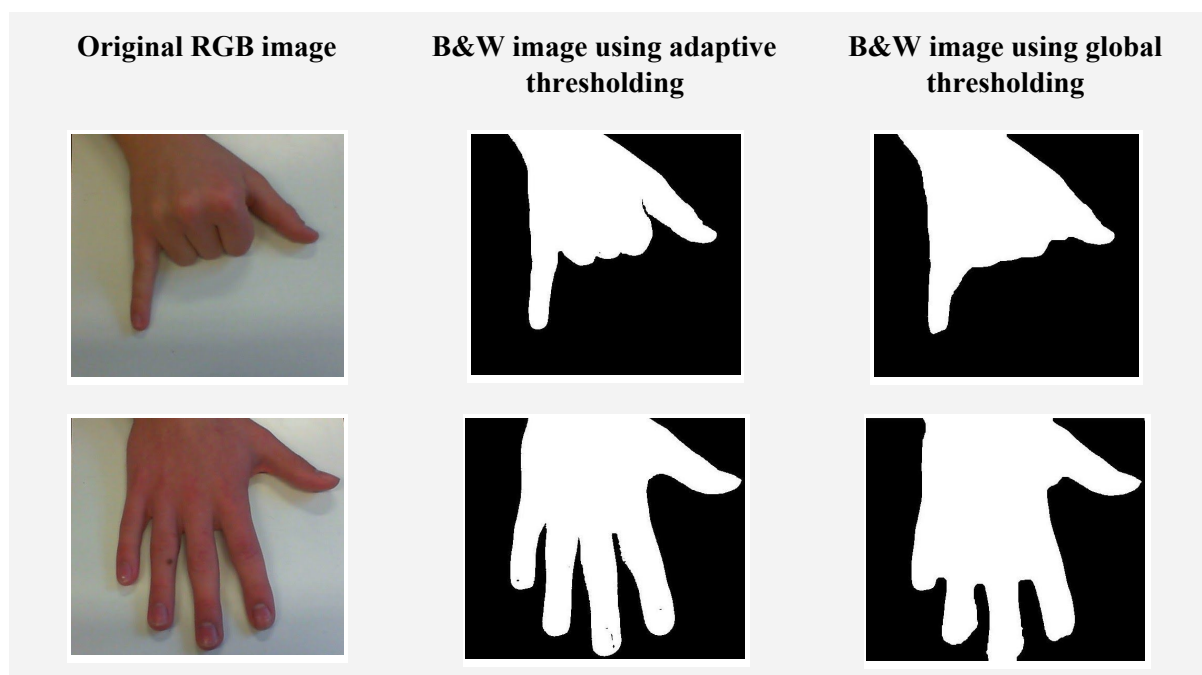


Figure 8 - Differences between adaptive and global thresholding

² More about adaptive thresholding: <https://people.scs.carleton.ca/~roth/iit-publications-iti/docs/gerh-50002.pdf>
<http://homepages.inf.ed.ac.uk/rbf/HIPR2/adpthrsh.htm>

Classifier - *classifier.m*

As we already mentioned in the System Design section, we used a rule-based classifier to classify the images into 5 classes {open hand, fist, L-sign, Y-sign, None}. The *classifier.m* function is responsible for this classification. It calls the *process_image.m* function, which returns the features of the hand. Then it proceeds to classify each gesture according to the following table:

	Open hand (Play/pause)	Fist (Exit program)	L-Sign (Next Song)	Y-Sign (Previous Song)
Form Factor	0.12-0.25	0.65-0.82	0.26-0.37	0.30-0.42
Solidity	-	-	0.65-0.75	0.69-0.80
Extern	-	-	0.40-0.50	0.40-0.54

Table 3: Classification Table

The function uses if-else statements to implement the above table correctly. After the image is classified, the function sets the sting variable *sign* to “exit”, “start_stop”, “next”, “previous” or “none”, depending on the class of the gesture. At the end, it returns this variable to the main function, which acts on it and controls the music player.

By observing the classification table, it is obvious that some values for the L and Y sign are overlapping. More specifically, when the form factor value is between 0.3-0.37, the solidity value is between 0.69-0.75 and the extern value is between 0.4-0.5, the existing classifier cannot determine which gesture is within the image. When the features range between these values, we need to re-process the image in a different way to get more features of the gesture.

Resolve conflict for L & Y Sign - *half_image_proccess.m*

This function is only used when all the feature values for the L and Y sign are overlapping. The *half_image_proccess.m* function takes the previous processed black & white image as an argument, cuts it in half and reprocess the left and the right part of the image (see figure 9). The idea is that we will be able to distinguish the L and Y signs by comparing the form factor of the left side to the form factor of the right side. In case we have a L-Sign, the difference between the form factors of the two subimages is quite similar and it has relatively high values because in each sub-image only one finger is away from the “hand body”. However, when we look at the L-sign, the form factor of one side has very low form factor values, whereas the other part can be viewed as a fist gesture resulting in high form factor values.

To make sure the hand gesture is perfectly cut in half, we crop the image according to the coordinates of the *bounding box* the hand BLOB has. Because we are dealing with an already processed black & white image with a single BLOB in, there is no need for further processing. After calculating the form factor of the left and right side, the function returns these values to the *classifier* function.



Figure 9: Illustration of the reprocessing where we create the left & right sub images

Classifier (Extended)

To be able to classify this new data, we look at the absolute value of the difference between the two form factors. This way, the classifier would work properly for both the left handed or the righted users. If the absolute difference is below 0.1, then the gesture is classified as Y-sign. Else, if the absolute difference is above 0.2, the gesture in our image it is the L-sign. Finally, if the absolute difference between the two form factors is above 0.1 but below 0.2, we determine the gesture by looking at the two form factors individually. If one of them is below 0.31 the gesture is classified as the L-sign, else the gesture is classified as the Y-sign.

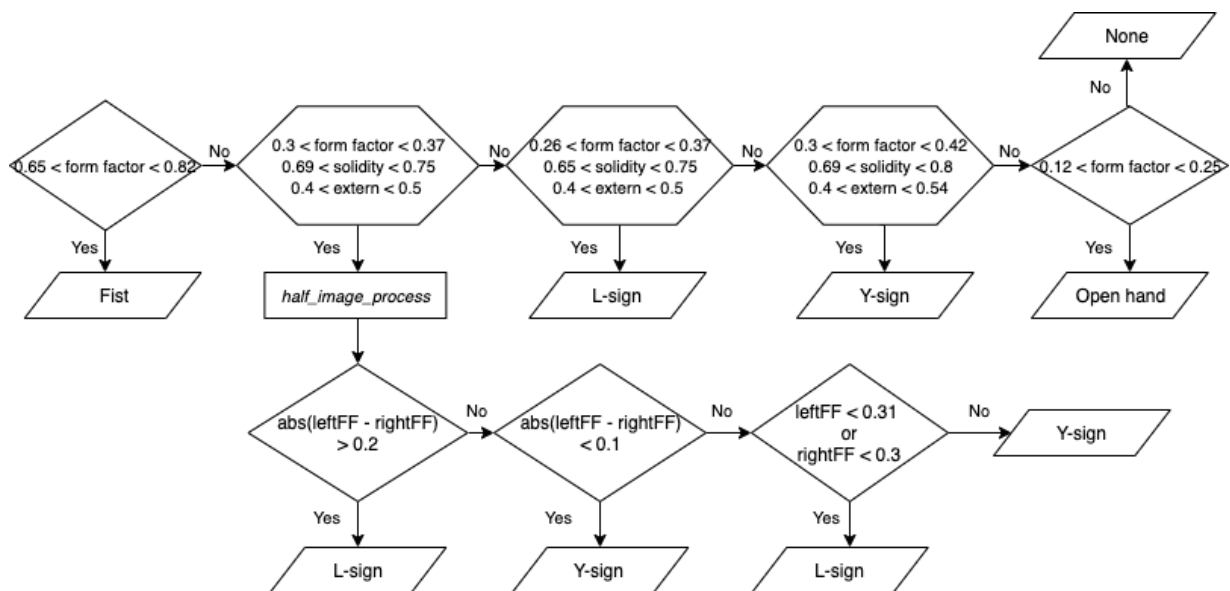


Figure 10: Classification Flowchart Overview

Project Plan

At the beginning of our project, we decided to create a flexible plan that we agreed to work on together (see table 4). Because we chose to follow the Agile project development model, we made sure that we documented all new developments after each sprint to prevent an insufficiently maintained documentation. Week 3 was the last week in which we completed each and every task together. After this week, we divided our tasks as follows:

- Athanasios: optimize processing and classification processes
- Pelle: start writing the project paper and make sure deadlines are made
- Zenit: create the required diagrams, write the system requirements section of the project paper and document all project developments

We have not made a risk/hazard analysis since our system is not safety-critical. The interactions between human and computer are not physical and there's no health risk involved in using this system. In the worst case, our system could crash, which means that the user has to restart it.

Tasks	Schedule
<ul style="list-style-type: none"> • Discuss the literature we read and how we are planning on reaching our final product • Divide the tasks necessary to complete this project • Create a project plan • Build the audio player in MatLab 	<p>Week 1</p> <p>21 November - 24 November 2019</p>
<ul style="list-style-type: none"> • Collect a large number of images to analyze: <ul style="list-style-type: none"> ◦ which features we should use ◦ the most efficient preprocessing method ◦ the most efficient classification method • Write the introduction, literature, and CONOPS sections of the report • Build the basics of the classifier 	<p>Week 2</p> <p>24 November - 2 December 2019</p>
<ul style="list-style-type: none"> • Optimize our classifier by introducing it to a larger and more diverse database of hand gestures • Write the system requirements and design sections of the report 	<p>Week 3</p> <p>2 December - 7 December 2019</p>
<ul style="list-style-type: none"> • Test our final product and prepare for the presentation. • Prepare for our presentation and create a well-working demo version of our product. • Finish the report of the project 	<p>Week 4</p> <p>8 December - 12 December 2019</p>

Table 4: *The project plan we followed*

Testing

At the end of our development, we've tested the components of the application to check if our system meets the requirements and everything works the way it should. Below, we will discuss the testing procedure we performed on the processing and classification of the image, which is the most important part of our system when it comes to performance.

Music Player

The testing of the Music Player mainly took place during its development. Moreover, the music player was tested every time whenever we ran a demo of our improved product. It is safe to say that the music player object works with 100% accuracy when the appropriate signal from the classifier is provided.

Classification

The processing and classification of the image with a hand gesture was a bit difficult to test. The tests allowed us to have a closer look at these aspects of our application, and helped us to improve our algorithms to a certain extent. Since the music player has 100% accuracy, the overall program accuracy is depending on the right classification and processing of the image.

For our final product, we tested our classifier with a test suite (set of test cases) containing 16 images³ for each gesture. We've used three different people's hands and used both the left and the right hand. The overall accuracy of the classifier is estimated at around 88%. In the confusion matrix (table 5), we can see that the "open hand" gesture is always classified accurately, and the "fist" gesture is classified with 94.4% accuracy. The main problems of our classification process arose with the L and Y signs. The L-sign has 77.7% accuracy, while the Y-sign has 66.6% overall accuracy. We have to notice that the L-sign is often misinterpreted as a Y-sign.

	Open Hand	Fist	L-Sign	Y-Sign	None	Total Accuracy
Open Hand	18	0	0	0	0	18 100% 0%
Fist	0	17	0	0	1	18 94.4% 5.6%
L-Sign	0	0	14	4	0	18 77.7% 22.3%
Y-Sign	0	0	1	12	5	18 66.6% 33.3%
None	0	0	0	0	18	18 100% 0%
Total Accuracy	18 100% 0%	17 100% 0%	15 93.3% 6.7%	16 75% 25%	24 75% 25%	90 87.8% 12.2%

Table 5: *Confusion matrix with the accuracy of the classifier*

³ The images used for testing can be found [here](#)

However, the low accuracy of the Y-sign might not be as big a problem as it looks. In 33.3% of the cases, the Y-sign is classified as “none”. This means that, if you don’t remove your hand and just wait for a short moment, there’s a 66.6% chance that the classifier now classifies your gesture correctly. This means that by using the same gesture twice, there’s an 88.9% ($=100*(1-(1/3)^2)$) chance that the classifier acts correctly at least once. The controller did not execute any incorrect operations in the meantime, and you’ll still end up with the same result. So, this 66.6% percentage might not be as bad as it looks. The same is true for the fist sign, where one of them was misclassified as “none”. If we take this into consideration and decide that a misclassification to “none” is not a “bad” classification, our system’s total accuracy increases to 94.4%, which is remarkably good.

Testing Automatization

To obtain the above-mentioned results we created a unit testing file (*tester.m*).

The tester scripts loads all the images that are inside the testing directory and test them automatically with the classifier. In order for the tester to know the right output of each image gesture, the image files have to be named properly by using this scheme:

- File containing a “open hand” gesture as `open[i].jpg`, where $[i] = 1, 2, \dots$
- File containing a “fist” gesture as `fist[i].jpg`, where $[i] = 1, 2, \dots$
- File containing a “L-sign” gesture as `next[i].jpg`, where $[i] = 1, 2, \dots$
- File containing a “Y-sign” gesture as `previous[i].jpg`, where $[i] = 1, 2, \dots$

By creating this tester, we received the testing results much faster compared to the manual process. Simultaneously, for any further development of this project, the tester can be used by the developers to immediately test the hand gestures against algorithmic improvements at the classification and processing operations.

Evaluation

During the development of this project, we came across the following major problems:

- time limitations
- different versions of MatLab (2019 vs 2017 & macOS vs Windows)

Time limitations have been the biggest constraint on the development of this project. We really wished we had the time to dive deeper into image segmentation, classification process, which has already been the hardest and most stressful part of this project, and find a way to implement neural networks to be able to recognize more hand gestures. Our original idea was to also implement the “Volume Up” and “Volume Down” commands, but the time restrictions did not let us.

As the development workspaces of our software, we used MatLab 2019 (on our own computers) and MatLab 2017 (installed in LAB computers), while also having different operating systems. When we started testing the first versions of our product, we soon discovered that MatLab 2017 does not support all of the features that MatLab 2019 support. So, we had a choice to make: which version of MatLab will we use and which operating system? We decided to use the MatLab 2017 version for Windows, which meant that we had to change some of our code, but we never came across problems like these again.

In our classification process, we chose to use the form factor, solidity, and extent as our features. These features restricted us to only use very basic hand gestures, so we also tried to use the Euler number to be able to implement more complex hand gestures. However, we found out that there was too much noise in the binary image we obtained after processing. As a consequence, the MatLab BLOB stats toolbox recognized more gaps in the BLOB than there actually were. However, this noise did not affect the features we were already using. Applying additional morphological operations solved this problem but, unfortunately, this changed the form factor of our BLOB's too much. So, that's why we decided to stick with our original four hand gestures.

As we mentioned in the code overview, we used the adaptive thresholding method, which allowed us to segment the images and get rid of the shadows and helped us with dealing with bad lighting. Still, adaptive thresholding is not a panacea, and sometimes we were not able to segment the image properly. For the first weeks, we had several serious discussions about the different segmentation methods, but we decided to stick with the adaptive thresholding, which was relatively easy while the first results we got from using it were positive and encouraging.

We might have come across many problems, but we are still very happy with the results from our testing processes. We've obtained very good results and we are very satisfied with our final product. The final model meets all the must-haves and should-haves requirements from the MoSCoW model. The could-haves are definitely possible to add to our current version in case we are given more time to finish the product. If we look at our planning, everything went perfectly fine. The communication went well and we managed to finish each new version before the deadlines.

In conclusion, we are happy with our final product and hope to inspire many more people with this revolutionary idea.

Appendix

gestify.m

Gestify A hand gesture music controller

This file is the main file of the application

```
% Clear everything at the workspace at start up.
clear all;

% Create Music Player object
musicPlayer = MusicPlayer();

% Initialize Video
video = videoinput('winvideo',1);
set(video,'ReturnedColorSpace','rgb');

% Start the video feed
start(video);

% Determine the coordinates for cropping
snapshot = getsnapshot(video);
[x,y,width,height] = getCropRect(snapshot);

% Main Loop
while true
    % Gets the snapsho and crop it
    snapshot = getsnapshot(video);
    handImageRGB = imcrop(snapshot, [x y width height]);

    % Classify the hand gesture
    sign = classifier(handImageRGB);

    % Use the hand gesture sign to contol the music player object
    if (sign == "exit")
        stop(video);
        musicPlayer.stop();
        break;

    elseif (sign == "next")
        musicPlayer.nextSong();
        pause(1);

    elseif (sign == "previous")
        musicPlayer.previousSong();
        pause(1);

    elseif (sign == "start_stop")
        if (musicPlayer.isPlaying())
            musicPlayer.stop();
        else
            musicPlayer.start();
        end
        pause(1);
    end

    pause(0.3);
end
```


MusicPlayer.m

```

classdef MusicPlayer < handle
    % MusicPlayer A music player that uses MATLAB audioplayer
    % This music player has 4 basic features:
    %     * Start
    %     * Stop
    %     * Go to next song
    %     * Go to previous song
    %
    % MusicPlayer Methods:
    %     start - Starts or unpause the music player
    %     stop - Pauses the music player
    %     isPlaying - Returns the status of the music player
    %     loadMusicDB - Loads the music files location into the musicDB array
    %     randomSong - The player picks a random songs out the the database
    %     nextSong - The player loads the next song and plays it
    %     previousSong - The player loads the next song and plays it

    properties (Access = private)
        player      % Audioplayer object
        musicDB      % Array with the songs titles as strings
        currentSong  % The index number of the current song in the MusicDB
        paused = false; % The state of the player as boolean
    end

    methods
        function obj = MusicPlayer()
            obj.loadMusicDB();
            obj.randomSong();
        end

        function start(obj)
            obj.paused = false;
            obj.player.resume();
        end

        function stop(obj)
            obj.paused = true;
            obj.player.pause();
        end

        function status = isPlaying(obj)
            status = obj.player.isplaying();
        end

        function obj = loadMusicDB(obj)
            musicFiles = dir('music/*.mp3');
            obj.musicDB = {musicFiles.name};
        end

        function obj = randomSong(obj)
            obj.currentSong = randi(size(obj.musicDB,2));
            songName = obj.musicDB{obj.currentSong};

            [data,Fs] = audioread(songName);
            obj.player = audioplayer(data,Fs);
            obj.player.StopFcn = @obj.stopCallback;
        end

        function obj = nextSong(obj)
            obj.stop();

            obj.currentSong = mod(obj.currentSong, size(obj.musicDB,2))+1;
            songName = obj.musicDB{obj.currentSong};

            [data,Fs] = audioread(songName);
            obj.player = audioplayer(data,Fs);
            obj.player.StopFcn = @obj.stopCallback;

            obj.start();
        end

        function obj = previousSong(obj)
            obj.stop();

            if (obj.currentSong == 1)
                obj.currentSong = size(obj.musicDB,2);
            else
                obj.currentSong = obj.currentSong-1;
            end
            songName = obj.musicDB{obj.currentSong};

            [data,Fs] = audioread(songName);
            obj.player = audioplayer(data,Fs);
            obj.player.StopFcn = @obj.stopCallback;

            obj.start();
        end

        function stopCallback(obj,-,-)
            % Callback function when the player stops.
            % It helps the music player to play then next song if the
            % current song finishes (without the interference of the user).

            if(obj.paused == false)
                obj.nextSong();
                obj.start();
            end
        end
    end
end
end

```

getCropRect.m

```
function [topLeftX,topLeftY,boxWidth,boxHeight] = getCropRect(backgroundImage)
    % Calculates the coordinates, the width and the height of the ROI
    % The region of interest is the white box the black tape define
    % (more info at the setup section of the paper)
    %
    % Arguments: RGB Image of the setup
    %
    % Returns:
    %     topLeftX - X Coordinate of the top left pixel in the ROI
    %     topLeftY - Y Coordinate of the top left pixel in the ROI
    %     boxWidth - The width of the ROI
    %     boxHeight - The height of the ROI

    % Convert the RGB image to grayscale and calculate the threshold
    grayscaleImage = rgb2gray(backgroundImage);
    grayscaleLevel = graythresh(grayscaleImage);

    % Convert the image to black & white with thresholding
    bwImage = im2bw(grayscaleImage,grayscaleLevel);

    % Apply a mild morphological correction
    bwImage = imopen(bwImage,strel('line',5,0));

    % Get the width and height of the image
    width = size(bwImage,1);
    height = size(bwImage,2);

    % Find the top left pixel where the ROI starts. This pixel has to be a
    % white with its left and top neighbor as black pixels.
    topLeftX = 1;
    topLeftY = 1;
    minimumSum = width/3 + height/3;
    for i = 2:width/3
        for j = 2:height/3
            if ((bwImage(i,j) == 1) && (bwImage(i-1,j) == 0) && (bwImage(i,j-1) == 0))
                % In case of multiple pixels with this features pick the
                % smallest one (aka top-left pixel of ROI).
                if(i+j < minimumSum)
                    topLeftX = j;
                    topLeftY = i;
                    minimumSum = i+j;
                end
            end
        end
    end

    % Determine the width by checking every pixel in the same row as the
    % found top-left pixel. If a black pixel is found then we hit a border.
    boxWidth = topLeftX;
    while(bwImage(topLeftY,boxWidth) == 1)
        boxWidth = boxWidth+1;
    end
    boxWidth = boxWidth-1;

    % Determine the height by checking every pixel in the same column as the
    % found top-left pixel. If a black pixel is found then we hit a border.
    boxHeight = topLeftY;
    while(bwImage(boxHeight,topLeftX) == 1)
        boxHeight = boxHeight+1;
    end
    boxHeight = boxHeight-1;
```

classifier.m

```
function sign = classifier(image)
    % Classifies an RGB Image into four classes of hand gestures.
    % The classifier uses up to three features of the hand image and
    % according to them it classifies the image as:
    %     * fist gesture
    %     * Open hand gesture
    %     * L-Sign gesture
    %     * Y-Sign gesture
    %
    % Arguments: RGB Image
    %
    % Returns: A string (sign) with 5 possible values: exit, start_stop,
    %         next, previous, none, which are used by the main program
    %         as signals for the music player.
    %
    % See also: process_image

    % Calls the process_image function to get the feature of the hand.
    [formFactor,solidity,extent,bwImage] = process_image(image);

    % Fist
    if (formFactor > 0.62 && formFactor < 0.83)
        sign = "exit";

    % L or Y Sign
    elseif ((0.30 < formFactor && formFactor < 0.37) && ...
            (0.69 < solidity && solidity < 0.75) && ...
            (0.4 < extent && extent < 0.5 ))

        % Reprocess the image
        [leftFormFactor,rightFormFactor] = half_image_process(bwImage);

        % Classify according to the new features of the left & right side
        % of the hand.
        if(abs(leftFormFactor-rightFormFactor) > 0.2)
            sign = "next";
        elseif (abs(leftFormFactor-rightFormFactor) < 0.1)
            sign = "previous";
        else
            if (leftFormFactor < 0.31 || rightFormFactor < 0.31)
                sign = "next";
            else
                sign = "previous";
            end
        end

    % L sign
    elseif ((0.26 < formFactor && formFactor < 0.37) && ...
            (0.65 < solidity && solidity < 0.75) && ...
            (0.4 < extent && extent < 0.5))
        sign = "next";

    % Y sign
    elseif ((0.30 < formFactor && formFactor < 0.42) && ...
            (0.69 < solidity && solidity < 0.8) && ...
            (0.4 < extent && extent < 0.54))
        sign = "previous";

    % Open hand
    elseif (formFactor > 0.12 && formFactor < 0.25)
        sign = "start_stop";

    else
        sign = "None";
    end
end
```

process_image.m

```
function [formFactor,solidity,extent,bwImage] = process_image(image)
% Process the image with adaptive thresholding.
% With this function we are trying to find the hand gesture as a
% single Binary Large Object
% More info: http://homepages.inf.ed.ac.uk/rbf/HIPR2/adpthrsh.htm
%
% Arguments: RGB Image with the hand gesture
%
% Returns:
%   formFactor: The form factor of the hand BLOB (0 if not found)
%   solidity: The solidity of the hand BLOB (0 if not found)
%   extent: The extent of the hand BLOB (0 if not found)
%   bwImage: The post process b&w image

% Convert the RGB hand image to grayscale
grayscaleImage = rgb2gray(image);

% Try the adaptive thresholding with different windows sizes
for windowSize = 5:2:17
    % Adaptive thresholding
    filteredImage = imfilter(image,fspecial('average',windowSize),'replicate');
    filteredImage = filteredImage-grayscaleImage-windowSize;

    % Convert the image to black and white
    bwImage = im2bw(filteredImage,0);

    % Apply a mild morphological correction
    bwImage = imopen(bwImage,strel('disk',12));

    [labels,numLabels]=bwlabel(bwImage);
    if (numLabels == 1)
        break;
    end
end

% Get the features of the hand BLOB if found, else return 0 values.
if (numLabels == 1)
    stats = regionprops(labels, 'all');

    area = stats(1).Area;
    perimeter = stats(1).Perimeter;

    formFactor = 4*pi*area/((perimeter)^2);
    solidity = stats(1).Solidity;
    extent = stats(1).Extent;
else
    formFactor = 0;
    solidity = 0;
    extent = 0;
end
```

half_image_process.m

```
function [leftFormFactor,rightFormFactor] = half_image_process(bwImage)
% Cut a black & white image in half with a single BLOB in it.
% This function is used to further analyze an L or Y sign by
% calculating the form factor of the left and right sub-images.
% See also: classifier
%
% Arguments: Black & White with a single BLOB in it.
%
% Returns:
%   leftFormFactor - The form factor of the left sub-image
%   rightFormFactor - The form factor of the right sub-image

% Use the Bounding Box feature to locate the hand location.
[labels,~] = bwlabel(bwImage);
stats = regionprops(labels, 'BoundingBox');
box = stats(1).BoundingBox;

% Have the black & white image with the hand center in the image
bwImage = imcrop(bwImage, [box(1) box(2) box(3) box(4)]);

% Calculate the center of the image and create the left & right images
center = size(bwImage,2)/2+.5;
leftBWImage = bwImage(:,1:center);
rightBWImage= bwImage(:,center:size(bwImage,2));

% Left side image
[labels_left,~] = bwlabel(leftBWImage);
leftStats = regionprops(labels_left, 'all');
leftArea = leftStats(1).Area;
perimeter_left = leftStats(1).Perimeter;
leftFormFactor = 4*pi*leftArea/((perimeter_left)^2);

% Right side image
[labels_right,~] = bwlabel(rightBWImage);
rightStats = regionprops(labels_right, 'all');
rightArea = rightStats(1).Area;
rightPerimeter = rightStats(1).Perimeter;
rightFormFactor = 4*pi*rightArea/((rightPerimeter)^2);
```

tester.m

Contents

- [Test open hand gestures](#)
- [Test Fist gestures](#)
- [Test L-Sign gestures](#)
- [Test Y-Sign Gestures](#)

```
% This script serves as an automatic tester for the classifier.
% It loads the testing files sequentially and test them against the
% classifier.
% The files must be named in a correct manner:
% * Open hand as open[i].jpg, where [i] = 1,2,...
% * Fist as fist[i].jpg, where [i] = 1,2,...
% * L-Sign as next[i].jpg, where [i] = 1,2,...
% * Y-Sign as previous[i].jpg, where [i] = 1,2,...
%
% We used this file to construct the confusion matrix.
```

Test open hand gestures

```
imageFiles = dir('images/testing/open*.jpg');

openHandPassed = 0;
openHandFailed = 0;

for i=1:size(imageFiles,1)
    image = imread(imageFiles(i).name);
    sign = classifier(image);
    if(sign == "start_stop")
        openHandPassed = openHandPassed+1;
    else
        openHandFailed = openHandFailed+1;
    end
end
```

Test Fist gestures

```
imageFiles = dir('images/testing/fist*.jpg');

fistPassed = 0;
fistFailed = 0;

for i=1:size(imageFiles,1)
    image = imread(imageFiles(i).name);
    sign = classifier(image);
    if(sign == "exit")
        fistPassed = fistPassed+1;
    else
        fistFailed = fistFailed+1;
    end
end
```

Test L-Sign gestures

```
imageFiles = dir('images/testing/next*.jpg');

nextPassed = 0;
nextFailed = 0;

for i=1:size(imageFiles,1)
    image = imread(imageFiles(i).name);
    sign = classifier(image);
    if(sign == "next")
        nextPassed = nextPassed+1;
    else
        nextFailed = nextFailed+1;
    end
end
```

Test Y-Sign Gestures

```
imageFiles = dir('images/testing/previous*.jpg');

previousPassed = 0;
previousFailed = 0;

for i=1:size(imageFiles,1)
    image = imread(imageFiles(i).name);
    sign = classifier(image);
    if(sign == "previous")
        previousPassed = previousPassed+1;
    else
        previousFailed = previousFailed+1;
    end
end
```


References

[1] Blue silhouette of hand holding the musical note vector image. [Photo]. Retrieved from <https://www.vectorstock.com/royalty-free-vector/blue-silhouette-of-hand-holding-the-musical-note-vector-14534526>

[2] Premaratne, P. (2014). Human Computer Interaction Using Hand Gestures. Singapore: Springer Singapore.

[3] Parab, Aditya & Ghosalkar, Janhavi & Akshay, V & Medge, B & Meena, Mamta. (2016). Music Controller using Gesture Recognition to Control Music Playback.

[4] The MathWorks - audioplayer. Retrieved from <https://nl.mathworks.com/help/matlab/ref/audioplayer.html>.

[5] 15 Free Gesture Icon Sets for Mobile Developers. [Photo]. Retrieved from <https://speckyboy.com/wp-content/uploads/2015/11/gesture-icons-free-set-14.png>

All figures have been created by using draw.io and creately.com.