Multimedia Computing Project Specification (Semi-Automatic Timeline)

David Mendes #44934
Ricardo Esteves #44930

ABSTRACT
In this study, we will demonstrate not only the study behind the creation of a system capable of semi-automatically generating a timeline, but also the preliminary interface designs and behaviors of the system. Our study was mainly based on three different papers, which we will review and point the main elations we took from each in order to improve our own system.

KEYWORDS
Timeline, Metadata, similarity search, interface, content-based, features extraction, video, audio, files, Luminance, Color based, Texture, Rhythm, Video, Frame, Pixels.

1. INTRODUCTION
The main goal of the project is to create a system that should be capable of building a semi-automatic timeline, based on the similarity of several parameters. Not only should it be able to generate said timeline, but it must also allow some liberties for the user, so he can re-order or delete clips of the generated timeline, and also add transitions between said clips.

Related to the study behind the system development we will shortly resume and review the paper contents, while focusing on constructively criticizing the papers content.

The knowledge gathered from the papers coupled with further analysis and consideration on the matters, was used to boost our own interface design, bearing in mind the importance of metadata and intuitive navigation.

2. PAPER'S REVIEW
2.1 VFerret: Content-Based Similarity Search Tool for Continues Achieved Video
Paper [1] describes VFerret, this system allows users to perform content-based similarity search using visual and audio features, while keeping a small memory footprint due to the very low amount of metadata needed to be archived along with the videos.

The main useful feature we detect with this system, is the ability to search unlabeled continuous archived data. The use of the extracted visual and audio features gives a tremendous amount of information, that enables it to perform remarkably well. Results show an impressive 0.79 search quality using only the extracted information, while the addition of annotation-based metadata is also possible and should boost its effectiveness even further.

The system works basically in the following manner: the video is firstly evenly segmented into 5-minute clips, after said splitting the metadata is extracted along with the visual and audio features. The metadata is stored in a database, so the user can query the clips with the attributes he desires and only then the clips are clustered based on the extracted visual and audio features. After the clustering the user can now proceed with its content-based search to find the desired clip.

"For each cluster, only one representative video clip is shown, so the user can quickly locate a video clip that is similar to the desired clip. Once a similar clip is found (this usually is much easier than finding the precise desired clip), the user can initiate the similarity search to find the clips of interest quickly."

The user search process can be split into three different steps, each being very important to iteratively improve the search quality of the intended clip.

2.1.1 Timeline-based search step
The first step is the most natural to users, as we naturally associate events with a specific time range. It is also the most effective when a specific clip has a distinctive date association (e.g. birthday party). This step is also the easier to accomplish in the system execution as the date is almost 100 % of the times, implicitly described in the clips metadata. Date range specification boosts significantly the search quality of the following steps, as it reduces the clips needed to be clustered.

2.1.2 Clustering step
Machine learning algorithms are used to cluster similar clips together, and a representative clip is chosen to be displayed to the user per cluster. After the cluster creation the user now selects the cluster that shows the most similarities with the clip he is looking for.

2.1.3 Content-based similarity search step
"Once user has a query clip, he can initiate the similarity search and iteratively refine the search to find the desired result ... This process will provide higher quality results iteratively and help the user quickly pinpoint the desired clips without browsing through the entire candidate set."

Generally speaking VFerret is a capable and intuitively designed system, that doesn't strike us as having major flaws, but instead shows us the importance and capabilities of feature extraction to boost the quality of search. Said importance acknowledgment will surely be reflected on the design of our own filter selection and results presented to build the timeline.

2.2 Video Exploration: From Multimedia Content Analysis to Interactive Visualization

From the paper [2], we can extract valuable information not only about multimedia content, but also, we can focus on the importance of its visualization and the relevance of it being intuitive while maintaining a great degree of informative and enjoyable content.

In an era, where the amount of content increases exponentially, there must be ways to display it on a human scale, something that can be seen, interpreted and chosen with user criteria, avoiding overwhelming and uninteresting information to a specific user. The paper presents three different interfaces with some common goals but major differences on the execution.

2.2.1 Common ground

For all interfaces, there is some common ground on data analysis and shot representation. The segmentation process of the videos relies heavily on visual content analysis as the features get extracted from each frame of a specific stream. The feature extraction not only enables content clustering but also, provides a way to represent a shot, being the result of visual descriptors clustering, a shot provides a way to summarize visual content, by displaying representative images of a stream, but also makes it easier to mine data on a shot, given the fact that it is a lower amount of data to be processed at the shot level compared to the stream. This process is mainly boosted by machine learning algorithms that intend to reduce the number of descriptors per shot and select said descriptors to "describe" the shot.

Given the fact, that all interfaces interact with TV programmes, we have to bear in mind that there is a considerable amount of metadata that is natural to this type of content, this kind of metadata is not derived from any kind of data analysis but from annotations created by the channels or archivists. Said metadata, gives important information about the type of content and enables a more precise segmentation.

2.2.2 Stream explorer

This interface uses the metaphor of the 60's tape recorder. The use of metaphors is supposed to give the user an intuitive feel for the system, as the metaphor represents something the user is accustomed to use and is able to associate its behavior within the application. In our opinion this metaphor can be somewhat confusing for someone who has not come in contact with a tape recorder. Therefore, we believe that this interface will have different acceptance standards according to the age demography of the users.

Not getting into much detail, about the complexity of the interface execution, the following features represent what we thought was the most pleasant and usable elements of the interface.

The stream explorer segments the video stream into classes, distinguishable by colors. This makes it easy to observe the intended segments within a specific time frame. Another good feature is the display of the metadata related to the current segment, taking a central position for good visibility.

2.2.3 TV Programme Explorer

This interface has a different take on the segmentation aspect of the video stream. Segmentation is made first in chronological order, then images that represent the same segment are clustered together, and the most relevant images of said segment are presented in a loop. The loop length, gives us a good notion of duration. The horizontal axis also has a representative image per segment.

Metadata is displayed close to the video being played, and textual search is also included in the interface. In our opinion the textual search is not really in-line with the rest of interface visually, and the search feedback could be improved, as highlighting the clips seems a bit underwhelming given the number of static images in the interface. We would suggest a restructure of the timeline given the textual search, so only the clips relevant to the search would show up, and therefore we reduce the "noise" in the interface.

2.3 Advanced User Interfaces for Dynamic Video Browsing

In this paper [3] they show us three different interface designs that enable users to browse a video by moving a slider along the timeline.

One problem of sliders is when a document is very long even the smallest unit to move the slider might result in a large step in the file. Hence, moving the slider allows users to navigate or browse through the file, but is not able to access an exact position due to the scaling problem. In addition to the previous information, jumps in the file when moving a slider continuously result in a "*bumpy*" visual feedback.

The first interface presented was the *ZoomSlider*, because it enables a user to continuously zoom in and out of a slider scale while browsing a video file. This enables finer or wider granularity, while being able to skim through the data.

The second interface is *NLslider*, the main goal of this approach is to solve the scaling problem by using nonlinear slider scale, the main advantage is the scale always stays within the application window and does not expand across the window's border. In our opinion this one is less intuitive and took way more time to understand, so we wouldn't exactly follow this approach.

The third and last interface proposed solves the scaling problem not by modifying the scale of the slider but instead allows the user to change the scrolling speed, by enlarging the distance between the thumb and the pointer. This approach is more visually appealing and intuitive as the "elasticity" metaphor works really well to give users a "feel" for the system while maintaining functionality.

In our opinion there is no best solution, as they depend on the usage scenario, but with our intended usage, the third interface is the one that better fills the bullet points of our intended purpose for video browsing.

3. INTERFACE DESIGN

In the following schematics we present the preliminary mockups of the application.

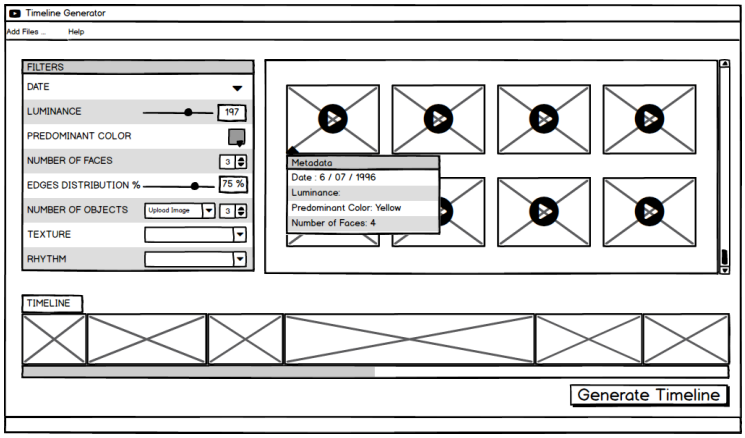

Figure 1 - Main Application Window

A top bar is provided to allow the selection of movie clips, by displaying the default Windows file browser, after pressing the "Add Files …" button. The help button will display some basic instructions on an overlay along with Author credits.

The Main Application Window features a filter menu, in the upper left side. The filter menu, presents a wide range of options, adjusted by sliders, dropdown boxes, input fields and a complex date range picker.

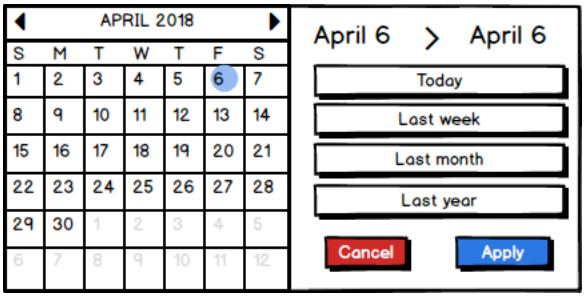

Figure 2 - Date Picker

The Date Picker gets displayed on top of the filter menu when the user wants to delimit the date of the clips. Pre-defined ranges are included for ease of use.

The upper right side shows the filtered clips by chronological order. Hovering on top of a specific clip brings up the metadata information on said clip.

Pressing the "Generate Timeline" button will create the timeline with the filtered clips and redirect to a new window.

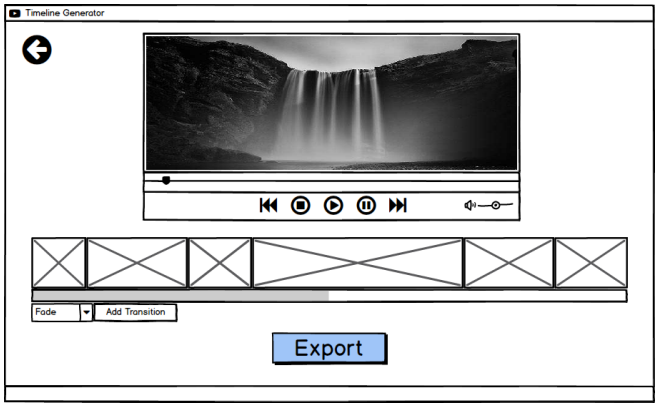

Figure 3 - Timeline Player and Editor

The upper left button (backwards) enables the user to return to the Main Application Window to readjust the clip selection.

Basic controls are offered on the player itself. Controls include a Fast Backward, Stop, Pause, Play and Fast Forward Button, along with a volume slider and a moving slider representing the video length covered.

Below the player, we present the clips that compose the timeline. The clips can be reordered by long pressing and dragging each clip to the new position. Delete functionality works the same way but the Export Button transforms into a Trash Can, so the user can drag the clip to the Trash Can to remove it.

Transitions can also be added using the provided combo-box, and finally the user can export the video to a specific directory.

4. ALGORITHMS AND TECHNIQUES

4.1 Edge Distribution

To implement this filter, we first apply a gaussian blur to reduce image noise. Then, we transform each frame into a grayscale image, because luminance is far more important and provides better edge information than any of the R, G, or B planes.

Only when we have the grayscale image, we start applying the "Edge Detection" algorithm. We could have used Sobel [4], Prewitt, but we ended up using Scharr, as it is recommended by the openCV documentation [5]. The Scharr matrixes are applied through the image and the results are computed for both X and Y. We then define a threshold, and compare each pixel color value to that threshold, when the value exceeds the threshold, we consider that pixel to be an edge. This process is repeated for both the X and Y gradients. The number of pixels whose color value exceed the threshold on the X gradient are considered to be vertical edges, while on the Y gradient are considered horizontal edges.

We then compute the percentage of vertical edges on a frame to be the ratio between vertical edges and the sum of vertical edges and horizontal edges. The process is repeated for every frame in the video and the results are averaged.

The input for the filter is simply the percentage of vertical edges desired for the videos. Given the fact that the percentage of vertical edges is complementary to the percentage of horizontal edges, the choice between them was arbitrary.

4.2 Luminance

To calculate this filter, we go through every pixel of every frame of the video and extract its lightness (the average of the R, G and B components). To compute the final Luminance value, we gather the value calculated above and we divide by the width of the frame multiplied by the height and multiplied by the total number of frames of the video, to get the average luminance of each pixel of the video.

The input of this filter is the average luminance of a video, ranging from 0 to 255.

4.3 Color Based on the First Moment

In this filter, we go through every pixel of every frame of the video and save all RGB colors separately. In the end, we will divide them by the total amount of pixels of the video to get the average of each RGB color. In this specific filter, we considered a Hexadecimal color pallet for the input. As we stated above we

have the color based in RGB and the input is Hexadecimal color, to resolve this problem we convert the RGB into Hexadecimal to match the input type and add a predefined range, because it would be too hard for the user to select the exact color of the videos.

4.4 Number of key point matches between an input image and a video

In this filter, we match an input image against the representative frames of the videos. We don't go through all the frames because we can't pre-process an input image; as so, going through every frame of every video would not be reasonable as the computation takes a considerable amount of time.

Before applying any kind of key point detection, we start by converting the images into grayscale, because as said before edge detection works best on the luminance level.

We produce an instance of a key point detector, with a SURF threshold of 400, that is used to "control the balance between the number of key points and their stability" [6]. In the next step, we will compute the descriptor extractor for each of the key points. Now we have the descriptors for both images.

Next, and because this is done at runtime, we use the FLANN (Fast Library Approximate Nearest Neighbors) matcher to improve the speed of the matching. We get the best matches using 2 as the max for the knn algorithm.

Next up we apply Dr.Loe's ratio test for computing good matches between the key points, ignoring the descriptors that do not fit the criteria. [7,8].

For each video the maximum number of key points between all representative frames is returned and can be filtered according to the input of the user.

4.5 Number of Faces Appearing in the video

In this specific filter, we have the goal to find how many faces appear in a video, therefore we found a slider to be the best way to represent this filter input. We implement this filter, using the *ofxCvHaarFinder* class of ofx and the function *findHaarObjects()*. This function returns the number of faces in each frame of the video. As our goal is to find how many people have been or are in the video we compute a simple max function (within all the frames) to find this value.

4.6 Texture

Texture characteristics can be estimated using Gabor Filters. To implement this filter, we started off with the information provided by the teacher. First, we create a Gabor filter using the function *getGaborKernel()*[12], with the five parameters: size of the matrix, sigma, theta, lambda, and gamma. We decided to maintain most of the values given in the example [12], but varying the wavelength (lambda) and orientation (thetha). The second step is to apply the filter created, so we use the function *filter2D()*. One particular aspect that we decided relevant to this function is the depth value, we use -1 as the value of depth, because of this value the output image will have the same depth as the source image.

Finally, we find the Gabor value by calculating the mean of the output image matrix and once again averaging the Gabor value between all the variables permutations made. We found once more that the use of a slider was the simplest way to choose the number, as we found no simple (and well-founded) way to categorize these results.

4.7 Rhythm

This filter measures the difference between two frames in the videos. To make this possible we calculate the average luminance of each pixel of each frame, in this aspect is very similar to the luminance filter, but the difference between them is the rhythm calculates the average **difference** between two frames in a row and after that calculates the mean variation between all frames.

As happened with previous filters the output will be a unique value, and to manage that we consider a slider as an input for this filter.

4.8 Cut Detection

Cut detection is based on rhythm analysis. As discussed previously, rhythm, considers luminance values. To determine the best frames to display as representative images to the user, we calculate the maximum variations between consecutive frames, as these usually represent a scene change. We established a limit of 5 representative frames per video, and so, we searched for the 5 biggest variations between frames and set those as the representative frames. These frames are saved and cycled through in the display of the videos inside the video box.

5. CLASS DESCRIPTION

In this topic, we explain the class organization of this project. This organization results from a deep analysis of the problem and we tried to find the simplest solution that fitted our needs.

The *offApp* class is the class that not only holds every other element, but also ultimately controls what is displayed on the screen. This class is also responsible for controlling the behavior of the two main buttons (Generate and Modify Timeline) and perform the necessary operations on the other classes to accommodate the state changes these incur.

We decided to create a class filter that is responsible for creating our filters box, this class instantiates Luminance, Face Detection, Rhythm, Texture and edge distribution sliders. For the predominant color filter, we choose to use a color picker. Our Date Filter is a drop-down with two inputs, one is for the Start date and the other for the End date. In the key point detection filter we use an image input, to allow the user to import the desired image and an input to put an integer number to represent the max number of key point matches between the input image and the video. For all our filters we create a toggle with the aim of allowing the user to activate and disable the desired filters. If the checkbox is activated this filter will contribute to the selection of the videos in the timeline.

This class holds and provides all the information necessary to apply the filtering of the videos, mostly are "getter" methods that give the *ofApp* information about what the user inserted in the filters.

The *videoBox* class holds all the videos the user imports and displays them in a grid. Each video contained is a *VideoElement*, in this class we manage the visualization of the thumbnail, the video highlighting (for delete), and the play button display and functionality.

Our next class is the *Timeline*, this class is responsible for holding the videos on the timeline with a particular order and position. Inside the Timeline, we have *TimelineElements*, this class is responsible for drawing the thumbnail of the respective video in the correct position. The *Timeline* also holds *switchElement*s that are the arrows that allow for video reordering. The *Timeline* controls whether these are visible or not.

If we want to play only one video we can press the play button on the video thumbnail and play only that selected video, this is being done by the *SingleVideoPlay* class. In there we have the controls to play, pause and stop the video for a better experience of visualization.

Once we click on the modify Timeline button, the *generatedTimeline* class comes into play. This class holds all the logic to make it so the multiple videos in the timeline can be reproduced as a single video in the most seamless way possible. This class also holds the dropdown and the logic to apply the fade animation to all videos in the timeline.

Another class that is a core of the program is the *XMLUtils*, since there is where we do all the computation for all the algorithms implemented in the program and also where we have the functions to extract information from the XML files.

We also have a namespace (*EventManager)* that holds extern events, so they can be accessed by every class.

6. APPLICATION IMPLEMENTATION

Next up we will try to explain in broad lines how every class accomplishes the desired behavior. For more detailed information on the intricacies of the program, the code is well commented and should clarify any doubt.

6.1 ofApp.cpp

The drawing of the main areas is done by the ofApp, and said draw is controlled by two booleans. There are three possible screens. The MainScreen, Single Video Reproduction and Generated Timeline Reproduction. Each screen is drawn according to a combination of said boolean variables. The ofApp creates, draws and handles the events generated by the generateTimeline and modifyTimeline buttons.

For the Timeline generation, the app gets all the information available in the filters class and creates the timeline accordingly.

The modifyTimeline button simply changes the state boolean variables to draw the desired screen. It also enables modifications to be made on the timeline.

6.2 filters.cpp

This class simply instantiates and draws the filters GUI, with the help of the ofxDatGui addon [13], it also has all the "getter" methods for said filters.

6.3 videoBox.cpp

Draws and holds all the videos the user imports. Has listeners for adding a video or deleting a video to/from the videoBox.

6.4 videoElement.cpp

Represents a video in the VideoBox, retrieves and displays the video's representative frames, and cycles through them at a rate of 2 seconds. Also shows the metadata of the video when the user hovers and creates the button for Single Video Reproduction and triggers the related event when the user clicks it. It also controls the toggle and consequent highlighting of the video for deletion.

6.5 Timeline.cpp

Holds the videos that get filtered and handles their reordering or deletion. Reordering and deletion is done by attaching listeners for those specific events.

The timeline has two vectors, that hold the Timeline Elements and the SwitchElements.

When the delete key is pressed, the timeline checks if it is in the modifiable state, and if so, collects all the toggled videos and removes them from the stored vector and rearranges the positions and size of the new videos. It also notifies the Generated Timeline that the videos that are in the timeline changed, so the playback changes accordingly.

6.6 TimelineElement.cpp

Responsible for displaying the video in the timeline, it shows the first frame of the video on the position and with the size, that gets passed in the setup. It controls whether the video is toggled for deletion, by checking if there was a click inside it's boundaries. It also has methods for getting the old and setting a new position, that is helpful for reordering of the videos in the timeline.

6.7 switchElement.cpp

Displays the arrows for reordering of the timeline when active. Is created with two indexes that represent the adjacent videos indexes. It checks for user clicks inside it's boundaries and when it happens triggers the events for reordering of the timeline, passing the videos indexes to be switched.

6.8 singleVideoPlay.cpp

Draws the video to be reproduced, and the controls. Similar to the class that will be explained in the following paragraph but with reduced complexity as it only displays one video unlike the next.

6.9 generatedTimeline.cpp

Perhaps the trickiest class to understand as the conjoint videos playback should seem as one single video.

Holds a vector of videos for playback, alongside their durations to know when to switch to the next video.

Has a listener for reordering, that swaps the position of the videos in the vector and the corresponding durations get recalculated. The other listener is responsible for getting the video urls to be reproduced. Every time a new list of videos is fed to the generatedTimeline every duration is recalculated and playback is reset.

The slider also has a listener to enable the user to seek within the timeline. The value is retrieved, and the corresponding video is played on the exact position of the slider. The accumulative durations vector is helpful here as its content is similar to the range in the slider and it becomes easier to play the correct video without needing to iterate through every duration and adding them up, every time the user uses the slider.

The update method constantly checks if the current video has ended so it plays the next video. It is also responsible for making the slider accompany the video reproduction by constantly setting the slider according to the current frame that is being played.

This class also displays the dropdown that allows for the fade effect to be applied. Whenever it is active, and the video is in the last 50 frames we start progressively reducing the alpha until the last frame of the video.

6.10 XmlUtils.cpp

This class only has static methods. Has the name indicate, they are utility methods that are used by almost every class (as displayed in the ClassDiagram). It implements the algorithms for videoProcessing as stated in section 4. And is responsible for reading values from the Xml files given the video path.

7. CONCLUSION

Throughout this work, we faced some difficulties, mainly due to the openFrameworks behavior that was new for us.

Surpassed the initial challenges we believe that we achieved a good result as we tried to make wise decisions, in regard to the program's usability and its internal components (we thrived for making them as comprehensible as possible).

After doing this work, we think that the knowledge gathered from the papers made us more alert to a more interactive and dynamic visualization of video, alongside the acknowledgment of the complexity hidden in this type of systems.

8. REFERENCES

[1] Zhe Wang, Matthew D. Hoffman, Perry R. Cook, Kai Li. VFerret: Content-Based Similarity Search Tool for Continuous Archived Video. Computer Science Dept, Princeton University, Princeton NJ, 08540

[2] ML Viaud, O Buisson, A Saulnier, C.Guenais INA. Video Exploration: From Multimedia Content Analysis to Interactive Visualization. 4 avenue de l'europe 94366 Bry Sur Marne, France

[3] Wolfgang Hürst, Georg Götz, Philipp Jarvers. Advanced User Interfaces for Dynamic Video Browsing. nstitut für Informatik, Albert-Ludwigs-Universität Freiburg, D-79110 Freiburg, German

[4] "Sobel operator," *Wikipedia*. 21-Apr-2018.

[5] "OpenCV: Sobel Derivatives." [Online]. Available: https://docs.opencv.org/3.2.0/d2/d2c/tutorial_sobel_derivatives.html. [Accessed: 01-Jun-2018].

[6] «Features2d — OpenCV 2.4.13.6 documentation». [Em linha]. Disponível em: https://docs.opencv.org/2.4/doc/user_guide/ug_features2d.html. [Acedido: 28-Mai-2018].

[7] "Object detection with SURF, KNN, FLANN, OpenCV 3.X and CUDA| Machine Learning | Software Development," *Robotic Vision | Machine Learning | Software Development*, 27-Jun-2016. .

[8] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[9] «Image Filtering — OpenCV 3.0.0-dev documentation». [Em linha]. Disponível em: https://docs.opencv.org/3.0-beta/modules/imgproc/doc/filtering.html. [Acedido: 29-Mai-2018].

[10] «ofxCvHaarFinder | openFrameworks». [Em linha]. Disponível em: https://openframeworks.cc/documentation/ofxOpenCv/ofxCvHaarFinder/. [Acedido: 29-Mai-2018].

[11] «public:gabor_filter [juergen's work wiki]». [Em linha]. Disponível em: http://www.juergenwiki.de/old_wiki/doku.php?id=public:gabor_filter. [Acedido: 29-Mai-2018].

[12] "documentation | openFrameworks." [Online]. Available: https://openframeworks.cc/documentation/. [Accessed: 01-Jun-2018].

[13] "OpenFrameworks UI – braitsch." [Online]. Available: http://braitsch.github.io/ofxDatGui/index.html. [Accessed: 02-Jun-2018].