# Team 23

# Real Time Drawing System

Nicola Bresciani 168456
nicolabresciani91@gmail.com
Daniele Marinelli 168026
daniele.marinelli91@gmail.com

## Abstract

Nowadays one of the most adopted technology that enables a user to hand-draw images, graphics and have a real time digitalized version of the sketch is the graphic table.
This kind of devices are expensive and fit the needs only of the professional customers. Usually the non-professional users are not willing to pay a consistent amount of money for a device that will be used occasionally. The aim of this self-proposed project is the implementation of an application able to track the creation of a sketch on a sheet of paper and display it in real time on a digitalized version. Moreover, the user can interact with the system by creating personalized buttons directly on the sheet: the color selection of the line is an example of this functionality. With this application we want to provide a cheap and fast alternative to the expensive graphic tables. The only hardware needed is a webcam, a color printed version of the provided sheet and a pen. For example our application can be used during live lectures by speakers to send notes in real time to the audience. The application is developed in such a way that the freedom of the user during the drawing phase is always guaranteed.

## 1. Objectives

The main objective of our application is to provide to the user a cheap and funny solution for drawing sketches. We developed a framework that is easy to use and that doesn't introduce huge constraints and limitations to the user.
To allow the interaction between the user and the application we introduce the possibility to create personalized buttons in a predefined area of the sheet. For each new button the user has to specify the action that the system will do once the button is pressed. Thanks to the described characteristics the user can express his ideas and artistic capabilities and have the digitalized version at his disposal in real time. In such a way the user can share his creation in real time with an audience or on the Web.

## 2. Achievements

With this project we provide an application able to track the real time creation of a sketch on a sheet of paper.
In the project plan we scheduled to reach the following achievements:

- real time visualization of the digitalized output,
- robustness to sheet or camera movements,
- creation of personalized buttons directly on the sheet.

### Real time visualization of the digitalized output

The application is able to detect the line produced by a pen (or pencil, or marker…) on a sheet of paper. The latter is an A4 paper with colored markers. Considering the horizontal orientation, the sheet includes: two black crosses placed on the top borders, two red crosses placed on the bottom borders and two green crosses placed just below the black ones. The image acquisition is performed by a webcam that can be placed everywhere, with the constraint that all the crosses are visible during the calibration phase. Is worth nothing that the resolution of the webcam is always set to the maximum available. This to achieve the best quality. The new parts of the draw are shown as soon as they are not occluded by the hand. The rectification of the draw is performed by a homography operation. It is a transformation that needs the coordinates of 4 reference points (at least) and the distances between them. The digitalized version of the draw is visible in a dedicated window and can be saved as a JPEG image without deformations due to angular acquisition.

### Creation of personalized buttons directly on the sheet

The user can create personalized buttons in the top area of the sheet defined by the black and green crosses. We will refer to this region as the "area of interactive functionalities". The software is able to automatically detect new buttons. Once a new button is detected, the user has to decide the action to be performed. In order to select a button, the user has to simply press it (i.e. performing the occlusion of the area).
In order to show the capabilities of interactive functionalities, a color line decision has been implemented: every button has a color associated. The user can select the color through a palette.

### Robustness to sheet or camera movements

The first version of the software allowed sheet and camera movements. However, this set-up introduced a strong limitation to the user: in order to correctly acquire the new parts of the draw, all the crosses had to be visible by the webcam. Drawing in this condition was uncomfortable and forced the user to sketch in unnatural position. Because of this we decide to understand how much is important the possibility to move the paper by analyzing the videos of some famous cartoonists. The result of this analysis led us to the conclusion that the assumption of no relative motion between camera and paper is fair. This constraint can be considered "soft" and plausible. In case of movements the final draw will exhibit some artifacts, such us multiple parallel lines or salt-and-pepper noise. To reduce this type of noise the user can apply a bilinear filter thought the GUI. Moreover, we noticed that the autofocus feature can introduce additional artifacts. Therefore we strongly suggest to disable this feature via the camera options panel.
In conclusion our software is a cheap and high quality solution for the creation of cartoons, real time presentations and sketches.

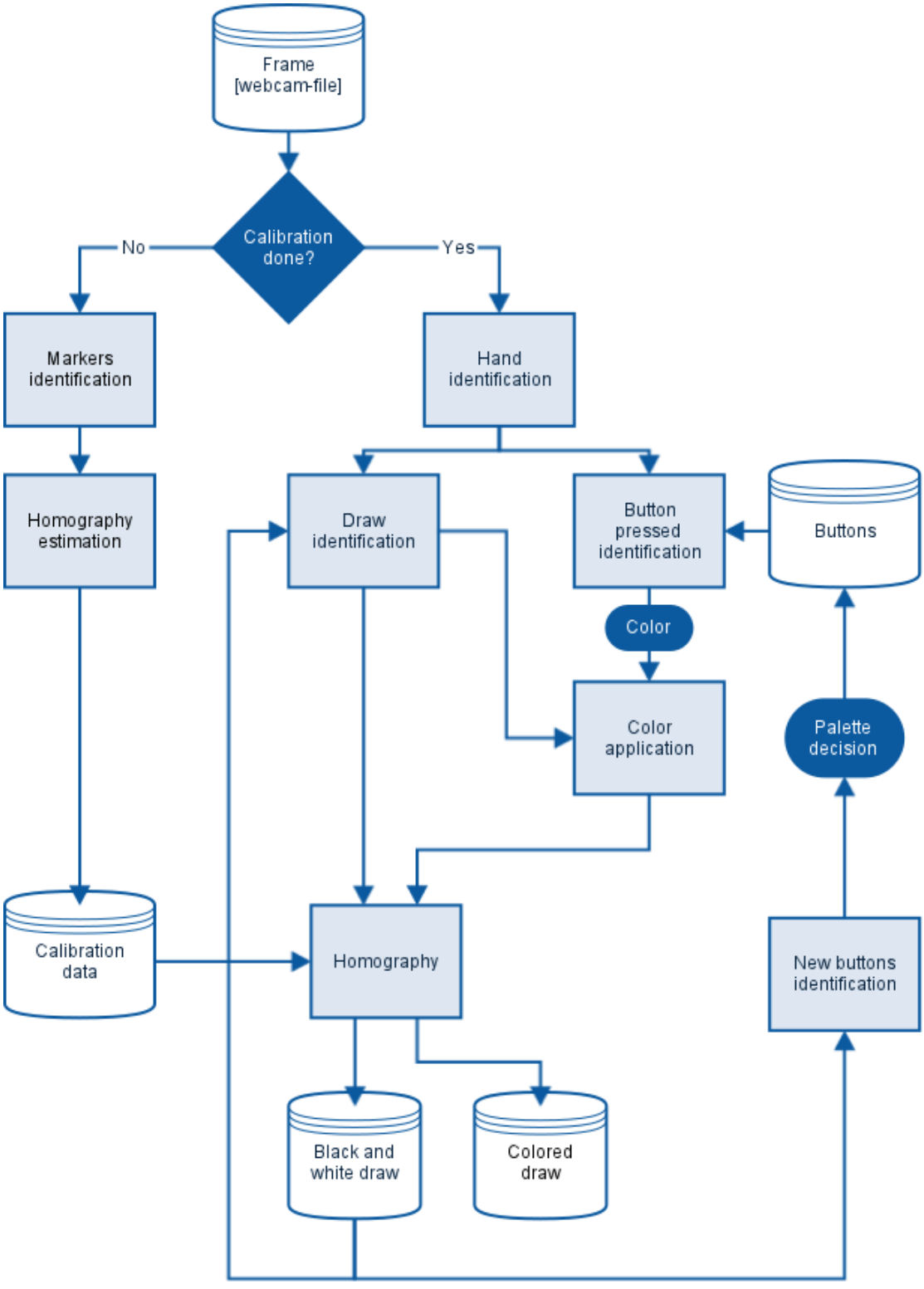


*Figure 1 Software Architecture*

Figure 1 shows the architecture of the software. In the following the single blocks are briefly described:

- **Markers identification:** aims to find the six colored crosses.
- **Homography estimation:** extracts the homography matrix by using 4 reference points (the centers of the bordered crosses).
- **Hand identification:** creates an area representing the hand of the user.
- **Draw identification:** identifies the new parts of the draw.
- **Button pressed identification:** detects the selection of a button.
- **Color application:** changes the color of the line according with the currently selected color.
- **Homography:** performs the rectification of the colored and the black-and-white draws using the parameters estimated during the calibration.
- **New buttons identification:** finds new buttons in the area of interactive functionalities. After the identification of a new button, a palette for color decision is presented to the user.

The software has been developed using OpenCV libraries v2.4.9. We made this choice because the libraries:

- allow basic and advance functionalities in computer vision,
- have good performance,
- are widely adopted and there is a vast documentation available.

Regarding the programming language, our first choice was Java due to its simplicity in implementation and error handling. We soon realized that OPENCV libraries are not fully compliant and offer a limited set of functionalities in the Java release. Due to the fact that OpenV libraries are C++ based, we decided to migrate from Java to C++. Moreover C++ is a more suitable language for real time computations. Regarding the GUI, it has been implemented using Qt libraries v5.4 due to their easy integration with C++.

Hereinafter we provide a brief explanation of the self-contained package.
The resource files are:

- C++ source files (.h and .cpp),
- "palette.jpg" is the palette for color decision,
- "sheet.pdf" is the A4 paper to be used.

The executable file (.exe) launches the graphic user interface. Through the GUI the user can

- adjust some parameters of the algorithm ,
- select the input: real time acquisition or pre-recorded video,
- save the sketch on file,
- show additional windows (for debugging purpose),
- open the camera option panel,
- clean the image if there are some artifacts,
- apply a bilinear filter to a draw previously saved.

## 3. Project Implementation

The project has been implemented in C++ using the OpenCV libraries. The application is divided in three main blocks:

- Calibration,
- frame processing,
- buttons management.

In the following sections the three blocks will be analyzed in detail.

### Calibration

The calibration of the application is performed only at the beginning of the frame acquisition and it is needed to identify the sheet of paper and to estimate the parameters for the homography mapping. The system executes the calibration function *sheetIdentification* on each frame until it is completed successfully. Subsequently the estimated parameters are saved in a data structure and the drawing acquisition starts.

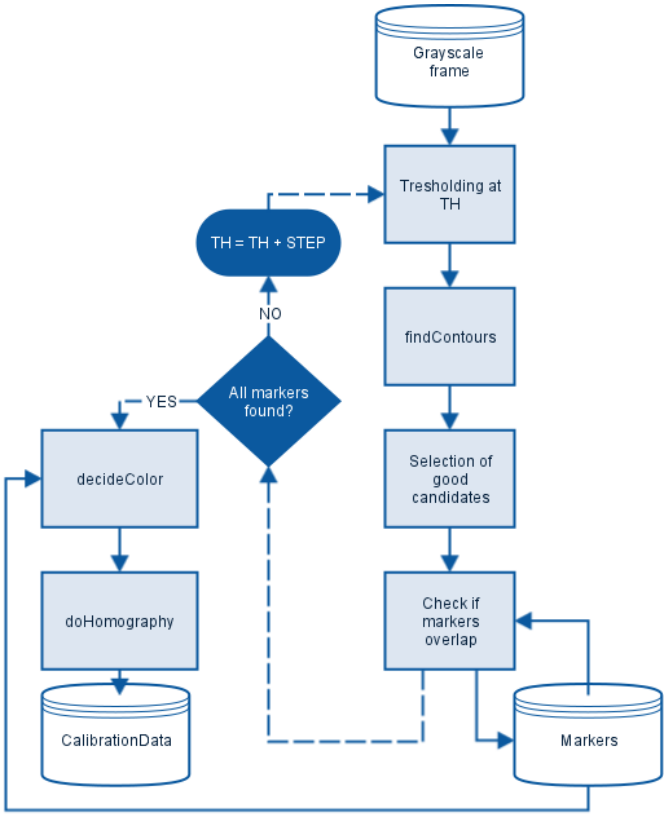


*Figure 2 Calibration procedure. Dashed lines represent only the operational flow. They do not carry any data.*

The paper identification is performed by searching for the markers that define it. The marker identification is performed by the function *findCrosses* as following. At first the frame is converted in grayscale; subsequently a binary threshold is applied and then using the *findContours* function all the possible contours are identified. After this operation we approximate the contours to a polygon. To filter out all the incorrect candidates we simply check some properties of the contours like convexity, number of edges and angles dimensions. After filtering the remaining contours are, with high probability, the correct markers.

It's important to say that executing this operation only once (i.e. using a single threshold) would make the markers identification extremely dependent from the scene illumination. To avoid this problem the operations described above are executed many times in order to adapt the threshold value. At every iteration the application executes the described operations, increases the threshold, and eliminates the identified crosses that overlap with the markers found at the previous iterations. The loop goes on until all the markers are identified or all the threshold values have been used. If the last condition is verified the calibration fails and the next frame is analyzed.

After the successful identification of the six crosses the function *computeHSV* estimates an HSV color value for each marker by computing an average of a $8 \times 8$ pixels region centered on the barycenter of the markers. After this, the function *DecideColor* identifies the black crosses as the one with lowest hue. The discrimination between red and green

crosses is performed by applying a threshold on the value component of the HSV color space.
Finally using the markers placed at the edges of the sheet (and ordered in a clockwise manner) the function *doHomography* estimates the homography transformation matrix. By applying this transformation to the frame we obtain a rectified version of the sheet of paper. It's worth noting that the transformation is not ideal and it introduces some distortions in the rectified version of the sheet. The solution of this problem will be described in the section of the frame processing. To improve performance (considering the hypothesis of no displacements of webcam and paper) the application uses the identified markers to crop all subsequent frames keeping only the region that contains the worksheet. It is important to say that this operation is applied at every frame before all the operation of the frame analysis. As a consequence the size of the image to analyze is reduced.
To have a visual clarification of the flow of the calibration process see Figure 2. The flowchart refers to the calibration procedure applied to a single frame.

### Frame processing

The frame analysis part refers to all the operations needed for the identification of the drawing. All the methods and procedures described in the following are used only after the calibration has completed successfully. From now on we will refer to the frame as the one cropped and converted to grayscale.
As described in the previous section the homography operation can introduce some distortions (i.e. blurring) in the rectified version of the frame. During testing we noticed that the distortions can impair the drawing identification especially in the areas where the acquisition angle is high. Indeed the blurring effect lower the values of the sketch and by doing so some part of it does not pass the threshold. The best solution to this issue is to perform the sketch identification on the non-rectified frame and to use the homography transformation only to display the result and to identify the buttons. By doing so the transformation is applied to a binary image and so the blurring is not a problem.
One of the most important step is the identification of the hand of the user to avoid the identification of the latter as a part of the drawing. To perform this task at each frame the application computes the absolute value of the difference between:

- the current frame,
- a reference frame in which the hand is not present. It is worth noting that this frame is acquired and stored immediately after calibration. To reduce the sensibility to variation of light condition the reference frame is re-acquired every time that the hand is not present.

By doing so the hand appears on the difference image with high values. An important things to underline is that with a simple difference also the drawing would appear as bright in the image and so it would disturb the hand detection process. To avoid this issue we use the image representing the identified sketch to set at 0 the part where the sketch is present. This is done by performing an AND operation.
After this the function *handIdentification* applies a threshold to the difference image. The result of this thresholding is a big white region representing the hand, some smaller region due to light variation and parts of the draw that are still not identified. To eliminate the unwanted region an erosion operation followed by 5 iterations of dilation with a $5 \times 5$ structuring element is applied. After this the methods finds the contours of all the region and keeps the only one with the biggest area. From the resulting area it generates two ROIs images (with size equal to the one of the frame):

- ROI defined by the bounding rectangle of the contour of the area,
- ROI defined by the convex hull of the contour of the area.

The first one is used in the phase of sketch detection while the second one is used to check if a button has been pressed. The choice of using a rectangular ROI, for the drawing detection, is driven by the fact that we preferred to partially sacrifice the reactivity in favor of robustness against the artifacts that can be generated by the erroneous identification of the hand as a part of the sketch. On the other hand, for the detection of the push of a button the ROI based on the convex hull is used to guarantee a greater precision. The button management will be analyzed in detail in a dedicated section.
After the hand detection the method *drawIdentification* performs the draw identification. First of all an adaptive threshold is applied. This operations identifies the possible parts of the sketch. Not all of the latter are real part of the sketch; some of them are artifacts generated by the hand or the pen. This is due to the fact that the hand identification procedure is not perfect. To eliminate this issue we developed a technique based on a buffer representing the identified sketch. At each frame the application uses the adaptive thresholding to identify the sketch. The result of this operation is a binary image where at 0 there are the pixels that do not belong to the sketch and at 1 the ones that may do. It's worth noting that the region defined by the ROI of the hand is set to 0. This image is added to the buffer by a simple addition and a counter is increased. This operation is performed a certain number of times defined by a maximum value *Bs*. When the counter reaches *Bs,* the result is an image that can be compared to a probability map where high values correspond to pixel with high likelihood of being part of the sketch. At this point a thresholding operation with threshold *Bs-1* is applied. By doing so all the pixels that have always been "classified" as sketch are identified as drawing while the other are discarded. In such a way the artifacts that appear for a number of times lower than *Bs* are discarded. It's worth noting that the function *drawIdentification* identifies the whole sketch every time the buffer is full but it add only the new parts to the digital paper. This is fundamental to give the possibility to the user to change the color of the line.
These methodologies, combined with the hand identification, strongly reduce the presence of artifacts. There are still cases in which some artifacts may appear. To face this problem a method *cleanDraw* has been defined. The method can be only called by the user via GUI. It simply performs an estimation (on a single frame) of the sketch on all the sheet and then sets to white all the pixels that are not part of the drawing.
The *drawIdentification* method generates two digital paper:

- a black and white version used to identify the new part of the sketch (in the following iterations) and the new buttons,
- a colored version used only for display the result.

Every time the counter reaches *Bs* the two images are rectified. It would be worthless to perform the homography operation at each frame because the two digital papers are update only every *Bs* frames.
Regarding the use of colors, every time the digital colored paper is updated the application uses the selected color to draw the new part. The color selection is performed by the user by pressing the personalize buttons on the top of the sheet. In the following sections the management of the latter will be analyzed.
To have a visual clarification see image 3. The flowchart refers to the frame processing applied to a single frame.

### Buttons management

The buttons management is performed by two methods:

- *findNewButtons:* identifies the new buttons and gives the user the possibility to assign a color to them via a GUI selection on a color palette,

- *buttonPressed*: checks using occlusion if a button is pressed and changes the current color.

From now on all the operation will be performed on the black and white rectified version of the draw. It's worth noting that, as described in the achievements section, only the upper part of the sheet is dedicated to the interactive functionalities and only so this part will be used for the buttons management operations.
The *findNewButtons* methods is called only when the draw is updated and when no hand is identified in the region of the buttons. This to prevent the identification of new (partial) buttons before they are completely drawn. To identify the possible candidates, the *findContours* method is applied. The search is applied only in the region where there are no buttons found in the previous searches. Subsequently it eliminates all the candidates that have a smaller area than a threshold or that are concave.

At this point it performs five operations for each new button:

- approximate the contour with a polygon,
- associate to the button an unique ID,
- add the region defined by the polygon to an image representing the buttons called *areaWithButtons*. The value of the pixels inside the region is equal to ID,
- add the bounding rectangle of the contours to a binary mask to avoid that the button will be re-identified in the following executions of *findNewButtons,*
- ask the user for color selection via a color palette image; if the user does not perform the selection the associated color is set to black.

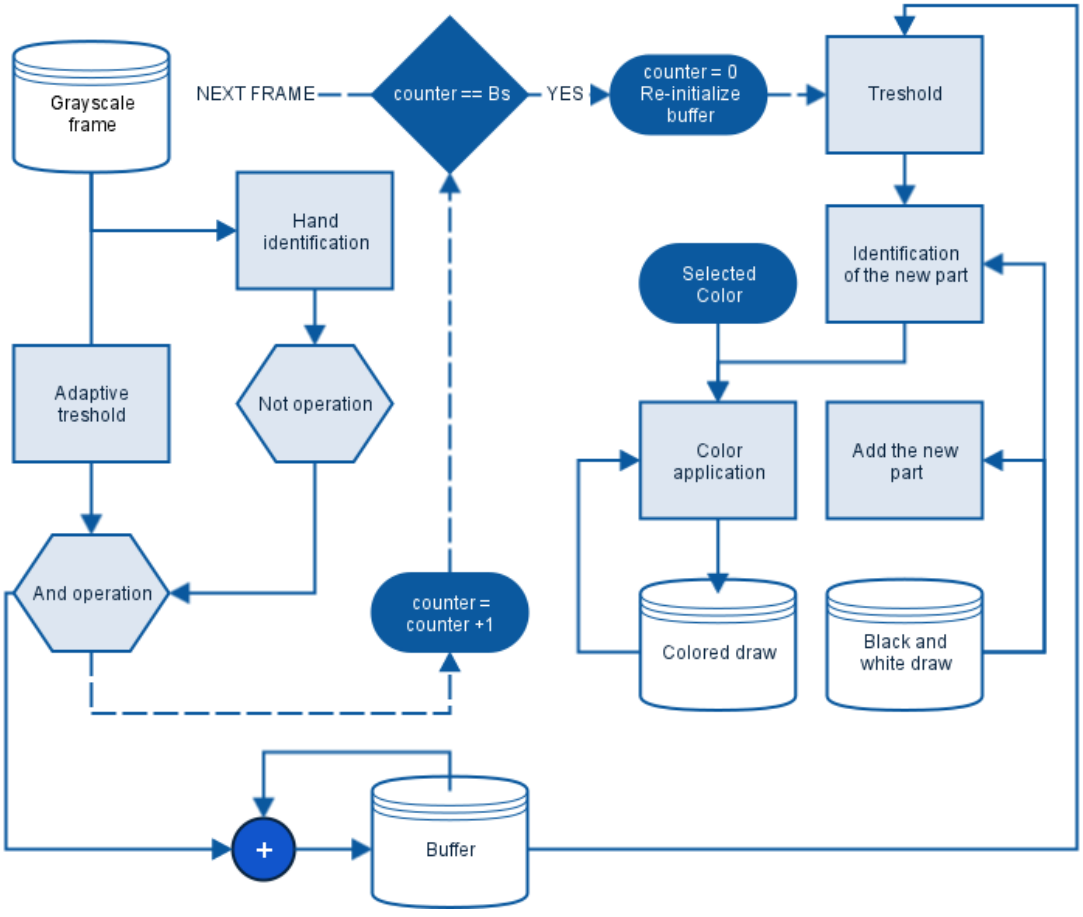


*Figure 3 Draw identification. Dashed lines represent only the flow. They do not carry any data.*

Every time the hand is identified in the region of the buttons the method *buttonPressed* is called. The method performs the bitwise AND between the convex ROI of the hand and *areaWithButtons*. The result of this operation is an image with zero values where the hand or buttons are no present or values corresponding to the ID of the buttons that are occluded by the hand. At this point the method checks how many unique values are inside the resulting image. The possible cases are three:

1. one unique value: the only possible value is zero and so it means that no button is pressed,
2. two unique values: one value is zero and the other is the ID of a single button,
3. three or more unique values: one values is 0 and the other are IDs of different buttons. In this case it is not possible to decide safely if the user wanted to press a button or not.

In the cases 1 and 3 the algorithm decides that no button has been pressed and so the previous selected color is maintained. Case two is the only one in which a correct selection is possible. To avoid unwanted selection the method checks the percentage of the total area of the button covered by the hand. If this percentage is greater than 80% the selection is confirmed and the color is set to the one associated to the button.

## Github repository

Link to github repository, https://github.com/mrban91/RealTimeDrawingSystem.git

## 4. Conclusions

As described in the previous sections all the objectives have been achieved except the robustness against sheet and camera movements. The digitalized version of the draw represents with high fidelity the real one.
Nonetheless there are many possibilities to improve the functionalities of the software. First of all it would be useful to avoid the artifacts generated by small sheet movements and vibrations by implementing some images-registration techniques. Secondly, the set of interactive functionalities can be extended in such a way to allow new line styles, filters and the insertion of graphic objects (e.g. images, shapes and so on and so forth). Moreover, the sheet identification procedure could be achieved by using less crosses. In order to do this it would be useful to estimate the position of the paper taking into account little movements (rotations, translations…). An innovative approach would be the estimation of the occluded crosses by using the visible one. This would probably make possible sheet and the camera movements without forcing the drawing position of the user. This would be definitely a positive turning point. Furthermore another improvement would be the integration of some OCR techniques both in the drawing area and in the interactive functionalities region.

## References

[1] OpenCV documentation, http://docs.opencv.org/