UNIVERSITY of RWANDA

COLLEGE OF SCIENCE AND TECHNOLOGY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

IMPLEMENTATION OF A REAL-TIME MANUSCRIPT SCANNER SOFTWARE FOR HANDWRITTEN LECTURING DIGITIZATION.

Undergraduate Final Year Project

Project Team Members: HABIYAREMYE NTWALI JANVIER: 218007332
MALIZA CONSTANTINE: 218005312
MBONIMPA PACOME SIMON: 218000124

Project Supervisors: Dr. Philibert Nsengiyumva
Mr. Gratien Muhirwa

February, 2022

Bachelor of Science (Honours) in Electronics and Telecommunication Engineering.

Project ID: **EEE/ETE/2020-2021/13**

APPROVAL AND CERTIFICATION

We confirm that this final year project has been done at the University of Rwanda, College of Science and Technology, School of Engineering, Department of Electrical and Electronics Engineering for the award of bachelor's degree in Electronics and Telecommunication Engineering, under the supervision of Dr. Philibert Nsengiyumva and Mr. Gratien Muhirwa. We also confirm that to the best of our knowledge, it is our original work. Contributions from other sources have been properly acknowledged.

Students:

1. Names: **HABIYAREMYE NTWALI JANVIER**
   Enrollment Number: **218007332** Signature:……….……….….………

2. Names: **MALIZA CONSTANTINE**
   Enrollment Number: **218005312** Signature:……….……….….………

3. Names: **MBONIMPA PACOME SIMON**
   Enrollment Number: **218000124** Signature:……….……….….………

Supervisors:

I. Names: **Dr. Philibert Nsengiyumva**

   Signature: ………………………………………

II. Names: **Mr. Gratien Muhirwa**

   Signature: ………………………………………

Head of Department, EEE:

Names: **Dr. Pierre Bakunzibake**

Signature: …………………………………

ACKNOWLEDGEMENT

We would like to express our gratitude towards our supervisors and lecturers for their guidance and constant supervision as well as their mentorship and support in completing the project. We acknowledge the efforts invested by the Department of Electrical and Electronics Engineering at the College of Science and Technology, University of Rwanda, for the sake of ensuring a satisfying curriculum of studies for future engineers.

Our thanks and appreciations also go to our colleagues, classmates, students and people who have willingly helped us with their abilities to deal with difficulties in this project.

ABSTRACT

This report describes our final year project entitled “Implementation of a Real-Time Manuscript Scanner Software for Handwritten Lecturing Digitization”. The report describes a system designed for tackling the problem of additional necessity of a digital whiteboard for remote lecturing purposes, in spite of the ubiquity of other personal electronic devices (smartphones, personal computers, cameras, etc.). The project aimed at addressing that problem by providing an emulation of a digital whiteboard using an A4 white paper, a camera and a computer software (powered by real-time image processing algorithms), thus cutting off the necessity of a digital whiteboard. The objectives of this project targeted specifically a software whose performance (in terms of output video frame rate, image enhancement and obstruction-free output) emulates a digital whiteboard output for the viewer. The research methodology and the software implementation relied on MATLAB toolboxes (Image Processing Toolbox, MATLAB App Designer and MATLAB Compiler) and hardware components: a personal computer and a camera. The outcome of the project was a software that complied with the set objectives but with limitations related to input image resolution, noise and illumination non-uniformity, perspective of the camera and output video file compression. The report concludes by recommending approaches for further improvement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

FETT	: Feature Extraction Through Time
HSI	: Hue, Saturation and Intensity
RGB	: Red, Green and Blue

1. INTRODUCTION

1.1. Background of the study

In the worldwide COVID-19 pandemic, many lecturers and students faced a sudden thrust into an online-only working environment, making paper document sharing and physical whiteboard usage difficult. Despite this move to an online-only environment, students enjoy the accessibility to oral and written lectures feature (as it was provided through the usage of physical whiteboards) but are not able to use such interaction. With the availability of digital cameras, smartphones and computers, digitization is becoming a simpler task.

Digital whiteboards (both physical and software based) can be used as an option in online lectures. They provide means of capturing video frames from the board, thus helping the recording and sharing of information. The physical digital whiteboards also provide means of accessibility when it comes to the need of handwriting and/or drawings, where the user interacts as if he/she is drawing on a real board. A challenge that consumers face with physical digital whiteboards is that they impose additional expenses besides other ubiquitous devices (like smartphones, digital cameras and laptops) that are also needed for remote collaborations.

So, we had an idea: what if we could digitize images of a white paper as it is being written on and turn that into a digital whiteboard-like system? The implementation of this idea had to benefit from the availability of simple devices (a smartphone/camera and a personal computer) and provide a computer software that could provide an emulation of a digital whiteboard. The project had to leverage image processing and computer vision programming libraries, and provide a software that can be interfaced to a camera connected to a computer, and be able to scan and enhance in real time the images of a white paper as it is being written on and also removing anything from the images that may appear obstructing any part of the white paper in the image (for example, the writer’s hand and its shadows).

1.2. Problem statement

Before the emergence of online-only lecturing, the normal way of teaching was to use a white board or black board so as to provide handwritten presentations and illustrations. The move to the online-only lecturing removed that accessibility that was provided through handwritten lectures. As it is shown in the background of this project, there are gaps left by the “handwritten lecturing digitization” solutions developed to tackle the challenge of remote handwritten lecturing. The developed solutions require manufacturing expensive high-resolution electronic devices (electronic whiteboards / screens, electronic pens, …) as additional devices to be interfaced with simpler personal devices (cameras, smartphones and laptops).

This research aimed at addressing that problem by providing an emulation of a digital whiteboard using an A4 white paper, a camera and a computer software, thus cutting off the necessity of an expensive digital whiteboard. This project also aimed at providing a solution whose performance (in terms of output video frame rate, image enhancement and obstruction-free output) emulates a digital whiteboard output for the viewer, as we had not yet found any known emulator that had attained such performance using only a camera and real-time image processing algorithms.

1.3. Objectives

1.3.1. General Objective

To design and implement a software that captures images of a white paper as the writer is writing and processes the images into a video that shows only the enhanced obstruction-free white paper and what is written on the paper (the pen strokes).

1.3.2. Specific Objectives

- To develop basic image capturing and processing functions using MATLAB’s Image Processing Toolbox.
- To determine and implement the most efficient image processing algorithms, to increase the output video’s frame rate.
- To apply the developed image processing algorithms in the implementation of a video processing standalone computer program using MATLAB App Designer and MATLAB Compiler.

1.4. Research Questions.

- Can a generalized image processing algorithm be implemented such that it processes raw white paper images (with pen inscriptions) into images of perfect white background with color-saturated pen inscriptions, despite random noises and illumination?
- By referring to the above question, can the efficiency of that algorithm be improved enough to run in real-time, in compliance with the human speed of writing and provide a real-time video of processed images, with a satisfactory frame rate for the viewers?
- Can the aspects of the algorithm be implemented into a graphical user interface software, with features comprehensible to end users, in terms of manipulation?

As we expected to get a positive answer for those question from the research, the hypothesis that had to be tested by this research was:
“It is possible to implement a comprehensible system that captures in real-time a video frames sequence of pen strokes inscription on a white paper and simultaneously, processes them into enhanced images, compensating for any pixel of the white paper obstructed from the camera by a significant opaque object, thus neutralizing the obstruction”.

1.5. Scope

1.5.1. Geographical scope

During the execution of this research, the reported image processing experiments were carried out in the environment of the Nyarugenge campus of the University of Rwanda, in Kigali, Rwanda.

1.5.2. Content scope

The project was limited to deploying basic spatial domain techniques of image processing including morphological/binary image processing. The resulting software was expected to provide features limited to image processing (image enhancement, noise removal and removal of parts obstructing the white paper using segmentation), a real-time full screen presentation of processed images for online screen sharing, and also audio and video recording by combining a recorded audio with the sequence of processed images to make a recorded video file.

1.5.3. Time scope

The implementation of this project took four months, as its proposal was approved on the 28th of September, 2021. The final report was expected to be presented in the last week of January 2022. All the activities related to this project were carried along with other academic activities done by the stakeholders.

1.6. Significance of the study

1.6.1. To the organization

This study was expected to not only contribute to the repository of research projects of the University of Rwanda, but also its success was expected to provide a tool (a software) that can be used widely by the lecturers as well as students.

1.6.2. To the researchers

The study that had to be covered by this project would contribute to the knowledge and hands-on skills possessed by the researchers, as it would challenge the researchers to study and apply digital image processing and programming skills, so as to reach the projected objectives.

1.6.3. To the future researchers

This project was expected to highlight critical aspects of image processing in terms of algorithm efficiency. Future researchers would learn from our methods of dealing with the challenge of time efficiency, and also how we dealt with the limitations we met.

2. LITERATURE REVIEW

2.1. Introduction

Even if the idea expressed in this report may sound as new to anyone, we were not the first people to deal with image processing as a tool for real-time emulation of a presentation board. Besides the fame of digital whiteboards, electronic pens or any other hardware developed for lecturing, there are other few solutions reported in various technical papers that tackled the same challenge, using cameras and image processing algorithms. In this chapter we are going to discuss about the solutions we found to be the most relevant, their critics and a conceptual framework we developed from them.

2.2. Review of Related Work

In [1], L. He and Z. Zhang presented a "Real-Time Whiteboard Capture and Processing Using a Video Camera for Remote Collaboration". This project aimed at scanning and enhancing images of pen strokes written on a whiteboard (not a white paper) in real time and filtering out the effect of the obstruction that occurs as the writer (lecturer) is moving between the whiteboard and the camera.

This system had a challenge: since the whiteboard is large, its contrast and light reflection would be influenced by the non-uniform illumination in the room which greatly depended on the weather and/or the direction of the sun rays entering through the windows of the room. In response to this challenge, the project was limited to assuming that the writer (lecturer) is not stationary for more than a period of four video frames, and also that for any small segment of the whiteboard, the pixels that belong to the whiteboard background are typically the majority. The approach proposed by the project was to process each input image by dividing it into a grid of segments, doing the analysis for each segment uniquely by categorizing and updating the obstructed segments and uniquely enhancing the segments impacted by the lighting in the room.

The segments were considered to contain the obstruction if the obstruction lasted less than a duration of four video frames. If the obstruction lasted longer, the obstruction was considered as new pen inscriptions. Therefore, the system failed to remove the obstruction in case the writer (the lecturer) was stationary in front of the camera, for a longer period. This segment-by-segment processing also cost a great time for processing each input image (which was 1.3 Megapixels of resolution), and it led to a low frame rate which was about 1 second per frame, using a computer of 2.4 GHz processor.

In [2], E. Coleshill, D. D. Stacey, and A. Ferworn presented a project entitled “Obstruction removal using feature extraction through time for video conferencing processing”. This project aimed at dealing with the challenge of shadow removal when a presenter obstructs light from a projector to the screen. Using a sequence of video frames taken through time, shadows would be detected and removed, leaving the content of the presentation material on the display without distortion. This would be accomplished by implementing an algorithm that compares intensity variations between consecutive video frames and decide which pixels are obstructed by shadows so as to compensate them with pixels from previous video frames when the section was not obstructed. This approach was called “Feature Extraction Through Time (FETT)”. The Feature Extraction Through Time approach the project relied on was adopted also in our project for the Obstruction Removal process. The aspect found from this project that made it different to ours was that the shadows on the screen are most of the time having an extremely low intensity, when compared to the light reflected by the screen. This indicates that the obstruction removal process in our project has to be a bit complex, since there would be cases where the skin color may impact the intensity of obstruction.

By reviewing [1] and [2], we recognized basic aspecs that the used methodologies relied on. First, the developed algorithms had to ensure that the target object (the whiteboard for [1] or the screen for [2]) was stationary with reference to the camera’s perspective, so as to be able to compare consecutive images and handle the new obstructions that emerged in. For our case, the target object was a whitepaper, which was expected to be moved accidentaly as the writer is writing on it. To ensure that the paper is in a stational position from frame to frame, we had to rely on measuring the correlation between samples taken from each two consecutive frames. A low correlation would indicate that the paper had moved, and thus restart the operations. As it was presented in [3], correlation coefficient calculation requires too much time for computation. Therefore, we had a challenge of developing a more time efficient solution. To emulate the output of a digital whiteboard, we had to also enhance images by increasing the saturation of the pen strokes. As it can be found in [4], conversion from Red-Green-Blue (RGB) to Hue-

Saturation-Intensity (HSI) color models (and vice-versa) requires non-linear computations that take too much time. This also imposes on us a challenge of inventing a time efficient alternative.

As it was reported in [1], the time delay for processing an input video frame would take about one second. This means that the frame rate of the system is approximately equal to 1 frame per second. As we had an idea of improving the performance, we had to set a minimum frame rate that our system had to achieve. Therefore, we referred to the data reported in a forensic related research paper [5], whose objective was to determine the highest speed of handwriting that can be achieved by a person, from a selected sample of people. It was reported that in undisputed police interview records, the highest speeds observed fell in the range 120–155 characters per minute. By conversion, that range is equivalent to about 2 - 2.58 characters per second. For our case, the targeted frame rate was to be set such that it would at least exceed the highest writing speed expected. Therefore, all the parameters used in the program had to be tuned to match an output frame rate of at least 3 frames per second, for a visually convenient image resolution to be determined through experiments.

3. METHODOLOGY.

3.1. Introduction.

Our idea arose from the fact that a human being is able to remember previous scenes of varying video frames from his/her temporary memory. To illustrate this, let us take an example (refer to Fig. 1. for illustrations): imagine that you are watching a video of an artist drawing a small labelled rectangle on a white paper, using a pen. Then after few seconds, the artist starts drawing a larger curve around that rectangle. As the artist draws the curve, his/her hand will roll over the labelled rectangle he/she previously drew, obstructing it from your perspective (view). Your temporary memory will keep reminding you that still there is a rectangle drawn, even when the artist's hand is obstructing it. Our project will rely on this idea of temporary memory, by providing an algorithm which will be able to compensate for obstructed pixels of a white paper being written on in a sequence of recorded video frames.

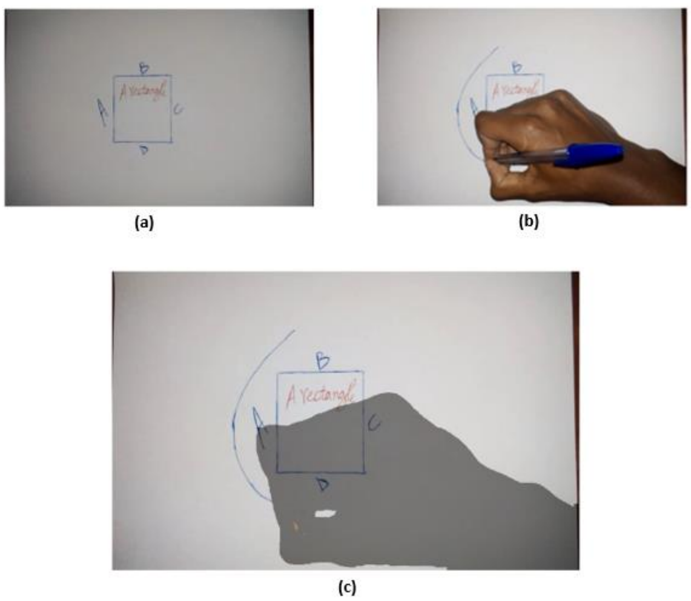


Fig. 1. Illustration of the image stored in a human temporary memory. (a) represents the initial image, (b) represents the obstructed image and (c) represents the perception of the obstructed image, referring to the temporary memory.

Technically, A video is a sequence of images called frames. Each frame is a digital image, a two-dimensional grid of pixels. By that definition, a video of a white paper being written on by someone is a sequence of digital images, where every image shows a state of the white paper throughout the sequence. The main objective of this project will be achieved by enhancing each frame and compensating for the white paper's area obstructed by the writer's hand or anything else. The compensation will be done through two segmentation processes: thresholding and masking. By thresholding, the large object (the hand, or something else between the camera and the paper) will be represented by zero intensity pixels, and by masking, those zero intensity pixels will be replaced by their corresponding pixels from the previous video frame when the hand was not hiding them. The output image will be appended to the sequence frames for the output video, and also be used to compensate for obstruction in the next frame. If the user wishes, all this process will be done while recording an audio which will be combined with the processed frames and produce a video of pen strokes being inscribed into a white paper, without showing any other thing else. The user will be able to manipulate the system using the software's Graphical User Interface, which will allow a live full screen presentation of the output.

3.2. Description of Tools Used During the Research and the Implementation.

This project was implemented using MATLAB R2021a edition software installed on a personal computer (a laptop). Computer program written in MATLAB file format were used to simulate and to assess the performance of the proposed algorithm. The used laptop has an Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz processor, a 64-bit Windows 10 operating system, Random Access Memory of 12 GB and a storage of 1TB. The used camera (webcam) had an input resolution of 8 Megapixels, and could also emulate lower resolutions up to 0.23 Megapixels. This project profited from different MATLAB tools, including MATLAB Image Processing Toolbox for digital image processing; MATLAB Timing Functions: tic(), toc() and timeit() functions to measure the time efficiency of algorithms, MATLAB App Designer and MATLAB Compiler to design and compile stand-alone applications for Microsoft Windows operating system.

3.3. Ideal Illustration of the Proposed Main Algorithm.

From MATLAB programming language, a "cell array" is a data structure with indexed data containers called cells, where each cell can contain any type of data. Let us consider the sequence of the captured input frames as a MATLAB cell called "images". In MATLAB syntax, a single input frame will be referred to as “images{n}” where ”n” is the number of the frame (this means that the first frame will be images{1}, the second will be images{2} and son on). As the software will command the camera to capture a video frame (image), the video frame will be processed through three consecutive main processes: Image enhancement, Mask creation and Obstruction removal. In Fig.2., a flow chart for this algorithm is illustrated.

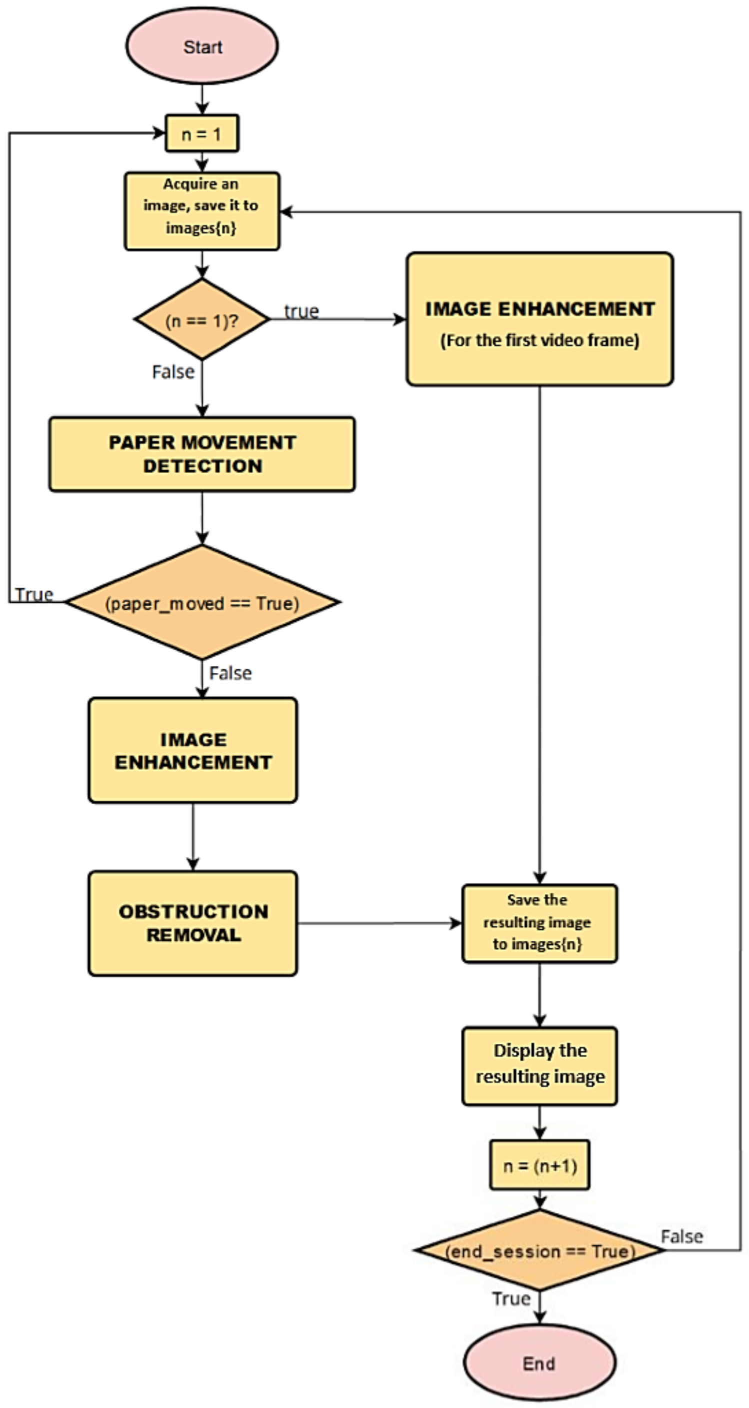


Fig. 2. The Proposed Main Algorithm.

3.3.1. Description of Paper Movement Detection Process

This process will help to ensure that the paper being written on is not being moved with respect to the camera. This will be achieved by computing correlation coefficients of sample data for every two consecutive image frames. A low correlation coefficient will indicate that the paper has moved, thus the algorithm will need to reset, and restart.

3.3.2. Description of the Image Enhancement Process

This process will truncate the input RGB (Red Green Blue) image so as to extract at the maximum the part showing the white paper, enhance the extracted part by increasing the saturation. The purpose of this process is to enhance the colors of the pen strokes inscribed on the white paper. The first frame will undergo its unique enhancement process so as to be transformed into a perfect frame to be a good reference to the next frame for obstruction removal. Fig. 3. illustrates an ideal result of the desired image enhancement process.

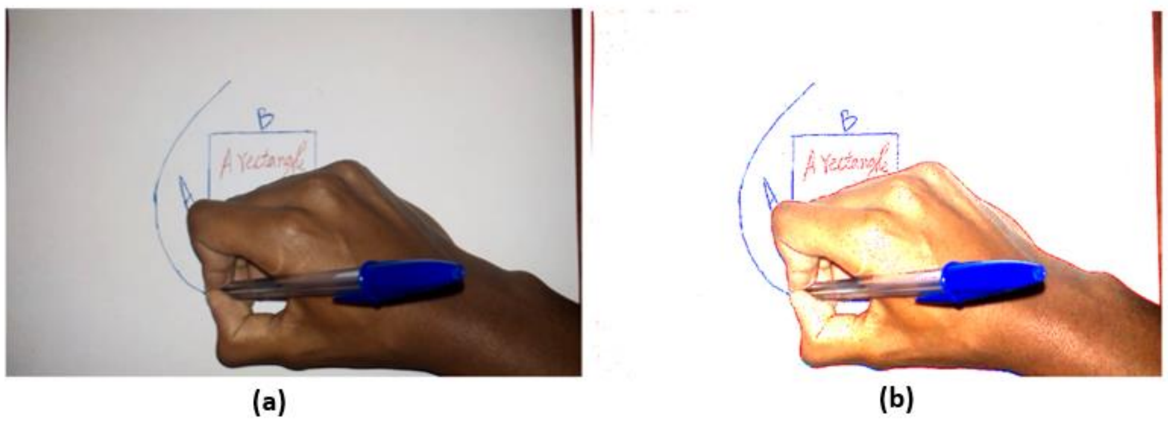


Fig. 3. Image enhancement process: (a) represents the raw image, (b) represents the enhanced image.

3.3.3. Description of the Obstruction Removal Process

This process will threshold the input image. The result will be a binary image, where the obstructed part will be having zero intensity. The binary image will be used in the next task as a mask indicating the obstructed part to be compensated for. Fig. 4. illustrates an ideal desired binary mask.

Fig. 4. The Binary Mask created after thresholding.

The process will use the enhanced image and the created mask, process them along with the previous frame, images{n-1} so as to compensate for the obstructed image, and provide a non-obstructed and enhanced image as the output. By using the illustrations from Fig. 5., let's consider the mask as a binary matrix, M and the inverse (the one's complement) of each digit in the mask as Mc. Since the pixels of matrix M where the white paper is obstructed have zero intensities, and the remaining pixels have intensity of 1, element-wise multiplication of matrix M with the enhanced images{n} will give an output with the obstructed part having intensity of zero. Let us call that output matrix, "A". By multiplying (element-wise) the previous image (frame) matrix (the one we called images{n-1}) with the complemented mask matrix, Mc, this will give an output with the non-obstructed part having intensity of zero, but the obstructed part of the white paper will be revealed. Let us call this output image matrix, "B". By adding the matrix B to matrix, A, the result will be the final desired image where the obstruction will have been removed. Fig. 5. and Fig. 6. illustrate well the obstruction removal approach explained above.

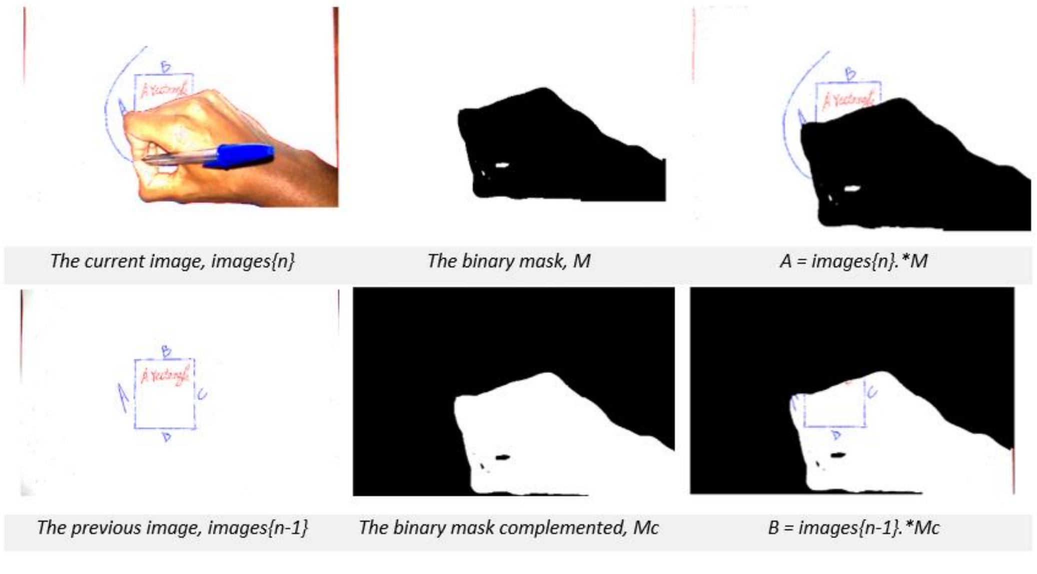

The current image, images{n}
The binary mask, M
A = images{n}.*M
The previous image, images{n-1}
The binary mask complemented, Mc
B = images{n-1}.*Mc

Fig. 5. Obstruction removal processes: Deduction of the obstructed part.

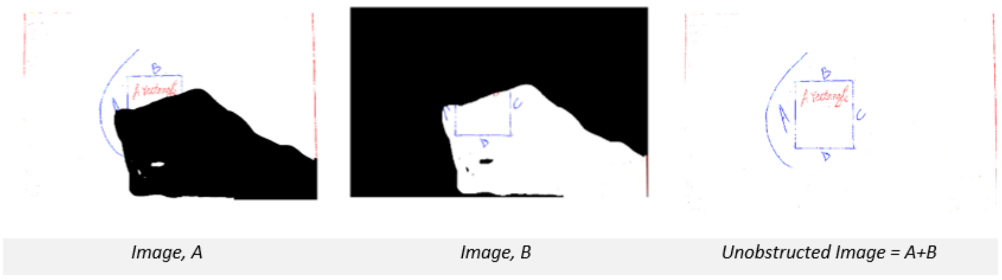


Fig. 6. Obstruction removal: compensating for obstructed pixels.

4. SYSTEM ANALYSIS, DESIGN, AND IMPLEMENTATION.

4.1. Introduction.

In this chapter, we are going to describe the proposed algorithms to deal with the basic parts of our implementation. Those parts include processes of Paper Movement Detection, Image Enhancement and Obstruction Removal. For each part (process), we will provide a flow chart indicating what the process should start with, how it will proceed and end. Explanations related to steps, together with sample output images are provided for illustration purposes. Note that the MATLAB code used during preparation for the 4.2, 4.3 and 4.4 sections is illustrated in Appendix A: ”MATLAB Code used in Illustrating the Main Algorithm’s Processes”.

4.2. Algorithm Design for the Paper Movement Detection Process.

To ensure that the obstruction can be removed, the system will need to first make sure that the paper has not been moved with respect to the camera's perspective. The flow chart shown in Fig.7. indicates where the Paper Movement Detection process belongs to in the main algorithm.

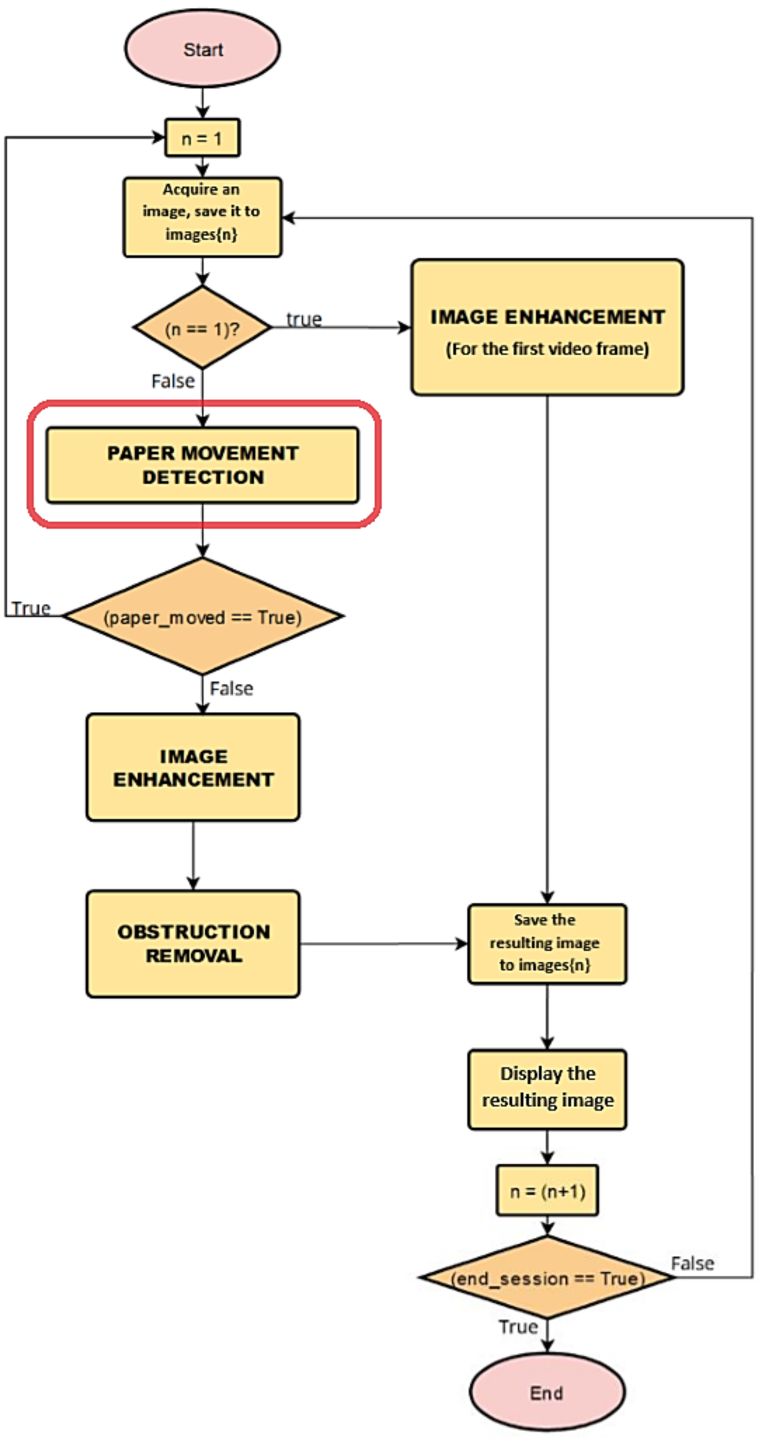


Fig. 7. Indication of where the Paper Movement Detection process belongs to in the Main Algorithm.

4.2.1. Methodology Illustration.

In this section, we are going to illustrate the algorithm of paper movement detection. The flowchart shown in Fig.8. illustrates basic information on the paper movement detection algorithm. Note that the indicated numbers are the number of processes (steps) that are going to be explained.

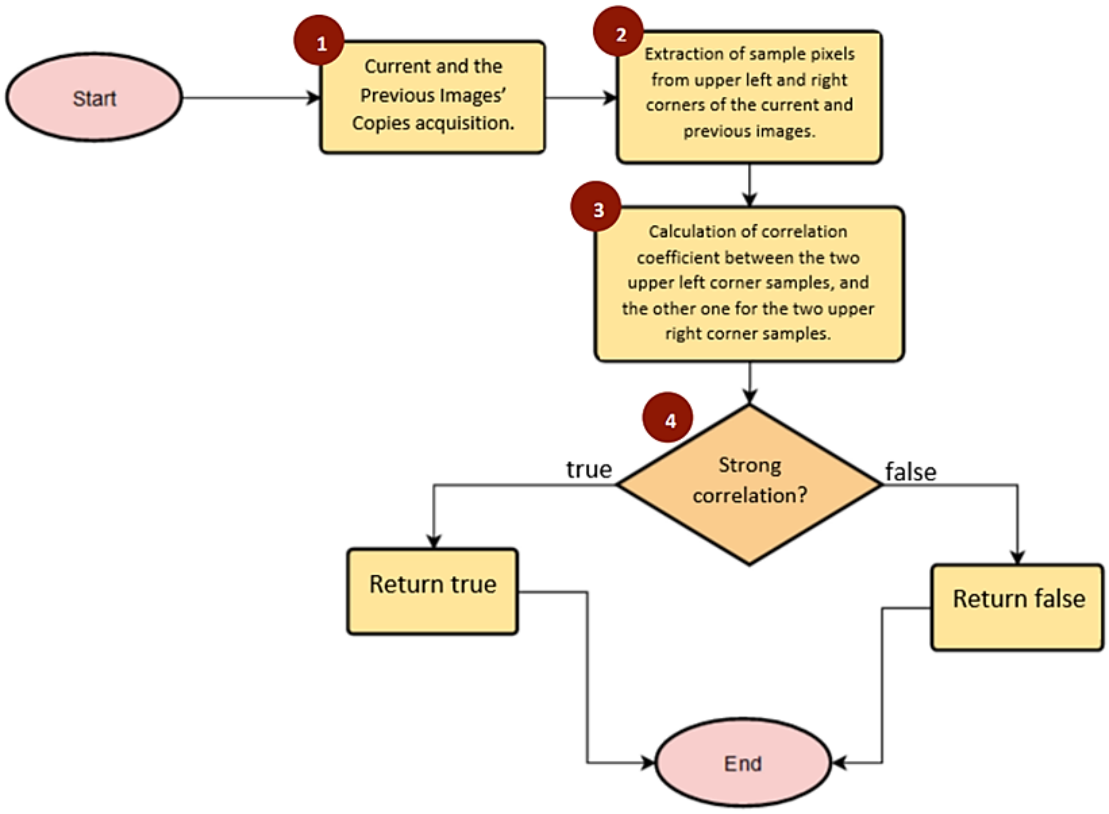

Start
1
Current and the Previous Images' Copies acquisition.
2
Extraction of sample pixels from upper left and right corners of the current and previous images.
3
Calculation of correlation coefficient between the two upper left corner samples, and the other one for the two upper right corner samples.
4
Strong correlation?
true
false
Return true
Return false
End

Fig. 8. Illustration of the proposed Paper Movement Detection algorithm.

Step 1: Current and Previous Images' Copies Acquisition.

In this demonstration, we will assume two images, where one represents a paper positioned well, and another represents the paper moved a bit from its initial position. Here at Step 1, this process will acquire a copy of the current image (frame) and the previous image (frame). It will also trim the images, so as to show the white paper section at maximum, as it is shown in Fig.9.

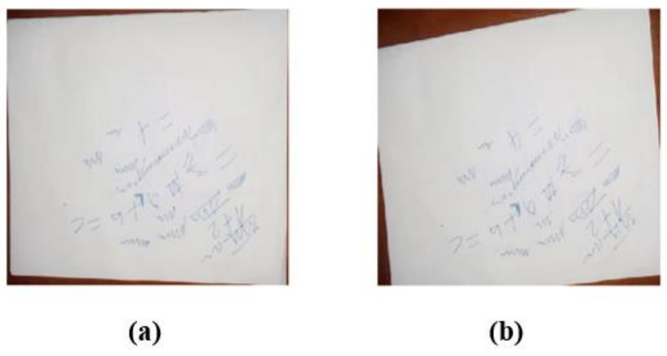


Fig. 9. Illustration of the previous image (a) which was acquired before moving the paper, and the current image, (b), which was acquired after moving the paper.

Step 2: Sample pixels extraction from the upper corners of the two acquired images.

This process will take samples from the upper left and upper right corners of the two images. The extracted segments' height and width will be ten percent of the height and width of their respective images. The acquired corner samples are shown in Fig.10.

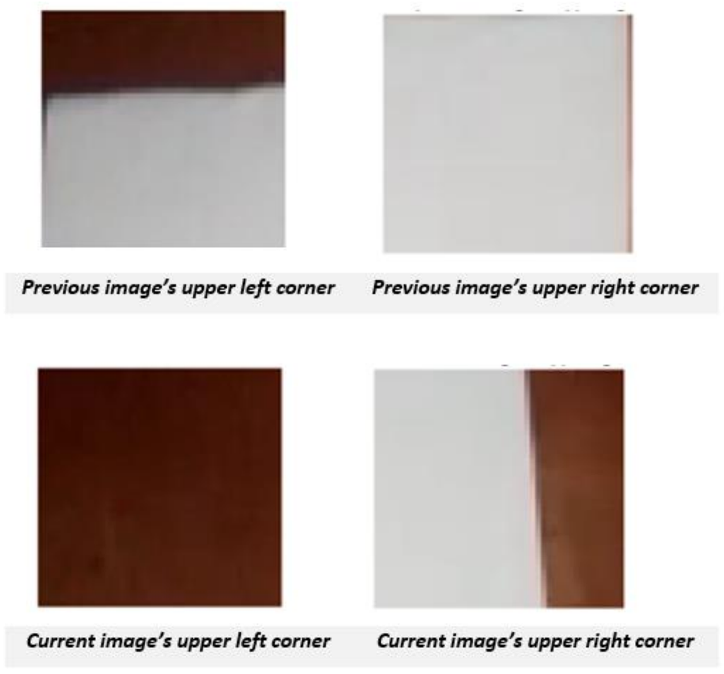

Previous image's upper left corner
Previous image's upper right corner
Current image's upper left corner
Current image's upper right corner

Fig. 10. Acquired samples from each image's upper corners.

Step 3: Correlation coefficients calculations between the equivalent corners of the two acquired images.

To enhance the contrast, this process will first equalize the taken samples. Next, it will transform them into single column vectors, then compute two correlation coefficients: correlation coefficient between upper right corner of current image and the upper right corner of previous image, then correlation coefficient between upper left corner of current image and the upper left corner of previous image. The results shown below were got from MATLAB after calculating the coefficients.

```
correlation_from_corner1 = 0.4252
correlation_from_corner2 = 0.6665
```

Step 4: Returning a result of a test for a high correlation coefficient.

For this process, the program has to decide whether or not the two correlation coefficients satisfy a minimum correlation coefficient. Normally, a correlation coefficient will range from 0 to 1, such that a value near to 1 indicates a high correlation. By experimentation, we found that in different conditions, 0.65 is the minimum coefficient of correlation that resulted when the paper was not moved. Therefore, this process will give a true answer if each of the two correlation coefficients satisfies the threshold condition of being greater than 0.65.

4.2.2. Algorithm Improvement.

Given that the calculation of correlation coefficient takes too much time due to its high computational complexity [3], we developed a new approach to tackle the challenge of time efficiency, as a real-time constraint. The new approach proposed had to start by thresholding the equalized pixel samples from the papers' upper corners. As various thresholding models rely on contrast enhancement ([6] and [7]), the image equalization will enhance the contrast significantly, to prepare the image for the thresholding. The process had then to compare the resulting binary patterns using the binary XOR operation. The resulting data would indicate which pixels have changed, and those that didn't change, if the paper had been moved. By experimentation, different trials showed that the maximum percentage of pixels that changed when the paper was not moved was 30%. This was due to the presence of noise that varied in different cases.

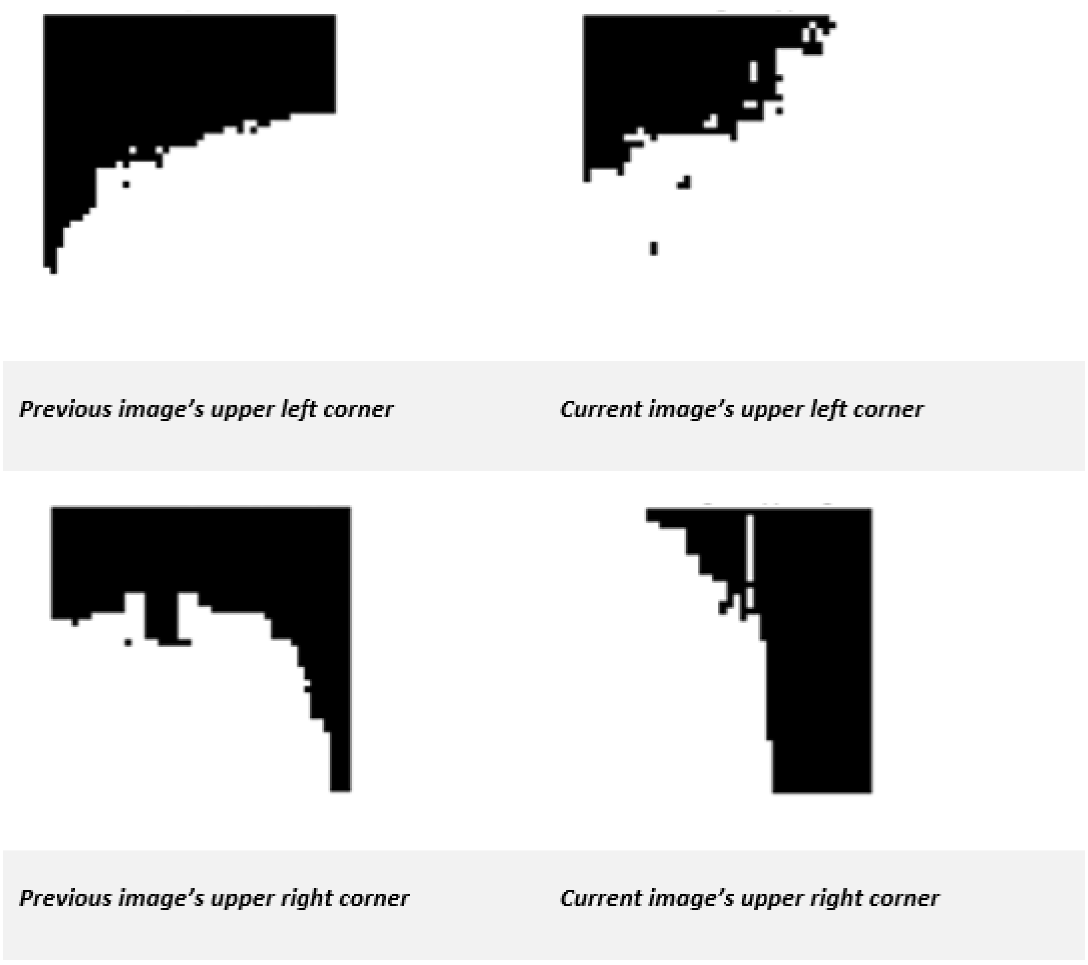

Previous image's upper left corner
Current image's upper left corner
Previous image's upper right corner
Current image's upper right corner

Fig. 11. Results after thresholding the acquired samples.

```
ratio_xor_from_corner1 = 0.1929
ratio_xor_from_corner2 = 0.3034
```

The "ratio_xor_from_corner1" and "ratio_xor_from_corner2" provides the fractions of number of pixels that changed, by referring to the samples extracted from the upper left and upper right corners, respectively.

4.2.3. Comparison of Time Usage Between the Correlation-Coefficient-Based Approach and the XOR-Based Approach.

In this section, we are going to compare the time usage of the two approaches: correlation-coefficient-based approach against the XOR-based approach. We will emulate the paper movement detection by using corner samples from images of 1 megapixel to 8 megapixels and graph the time it will take for the computer to process each image using each of the two approaches. For each approach, the computer will execute the step number 3, by considering the corner samples of each image as if they are from the current and the previous images (frames).

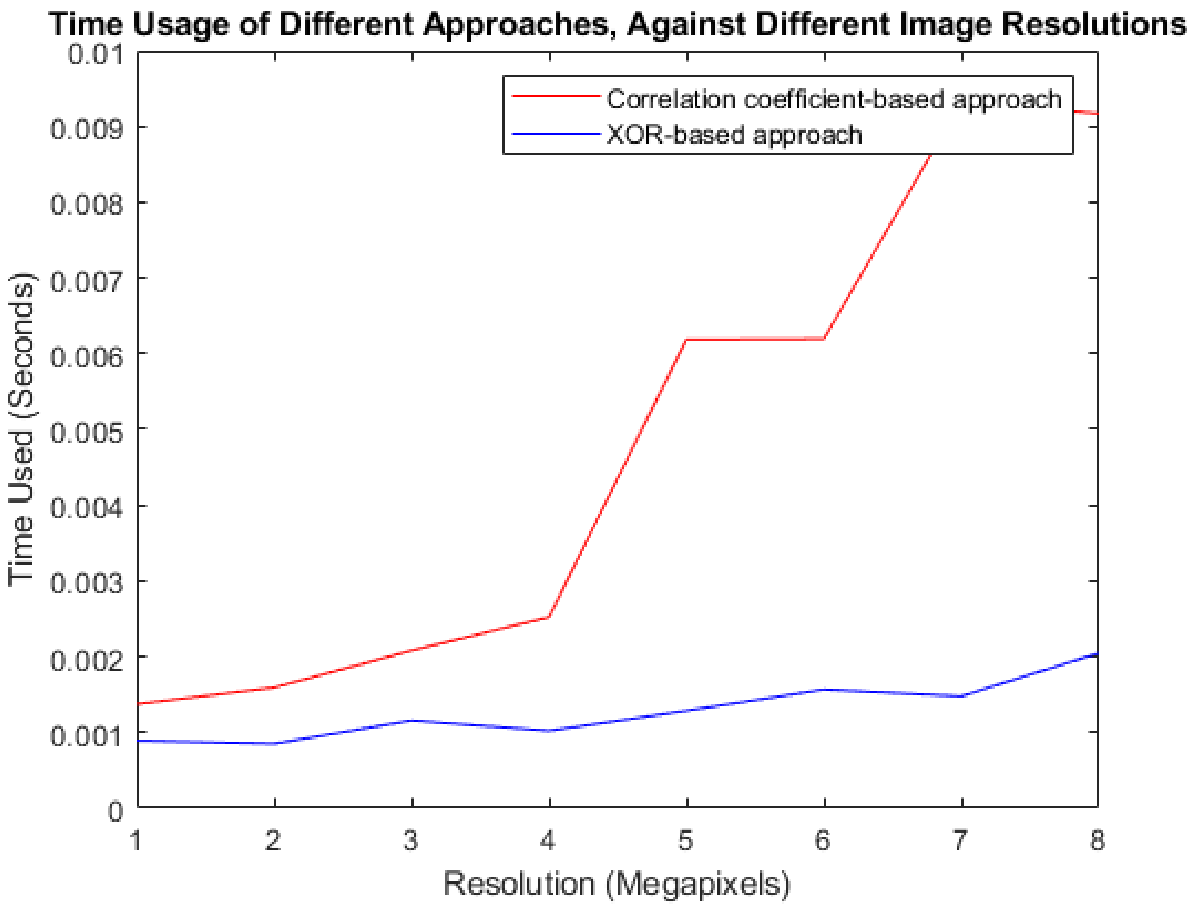


Fig. 12. Time usage of the Correlation-Coefficient-based approach and the XOR-based approach, with respect to different input images' resolutions.

From Fig. 12., it can be seen that the XOR-based approach uses less time than the correlation coefficient-based approach. As we targeted a minimum frame rate of 3 frames per second, about 0.333 seconds has to be allocated to the overall frame processing time. The XOR-based approach was chosen to be implemented since it requires less time which ranges from 0.001 seconds to 0.002 seconds (for input resolutions ranging from 1 megapixel to 8 megapixels), equivalent to the range of 0.3% to 0.6% of the minimum targeted frame processing time (which is 0.333 seconds).

4.3. Algorithm Design for the Image enhancement Process.

To ensure that the pen strokes are enhanced, the system will need to remove noises from the image and then enhance specifically the traces of the pen's ink. Fig. 13. shows a flowchart that indicates where the Image Enhancement process belongs to in the main algorithm.

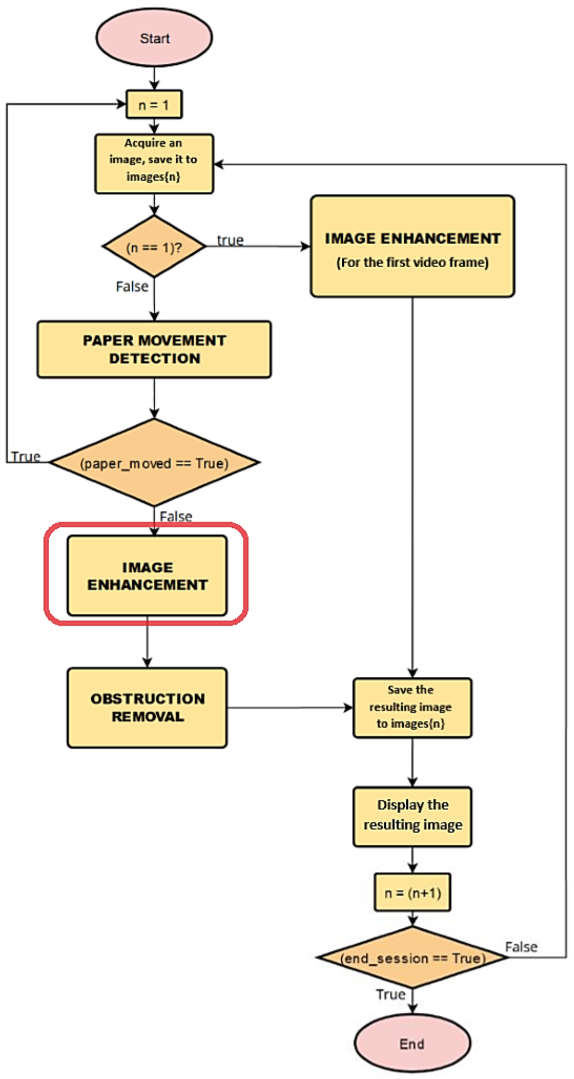


Fig. 13. Indication of where the Image Enhancement process belongs to in the Main Algorithm.

4.3.1. Proposed output of the "Image Enhancement" process:

It is desired that the Image Enhancement process provide an output image with these characteristics:

• The image should be truncated to show at maximum the enhanced white paper,

• The pen strokes on the image should be enhanced, with an increased saturation,

• The first frame will undergo its unique enhancement process so as to be transformed into a white frame and be a good reference to the next frames.

4.3.2. Methodology Illustration.

In this section, we are going to illustrate the algorithm of image enhancement. The flow chart shown in Fig. 14. presents basic information on the algorithm used in this presentation for the process of image enhancement. Note that the indicated numbers are the number of processes (steps) that are going to be explained.

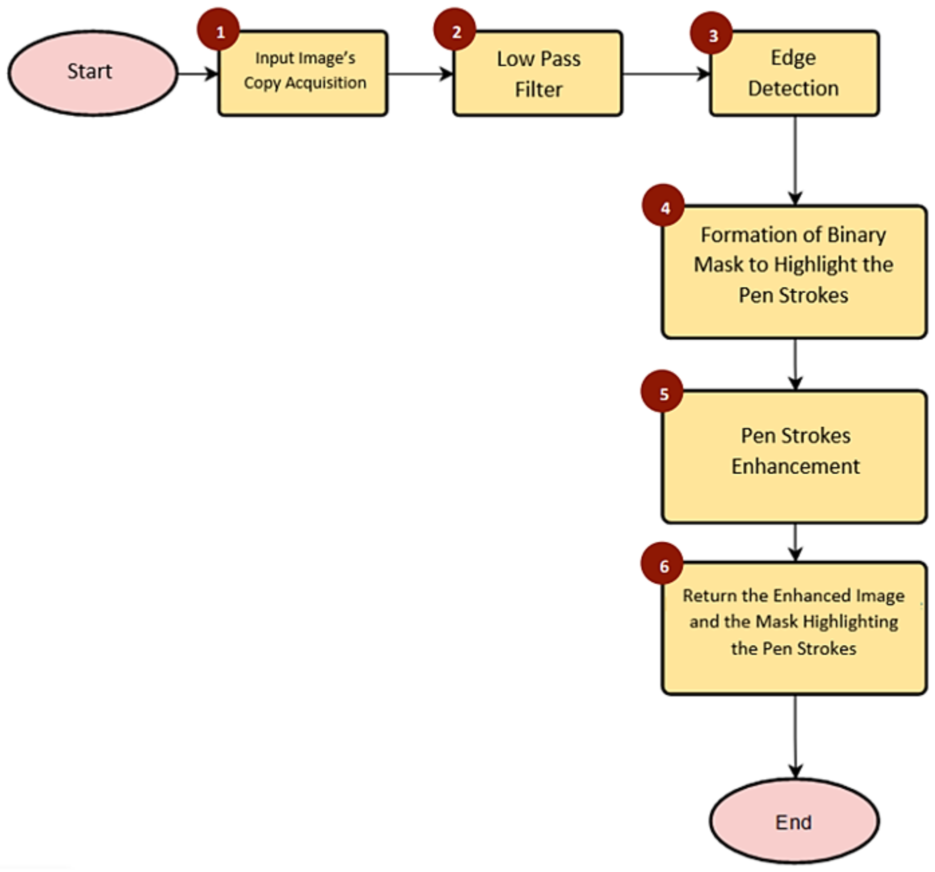

Start
1
Input Image's Copy Acquisition
2
Low Pass Filter
3
Edge Detection
4
Formation of Binary Mask to Highlight the Pen Strokes
5
Pen Strokes Enhancement
6
Return the Enhanced Image and the Mask Highlighting the Pen Strokes
End

Fig. 14. Illustration of the proposed Image Enhancement algorithm

Step 1: Acquisition of the Input Image's Copy.

At this step, the program will acquire a copy of the input image and trim it so as to get at maximum the part that shows the white paper.

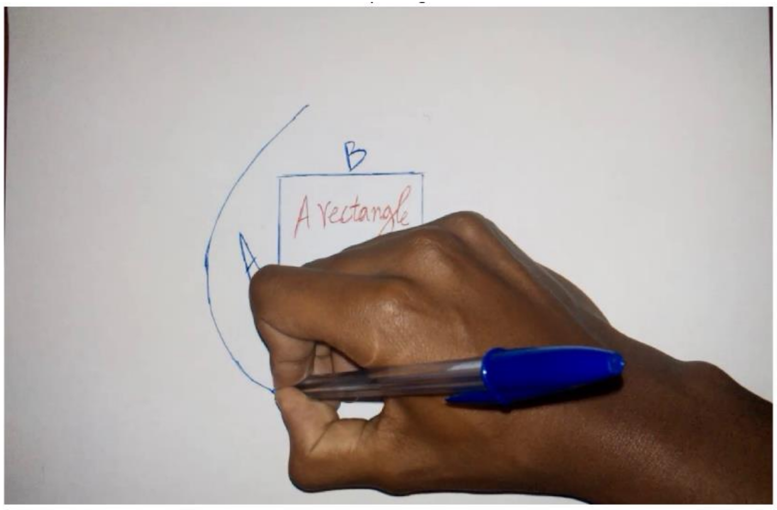


Fig. 15. The acquired input image.

Step 2: Image smoothing using a low pass filter.

This process will use an averaging filter (a low pass filter) to filter out noises in the image after converting it to grayscale. As it was stated in [8, p. 152], a major use of averaging filters is in the reduction of “irrelevant” detail in an image. By “irrelevant” we mean pixel regions that are small with respect to the size of the filter mask. Therefore, noting that the size of the filter matters, this process uses a filter whose height and width are 0.3% of the height and width of the input image (which means that the area of the filter is 0.09% of the area of the input image). That factor of 0.3% was selected after numerous try-and-see image enhancement experimentations.

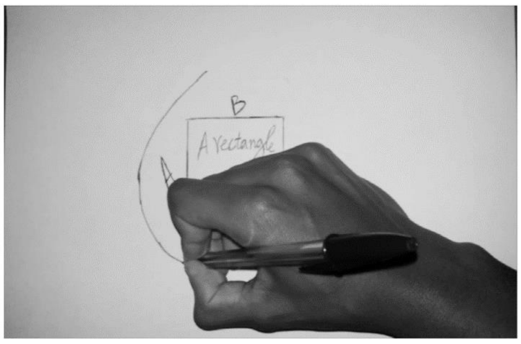


Fig. 16. The input image smoothed after transforming it into a grayscale image.

Step 3: Edge Detection.

This step will use a 3x3 Laplacian filter to detect edges in the image, then threshold the output so as to highlight areas where the ink has traced. Since the pen strokes can have any direction, we need an edge detector that is isotropic. As it is stated in [8, p. 699], the Laplacian detector is isotropic, so its response is independent of direction (with respect to the four directions of the Laplacian mask: vertical, horizontal, and two diagonals).

```
Laplacian_Filter = 3×3
      1     1     1
      1    -8     1
      1     1     1
```

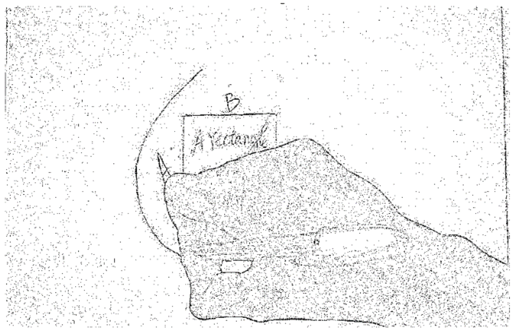


Fig. 17. The detected edges.

Step 4: Formation of a binary mask to highlight the pen strokes.

This step will do a dilation morphological operation so as to remove remaining noises. As it is proved in [8, p. 632], the used structuring element's size matters. Therefore, this process uses a structuring element whose height and width are 0.2% of the height and width of the input image (which means that the area of the filter is 0.04% of the area of the input image). That factor of 0.2% was selected after numerous try-and-see image enhancement experimentations.

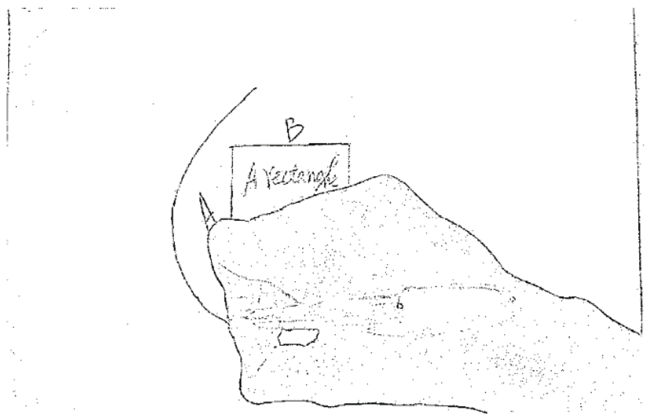


Fig. 18. A binary mask highlighting the pen strokes, excluding noises.

Step 5: Pen Strokes Enhancement.

This process will do an element-wise multiplication between the input image and the created mask (complemented), so as to highlight the pen strokes. To increase the saturation of the pen strokes, the resulting image was converted to the HSI model, and the saturation dimension was multiplied by a relatively large number (10 for our case). After the enhancement, the image was re-converted to RGB for the next steps.

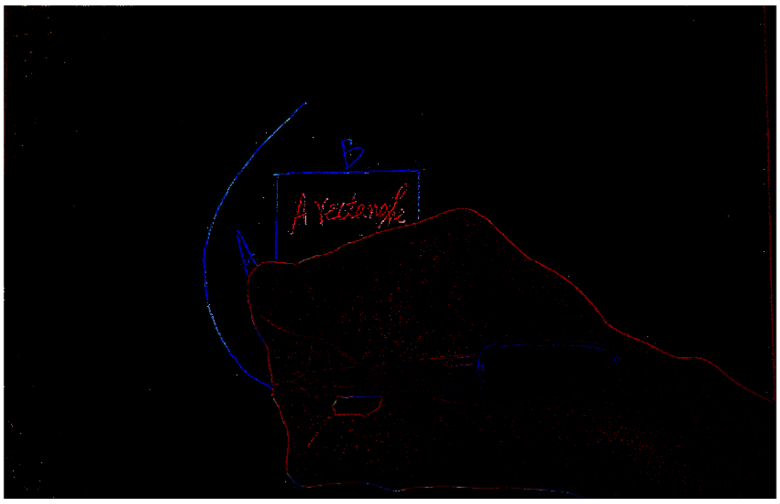
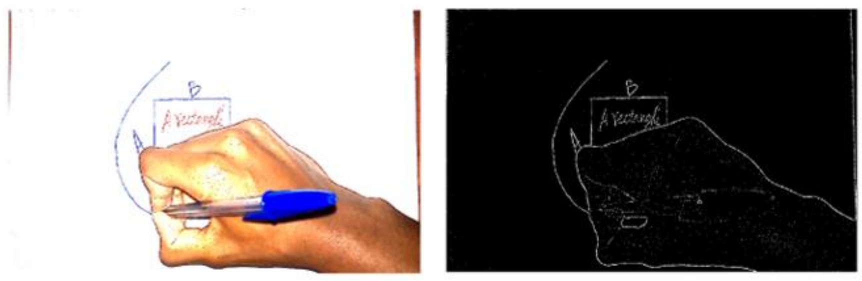


Fig. 19. Pen strokes enhanced.

Step 6: Returning the enhanced image and the binary mask.

This process will set the remaining part (the part which does not represent pen strokes) to higher intensity levels, by multiplying it with a constant. The part is then added to the part that represents pen strokes. After that, it will return the enhanced image and a binary mask to be later used in obstruction removal process, so as to protect pen strokes from being considered as obstruction (see the next topic: Obstruction Removal)



Enhanced Image	***Pen strokes binary mask (to be used in the next process: Obstruction Removal)***

Fig. 20. The returned data after the Image Enhancement process.

4.3.3. Algorithm Improvement.

By experimentation, it was found that step 5 (Pen strokes Enhancement) took too much time for execution, due to the conversion from RGB to HSI and vice-versa. The process of RGB-HSI conversion requires complex calculations such as trigonometric and inverse trigonometric functions. These calculations result in low algorithmic efficiency (time efficiency). For this reason, various approaches to improving the RGB to HSI conversion has been proposed, as it can be found in [4]. Given that our case needed to increase only the saturation parameter, we developed our own approach so as to tackle the challenge of time efficiency, as a real-time constraint. The new approach proposed was to divide by three the sum of the Red, Green and Blue components' intensities, so as to get the average intensity for each pixel. Next, the original intensities of the Red, Green and Blue components were compared to the resulting average so as to determine which intensities are major in each pixel. The intensities that were found to be greater than the average were set to the maximum intensity (which is 255), while the intensities that were found to be less than the average were set to the lowest intensity (which is 0). Fig. 21. shows the resulting enhanced image after using this new (improved) approach.

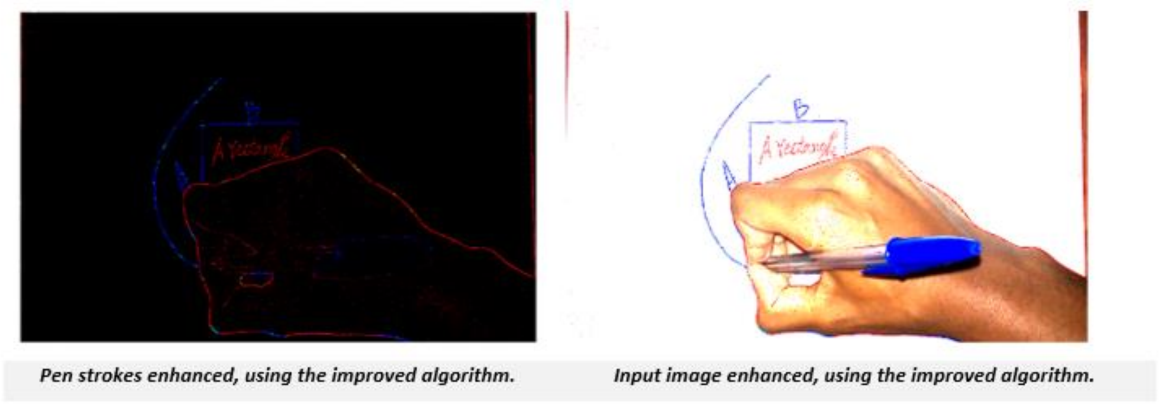

A Yectangl
B
A
A Yectangl
B
A
Pen strokes enhanced, using the improved algorithm.
Input image enhanced, using the improved algorithm.

Fig. 21. Results after improving the image enhancement algorithm.

4.3.4. Comparison between the HSI-based approach and the improved approach.

In this section, we are going to compare the time usage of the two approaches: the HSI approach against the improved approach. We will emulate the saturation enhancement process of the system by using sample images of 1 megapixel to 8 megapixels and graph the time it will take for the computer to execute each dataset using each of the two approaches. For each approach, the computer will execute the step number 3, assuming that the given image represents the RGB image to be processed.

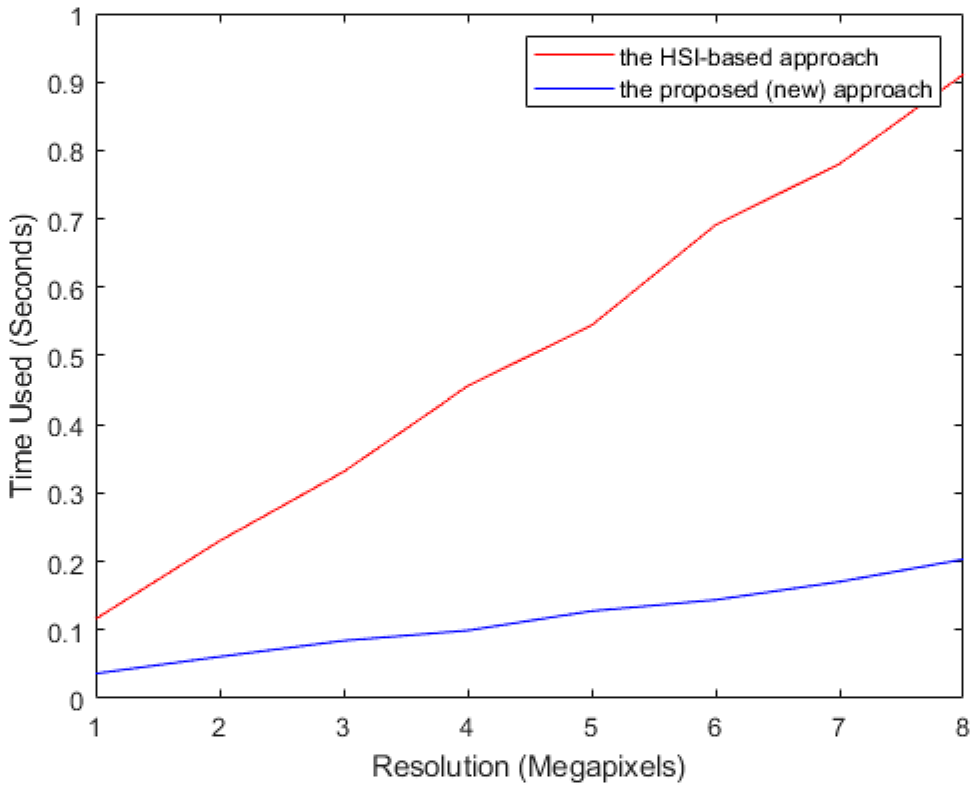


Fig. 22. Comparison between the HSI-based approach and the new (improved) approach.

By referring to Fig.22., the new (improved) approach uses less time than the HSI approach. As we targeted a minimum frame rate of 3 frames per second, about 0.333 seconds has to be allocated to the overall frame processing time. The new (improved) approach was chosen to be implemented since it requires less time which ranges from about 0.04 seconds to 0.2 seconds (for input resolutions ranging from 1 megapixel to 8 megapixels), equivalent to the range of 12% to 60% of the minimum targeted frame processing time (which is 0.333 seconds). The HSI-based approach was rejected as it requires much time that even exceed the minimum targeted frame processing time (which is 0.333 seconds) for resolutions higher than 3 megapixels.

4.4. Algorithm Design for the Obstruction Removal Process.

After the Image Enhancement process, the system will need to compensate for the obstructed part of the input image. Fig. 23. illustrates a flowchart which indicates where the Obstruction Removal process belongs to in the main algorithm.

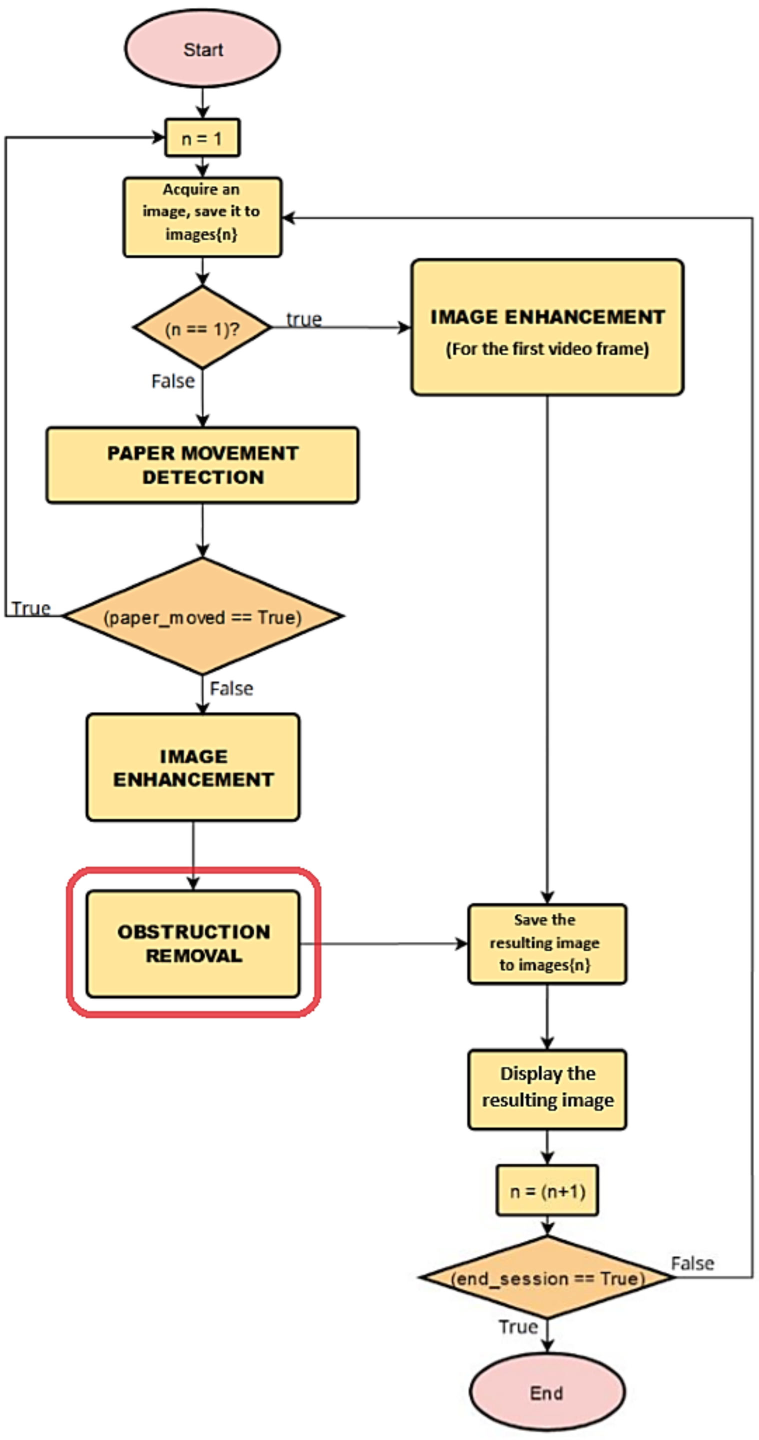


Fig. 23. Indication of where the Obstruction Removal process belongs to in the Main Algorithm.

4.4.1. Methodology Illustration.

In this section, we are going to illustrate the algorithm of obstruction removal. The flowchart illustrated in Fig. 24. shows basic information on the Obstruction removal algorithm that we are going to explain here below. Note that the indicated numbers are the number of processes (steps) that are going to be explained.

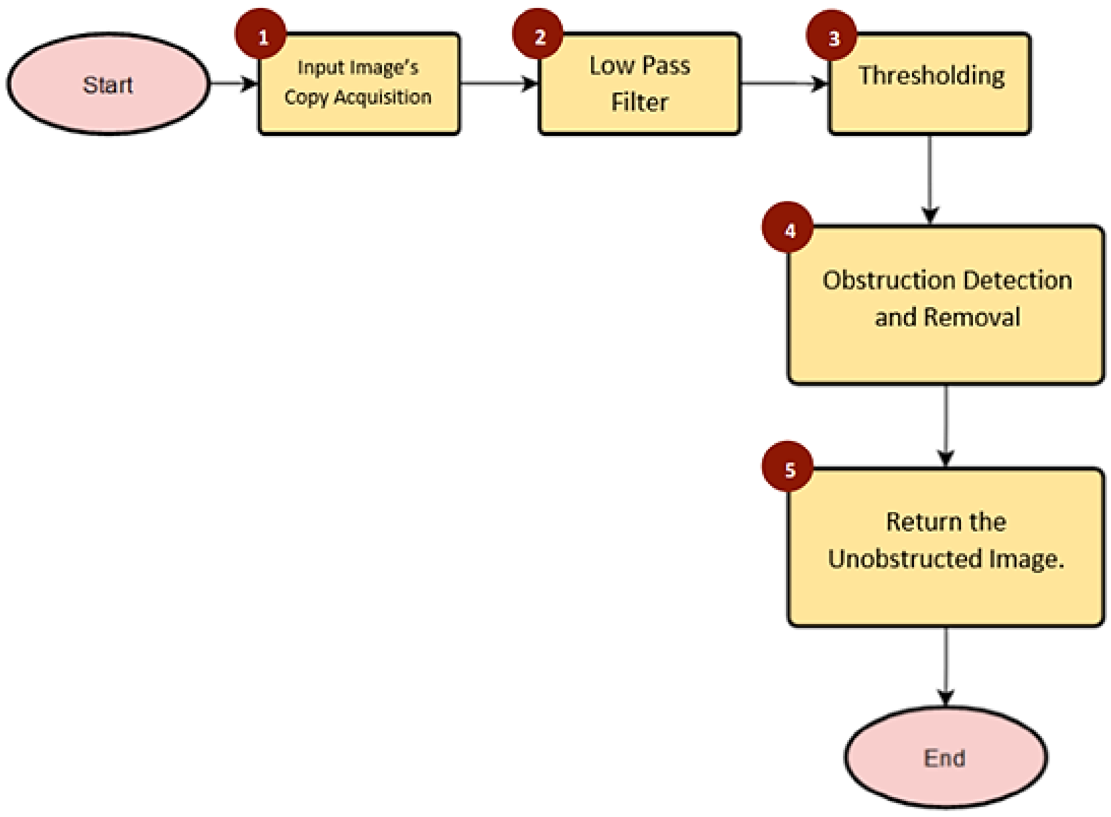

Start
1
Input Image's Copy Acquisition
2
Low Pass Filter
3
Thresholding
4
Obstruction Detection and Removal
5
Return the Unobstructed Image.
End

Fig. 24. Illustration of the proposed Obstruction Removal algorithm

Step 1: Input Image's Copy Acquisition

At this step, the program will read the input image and trim it so as to get at maximum the part that shows the white paper.

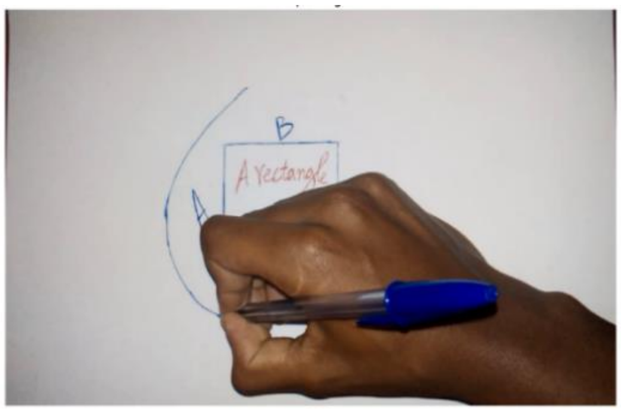


Fig. 25. the input image.

Step 2: Image smoothing using a low pass filter.

This process will use a low pass filter to filter out any object that appears in the image, except only the obstruction (the hand). As it was stated in [8, p. 152], a major use of averaging filters is in the reduction of "irrelevant" detail in an image. By "irrelevant" we mean pixel regions that are small with respect to the size of the filter mask. Therefore, noting that the size of the filter matters, this process uses a filter whose height and width are 1.5% of the height and width of the input image (which means that the area of the filter is 2.25% of the area of the input image). That factor of 1.5% was selected after numerous try-and-see image enhancement experimentations.

The process will use the binary mask from the image enhancement process so as to highlight pen strokes, and make sure that they do not contribute to the obstruction pixels by setting them to the highest intensity value, 255 (This will also prevent the effect of unequal illumination from obstructing the pen strokes). Then the resulting image will be filtered by smoothing.

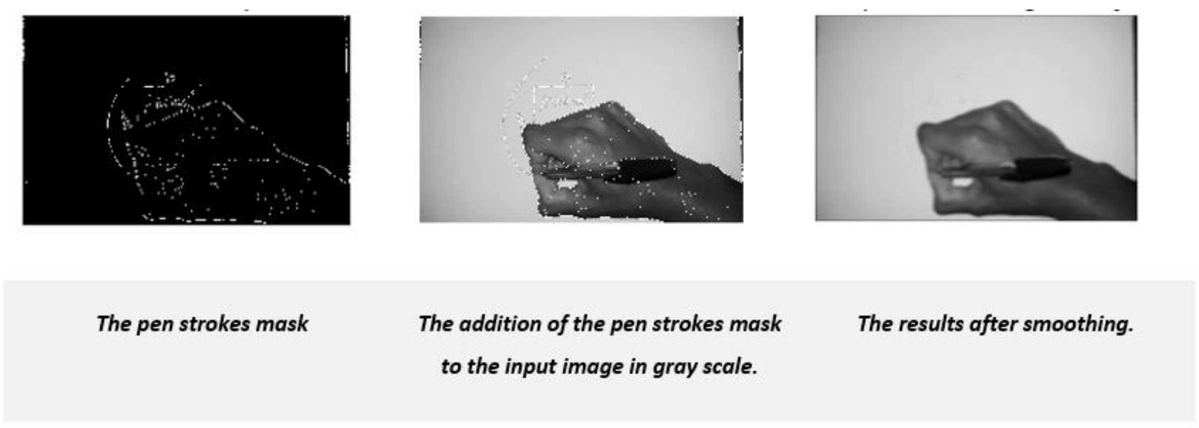
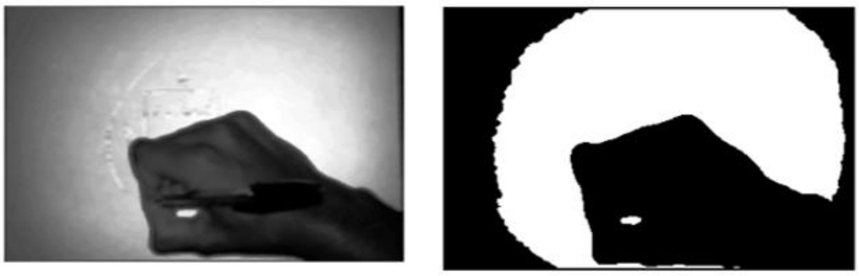

The pen strokes mask
The addition of the pen strokes mask to the input image in gray scale.
The results after smoothing.

Fig. 26. The process that leads to smoothing, ensuring that the pen strokes are protected from obstruction.

Step 3: Thresholding

This process will threshold the smoothed output image from step 2. As various thresholding models rely on contrast enhancement ([6] and [7]), the image equalization will enhance the contrast significantly, so as to prepare the image for the thresholding. By thresholding the image, the process' results will make sure that the obstruction will be represented by 0 intensity pixels.



The equalized image

The Results of thresholding: the Obstruction Mask

Fig. 27. The process of thresholding.

Step 4: Obstruction Removal.

This process will replicate the created obstruction mask into three dimensions so as to multiply (element-wise multiplication) with the input image, thus remove the obstruction by setting the obstruction to zero intensity. The zero intensities will be compensated by the previous frame through addition.

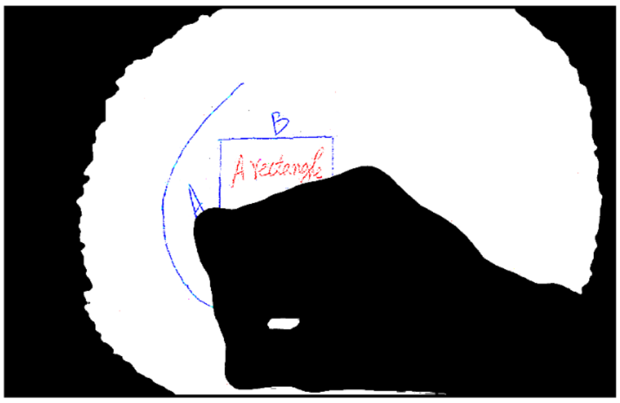
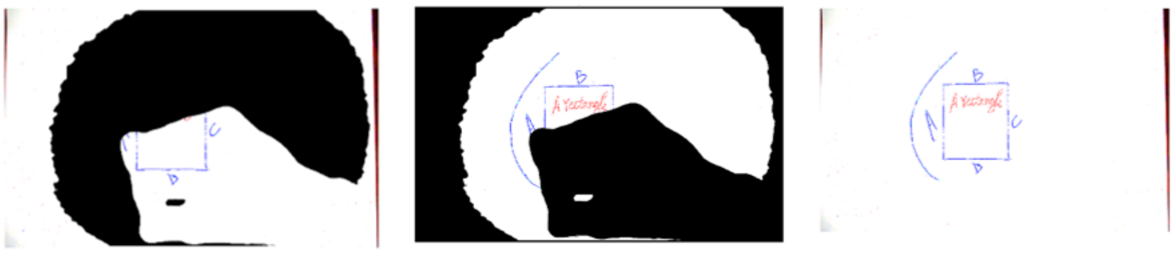


Fig. 28. The obstructed pixels' intensities are set to zero.

Even though the illumination correction is a challenge, all the pen strokes were protected from obstruction. This was due to that, at step 2, we used the pen strokes mask to protect the pen strokes.

Step 5: Return the output image.

This step will return the output unobstructed image to the main program. It will compute the image by compensating the obstructed part by the pixels of the previous image.



(a) The obstructed pixels are acquired from the previous image

(b) The non-obstructed pixels are acquired from the current image

(c) The final obstruction-free image is got as the addition of (a) to (b)

Fig. 29. The final step that generates an obstruction-free output image.

4.5. Frame Rate Related Issues and Code Optimization.

As it is mentioned in the introduction, the goals of this project were set with an envision that it will provide a new alternative for digitized lecturing. Therefore, the time efficiency of the developed algorithms should match with real-time constraints of handwriting. To tackle this challenge of frame rate determination, we referred to the data reported in a forensic related research paper [5]. The objective of that research was to determine the highest speed of handwriting that can be achieved by a person, from a selected sample of people. As it was reported, in undisputed police interview records, the highest speeds observed fell in the range 120–155 characters per minute. By conversion, that range is equivalent to about 2 - 2.58 characters per second. For our case, the targeted frame rate was to be set such that it would at least exceed the highest writing speed expected. Therefore, all the parameters used in the program were tuned so as to match an output frame rate of at least 3 frames per second, for a convenient image resolution. By experimentation, the highest resolution that complied with that constraint was found to be 2 Megapixels. This means that the frame rate would increase if lower resolutions were used, but that would sacrifice the image quality if the resolution was too much lower. The lowest resolution that we found to be tolerable was 0.5 megapixels. Fig. 30. shows a graph that indicates the recorded average frame rate achieved with respect to the resolution of input images. Fig. 31. shows screenshots of outputs of different resolutions. Note how artefacts appears as the input resolution diminishes.

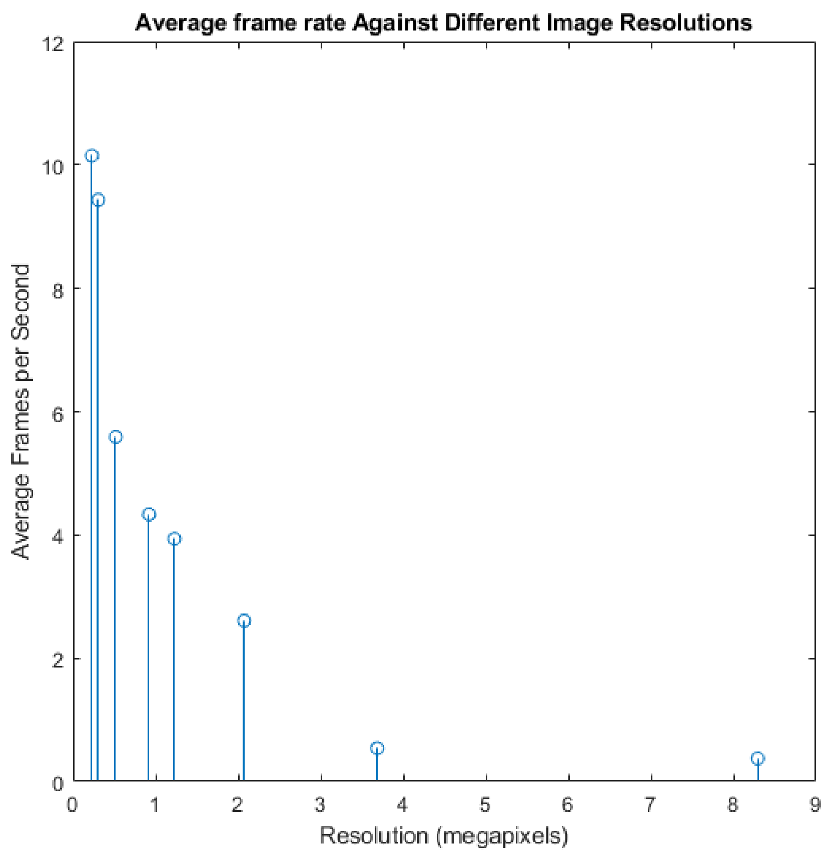
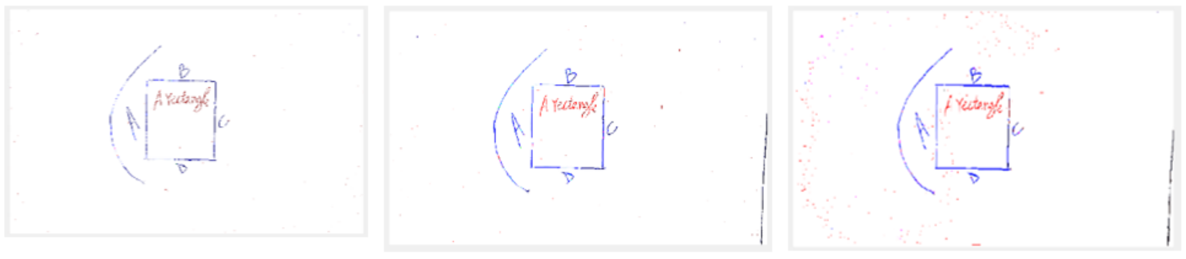


Fig. 30. Average frame rate against different image resolutions. Two Megapixels resolution was found to be the maximum resolution whose output video frame rate would reach in the proposed range.



Output of a 1920x1080 resolution input. ***Output of a 960x540 resolution input.*** ***Output of a 640x360 resolution input.***

Fig. 31. Screenshots of outputs of different resolutions. Note how artefacts appears as the input resolution diminishes.

Regarding the code optimization challenge, we had to rely on MATLAB programming language's syntax. As it is suggested in [9], we had to consider two important approaches for MATLAB code optimization: pre-allocating arrays and vectorizing loops. Here, the term "pre-allocation" refers to initializing the variables (arrays) with values before entering a "for" loop that computes and assigns elements to the array. Vectorization in MATLAB refers to techniques for eliminating loops altogether, using a combination of matrix/vector operators, indexing techniques, and existing MATLAB or toolbox functions.

4.5.1. Array pre-allocation Illustration.

To illustrate the concept of array pre-allocation, let us take an example: suppose that, using a for loop, we want to generate a 1x10^8 array, R, of random numbers. Without pre-allocating the array, we would implement our code like this (here the **tic** and **toc** keywords were used to time the execution time):

```
clear R; % we start by clearing the R variable from the memory.
tic; % tic keyword is used to calculate the time of execution.
for k=1:1:10^8
 R(k)=rand(1);
end
toc % toc keyword is used to calculate the time of execution.
```

```
Elapsed time is 17.495172 seconds.
```

```
R
```

```
R = 1×100000000
    0.7940    0.0866    0.4583    0.5840    0.8606    0.4653    0.2677 ⋯
```

By doing different tries, it was seen that it took about 17 seconds for the loop to end its execution.

Let us see the time it will take if we pre-allocate the array, R using the function ones(), so that its elements are initialized as 1s before entering the loop:

```
tic; % tic keyword is used to calculate the time of execution.
R=ones(1,10^8);
for k=1:10^8
 R(k)=rand(1);
end
toc % toc keyword is used to calculate the time of execution.
```

```
Elapsed time is 4.490343 seconds.
```

```
R
```

```
R = 1×100000000
    0.6575    0.4060    0.9396    0.1912    0.2177    0.0114    0.9936 ⋯
```

By doing different tries, It was seen that it took about 3.5 to 4.5 seconds for the loop to end its execution, which is less than the time it took when we did not pre-allocate the R array.

Therefore, in our project, array pre-allocation was a practice followed so as to ensure time efficiency.

4.5.2. Vectorizing loops illustration

Vectorization in MATLAB refers to techniques for eliminating loops, using a combination of matrix/vector operators and indexing techniques. To illustrate this idea, we assume that we have an array, R, whose elements are random decimal numbers, ranging from 0 to 1. We want to threshold the elements of the array, such that the elements of R greater than or equal to 0.5 will be set to 1, and those less than 0.5 will be set to 0. By using a for loop, without vectorizing, our code will be written as given below (here the **tic** and **toc** keywords were used to time the execution time):

```
R=rand(1,10^7);
tic; % tic keyword is used to calculate the time of execution.
for k=1:10^7
    if (R(k) < 0.5)
        R(k) = 0;
    else
        R(k) = 1;
    end
end
toc % toc keyword is used to calculate the time of execution.
```

```
Elapsed time is 0.119271 seconds.
```

By various experimentations, the loop took about 0.1 to 0.15 seconds to finish its execution.

Let us then see the time it will take if we vectorize the loop. The syntax of the loop will even look simpler, since it will take only operators to indicate the loop.

```
R=rand(1,10^7);
display("R array's variables before the loop execution")
```

```
    "R array's variables before the loop execution"
```

```
R
```

```
R = 1×10000000
     0.8910     0.6911     0.7243     0.1794     0.7174     0.4693     0.4250  ⋯
```

```
tic; % tic keyword is used to calculate the time of execution.
R=(R >= 0.5);
toc % toc keyword is used to calculate the time of execution.
```

```
Elapsed time is 0.015905 seconds.
```

```
display("R array's variables after the loop execution")
```

```
    "R array's variables after the loop execution"
```

```
double(R)
```

```
ans = 1×10000000
      1     1     1     0     1     0     0     1     1     0     0     1
1 ⋯
```

Through various experimentation, the time of execution diminished to about 0.01 to 0.03 seconds. Therefore, in our project, loops vectorization was a practice followed so as to ensure time efficiency.

4.6. The Graphical User Interface.

As it was mentioned, this project aimed at developing a computer software for end-users. This means that the image processing algorithms and functions explained above will be executed as the user manipulates an interface that has graphics (buttons, plots of images, sliders, …). MATLAB has an application called "MATLAB App Designer" that allows the development of graphical user interfaces [10]. MATLAB also allows to compile such graphical user interface software into a standalone software for Windows and Mac Operating systems, through MATLAB Compiler [11]. As we were investigating different ways of implementing the image processing algorithms to be used in this project, we also indicated some variables that the user will be able to manipulate through the graphical user interface, depending on the output he/she wishes. As it can be seen on the screenshot below, those variables include the camera, the image trimmer sensitivity and the movement detector sensitivity.

4.6.1. The main components of the graphical user interface.

The figure below shows the first window a user sees when he/she opens our application. The final software can be accessed (with a graphical user interface) through downloading our folder of scripts from [12], add all those files to a local folder, open that folder in MATLAB (version R2021a or later), and run the matApp.m file.

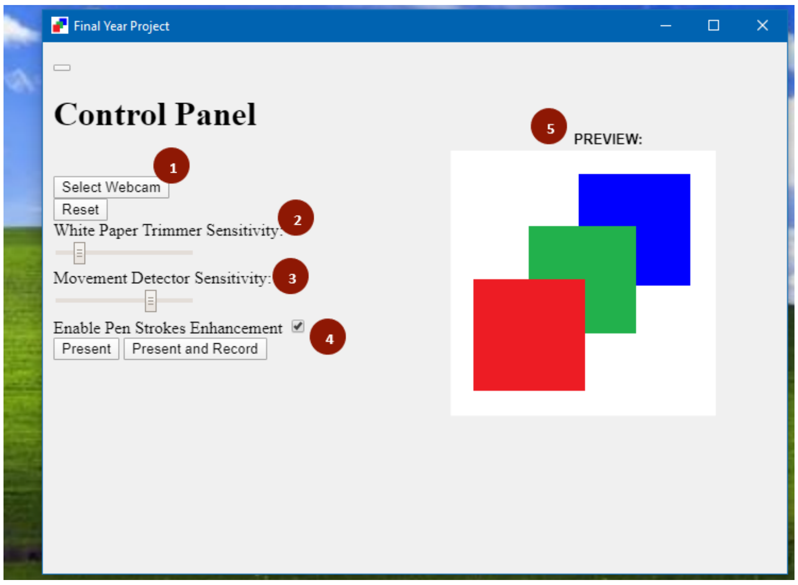

Final Year Project
Control Panel
Select Webcam
Reset
White Paper Trimmer Sensitivity:
Movement Detector Sensitivity:
Enable Pen Strokes Enhancement
Present
Present and Record
PREVIEW:
1
2
3
4
5

Fig. 32. The main components of the graphical user interface.

1. **The "Select Webcam" button:** allows the user to select an installed camera and available resolution and then preview a sample output of the camera on the **"PREVIEW" pane (labelled as 5 on the figure)**. Fig.33. illustrates the required steps of selecting a webcam.

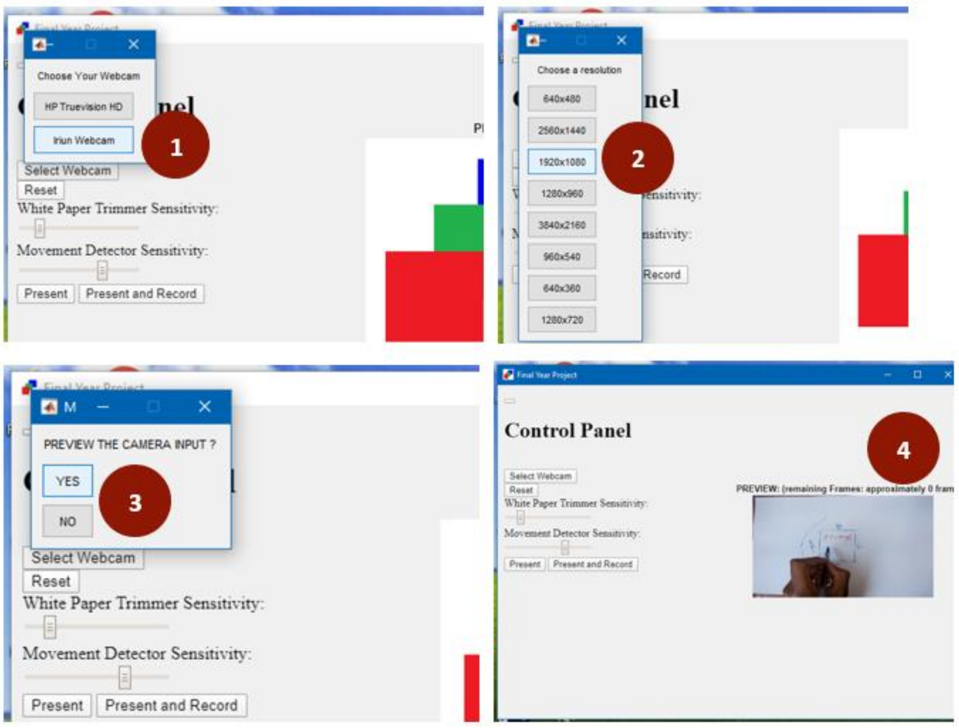


Fig. 33. Process of selecting a webcam. (1: select a webcam by name, 2: select a resolution, 3: choose whether or not you want to preview a video from your selected webcam, 4: Watch a preview for a limited number of frames)

2. **The “White Paper Trimmer Sensitivity” slider**: indicates how much the user wants to trim the output video with respect to the white paper position. As the user manipulates it, it returns a value between 0.1 and 0.9 which indicates the threshold percentage of white pixels the software has to detect on each of the four edges while trimming. By default, the value returned is programmed to be 0.3.

3. **The “Movement Detector Sensitivity” slider:** indicates how sensitive the system should be towards resetting in case the paper is moved with respect to the camera position. It returns values between 0.1 and 0.9 which indicates the percentage of pixels that have not changed from one video frame to another, by referring to the pixel samples trimmed from the upper corners of those frames. By default, the value returned is programmed to be 0.7.

4. **The “Enable Pen Strokes Enhancement” Checkbox and the “Present” and “Present and Record” buttons:** The checkbox enables the user to set if he/she needs that the system should include image enhancement processes (that might delay the frame rate), or not. For the two remaining buttons each enables the user to enter into a full screen presentation of his/her writings. In addition to that, the second button allows to record the voice and the video frames and later combines them into a video file.

As it is stated above, the final software can record the video along with the audio. MATLAB has the VideoWriter object that allows to process digital images as video frames, select the frame rate and combine them into a video file. MATLAB also has the audiorecorder object that can be used to record audio and write it into a computer file. MATLAB also supports any kind of webcam (camera) provided that it can be recognized by the computer’s operating system. The webcamlist object shows a list of webcams connected to the computer, the webcam() and snapshot() functions respectively, allow you to select the webcam and to take photos instantly. On the graphical user interface we built, the user can choose to present the output and also record the stream at the same time. The two buttons: "Present" and "Present and Record" allows to Only Present or Present and Record, respectively.

5. EXPERIMENTS' RESULTS AND RELATED DISCUSSIONS.

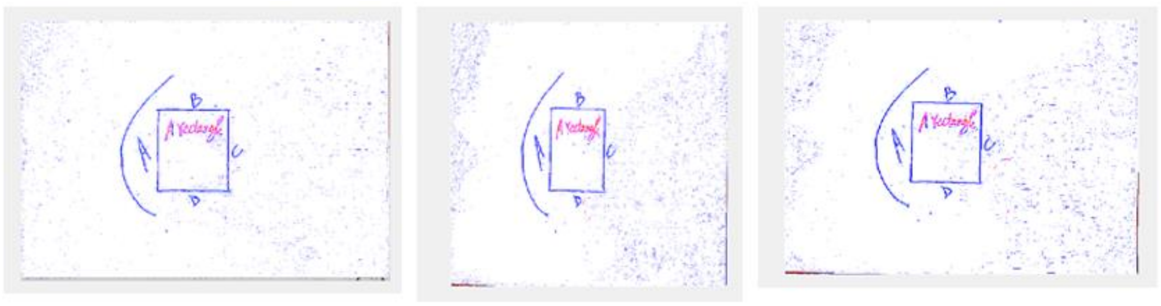
In this section, we are going to compare the results of the research and the outcome we expected. The hypothesis of our research stated, **"It is possible to implement a comprehensible system that captures in real-time a video frames sequence of pen strokes inscription on a white paper and simultaneously, processes them into enhanced images, compensating for any pixel of the white paper obstructed from the camera by a significant opaque object, thus neutralizing the obstruction"**.

The expected result of this project was a graphical user interface software that would capture images (frames) of a white paper as the writer is writing and process the images to show only the enhanced white paper and what is written on the paper (the pen strokes). The system had to use segmentation processes to remove any part which was obstructing any part of the white paper, including the writer's hand. The system was also expected to allow the user to record a video along with the audio, and the video will only be showing enhanced pen strokes being inscribed on a purely white paper.

While the result showed that the results matched the most parts of the hypothesis, there are some few elements that did not comply. In this section we are going to review critical aspects that characterize the limitations of the performance of our final software.

5.1. The output is not always as appealing as expected.

The imperfect elegance of the output video frames is one of the aspects a user can perceive while manipulating our software in differently illuminated locations. This is because our system relies on a long-generalized sequence of image processing algorithms that include filtering out noises, detecting edges of pen strokes, sequences of morphological processes, etc. All those were designed in a generalized way for the purpose of protecting the thin pen strokes, while also removing the thinner noises, and the huge obstruction (the hand) in different situations. Due to random fluctuations of illumination and noises that change time by time and location by location, some details of pen strokes can disappear, and also unexpected noises may appear. Also, by referring to the fact the system favors resolutions less than two megapixels (for the sake of preserving the frame rate, for Intel(R) Core(TM) i7-5500U CPU @ 2.40 GHz processor computer), the output video may not look attractive for large screens (large as digital whiteboards, for example) viewers, as we expected to extend up to 8 megapixels.



Output of a 1920x1080 resolution input. ***Output of a 1280x960 resolution input.*** ***Output of a 960x540 resolution input.***

Fig. 34. The output can be impacted by random fluctuation of non-uniform illumination. The screenshots were taken during an experiment done in a low illuminated place.

5.2. The system was not tested on various people with different skin colors.

Due to time limits, the final software was finished late and thus it was tested only on the group members who conducted this research. The only emulation that could be done was to turn the sides of the hand in front of the camera and see if the obstruction would still be removed regardless of the skin color changes. As a result, the system was able to remove the obstruction for inputs with resolutions higher or equal to 1 megapixel. For resolutions lower than 1 megapixel, the system showed some part of the hand's edges as pen strokes. Fig. 35. shows a screenshot taken to illustrate the artefacts, for an input of a low resolution (640x360).

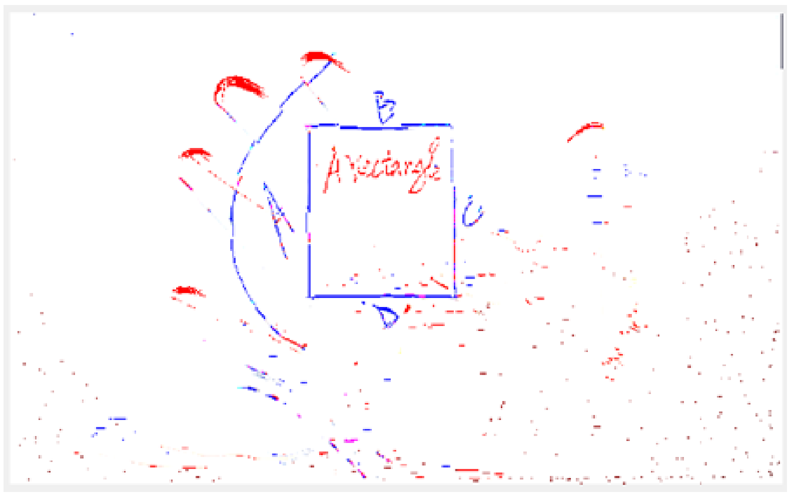


Fig. 35. Artefacts due to failure of obstruction removal process, for resolutions lower than 1 megapixel. (Here we used a resolution of 640x360.)

5.3. The user should always be aware of the camera's perspective with respect to the paper.

The user should know that if a region of the white paper is never exposed to the camera, the software will not be able to figure out the pen inscriptions from that region. It is recommended that the user should place the camera in a way that the line of sight from the camera to the whitepaper will not be obstructed most of the time. The user should also note that the system considers the first frame as an unobstructed reference for removing the obstruction in the next frames. Therefore, the user should not start presenting while the whitepaper is not fully exposed to the camera. This also applies to the case of resetting, when the paper is moved. The software was programmed to dim the intensity and beep twice when the system resets due to paper movement or illumination changes. The user should remove his/her hand from the paper for a second so as to ensure that the image processing loop resets correctly. Fig. 36. shows the resulting output when the first frame is obstructed by the hand.

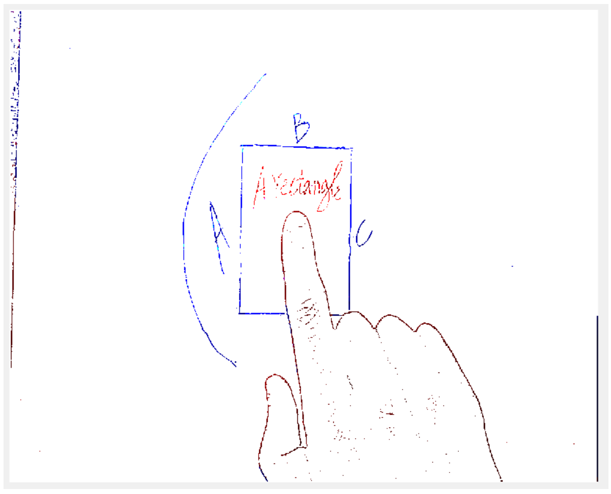


Fig. 36. The resulting output when the first frame is obstructed by the hand.

5.4. Problems related to output video file compression.

MATLAB provides functions that allow combining images into a video file, and other functions that allow a programmer to combine recorded audio samples into an audio file. When it came to combining the video frames together with audio samples into a single file, a compression challenge arose: the output file was extremely huge. We tried the two compressors available on the platform ('DV Video Encoder' and 'MJPEG Compressor'), both did not provide a satisfactory performance, as a video of less than 10 seconds took more than 5 Megabytes, with a resolution lower than 0.1 megapixels. Therefore, we searched for another alternative solution to this problem out of MATLAB’s toolboxes. The other alternative we developed was to use a Python script that relies on a Python library called “moviepy” [13]. The library provides functions that directly provide video editing solutions, with a satisfying compression. This alternative developed forces the user to install Python as well as the moviepy library in order to be able to combine the recorded audio together with the recorded video in a single video file. This also imposed another limitation, since python can not interact with MATLAB compiled applications. Therefore, the automatic combination of the two files (audio and video) together will be available for a user using a non compiled version of our program, running in an installed MATLAB R2021a edition.

6. CONCLUSION AND RECOMMENDATIONS FOR FUTURE RESEARCH.

The general objective of this project was to design and implement a software that would capture images (frames) of a white paper as the writer is writing and process the video frames to show only the enhanced white paper and what is written on the paper (the pen strokes). The research methodology and the software implementation relied on MATLAB toolboxes (Image Processing Toolbox, MATLAB App Designer and MATLAB Compiler) and hardware components: a personal computer and a camera. The outcome of the project was a software that complied with the set objectives but with limitations related to input image resolution, noise and illumination non-uniformity, perspective of the camera and output video file compression.

As it was explained in the previous chapter of this report, the overall output was not attractive as it was expected. The overall report suggests that the resolution input for which the system will perform its best is 1 megapixel. For further research, we recommend trying different alternatives that can support high resolution images while also conserving the needed frame rate. This would require using a computer with a more advanced processor to reduce the execution time. Another approach to recommend for further research is testing the system with various users of different skin color. This would give a more assurance on how the system is able to generalize and detect obstruction.

Regarding the problem of re-adjusting the processed images sequence with the change in position / perspective of the camera to the paper, we may recommend an implementation of geometric transformations, so as to re-align video frames whenever the paper is moved, and thus remove the obstruction easily, and remove the burden of resetting. As the time did not allow us to tackle the problem of video file compression, we would also recommend future researchers to consider developing other video compressing libraries different from those reported in the previous chapter of this report.

To ensure that the research schedule had to match with the researcher's academic calendar, some of the remaining challenges were not solved. Those challenges, as are reported in the previous chapter, required refinement after testing the software in different conditions of noise and illuminations, and on users with different skin colors. For a more exhaustive research, further investigations to tackle those challenges are suggested.

REFERENCES

[1] L. He and Z. Zhang, “Real-time whiteboard capture and processing using a video camera for remote collaboration,” *IEEE Transactions on Multimedia*, vol. 9, no. 1, pp. 198–206, Jan. 2007.

[2] E. Coleshill, D. D. Stacey, and A. Ferworn, “Obstruction removal using feature extraction through time for videoconferencing processing,” *Innovative Algorithms and Techniques in Automation, Industrial Electronics and Telecommunications*, pp. 131–133, 2007.

[3] A. Mahmood and S. Khan, “Correlation-coefficient-based fast template matching through partial elimination,” *IEEE Transactions on Image Processing*, vol. 21, no. 4, pp. 2099–2108, Aug. 2010.

[4] S. Zhi, Y. Cui, J. Deng, and W. Du, “An FPGA-based simple RGB-HSI space conversion algorithm for hardware image processing,” *IEEE Access*, vol. 8, pp. 173838–173853, Oct. 2020.

[5] R. A. Hardcastle and C. J. Matthews, “Speed of writing,” *Journal of the Forensic Science Society*, vol. 31, no. 1, pp. 21–29, Jan. 1991.

[6] P. Kandhway and A. K. Bhandari, “An optimal adaptive thresholding based sub-histogram equalization for brightness preserving image contrast enhancement,” *Multidimensional Systems and Signal Processing*, vol. 30, no. 4, pp. 1859–1894, Feb. 2019.

[7] S. N and V. S, “Image segmentation by using thresholding techniques for medical images,” *Computer Science & Engineering: An International Journal*, vol. 6, no. 1, pp. 1–13, Feb. 2016.

[8] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed. Upper Saddle River, New Jersey: Pearson Prentice Hall, 2008.

[9] R. C. Gonzalez, S. L. Eddins, and R. E. Woods, “Fundamentals,” in Digital image processing using MATLAB, 2nd ed., S.L., California: Gatesmark Publishing, 2009, pp. 65–71.

[10] S. Attaway, in *MATLAB: A practical introduction to programming and problem solving*, Oxford, United Kingdom: Elsevier, B/H, Butterworth-Heinemann, 2019, pp. 475–488.

[11] R. Lee and R. Bachnak, “The Matlab Compiler Suite: M files to c/c++ executable programs,” *2002 American Society for Engineering Education Annual Conference & Exposition*, p. 7.1164.2, 2002.

[12] P. S. Mbonimpa, J. Habiyaremye Ntwali, and C. Maliza, “RealTimePenStrokesScanner,” *GitHub*, 21-Jan-2022. [Online]. Available: https://github.com/pacomesimon/RealTimePenStrokesScanner/tree/main/Real_Time_testing. [Accessed: 22-Jan-2022].

[13] M. Beacom, T. Burrows, and Zulko, “Moviepy,” *PyPI*, 05-Oct-2020. [Online]. Available: https://pypi.org/project/moviepy/. [Accessed: 21-Jan-2022].

A. Appendix A: MATLAB Code used in Illustrating the Main Algorithm’s Processes.

A.1. Introduction.

In this section, we are going to provide explanations along with MATLAB codes of the proposed algorithms to deal with the basic parts of our implementation. Those parts include processes of Paper Movement Detection, Image Enhancement and Obstruction Removal. Note that the codes shown here corresponds to the 4th chapter of this report, specifically the 4.2, 4.3 and 4.4 sections. The codes are presented here for illustration purposes only.

A.2. Algorithm Design for the Paper Movement Detection Process.

To ensure that the obstruction can be removed, the system will need to first make sure that the paper has not been moved with respect to the camera's perspective.

A.2.1. Methodology Illustration.

In this section, we are going to illustrate the algorithm of paper movement detection.

Step 1: Current and Previous Images’ Copies Acquisition.

In this demonstration, we will assume two images, where one represents a paper positioned well, and other represents the paper moved a bit from its initial position.

Here at Step 1, This process will acquire a copy of the current image (frame) and the previous image (frame). The input images are supposed to be trimmed, thus showing the white paper section at maximum.

```
rawImage1= imread("outImageCr1.png");
rawImage2=imread("outImageCr2.png");
```

Step 2: Sample pixels extraction from the upper corners of the two acquired images.

This process will take samples from the upper left and upper right corners of the two images.

```
cropCornerArea=uint16(10); % area cropped from the corner, in percentage;
rowCr1=1;
```

```
colCr1=1;
[rowCr2 colCr2 d]=size(rawImage2);
total_numberOfRows=rowCr2 - rowCr1 + 1;
total_numberOfColumns = colCr2 - colCr1 + 1;
farthest_row = total_numberOfRows * cropCornerArea / 100 ;
farthest_column = total_numberOfColumns * cropCornerArea / 100 ;
cropCorner1{1}=rawImage1(1:farthest_row,1:farthest_column,:);
cropCorner1{2}=rawImage1(1:farthest_row,end-farthest_column:end,:);
cropCorner2{1}=rawImage2(1:farthest_row,1:farthest_column,:);
cropCorner2{2}=rawImage2(1:farthest_row,end-farthest_column:end,:);
```

Step 3: Correlation coefficients calculations between the equivalent corners of the two acquired images.

To enhance the contrast, this process will first equalize the taken samples. Next, it will transform them into single column vectors, then compute two correlation coefficients: correlation coefficient between upper right corner of current image and the upper right corner of previous image, then correlation coefficient between upper left corner of current image and the upper left corner of previous image.

```
Corner1{1}=histeq(cropCorner1{1});
Corner1{2}=histeq(cropCorner1{2});
Corner2{1}=histeq(cropCorner2{1});
Corner2{2}=histeq(cropCorner2{2});

correlation_from_corner1=corr(double(Corner1{1}(:)),double(Corner2{1}(:)))
correlation_from_corner2=corr(double(Corner1{2}(:)),double(Corner2{2}(:)))
```

Step 4: Returning a result of a test for a high correlation coefficient.

For this process, the program has to decide whether or not the two correlation coefficients satisfy a minimum correlation coefficient. Normally, a correlation coefficient will range from 0 to 1, such that a value near to 1 indicates a high correlation. By experimentation, we found that in different conditions, 0.65 is the minimum coefficient of correlation that resulted when the paper was not moved. Therefore, this process will give a true answer if each of the two correlation coefficients satisfies the threshold condition of being greater than 0.65.

```
Decision_ = ((correlation_from_corner1 > 0.65) && (correlation_from_corner2 > 0.65))
```

A.2.2. Algorithm Improvement.

Given that the calculation of coefficient of correlation takes too much time due to its high computational complexity, we developed a new approach to tackle the challenge of time efficiency, as a real-time constraint. The new approach proposed had to start by thresholding the equalized pixel samples from the papers' upper corners. The process had then to compare the resulting binary patterns using the binary XOR operation. The resulting data would indicate which pixels have changed, and those that didn't change, if the paper had been moved. By experimentation, we found that the maximum percentage of pixels that changed when the paper was not moved was 30%. This was due to the presence of noise that varied in different cases.

```
Corner1{1}= rgb2gray(Corner1{1});
Corner1{2}= rgb2gray(Corner1{2});
Corner2{1}= rgb2gray(Corner2{1});
Corner2{2}= rgb2gray(Corner2{2});

Corner1{1}=imbinarize(Corner1{1},0.5);
Corner1{2}=imbinarize(Corner1{2},0.5);
Corner2{1}=imbinarize(Corner2{1},0.5);
Corner2{2}=imbinarize(Corner2{2},0.5);

xor_from_corner1=xor((Corner1{1}(:)),(Corner2{1}(:)));
xor_from_corner2=xor((Corner1{2}(:)),(Corner2{2}(:)));

sum_xor_from_corner1=sum(xor_from_corner1(:));
sum_xor_from_corner2=sum(xor_from_corner2(:));

ratio_xor_from_corner1 = sum_xor_from_corner1/length(xor_from_corner1)
ratio_xor_from_corner2 = sum_xor_from_corner2/length(xor_from_corner2)
```

A.2.3. Comparison of Time Usage Between the Correlation-Coefficient-Based Approach and the XOR-Based Approach.

In this section, we are going to compare the time usage of the two approaches: coefficient correlation approach against the XOR approach. We will generate different random images of 1 megapixel to 5 megapixels and graph the time it will take for the computer to execute each dataset using each of the two approaches. For each approach, the computer will execute the step number 3, assuming that the given image represents the corner samples of the current and the previous images (frames).

```
i1_MegaPixels=uint8(255*rand(1000,1000,3));
i2_MegaPixels=uint8(255*rand(2000,1000,3));
i3_MegaPixels=uint8(255*rand(3000,1000,3));
i4_MegaPixels=uint8(255*rand(4000,1000,3));
i5_MegaPixels=uint8(255*rand(5000,1000,3));
tic;
corr_Approach(i1_MegaPixels);
t_correlation(1)=toc;
tic;
corr_Approach(i2_MegaPixels);
t_correlation(2)=toc;
tic;
corr_Approach(i3_MegaPixels);
t_correlation(3)=toc;
tic;
corr_Approach(i4_MegaPixels);
t_correlation(4)=toc;
tic;
corr_Approach(i5_MegaPixels);
t_correlation(5)=toc;

tic;
xor_Approach(i1_MegaPixels);
t_xor(1)=toc;
tic;
xor_Approach(i2_MegaPixels);
t_xor(2)=toc;
tic;
xor_Approach(i3_MegaPixels);
t_xor(3)=toc;
tic;
xor_Approach(i4_MegaPixels);
t_xor(4)=toc;
tic;
xor_Approach(i5_MegaPixels);
t_xor(5)=toc;
subplot(1,1,1)
plot(1:1:5,t_correlation,'r',1:1:5,t_xor,'b')
title("Time Usage of Different Approaches, Against Different Image Resolutions")
xlabel("Resolution (Megapixels)")
ylabel("Time Used (Seconds)")
legend({'Correlation coefficient approach','XOR approach'})
```

A.3. Algorithm Design for the Image enhancement Process.

To ensure that the pen strokes are enhanced, the system will need to remove noises from the image and then enhance specifically the traces of the pen's ink.

A.3.1. Proposed output of the "Image Enhancement" process:

The output Image should match these characteristics:

- The image should be truncated to show at maximum the enhanced white paper,
- The pen strokes on the image should be enhanced, with an increased saturation,
- The first frame will undergo its unique enhancement process to be transformed into a white frame and be a good reference to the next frames.

A.3.2. Methodology Illustration.

In this section, we are going to illustrate the algorithm of image enhancement.

Step 1: Acquisition of the Input Image's Copy.

At this step, the program will acquire a copy of the input image, supposing that it is trimmed to get at maximum the part that shows the white paper.

```
inputIm=imread("theImageWithHand.png");
rowCr1 = 1;
colCr1 = 1;
[rowCr2 colCr2 d] = size(inputIm);

total_numberOfRows=rowCr2 - rowCr1 + 1;
total_numberOfColumns = colCr2 - colCr1 + 1;
```

Step 2: Image smoothing using a low pass filter.

This process will use an averaging filter (a low pass filter) to filter out noises in the image after converting it to grayscale. Given that the size of the filter matters, this process uses a filter whose height and width are 0.3% of the height and width of the input image (which means that the area of the filter is 0.09% of the area of the input image). That factor of 0.3% was selected after numerous try-and-see image enhancement experimentations.

```
theRowPeriod = ceil((0.3/100) * total_numberOfRows);
theColPeriod = ceil((0.3/100) * total_numberOfColumns);
global LP_filter_PS
LP_filter_PS = (1/(theColPeriod*theRowPeriod)) * ones(theRowPeriod,theColPeriod);
inputIm_Smoothed=rgb2gray(inputIm);
inputIm_Smoothed = imfilter(inputIm_Smoothed,LP_filter_PS);
```

Step 3: Edge Detection.

This step will use a 3x3 Laplacian filter to detect edges in the image, then threshold the output so as to highlight areas where the ink has traced. Since the pen strokes can have any direction, we need an edge detector that is isotropic, the Laplacian filter.

```
Laplacian_Filter=[1 1 1; 1 -8 1; 1 1 1]
F=[1 1 1; 1 -8 1; 1 1 1] ;
inputIm_Edges=imfilter(inputIm_Smoothed_Gray,F);
inputIm_Edges = histeq(inputIm_Edges);
inputIm_MaskForPenStrokes = (inputIm_Edges < 250);
```

Step 4: Formation of a binary mask to highlight the pen strokes.

This step will also do a dilation so as to remove remaining noises. Given that the used structuring element's size matters, this process uses a structuring element whose height and width are 0.2% of the height and width of the input image.

```
theRowPeriod = ceil((0.2/100) * total_numberOfRows);
theColPeriod = ceil((0.2/100) * total_numberOfColumns);
global Dilution_se
Dilution_se = ones(theRowPeriod,theColPeriod);
pattern = Dilution_se;
inputIm_MaskForPenStrokes=imdilate(inputIm_MaskForPenStrokes,pattern);
```

Step 5: Pen Strokes Enhancement.

This process will do an element-wise multiplication between the input image and the created mask (complemented), to highlight the pen strokes. To increase the saturation of the pen strokes, the resulting image was converted to the HSI model, and the saturation dimension was multiplied by a relatively large number (10 for our case). After the enhancement, the image was re-converted to RGB for the next steps.

```

inputIm_MaskForPenStrokes = uint8(inputIm_MaskForPenStrokes);
my_white_blank = 255*ones(size(inputIm));
inputIm_WhiteArea= inputIm_MaskForPenStrokes.*uint8(my_white_blank) ;
inputIm = uint8(~inputIm_MaskForPenStrokes) .* inputIm;
inputIm_Penstrokes_Unsaturated = inputIm;
inputIMAGE_HSV=rgb2hsv(inputIm);
inputIMAGE_HSV(:,:,2)=inputIMAGE_HSV(:,:,2)*10;
inputIm=hsv2rgb(inputIMAGE_HSV);
```

Step 6: Returning the enhanced image and the binary mask.

This process will add the remaining part (the part which does not represent pen strokes) to the part that represents pen strokes. After that, it will return the enhanced image and a binary mask to be later used in obstruction removal process, to protect pen strokes from being considered as obstruction (see the next topic: Obstruction Removal)

```
outputIMAGE= uint8(inputIm_WhiteArea) + uint8(inputIm*255);
global mypenstroke_mask
mypenstroke_mask = ~inputIm_MaskForPenStrokes;
```

A.3.3. Algorithm Improvement.

By experimentation, It was found that step 5 (Pen strokes Enhancement) took too much time for execution, due to the conversion from RGB to HSI and vice-versa. Given that our case needed to increase only the saturation parameter, we developed our own approach so as to tackle the challenge of time efficiency, as a real-time constraint. The new approach proposed was to divide by three the sum of the red, green and blue components' intensities, so as to get the average intensity for each pixel. Next, the original intensities of the red, green and blue components were compared to the resulting average so as to determine which intensities are major in each pixel. The intensities that were found to be greater than the average was set to the maximum intensity (which is 255), while the intensities that were found to be less than the average were set to the lowest intensity (which is 0).

```
inputIm_average = ...
    inputIm_Penstrokes_Unsaturated(:,:,1)/3+ ...
    inputIm_Penstrokes_Unsaturated(:,:,2)/3+ ...
    inputIm_Penstrokes_Unsaturated(:,:,3)/3;
penstrokes_Saturated = inputIm_Penstrokes_Unsaturated;
penstrokes_Saturated(:,:,1) = 255*uint8(inputIm_Penstrokes_Unsaturated(:,:,1) > ...
    inputIm_average);
penstrokes_Saturated(:,:,2) = 255*uint8(inputIm_Penstrokes_Unsaturated(:,:,2) > ...
    inputIm_average);
penstrokes_Saturated(:,:,3) = 255*uint8(inputIm_Penstrokes_Unsaturated(:,:,3) > ...
    inputIm_average);
outputIm= (inputIm_WhiteArea) + penstrokes_Saturated;
```

A.3.4. Comparison between the HSI-based approach and the improved approach.

In this section, we are going to compare the time usage of the two approaches: the HSI approach against the improved approach. We will generate different random images of 1 megapixel to 5 megapixels and graph the time it will take for the computer to execute each dataset using each of the two approaches. For each approach, the computer will execute the step number 3, assuming that the given image represents the RGB image to be processed.

```
i1_MegaPixels=uint8(255*rand(100000,10,3));
i2_MegaPixels=uint8(255*rand(200000,10,3));
i3_MegaPixels=uint8(255*rand(300000,10,3));
i4_MegaPixels=uint8(255*rand(400000,10,3));
i5_MegaPixels=uint8(255*rand(500000,10,3));
tic;
hsi_Approach(i1_MegaPixels);
t_hsi(1)=toc;
tic;
hsi_Approach(i2_MegaPixels);
t_hsi(2)=toc;
tic;
hsi_Approach(i3_MegaPixels);
t_hsi(3)=toc;
tic;
hsi_Approach(i4_MegaPixels);
t_hsi(4)=toc;
tic;
hsi_Approach(i5_MegaPixels);
t_hsi(5)=toc;

tic;
new_Approach(i1_MegaPixels);
t_new(1)=toc;
tic;
new_Approach(i2_MegaPixels);
t_new(2)=toc;
tic;
new_Approach(i3_MegaPixels);
t_new(3)=toc;
tic;
new_Approach(i4_MegaPixels);
t_new(4)=toc;
tic;
new_Approach(i5_MegaPixels);
t_new(5)=toc;

plot(1:1:5,t_hsi,'r',1:1:5,t_new,'b')
title("Time Usage of Saturation Enhancement Approaches, Considering Image Resolutions.")
xlabel("Resolution (Megapixels)")
ylabel("Time Used (Seconds)")
legend({'the HSI approach','the proposed (new) approach'})
```

A.4. Algorithm Design for the Obstruction Removal Process.

After the Image Enhancement process, the system will need to compensate for the obstructed part of the input image.

A.4.1. Methodology Illustration.

In this section, we are going to illustrate the algorithm of obstruction removal.

Step 1: Input Image's Copy Acquisition

At this step, the program will read the input image and trim it so as to get at maximum the part that shows the white paper.

```
 inputIm=imread("theImageWithHand.png");
 rowCr1 = 1;
 colCr1 = 1;
 [rowCr2 colCr2 d] = size(inputIm);
```

Step 2: Image smoothing using a low pass filter.

This process will use a low pass filter to filter out any object that appears in the image, except only the obstruction (the hand). Given that the size of the filter matters, this process uses a filter whose height and width are 1.5% of the height and width of the input image (which means that the area of the filter is 2.25% of the area of the input image). That factor of 1.5% was selected after numerous try-and-see image enhancement experimentations.

The process will use the binary mask from the image enhancement process to highlight pen strokes, and make sure that they do not contribute to the obstruction pixels by setting them to the highest intensity value, 255 (This will also prevent the effect of unequal illumination from obstructing the pen strokes). Then the resulting image will be filtered by smoothing.

```
 total_numberOfRows=rowCr2 - rowCr1 + 1;
 total_numberOfColumns = colCr2 - colCr1 + 1;
 theRowPeriod = ceil((1.5/100) * total_numberOfRows);
 theColPeriod = ceil((1.5/100) * total_numberOfColumns);
 LP_filter_Hand = (1/(theColPeriod*theRowPeriod)) *
ones(theRowPeriod,theColPeriod);
```

```
global mypenstroke_mask
inputIm_Smoothed=rgb2gray(inputIm);
inputIm_Smoothed(mypenstroke_mask)= uint8(255);
inputIm_Smoothed = imfilter(inputIm_Smoothed,LP_filter_Hand);
```

Step 3: Thresholding

This process will threshold the smoothed output image from step 2. The image equalization will enhance the contrast significantly, to prepare the image for the thresholding. By thresholding the image, the process' results will make sure that the obstruction will be represented by 0 intensity pixels.

```
inputIm_Smoothed_eq=histeq(inputIm_Smoothed);
inputIm_Smoothed_eq_gray = (inputIm_Smoothed_eq);
inputIm_MaskOfObstruction_Binary = imbinarize(inputIm_Smoothed_eq_gray,0.5);
```

Step 4: Obstruction Removal.

This process will replicate the created obstruction mask into three dimensions to multiply (element-wise multiplication) with the input image, thus remove the obstruction by setting the obstruction to zero intensity. The zero intensities will be compensated by the previous frame through addition.

```
inputIm_MaskOfObstruction_Binary=uint8(inputIm_MaskOfObstruction_Binary);
clear Mask_RGB;
Mask_RGB(:,:,1)=inputIm_MaskOfObstruction_Binary;
Mask_RGB(:,:,2)=inputIm_MaskOfObstruction_Binary;
Mask_RGB(:,:,3)=inputIm_MaskOfObstruction_Binary;

outputIm=inputIm.*(Mask_RGB);
```

Step 5: Return the output image.

This step will return the output image to the main program.