

Department of Informatics

School Of Computer Science

UNIVERSITY OF PETROLEUM & ENERGY STUDIES,

DEHRADUN- 248007. Uttarakhand

SOFTWARE REQUIREMENTS SPECIFICATION

For

**VIRTUAL PAINTER USING OPENCV IN C++**

**22nd October 2021**

Prepared by

|  |  |  |
| --- | --- | --- |
| **SPECIALIZATION** | **SAP ID** | **NAME** |
| B.Tech CSE (CCVT) | 500076480 | Akansh Mowar |
| B.Tech CSE (CCVT) | 500075149 | Aviral Jindal |
| B.Tech CSE (CCVT) | 500075804 | Kriti Niranjan |
| B.Tech CSE (CCVT) | 500075894 | Bhavya Jain |

Table of Contents

|  |  |  |
| --- | --- | --- |
| **Topic** | | **Page No** |
| 1 | Introduction |  |
|  | 1.1 Purpose of the Project | 4 |
|  | 1.2 Target Beneficiary | 4 |
|  | 1.3 Project Scope | 4 |
|  | 1.4 References | 4-5 |
| 2 | Project Description | 5-6 |
|  | 2.1 Reference Algorithm | 6-8 |
|  | 2.2 Data/ Data structure |  |
|  | 2.3 SWOT Analysis |  |
|  | 2.4 Project Features |  |
|  | 2.5 User Classes and Characteristics |  |
|  | 2.6 Design and Implementation Constraints |  |
|  | 2.7 Design diagrams | 8 |
|  | 2.8 Assumption and Dependencies |  |
| 3 | System Requirements |  |
|  | 3.1 User Interface |  |
|  | 3.2 Software Interface |  |
|  | 3.3 Database Interface |  |
|  | 3.4 Protocols |  |
| 4 | Non-functional Requirements |  |
|  | 4.1 Performance requirements |  |
|  | 4.2 Security requirements |  |
|  | 4.3 Software Quality Attributes |  |
| 5 | Other Requirements |  |
| Appendix A: Glossary | |  |
| Appendix B: Analysis Model | |  |
| Appendix C: Issues List | |  |

**INTRODUCTION**

**Purpose of the Project:**

Writing is a cohesive form of communication that can effectively convey our thoughts. Typing and writing are the standard methods of recording information today. Characters or words are written in the free space with a marker or finger. It differs from traditional writing methods in that the pen does not move up and down. With the development of intelligent wearable devices, the digital world can now be controlled by human gestures. These wearable devices can recognize and understand our actions. Recognizing and interpreting a continuous sequential gesture stream from the given set of input data is gesture recognition. Gestures are non-verbal information used to improve computer language understanding. Human gestures are perceived by vision, and computer vision is used to analyze various gestures. The project takes advantage of this gap and focuses on developing a motion-to-text converter that can potentially serve as a software for intelligent wearable devices for writing from the air. The system will use computer vision to trace the finger's path, and in that way, one can write from above. The generated text can also be used for various purposes, such as sending messages, e-mails, etc. It will be a powerful means of communication for the deaf. It is an effective communication method that reduces cellphone and laptop usage by eliminating the need to write.

**Target Beneficiary:**

Drawing or Sketching using hand is everyone’s wish. Some or the other time we imagine writing in air using our hand. So, here came the project from this concept where we create a canvas and pick the colours required using our hand and draw the required design or write anything you wish.

**Project Scope:**

• To ensure that, the interface is very simple and easily understandable by the user.

• The user should be able to draw what he wishes to draw without any interruptions.

• In future, this is useful for making kids to learn drawing in schools in an interactive way.

**Références :**

* [**https://www.youtube.com/watch?v=2FYm3GOonhk&t=9894s**](https://www.youtube.com/watch?v=2FYm3GOonhk&t=9894s)
* [**https://ijsrst.com/paper/7305.pdf**](https://ijsrst.com/paper/7305.pdf)
* [**https://techvidvan.com/tutorials/detect-objects-of-similar-color-using-opencv-in-python/**](https://techvidvan.com/tutorials/detect-objects-of-similar-color-using-opencv-in-python/)
* [**https://learnopencv.com/contour-detection-using-opencv-python-c/**](https://learnopencv.com/contour-detection-using-opencv-python-c/)

**Project Description**

1. **Importing Libraries :**

<opencv2/highgui.hpp>

<opencv2/imgproc.hpp>

<iostream>

<opencv2/imgcodecs.hpp>

**2. Color Detection in OpenCV:**

Import necessary packages and read the image.

Detect the color from the input image and create a mask.

Removing unnecessary noise from masks.

Apply the mask to the image.

Draw a Boundary of the detected objects.

* Step 1 – Import necessary packages and Initialize the camera
* Step 2 – Detect the color from the input image and create a mask :

Why HSV?

HSV color space is useful when we’re working with color information. It stands for HUE, SATURATION, and VALUE (or brightness). It is a cylindrical color space.

HUE: The hues are modeled as an angular dimension that encodes color information.

SATURATION: Saturation encodes the intensity of color.

VALUE: Value represents the amount to which that respective color is mixed with black.

* Step 3 – Removing unnecessary noise from masks
* Step 4 – Apply the mask on the image
* Step 5 – Draw a Boundary of the detected objects

**3. Contour Detection using OpenCV:**

Using contour detection, we can detect the borders of objects, and localize them easily in an image. It is often the first step for many interesting applications, such as image-foreground extraction, simple-image segmentation, detection and recognition.

Steps for Detecting and Drawing Contours in OpenCV:

1. Read the Image and convert it to Grayscale Format : Converting the image to grayscale is very important as it prepares the image for the next step. Converting the image to a single channel grayscale image is important for thresholding, which in turn is necessary for the contour detection algorithm to work properly.

2. Apply Binary Thresholding: This converts the image to black and white, highlighting the objects-of-interest to make things easy for the contour-detection algorithm. Thresholding turns the border of the object in the image completely white, with all pixels having the same intensity. The algorithm can now detect the borders of the objects from these white pixels.

3.Find the Contours: Use the findContours() function to detect the contours in the image.

4. Draw Contours on the Original RGB Image: Once contours have been identified, use the drawContours() function to overlay the contours on the original RGB image.

**4. How to draw lines, rectangles, circles etc. over images :**

OpenCV provides easy to use functions for drawing over an image. The most common drawing operations are given below:

* Line : cv::line (InputOutputArray img, Point pt1, Point pt2, const Scalar &color, int thickness=1, int lineType=LINE\_8, int shift=0)
* Circle: cv::circle (InputOutputArray img, Point center, int radius, const Scalar &color, int thickness=1, int lineType=LINE\_8, int shift=0)
* Rectangle: cv::rectangle (InputOutputArray img, Point pt1, Point pt2, const Scalar &color, int thickness=1, int lineType=LINE\_8, int shift=0)
* Text: cv::putText (InputOutputArray img, const String &text, Point org, int fontFace, double fontScale, Scalar color, int thickness=1, int lineType=LINE\_8, bool bottomLeftOrigin=false)

**5. Background Substraction:**

Background subtraction is a major preprocessing step. In many vision-based applications. For example, consider the case of a visitor counter where a static camera takes the number of visitors entering or leaving the room, or a traffic camera extracting information about the vehicles etc. In all these cases, first you need to extract the person or vehicles alone. Technically, you need to extract the moving foreground from static background.

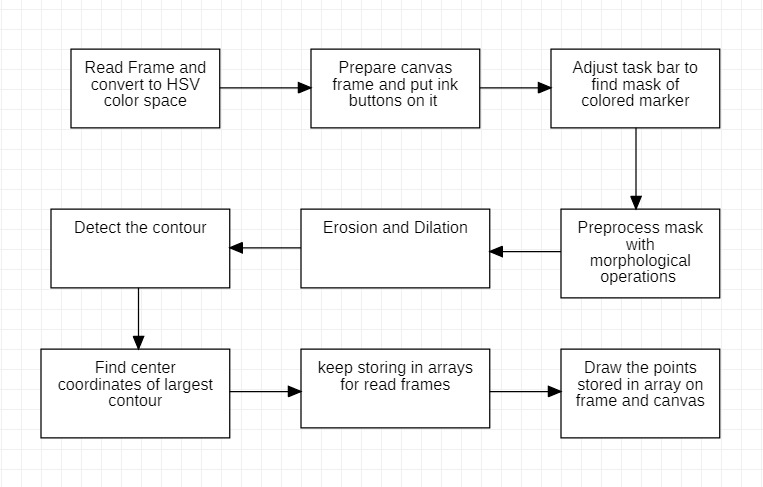
**6. Skin Detection Method:**

Process of finding skin-colored pixels and regions in an image or a video . Often used as a cue for detecting, localizing and observing targets containing skin(like faces and hands in an image) . Plays an important role in human motion analysis and face detection.

**REFERENCE ALGORITHM**

Background Subtraction :-

* [cv::BackgroundSubtractor](https://docs.opencv.org/4.x/d7/df6/classcv_1_1BackgroundSubtractor.html) object will be used to generate the foreground mask. In this example, default parameters are used, but it is also possible to declare specific parameters in the create function.
* //create Background Subtractor objects
* Ptr<BackgroundSubtractor> pBackSub;
* if (parser.get<[String](https://docs.opencv.org/4.x/dc/d84/group__core__basic.html#ga1f6634802eeadfd7245bc75cf3e216c2)>("algo") == "MOG2")
* pBackSub = [createBackgroundSubtractorMOG2](https://docs.opencv.org/4.x/de/de1/group__video__motion.html#ga2beb2dee7a073809ccec60f145b6b29c)();
* else
* pBackSub = [createBackgroundSubtractorKNN](https://docs.opencv.org/4.x/de/de1/group__video__motion.html#gac9be925771f805b6fdb614ec2292006d)();
* A [cv::VideoCapture](https://docs.opencv.org/4.x/d8/dfe/classcv_1_1VideoCapture.html) object is used to read the input video or input images sequence.
* VideoCapture capture( [samples::findFile](https://docs.opencv.org/4.x/d6/dba/group__core__utils__samples.html#ga3a33b00033b46c698ff6340d95569c13)( parser.get<[String](https://docs.opencv.org/4.x/dc/d84/group__core__basic.html#ga1f6634802eeadfd7245bc75cf3e216c2)>("input") ) );
* if (!capture.isOpened()){
* //error in opening the video input
* cerr << "Unable to open: " << parser.get<[String](https://docs.opencv.org/4.x/dc/d84/group__core__basic.html#ga1f6634802eeadfd7245bc75cf3e216c2)>("input") << endl;
* return 0;
* }
* Every frame is used both for calculating the foreground mask and for updating the background. If you want to change the learning rate used for updating the background model, it is possible to set a specific learning rate by passing a parameter to the apply method.
* //update the background model
* pBackSub->apply(frame, fgMask);
* The current frame number can be extracted from the cv::VideoCapture object and stamped in the top left corner of the current frame. A white rectangle is used to highlight the black colored frame number.
* //get the frame number and write it on the current frame
* rectangle(frame, cv::Point(10, 2), cv::Point(100,20),
* cv::Scalar(255,255,255), -1);
* stringstream ss;
* ss << capture.get(CAP\_PROP\_POS\_FRAMES);
* string frameNumberString = ss.str();
* putText(frame, frameNumberString.c\_str(), cv::Point(15, 15),
* FONT\_HERSHEY\_SIMPLEX, 0.5 , cv::Scalar(0,0,0));
* We are ready to show the current input frame and the results.
* //show the current frame and the fg masks
* imshow("Frame", frame);
* imshow("FG Mask", fgMask);



**DESIGN DIAGRAM**