# LAB: Color Image Segmentation

**Date:** 2025-Apr-15

**Author:** Joungbin Choi 22200757

**Github:** repository link

**Demo Video:** Youtube link

---

# Introduction

## 1. Objective

**Goal**: Find the HSV of a certain color and make an object as invisible.

This lab is focusing on color detection. After color detecting, making invisible object with masking and background substitution.

## 2. Preparation

### Software Installation

- OpenCV 3.83, Clion2024.3.5
- CUDA 12.6, cudatoolkit 12.6, C++

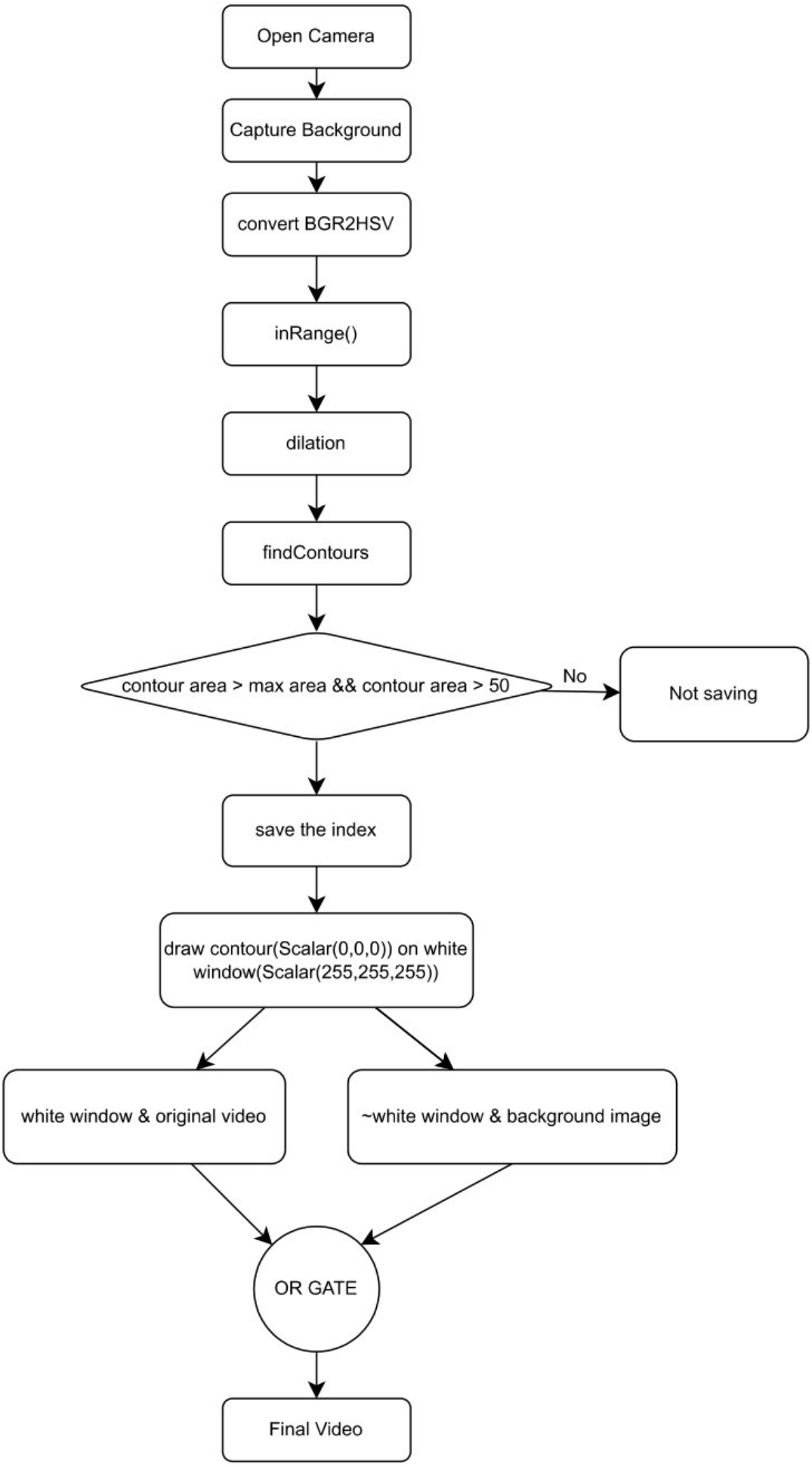
### Dataset

In test video, there is a man with a rectangular box. The goal is to make a box as invisible.

**Dataset link:** Download the test video

# Algorithm

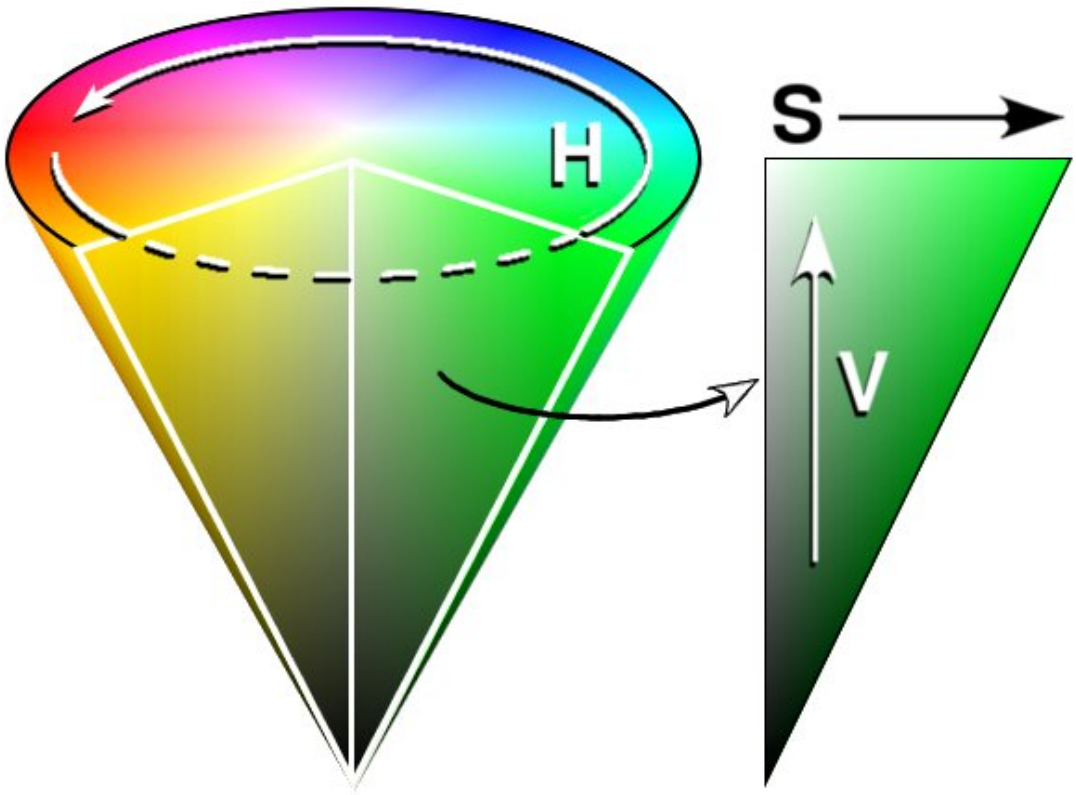
## 1. Overview


Open Camera
Capture Background
convert BGR2HSV
inRange()
dilation
findContours
contour area > max area && contour area > 50
No
Not saving
save the index
draw contour(Scalar(0,0,0)) on white window(Scalar(255,255,255))
white window & original video
~white window & background image
OR GATE
Final Video

# 2. Procedure

## HSV analysis

HSV color space is one of the color space to express the color. Hue represents the color type, Saturation indicates the intensity or how much white is included, and Value defines the brightness.




In OpenCV, Hue range is [0, 179], Saturation range is [0, 255], and Value range is also [0, 255]. To detect a certain color, it is possible to set H, S, V values and use `inRange()` function.

```
inRange(src, Scalar(MIN(hmin, hmax), MIN(smin, smax), MIN(vmin, vmax)),
Scalar(MAX(hmin, hmax), MAX(smin, smax), MAX(vmin, vmax)), dts);
```

## Dilation

To decrease the noise, morphology process is included. Since we want to mask larger space than the origin, it is necessary to process the dilation.

```
Mat kernel = getStructuringElement(MORPH_ELLIPSE, Size(13, 13));
dilate(video_erase, video_erase, kernel);
```

## Contour

After dilation, find the contour to draw a mask. Additionally, find the index of contour which has the maximum area. Draw the filled contour on white_window to get binary window.

```
/// Find the contours ///
findContours(video_erase, contours, RETR_EXTERNAL, CHAIN_APPROX_NONE);

/// Find the Contour with the largest area ///
double threshold_Area = 1500;
double maxArea = 0;
int maxArea_idx = 0;
if (contours.size() > 0) {

    for (int i = 0; i < contours.size(); i++)
    if (contourArea(contours[i]) > maxArea && contourArea(contours[i]) >
threshold_Area){
        maxArea = contourArea(contours[i]);
        maxArea_idx = i;
    }

    /// Drawing the Contour Box  ///
    drawContours(white_window, contours, maxArea_idx, Scalar(0, 0, 0), FILLED);

}
```

## Masking

Create the white window with the size of original video frame. In white_window, the area where we want to delete is filled with black( `Scalar(0,0,0)` ). When that white_window is calculated with original video with `AND` gate, we will get the video which the certain object has been deleted.

To fill the deleted part with background image, inverse of white_window is calculated with background image with `AND` gate.

Finally, multiply those two video with `OR` gate to get the video with invisible object.

```
/// Operating bitwise to get object and background image ///
Mat video_object = white_window & video;
Mat video_background = ~white_window & Background_image;

/// The result ///
video_disp = video_background | video_object;
```

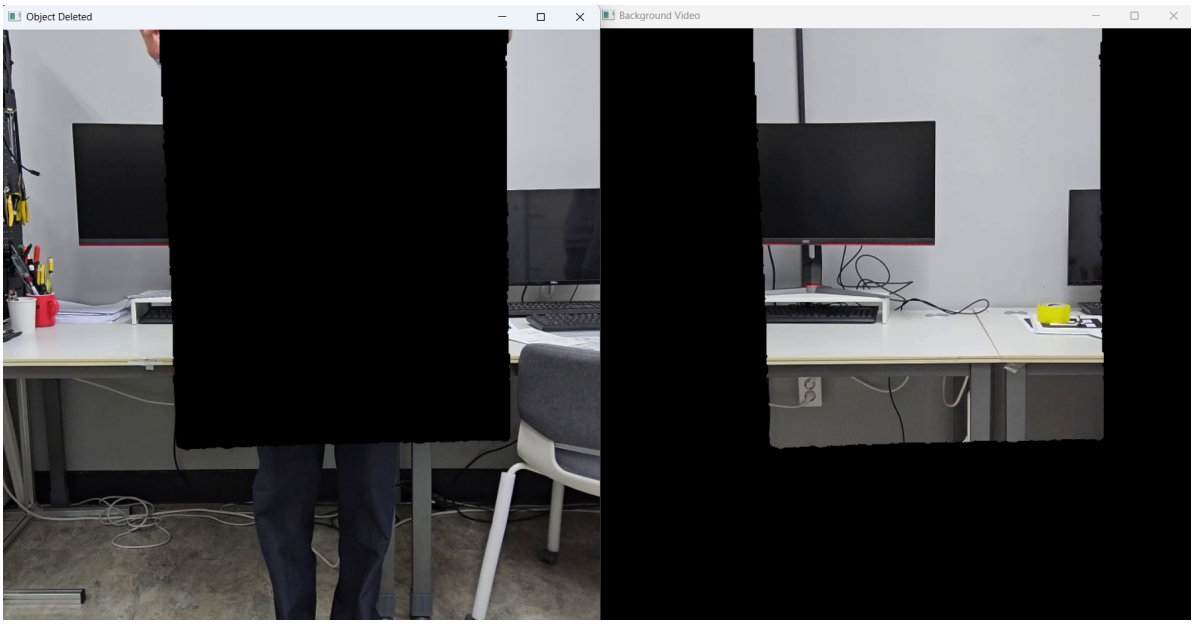

# Result and Discussion

## 1. Final Result

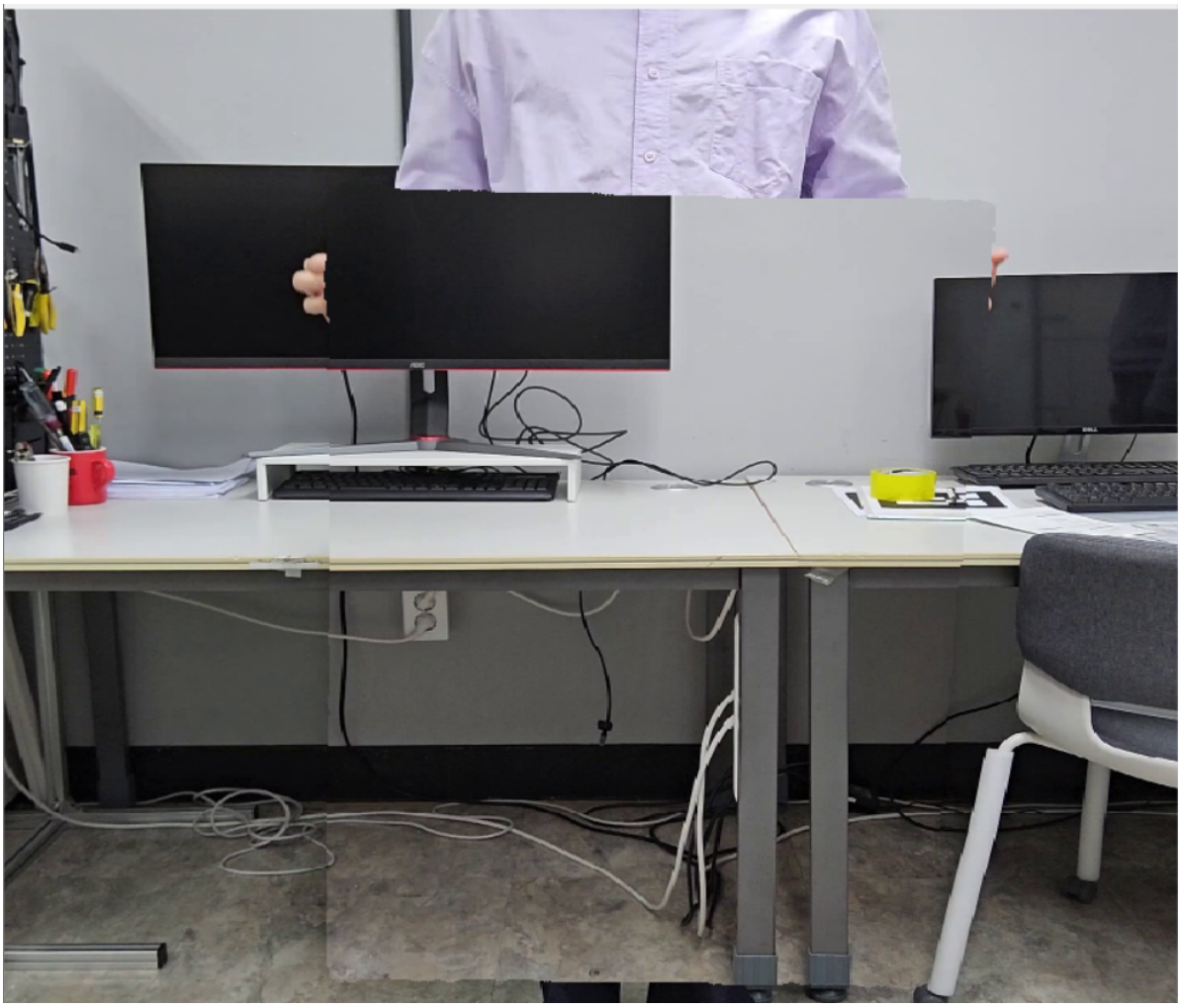
### Sample 1


The result image of converting gray to HSV.


The result image of dilation and white window.




The result image of mask and background after bit-wise operation.



The final result image.

### Sample 2

video

## 2. Discussion

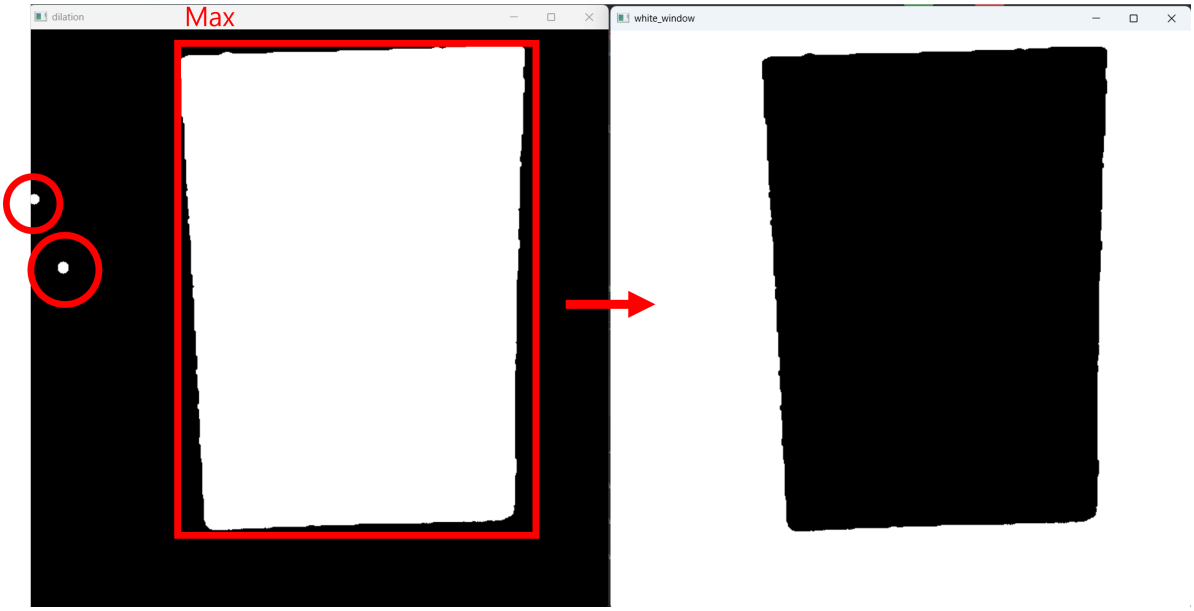
The HSV values are selected by experimental results.

```
int hmin = 13, hmax = 18, smin = 108, smax = 150, vmin = 130, vmax = 230;
```

The background image is capture when the video has been opened. It is captured before the `while()`

```
VideoCapture cap("../Image/LAB_MagicCloak_Sample1.mp4");
cap.read(Background_image);
```

There was a noise after dilation. Therefore, contours, which are less than threshold area, are deleted. Among them, the index of a contour with the max area is saved to draw on white window.




# Conclusion

The goal of this lab is to detected an object with a certain color and replace it with background image, so it seems like an object is invisible. To achieve that goal, understanding HSV color space is necessary. In OpenCV, the min and max of Hue, Saturation, and Value are inserted in `inRange()` function to get the classified color image. After that, the max area contour is selected as an object and draw in binary image. Drawing contour on binary image plane is to operate bit wise calculation.

The objects in sample 1 and 2 are well detected and replaced with background.

---

# Appendix

```
//
// Created by joung on 25. 4. 11.
//

// Sample 1

#include <iostream>
#include <opencv2/opencv.hpp> //Clion OpenCV Library

using namespace cv;
using namespace std;

Mat Background_image, video;
int hmin = 13, hmax = 18, smin = 108, smax = 150, vmin = 130, vmax = 230;

int main()
{
    Mat video_disp, video_hsv, video_erase;
    vector<vector<Point> > contours;
```

```
    /*  open the video  */
    VideoCapture cap("../Image/LAB_MagicCloak_Sample1.mp4");
    if (!cap.isOpened())
    {
        cout << "Cannot open the video cam\n";
        return -1;
    }

    /*  open Background Image  */
    cap.read(Background_image);

    while (true) {
        /*  capture the frame of video  */
        bool bSuccess = cap.read(video);
        if (!bSuccess)
        {
            cout << "Cannot find a frame from  video stream\n";
            break;
        }

        /// Final video and white window ///
        video.copyTo(video_disp);
        Mat white_window = Mat::zeros(video.size(), CV_8UC3) +
Scalar(255,255,255);

        /// convert BGR to HSV ///
        cvtColor(video, video_hsv, COLOR_BGR2HSV);

        /// Find the area which is in the desirable range ///
        inRange(video_hsv, Scalar(MIN(hmin, hmax), MIN(smin, smax), MIN(vmin,
vmax)), Scalar(MAX(hmin, hmax), MAX(smin, smax), MAX(vmin, vmax)), video_erase);

        /// Dilating to decrease the noise ///
        Mat kernel = getStructuringElement(MORPH_ELLIPSE, Size(13, 13));
        dilate(video_erase, video_erase, kernel);

        /// Find the contours ///
        findContours(video_erase, contours, RETR_EXTERNAL, CHAIN_APPROX_NONE);

        /// Find the Contour with the largest area ///
        double threshold_Area = 1500;
        double maxArea = 0;
        int maxArea_idx = 0;
        if (contours.size() > 0) {
            for (int i = 0; i < contours.size(); i++)
                if (contourArea(contours[i]) > maxArea &&
contourArea(contours[i]) > threshold_Area) {
                    maxArea = contourArea(contours[i]);
                    maxArea_idx = i;
                }

            ///  Drawing the Contour Box  ///
            drawContours(white_window, contours, maxArea_idx, Scalar(0, 0, 0),
FILLED);

            /// Operating bitwise to get object and background image ///
            Mat video_object = white_window & video;
            Mat video_background = ~white_window & Background_image;
```

```
            /// The result ///
            video_disp = video_background | video_object;
            imshow("Source", video_disp);
        }

        if (waitKey(10) == 27)
            break;
    }
    cap.release();
    destroyAllWindows();
}
```

```
//
// Created by joung on 25. 4. 15.
//

// Sample 2

#include <iostream>
#include <opencv2/opencv.hpp>

using namespace cv;
using namespace std;

Mat Background_image;
Mat video;
int hmin = 50, hmax = 100, smin = 60, smax = 255, vmin = 90, vmax = 255; //
Sample 2(Green)

int main()
{
    Mat video_disp, video_hsv, video_erase;
    vector<vector<Point> > contours;

    /*  open the video  */
    VideoCapture cap("sample2-1.mp4");
    // VideoCapture cap("sample2-2.mp4"); // Switch the video
    if (!cap.isOpened())
    {
        cout << "Cannot open the video cam/n";
        return -1;
    }

    /*  open Background Image  */
    cap.read(Background_image);

    namedWindow("Source", 0);

    while (true) {
        /*  capture the frame of video  */
        bool bSuccess = cap.read(video);
        if (!bSuccess)
        {
            cout << "Cannot find a frame from  video stream\n";
            break;
        }
```

```
        /// Final video and white window ///
        video.copyTo(video_disp);
        Mat white_window = Mat::zeros(video.size(), CV_8UC3) +
Scalar(255,255,255);

        /// convert BGR to HSV ///
        cvtColor(video, video_hsv, COLOR_BGR2HSV);

        /// Find the area which is in the desirable range ///
        inRange(video_hsv, Scalar(MIN(hmin, hmax), MIN(smin, smax), MIN(vmin,
vmax)), Scalar(MAX(hmin, hmax), MAX(smin, smax), MAX(vmin, vmax)), video_erase);

        /// Dilating to decrease the noise ///
        Mat kernel = getStructuringElement(MORPH_ELLIPSE, Size(13, 13));
        dilate(video_erase, video_erase, kernel);

        /// Find the contours ///
        findContours(video_erase, contours, RETR_EXTERNAL, CHAIN_APPROX_NONE);

        /// Find the Contour with the largest area ///
        double threshold_Area = 1500;
        if (contours.size() > 0) {
            for (int i = 0; i < contours.size(); i++)
                if (contourArea(contours[i])>threshold_Area) {
                    drawContours(white_window, contours, i, Scalar(0, 0, 0),
FILLED); //  Drawing the Contour Box
                }

            /// Operating bitwise to get object and background image ///
            Mat video_object = white_window & video;
            Mat video_background = ~white_window & Background_image;

            /// The result ///
            video_disp = video_background | video_object;

        }

        imshow("Source", video_disp);
        if (waitKey(10) == 27)
            break;
    }
    cap.release();
    destroyAllWindows();
}
```