

LAB2: Color Image Segmentation

Date: 2025-May-8

Author: Yechan Kim 22100153

Introduction

I. Objective

Goal: The objective is making the designated color region transparent, allowing the background to be visible through those areas using OpenCV's color segmentation technique.

II. Preparation

Software Installation

- OpenCV 4.9.0, Visual Studio 2022

Dataset

source video:

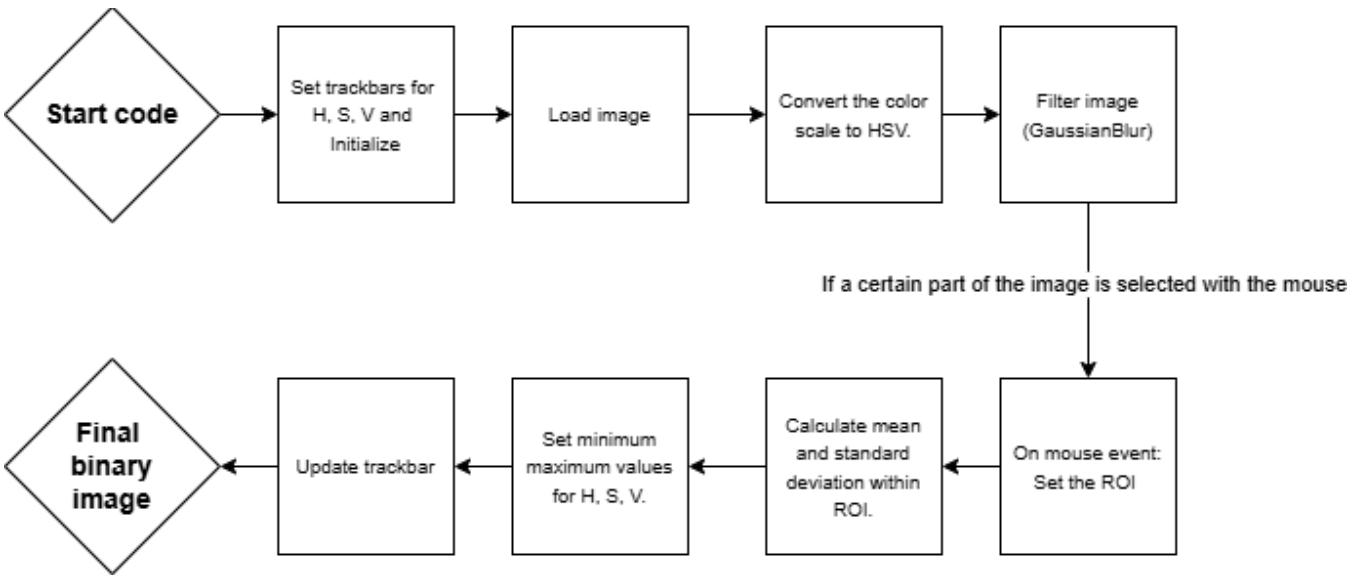
[source video](#)

I. Overview

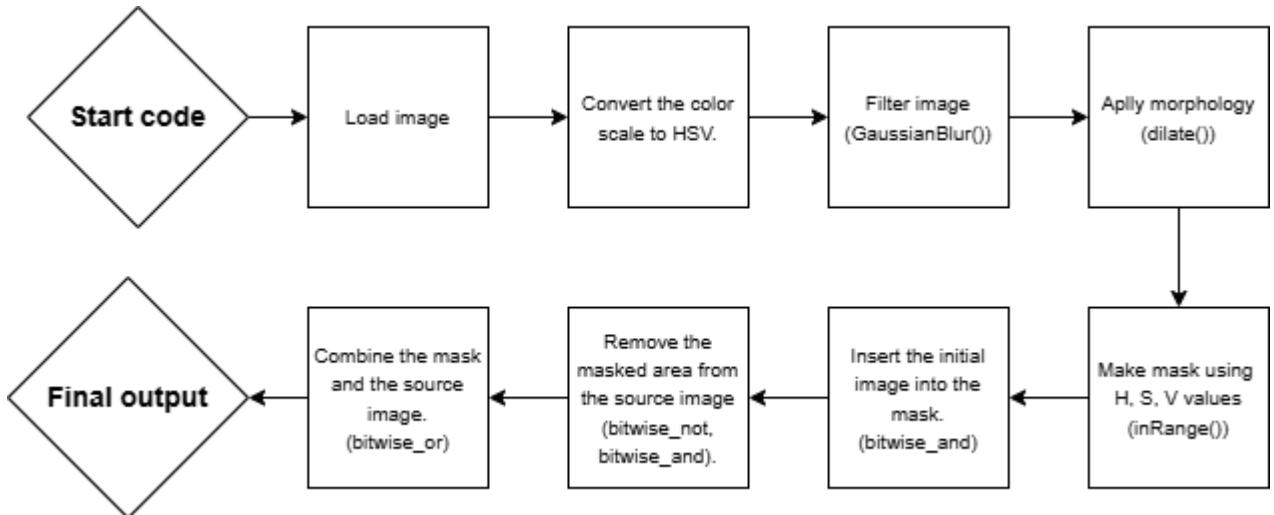
This project was carried out using two types of source code as follows:

- Finding H,S, and V values
- Color image segmentation

Flow chart for finding H,S, and V ranges:



Flow char for color image segmentation:



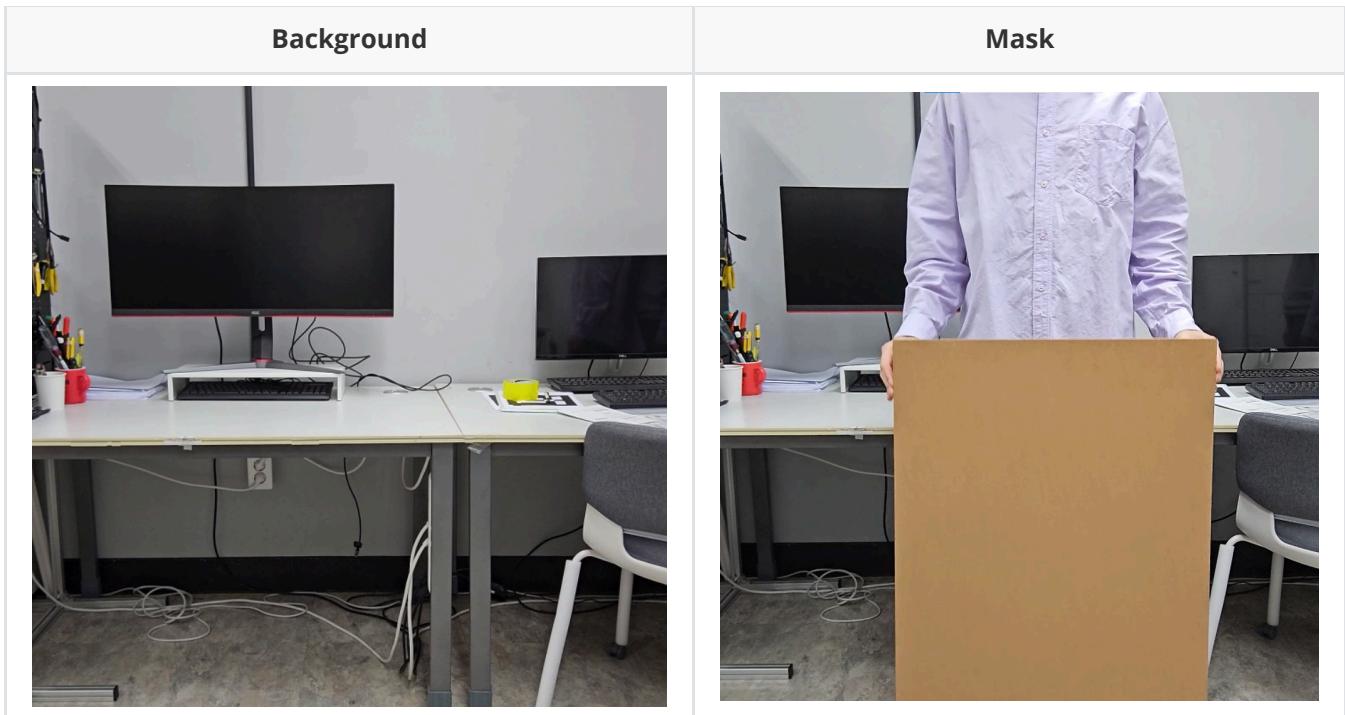
II. Procedure

1. Finding H, S, V values

This process involves displaying the source image, allowing the user to drag and select a desired region with the mouse, and then representing the H, S, and V values of the selected area using track bars.

When an ROI is set using the mouse, the areas in the source image that have similar colors to the ROI will be represented with a value of 255 in the final image.

Source images are as follows:



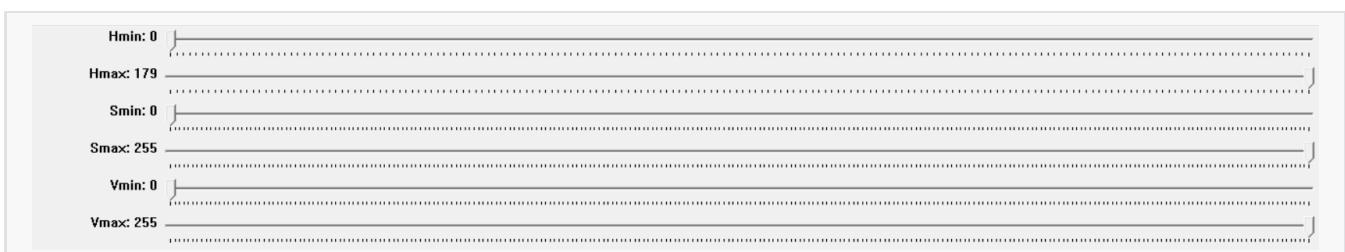
1-1. Setting the track bars and initializing H, S, and V values

To obtain the H, S, and V values of the selected region from the source image, it is first necessary to initialize the track bars.

The track bars consist of six in total, with minimum and maximum values for each of the H, S, and V components. The maximum and minimum values for each are as follows:

- H: 0 ~ 179
- S: 0 ~ 255
- V: 0 ~ 255

Bellow is initialized track bar and dst:



Code for this result is as follows:

```
// hsv스케일의 전체 범위
int hmin = 0, hmax = 179, smin = 0, smax = 255, vmin = 0, vmax = 255;

// 설정한 색깔의 hsv정보를 나타낼 트랙바 설정
namedWindow("Source", 0);
```

```

//콜백함수를 onMouse로 설정
setMouseCallback("Source", onMouse, 0);

//트랙바 설정
createTrackbar("Hmin", "Source", &hmin, 179, 0);
createTrackbar("Hmax", "Source", &hmax, 179, 0);
createTrackbar("Smin", "Source", &smin, 255, 0);
createTrackbar("Smax", "Source", &smax, 255, 0);
createTrackbar("Vmin", "Source", &vmin, 255, 0);
createTrackbar("Vmax", "Source", &vmax, 255, 0);

```

1-2. Converting color scale and filtering

First, capture the image from each frame, convert color scale of the image from the BGR scale to the HSV. After that, apply Gaussian filter to reduce noises.

```

// 동영상을 한 프레임씩 가져온다.
Mat src;
VideoCapture cap("LAB_MagicCloak_Sample2.mp4");
bool bSuccess = cap.read(src);

if (!bSuccess) {
    cout << "End of video stream or error.\n";
    destroyAllWindows(); // 창 닫기
    break;
}

//받아온 이미지를 hsv스케일로 변환시킨다.
cvtColor(src, hsv, COLOR_BGR2HSV);

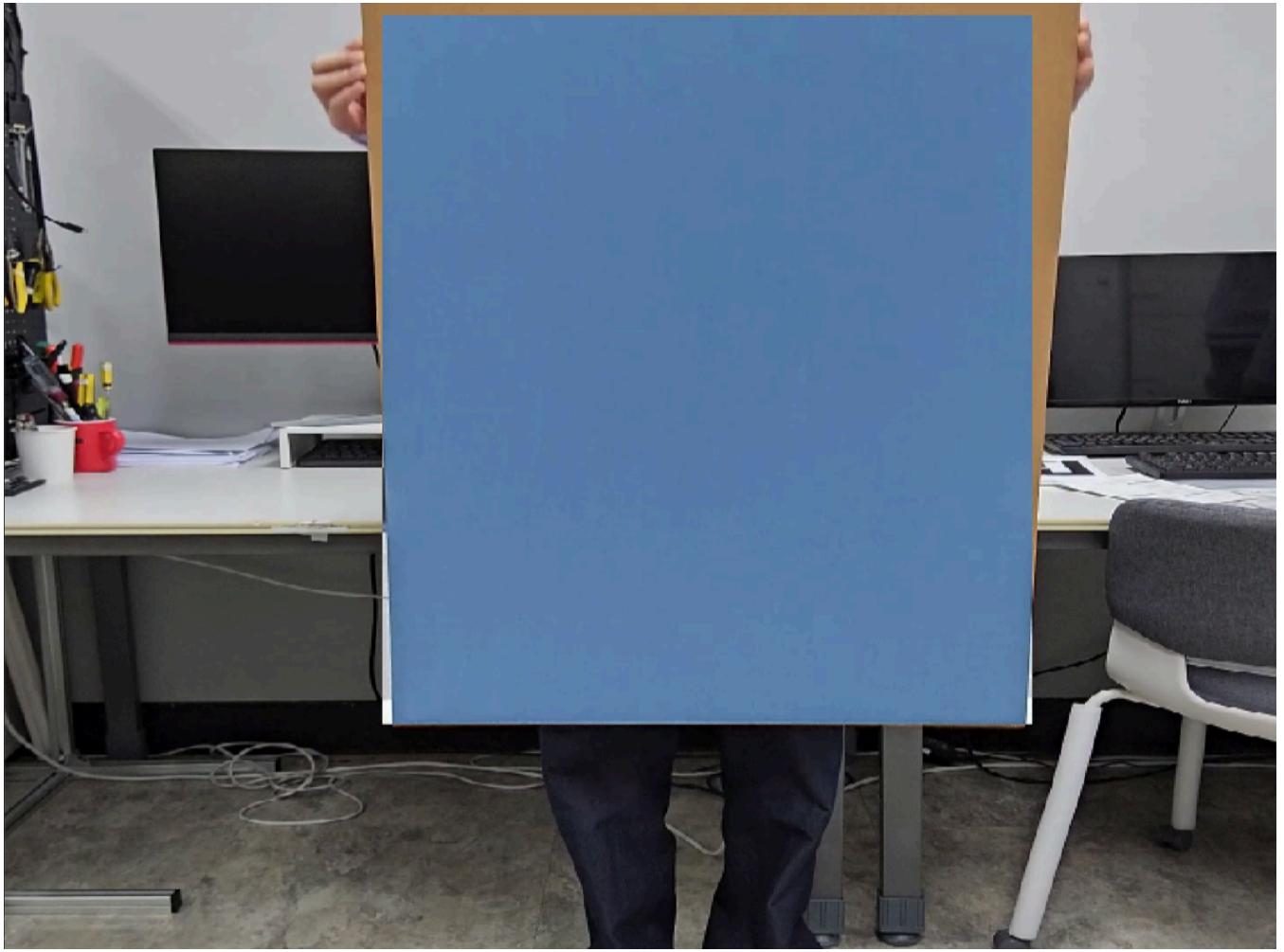
//노이즈 처리를 위해 smoothing한다.
GaussianBlur(hsv, hsv, Size(3, 3), 0);

```

1-3. Selecting ROI

Use the `onMouse()` event to select the area that will be used to create the mask.

Using this function, ROI is selected as follows:



```
* onMouse(int event, int x, int y, int, void*)
```

This function allows the user to click and drag the mouse to select a region of the image, which is then set as the ROI. If specific area is selected, the selected region is displayed using the bitwise_and function Inside the while loop.

```
//on mouse 이벤트
static void onMouse(int event, int x, int y, int, void*)
{
    if (src.empty()) return;

    //선택된 영역을 roi로 설정한다.
    if (selectObject && !src.empty())
    {
        //드래그를 시작한 순간과 이후 좌표를 비교하여 selection의 상단 좌표를 설정한다.
        selection.x = MIN(x, origin.x);
        selection.y = MIN(y, origin.y);

        //roi의 높이와 넓이를 설정한다.
        selection.width = abs(x - origin.x) + 1;
        selection.height = abs(y - origin.y) + 1;

        //roi를 적용시킨다.
    }
}
```

```

        selection &= Rect(0, 0, src.cols, src.rows);
    }

//마우스 버튼을 누르고 일정 영역을 선택하면 selectObject가 참이 된다.
switch (event)
{
case EVENT_LBUTTONDOWN:
    selectObject = true;
    origin = Point(x, y);
    break;
case EVENT_LBUTTONUP:
    selectObject = false;
    if (selection.area())
        trackObject = true;
    break;
}
}

if (selectObject && selection.area() > 0) // Left Mouse is being clicked and dragged
{
    //Mouse Drag을 화면에 보여주기 위함
    Mat roi_RGB(image_disp, selection);
    bitwise_not(roi_RGB, roi_RGB);

}

```

1-4. Updating track bar and making mask

If a region is selected by dragging, `meanStdDev()` calculates the mean and standard deviation of the HSV values for that region. Using these, the H, S, and V value ranges are determined as follows:

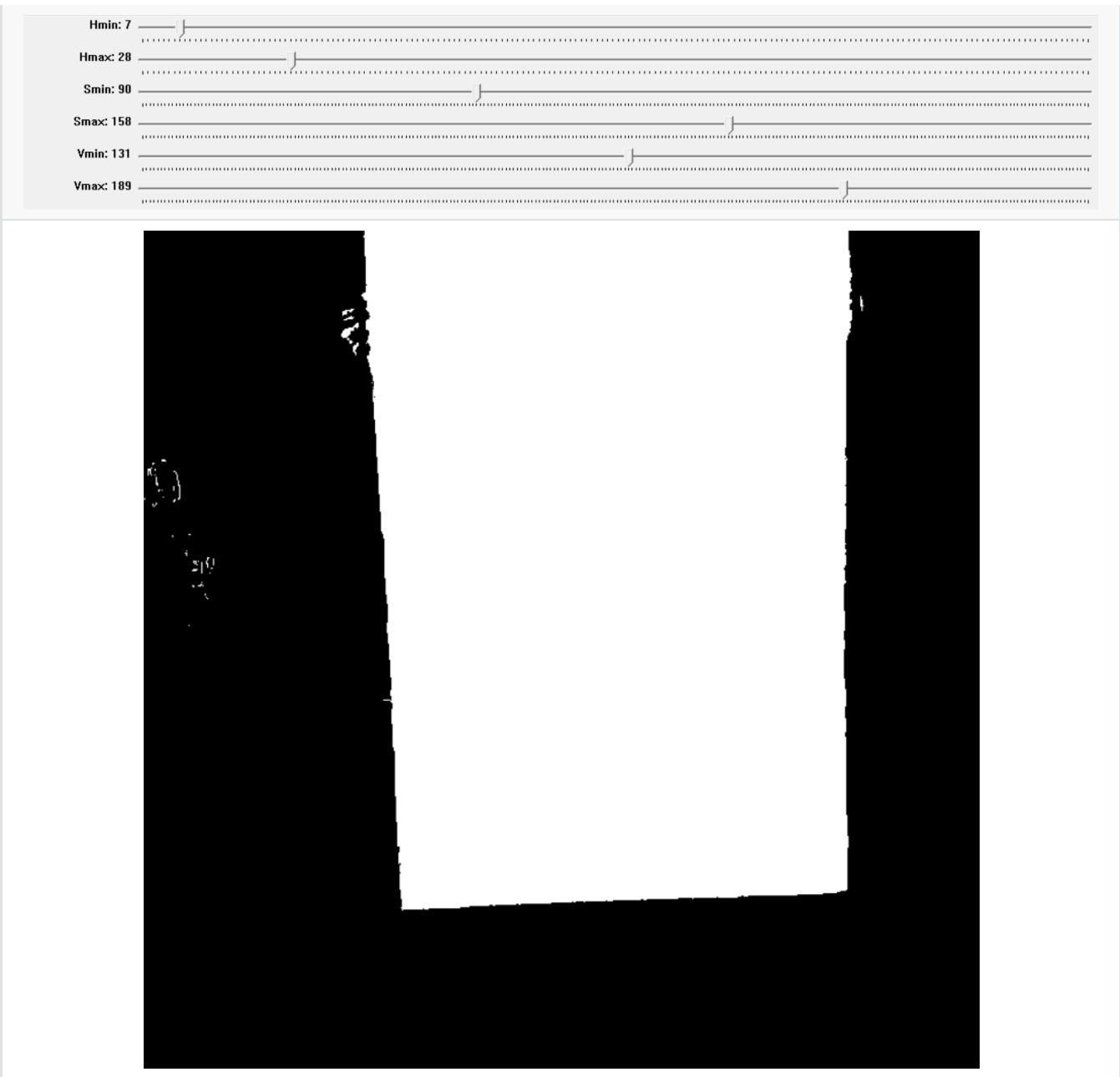
- min = (mean) - (standard deviation)
- max = (mean) + (standard deviation)

Function `onMouse()` acts as the callback function for the track bars and updates their values to the ranges calculated above.

Mask is made by using `inRange()` function which makes pixels that fall within the specific range keep white.

Obtained H, S, and V ranges are as follows:

- h: 7 ~ 28
- s: 90 ~ 158
- v: 131 ~ 189



```

inRange(hsv, Scalar(MIN(hmin, hmax), MIN(smin, smax), MIN(vmin, vmax)),
        Scalar(MAX(hmin, hmax), MAX(smin, smax), MAX(vmin, vmax)), dst); // 보고자 하는 색상정보

if (trackObject)
{
    //해당 이벤트가 계속하여 발생하지 않도록 초기화한다.
    trackObject = false;

    //hsv스케일로 선택된 영역에 대해 roi영역을 만들기 위한 변수
    Mat roi_HSV(hsv, selection);

    //선택된 영역의 각 h,s,v에 대해 평균과 표준편차를 구한다.
    Scalar means, stddev;
    meanStdDev(roi_HSV, means, stddev);
    cout << "\n Selected ROI Means= " << means << "\n stddev= " << stddev;
}

```

```

//트랙바의 값을 평균과 표준편차를 이용하여 변환시킨다. (평균-표준편차=최솟값, 평균+표준편차=최댓값)
hmin = MAX((means[0] - stddev[0]), 0);
hmax = MIN((means[0] + stddev[0]), 179);
setTrackbarPos("Hmin", "Source", hmin);
setTrackbarPos("Hmax", "Source", hmax);

smin = MAX((means[1] - stddev[1]), 0);
smax = MIN((means[1] + stddev[1]), 255);
setTrackbarPos("Smin", "Source", smin);
setTrackbarPos("Smax", "Source", smax);

vmin = MAX((means[2] - stddev[2]), 0);
vmax = MIN((means[2] + stddev[2]), 255);
setTrackbarPos("Vmin", "Source", vmin);
setTrackbarPos("Vmax", "Source", vmax);

}

```

* `meanStdDev(InputArray src, OutputArray mean, OutputArray stddev, InputArray mask = noArray());` : It calculates the mean and the standard deviation of array elements, independently for each channel. It is often used in image processing to analyze the brightness and contrast characteristics of an image or a specific region of interest (ROI).

* `inRange(src, lowerb, upperb, dst);` : It checks each pixel to see if it falls within a specified minimum and maximum value for each channel. If a pixel's value falls within the range, the output pixel is set to 255 (white); otherwise, it is set to 0 (black). This results in a binary mask highlighting the desired areas.

2. Color segmentation

In this step, Magic cloak is created using the H, S, and V ranges of the ROI obtained from the previous step.

2-1. Loading image and converting color scale

Load the image and then convert its color space from BGR to HSV. Additionally, apply `GaussianBlur()` to reduce noises

```

//주어진 동영상을 가져온다. (Sample 1, 2 각각 따로)
VideoCapture cap("LAB_MagicCloak_Sample1.mp4");
//VideoCapture cap("LAB_MagicCloak_Sample2.mp4");

// 매 프레임을 읽어온다.
Mat src;
bool bSuccess = cap.read(src);

//이미지처리를 위해 소스이미지를 복사하여 가져온다.
image = src.clone();

//기존 BGR스케일이던 이미지를 HSV스케일로 바꾼다.

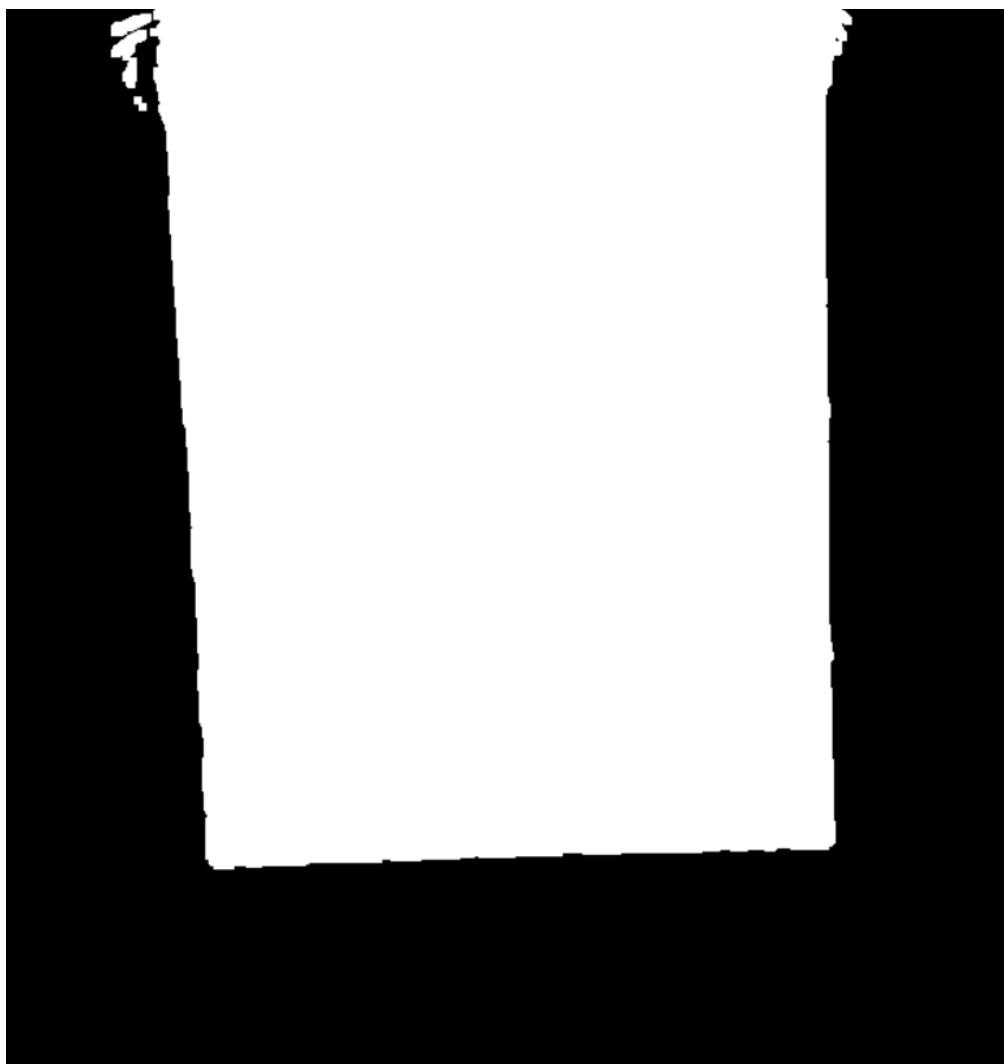
```

```
cvtColor(image, hsv, COLOR_BGR2HSV);  
  
//Smoothing하여 이미지의 노이즈를 제거한다.  
GaussianBlur(hsv, hsv, Size(3, 3), 0);
```

2-2 Creating mask and applying morphology

To create areas where the background appears transparent, the `inRange()` function was used to generate a mask for the ROI area, and Morphology were applied to the mask(Dilation for 2 times).

Results are as follows:

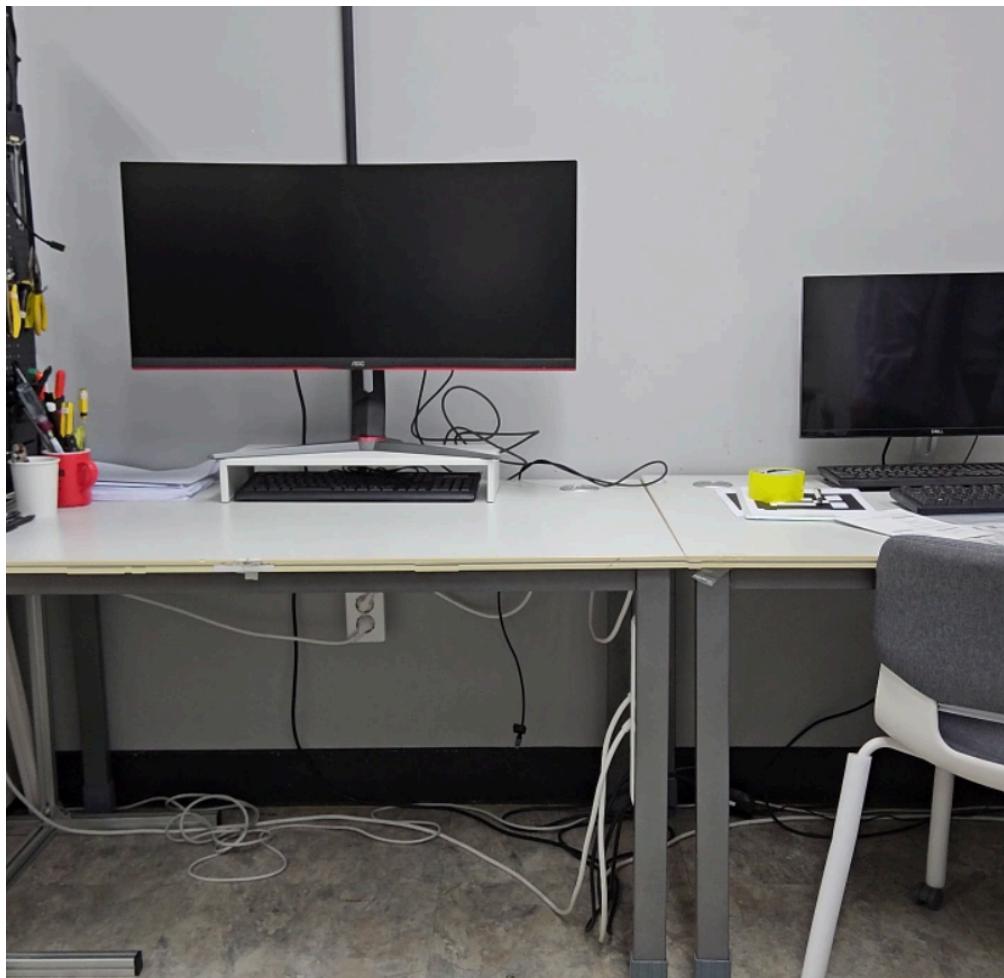


```
int hmin = 7, hmax = 28, smin = 90, smax = 158, vmin = 131, vmax = 189  
  
//기준에 찾았던 나무판자의 hsv값을 기반으로 mask를 만든다.  
inRange(hsv, Scalar(hmin, smin, vmin), Scalar(hmax, smax, vmax), mask);  
  
//마스크 이미지가 깨지는 것을 막기 위해 morphology를 진행한다.  
morphologyEx(mask, mask, MORPH_DILATE, kernel, Point(-1, -1), 1);
```

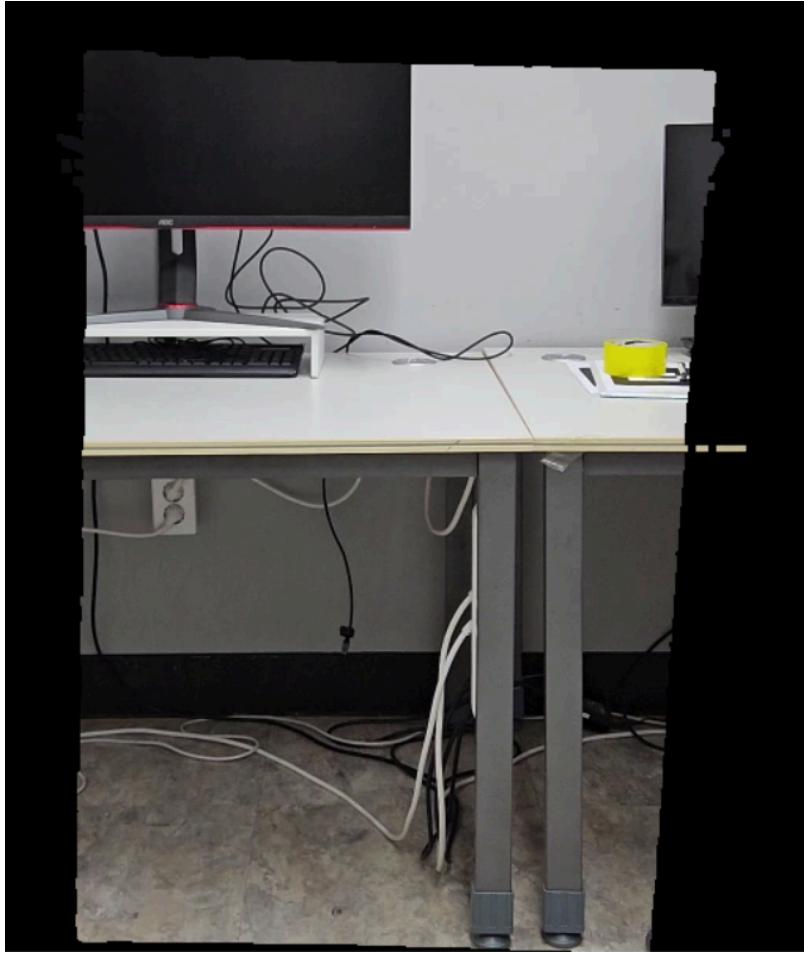
2-3. Creating cloak

To display the background in the ROI area, the first frame of the video—before the object used as the ROI appears—is saved as the background.

Background image is as follows:



Then, `bitwise_and()` is used to apply this background image to the mask created in the previous step.



```
//while loop 이전  
Mat background;  
cap.read(background);  
  
//ROI부분에 초기이미지를 넣기 위한 이미지  
Mat cloak;  
  
//Mask에 초기이미지 넣기  
bitwise_and(background, mask, cloak);
```

2-4. Creating the outer area of the cloak.

The masked regions in the original image are set to zero, allowing only the unmasked areas to be displayed by using `bitwise_not` for the mask, and `bitwise_and`.



```
//ROI부분이 0인 이미지를 넣기 위한 행렬  
Mat outside_cloak;  
  
//기존 이미지에서 마스크부분을 0으로 처리한다.  
bitwise_not(mask, mask);  
  
//마스크 부분만 검정색인 이미지를 만들어낸다.  
bitwise_and(image, mask, outside_cloak);
```

2-5. Creating final output

The initial mask containing the image is combined with the background, where the masked areas are set to zero using `bitwise_or()` , to produce the final output.

```
//최종 결과물을 송출할 이미지.  
Mat image_disp;  
  
//각각의 이미지를 더하여 최종 결과물을 만들어낸다.  
bitwise_or(outside_cloak, cloak, image_disp);
```

Result and Discussion

I. Final Result

Final results are as follows:



Same process was conducted on Sample 2. The result is as follows:



Demo video of Sample 2:

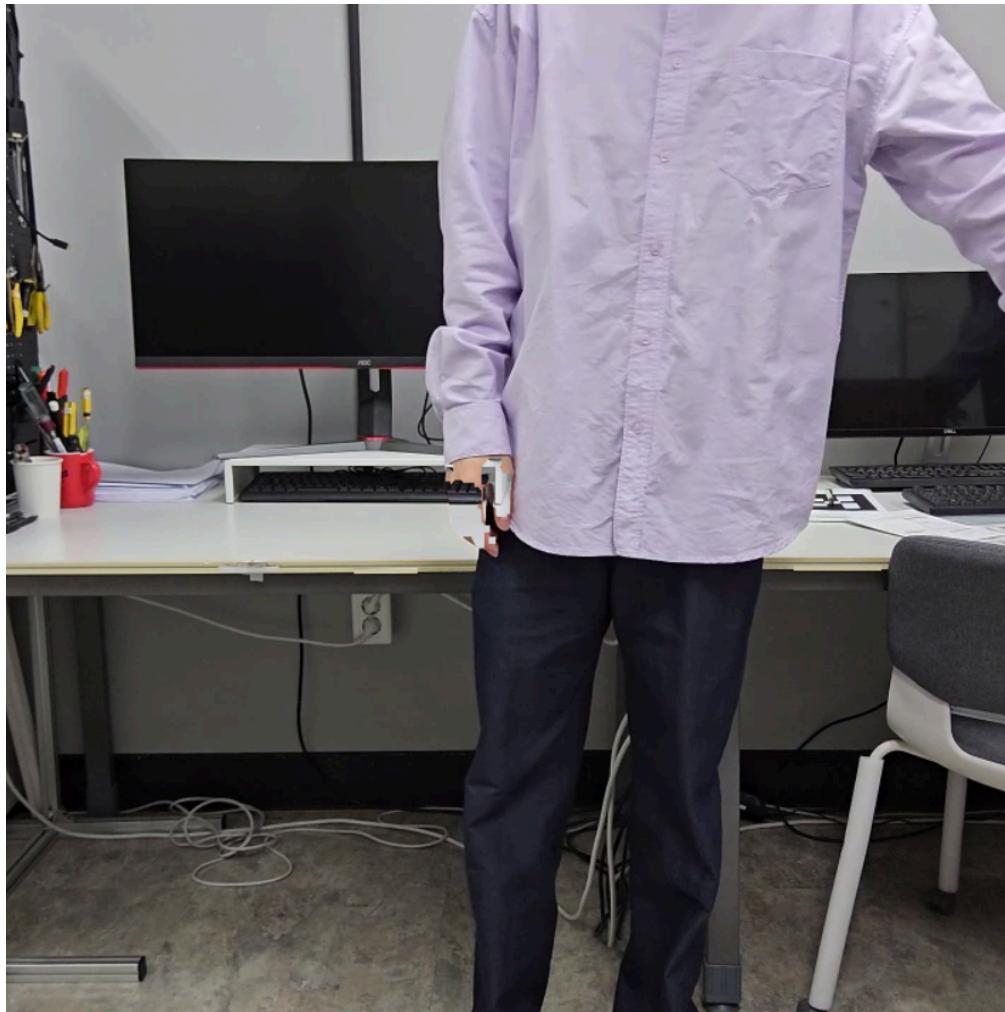
[Result of Sample 2](#)

II. Discussion

Overall, the designated color region appeared transparent as intended. However, a drawback was that some parts of the external image falling within the same H, S, and V range were also rendered transparent. On sample 1, parts of the hand and the desk in the video had similar colors to the wooden board set as the ROI, which limited the ability to cleanly isolate the wooden board area and reveal only the background.

When such issues occur, the limitations become more apparent during the application of morphology. Techniques like dilation, used to preserve the ROI area without damage, also increase the distortion of background regions with similar colors.

Below shows the limitation of Sample 1:



To avoid results like the above, a bright green color, which was not present in the surroundings, was used for the area to be set as the ROI when recording Sample 2, ensuring maximum separation from the background.

Below is input image of Sample 2:



As a result, there was no background distortion as follows:



This highlighted that The fewer similar colors there are around the object set as the ROI, the cleaner the resulting output will be and if we made more precise range for H, S, and V, it going to be led a better result.

Conclusion

The objective of this study was making the designated color region transparent, allowing the background to be visible through those areas using OpenCV's color segmentation technique. To do this, first we selected ROI using `onMouse()` that allows us to make area by dragging mouse. Then we found H, S, and V range by using `meanStdDev()` and `inRange()`. With this range, the mask of the ROI was created, we applied morphology on it. By using `bitwise_and()` and `bitwise_not`, we allowed the mask to show the background image and created outside area. Finally, we combined two images using `bitwise_or()`, got the final result.

The result was successful, but there was limitation. If some parts of the external image falling within the same H, S, and V range were also rendered transparent. This demonstrated that, when recording the source video, the ROI area should be set to a color distinct from the surroundings, and that a more precise range setting is necessary.

Appendix

Source code of finding H, S, and V values

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

using namespace cv;
using namespace std;

//루프 속에서 동영상의 프레임을 받아올 변수
Mat src;

//selection을 만들 때 기준 좌표를 표시하기 위함
Point origin;

//ROI가 지정될 영역
Rect selection;

//마우스로 특정 영역이 지정되었는지 확인하기 위함.
bool selectObject = false;
bool trackObject = false;

//HSV스케일의 전체 범위
int hmin = 0, hmax = 179, smin = 0, smax = 255, vmin = 0, vmax = 255;

// on mouse 이벤트
static void onMouse(int event, int x, int y, int, void*);

int main()
{
    //최종적으로 화면에 보일 이미지
    Mat image_disp;

    //HSV스케일로 바꾼 이미지를 저장할 공간
    Mat hsv;

    //inRange의 결과물이 저장될 공간
    Mat dst;

    //샘플 동영상
    videoCapture cap("../LAB_MagicCloak_Sample2.mp4");
    //videoCapture cap("../LAB_MagicCloak_Sample1.mp4");

    //동영상을 불러오지 못했다면 에러 메세지를 출력한다.
```

```

if (!cap.isOpened())
{
    cout << "Cannot open the video cam\n";
    return -1;
}

//설정한 색깔의 hsv정보를 나타낼 트랙바 설정
namedWindow("Source", 0);

//콜백함수를 onMouse로 설정
setMouseCallback("Source", onMouse, 0);

//트랙바 설정
createTrackbar("Hmin", "Source", &hmin, 179, 0);
createTrackbar("Hmax", "Source", &hmax, 179, 0);
createTrackbar("Smin", "Source", &smin, 255, 0);
createTrackbar("Smax", "Source", &smax, 255, 0);
createTrackbar("Vmin", "Source", &vmin, 255, 0);
createTrackbar("Vmax", "Source", &vmax, 255, 0);

//waitKey함수를 위한 변수 설정
int key = 0;

while (true)
{
    //동영상을 한 프레임씩 받아와 src에 저장한다.
    bool bSuccess = cap.read(src);

    if (!bSuccess) {
        cout << "End of video stream or error.\n";
        destroyAllWindows(); // 창 닫기
        break;
    }

    //30ms간 기다리고, esc가 눌리면 루프를 빠져나간다.
    key = waitKey(30);
    if (key == 27)
    {
        cout << "ESC key is pressed by user\n";
        break;
    }

    //받아온 이미지를 hsv스케일로 변환시킨다.
    cvtColor(src, hsv, COLOR_BGR2HSV);

    //노이즈 처리를 위해 smoothing한다.
    GaussianBlur(hsv, hsv, Size(3, 3), 0);

    //영역이 선택되기 이전 hsv값을 초기화하여 dst에 저장한다.
}

```

```

inRange(hsv, Scalar(MIN(hmin, hmax), MIN(smin, smax), MIN(vmin, vmax)),
        Scalar(MAX(hmin, hmax), MAX(smin, smax), MAX(vmin, vmax)), dst); // 보고자 하는 색
상정보

imshow("dst", dst);
//마우스로 선택된 영역이 0보다 클 경우 if문이 작동한다.
if (trackObject)
{
    //해당 이벤트가 계속하여 발생하지 않도록 초기화한다.
    trackObject = false;

    //hsv스케일로 선택된 영역에 대해 roi영역을 만들기 위한 변수
    Mat roi_HSV(hsv, selection);

    //선택된 영역의 각 h,s,v에 대해 평균과 표준편차를 구한다.
    Scalar means, stddev;
    meanStdDev(roi_HSV, means, stddev);
    cout << "\n Selected ROI Means= " << means << "\n stddev= " << stddev;

    //트랙바의 값을 평균과 표준편차를 이용하여 변환시킨다. (평균-표준편차=최솟값, 평균+표준편차
    //=최댓값)
    hmin = MAX((means[0] - stddev[0]), 0);
    hmax = MIN((means[0] + stddev[0]), 179);
    setTrackbarPos("Hmin", "Source", hmin);
    setTrackbarPos("Hmax", "Source", hmax);

    smin = MAX((means[1] - stddev[1]), 0);
    smax = MIN((means[1] + stddev[1]), 255);
    setTrackbarPos("Smin", "Source", smin);
    setTrackbarPos("Smax", "Source", smax);

    vmin = MAX((means[2] - stddev[2]), 0);
    vmax = MIN((means[2] + stddev[2]), 255);
    setTrackbarPos("Vmin", "Source", vmin);
    setTrackbarPos("Vmax", "Source", vmax);

}

src.copyTo(image_disp);

if (selectObject && selection.area() > 0) // Left Mouse is being clicked and
dragged
{
    //Mouse Drag을 화면에 보여주기 위함
    Mat roi_RGB(image_disp, selection);
    bitwise_not(roi_RGB, roi_RGB);

}

//최종적으로 처리된 이미지를 출력한다.
imshow("Source", image_disp);

}

```

```

    return 0;
}

//on mouse 이벤트
static void onMouse(int event, int x, int y, int, void*)
{
    if (src.empty()) return;

    //선택된 영역을 roi로 설정한다.
    if (selectObject && !src.empty())
    {
        //드래그를 시작한 순간과 이후 좌표를 비교하여 selection의 상단 좌표를 설정한다.
        selection.x = MIN(x, origin.x);
        selection.y = MIN(y, origin.y);

        //roi의 높이와 넓이를 설정한다.
        selection.width = abs(x - origin.x) + 1;
        selection.height = abs(y - origin.y) + 1;

        //roi를 적용시킨다.
        selection &= Rect(0, 0, src.cols, src.rows);
    }

    //마우스 버튼을 누르고 일정 영역을 선택하면 selectObject가 참이 된다.
    switch (event)
    {
        case EVENT_LBUTTONDOWN:
            selectObject = true;
            origin = Point(x, y);
            break;
        case EVENT_LBUTTONUP:
            selectObject = false;
            if (selection.area())
                trackObject = true;
            break;
    }
}

```

Source code of Sample 1

```

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

using namespace cv;
using namespace std;

```

```

int main()
{
    //동영상의 한 프레임을 가져와 이미지처리를 하기 위한 행렬
    Mat image;

    //hsv스케일로 이미지처리를 위함
    Mat hsv;

    //inRange함수를 거친 이후 나무판자와 같은 색깔의 부분만 255, 나머지는 0으로 채운 binary 이미지
    Mat mask;

    //dilate를 위한 커널
    Mat kernel = getStructuringElement(MORPH_RECT, Size(5, 5));

    //나무판자의 hsv값
    int hmin = 7, hmax = 28, smin = 90, smax = 158, vmin = 131, vmax = 189;

    //비디오가 녹화중인지 여부
    bool bRec = false;

    //주어진 동영상을 가져온다.
    videoCapture cap("../LAB_MagicCloak_Sample1.mp4");

    //동영상을 가져오지 못했을 경우 에러메세지를 띄운다.
    if (!cap.isOpened())
    {
        cout << "Cannot open the video cam\n";
        return -1;
    }

    //원본 이미지에서 마스크 부분에 뒷 배경이 나오게 하기 위해 맨 처음 프레임을 저장한다.
    Mat background;
    cap.read(background);

    while (true)
    {
        // 매 프레임을 읽어온다.
        Mat src;
        bool bSuccess = cap.read(src);

        //이미지처리를 위해 소스이미지를 복사하여 가져온다.
        image = src.clone();

        //최종 결과물을 송출할 이미지.
        Mat image_disp;

        //배경이 보이는 나무판자를 넣기 위한 행렬
        Mat cloak;

```

```

//나무판자부분이 0인 이미지를 넣기 위한 행렬
Mat outside_cloak;

//이미지를 가져오지 못했을 경우 루프를 종료시킨다.
if (!bSuccess)
{
    cout << "Cannot find a frame from video stream\n";
    break;
}

//기존 BGR스케일이던 이미지를 HSV스케일로 바꾼다.
cvtColor(image, hsv, COLOR_BGR2HSV);

//Smoothing하여 이미지의 노이즈를 제거한다.
GaussianBlur(hsv, hsv, Size(3, 3), 0);

//기준에 찾았던 나무판자의 hsv값을 기반으로 mask를 만든다.
inRange(hsv, Scalar(hmin, smin, vmin), Scalar(hmax, smax, vmax), mask);

//마스크 이미지가 깨지는 것을 막기 위해 morphology를 진행한다.
morphologyEx(mask, mask, MORPH_DILATE, kernel, Point(-1, -1), 1);

//BGR채널 이미지와 bitwise연산을 하기 위해 mask의 스케일을 바꿔준다.
cvtColor(mask, mask, COLOR_GRAY2BGR);

//마스크 부분에 배경이 나오도록 처리한다.
bitwise_and(background, mask, cloak);

//기존 이미지에서 마스크부분을 0으로 처리한 후 원본 이미지에 마스크를 제외한 부분이 나오도록 처리한다.
bitwise_not(mask, mask);
bitwise_and(image, mask, outside_cloak);

//각각의 이미지를 더하여 최종 결과물을 만들어낸다.
bitwise_or(outside_cloak, cloak, image_disp);

//최종 이미지를 출력한다.
imshow("image disp", image_disp);

//esc키를 누르거나 영상이 끝나면 동영상을 최종적으로 저장한다.
char ch = waitKey(10);
if (ch == 27)
{
    cout << "ESC key is pressed by user\n";
    break;
}

return 0;
}

```

Source code of Sample 2

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <stdio.h>

using namespace cv;
using namespace std;

int main()
{
    //동영상의 한 프레임을 가져와 이미지처리를 하기 위한 행렬
    Mat image;

    //hsv스케일로 이미지처리를 위함
    Mat hsv;

    //inRange함수를 거친 이후 나무판자와 같은 색깔의 부분만 255, 나머지는 0으로 채운 binary 이미지
    Mat mask;

    //dilate를 위한 커널
    Mat kernel = getStructuringElement(MORPH_RECT, Size(5, 5));

    //휴대폰의 hsv값
    int hmin = 25, hmax = 45, smin = 90, smax = 255, vmin = 112, vmax = 224;

    //주어진 동영상을 가져온다.
    videoCapture cap("LAB_MagicCloak_Sample2.mp4");

    //동영상을 가져오지 못했을 경우 에러메세지를 띠운다.
    if (!cap.isOpened())
    {
        cout << "Cannot open the video cam\n";
        return -1;
    }

    //원본 이미지에서 마스크 부분에 뒷 배경이 나오게 하기 위해 맨 처음 프레임을 저장한다.
    Mat background;
    cap.read(background);

    while (true)
    {
        // 매 프레임을 읽어온다.
        Mat src;
        bool bSuccess = cap.read(src);
```

```

//이미지처리를 위해 소스이미지를 복사하여 가져온다.
image = src.clone();

//최종 결과물을 송출할 이미지.
Mat image_disp;

//이미지를 가져오지 못했을 경우 루프를 종료시킨다.
if (!bSuccess)
{
    cout << "Cannot find a frame from video stream\n";
    break;
}

// esc키를 30ms간 기다리고, esc키가 눌리면 프로그램을 종료한다.

//기존 BGR스케일이던 이미지를 HSV스케일로 바꾼다.
cvtColor(image, hsv, COLOR_BGR2HSV);

//smoothing하여 이미지의 노이즈를 제거한다.
GaussianBlur(hsv, hsv, Size(3, 3), 0);

//기존에 찾았던 나무판자의 hsv값을 기반으로 mask를 만든다.
inRange(hsv, Scalar(hmin, smin, vmin), Scalar(hmax, smax, vmax), mask);

//마스크 이미지가 깨지는 것을 막기 위해 Morphology를 적용한다.
erode(mask, mask, kernel);
morphologyEx(mask, mask, MORPH_DILATE, kernel, Point(-1, -1), 2);

//BGR채널 이미지와 bitwise연산을 하기 위해 mask의 스케일을 바꿔준다.
cvtColor(mask, mask, COLOR_GRAY2BGR);

//배경이 보이는 나무판자를 넣기 위한 행렬
Mat cloak;

//마스크 부분에 배경이 나오도록 처리한다.
bitwise_and(background, mask, cloak);

//나무판자부분이 0인 이미지를 넣기 위한 행렬
Mat outside_cloak;
//기존 이미지에서 마스크부분을 0으로 처리한다.
bitwise_not(mask, mask);

//마스크 부분만 검정색인 이미지를 만들어낸다.
bitwise_and(image, mask, outside_cloak);

//각각의 이미지를 더하여 최종 결과물을 만들어낸다.
bitwise_or(outside_cloak, cloak, image_disp);

//최종 이미지를 출력한다.
imshow("image disp", image_disp);

```

```
//esc키를 누르거나 영상이 끝나면 동영상을 최종적으로 저장한다.
char ch = waitKey(10);
if (ch == 27)
{
    cout << "ESC key is pressed by user\n";

    break;
}

return 0;
}
```