Snehashis Pal
M.Tech CSE
Roll no: 2018201072
CSE578: Computer Vision
Assignment 0

All programs were written using C++ in linux with OpenCV installed on the system from the ubuntu repository. The functional parts of the code are provided in this pdf. The code is also available in the repository https://github.com/snehashispal1995/cv_assign_0.git . GNU compiler collection(gcc) was use to compile all codes. The library requirements of opencv can be satisfied either by manually providing the libraries or through *pkg-config* utility as was done here.

I.a. Dividing image into frames

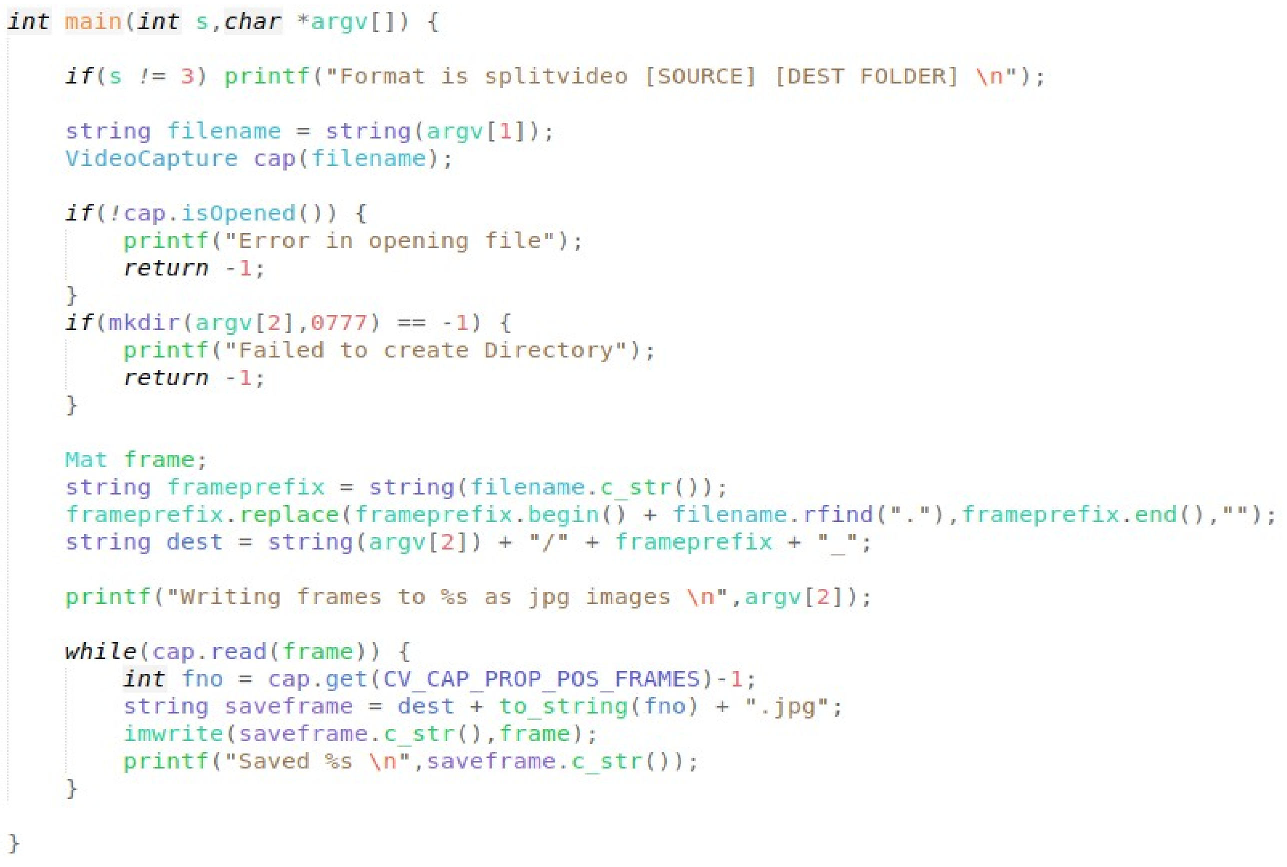
Every video is a set of temporally related images played one after another in a set period of time.
This task asks us to divide a video into its images using the opencv library. The general idea is read each frame of the video one by one and store them in a folder specified by the user.

The following procedure was used to achieve this.

1. Load the video using an instance of *VideoCapture*.
2. Read the frames of the video one by one by *VideoCapture::read*() storing each frame into a temporary 3 channel *Mat*.
3. Store the temporary *Mat* instance using *imwrite*().

All functions are a part of the opencv library. The program uses cmd line parameters as inputs.

C++ Code
https://github.com/snehashispal1995/cv_assign_0/blob/master/Q1/code1.cpp

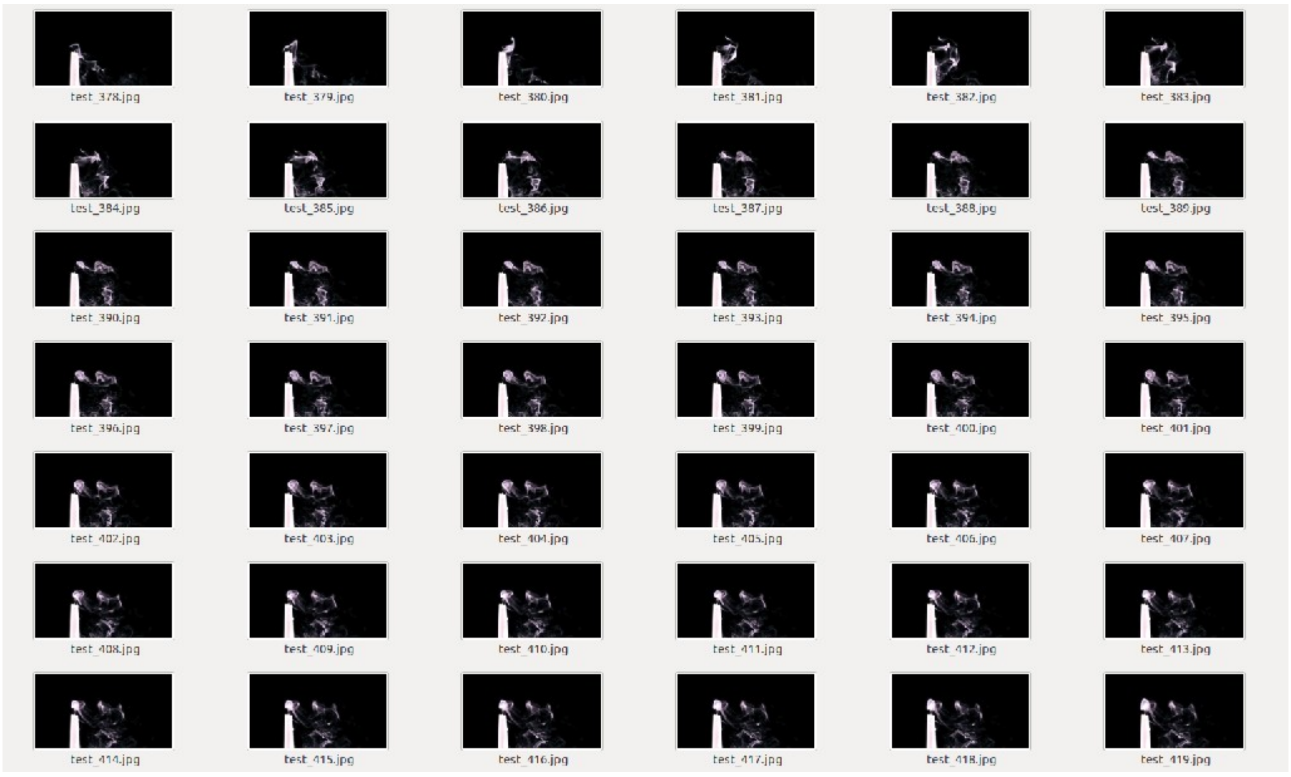


```
int main(int s,char *argv[]) {

    if(s != 3) printf("Format is splitvideo [SOURCE] [DEST FOLDER] \n");

    string filename = string(argv[1]);
    VideoCapture cap(filename);

    if(!cap.isOpened()) {
        printf("Error in opening file");
        return -1;
    }
    if(mkdir(argv[2],0777) == -1) {
        printf("Failed to create Directory");
        return -1;
    }

    Mat frame;
    string frameprefix = string(filename.c_str());
    frameprefix.replace(frameprefix.begin() + filename.rfind("."),frameprefix.end(),"");
    string dest = string(argv[2]) + "/" + frameprefix + "_";

    printf("Writing frames to %s as jpg images \n",argv[2]);

    while(cap.read(frame)) {
        int fno = cap.get(CV_CAP_PROP_POS_FRAMES)-1;
        string saveframe = dest + to_string(fno) + ".jpg";
        imwrite(saveframe.c_str(),frame);
        printf("Saved %s \n",saveframe.c_str());
    }

}
```

To run the program use the format : *splitvideo* [SOURCE VIDEO] [DESTINATION FOLDER].
Replace the name with whatever binary created if compiled locally.

Example.

Here is a section of the video https://www.youtube.com/watch?v=QwdHbh2kHUY as split frames.



I.b. Combining a set of images into a video.

This is the opposite of a as we need to stich together a set of frames into a video. In the process we need to additionally provide the framerate of the resultant video to be created. OpenCV has the functionality of creating videos using an instance of the *VideoWriter* class from a set of images. However it is upto the programmer in which order the frames are sequenced. As a result we must have a standardized procedure to select which frame comes after one another.

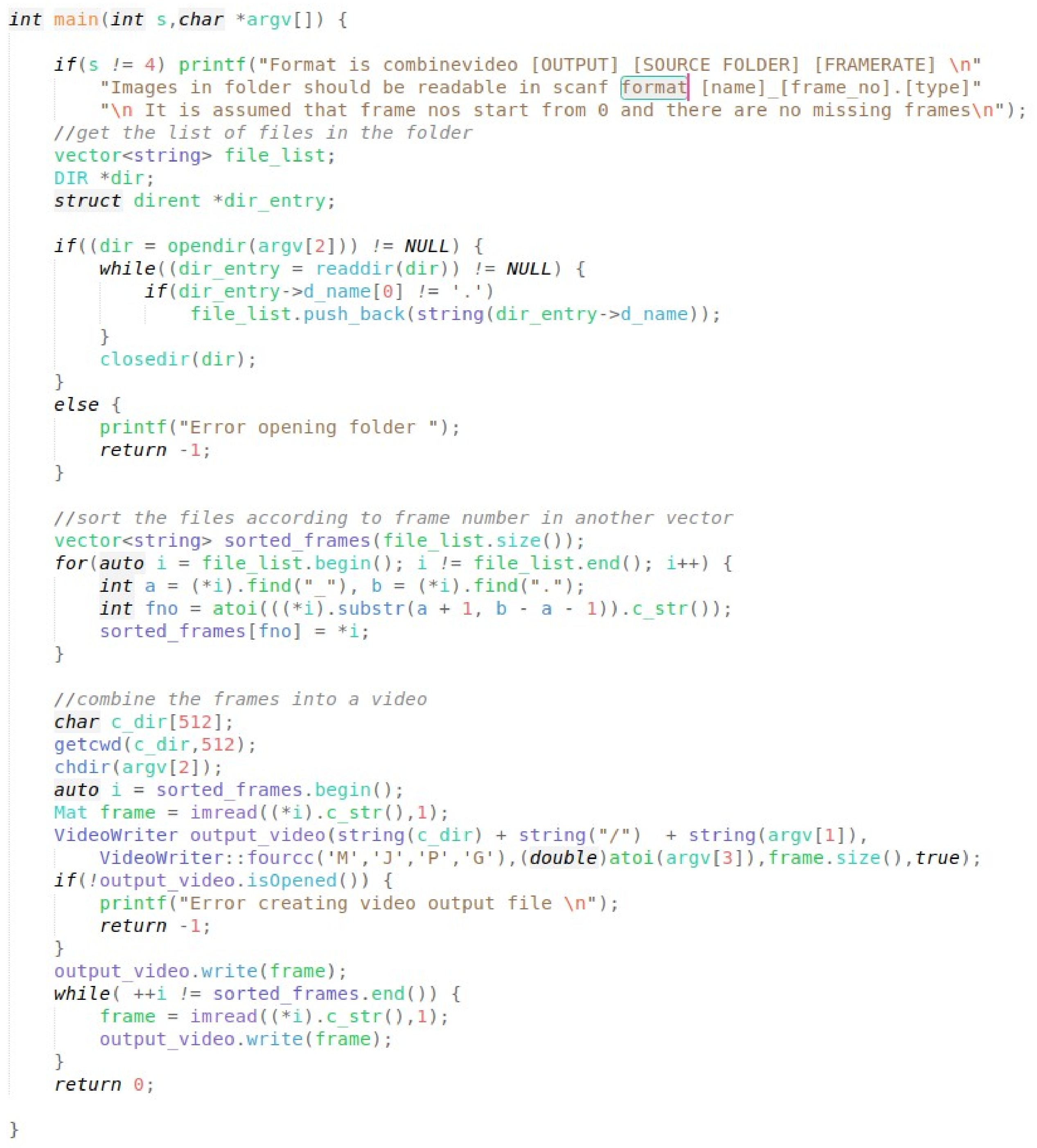
- All the frames of a video are kept in a single folder.
- All the frames are named according to its position in the video i.e. if a frame image is the 6th frame in sequence in the video its name will be [name]_6.[ext].
- There are no skipped frame images i.e the set of frames numbers are contigous.

We need to read the frames as images one by one. However its is not guaranteed by the OS that the image read next from the folder will be the next frame in the sequence. Hence we need to explicitly find out the order of the images in sequence.

1. Read the image names from the folder one by one. The exact details are OS dependent. The code uses native system calls of the linux environment hence will only work in linux. Store them in a vector of image names.
2. Sort the image names according to their position as determined in their names. This can be done in linear time since all images are continous in the range 1 to n(no of frames) using direct addressing to store it in its correct place.

3. Once we have the sequence of images to read create a *VideoWriter* instance providing the compression coder to be used and the framerate as given by the user. Then use *VideoWriter :: write*() to store each frame into the video.

C++ Code
https://github.com/snehashispal1995/cv_assign_0/blob/master/Q1/code2.cpp



```cpp
int main(int s,char *argv[]) {

    if(s != 4) printf("Format is combinevideo [OUTPUT] [SOURCE FOLDER] [FRAMERATE] \n"
        "Images in folder should be readable in scanf format [name]_[frame_no].[type]"
        "\n It is assumed that frame nos start from 0 and there are no missing frames\n");
    //get the list of files in the folder
    vector<string> file_list;
    DIR *dir;
    struct dirent *dir_entry;

    if((dir = opendir(argv[2])) != NULL) {
        while((dir_entry = readdir(dir)) != NULL) {
            if(dir_entry->d_name[0] != '.')
                file_list.push_back(string(dir_entry->d_name));
        }
        closedir(dir);
    }
    else {
        printf("Error opening folder ");
        return -1;
    }

    //sort the files according to frame number in another vector
    vector<string> sorted_frames(file_list.size());
    for(auto i = file_list.begin(); i != file_list.end(); i++) {
        int a = (*i).find("_"), b = (*i).find(".");
        int fno = atoi(((*i).substr(a + 1, b - a - 1)).c_str());
        sorted_frames[fno] = *i;
    }

    //combine the frames into a video
    char c_dir[512];
    getcwd(c_dir,512);
    chdir(argv[2]);
    auto i = sorted_frames.begin();
    Mat frame = imread((*i).c_str(),1);
    VideoWriter output_video(string(c_dir) + string("/")  + string(argv[1]),
        VideoWriter::fourcc('M','J','P','G'),(double)atoi(argv[3]),frame.size(),true);
    if(!output_video.isOpened()) {
        printf("Error creating video output file \n");
        return -1;
    }
    output_video.write(frame);
    while( ++i != sorted_frames.end()) {
        frame = imread((*i).c_str(),1);
        output_video.write(frame);
    }
    return 0;

}
```

To run the program use the format : *combinevideo* [OUTPUT] [SOURCE FOLDER] [FRAMERATE]
If compiled locally replace the name with whatever output binary was created.

II. Capturing Images

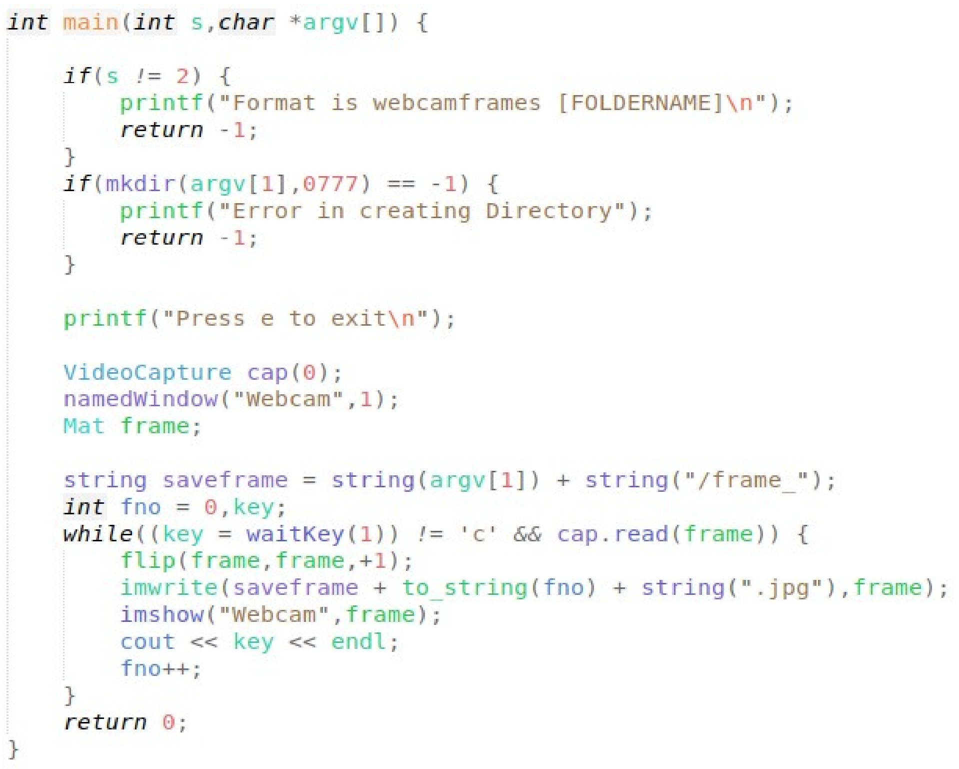
The objective is to capture and store a video frame by frame. The *VideoWriter* class also has mechanisms to directly read image frames from devices such as webcams and thus, provides an easy way to acquire live image data.

The procedure applied is

1. Read image frames from the *VideoWriter* instance with a delay of 1ms using the *waitKey*() function. Setting the *waitKey* value to 1 will allow the camera to read at the maximum possible refresh rate.
2. The resulting image is laterally inverted so we need to filp it horizontally using the *flip*() function.
3. The flipped image can now be saved in the folder specified by the user.

C++ Code

https://github.com/snehashispal1995/cv_assign_0/blob/master/Q2/code.cpp



```
int main(int s,char *argv[]) {

    if(s != 2) {
        printf("Format is webcamframes [FOLDERNAME]\n");
        return -1;
    }
    if(mkdir(argv[1],0777) == -1) {
        printf("Error in creating Directory");
        return -1;
    }

    printf("Press e to exit\n");

    VideoCapture cap(0);
    namedWindow("Webcam",1);
    Mat frame;

    string saveframe = string(argv[1]) + string("/frame_");
    int fno = 0,key;
    while((key = waitKey(1)) != 'c' && cap.read(frame)) {
        flip(frame,frame,+1);
        imwrite(saveframe + to_string(fno) + string(".jpg"),frame);
        imshow("Webcam",frame);
        cout << key << endl;
        fno++;
    }
    return 0;
}
```

Run code as *webcamframes* [STORE DIRECTORY] or by replacing with the name of the binary created locally.

III. Chroma Keying

This task involves creating a *matte* separating a background from the freground of an image. This process of separating a background from a composite image where the background is any arbitary amalgamation of objects is a difficult process. However the process can be simplified using a technique called chroma keying. In this process the background of a video consists of a singular color (usually green or blue) theoretically. The foreground is kept so that no part of the background color is present in the foreground objects. The task then is to seperate the green or blue background from the foreground objects which are ideally of different color. We can then alpha blend a background image/video in the locations where the background pixels occur to get a composite video.

The background is seperated by comparing each pixel in the image with a pixel value given by the user. Pixel values which are close to the user given value need to be removed from the foreground video and replaced with the corresponding pixel form the background video. Since the pixel value to be matched must be one of the pixels of the image we take the input from the user accordingly.

1. Create a *namedWindow*() instance and attach a mouse callback to it which determines the point in the window at which the user clicked.
2. Show the first frame of the foreground video (initially) in the *namedWindow and* wait for the user to click on the image.
3. Determine the pixel value in the corresponding point at which the user clicked. This is our chroma keying color value.
4. Keep an option for the user to go to the next frame of the video.

With the pixel value determined we now need to match which parts of the image have pixel values identical or close to the chroma keying color value. We let the user determine just how close the pixel need to be for it to be completely replaced by a background pixel through a command line parameter.

Firstly we must find a way of comparing two different pixel values. Normal video frames are in a 3 channel RGB format. Naively we can think of comparing each component of every pixel with the corresponding component of the choroma key pixel and find out how close the pixel is in the color space. For a neighbourhood of pixel values in the color space we may consider a range for each component within which each pixel should lie for it to be replaced by the background pixel. In practice using this method tended to eliminate unwanted pixel values also as keeping the neighbourhood large matched pixel where the non – dominant color component fell into this range. If the neighbourhood was kept low and all the background pixels was not of uniform color it tended to leavo out some pixels pirticularly at the edges of the foreground objects. This also limits any transparency as we are making binary decisions to either include or exclude a pixel. Therfore we need to find a way comparing pixel values based on a single value, calculated by taking into consideration all the three channel. Fortunately the HSV format (Hue, Separation, Value) with its Hue component gives us just that.

We can manually convert each pixel in RGB to HSV using the equations. Initally the RGB components are in [0,1] range.

$$V \leftarrow \max(R, G, B)$$

$$S \leftarrow \begin{cases} \frac{V-\min(R,G,B)}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$H \leftarrow \begin{cases} 60(G-B)/(V-\min(R,G,B)) & \text{if } V = R \\ 120 + 60(B-R)/(V-\min(R,G,B)) & \text{if } V = G \\ 240 + 60(R-G)/(V-\min(R,G,B)) & \text{if } V = B \end{cases}$$

If $H < 0$ then $H \leftarrow H + 360$. On output $0 \leq V \leq 1$, $0 \leq S \leq 1$, $0 \leq H \leq 360$.

OpenCV also has the *cvtColor* function for easy color conversion.

We can now compare between pixel values and assign transparency of foreground pixels using their Hue values. The formulae used to determine the *alpha* value or the transparency at each pixel of the foreground is..

$$\alpha = 1 - \frac{diff}{|hue_{(x,y)} - hue_{chroma}| + 1}$$

where *diff* is a user given value used to determine how the *alpha* value changes as the hue value of the pixel moves away from the choroma key hue value. If *alpha* lies outside the [0,1] range we truncate its value to either 0 or 1 depending on which side it lies. This means that *alpha* value of the pixel increases as the hue value of the pixel moves away from the hue value of the chroma color. We can use this as the transparency parameter and use it to blend the foreground and background using

$$P_{(i,j)} = \alpha * F_{(x,y)} + (1-\alpha) * B_{(x,y)}$$

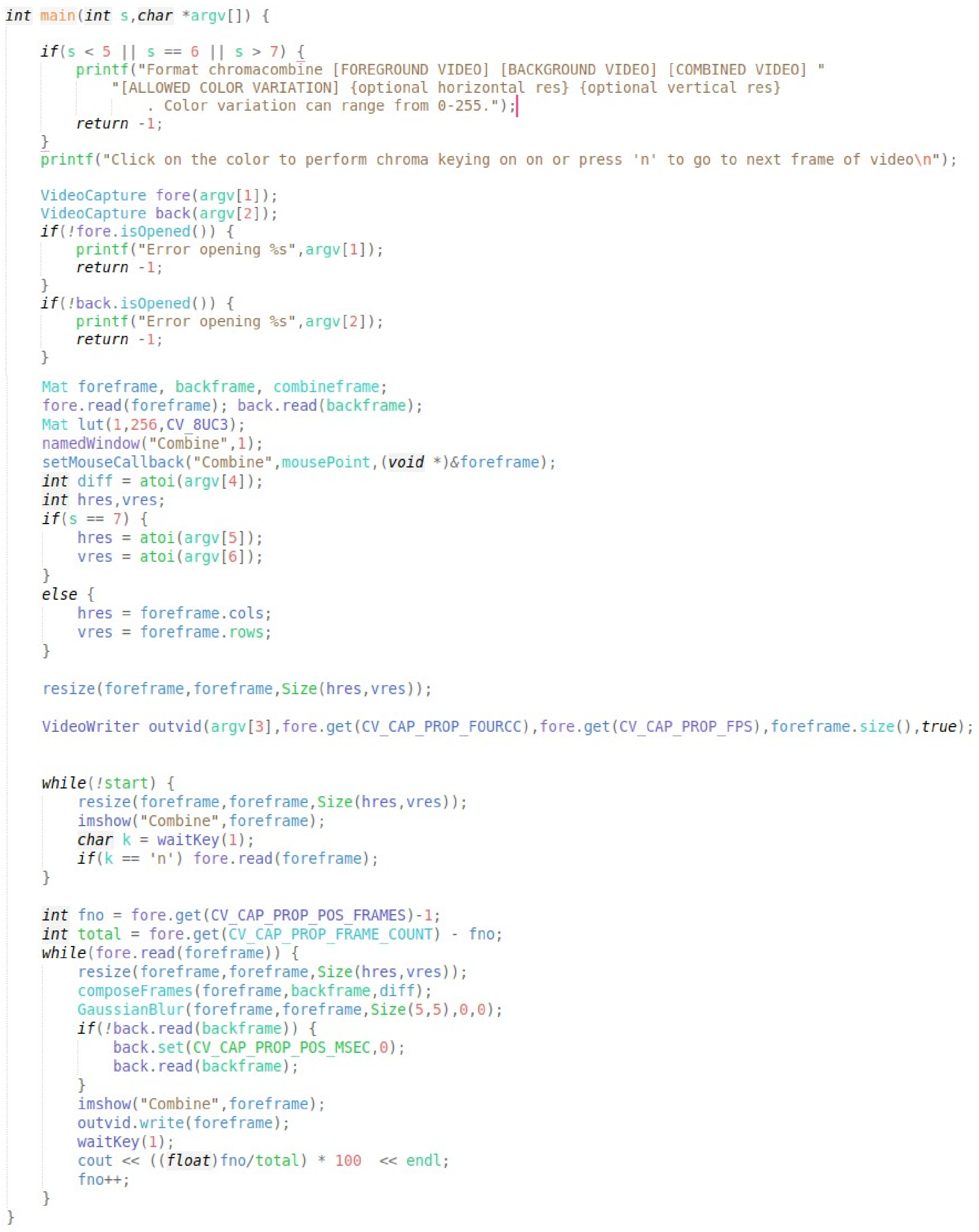
Thus is the alpha value decreases we get more of the background color and vice versa. To summarize the alpha blending procedure.

1. Convert the chroma key color given by the use to HSV format.
2. Resize the background image to be of the same dimensions os the foreground image.
3. For every pixel in the foreground image
    1. find the *alpha* value using the equation above.
    2. Combine the foreground and background pixel using *alpha* blending as described above.
    3. (Optional) Use a gaussian blurring (we have used one with a kernel of 5x5 pixels) to smooth out the edges of the video.
    4. Store the combined frame using a *VideoWriter* instance to
4. Do steps 2 – 3 for every frame in the foreground video.

C++ Code

https://github.com/snehashispal1995/cv_assign_0/blob/master/Q3/code.cpp

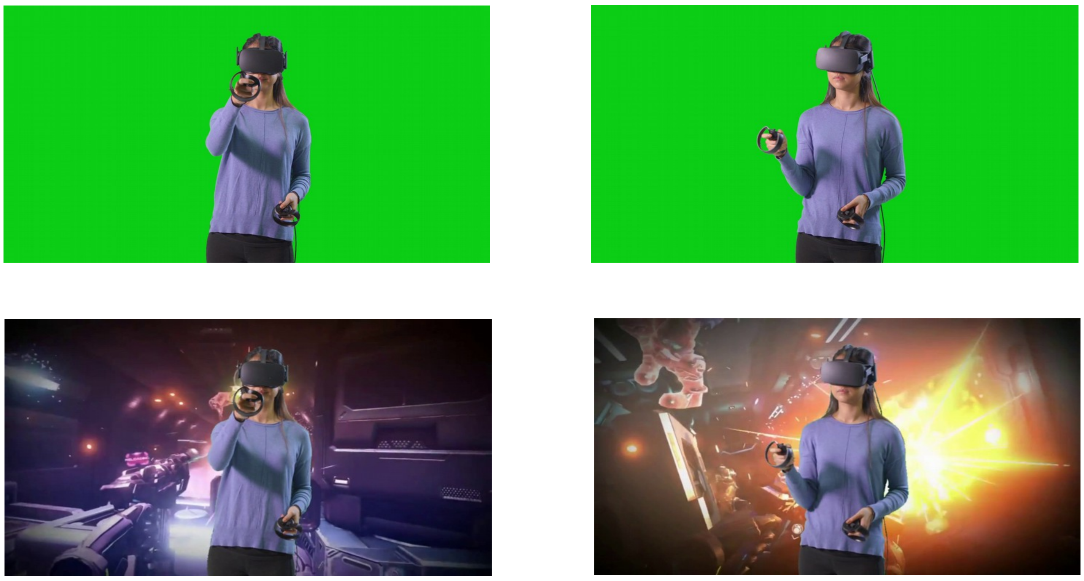
```cpp
int start;
Vec3b Color;

//function to determine the color or the pixel to be used for chroma keying
void mousePoint(int event, int x, int y, int flags, void *data) {

    Mat *frame = (Mat *)data;
    if(event == EVENT_LBUTTONUP) {
        Mat3b c(frame->at<Vec3b>(y,x)), col;
        cout << c << endl;
        if(!start) {
            cvtColor(c,col,COLOR_BGR2HSV);
            Color = col.at<Vec3b>(0,0);
        }
        start = 1;
    }
    return;
}

//combine the foreground and background frames
void composeFrames(Mat &fore, Mat &back, int diff) {

    Mat resback,ret;
    resize(back,resback,fore.size());

    for(int i = 0; i < fore.rows; i++) {
        for(int j = 0; j < fore.cols; j++) {
            Mat3b c(fore.at<Vec3b>(i,j)), hsvc;
            Vec3b bc = resback.at<Vec3b>(i,j);
            cvtColor(c,hsvc,COLOR_BGR2HSV);
            Vec3b fc = hsvc.at<Vec3b>(0,0);
            double alpha = 1;

            alpha = 1 - (HUE_W * diff/(abs(fc[0] - Color[0]) + 1) +
                SAT_W * diff/(abs(fc[1] - Color[1]) + 1) +
                VAL_W * diff/(abs(fc[2] - Color[1]) + 1));
            if(alpha > 1) alpha  = 1;
            else if(alpha < 0) alpha = 0;
            fore.at<Vec3b>(i,j)[0] = alpha * fore.at<Vec3b>(i,j)[0] + (1 - alpha) *
                resback.at<Vec3b>(i,j)[0];
            fore.at<Vec3b>(i,j)[1] = alpha * fore.at<Vec3b>(i,j)[1] + (1 - alpha) *
                resback.at<Vec3b>(i,j)[1];
            fore.at<Vec3b>(i,j)[2] = alpha * fore.at<Vec3b>(i,j)[2] + (1 - alpha) *
                resback.at<Vec3b>(i,j)[2];
        }
    }

}
```



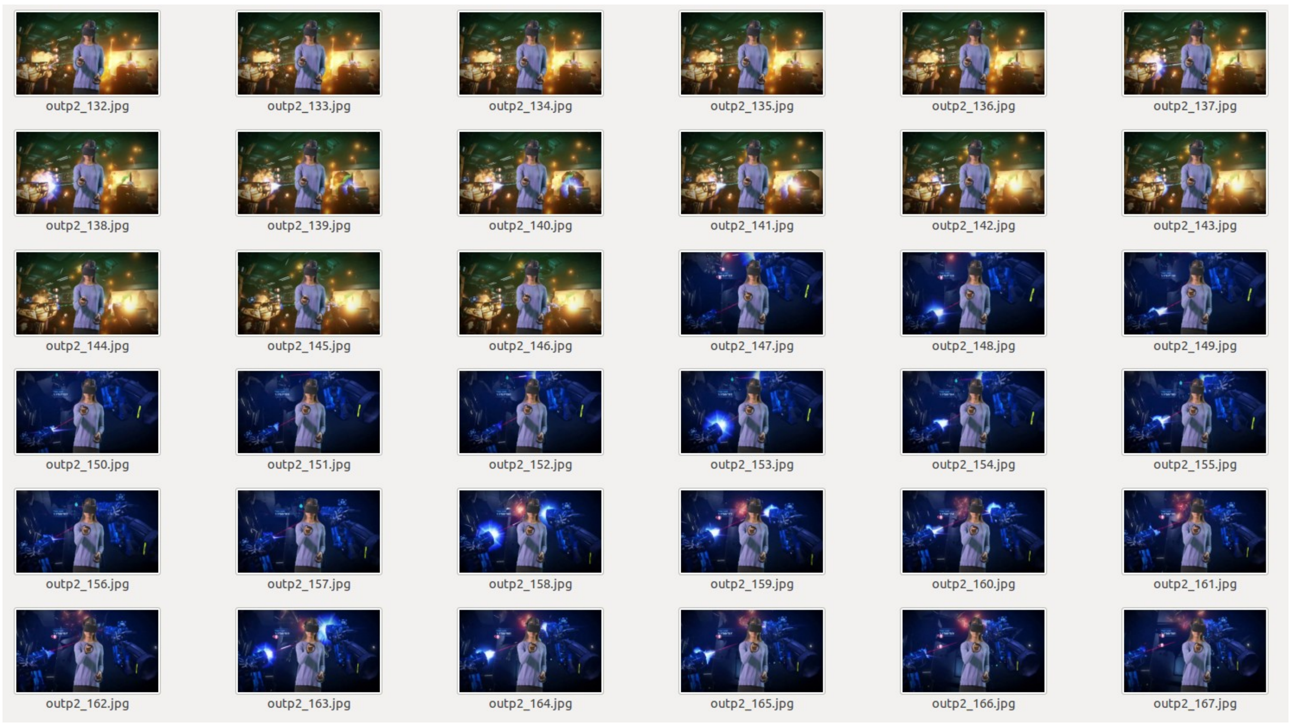
```
int main(int s,char *argv[]) {

    if(s < 5 || s == 6 || s > 7) {
        printf("Format chromacombine [FOREGROUND VIDEO] [BACKGROUND VIDEO] [COMBINED VIDEO] "
            "[ALLOWED COLOR VARIATION] {optional horizontal res} {optional vertical res}
                . Color variation can range from 0-255.");
        return -1;
    }
    printf("Click on the color to perform chroma keying on on or press 'n' to go to next frame of video\n");

    VideoCapture fore(argv[1]);
    VideoCapture back(argv[2]);
    if(!fore.isOpened()) {
        printf("Error opening %s",argv[1]);
        return -1;
    }
    if(!back.isOpened()) {
        printf("Error opening %s",argv[2]);
        return -1;
    }

    Mat foreframe, backframe, combineframe;
    fore.read(foreframe); back.read(backframe);
    Mat lut(1,256,CV_8UC3);
    namedWindow("Combine",1);
    setMouseCallback("Combine",mousePoint,(void *)&foreframe);
    int diff = atoi(argv[4]);
    int hres,vres;
    if(s == 7) {
        hres = atoi(argv[5]);
        vres = atoi(argv[6]);
    }
    else {
        hres = foreframe.cols;
        vres = foreframe.rows;
    }

    resize(foreframe,foreframe,Size(hres,vres));

    VideoWriter outvid(argv[3],fore.get(CV_CAP_PROP_FOURCC),fore.get(CV_CAP_PROP_FPS),foreframe.size(),true);


    while(!start) {
        resize(foreframe,foreframe,Size(hres,vres));
        imshow("Combine",foreframe);
        char k = waitKey(1);
        if(k == 'n') fore.read(foreframe);
    }

    int fno = fore.get(CV_CAP_PROP_POS_FRAMES)-1;
    int total = fore.get(CV_CAP_PROP_FRAME_COUNT) - fno;
    while(fore.read(foreframe)) {
        resize(foreframe,foreframe,Size(hres,vres));
        composeFrames(foreframe,backframe,diff);
        GaussianBlur(foreframe,foreframe,Size(5,5),0,0);
        if(!back.read(backframe)) {
            back.set(CV_CAP_PROP_POS_MSEC,0);
            back.read(backframe);
        }
        imshow("Combine",foreframe);
        outvid.write(foreframe);
        waitKey(1);
        cout << ((float)fno/total) * 100  << endl;
        fno++;
    }
}
```

Format : *chromekey* [FOREGROUND VIDEO] [BACKGROUND VIDEO] [COMBINED VIDEO] [DIFF VALUE] {OPTIONAL RENDER [HORIZONTAL] AND [VERTICAL] RESOLUTION}

Example



Chroma keying with green screen.



A set of frames where the background was replaced with another video.