Nixon Puertollano, Christopher Kruger, Christopher Youngclaus

Professor Man

CPE 462

14 May 2022

Final Project Report - Virtual Painter

Abstract

Augmented reality is an upcoming technology that utilizes what is seen by a computer to add functionality, such as superimposing colors and images onto the live feed of video. To learn more about this technology by using OpenCV, the group has created a program that allows the user to utilize their camera as a canvas and allows different household objects to act as a paint brush. By holding up objects of unique colors (in our case being orange, green, or red), the program will take the RGB values to draw over the webcam in the same colors, tracing over the tip of the object. The project worked as expected. By holding up for example an orange sharpie (over a non orange background), the tip of the sharpie will begin to draw a solid orange line. The incorporation of OpenCV has successfully allowed the group to create a superficial drawing application over any background within the web camera.

Introduction

The goal of this project is to utilize OpenCV libraries to allow a user to draw over the webcam using an object of their choice. This utilization has been used previously in many different programs, often used as an augmented reality function. This new program can be fine tuned to a user's specific needs, where it can then be used as an artistic expression. The user may draw with the built in three colors (or add more as desired) to paint as they please. Another usage

of the application can be used within the educational workforce, where a real time image displayed with the camera can be drawn over to teach a topic.

Method

Using Visual Studio Code and OpenCV on a C++ file, the group was able to detect and identify color through use of a camera by the end of this project, in addition to learning how to draw contour lines around objects that clearly separate them from the background of the image. Using these two main features, the group was able to create a program that identified where objects were in the scene, detected what color they were, and displayed the correct color of the object to the user on-screen in the top-right corner of the GUI. The code currently runs three sample colors to prevent overloading the program with an image containing many different colors: red, orange, and light green. According to Figures 3 and 4, if the user wants to examine different colors, the user should add their ColorPickerProgram values to the myColors vector, the corresponding bgr value to the myColorValues vector, and also the name of the color to the myColorNames vector.

The idea and framework for the project was found in a video linked at the end of this report; the one other file that doesn't help our program run, rather helps the user to use our program correctly, is the ColorPickerProgram file that the group designed to help fine tune the user's camera to identify one color.[1] This file shows several sliders that adjust image settings like hues, in order to find the right settings for their particular camera (See Figure 1). To gain initial functionality in the core program, the group wrote a function that finds objects that are on camera and makes lines around the general shape of the object mainly using the findCountours function that is included in OpenCV imports. The group initially had the program draw several

[1] Murtaza's Workshop Video: https://www.youtube.com/watch?v=2FYm3GOonhk

purple lines around the object, but soon realized that green is a much better contrasting color (in most cases) and also that a simple rectangle around the object alleviates a lot of the stress on both the program and the user, as exemplified in Figure 6. Then, the findColor function uses findContours by telling it what color to look for in the first place by initializing a mask for each color that is defined by the user in the myColors vector (these colors are found using the aforementioned ColorPickerProgram).

In the main function, displayed in Figure 8, after detecting the video capture device that the user is using, the program will loop through reading the image, identifying where the color is on the camera, printing what user-chosen colors are currently on the screen in the top-right corner, and forwarding the video image being sent to the code back to the user with the previously mentioned adjustments. The newPoints variable calls back to findColor, then the detectColorText (See Figure 7) uses newPoints and myColorValues to display a dot on the top of the rectangle encompassing the object that is the same color as that object, which will also produce the name of the color in the top-right of the image. This is done by running detectColorText immediately after defining newPoints. imshow will reproduce the edited image with the rectangle, colored dot, and color name. The color name will refresh every time the program refreshes, but the colored dots stay on screen and stack on top of one another to show the path that the image processor is following along with the path of the object on-screen.

Compile/Execution

The first step in compiling our code is to have OpenCV correctly installed and functional. Following this, please run the ColorPickerProgram.cpp file to detect the HSV (Hue Saturation Value) of an object you would like to act as a paint brush (**Figure 1**). Using the sliders in the

GUI, allow only the color you want to be used as a paintbrush to be shown as white with the rest being black. For reference, **Figure 2** shows a depiction of this process. The left window is the raw input feed of the camera, the middle window is the HSV adjusted windows to only "see" the orange, and the right window is the GUI. By finding these values, please edit the values associated with each variable (**Figure 3**). The user will also have to manually select the colors that are to be drawn with by incorporating the RGB values (**Figure 4**). Once these values are incorporated within the program to match the user's object and webcam, the program will begin to draw in the corner of the counter being attached to the object (**Figure 5**). Note the drawing function and the function that shows the color being detected.

Figures

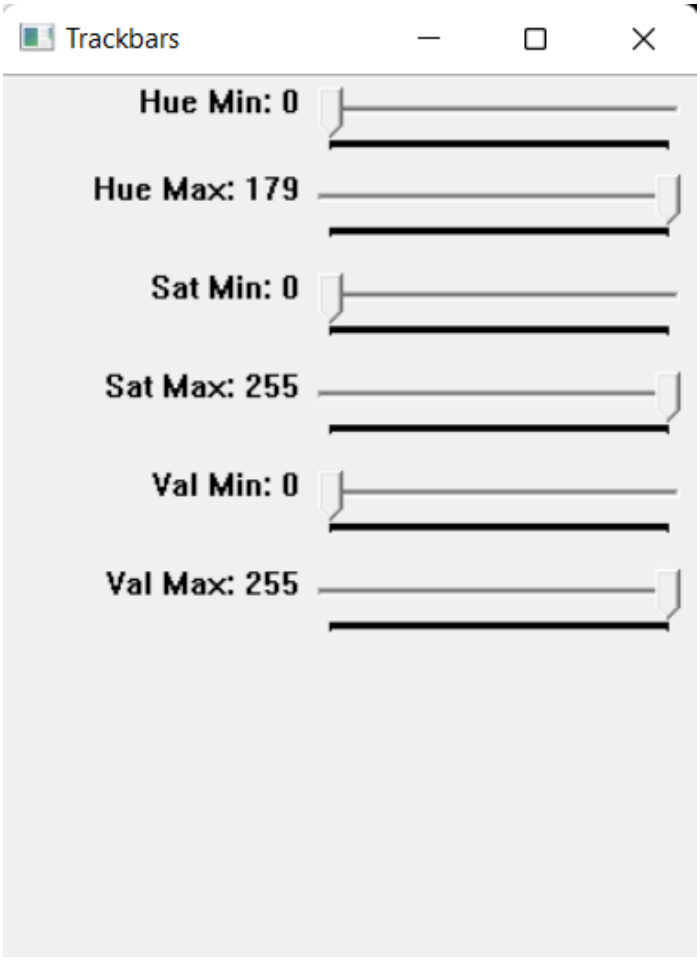

Figure 1: ColorPickerProgram

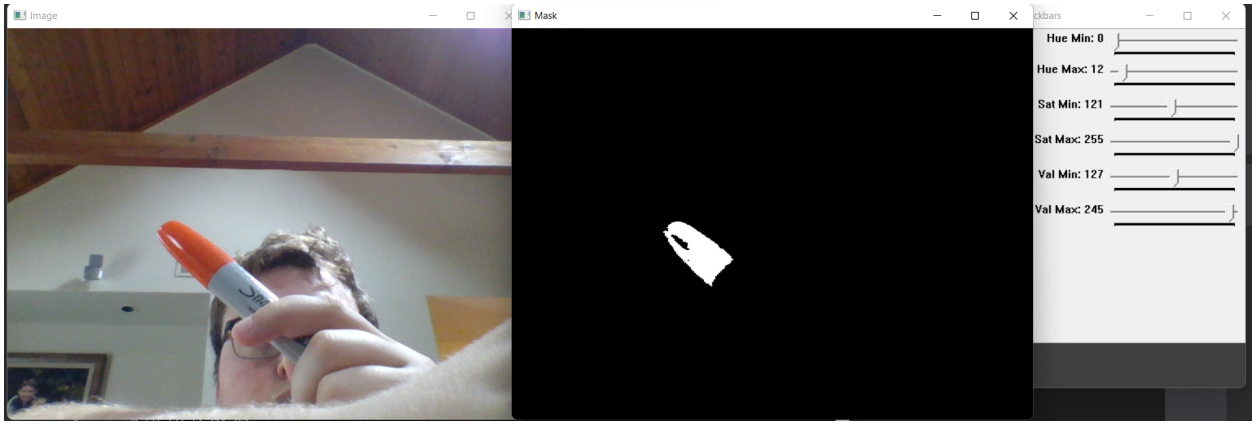

Figure 2: Running ColorPickerProgram to Identify Orange

```
// hmin, smin, vmin, hmax, smax, vmax, ypos(for text position)
vector<vector<int>> myColors{ {51,56,116,128,170,255,100},            // light green highlighter
                              {162,130,69,179,201,134,150},          // red colored pencil
                              {0,94,154,13,255,255,200}                  // orange colored pencil
};
```

Figure 3: ColorPickerProgram Values

```
vector<Scalar> myColorValues{ {0, 255, 0},
                              {0, 0, 255},
                                  {0, 165, 255}
};

vector<string> myColorNames{ "Green", "Red", "Orange" };
```

Figure 4: Bgr Values and Color Names

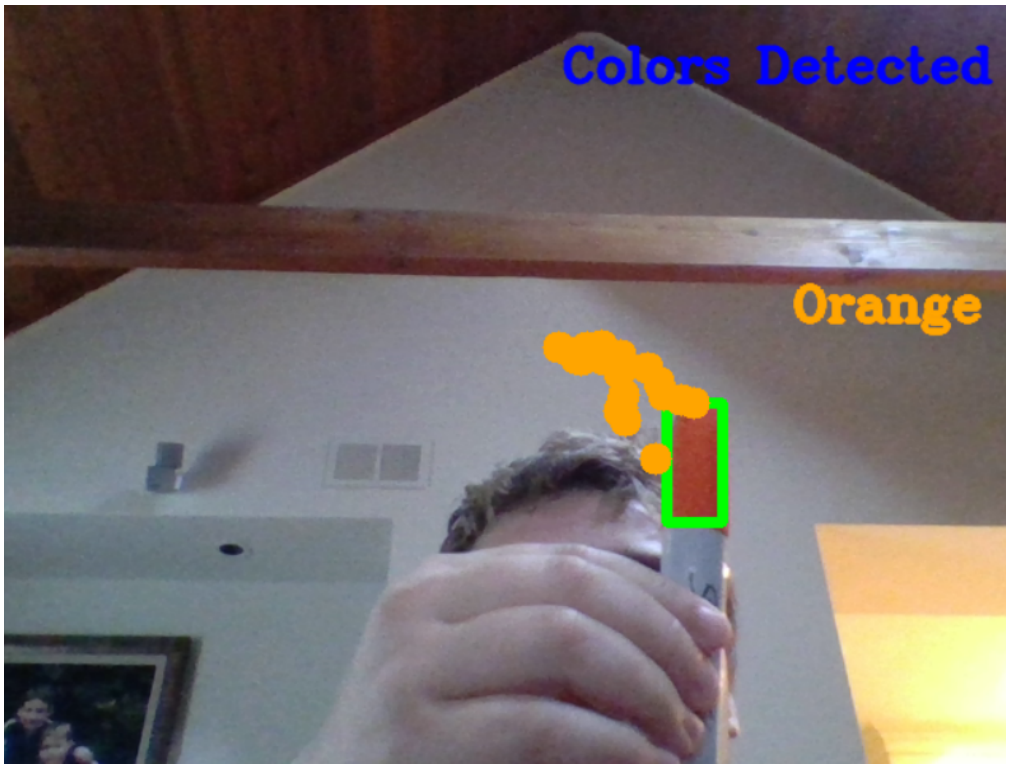

Figure 5: Fully Compiled Program

```
if (area > 1000)
{
    float peri = arcLength(contours[i], true);
    approxPolyDP(contours[i], conPoly[i], 0.02 * peri, true);

    cout << conPoly[i].size() << endl;
    boundRect[i] = boundingRect(conPoly[i]);
    myPoint.x = boundRect[i].x + boundRect[i].width / 2;
    myPoint.y = boundRect[i].y;

    // drawContours(img, conPoly, i, Scalar(255, 0, 255), 2);                    // draw purple contours around object
    rectangle(img, boundRect[i].tl(), boundRect[i].br(), Scalar(0, 255, 0), 5);  // draw green rectangle around object

}
```

Figure 6: Contour Visualizer

```
void detectColorText(vector<vector<int>> newPoints, vector<Scalar> myColorValues)
{
    putText(img, "Colors Detected", Point(355, 50), FONT_HERSHEY_COMPLEX, 1, (255, 0, 255), 2, LINE_AA);
    for (int i = 0; i < newPoints.size(); i++)
    {
        putText(img, myColorNames[newPoints[i][2]], Point(500, newPoints[i][3]), FONT_HERSHEY_COMPLEX, 1, myColorValues[newPoints[i][2]], 2, LINE_AA);
    }
}
```

Figure 7: DetectColorText Function

```
void main()
{
    VideoCapture cap(0);
    while (true) {
        cap.read(img);
        newPoints = findColor(img);
        detectColorText(newPoints, myColorValues);

        imshow("Image", img);
        waitKey(1);
    }
}
```

Figure 8: Main Function

Conclusion

The group has successfully utilized the OpenCV library to learn more about not just augmented reality, but also how computer vision works. With the new experience in computer vision, members of the group can now incorporate our knowledge into future computer vision aided projects as well as in our future careers. The group has also learned about color detection and how to incorporate GUI's into a program. Overall, the project was a success. Added functionality for the future can include an automatic color detection system, rather than having to manually edit it each time when changing the camera background or object to paint with.

Contributions

Nixon Puertollano, Christopher Kruger, and Christopher Youngclaus equally contributed to the base code from our source. Nixon Puertollano wrote the added detectColorText function. Christopher Kruger and Christopher Youngclaus wrote the majority of the report.

Sources

https://www.youtube.com/watch?v=2FYm3GOonhk