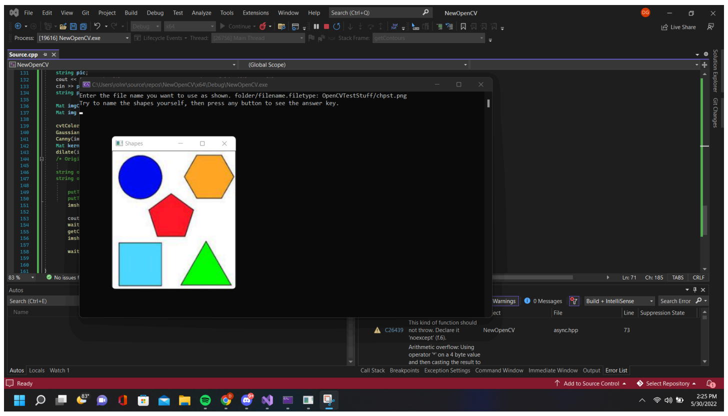
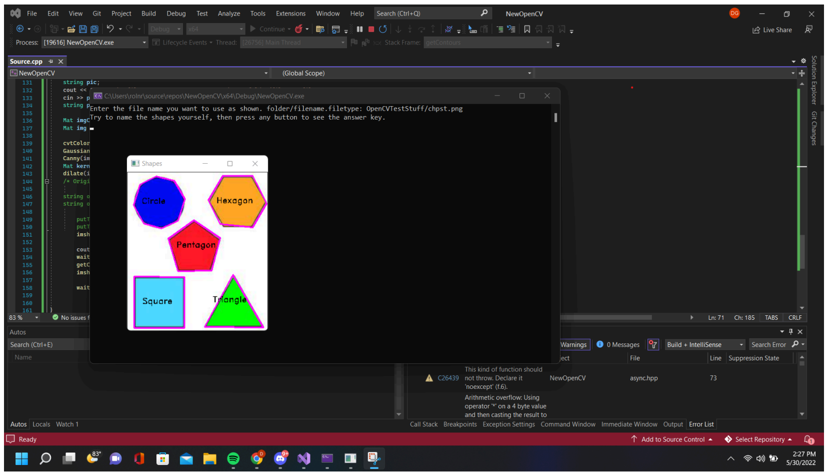
Daniel Goldberg
CPE462
Professor Man

Shape Identification Project

When Covid first came about, everyone was forced into quarantine, and while this hurt a lot of people's lifestyles, it gave parents more time to spend with their young children, and help them learn what they need to succeed in life. With said restrictions loosening, parents are going back to work, and these kids are receiving less attention and life skills. This project is intended to help young children learn their shapes, without the need for a teacher or parent to be present. Using OpenCV and Visual Studios c++, this program identifies the edges and vertices of the shapes presented and identifies them based on this. Any image can be used, but due to coding restrictions, the program is only able to identify circles, squares, rectangles, triangles, pentagons, and hexagons. The program is also capable of filtering out potential "noise" that may be included in these pictures as well.

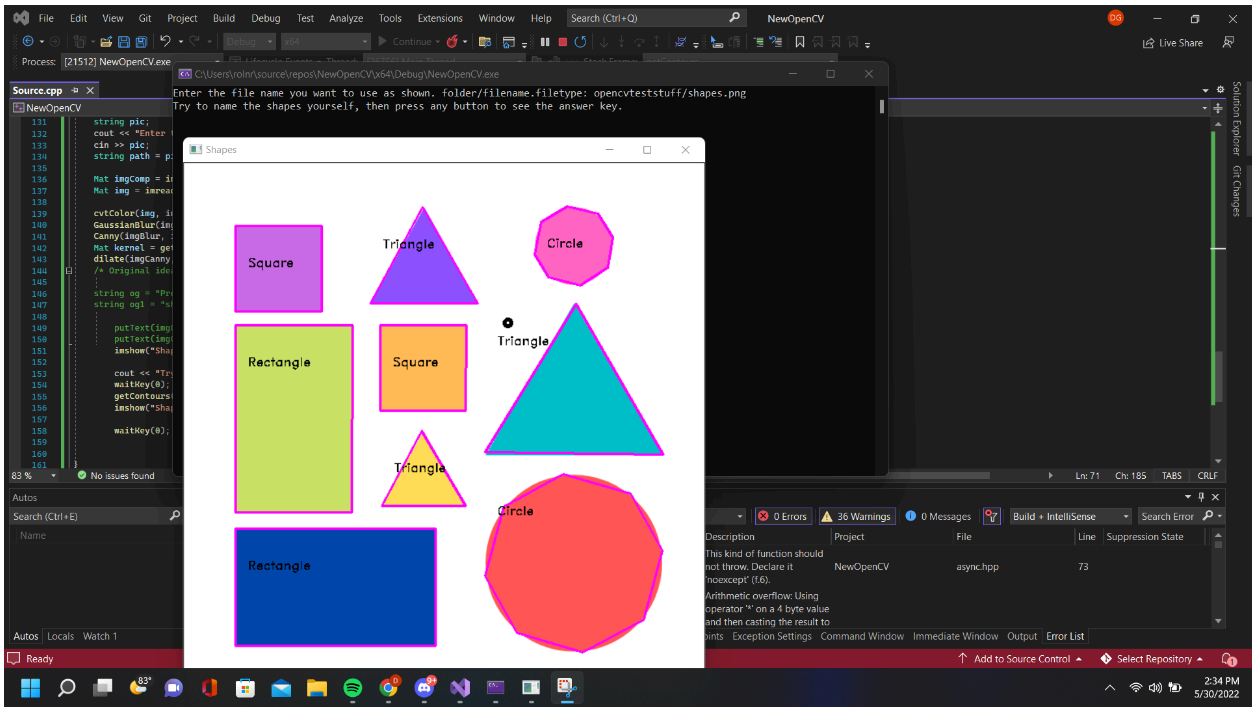
The program was originally intended to be interactable, in the sense that it would number the shapes in the image provided and ask for the user to type the name of the shape into the debug console, but it proved to be much more complicated than that. Due to a multitude of errors, it was decided that the program would present the original image, and wait for an input to then show the same image but with the names on top. The program works flawlessly with the aforementioned shapes, but if any shape that has more than 6 sides is used, it will be identified as a circle, as that is how the program was coded. The contour function that was created maps line segments along the sides of the shapes presented, but in the case of a circle, it will do its best with the segments, mapping out 8 sides despite the circle being one continuous curve, so, for this reason, it would identify an octagon (8 sided figure) the same as a circle.


```
C:\Users\rolnr\source\repos\NewOpenCV\x64\Debug\NewOpenCV.exe
Enter the file name you want to use as shown. folder/filename.filetype: OpenCVTestStuff/chpst.png
Try to name the shapes yourself, then press any button to see the answer key.
```


```
C:\Users\rolnr\source\repos\NewOpenCV\x64\Debug\NewOpenCV.exe
Enter the file name you want to use as shown. folder/filename.filetype: OpenCVTestStuff/chpst.png
Try to name the shapes yourself, then press any button to see the answer key.
```

Some images that the user may want to provide to the program may have some imperfections, such as pencil markings, unusual blots from scanning, etc. The program is created in such a way that it will filter out most if not all of these small imperfections, or “noise”, and only focus on the larger shapes presented.

Of all the basic shapes, squares and rectangles are among some of the first that are taught. It’s easy enough to describe to a person the difference between these shapes, a square’s sides are all even while a rectangle has two sets of sides that share different lengths. The code however didn’t originally recognize these two as different, due to the fact that it uses the number of vertices in the shape to identify it. In order to combat this, an “if” statement was used that compares the height and width of the identified shape, and if the two are the same (or close enough to a point where the human eye wouldn’t notice), it is labeled a square. Otherwise, the shape will be labeled as a rectangle.



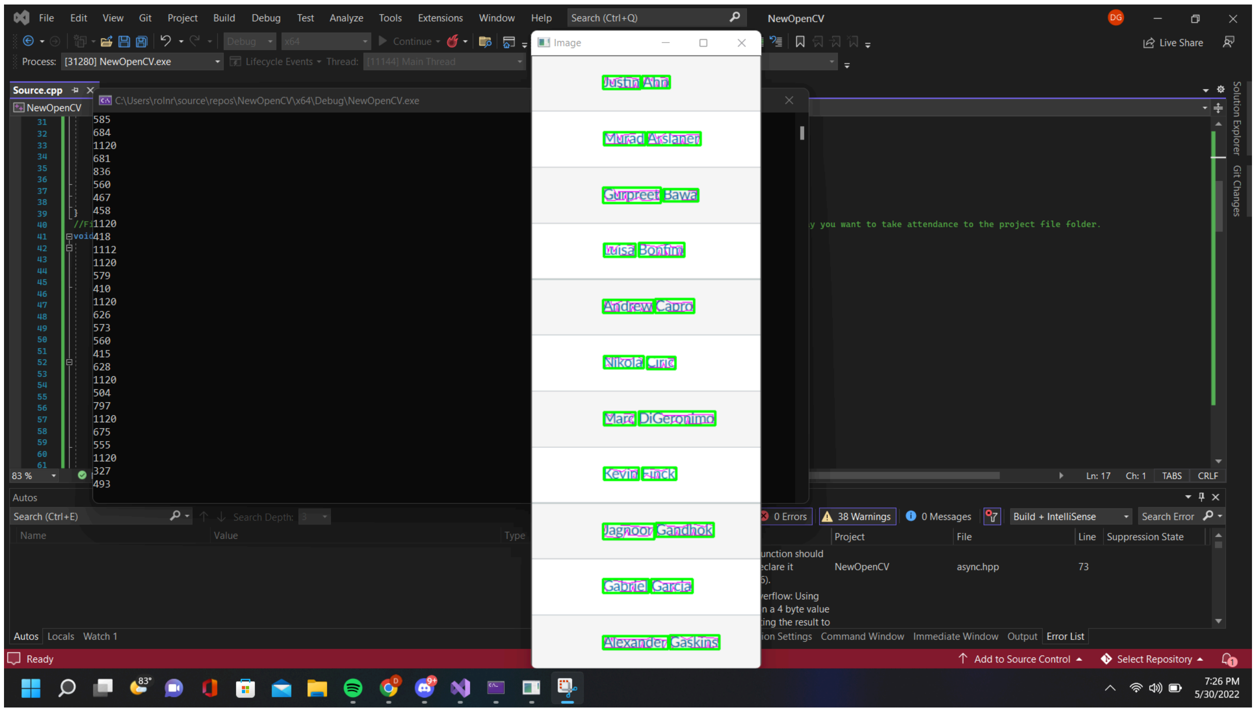
```
C:\Users\rolnr\source\repos\NewOpenCV\x64\Debug\NewOpenCV.exe
Enter the file name you want to use as shown. folder/filename.filetype: opencvteststuff/shapes.png
Try to name the shapes yourself, then press any button to see the answer key.
```

```
    string pic;
    cout << "Enter
    cin >> pic;
    string path = p

    Mat imgComp = i
    Mat img = imrea

    cvtColor(img, i
    GaussianBlur(im
    Canny(imgBlur,
    Mat kernel = ge
    dilate(imgCanny
    /* Original ide

    string og = "Pr
    string og1 = "s

        putText(img
        putText(img
        imshow("Sha

        cout << "Tr
        waitKey(0);
        getContours
        imshow("Sha

        waitKey(0);

}
```

```
0 Errors    36 Warnings    0 Messages    Build + IntelliSense    Search Error
Description                                Project       File         Line   Suppression State
This kind of function should               NewOpenCV     async.hpp    73
not throw. Declare it
'noexcept' (f.6).
Arithmetic overflow: Using
operator '*' on a 4 byte value
and then casting the result to
Exception Settings   Command Window   Immediate Window   Output   Error List
Ln: 71   Ch: 185   TABS   CRLF
2:34 PM
5/30/2022
```

In order to successfully compile and run this code, one really only needs the basics. An installation of opencv for c++ is required, and visual studios c++. The only extra thing needed would be file(s) containing images of shapes, although the files in the images above will be provided along with the source code. Any image can be used and should work as stated, though the ones provided have proven to work flawlessly. Once the files are gathered, insert them into a folder (it is recommended to keep the file and folder names short, as these will need to be typed into the debug console to pull them up) and move this folder into the project folder. From this point, all that needs to be done is to run the code, when prompted enter the folder name and file name like so (foldername/filename.filetype (it is not case sensitive)) and the project will pull up the specified image. The debug console gives instructions on how to navigate the project, once the original image is pulled up simply press any button and the original image will reappear, with the proper names of the shapes presented on top of the image.

My original plan for this project was to create a program that would take provided screenshots of the class list and the zoom participant list for the day, and automatically mark whether the names on the participant list match those in the class roster, thus filling in the attendance for the day. I had gotten the idea from a LinkedIn post I found of a student doing something similar in Python. The original code for my attempt is still in the shapes project at the top, but it is all commented out as to prevent it from making my new project fail to compile. I had gotten it to a point where it would identify the text provided in a screenshot from the class roster on canvas and surround each first and last name with a bounding box. It would then provide the area within each box that surrounds these names in the debug console. I had originally run into an issue where the detection system would detect the lines between the names as well, but created a workaround similar to the noise exclusion used in the shapes program, utilizing an "if" statement, and made a case where it would ignore the areas depicted in the debug console that represented these lines.



```
585
684
1120
681
836
560
467
458
1120
418
1112
1120
579
410
1120
626
573
560
415
628
1120
504
797
1120
675
555
1120
327
493
```

Justin Ahn
Murad Arslaner
Gurpreet Bawa
Luisa Bonfini
Andrew Capro
Nikola Cirić
Marc DiGeronimo
Kevin Finck
Jagnoor Gandhok
Gabriel Garcia
Alexander Gaskins

My original idea was that if the areas representing the first and last name of the students in the image provided matched the areas of the names in the zoom participant list screenshot list, it would mark that specific case as a student who is present. Amongst many other things overlooked, the main thing that caused me to scrap this project was that the zoom participant list doesn't use the same font or size as the student list on canvas, so there would be two different values for the same name, which would render my program useless. In my search I had discovered a potential workaround using a program for OpenCV called Tesseract, which is labeled a text detection software. I was able to find plenty of things online that explained how to implement and use this text recognition engine using Python, but unfortunately there was (after searching the internet to my best ability) little to no online assistance for C++, so I had tried to use my own workaround to detect and match text to one another.

I was able to scrap some of the code used for this original project and repurpose it for the shapes project. While the shapes project is fairly simple, there was a lot of learning from very online sources that went into the making of it, which opened me up to some of the other capabilities of the software we have been using in this class, which was very interesting to explore.