Compsci 373 Assignment extension report

Motivation

The python code I wrote for the main part of this assignment completes its objective, but it does it very slowly, taking 10 - 15 seconds on the images provided. In the real world this sort of performance would be unacceptable.
Part of the reason the process takes so long is that I wanted to implement a relatively robust detection algorithm, which does more processing to ensure the qr code is detected. This combined with the fact that python isn't particularly fast results in the long delay.
Despite the attempts to make the detection robust enough to work for all of the challenging images, it doesn't properly detect one of the codes (playground.png).

Goal

I concluded that taking the opposite approach, a less robust but much faster method, may yield better results.
From the lectures I recalled openCV being used to perform contrast stretching on a video input in real time, so I decided to use it in the hopes of real time qr code detection.

Approach

OpenCV includes functions to perform the tasks I implemented manually in python, however they are written in c++ which means they will run orders of magnitude quicker than my implementation.
The initial process of manipulating the input is almost identical the one I implemented / the one suggested by the assignment:

```
kernel = np.ones((5, 5), "uint8")

# convert to grayscale
grey = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# slight blur to remove fine noise
blur = cv2.blur(grey, (3, 3))
# canny edge detection
edges = cv2.Canny(blur, 40, 140)
# blur edges
edges_blurred = cv2.blur(edges, (3, 3))
# threshold
_, threshold = cv2.threshold(edges_blurred, 20, 255, cv2.THRESH_BINARY)
# dilate
dilated = cv2.dilate(threshold, kernel)
```

The main differences between this and my method is that a slight blur is applied before edge detection (due to noise from my camera), and that no erosion step is performed, as it didn't seem necessary.

OpenCV also provides methods for creating bounding boxes around detected contours, the function `cv2.minAreaRect(contour)` finds the smallest rotated rectangle around the given shape contours. The main drawback of this is that it will always provide a rectangular

shape, so if the qr code is perspective warped, the rectangle becomes inaccurate (This doesn't occur in my implementation of the bounding rectangle creation).
This rectangle is then drawn on the input frame and displayed.

Results

The detection runs in (slightly faster than) real time, with considerable accuracy. While it is not as robust as the pure python version, it will easily detect a qr code if the camera is pointed at it in most cases.
The bounding box becomes inaccurate if the qr code is perspective warped, as mentioned above. I did not have time to try and fix this as I am still pretty new to openCV.

For a demonstration of the code working see qrExample.mp4 in the project folder

The bounding box handles rotated codes well

But not perspective warped

Conclusion

In the real world, people usually try to make it easy for the device they are using to detect a qr code, in almost all cases, they will hold their phone up the qr code so that it fills the frame and is easy to detect. The challenging scenarios (i.e playground.png) are unlikely edge cases, where the user would probably move their phone closer to get a better view of the code. I think speed is an important factor and chose to implement that with decent success.