CS5330 Project 4

Philip Mathieu

March 20, 2023

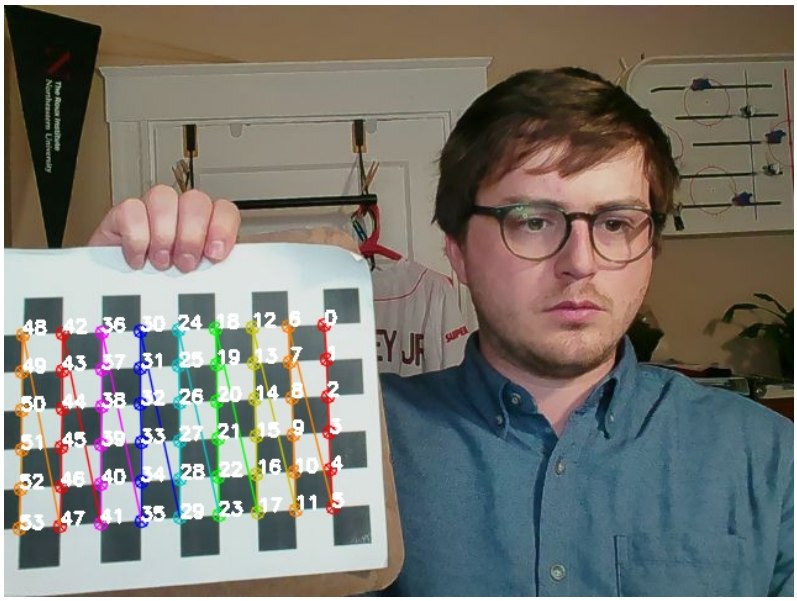
Project Description

This project demonstrates basic augmented reality functions using a chessboard marker. The first program finds chessboards in a live video feed and gives the viewer the option to add the view to a list for camera calibration. When sufficient images have been collected, it calculates the camera calibration matrix and distortion coefficients. The second program reads these calibrations and uses them to place 3D objects into the projected space. The third program finds and highlights Harris markers in an image. Last, as an extension, I rewrote the augmented reality visualization program in python, following the C++ code nearly line-for-line, and demonstrate that performance (as measured by frames per second) is similar under some circumstances but worse under others.

Required Images

Calibration Image

A calibration image with chessboard corners highlighted.



Calibration Matrices

Initial Camera Matrix:

```
[ 1.00   0.00   320.00 ]
[ 0.00   1.00   240.00 ]
[ 0.00   0.00   1.00 ]
```

Calibrated Camera Matrix:

```
[ 691.95   0.00   313.66 ]
[ 0.00   691.95   251.46 ]
[ 0.00   0.00   1.00 ]
```

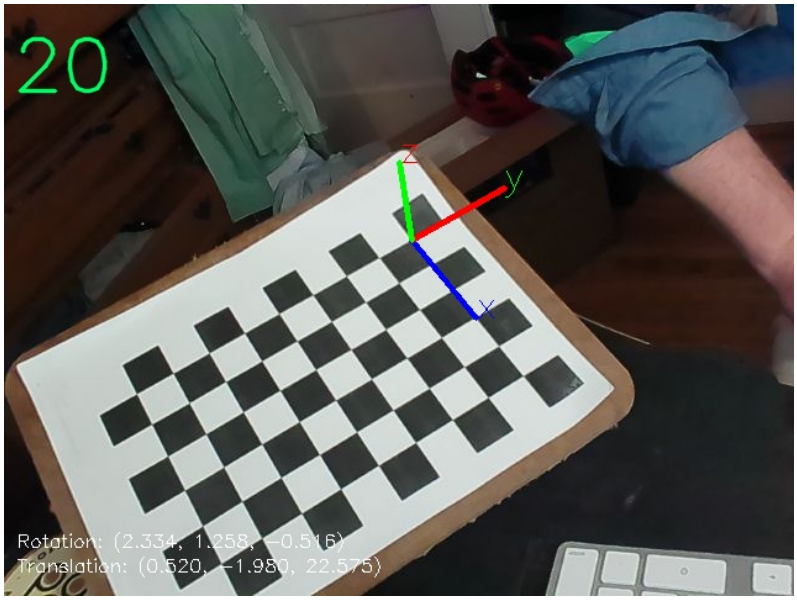
Distortion Coefficients:

```
[ -0.08 ]
[ 0.48 ]
[ 0.00 ]
[ -0.00 ]
[ -0.63 ]
```

```
Total Average Error:  0.496
```

Projected Axes

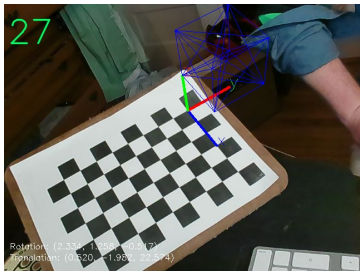
I decided to go with projecting the x, y, and z axes rather than highlighting the outer corners. This proved very useful for debugging the orientation of the coordinate system.



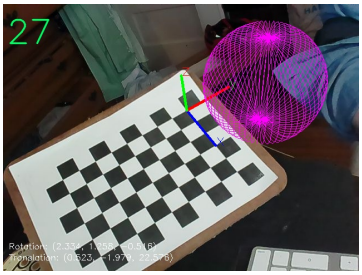
Virtual Objects

I decided that it would be fun to use this as an opportunity to brush up on my parametric solids. I used the equations for a cube, sphere, torus, and topological knot to create some relatively complex shapes relatively easily.

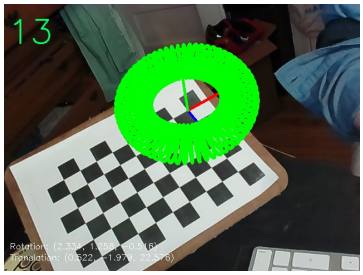
Each of these shapes can be triggered on or off individually with a keypress (indicated in parentheses).



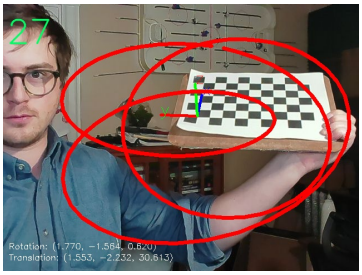
Cube (c)



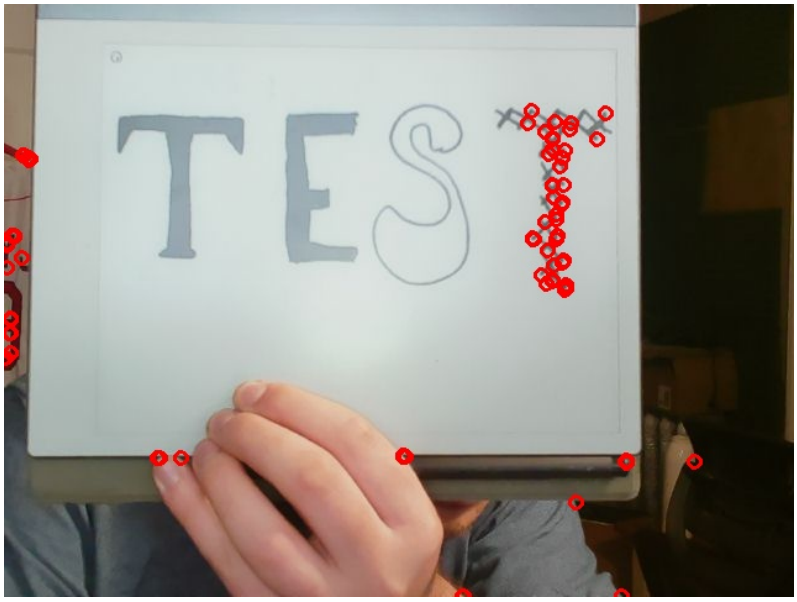
Sphere (o)



Torus (t)



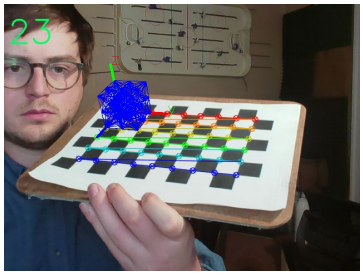
Topological Knot (k)



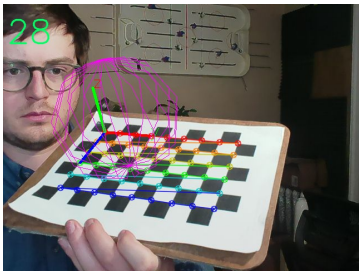
Robust Features I used the built-in Harris feature detection algorithm. The threshold for a reasonable number of features is very sensitive. To explore their sensitivity to texture, I wrote the word "TEST" in some whimsical fonts. Setting the threshold such that a number of features were detected in the hatched letter meant that almost no other features were detected; setting the threshold any higher resulted in tons of features being detected, which slowed my script down dramatically.

Extension

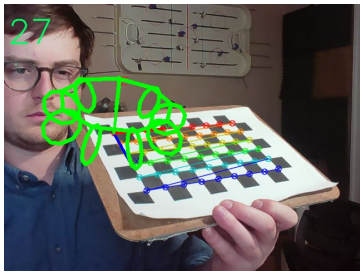
For my extension, I 1) added a live FPS readout to the program and 2) rewrote the program in python to explore the differences in the two APIs and their performances. Below are some saved images from the python implementation. Here are my overall takeaways from the process:



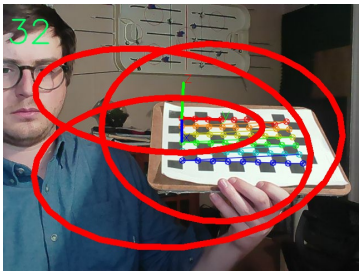
Cube (c)



Sphere (o)



Torus (t)



Topological Knot (k)

1. The python implementation was shorter in general because of the ability to use list comprehensions and other inline programming techniques to simplify functions.

2. The OpenCV FileSystem (a variant of XML/YML) worked well to pass the camera calibrations between the C++ program and the python program. However, I had to borrow a function from StackExchange to parse the XML in python because the built-in OpenCV function threw errors.

3. Frame rate was equal (and I think actually slightly smoother) on the python implementation than on the C++ implementation. However, python performed even worse than C++ when no chessboard was in the frame. I believe that this difference may actually have to do with the GUI interface - the python package installed with Qt, whereas the C++ only uses the more basic highgui include. However, it's not clear to me why the more complex Qt GUI would be *faster*...

4. Overall, I maintain that I could probably complete these assignments in about half the time in python with only limited difference in the performance of the algorithm. However, from a learning standpoint I've certainly learned more from grappling with C++. (And it turns out, based on a recent Co-Op application, that data science majors who know a "real" language are sought after...)

Reflection

I quite enjoyed working on this project. Even though my extension was a little bit less creative than some of my previous extensions, it was probably more useful in the long run, especially as we move into the deep learning part of the course and as I start to think about my final project. If I had more time it would have been fun too use an STL library to load some more interesting objects into the python program. That would probably demonstrate another reason why python can be advantageous for OpenCV projects.

Acknowledgements

I used the OpenCV documentation extensively. I also referred to several OpenCV tutorials for various parts of the extension. Lastly, my python implementation would not have worked without an XML parsing function that I found in https://stackoverflow.com/questions/11025477/error-loading-opencv-xml-file-with-python.