Alexander Buchholz

CPE 645

Final Project

**Real-Time Template Matching with OpenCV**

**Abstract**

Image processing tools are becoming increasingly common, and open source libraries like OpenCV make it simple to build all manners of capabilities without having to rewrite low-level code. This abundance of image processing tools, combined with higher processing speeds, opens up new opportunities for software developers to create software products centered on object detection. This project is an attempt at using OpenCV for real-time object detection through multi-scale template matching. The goal is to design a scalable solution that can collect object data to use as an input into other systems, such as entries into a database or nodes in a neural network.

**Overview**

ObjectCapture.py is a Windows-based python script that continuously takes screenshots and finds locations where the letters "h" or "H" are displayed on the screen. The object detector can be thought of in terms of several subsystems that handle inputs (screen capture, templates), preprocessing, template matching, and outputs (display window). A system block diagram is shown in Figure 1. In this case, screenshots are used as the input image, the letters "h" and "H" as input templates, and a display window as the output, but the same methods used to create ObjectCapture.py can be used in other object detection applications.

**Screenshot**

**Templates**

**Display Image**

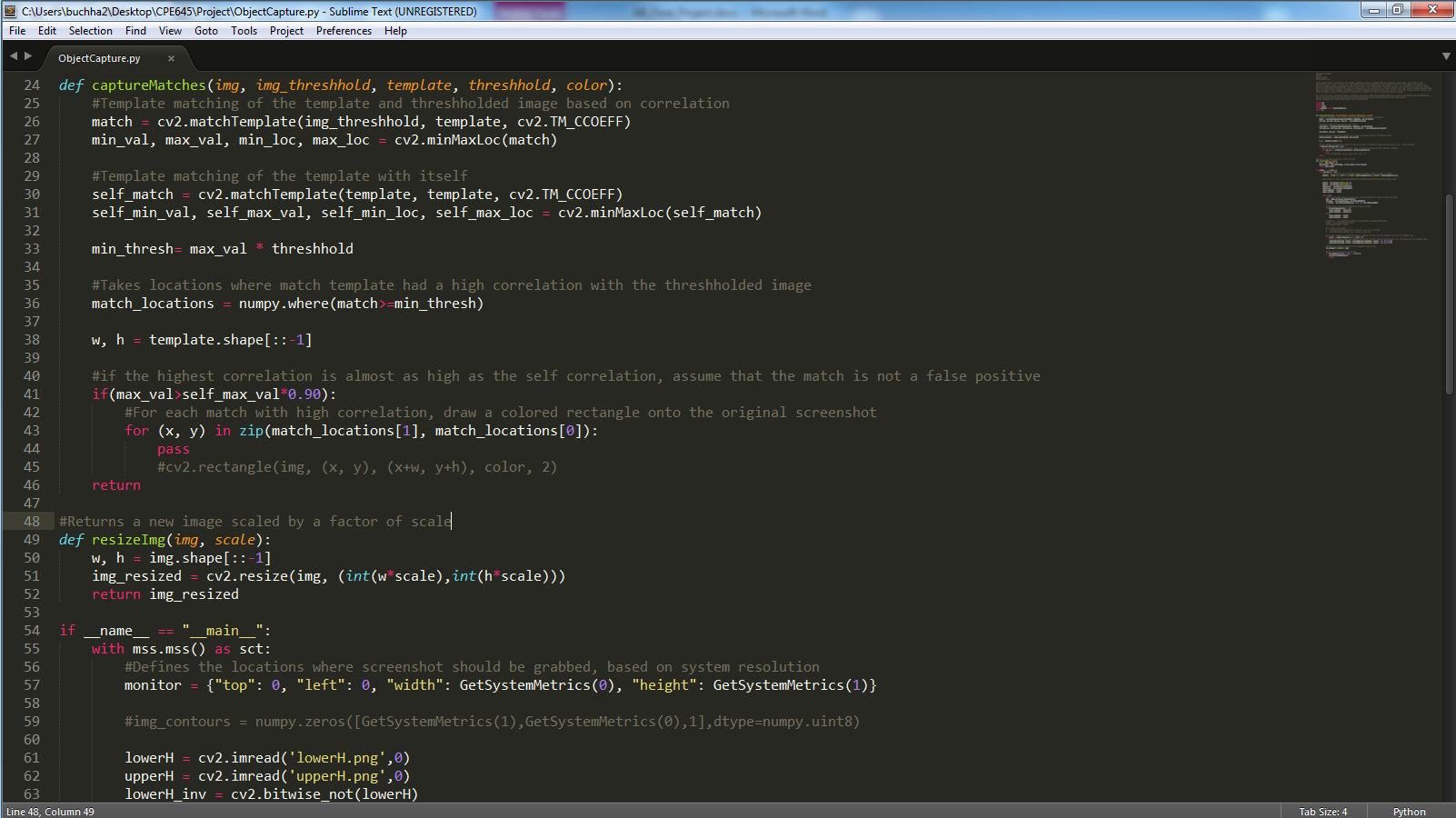
**Preprocessing**

**Preprocessing**

Figure 1 – System Block Diagram

**Inputs**

A few different libraries were investigated for screen sampling: Pillow, the Win32 API, and MSS. Since real-time screen capture was a concern, libraries were judged based on both ease of implementation and their performance time. Pillow was the easiest to implement, and Win32 gave the best performance with OS level commands, but ultimately MSS was used for screen captures. Because MSS utilizes ctypes, the platform offers better performance with a similar level of simplicity. On a 2012 Windows 7 test laptop, MSS was able to capture the full screen at a rate of around 30FPS, compared to the 15FPS offered by PIL. Figure 2 shows example screenshots used as test images. The first is a screenshot of the source code, which has bright text, while the second is a screenshot of the Wikipedia article for “H”, which has dark text.



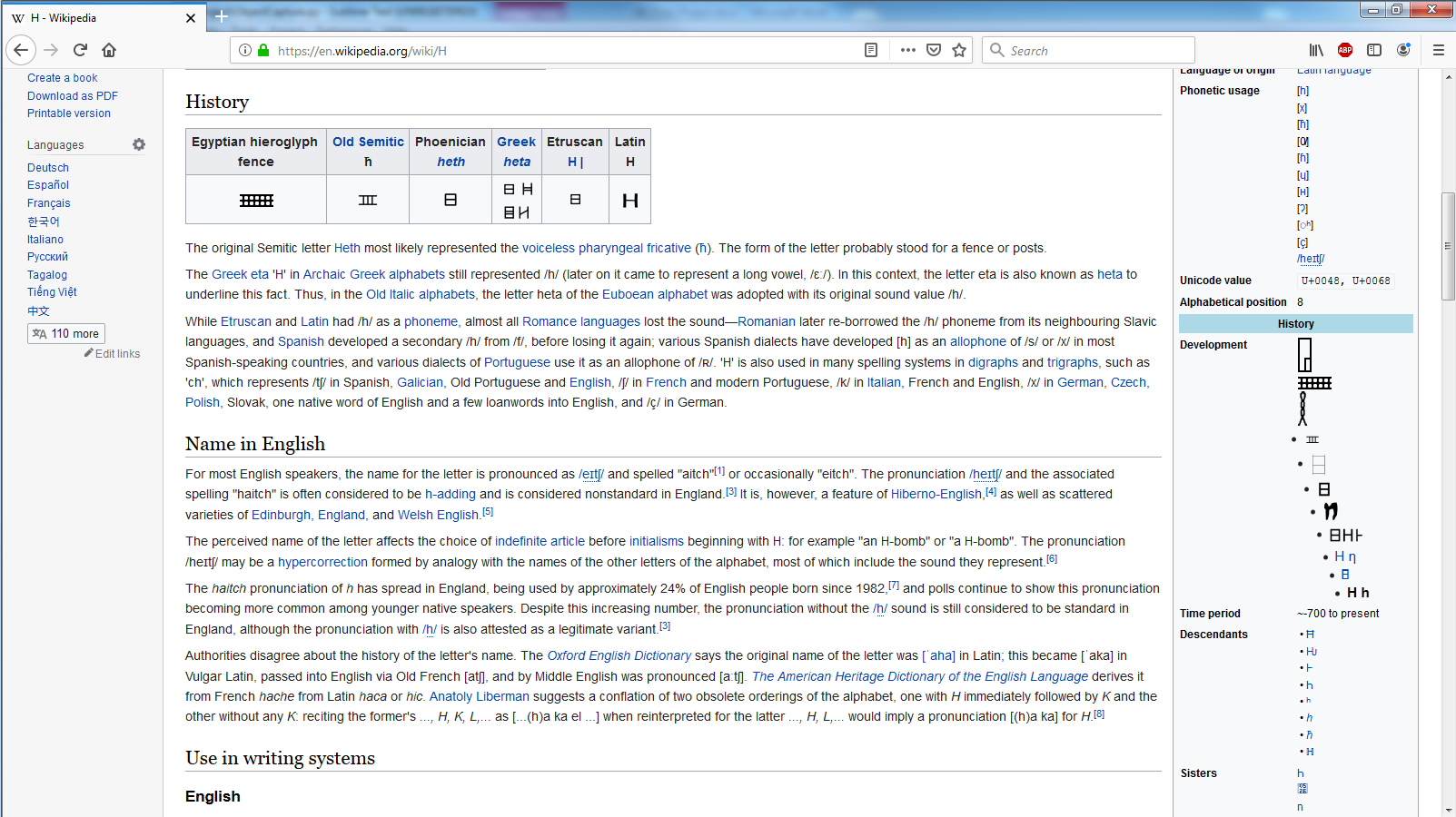


Figure 2 - Screenshots

For template inputs, the following images of the letters "h" and "H" were used. One goal of ObjectCapture.py was to capture letters of any color. To accomplish this, black and white templates are used, and screen captures are converted to binary during preprocessing. Using inverted templates allows the script to handle any colors of dark text on a light background, as well as light text on a dark background. The input template images are shown in Figure 3.

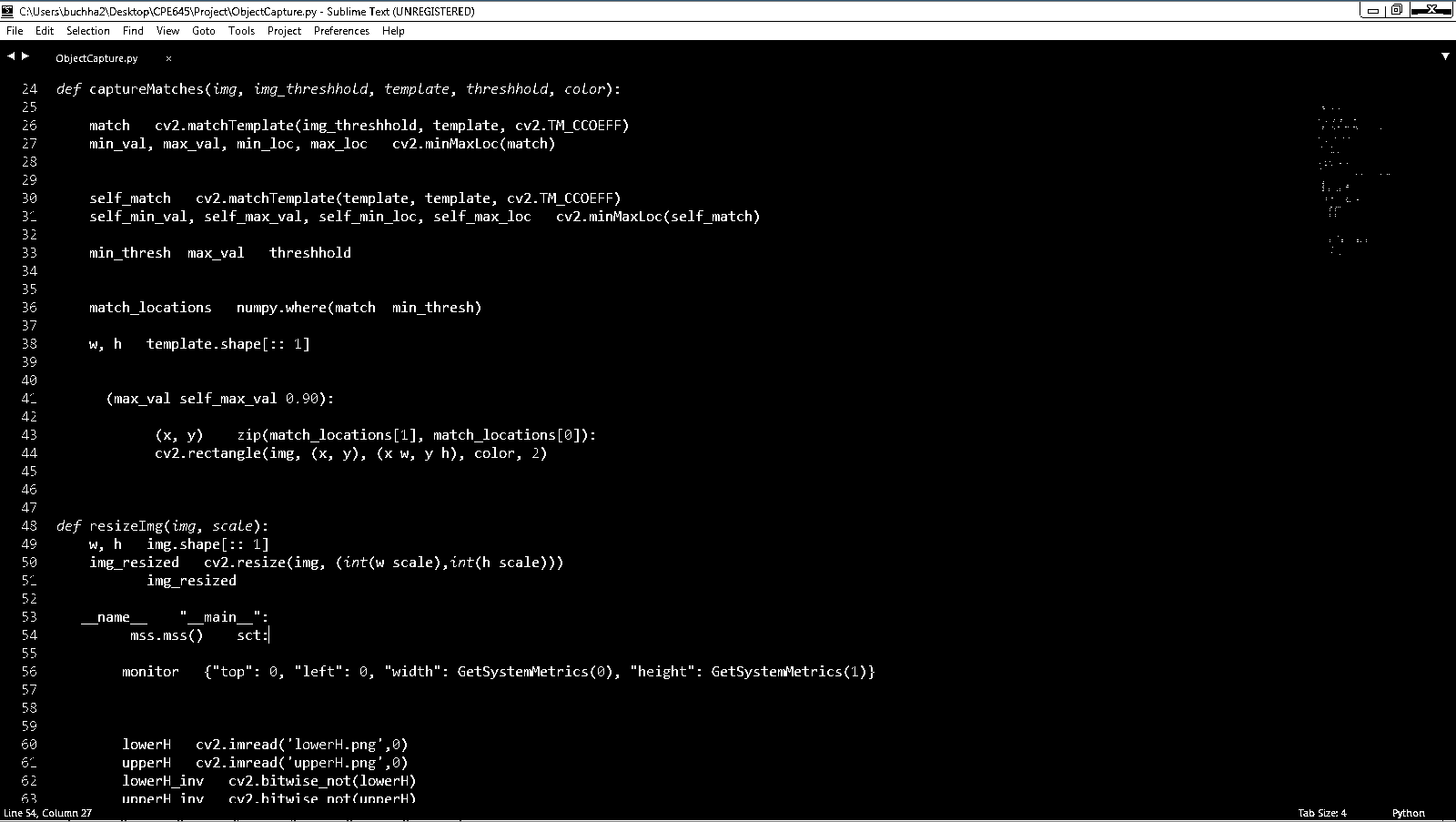
   

Figure 3 - Templates and Inverted Templates

**Preprocessing**

Grayscale screenshots need to be converted to binary images in order to perform accurate template matching. Originally, this was done by creating a binary image containing all of the contours in the image. Contours borders would then be filled with a solid color and output to a new binary image that would be used for the template match. Unfortunately, OpenCV's contour function was not always consistent with the color filling, making it difficult to create a color-blind binary image.

Since contouring did not handle bright images well, simple thresholding was used instead. ObjectCapture.py thresholds all values above 127 to white and all other values to black. Dark text is more likely to appear on a bright background, and bright text on a dark background, so the preprocessing phase also calculates the average brightness of the image and chooses the inverted template if necessary. There are some scenarios where dark text can appear on a mostly dark screen, and bright text on a mostly bright screen, but this simple thresholding is sufficient for most purposes. Figure 4 shows the thresholded images, which do not have the filling inconsistencies that the contour functions were providing.



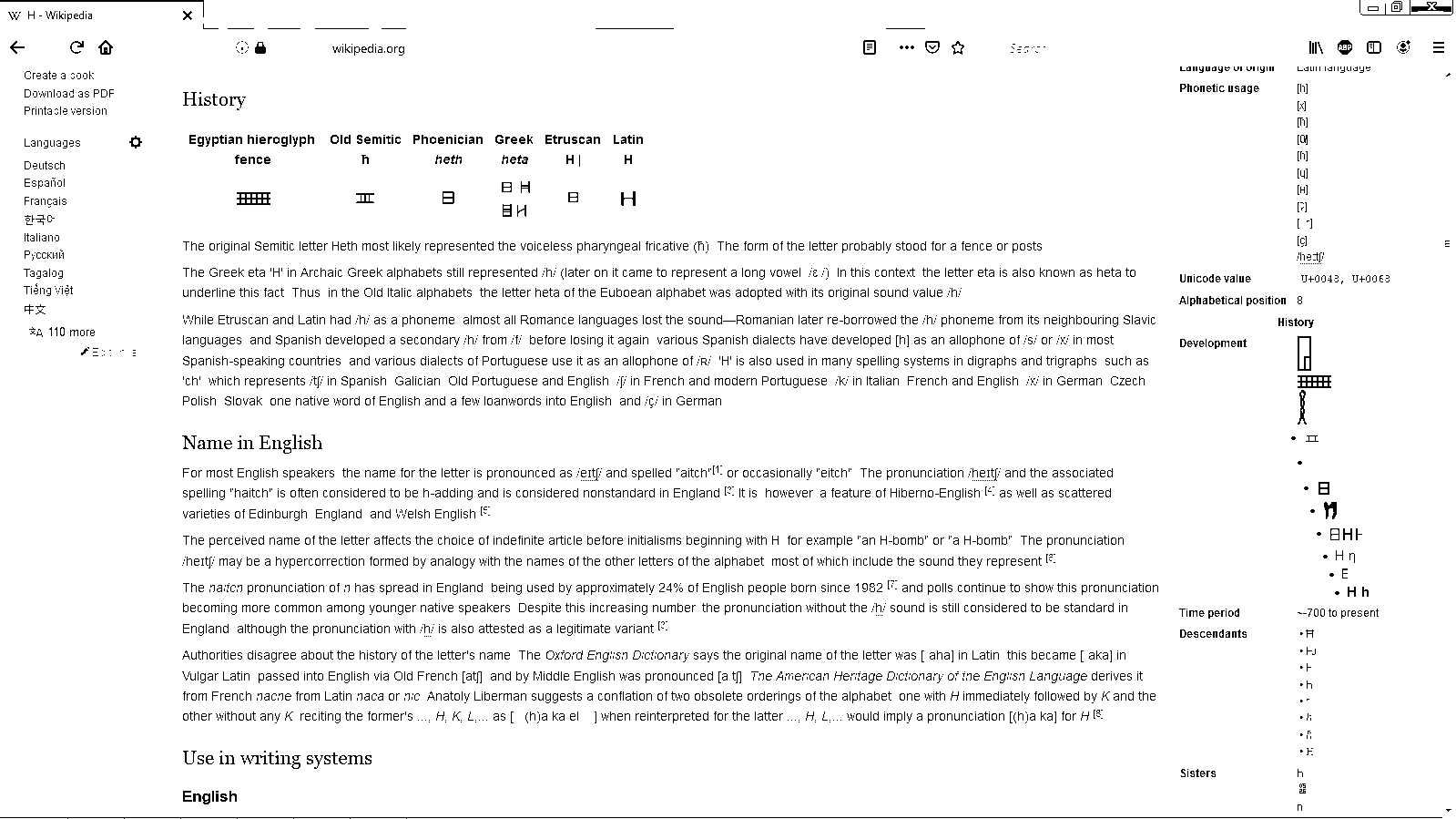


Figure 4 - Thresholded Screenshots

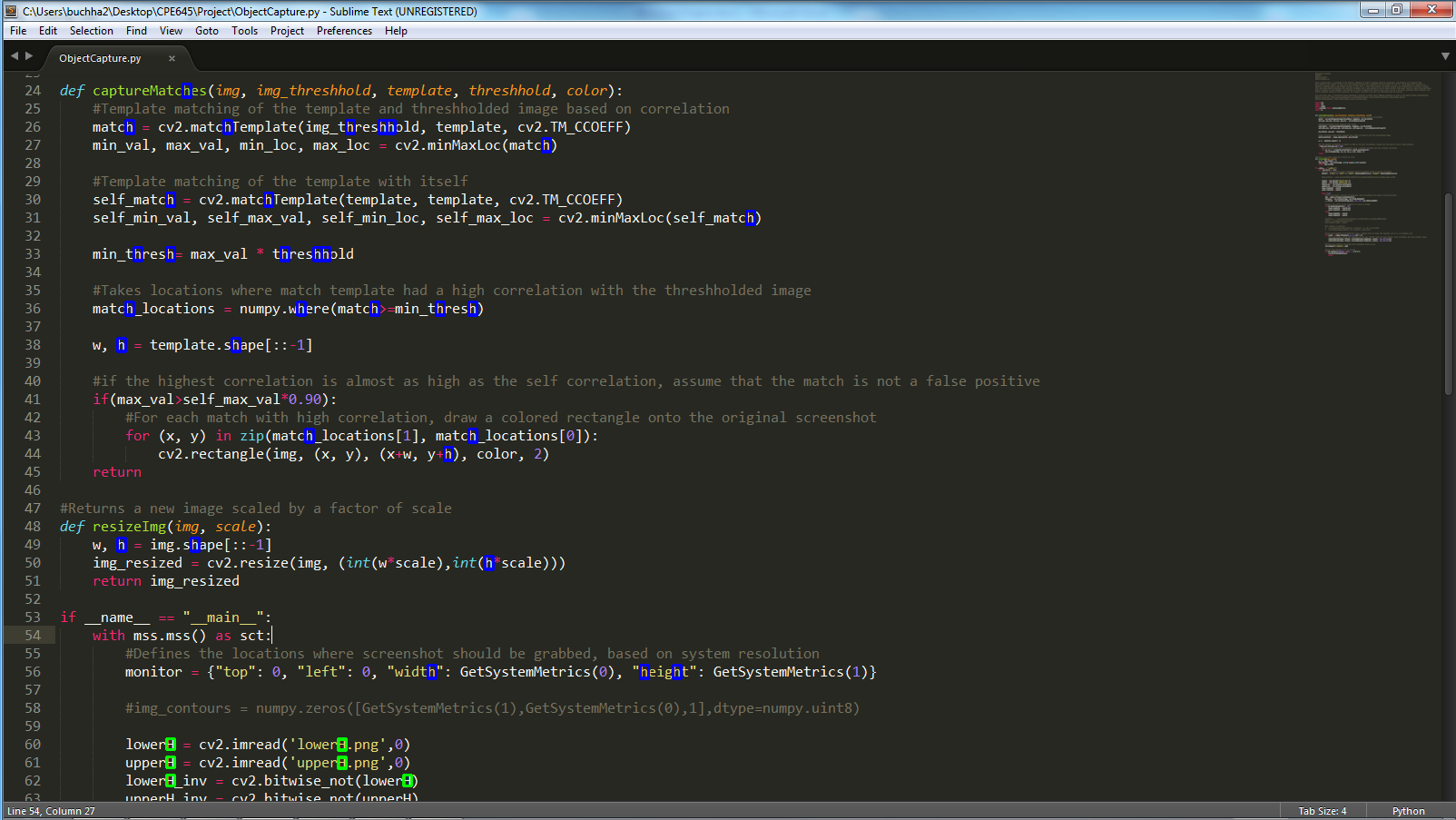
**Template Matching**

High accuracy template matching is accomplished in several different steps. First, templates are scaled down to different sizes. An example is shown in Figure 5. For the examples in this program, twenty different sizes ranging from 0.4 to 1.0 resulted in the best success rate at a decent performance. The templates are passed over the binary image to generate a new image based on correlation coefficient. To remove false positives, the templates are also passed over themselves, to get a reference for how high the correlation should be for a complete match. In locations where the correlation is above a fraction of the maximum correlation, as well as above a fraction of the self-match, rectangles are drawn onto the original image to indicate a likely positive match. When all sizes have been tested, the output image is displayed. This is repeated in a loop, continuously, until the user exits the program.

**** 

Figure 5 - Template Scaling

Although the speed is not as great as desired, ObjectCapture.py is able to output a new image on the test laptop at a couple times a second, which is fast enough for usable feedback. The script also does a pretty good job at eliminating false positives, which is difficult to accomplish in the context of text recognition, where the images are small and simple with few defining features. Performance would likely be faster on a stronger computer, but even on the test laptop, the performance is more than good enough for continuous feedback while browsing the web or reading papers. A second monitor is also practically mandatory for running the program too; without a second monitor to store the output image, the output window covers the entire screen. Figure 6 shows an example of image outputs from ObjectCapture.py. Capital letters are highlighted in green, while lowercase letters are highlighted in blue. Although there were a couple of false positives, or missed letters when there are severe font, color, or distortion differences, the program overall does an excellent job at matching letters.

****

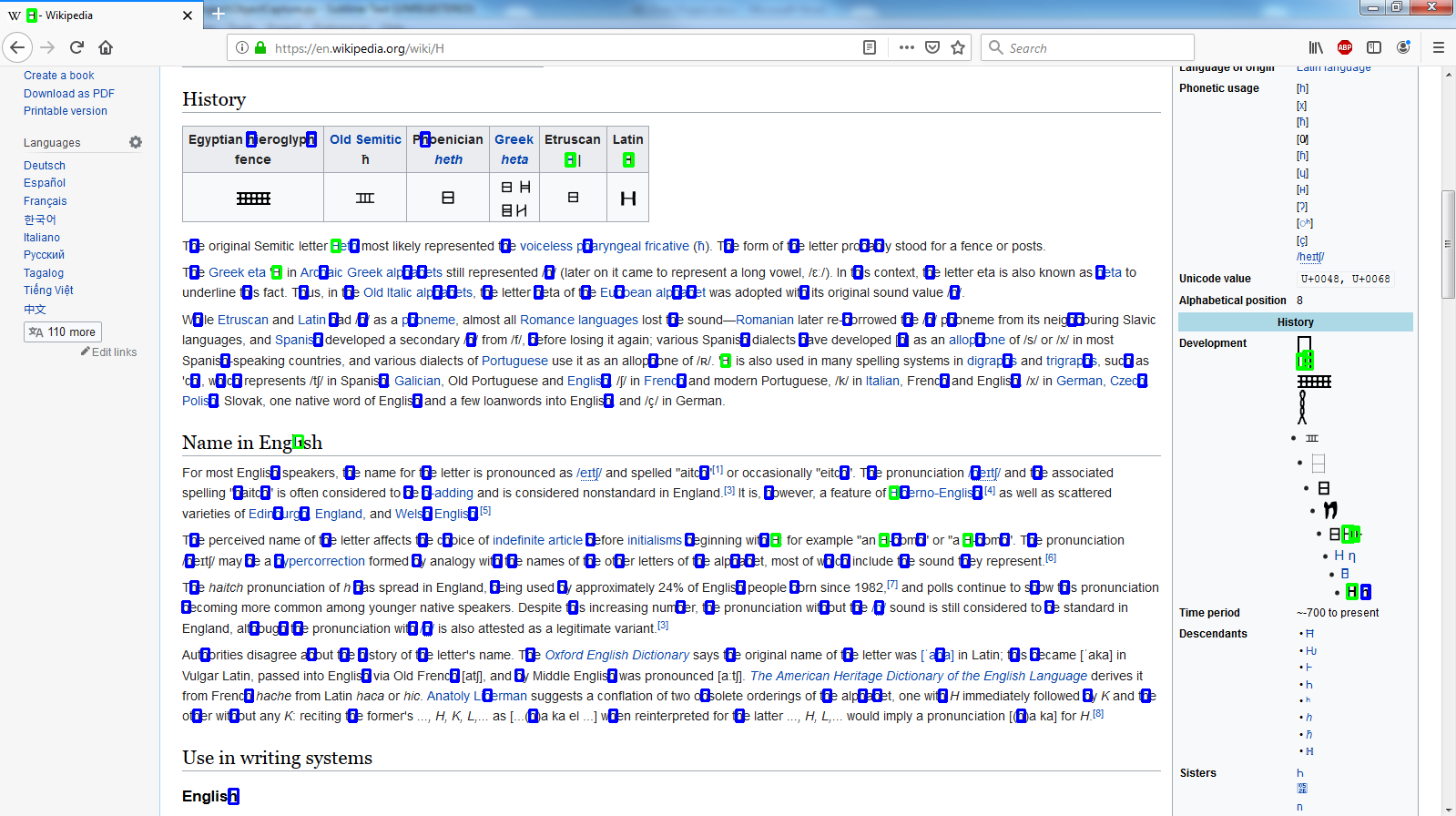
****

Figure 6 – Output images with "H" detection

**Conclusion**

Open source libraries like OpenCV and higher processing speeds are making it easier to incorporate powerful image processing tools into other software products. ObjectCapture.py demonstrates the relative ease of building image recognition tools based on template matching.