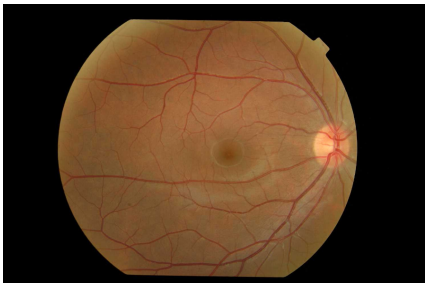
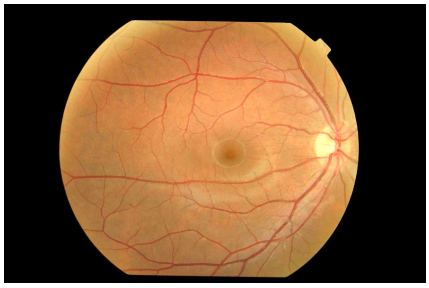
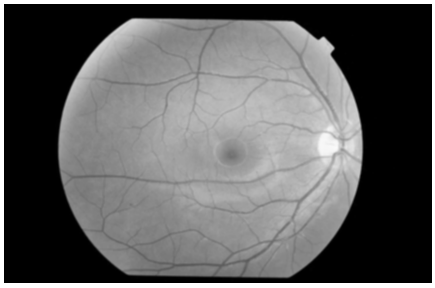
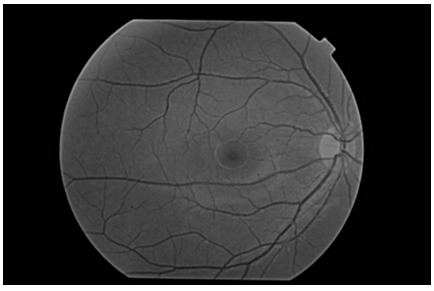
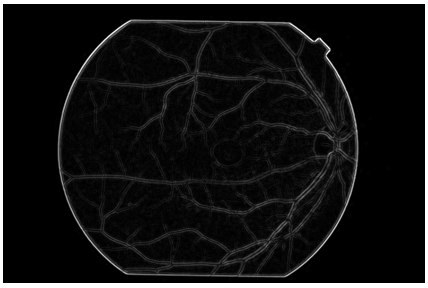
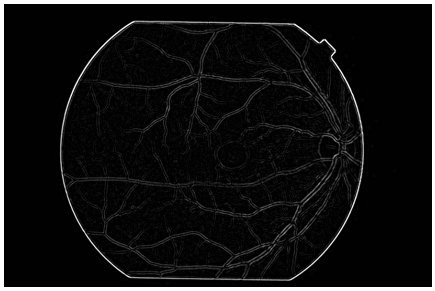
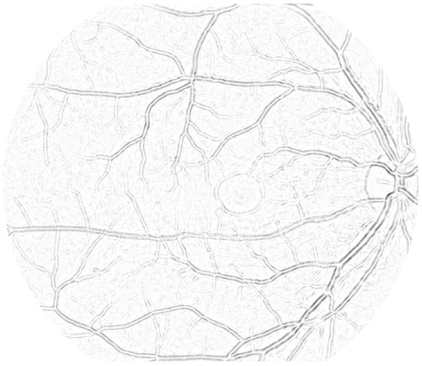
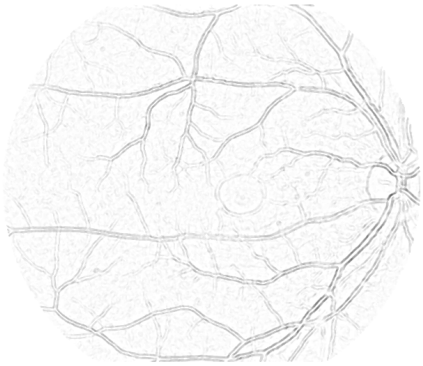
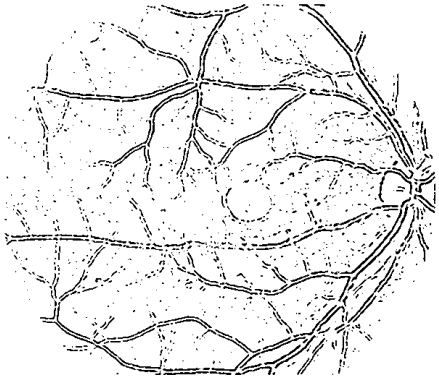
Assignment 4 Report

Created by: David Nguyen Huu & Ralph Huey Olegario

Computer Vision Pipeline

1. Resize image to 60% of its original size.
2. Increase contrast of the image.
3. Gaussian blur to smoothen the image, then greyscale.
4. Gaussian sharpen.
5. Sobel edge detection.
6. Apply the gaussian sharpen again.
7. Apply mask, invert colour and crop.
8. Median blur to further smoothen and reduce noise.
9. Adaptive threshold to binarize image.

Increasing Contrast

We upped the brightness of images as we wanted images to have uniform brightness in order to make it easier to compare images. We looked at three different contrasting methods: CLAHE, Equalise Histogram and raising the gamma.

- **Equalise Histogram**
  This contrasting method gave us quite a bit of trouble. Equalise histogram had a contrasting algorithm that deleted the veins due to veins blending into the background. So we scrapped it
- **Raising the gamma values**
  It is easier to control the gamma values of the image you want to change. Since I only needed to change 1 parameter, it is much easier to control compared to other contrast algorithms
- **CLAHE**
  When using this algorithm, it had a promising start of highlighting the veins quite well. But we ran into trouble when trying to blur this out would produce a lot of noise when converted from grayscale to binary image. So we scrapped CLAHE.

Noise Reduction

A way to reduce the noise in the images would be to blur them. Blurring also makes the veins pop more. When deciding which blurring algorithm we should use, we looked around for which blurring algorithms work the best. OpenCV had 3 blurring algorithms that we tried using. **Bilateral Filters, Gaussian Blur, and Median Blur.**

- **Bilateral Filter**
  Bilateral Filters were described to be a great way to define the edges. It does its job properly of making the veins much more apparent when thresholding the images. Due to making the veins more apparent, but it also made the noise too defined compared to gaussian and median blur.
- **Gaussian Blur**
  Gaussian blurring is the blurring method we use the most. Gaussian blurring blurs the image quite a bit. Which was a good sign. We played around with the values and decided that a kernel size of 1, looping again to have a size of 3 made the image blur just right. We use this blurring before the grey scaling in order to perform some form of noise cancellation.
- **Median Blur**
  At the start, we discounted median blur as gaussian blur just does the job of noise cancellation compared to median blur. But Median blur has one advantage over gaussian blur, and that is not cancelling out the gaussian sharpen we did on the image.

Edge Detection

We tried two different Edge Detection methods for this assignment: Sobel and Laplacian.

- **Laplace Transform**
  Laplace transform is the first edge detection algorithm we have tried. The problem with Laplace Transform is that doing adaptive thresholding makes the veins look blurred or look like paint splotches. It was not something that we wanted the picture to look like.

- **Sobel Transform**

Sobel transform felt like a better edge detection algorithm than laplace from our experimentation. One thing that sobel did better was identifying the edges better and making the outline look like what we wanted. Still a bit noisy, but it seemed doable

Sharpening

Sharpening allows for more defined veins. Sharpening has the downside of making the fake edges much more apparent. In this case, Ralph defines fake edges as non-veins.

- **Gaussian Sharpen**
  Gaussian sharpening was used when we wanted to have more defined veins. This worked well when we positioned sharpening before edge detection in order to make the veins more apparent.

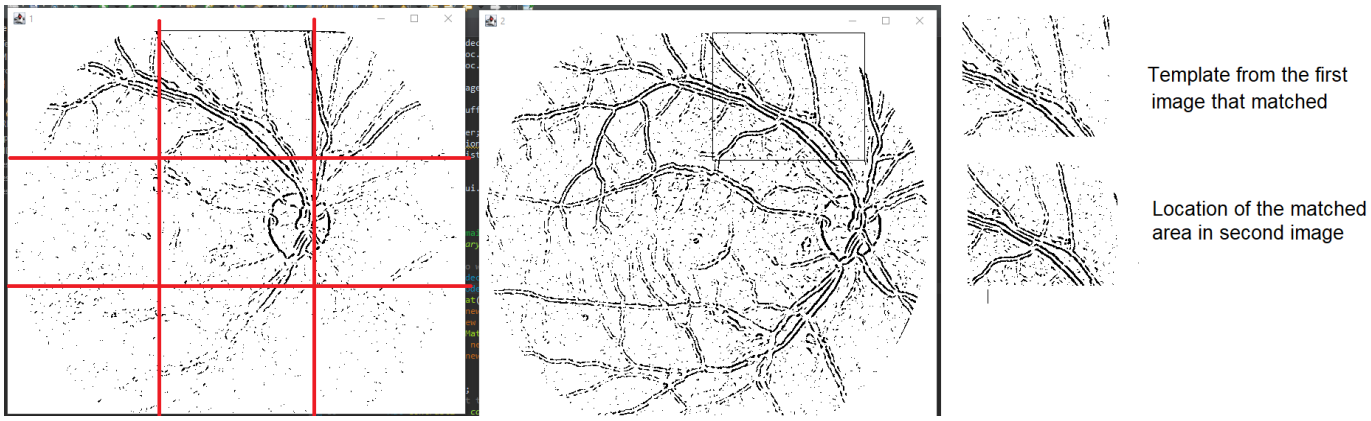
Masking and cropping

After edge detection, the eyeball had white veins as well as a white outline of the eyeball and a lot of unnecessary empty space. To remove the white outline, we masked it out by creating a circle mask that was just a bit smaller than the eyeball so that the outline wasn't included. For the cropping part, we moved the image into the top left corner as we can without losing any veins and then we cropped the image. After masking and cropping the image, we then inverted the colour because it gave us a nicer result when we later binarized the image.

Matching Algorithm

When coming up with the matching algorithm, the plan was to split one of the processed images into a square grid (4x4). This would give us '$4^2 = 16$' templates to use for template matching on the other processed image. A loop will run that will use the 16 templates and try to match them with the other processed image. If the similarity passes a certain threshold, then a match is found and the algorithm returns true. If no similarities are found that pass the threshold, the algorithm returns false. Only one template needs to pass the threshold for it to be a match.

Example with a 3x3 grid using IMG000001_7 and IMG000002_7:



Matching Accuracy

To test the accuracy of our program, a test was implemented which processes every image in RIDB and runs the matching between every *ordered* pair that can be made (the order of the pair matters in our matching). The test finishes after about an hour and results in only 2 failed matches, where each pair matched when it shouldn't.

=> (9900 - 2) / 9900 = 0.9998 = 99.98% accuracy