**Detailed Report**

I’ve briefly gone over the details about the project in my summary report but in a more detailed report, I’ll be going into more details and discussing why I did what I did. To start off, I began my project which dealt with key point matching by researching what I should use to match the images. It was at this stage that I compared the benefits of using a FLANN matcher, which stood for FAST Library for Approximate Nearest Neighbor compared to a BF Matcher, which stood for Brute Force. In lab 10 I had used a FLANN matcher, but for this final project I decided to use a BF Matcher because while looking at the benefits of each, I discovered that FLANN matchers are faster but less accurate than a BF Matcher because the BF Matcher tries all possibilities, whereas the FLANN matcher finds an approximation. I did discover that the FLANN matcher’s accuracy can be increased at the expense of the speed but in our case, I found doing that would be unnecessary as our program would run fast even while using the Brute Force matcher. After deciding on what matcher, I can use, I started the project by reading both images and validating them. And declaring the ORB detector, at this stage I didn’t go into too much detail regarding other detectors as I decided to use the ORB one because it was the one, I was most familiar with. I then declared the 2 key points and 2 descriptors. And ran the detector’s detectAndCompute function with their respective images, validated them and then declared the BF Matcher. I created the vector to contain the matches found and called the match function which returns all of the matches found in the vector I had created. This was where I encountered my first hurdle, the details of which I have already gone over in my summary report. Upon compiling, the program would crash at the match function. I traced the problem further back and discovered that the detect function had returned an empty key points array, when that was passed to the match function the program crashes. This problem was later discovered to be a problem with the detector which I fixed by adding a parameter to the creation of the ORB detector changing the edge threshold from 31 to 15. The program now worked with the image template\_A.png and found the key points. I then sorted the vector from best to worst matches and appended the top 15 matches to a good\_matches vector before drawing those matches in a third image, this was done to reduce clutter and only display the 15 best matches. This new image will draw lines connecting the matches found in the two images. And as stated in my summary report, this is where I encountered my second hurdle. The image was cluttered with individual key points found in the two images in the form of colored circle which I removed with the parameter NOT\_DRAW\_SINGLE\_POINTS, this resulted in the image only showing the lines connecting the two images without unnecessary clutter. I finished the project by resizing the image to an appropriate size before displaying it as I noticed that some of the images were too large to display properly. Now that I have a working program, I went back to earlier parts of it and printed the distance and key point matches to the console when compiled. Mostly complete, I reread the project details, and noticed that I was supposed to add salt and pepper noise to the image and document its effect on the match accuracy. So, I added the noise as we had done earlier in the semester in one of the labs and tested the feature matching with the noise set at various probability values. I first tried it with the probability value set at 0.05 and slowly increased until a maximum value of 0.9. I noticed with a lower value, the difference wasn’t very noticeable but as I increased the values, the matches were more wide spread. Because the noise was only added to the larger image, the number of matches remained the same because the number of key points found in the template image remained the same. But even at a value of 0.9, matches were found in the larger image despite the original image no longer being visible and the match lines lead all throughout the image with a drastically reduced accuracy. In conclusion, the matching can be fairly accurate depending on the image, but in our case when we compared braille the matcher had difficulty with all the white space and a lot of the results lacked accuracy. In the case of the template image template\_A.png, all matches found were incorrect. This could potentially be due to the amount of white space in the image seeing as how the image was a single dot with a white background. Because we are comparing braille, all letters found in the original image consisted of the dots with a white background. So, a possible fix for this would be getting the feature matching to recognize when the dot has nothing around it so when matching, the braille letters with another dot too close are eliminated. But overall, the key point matching works fairly well, with the exception of some braille characters.