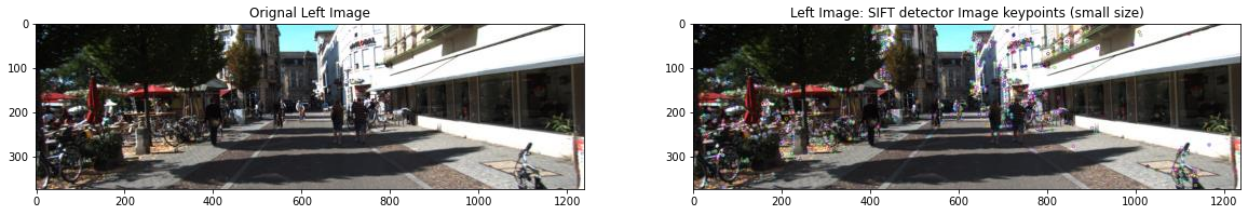
AER 1515 Assignment 2

Tushar Aggarwal - 999356913

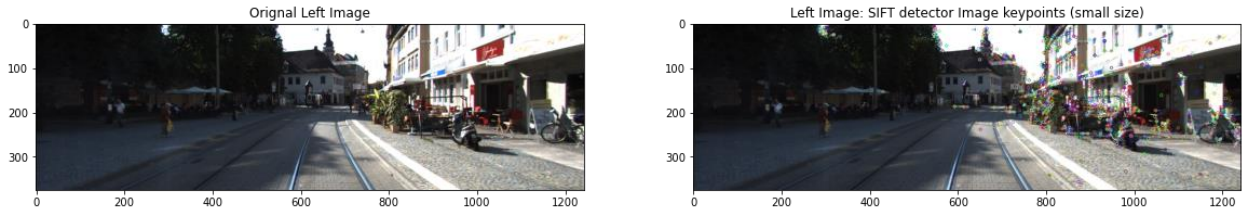
Part 1: Feature Detection

Images from Test Set

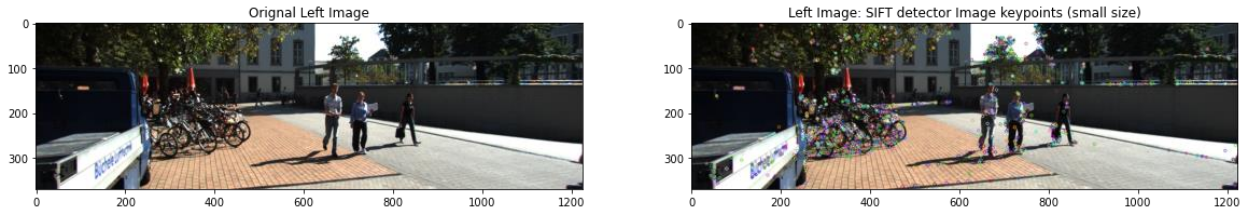
11



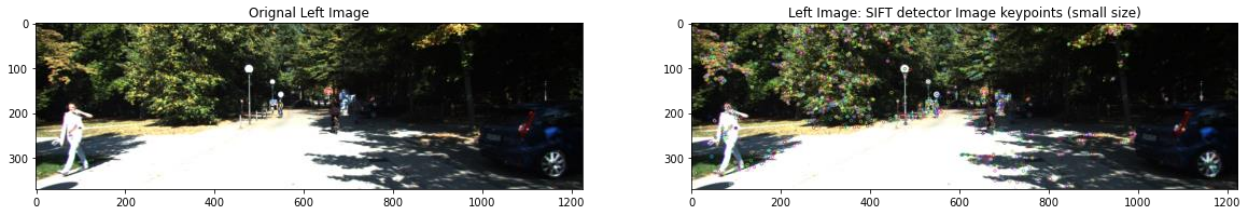
12



13



14



15

Please note I had shown original and after detector implementation in my Jupyter, hence I just ended up using that for left images.

The detector and descriptor algorithm used for feature detection is: SIFT **(Scale-Invariant Feature Transform**

Looking at the images many unreliable features can be noticed. Image 11 we can see there are some keypoints detected on the left side of the building plain wall. Additionally, there are some key points detected on the tree branches on the left side which do not clearly give any information and hence are not reliable. Images 15 and 14 have key points on the floor of the images. Some of these points are detecting the edge of the shadows due to illumination and some of the other points are just detecting the floor surface where it is a bright spot without any edge. Such points do not give us much information about the feature and may not be reliable keypoints. Image 15 is an example of this as well where we have some keypoints in the center of the image where the sky is overexposed.

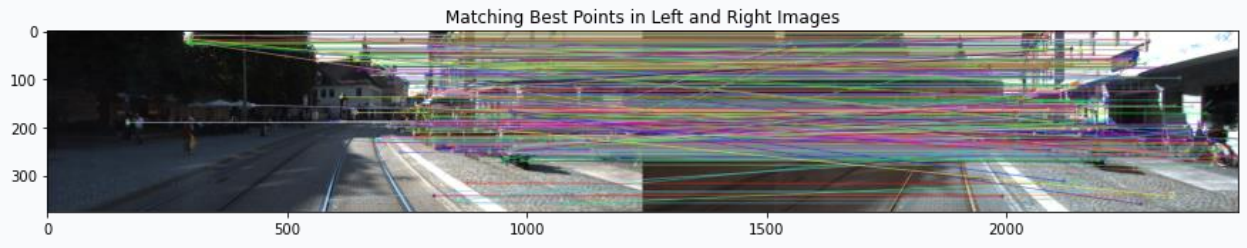
Also, the images are not very high resolution which may cause some issues for the detector to detect good key points. In Image 11 and 13 a lot of the keypoints are clustered together in one area on the left side. Due to low resolution of the image it could be difficult for the matcher to match some of these kepoints as the pixel value can be completely different as well.

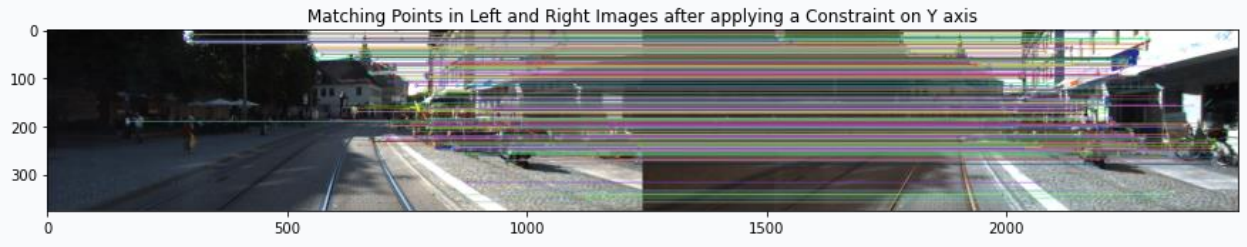
Part 2: Feature Matching

Images (Top: Without epipolar constraint, Bottom: with epipolar constraint of Y<=1)

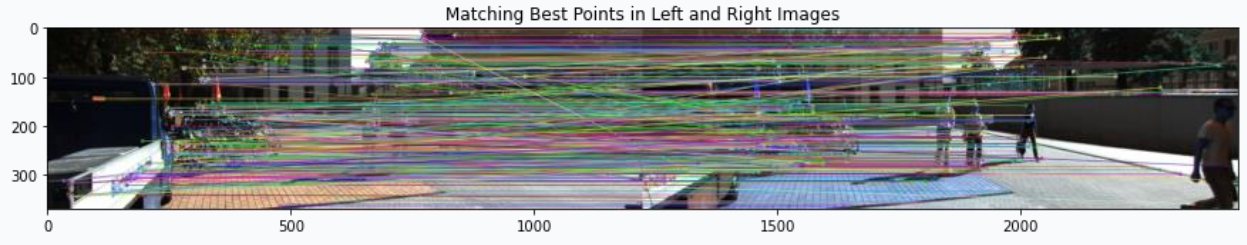
11

12

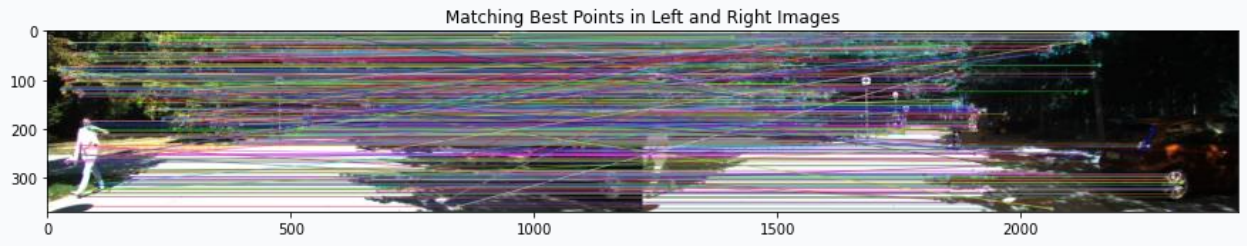


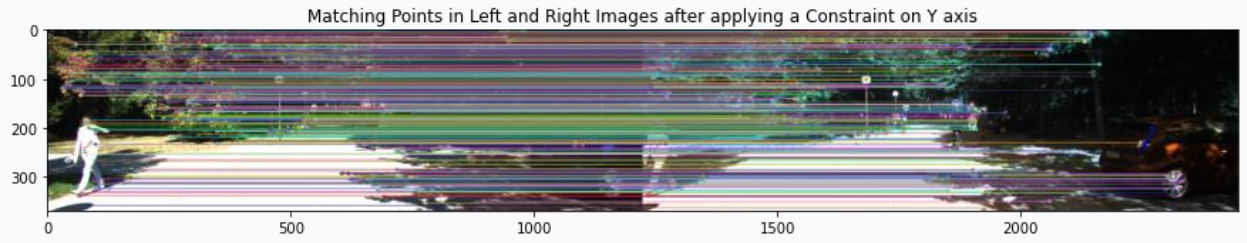


13

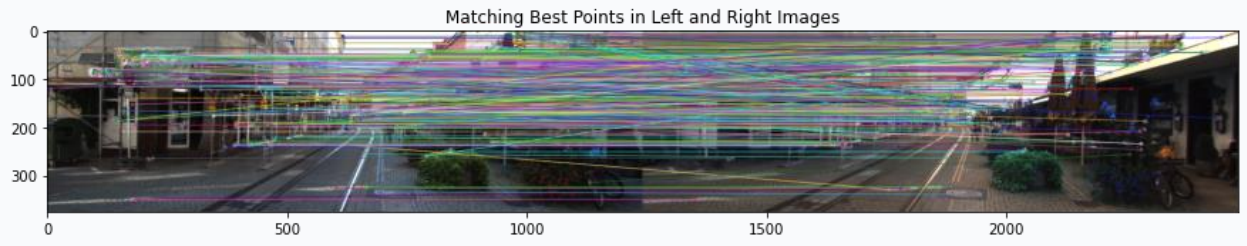


14





15



We used Brute Force Matcher for the second part of feature matching using OpenCV with Crosschecking enabled to match the feature descriptor for left & right image we obtained from part 1. Cross checking is useful as it helps with evaluating right image features to the left image features and give better results for out match. I also enabled L2 Norm for feature keypoint matching.

Additionally, since the image given to us were rectified, we applied epipolar constraint on the y axis. If points that matched from left and right image have an absolute distance of less than 1. We kept those matches, and the result can be seen in the second image shown above. The feature matching lines are more straight resulting in much less correspondence between left and right image features.

Part 3: Outlier Rejection

11

12

13

14

15

For part 3 we applied RANSAC outlier rejection we calculated that the number of keypoints feature that were matched reduced compared to our epipolar constraint in Part 2.

We used the OpenCV's findFundamentalMat function on our keypoint point to get the inlier values. Evaluating the algorithm, we found that the optimal points were received at RANSAC threshold of 0.5 and confidence of 0.99. The function uses RANSAC algorithm with 8 points.

RANSAC implementation was fine tuned for our outlier rejection by using multiple hyperparameter values. I experimented with threshold values 0,0.1,0.5,0.7,0.9,1 additionally, confident interval was set to 0.9, 0.98, 0.99 and 1.0.

Values 0 for threshold and 1 for confidence level give the best for the maximum number of images however, 0.5,0.99 gave the best result for maximum number of images.

Disparity was also calculated in this section by subtracting the X-axis value for keypoints in the left and right images. Depth was calculated using disparity using the formula:

$$\text{Depth} = (\text{focal length} * \text{baseline})/\text{disparity}$$

During our matching process it is important to note that only matches where our inliers value was nonzero (ie the x value was not 0) were selected.

Therefore, comparing the matches from Part 2 and in Part 3 it can be seen that RANSAC did a much better job. Additionally, number of keypoints selected drop showing that RANSAC was being selective and was implemented well. Many points that were standing our as mentioned in part 1 were also rejected after RANAC implementation.

I would also like to point out that findHomography function was also tried (currently commented in the code), however, I found findFundamentalMat function worked better.

Resources used:

- OpenCV documentation
- Lecture notes
- Stackoverflow (for error evaluations)