AER 1515 – Perception for Robotics

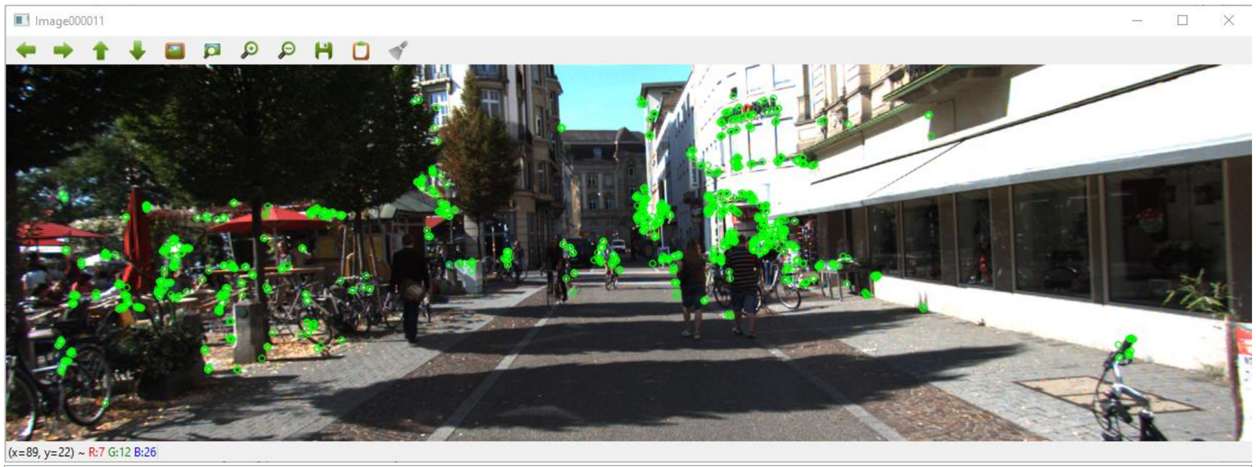
Assignment 2

Chokpisit Kasemphaibulsuk

1. Feature Detection

<u>Feature Detection Result:</u>

Image11



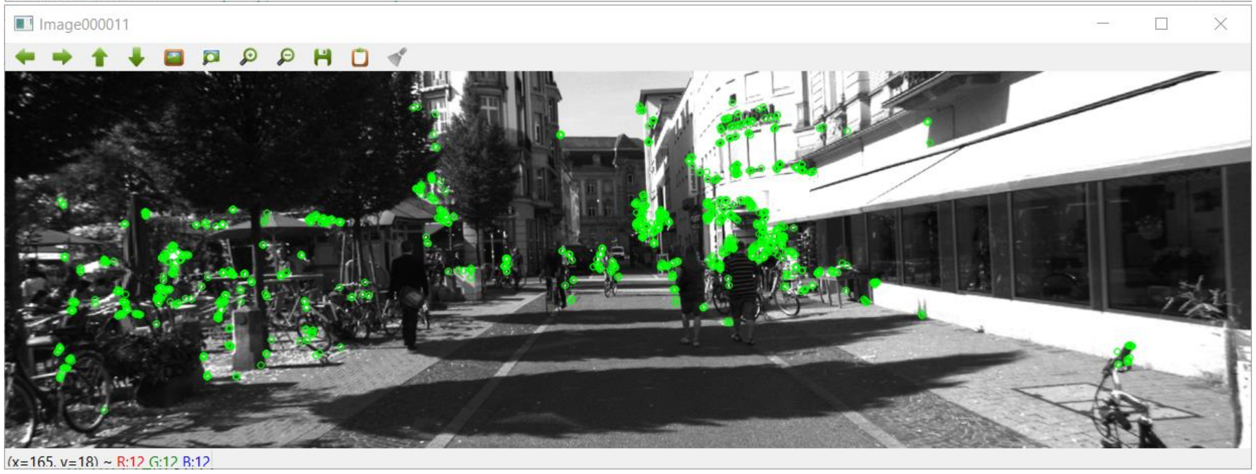
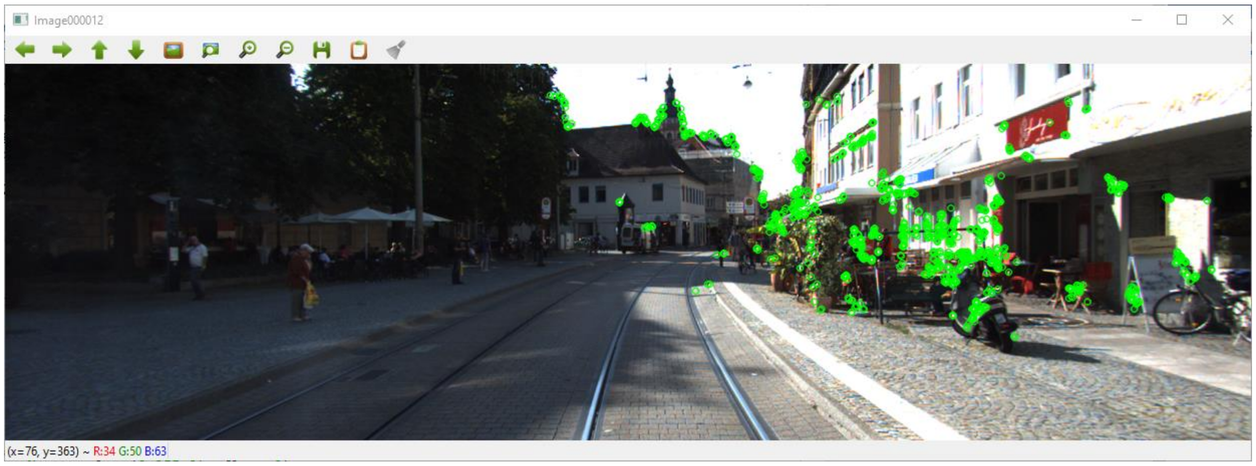


Image12



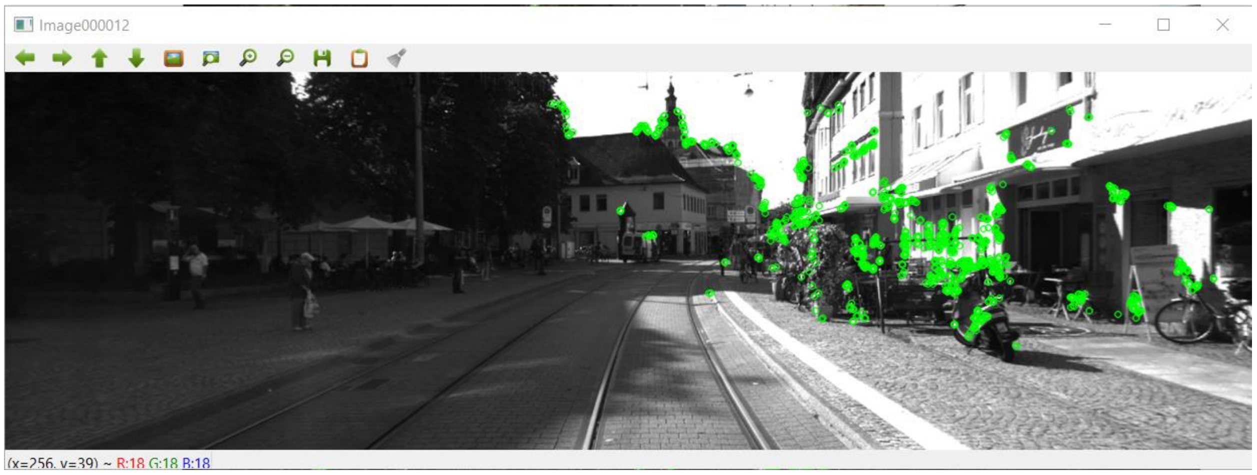


Image000012
(x=256, y=39) ~ R:18 G:18 B:18

Image13

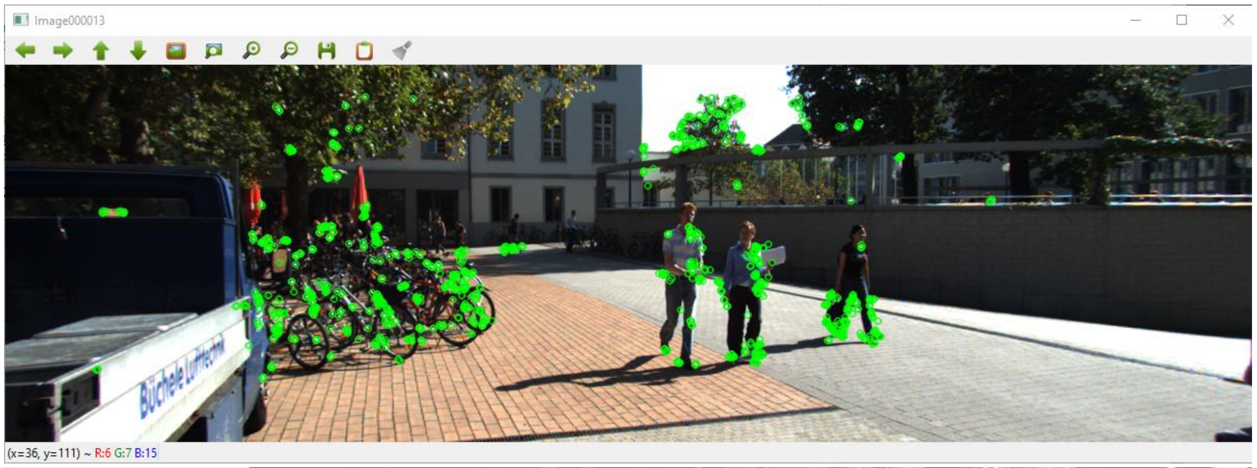


Image000013
(x=36, y=111) ~ R:6 G:7 B:15

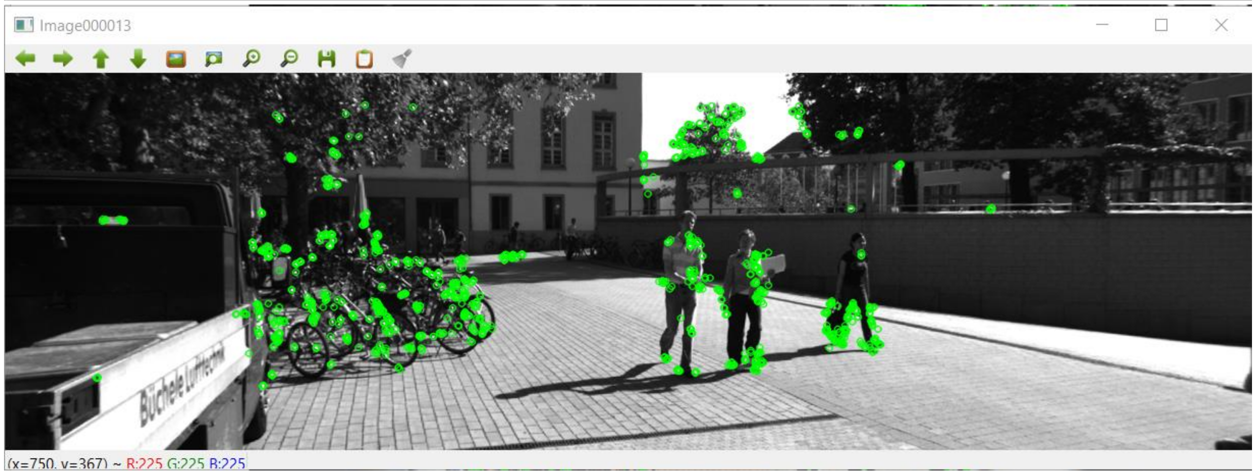
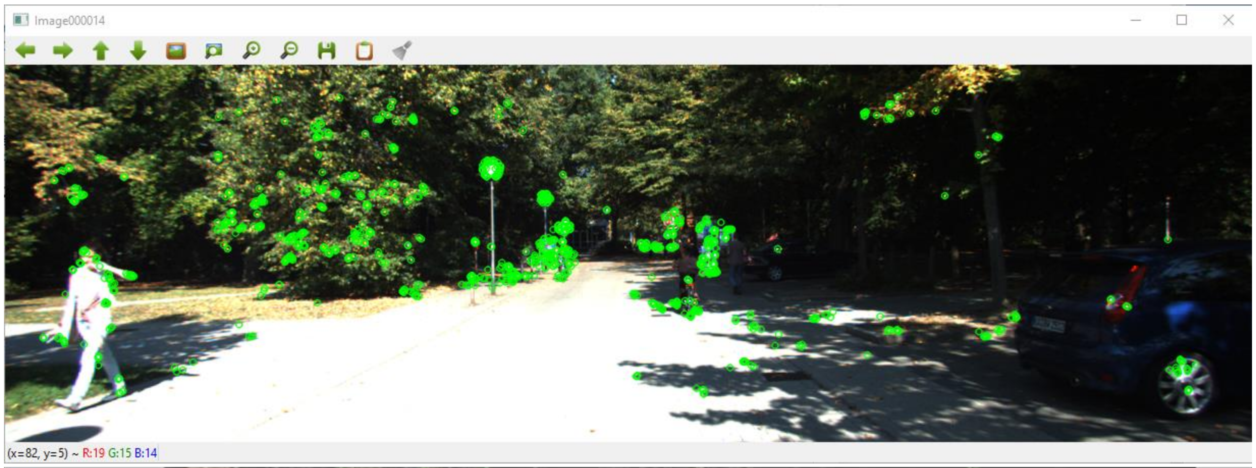


Image000013
(x=750, y=367) ~ R:225 G:225 B:225

Image14



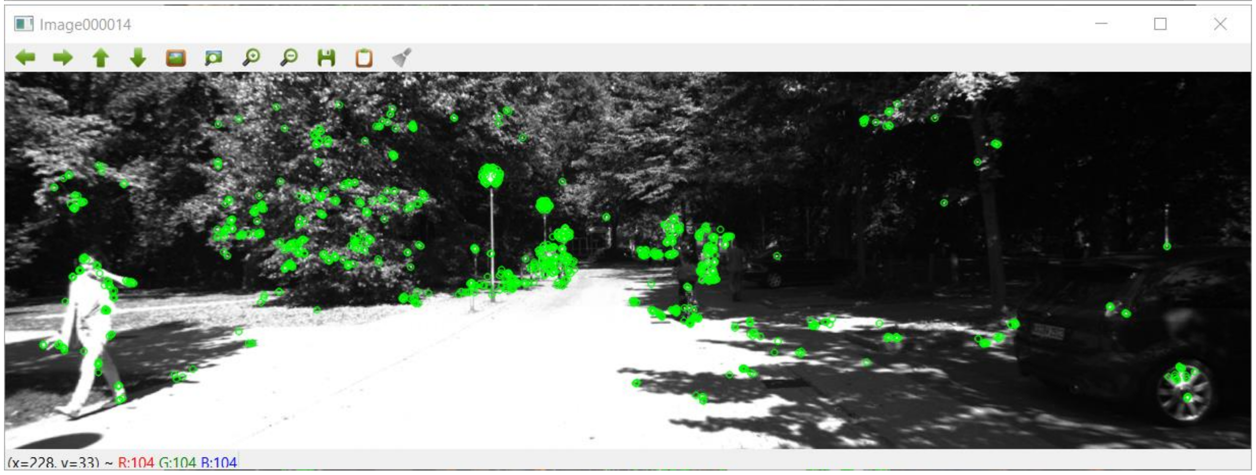
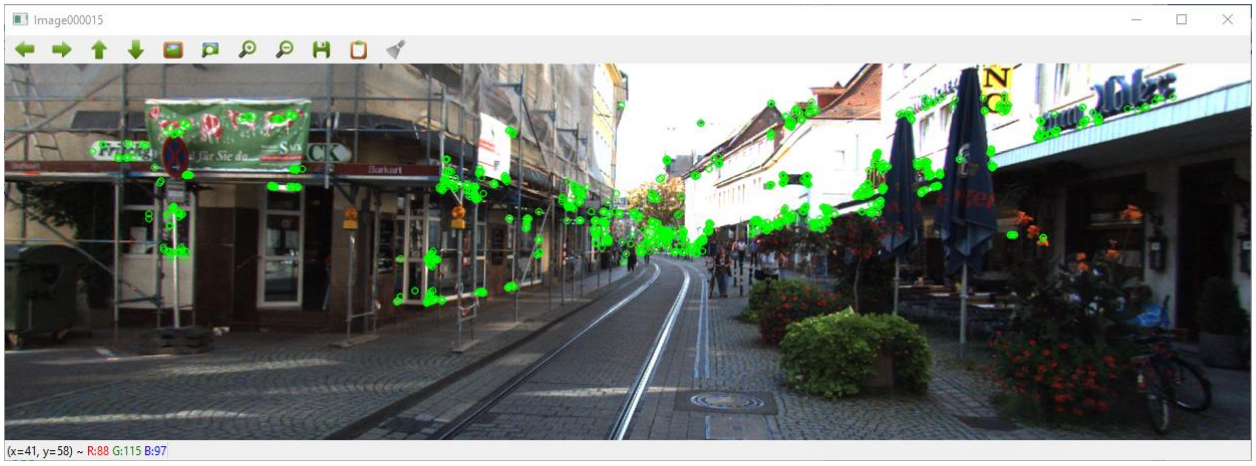
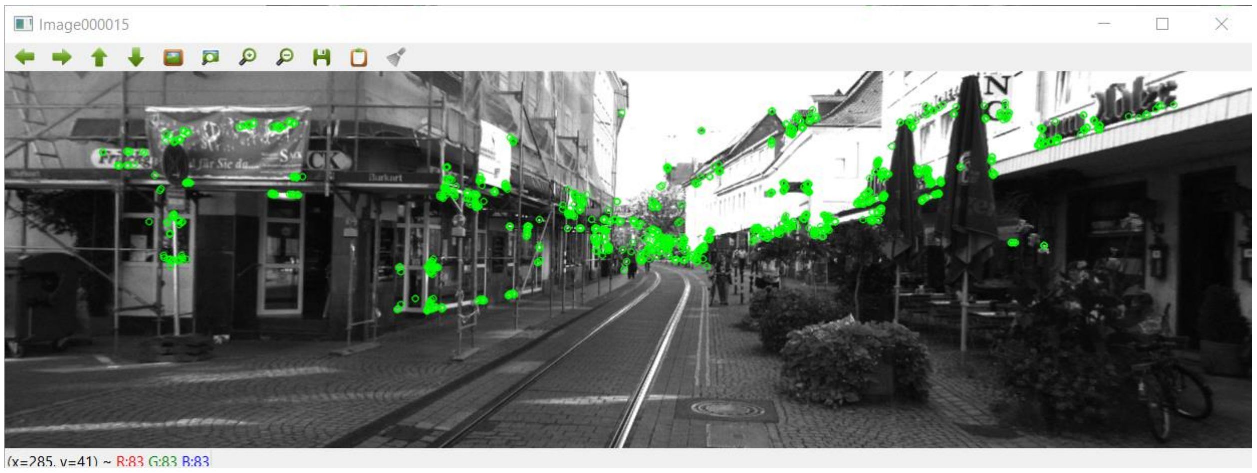


Image15





Discussion and Observation:

I use ORB (Oriented FAST and Rotated BRIEF) detectors and descriptors for finding the keypoints. The keypoints that are not as reliable and may lead to matching difficulty and error are one around the edges either of a building or a pole of the sign like in figure 14 and 15. Edges are not a good feature because the descriptor along the same edges will be the same, therefore it won't allow us to match two close by features for matching. On the other hand, corner would be a better keypoint because there is only one descriptor in each image so it will match correctly.

Another unreliable keypoint is the one occurring on the wall or ground which has no unique feature. For the wall and ground, the detector detects the feature from the gradient and change in intensity from the shadow, but it is not reliable because it's not robust since the shadow and light could be altered from the movement of surrounding object and environment. Originally, those planes have no specific feature and should not be use as keypoint. It is better to use something that is rigid and static as keypoint, which are usually manmade object like building corner, car, sign, etc.

Lastly, keypoints on nature object such as tree leaves and branch may not be a good feature for matching. They are not distinctive because all the leaves look the same in low resolution and are too small to find distinctive feature, so it won't be robust and repeatable. Trees' trunks and branch could also be thought of as edges and could not be distinguish easily for other trees.

And as you can see, the feature in RGB image and grayscale image are almost identical. Maybe because the algorithm convert image into grayscale before finding feature or that the feature is still detectable with the method of the detector, which could be something like finding the feature in each color channel. I suspect that since grayscale image still have the same gradient just like the RGB image, the detector still detects the same feature. But after observing, certain area with feature in RGB image does not exist in grayscale. Grayscale would remove the insignificant gradient and make the detectors to detect only significant features.

2. Feature Matching

Feature Matching Description:

The features are match using brute-force matching method with the BFMatcher object. The chosen parameter for my features matching are cv2.NORM_HAMMING and crossCheck=True.

Brute force feature matching algorithm simply choose the feature with the minimum distance to be its best match. Because I am using ORB feature detection which use a binary descriptor, the parameter NORM_HAMMING is chosen to use the hamming distance as the distance function for matching since that equation is for binary descriptor. The equation is shown below.

Hamming Distance:
$$d(f_i, f_j) = \sum_{k=1}^{D} XOR(f_{i,k}, f_{j,k})$$

When crosscheck is true, for each feature in left image, the BFMatcher will only return the nearest pair which is the consistent pairs. The result of feature matching is shown below:

<u>Feature Matching Result:</u>

Image11

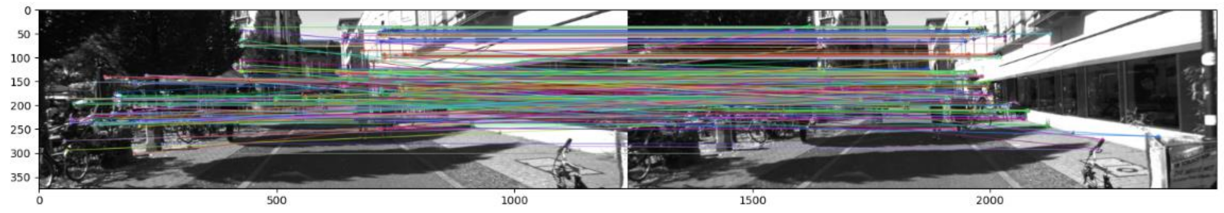


Image12

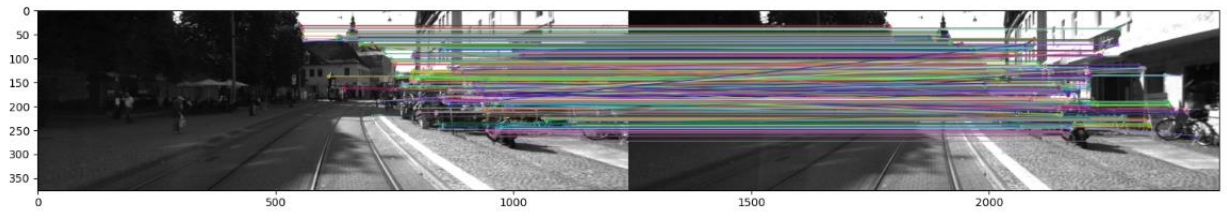


Image13

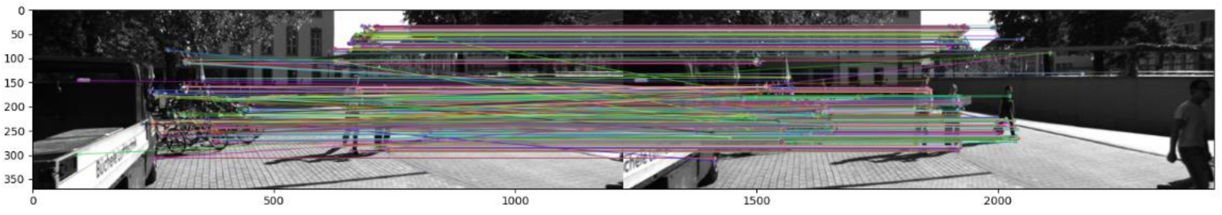


Image14

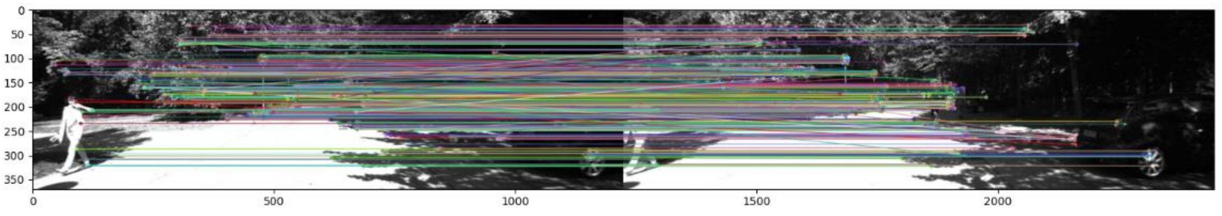
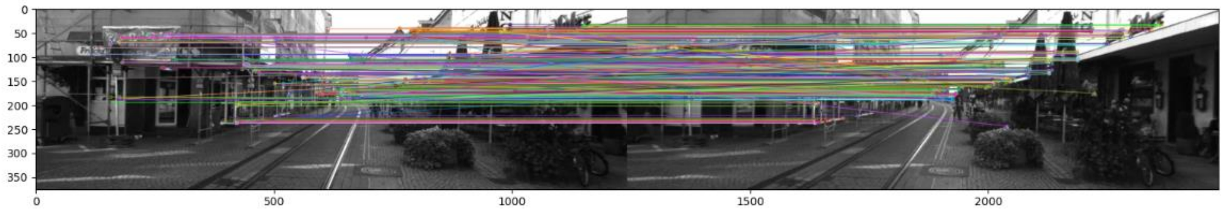


Image15



Feature Matching Evaluation:

To evaluate the accuracy of my feature matching, I calculate the depth of that feature and compare it with the depth measured with Lidar sensor. When trying to compare the depth image generating by calculating the depth from disparity and the depth from lidar, I have made a few observation. Some feature point give me a disparity of 0 so it wasn't able to calculate the depth. Assuming the epipolar constraint, the disparity could be calculate with the following equation: disparity = ul – ur, which mean the difference of that feature u position (in the x direction) from left image and right image. If disparity is 0, it means that the feature happen to be on the same pixel location for both image whch mean the depth is infinite. In reality, this could never happen because true zero disparity mean infinite distance. However, zero disparity here is the result from finite accuracy and rounding effect. Therefore, we'll be ignoring that depth. Depth is calculated by the following equation: Depth = focal length * baseline / disparity. Since distance is inversely proportional to disparity, as the depth get closer to infinity, disparity would be getting closer to zero. As the object get further away, the disparity get smaller until it was round to be on the same pixel. The performance of the feature matching is measure using the RMSE (root mean square error) of the estimate depth from feature matching and measured depth from Lidar sensor. The RMSE result of training image is shown in part 3.

3. **Outlier Rejection**

Outlier Rejection Matches Results:

Image11

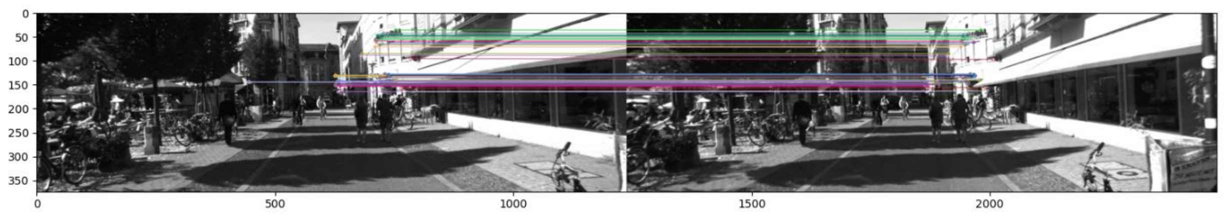


Image12

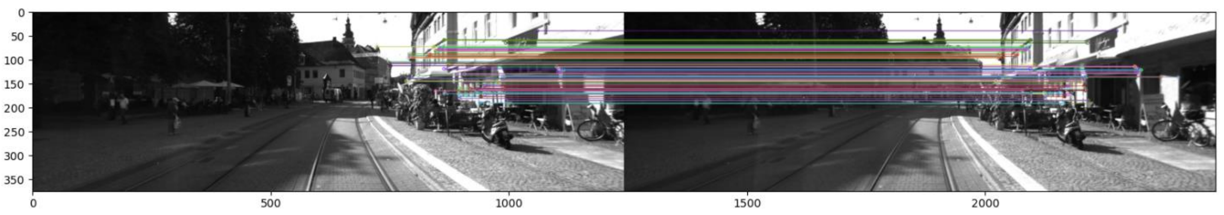


Image13

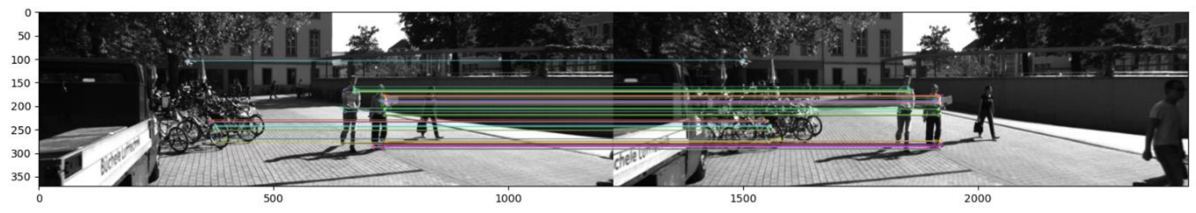


Image14

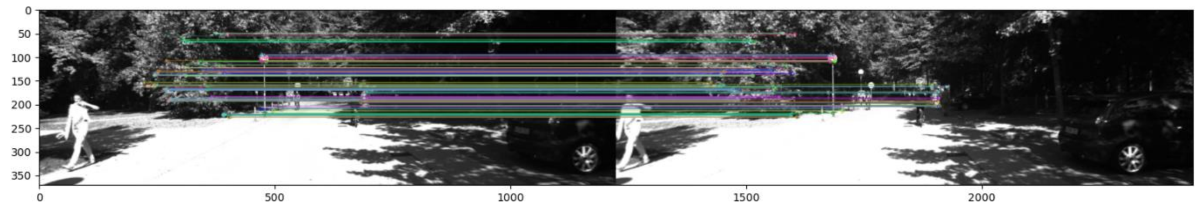
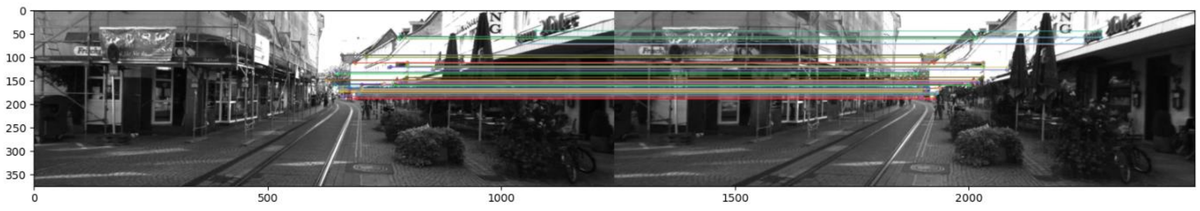


Image15



As you can see, the number of matches decrease because the matches which are outliers were removed. Now all the matches are mostly horizontal line because the images are stereo images so the matching feature should follow the horizontal epipolar line, meaning that the transformation should just be translation in horizontal direction. The incorrectly match features such as matches from the feature of a tree which is diagonal are removed because they are classified as outliers according to the RANSAC algorithm, which will be explain in the next section.

Outlier Rejection Algorithm Explain:

The outlier rejection algorithm that I chose is RANSAC implemented by OpenCV. The RANSAC algorithm is run through the opencv findHomography() function. The parameters of findHomography() are the srcPoints, dstPoints, method, ransacReprojThreshold, maxIters, and confidence. Here are the description:

- srcPoints – x and y coordinate of points in the first image
- dstPoints – x and y coordinates of the corresponding points in the second image (target image) which match the srcPoints. The same index matches each other.
- method – method for computing homography matrix and outlier rejection. Here I choose RANSAC
- ransacReprojThreshold – threshold for determining wheter a point is inlier or outlier with this equation:
  $\| \text{dstPoints}_i - \text{convertPointsHomogeneous}(\text{H} * \text{srcPoints}_i) \|_2 >$ $\text{ransacReprojThreshold}$
- maxIters – maximum number of iterations
- confidence – level of confidence between 0 and 1, this is a the inlier ratio threshold

The RANSAC algorithm works as follow:

1. Randomly select the fewest number of sample need to compute the model. Here the model is a 3x3 matrix consisting of rotation and translation. It can be computed with only 4 points (4 features match from srcPoints and dstPoints). If there is less than 4 points, then the function will give error.

2. Compute the model parameter (the 3x3 transformation matrix) from those 4 points
3. Calculate how many matches fit that model within the ransacReprojThreshold, which we will called this number of matches inliers C
4. If inliers C > inlier ratio threshold or the programe reach the maximum number of iteration, go to step 5, else go to step 1.
5. Recompute the model parameters from entire best inlier set

After the findHomography() function finish the RANSAC, it return the Homography (the 3x3 transformation matrix) and the mask which is a vector of 0 and 1 telling which matches are outlier or inlier.

Tuning Parameter:
There are three main paramenter that was used to tune the algorithm: the ransacReprojThreshold, the maxIters, and confidence. The parameters are tune until the RMSE is minimized in the training set.

The ransacReprojThreshold is set to 1. This threshold will decide if the keypoint is inlier or outlier with the equation
$\| \mathrm{dstPoints}_i - \mathrm{convertPointsHomogeneous}(\mathrm{H} * \mathrm{srcPoints}_i) \|_2 > \mathrm{ransacReprojThreshold}$.
If this statement is true, i.e., the norm of the distance difference between the destination point and the transform source point using the model is more than that threshold, then it will be classified as outliers. Thus, the mask output will classify the corresponding matches index as 0 if its outlier, and 1 if it's inlier. Setting this threshold value as 1 means that the srcPoints after transformation should be really close to dstPoints. After testing, this value minimized the RMSE.

The chosen max iteration for RANSAC is 5000. Increasing the iteration could increase the processing time if RANSAC could not meet the desire amount of inlier, but with only 1000 features point as a start, eventually it will reach the desire point within short period of time. Therefore, 5000 iteration is enough to ensure the algorithm can classifier all the outlier and inlier and calculate the correct model. The RMSE result does not change when the max is set to 4000, but giving it 5000 just to give it additional iteration to ensure it find the best homography and remove all outliers.

The confidence level is chosen to be 0.925 which mean that the number of inlier C (number of point classifier as inlier) is about 92.5% of the total number of matches. So if I have 1000 matches, then at least 925 of them must be classified as inliers. The default value for this param is 0.995, but after tuning this parameter, by lowering this value I observe that it will prevent the model from overfitting and lower RMSE.

Assumption:
One major assumption of this algorithm is that it already assume the srcPoints is correctly correspond to the same index of dstPoints. From the BFmatching, if too many source points are incorrectly matches with the destination points either because there are not enough good

feature for matching or just have edges or plain wall which could lead to incorrect matches, then we could never get the correct transformation matrix and the mask would also be incorrect. Since RANSAC implement within the findHomography will not be performing any matching or correct any mismatches by finding the right match, so we have to assume that most the the feature matches are correct. It is also assume that the number of srcPoints and dstPoints are the same, or else the program will give out an error. Lastly, we are assuming that the object with minimal distance is the only perfect match because crosscheck is set to true. For the case of edges and plain wall where descriptor looks the same, if crosscheck is turn false then the single feature in source point could be matched to multiple destination feature point, and RANSAC could try to eliminate the outlier and incorrect matches.

<u>RMSE Result:</u>

The table below show the RMSE of from only BF matching, after RANSAC, after applying epipolar constraint by only allow feature on same y value to match, and applying both RANSAC and epipolar constraint.

	RMSE			
Training Image #	Unprocess (No RANSAC or Epipolar Constraint)	With RANSAC	With Epipolar Constraint	Both RANSAC and Epipolar Constraint
1	6.8661	2.0173	3.4719	1.9024
2	5.3133	1.2645	1.0196	1.0940
3	17.1990	1.9847	10.0108	2.4928
4	22.7829	4.9644	5.0450	5.1307
5	15.8060	12.4421	12.5660	11.6877
6	8.4690	3.0064	4.2324	3.0064
7	21.6017	6.5186	13.0214	6.5186
8	41.0740	12.7653	31.4057	13.1967
9	14.2302	6.3005	10.1983	1.1529
10	8.0295	4.0247	6.8845	4.0247
Total	161.3717	55.2885	97.8557	50.2068

After applying outlier rejection such as RANSAC, the RMSE significantly become lower. Comparing the result from applying RANSAC and epipolar constraint, RANSAC perform much better because it remove the match which have incorrect horizontal transformation. The result from apply both doesn’t decrease RMSE by much because most of the incorrect matching has been remove by RANSAC.