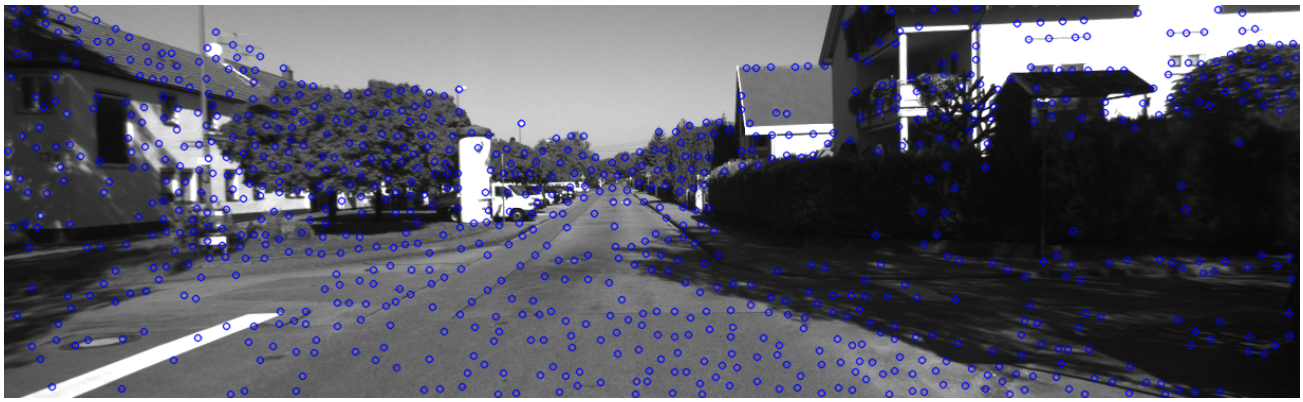
Ehud Gordon 203861422

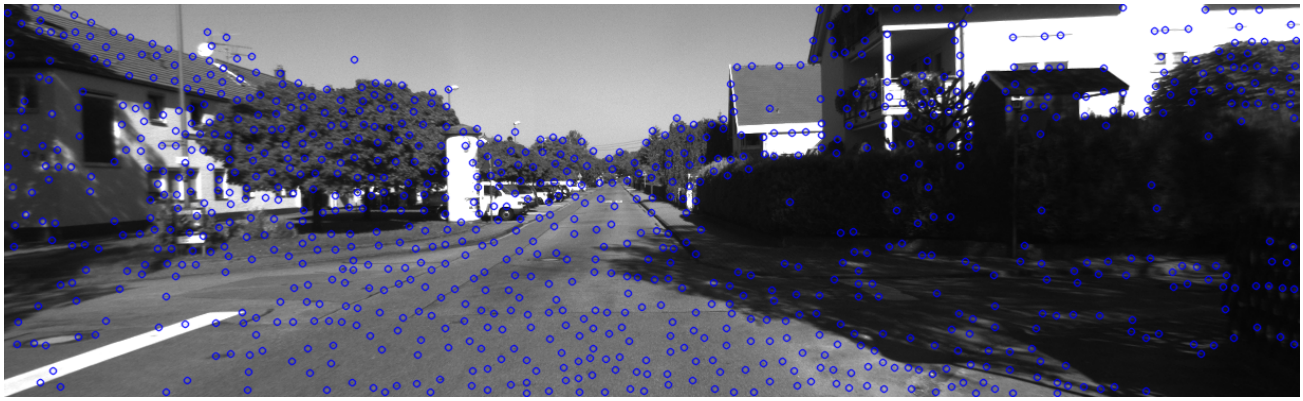
NAV Stage 1

https://github.com/ehud-gordon/VAN_ex

I have in main.py method named ex1_main()

Q1.1





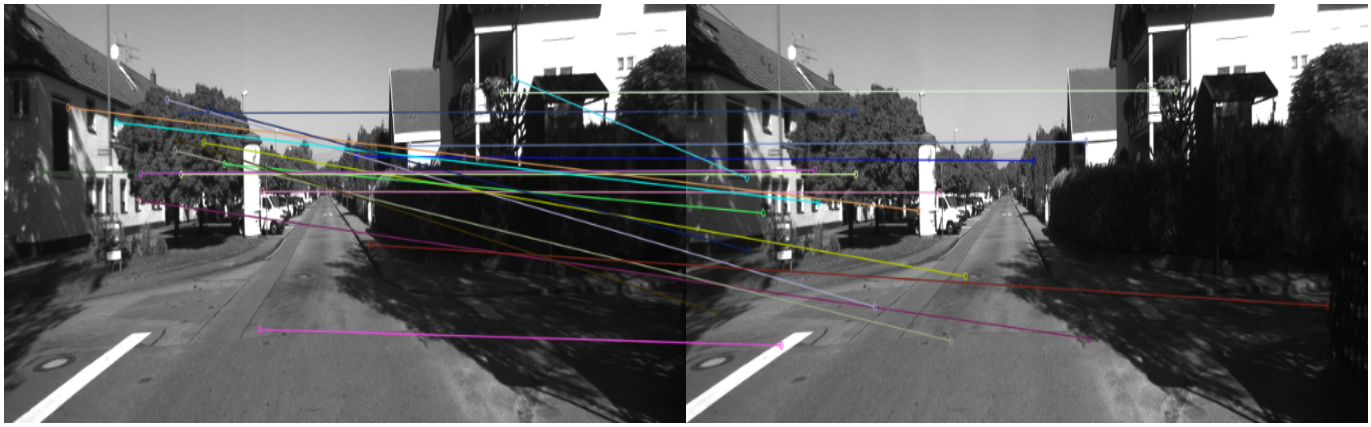
I've used FAST keypoint detector + scc algorithm to distribute the keypoints, since using ORB gave me keypoints that weren't homogeneously distributed across the image. Also FAST is fast. I've then used BRIEF descriptor, because it's fast and I saw it somewhere described with SLAM.

Q1.2

I've used BRIEF descriptor. Here's an example of descriptors of the closest matching key-points, we can see how similar they are:

```
array([ 94, 199, 185, 151, 226,  93,  57,  85, 111,   4,  46, 125,  16,
       122, 240, 107,  19,   5,  97, 164, 170, 215,  53,  66, 251, 146, 26, 213,  10,  64,  31, 168],
dtype=uint8)
array([ 94, 199, 185, 151, 224,  93,  57,  85, 111,   4,  46, 125,  16,
       122, 240, 107,  19,   5,  97, 164, 170, 215,  53,  66, 251, 146, 26, 213,  10,  64,  31, 168],
dtype=uint8)
```

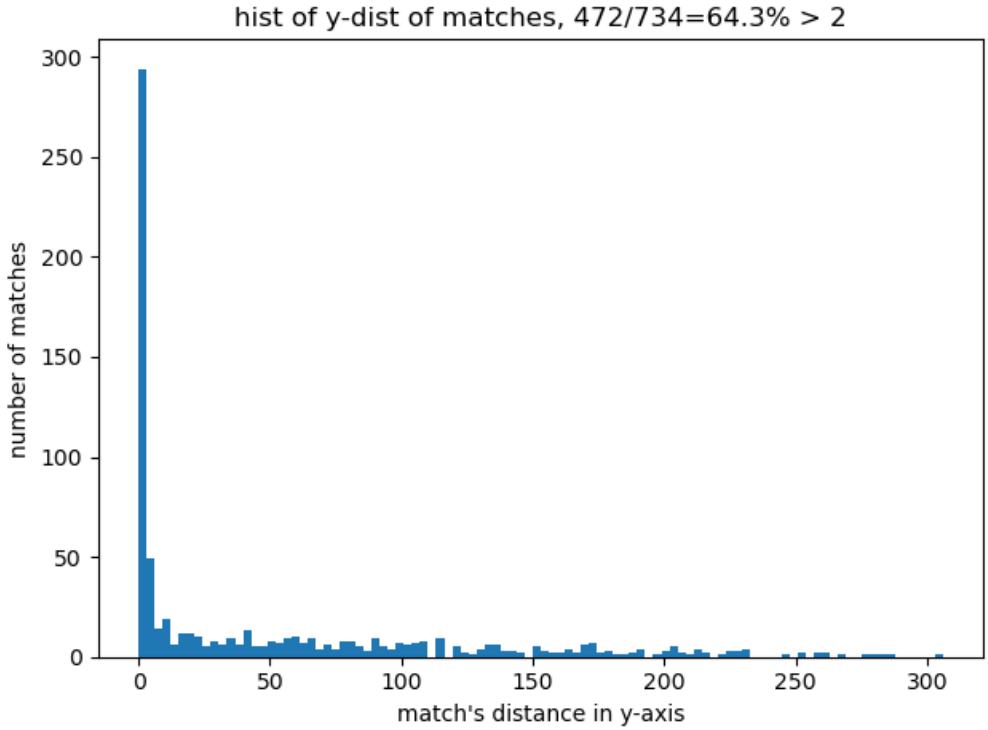
Q1.3



I've used FLANN based matcher, because It's quick, and we're about to do a lot of matching throughout SLAM. The distance is computed based on HAMMING_NORM. Before filtering, the matches are truly awful.

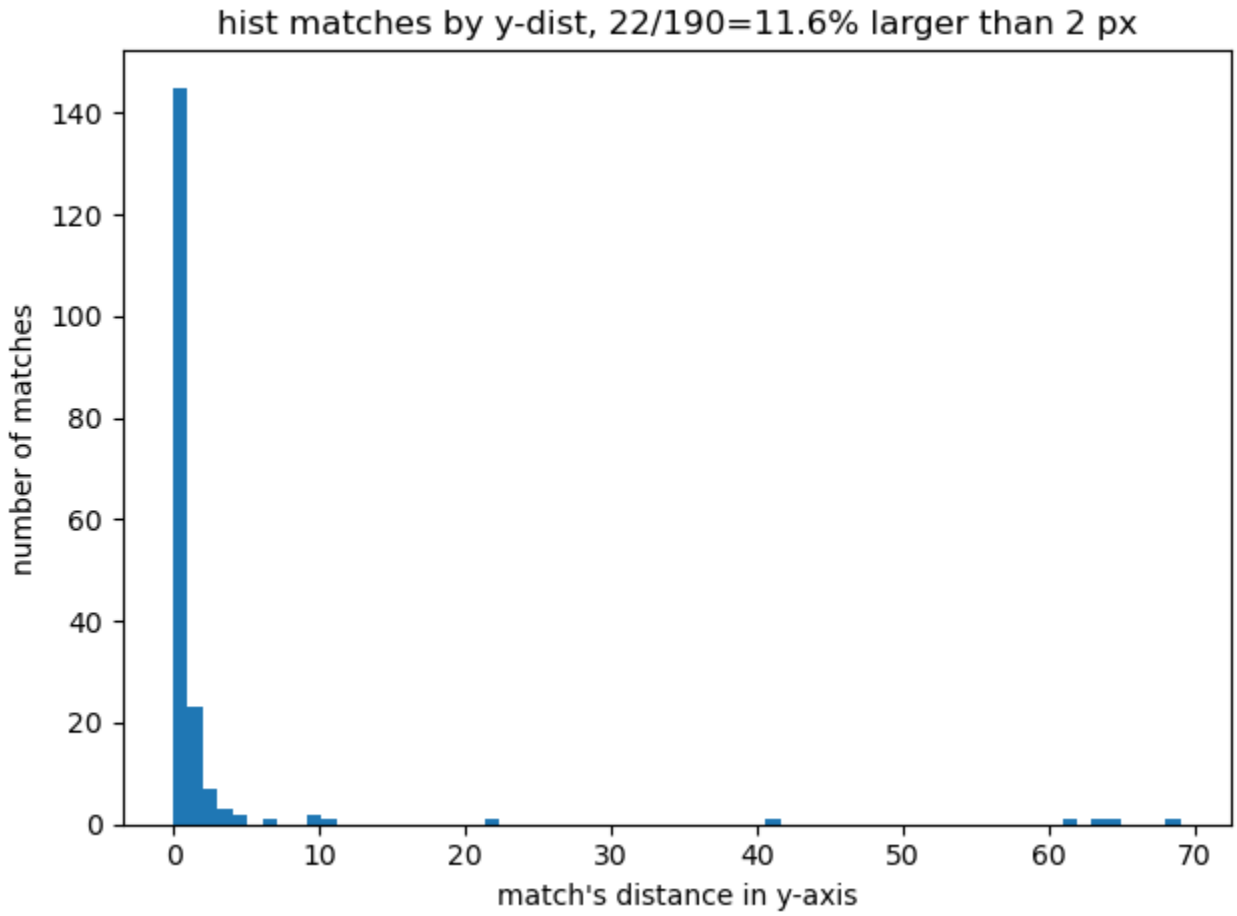
Q1.4

All of the correct matches will be of keypoints that're roughly on the same line in the image. This is because the rectified stereo images are rectified as if the two cameras were on the same plane when the images were taken (and also at the same time), and hence inliers must lie on the same scanline.



Without significance filtering, 734 matches found, 472/734 = 64.3 % of them have y_dist > 2 pixels.

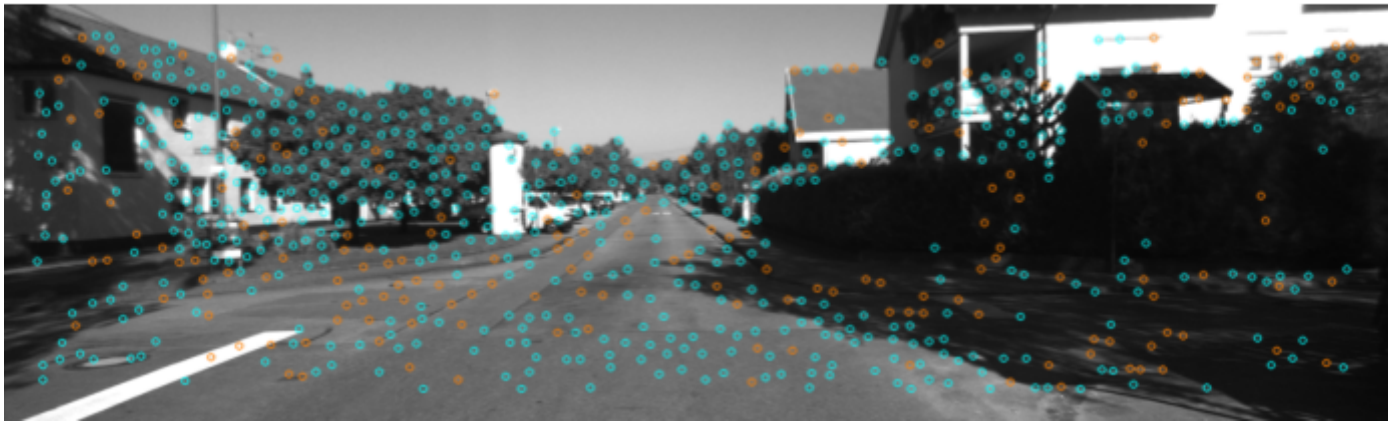
Q1.5



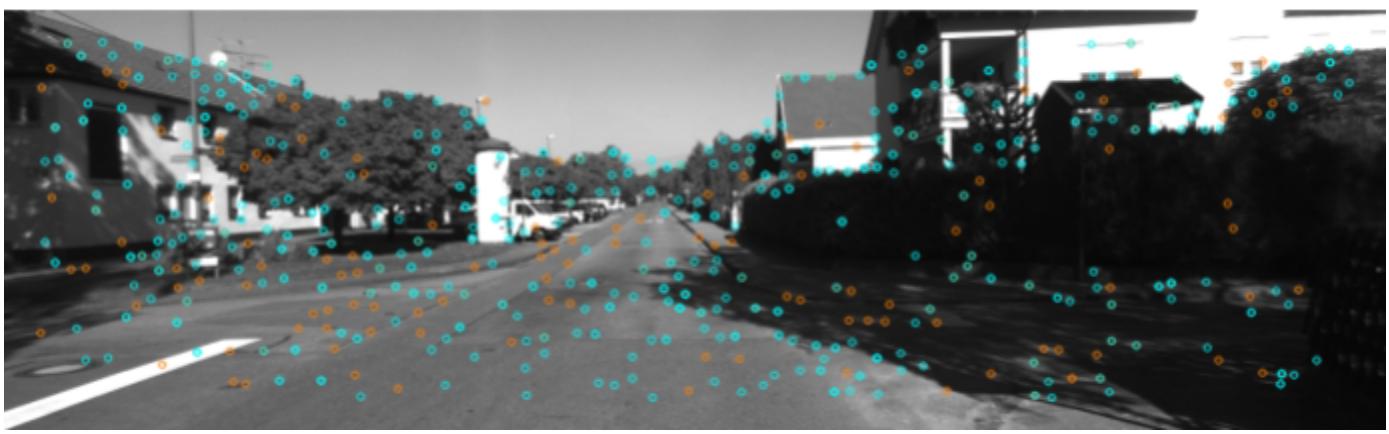
With lowe's ratio test, only 190 (out of 784) matches passed. Of those, 22/190=11.6% have y_dist > 2 pixels.

Q1.6

Left image Inliers (Orange) and Outliers (Cyan) after sig+stereo



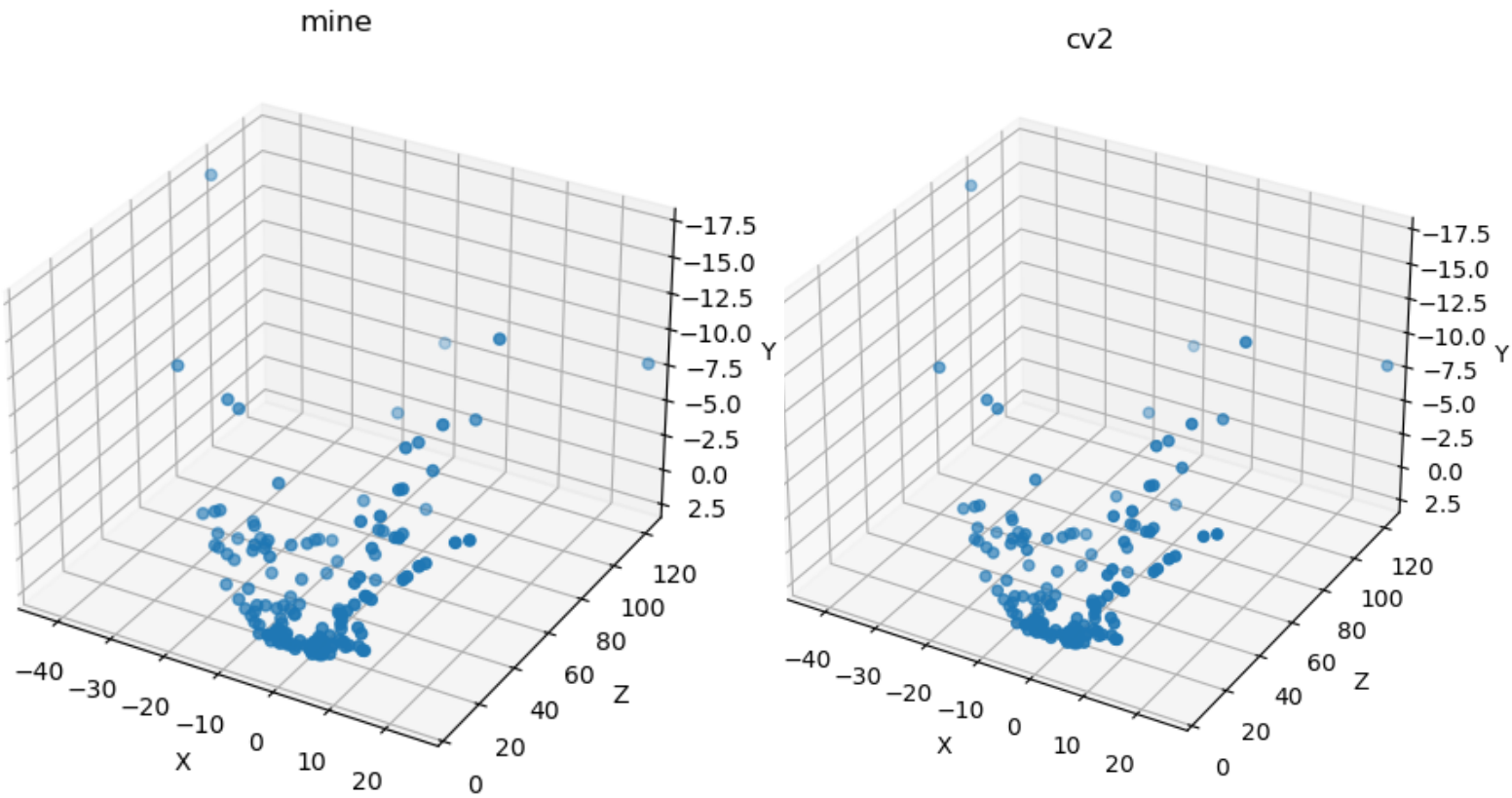
Right image Inliers (Orange) and Outliers (Cyan) after sig+stereo



Using the stereo rejection policy, erroneous matches will be only due to keypoints on the same line that have similar descriptors. Thus the rate of erroneous matches will decrease.

Before stereo rejection (but with significance), I got 190 matches. Using Stereo constraint, I've rejected 22 matches
With the assumption that the Y-coordinate of the erroneous matches is distributed uniformly, this means that the per-row, we have $\frac{22}{370-4} = 0.06$ stereo-incorrect matches. This means that after our stereo-rejection, since we've excluded 4 rows, we'll have $0.06 \ * 4 = 0.24$ wrong stereo-incorrect matches. This means that out of the final $190 - 22 = 168$ matches remaining after stereo+sig rejection, $\frac{0.24}{168} = 0.14\%$ would be stereo-incorrect matches.

Q1.7



The axes are not a mistake (I hope). Look at the [KITTI setup](#).
They're the same. If you look at my code, I've written a function `vis_triangulation()` that prints circles on the original image, corresponding to matched points, along with their triangulated 3D location. I get the same results for both cv2 and my triangulation. In fact, the iterative part of LS wasn't really necessary.

Q1.8

I've noticed some incorrect locations. Specifically, some values are impossible (such as negative z-values) and some are unlikely (extremely large z-values / y-values / x-values). I've filtered all of these out using common sense, but I'm sure there's a way to eliminate those.
I've several ideas on how to eliminate incorrect locations:

1) There must be a way to reconstruct the original camera matrix from these 3D points and the matches. For any group of points, we can use something like RANSAC to estimate camera matrix, and then compare it to the true camera matrix - and thuse remove outliers.
2) Some points lie on a plane (a building side for example) that is parallel to us, and thus should have the same z-axis. When one of these points have a vastly different z-axis than the others, it's probable to say that this is an incorrect location.