Vision Aided Navigation 2024 - Exercise 2

Amitai Ovadia 312244254

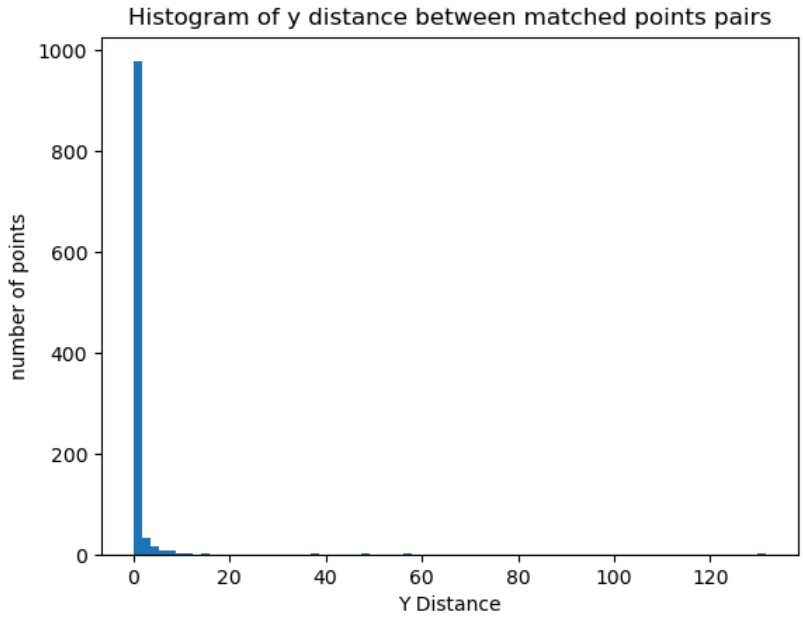
https://github.com/AmitaiOvadia/SLAMProject/tree/45060eaae7ce7805d8d3f38127479b003ef0df38/VAN_ex/code/ex2

And also

https://github.com/AmitaiOvadia/SLAMProject/blob/45060eaae7ce7805d8d3f38127479b003ef0df38/VAN_ex/code/utils

(2.1) We are working with a pair of rectified stereo images.

- In rectified stereo images, the camera arrangement is such that they are positioned parallel to each other, facing the same direction (sharing x and z axis) and only have offset in the y axis.
  Also their internal camera parameters are the same.
  Because of that, the epipolar lines of one camera with respect to the other are parallel to the y axis and we will expect corresponding points to have the same y coordinates.
- Create a histogram of the deviations from this pattern for all the matches:

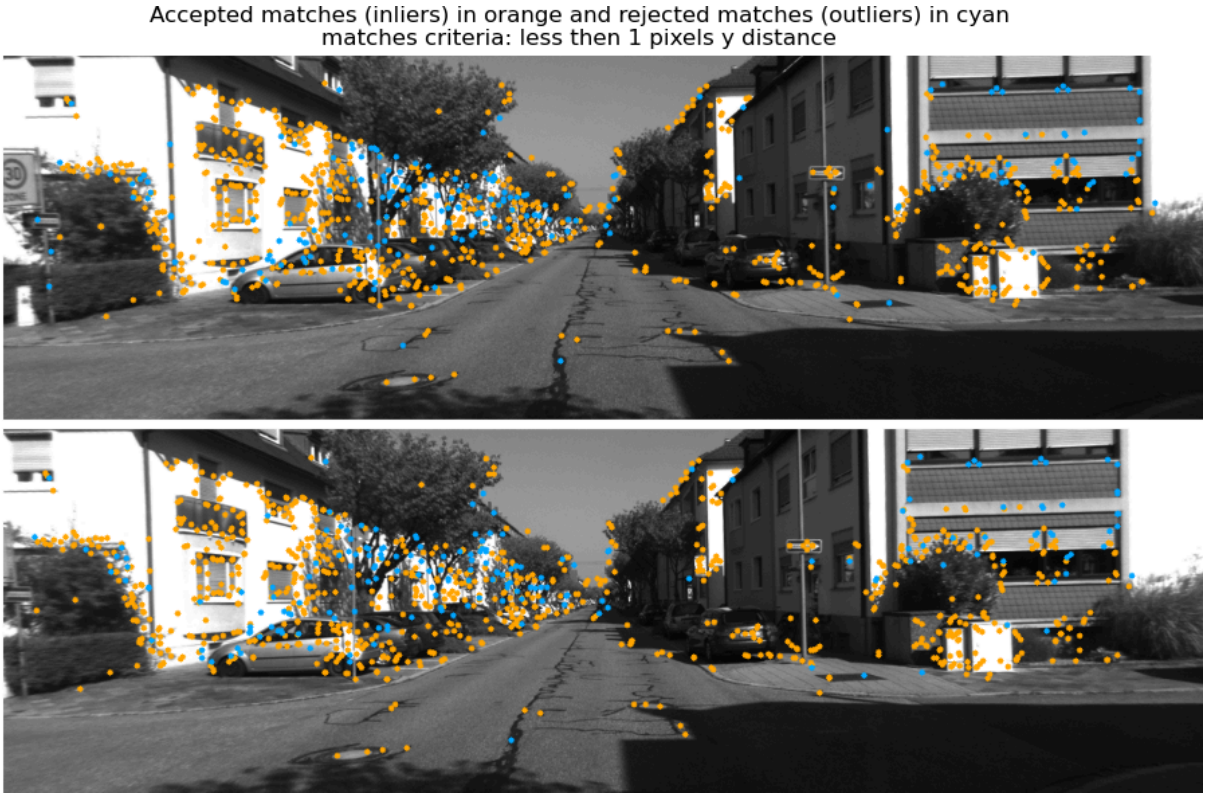


- Print the percentage of matches that deviate by more than 2 pixels.

```
the percentage of matches that deviate by more than 2 is 8.75%
```

(2.2) Use the rectified stereo pattern to reject matches:

- Present all the resulting matches as dots on the image pair. Accepted matches (inliers) in orange and rejected matches (outliers) in cyan.
  I used a more strict threshold of **1 pixel** distance in y axis between corresponding points:



- How many matches were discarded?

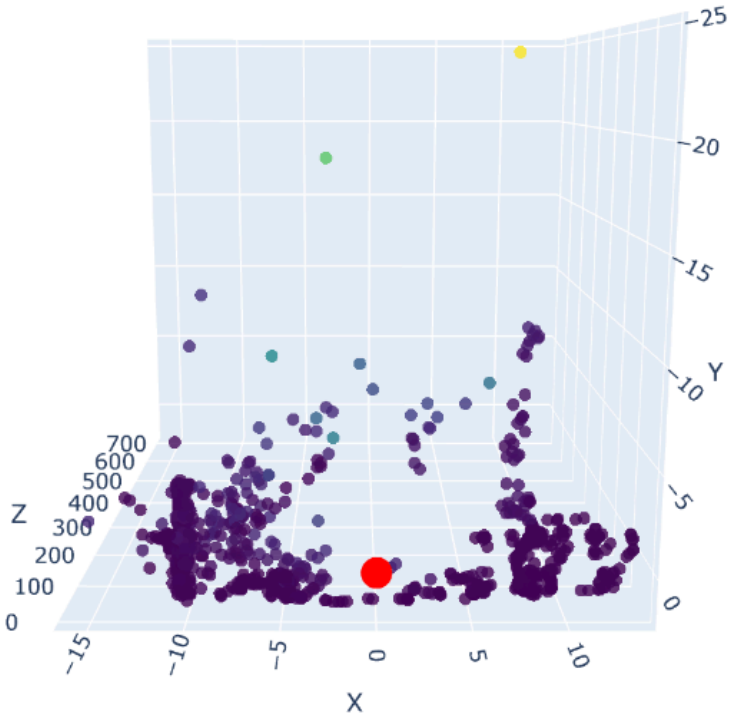
```
percentage of discarded matches: 18.32
```

- Using **1 pixel** difference in y axis as a rejection policy:
  If erroneous matches are distributed uniformly across the image, then their y coordinate is distributed y ~ uniform(0, IMAGE_HEIGHT)
  So given a point in the left image (x0, y0) then the chances of an erroneous match not to be thrown out is 2 / IMAGE_HEIGHT (because it's 1 pixel in each direction)
- This assumption is unrealistic because the features tend to be concentrated on parallel lines like in this example here:

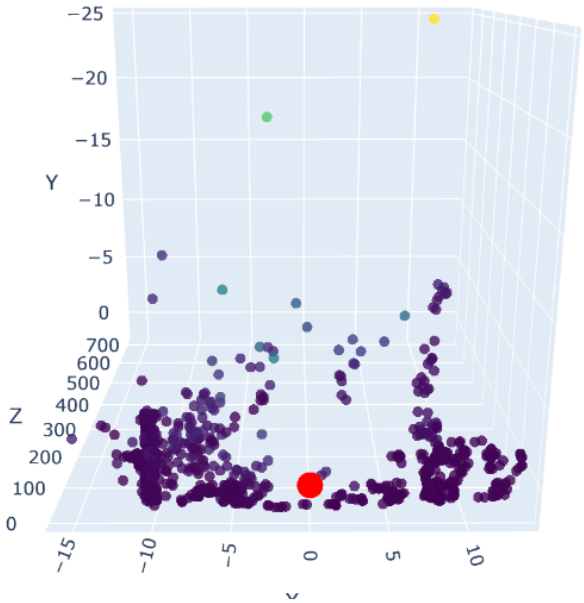
  And there are a lot of repetitive patterns as well that can result in points being matched using the AKAZE algorithm for example, that are also on the same line that is parallel to the camera's y axis, but are actually bad matches. So in my opinion the actual number of accepted erroneous matches is much higher than then this theoretical ratio.

(2.3) Present a 3D plot of the calculated 3D points.

- The red dot is the origin (camera center) at (0,0,0)



- Repeat the triangulation using ‘cv2.triangulatePoints’ and compare the results.
    - Display the point cloud obtained from opencv and print the median distance between the corresponding 3d points: Basically the same.



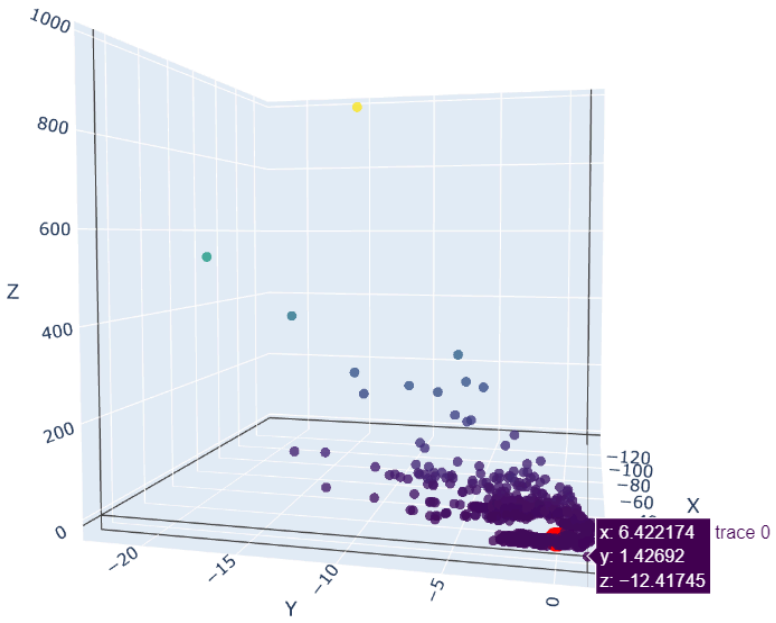

```
The median distance between the 3D points triangulated using
OpenCV function and my function is 8.772869659673792e-14 meters
```

One big difference was the calculation time:
I tried 3 different triangulation approaches using the `check_times` function

```
Time for (1) naive implementation: 0.018716 seconds
Time for (2) no loops implementation: 0.003145 seconds
Time for (3) OpenCV implementation: 0.001034 seconds
```

The naive implementation (1) means that I iterated over each pair of points and calculated the SVD separately.
The 'no loops' implementation (2) meant that the SVD was calculated over a (N, 4, 4) matrix corresponding to N points pairs.
And the opencv implementation (3) is the built in opencv function.
It worth noting that (2) was around 6 times faster than (1), and that (3) is 3 times faster than (2)

(2.4) Look at the matches and 3D points, can you spot any 3D points that have an obviously erroneous location?
Here this point has a negative z coordinate which cannot be true because the cameras are facing forward parallel with the z axis and located in (0,0,0)



- Given a possible match were 1 corresponds to the left camera and 2 corresponds to the right camera (x1, y1) (x2, y2),
  For a good match We get x1 > x2, but if x2 > x1 it leads to the point being calculated behind the camera (negative z) instead of in front of it.
- Can you think of a relevant criterion for outlier removal?
  - Yes, for example checking that indeed x1 > x2.