Vision Aided Navigation 2022 - Exercise 2

Prefix:

In **exercise 1** we got familiar with the feature handling abilities of OpenCV: detect, extract and match. We explored a common outlier rejection policy and realized its limitations.

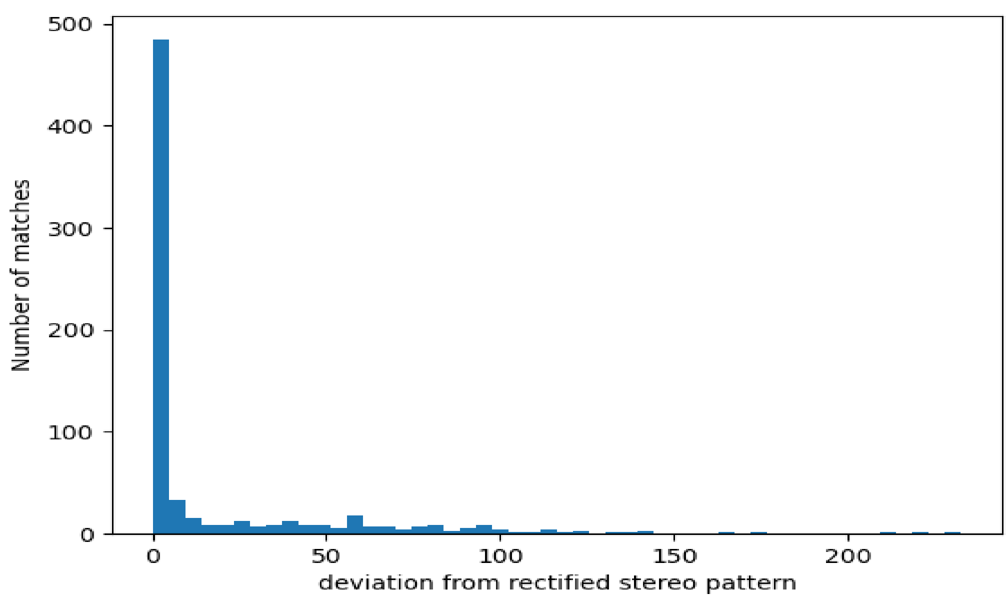
In this exercise we explore a geometric outlier rejection policy and use the stereo matches for triangulation to produce a 3D point cloud.

As in exercise 1, we are working with matches between the 1st stereo pair:
Left_0: VAN_ex\dataset\sequences\00\image_0\000000.png
Right_0: VAN_ex\dataset\sequences\00\image_1\000000.png
We work with all the matches (i.e., without using the significance test)

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

- Explain the special pattern of correct matches on such images. What is the cause of this pattern?
- 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.

```
Useful code: matplotlib.pyplot.hist
```



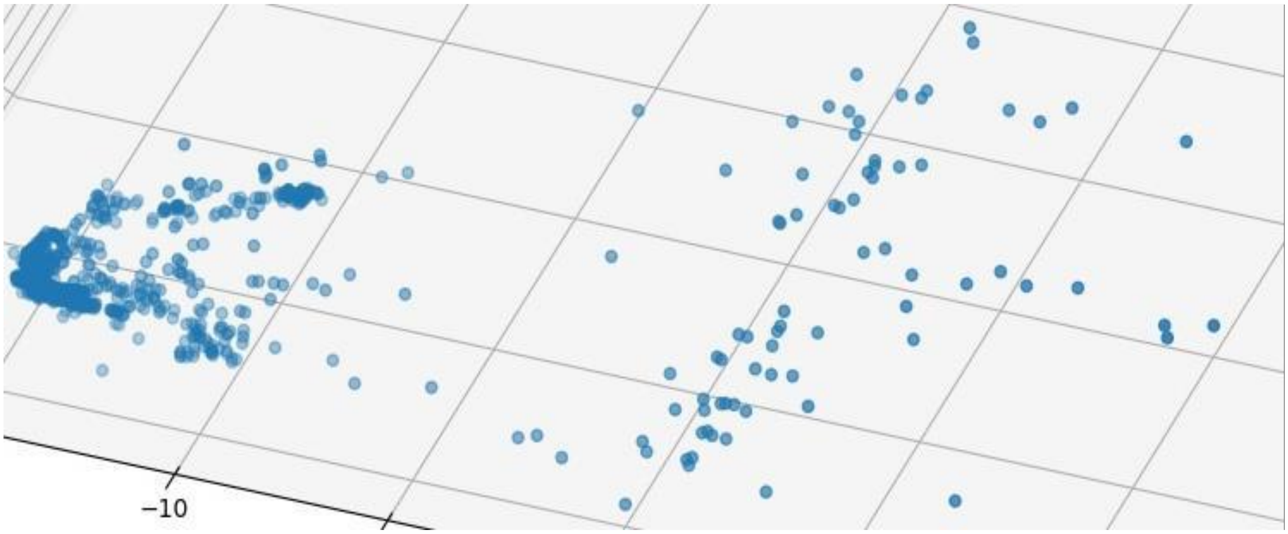
2.2 Use the rectified stereo pattern to reject matches.

- Present all the resulting matches as dots on the image pair. Accepted matches (inliers) in orange and rejected matches (outliers) in cyan.
- How many matches were discarded?
- Assuming the Y-coordinate of erroneous matches is distributed uniformly across the image, what ratio of matches would you expect to be wrong with this rejection policy? (i.e., how many are still wrong even after the rejection?)
- Is this assumption (uniform distribution along the Y-axis) realistic?
  Would you expect the actual number to be higher or lower? Why?

Matches: Inliers (orange), outliers (cyan)

2.3 Read the relative camera matrices of the stereo cameras from ‘calib.txt’.
Use the matches and the camera matrices to define and solve a linear least squares triangulation problem. Do **not** use the opencv triangulation function.

- Present a 3D plot of the calculated 3D points.
- Repeat the triangulation using ‘cv2.triangulatePoints’ and compare the results.



Run this process (matching and triangulation) over a few pairs of images.

- Look at the matches and 3D points, can you spot any 3D points that have an obviously erroneous location?
- What in your opinion is the reason for the error?
- Can you think of a relevant criteria for outlier removal?

```
Useful code:
mpl_toolkits.mplot3d.Axes3D.scatter, cv2.triangulatePoints
```

```
def read_cameras():
  with open(DATA_PATH + 'calib.txt') as f:
    l1 = f.readline().split()[1:]      # skip first token
    l2 = f.readline().split()[1:]      # skip first token
  l1 = [float(i) for i in l1]
  m1 = np.array(l1).reshape(3, 4)
  l2 = [float(i) for i in l2]
  m2 = np.array(l2).reshape(3, 4)
  k = m1[:, :3]
  m1 = np.linalg.inv(k) @ m1
  m2 = np.linalg.inv(k) @ m2
  return k, m1, m2
```