Vision Aided Navigation 2023 - Exercise 1

Prefix:

Over the semester we will gradually develop a large-scale project that employs a suite of algorithms to provide local visual odometry and global trajectory estimation. Simply put, we will find the vehicle locations using only its cameras.

The system will run on the [KITTI](http://www.cvlibs.net/datasets/kitti/) stereo dataset and will tackle these challenges using image processing, image understanding and 3D optimization techniques to estimate the vehicle path and the geometric structure of the world.

In this exercise we will develop the feature-tracking system, which is a key-component in any vision-based navigation system. Basically, the feature-tracking system receives as input a series of images (or, in our case, stereo pairs) and outputs trajectories of points in the image, where each track represents the different pixel locations of a certain object in the scene.

Part 0

Open a new folder named "VAN_ex". All future paths will be relative to it.
Open the following subfolders:

VAN_ex/dataset
VAN_ex/code
VAN_ex/docs

We will use data from the KITTI benchmark http://www.cvlibs.net/datasets/kitti/, specifically a route within visual odometry sequence #5.
Download it from:
https://drive.google.com/file/d/1CFWOJuKV6qrkry3guHNdj8Ipb7XaFRy_/view?usp=sharing
Into the path: *VAN_ex/dataset/*
And unzip it there.

First, we'll examine the first stereo pair:

Left_0: VAN_ex\dataset\sequences\05\image_0\000000.png
Right_0: VAN_ex\dataset\sequences\05\image_1\000000.png

Add the code for the exercises to a GitHub repository with permissions to the course teachers.
Submit a pdf file with a link to the repository and the answers.

Part 1

1.1 Detect and extract at least 500 key-points on each image of the first stereo pair. You can use any popular feature detection method.

- Present the key-points pixel locations on both images.

```
Useful code:
cv2.ORB_create / cv2.AKAZE_create / cv2.SIFT_create , and many more…
detectAndCompute
```

```
DATA_PATH = r'...\VAN_ex\dataset\sequences\00\\'
def read_images(idx):
  img_name = '{:06d}.png'.format(idx)
  img1 = cv2.imread(DATA_PATH+'image_0\\'+img_name, 0)
  img2 = cv2.imread(DATA_PATH+'image_1\\'+img_name, 0)
  return img1, img2
```

1.2 Calculate feature-descriptors for each key-point in both key-points lists. You can use any popular feature-descriptor method.

- Print the descriptors of the two first features.

1.3 Match the two descriptors list: find, for each descriptor in the left image, the closest feature in the right image. The distance function should not consider the pixel location, but only the feature descriptor. You can use any popular distance function.

- Present 20 random matches as lines connecting the key-point pixel location on the images pair. There may be some mismatches, it's okay for now.

```
Useful code: cv2.BFMatcher, cv2.drawMatches, pyplot.imshow
```

1.4 Use significance test[1] to reject matches

- generate an output with 20 of the resulting matches (as in sections 1.3).
- What ratio value did you use?
- How many matches were discarded?
- Present a **correct** match (as a dot on each image) that failed the significance test. (If you cannot find such match, strengthen the significance test (how?) until you find one.)

[1] AKA distance ratio test, see:
https://en.wikipedia.org/wiki/Scale-invariant_feature_transform#Feature_matching_and_indexing