Vision Aided Navigation - Exercise 2

Prefix:

In **exercise 1** we got familiar with the feature handling abilities of OpenCV: detect, extract and match. We triangulated the resulting matches into a 3D point cloud. We also found that despite our best efforts to avoid mismatches, there are still erroneous 3D points in our point cloud.

In this exercise we will move forward in time to the next stereo pair and match the left image to the previous left image and run PnP using RANSAC iterations to estimate the relative motion. We will also use the extra information - we now have two stereo pairs - to reject (almost all of) the remaining outliers.

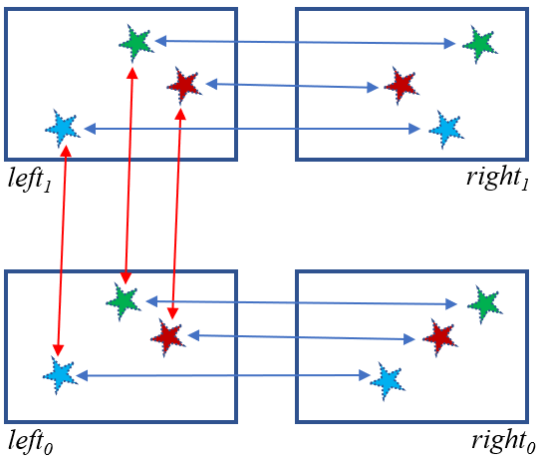
An important part of the exercise is implementing a suitable database for the features. Correct Bookkeeping will be important in future stages when we will build the Bundle Adjustment / Loop Closure optimization.

2.1 Use the code from exercise 1 to create a point cloud for the next images pair (i.e. match features, remove outliers and triangulate):

$left_1$: VAN_ex\data\dataset05\sequences\05\image_0\000001.png
$right_1$: VAN_ex\data\dataset05\sequences\05\image_1\000001.png

We now have two 3D point clouds, one for pair 0 (from exercise 1) and one for pair 1.

2.2 Match features between the two left images ($left_0$ and $left_1$) (code from exercise 1 steps 1.1-1.5)



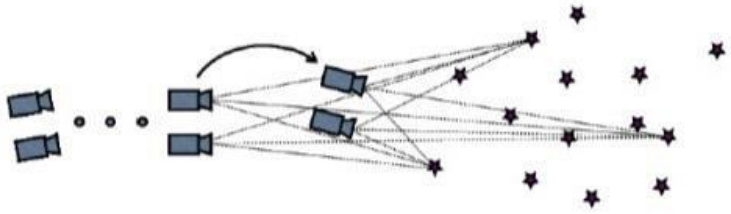
Note that this is not a stereo pair, and therefore not rectified, so we skip step 1.6

2.3 Choose 4 key-points that were matched on all four images. This means we have both 3D locations from the triangulation of pair 0 and matching pixel locations on pair 1. Apply the PNP algorithm between the point cloud and the matching pixel locations on $left_1$ to calculate the

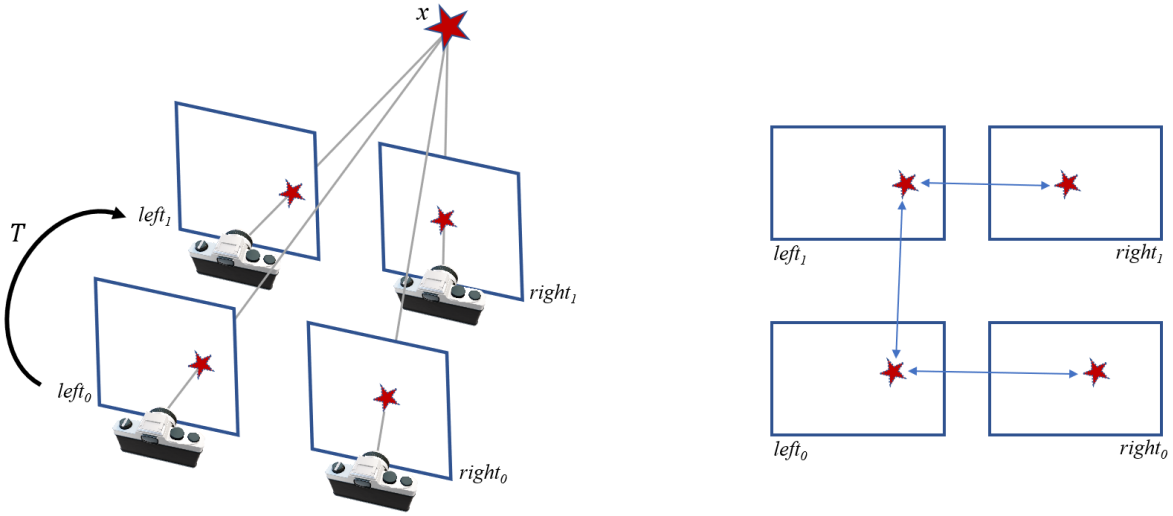
extrinsic camera matrix $[R|t]$ of $left_1$, with R a 3×3 rotation matrix and t a 3×1 translation vector.

- Describe how to define from $[R|t]$ a transformation T that transforms from $left_0$ coordinates to $left_1$ coordinates.
- If camera A has extrinsic matrix $[I|0]$, transformation $T_{A\to B}(x) = R_1 x + t_1$ transforms from the coordinates of A to the coordinates of camera B and transformation $T_{B\to C}(x) = R_2 x + t_2$ transforms from the coordinates of B to the coordinates of camera C, express transformation $T_{A\to C}$ and the extrinsic matrix of C using R_1, R_2, t_1, t_2.

- For a camera with extrinsic matrix $[R|t]$, what is the location of the camera in the global coordinate system?
  Hint: if the location is $(x\ y\ z)^T$, what would we expect the result of $[R|t](x\ y\ z\ 1)^T$ to be?
- Plot the relative position of the four cameras (from above).



2.4 For each 3D point x that was matched to $left_1$ we have four associated pixel locations (on $left_0$, $right_0$, $left_1$, $right_1$) if these are correct and the transformation T we calculated is also correct, we can expect the projection of x to fall close to these pixel locations on all four images.



We consider a point x that projects close to the matched pixel locations in all four images a supporter of transformation T.
Use a distance threshold of 2 pixels to recognize the supporters.

- Plot on images $left_0$ and $left_1$ the matches, with supporters in different color.

2.5 Use a RANSAC framework, with PNP as the inner model, to find the 4 points that maximize the number of supporters. We call this maximal group the 'inliers'.
Note: Implement RANSAC yourself, do not use 'cv2.solvePnPRansac'
Refine the resulting transformation by calculating transformation T for all the inliers.

Use the resulting T to transform the first point cloud (that belongs to pair 0)

- Plot the two 3D point clouds from above. Use different colors for the two clouds.
- Plot on images $left_0$ and $left_1$ the inliers and outliers in different colors.

```
Useful code: numpy.random.randint, cv2.solvePnP
```

```
def rodriguez_to_mat(rvec, tvec):
  rot, _ = cv2.Rodrigues(rvec)
  return numpy.hstack((rot, tvec))
```

2.6 We call a 3D landmark that was matched across multiple pairs of stereo images a 'track'. In the previous sections we recognized tracks of length 2 (matched over two pairs of images), but we hope to lengthen the tracks by matching features over more pairs.
Implement a suitable database for the tracks.

- Every track should have a unique id, we will refer to it as TrackId.
- Every image stereo pair should have a unique id, we will refer to it as CameraId.
- Implement a function that returns all the TracksIds that appear on a given CameraId.
- Implement a function that for a given (CameraId, TrackId) pair returns:
  - Feature location on the left camera (x,y)
  - Feature location on the right camera (x,y)
- Implement an ability to extend the database with new tracks on a new image pair as we match new stereo pair to the previous one.
- Implement functions to serialize the database to a file and read it from a file.

2.7 Repeat steps 2.1-2.5 for the whole movie in: *VAN_ex\data\dataset05\sequences\05*
for all the images. Fill the database you created with the resulting tracking.

- How long did the tracking take?

Keep track of all the relative transformations T and produce an estimation for the extrinsic matrix of all the left cameras.

- Plot a trajectory of all the left camera locations in the coordinates of camera 0, as viewed from above.
- Read the ground-truth extrinsic matrices and add to the plot the ground truth locations in a different color.
  - The ground truth matrices are in *VAN_ex\data\dataset05\poses\05.txt*
  - Each line has 12 numbers that represent the extrinsic matrix of the corresponding left camera in row-major order.

- Important note: The ground truth matrices in *05.txt* are from camera coordinates to world coordinates, NOT the other way around like we learned in class. i.e. given matrix M and point x_c in camera coordinates the expression $M\begin{pmatrix} x_c \\ 1 \end{pmatrix}$ gives the point in world coordinates.