Exercise 11: Visual Odometry

02504 Computer vision

Morten R. Hannemose, mohan@dtu.dk, DTU Compute

April 19, 2023

This is a longer exercise, and you have both week 11 and 12 to work on it. The second part of the exercise is again more free where you have options for improving your algorithm.

Download Glyp.zip which contains the images and camera matrix you will need for the exercise.

Load the camera matrix with
`K = np.loadtxt('K.txt')`

Load the first three images (`000001.png`, `000002.png`, `000003.png`) into Python as `im0`, `im1` and `im2`.

Exercise 11.1

Find SIFT keypoints (`kp0`, `kp1`, `kp2`) in all three images and compute their corresponding descriptors (`des0`, `des1`, `des2`). For speed reasons, you can limit the number of SIFT features to 2000. Convert the features to numpy arrays of 2D points
`kp = np.array([k.pt for k in kp])`

Match the SIFT features between `im0` and `im1` (`matches01`), and between `im1` and `im2` (`matches12`). Convert the matches to numpy arrays of the indices
`matches = np.array([(m.queryIdx, m.trainIdx) for m in matches])`.

Exercise 11.2

Estimate the essential matrix between `im0` and `im1` with RANSAC. You can use the OpenCV function `cv2.findEssentialMat` to do this.

Decompose the essential matrix and find the correct relative pose (`R1`, `t1`). For this we can again use an OpenCV function namely `cv2.recoverPose`.

The `mask` returned by `cv2.recoverPose` indicates which of the matches are inliers, that lie in front of both cameras. Remove the matches that are not inliers from `matches01`, so that only contains the inliers.

Exercise 11.3

Use matches01 and matches12 and find the subset of matches such that we can match features all the way from image 0 to image 2. In other words, create three lists such that points0[i], points1[i], and points2[i] are the 2D locations of the same point in the corresponding images. For this you can use

```
_, idx01, idx12 = np.intersect1d(matches01[:,1], matches12[:,0], return_indices=True)
```

Exercise 11.4

For the points that have been tracked through all three images, use the 2D positions in image 0 and 1 to triangulate the points in 3D (Q). Using the 2D positions in image 2, estimate the pose of image 2 with RANSAC. Use cv2.solvePnPRansac to do this. As the lens distortion is already corrected, you can set distCoeffs = np.zeros(5).

Visualize the 3D points that are also inliers for solvePnPRansac.

```
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.scatter(*Q[inliers.flatten()])
```

Also plot the position of the cameras. Recall that the position of the camera is not the translation. How do you find the position?

Expand your algorithm

Now you will expand your algorithm so it can work for more than three images.

For this I suggest that you create lists of store the relevant objects for each frame (Rs, ts, kps, etc.)

Exercise 11.5

Iterate through all images in the folder, repeating the steps in Exercises 11.3 and 11.4 for the previous three images each time.

Visualize all the 3D points and camera positions. Does it look correct?

Exercise 11.6

Instead of just triangulating the 3D points from the previous two frames, use all frames that the feature has been seen in to triangulate it.

Optional exercises

Exercise 11.7

Use the calibration you did of your phone and capture your own sequence of images to try out.

Exercise 11.8

Implement bundle adjustment after your algorithm has run, where you optimize over the camera poses and 3D points in all frames at the same time. Does this improve the result?

Exercise 11.9

For a more challenging evaluation, you can try your code on a sequence from the KITTI dataset. Download KITTI09.zip that contains the images from left camera in sequence 09 in the KITTI dataset, and the corresponding ground truth camera positions and orientations.

Load the ground truth by running

```
gt = np.loadtxt('09.txt').reshape(-1, 3, 4)
```

Now, `gt` contains the ground truth rotation and position (NOT the pose) of each camera in the sequence. The translations are true to scale, so it's allowed to scale your positions to match as good as possible.