

IN2106 Practical Course – Vision-based Navigation: Exercise #4

Topic: Visual Odometry

Min-An Chao (03681062) | TUM MS Informatics | ga83fok@mytum.de

24.05.2018

1 Feature based visual odometry

Task 1 The feature points are labeled with angles, as shown in Fig. 1. The `atan()` function is used to compute the angle, along with wrap-around check for negative m_{10} and radius-to-degree conversion. (OpenCV use degree as unit for plotting.)



Figure 1: Feature points with angles labeled

Task 2 Based on the rotation angle of each feature point, we compute the rotated point based on the sample points given by `ORB_pattern`. The rotated points outside the image will be marked as "bad" points, and below shows the number of such bad points. Other good points, where we use to get the pixel values in two images, will come up of a corresponding Boolean value, labeling if pixel from image 1 is larger than another.

```
1 bad/total: 41/638
2 bad/total: 6/595
```

Task 3 By computing the nearest point in image 2 for each point in image 1, we obtain a vector `matches` stored as `vector<cv::DMatch>`. However, there will be some 2 or more matches in the vector `matches`, that share the same point in image 2 as the nearest point. We can mark such matches as conflicts. Since the matches should be exclusively one-to-one in 2 images, we delete all such conflicts inside the vector `matches`. After this we obtain what we expected, as shown in Fig. 2.

Task 4 2D-2D pose estimation can be done by OpenCV functions `findEssentialMat()` and `recoverPose()`. The results are shown below.

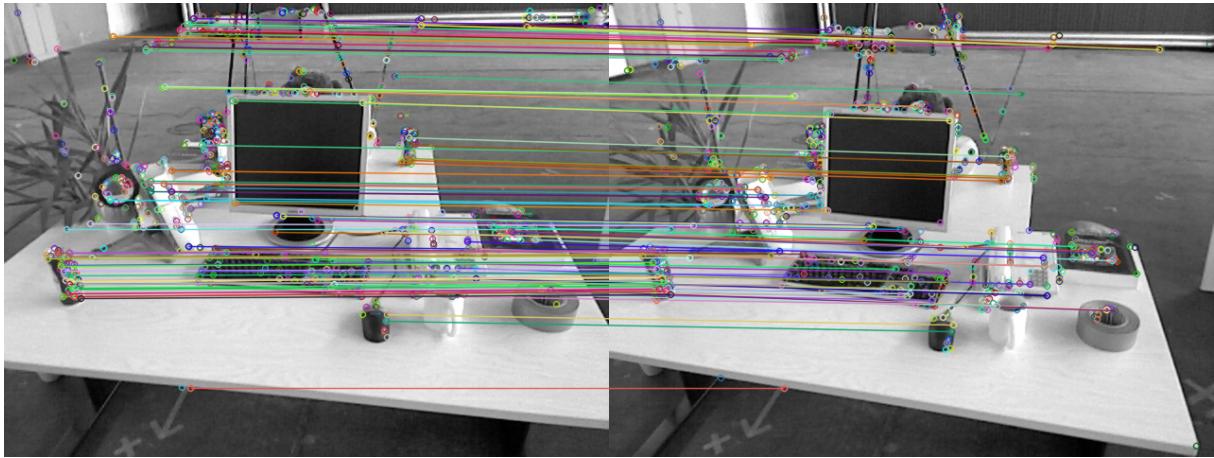


Figure 2: Matches made by brute force searching from feature points

```

1 essential_matrix is
2 [-0.01689316757952954, -0.4001360984609538, -0.05206126359774105;
3  0.382602780486801, -0.03388384497843067, 0.5922787440926547;
4  0.0145439817260907, -0.5814930982650205, -0.0149159111406146]
5 estimated motion using 2D-2D method
6 Rotation from 1 to 2:
7   0.998461 -0.0461534  0.030742
8   0.0453012  0.998585  0.0278645
9  -0.0319845 -0.0264289  0.999139
10 translation from 1 to 2:
11  -0.820848 -0.0578427  0.568211

```

After processing depth information from RGB-D camera, we obtain the 3D information of keypoints, where (u, v) is transformed to (x, y) coordinate and then applied with depth information to be $(x', y', z') = (xd, yd, d)$. With OpenCV functions `solvePnP()` and `Rodrigues()`. The results are shown below. Note the rotation matrix is similar to 2D-2D result, with translation only scaled down compared to 2D-2D.

```

1 estimated motion using 3D-2D method
2 Rotation from 1 to 2:
3   0.998263 -0.0504054  0.0304982
4   0.04982   0.998565  0.0196607
5  -0.0314454 -0.0181071  0.999341
6 translation from 1 to 2:
7  -0.11344  0.00391171  0.0685656

```

Finally, we transform keypoints from both image 1 and 2 with the same procedures to obtain their 3D information. Subtracted with mass center $\bar{\mathbf{p}}_1$ and $\bar{\mathbf{p}}_2$ for each array, we have $\mathbf{q}_1^{(i)}$ and $\mathbf{q}_2^{(i)}$, and we compute $\mathbf{W} = \sum_i \mathbf{q}_1^{(i)} \mathbf{q}_2^{(i)T}$. Then by SVD decomposition of $\mathbf{W} = \mathbf{U}\mathbf{S}\mathbf{V}^T$, we get rotation $\mathbf{R} = \mathbf{U}\mathbf{V}^T$ and $\mathbf{t} = \bar{\mathbf{p}}_1 - \mathbf{R}\bar{\mathbf{p}}_2$. The results are shown below. Perhaps due to the noise in depth information, the results do not look so similar.

```

1 estimated motion using 3D-3D method
2 Rotation from 1 to 2:
3   0.995954  0.0787748 -0.0432516
4  -0.0707982  0.984202  0.162275
5   0.0553515 -0.158556  0.985797
6 translation from 1 to 2:
7   0.116904 -0.289467  0.0741315

```

2 LK optical flow

Task 1 The Jacobian matrix is derived from error related to the motion as

$$\mathbf{J}_i = \left(-\frac{\partial I_2(x_i + \Delta x_i, y_i + \Delta y_i)}{\partial x_i} \quad -\frac{\partial I_2(x_i + \Delta x_i, y_i + \Delta y_i)}{\partial y_i} \right)^T, \quad (1)$$

along with

$$\begin{aligned} \mathbf{H} &= \sum_i \mathbf{J}_i \mathbf{J}_i^T, \\ \mathbf{b} &= \sum_i -e_i \mathbf{J}_i, \\ (\partial x_i, \partial y_i)^T &= \mathbf{H}^{-1} \mathbf{b}. \end{aligned} \quad (2)$$

Task 2 We change the optimization object in Eq. 4 to derive an inverse version:

$$\mathbf{J}_i = \left(-\frac{\partial I_1(x_i, y_i)}{\partial x_i} \quad -\frac{\partial I_1(x_i, y_i)}{\partial y_i} \right)^T. \quad (3)$$

The result of single layer optical flow is shown in Fig. 3.

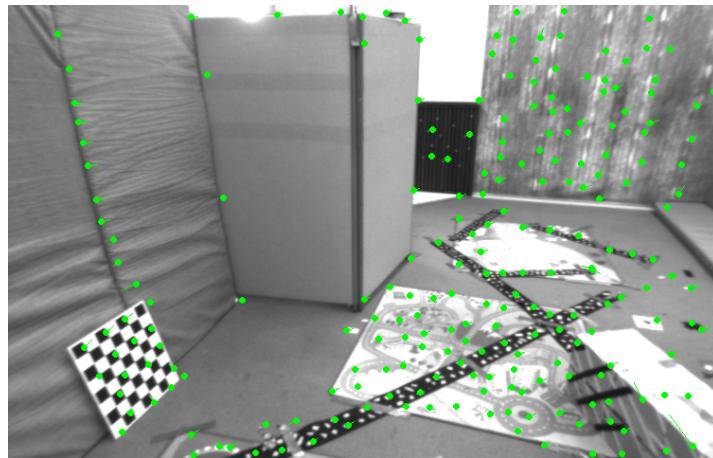
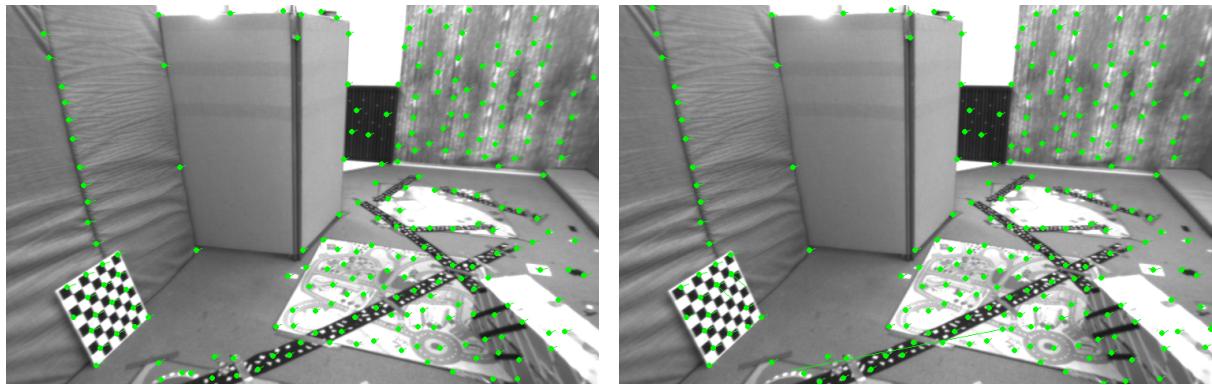


Figure 3: LK flow using single layer

Task 3 In multi-layer method, coarse-to-fine means we downsample the original image to get a coarse optical flow first, then upsample back the candidates to gradually refine our optical flow estimates. The pyramid method used in pixel level or feature point level is basically the same. Only the projection needs to be done if feature points are stored in 3D world coordinate.

The results are in Fig. 4, with multi-layer LK flow slightly better than OpenCV implementation.



(a) multi-layer LK

(b) OpenCV library

Figure 4: LK flow results

3 Direct method

Task 1 The 6×1 Jacobian matrix can be factorized as two parts, the 3D-2D projection part $2 \times 6 \mathbf{J}_\xi$, and the image pixel gradient part $2 \times 1 \mathbf{J}_p$.

$$\begin{aligned}
 \mathbf{J}_i &= \mathbf{J}_{\xi,i}^T \mathbf{J}_{p,i}, \\
 \mathbf{J}_{\xi,i} &= \begin{pmatrix} -\frac{f_x}{z_i} & 0 & \frac{f_x x_i}{z_i^2} & \frac{f_x x_i y_i}{z_i^2} & -f_x - \frac{f_x x_i^2}{z_i^2} & \frac{f_x y_i}{z_i} \\ 0 & -\frac{f_y}{z_i} & \frac{f_y y_i}{z_i^2} & f_y + \frac{f_y y_i^2}{z_i^2} & -\frac{f_y x_i y_i}{z_i^2} & -\frac{f_y x_i}{z_i} \end{pmatrix}, \\
 \mathbf{J}_{p,i} &= \left(\frac{\partial I_{\text{cur}}(u_i, v_i)}{\partial u_i} \quad \frac{\partial I_{\text{cur}}(u_i, v_i)}{\partial v_i} \right)^T, \\
 \text{where } (u_i, v_i) &= \left(\frac{x_i}{z_i} \cdot f_x + c_x, \frac{y_i}{z_i} \cdot f_y + c_y \right). \tag{4}
 \end{aligned}$$

The size of window, or patch, is chosen to be 4×4 in this task. However it also makes sense to just choose a single pixel, just with weaker ability to fight against noise and sudden illumination changes.

Task 2 The results are shown with trajectories of sample points in Fig. 5 for each frame. The pose shift portions from the first to the fifth image are shown as follows.

```

1 t1 = (-0.00153993, 0.00281099, -0.72449)^T
2 t2 = (0.0072663, -0.00159868, -1.47007)^T
3 t3 = (0.00767712, 0.00385026, -2.20899)^T
4 t4 = (0.010734, 0.00301489, -2.99649)^T
5 t5 = (0.0171474, -0.00840279, -3.78821)^T

```

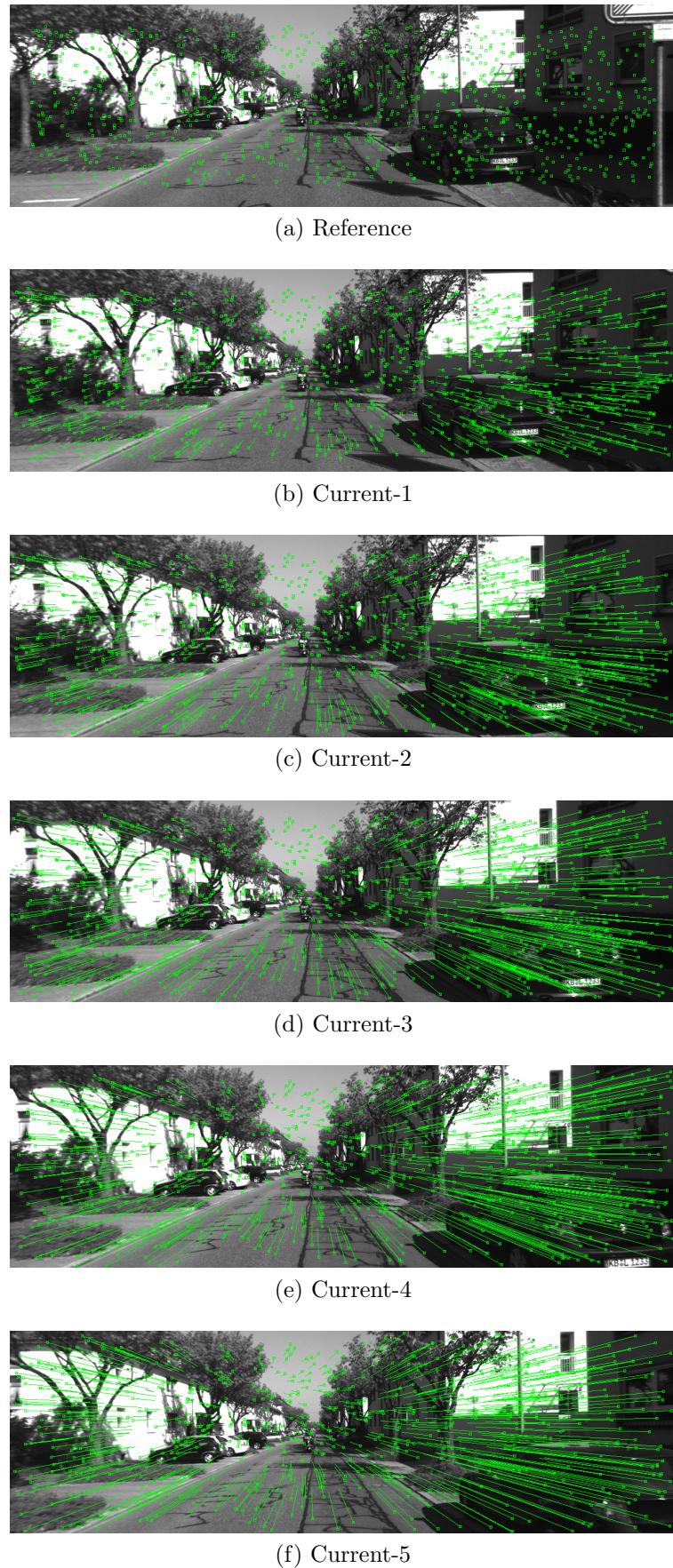


Figure 5: Results of direct method