ME41030 - 3D Robot vision Assignment 5 - Visual Odometry/SLAM

Joris Domhof

March 15th 2018

1 Introduction

This assignment is about simultaneous localization and mapping. This assignment consists of two parts:

1. Robust ego motion estimation: remove outliers in ego-motion estimation.
2. Bundle adjustment: refinement of ego-motion parameters and a set of 3d points (aka map) over a sequence of frames.

ℹ the assignment will be performed in Matlab. We assume you already have some basic knowledge of this software.

A report in pdf with the answers of this assignment should be uploaded to Brightspace. In addition to the answers to each question we ask you to provide the (commented) Matlab code that computes the entire code.

1.1 Getting help

The practicum session is the main opportunity to get help. Outside practicum hours, please use the Brightspace forum to ask any question regarding the assignment! We will not answer questions via email. This way, everybody will have access to the same feedback.

2 Assignment 1: Robust Visual Odometry

In this section,we will look into a more challenging scenario for ego motion estimation. In order to improve the results, Random sample consensus (RANSAC) will be implemented.
The `ransac_data.mat` file contains a struct with the following parameters:

- `frames(i).image_index`.prev: index to image `img_files` with index to previous frame (prev)
- `frames(i).image_index`.now: index to image `img_files` with index to current frame (now)
- `frames(i).features.prev`: matched image features in image previous image
- `frames(i).features.now`: matched image features in image current image

(a) Estimate the egomotion of the car on the new dataset. In the mat file (`ransac_data.mat`) the features can be found. Use your code of the previous assignment to estimate the trajectory. Visualize clearly the trajectory in a 3D plot.
ℹYou should not expect a perfect trajectory.
ℹCheck the comment in Matlab about the `find_transform.m` function.

(b) Explain why visual odometry does not work that well in the previous question. (maximum 4 lines).

(c) As you have seen in the previous question, the trajectory is not very impressive. That is why we are going to use RANSAC to improve the results. What is the number of times you need to run RANSAC now to obtain a success rate of (at least) 99.9% on a this data set where 50% or 10% of the points are inliers? How many rounds will you use for your RANSAC implementation on this dataset?

(d) In a RANSAC implementation, you should define whether a point is an inlier or an outlier for the fitted/estimated model. How can we determine if a point is an inlier or an outlier in this case?

(e) Implement RANSAC on this dataset in order to remove the outliers. Visualize the following two things:

- the RANSAC egomotion trajectory in 3D as well as the trajectory of question a (using all points).
- Plot two images (for the last frame) side by side and plot in both images the inliers in blue and the outliers in red.

(f) Experiment with different values of the threshold for determining the inliers and outliers. You should use values smaller than 0.5 m. Plot the trajectories for the settings you have used.

(g) In question **e**, you have plotted the inliers and outliers in a pair of images. Which points do you expect to get removed (outliers)? Which points are removed? (Use maximum 4 lines to answer this question.)

3 Assignment 2: Bundle Adjustment

RANSAC allows us to remove outliers on each frame, thereby creating a more optimal transform from frame to frame. However, all the errors made in each frame-to-frame transform will accumulate over time, resulting in drift. To combat this, we can use bundle adjustment to optimize the transform simultaneously over multiple frames. We assume that we have an initial map of 3D points. Furthermore, we have n inital poses and feature matches positions. We will perform bundle adjustment on the trajectory meaning that we perform a non-linear optimization of the reprojection error.

The `bundle_adjustement_data.mat` file contains a struct with the following parameters:

- `nr_poses`: number of frames in this sequence.
- `points3d`: estimated location of the feature points in $[x, y, z]^T$.
- `poses`: 4x4 poses matrix (orientation and position) per frame.
- `indices_features`: indices of the features seen in this pose. The indices refer to the 3D points.
- `feature_locations_px`: pixel locations of the features in the camera image.

In the literature, bundle adjustment is optimized for the resulting error in the image plane, rather than the error in 3D coordinates as we have done before. To do this, we need a function to reproject 3D points onto the image plane.

(a) Write a function `[u,v] = map2image(P,p,camera)` function which first transforms the points `p` using the pose `P`, and then projects these points onto the image plane. The third argument of this function is the camera parameters from `camera.mat`.
ℹLook into `compute_3d.m` to see how in this function the 3D points in camera reference frame are mapped to vehicle reference frame. You should consider this transformation as well in your `map2image()` function.
ℹRemember you could use `compute_3d.m` file to verify if your function works.

The bundle adjustment optimization cannot be done with a direct equation, as was the case with the frame-to-frame estimation for RANSAC, so the optimization will be done by the matlab function `fminunc`. The optimization function puts two requirements on the parameters that we want it to optimize. First, they must be supplied as a vector. Secondly, we cannot directly tell this function to directly optimize the 3×3 rotation matrix R, because that would probably result in a 3×3 matrix that no longer adheres to the constraints of a rotation matrix: $RR' = I$, all eigenvalues equal to one. Instead, we need to supply the optimization function with a minimal representation of the rotation matrix, also known as euler angles. Transforming from rotation matrices to euler angles and back can be done with `rotm2eul` and `eul2rotm`, respectively.

(b) Turn the poses (that is both a rotation and a translation) of all the frames that we will optimize for into one large vector.
ℹMake the amount of frames that are used dependent on `ba_data.nr_poses`, so you can easily vary the amount of frames that you will optimize.

(c) Write a function `e = compute_reprojection_error(X)` (which can be found in `optimize_reprojection_error.m`) that computes the total reprojection error for a given vector X that contains the poses and 3D points. The total reprojection error is equal to the sum of squared errors (SSE):

$$\epsilon_{ij} = d(K \cdot P_i \cdot p_j, x_{ij}) \tag{1}$$

$$e = \sum_{i=1}^{n} \sum_{j=1}^{m_i} \epsilon_{ij}^2 \tag{2}$$

where `d` represent the euclidean distance between image point $K \cdot P_i \cdot p_j$ and x_{ij}, so ϵ_{ij} is the error for feature point j (x_{ij}) for pose i (P_i). Furthermore, p_j is the 3D location of feature point j and matrix K is the intrinsic camera matrix. `n` is the number of poses and `m_i` is the number of visible features points for pose i.
In addition, the second output argument should be the root means squared error. Provide your root mean square error for the initial estimate of X.
ℹIf you set `ba_data.nr_poses = 5` you should expect a root mean squared error of approximately 2 pixels.

We will now try to optimize the poses over multiple frames. We do this for only a part of the trajectory, because the optimization use very computationally demanding.
(d) Use the Matlab function `fminunc` to minimize the reprojection error in the function `optimize_reprojection_error.m` Use the following optimization options for this function:
```
opts = optimoptions('fminunc', 'Display', 'iter', 'UseParallel', true,
'MaxFunctionEvaluations', 1e5);
```
The optimization will terminate prematurely, but it will be close to the optimum. Report your new reprojection error which should be lower than the reprojection error

before bundle adjustment.

$$\text{minimization term}: \quad \min_{P_i,\vec{p}_j} \sum_{i=1}^{n} \sum_{j=1}^{m_i} \epsilon_{ij}^2$$

ℹIn an optimal implementation we would supply the derivative of the function. In this case however we let Matlab estimate it for us. The algorithm uses this gradient to move to a local minimum of the error function.

(e) Plot the optimized and initial trajectory for all poses. In addition also plot the 3D points for the initial and optimized feature point positions.

(f) Plot the initial and optimized trajectory in a series of subplots: x versus time, y versus time and z versus time. For that use the `subplot(1,3,i)` with `i=1,2,3` command.