Mini project: Your own Visual Odometry pipeline!

Contents

This document describes the goal of the mini-project, the expected result, and gives you a number of guidelines regarding what to do and where to start.

1 Preliminaries

The electronic version of this document contains some additional links that you might find useful. Please consult it in addition to your (eventual) printed copy.

1.1 Additional resources

- A Youtube playlist (link in electronic version) showing our progress on the reference implementation is available.
- You might also want to check regularly the FAQ page (link in electronic version) where we will give some answers to the most frequently asked questions about the mini-project

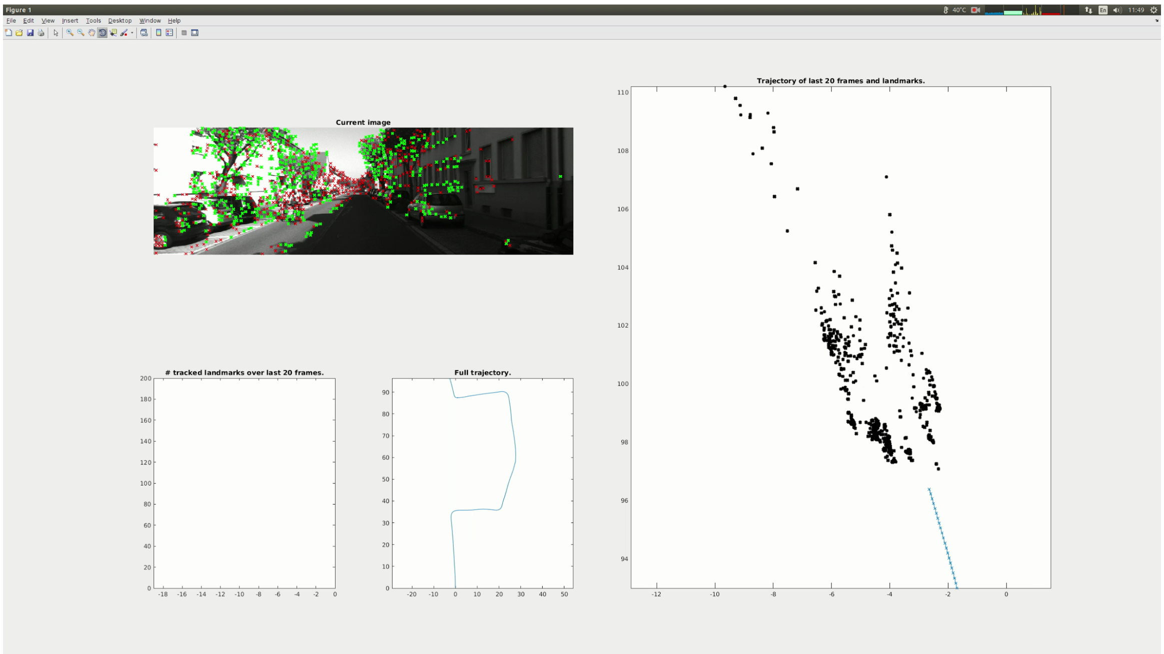

Figure 1: Preview of our implementation on the KITTI dataset.

1.2 Goal of the project

The goal of this mini-project is to implement a simple, monocular, visual odometry (VO) pipeline with the most essential features: initialization of 3D landmarks, keypoint tracking between two frames, pose estimation using established 2D ↔ 3D correspondences, and triangulation of new landmarks.

You will use the knowledge acquired during the class, and specifically the different modules developed during the exercise sessions.

1.3 Datasets

You are provided with three datasets to test your pipeline:

- The parking dataset (monocular)
- The KITTI 00 dataset (stereo)
- The Malaga 07 dataset (stereo)

1.4 Grading

You will be graded according to three things:

1. The quality of the pose estimation from your VO pipeline. We will run your code, and read it (please note that particularly unreadable code will be penalized).
2. The text report that you will have to hand in.
3. Whether you implemented one or multiple bonus features, or did something in addition to the basic requirements (and explained it in the report). A (non-exhaustive) list of ideas is given in 1.4.

We will **not** do a strict quantitative evaluation of the quality of your VO pipelines (i.e. we will not compute any positional error compared to the ground truth). Instead, we will run your code on several datasets, and judge it from a qualitative point of view, paying particular attention to the following points (sorted by decreasing order of importance):

- The features you implemented. The maximum grade can only be achieved with a completely monocular pipeline.
  - If your pipeline requires stereo frames in continuous operation, your grade will be ≤ 4.5.
  - If your pipeline requires stereo frames for initialization only, your grade will be ≤ 5. Note that no penalty is given if a bonus feature (see 1.4) requires stereo frames.
- How far does the pipeline go without failing or deviating significantly from the ground truth. Note that with the KITTI and Malaga sequences, scale drift will be difficult to avoid, we will take that into account when judging your VO pipelines).
- How fast does your code run (in particular for Matlab code, we will check whether you paid attention to using vectorized operations instead of for loops whenever possible).

Project report The project report should summarize the work that you did and specify exactly what your VO pipeline does. For example, whether it is monocular or stereo, and how well it performs on the provided datasets (with plots showing the trajectory estimated). The project report is also the place to describe the (eventual) bonus features that you implemented, or the additional work you did beyond the basic VO pipeline required, and how this impacts the quality of your VO pipeline. Note that having implemented an additional feature which degrades the quality of your VO pipeline will be accepted and valued as long as: i) the implemented feature is properly motivated and described, and ii) an analysis showing the effect of your additional feature is provided in the report. You should also upload or attach a video of your working pipeline.

(Non-exhaustive) list of bonus features/improvements This list is ordered by increasing weight in terms of bonus points.

- An custom, appealing plotting of your results (trajectory, 3D landmarks, keypoints being tracked, etc.).
- Any idea that you come up with and think will improve the quality of your VO pipeline (you may write an email to us for confirmation).
- An automatic way to select the frames that will be used for initialization (in the basic VO pipeline, these frames are hard-coded).
- Refine the estimated pose (after P3P) by minimizing the reprojection error in a nonlinear fashion. This will be more clear after lecture 13 about Bundle Adjustment.
- Record your own dataset - with your smartphone's camera (that you will have calibrated beforehand using the Matlab camera calibration toolbox).
- Propose and implement improvements to combat scale drift (without using stereo frames).
- A detailed, quantitative analysis that compares several approaches that can be applied to the same component of the VO (for example, keypoint tracking via block matching versus KLT tracker - what is the impact on speed, accuracy, etc.).
- Using your VO pipeline for some specific application (for example, estimating the pose for an virtual/augmented reality headset).
- Structure-Only Bundle Adjustment (i.e. refinement of landmark positions).
- Full Bundle Adjustment (motion and structure) on a selected set of recent frames (keyframes), to improve the current estimate.
- Loop detection using a Bag of Words approach (see the upcoming lecture 12 and exercise 9) + include the loop closure constraints in your Bundle Adjustment step.

1.4.1 Code building blocks and use of external libraries

You can of course use the modules that you built during the exercise sessions. However, we would highly recommend you to use the implementation provided to you to limit the potential amount of bugs. For your convenience, you will find packaged in the provided archive the code from all the previous classes.

External libraries Although we encourage you to use the modules that have been developed during the exercises sessions, you are allowed to use external functions (for example, functions from Matlab or OpenCV) for everything that has been covered during the exercises. For example, for tracking a keypoint between two successive frames, you can either use the Lucas-Kanade tracker developed during exercise 8, or the vision.PointTracker in Matlab's Computer Vision System Toolbox, or OpenCV's implementation.

Exception: Fundamental matrix estimation with RANSAC You are **not** allowed to use an external function to estimate the fundamental (or essential) matrix using the 8-point algorithm and RANSAC (such as Matlab's `estimateFundamentalMatrix` or OpenCV's `findFundamentalMat` with the flags `CV_FM_RANSAC` or `CV_FM_MEDS`). Some implementation hints to help you with this task are given in Paragraph 3.2.1.
The reason for that is that the 8-point algorithm in conjunction with RANSAC has **not** been covered during the exercises, and is an important part of the VO pipeline that we want you to implement from scratch. Of course, you **are** allowed to reuse some code from exercise 6 (which combines P3P and RANSAC).

We won't provide help with using external functions or libraries.

Note that we will ask you detailed questions about the theory of the separate modules during the oral exam, so implementing them (as was done during the exercises) is very a good idea to prepare.

1.4.2 Hand-out

Code

Matlab You should provide A ZIP archive containing a `main.m` file which we can run directly without any change in the code. Please also indicate in the report and in the code which version of Matlab you are using. Note that we will set the variables corresponding to the dataset paths (e.g. `parking_path`, `kitti_path`, `malaga_path`) prior to running your code.

Since we will go through your code, **we will also penalize particularly unreadable code**. Make sure to use concise variable names, and regularly remove code that doesn't do anything.

Other languages You are allowed to use any language of your choice (for example Python, C++, etc.). However, it is your responsibility to ensure that we can run your code with minimal effort, on Ubuntu 14.04 (with Python 2.7 and gcc 4.9.2). As mentioned above, you are allowed to use external dependencies (such as OpenCV for example) provided that the functions you use have been implemented in the exercises. In that case, you must:

- Provide detailed explanations regarding how to run your code (provide a Makefile for C/C++ for example, list the necessary dependencies and how to install them).
- The necessary dependencies **must** be installable with apt-get or pip, otherwise you need to request for an exception beforehand (ask the TAs).

Report and Video The content of the report has been detailed above. Note that the maximum number of pages allowed for the report is **5 pages** (excluding plots, images and installation/run instructions). If you have a working pipeline, you should also upload or attach a video of it.

2 Overview of the proposed pipeline

We first give a global overview of the proposed pipeline and the different components involved. We will then go into more details.

2.1 Notations

Throughout the next section, we will use the following notations:

- We denote the set of all N frames in a dataset by $\{I^i \triangleq I(t^i)\}_{i=1..N}$,
- We denote the pose of the camera at time t^i by ${}^W T_C^i$.

2.2 Overview

The proposed pipeline is illustrated in Figure 2, and is composed of two main components:

- An initialization module that extracts an initial set of 2D $\leftrightarrow$ 3D correspondences from the first frames of the sequence.
- A continuous VO module that processes each frame I^i, estimates the current pose of the camera ${}^W T_C^i$ (using an existing set of landmarks), and regularly triangulates new landmarks.

These two modules can be developed independently from one another, although to test the continuous VO module, you will need to have a working initialization module (see section 3); for KITTI, you may use the 2D $\leftrightarrow$ 3D correspondences provided in the RANSAC exercise.

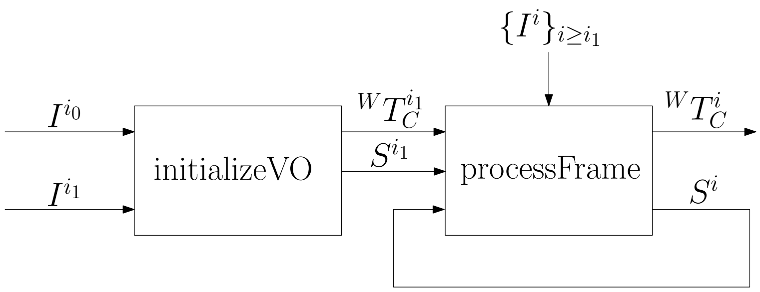

Figure 2: Overview of the proposed VO pipeline. The initialization module (function `initializeVO`) takes as input two (user-selected) frames I^{i_0} and I^{i_1}, and returns the first state S^{i_1} (that contains the first set of 2D $\leftrightarrow$ 3D correspondences), and the first camera pose ${}^W T_C^{i_1}$. The function `processFrame` is called iteratively for all the remaining frames $\{I^i\}_{i \geq i_1}$ and estimates the camera pose ${}^W T_C^i$ for every frame. Note that `processFrame` propagates the state S^i to the next frame (see section 4 for more details).

3 Initialization

The continuous VO pipeline described below requires, for initialization, a set of keypoint landmarks (i.e. 2D $\leftrightarrow$ 3D) correspondences. We propose two different ways to extract such an initial set of correspondences: the first uses a stereo pair of images to triangulate landmarks, while the second uses two-view geometry.

We suggest you to proceed as follows:

- Implement the first initialization approach (which is easier to implement; or use the data given for KITTI in the RANSAC exercise).
- Implement the continuous VO pipeline (described in the next section).

- Test it (with the first initialization approach, you need a stereo dataset, e.g. either the KITTI or the Malaga dataset).
- Once the continuous pipeline works correctly, implement the second initialization approach, which is more general and works for monocular VO as well.

3.1 Initialization from a stereo pair of images

Although your VO pipeline will be monocular, i.e. using only the left frames of the sequence $\{I^i_{left}\}$, we propose, as a first approach, to use the first stereo pair of frames $\{I^0_{left}, I^0_{right}\}$ to initialize a set of 3D landmarks.

Here is how you can proceed:

- Extract a set of keypoints $\{p_{k,left}\}_{k=1..K}$ in I^0_{left} (hint: exercise 3).
- Establish the correspondences $\{p_{k,left} \leftrightarrow p_{k,right}\}$ in I^0_{right} using stereo matching (hint: exercise 4).
- Triangulate the keypoint correspondences $\{p_{k,left} \leftrightarrow p_{k,right}\}$ to get an initial set of 3D landmarks $\{X^0_k\}$ (see exercise 4).

You can now initialize the continuous VO pipeline with the newly established set of keypoint landmark correspondences $\{p_{k,left} \leftrightarrow X^0_k\}$.

3.2 Initialization using two-view geometry, for monocular VO

As you learnt in lecture 6, an alternative way to proceed is to use two-view geometry to estimate the relative pose between two (sufficiently distant) frames, and triangulate a point cloud of landmarks.

You can proceed as follows:

- Manually select two frames I^{i_0} and I^{i_1} at the beginning of the dataset (at times i_0 and i_1).
- Establish keypoint correspondences between these two frames (hint: exercise 3).
- Estimate the relative pose between the frames and triangulate a point cloud of 3D landmarks (hint: exercise 5).
- Since the keypoint correspondences from the previous step will inevitably contain some outliers, you will need to use RANSAC (some hints are given below) to filter them out.

You can now initialize the continuous VO pipeline with the inlier keypoints and their associated landmarks.

3.2.1 Implementation hints

Make sure that the baseline between the two initialization frames is large (i.e. it is better not to use two adjacent frames). Don't pick too distant frames either, otherwise it becomes more difficult to establish keypoint correspondence. For the KITTI dataset, we obtained good results using frame 1 and frame 3 ($i_0 = 1$ and $i_1 = 3$).

Eight-point algorithm with RANSAC You have seen how to implement robust model estimation with RANSAC in exercise 6. A key ingredient of RANSAC is a way to decide whether a given sample should be considered an inlier, given a model. In the case of RANSAC for fundamental matrix estimation, you can use the epipolar line distance (see lecture 8, page 35) to discriminate inliers from outliers. Specifically, for a given candidate fundamental matrix F, a point correspondence should be considered an inlier if the epipolar line distance is less than a threshold (a threshold of 1 px should give good results). Implementing the eight-point algorithm with RANSAC can be quite tricky, so make sure you do proper testing before plugging your initialization code into the continuous VO pipeline. One way you can do this is extend exercise 5 to work with RANSAC. To test it, you may add artificial outliers to the supplied inlier keypoint correspondences.

4 Continuous operation

The continuous VO pipeline is the core component of the proposed VO implementation. Its responsibilities are three-fold:

1. Update the set of keypoints landmarks correspondences across frames.
2. Estimate the current camera pose based on an incoming frame and the current set of keypoints landmarks (2D 3D) correspondences.
3. Regularly triangulate new landmarks for use in the subsequent frames.

Step 1. can be carried out by propagating the 2D 3D correspondences across keyframes through keypoint tracking between successive frames.
Step 2. can be achieved using PnP (see lecture 3).
Step 3. can be done by maintaining keypoint tracks, i.e. sequences of keypoints that are matched in multiple subsequent frames, but which do not yet have an associated 3D landmark. These tracks can be later used to triangulate a new 3D landmarks as soon as this can be done reliably enough.

Steps 1. and 2. are sufficient to make a VO pipeline with basic functionality (i.e. which will use the landmarks triangulated in the initialization phase only), while step 3. is needed for doing VO on a larger scale. For this reason, we suggest to implement first steps 1. and 2., test them on the beginning of the datasets, and then implement step 3.

4.1 Steps 1. and 2.

To perform steps 1. and 2. for every incoming frame, we propose to implement a single function `processFrame` which will be called for every frame I^i in the dataset, whose inputs and outputs are given in Figure 2.

The state S^i of a frame at time t^i contains a set of K keypoint $\leftrightarrow$ landmark correspondences $\{p_k^i \leftrightarrow X_k\}_{k=1..K}$, where p_k^i is a 2D keypoint extracted in frame I^i and X_k its corresponding landmark in the scene. Note that the landmarks $\{X_k\}$ are fixed 3D points in the scene, and they do not depend on the current camera pose; thus, they are not indexed by time and remain constant throughout the frames.

Said otherwise, the function `processFrame` has the following form:

$$S^i, {}^W T_C^i = processFrame(I^i, I^{i-1}, S^{i-1})$$

The key idea in this design is that the function inputs solely depend on the output of the previous function call (and the new frame to process), i.e. it has the Markov property. That means we don't need to build a data structure to maintain the history of the past frames, all that is needed is contained in the current state.

As discussed before, `processFrame` should do two things:

- Propagate the keypoint landmark correspondences from time t^{i-1} to t^i, i.e. propagate the state S^{i-1} to S^i.
- Estimate the current camera pose ${}^W T_C^i$ based on the new state S^i.

We now give further indications for each step.

4.1.1 State propagation

To propagate the 2D $\leftrightarrow$ 3D associations $p_k^{i-1} \leftrightarrow X_k$ (that were established at time t^{i-1} in frame I^{i-1}) to the current frame I^i, we first establish correspondences $p_k^{i-1} \leftrightarrow p_k^i$ by tracking the keypoints p_k^{i-1} from frame I^{i-1} to frame I^i. Then, all that is left to do is to update the correspondences $p_k^{i-1} \leftrightarrow X_k$ to $p_k^i \leftrightarrow X_k$.

You can proceed as follows:

- For every keypoint $p_k^{i-1} \in S^{i-1}$, track that keypoint (4.1.3) to the current frame I^i (to obtain p_k^i):
  - Upon tracking success, update the correspondence $p_k^i \leftrightarrow X_k$.
  - Upon tracking failure, discard the correspondence p_k and its associated landmark X_k, i.e. do not include them in the updated state S^i.

4.1.2 Pose estimation

Use the updated correspondences $p_k^i \leftrightarrow X_k^i$ to estimate the pose ${}^W T_C^i$ using PnP and RANSAC (this has been done exactly in exercise 6). In addition to estimating the pose, RANSAC provides you with a list of outlier correspondences. We suggest you to additionally discard these correspondences from the state S^i.

4.1.3 Implementation tips

Keypoint tracking between successive keyframes You have two options to implement keypoint tracking between two successive keyframes:

- Use the KLT algorithm, that you will learn about in lecture 11 and implement in exercise 8. If you use Matlab. you can use the KLT algorithm implementation from the Computer Vision System Toolbox meanwhile.
- Use the keypoint description and matching algorithms implemented in exercise 3 . For best results, you may modify the function `matchKeypoints` to search for matches in the next frame only in a small neighborhood of the keypoint in the current frame. This works because frame-to-frame motion is limited. Implementation hint: to save computations, you may also store the descriptors associated with each keypoint in the state to reuse them when trying to match keypoints in the next frame.

We recommend using the first option (KLT tracker), since it is computationally more efficient, and it can also provide matches with subpixel-accuracy.

Pose estimation from a set of 2D $\leftrightarrow$ 3D correspondences You have implemented exactly this in exercise 6, using P3P and RANSAC. In this exercise, we had suggested to refine the P3P guess with a DLT solution for all inliers, once the maximum set of inliers has been determined. However, while developing the reference VO, we have found that the DLT solution is very often worse than the P3P guess.

Data storage for the state Although we described the state in terms of "sets" (i.e. collections of objects), keep in mind that in Matlab all the necessary data can be stored as simple 2D matrices.

4.2 Step 3.: Triangulating new landmarks

So far, the pipeline can use the landmarks X_k from the initialization phase to localize subsequent frames. However, once the camera has moved far enough, these landmarks might not be visible any more. It is thus necessary to continuously create new landmarks.

We propose an approach which maintains the Markov property of our design and provides new landmarks asynchronously, as soon as they can be triangulated reliably. The idea is to initialize, for each new frame I, a set of **candidate keypoints** (which of course do not have an associated landmark yet), and try to track them through the next frames.

Thus, at every point in time, we maintain a set of M candidate keypoints $\{c_m^i\}_{m=1..M}$ which have been tracked from previous frames. For every candidate keypoint c_m^i, we call the sequence $\Gamma_m = \{c_m^{i-L_m}, c_m^{i-L_m+1}, \ldots, c_m^i\}$ of tracked keypoints from frame I^{i-L_m} to frame I^i a keypoint track (of length L_m). As soon as a given keypoint track Γ_m meets some conditions (more details given in 4.2.2), we can reliably triangulate a new landmark from the keypoint observations Γ_m, and the camera poses $\{{}^W T_C^{i-L_m}, \cdots, {}^W T_C^i\}$.

We assume that the best triangulation for a given track can be achieved using the most recent observation c_m^i, the first ever observation of the keypoint $c_m^{i-L_m}$, and the poses ${}^W T_C^i$ and ${}^W T_C^{i-L_m}$. Hence, all we need to remember for a given keypoint track Γ_m is the first observation $c_m^{i-L_m}$ and the camera pose at the time of the observation ${}^W T_C^{i-L_m}$.

We can add the following data to the state S^i to reflect this:

- The set of candidate keypoints $\{c_m^i\}_{m=1..M}$.
- A set containing the first observations for each keypoint track Γ_m: $\{c_m^{i-L_m}\}_{m=1..M}$.
- For each candidate keypoint c_m^i, the camera pose ${}^W T_C^{i-L_m}$ at the first observation of the keypoint (see 4.2.2 for suggestions regarding efficient storage of these data).

Again, we do not store the entire keypoint tracks Γ_m, but only the first observation and the associated camera pose.

We now detail the necessary steps to complete step 3).

4.2.1 Keypoint track initialization and landmark triangulation

Update the function $processFrame(I^i, I^{i-1}, S^{i-1})$ to perform the following additional steps after steps 1. and 2.:

Initialization of new candidate keypoints

- Detect new candidate keypoints c_m^i in I^i to initialize new keypoint tracks (the number of keypoints to detect is up to you).
- For every newly initialized keypoint c_m^i, remember in the state S^i that this is the first observation of the new keypoint track Γ_m , i.e. add c_m^i and ${}^W T_C^i$ in the set of first keypoint observations.

Triangulation of keypoint tracks

- For every candidate keypoint $c_m^{i-1} \in S^{i-1}$ (in frame I^{i-1}), try to track it to the current frame I^i:
  - Upon tracking success, update the current state with the tracked keypoint c_m^i.
  - Upon tracking failure, discard the keypoint track Γ_m.
- Check the triangulability of every candidate keypoint c_m^i, using the first observation $c_m^{i-L_m}$ and the camera pose ${}^W T_C^{i-L_m}$ stored in the state (see 4.2.2 for the triangulability check). If it is possible, triangulate a new landmark X_{new} using c_m^i, $c_m^{i-L_m}$, ${}^W T_C^i$ and ${}^W T_C^{i-L_m}$.
- Update the state as follows:
  1. Add the newly established correspondence $\{c_m^i \leftrightarrow X_{new}\}$ to the state (i.e. update the list of keypoints $\{p_k^i\}$ and landmarks $\{X_k\}$.
  2. Discard the keypoint track Γ_m. If triangulation is not possible yet, do nothing.

4.2.2 Implementation hints

Data storage

- To store the first observation of every candidate keypoint c_m^i, you have multiple options: Record directly $c_m^{i-L_m}$ and ${}^W T_C^{i-L_m}$ in the state. To store the latter with a single matrix, you can reshape the 4×4 transformation matrix ${}^W T_C^{i-L_m}$ to a 16×1 vector and store all the transformation matrices in a $M \times 16$ matrix (where M is the current number of keypoint tracks). Alternatively, you can compute and store a 3D bearing vector $b_m^{i-L_m}$, which will encode directly the direction of the 3D ray corresponding to $c_m^{i-L_m}$ (express it in the world frame). In that case, you will also need to store the 3D position of the camera center (expressed in the world frame) and modify a bit the linear triangulation algorithm.

Triangulation

- Use the linear triangulation algorithm developed in exercise 5 to triangulate new landmarks.
- A simple triangulability check to test whether a keypoint track is ready for good triangulation is to compute the angle between the bearing vector of the first observation, and the current bearing vector: if it is above some threshold, triangulation will be likely to yield a good 3D landmark.
- You can be pretty sure that a landmark triangulated behind the camera is incorrect; discard it.

General hints

- In general, proceed step by step and verify your intermediate results visually. Concretely, make sure matching, localization and landmark propagation work properly before triangulating new landmarks.
- We have given pseudo-code indicating what you should do on a keypoint-per-keypoint basis, for clarity. Of course, you should batch these operations for all keypoints. The webpage Matrix indexing in Matlab (link in electronic version) will be very handy for this. We strongly recommend to read ALL of this page. Note that indexing can be used both on the right and the left hand side of assignments. Also note that you can index recursively: if you have e.g. A = [1 12 14 17], then B(A([2 4])) will access the 12th and 17th elements of B. Note that this might not work as expected when A is a logical index - use the `find` function in that case.