Technische Universität München
Fakultät für Informatik
PD Dr. Slobodan Ilic

WS 18/19
Exercise Sheet 1.1
November 8, 2018

Exercises in Tracking & Detection

In 3D Computer Vision object pose estimation is one of the key elements for model based tracking. Its applications are evident in many application areas like Augmented Reality, Robotics, Machine Vision and Medical Imaging. In this project you will develop a method that will estimate the pose of a camera in respect to the 3D model of the object exploiting the texture information of the object described by its visual features. You are given a calibrated camera with its known intrinsic parameters, 3D model of the object described as a triangular mesh containing vertices and faces and finally a number of input RGB camera images. Initially the object doesn't contain associated texture, but you will be given a couple of images of the object seen from different viewpoints, which will serve to associate texture information to the object. The implementation has to be done in MATLAB. Please follow the requirements below in terms of the functions that can be used from MATLAB Toolboxes. If not specified that available functions from MATLAB can be used, it means that the functions must be implemented by yourself.

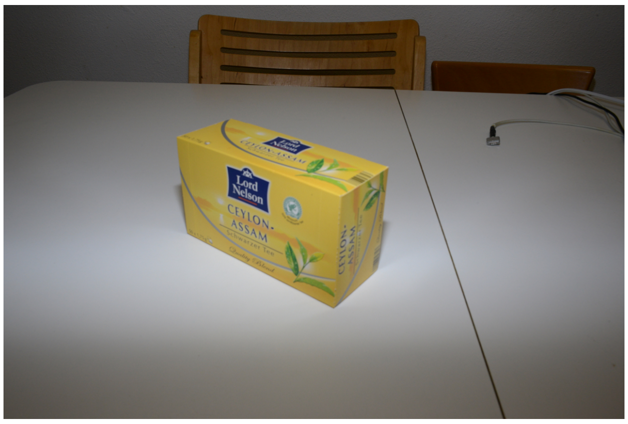
Figure 1: Teabox image for texturing.

Task 1 Model preparation

As a first step, you will need to associate the texture information to the given 3D model from a couple of input images depicting the object from different viewpoints. The 3D model called *teabox.ply* is given as an ASCII file in PLY format. The description of the format can be found at `http://paulbourke.net/dataformats/ply/`. Use provided MATLAB function **read_ply** to read PLY files.

The model contains 8 vertices accompanied with per vertex normals and 12 triangles (faces). It can be visualized using Meshlab `http://www.meshlab.net/`.

In addition to the *teabox.ply* file describing the object's geometry you are also given the intrinsic camera parameters: $f_x = f_y = 2960.37845$ $c_x = 1841.68855$ $c_y = 1235.23369$. The origin of the world coordinate system is attached to the lower left corner of the object and it is right handed.

In the first step of the exrecise we would like to estimate poses of the object in the training images (can be found in **images/init_texture**). The recommended option is to label locations of object's visible corners in each image. Then, given coordinates of the corners in the image and the corresponding 3D coordinates, object's pose can be estimated using PnP. For PnP you can use *estimateWorldCameraPose* function from MATLAB's Computer Vision System Toolbox.

You must visualize the computed camera poses as shown in Figure 2. Note that we do not require you to project texture onto the object. For visualization you should use *plotCamera* function from MATLAB's Computer Vision System Toolbox.

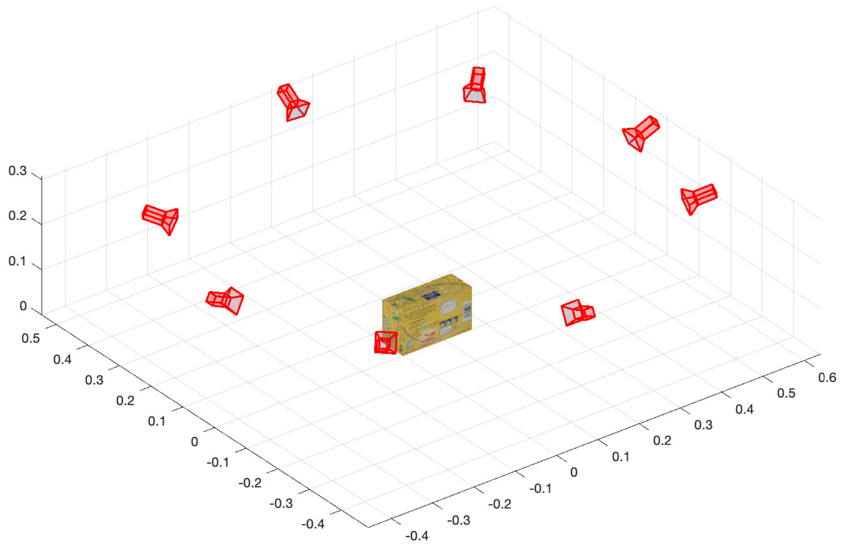

Figure 2: Camera poses

The next step is to detect SIFT keypoints in the training images and compute their 3D locations on the model. These additional 2D-3D correspondences will be used for automatic object detection and pose estimation in the following exercises. SIFT keypoints can be extracted using VLFeat library `http://www.vlfeat.org/`.

Once the 2D features are extracted, we can back-project them onto the 3D model. Without loss of generality, a 2D point $\mathbf{m} = [x, y]^T$, map to a 3D ray using the inverse intrinsics matrix and the camera optical center C:

$$\mathbf{r}(\mathbf{m}) = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \mathbf{C} + \lambda \mathbf{Q}^{-1}\mathbf{m}$$

where $\mathbf{Q}$ is part of the projection matrix $\mathbf{P} = [\mathbf{Q}\ \mathbf{q}]_{3\times4} = \mathbf{K}[\mathbf{R} \mid \mathbf{t}] = [\mathbf{KR} \mid \mathbf{Kt}]$ and $\mathbf{C} = -\mathbf{Q}^{-1}\mathbf{q}$ is the optical center of the camera. Given the ray $\mathbf{r}(\mathbf{m})$, we need to find where it intersects the model in 3D space. We provide function *TriangleRayIntersection* for this purpose. This will give coordinates in the world coordinate system, which coincides with the model coordinate system.

In order to verify that backprojection has been implemented correctly, visualize 3D locations of SIFT keypoints using *scatter3* MATLAB function. As an example, see provided file **sift3d.fig**. For the future exercises for each SIFT keypoint you need to save its descriptor and 3D location in the model coordinate system.