Computer Vision Project
6D Object Pose Estimation

You are given one RGB image containing a number of objects. The task is to estimate the pose of each of these objects, relative to the camera. You are also provided a set of 2D-3D correspondences (3D points and their projections).

This is essentially the resection problem, which you have seen before, but there is one difference from how you were presented to it in the course material. In both cases you have 2D-3D correspondences, but in this case the 3D correspondences are in object specific coordinate systems – not in a unified global coordinate system. The local object coordinate systems are specific for, and centered at, each object.

We are looking for the transformations T_i, which transform points in the respective local object coordinate systems, to the global coordinate system:

$$T_i = \begin{bmatrix} R_i & t_i \\ 0 & 1 \end{bmatrix} \tag{1}$$

Although the camera is the same for all objects (we may assume $P = \begin{bmatrix} I & 0 \end{bmatrix}$), we can actually rephrase the problem so as to let object-specific camera matrices P_i capture both P as well as the euclidean transformations T_i that we are looking for:

$$P_i = PT_i = \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} R_i & t_i \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_i & t_i \end{bmatrix} \tag{2}$$

The 2D-3D correspondences are dense, i.e. for every pixel of an object there is a corresponding estimated 3D point, see Figure 1. Such correspondences could potentially be derived from a neural network trained for the task, but in this case they are synthetic. They do however suffer from noise as well as outliers (to a varying degree).

You may use any method for estimating the pose, e.g. DLT / Levenberg-Marquardt. Comparing multiple methods is encouraged.

In the `data/` directory you are provided 9 images, along with 9 `.mat` files, containing the following variables:

- `u` – Cell array of 2D points for each object (normalized). `u(i)` is a $(2 \times n_i)$ array containing all 2D points for object i.
- `U` – Cell array of 3D points for each object. `U(i)` is a $(3 \times n_i)$ array containing all 3D points for object i.
- `poses` – Cell array of ground truth poses for each object. `poses(i)` is the ground truth calibrated camera matrix $P_i^{gt} = [R_i^{gt} \quad t_i^{gt}]$ for object i.
- `bounding_boxes` – Cell array of 3D corner points defining a bounding box for each object. This should not be needed, except when passed to the provided evaluation / plotting functions.

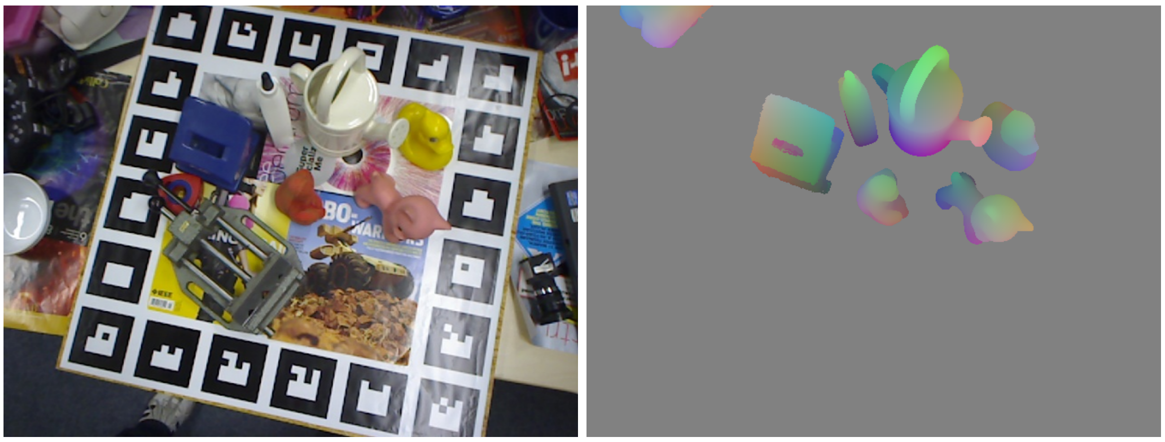

Figure 1: An example RGB image, along with an illustration of its ground truth correspondence map. For every pixel where an object is visible, the RGB-values (conveniently happening to be exactly 3 channels) encode a corresponding 3D point in the local object coordinate system.

The following functions are provided:

- `minimalCameraPose.m` takes 3 normalized point correspondences, and returns all possible camera matrices providing a solution to these correspondences. If multiple solutions are found, you can check the number of inliers among all correspondences for each of the solutions to determine which is the relevant one.
- `eval_pose_estimates.m` implements a metric for evaluating estimated poses by comparing to ground truth. Returns the score for each object, alphabetically ordered (ape, can, cat, duck, eggbox, glue, holepuncher).
- `draw_bounding_boxes.m` plots projected 3D bounding boxes of both ground truth poses and estimated poses, to be compared.

Note that the calibration matrix K is known, but should not be needed since all 2D points are normalized. K is however defined and used in `draw_bounding_boxes.m`, for plotting purposes.

Useful matlab commands:

```
%% Sample 3 correspondences and find candidate camera matrices for one object
ind = randsample(size(U{obj_idx},2), 3);
Ps = minimalCameraPose(pextend(u{obj_idx}(:, ind)), U{obj_idx}(:, ind));

P_est = ... % cell array with your final pose estimates for each object.

%% Plot bounding boxes of all objects, overlayed on image.
% You can try out the plotting fucntionality immediately,
% by setting P_est = poses (ground truth).
draw_bounding_boxes(I, poses, P_est, bounding_boxes);

% Compute (and print out) the score for each estimated object pose.
scores = eval_pose_estimates(poses, P_est, bounding_boxes);
```

For the report: Please include a description of your algorithm, a table of scores for all objects in all images, and the total average of all scores in the table. Also, submit an m-file that executes your algorithm and produces all scores.

We will create a top-list of the best performing algorithms. Note: You are not allowed to use the ground truth poses in your algorithm! Good Luck!