Computer Vision Project
Rigid 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 resectioning problem, which you have seen before, but there is a difference from how you were presented to it in the course material. In both cases you have 2D-3D point correspondences, but in this case the 3D points 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:

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

Note that all P_i are necessarily calibrated in this formulation. This is something you will have to take into account.

The 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).

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.

At a minimum (for a passing grade), you should:

- Filter the correspondences from outliers.
- Run any resectioning algorithm of your choice to fit a calibrated camera to all filtered points. The algorithm needs to be implemented by you (no high level optimization calls or similar), and is intended to yield a refinement*.
- Compare the result with just using the provided minimal solver (see below).
- (+Provide everything that is asked for in the gray box furher down.)

*Note that as long as the refined solution is reasonable, the actual score is not so important from the grading point of view. In particular, one can not expect very good performance from methods that handle the rotation constraint of calibrated camera matrices very crudely, but such methods are still accepted (e.g. enforcing the rotation constraint after DLT). Also, note that implementing Levenberg-Marquardt for this problem based on the supplied matlab functions does still count as your own implementation.

For acquiring optional points, the project can be extended in two ways, both contributing with points:

- Run additional refinement algorithms and compare the results between them (e.g. DLT + Levenberg-Marquardt, but virtually any approach you like). For full points on this part consider 1+ additional algorithms, which you have also implemented yourself, or alternatively consider 2+ additional algorithms, but which need not be your own implementation.
- Try advanced RANSAC schemes for outlier filtering – this is the major part considered for optional points. Evaluate the benefit in doing so (Are fewer iterations needed for a good result? Given a fixed budget of iterations, will the advanced scheme typically yield a better pose fit?). Implement one of the methods presented in the research papers below (and distributed along with the project data):
  - Preemptive RANSAC: `nister2005_preemptive_ransac.pdf`
  - R-RANSAC: `chum2002_randomized_ransac.pdf`
  - WaldSAC: `matas2005_waldsac.pdf`

No matter what your ambitions are, you always need to give a somewhat detailed description in the report, for whatever algorithms you use.

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.

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.

In addition to the above, there some functions taken from assignment 5, which can be used as a basis for your Levberg-Marquardt implementation if you want. However, you can not use the implementation directly, since it is not intended for resectioning but for Structure-from-Motion, in which case both cameras and 3D points are unknown.

- `ComputeReprojectionError.m` This should work out-of-the-box for you. Although it expects a collection of camera matrices and image points for different views, you can submit only a single view.
- `LinearizeReprojErr.m` Calculates the Jacobian J for SfM. The rows correspond to x & y residuals in all different camera views, while the columns correspond to $3 \cdot$ `n_pts` $- 1$ 3D point parameters, followed by $6 \cdot$ (`n_cams` $- 1$) camera parameters. Note that one 3D point parameter, and 6 camera parameters (the parameters for the first camera matrix) are disregarded in J. This is due to locking the global coordinate system, which (unlike resectioning) for SfM is arbitrary to start with. Also note that the Jacobian computed here is not exactly what you need – you need to make some adjustments so that it fits your case with resectioning.
- `update_solution.m` Takes as input a parameter increment (the δv for a $v \to v + \delta v$ update), as well as the previous estimated cameras and 3D points (corresponding to v). The output is the updated cameras and 3D points corresponding to $v + \delta v$. Also in this function, the first camera (and the first coordinate of the first 3D point) is assumed to be constant. This function will need to be modified for your purposes.

Finally, 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);
```

Submit a PDF report, in which you include a description of your algorithm(s). Also, for each pose estimation method you use, include a table of corresponding scores for all objects in all images, a histogram over all scores in the table, and the total average of all scores in the table.
Also, briefly state to what extent you have contributed to the implementation(s) of the algorithm(s) considered. Finally, submit matlab code that executes your algorithm(s) and produces all scores.

Note: You are not allowed to use the ground truth poses in your algorithm! Good Luck!