

## Assignment-3

*Released: September 16<sup>th</sup>**Deadline: October 5<sup>th</sup>*

## Instructions

- This assignment is designed to get you familiar with stereo reconstruction, and non-linear optimization.
- Code can be written in Python or C++ only. OpenCV or any equivalent library can be used for image input & output, and where ever specified. Make sure your code is modular since you might be reusing them for future assignments.
- Submit your code files and a report (or as a single Jupyter notebook if you wish) as a zip file, named as `<team_id>_assignment3.zip`.
- The report should include all outputs and results and a description of your approach. It should also briefly describe what each member worked on. The scoring will be primarily based on the report.
- Refer to the late days policy and plagiarism policy in the course information document.
- Start early! This assignment may take fairly long.

**Q1) Stereo dense reconstruction**

3D point clouds are very useful in robotics for several tasks such as object detection, motion estimation (3D-3D matching or 3D-2D matching), SLAM, and other forms of scene understanding. Stereo cameras provide us with a convenient way to generate dense point clouds. *Dense* here, in contrast to *sparse*, means all the image points are used for the reconstruction. In this part of the assignment you will be generating a dense 3D point cloud reconstruction of a scene from stereo images as shown.



Figure 1: A reference reconstruction. The empty spaces are regions that could not be matched properly or that have very little disparity. Courtesy: Dellaert et al.

Download the data from [\[here\]](#). It includes a set of rectified and synchronized stereo image pairs from a KITTI sequence, the calibration data, and the absolute ground truth poses of each stereo pair. The procedure is as follows,

- Generate a disparity map for each stereo pair. Use OpenCV (e.g. StereoSGBM) for this. Note that the images provided are already rectified and undistorted.
- Then, using the camera parameters and baseline information generate colored point clouds from each disparity map. Some points will have invalid disparity values, so ignore them. Use [Open3D] for storing your point clouds.

- Register (or transform) all the generated point clouds into your world frame by using the provided ground truth poses.
- Visualize the registered point cloud data, in color. Use Open3D for this.

Report your observations with a screenshot of your reconstruction. [Bonus] Compare different stereo matching algorithms.

## Q2) Motion estimation using iterative PnP

Using the generated reconstruction from the previous part, synthesize a new image taken by a virtual monocular camera fixed at any arbitrary position and orientation. Your task in this part is to recover this pose using an iterative Perspective-from-n-Points (PnP) algorithm. The steps are as follows,

- Obtain a set of 2D-3D correspondences between the the image and the point cloud. Since here you're generating the image, this should be easy to obtain.
- For this set of correspondences compute the total reprojection error  $c = \sum_i \|x_i - P_k X_i\|^2$  where  $P_k = K[R_k | t_k]$ ,  $X_i$  is the 3D point in the world frame,  $x_i$  is its corresponding projection.
- Solve for the pose  $T_k$  that minimizes this non-linear reprojection error using a Gauss-Newton (GN) scheme. Recall that in GN we start with some initial estimated value  $x_o$  and iteratively refine the estimate using  $x_1 = \Delta_x + x_0$ , where  $\Delta_x$  is obtained by solving the normal equations  $J^T J \Delta_x = -J^T e$ , until convergence.

The main steps in this scheme are computing the corresponding Jacobians and updating the estimates correctly. For our problem, use a  $12 \times 1$  vector parameterization for  $T_k$  (the top  $3 \times 4$  submatrix). Run the optimization for different choices of initialization and report your observations.

- [Bonus++] Solve the same optimization problem but using a smaller  $1 \times 6$   $se(3)$  parameterization for the pose ( $\xi$ ). The result in this case should theoretically be better. The Jacobian here would be,

$$\frac{\partial e_i}{\partial \xi_k} = \begin{bmatrix} f_x(\frac{1}{z'}) & 0 & -f_x(\frac{x'}{z'^2}) & -f_x(\frac{x'y'}{z'^2}) & f_x(1 + \frac{x'^2}{z'^2}) & -f_x(\frac{y'}{z'}) \\ 0 & f_y(\frac{1}{z'}) & -f_y(\frac{y'}{z'^2}) & -f_y(1 + \frac{y'^2}{z'^2}) & f_y(\frac{x'y'}{z'^2}) & f_y(\frac{x'}{z'}) \end{bmatrix}$$

where  $e_i = x_i - P_k X_i$ , and  $x', y', z'$  are the coordinates of the 3D point transformed by  $\xi_k$ . The update step here would not be just  $\xi_{k+1} = \xi_k + \Delta_\xi$  but  $\xi_{k+1} = \log(\exp(\Delta_\xi) \cdot \exp(\xi_k))$ . More details are available [here].