Geometric Computer Vision: Homework assignment 1

Desheulin Oleg

2021/03/22

1 TODOs description

1.1 Write your code to constrict a world-frame point cloud from a depth image, using known intrinsic and extrinsic camera parameters

TODO: write your code to constrict a world-frame point cloud from a depth image, using known intrinsic and extrinsic camera parameters. Hints: use the class 'RaycastingImaging' to transform image to points in camera frame, use the class 'CameraPose' to transform image to points in world frame.

Solution: First, we need to obtain CameraPose item using necessary extrinsic matrix. Second, we can obtain imaging class using our intrinsics parameters stored in intrinsics_dict. Using imaginging class we can transform image to points in camera frame. Using already obtained CameraPose we can transform points to world frame and finally obtain points.

1.2 Use functions from CameraPose class to transform points_j into coordinate frame of view_i

TODO: your code here: use functions from CameraPose class to transform 'points_j' into coordinate frame of 'view_i'

Solution: We already have CameraPose class instance and we can use it's method world_to_camera to reproject points to specified camera from world frame.

1.3 Use cKDTree to find k=nn_set_size indexes of nearest points for each of points from reprojected_j

TODO: your code here: use cKDTree to find k=nn_set_size indexes of nearest points for each of points from reprojected_j.

Solution: We can use cKDTree class to build kd-tree upon (u, v) coordinates of points_i in the pixel grid of view_i. We can find indexes for point_j in our builded set using method query. We need only first 2 rows from 1 dimension of our point_j to obtain this indexes.

1.4 Use point_nn_indexes found previously and distance values from image_i indexed by the same point_nn_indexes

TODO: your code here: use point_nn_indexes found previously and distance values from image_i indexed by the same point_nn_indexes

Solution: We need to transform point_nn_indexes to pixel indexes. This can be done with unravel_index function from numpy for all points in one row for specified shape. Then we can obtain distances from image_i with this reindexed point_nn_indexes.

1.5 Compute a flag indicating the possibility to interpolate

TODO: compute a flag indicating the possibility to interpolate by checking distance between point_from_j and its point_from_j_nns against the value of distance_interpolation_threshold

Solution: We can calculate distances to nearest from depth value of this specifed point. Depth value can be obtained from the last channel of point_from_j and compare it with calculated values in point_from_j_nns. We need all point to be inside of our radius distance_interpolation_threshold, so we can check only maximum distance.

1.6 Use the interpolator to compute an interpolated distance value

TODO: your code here: use interpolate.interp2d to construct a bilinear interpolator from distances predicted in view_i (i.e. distances_i) into the point in view_j. Use the interpolator to compute an interpolated distance value.

Solution: We need to find interpolator $f(u, v)$, where u, v are coordinates obtained previously with unravel_index function. And target is distances_i in this points. For now interpolator is interpolate.interp2d and is fitted direclty. Then we can obtain interpolation in point_from_j.

2 Bonus points

2.1 Interpolation (2pts)