This file describes the functions in diy_icp.cpp, and gives instructions on installing the required libraries. The instructions are based on Ubuntu systems (tested on 18.04.06 LTS Bionic).

Installation:

The direct dependency for the program is the PCL library which can be installed, along with all sub-dependencies, using one of the following methods:

Using Advanced Package Tool (basic - v1.8)

```
sudo apt install libpcl-dev
```

Installation from source (advanced - v1.12.1)

Install dependencies:

```
sudo apt-get -y install g++ cmake cmake-gui doxygen mpi-default-dev openmpi-bin
openmpi-common
sudo apt-get -y install libusb-1.0-0-dev libqhull* libusb-dev libgtest-dev
sudo apt-get -y install git-core freeglut3-dev pkg-config build-essential libxmu-dev
libxi-dev
sudo apt-get -y install libphonon-dev libphonon-dev phonon-backend-gstreamer
sudo apt-get -y install phonon-backend-vlc graphviz mono-complete
sudo apt-get -y install qt5-default
sudo apt-get -y install libflann-dev

sudo apt-get -y install libflann1.9
sudo apt-get -y install libboost-dev
sudo apt-get -y install libeigen3-dev
```

Install VTK 8.1.1:

```
cd ~/Downloads
wget http://www.vtk.org/files/release/8.1/VTK-8.1.1.tar.gz
tar -zxvf VTK-8.1.1.tar.gz
cd VTK-8.1.1
mkdir build
cd build
cmake ..
make
sudo make install
```

Install desired PCL verison:

```
wget https://github.com/PointCloudLibrary/pcl/archive/pcl-1.12.1.tar.gz
tar -zxvf pcl-1.12.1.tar.gz
cd pcl-pcl-1.12.1
mkdir build
cd build
ccmake ..
# modify the parameters to your needs
(based on: https://pointclouds.org/documentation/tutorials/building_pcl.html)
make
sudo make install
cmake –build . –prefix "path/to/pcl"
cmake –install
```

Program Description:

The program utilizes PCL classes and commands to create functions that manipulate 3D scanner data to achieve the goal of scan registration.

To run the programs, the paths to the scans need to be called along with the name of the new file where the combined clouds are to be saved.

Example: ./diy_icp scans/scan1.pcd scans/scan2.pcd scans/scan3.pcd scans/scan4.pcd scans/combinedScan1234.pcd

`using-declarations:`

```
using point = PointXYZ;
using normal = PointNormal;
using pointCloud = PointCloud<point>;
using normalCloud = PointCloud<normal>;
```

`void loadFile (const char* fileName, pointCloud &cloud):`

Loads a pcd file to a PointCloud object.

`fileName`: path to the pcd file.
`cloud`: the PointCloud object.

Reference:
https://pointclouds.org/documentation/group__io.html#gacd22a31e8ec257c0367f4db23ab0212f
https://pointclouds.org/documentation/namespacepcl.html#a89aca82e188e18a7c9a71324e9610ec9

`void cropClouds (pointCloud::Ptr &source_cloud, pointCloud::Ptr &target_cloud,pointCloud::Ptr &source_overlap, pointCloud::Ptr &target_overlap, string dir):`

Crops the overlap regions in the source and target clouds based on their position values and the scanning direction.

`source_cloud:` the source cloud object pointer.
`target_cloud:` the target cloud object pointer.
`source_overlap:` the source overlap-region object pointer.
`target_overlap:` the target overlap-region object pointer.
`dir:` the scanning direction between the two clouds.
`target:` the target cloud positions-structure pointer.
`source:` the source cloud positions-structure pointer.

Reference:
https://pointclouds.org/documentation/classpcl_1_1_pass_through.html
https://pointclouds.org/documentation/classpcl_1_1_crop_box.html

`void normalSpaceSample (pointCloud::Ptr &cloud, normalCloud::Ptr &normals, pointCloud::Ptr &ds_cloud, float ds_factor = 0.002):`

Downsamples a point cloud using the Normal Space Sampling method.

`cloud:` object pointer of the point cloud.
`normals:` object pointer of the respective normals point cloud.
`ds_cloud:` object pointer for the downsampled point cloud.
`ds_factor:` percentage factor of the downsampling (0.2% by default).

Reference:
https://pointclouds.org/documentation/classpcl_1_1_normal_space_sampling.html

`Eigen::Matrix4f findTF (pointCloud::Ptr &source_cloud, pointCloud::Ptr &target_cloud, pcl::CorrespondencesPtr &corr_list):`

Estimates and returns the rigid transformation between a source and a target point cloud using SVD.

`source_cloud:` object pointer of the source cloud.
`target_cloud:` object pointer of the target cloud.
`corr_list:` object pointer of the correspondences between the clouds.

Reference:
https://pointclouds.org/documentation/classpcl_1_1registration_1_1_transformation_estimation_s_v_d.html

[1/3] **`void cloudsViewer (pointCloud::Ptr &source_cloud, pointCloud::Ptr &target_cloud, normalCloud::Ptr source_normals, normalCloud::Ptr target_normals, pcl::CorrespondencesPtr &corr_list, bool view_normals = true):`**

Views the passed 3D clouds along with their correspondences, and optionally their normal vectors.

`source_cloud`: object pointer of the source cloud.
`target_cloud`: object pointer of the target cloud.
`source_normals`: object pointer of the source normals point cloud.
`target_normals`: object pointer of the target normals point cloud.
`corr_list`: object pointer of the correspondences between the clouds.
`view_normals`: toggle for viewing the normal vectors.

Reference:
https://pointclouds.org/documentation/classpcl_1_1visualization_1_1_p_c_l_visualizer.html

[2/3] **`void cloudsViewer (pointCloud::Ptr &source_cloud, pointCloud::Ptr &target_cloud):`**

Views two 3D clouds simultaneously.

`source_cloud`: object pointer of the source cloud.
`target_cloud`: object pointer of the target cloud.

[3/3] **`void cloudsViewer (pointCloud::Ptr &cloud):`**

Views a single 3D cloud.

`cloud`: object pointer of the cloud.

`void saveFile (string file_name, pointCloud::Ptr &cloud):`

Saves a point cloud into a pcd file type.

`file_name`: name of the file.
`cloud`: object pointer of the cloud.

Reference:
https://cplusplus.com/reference/ostream/ostream/

`string getDirection (pointCloud::Ptr &source_cloud, pointCloud::Ptr &target_cloud):`

Detects the scanning direction based on the clouds' minimum and maximum positions.

`source_cloud`: object pointer of the source cloud.
`target_cloud`: object pointer of the target cloud.

`pointCloud::Ptr combineClouds (pointCloud::Ptr &source_cloud, pointCloud::Ptr &target_cloud, string dir):`

Combines two clouds into one while eliminating one of the double overlap areas based on critical position values of the clouds.

`source_cloud:` object pointer of the source cloud.
`target_cloud`: object pointer of the target cloud.
`dir:` the scanning direction between the two clouds.

Reference: https://pointclouds.org/documentation/classpcl_1_1_pass_through.html

`void findNormalCorrespondences (pointCloud::Ptr &source_cloud, normalCloud::Ptr &source_normals, pointCloud::Ptr &target_cloud, normalCloud::Ptr &target_normals, pcl::CorrespondencesPtr &corr, int kfactor = 8, float dist = 10):`

Estimates the list of correspondences between the two clouds based on the normal shooting technique.

`source_cloud:` object pointer of the source cloud.
`source_normals:` object pointer of the source normals point cloud.
`target_cloud:` object pointer of the target cloud.
`target_normals:` object pointer of the target normals point cloud.
`corr_list:` object pointer of the output correspondences between the clouds.
`kfactor:` the number of nearest neighbours to be considered.
`dist:` maximum distance between the source normal and the respective target point.

Reference: https://pointclouds.org/documentation/classpcl_1_1registration_1_1_correspondence_estimation_normal_shooting.html

`void findIndNormalCorrespondences (pointCloud::Ptr &source_cloud, normalCloud::Ptr &source_normals, pointCloud::Ptr &target_cloud, normalCloud::Ptr &target_normals, pcl::CorrespondencesPtr &corr, pcl::IndicesPtr &source_ind, pcl::IndicesPtr &target_ind, int kfactor = 8, float dist = 10):`

Estimates the list of correspondences between specific indices of the two clouds based on the normal shooting technique.

`source_cloud`: object pointer of the source cloud.
`source_normals`: object pointer of the source normals point cloud.
`target_cloud`: object pointer of the target cloud.
`target_normals`: object pointer of the target normals point cloud.
`corr_list`: object pointer of the output correspondences between the clouds.
`source_ind:` object pointer of the input source indices list.
`target_ind:` object pointer of the input target indices list.
`kfactor`: the number of nearest neighbours to be considered.
`dist`: maximum distance between the source normal and the respective target point.

Reference:
https://pointclouds.org/documentation/classpcl_1_1registration_1_1_correspondence_estimation_normal_shooting.html

`void normalCorrRejector (pcl::CorrespondencesPtr &corr_list, pcl::CorrespondencesPtr &corr_out, normalCloud::Ptr &source_normals, normalCloud::Ptr &target_normals, pcl::IndicesPtr rej_ids, float angle_diff):`

Rejects faulty correspondences based on the surface normal rejector.

`corr_list`: object pointer of the input correspondences between the clouds.
`corr_out`: object pointer of the output filtered correspondences.
`source_normals`: object pointer of the source normals point cloud.
`target_normals`: object pointer of the target normals point cloud.
`rej_ids`: object pointer of the output rejected point indices list.
`angle_diff`: the thresholding angle between the normals for rejection in degrees.

Reference:
https://pointclouds.org/documentation/classpcl_1_1registration_1_1_correspondence_rejector_surface_normal.html

`void one2oneRejector (pcl::CorrespondencesPtr &corr_list, pcl::CorrespondencesPtr &corr_out):`

Eliminates duplicate match indices in the correspondences, leaving only the one with the smallest distance.

`corr_list`: object pointer of the input correspondences between the clouds.
`corr_out`: object pointer of the output filtered correspondences.

Reference:
https://pointclouds.org/documentation/classpcl_1_1registration_1_1_correspondence_rejector_one_to_one.html#details

`void findNormals (pointCloud::Ptr &cloud, normalCloud::Ptr &normals):`

Estimates the surface normals at each 3D point.

`cloud`: object pointer of the input cloud.
`normals`: object pointer of the output normals cloud.

Reference:
https://pointclouds.org/documentation/classpcl_1_1_normal_estimation.html

`pcl::IndicesPtr filterByAngle (normalCloud::Ptr &normals, const Eigen::Vector3f refVector, float threshAngle, string LH)`

Calculates the angle between the normal vectors and a reference vector, and filters out the points with angles lower or higher than the threshold value. Returns an object pointer of the remaining point indices.

`normals`: object pointer of the input normals cloud.
`refVector`: input reference vector.
`threshAngle`: the threshold angle in degrees.
`LH`: “higher” or “lower”, specifies the respective comparison operation.