Feature Extraction functionalities 20220202

key definitions:

scanline	- pointcloud data from a single laser
scan	- a collection of scanline(s), generally from one full revolution of the Lidar
pointcloud	- a collection of points, generally from scan(s)

overall design:

1) take in a pointcloud containing a single scan (can be multiple scanlines)

2) separate into continuous clusters using delta_range and delta_horizontal_angle as a way to distinguish clusters

3) throw away clusters of insufficient size (please use a macro to define numerical parameters)

4) apply algorithm on each cluster to determine corners, return indices of corners

5) use indicies to extract points from original pointcloud and save as separate pointcloud

main function:

```
/** @brief take in a pointcloud containing a single scanline and return the indices of the corner points

 * @param scanline pointcloud containing one scanline

/* @return a vector containing indices of all corner points, returns empty vector if non found

std::vector<size_t> find_corners(const pcl::PointCloud<pcl::PointXYZ> & scanline);
```

sub functions:

```
/** 1) read in point cloud sample code */
pcl::PointCloud<pcl::PointXYZ>::Ptr uav_pointcloud(new pcl::PointCloud<pcl::PointXYZ>);
if(pcl::io::loadPCDFile<pcl::PointXYZ>(file_path_string, *uav_pointcloud) == -1)
{
   std::cerr << "failed to read pointcloud file. \n";
   return -1;
}
else
   std::cout << "file loaded, size: " << (*uav_pointcloud).size() << "\n";
```

```
/** 2) separate a scanline into clusters of points
* 3) this function should also use a size check at end to filter out clusters of insufficient size
@return a vector of scanline clusters*/
std::vector<pcl::PointCloud<pcl::PointXYZ>>
cluster_scanline_with_range_angle(const pcl::PointCloud<pcl::PointXYZ> & scanline, float
range_threshold, float angle_threshold);
```

```
/** 4) corner finding algorithm
@return indices of corner points*/
std::vector<size_t> find_corners_in_scanline_cluster (const pcl::PointCloud<pcl::PointXYZ> &
scanline_cluster);
```

```
/** 5) extract and save pointcloud based on indices */
// to save pcd, use pcl::io::savePCDFileASCII, make sure to check that the pointcloud is not empty
before saving, otherwise it will crash the software
```