import open3d as o3dimport numpy as npdef extract\_human\_points(cloud, eps=0.5, min\_points=50, max\_points=5000, visualize=False): """ Extract clusters of points corresponding to humans using DBSCAN. Args: cloud (o3d.geometry.PointCloud): Input point cloud. eps (float): Maximum distance between points to form a cluster. min\_points (int): Minimum points required to form a cluster. max\_points (int): Maximum points allowed in a cluster (to filter out large objects). visualize (bool): Whether to visualize the results. Returns: human\_clusters (list): List of point clouds corresponding to detected human clusters. """ # Fixed 50 distinct colors for clusters fixed\_colors = [ [1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0], # Red, Green, Blue [1.0, 1.0, 0.0], [1.0, 0.0, 1.0], [0.0, 1.0, 1.0], # Yellow, Magenta, Cyan [0.5, 0.5, 0.5], [1.0, 0.5, 0.0], [0.5, 0.0, 1.0], # Gray, Orange, Purple [0.0, 0.5, 0.5], [0.5, 0.5, 0.0], [0.5, 0.0, 0.5], # Teal, Olive, Violet [0.3, 0.7, 0.2], [0.7, 0.3, 0.2], [0.2, 0.3, 0.7], # Custom colors [0.7, 0.2, 0.3], [0.3, 0.2, 0.7], [0.2, 0.7, 0.3], [0.6, 0.4, 0.8], [0.8, 0.6, 0.4], [0.4, 0.8, 0.6], [0.6, 0.8, 0.4], [0.8, 0.4, 0.6], [0.4, 0.6, 0.8], [0.9, 0.1, 0.1], [0.1, 0.9, 0.1], [0.1, 0.1, 0.9], [0.9, 0.9, 0.1], [0.9, 0.1, 0.9], [0.1, 0.9, 0.9], [0.3, 0.6, 0.9], [0.6, 0.3, 0.9], [0.9, 0.6, 0.3], [0.3, 0.9, 0.6], [0.6, 0.9, 0.3], [0.9, 0.3, 0.6], [0.2, 0.8, 0.2], [0.8, 0.2, 0.2], [0.2, 0.2, 0.8], [0.8, 0.8, 0.2], [0.8, 0.2, 0.8], [0.2, 0.8, 0.8], [0.4, 0.7, 0.3], [0.7, 0.4, 0.3], [0.3, 0.7, 0.4], [0.7, 0.3, 0.4], [0.4, 0.3, 0.7], [0.3, 0.4, 0.7] ] # Compute clusters using DBSCAN labels = np.array(cloud.cluster\_dbscan(eps=eps, min\_points=min\_points)) # Extract clusters human\_clusters = [] for label in range(labels.max() + 1): cluster\_indices = np.where(labels == label)[0] cluster = cloud.select\_by\_index(cluster\_indices) # Filter clusters based on size if len(cluster.points) >= min\_points and len(cluster.points) <= max\_points: human\_clusters.append(cluster) if visualize: # Assign fixed colors to clusters clustered\_cloud = o3d.geometry.PointCloud() for i, cluster in enumerate(human\_clusters): color = fixed\_colors[i % len(fixed\_colors)] # Use fixed color for up to 50 clusters cluster.paint\_uniform\_color(color) clustered\_cloud += cluster print(f"Detected {len(human\_clusters)} human clusters.") o3d.visualization.draw\_geometries([clustered\_cloud]) return human\_clustersif \_\_name\_\_ == "\_\_main\_\_": # Load the input point cloud (replace "example.pcd" with your point cloud file) cloud = o3d.io.read\_point\_cloud("example.pcd") print(f"Loaded point cloud with {len(cloud.points)} points.") # Extract human points human\_clusters = extract\_human\_points(cloud, eps=0.5, min\_points=50, max\_points=5000, visualize=True) # Save each detected human cluster as a separate PCD file for i, cluster in enumerate(human\_clusters): filename = f"human\_cluster\_{i + 1}.pcd" o3d.io.write\_point\_cloud(filename, cluster) print(f"Saved: {filename}")import open3d as o3dimport numpy as npdef filter\_front\_clusters(clusters, axis='z', distance\_threshold=5.0): """ Filter the front side of clusters based on their position along a specified axis. Args: clusters (list): List of clustered point clouds. axis (str): Axis to filter along ('x', 'y', or 'z'). Default is 'z'. distance\_threshold (float): Maximum distance along the axis to include points. Returns: filtered\_clusters (list): List of filtered point clouds. """ axis\_index = {'x': 0, 'y': 1, 'z': 2}[axis] filtered\_clusters = [] for cluster in clusters: points = np.asarray(cluster.points) front\_points = points[points[:, axis\_index] < distance\_threshold] # Keep points within threshold if len(front\_points) > 0: front\_cluster = o3d.geometry.PointCloud() front\_cluster.points = o3d.utility.Vector3dVector(front\_points) front\_cluster.colors = cluster.colors # Keep the same color filtered\_clusters.append(front\_cluster) return filtered\_clustersdef extract\_human\_points(cloud, eps=0.5, min\_points=50, max\_points=5000, distance\_threshold=5.0, visualize=False): """ Extract clusters of points corresponding to humans using DBSCAN, and filter the front side. Args: cloud (o3d.geometry.PointCloud): Input point cloud. eps (float): Maximum distance between points to form a cluster. min\_points (int): Minimum points required to form a cluster. max\_points (int): Maximum points allowed in a cluster (to filter out large objects). distance\_threshold (float): Maximum depth (front-side) to keep clusters. visualize (bool): Whether to visualize the results. Returns: front\_clusters (list): List of point clouds corresponding to filtered front-side human clusters. """ # Fixed 50 distinct colors for clusters fixed\_colors = [ [1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0], # Red, Green, Blue [1.0, 1.0, 0.0], [1.0, 0.0, 1.0], [0.0, 1.0, 1.0], # Yellow, Magenta, Cyan # Add additional colors as needed (50 colors total) ] # Compute clusters using DBSCAN labels = np.array(cloud.cluster\_dbscan(eps=eps, min\_points=min\_points)) # Extract clusters human\_clusters = [] for label in range(labels.max() + 1): cluster\_indices = np.where(labels == label)[0] cluster = cloud.select\_by\_index(cluster\_indices) # Filter clusters based on size if len(cluster.points) >= min\_points and len(cluster.points) <= max\_points: human\_clusters.append(cluster) # Filter clusters to only keep the front side front\_clusters = filter\_front\_clusters(human\_clusters, axis='z', distance\_threshold=distance\_threshold) if visualize: # Assign fixed colors to clusters clustered\_cloud = o3d.geometry.PointCloud() for i, cluster in enumerate(front\_clusters): color = fixed\_colors[i % len(fixed\_colors)] # Use fixed color for up to 50 clusters cluster.paint\_uniform\_color(color) clustered\_cloud += cluster print(f"Detected {len(front\_clusters)} human clusters on the front side.") o3d.visualization.draw\_geometries([clustered\_cloud]) return front\_clustersif \_\_name\_\_ == "\_\_main\_\_": # Load the input point cloud (replace "example.pcd" with your point cloud file) cloud = o3d.io.read\_point\_cloud("example.pcd") print(f"Loaded point cloud with {len(cloud.points)} points.") # Extract human points from the front side front\_clusters = extract\_human\_points( cloud, eps=0.5, min\_points=50, max\_points=5000, distance\_threshold=5.0, # Front-side filtering based on depth visualize=True ) # Save each filtered cluster as a separate PCD file for i, cluster in enumerate(front\_clusters): filename = f"front\_human\_cluster\_{i + 1}.pcd" o3d.io.write\_point\_cloud(filename, cluster) print(f"Saved: {filename}")import open3d as o3dimport numpy as npdef select\_points\_in\_roi(cloud, bbox\_min, bbox\_max): """ Select points within a rectangular region of interest (ROI). Args: cloud (o3d.geometry.PointCloud): Input point cloud. bbox\_min (tuple): Minimum x, y, z of the bounding box (ROI corner 1). bbox\_max (tuple): Maximum x, y, z of the bounding box (ROI corner 2). Returns: cropped\_cloud (o3d.geometry.PointCloud): Point cloud cropped to the ROI. """ points = np.asarray(cloud.points) mask = ( (points[:, 0] >= bbox\_min[0]) & (points[:, 0] <= bbox\_max[0]) & # x-axis (points[:, 1] >= bbox\_min[1]) & (points[:, 1] <= bbox\_max[1]) & # y-axis (points[:, 2] >= bbox\_min[2]) & (points[:, 2] <= bbox\_max[2]) # z-axis ) indices = np.where(mask)[0] cropped\_cloud = cloud.select\_by\_index(indices) return cropped\_clouddef extract\_single\_cluster(cloud, eps=0.5, min\_points=50, max\_points=5000, bbox\_min=None, bbox\_max=None, visualize=False): """ Extract a single cluster from a specified rectangular region (ROI) in the front side. Args: cloud (o3d.geometry.PointCloud): Input point cloud. eps (float): Maximum distance between points to form a cluster. min\_points (int): Minimum points required to form a cluster. max\_points (int): Maximum points allowed in a cluster. bbox\_min (tuple): Minimum x, y, z of the bounding box (ROI corner 1). bbox\_max (tuple): Maximum x, y, z of the bounding box (ROI corner 2). visualize (bool): Whether to visualize the results. Returns: single\_cluster (o3d.geometry.PointCloud): Extracted cluster in the ROI. """ # Crop the point cloud to the ROI if bbox\_min is not None and bbox\_max is not None: cloud = select\_points\_in\_roi(cloud, bbox\_min, bbox\_max) # Compute clusters using DBSCAN labels = np.array(cloud.cluster\_dbscan(eps=eps, min\_points=min\_points)) # Extract the largest cluster (optional: select based on size or index) unique\_labels = np.unique(labels) if len(unique\_labels) == 0 or unique\_labels[0] == -1: # No clusters found print("No clusters found in the selected region.") return None # Find the largest cluster clusters = [] for label in unique\_labels: if label == -1: # Ignore noise continue cluster\_indices = np.where(labels == label)[0] cluster = cloud.select\_by\_index(cluster\_indices) if len(cluster.points) <= max\_points: clusters.append(cluster) # Select the desired cluster (e.g., first or largest cluster) single\_cluster = max(clusters, key=lambda c: len(c.points)) if clusters else None if single\_cluster and visualize: single\_cluster.paint\_uniform\_color([1.0, 0.0, 0.0]) # Color the cluster red o3d.visualization.draw\_geometries([single\_cluster]) return single\_clusterif \_\_name\_\_ == "\_\_main\_\_": # Load the input point cloud (replace "example.pcd" with your point cloud file) cloud = o3d.io.read\_point\_cloud("example.pcd") print(f"Loaded point cloud with {len(cloud.points)} points.") # Define the ROI (rectangular area) bbox\_min = (-1.0, -1.0, 0.0) # Minimum corner of the bounding box (x\_min, y\_min, z\_min) bbox\_max = (1.0, 1.0, 2.0) # Maximum corner of the bounding box (x\_max, y\_max, z\_max) # Extract a single cluster in the ROI single\_cluster = extract\_single\_cluster( cloud, eps=0.5, min\_points=50, max\_points=5000, bbox\_min=bbox\_min, bbox\_max=bbox\_max, visualize=True ) # Save the extracted cluster if single\_cluster is not None: o3d.io.write\_point\_cloud("single\_cluster.pcd", single\_cluster) print("Saved single cluster as 'single\_cluster.pcd'.")