```
// Pseudo Code for Identifying True Poles using Effective Point Cloud Techniques

BEGIN PoleDetectionProcess

    // Step 1: Ground Plane and Noise Filtering
    INPUT: raw_pointcloud_data
    OUTPUT: filtered_cloud_without_ground

    FUNCTION RemoveGroundPlane(raw_pointcloud_data)
        INITIALIZE ground_plane_coefficients
        PERFORM RANSAC_Plane_Segmentation on raw_pointcloud_data
        EXTRACT points NOT part of ground plane -> filtered_cloud_without_ground
    END FUNCTION

    // Step 2: Pass-Through Filter for Height-Based Removal
    INPUT: filtered_cloud_without_ground
    OUTPUT: height_filtered_cloud

    FUNCTION ApplyHeightFilter(filtered_cloud_without_ground, min_height, max_height)
        SET min_height_threshold = 0.5m (example value)
        FILTER points from filtered_cloud_without_ground where height > min_height_threshold
        RETURN remaining points as height_filtered_cloud
    END FUNCTION

    // Step 3: Height Filtering for Vertical Structures
    INPUT: height_filtered_cloud
    OUTPUT: vertical_structures_cloud

    FUNCTION FilterVerticalStructures(height_filtered_cloud, min_height, max_height)
        SET height_range = [min_height, max_height]  // Define expected pole height range
        FILTER points in height_filtered_cloud based on height_range
        RETURN points as vertical_structures_cloud
    END FUNCTION

    // Step 4: Euclidean Cluster Extraction for Pole Identification
    INPUT: vertical_structures_cloud
    OUTPUT: pole_clusters

    FUNCTION ClusterExtraction(vertical_structures_cloud, min_cluster_size, max_cluster_size)
        INITIALIZE clustering_parameters with min_cluster_size, max_cluster_size
        APPLY EuclideanClusterExtraction on vertical_structures_cloud using
clustering_parameters
        IDENTIFY clusters that match expected size criteria -> pole_clusters
    END FUNCTION
```

```
    // Step 5: Cluster Analysis to Confirm Pole-Like Structures
    INPUT: pole_clusters
    OUTPUT: detected_poles

    FUNCTION AnalyzeClusters(pole_clusters)
        FOR EACH cluster IN pole_clusters
            CALCULATE cluster_height
            IF cluster_height >= min_pole_height_threshold THEN
                MARK cluster as detected_pole
        END FOR
        RETURN list of detected_poles
    END FUNCTION

    // Calculate midpoints for path generation if left and right poles are detected
    INPUT: detected_poles
    OUTPUT: midpoint_for_path_generation

    FUNCTION CalculateMidPoint(detected_poles)
        IF detected_poles contains both LEFT_POLE and RIGHT_POLE THEN
            CALCULATE centroid of LEFT_POLE and RIGHT_POLE
            COMPUTE midpoint as the average of the two centroids
            RETURN midpoint_for_path_generation
        ELSE
            RETURN NULL  // Skip midpoint calculation if no valid poles detected
    END FUNCTION

END PoleDetectionProcess
```