Wood Pellets Volume Estimation

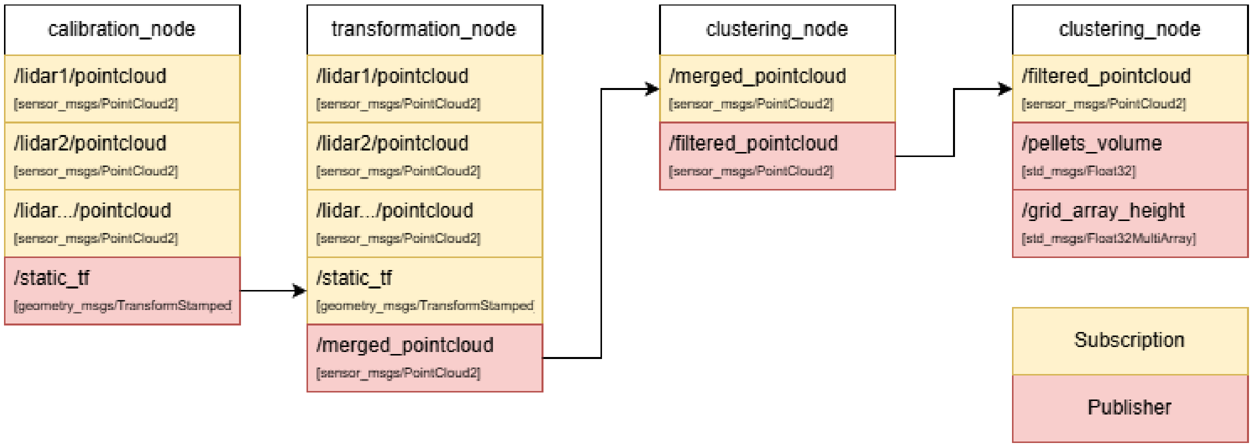
Task description and requirements

This project focuses on estimating the volume of wood pellets stored in a warehouse. The system uses multiple LiDAR sensors mounted above the pile to capture the data. The overall task involves merging the pointclouds from these sensors and then calculating the final volume estimate.

Task Requirements:

- **Input:** Pointcloud data from the LiDAR sensors.
- **Goal:** Estimate the volume of the pellet pile.
- Transformations provided by the measurements can be assumed to have no error.
- Other object present in the scene must be filtered out before estimating the volume.

System Diagram



The system consists of three main nodes: the **Calibration Node**, **Transformation Node**, the **Clustering Node**, and the **Volume Estimate Node**.

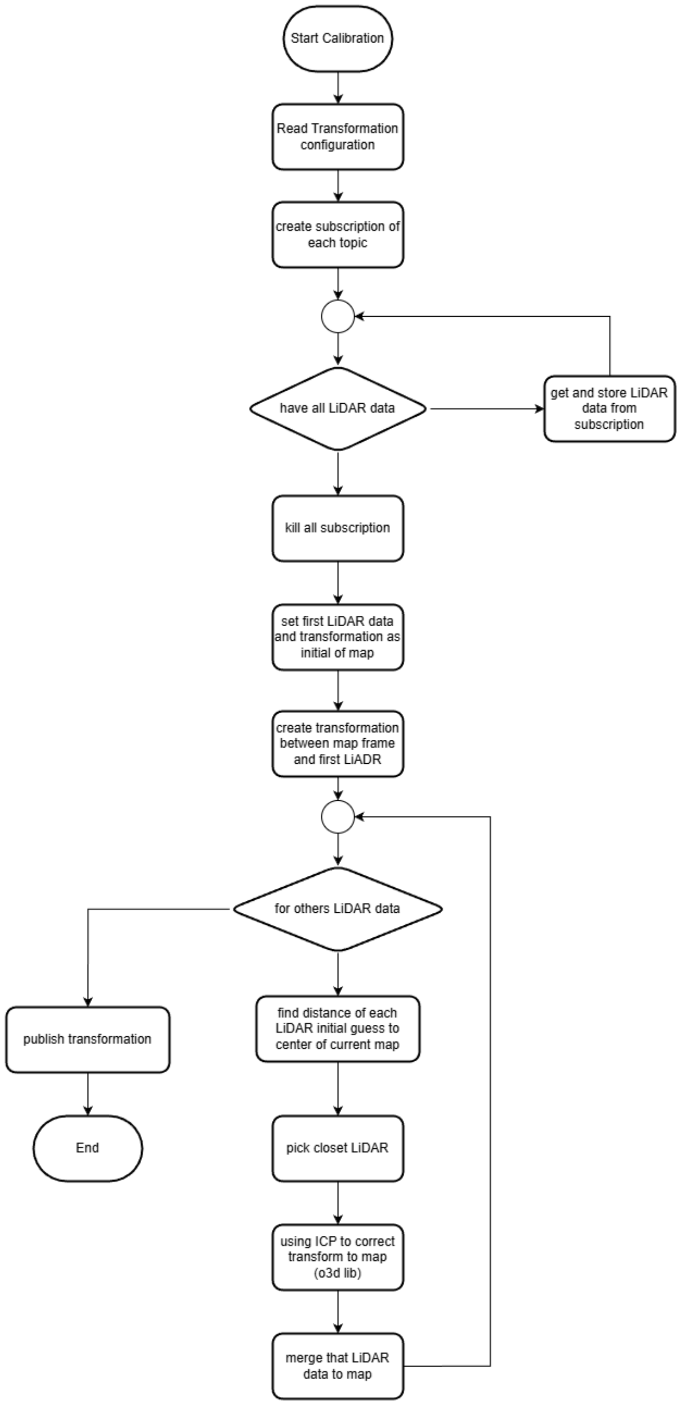
- **Calibration node:** Receives initial transformation data and pointcloud data from multiple sensors and corrects and publish align transformation.
- **Transformation node:** Receives pointcloud data from multiple LiDAR sensors and merges them into a single pointcloud.
- **Clustering node:** Receives the merged pointcloud, filters out objects that are not pellets, and outputs a filtered pointcloud.
- **Volume estimate:** Receives the filtered pointcloud and calculates the total volume of the pellet pile.

Module Details

Calibration node

This module corrects transformation data of each sensor from first initial transformation data from setting.

Inputs: Transformation data, Pointcloud data. **Output:** TF.


Start Calibration
Read Transformation configuration
create subscription of each topic
have all LiDAR data
get and store LiDAR data from subscription
kill all subscription
set first LiDAR data and transformation as initial of map
create transformation between map frame and first LiADR
for others LiDAR data
publish transformation
End
find distance of each LiDAR initial guess to center of current map
pick closet LiDAR
using ICP to correct transform to map (o3d lib)
merge that LiDAR data to map

Initialization and Data Collection:

- **Configuration Reading:** The node begins by reading the transformation configuration files, which contain the "initial guess" of where each LiDAR is located relative to the target map frame.
- **Synchronized Subscription:** It creates subscriptions for all LiDAR topics and enters a loop to collect data.
- **Data Buffering:** Once a complete set of data from all sensors is captured and stored, the node terminates the active subscriptions to conserve processing resources.

Map Initialization:

- **Base Frame Setup:** The first LiDAR's data and its initial transformation are used to establish the "initial of map."
- **Frame Transformation:** A transformation is created between the map frame and this primary LiDAR sensor using initial guess.

Iterative Registration (ICP):

- **Neighbor Selection:** For the remaining LiDAR sensors, the node calculates the distance from their initial guess to the center of the current map.
- **Closest Sensor Priority:** It selects the closest LiDAR to process next, ensuring a more stable and accurate merge.
- **ICP Correction:** Using the **Iterative Closest Point (ICP)** algorithm via the Open3D (o3d) library, the node refines the initial transformation guess to align the sensor's pointcloud precisely with the existing map.
- **Map Expansion:** Once corrected, that LiDAR's data is merged into the map, and the process repeats for the next sensor.

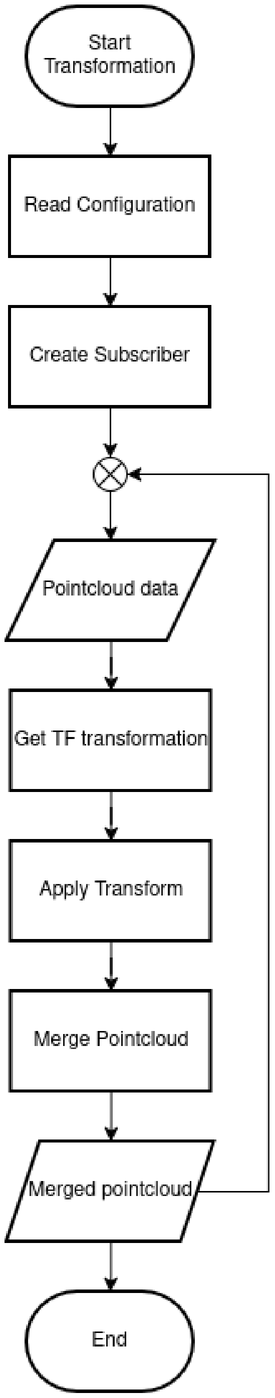
Output:

- **Publish Transformation:** After all sensors are processed and aligned, the final corrected transformation data is published to the /static_tf topic for use by other nodes in the system.

Transformation node

This module handles the application of transformations to the incoming data and merging of the pointclouds.

Inputs: TF, Pointcloud data. **Output:** Merged pointcloud.



Setup Steps (Before Data Arrives)

1. **Read Configuration:** The program first checks what topic of data nedd to subscription by reading the parameters file.
2. **Create Pointcloud Subscriber:** The system sets up a subscriber to listen for pointcloud data. It uses a Time Synchronizer to handle all subscriptions callback.

Working Loop (When Data Arrives)

1. **PointCloud Data Arrives:** The data arrives via the subscriber callback, and the process begins.
2. **Read transformation from TF:** The program uses the data's topic name to look up the correct transformation matrix in the TF which publish from Calibration node.
3. **Apply Transform and add to list:** The matrix is used to change the pointcloud's position and orientation, and the transformed data is saved to array.
4. **Merge point in list:** All pointclouds in the array are merged pointcloud.

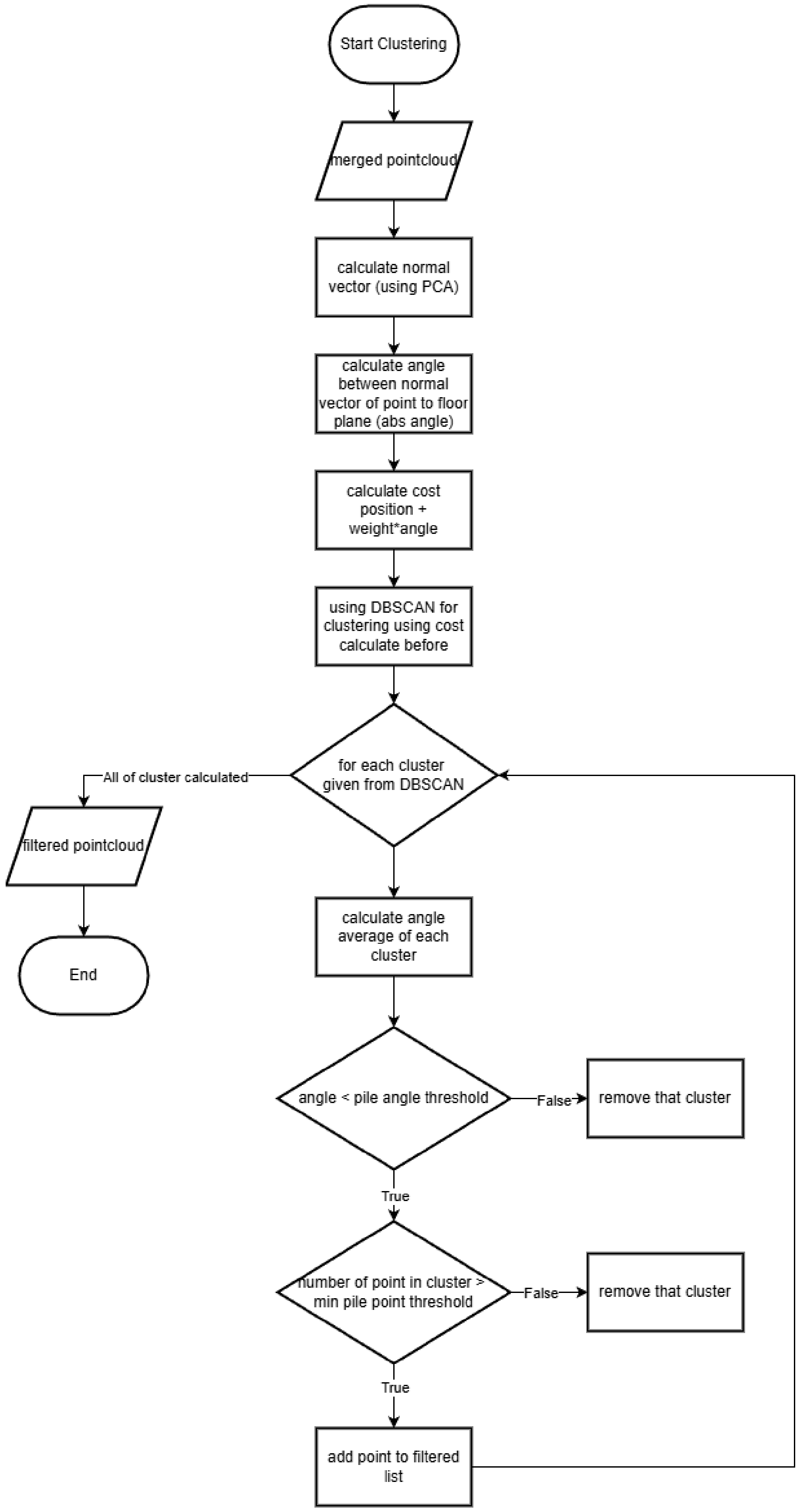
Output and Finish

- The final result is the **Merged pointcloud**.

Clustering node

This module processes the merged pointcloud to identify and isolate the pellet piles by checking their defining geometric characteristic: the angle of repose. The main purpose is to filter out objects that are not part of the pile.

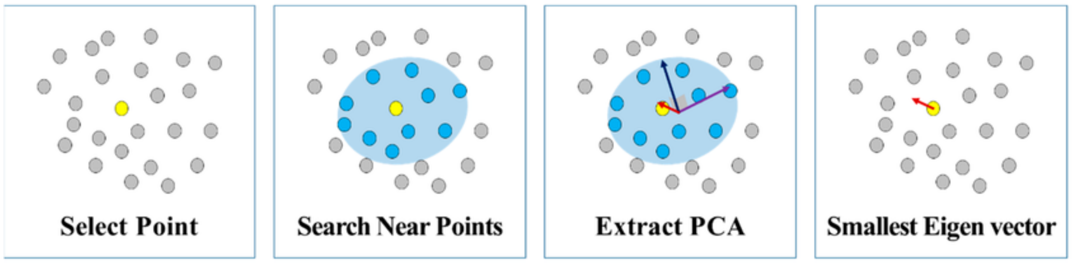
Input: Merged pointcloud. **Output:** Filtered pointcloud.


Start Clustering
merged pointcloud
calculate normal vector (using PCA)
calculate angle between normal vector of point to floor plane (abs angle)
calculate cost position + weight*angle
using DBSCAN for clustering using cost calculate before
for each cluster given from DBSCAN
All of cluster calculated
filtered pointcloud
End
calculate angle average of each cluster
angle < pile angle threshold
False
remove that cluster
True
number of point in cluster > min pile point threshold
False
remove that cluster
True
add point to filtered list

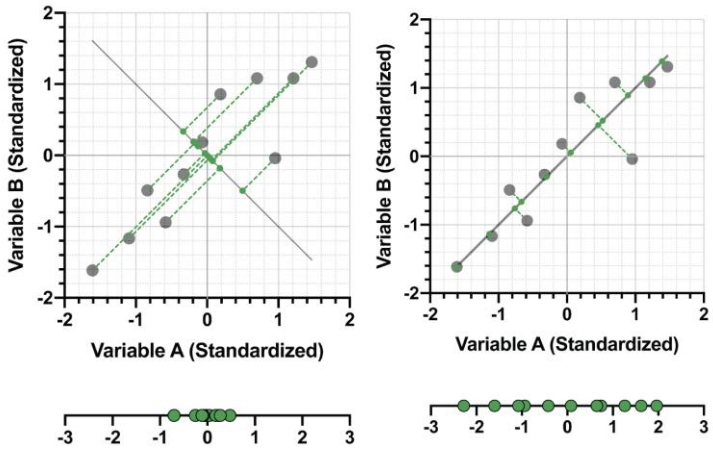
Preparation: Capturing the Slope

The program must measure the slope of the pile to cluster pile from others object.
(A note: If clustering relied only on position, far points might be incorrectly grouped due to lower LiDAR resolution.)

- **Calculate Normal Vector:** Principal Component Analysis (PCA) is used to calculate the normal vector for each point.
    - The process involves finding the point's nearest neighbors and then calculating the normal vector via the covariance matrix.



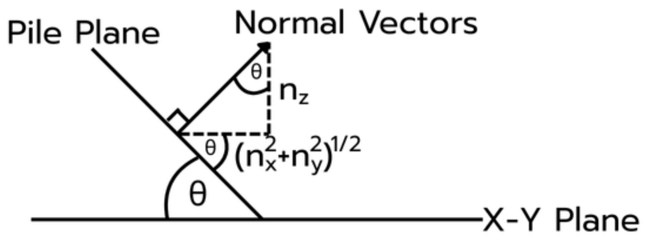
Step for finding Normal Vector using PCA.



Candidate line of first principal component (Left: low variance, Right: high variance (pick PC1))

- **PCA Principle:** PCA calculates a line that passes through the average of the points and gives the highest variance when the points are projected onto it. This is the first principal component (PC1). Subsequent components (PC2, PC3) must be perpendicular to the others. The normal vector corresponds to the direction of least variance (the smallest eigenvector).
- This part using library sklearn using KNN to find near neighbors and PCA for find normal vector.

- **Calculate Cost:** The system combines the point's spatial location (position) with its slope information (angle between pile plane to X-Y plane) to create a cost metric.



- To find the angle θ, we can calculate it using

$$\cos^{-1}\frac{|n_z|}{|normal\ vector|}$$

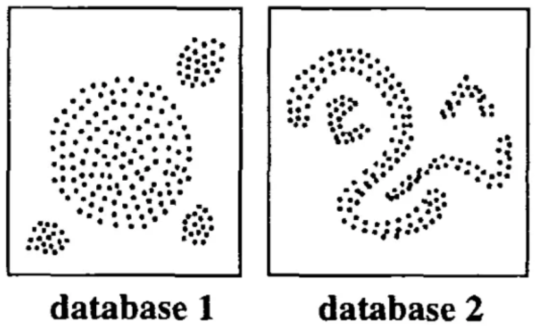
- Since the output is a unit normal vector, the size (length) of the vector is exactly 1. Therefore, the equation becomes simpler:

$$\cos^{-1}|n_z|$$

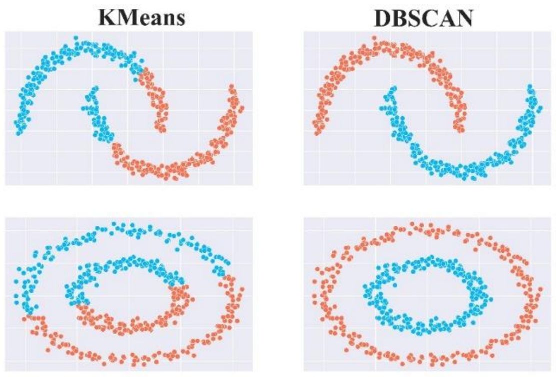
Grouping by Shape (DBSCAN)

The clustering step uses calculated cost with DBSCAN:

- **DBSCAN:** This algorithm is used because it groups points based on density and similarity in the calculated cost.
    - Since the cost includes the slope angle, DBSCAN groups points that are close together and have a similar slope. This is crucial for ensuring that the resulting clusters actually represent a surface of the pile.
    - Method Selection: Other methods, like K-means, separate points based only on distance from a centroid, making them unsuitable for complex, non-separated shapes (like those in the database 2 image). DBSCAN's use of density allows it to separate complex shapes.

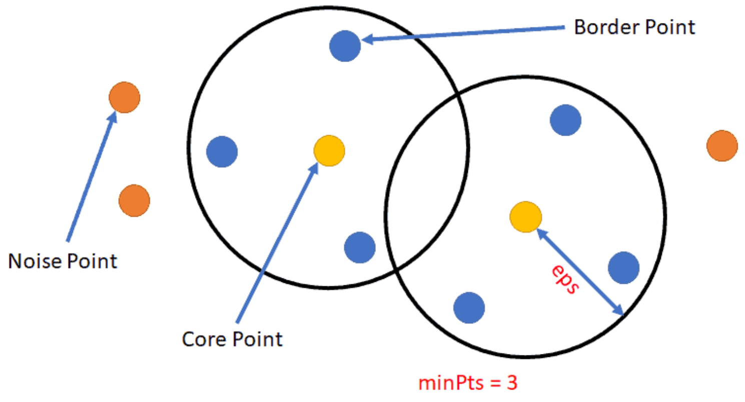


Example dataset shape.



Result compares KMeans and DBSCAN algorithm.

- DBSCAN Operation: For each data point, DBSCAN defines a core point if the number of neighbors within a radius (eps) is equal to or greater than a threshold (minPts). All points in the core point's radius belong to its cluster. If two core points share a neighbor, their clusters are merged. Points that are not part of any core point's neighborhood are classified as Noise Points.

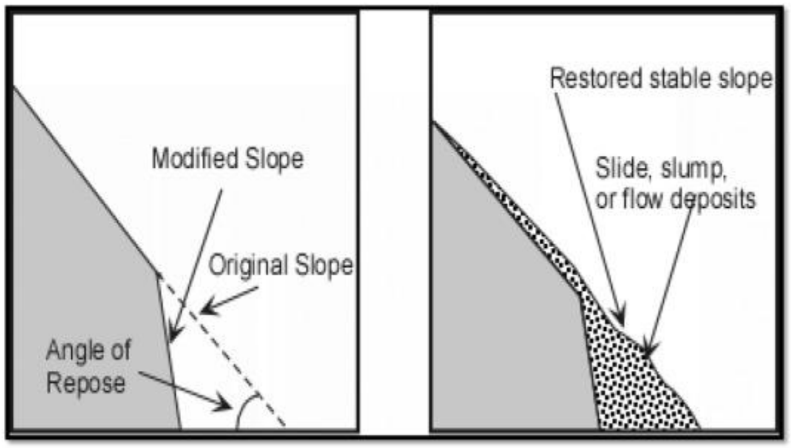


DBSCAN Algorithms

- This part using library sklearn function call DBSCAN.

Filtering: Checking the Characteristic Slope

After clustering, a final filtering step checks the remaining clusters against the material's characteristic angle of repose.



Angle of repose

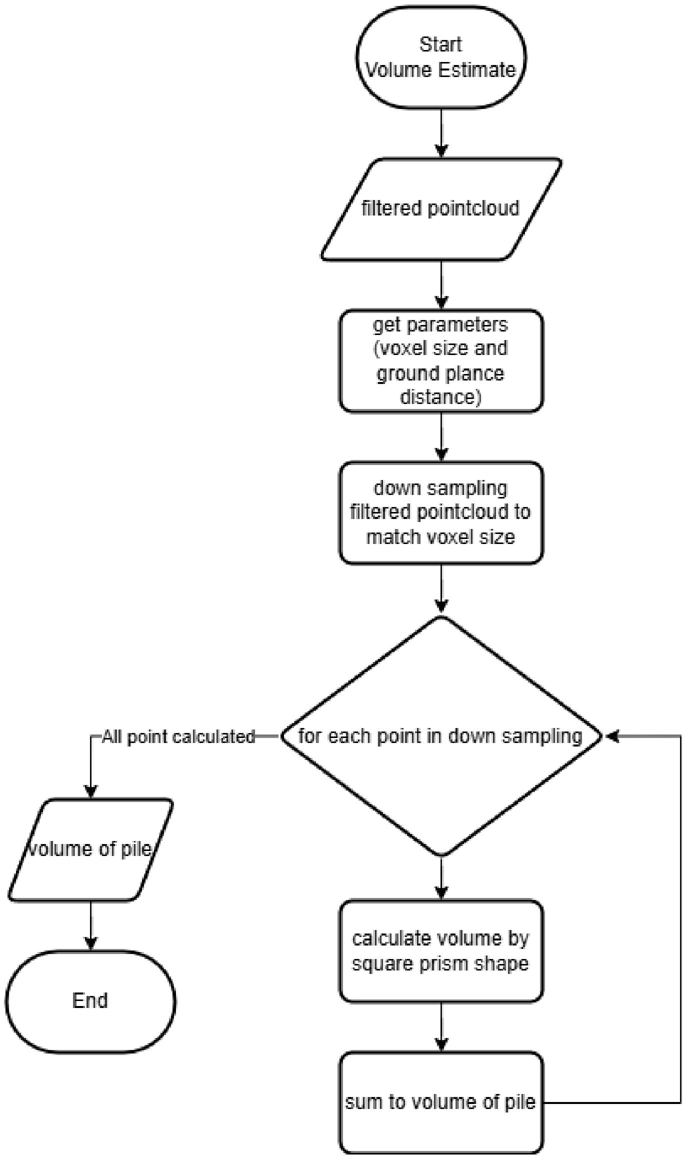
1. **Filter by Tilt (Normal Vector Threshold):**
   - The program calculates the average normal vector (average slope) of each cluster.
   - This average slope is checked against a pile normal vector threshold, which is set to match the maximum or minimum angle of repose expected for the wood pellets.
   - If the cluster's average slope is outside this range (e.g., too flat like the ground, or too steep like a wall), the cluster is removed because it does not have the required pile characteristic shape.
2. **Filter by Size (Point Threshold):**
   - The cluster's size is checked to ensure it contains a number of points greater than a minimum pile point threshold. This ensures the cluster is a meaningful object and not just noise.

The final **filtered pointcloud** contains only the clusters that meet both the expected size and the specific slope defined by the material's **angle of repose**.

Volume estimate

This module calculates the pile's volume using the clean, filtered pointcloud, based on the height of the points above the ground.

Input: Filtered pointcloud. **Output:** Volume of the pile


Start Volume Estimate
filtered pointcloud
get parameters (voxel size and ground plance distance)
down sampling filtered pointcloud to match voxel size
for each point in down sampling
All point calculated
volume of pile
End
calculate volume by square prism shape
sum to volume of pile

Setup and Preparation

1. **Get Parameters:** The module reads two essential parameters:
   - **Voxel size:** Controls the dimensions of the volume blocks (prisms) used for calculation. Smaller voxels increase precision but also computational load.
   - **Ground plane distance:** Defines the bottom plane (Z=0) of the pile.
2. **Down Sampling:** The filtered point cloud is down-sampled to match the defined voxel size (in the X and Y dimensions).
   - The x and y position of the down-sampled point is the middle of the voxel, and the z is the average of the z data grouped within that voxel.

Volume Calculation Loop

The module iterates through the reduced set of down-sampled points:

1. **For each point** (representing the top of a volume block in that area).
2. **Calculate Volume:** The volume is calculated using a **square prism shape**. The area around the point is treated as a vertical block (prism) with a base defined by the voxel size and a height determined by the point's distance from the ground plane.

   Volume = voxel_res * voxel_res * z_height

3. **Sum Total:** It **sum to volume of pile**, adding the volume of that single block to the running total.

Output

- The loop continues until all points are calculated. The final total is the volume of the pile.
- Also publish height of each voxel as Float32MultiArray.