Box Detection - Discussion
Hatef Rahimi & Kiarash Mohsenipour

1. Abstract

This report describes a simple plane‑segmentation pipeline to detect a floor and the top face of a single box from a time‑of‑flight (ToF) point cloud. We fit planar models via RANSAC, generate binary masks, clean them with morphology, and finally estimate the box's height, length, and width. We discuss implementation details, show visual results, identify algorithmic weaknesses, and suggest improvements for robustness, accuracy, and speed.

2. Introduction

Accurate 3D object detection in raw ToF data remains challenging due to sensor noise, missing returns, and complex scene geometry. In this exercise, we target a simple scene containing a flat floor and a rectangular box. By detecting the two dominant planar surfaces, we can extract the box's dimensions from the ToF cloud alone.

3. Methods

3.1 Data Preparation

- **Input**: MATLAB file containing
    1. `cloud4` – an H×W×3 point cloud (X, Y, Z in meters)
    2. `amplitudes4` – matching ToF amplitude image
- **Preprocessing**:
    1. Extract the depth channel (Z) into a 2D array, zero‑masking invalid points (Z = 0).
    2. Randomly subsample 10 000 points for 3D scatter visualization.

3.2 RANSAC Plane Fitting

We implement `ransac_plane_fit(points, threshold, max_iterations)` as follows:

1. **Sampling**: pick three non-collinear points at random (`np.random.choice`).
2. **Plane model**: compute normal $n = (p_2 - p_1) \times (p_3 - p_1)$, normalize to unit length, and form the plane equation

$$ax + by + cz = d,\ d = n \cdot p_1$$

3. **Inlier counting**: measure perpendicular distances $|n \cdot p - d|$ for all points; label those below `threshold` as inliers.
4. **Best model selection**: keep the plane with the largest inlier set over `max_iterations`.

This procedure is run twice:

- First on the full valid-depth cloud to detect the **floor** (horizontal plane).
- Then on the *remaining* points (non-floor) to detect the **box top** (tilted planar face).

3.3 Morphological Cleaning

The raw floor mask often contains small holes and spurious islands. We apply:

1. **Binary closing** (5×5 structuring element) to fill holes inside the floor region.
2. **Binary opening** (3×3) to remove tiny disconnected blobs.

3.4 Connected-Component Filtering

For the box mask (inliers of the second RANSAC), we label all connected components and retain only the largest, under the assumption that the box top is the dominant planar patch aside from the floor.

3.5 Dimension Measurement

- **Height**: distance between the two detected planes, computed by

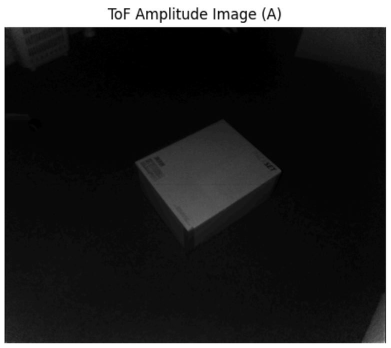
$$h = \frac{|d_{box} - d_{floor}|}{||n||}$$

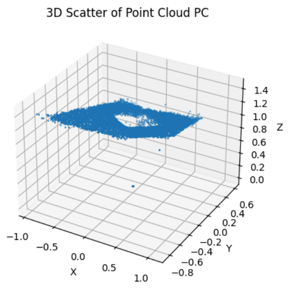
- **Length & Width**: project box-top inliers onto the XY-plane and compute the axis-aligned bounding-rectangle extent:

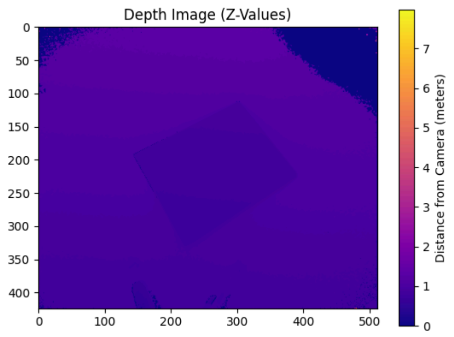
$$length = max(X) - min(X),\ width = max(Y) - min(Y)$$

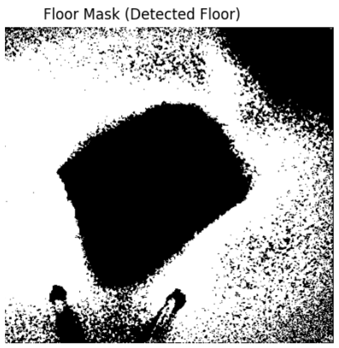
4. Results (for the `example4kinect.mat`)

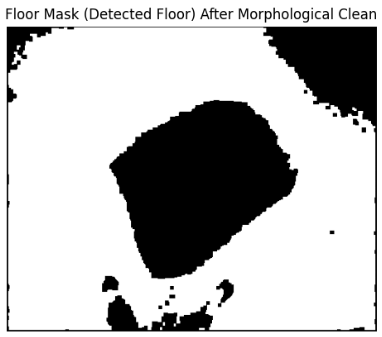
Quantity	Value (m)
Height	0.1837
Length	0.5671
Width	0.4886

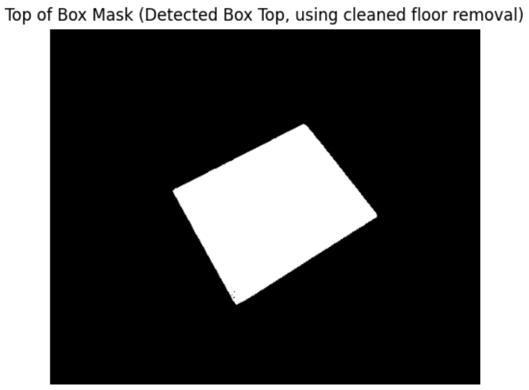


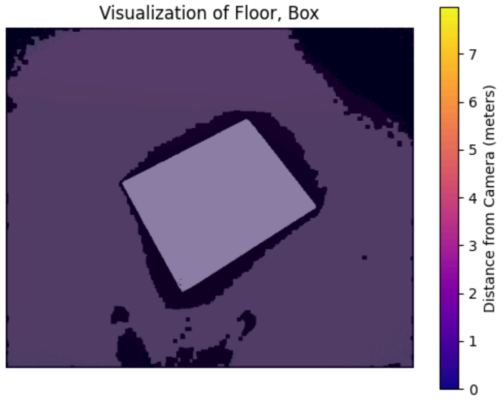









Top of Box Mask (Detected Box Top, using cleaned floor removal)



5. Discussion

Key issues with using plain RANSAC for our box–floor segmentation are:

1. **Threshold Sensitivity**
    - You must pick a fixed “inlier” distance threshold. If it’s too small, genuine plane points get rejected; if too large, you include outliers and skew the fit.
2. **Non-Deterministic & Iteration-Bound**
    - RANSAC relies on random samples. With low inlier ratios (e.g. a small box top amidst many non-planar points), you may need an enormous number of iterations to hit three good points—and still not guarantee the optimal model.
3. **Single‑Model Limitation**
    - Standard RANSAC extracts only one plane at a time. To find a second plane (the box top), you subtract the first inliers and rerun—propagating any errors from the first fit into the second.
4. **Computational Cost**
    - Each iteration tests every point against the candidate plane. Runtime is O(N·Iters), which becomes prohibitive on full-resolution clouds or when you chain multiple RANSACs.

5. **Bias under Structured Outliers**

    - If there are other large planar surfaces (e.g. walls, tables), RANSAC may mistakenly pick those instead of the true floor or box top—since it doesn't enforce any scene-specific priors beyond "largest consensus set." Additionally If your surface is curved or irregular, a single plane won't fit well, so RANSAC's output will be poor.

6. **No Built-In Confidence**

    - RANSAC gives you a model but no measure of how well it describes the data overall (beyond inlier count). You don't know if the chosen threshold or iteration count was sufficient, so you can't automatically detect a "bad" fit.

These weaknesses make a vanilla RANSAC brittle in real-world ToF data—requiring adaptive thresholds, informed sampling (e.g. guided by amplitude or normals), or accelerated/multi-model variants to be robust, accurate, and fast.

6. Recommendations for Robustness, Accuracy & Speed

1. **Adaptive RANSAC**

    - Estimate local depth noise (e.g. via median absolute deviation) and set thresholds per region.

2. **Pre-Filtering**

    - Apply edge-preserving smoothing (e.g. bilateral filter) to reduce outliers before plane fitting.

3. **Rotated Rectification**

    - Compute a 2D convex hull of box-top inliers and fit a minimum-area enclosing rectangle to measure true length/width and orientation.

4. **Area-Based Morphology**

    - Remove connected components by size rather than fixed kernels; fill holes up to a maximum area.

5. **Library Acceleration**

- Use optimized C++/CUDA implementations (PCL, Open3D) for fast plane segmentation and clustering.

6. **Multi‑Model Strategy**

   - Sequentially extract planes sorted by expected normal directions (horizontal, then near‑vertical) to avoid picking walls or other surfaces.

7. Conclusion

The simple RANSAC‑based pipeline successfully measured box dimensions in a controlled ToF scene. However, fixed thresholds, morphological artifacts, and orientation biases limit its robustness. The proposed enhancements—adaptive thresholding, rotated‑rectangle fitting, area‑based cleaning, and optimized libraries—can yield more accurate and faster performance in realistic, cluttered environments.