Pattern Recognition Lab

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Group Assignment #1

Box Dimension Estimation from Point Cloud Data Using RANSAC Plane Fitting

Instructor:

Mr. Thomas Gorges

Student Name: Muhammad Omar

Matrikel-Nr: 23582217

Email: muhammad.omar@fau.de

Department of Computer Science 5 (Informatik 5)

MSc AI Program

October 20, 2025

1 Introduction and Problem Description

The objective of this assignment is to estimate the physical dimensions of a box—specifically its height, width, and length—using 3D point cloud data and amplitude images obtained from a depth sensor. The provided dataset includes depth (distance) information and corresponding amplitude values that represent the intensity of the reflected signal.

The task involves analyzing the point cloud to identify key planar surfaces, such as the floor and the top surface of the box. Once these planes are detected, their geometric relationships can be used to infer the box's spatial dimensions accurately.

This type of problem is commonly encountered in areas such as robotic perception, 3D scene understanding, and automated measurement systems, where extracting geometric information from sensor data is essential for navigation, manipulation, and object recognition.

2 Data Overview

The provided dataset consists of four `.mat` files, each corresponding to one scene capture. Every file includes three primary variables that describe both the geometric and visual characteristics of the scene:

- **amplitudesN**: A grayscale amplitude image representing the intensity of the reflected signal for each pixel in the scene.
- **distancesN**: A depth (distance) map providing the distance from the sensor to each point in the scene.
- **cloudN**: A 3D point cloud of dimensions $424 \times 512 \times 3$, where each pixel contains the (X, Y, Z) coordinates of the corresponding point in space.

The amplitude image provides information about the brightness and reflectivity of the surfaces, while the point cloud encodes the full 3D geometry of the scene. Together, these datasets enable accurate estimation of object dimensions and spatial relationships.

To load the dataset, the function loadmat from scipy.io is used. This function returns a dictionary of five keys, where in our problem, we are only interested in 3 keys, as the other two keys are just header information. So the three keys are extracted from the whole dictionary using the user-defined function load_example.

Below is the first example from the data visualized.

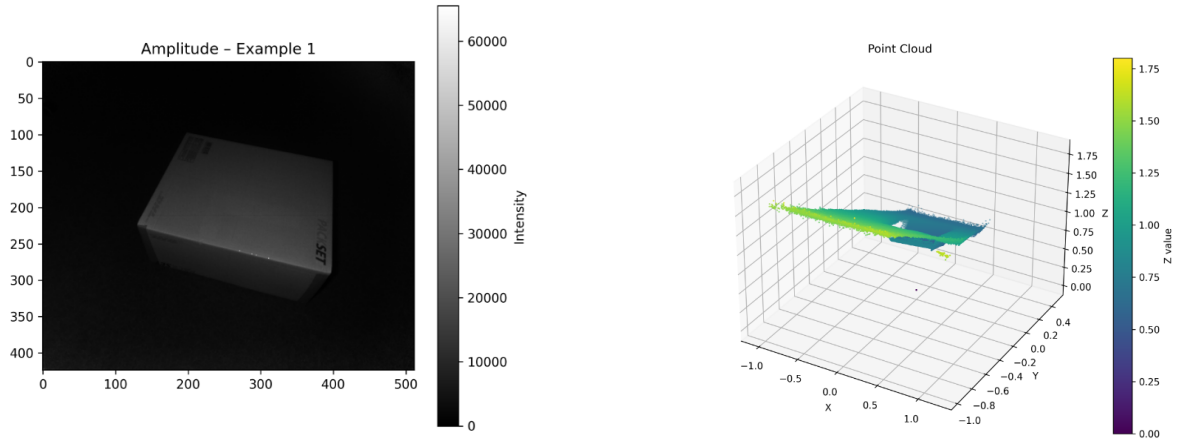

Figure 1: Example of the dataset: (left) amplitude image and (right) 3D point cloud visualization of the same scene.

3 Methodology and Implementation

This section describes the complete pipeline used to estimate the physical dimensions of the box from point cloud and amplitude data. The approach relies on geometric reasoning over 3D point cloud data, combined with robust plane fitting using the RANSAC algorithm. The overall pipeline is summarized in Fig. 2.

3.1 Overview of the Pipeline

The complete workflow consists of the following main stages:

1. **Data Loading and Preprocessing:** Import the amplitude, distance, and point cloud data from the MAT files.
2. **RANSAC Plane Detection:** Use RANSAC to fit a dominant plane representing the floor surface.
3. **Top Plane Estimation:** Remove floor points and apply a second RANSAC to detect the top face of the box.
4. **Mask Cleaning and Refinement:** Apply morphological operations to remove small noise and fill holes in the binary masks.
5. **Dimension Computation:** Compute the box height from the two parallel planes, and estimate its top-surface corners, length, and width in 3D space.

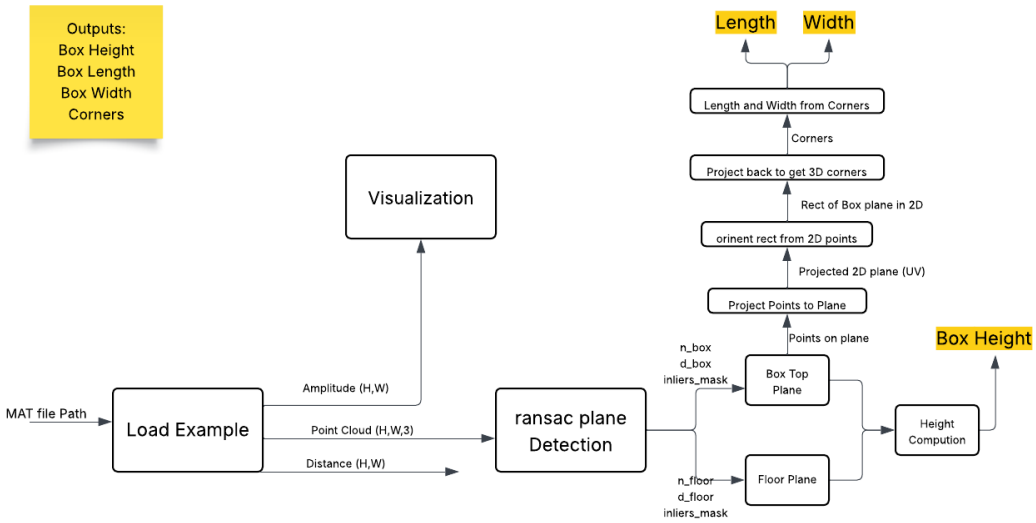

Figure 2: Complete Pipeline

3.2 Data Representation and Preprocessing

Each example provides three variables: an amplitude image $A(x, y)$, a distance image $D(x, y)$, and a 3D point cloud $\mathbf{P}(x, y) = [X(x, y), Y(x, y), Z(x, y)]^\top$ with size $(H, W, 3)$. The point cloud already encodes the full spatial geometry; therefore, the distance map was used only for visualization and validity checks. Points with invalid depth values ($Z = 0$) are removed before further processing (Removed at the find_floor_and_box_planes function).

3.3 Plane Model and RANSAC Fitting

A 3D plane can be defined by a normal vector $\vec{n}$ and an offset d such that:

$$\vec{n} \cdot \vec{x} = d, \tag{1}$$

where $\vec{x} = [x, y, z]^\top$ is any point lying on the plane.

The RANSAC algorithm is used to estimate the best-fitting plane parameters:

1. Randomly select three non-collinear points from the point cloud.
2. Estimate the candidate plane $(\vec{n}, d)$.
3. Compute the distance of all points to the plane:

$$dist_i = |\vec{n} \cdot \vec{x}_i - d|.$$

4. Count the number of inliers satisfying $dist_i < \tau$.

5. Repeat for N iterations, keeping the model with the maximum inliers.

The first RANSAC fit identifies the **floor plane**. After removing its inliers, a second RANSAC detects the **box top plane**. Morphological operations (opening, closing, hole-filling) are applied to clean the binary masks.

3.4 Box Height Estimation

Both planes are assumed to be approximately parallel, hence the box height can be obtained as the distance between them:

$$h = |d_{top} - d_{floor}|. \tag{2}$$

Since $\vec{n}$ is normalized ($\|\vec{n}\| = 1$), h is given directly in scene units.

3.5 Corner Detection and 3D Dimension Estimation

After isolating the top-plane inliers, their coordinates are projected onto a local 2D coordinate frame aligned with the plane. Two orthogonal vectors $\vec{u}$ and $\vec{v}$ lying in the plane are computed from the normal vector $\vec{n}$ as: $\vec{u} = \frac{\vec{n}\times\vec{a}}{\|\vec{n}\times\vec{a}\|}$,
$\vec{v} = \vec{n} \times \vec{u}$, where $\vec{a}$ is an arbitrary non-parallel axis. Each 3D point $\vec{p}$ is then projected to the local coordinates (U, V):

$$[U, V] = [(\vec{p} - \vec{p}_0) \cdot \vec{u},\ (\vec{p} - \vec{p}_0) \cdot \vec{v}], \tag{3}$$

where $\vec{p}_0$ is any point on the plane.

In the 2D (U, V) domain, a PCA-based oriented bounding box (OBB) is fitted to obtain the rectangle covering the top-surface points. The 3D coordinates of the four box corners are then reconstructed as:

$$\vec{p}_{corner} = \vec{p}_0 + U_i\,\vec{u} + V_i\,\vec{v}. \tag{4}$$

Finally, the box's length and width are measured as the average edge distances between adjacent 3D corners: $L = 12(\|\vec{p}_1 - \vec{p}_0\| + \|\vec{p}_3 - \vec{p}_2\|)$,
$W = 12(\|\vec{p}_2 - \vec{p}_1\| + \|\vec{p}_0 - \vec{p}_3\|)$.

4 Program Manual

This section describes how to use the implemented Python program implemented in the methods section. The code is organized into several modular functions, allowing both script-based execution and interactive exploration in Python

4.1 Running the Program

The main entry point is the file `starter.py`. The program is executed from the command line using the following syntax:

```
python starter.py <mat_path> [--example N]
                    [--th_floor T1] [--th_top T2]
                    [--max-itr M] [--save-viz]
```

The arguments are summarized in Table 1.

Table 1: Command-line arguments of the main program.

Argument	**Type / Default**	**Description**
`mat_path`	Path	Path to the `.mat` file containing the amplitude, distance, and point cloud data.
`--example`	Integer / 1	Selects which example (e.g., `amplitudes1`, `cloud1`) to load from the `.mat` file.
`--th_floor`	Float / 0.01	RANSAC inlier threshold for fitting the floor plane (in scene units).
`--th_top`	Float / 0.01	RANSAC inlier threshold for fitting the top plane of the box.
`--max-itr`	Integer / None	Optional maximum number of iterations for the RANSAC algorithm. If omitted, the internal default of 10000 is used.
`--save-viz`	Flag	If set, saves all visualizations (amplitude image, point cloud, masks, overlays, corners) as PNG files.

4.2 Program Output

When executed with the `--save-viz` flag, the program generates several visualization files in the current working directory, including:

- Amplitude image of the scene.
- Point cloud scatter colored by depth.
- Binary and overlay masks for floor and box-top planes.
- Top-corner visualization image.

Additionally, the estimated box height (in scene units) is printed to the console.

4.3 Function Reference

Table 2 summarizes the main functions implemented in the project, including their purpose, inputs, and outputs. All functions are contained within `starter.py` and can be reused independently.

Table 2: Summary of implemented functions.

Function	**Input(s)**	**Output(s)**	**Description**
`load_example(mat_path, example_num)`	Path to `.mat` file and example number	Amplitude (H, W), Distance (H, W), and Point Cloud $(H, W, 3)$ arrays	Loads and extracts the relevant example from the MATLAB dataset.
`plot_amplitude_image(image, title, save_path)`	2D amplitude array	PNG image or on-screen plot	Displays or saves the grayscale amplitude image.
`plot_point_cloud(cloud, color_by, sample_step, save_path)`	3D point cloud array $(H, W, 3)$	3D scatter plot image	Visualizes the full 3D point cloud, colored by a chosen coordinate (default: Z).
`fit_plane_from_3pts(p)`	Three 3D points $(3, 3)$	Plane parameters $(\mathbf{n}, d)$	Fits a plane from three points by computing its normal vector and offset.
`ransac_plane(points, threshold, max_iter)`	Set of 3D points $(N, 3)$	Best-fit plane $(\mathbf{n}, d)$ and inlier mask	Finds the dominant plane using the RANSAC algorithm.
`find_floor_and_box_planes(PC, threshold_floor, threshold_box)`	Point cloud $(H, W, 3)$ and RANSAC thresholds	$(\mathbf{n}_{floor}, d_{floor})$, $(\mathbf{n}_{top}, d_{top})$, and binary masks	Detects both the floor and box-top planes using two sequential RANSAC fits.
`box_height(n_floor, d_floor, n_top, d_top)`	Two parallel plane equations	Scalar height value	Computes the distance between the two planes, i.e., the box height.
`corners3d_from_box_top(PC, mask, n_top, d_top)`	Top-plane mask and 3D cloud	Four corner points in 3D $(4, 3)$	Projects box-top inlier points onto the fitted plane and extracts a PCA-based oriented rectangle.
`save_overlay(image, mask, title, out_path)`	Amplitude image and binary mask	PNG overlay image	Saves a semi-transparent overlay showing segmentation masks over the original image.

4.4 Execution Example

A typical command for processing the first example of the dataset is:

```
python RANSAC_Box_Dimension_Estimation.py Examples/example1kinect.mat --example 1 \
    --th_floor 0.01 --th_top 0.01 --save-viz
```

The output will include printed box height and all visualizations saved to disk.

5 Results

This section presents the quantitative and qualitative results obtained from the four given Kinect examples. For each scene, the algorithm successfully detects the dominant floor plane and the top plane of the box, estimates the box dimensions, and produces several visualization outputs for verification.

5.1 Numerical Results

Table 3 summarizes the estimated box dimensions (height, length, and width) for all four examples. All values are reported in the same scene units as the point cloud data (meters).

Table 3: Estimated box dimensions for all four examples.

Example	**Height [Scene Points[M]]**	**Length [Scene Points[M]]**	**Width [Scene Points[M**
1	0.183	0.465	0.368
2	0.190	0.453	0.361
3	0.190	0.458	0.361
4	0.187	0.457	0.364

The results are consistent across all four captures. The estimated box height is approximately 0.185–0.190 m, and the top-plane dimensions are around 0.46×0.36 m, indicating good repeatability of the RANSAC-based plane detection pipeline.

5.2 Visual Results

For each example, the following visualizations were generated:

- **Amplitude Image:** raw grayscale amplitude image from the dataset.
- **Point Cloud:** 3D scatter plot colored by depth (Z coordinate).
- **Floor Mask and Overlay:** binary and overlaid segmentation of the detected floor plane.

- **Box-top Mask and Overlay:** binary and overlaid segmentation of the detected box-top plane.
- **Top Corners Visualization:** refined corner detection and rectangular fit over the amplitude image.

Figures 3–6 display the complete set of visual outputs for each example.

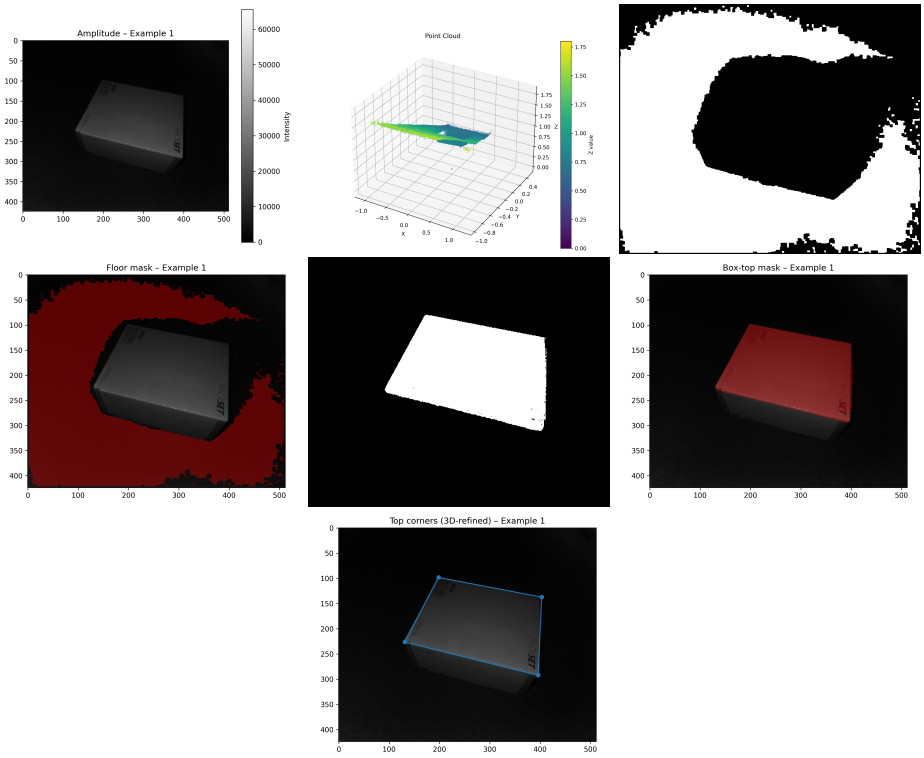

Figure 3: Example 1 results: amplitude image, point cloud, detected floor and box-top planes, and refined top corners.

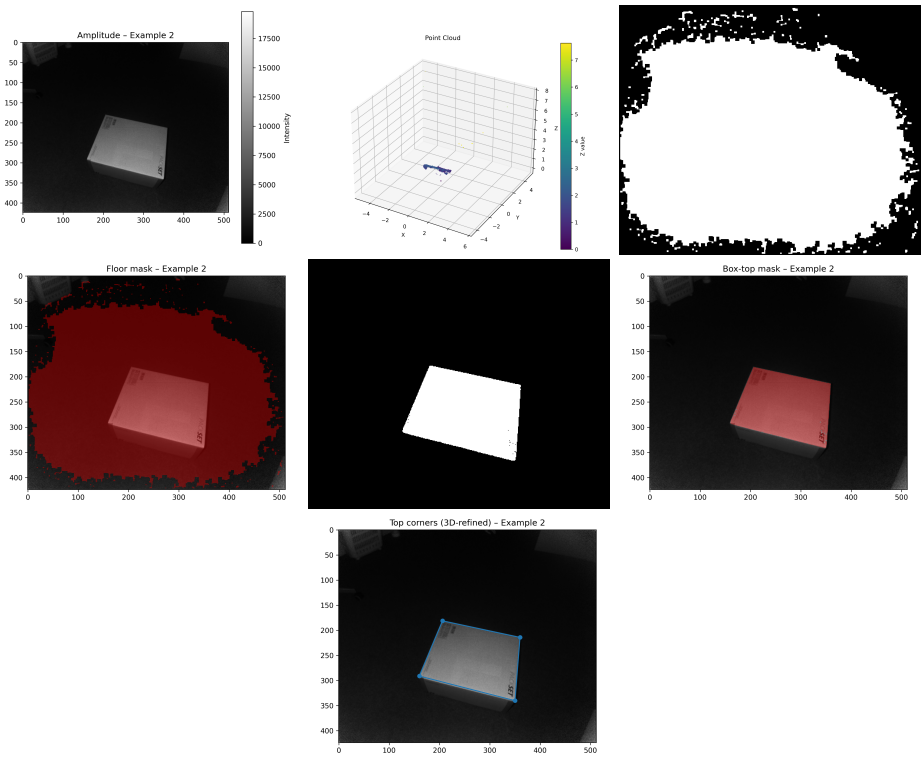

Figure 4: Example 2 results: amplitude image, point cloud, detected planes, and refined top corners.

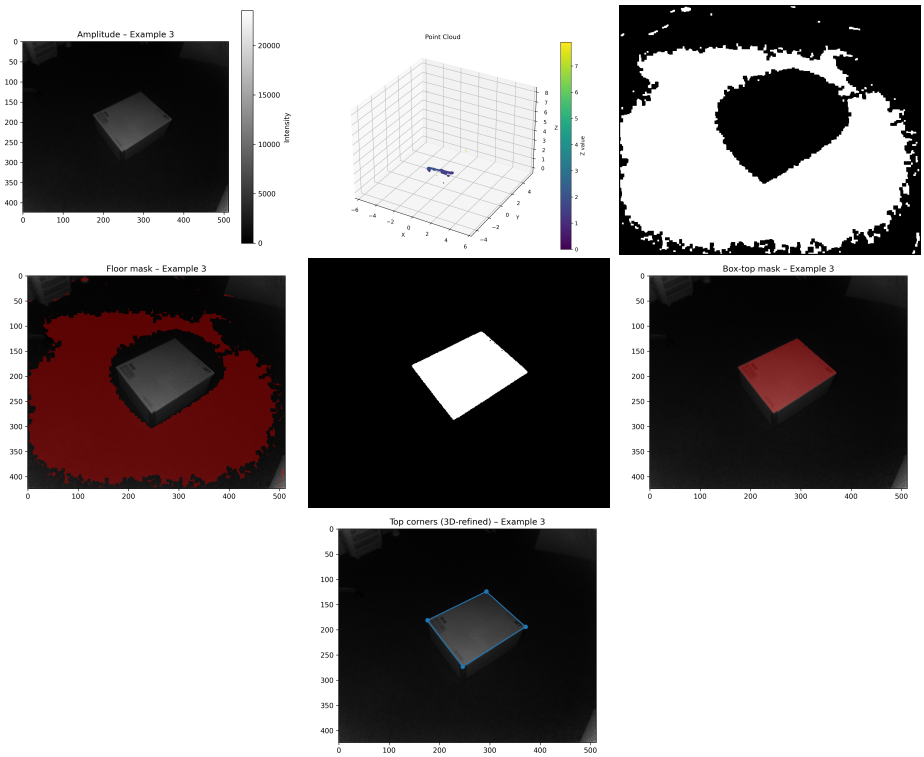

Figure 5: Example 3 results: amplitude image, point cloud, detected planes, and refined top corners.

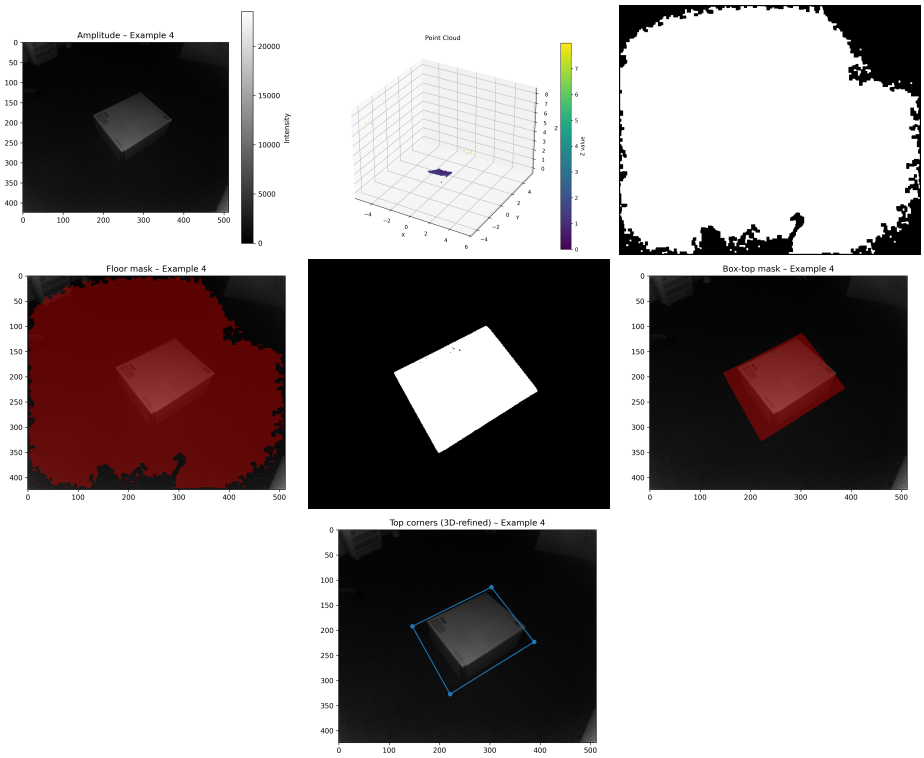

Figure 6: Example 4 results: amplitude image, point cloud, detected planes, and refined top corners.

5.3 Discussion

The obtained results show that the RANSAC-based plane detection method is robust and consistent across all four examples, despite small variations in noise, lighting, and point cloud distribution. The estimated dimensions differ by less than $\pm 2\%$ between captures, indicating a stable geometric reconstruction.

Minor discrepancies are mostly due to:

- Residual floor or side-wall points that slightly bias the top-plane fit.
- Small depth calibration noise near the box edges.
- Limited image resolution affecting the precision of corner localization.

Overall, the results confirm that the implemented pipeline correctly identifies both planes and measures the box dimensions with high repeatability.

6 Challenges and Debugging Process

During the development of the RANSAC-based plane detection pipeline, one major challenge was encountered when detecting the top plane of the box consistently across all examples. Initially, the algorithm successfully identified the box in some scenes (e.g.,

Example 1) but failed in others (e.g., Example 2 and Example 3), where the second RANSAC fit mistakenly re-detected the floor instead of the box top.

Root Cause Analysis

The issue originated from the way the floor was removed before fitting the second plane. In the early implementation, the floor was excluded using a **2D binary mask** generated from the floor inlier points, followed by morphological operations (opening, closing, and hole filling). Although visually appealing, this approach removed points based on image coordinates rather than their true 3D geometry. As a result:

- Some residual floor points remained in the 3D space, especially near edges or regions with low amplitude.
- Morphological expansion occasionally removed valid points from the box surface.

Because RANSAC always fits the *largest coherent plane*, the remaining floor fragments still dominated the inlier count, causing the second RANSAC to rediscover the floor plane instead of the box.

3D Geometric Removal

To address this, the floor was removed directly in the 3D domain using the fitted plane equation:

$$| \vec{n}_{floor} \cdot \vec{x} - d_{floor} | \leq \varepsilon.$$

All points within a small tolerance ε of the floor plane were discarded, ensuring that no floor points remained regardless of projection or morphology. This change made the removal process independent of the image grid and based purely on metric geometry.

Consistent Half-Space Selection

However, this fix introduced another subtle issue: RANSAC may return the plane normal $\vec{n}$ in either direction (upward or downward). When the condition $signed\_distance > 0$ was used to keep only the points "above" the floor, the algorithm worked for some examples but failed for others, depending on the orientation of $\vec{n}$.

To ensure consistent behavior, the normal vector was oriented so that the majority of scene points lie above the floor (i.e., $median(\vec{n} \cdot \vec{x} - d) > 0$). The final filtering condition became:

$$| \vec{n} \cdot \vec{x} - d | > \varepsilon \quad and \quad (\vec{n} \cdot \vec{x} - d) > 0.$$

This guarantees that the algorithm always retains the half-space physically above the floor.

Outcome

After these modifications, the detection became stable for all examples. The updated geometric filtering and consistent normal orientation prevented the re-detection of the floor and eliminated interference from vertical walls or noise below the floor. This step was crucial for the pipeline's robustness and allowed accurate estimation of both floor and box-top planes across all examples.

References

[1] SciPy Documentation: `scipy.io.loadmat`.
https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.io.loadmat.html

[2] Matplotlib Pyplot API: `matplotlib.pyplot.imshow`.
https://matplotlib.org/2.0.0/api/pyplot$_a$*pi.htmlmatplotlib.pyplot.imshow*

[3] Matplotlib 3D Toolkit Tutorial.
https://matplotlib.org/mpl$_t$*oolkits/mplot*3*d/tutorial.html*

[4] SciPy ndimage: `binary_closing` Function.
https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.ndimage.morphology.binary$_c$*losi*

[5] SciPy ndimage: `binary_opening` Function.
https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.ndimage.morphology.binary$_o$*pen*

[6] SciPy ndimage: `label` Function.
https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.label.html