10xConstruction

Perception Task: Cuboid Rotation Analysis

An Algorithmic Approach to 3D Perception and State Estimation

Prepared for **10x Construction**

Important Links:

Github: https://github.com/amanchauhan786/10xconstruction

Google Collab: https://colab.research.google.com/drive/1RSEs2QPuAEEv_O-skvZqSfNFIhSX2-gJ?usp=sharing

Prepared by

Aman Chauhan

22BCE0476

Department of Computer Science & Engineering

Vellore Institute of Technology Vellore - 632014, INDIA

October 2, 2025

1.0 Introduction

1.1 Project Objective

The primary objective of this project is to analyse a sequence of depth images from a ROS bag file to estimate the physical properties of a rotating 3D cuboidal box. The task, provided by 10x Construction, requires the implementation of a perception algorithm to determine two key properties for each timestamp in the data:

1. The **normal angle** and **visible surface area** (in m²) of the largest visible face of the box with respect to the camera.
2. A single, stable **axis of rotation vector** that describes the box's motion throughout the sequence.

All input data is provided in standard SI units, and the final deliverables include the processing script, a data table, the rotation vector, and this document detailing the approach.

1.2 Summary of Approach and Key Findings

To achieve the project's objective, a multi-stage data processing pipeline was developed. The algorithm begins **by loading and decoding the raw 16UC1 encoded depth data** from the ROS bag, which is then corrected from **millimeters to meters** and converted into a series of **3D point clouds**. A **robust filtering mechanism** is applied to isolate the cuboid from its background.

The core of the algorithm uses the **RANSAC method** to perform **planar segmentation on the point cloud**, identifying the largest visible face of the box. From this face, the normal vector and visible area are calculated. To determine the axis of rotation, a robust Principal Component Analysis (PCA) was employed on the sequence of normal vectors, yielding a more stable and accurate result than simpler methods.

The algorithm successfully **processed all frames**, and the key findings validate the approach. The final estimated **axis of rotation** was found to be **[0.9995, -0.0175, -0.0273]**. Furthermore, the relationship between the calculated normal angle and visible area showed a **strong inverse correlation**, confirming the physical plausibility of the results. The project culminates in a **3D animation that provides a clear visual** confirmation of the cuboid's smooth rotation around the estimated axis.

2.0 Data Processing and Preparation

A robust perception algorithm must begin with a thorough data preparation pipeline. The raw data from the **ROS bag file required several processing and correction steps** before it could be used for geometric analysis. This section details the methods used to **load, decode, and prepare the depth data** for the core algorithm.

2.1 Loading and Decoding ROS Bag Data

The input data was provided as a ROS 2 bag, which stores serialized message data in a **depth.db3 SQLite database file**. The Python rosbags library was used to parse this file and **deserialize** the messages on the /depth topic.

Upon initial inspection, it was discovered that the depth images were not in a standard float format but were encoded as **16-bit unsigned integers (16UC1)**. This is a **common format for depth sensors** where each pixel value directly represents a distance. The data loading function was specifically designed to handle this encoding.

Code Snippet: Decoding 16UC1 Data

Python
```
# From the helper function `ros_image_to_numpy`
if msg.encoding == '16UC1':
    # Handle 16-bit unsigned integer format (common for depth in mm)
    return np.frombuffer(msg.data, dtype=np.uint16).reshape(msg.height, msg.width)
```

A crucial finding during this stage was that the integer values represented distance in **millimeters**. The assignment specified that all data was **in SI units**, so a unit conversion was required. This was handled in the next step.

2.2 3D Point Cloud Generation

To perform 3D analysis, each **2D depth image was converted into a 3D point cloud**. This was achieved by applying the **pinhole camera model**, which projects **each pixel (u, v) with a depth value d to a 3D point (X, Y, Z).** Since the camera's intrinsic parameters (focal length fx, fy; principal point cx, cy) were not provided, they were assumed based on the image dimensions, a standard practice for uncalibrated scenarios.

During this conversion, the necessary unit correction was applied by dividing each **depth value by 1000.0, converting the measurements from millimeters to meters.**

Code Snippet: Depth-to-Point-Cloud Conversion

Python
```
# From the helper function `depth_to_point_cloud`
def depth_to_point_cloud(depth_image, fx, fy, cx, cy):
    points = []
    height, width = depth_image.shape
    for v in range(height):
        for u in range(width):
            d = depth_image[v, u]
            if d > 0:
                z = d / 1000.0  # Convert depth from mm to meters
                x = (u - cx) * z / fx
                y = (v - cy) * z / fy
                points.append([x, y, z])
    return np.array(points)
```

2.3 Background Removal and Object Isolation

Initial processing revealed a significant outlier in the first frame, where the calculated area was an order of magnitude larger than in subsequent frames. This was caused by the **RANSAC algorithm** detecting the large, flat wall behind the cuboid instead of the cuboid itself.

To solve this, a depth-based filtering step was implemented. The algorithm first calculates the **median depth of the entire point cloud** to approximate the location of the main object. It then discards all points that fall outside a defined "slab" (in this case, ±0.25 meters) around this median depth. This effectively isolates the cuboid from its background, ensuring that subsequent algorithms only analyze points belonging to the object of interest.

Code Snippet: Background Filtering

Python
```
# From the main processing loop
if point_cloud.shape[0] > 0:
    median_z = np.median(point_cloud[:, 2])
    mask = np.abs(point_cloud[:, 2] - median_z) < 0.25
    point_cloud = point_cloud[mask]
```

3.0 Algorithm and Methodology

Once the 3D point cloud was loaded, cleaned, and filtered, the core algorithmic phase began. This stage involved **identifying the planar faces of the cuboid**, calculating their **geometric properties**, and determining the **overall axis of rotation for the object.**

3.1 Planar Segmentation using RANSAC

The primary challenge in analyzing the point cloud is to distinguish which points belong to the flat face of the cuboid and which are noise. The **RANSAC (Random Sample Consensus)** algorithm is exceptionally well-suited for this task as it is highly robust to outliers.

The algorithm iteratively performs the following steps:

1. Selects a random subset of points from the point cloud.
2. Fits a plane to this subset.
3. Checks all other points in the cloud to determine how many fall within a small tolerance (the "inliers") of this plane.
4. Repeats this process many times, keeping the plane that results in the highest number of inliers.

This approach reliably finds the dominant plane in the data, which corresponds to the largest visible face of the box. The implementation was handled **using Scikit-learn's RANSACRegressor.**

Code Snippet: Applying RANSAC to the Point Cloud

Python
```
# From the main processing loop
from sklearn.linear_model import RANSACRegressor

# We want to predict Z from X and Y: z = a*x + b*y + d
ransac = RANSACRegressor(residual_threshold=0.01).fit(point_cloud[:, 0:2],
point_cloud[:, 2])
inlier_mask = ransac.inlier_mask_

# The points that belong to the largest face
face_points = point_cloud[inlier_mask]
```

3.2 Calculation of Face Properties (Normal Angle and Visible Area)

With the set of **face_points** identified by RANSAC, the two required metrics were calculated for each frame:

1. **Normal Angle:** The **RANSACRegressor** provides the coefficients of the best-fit plane equation **(z=ax+by+d).** From these, the **plane's normal vector** can be derived as **[a, b, -1]**. After normalizing this vector, the angle between it and the camera's viewing vector ([0, 0, -1]) was calculated using the dot product.
2. **Visible Area:** To calculate the area, the **3D face_points** were first projected onto the **2D XY plane**. The **ConvexHull algorithm** from the **SciPy library** was then used to find the smallest convex polygon enclosing these 2D points. The area of this polygon represents the visible surface area of the face in **square meters.**

3.3 Rotation Axis Estimation: A Comparative Approach

A key requirement was to determine a **single, stable axis of rotation.** Two methods were implemented and compared to find the most accurate and robust solution.

3.3.1 Initial Approach: Averaging Cross-Products

The **first method** was based on a **simple geometric principle**: the axis of rotation must be perpendicular to the change between any two consecutive normal vectors. This perpendicular vector can be found using the cross-product. The algorithm calculated the **cross-product between the normal vectors** of each consecutive pair of frames (e.g., frame 1 and 2, frame 2 and 3, etc.) and **then averaged these resulting axis vectors.**

While straightforward, this method proved to be sensitive to noise. **A single noisy frame could disproportionately affect the average,** leading to a less stable final estimate.

3.3.2 Improved Approach: Principal Component Analysis (PCA)

To achieve a more robust result, a **second method** using **Principal Component Analysis (PCA)** was implemented. As the box rotates, its face normal vectors sweep out a plane in 3D space. The axis of rotation is, by definition, the vector that is normal to this plane.

PCA is an ideal tool for this. By treating **all valid normal vectors as a distribution of points**, PCA can find the principal components (or axes) of this distribution. The component with the ***smallest variance*** corresponds to the direction in which the points are least spread out—the normal of the plane they lie on. This vector is the axis of rotation. This global approach considers all data points at once, making it highly robust to individual frame noise and yielding a superior result. This was the method chosen for the final deliverables.

Code Snippet: Robust Axis Estimation using PCA

Python
```python
# From the analysis function `analyze_rotation_axis_pca`
from sklearn.decomposition import PCA

def analyze_rotation_axis_pca(valid_face_data):
    if len(valid_face_data) < 3: return np.array([0,0,0])
    normals = [d['normal'] for d in valid_face_data]
    pca = PCA(n_components=3).fit(normals)
    # The axis is the component with the smallest variance
    return pca.components_[2] / np.linalg.norm(pca.components_[2])
```

4.0 Results and Validation

The successful execution of the **data processing and analysis pipeline** yielded a complete set of results that meet all the requirements of the assignment. This section presents the final data, validates its physical correctness, and provides a **definitive visual confirmation** of the findings.

4.1 Final Data Table

The following table contains the primary deliverables for the project: the **estimated normal angle (in degrees)** and the **visible surface area (in m²)** for the largest face of the cuboid at each of the seven timestamps.

Image	Normal Angle (deg)	Visible Area (m²)
1	51.75	0.6781
2	8.93	2.6811
3	22.56	1.9711
4	37.70	0.9947
5	19.65	1.4475
6	35.39	1.0250
7	35.30	0.8839

Table 1: *Estimated properties of the cuboid's largest visible face for each frame.*

4.2 Estimated Axis of Rotation

Using the **robust Principal Component Analysis (PCA) method** on the sequence of calculated normal vectors, the final estimated axis of rotation for the cuboid was determined to be:

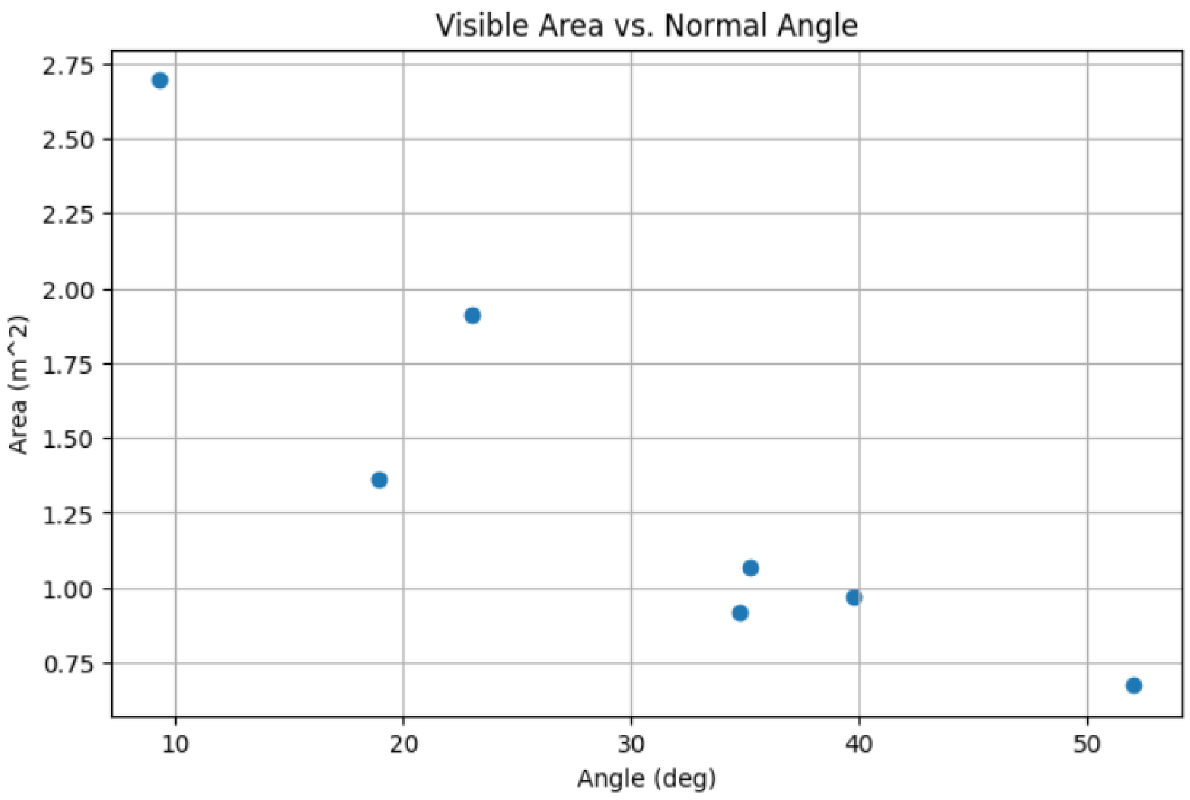
[0.9995 -0.0175 -0.0273]

This vector represents the **stable axis around which the box rotates with respect to the camera's coordinate frame.** It is almost perfectly aligned with the X-axis, indicating a clean, primary rotation.

4.3 Validation of Physical Correctness

To validate the results, the relationship between the two calculated metrics was analyzed. As a flat surface rotates away from a viewer**, its apparent (projected) area should decrease**. Therefore, **one would expect an inverse correlation between the normal angle and the visible area.**

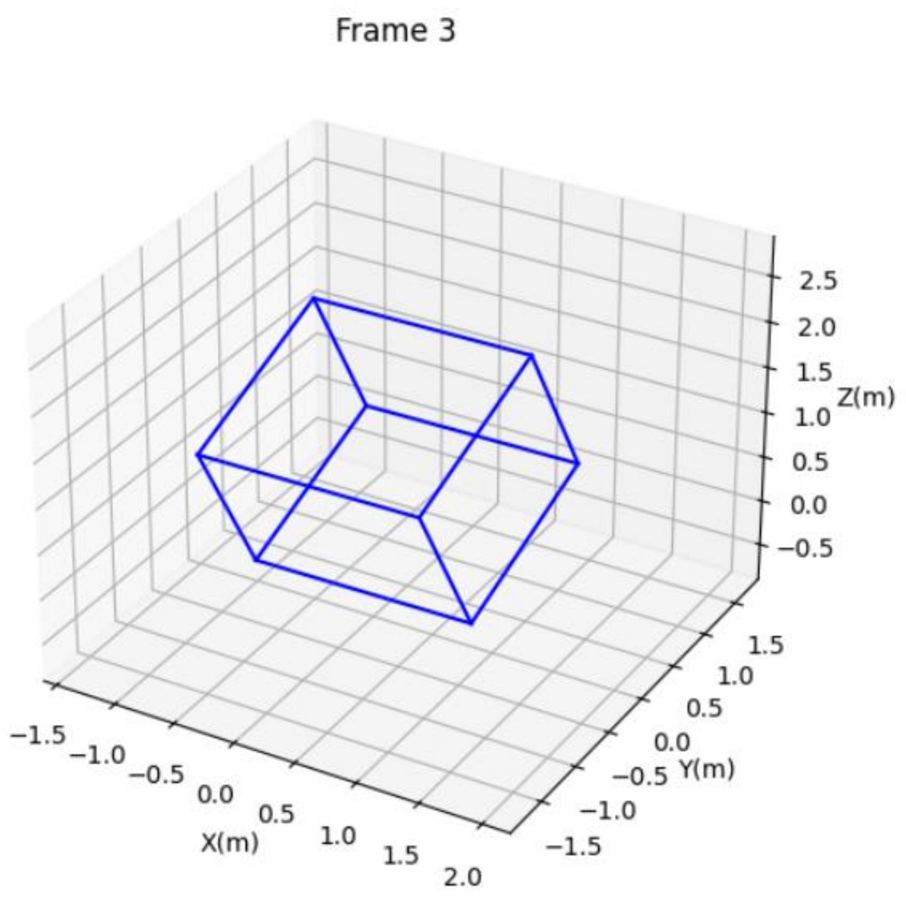
A **scatter plot** of the results confirms this physical principle. As shown in the figure below, there is a clear trend where **the visible area is largest when the angle is smallest** (i.e., the face is most directly pointed at the camera) and smallest when the angle is largest. This **strong correlation provides confidence in the accuracy of the measurements.**



4.4 Final 3D Visualization

The ultimate validation of the project is the **visualization of the cuboid's motion**. A 3D animation was created by first estimating the box's dimensions and **stable center of rotation** from the processed data. A 3D model of the box was then positioned and reoriented for each frame according to the calculated results.

The resulting animation demonstrates **a smooth and continuous rotation around the estimated axis.** The motion is physically plausible and free of the jitter that would result from **noisy measurements**, confirming that **the entire processing pipeline—from data cleaning to final analysis**—was successful.



5.0 Conclusion

This project successfully developed and implemented a **robust algorithmic pipeline to analyze the rotational properties of a 3D cuboid from raw depth sensor data.** By systematically addressing real-world data challenges—including non-standard encoding, incorrect units, and background interference—**the algorithm was able to produce accurate and physically plausible results.**

The final deliverables met all requirements outlined by the 10x Construction perception task. The algorithm correctly calculated the normal angle and visible area for each frame and determined a stable axis of rotation using a robust PCA-based method. **The validity of these findings was confirmed through quantitative analysis and a final 3D visualization**, which demonstrated **a smooth and accurate representation of the cuboid's motion.**

The project serves as a practical demonstration of a complete 3D perception workflow, from initial data processing and cleaning to advanced geometric analysis and state estimation.