**Approach and Algorithm Description**

The goal of this project was to build a complete pipeline that could extract depth data from a ROS 2 bag file, identify planar surfaces in each frame, and calculate their orientation and surface area. The system also estimates an overall rotation axis by analyzing how these planes move or tilt across multiple frames. The pipeline works entirely from a recorded ROS bag (.db3) and processes it frame by frame using a combination of ROS 2 libraries, NumPy, SciPy, and OpenCV.

**Extracting Depth Frames**

The first step was to extract depth frames from the ROS 2 bag. Using the rosbag2\_py library, the code reads messages stored under a given topic (for example, /depth). Each message was a ROS image that contained depth data — where each pixel’s value represented its distance from the camera.

The cv\_bridge library was used to convert these ROS image messages into NumPy arrays so that they could be processed in Python. Depth values were normalized from millimeters to meters, and each frame was saved as a .npy file inside a folder named frames.

At the end of this step, we had a clean sequence of numerical depth images that could be processed without any ROS dependencies.

**Converting Depth to 3D Points**

Each depth image is essentially a 2D grid of distance values, but to analyze surfaces we had to represent it in 3D space.

Using the pinhole camera model, every valid pixel (u, v) is backprojected into 3D coordinates:

where and were the camera focal lengths and were the image center coordinates.

This converted the depth map into a cloud of 3D points in the camera’s coordinate frame. To speed up computation, the algorithm only uses every few pixels (based on a step size), and ignores any invalid or distant readings.

**Detecting the Dominant Plane with RANSAC**

Once I had a 3D point cloud, I fit a plane to it using the RANSAC (Random Sample Consensus) algorithm. The idea behind RANSAC was to find a model (in this case, a plane) that best fit the majority of the data while ignoring outliers or noisy points.

In each iteration, the algorithm randomly selects 3 points and computes the plane that passes through them. It then calculates the distance from every other point in the cloud to this plane. Points that are close enough (within a small threshold) are considered inliers. The plane with the highest number of inliers after all iterations is selected as the best fit. After finding the best plane, the algorithm refines its orientation using Singular Value Decomposition (SVD) on the covariance of inlier points. This gives a more stable and accurate normal vector for the detected plane.

**Computing Area and Orientation**

Once the main plane in a frame was found, I computed two key properties — the **area** and its **angle** relative to the camera. To find the area, all inlier points were projected onto a 2D coordinate system defined by two tangent vectors on the plane. Essentially, this flattened the 3D points into a top-down view of the plane.

I then computed the Convex Hull of these 2D points — the smallest polygon that encloses them — and calculated its area. This represented the visible portion of the plane in square meters. I also filter out distant outlier points before computing the convex hull, which helps avoid overestimating the area due to noisy edges.

The orientation angle is computed by measuring how much the plane’s normal vector tilts relative to the camera’s Z-axis (the direction the camera faces). This gives a clear idea of whether the plane is flat, slanted, or vertical.

**Analyzing Motion Across Frames**

After processing all frames, the system had a list of plane normals — one for each depth frame.  
By analyzing these normals together, I estimated a rotation axis that described how the detected plane or object was moving relative to the camera.

This was done by stacking all the normal vectors and applying SVD again to find the direction that changed the least — effectively the axis of rotation.

**Outputs**

At the end of the process, the following files are generated:

* **frames/** – Contains all extracted depth frames as .npy arrays.
* **results.csv** – A table listing each frame’s orientation angle and visible area.
* **axis\_of\_rotation.txt** – A text file containing the estimated unit vector for the rotation axis.

Each row in the CSV file corresponds to one depth frame and provides metrics like.