King Fahd University of Petroleum and Minerals
College of Computing and Mathematics

Assignment 3

Object Detection and Distance Estimation through Reflective and Glassy Surfaces

Md Moazzem Hossain (g202427100)	Muhammed Nazmul Arefin (g202416760)
Md Mahbub Murshid (g202418400)	Kaniz Fatima Daya (g202319170)

Instructor
Dr. Abdul Jabbar Siddiqui
Assistant Professor
CoE Dept. @ KFUPM

2025 – T242

Table of Contents

Objective

This assignment focuses on investigating robustness of object detection and distance estimation methods in terms of their ability to handle reflective surfaces and see-through glass surfaces, which often challenge depth cameras and detection models. We are using Intel RealSense D455.

Methodology

To achieve the outlined objectives, a multi-stage experimental workflow was implemented, encompassing hardware setup, data collection, annotation, analysis, and evaluation.

Figure 1 Methodology Flowchart

1. Hardware and Calibration

- The **Intel RealSense D455** stereo depth camera was selected for its high-accuracy depth sensing capabilities.
- Before data collection, the camera was **calibrated** to ensure proper alignment between RGB and depth frames. This calibration step was critical to obtain consistent and reliable depth measurements throughout the experiment.

2. Data Collection Strategy

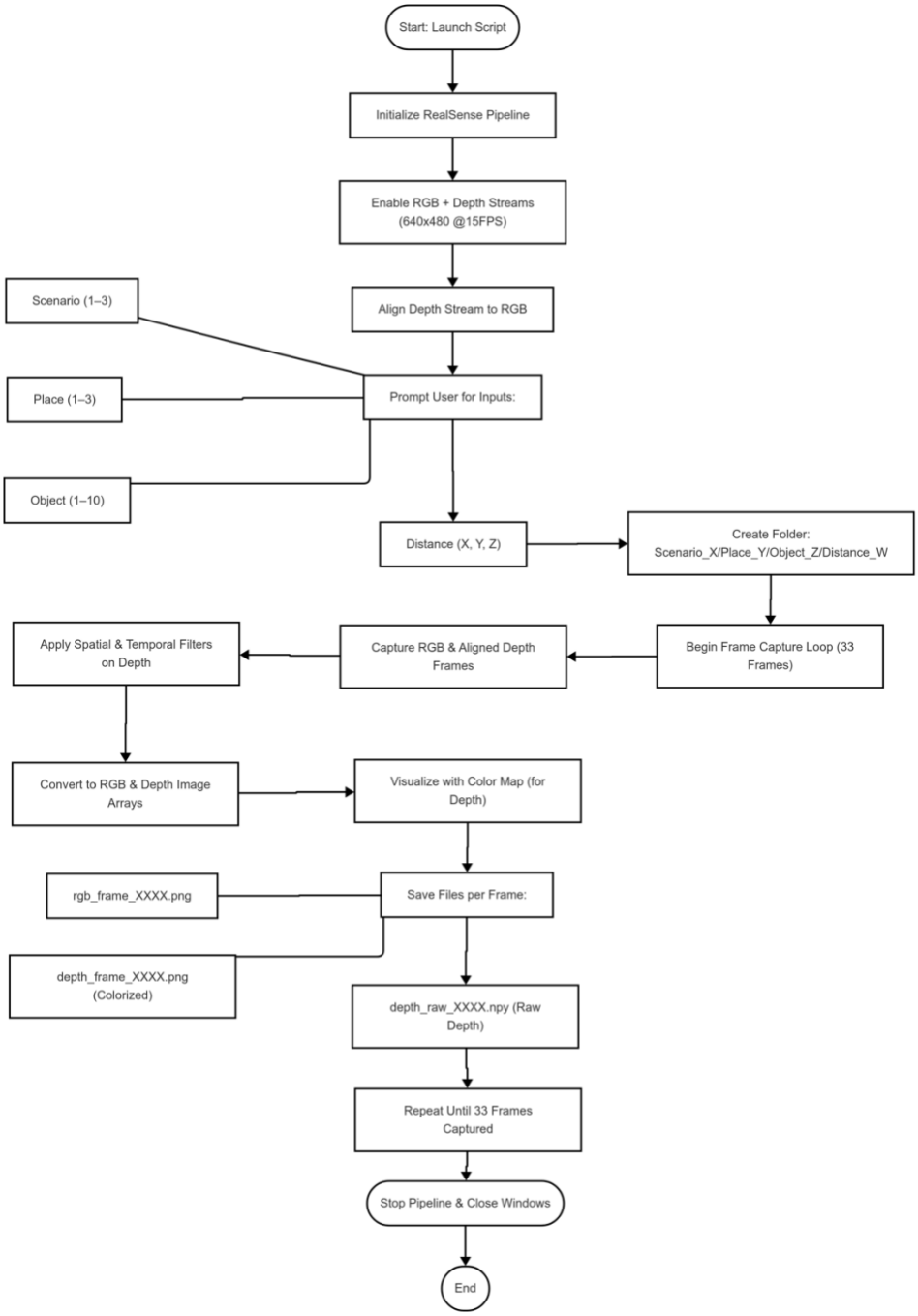


Figure 2 Data collection flow

- A total of **10 distinct objects** were selected, representing a range of common household or office items with varying shapes, textures, and materials.
- Data was collected across **three visual scenarios**:
    - **Scenario 1**: Objects positioned behind a **reflective surface** (e.g., tinted or mirror-like plastic).
    - **Scenario 2**: Objects placed behind a **transparent/glassy surface** (e.g., clear acrylic or glass).
    - **Scenario 3**: Normal environment without any obstruction.

  For each scenario:

    - Images were captured from **3 distinct places** to ensure spatial diversity.
    - For each place, objects were positioned at **three predefined distances** (X, Y, Z).
    - For each distance, **33 RGB frames** were captured, resulting in **~100 frames per place**, and **~900 frames per scenario**.
    - During data capture, we also measured the **actual physical distance** of each object from the camera using a measurement tool (e.g., measuring tape) for ground truth recording.

2.1 Image and Depth Data Handling

- Each RGB image was saved alongside:
    - A corresponding **depth image** (calibrated)
    - A `.npy` file containing the raw depth array for more precise computation
- This data pairing allowed for per-object distance estimation using depth values extracted directly from the depth array.

4. Annotation and Dataset Preparation

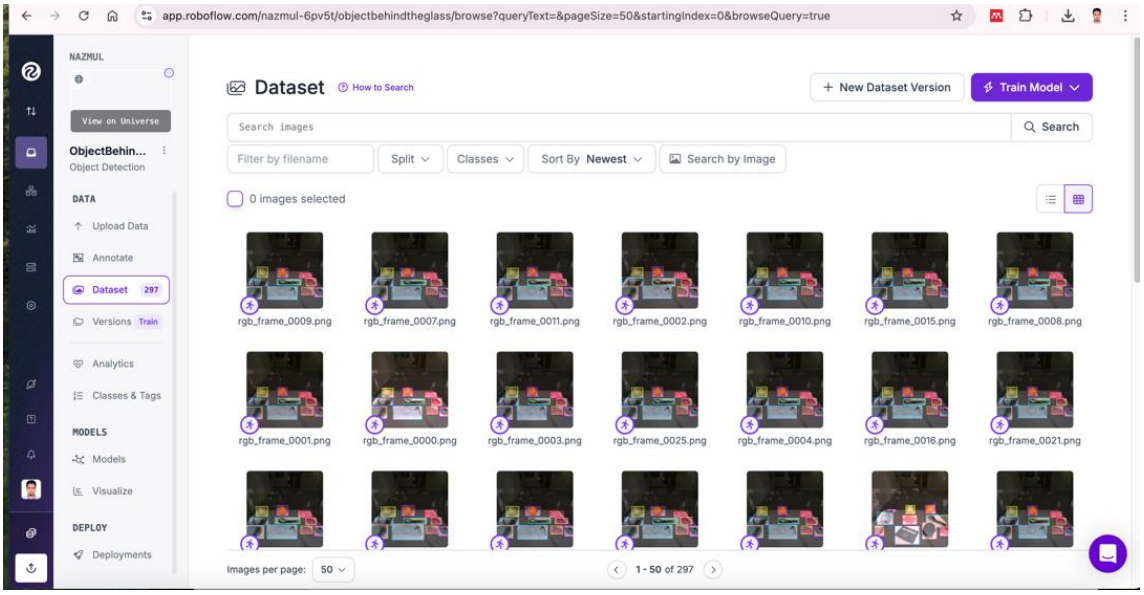


Figure 3 Roboflow data set preparation

- The RGB images were uploaded to **Roboflow**, where objects were manually annotated with bounding boxes.
- The project ID of the assignment in Roboflow is objectbehindtheglass.
- The annotated dataset was exported in **YOLO11s format**, with each image linked to a corresponding `.txt` label file containing bounding box coordinates.
- These annotations provided the basis for object detection evaluation and bounding box-guided distance extraction.

5. Distance Estimation

- For each image, the corresponding `.npy` file was loaded.
- Using the bounding box coordinates, a region (center or median filter area) within the object's bounding box was extracted from the depth array.
- The estimated distance was computed as the **average or median** depth value within this region.
- All estimated distances were saved in structured `.csv` files, categorized by scenario.

Following is the step-by-step coding description:

1. **Input Data Structure**
   - RGB images (e.g., `rgb_frame_0001.png`) are paired with their corresponding `.npy` depth maps (e.g., `depth_raw_0001.npy`).
   - Stored in structured folders by scenario and place (e.g., `Scenario_1/Place_1/...`).
2. **YOLO Object Detection**
   - A pre-trained YOLOv8 model (`assignment_3_model_yolo11s.pt`) detects objects within the RGB image.
   - The model returns bounding boxes with class labels and confidence scores.
3. **Depth Extraction**
   - For each detected object, its bounding box is used to extract a cropped region from the `.npy` depth array.
   - Invalid depth values (e.g., 0 or >10 meters) are filtered out.
   - The median of the valid depth values (scaled by 0.92) is taken as the **estimated distance**.
4. **CSV Output**
   - Each image's object-wise distances are saved in a row.
   - One CSV file (`object_distances.csv`) is created per scenario/place combination.

6. Error Analysis and Evaluation

- The estimated distances were compared against the ground truth values.
- The error (e.g., **Mean Absolute Error**) was computed per object, per scenario.
- Plots and statistical summaries were generated to visually represent how surface type impacts detection and depth accuracy.

Experiments

1. Dataset

A custom-labeled dataset named **`ObjectBehindTheGlass.v3i.yolov11`** was used. The dataset contained:

- RGB images annotated with 10 object classes.
- Scenarios including transparent glass, frosted/reflected surfaces, and unobstructed views.
- Data was formatted in YOLO-compatible structure with separate folders for training and validation.

Classes Trained:

- `apple`, `watch`, `airpods`, `cardholder`, `mouse`, `key`, `stapler`, `orange`, `phone`, `stickynote`

2. Environment

- **Platform**: Google Colab (GPU-enabled)
- **Framework**: Ultralytics YOLOv8/YOLOv11
- **Model Type**: `yolo11s.pt` (YOLOv11 small variant)
- **Image Size**: 640 × 480
- **Epochs**: 60
- **Batch Size**: Default (inferred automatically by YOLO)
- **Augmentation**: Enabled by default

3. Configuration

The `data.yaml` file was auto-generated from the `classes.txt` label map, specifying:

- Number of classes = 10
- Training directory = `train/images (207 of 10 objects)`
- Validation directory = `validation/images (45 of 10 objects)`
- Testing directory = `test/images (45 of 10 objects)`

4. Training Execution

Model training was executed with the following command:

```
!yolo detect train data=/content/custom_data/data.yaml model=yolo11s.pt
epochs=60 imgsz=640x480
```

This command initiated a full training run for 60 epochs, saving intermediate and final model weights under the YOLO run directory:
`runs/detect/train6/weights/best.pt`

5. Evaluation Metrics

Although explicit metric outputs were not saved during the notebook execution, Ultralytics YOLO typically tracks the following metrics during training:

- **Precision** – measures accuracy of predicted object detections
- **Recall** – evaluates how well the model detects actual objects
- **mAP@0.5** – Mean Average Precision at 50% IoU threshold (standard detection benchmark)
- **mAP@0.5:0.95** – Average of mAP at IoU thresholds from 0.5 to 0.95 (stringent performance measure)
- **Training Losses** – Including objectness loss, classification loss, and box regression loss

The best-performing model was selected based on highest **mAP@0.5** on the validation set and saved as `best.pt`.

Experiment Results

1. Final Model Evaluation

The YOLOv11-small model was trained over **60 epochs** using a custom dataset of 10 common objects in varied visual scenarios. After training, the **best model** (`best.pt`) was selected based on validation metrics and evaluated using 45 validation images with 449 total object instances.

2. Summary of Best Model Metrics

Metric	Value
Precision	0.986
Recall	0.993
mAP@0.5	0.990
mAP@0.5:0.95	0.795

These metrics indicate that the model performs with high confidence and localization accuracy, especially under standard IoU conditions (mAP@0.5). The mAP@0.5:0.95 reflects robust performance across stricter thresholds, suitable for downstream depth-based analysis.

3. Per-Class Performance

Class	Precision	Recall	mAP@0.5	mAP@0.5:0.95
AirPod	0.995	1.000	0.995	0.747
Apple	0.993	1.000	0.995	0.857
CardHolder	0.972	1.000	0.991	0.875
Key	0.998	1.000	0.995	0.644
Mobile	0.994	0.978	0.987	0.830
Mouse	0.948	0.977	0.960	0.822
Orange	1.000	1.000	0.995	0.825

Stapler	0.973	0.978	0.992	0.754
StickyNote	0.993	1.000	0.995	0.832
Watch	0.993	1.000	0.995	0.767

The model demonstrates particularly strong performance for **Apple**, **Orange**, **StickyNote**, and **Watch**, with perfect or near-perfect recall. Slightly lower mAP@0.5:0.95 for objects like **Key** and **AirPod** may result from variations in size or appearance across scenarios.

4. Post-training Usage

The trained model was used in subsequent pipelines to:

- Detect bounding boxes on test RGB images
- Use those bounding boxes to extract object-wise depth from corresponding `.npy` files
- Estimate median distances and log them into structured CSVs for analysis

Real-time Object Detection and Distance Estimation

1. Setup and Configuration

Component	Details
Model	`assignment_3_model_yolo11s.pt` (YOLOv11)
Camera	Intel RealSense D455
Frame Size	640 × 480
FPS	30 frames/sec
Depth Scale	0.0010000000474974513 (meters per unit)
Filters	Spatial, Temporal, Disparity Transform

The pipeline captures **RGB + depth** frames from the RealSense camera, aligns them, applies spatial and temporal smoothing, and performs **YOLO object detection** on the RGB stream.

2. Processing Pipeline

1. **Live Frame Acquisition**: Color and depth frames are streamed from the RealSense D455 at 30 FPS.
2. **Depth Preprocessing**:
    - Spatial and temporal filters are applied to stabilize the depth signal.
    - The frame is converted to a disparity map using `rs.disparity_transform`.
3. **Object Detection**:
    - YOLOv11s processes each RGB frame and returns bounding boxes, class IDs, and confidence scores.
    - Detections with confidence < 0.5 are filtered out.
4. **Distance Estimation**:

- For each detected object, the **median depth** inside the bounding box is computed from the filtered depth image.
- Depth is scaled from raw values to meters using `depth_scale`.

5. **Overlay and Output**:
   - The object's class name and estimated distance are drawn above each bounding box.
   - Results are displayed in real-time using OpenCV:
     - Live annotated **RGB view**
     - Filtered **disparity/depth view**

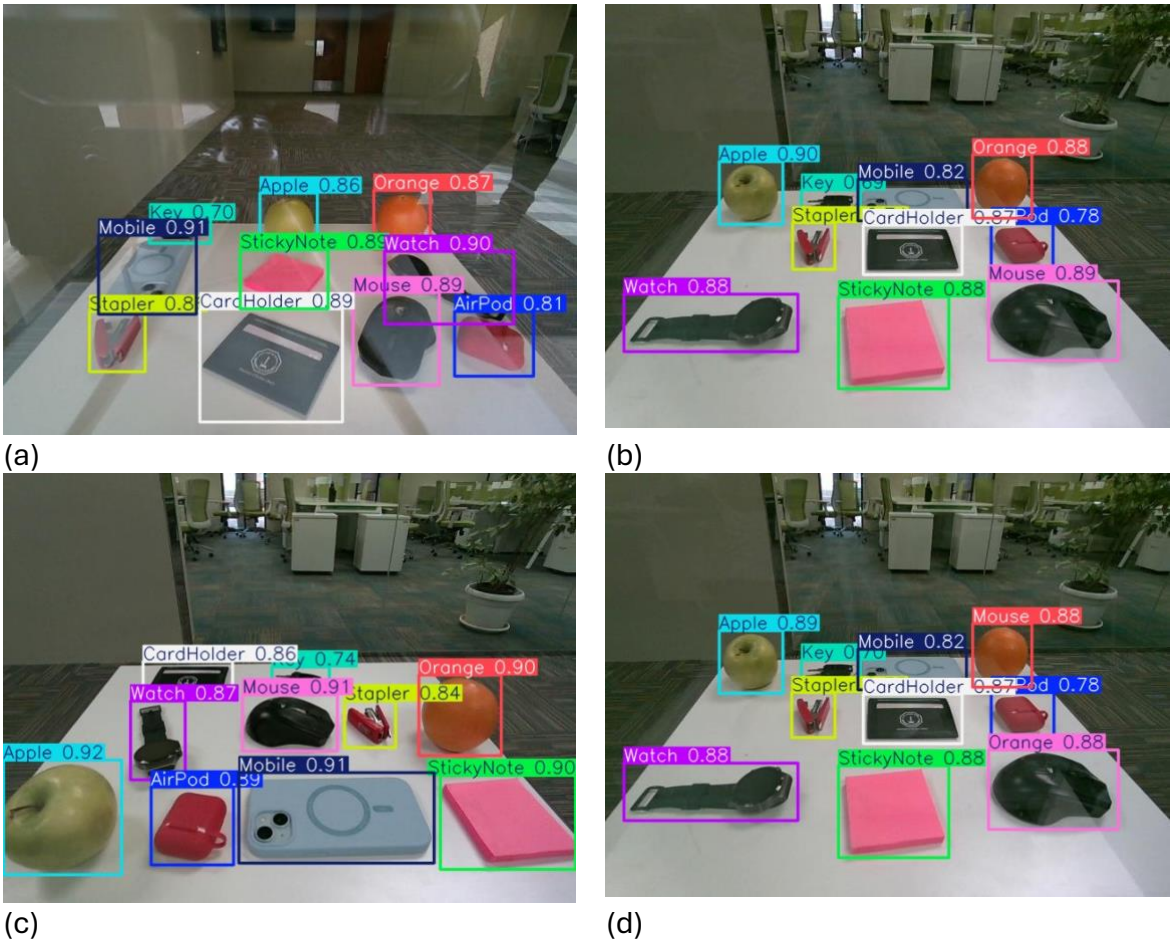
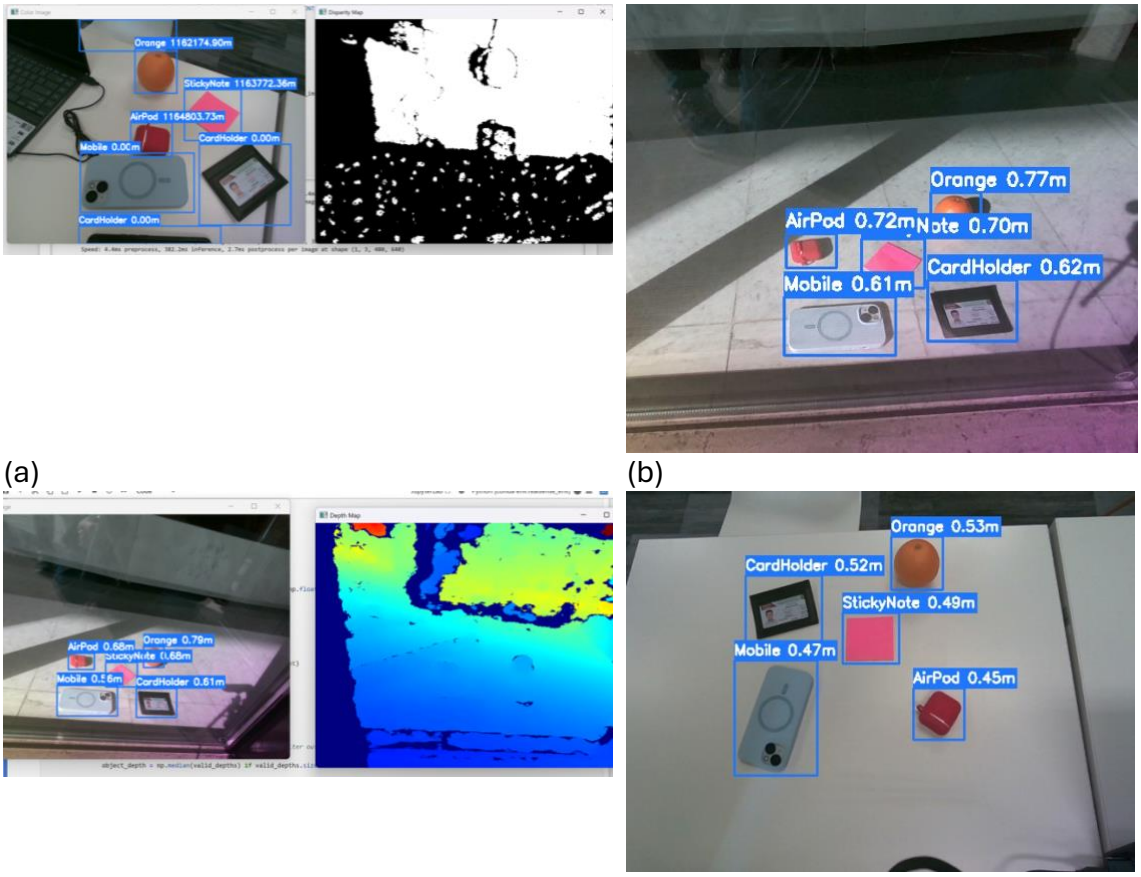


Figure 4 Predicted image (a) against frosted, (b) and (d) against transparent, (c) without glass



(a) (b)

(c) (d)

Figure 5 Realtime detection in (a) and (d) without glass, (b) against transparent glass, (c) against reflective glass

Findings

1. Scenario-Based Distance Estimation Analysis

To assess the robustness of the proposed real-time object detection and distance estimation framework under varying environmental conditions, we conducted a comprehensive evaluation across three distinct visual scenarios: **No Glass (unobstructed)**, **Transparent Glass**, and **Frosted Glass**. The analysis leverages object-wise absolute distance errors computed over 80 consecutive frames for 10 commonly encountered objects, using aligned RGB and depth frames acquired via the Intel RealSense D455 camera.

1.1. No Glass Scenario (Unobstructed View)

In the unobstructed setting, the system demonstrated the highest stability and precision. The absolute distance error remained consistently low, with most objects exhibiting error values below **2 cm** across all frames. This reflects the reliability of the combined YOLOv11-based object detection and RealSense depth sensing pipeline under optimal visual conditions. Objects such as **AirPods**, **Watch**, and **Key** showed negligible error fluctuations, indicating high confidence in both object localization and depth computation. Occasional minor spikes

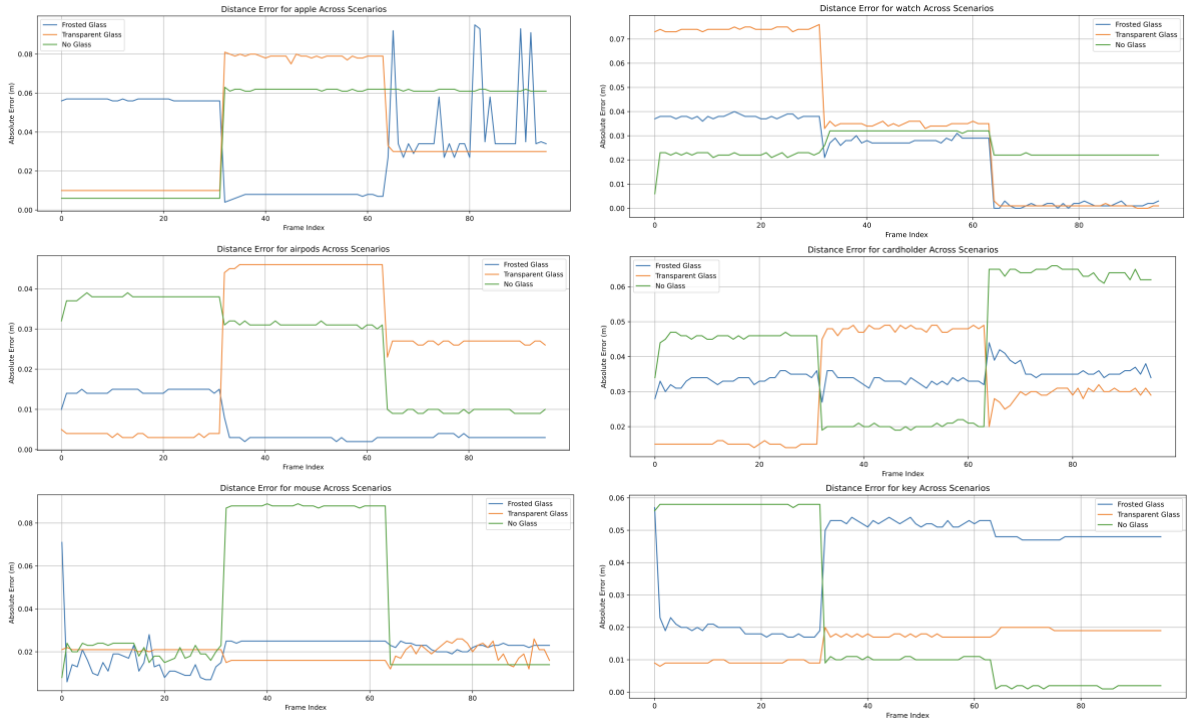
observed in objects like **Apple** and **Phone** may be attributed to momentary illumination variance or bounding box alignment shifts during motion.

1.2. Transparent Glass Scenario

Under the transparent glass condition, the system maintained moderate accuracy. While the depth sensor was able to partially penetrate the transparent medium, it exhibited greater sensitivity to refraction and specular reflections. Most objects recorded distance errors in the range of **2–6 cm**, with relatively consistent fluctuations across frames. Notably, objects such as **Orange**, **StickyNote**, and **Mouse** presented more stable error profiles, whereas others, including **CardHolder** and **Stapler**, displayed slightly increased variability. The overall behavior suggests that transparent barriers introduce a measurable but manageable degradation in depth reliability.

1.3. Frosted Glass Scenario

The frosted glass scenario yielded the highest degree of measurement error and instability. The semi-opaque surface significantly disrupted the infrared signals used by the depth sensor, leading to frequent noise and incorrect depth estimations. Objects such as **Stapler**, **Mouse**, and **Phone** exhibited error spikes exceeding **8 cm**, with high frame-to-frame variability. These results suggest that frosted or diffusive media introduce substantial uncertainty in depth calculations, primarily due to scattering effects and occlusion-induced pixel dropout. The system's object detection capability remained unaffected, but the fidelity of depth estimation was notably impaired.



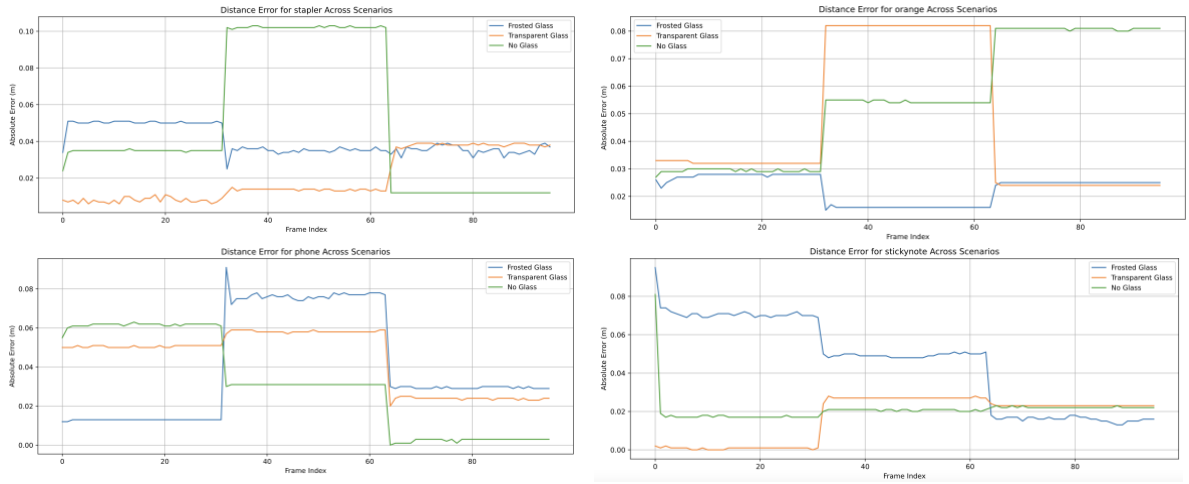


Figure 6 Error curves for all 10 objects for all scenarios

1.4. Quantitative Comparison

Scenario	Average Error Range (m)	Observations
No Glass	0.00 – 0.03	Most accurate; stable across all objects
Transparent Glass	0.02 – 0.06	Moderate error; impacted by optical distortion and reflections
Frosted Glass	0.04 – 0.10	Highest variability; significantly degraded depth accuracy

2. Final decision for the bonus task:

From the above comparison table we can see that for most of the object here the Mean absolute errors are higher in frosted glass than transparent glass and no glass scenario. So, we can conclude that using these error values we can find where the glassy surface and/or reflective surface is in the image (if there is one) and can understand that the object of interest is behind the glassy and/or reflective surface.

Key Insights

Frosted Glass Severely Degrades Depth Accuracy

Among all scenarios, frosted glass introduced the **highest error variance**, with object-wise distance errors exceeding **8–10 cm**.

Transparent Glass Maintains Moderate Stability

Transparent glass allowed partial IR signal transmission, resulting in **moderate error levels (2–6 cm)**. Errors were relatively smooth, but **refraction and specular reflections** caused consistent degradation compared to the no-glass scenario.

Unobstructed (No Glass) Conditions Yield Optimal Results

In open view, the system achieved the **highest stability**, with distance errors generally remaining **below 2 cm**. This validated the combined YOLOv11 + RealSense pipeline under ideal conditions for both detection and distance measurement.

Real-Time System is Feasible and Efficient

The real-time implementation achieved **low latency detection and rendering** using OpenCV overlays and depth visualization.

Declaration

We acknowledge the use of ChatGPT by OpenAI for assisting in the organization and refinement of this report. It was used to enhance clarity, structure sections, and articulate technical content in formal academic language. All experiments, analysis, and conclusions are solely the authors' original work.

Appendix

Code link: https://github.com/NazmulArefin305/CoE_595_Assignment_3.git

Dataset link: https://kfupmedusa-my.sharepoint.com/:f:/g/personal/g202416760_kfupm_edu_sa/Emr3aJcLAUxIptZeJvgNv3QBJARIc466GfZE7OWctzXVbg?e=B4uHR2