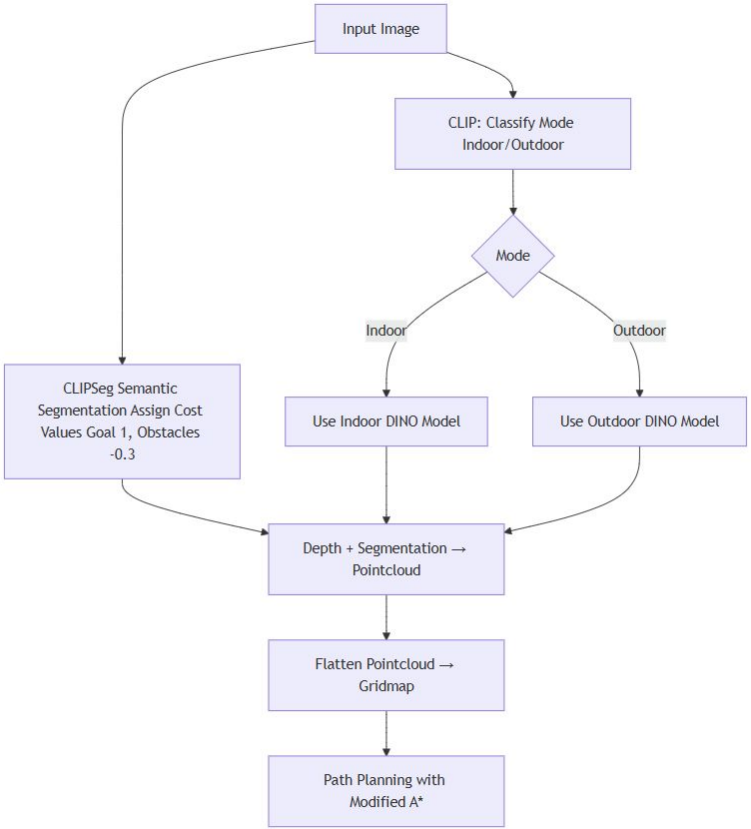
Scene Specific Navigation using CLIPSeg

Aditya Potnis (apotnis2), Aidan Wefel(awefel2), Sai Aitha (sa60)

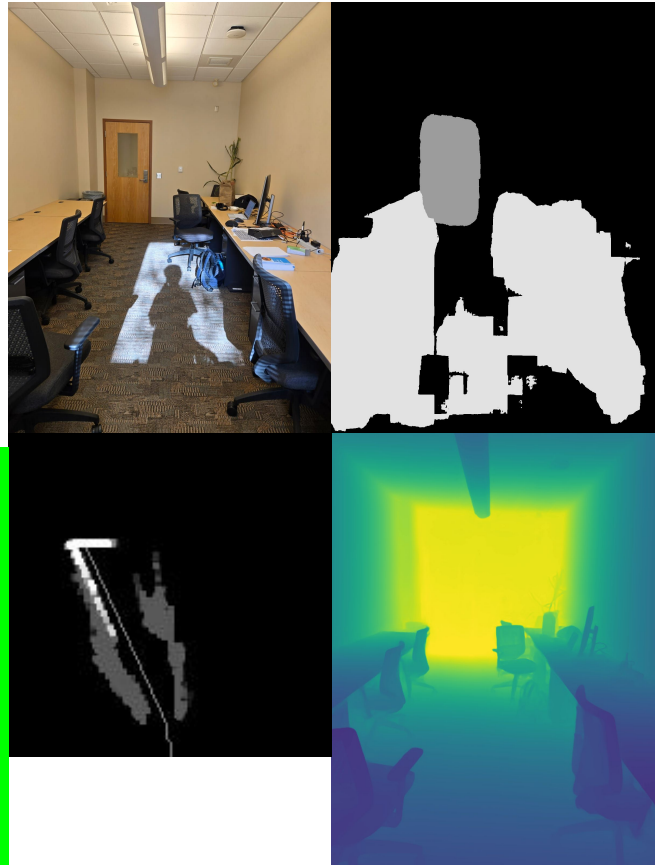
What are we doing?

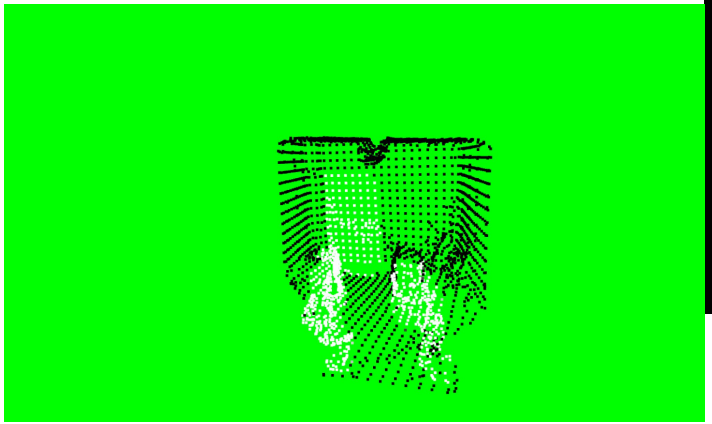
We propose a combined system for path finding and navigation to a goal object using CLIP for scene understanding to select relevant scene specific monocular depth estimation model (Depth Anything V2), utilize open vocabulary segmentation using CLIPSeg to identify risky obstacles and an implementation of a path finding algorithm using A* to allow actual pathfinding to the goal object. We test robustness of each submodule to validate system performance and propose some use cases.



Key Technological Innovation

We combine Image segmentation, depth estimation, and point cloud construction techniques to make a system that is flexible across different use cases and scenarios.





Scene Understanding CLIP v. DINO ViTs

CLIP

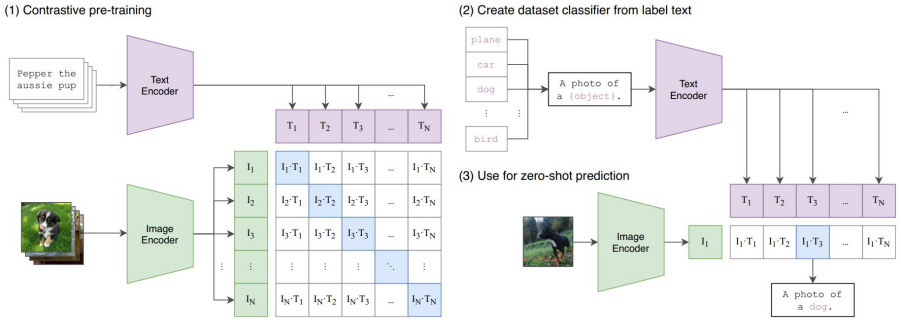
CLIP Model (Contrastive Language-Image Pre-Training) from OpenAI that learns to connect images and text. It is trained on hundreds of millions of image–text pairs from the internet, enabling it to understand and relate visual and textual information in a shared space

- Image Encoder: (e.g., Vision Transformer) converts images into vectors.
- Text Encoder: (e.g., Transformer) converts text into vectors

Trains both encoders so that matching image-text pairs have similar vectors, while mismatched pairs are far apart

Pro: Has good semantic similarity understanding for scenes

Cons: Spatial understanding is not as good



Scene Understanding CLIP v. DINO ViTs

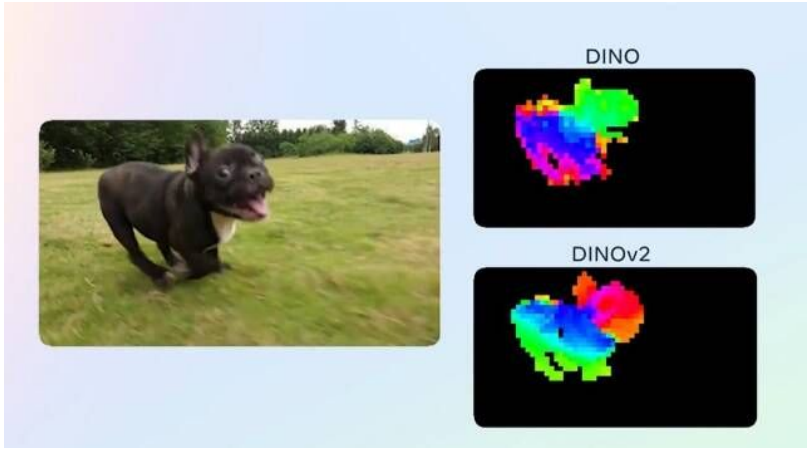
DINO & DINOv2 (Self-Supervised Vision for Universal Representations)

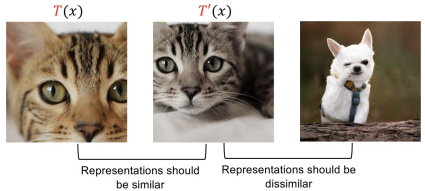
DINOv2 is a self-supervised vision model by Meta AI that learns robust visual features without labeled data. Trained on 142M+ images, it serves as a backbone for tasks like depth estimation, segmentation, and retrieval-without task-specific fine-tuning.

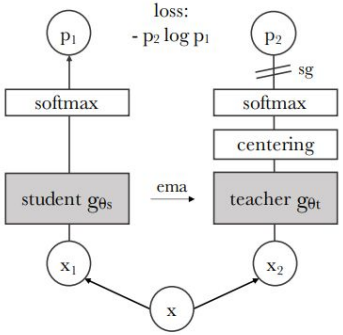
ViT is pre-trained on a global [CLS] patch and local patches in a student teacher network using only positive siamese pairs. The teacher gradient is stopped, and instead updated using exponential moving averages to prevent model collapse.

Pros: No labelling, good spatial accuracy

Cons: Poorer semantic scene understanding

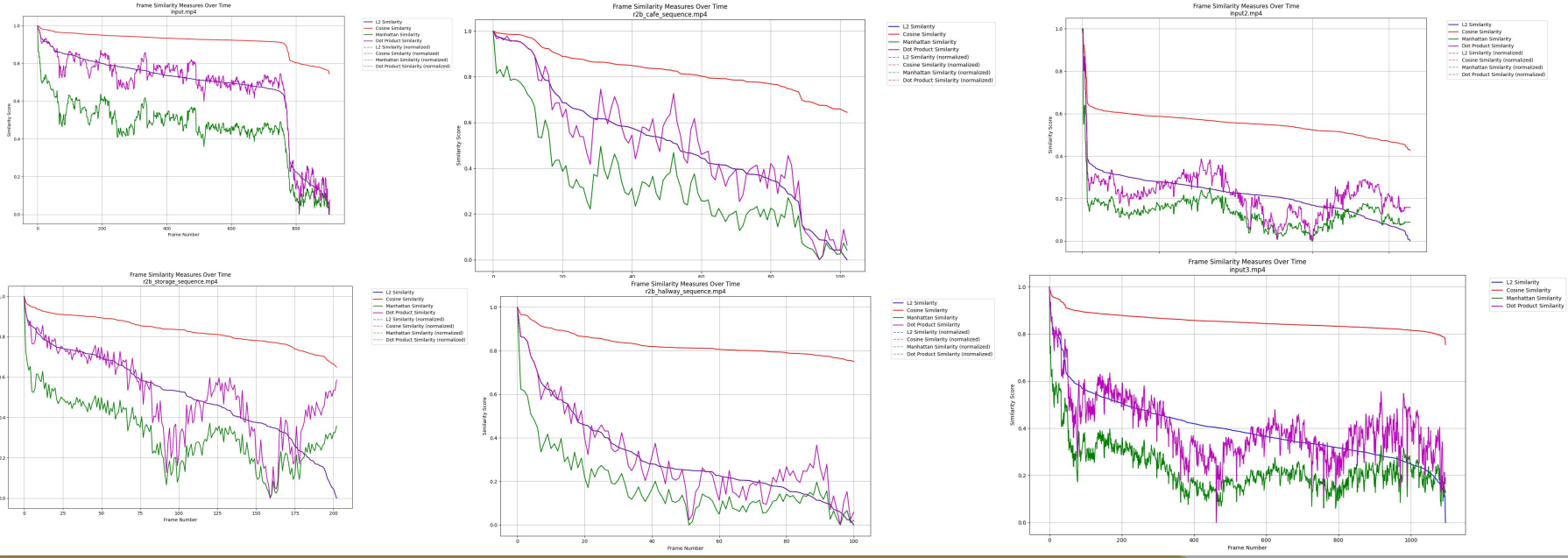




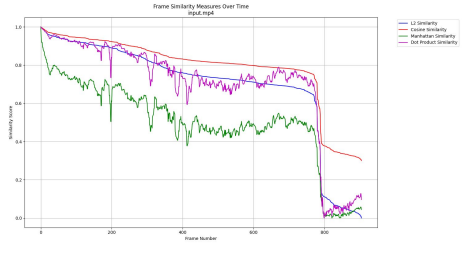


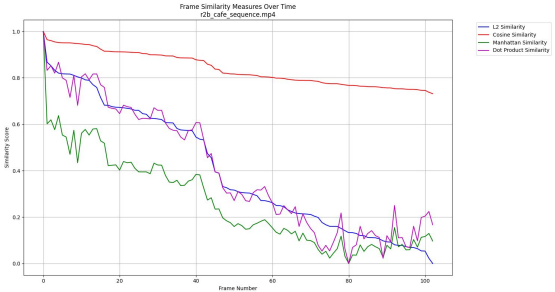
Comparing using variety of scene datasets and our images, we use cosine similarity as the primary metric for comparison

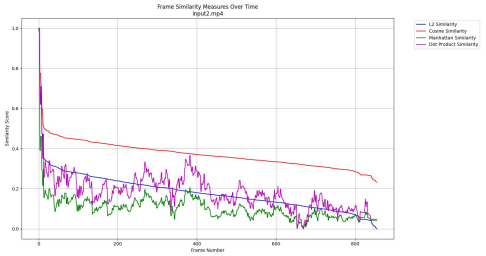
CLIP similarity between 1st and rest embeddings (smoother, less noise to change of position, better semantic scene understanding)

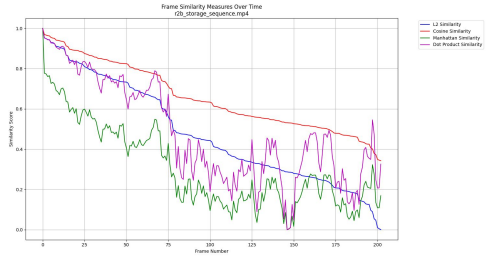


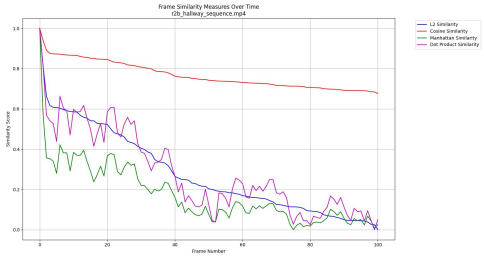
DINOv2 similarity between 1nd and rest embeddings (noisier, to change of position, more spatial)

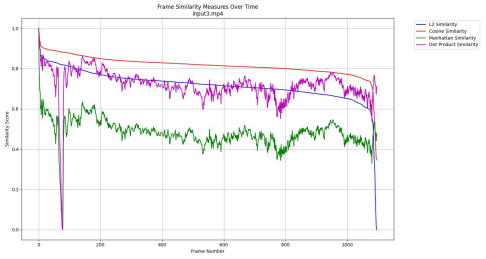










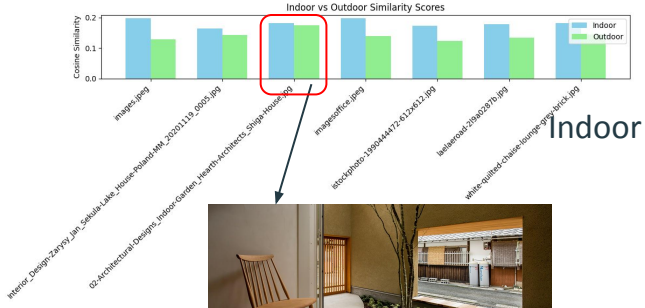


CLIP text and image scene similarity detection

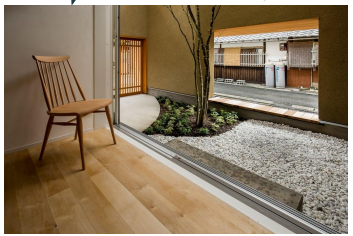
CLIP is fairly consistent at comparing scene description. We tested using a variety of 30+ images from a variety of datasets and indoor/ outdoor images online along with matterport and kitti datasets. Overall success percentage was 94% (NOTE: misc data was not used)

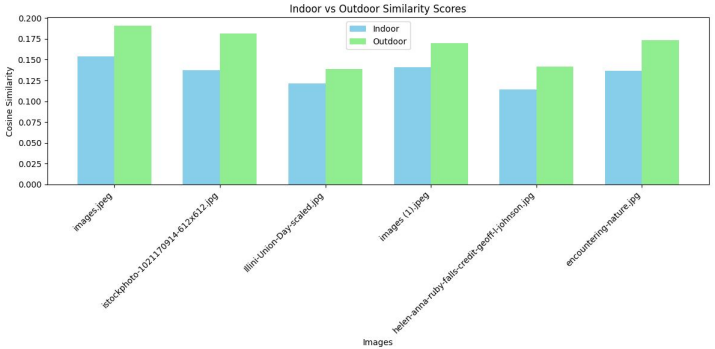
The absolute value of cosine similarity are not constant, hence we use argmax to find highest cosine similarity among a few scene descriptors such as “indoor” and “outdoor”. This allows us to create necessary scene descriptors for special purposes as needed. To extend the design.

We saw a few edge cases where indoor and outdoor scenes were hard to differentiate like this house which had a large tree inside of it



Indoor images





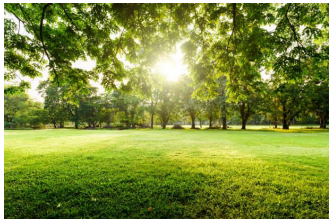
Outdoor images

Sample images used

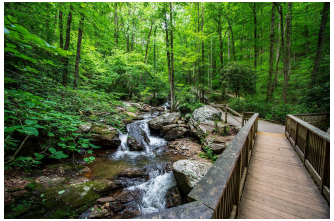




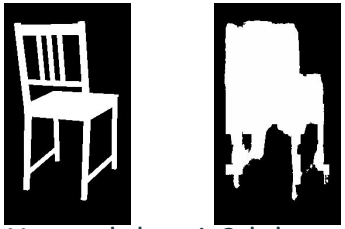


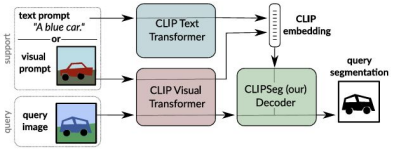





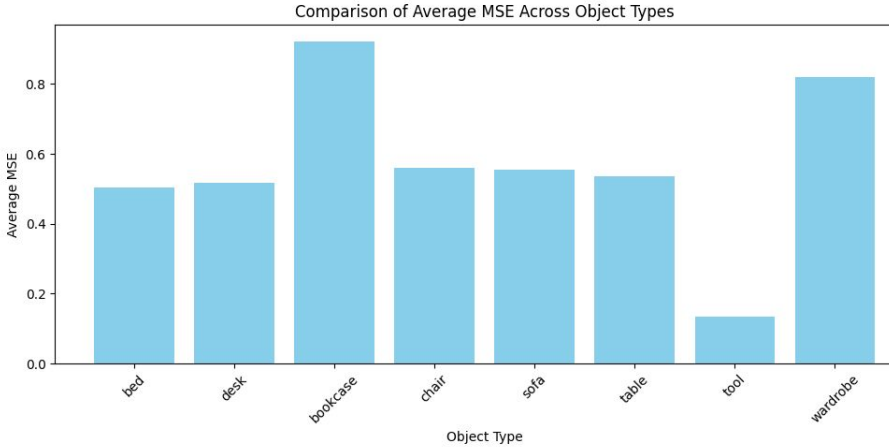



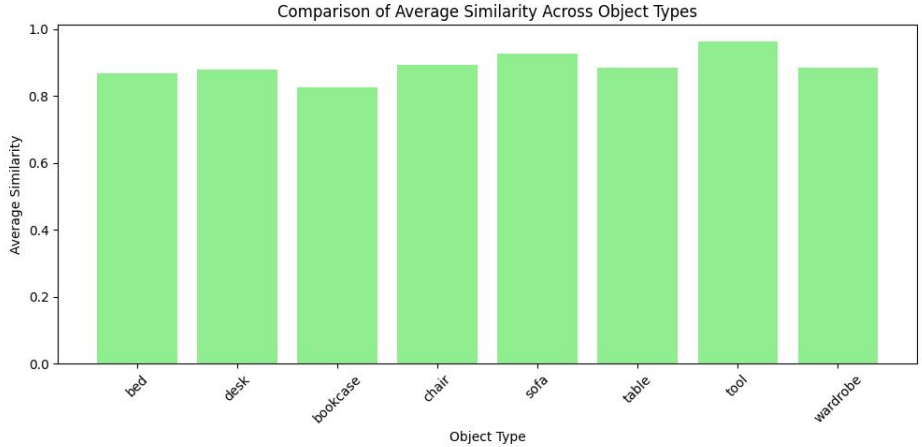
Open Vocabulary segmentation - CLIPSEG





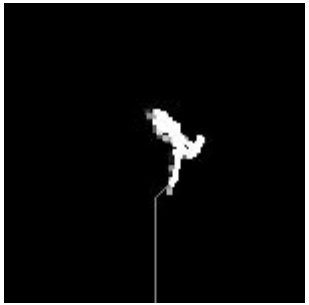
We used the pix3d dataset to evaluate CLIPSEG's performance, comparing the generated masks with the ground truth masks

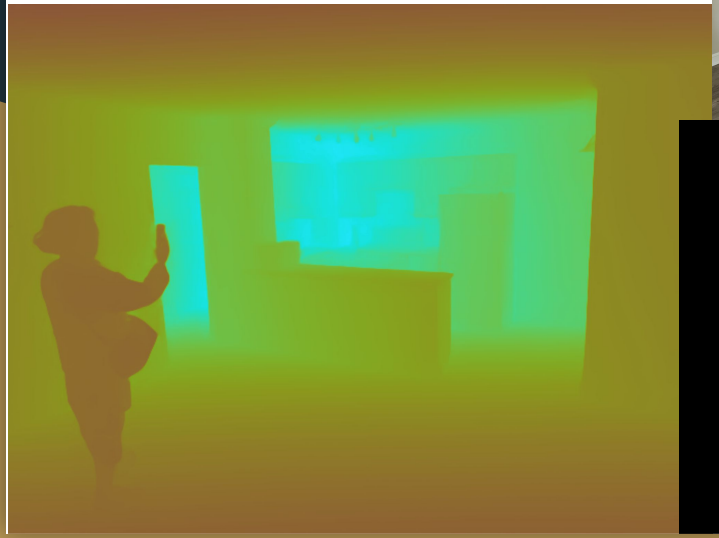


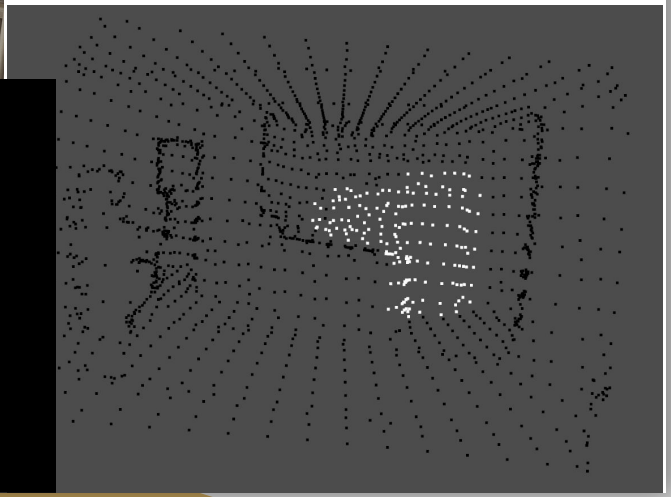


Evaluations path_image_0

Target: Fridge

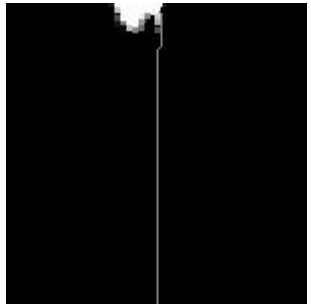


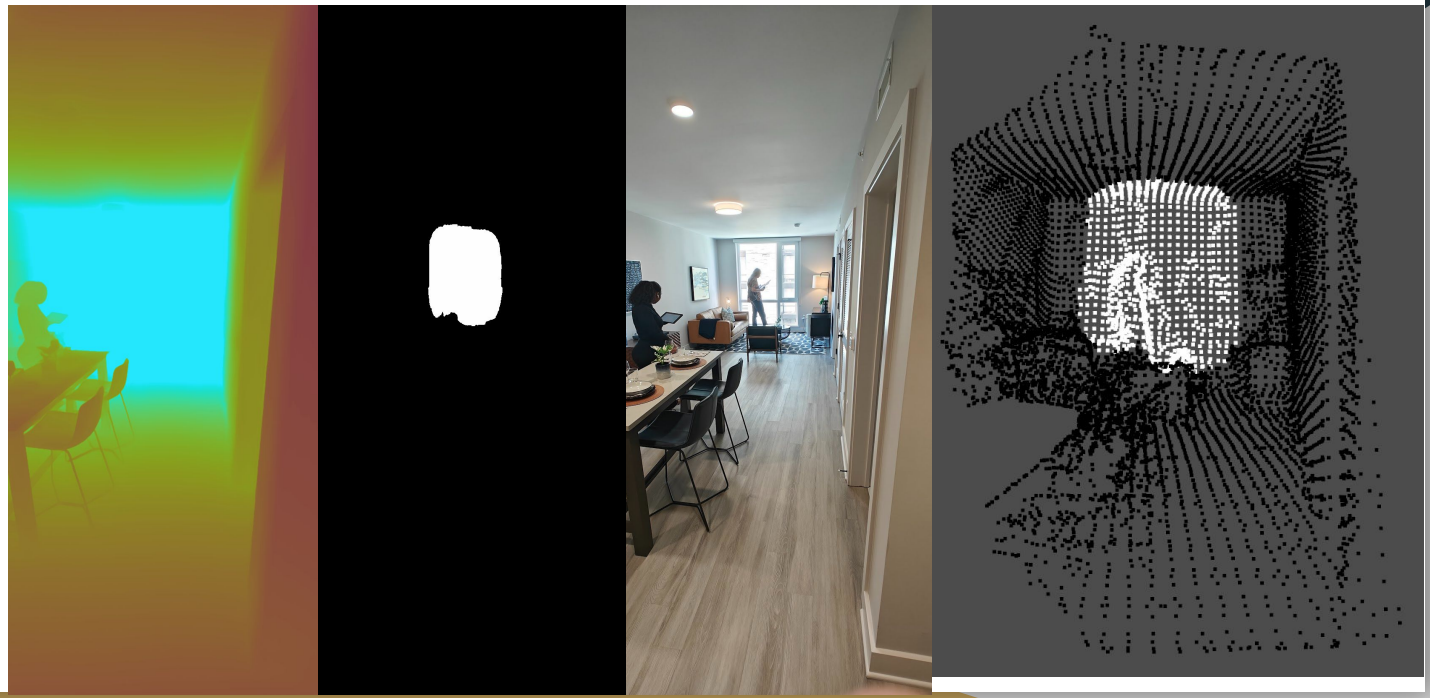




Evaluations path_image_1

Target: Window

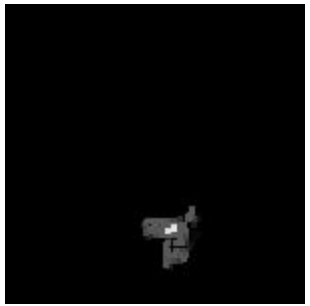


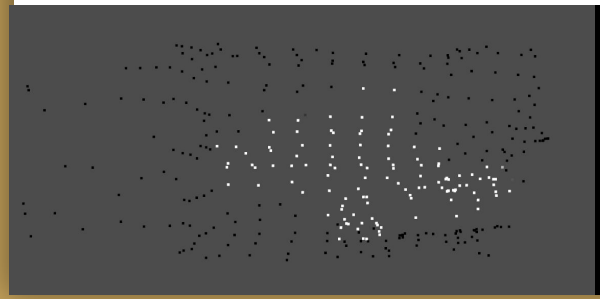


Evaluations path_image_2

Target: Stove

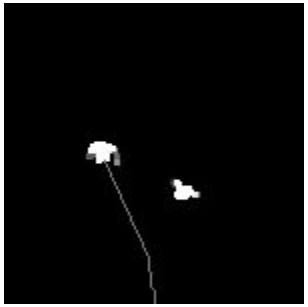
Avoid: Counter

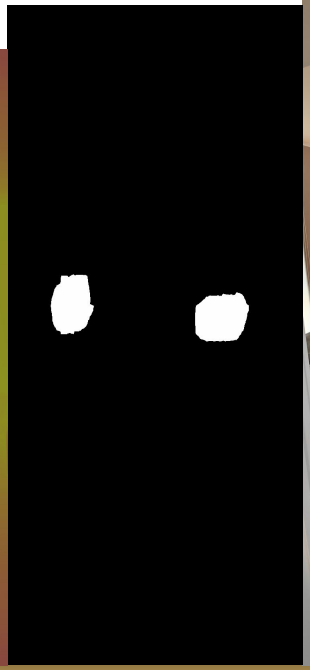


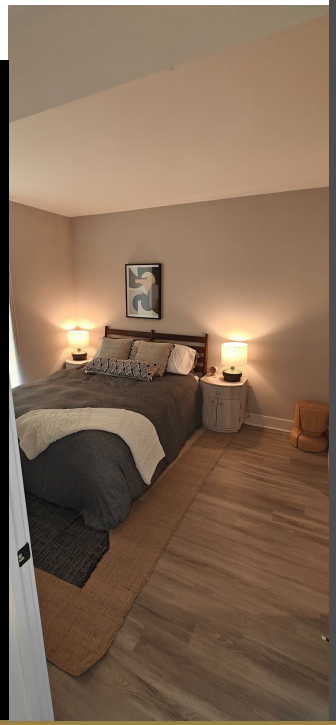


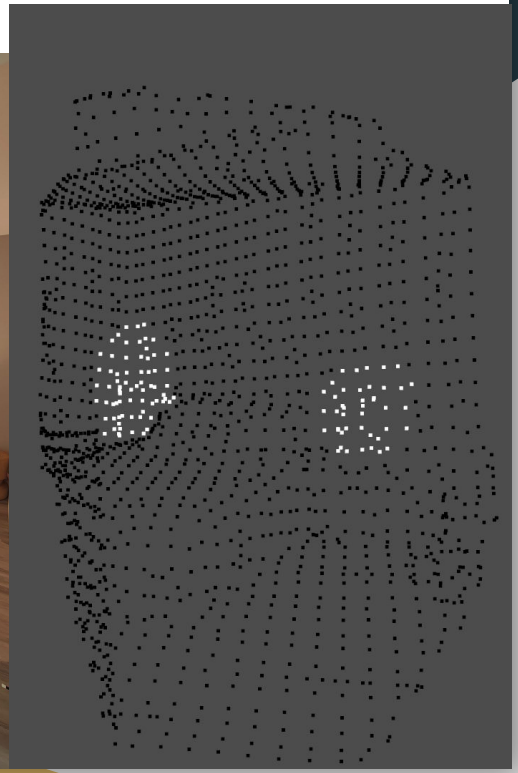
Evaluations path_image_3

Target: Lamp



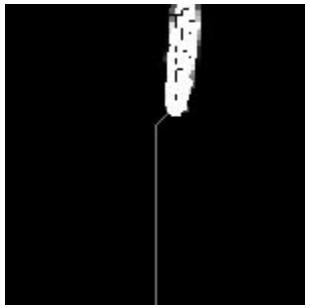


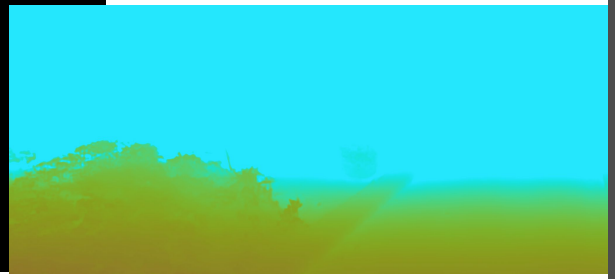


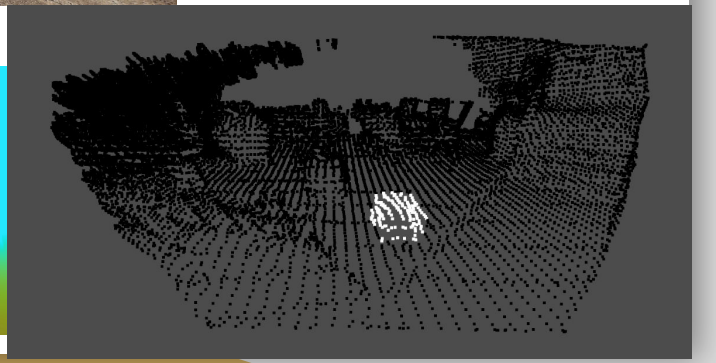


Evaluations path_image_6

Target: Flowers

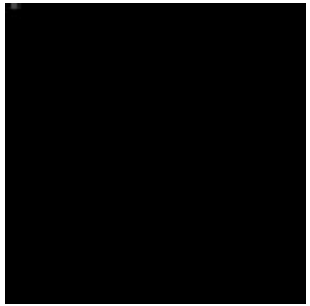


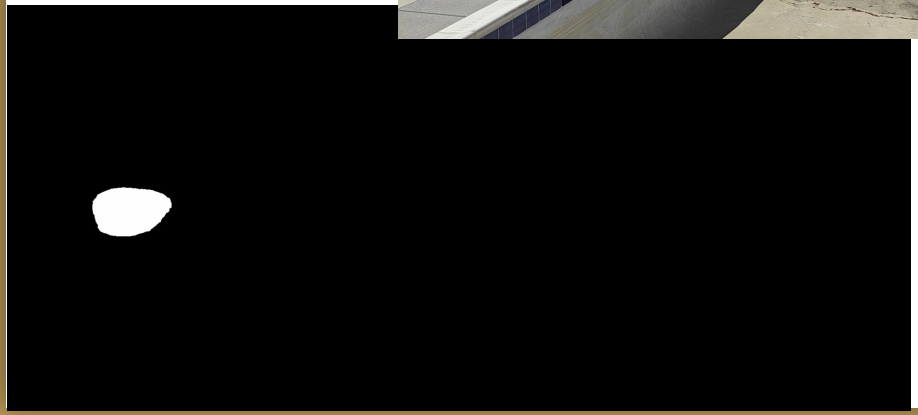


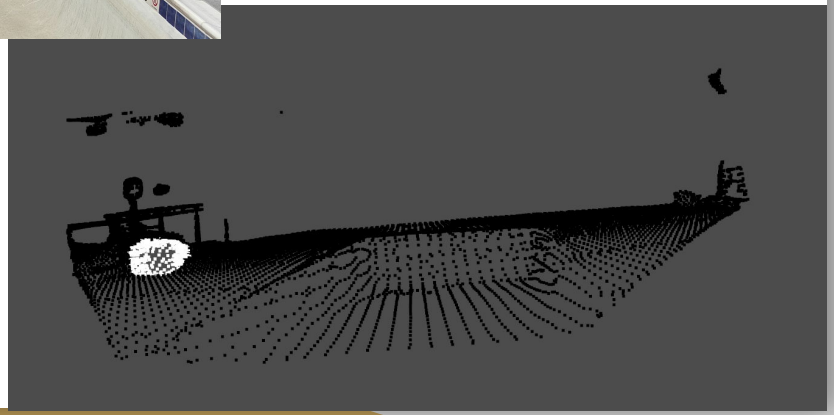


Evaluations path_image_8

Target: Pot

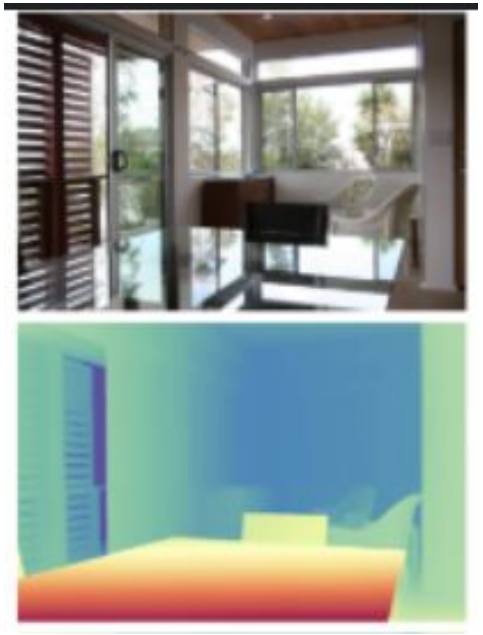






Monocular depth - Depth-Anythingv2?

We used the pretrained Depth-Anything V2 model to estimate depth from a single RGB image (monocular input). The model predicts depth maps in real-world metric units using only visual cues. We used both the indoor base and outside base variants which was trained on synthetic indoor datasets.



Evaluations: Depth-Anything

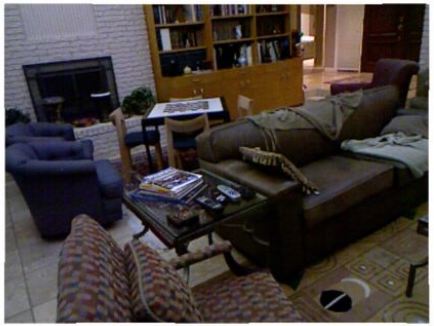
We validated our Depth-Anything-V2 model with the NYU depth data

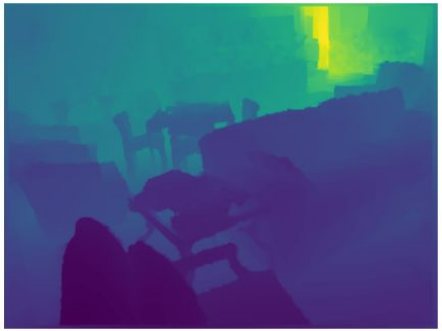
--- Final Evaluation Table ---

Metric	Value
RMSE	1.206
MAE	0.947
AbsRel	0.407
$\delta < 1.25$	0.539
$\delta < 1.25^2$	0.820
$\delta < 1.25^3$	0.949

The model achieved a decent RMSE of 1.206 and an accuracy of 54% under the $\delta < 1.25$ threshold. While $\delta < 1.25^3$ reached nearly 95%, these results are still subpar compared to state-of-the-art benchmarks. That's because Depth-Anything hasn't been fine-tuned on NYU specifically — and our validation set likely had limited visual diversity, with mostly similar indoor scenes.

With fine-tuning on more varied images — especially ones closer to the NYU distribution — we expect performance to improve significantly.



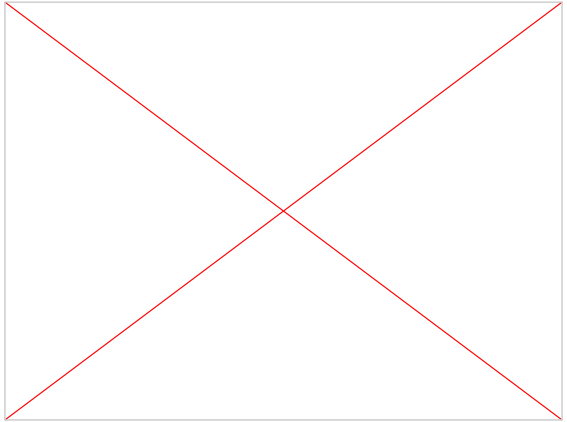


Point Cloud generation

We used open3d package as it is easier for generating point clouds along with reliable visualization.

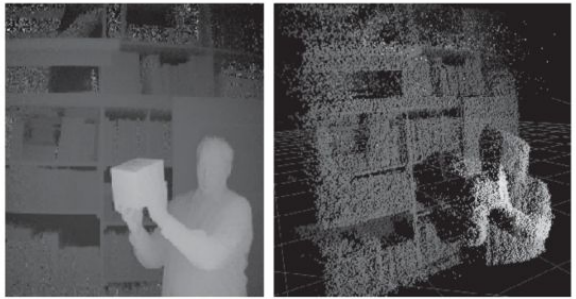
It is governed by an intrinsic matrix and extrinsic position properties in the scene translation and rotation.

Currently we assume the height to the camera to be approximately constant.



$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

2D Image Coordinates | Intrinsic properties (Optical Centre, scaling) | Extrinsic properties (Camera Rotation and translation) | 3D World Coordinates



Gridmap from point cloud

We convert the goal points and the obstacles in the scene into a 2d gridmap. Currently it is a local gridmap as a global map is not needed in this case.

The brightness signifies the goal vs risk in the scene. The Black background is empty/ unknown data. The White points are Goal points in the scene. The grey points are the obstacles in the scene, we avoid these and prevent collisions with these in the planner.

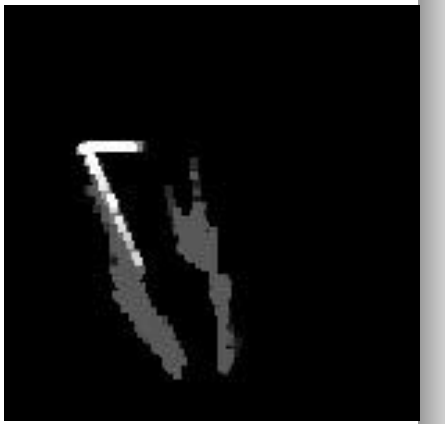
The grid is 150x150 with the grid pixel size of 0.1m.

The points from the point cloud were a bit noise since we use voxel downsampling to reduce computation and memory usage.

We used a 3x3 kernel pixel dilation to expand the objects and create cleaner gridmaps with lesser noise.

We also tried gaussian blur but found it led to loss of clarity and some goal points were lost.

Pointcloud to grid, empty cell due to downsampling



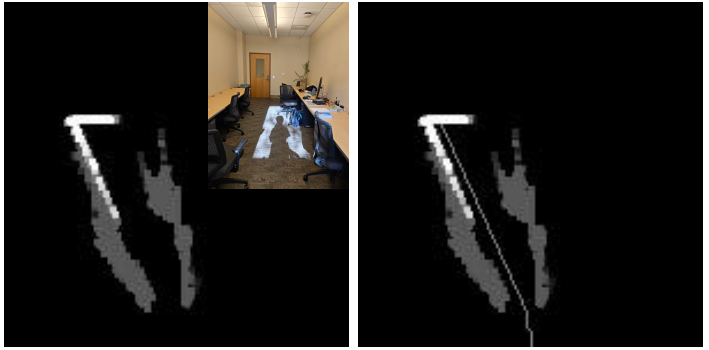
Dilation of points to reduce empty data

Path planning with modified A*

The brightness signifies the goal vs risk in the scene. The Black background is empty/ unknown data. The White points are Goal points in the scene. The grey points are the obstacles in the scene, we avoid these and prevent collisions with these in the planner. The grid is 150x150 with the grid pixel size of 0.1m.

The Goal of the planner is identified by finding the centroid of the goal object. We use `opencv::connectedComponentsWithStats`. If we use the centroid, the path may consider objects which are part of goal points as a collision object.

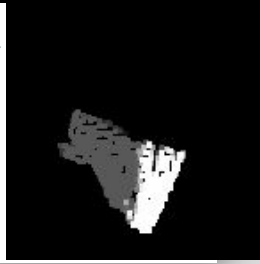
We use a modified version of A* with additional check for the object belonging to to the set of goal objects. If path is not found to the centroid, we instead use the path to the first found point that is of the goal object category (>250 value). We use a euclidean distance heuristic.



Narrow light grey line is path

Path to the centroid of the goal object

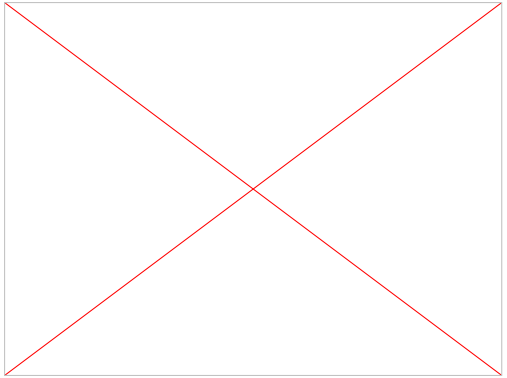
Path to the first discovered point of the object



Path planning with modified A* - Pseudocode

```
function AStarWithHighValue(grid, start, goal):
    // Initialize
    openSet = {start}
    closedSet = {}
    cameFrom = {}
    gScore = {start: 0}
    fScore = {start: heuristic(start, goal)}
    firstHighValuePath = null

    while openSet not empty:
        current = node with lowest fScore in openSet

        if current == goal:
            return reconstructPath(cameFrom, current, start)

        for each neighbor of current:
            if neighbor is out of bounds:
                continue

            if grid[neighbor] > 250 and firstHighValuePath is null:
                firstHighValuePath = reconstructPath(cameFrom, current, start) + [neighbor]
                continue

            if grid[neighbor] is obstacle:
                continue

            newGScore = gScore[current] + distance(current, neighbor)

            if newGScore < gScore[neighbor]:
                cameFrom[neighbor] = current
                gScore[neighbor] = newGScore
                fScore[neighbor] = newGScore + heuristic(neighbor, goal)
                add neighbor to openSet

        move current from openSet to closedSet

    return firstHighValuePath if exists else empty path
```

Conclusion / Discussion



Our system is able to generate point clouds for a variety of scenes.

Distance accuracy was in range of +/- 9% of actual distance (Measured using measuring tape in the picture scenes) from the photos we have taken.

90% of the tests rendered a solution from the path planner.

The lighting is an important factor and has to be considered to ensure accurate depth estimation.

Some calibration / tuning was needed to match the metric depth estimation to the actual distance measurement. We attribute this to the camera intrinsic matrix calibration having slightly inaccurate values initially.

Possible Uses

1) Using with Project Aria style glasses to help visually impaired find objects in unknown environments.
2) Robot navigation in indoor and outdoor spaces.
3) Interactive part repair segmentation and instructions

Conclusion

We’ve achieved a robust path planner for images and new environments.

Limitations:

- Lighting is a major hurdle for our system
- Depth-anything needs some calibration for estimation
- Currently only local information

Extensions:

- Identify traversable parts of the image according to use case and pathfind only along those
- 3d path planning instead of only 2d