GAS - A Visual Navigation Framework for Producing 2D and 3D Semantic Maps from Video

Alexander ”Sasha” Kuznetsov
skz@stanford.edu

Ghanshyam Bhutra
gbhutra@stanford.edu

Swaroop Pal
swaroop3@stanford.edu

Abstract

We introduce a visual navigation framework that produces 2D and 3D semantic maps, with accurate object and landmark labels, from video.

This is achieved by consuming monocular, stereo, or RGB-D video to estimate camera poses and a dense 3D map of the environment. Simultaneously, the framework detects objects in the video feed and attaches semantic labels to key items in the environment. The 3D map and detected object information are then converted into a higher abstraction, 2D map, which can be used to aid indoor navigation and motion planning for humans, robots, and AI agents.

Our framework leverages GO-SLAM [2] to estimate an environment mesh from video, and utilizes state-of-the-art models including a fine-tuned version of Detectron2's [9] Faster R-CNN [10] for object detection and a Segment Anything [11] model for object segmentation.

Our finetuned object detector is trained on multiple datasets from various sources which contain relevant indoor objects and landmarks of interests, such as chairs and entrances/exits.

We qualitatively evaluate the mapping performance on environments from the Replica dataset [3]. We evaluate our fine-tuned object detector on various quantitative metrics.

1. Introduction

Indoor mapping is an important topic for both humans and robots. Environments can change over time, so mapping environments quickly and cheaply is key, especially at a global scale. While satellite, aerial, and streetview footage can be used to automatically generate outdoor maps at a global scale, indoor mapping is a more challenging problem, due to the lack of high quality data. This is demonstrated by the lack of global, publicly available, up-to-date indoor maps.

One solution is to produce indoor maps from video. Cameras are a relatively cheap and readily available sensor that, even in the absence of LiDAR, can be used to generate maps, and can scale globally.

To solve this, we've created a system that consumes monocular, stereo, or RGB-D video to produce accurate, semantically and spatially labelled, 2D and 3D maps of indoor environments. These maps can then be used to aid indoor navigation and motion planning for humans, robots, and AI agents. 2D floor maps provide a level of abstraction that is natural for human navigation, while 3D maps describe the environment in a manner that a robot or AI agent can use for planning and acting.

Our framework leverages GO-SLAM [2] to estimate an environment mesh from video using deep-learning techniques for feature extraction and matching, and utilizes state-of-the-art models like a fine-tuned version of Detectron2's [9] Faster R-CNN [10] for object detection and a Segment Anything [11] model for object segmentation. These pieces are combined to generate 2D and 3D maps.

2. Related Work

Deep learning for visual localization and mapping is a rich and deep field. A survey of techniques in this area can be found at [19].

We reviewed key SOTA papers and models in the areas of visual SLAM, object detection and image segmentation, which are the building blocks of this project. We combined them in a novel way to produce semantically labelled 2D and 3D maps.

There exists a project [13] that labelled objects during visual SLAM, but it was limited to labelling objects along walls and only with rectangular bounding boxes. Our framework extends this by leveraging more fine object segmentation masks, thereby producing more accurate spatial labels, and by projecting properly into world coordinates, can place objects anywhere in the 3D map.

Another relevant and interesting project is [18], which describes an approach for navigating to a visual target in an indoor environment in the minimum number of steps. This approach does not require a map, and doesn't produce one, although it could probably be extended to produce one. In contrast, our framework has the explicit goal of producing 2D and 3D maps of indoor scenes.

For 3D visual SLAM, we reviewed several SOTA frameworks, particularly DIM-SLAM [17], DROID-SLAM [15], and DK-SLAM [16]. In the end we made heavy use of the existing GO-SLAM framework [2]. We chose this framework due to its higher 3D mesh accuracy, versatility (it can run on mono, stereo, or RGB-D inputs), and low memory requirement. We left this framework largely unchanged for producing an estimated environment mesh, but adapted it to our needs by utilizing its intermediate outputs: estimated camera poses and corresponding estimated depth maps for each frame. These intermediate inputs were fed into our framework pipeline.

For object detection, we reviewed existing object detectors including fasterrcnn_resnet50_fpn from PyTorch [14] and the Detectron2 Faster R-CNN model [9]. We adapated the Detectron2 model by finetuning it on additional datasets with additional relevant labels for our task of mapping indoor environments.

We also explored using DETR for object detection [1] but we abandoned it due to difficulty fine-tuning it on additional datasets.

For object segmentation, we reviewed SOTA models, particularly the Segment Anything [11] model, which we could adapt for our purposes without modification.

3. Methods

3.1. Framework Overview

The mechanism consists of 4 stages, executed sequentially (see Figure 1). They are:

1. *3D SLAM and pose estimation from a video feed using GO-SLAM [2]*
   Inputs:
   - mono, stereo, or RGB-D video frames

   Outputs:
   - dense mesh of the environment
   - estimated camera poses for to each frame
   - estimated depth map for each frame
2. *Object detection and segmentation, run on each frame*
   Inputs:
   - video frames
   - estimated depth maps, from previous stage
   - estimated camera poses, from previous stage

   Outputs:
   - object bounding boxes, labels, and segmentation masks for each frame
   - 3D point cloud for each object
3. *Object merging, which merges images across frames*
   Inputs:
   - object point clouds, generated in previous stage

   Outputs:
   - unique point clouds, merged between frames
4. *Map generation, which aligns the SLAM'd environment with the x-y-z frame*
   Inputs:
   - dense environment mesh, generated in first stage
   - unique object point clouds and labels, generated from previous stage

   Outputs:
   - 3D map with spatial object and landmark labels
   - 2D floor map with spatial object and landmark labels

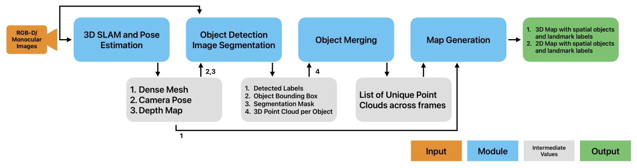

RGB-D/ Monocular Images
3D SLAM and Pose Estimation
Object Detection Image Segmentation
Object Merging
Map Generation
1. 3D Map with spatial objects and landmark labels
2. 2D Map with spatial objects and landmark labels
1. Dense Mesh
2. Camera Pose
3. Depth Map
2,3
1. Detected Labels
2. Object Bounding Box
3. Segmentation Mask
4. 3D Point Cloud per Object
4
List of Unique Point Clouds across frames
1
Input
Module
Intermediate Values
Output

Figure 1. GAS - A Visual Navigation Framework for Producing 2D and 3D Semantic Maps

3.2. Execution Stage Detail

A detailed description of each stage of the pipeline follows.

3.2.1 3D SLAM

The first step in the pipeline executes GO-SLAM on the video frames, which estimates a mesh of the scene and the camera trajectory. During this step, the mesh, camera pose corresponding to each frame, and estimated depth maps corresponding to each frame are saved (see Figure 2).

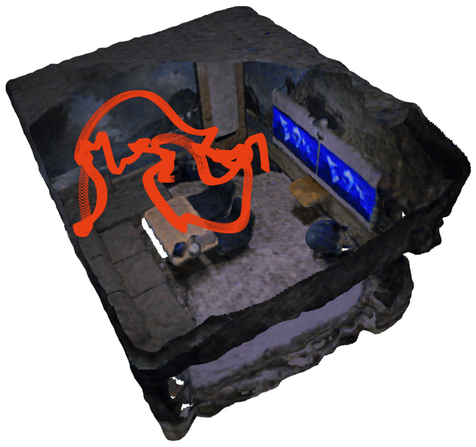


Figure 2. Estimated environment mesh and camera poses produced by 3D SLAM

The depth maps for each frame are generated by providing the mesh, the camera pose, the camera intrinsic matrix to a rendering program (see Figure 3).

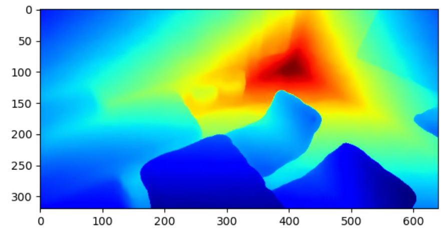


Figure 3. Estimated depth map corresponding to a video frame, produced from estimated mesh, estimated camera pose, and camera matrix

3.2.2 Object Detection

Next, we run an object detector model on a subsampled stream of frames. For this purpose, we fine-tuned our own Detectron2 model on datasets of indoor objects and landmarks of interest. We experimented both with our own model and fasterrcnn_resnet50_fpn from the PyTorch model zoo.

The object detector returns a bounding box in the original image (see Figure 4).

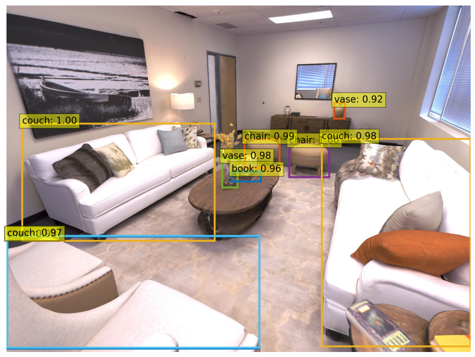


Figure 4. Object detector result on a frame from the Replica dataset

We filter objects by confidence score and then map the bounding box to the estimated depth map. We then segment the objects of interest by applying the Segment Anything model to the centre of each box (see Figure 5).

Segmentation masks for each image are projected into 3D point clouds using the camera intrinsic matrix and the corresponding depth mask, extracted from the estimated depth map. See Figure 6, which shows the projection from image frame coordinates to 3D world coordinates, and Figure 7, which shows this projection when combined with an image segmentation mask.

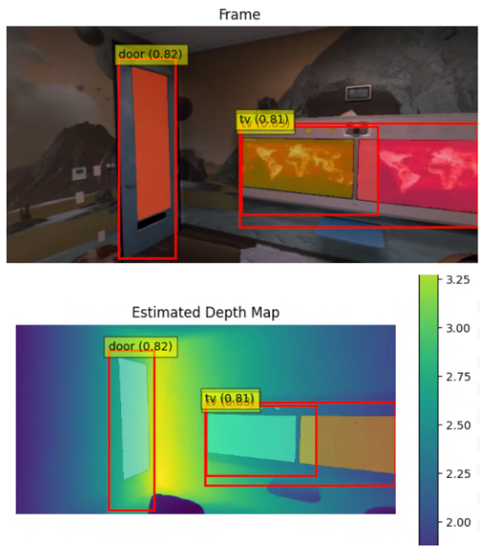


Figure 5. Object boxes and masks in the image and corresponding masks in the depth map

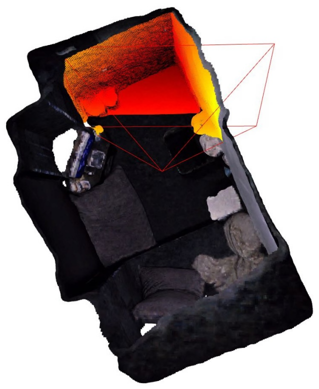


Figure 6. Image projection from camera to 3D world, produced from frame pose estimate, the camera matrix, and the estimated depth map

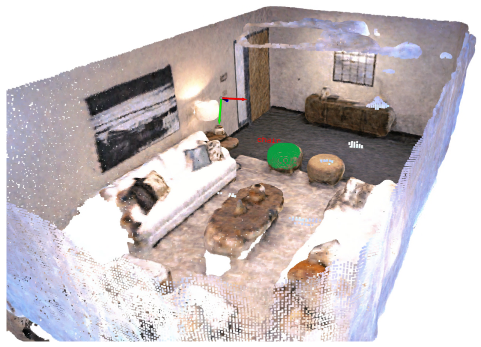


Figure 7. Chair segmentation mask (green) projected into 3D world coordinates, with object label above it (red)

3.3. Object Merging

At this point in the pipeline, we have the point clouds for each object from each frame (see Figure 8). We must now merge instances of the same object, seen across frames, into the same object.

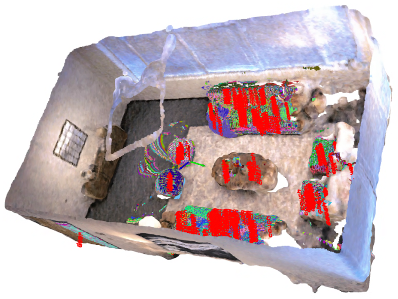


Figure 8. All the objects extracted from all frames, before merging

We merge two instances of an object if they are likely to be the same object. To determine whether two instances should be merged, we:

1. Quantize the objects' point clouds to decimeter level
2. Calculate the intersection of the two point clouds, and normalise by the smaller point cloud
3. If the normalised intersection is above some threshold, we consider the objects to likely be two instances of the same object, and merge them

To merge, we combine the two point clouds and remove duplicate points (quantized to cm level).

We finalise the list by imposing a minimum threshold on the number of frames that must have seen the same object. If an object label was detected in the same position with the same label from multiple frames, it significantly reduces the chances of it being a false positive object detection from a single frame, which we observed in some experiments.

Finally, we have all the point clouds corresponding to each unique object in the scene (see Figure 9).

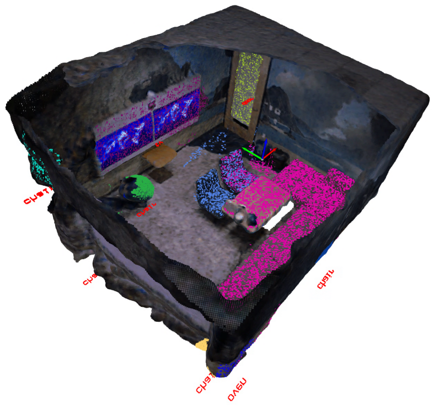


Figure 9. Labelled object point clouds in 3D space

3.3.1 Map Generation

At this point, we have created a 3D map with spatial labels for landmarks and objects. Now we align the scene with the xyz axes so that we can project the map into 2D. This process starts by using a RANSAC algorithm to identify and align the largest plane, likely the floor or ceiling, with the x-y plane. After removing this plane, we identify and align the third largest plane, assumed to be a wall, with the x-z plane. This alignment ensures accurate projection of the 3D meshes and labels into 2D, as depicted in Figure 10 showing the scene with segmented planes removed and properly oriented.

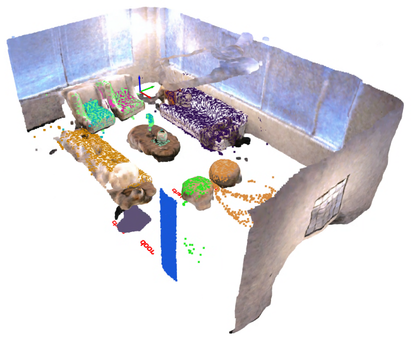


Figure 10. Labelled mesh with segmented floor, ceiling, and wall removed

Next we project the 3D mesh into 2D by removing the z axis, and then compute the 2D histogram of points (see Figure 11). We apply a median filter to reduce spikes, and then apply a Gaussian filter to smooth the density map further.

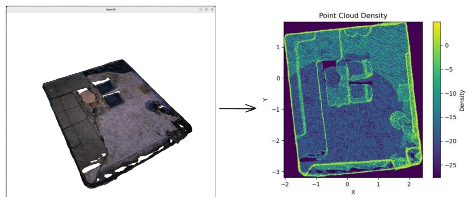


Figure 11. Creating a map from 2D point cloud density

This highlights environment features, particularly walls and objects, very well. To create patches corresponding to objects on the map, we take each object's pointcloud, projected into 2D, and compute its convex hull. This gives the exterior points of the object in the 2D space, from which we then create a polygon patch for each object (see Figure 17).

3.4. Fine-tuned Object Detector

We improved the performance of our framework by fine-tuning a Detectron2 Faster R-CNN model with additional datasets with relevant objects of interests, particularly entrances.

For our specific application in indoor navigation, we have fine-tuned the Faster R-CNN model on a custom dataset containing images of common indoor objects including doors, chairs, tables, and couches. This fine-tuning process involves adjusting the last few layers of the pre-trained network to better recognize relevant categories. This is effective for our task because it leverages the learned features of the pre-trained model, which are generally applicable to various objects, and adapts them to the specific characteristics of common objects in indoor environments.

The fine-tuning process was inspired by the comprehensive guide and scripts available at Detectron2's official documentation [9] and a custom script, finetunedetectron2 [12]. These resources provided foundational code and methodologies, which we built upon by integrating our datasets and adjusting hyperparameters to optimize the detection performance in indoor settings.

4. Data

4.1. Dataset for Fine Tuning Detectron2's Faster R-CNN [10]

In this project, we focus on detecting landmark objects that typically appear indoors and remain static. The categories include door, cabinet, refrigerator, window, chair, table, couch, bed, oven, and TV. To achieve this, we consolidated five different datasets as no single dataset met all our requirements.

The merged dataset comprises a total of 12,427 images, with 8,259 images for training, 2,971 images for validation, and 1,197 images for testing. Below is a brief overview of each individual dataset that was merged:

- **ditonadoorkasidikasya Dataset** [4]: 947 images annotated in COCO format, split into 800 training images, 102 validation images, and 45 test images. Preprocessing included auto-orientation and resizing to 640x640. No augmentations were applied.
- **furniture-ngpea Dataset** [5]: 689 images annotated in COCO format, with 454 images for training, 161 for validation, and 74 for testing. No preprocessing or augmentation was applied.
- **furniture-6ekum Dataset** [6]: 1,653 images in COCO format, with 1,446 for training, 138 for validation, and 69 for testing. Preprocessing steps were auto-orientation and resizing to 640x640, with extensive augmentation techniques applied such as horizontal flipping, random cropping, rotation, shear, Gaussian blur, and noise addition.
- **household appliances Dataset** [7]: 7,789 images annotated in COCO format, divided into 5,452 training images, 1,558 validation images, and 779 test images. Preprocessing involved auto-orientation and resizing to 416x416, with no augmentations applied.
- **Indoor objects dataset** based on [8]: 1,349 images annotated in YOLOv5 format, split into 107 training images, 1,012 validation images, and 230 test images. This dataset required conversion from YOLO to COCO format and category adjustments to align with the target categories.

4.1.1 Datasets Assemble!

For the merged dataset, specific preprocessing and augmentation steps were necessary to ensure consistency and enhance the model's robustness. The preprocessing steps were as follows:

- **Conversion from YOLO to COCO:** The Indoor objects dataset, originally in YOLO format, was converted to the COCO format using a custom script. This involved reading the YOLO annotations, converting the normalized coordinates to absolute coordinates, and saving the annotations in COCO format.
- **Category Mapping and Unification:** Categories from all datasets were mapped and unified to align with the target categories. This involved merging similar categories and removing irrelevant ones. For example, the categories "sofa" and "couch" were merged into a single category "couch." Categories like laptop and microwave were removed. Images and annotations from all datasets were combined into a single dataset for each split (train, validation, test).

4.1.2 Detectron2's Magical Configuration

Since the above section for getting the merged dataset results in COCO annotation which has images of different image sizes from different datasets we leverage Detectron2's [9] default configuration to handle the final input to the model. During the final preprocessing stage, Detectron2 dynamically resizes the images to handle different image sizes in the merged dataset which is used for fine tuning and training the Faster R-CNN model. For training, the shorter side of each image is resized to one of the predefined sizes (640, 672, 704, 736, 768, or 800 pixels) while ensuring the longer side does not exceed 1333 pixels. This dynamic resizing helps maintain consistency in image dimensions, facilitating efficient processing by the model. During testing, images are resized such that the shorter side is 800 pixels, with the longer side limited to 1333 pixels. Aspect ratio grouping is employed to group images with similar aspect ratios into the same mini-batches, enhancing training efficiency. Additionally, empty annotations are filtered out to ensure only images with valid annotations are used. Random horizontal flipping is applied as a data augmentation technique to improve model robustness. These preprocessing steps ensure that the model can effectively handle the variability in image sizes across the merged dataset, leading to better performance during training and testing.

These preprocessing and augmentation steps ensured a consistent and diverse dataset, facilitating effective training and evaluation of the object detection model.

4.2. Dataset for SLAM and Mapping Evaluation

We qualitatively evaluated our framework on indoor scenes from the Replica dataset [3]. After training our fine-tuned Detectron2 model on labelled datasets, we also qualitatively examined the performance of our model on frames from the Replica dataset to verify that it could it detect common objects in the video feeds.

To evaluate the performance of our mapping system, we qualitatively compared the spatially lablled semantic maps, in 2D and 3D, against the ground truth meshes provided by Replica (see Figures 16, 17 and 9).

5. Experiments and Discussion

5.1. Mapping

We experimented with using both an existing faster-rcnn_resnet50_fpn model and our own fine-tuned Detectron2 model (see Figures 18 and 17 respectively, with corresponding ground truth in Figure 16). We found that the Detectron2 model performed inference more quickly, reducing computation time or allowing us to improve the quality of the map generation by increasing the frame sampling rate. Further, fine-tuning our own model with additional object classes allowed us to enrich the information provided by the map (in Figure 17 we added doors, which are a key feature for navigating indoor environments). Our fine-tuned model also produced more accurate labels. For example, the faster-rcnn_resnet50_fpn incorrecntly labelled a round chair as a sports ball in Figure 18.

We observed that the quality of the map was heavily sensitive to the estimate of the camera instrinsics. If the estimate is not accurate, such as if the footage is obtained by an unknown camera, then the quality of the map will suffer. This is because the camera instrinsics govern how segmented points from frames are projected into world coordinates, so inaccuracies will distort the world coordinate projection.

We observed that incorporating segmentation masks improved the quality of spatial labels dramatically. Projecting bound boxes into world coordinates catches many points that are not part of the object of interest, which are often further in the background or foreground. When projected into 3D, these points place the edges of the object incorrectly, and also inflate the size of objects.

However, we still observe errors with segmentation masks. Sometimes the automatic segmentation incorrectly selects a portion of an object. This is somewhat ameliorated by observing the same object from multiple angles, and then merging the masks. The other error is introduced when the masks catches edges of objects in the foreground or background. For example, in Figure 17, the couch mask catches some pixels from the vase on the table, extending the couch patch towards the table.

This introduces another issue we observed. Our model sometimes confused adjacent chairs with couches, as seen in Figure 17. Hence, both classes of objects are labelled in that region: two adjacent chairs and a couch.

Similarly, we saw confusion from the faster-rcnn_resnet50_fpn, where it confused objects with similar features. For example, it mistook lamp shades for bowls.

We experimented with different input channels as well. RGB-D input produced higher fidelity maps, due to the additional depth information. Our framework also produced maps from monocular feed, but at lower fidelity.

5.2. Object Detection

In our experiments training Detectron2's Faster R-CNN model for object detection, we evaluated various configurations to understand the influence of batch size, learning rate, and the focus of layer training. To clarify the metrics, we describe them below:

- AP (Average Precision) measures precision across all classes and IoU thresholds.

  IoU, or Intersection over Union, is a metric used to evaluate the accuracy of an object detector. It is calculated as the ratio of the area of overlap between the predicted bounding box and the ground truth bounding box to the area of their union.

  $$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

  Where:

  – **Area of Overlap** is the area shared by both the predicted and the ground truth bounding boxes.

  – **Area of Union** is the combined area of the predicted and ground truth bounding boxes, minus the overlap.

- APs, APm, APl represent the average precision for small, medium, and large objects, respectively, demonstrating the model's performance across different object sizes.

- AP-{class} such as AP-door or AP-tv, measures the precision for specific classes, providing insights into class-specific detection capabilities.

We analyzed five different configurations:

Table 1. Overall Metrics for Fine tuned Model

Config	Batch	LR	AP	APs	APm	APl
Pretrained	-	-	40.22	24.16	43.53	51.98
1	2	0.00025	18.90	0.83	10.70	24.66
2	10	0.001	24.86	1.59	13.35	32.38
3	10	0.005	20.65	1.68	10.74	27.10
4	10	0.01	18.87	1.25	8.28	25.56
5	10	0.001	25.38	2.04	13.34	32.84

Table 2. Category-Specific Metrics Part 1

Config	AP-door	AP-cabinet	AP-refrigerator
Pretrained	-	-	53.15
1	23.86	0.00	20.52
2	33.57	0.55	32.70
3	28.02	1.06	25.67
4	17.65	0.32	19.32
5	30.49	0.59	35.51

Table 3. Category-Specific Metrics Part 2

Config	AP-chair	AP-table	AP-couch
Pretrained	26.23	25.91	40.74
1	28.95	12.00	32.27
2	31.44	25.76	30.92
3	27.96	27.56	35.89
4	29.32	25.51	23.70
5	32.71	26.37	33.85

Table 4. Category-Specific Metrics Part 3

Config	AP-bed	AP-oven	AP-tv
Pretrained	38.28	33.54	54.75
1	18.13	17.42	35.85
2	19.98	27.99	45.70
3	6.95	17.42	35.99
4	16.09	21.08	35.75
5	23.11	25.37	45.79

1. Configuration 1: Utilized a low learning rate and a small batch size.
2. Configuration 2: Increased the batch size and adjusted the learning rate to optimize learning.
3. Configurations 3 & 4: Experimented with higher learning rates to test the model's stability and performance under different learning conditions.
4. Configuration 5: Applied layer freezing with an optimal batch size and learning rate, focusing training on the final layers.
5. Configurations 6 & 7: Tested the limits of batch size in a resource-constrained environment, which resulted in memory allocation failures.

Among these, Configuration 2 and Configuration 5 showed the best overall performance, particularly excelling in object-specific classes such as refrigerators and TVs. Notably, Configuration 5, which involved freezing earlier layers, slightly outperformed others, suggesting that limiting training to the final layers can effectively optimize model adaptation to new tasks. This approach is particularly beneficial in transfer learning, allowing the model to fine-tune more specific features while retaining well-trained general features.

When compared to a pretrained model [21], all custom configurations demonstrated lower performance, which can be attributed to the limited training and fewer resources available in a Google Colab environment. The pretrained model showcased higher APs across all sizes and specific classes, benefiting from comprehensive training across diverse datasets and more refined hyperparameter tuning. This disparity highlights the limitations faced when training in a resource-constrained environment like Colab, particularly evident in Configurations 6 and 7, where increased batch sizes led to memory allocation failures.

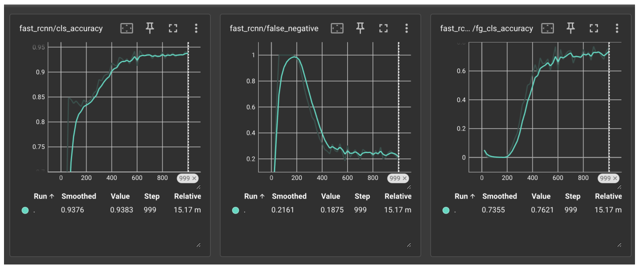
Below we provide plots from the training for configuration 2 and 5 with the following metrics:

cls_accuracy: Measures the classification accuracy, indicating successful class identification within bounding boxes.

false_negative: Represents the rate at which the model fails to detect actual objects, quantifying missed detections.

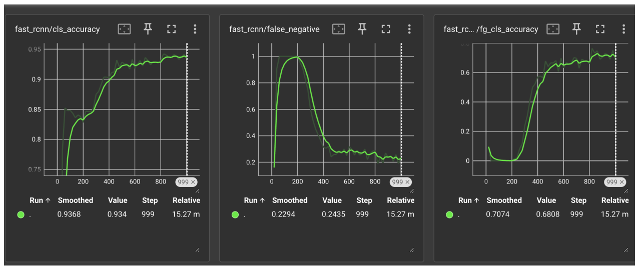
fg_cls_accuracy: Shows the accuracy for classifying foreground objects, excluding the background.

total_loss: Indicates the overall loss during training, combining losses from box regression, class prediction, and RPN tasks.



Run ↑	Smoothed	Value	Step	Relative
.	0.9376	0.9383	999	15.17 m
.	0.2161	0.1875	999	15.17 m
.	0.7355	0.7621	999	15.17 m

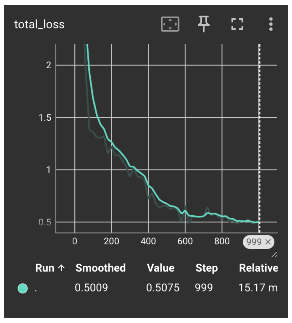
Figure 12. Results Configuration 5



Run ↑	Smoothed	Value	Step	Relative
.	0.9368	0.934	999	15.27 m
.	0.2294	0.2435	999	15.27 m
.	0.7074	0.6808	999	15.27 m

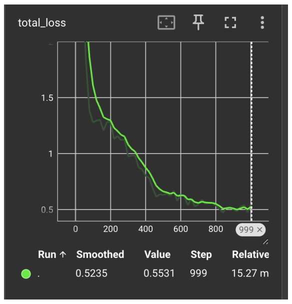
Figure 13. Results Configuration 2

Overall, the results emphasize the importance of strategic hyperparameter adjustments and training focus to op-



Run ↑	Smoothed	Value	Step	Relative
● .	0.5009	0.5075	999	15.17 m

Figure 14. Loss Configuration 5



Run ↑	Smoothed	Value	Step	Relative
● .	0.5235	0.5531	999	15.27 m

Figure 15. Loss Configuration 2

timize performance, especially when computational resources are limited. Configuration 5 emerges as the most effective approach under such constraints, demonstrating the value of freezing earlier layers to focus computational efforts on fine-tuning the model's final layers. This strategy not only conserves resources but also enhances the model's ability to adapt to specific new tasks, offering a practical approach for efficiently leveraging pretrained models in new application domains.

6. Conclusion

Fine-tuning our own object detector allowed us to detect additional objects and landmarks of interest, enriching the quality of the maps produced.

As a result, we able to produce detailed 3D and 2D maps of environments, with spatial object and landmark labels, from a video stream. See Figure 16, showing the ground truth scene, and the corresponding semantically labelled maps in Figures 17 and 10.

However, the maps are limited by the vocabulary of the object detector, and false positives can introduce incorrect labels. Object localization is improved by incorporating segmentation masks and projecting them into world coordinates, instead of only bounding boxes. Improving the quality of the segmentation can improve object localization.

The quality of the localization is also sensitive to camera intrinsics, which must be known or estimated accurately.

Localization is further improved by incorporating depth information via RGB-D or stereo video, although the system can also produce maps of environments from only mono feeds.

6.1. Future Work

Our framework could evolve to include zero-shot learning techniques, such as those explored in "Vision-Language Frontier Maps for Zero-Shot Semantic Navigation" [20]. By recognizing objects that have never been seen during training, using only textual or semantic clues, Zero-Shot techniques could significantly enhance our project's capabilities. Our framework could generate richer semantic maps and handle dynamic changes in indoor settings more effectively, responding to new objects in new environments.

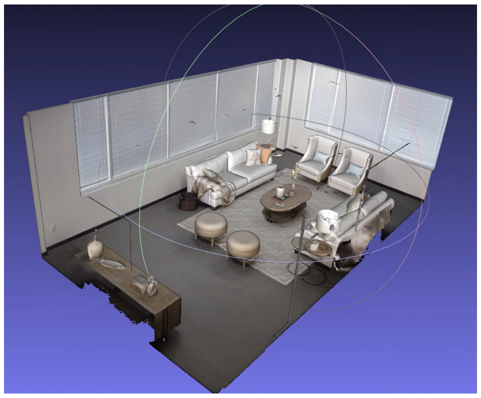


Figure 16. Ground Truth

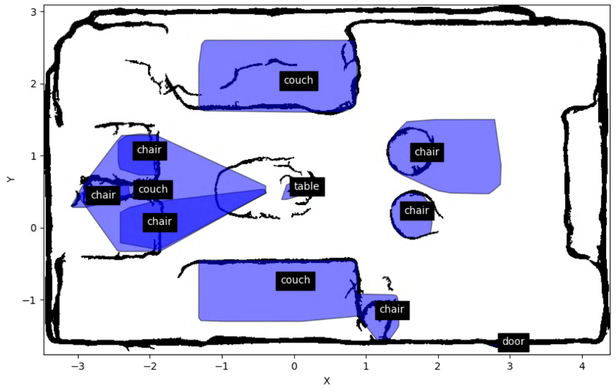


Figure 17. Generated 2D map with our finetuned object detector

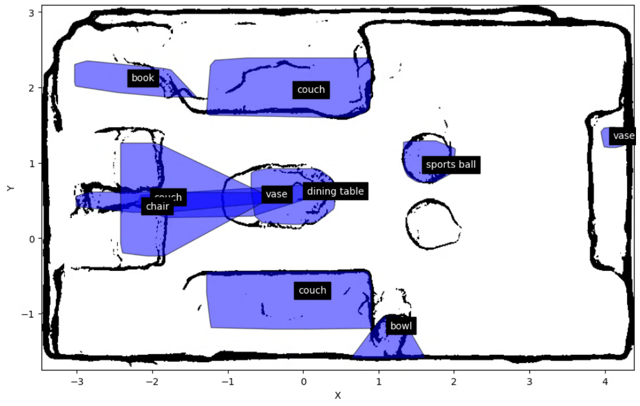


Figure 18. Generated 2D map with fasterrcnn_resnet50_fpn

7. Acknowledgements

We thank the authors of GO-SLAM [2], Detectron2 Faster R-CNN [9], and Segment Anything [11] for their foundational models which we have leveraged for our framework. We also thank various authors of datasets we

used to train our object detection model [3] [4] [5] [6] [7] [8]. We also thank the author of the finetunedetectron2 [12] for their fine-tuning script.

8. Contributions

Alexander Kuznetsov contributed most of the mapping code, heavily leveraging the GO-SLAM visual SLAM framework, available here: https://github.com/youmi-zym/GO-SLAM and modified version with mapping code available here: https://github.com/skzv/GO-SLAM. Ghanshyam Bhutra and Swaroop Pal collected and pre-processed datasets and fine-tuned the Detectron2 model and ran various hyperparameter experiments. https://github.com/ozyphus/GAS

References

[1] Facebook AI Research. (2020). DETR: End-to-End Object Detection with Transformers. Retrieved from https://huggingface.co/facebook/detr-resnet-50.

[2] Youmi, Z. (2023). GO-SLAM: A Novel SLAM Framework. Retrieved from https://youmi-zym.github.io/projects/GO-SLAM/.

[3] Julian Straub et al., ”The Replica Dataset: A Digital Replica of Indoor Spaces,” arXiv preprint arXiv:1906.05797v1, 2019. [Online]. Available: https://arxiv.org/abs/1906.05797v1.

[4] PLM Project, ”ditonadoorkasidikasya Dataset,” Roboflow Universe, Open Source Dataset, 2024. [Online]. Available: https://universe.roboflow.com/plm-project/ditonadoorkasidikasya. [Accessed: 04- June- 2024].

[5] Roboflow 100, ”furniture Dataset,” Roboflow Universe, Open Source Dataset, 2023. [Online]. Available: https://universe.roboflow.com/roboflow-100/furniture-ngpea. [Accessed: 04- June- 2024].

[6] project-ombay, ”furniture Dataset,” Roboflow Universe, Open Source Dataset, 2023. [Online]. Available: https://universe.roboflow.com/project-ombay/furniture-6ekum. [Accessed: 04- June- 2024].

[7] A A, ”household appliances Dataset,” Roboflow Universe, Open Source Dataset, 2022. [Online]. Available: https://universe.roboflow.com/a-a-emd32/household-appliances. [Accessed: 04- June- 2024].

[8] Indoor objects dataset for YOLOv5 format is a modified version of the dataset - Arduengo, Miguel, Carme Torras, and Luis Sentis. ”Robust and adaptive door operation with a mobile robot.” *Intelligent Service Robotics* (2021). [Online] from Kaggle. Modified version available here: https://www.kaggle.com/datasets/thepbordin/indoor-object-detection/data. [Accessed: 04- June- 2024].

[9] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, ”Detectron2,” 2019. [Online]. Available: https://github.com/facebookresearch/detectron2.

[10] S. Ren, K. He, R. Girshick, and J. Sun, ”Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” arXiv preprint arXiv:1506.01497, 2015. [Online]. Available: https://arxiv.org/abs/1506.01497v3.

[11] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, ”Segment Anything,” arXiv preprint arXiv:2304.02643, 2023.

[12] Wok woctezuma https://github.com/woctezuma https://colab.research.google.com/github/woctezuma/finetune-detr/blob/master/finetune_detectron2.ipynb

[13] R. Martins, D. Bersan, M. F. M. Campos, and E. R. Nascimento, “Extending Maps with Semantic and Contextual Object Information for Robot Navigation: a Learning-Based Framework Using Visual and Depth Cues,” *Journal of Intelligent & Robotic Systems*, vol. 99, no. 3–4, pp. 555–569, Feb. 2020. DOI: 10.1007/s10846-019-01136-5

[14] PyTorch, “Documentation for fasterrcnn_resnet50_fpn model,” *PyTorch Vision Documentation*, [Online]. Available: https://pytorch.org/vision/main/models/generated/torchvision.models.detection.fasterrcnn_resnet50_fpn.html. [Accessed: Insert-Current-Date-Here].

[15] Z. Teed and J. Deng, *DROID-SLAM: Deep Visual SLAM for Monocular, Stereo, and RGB-D Cameras*, arXiv preprint arXiv:2108.10869, 2021. https://arxiv.org/abs/2108.10869

[16] Hao Qu, Lilian Zhang, Jun Mao, Junbo Tie, Xiaofeng He, Xiaoping Hu, Yifei Shi, Changhao Chen, *DK-SLAM: Monocular Visual SLAM with Deep*

Keypoints Adaptive Learning, Tracking and Loop-Closing, arXiv preprint arXiv:2401.09160, 2024. https://arxiv.org/abs/2401.09160

[17] Authors, *Dense RGB SLAM with Neural Implicit Maps*, arXiv preprint arXiv:2301.08930, 2023. https://arxiv.org/abs/2301.08930

[18] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi, *Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning*, arXiv preprint arXiv:1609.05143, 2016. https://arxiv.org/abs/1609.05143

[19] Changhao Chen, Bing Wang, Chris Xiaoxuan Lu, Niki Trigoni, Andrew Markham, *Deep Learning for Visual Localization and Mapping: A Survey*, arXiv preprint arXiv:2308.14039, 2023. https://arxiv.org/abs/2308.14039

[20] Naoki Yokoyama, Sehoon Ha, Dhruv Batra, Jiuguang Wang, Bernadette Bucher, *VLFM: Vision-Language Frontier Maps for Zero-Shot Semantic Navigation*, https://arxiv.org/abs/2312.03275

[21] https://dl.fbaipublicfiles.com/detectron2/COCO-Detection/faster_rcnn_R_50_FPN_3x/137849458/metrics.json https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md