Language-Based Open-Vocabulary Robot Navigation System

Parth Rawri
Stanford University
rawri@stanford.edu

Chethan Bhateja
Stanford University
chethanb@stanford.edu

Abstract

This paper presents a comprehensive study and implementation of a language-based open-vocabulary robot navigation system. Our system enables a simulated wheeled robot to understand and execute natural language commands to navigate complex environments in ROS2. Leveraging advanced Simultaneous Localization and Mapping (SLAM) techniques and Visual Language Models (VLMs), our approach combines real-time 3D scene reconstruction with semantic understanding. We investigate two primary methods: augmenting 3D maps with semantic information for language indexing during navigation, and real-time navigation using systematic exploration and object detection.

Experiments conducted using ROS2 and Gazebo demonstrate our system's efficacy in navigating and interacting within diverse environments, but also reveal some of the limitations in how we combine VLMaps, object detection, and path planning that may yield promising directions for future work.

1. Introduction

Language has the advantage of enabling humans to cheaply specify desired behavior. For example, in the case of navigation, telling someone that you will meet them in front of the Hoover Tower is much cheaper than sending them a map pin of the exact location where you will meet them. Therefore, the goal of this project is to build an end-to-end language-based navigation system in ROS2, enabling a simulated wheeled robot to understand and execute language commands to navigate to a specific environment location, such as

"Hey Bot, go to the TV."

Our final approach combines VLMap [4], where we use SLAM with segmentation to build a language-embedded semantic map of our environment, with ROS2 packages for path planning during navigation. We also consider an alternate approach using object detection instead of building a map augmented with semantic information. We cover related work, our approach and other potential approaches considered, experiments we conducted and their results, challenges we faced and how we tackled them, and potential future work.

2. Background and Related Work

An autonomous robotic system includes:

- Sensing and Data Fusion (collecting and combining sensor data)
- Perception (interpret sensory data to understand the environment)
- Motion Planning (computing actions to move the robot while avoiding obstacles)
- Localization (determining the robot's position using sensors)
- Mapping (creating a map of the environment, for instance using SLAM)
- Control System (Converting motion plans into commands for actuators).

Robot navigation, in particular involves 2 steps:

1. Create a map of the world (movement space)
2. Navigate the robot from point A to point B (motion planning)

In recent years, LLMs and VLMs have widely been used to build truly autonomous navigation systems. One key advantage of using language in navigation is that it enables humans to cheaply specify their desired behavior, in contrast to more expensive forms of specification like location goals or demonstrations. In this work, we aim to leverage the semantic information from VLMs and LLMs to navigate to language goals.

Huang et al. (2023) [4] introduced VLMaps, a method to create spatial maps by fusing pre-trained visual language models with 3D scene reconstructions. They define

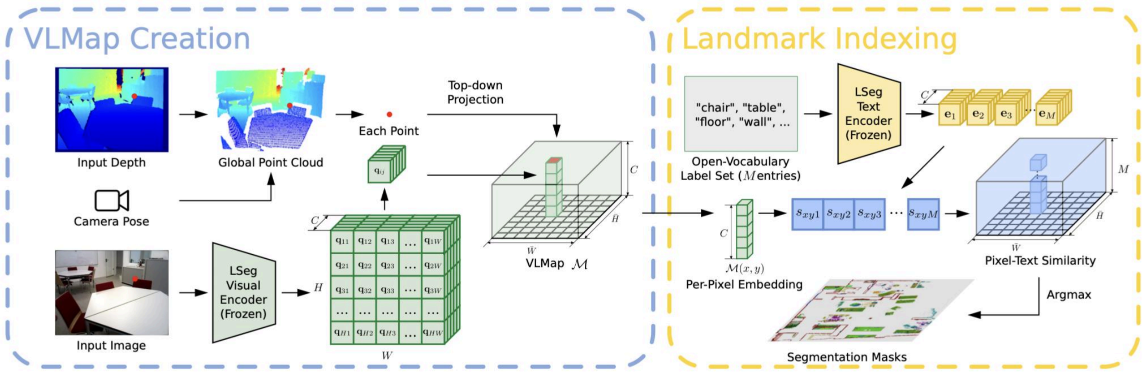

Figure 1. VLMap System Overview [4]. The VLMap is built by running SLAM with an RGB-D camera and segmenting each pixel (left). After this, we can perform natural language indexing by finding similarity in the map and taking an argmax (right).

a VLMap as a map in $\mathbb{R}^{H\times W\times C}$, where C is the language embedding size, and build this map using SLAM. Building this type of map enables language indexing of the map with open-vocabulary goals, such as "between the couch and the TV." The flexibility of this map is ideal for our goal of performing open-vocabulary navigation.

Wu et al. (2023) developed TidyBot [9], a robot that uses large language models (LLMs) for personalized assistance. TidyBot leverages the reasoning capabilities of LLMs to understand and execute complex commands, facilitating tasks such as tidying up spaces and organizing objects. The system combines object detection with language understanding to perform user-specific tasks efficiently. However, it assumes access to an overhead camera, effectively removing the step of map building, and does not embed the map with semantic information.

Zhi et al. (2024) [10] presented a system that integrates GPT-4V with mobile manipulation robots. This approach allows robots to understand and execute open-vocabulary commands by leveraging the generative capabilities of GPT-4V. The system demonstrates closed-loop control, where the robot's actions are continually refined based on real-time feedback from the environment. This approach is similar to running object detection on a camera feed rather than building a map.

Rana et al. (2023) introduced SayPlan [8], a method for task planning using 3D scene graphs grounded in large language models. SayPlan leverages the spatial and semantic understanding of 3D scenes to enable scalable task planning for robots. This approach allows robots to understand and execute complex tasks by interpreting language commands within the context of a 3D environment. Scene graphs are an alternate representation to our pixel-based map, which are perhaps more efficient at the expense of some precision.

Ahn et al. (2022) [1] developed SayCan, which grounds language in robotic affordances to enable more intuitive human-robot interactions. The system uses pre-trained language models to understand commands and map them to feasible robotic actions. This approach allows robots to interpret high-level language instructions and translate them into a series of executable actions.

Gadre et al. (2022) [3] introduced "CLIP on Wheels," a framework for zero-shot object navigation leveraging CLIP, a model trained on large-scale image-text pairs. This approach treats object navigation as a combination of object localization and exploration tasks. The system uses CLIP's visual-textual understanding to identify and navigate towards objects based on natural language descriptions, without the need for explicit training on specific objects. This capability significantly enhances the flexibility and scalability of robotic navigation systems in real-world environments, but as with many other works utilizing pre-trained models it does not combine semantic information with a map.

Minderer et al. (2022) [6] proposed a method for open-vocabulary object detection using Vision Transformers (ViTs). This approach leverages the generalization capabilities of ViTs to detect objects not seen during training, facilitating zero-shot object detection. The method enhances the ability of robots to recognize and interact with a wide range of objects based on natural language descriptions.

Ramakrishnan et al. (2021) [7] introduced the Habitat-Matterport 3D shiftset (HM3D), a large-scale shiftset comprising 1000 diverse 3D environments for embodied AI research. The shiftset includes detailed RGB-D distribution, enabling robust training and evaluation of navigation and perception models in realistic scenarios. We import one of these scenes into Gazebo and use it for navigation.

These comprehensive summaries of the related works establish a strong foundation for the development of our language-based open-vocabulary robot navigation system capable of understanding and executing complex natural language commands, facilitating effective navigation and interaction within diverse and dynamic environments. Our work builds off of VLMap [4], but we build our system in ROS2 and also consider an alternative approach using object detection.

3. Method

To achieve our goal of an end-to-end language-based navigation system in ROS2, we worked on two approaches:

3.1. Semantic Map

As in VLMap [4], we generate 3D maps augmented with semantic information, which then enables language-based indexing of the map during navigation.

3.1.1 Building the VLMap

For the purposes of this project we treated the VLMaps paper as a black box and ran their code to generate the language embedded map. To build a VLMap [4], we first generated a dataset of RGB-D images with corresponding camera pose estimates by running SLAM with an RGB-D camera. After this, each pixel is embedded using segmentation and since we have camera pose and depth, we are able to project each pixel into a 3D global point cloud as shown in Figure 1. As in VLMap [4], a given pixel, $\mathbf{u} = (u, v)$ with depth $z \in \mathbb{R}$ is converted into homogeneous coordinates $\tilde{\mathbf{u}} = (u, v, 1)$ and projected via the intrinsic camera matrix K of the RGB-D camera to project it into the 3D camera frame via the equation $\mathbf{P}_c = zK^{-1}\tilde{\mathbf{u}}$. Then, the 3D position in the world frame $\mathbf{P}_W$ is found via the equation $\mathbf{P}_W = [R, t]\mathbf{P}_c$ where $[R, t]$ are the rotation and translation to the world frame. Next, the point $\mathbf{P}_W$ in world coordinates is projected onto the ground plane via the equations

$$p^x_{map} = \left\lfloor \frac{\bar{H}}{2} + \frac{P^x_W}{s} + 0.5 \right\rfloor,\ p^y_{map} = \left\lfloor \frac{\bar{W}}{2} - \frac{P^y_W}{s} + 0.5 \right\rfloor$$

[4] where p^x_{map} and p^y_{map} represent the coordinates of the projected point on the cost grid map. Finally, we average over embeddings from multiple camera views over xy positions to obtain a well-grounded semantic map in $\mathbb{R}^{H\times W\times C}$. We experimented with ORB-SLAM3 [2] but found it very difficult to build in ROS2 and ultimately stuck to using RTAB-SLAM, which was pre-built into the ROS2 navigation2 package and was also used by the authors of VLMap [4]. For our segmentation model, we used LSeg [5], which segments pixels into the CLIP feature space.

3.1.2 Navigating with the VLMap

To navigate with the VLMap [4], we first take in a user text query and use the LSeg text encoder [5] to embed the query into the CLIP feature space. Then we take the similarity with all xy pixels in the VLMap via dot products and choose the argmax of the similarity as the goal location. Finally, we use the navigation2 package from ROS2 to perform path planning with the A* algorithm to the desired location.

3.2. LLM with Object Detection

An alternative approach that we considered was to use a LLM supplemented with an Object detection and/or a Classification model as in TidyBot [9]. Here the map entails just the spatial 3D reconstruction information through the occupancy map without any semantic information. The occupancy map in this approach is just used to systematically explore the environment during the exploration phase. Here, we wander around until the object of interest is in direct FOV, in which case the bot moves straight to the object. For the purposes of exploration, different techniques can be employed such as Frontier-based exploration. Such exploration techniques can be implemented through the readily available pre-built ROS2 packages viz. *explore-lite*. For this project, we were able to run object detection from the ROS2 camera feed and were able to navigate with the ROS2 nav2 package, but didn't manage to get this approach working end-to-end.

The downside of this naive approach is that since we don't maintain a database of seen objects with their mappings to the environment, our turtlebot3 will have to find objects it may have already seen previously from scratch for every user query. In contrast, including semantic information in the map as in VLMap has the advantage of being able to remember where objects are based on language, but this comes at the cost of map creation being much more expensive, particularly when using semantic segmentation.

4. Experiments & Results

4.1. Environment Setup

We completed our initial environment setup with ROS2 Humble and Gazebo Fortress on Ubuntu 22.04.4 Jammy Jellyfish and tested the environment using the TurtleBot3 Burger and Waffle.

After getting the TurtleBot3 rendering working in Gazebo on our VM, we were able to use teleop commands to make it move around. After this, we thought the Nav2 package looked like a useful package to build off of for our navigation methods since it contains methods for localization, mapping, and motion planning. We were able to get an initial example working with Adaptive Monte Carlo Localization (AMCL) and see how the localization estimates updated as the robot explored in the TurtleBot3 World environment, as shown in Figure 6.

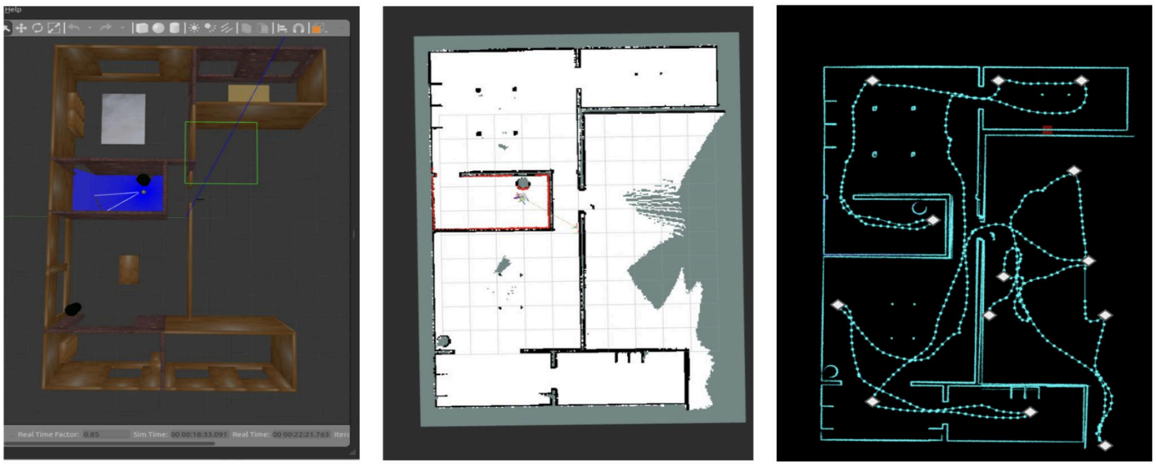
Figure 2. House Environment (Left), Generated Map (Center), Navigation paths (Right)

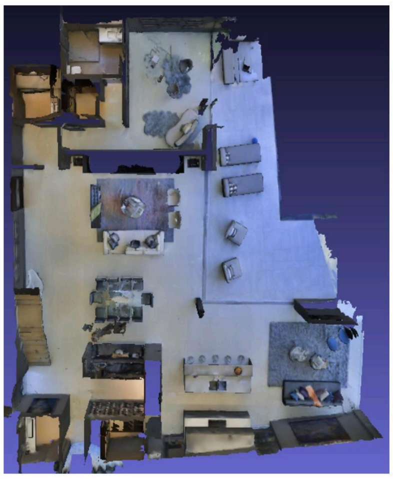
Figure 3. Matterport3D Scene

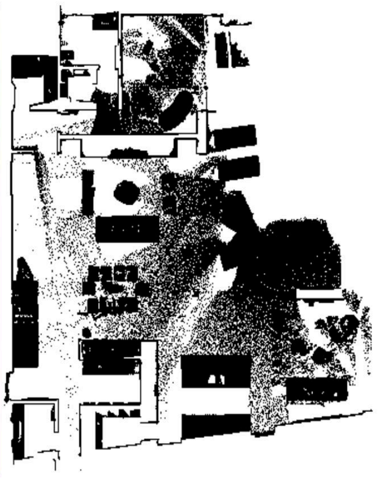
Figure 4. Occupancy Map

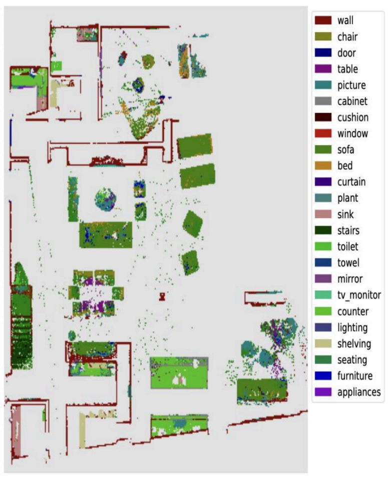

Figure 5. corresponding VLMap [4]

4.2. Core-Pipeline Setup

As a next step we implemented a core pipeline that would function as the base for our future work. This core pipeline comprised of the pre-built *ROS2 RTAB-SLAM* package for map reconstruction, the *turtlebot3-teleop* package for teleoperation, and the *turtlebot3-navigation* package for navigation. The core pipeline was implemented using a ROS2 launch file to run a TurtleBot3 Waffle. TurtleBot3 Waffle is equipped with both LiDAR and RGB-D cameras. Figure 2 shows the functionality of the pipeline in Gazebo's House Environment. The diamond marks on the navigation paths in the rightmost figure denote the different goal points assigned during our testing process.

4.3. Implementing Object Detection

We then modified the launch file of our core pipeline to include object detection with navigation. Initially, we implemented a custom-built ROS2 package for object detection which used MobileNet for object detection. MobileNet misclassified many objects in the simulated environment, as shown in Figure 7. Later, we moved to YOLOv8 object detection model offered in the ROS2 openvino-toolkit which had much higher overall accuracy. We suspect the data distribution shift to be the reason behind the misclassifications observed as the training and test data differed significantly. This was implemented for approach 2 mentioned in our methods section.

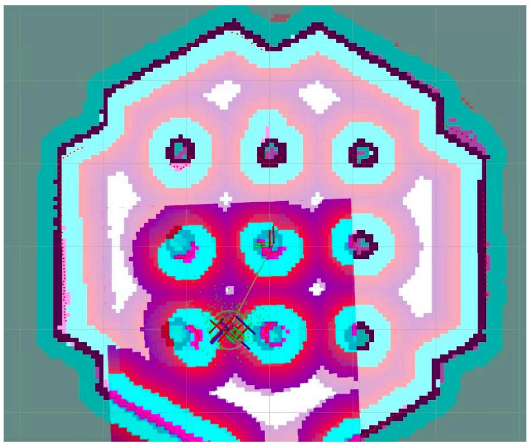
Figure 6. AMCL Localization Distribution after Some Navigation

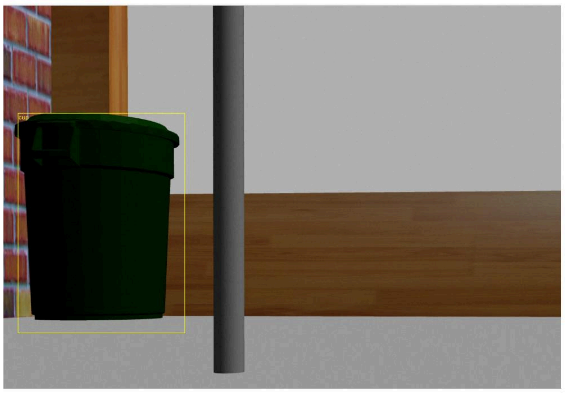
Figure 7. MobileNet misclassifies an image of a trash can from the Turtlebot camera as a cup

4.4. Generating VLMap on Matterport3D Scenes

Following our implementation of the core pipeline we imported one of the 1000 available scenes from the *Matterport3D* dataset into the Gazebo environment.

4.4.1 Matterport3D Pre-processing

The imported scene had to be configured with the right scaling, orientation, material properties, and lighting conditions to emulate the actual environment in Matterport to get only the first floor of the house environment. Furthermore, we imported the mesh into Gazebo for collisions to prevent Turtlebot3 from moving through the walls. Unfortunately, Gazebo threw errors and refused to import some of the textures from these scenes, but we were able to import the meshes themselves as shown in Figure 8, so we now had a Gazebo environment and corresponding VLMap to use for navigation.

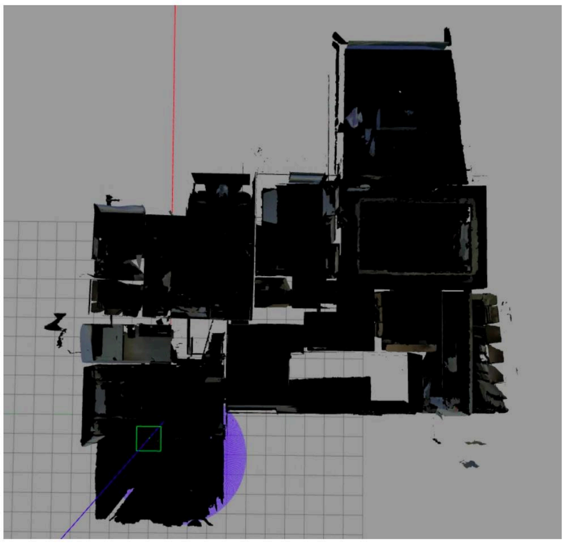
Figure 8. Matterport3D Scene imported into Gazebo environment. Some textures couldn't be imported.

4.4.2 Map generation

We then generated the occupancy map. The occupancy map as seen in Figure 4 is generated by traversing the turtlebot3 through the first floor of the imported scene. We perform the traversing using the pre-built ROS2 *turtlebot3-teleoperation* package. This generated occupancy map was then augmented with language indexing to perform navigation. The generated map features a lot of noisy black pixels scattered across the floor which we suspect is due to the non-uniformity in floor height in the chosen scene, as is mentioned in the VLMap [4] paper.

5. Evaluation

For the purposes of evaluation we performed qualitative analysis where we monitored the success of navigation by assigning several end points. Navigation trajectories in Figure 1 show the movement of TurtleBot3 when multiple goal states were defined. We also noticed a few scenarios where our model failed to navigate through close tight spaces. For instance in the Gazebo House Environment, we saw a few failures where we assigned an initial state outside the house. In these cases, the TurtleBot3's motion plan would go through the letter box and it would get stuck as shown in Figure 9. Furthermore, we encountered several failure cases while navigating through the occupancy grid generated via. MatterPort3D in Figure 4 due to noisy map reconstructions. In such scenarios, we found adjusting the oc-

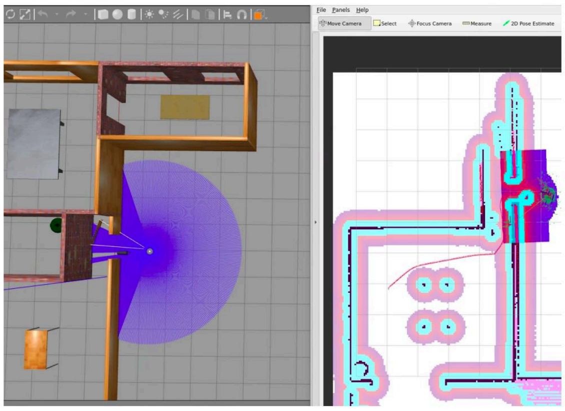

Figure 9. Our motion plans sometimes went through the letter box, causing the bot to get stuck

cupancy threshold parameter in the navigation2 package's map yaml file helpful. This threshold filters out noisy pixels with less certain occupancy values. However, this could also be addressed by averaging similarities over larger windows, as we have seen in other systems in this class.

6. Conclusion

Through this project, we attempted to build an end-to-end robot navigation system that takes in language commands and navigates to the corresponding desired locations.

6.1. Challenges

Rendering We encountered rendering errors when using X11 forwarding, and eventually deduced that these errors seemed to be specific to MacOS but we were never able to fix them. Attempting to complete the setup locally using the `osrf/ros:iron-desktop-full` Docker image also led to issues with running Gazebo Fortress. We eventually discovered that reinstalling the default CUDA drivers on our VM enabled us to render on our local Ubuntu system with *ROS Humble* and *Gazebo Fortress* but found that *Gazebo* rendered very slowly. We also attempted to install *Webot* for our simulations but encountered issues, leading us to use *Gazebo* for all our experiments. Eventually, we found that using *Google Remote Desktop* avoided SSH authentication issues and sped up rendering.

GCP VMs While building the core pipeline we encountered several errors particularly with our GCP VM. Due to the poor availability of VMs, we were locked out for several hours everyday during peak hours and were forced to create an image of our VM and spin up a new VM in a different *zone* (solution recommended by Google). This was a recurring issue and was particularly challenging because the new VM had to be created in a new *project group* because of Google's GPU quota limitation (we could only request 1 GPU per zone per project). Our requests for an increase in GPU quota beyond 1 were denied. ROS2 environments were quite fragile and we ended up rebuilding our setup from scratch multiple times. We learnt our lesson and backed up an image of our instance each time we hit a milestone, using multiple projects to get around the limits of 1 GPU/project, and using the GCP $300 free trial to somewhat reduce our expenses.

6.2. Limitations and Future Work

Given the scale of the project, we resorted to primarily using pre-built packages for our pipeline, which we connected with some custom launch files and nodes.

Object Detection Since the open-source MobileNet COCO detector algorithm is not trained to detect in a simulated environment, we observed a loss in accuracy while performing object detection in the simulated Gazebo environments. To address the distribution shift we would need to retrain the model on simulated data in the future.

Exploration Manual mapping was used to map out the environment. In future work, different approaches can be explored such as frontier-based mapping so that a robot can perform exploration/reinforcement learning based policies and mapping at the same time, removing the need to map out the environment before accepting user queries.

Evaluation As of now we rely on qualitative analysis to evaluate our navigation system. Importing the full dataset of Matterport3D scenes [7] and having automatic detection of success would enable us to use more quantitative methods similar to those mentioned in VLMaps [4].

Efficient Segmentation VLMaps [4] segment every single pixel from every single camera image, which can be very inefficient. We also considered using a different VLM such as OWL-ViT [6] that outputs bounding box coordinates instead of segmentation masks as in LSeg and Grounded SAM for approach 2. Using bounding boxes instead of segmenting every single pixel could perhaps help us run with much less memory and time, enabling us to use this on large and complex scenes.

7. Supplementary Material

The code for this project can be found here: https://github.com/chethus/vlmap_ros

8. Merging Class Project

Parth Rawri has merged this project with "AA273: State Estimation and Filtering for Robotic Perception". For the CS231A part of the project, he built ORB-SLAM3 and augmented the pipeline with the YOLOv8 object detection model for navigation purposes.

References

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can and not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*, 2022. 2

[2] Carlos Campos, Richard Elvira, Juan J. Gomez Rodriguez, Jose M. M. Montiel, and Juan D. Tardos. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6):1874–1890, Dec. 2021. 3

[3] Samir Yitzhak Gadre, Mitchell Wortsman, Gabriel Ilharco, Ludwig Schmidt, and Shuran Song. Clip on wheels: Zero-shot object navigation as object localization and exploration. *arXiv preprint arXiv:2203.10421*, 2022. 2

[4] Chenguang Huang, Oier Mees, Andy Zeng, and Wolfram Burgard. Visual language maps for robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, London, UK, 2023. 1, 2, 3, 4, 5, 6

[5] Boyi Li, Kilian Q. Weinberger, Serge Belongie, Vladlen Koltun, and René Ranftl. Language-driven semantic segmentation, 2022. 3

[6] Matthias Minderer, Alexey Gritsenko, Austin Stone, Maxim Neumann, Dirk Weissenborn, Alexey Dosovitskiy, Aravindh Mahendran, Anurag Arnab, Mostafa Dehghani, Zhuoran Shen, Xiao Wang, Xiaohua Zhai, Thomas Kipf, and Neil Houlsby. Simple open-vocabulary object detection with vision transformers, 2022. 2, 6

[7] Santhosh Kumar Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alexander Clegg, John M Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X Chang, Manolis Savva, Yili Zhao, and Dhruv Batra. Habitat-matterport 3d dataset (HM3d): 1000 large-scale 3d environments for embodied AI. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021. 2, 6

[8] Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. Sayplan: Grounding large language models using 3d scene graphs for scalable task planning. In *7th Annual Conference on Robot Learning*, 2023. 2

[9] Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas Funkhouser. Tidybot: Personalized robot assistance with large language models. *Autonomous Robots*, 2023. 2, 3

[10] Peiyuan Zhi, Zhiyuan Zhang, Muzhi Han, Zeyu Zhang, Zhitian Li, Ziyuan Jiao, Baoxiong Jia, and Siyuan Huang. Closed-loop open-vocabulary mobile manipulation with gpt-4v, 2024. 2