Let it SIMmer: Lazily-Evaluated Embeddings in Robotic Navigation and Digital Twins

Will Huey
Cornell University
Ithaca, NY
wph52@cornell.edu

Sean Brynjólfsson
Cornell University
Ithaca, NY
smb459@cornell.edu

***Abstract*—A child, an architect, and a structural engineer go on a visit to the Sagrada Família— an exotic, unfinished cathedral in Barcelona that has been in development for over a century.**

The child looks up in wonder, eyes wide and gleaming. For the child, the cathedral is a fairy tale castle, with its many spires like giant candy canes, its tourists whimsical inhabitants, and its intricate facades reminiscent of an enchanted forest.

The architect looks up with a more discerning gaze, dissecting and labeling every mix of Gothic and Art Nouveau form. To her, the Sagrada Família is a masterclass in design. She marvels at the geometry, the incorporation of hyperbolic paraboloids, helicoids, and conoids—every detail full of innovation, yet deeply rooted in history.

The structural engineer, in contrast, focuses on the pillars, arches, and the scaffolding that support the growing basilica. He recognizes the tree-like columns which Gaudí used to distribute weight and mimick nature's strength. He looks for flaws, he sees the fixes, he ponders the technical difficulties behind the magnificent structure.

And now a robot enters behind the three. What does it see?

NOTE WELL

The write-up here is foremost a summary for internal use. We explored so many tangents that much of the information here is irrelevant to any single focus, but for comprehension and completeness we wanted to document it all.

I. CONCRETE

While seeing robots on a construction site might sound futuristic, we envision them playing crucial roles in tasks such as BIM modeling, scanning construction sites for mistakes/out-of-tolerances, and playing a significant part in the creation and maintenance of digital twins. However, there are still significant challenges to face before deploying autonomous robots in a generic environment.

We started with one fundamental challenge in mind–how can we keep a robot safe while it navigates autonomously through a hazardous construction site?

We decided to build on top of Isaac Sim's Orbit framework. Its photorealistic capabilities allow us to simulate and test our robots in a virtual environment which closely mirrors real-world construction sites. This technology is already in use training robots for Amazon's warehouses, so construction sites are a natural extension.

In June, we discovered a paper named *Wild Visual Navigation* had been recently published by researchers at the Robotics Systems Lab, in which a robot learns traversability online using only image data [3]. The previous semester, we had started to assemble our own method for traversability estimation, but *Wild Visual Navigation* seemed to have better potential for use in construction sites. We got in touch with the first author and set out to implement the method inside Orbit, where we could test it on our own environments.

Along the way, we were introduced to a paper from Facebook Research: Open Vocabulary Segmentation. Known henceforth as OVSeg, this paper unexpectedly took our project in an entirely new direction. OVSeg introduces semantic segmentation with an open vocabulary (i.e., any text can be segmented at inference time, with no extra training). Initially, we intended to use it to identify fire extinguishers, pinpointing specific hazards, such as cones, areas marked off with tape, or wet concrete. We also recognized that such semantic intelligence would enhance the accuracy and depth of digital twin reconstructions.

drHowever, we stumbled upon an unexpected discovery. While working with OVSeg, we serendipitously discovered it was skilled at handling labels related to more abstract concepts, such as traversability. Intrigued by this, we shifted our focus to explore the various possibilities of these embeddings. In this discussion, we'll start by explaining the fundamental technologies at the foundation of our work and then delve into our progress in applying them.

II. THE EMERGING EMBEDDING

In machine learning, an embedding is the conversion of categorical data, like words or items, into continuous vectors within a multi-dimensional space where position conveys meaning. This is the foundation of many emerging technologies in natural language and computer vision. Mapping data into a high-dimensional space can approximate the semantic relationships between diverse objects, facilitating systems that can rely on structure to infer meaning. This helps models be robust to data that they have not explicitly seen before, making them useful in a zero-shot setting.

The idea can be taken even further, where instead of just mapping inputs of a single format into the embedding space, establish semantic correspondences between completely separate modalities can be established. Recent examples of

shared embedding spaces include video-to-animation, protein-to-molecule, and for our purposes, text to image.

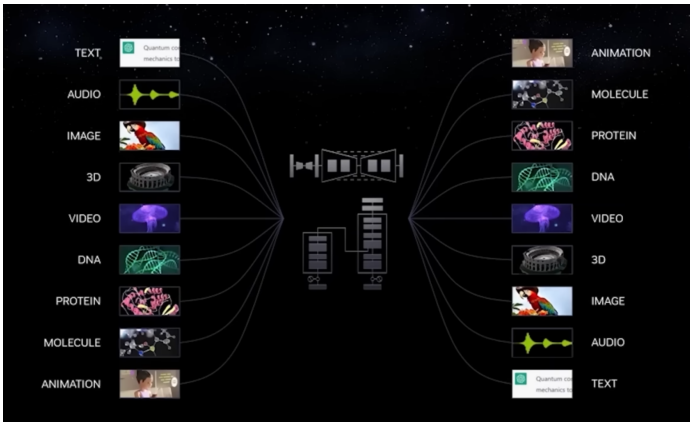

Fig. 1. A slide from Jensen Huang's keynote speech at 2023 Computex demonstrating NVIDIA's target to support the interoperability of different modalities; "anything that has structure, we can learn that language"— Huang.

A unified embedding space helps models seamlessly integrate knowledge from various sources, leading to deeper understanding and more versatile applications.

III. An Introduction to Open Vocabularies

Before the transformer, deep learning based classification methods required supervised learning over vast training sets of carefully labeled data. General datasets, such as COCO, Ade20k, and ImageNet (and countless other domain specific datasets) have been created for the purpose of training CNNs and Resnets. However, these are limited by the fact that they only work on a predetermined set of object classes. Embeddings have opened up a new opportunity: open vocabulary classification. Open vocabulary means that the classes are only known at inference time (i.e., upon query), and not at training time.

Current methods accomplish this by training a joint embedding space where both representations can coexist. For example, a model that learns to map both visual features and linguistic features to a common space could compare a *picture* of a cat to a *picture* of a dog and—simultaneously—to the text *description* "fluffy". In this space, semantically similar image-text pairs are closer to each other, while dissimilar pairs are farther apart. When querying with open vocabulary terms at inference time, these models can then locate the closest visual representation of a textual description in the shared space, enabling classification of images based on never-before-seen labels. This paradigm shift allows for more flexible and adaptive computer vision systems that are not strictly bound by the labels they were trained on.

IV. Zero Shot Segmentation

One such example of a model that utilizes a shared image-language space is OpenAI's CLIP, which exhibits impressive zero-shot and few-shot learning capabilities [6]. However it was trained specifically for image classification. While it can provide an embedding for an entire image, it is not able to provide contextual information about areas within an image on its own. For tasks that require the localization of objects (rather than just their detection), it is necessary to have semantically labeled segments. Where classification answers the question of "is this an image of a cat?", semantic segmentation answers "where are the cats in this image." Traditionally, semantic segmentation has required significant resources to label masks for large sets of images. Now, various models have been proposed to perform open vocabulary segmentation. Recently, OVSeg was able to match the performance of state of the art Resnets. OVSeg uses a method called MaskFormer to gain per-pixel visual transformer embeddings, and a CLIP adapter that maps these embeddings to the CLIP space.

Having a performant open vocabulary model is useful in scenarios when important features are not known in advance and/or when the set of important features is dynamic. These tools developed on top of such technology can then take on new forms depending on the context at hand. This comes with the benefit of additional interactivity; because the input format is both human-readable and human-writable, tweaking the model amounts to rephrasing a question.

The crux of our exploration is this: in the same way that machine learning aims to solve code we do not know how to write, an open vocabulary model attempts to take the place of data we can only describe.

V. Voxel Semantic Segmentation (VoxSeg)

For the purpose of robotic navigation, it is useful to have a full 3D representation of the environment, not just semantically segmented images. Using the camera's location and intrinsic properties at the time it took the image, the per-pixel embeddings can be projected into a 3D setting. To visualize these embeddings in omniverse, we chose to discretize the 3D space into voxels (the simplest possible implementation).

We evaluated two different methods for obtaining per-pixel image embeddings: MaskFormer [1] and OVSeg [4]. Maskformer is faster, but is only trained on the classes in the COCO dataset, whereas OVSeg implements a clip adapter for better generalization. Given the camera's intrinsics, transformation in world space, and depth for each pixel, these embeddings are then projected back into 3D coordinates. To provide a structured representation of the world, we use a simple voxel grid. If multiple embeddings from different images fall in the same voxel, they are simply averaged. We choose to use the mean for averaging rather than an exponential or Bayesian average because we are primarily dealing with static environments. There is no reason to assume that more recent images will provide embeddings closer to ground truth, and it is important not to "forget" old views of a voxel when new images are taken.

The pipeline is summarized as follows:

1) An RGB image is obtained alongside camera intrinsic and extrinsic values.
2) CLIP embeddings are computed for each pixel in the RGB image.

3) Depth information is reintegrated (e.g. from the original RGB(D) or derived later by intersection with mesh geometry) and used to project the embeddings to the nearest voxel.
4) For each voxel, sample ONE embedding from all embeddings that project to that voxel, if any.
5) The average of all embeddings which that voxel has seen is kept (all weighted equally).

OVSeg inference is quite slow, with inference on the largest model taking around 2 seconds per image and the smaller model taking 1.2 seconds on an Nvidia A6000. The CLIP adapter requires running MaskFormer an additional N times on each image to obtain foreground and background mask proposals, and then running the CLIP model on each of those proposals. When the CLIP adapter was removed, we found that runtime significantly decreased to around .8 seconds per image with the large backbone and .6 seconds with the small backbone. Despite MaskFormer being only trained on labels in the COCO dataset, we did not observe a significant performance drop on most out of domain labels.

Additionally, projecting the embeddings for every pixel in the image onto the voxel grid was quite slow. To decrease the runtime, we decided to randomly sample an embedding for each voxel that appears in the image. Given just one image, this could cause loss of information. For example, if the majority of a voxel contains a fire hydrant, but the one pixel outside of the fire hydrant is selected, its embedding will be much further from "fire hydrant". However, the entire purpose of Voxseg is to average the embeddings over different viewpoints. As the number of images with a pixel correspondence to a certain voxel increases, the embedding within that voxel approaches its expected embedding. In the limit of images, the performance of the model with respect to the ground truth label of the voxel is entirely determined by the variance within the voxel, which is a function of its resolution. To improve model performance, we just need to increase the resolution, which comes at the expense of compute speed.

OVSeg and CLIP use a Vision Transformer (ViT) to obtain embeddings. In the original Vision Transformer paper, it was shown that the positional embeddings for nearby image patches have a lower cosine distance. Furthermore, many of the attention heads have a mean attention distance that indicates global attention [2]. This means that if an image contains a fire hydrant, the embedding for each pixel in that image will be closer to that image's fire hydrant embedding. We assume that there is some ground truth fire hydrant embedding, and that it can be obtained by averaging the embeddings for every fire hydrant pixel, in every image, from every possible viewpoint. Thus, it is less important to average embeddings over pixels within an image, because this has (to some extent) already been done by the attention layers. Instead, by averaging just one embedding for each voxel over many images, we hope to achieve a more accurate voxel represenation.

The final computation issue involves storing the voxels. Each voxel cell contains an embedding of size 768. Existing robotic centric elevation mapping methods use an xy grid dimension on the order of 200x200. With these dimensions, storing an embedding in every grid cell requires 1.22 GB of memory. In order to even be stored on an Nvidia A6000 with 48GB VRAM, the z dimension of the voxel grid must then be less than 40. This memory requirement could be significantly reduced by using sparse voxel methods. However, due to constraints on time this summer, we were forced to limit the method to smaller environments.

VI. VoxVis

We set out to create an Omniverse extension to visualize the embeddings. However, we were limited by compute power and time to develop, and instead opted for an approach that performs the compute on one machine and the Omniverse rendering on the other, using ROS as a bridge.

1) Insufficient VRAM: Omniverse and OVSeg both have high computational demands.
2) We anticipated that the dependencies were not going to be easy to resolve, as Omniverse/Isaac-Sim itself has a very particular way in which it interacts with its python interpreter.
3) A ROS interface is particularly useful if we want to easily read in rosbags and potentially even work with a live robot.
4) It lets us split the computational burden between two machines.

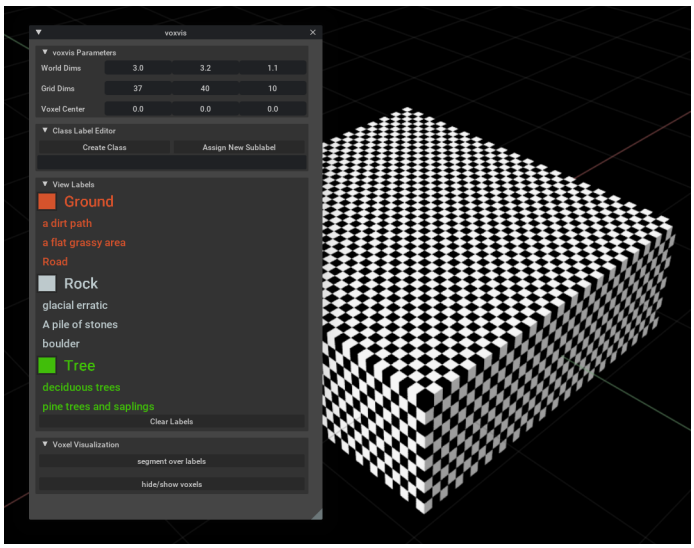

Fig. 2. **The Voxvis Extension** as it appears in Omniverse. The voxel checkerboard is a live-preview of the domain while the user is changing their voxel domain to help them position it in their scene. The interface is shown with dummy classes and their sub-labels filled in. There are elements to adjust the domain and resolution of the voxels, add classes, group sub-labels labels to classes, and modify the class colors.

VII. Open Vocabulary Traversability (OVT)

Lazily evaluated embeddings have another application: robotic navigation. Modern mobile robots need to navigate complex environments with boundless types of obstacles and

terrain. Construction sites are a prime example: the standard materials, equipment, and hazard signs vary wildly in different regions and countries. A robot might be interested in avoiding different types of objects, from wet concrete to humans. Given the lack of publicly available data of construction sites and the costs associated with labeling datasets, it is not practical to train a new model for every new case. With an open vocabulary model, the robot can easily adapt to new environments (for example, by being prompted to avoid red and white tape in Switzerland, and yellow tape in the United States).

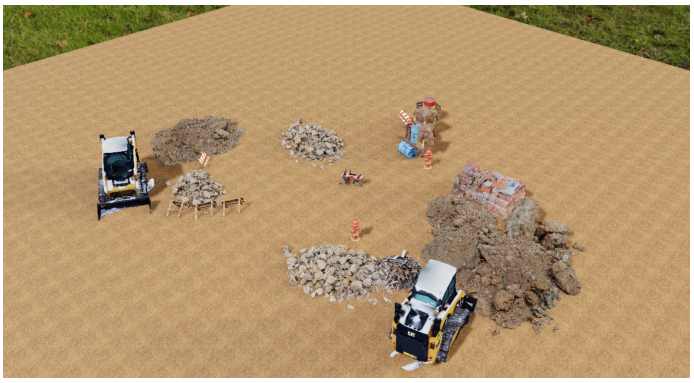

Fig. 3. A screenshot from Omniverse previewing a basic test construction scene we assembled to test our segmentation and visualization methods. The assets are meshed scans of real world objects we downloaded from SketchFab.

We have thus far discussed using nouns representing tangible objects to prompt the model. While testing out various prompts this summer, we decided to try a more abstract idea: traversability. Using our VoxSeg extension, we walked an Anymal C around in simulation, allowing it to take 60 pictures of its environment from various views. Then, we entered the classes “Traversable,” “Untraversable,” and “Obstacle.” The results are shown in Figure 4.

Fig. 4. Visualizing the Voxel Segments. The colors correspond to the classes as defined in the extension. Classification was performed online using the RGB and depth images taken by the ANYmal after walking it around in simulation.

From no more than the abstract concepts, the model was able to correctly identify the voxels along the ground as “traversable,” those approaching the dirt/rubble piles as “untraversable,” and the ones beside the heavy machinery as “obstacle.” It appears that embeddings are rich enough to encode information about traversability. Of course, this doesn’t take into account the differences in what an Anymal C robot will find traversable, versus a different type of quadruped, human, car, or anything else. We anticipate this method failing in areas that are robot-specific (our expectation is that it mimics human perception of traversability, because this is, on average, what the CLIP training set would provide). However, this may be more of a feature than a bug. Most existing methods for traversability analysis rely on some heuristic for defining a traversability threshold. In elevation mapping, this is usually the maximum stepping height. With this approach, the only human input required is the prompts used by the model.

Figure 5 shows the results of applying OVSeg to images of a construction site taken by an Anymal D. The algorithm is able to identify most obstacles and extremely untraversable areas. We are now working on empirically benchmarking OVT against the current state of the art in traversability analysis. Figure 6 shows the semantic traversability predictions overlaid on top of the robot-centric elevation map generated during one run through the site. Overall, the OVT predictions are quite similar to the elevation map.

Aside from being an interesting demonstration of the power of embeddings, open vocabulary traversability provides a few major benefits over traditional methods. By definition, it allows for more general semantic-based traversability analysis, without the need for collecting data or retraining models. As an entirely vision-based method, it does not rely on expensive lidar scanners, and it can avoid the various pitfalls that come with height mapping. Finally, it removes the need for careful tuning of parameters by providing the robot with a representation of the environment’s traversability that is closely aligned with how a human would label it.

Fig. 5. Randomly selected traversability masks generated using the prompts ”something an Anymal robot could walk on” and ”other”. Images were taken on a typical large construction site by an Anymal D, and the masks were generated using the largest OVSeg model. The rightmost mask shows a step being classified as traversable

Our current framework for these experiments is to integrate semantic labels into an existing path planner on the Anymal D robot. We are using a GPU-accelerated elevation mapping method developed at the RSL to composite our traversability

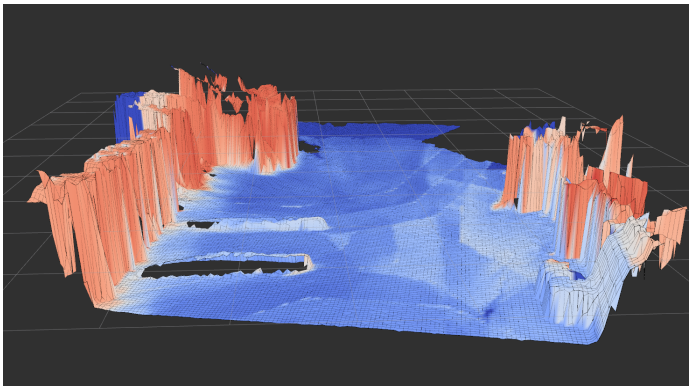

Fig. 6. Semantic traversability predictions overlaid on top of the elevation map of the environment. Red areas are predicted to be untraversable by OVT, and blue areas are highly traversable. On the right is a "step" that would normally require some heuristic for the maximum traversable step, if using only geometric methods.

signals over the estimated traversability from both geometry [5]. In the future, we hope to extend our "slow cooked" voxel approach to run on the robot, which would allow it to build a semantic representation of the environment instead of immediately evaluating the embedding classifications.

VIII. Wild Visual Navigation Orbit (WVN)

This is our implementation of Wild Visual Navigation (WVN) in Orbit. We have nearly replicated the WVN paper in Orbit, addressing most aspects and working on a few remaining elements for correctness. While the original implementation was built on top of ROS and rospy, we successfully recreated this functionality within Orbit, along with creating similar visualization tools and outputs. In order to facilitate randomized experiments and planning integration, we also developed Isaac Planner, a python package that provides environment graph generation, local planning, and path tracking functionality for quadruped robots in Isaac Sim.

Unfortunately, we are currently facing a major bug in the WVN learning pipeline that we have yet to narrow down. This bug is characterized by excessive confidence during the initial phase, but once the first traversability signal is received (after the robot has walked enough to start learning from its movements), everything suddenly drops to 0. Images below show masks taken from a test run. Each image shows the current traversability scores (left) and the confidence scores (right) overlaid on the current view of the robot in the forest environment.

IX. Isaac Stage

We have developed a Python repository named "Isaac Stage" with the capability to create randomized environments, parametrized over a set of assets and a terrain function (and some rules for more fine-grained control). One of the notable features of this repository is its versatility, as it can be seamlessly expanded to accommodate various types of terrain, each with its own set of rules for spawning assets. In our endeavor to establish a direct basis for comparison with Wild

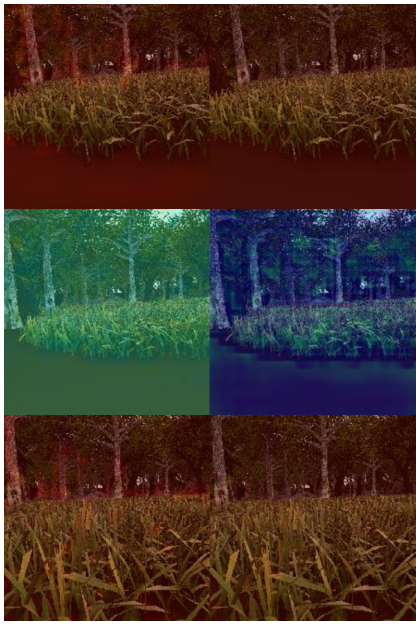

Fig. 7. The state of our incomplete implementation of *Wild Visual Navigation* in Isaac Orbit. The left image represents the traversability score and the right image is the confidence score (as defined in the WVN paper). Currently the training phase has unresolved issues but the core framework is implemented. These are signals are shown overlaid on top of the current view of the robot. **Image 1**: At the beginning, both confidence and traversability are near zero everywhere. **Image 2**: After mission nodes start to register, confidence begins to improve (blue means higher). **Image 3**: Once the supervision signal from the footpath is registered, both signals drop to zero everywhere.

Visual Navigation, a notable research paper, we opted to emulate a forest environment. This choice was influenced by the fact that the Wild Visual Navigation paper's testing was centered around pathways within a forest setting. To facilitate this emulation, we present a sample terrain that has been generated entirely through parametric means, devoid of any manual intervention. It's worth noting that the renderings we present are developed in-house and accurately represent the visual input received by the robotic system. This capacity for generating parametric terrains and the fidelity of the rendered images allow for meaningful evaluations and benchmarking against existing approaches like Wild Visual Navigation.

X. Real Construction Site Data

In our study, we centered on the analysis of real-world construction site data. The site we scanned is located at Schulstrasse 44, 8050 Zürich, Switzerland, though we have yet to finalize an appropriate nomenclature for the building. The data which we succeeded to record consists of approximately 30 minutes of data navigating a construction site from an Anymal. This data was fragmented into six distinct rosbags. The site itself had diverse lighting conditions (internal, external, skylit, dim hallway, etc.), several flights of stairs, different stories (which were completely distinct from each other), some puddles, glass, pits (though barricaded), and more.

While the Anymal was charging, we also conducted scans using a Leica point cloud scanner. These scans were carried

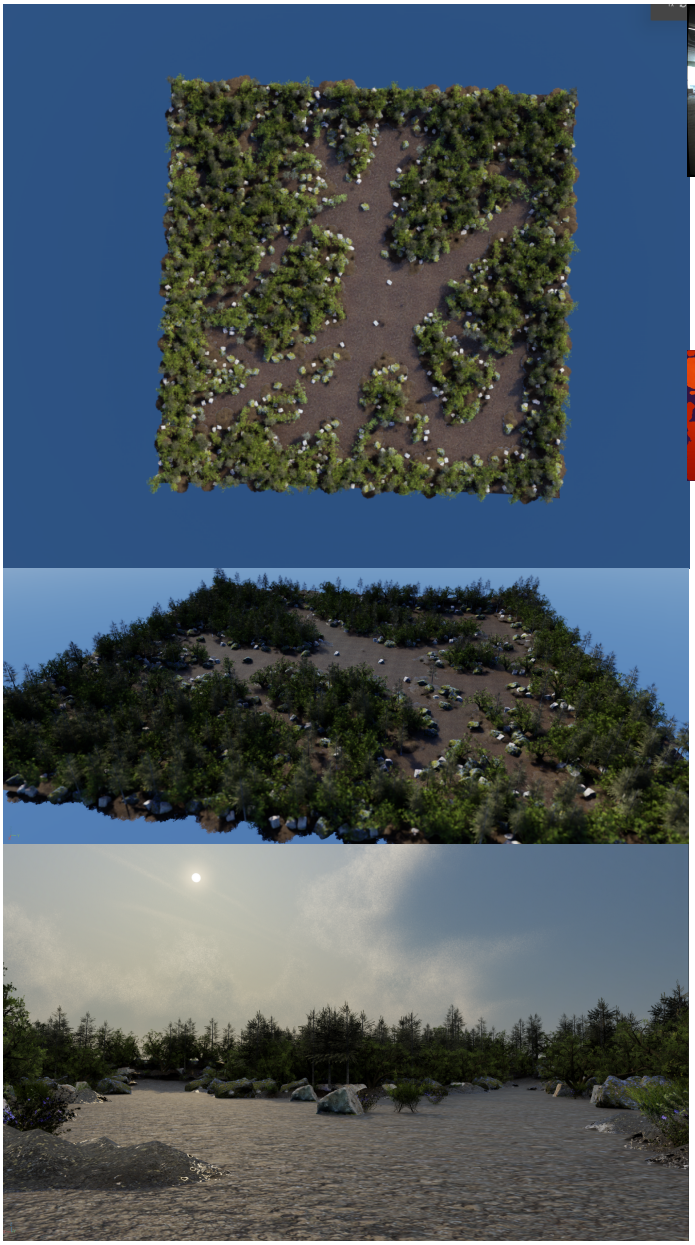

Fig. 8. A procedurally generated forest environment created as-is via IsaacStage. All three pictures taken of the same stage. No post-processing effects were added; screen captures taken directly from the Omniverse viewport.

out specifically around the first floor of the construction site, coinciding with the domain of the first rosbag.

Key to the live collection was David Hoeller; he transported the Anymal to the site and explained the procedure for operating and collecting data. He also piloted the Anymal during the runs. *(aside, Sean): It seems people familiar with these robots have a sixth sense just like a farmers does with cattle. Farmers can sense when a cow's about to kick (from inexplicable signals), and it seems a skilled Anymal operator knows much the same when it comes to the robots–they are much more hefty than one would expect.*

Fig. 9. Example of data taken from the RGB front wide angle camera, taken from the first mission. These images were taken in conditions identical to use-case, i.e., the robot is in active motion (no time to look pretty); the images arrive in the resolution 1080x1440 as as compressed jpg images–we are not sure what the compression ratio is.

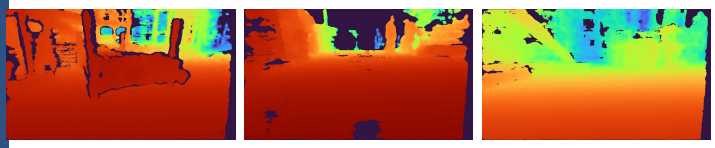

Fig. 10. Example depth camera data taken from the front RealSense camera, 848x480. The base format stores the image in units of millimeters, by observation, between 0.500m and 25.000m. The depth images were normalized and passed through the turbo colormap reversed–so red represents near and blue far. The dark purple is where the depth camera reported '0' due to a level of uncertainty. Typically caused by surfaces nearly parallel to the camera, high concavity, or being too far away.

Example of depth camera data taken from the front RealSense camera, taken from the first mission. The data was normalized and the Jet colormap was used.

XI. Future Plans

To summarize, these are our main accomplishments from this summer:

- Converted the *Wild Visual Navigation* framework into Orbit; even though learning itself does not succeed (from failing to keep the original model in tact while converting) the input/output channels involved are functional so it should be a plug-in fix.
- Developed Isaac Stage, a python package that automatically generates realistic environments in Isaac Sim given a set of assets and a function which governs terrain generation (and some rules for asset placement).
- Developed Isaac Planner, a python package that handles mobile robot motion planning and path tracking in Isaac Sim
- Developed VoxVis, an Isaac Sim extension whose scope is to visualize arbitrary voxel data quickly. The current implementation is tailored for our needs and has buttons to provide open vocabulary semantic queries of a scene and visualize the results.
- Adapted OVSeg and MaskFormer to run on top of an existing mapping framework for the Anymal (Elevation Mapping *Semantic* Cupy).
- Collected a dataset of about 30 minutes of odometry, images, and point cloud information from an Anymal D navigating a large construction site.
- Demonstrated accurate semantic traversability analysis live on an Anymal D running at near-interactive times.

From here, we have two main goals.

1) **Continue experiments with semantic traversability.** We can improve the results of our models in various ways without sacrificing performance (such as Visual Prompt Tuning and prompt engineering). We would also like to directly compare our approach with existing methods in traversability analysis, and see if we can achieve state of the art results by fusing them together. Given our promising initial results, we think that these experiments could result in a new method for robot navigation and provide insights into the structure of the CLIP embedding space.
2) **Rewrite and optimize the VoxVis extension.** This will include a refactor that uses sparse matrix methods and CUDA kernels to operate on the voxels, rather than our current method that stores a rectangular voxel grid. We believe that VoxVis demonstrates the powerful capabilities of Omniverse rendering and the potential for AI in digital twins, and we hope that it could be used by researchers and designers alike.

XII. Acknowledgements

We thank Dr. Don Greenberg for motivating this project, establishing the collaboration with the Robotic Systems Lab, and providing funding for the summer. We thank David Hoeller and Jonas Frey for introducing us to the RSL, answering our many questions about the Anymal software stack, and guiding our project. We thank Leul Testefaye and John Wolford for tirelessly helping us debug the more complicated parts of our project. We thank Dr. Marco Hutter for accepting two undergraduates from across an ocean into his lab for the summer, and Maria Trodella for making sure our arrival at the lab was seamless.

References

[1] Bowen Cheng, Alexander G. Schwing, and Alexander Kirillov. Per-pixel classification is not all you need for semantic segmentation, 2021.
[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.
[3] Jonas Frey, Matias Mattamala, Nived Chebrolu, Cesar Cadena, Maurice Fallon, and Marco Hutter. Fast traversability estimation for wild visual navigation, 2023.
[4] Feng Liang, Bichen Wu, Xiaoliang Dai, Kunpeng Li, Yinan Zhao, Hang Zhang, Peizhao Zhang, Peter Vajda, and Diana Marculescu. Open-vocabulary semantic segmentation with mask-adapted clip, 2023.
[5] Takahiro Miki, Lorenz Wellhausen, Ruben Grandia, Fabian Jenelten, Timon Homberger, and Marco Hutter. Elevation mapping for locomotion and navigation using gpu, 2022.
[6] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

Appendix

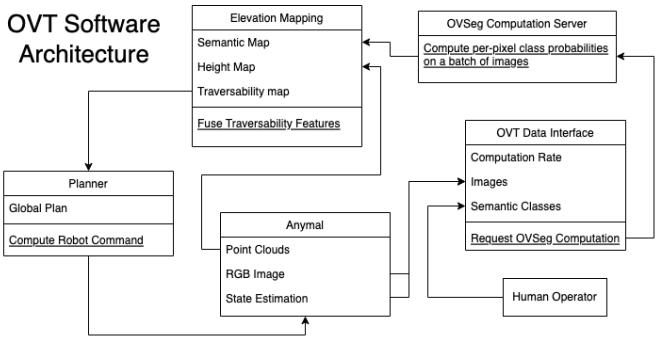

Fig. 11. The ROS system design for running OVT on the robot

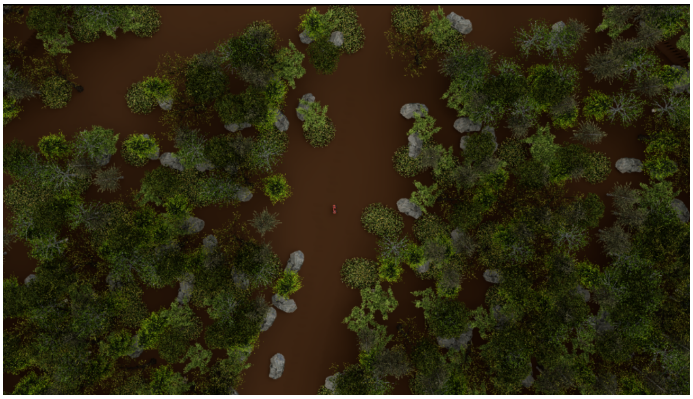

Fig. 12. A procedurally generated forest environment rendered in Omniverse which we used to compare the classifications shown below. We simulated the robot walking around for a few minutes to accumulate different perspectives on the environment.

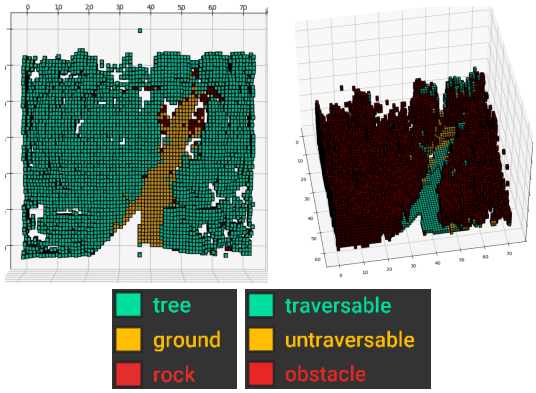

Fig. 13. The results of segmentation on the forest scene using VoxSeg performed on two sets of labels **Left Image**{tree, ground, rock} and **Right Image**{traversable, untraversable, obstacle}, each colored teal, gold, and red, respectively. Visuals created with Matplotlib.