Semantic Neural SLAM 2023

Bradley Yu bcyu@uchicago.edu
Douglas Williams douglasmsw@uchicago.edu
University of Chicago

Abstract

We present Semantic Neural SLAM, a method that leverages modern semantic embedding spaces to enable more consistent Neural Style Field view synthesis of a novel environment. When applied to environment mapping problems, it reduces the number of image data points necessary for training and increases scene model quality.

1 Introduction

When navigating a novel environment, robot agents must maintain an accurate estimate of their current pose while simultaneously modeling their surroundings. This has given rise to various Simultaneous Localization And Mapping (SLAM) algorithms over the years. These algorithms use observed landmarks as position fixes, so a robot whose estimated pose drifts astray can correct themselves upon seeing a familiar piece of geometry.

In order for this to work, the robot must be able to confidently identify and distinguish between landmarks. Any mistake could lead to wildly misguided position estimate updates. This has been dubbed the "data association problem."

A potential solution is to feed more context about the enviroment to the robot. Semantic Neural SLAM takes this to the next level by creating a granular, 3D representation of the environment based on what the robot has seen. Using input RGB images along with depth sensors, we are able to learn an implicit feature mapping of the surrounding environment, which can be queried for position and explicitly rendered as a 3D polygonal mesh.

1.1 Motivation

The field of computer graphics has long sought efficient ways of modeling scenes in 3D. Some of these recent innovations can be ported over to the mapping and data association problems in the field of robotics.

Robots are often equipped with simple RGB (red, green, blue) or RGB-D (red, green, blue, depth) cameras that capture information about the scene around them. We wanted to leverage this data source to build a robust representation of the robot agent's environment so it could accurately estimate its pose from novel positions.

To accomplish this, we turned to Neural Radiance Fields (NeRFs), which learn a function mapping between images and camera position. As the robot explores, they build an implicit representation of the scene stored as these mapping functions. The question becomes, how can we reduce the number of images necessary to build this implicit scene representation? And how can we help the function accurately predict how the scene will look from novel viewing angles?

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

2 Related Works

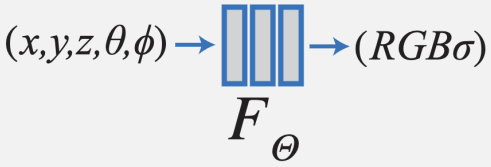
2.1 An Overview of NeRFs

NeRFs are fully-connected neural networks that approximate the function to map from a camera position, parameterized by coordinates and viewing angle (x, y, z, θ, ϕ), to output image, represented by a tensor of size $width \times height \times RGB \times \sigma$: $[w, h, 3, 1]$. σ here denotes the view-dependant emitted radiance from a point[1].

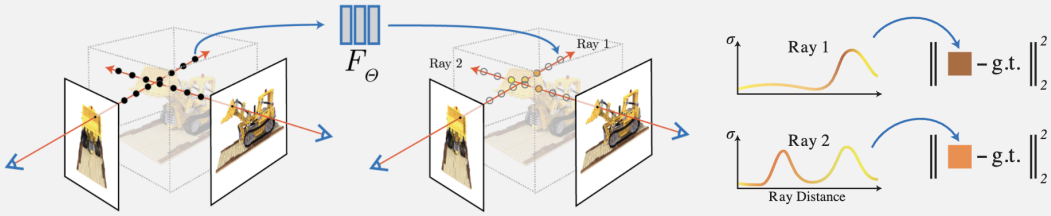
One can imagine a virtual camera as emitting rays from the lens, which hit geometry and capture local information like color and radiance. Each image pixel is thus the result of a ray hitting the scene. NeRFs query the coordinates along a camera ray to understand the scene geometry.

We can sample a set of sparsely clustered rays around a scene and back-propagate the photometric and depth comparison loss. Photometric loss captures the difference in pixel / ray color values via: $\mathcal{L}_{photometric} = ||C(I) - C(\hat{I})||_2^2$. I here is a camera observed $width \times height \times RGBD$ image-depth tensor while $\hat{I}$ is the image predicted by the network and the depth calculated from the network camera pose estimation. C(I) returns the color for the image. Depth loss is represented by the difference in the per-pixel distance from the camera between the ground truth and predicted images: $\mathcal{L}_{depth} = ||D(I) - D(\hat{I})||_2^2$, where D pulls the per-pixel depth from the RGBD data stored in I and $\hat{I}$.

We present a method that achieves state-of-the-art results for synthesizing novel views of complex scenes by optimizing an underlying continuous volumetric scene function using a sparse set of input views.



Our algorithm represents a scene using a fully-connected (non-convolutional) deep network, whose input is a single continuous 5D coordinate (spatial location (x, y, z) and viewing direction (θ, φ)) and whose output is the volume density and view-dependent emitted radiance at that spatial location.



We synthesize views by querying 5D coordinates along camera rays and use classic volume rendering techniques to project the output colors and densities into an image. Because volume rendering is naturally differentiable, the only input required to optimize our representation is a set of images with known camera poses. We describe how to effectively optimize neural radiance fields to render photorealistic novel views of scenes with complicated geometry and appearance, and demonstrate results that outperform prior work on neural rendering and view synthesis.

2.2 NeRF Extensions

While NeRFs were originally meant to map from pose to view to implicitly model a scene, recent works have expanded potential applications. A downside of the original NeRF paper was that in order to accurately model an object, one needed a lot of different views along with accurate camera poses. iNeRF[2] was the first to jointly optimize estimated camera pose alongside views. Bundle-adjusted neural radiance fields[3] (Barf) expanded by changing the optimization process along sampled rays to better estimate camera pose.

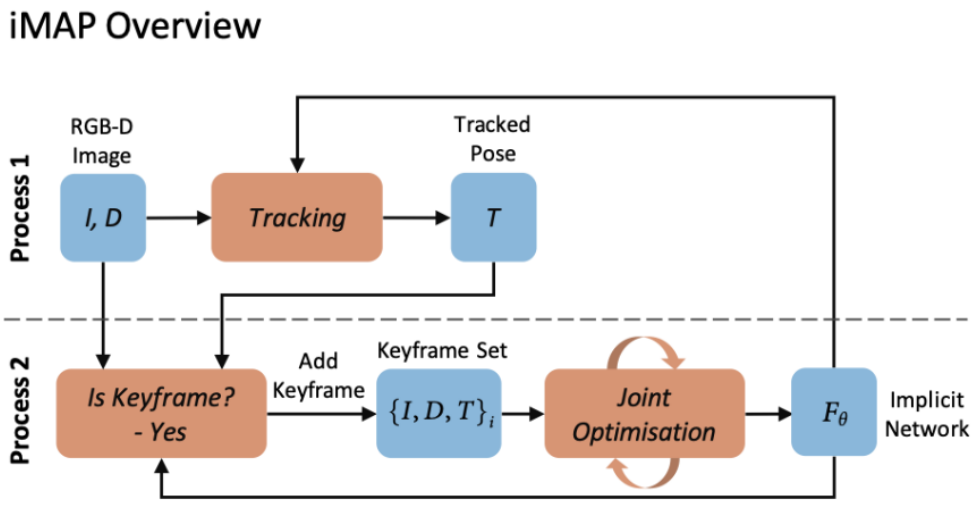
2.2.1 SLAM

Simultaneous Localization And Mapping (SLAM) represents a family of algorithms that, generally speaking, simultaneously maintain an estimate of the environment map and the robot's pose. The map is represented by a matrix of object ids and their parameterized pose information. The robot pose is similarly a set of coordinates, often (x, y, z) and some additions such as joint angles, roll, pitch, yaw, etc.

Early SLAM algorithms work by correlating the uncertainty in pose estimates of the robot and landmarks[4]. This means that when the robot can get a position fix upon a repeat observation of a landmark, it can reduce the uncertainty in all observed landmarks. This allows the robot to continually combat growing uncertainty as it expands its map and revisits more landmarks.

2.2.2 RGB-D SLAM

iMAP was the first to use neural implicit scene representations in SLAM. They used RGB-D images as input features and optimized the NeRF as a single large MLP to reduce mean-squared error of photometric and depth loss terms. The result was a dense scene representation and the ability for the robot to query its location via an image.[5]



However, using a single large MLP to represent the scene results in over smoothing and, for large scenes, tends to exhibit some amount of forgetting.

In order to remedy this, NICE-SLAM used hierarchical scene representation by using multiple MLPs to represent coarse and fine-grain scene details. Other works like NICER-SLAM[6] and NeRF SLAM[7] require only RGB inputs and just train on photometric loss (no need for depth data) to speed up the process and perform neural implicit SLAM in an online setting.

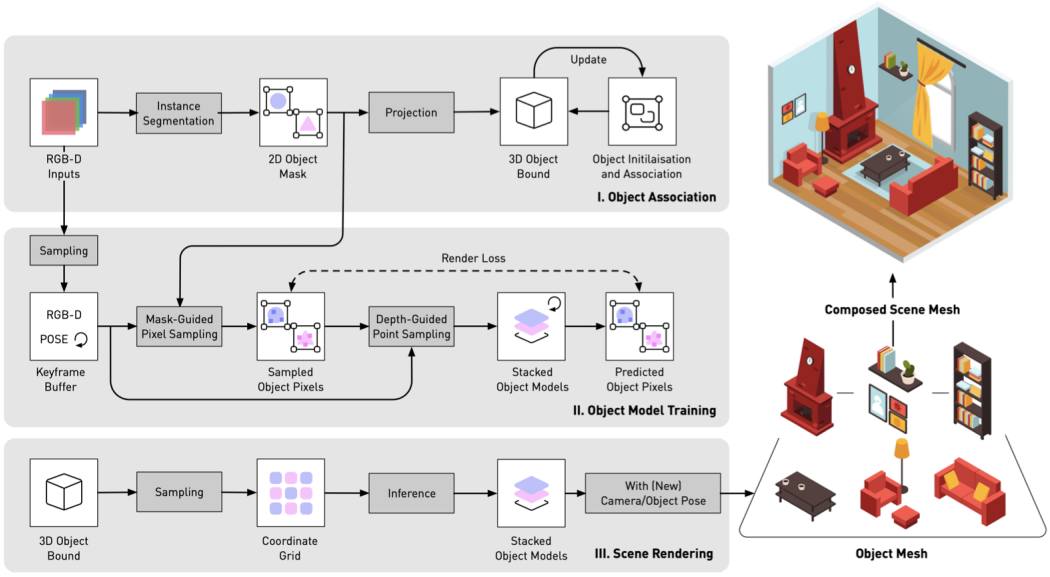
3 Methods

3.1 vMAP

This project is built on the vMAP mapping algorithm[8]. vMAP uses an off-the-shelf 2D image segmenter to find and classify objects in the scene. For an object to be considered a landmark it must a) be noticed by the segmenter, b) be in a frame whose estimated pose is spatially consistent with the object in the prior frames, c) be classified as the same semantic object class as was present in the prior frame. When a landmark is identified, it is added to the map and assigned a small, fully connected,

4-layer and 32-node hidden layer MLP and the keyframe index of the appearance is tracked. vMAP also uses a larger 4-layer 128-node hidden layer MLP to implicitly represent the entire scene.

By decomposing the scene into separate small NeRFs and tracking keyframe appearances, vMAP is able to vectorize the NeRF training for performance while also maintaining more detailed object-level representations.



3.2 Diet NeRF

NeRFs are very data hungry models because it is extremely difficult for a neural net to generalize and predict unknown views of complex geometric scenes. It often over smooths or has gaps the further from its training poses it gets. This is where Diet NeRF comes in[9]. Diet NeRF remedies this interpolation problem by starting with the base assumption that "a bulldozer looks like a bulldozer from any direction." It is hard for a NeRF to interpolate because it has little-to-no context for what the scene may look like from another view. But if the subjects of the view could be described semantically, for example "this is a bulldozer," and the network had some context for what bulldozers looked like, it could have better guidance when interpolating novel viewing angles.

Diet NeRF accomplishes this by introducing a semantic loss term into the NeRF training cycle, wherein the target and predicted images are rendered and then cast into a semantic embedding space. In doing so, one can compare the semantic similarity between the ground truth and the NeRF predictions via cosine similarity or the norm of the difference vector. The gradients then guide the network via semantic patters in the embedding space to create a more consistent scene.

We can thus extend the photometric and depth loss from earlier to include a semantic loss, where ϕ_s denotes the mapping operation over a flattened image vector of length $height \times width \times 3$ to the semantic embedding space.

$$\mathcal{L}_{semantic} = ||\phi_s(I) - \phi_s(\hat{I})||_{L1}$$

3.3 CLIP

Our choice of semantic embedding space comes from OpenAI's CLIP embedding space. CLIP is the result of training a variety of neural network and transformer architectures on over 400 million image and text pairs from the internet. The models in the CLIP Python module allow users to embed images and text in a shared image-text semantic embedding space. We chose to use their ViT-B/32 vision

transformer model as our ϕ_s in the semantic loss term above. This model uses a vision transformer architecture tuned to cast images to a 512-dimensional semantic embedding space.

3.4 Algorithm

Our algorithm can be summarized by the below algorithm statement. We chose CLIP's ViT-B/32 vision transformer architecture as our ϕ_s, $\hat{I}$ are our vMAP NeRF predictions, and I is the ground truth image from a given RGB-D dataset.

Data: Observed views $\mathcal{D} = \{(I, \mathbf{p})\}$, semantic embedding function $\phi(\cdot)$, pose distribution π, consistency interval K, weight λ, rendering size, batch size $|\mathcal{R}|$, lr η_{it}
Result: Trained Neural Radiance Field $f_\theta(\cdot, \cdot)$
Initialize NeRF $f_\theta(\cdot, \cdot)$;
Pre-compute target embeddings $\{\phi(I) : I \in \mathcal{D}\}$;
for *it from 1 to num_iters* **do**
  Sample ray batch $\mathcal{R}$, ground-truth colors $\mathbf{C}(\cdot)$;
  Render rays $\hat{\mathbf{C}}(\cdot)$ by (1);
  $\mathcal{L} \leftarrow \mathcal{L}_{\text{MSE}}(\mathcal{R}, \mathbf{C}, \hat{\mathbf{C}})$;
  if *it* $\% K = 0$ **then**
    Sample target image, pose $(I, \mathbf{p}) \sim \mathcal{D}$;
    Sample source pose $\hat{\mathbf{p}} \sim \pi$;
    Render image $\hat{I}$ from pose $\hat{\mathbf{p}}$;
    $\mathcal{L} \leftarrow \mathcal{L} + \mathcal{L}_{\text{SC}}(I, \hat{I})$;
  end
  Update parameters: $\theta \leftarrow Adam(\theta, \eta_{it}, \nabla_\theta \mathcal{L})$;
end

In practice, it is expensive to perform inference with a NeRF, so we only render and calculate semantic loss for samples every $K = 10$ input image frames. Semantic loss is only calculated over complete or masked images, never over ray samples. The semantic loss is weighted by another hyperparameter, λ, which for our experiments worked well at a value of 1.

3.5 Implementation

Furthermore, because of the expense of full renders, our photometric and depth loss are not computed using full renders from the scene. We keep track of the frames where a each landmark object appears and then use the image segmenter to create a mask. The mask is an image where all RGB pixel values are set to 0 (black) where the object is not present and 1 (white) when the object is. When a mask is multiplied by the corresponding image, it blacks out the image except for the pixels that are of the relevant object. Since the mask pixel values are 1 on relevant pixels, the original image's object pixel values remain unchanged after multiplication.

We apply the mask to our semantic image renders of specific images so we can capture an embedding that focuses on the landmark. No mask is applied before aquiring the semantic embedding of full scene renders because we do not want to occlude any of the image.

When training, we randomly sample a set, $\mathcal{R}$, of $|\mathcal{R}|$ = 20 * 120 pixels selected from a batch of $\mathcal{D}$ randomly chosen object-relevant keyframes. Each group of 120 pixels is composed into a 60 x 20 rectangular image that we then can get calculate loss terms from.

It is important to remember that the pixels are the result of ray information in the NeRF setting, wherein a ray is shot out of the virtual camera and returns the color and occupancy of regions of space along its path. The 3D point occupancy refers to probability, bounded between 0 and 1, that a 3D coordinate position sampled along the ray path is "occupied" by physical geometry. We can

encourage the network to represent regions where we would expect geometry to be occupied via an occupancy loss. In our ground truth image, occupancy can be represented as a binary variable, as the relevant object is either in the pixel or not. This is conveniently captured by the mask we calculated earlier, where the pixel has a value of 1 if the object is in it and 0 if it is not.

If we consider the composed image from which we pull our photometric, depth, and occupancy loss to be made up of $|\mathcal{R}|$ rays, we can write out our losses as follows. C and D map from an input ground truth ray to the color and depth, respectively, of said ray. $\hat{C}$ and $\hat{D}$ are the depth and color of the corresponding ray as predicted by the NeRF. We introduce O and $\hat{O}$ as analogous mapping functions from an input ray to the occupancy of that ray along its path. Lastly, M^I represents the mask being applied to composed ray image I. In practice, we perform this for every object by substituting the object mask below and then averaging all the results.

$$\mathcal{L}^I_{photometric} = \frac{1}{|\mathcal{R}|} M^I * \sum_{r \in \mathcal{R}^I} |C(r) - \hat{C}(r)|$$
$$\mathcal{L}^I_{depth} = \frac{1}{|\mathcal{R}|} M^I * \sum_{r \in \mathcal{R}^I} |D(r) - \hat{D}(r)|$$
$$\mathcal{L}^I_{occupancy} = \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}^I} |M^I * O(r) - \hat{O}(r)|$$

So our total loss function for an input image can be represented by:

$$\mathcal{L}^I = \lambda_1 \mathcal{L}^I_{photo} + \lambda_2 \mathcal{L}^I_{depth} + \lambda_3 \mathcal{L}^I_{occ} + \lambda_4 \mathcal{L}^I_{sem}$$

We used loss weights $\lambda_1 = 5$, $\lambda_2 = 1$, $\lambda_3 = 10$, and, after experimentation, landed on $\lambda_4 = 1$.

4 Results

4.1 Scene Comparisons

We evaluated on a portion of the Replica dataset. With more time, we would also evaluate on many real-world datasets.

We experimented with semantic loss weights λ_4 = 1 and 10 and compared scene renders to those output by a standard vMAP implementation. All shared hyperparameters were held constant between the three models. The top left is a table rendered by the vMAP scene NeRF, top right is our semantic scene NeRF with a weight of 1, and the bottom is ours with a semantic loss weight of λ_4 = 10.

As can be seen in the first vMAP image, the right side of the table is well observed in the data but the left is novel. The vMAP representation fails to reconstruct the left, and is filled with odd cavities and deformities. Our semantically guided reconstruction completes the table geometry so it looks like the same desk from various angles.

The next set of results below are arranged in the same way, with vMAP top left, semantic weight of 1 in the top right, and the semantic weight of 10 on the bottom. This shows another angle of the room

that was rich in photo data. A note that the orange tint on the bottom image is the result of a render error and was not from the NeRF.

Here, we see that semantically guided NeRFs do not offer much advantage over classic NeRFs when synthesizing views of data rich regions of the environment. The interpolation issue for classic NeRFs is bypassed entirely.

On the flip side, below shows the other side of the room. This was a data poor region with a small number of frames. Because each training step randomly samples frames, it is difficult to conclusively state whether the large quality difference between the classic and semantically guided NeRF representations are indicative of a superior methodology or an unlucky optimization for the NeRF. That said, this observed reconstruction gulf merits more experimentation to tease out the true comparative strength of semantically guided scene mapping.

Despite the apparent inference strength semantically driven NeRF demonstrated over the back right corner, it hallucinated strange ungulating left wall segments that grew more pronounced with a higher semantic weight term. This is likely a strange interaction where adding geometry in this strange optimization region allowed semantic NeRF to greedily reduce loss while harming the image

cohesion. This reinforces the fact that semantic guidance must be balanced and a lack of data can still harm representations. That said, we see promising results in reconstruction of regions where we are lacking sufficient observation data.

5 Conclusion

We have shown that incorporating semantic guidance to NeRF construction can potentially improve scene inference over data poor regions of an environment. But, if not tuned properly, the semantic guidance can enter strange greedy feedback loops that hallucinate dramatic, and nonexistent, geometry.

A major drawback of the method presented is the computational cost involved with training the NeRFs and inferring on large samples of rays. We ran training over 1,000 images on 4 NVidia a40 GPUs, and training often took upwards of 20 minutes. In the future, integrating hierarchical representation like NICE-SLAM[10] and leveraging novel render techniques such as those showcased in Mobile NeRF[11] could enable this mapping to be done in an online SLAM setting.

6 References

All project code can be found at: https://github.com/Douglasmsw/DAD_SLAM

All summary diagrams are pulled from the corresponding method paper.

[1] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, Ren Ng (2020) "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis". European Conference on Computer Vision

[2] Lin Yen-Chen, Pete Florence, Jonathan T. Barron, Alberto Rodriguez, Phillip Isola, Tsung-Yi Lin (2021) "iNeRF: Inverting Neural Radiance Fields for Pose Estimation". International Conference on Intelligent Robots and Systems

[3] Lin, Chen-Hsuan and Ma, Wei-Chiu and Torralba, Antonio and Lucey, Simon (2021) "BARF: Bundle-Adjusting Neural Radiance Fields". International Conference on Computer Vision

[4] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," (2006). IEEE Robotics Automation Magazine, vol. 13, no. 2, pp. 99-110, doi: 10.1109/MRA.2006.1638022

[5] Edgar Sucar, Shikun Liu, Joseph Ortiz, Andrew Davison (2021) "iMAP: Implicit Mapping and Positioning in Real-Time". International Conference on Computer Vision

[6] Zhu, Zihan; Peng, Songyou; Larsson, Viktor; Cui, Zhaopeng; Oswald, Martin R. (2023) "NICER-SLAM: Neural Implicit Scene Encoding for RGB SLAM". preprint arXiv:2302.03594

[7] Rosinol, Antoni and Leonard, John J and Carlone, Luca (2022) "NeRF-SLAM: Real-Time Dense Monocular SLAM with Neural Radiance Fields". preprint arXiv:2210.13641

[8] Kong, Xin and Liu, Shikun and Taher, Marwan and Davison, Andrew J. (2023) "vMAP: Vectorised Object Mapping for Neural Field SLAM". preprint arXiv:2302.01838

[9] Jain, Ajay and Tancik, Matthew and Abbeel, Pieter (2021) "Putting NeRF on a Diet: Semantically Consistent Few-Shot View Synthesis". International Conference on Computer Vision

[10] Zhu, Zihan and Peng, Songyou and Larsson, Viktor and Xu, Weiwei and Bao, Hujun and Cui, Zhaopeng and Oswald, Martin R. and Pollefeys (2022) "NICE-SLAM: Neural Implicit Scalable Encoding for SLAM". Conference on Computer Vision and Pattern Recognition

[11] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, Andrea Tagliasacchi (2023) "MobileNeRF: Exploiting the Polygon Rasterization Pipeline for Efficient Neural Field Rendering on Mobile Architectures". Conference on Computer Vision and Pattern Recognition