Research Article

LiLMaps: Learnable Implicit Language Maps

Evgenii Kruzhkov[1], Sven Behnke[2,3]

1. Autonomous Intelligent Systems, Computer Science Institute VI, University of Bonn, Bonn, Germany; 2. Autonomous Intelligent Systems, Computer Science Institute VI – Intelligent Systems and Robotics, University of Bonn, Bonn, Germany; 3. Center for Robotics and the Lamarr Institute for Machine Learning and Artificial Intelligence

One of the current trends in robotics is to employ large language models (LLMs) to provide non-predefined command execution and natural human-robot interaction. It is useful to have an environment map together with its language representation, which can be further utilized by LLMs. Such a comprehensive scene representation enables numerous ways of interaction with the map for autonomously operating robots. In this work, we present an approach that enhances incremental implicit mapping through the integration of vision-language features. Specifically, we (i) propose a decoder optimization technique for implicit language maps which can be used when new objects appear on the scene, and (ii) address the problem of inconsistent vision-language predictions between different viewing positions. Our experiments demonstrate the effectiveness of LiLMaps and solid improvements in performance.

1. Introduction

Classic robotic maps are commonly used for estimating distances to obstacles and costs of motions in navigation and localization tasks. However, more comprehensive tasks, as well as natural human-robot interaction, may require a deeper understanding of the environment, and thus imply more advanced map representations. For example, visual-language navigation is the task where a robot must interpret a natural language command from a non-expert user and proceed towards the goal according to the command. The environment might be unknown in advance, but the robot still must navigate in the shortest possible time. From this example, it is clear that a map that allows one to easily find correlation between the given language command and a partially or fully mapped environment can be more useful than a pure obstacle costmap.

In this work, we make a step towards creation of efficient yet compact natural language environment representations and introduce Learnable implicit Language Maps (LiLMaps). We chose an implicit representation because of its ability to compactly represent the data and for the possibility of further detailed reconstruction.

Recent studies in implicit mapping demonstrate outstanding results in geometry reconstruction. While in these studies, geometry decoders can be easily pre-trained or even trained in the first few iterations, in our work we

 doi.org/10.32388/D8Q0Y8

demonstrate that LiLMaps performs better compared to the pre-trained language decoders because some language features can be poorly represented in them. Moreover, pre-training the decoder to represent every possible language feature could make its structure and training process significantly more complicated, and it can reduce its flexibility for different applications.

Another challenging problem that frequently appears in incremental language learning is the inconsistency of measurements taken from different viewing positions. Precise range sensors, such as LiDARs and RGB-D cameras, are commonly used in implicit mapping, but usually they do not provide contradictory measurements. However, in visual-language navigation tasks, information about the environment is often derived from RGB images. Vision-language features extracted from RGB images may have many sources of inconsistency: a painting can be recognized as a wall from a greater distance; a bed object can be misclassified as a sofa at different angles of view; objects on image borders and occluded objects might not be visible enough to provide correct features; inaccurate detection on the object edge can spoil features of the objects behind them; etc.

LiLMaps focuses on incremental implicit language mapping, i.e., when new observations become available incrementally, one-by-one. This is a typical condition for SLAM, and LiLMaps can be integrated into existing implicit SLAM approaches with minimal changes. We achieve this with the following key techniques that are presented in this work:

- **Adaptive Language Decoder Optimization** dynamically updates the decoder to new discovered language features in the environment providing flexible and sufficient coverage of language representations.
- **Measurements Update Strategy** adjusts incoming measurements reusing accumulated and implicitly stored knowledge about the environment to reduce measurements inconsistency.

Our experiments show that LiLMaps enables incremental vision-language environment exploration with just a small overhead.

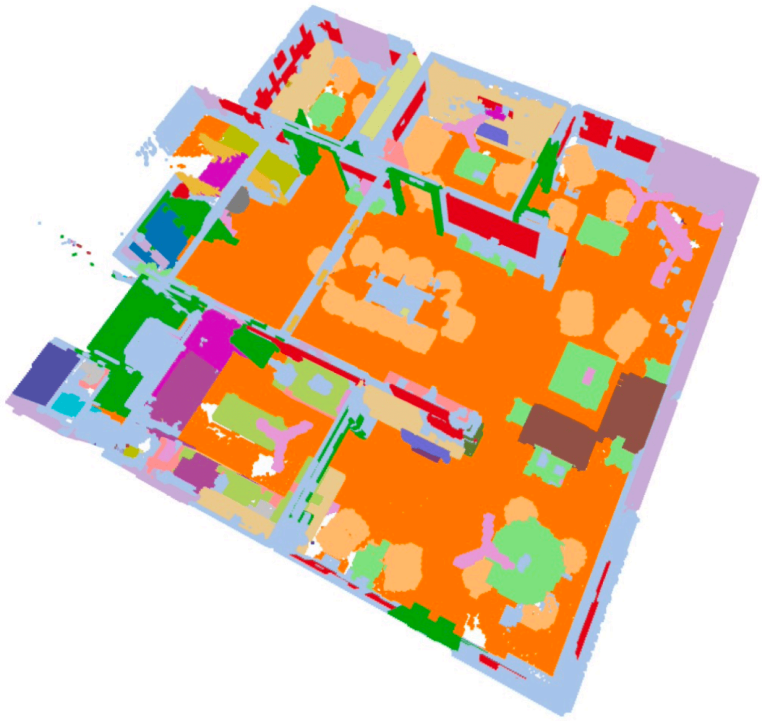

Figure 1. Reconstructed implicit language map built with LiLMaps. Semantic colors are assigned based on the similarity of reconstructed language features and CLIP[1] encodings of semantic categories from the Matterport3D dataset[2].

2. Related Works

Language and Vision-Language Models

Dynamic execution of natural language commands has been an active research topic for a long time[3][4]. Previous works often propose custom environment representations that are difficult to reuse in real applications. Recently, large language models (LLMs) have received increased attention in this field. The advantage of LLMs is their ability to be applied to a wide range of tasks. LLMs can enhance robot abilities to understand and execute natural language commands[5][6]. In addition, LLMs have been shown to be successful in guiding object grasping[7][8], navigation[9][10][11][12][13] and scene understanding[14][15][16]. Vision-Language Models (VLMs)[17][18][19][20], which extract language information about the environment from the provided images, are frequently used in conjunction with LLMs. For instance, CLIP[1] is one of the most widely used models capable of mapping images into a natural language space.

Often, VLMs transfer only single or batched images into a natural language vector space. However, some tasks may benefit from mapping the entire environment into the language space. For example, VLMaps[9] suggests enhancing 2D maps with language features and then demonstrates that navigation and detection tasks can be solved directly on these enhanced maps. Although VLMaps can be used alongside simultaneous localization and mapping (SLAM), its performance depends on localization and mapping quality, and the method is limited to building only 2D language maps. On the other hand, OpenScene[15] can generate maps as 3D point clouds with corresponding language features. However, the method performs best in batch-like operations, where all RGB images of the environment, their poses, and 3D point clouds are available in advance, meaning no real-time data is processed. Another approach, ConceptFusion[21], demonstrates that language features can be fused into 3D maps using traditional SLAM approaches. SAM3D[22] projects SAM segmentation masks[23] into 3D and creates 3D scene masks.

Implicit Representations

Implicit representations[24][25][26][27][28] have gained popularity for their compactness and ability to achieve high-resolution reconstructions. They demonstrate great capabilities in environment mapping. iMap[29] uses an RGB-D sensor to perform a real-time SLAM task. Nice-SLAM[30] extended the possible sizes of the mapped environment. SHINE-Mapping[31] demonstrated implicit mapping of outdoor environments. Recently, works based on Gaussian splatting[32] demonstrated exceptional results[28][33][34][35]. In addition to geometry, implicit maps demonstrate a successful reconstruction of semantic information[36], physical properties[37], and visual features[38].

Integration of language features into implicit representation is an actively researched topic. LERF[39] studies the fusion of language features into Radiance Fields, but is limited to small scenes. LangSplat[40] uses Gaussian Splatting[32] to achieve higher precision and training speed. However, LangSplat demonstrates only the reconstruction of small table-sized scenes and does not consider incremental mapping.

Implicit representations are highly dependent on the type of encoding they use. The encodings employed in the original NeRF work[24] were able to generalize the predictions[37][38], but were constrained by the limited size of the environments. Subsequent works successfully increased training and reconstruction speeds[25][41]. Gaussian Splatting[32] is currently one of the most popular encodings due to its speed, simplicity, and high quality reconstruction. However, some works may benefit from structured encodings such as grid-based[42], octree-based feature volumes[43], or combined ones[44].

Compared to the works discussed above, our approach can build large-scale 3D implicit language maps and can be seamlessly integrated with implicit SLAM methods. LiLMaps use a sparse octree-based representation[43] to

store learnable features, but our method is not tied to any particular representation and can be adapted to others with minimal effort.

3. Method

We address the task of building an implicit vision-language representation along with environment mapping. Sec. 3.1 describes our model architecture used in LiLMaps. During incremental mapping, the future observed objects and their encoded representations are unknown in advance, which makes it challenging to train a language decoder to represent all language features. To address this issue, in Sec. 3.2 we propose the adaptive language decoder optimization strategy that can effectively adjust the decoder to new language features while retaining previously observed ones. In this work, we employ a visual language encoder but pixel-wise language features are often inconsistent between frames. We address this issue in Sec. 3.3.

3.1. LiLMaps Architecture

Figure 2 shows the architecture of the proposed approach. Input data for our pipeline are point clouds associated with CLIP language features (language point clouds), as well as camera poses estimated by any external SLAM method. We produce language point clouds by extracting language features from an RGB image. Extracted language features are projected to the point clouds in the world coordinate system using the corresponding depth image and camera pose. The extraction of language features can be done using per-pixel visual language encoders such as LSeg, OpenSeg, Segment-Anything-CLIP[20], etc. For example, VLMaps[9] utilizes LSeg for this purpose. In this study, the visual language encoder is treated as an external module, and improving its performance is not our focus.

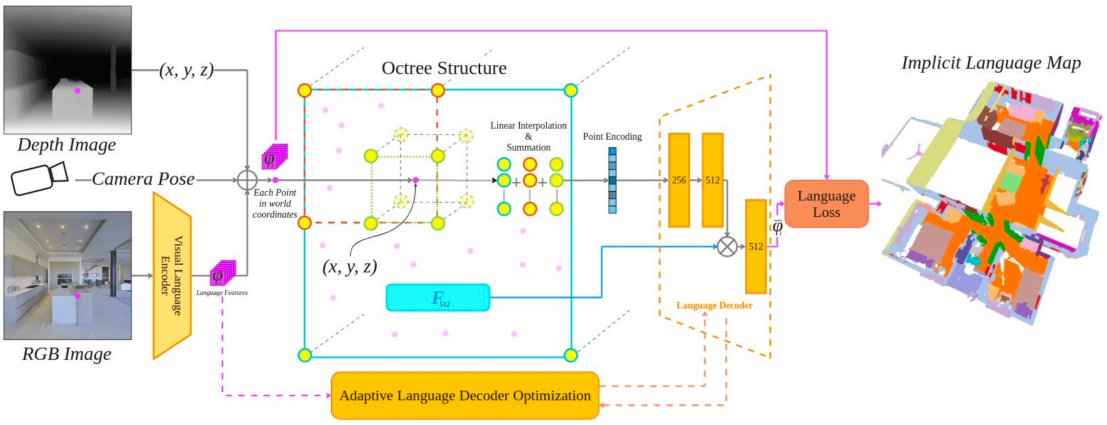

Figure 2. Implicit language mapping. Vision-language features φ are extracted from the RGB image. The corresponding points of the depth image are projected to the world coordinate system. Each point can be encoded using its coordinates and octree: the coordinates are used to find the corresponding octree voxels (blue, red, green); learnable features stored in the voxels' corners are interpolated and summed, producing the point encoding. F vectors are stored only in the voxels of the coarse octree level (blue). The language decoder reconstructs the language feature $\overline{\varphi}$ in the spatial coordinates of the point based on its encoding and the vector F. Language loss optimizes the learnable features and F vectors. After optimization, the language map can be reconstructed in arbitrary spatial coordinates. The language detector is optimized independently of the implicit mapping (Sec. 3.2).

Our goal is to enable implicit vision-language mapping under the conditions of environment exploration when future measurements are not available. We use the octree structure as positional encoding to build the implicit representation. Unless otherwise specified, we consistently use three different levels of the octree to store the features. Each level of the octree is made up of voxels, and each voxel from a higher level can encompass multiple voxels from lower, more detailed levels. It should be noted that we use a sparse octree representation, meaning voxels are only present where observations have been made. When the point clouds are projected to the world coordinates, we find the corresponding voxels in the octree for each point. Each voxel holds learnable features at its corners. These features are shared among voxels that have common corners.

The language features have a high dimensionality and a straightforward solution to encode them in the octree is to increase the size of the learnable features stored in the corners. However, storing high-dimensional learnable features consumes a significant amount of memory. Instead, we suggest storing one high-dimensional learnable feature vector F per voxel of the first (coarse) octree level, while keeping the corner features low-dimensional.

To train the implicit representation, we apply the cosine similarity loss between the features decoded from the octree $\overline{\varphi}$ and the vision-language features φ of the input point cloud:

doi.org/10.32388/D8Q0Y8

$$\mathcal{L}_{vl} = -\frac{1}{N}\sum_{i=1}^{N}\text{CosineSimilarity}(\varphi_i, \overline{\varphi}_i), \tag{1}$$

where N is the number of points with vision-language features in the point cloud.

Note that we encode every new available measurement into the learnable features using (1), but the weights of the language decoder are not updated with this loss. The optimization of the decoder is described in Sec. 3.2.

The decoder reconstructs the language feature $\overline{\varphi}$ in spatial coordinates using feature vector F and corners features of the corresponding voxels. Before feeding to the decoder, the corner features are linearly interpolated into the reconstruction point and summed across all octree levels. The decoder consists of three fully connected layers. The first two layers expand the dimensions of the corner features and produce the element-wise scaling vector for F, which is then multiplied by it and passed to the last fully connected layer, which outputs the predicted language feature $\overline{\varphi}$.

3.2. Adaptive Language Decoder Optimization

Equation (1) uses our MLP-based language decoder to predict vision-language features $\overline{\varphi}$ based on the encodings stored in the octree. However, the new data can contain features that have not been observed before. In this case, the decoder weights must be updated to be able to reconstruct new features without forgetting the old ones, but re-training of the whole previously mapped environment is computationally expensive.

We propose adaptive language decoder optimization in Algorithm 1. When a new point cloud with vision-language features arrives, we perform decoder optimization before the optimization of octree features described in Sec. 3.1.

Algorithm 1 Adaptive Language Decoder Optimization

```
1: global names
2:     LDec                                   ▷ Language decoder
3:     kFeatures ← ∅                          ▷ Known features
4:     Encodings ← ∅                          ▷ Learnable Parameter
5:     FVectors ← ∅                           ▷ Learnable Parameter
6:     τ                                      ▷ Cosine similarity threshold
7:     LOSS                                   ▷ Cosine similarity loss
8: end global names
9: procedure OPTIMIZE(inFeatures)
10:    uniqueFeatures ← UNIQUE(inFeatures, τ)
11:    newFeatures ← UNKNOWN(uniqueFeatures,
                             kFeatures, τ)
12:    if len(newFeatures) = 0 then
13:        return      ▷ No optimization if no new features
14:    end if
15:    encodings ← RANDOM(len(newFeatures), m)
16:    if len(FVectors) = 0 then
17:        fvectors ← RANDOM(1, L)
18:    else
19:        fvectors ← MEAN(FVectors, dim = 0,
                           keepdim = true)
20:    end if
21:    optimizer ← ADAM([encodings, fvectors, LDec])
22:    allEncodings ← Encodings ∪ encodings
23:    allFVectors ← FVectors ∪ fvectors
24:    allFeatures ← kFeatures ∪ newFeatures
25:    for i ← 0 to N_opt do
26:        φ̄ ← LDec(allEncodings, allFVectors)
27:        fvectors1 ← SHUFFLE(allFVectors)
28:        loss ← LOSS(allFeatures, φ̄)
29:        loss_F ← LOSS(fvectors1, allFVectors)
30:        optimizer.optimize(loss + loss_F)
31:    end for
32:    kFeatures ← allFeatures              ▷ Update features
33:    Encodings ← allEncodings             ▷ Update encodings
34:    FVectors ← allFVectors               ▷ Update FVectors
35: end procedure
```

The proposed optimization operates with a language decoder (Line 2) and learnable parameters. The learnable parameters are the inputs that are directly forwarded to the decoder. In our work (Figure 2), the learnable parameters are F vectors (Line 5) and the interpolated and summed point encoding (Line 4). Note that Algorithm 1 is not limited to our network architecture and can be adapted to other implicit representations by simply replacing the corresponding learnable parameters.

Firstly, we extract only unique language features (Line 10) from all available ones in the input point cloud and then filter out already known features (Line 11). In both cases, cosine similarity and a predefined threshold τ are used to estimate similarity.

For the new features (unobserved features without duplicates), we initialize the learnable parameters: the encodings and F vectors (Lines 15–19). Note that the initialized encodings (Line 15) correspond to the linearly interpolated and summed features of the corners of the octree (Point Encoding in Figure 2). We initialize only a single F feature vector for all new language features (Line 17) because the F feature vector stored in the coarse octree level may be used for points with different language features (Sec. 3.1). We also regularize new F vectors by enforcing them to be similar to existing ones (Lines 27 and 29). However, this regularization is optional and can be omitted or changed to any other regularization required by the corresponding implicit representation.

The proposed adaptive optimization approach optimizes the decoder for unobserved language features and finds the learnable parameters for them. Only the decoder and new learnable parameters are optimized (Line 21). The already known features and their encodings are not optimized, but used to prevent forgetting (Lines 22 to 24). After optimization, we update the list of known vision-language features and their learnable parameters (Lines 32 to 34).

The proposed optimization efficiently stores only a small number of known features for replay (Line 3), as demonstrated in experiments (Figure 6). It enables fast decoder optimization using vectorization. Moreover, the language decoder and the learnable parameters are optimized only if new language features are observed.

3.3. Measurement Update Strategy

During incremental mapping, new observations added to the map should not corrupt previous measurements. However, vision-language features predicted by the visual encoder may not be consistent between frames. VLMaps[9] averaged the language features of the objects received from different views. We note that the averaging can be done in a recursive form:

$$\mathcal{A}_n = \frac{1}{N}\sum_{i=1}^{N}\varphi_i = \frac{n-1}{n}\mathcal{A}_{n-1} + \frac{\varphi_n}{n}, \text{with } \mathcal{A}_0 = 0. \tag{2}$$

In this work, we propose to use as target for training in Eq. (1) a weighted average φ_n^* between observations φ_n and the features $\bar{\varphi}_{n-1}$ already stored in the map:

$$\varphi_n^* = \frac{n-1}{n}\bar{\varphi}_{n-1} + \frac{\varphi_n}{n}, \text{with } \bar{\varphi}_0 = 0. \tag{3}$$

This averaging is especially useful for noisy measurements such as vision-language features because they may significantly vary with distance to the objects or point of view (Sec. 4.1). Mapping with Eq. (3) forces the map to store all observations similarly to[9]. However, we observed better results when not all previous data are stored and decided to use exponential smoothing instead of averaging:

$$\varphi_n^* = \alpha\bar{\varphi}_{n-1} + (1-\alpha)\varphi_n \text{with-} \bar{\varphi}_0 = 0, \tag{4}$$

where α is set dynamically to higher values if new measurements φ_n are more different from the previously optimized map features $\bar{\varphi}_{n-1}$ and lower otherwise:

$$\alpha = \frac{\text{CosineSimilarity}(\varphi_i, \bar{\varphi}_i)}{0.5 + \text{CosineSimilarity}(\varphi_i, \bar{\varphi}_i)}. \tag{5}$$

4. Experiments

In the experiments, we validate that our method can be used for incremental implicit mapping of language features. We use depth, semantic, and RGB images provided by[9] through the Habitat simulator using Matterport3D[2]. Matterport3D provides ground truth meshes with each face assigned to a class label. To get ground truth point clouds with the corresponding language features, we sample the meshes and encode their labels using CLIP[1]. During all experiments, we project depth images into 3D using the current camera pose to obtain input point clouds. The parameters we use during the experiments are summarized in Table 1.

Parameter	Symbol	Value
Similarity threshold	τ	0.02
Used octree levels		8,9,10
Fine level resolution		0.05 [m]
Learnable features size	m	16
F vectors size	L	512
Iterations per decoder optimization	N_{opt}	100
Iterations per mapping loss Eq. (1)		100

Table 1. LiLMaps parameters and their values in the experiments.

doi.org/10.32388/D8Q0Y8

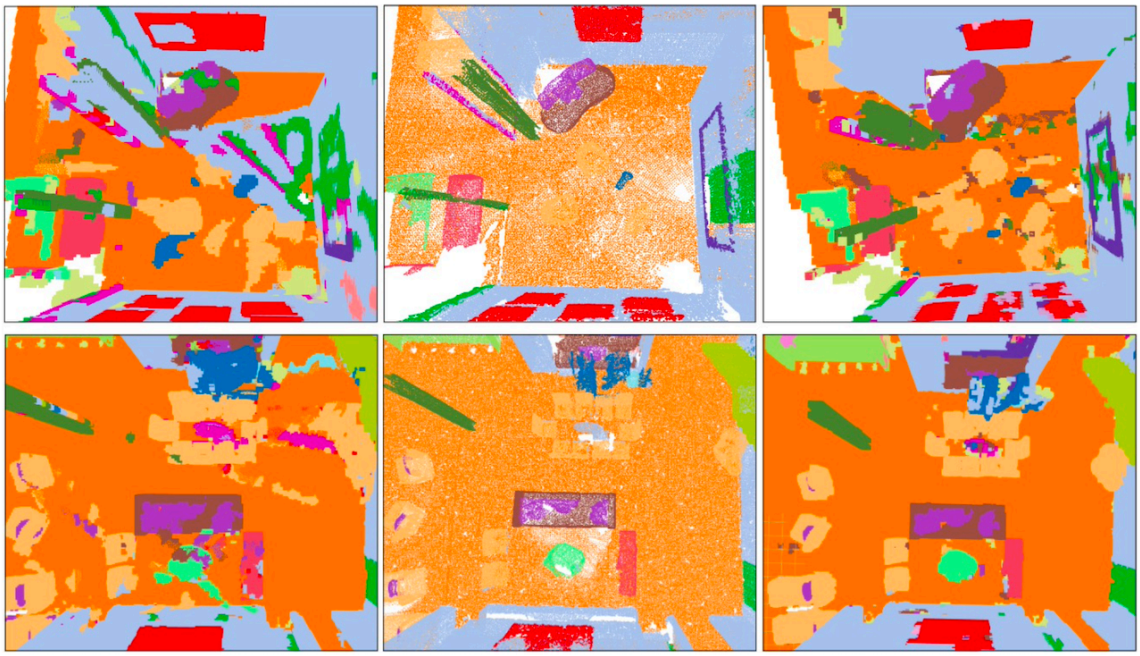

Figure 3. *Left:* Environments reconstructed without measurement update; *Middle:* Ground Truth; *Right:* Environments reconstructed with measurement update.

4.1. Mapping Quality

We evaluate accuracy, recall, precision, and intersection over union for our implicit language map. Accuracy is defined as the number of points with correctly reconstructed language features divided by the total number of points. The values are compared with the OpenScene 3D model[15] trained on Matterport3D[2]. The results of random sequences are presented in Table 2. To validate the measurement update strategy, we also present results with deactivated measurement update, marked by an asterisk (LiLMaps*).

doi.org/10.32388/D8Q0Y8

	5LpN3gDmAk7_1				YmJkqBEsHnH_1				gTV8FGcVJC9_1				jh4fc5c5qoQ_1				JmbYfDe2QKZ_2			
	A	**mR**	**mP**	**mIoU**	**A**	**mR**	**mP**	**mIoU**	**A**	**mR**	**mP**	**mIoU**	**A**	**mR**	**mP**	**mIoU**	**A**	**mR**	**mP**	**mIoU**
LiLMaps*$_{GT}$	97	97	96	93	98	93	92	89	98	98	95	94	98	96	88	85	95	96	92	89
LiLMaps$_{GT}$	97	97	93	91	96	94	96	90	97	98	93	92	98	98	92	89	95	96	90	87
LiLMaps*$_{SEM}$	84	77	70	57	84	77	82	65	85	85	84	73	83	78	73	61	78	77	77	63
LiLMaps$_{SEM}$	**88**	**86**	**75**	**66**	**90**	**82**	**87**	**72**	**90**	**90**	**86**	**79**	**90**	**84**	**82**	**71**	**85**	**88**	**82**	**74**
OpenScene[15]	68	45	67	36	63	50	77	41	61	49	60	36	77	52	59	39	56	51	67	41
LiLMaps*$_{LSeg}$	64	29	46	21	52	44	58	31	65	48	59	32	59	32	44	21	53	41	50	33
LiLMaps$_{LSeg}$	68	37	57	26	57	56	60	34	70	50	63	33	73	39	58	27	56	42	56	34
VLMaps	28	-	-	19	28	-	-	19	28	-	-	19	28	-	-	19	28	-	-	19

Table 2. Language mapping quality evaluation: accuracy (A), recall (mR), precision (mP) and mean IoU (mIoU) in [%].

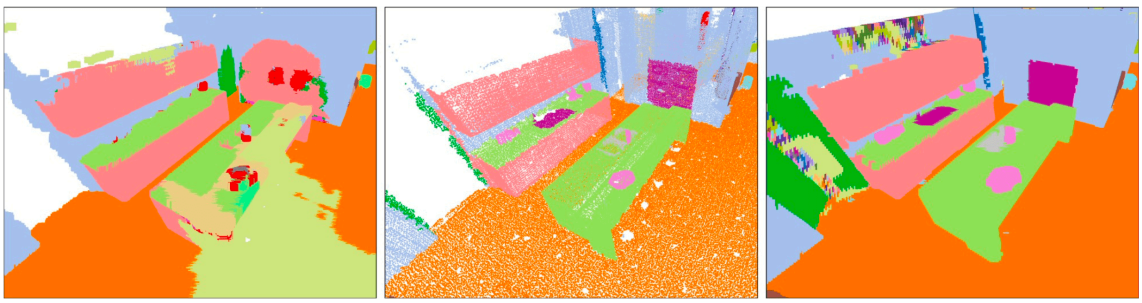

Figure 4. *Left:* Language map produced by OpenScene 3D[15]; *Middle:* Ground Truth; *Right:* Language map created by LiLMaps.

For LiLMaps$_{GT}$ and LiLMaps*$_{GT}$, the language features of a measurement are obtained directly from the closest points of the ground truth point cloud. This allows us to estimate upper-bound performance with close-to-ideal input data. However, simulation measurements and ground truth points sampled from uneven meshes do not always match perfectly. As a result, some input points may have wrong language features or do not have language features at all.

LiLMaps$_{SEM}$ and LiLMaps*$_{SEM}$ denote experiments in which language features are extracted from semantic images. Simulated semantic images have incorrect labels due to mesh discontinuities and on object edges. This

allows us to test LiLMaps when the input data are closer to the real ones, e.g. when the data are imprecise and inconsistent between frames.

Our approach demonstrates the best performance when used with the GT data. The introduction of the Measurements Update does not change the performance significantly, as the input features are precise and consistent between frames in this case.

Figure 5. Language map incrementally created with our adaptive optimization. *Bottom Left:* A region mapped in the beginning. *Bottom Right:* The same region after the mapping is completed. All initially mapped objects remain unchanged.

As expected, LiLMaps$_{\text{SEM}}$ and LiLMaps*$_{\text{SEM}}$ yield worse results due to the inconsistency of the input data, but both still outperform OpenScene[15]. LiLMaps$_{\text{SEM}}$ outperforms LiLMaps*$_{\text{SEM}}$ due to the proposed measurement update technique which addresses potential data inconsistencies. Figure 3 shows maps learned with activated

doi.org/10.32388/D8Q0Y8

and deactivated measurement update procedure. Enabling measurements update results in a cleaner final map, which is crucial for object detection and navigation.

Figure 4 compares a LiLMaps reconstruction with the prediction of the OpenScene 3D model. Despite OpenScene being trained on the same dataset, it struggles with certain labels and completely misses labels such as "TV monitor", "appliances", "stool", whereas our approach achieves high accuracy for these labels and does not completely miss objects.

Table 2 compares our approach combined with the LSeg model (LiLMaps$_{\text{LSeg}}$ and LiLMaps*$_{\text{LSeg}}$) and VLMaps (VLMaps) with metrics reported in[9]. LSeg frequently misses objects (e.g., segments a painting as a wall) or provides wrong language features (e.g., detects a bed as a sofa), which significantly influence the final results. Improving the quality of per-pixel language segmentation is beyond the scope of this research, however. In all cases, our 3D reconstructed language maps yield better results than the mean results reported in VLMaps[9] for their 2D maps.

4.2. Adaptive Language Decoder Optimization

We demonstrate the impact of our Adaptive Optimization Strategy on sequence 5LpN3gDmAk7_1 with GT labels. We compare the performance of the decoder trained with our Adaptive Optimization with other decoders built from pre-trained models. We chose OpenScene[15] 3D model's head as the pre-trained decoder because it can predict language features for arbitrary input point clouds. For fair comparison, we changed our language decoder architecture (LiLMaps$^{\text{simple}}$) to match the architecture of OpenScene's head. This head does not allow us to use the learnable vectors F, however, and therefore the results in Table 3 are presented when only the learnable corner features with $m = 96$ are optimized.

Decoders	**F1-Score**				
	↑100% – 90%	**↓90% – 80%**	**↓↓80% – 70%**	**↓↓↓70% – 50%**	**↓↓↓↓50% – 0%**
LiLMaps	20	2	1 (picture)	0	0
LiLMaps	20	2	0	1 (towel)	0
OpenScene	16	4	2	0	1 (objects)
OpenScene	16	5	1	0	1 (objects)
OpenScene	9	8	1	3	2 (picture, shelving)

Table 3. Number of classes falling into different F1-score ranges for compared decoders. A method is considered to be better if a larger number of classes are listed in the first column with $\mathrm{F1} > 90\%$.

Table 3 summarizes number of classes with distinct F1-score qualities using different optimization types: decoder trained with the proposed adaptive optimization (LiLMaps^{simple}); decoder pre-trained with the proposed optimization (LiLMaps^{pretrained}); and pre-trained and fixed headings of OpenScene^{MIT}[15], trained on different datasets (Matterport3D[2], nuScenes[45], ScanNet[46]).

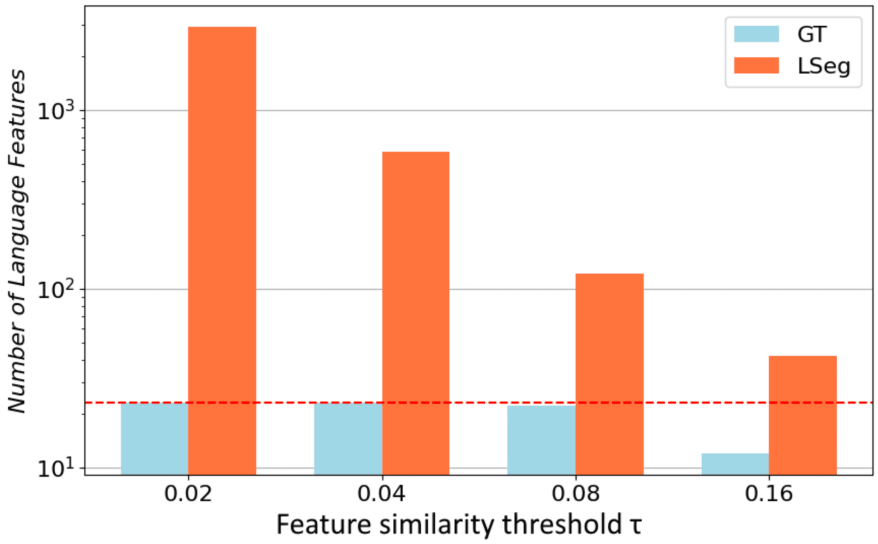

Figure 6. Number of language features stored for the adaptive optimization with varying feature similarity thresholds τ. Blue: Language features are extracted from ground truth (GT) data; Orange: Language features are extracted by LSeg; Red line: Total number of different GT classes presented in the scene.

To obtain LiLMaps^{pretrained} we extract all available labels from the scene, convert them to language features using CLIP and use our adaptive language decoder optimization with all features at once. The additional possibility of pre-training the decoder in advance without any real measurements may be useful for some applications. Our adaptive language decoder optimization allows to adjust pre-trained decoders online if necessary, but for this experiment, we do not update pre-trained models (LiLMaps^{simple}, OpenScene^{MIT}, OpenScene^{SC}, OpenScene^{MS}) during the mapping.

The final results of LiLMaps^{simple} are similar to those of LiLMaps^{pretrained} because every time a new object is observed, the corresponding language features are included in the Adaptive Optimization and the decoder of LiLMaps^{simple} is updated to represent new features without forgetting the old ones. Adaptive and pre-trained LiLMaps models may have minor differences in the results due to their initial states and optimization processes being different. In Figure 5 we demonstrate that the proposed adaptive optimization can incrementally extend

the decoder to represent new features without catastrophic forgetting of language features observed in the beginning.

The results of Table 3 show that the proposed adaptive optimization LiLMaps^{simple} performs better than pre-trained and fixed models. Our adaptive optimization fits the model to a specific scene while pre-trained decoders (in this case OpenScene's heads) are trained for general language prediction. If a model trained for general prediction is used, then some language features of the environment may be poorly represented in it (e.g. picture and shelving in Table 3), while well-represented features may be irrelevant for the specific scene. This can be seen in the results of OpenScene^{MIT}. OpenScene^{SC} is trained on Matterport3D[2] and has better results. OpenScene^{MS} is trained on ScanNet[46] which is similar to Matterport3D that explains the similar results. However, OpenScene^{HE} pre-trained on NuScenes[45] has significant degradation in the results because the NuScenes environment is more different from Matterport3D. Moreover, our adaptive language decoder optimization allows one to build a custom decoder architecture while employing pre-trained models could restrict available architecture options.

We analyze different values of the threshold τ used to extract unique and unknown features from all input features. Figure 6 shows the final number of features that were considered distinguished and were involved in the optimization at the end of mapping. Lower τ values lead to a larger number of features, but they are still memory efficient. For comparison, the stored features are collected from hundreds of high-resolution images, but their final number is less than 0.5% of the number of pixels in a single image with resolution 640×480. During all tests, adaptive language decoder optimization operated at a rate of 4 frames per second (fps). To achieve real-time performance, adaptive optimization can be executed in parallel to mapping.

5. Conclusion

In this work, we presented an implicit language mapping approach called LiLMaps. We address the problem of unseen language features that appear during the mapping process and the problem of inconsistencies between frames. Currently, LiLMaps is the only approach capable of large-scale incremental implicit language mapping. It can be used alone and enables a variety of interactions with the environment, for instance, 3D language-based object detection (7). Additionally, it can be integrated into existing implicit mapping approaches, introducing only slight overhead.

We evaluated LiLMaps on the public dataset commonly used in related works. Based on the results, we outperform similar works in terms of language mapping quality. However, LiLMaps significantly depends on the quality of visual language features produced by the encoder, which is considered to be an external module in our study. The importance of this dependency is reduced by the proposed Measurement Update strategy that handles inconsistency between frames. We demonstrated that LiLMaps can adapt to the environment

outperforming pre-trained decoders. The proposed Adaptive Optimization demonstrates the ability to prepare decoders given arbitrary language features without the need for actual observations.

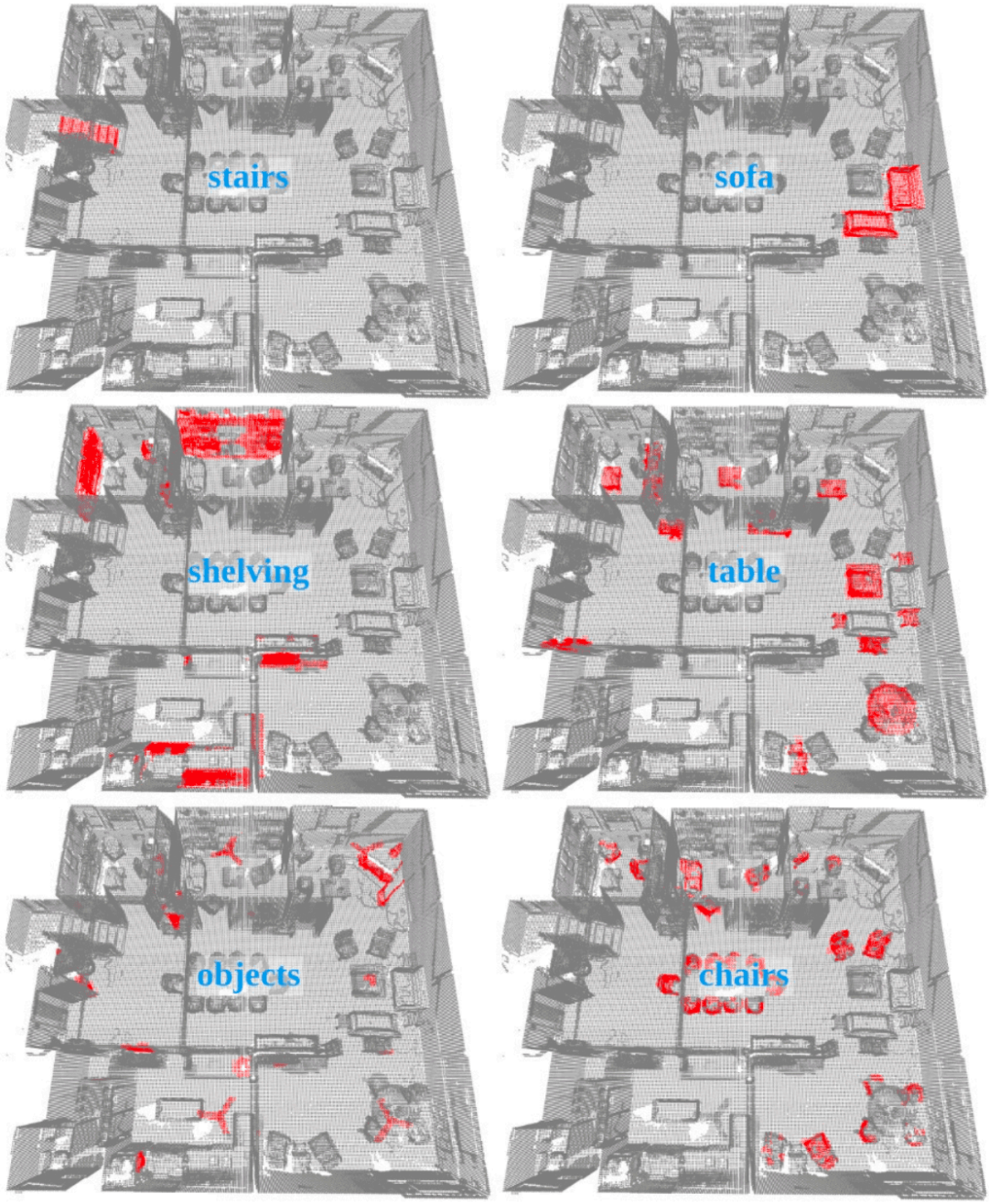

Figure 7. 3D language-based object detection performed on our language map. LiLMaps creates an implicit language map which is reconstructed and queried for different objects. The **highest correspondences** between the reconstructed language map and corresponding request (blue) are highlighted in red.

Acknowledgment

This research was funded by the German Federal Ministry of Education and Research (BMBF) in the project WestAI – AI Service Center West, grant no. 01IS22094A.

References

1. ^a, ^b, ^cRadford A, Kim JW, Hallacy C, Ramesh A, Goh G, Agarwal S, Sastry G, Askell A, Mishkin P, Clark J, et al. Learning transferable visual models from natural language supervision. In: International Conference on Machine Learning (ICLM). PMLR; 2021. p. 8748-8763.
2. ^a, ^b, ^c, ^d, ^eChang A, Dai A, Funkhouser T, Halber M, Niessner M, Savva M, Song S, Zeng A, Zhang Y (2017). "Matterport3D: Learning from RGB-D Data in Indoor Environments". International Conference on 3D Vision (3DV). pages 667--676.
3. ^Anderson P, Wu Q, Teney D, Bruce J, Johnson M, Sünderhauf N, Reid I, Gould S, Van Den Hengel A. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2018. p. 3674-3683.
4. ^Hong Y, Wang Z, Wu Q, Gould S (2022). "Bridging the gap between learning in discrete and continuous environments for vision-and-language navigation". In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 15439–15449.
5. ^Obinata Y, Kanazawa N, Kawaharazuka K, Yanokura I, Kim S, Okada K, Inaba M (2023). "Foundation Model based Open Vocabulary Task Planning and Executive System for General Purpose Service Robots". arXiv preprint arXiv:2308.03357. arXiv:2308.03357.
6. ^Hu Y, Xie Q, Jain V, Francis J, Patrikar J, Keetha N, Kim S, Xie Y, Zhang T, Zhao Z, et al. Toward general-purpose robots via foundation models: A survey and meta-analysis. arXiv preprint arXiv:2312.08782. 2023.
7. ^Dalal M, Chiruvolu T, Chaplot D, Salakhutdinov R. "Plan-Seq-Learn: Language model guided RL for solving long horizon robotics tasks." In: 12th International Conference on Learning Representations (ICLR); 2024.
8. ^Lynch C, Wahid A, Tompson J, Ding T, Betker J, Baruch R, Armstrong T, Florence P (2023). "Interactive language: Talking to robots in real time". IEEE Robotics and Automation Letters (RA-L). 2023.
9. ^a, ^b, ^c, ^d, ^e, ^f, ^g, ^hHuang C, Mees O, Zeng A, Burgard W (2023). "Visual Language Maps for Robot Navigation". In: IEEE International Conference on Robotics and Automation (ICRA).
10. ^Chen B, Xia F, Ichter B, Rao K, Gopalakrishnan K, Ryoo MS, Stone A, Kappler D (2023). "Open-vocabulary queryable scene representations for real world planning". In: IEEE International Conference on Robotics and Automation (ICRA). pp. 11509–11522.
11. ^Raman SS, Cohen V, Rosen E, Idrees I, Paulius D, Tellex S (2022). "Planning with large language models via corrective re-prompting". NeurIPS Foundation Models for Decision Making Workshop (FMDM).

12. ^*Ahn M, Brohan A, Brown N, Chebotar Y, Cortes O, David B, Finn C, Fu C, Gopalakrishnan K, Hausman K, et al. (20 22). "Do as I can, not as I say: Grounding language in robotic affordances". arXiv preprint arXiv:2204.01691.*

13. ^*Song CH, Wu J, Washington C, Sadler BM, Chao WL, Su Y (2023). "LLM-Planner: Few-shot grounded planning fo r embodied agents with large language models". In: IEEE/CVF International Conference on Computer Vision (ICC V). pp. 2998–3009.*

14. ^*Ha H, Song S (2022). "Semantic Abstraction: Open-world 3D scene understanding from 2D vision-language mo dels". In: Conference on Robot Learning (CoRL), volume **205** of Proceedings of Machine Learning Research, pages 643–653. PMLR.*

15. a, b, c, d, e, f, g, h*Peng S, Genova K, Jiang C, Tagliasacchi A, Pollefeys M, Funkhouser T, et al. "OpenScene: 3D scene u nderstanding with open vocabularies." In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CV PR); 2023. p. 815–824.*

16. ^*Chen W, Hu S, Talak R, Carlone L (2022). "Leveraging large language models for robot 3D scene understanding". arXiv preprint arXiv:2209.05629.*

17. ^*Li B, Weinberger KQ, Belongie SJ, Koltun V, Ranftl R. "Language-driven Semantic Segmentation." In: 10th Intern ational Conference on Learning Representations (ICLR); 2022.*

18. ^*Ghiasi G, Gu X, Cui Y, Lin T-Y. "Scaling open-vocabulary image segmentation with image-level labels." In: Euro pean Conference on Computer Vision (ECCV). Springer; 2022. p. 540–557.*

19. ^*Ranasinghe K, McKinzie B, Ravi S, Yang Y, Toshev A, Shlens J. "Perceptual Grouping in Contrastive Vision-Langu age Models." In: IEEE/CVF International Conference on Computer Vision (ICCV); 2023. p. 5548-5561.*

20. a, b*Li MF (2023). "Per-pixel Features: Mating Segment-Anything with CLIP". Available from: https://github.com/j ustin871030/Segment-Anything-CLIP.*

21. ^*Jatavallabhula KM, Kuwajerwala A, Gu Q, Omama M, Iyer G, Saryazdi S, Chen T, Maalouf A, Li S, Keetha NV, Tew ari A, Tenenbaum JB, de Melo CM, Krishna KM, Paull L, Shkurti F, Torralba A. "ConceptFusion: Open-set multimo dal 3D mapping." In: Robotics: Science and Systems XIX (RSS); 2023.*

22. ^*Yang Y, Wu X, He T, Zhao H, Liu X (2023). "SAM3D: Segment anything in 3D scenes". arXiv preprint arXiv:2306.0 3908. Available from: https://arxiv.org/abs/2306.03908.*

23. ^*Kirillov A, Mintun E, Ravi N, Mao H, Rolland C, Gustafson L, Xiao T, Whitehead S, Berg AC, Lo WY, et al. Segment anything. In: IEEE/CVF International Conference on Computer Vision (ICCV); 2023. p. 4015–4026.*

24. a, b*Mildenhall B, Srinivasan PP, Tancik M, Barron JT, Ramamoorthi R, Ng R (2021). "NeRF: Representing scenes as neural radiance fields for view synthesis". Communications of the ACM. **65** (1): 99–106.*

25. a, b*Reiser C, Peng S, Liao Y, Geiger A (2021). "KiloNeRF: Speeding up neural radiance fields with thousands of tiny MLPs". In: IEEE/CVF International Conference on Computer Vision (ICCV). pp. 14335–14345.*

26. ^Bloesch M, Czarnowski J, Clark R, Leutenegger S, Davison AJ. "CodeSLAM -- learning a compact, optimisable rep resentation for dense visual SLAM". In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 201 8. p. 2560--2568.

27. ^Ortiz J, Clegg A, Dong J, Sucar E, Novotn\uoofd D, Zollh\uoof6fer M, Mukadam M (2022). "iSDF: Real-Time Neur al Signed Distance Fields for Robot Perception". In: Robotics: Science and Systems XVIII (RSS), 2022.

28. a, bMatsuki H, Murai R, Kelly PHJ, Davison AJ. "Gaussian splatting SLAM". In: IEEE/CVF Conference on Computer V ision and Pattern Recognition (CVPR). 2024. p. 18039–18048.

29. ^Sucar E, Liu S, Ortiz J, Davison AJ (2021). "iMAP: Implicit mapping and positioning in real-time". In: IEEE/CVF In ternational Conference on Computer Vision (ICCV). pp. 6229–6238.

30. ^Zhu Z, Peng S, Larsson V, Xu W, Bao H, Cui Z, Oswald MR, Pollefeys M. "NICE-SLAM: Neural implicit scalable enc oding for SLAM." In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); 2022. p. 12786-1 2796.

31. ^Zhong X, Pan Y, Behley J, Stachniss C (2023). "SHINE-Mapping: Large-scale 3D mapping using sparse hierarchic al implicit neural representations". IEEE International Conference on Robotics and Automation (ICRA). pp. 8371–8377.

32. a, b, cKerbl B, Kopanas G, Leimkühler T, Drettakis G (2023). "3D Gaussian Splatting for Real-Time Radiance Field Rendering." ACM Transactions on Graphics (TOG). **42** (4): 139--1.

33. ^Zhu S, Qin R, Wang G, Liu J, Wang H (2024). "SemGauss-SLAM: Dense semantic Gaussian splatting slam". arXiv preprint arXiv:2403.07494.

34. ^Naumann J, Xu B, Leutenegger S, Zuo X (2024). "NeRF-VO: Real-time sparse visual odometry with neural radia nce fields". IEEE Robotics and Automation Letters (RA-L). **9** (8): 7278–7285.

35. ^Zhu L, Li Y, Sandström E, Schindler K, Armeni I (2024). "LoopSplat: Loop Closure by Registering 3D Gaussian Spl ats". arXiv preprint arXiv:2408.10154.

36. ^Zhi S, Sucar E, Mouton A, Haughton I, Laidlow T, Davison AJ (2021). "iLabel: Interactive neural scene labelling". arXiv preprint arXiv:2111.14637. Available from: https://arxiv.org/abs/2111.14637.

37. a, bHaughton I, Sucar E, Mouton A, Johns E, Davison AJ. Real-time mapping of physical scene properties with an a utonomous robot experimenter. In: Conference on Robot Learning (CoRL), volume 205 of Proceedings of Machine Learning Research. PMLR; 2022. p. 118-127.

38. a, bMazur K, Sucar E, Davison AJ (2023). "Feature-realistic neural fusion for real-time, open set scene understand ing". In: IEEE International Conference on Robotics and Automation (ICRA). pp. 8201–8207.

39. ^Kerr J, Kim CM, Goldberg K, Kanazawa A, Tancik M (2023). "LERF: Language embedded radiance fields". In: IEE E/CVF International Conference on Computer Vision (ICCV). pp. 19729–19739.

40. ^Qin M, Li W, Zhou J, Wang H, Pfister H (2024). "LangSplat: 3D language Gaussian splatting". In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 20051–20060.

41. ^Müller T, Evans A, Schied C, Keller A (2022). "Instant neural graphics primitives with a multiresolution hash encoding". ACM Transactions on Graphics (TOG). **41** (4): 1–15.

42. ^Wang J, Bleja T, Agapito L. "GO-Surf: Neural feature grid optimization for fast, high-fidelity RGB-D surface reconstruction." In: International Conference on 3D Vision (3DV). IEEE; 2022. p. 433-442.

43. a, bTakikawa T, Litalien J, Yin K, Kreis K, Loop C, Nowrouzezahrai D, Jacobson A, McGuire M, Fidler S (2021). "Neural geometric level of detail: Real-time rendering with implicit 3D shapes". In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 11358–11367.

44. ^Li J, Wen Z, Zhang L, Hu J, Hou F, Zhang Z, He Y (2024). "GS-Octree: Octree-based 3D Gaussian splatting for robust object-level 3D reconstruction under strong lighting". Computer Graphics Forum (CGF). **43** (7): i--xxii.

45. a, bCaesar H, Bankiti V, Lang AH, Vora S, Liong VE, Xu Q, Krishnan A, Pan Y, Baldan G, Beijbom O. "nuScenes: A multimodal dataset for autonomous driving." In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR); 2020. p. 11621-11631.

46. a, bDai A, Chang AX, Savva M, Halber M, Funkhouser TA, Nießner M. "ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes." In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2017. p. 2432–2443.

Declarations

Funding: No specific funding was received for this work.

Potential competing interests: No potential competing interests to declare.