LiLMaps: Learnable Implicit Language Maps

Evgenii Kruzhkov
University of Bonn
Germany
ekruzhkov@ais.uni-bonn.de

Sven Behnke
University of Bonn
Germany
behnke@cs.uni-bonn.de

Abstract

One of the current trends in robotics is to employ large language models (LLMs) to provide non-predefined commands 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 visual-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 visual-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 the 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 the 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 the 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).
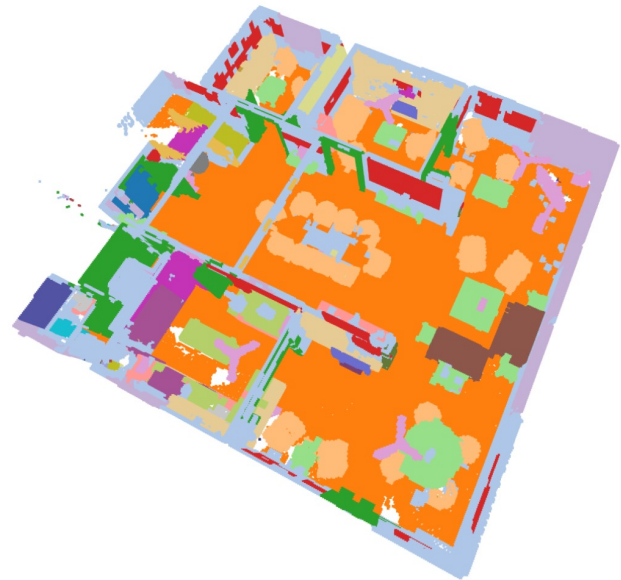

Figure 1. Reconstructed implicit language map built with LiLMaps. Semantic colors are assigned based on the similarity of reconstructed language features and CLIP [33] encodings of semantic categories from Matterport3D dataset [5].

We chose 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 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. Visual 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 visual-language environment exploration with just a small overhead.

2. Related Works

Language and Visual Language Models. Dynamic execution of natural language commands has been an interesting research topic for a long time [2], [13]. 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 robots' abilities to understand and execute natural language commands [29], [14]. In addition, LLMs have been shown to be successful in guiding objects grasping [9], [23], navigation [1, 6, 15, 34, 37] and scene understanding [7, 11, 31]. Visual Language Models (VLMs) [10, 20, 22, 35], which extract language information about the environment from the provided images, are frequently used in conjunction with LLMs. For instance, CLIP [33] is one of the most widely used models capable of mapping images into a natural language space.

Usually, VLMs transfer only single or batched images into the natural language vector space. However, some tasks may benefit from mapping the entire environment into the language space. For example, VLMaps [15] 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), their performance depends on localization and mapping quality, and they are limited to building only 2D language maps. On the other hand, OpenScene [31] 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 [16], demonstrates that language features can be fused into 3D maps using traditional SLAM approaches. SAM3D [41] projects SAM segmentation masks [19] into 3D and creates 3D scene masks.

Implicit Representations. Implicit representations [3, 24, 26, 30, 36] have gained popularity for their compactness and ability to achieve high-resolution reconstructions. They demonstrate great capabilities in environment mapping. iMap [38] uses a RGB-D sensor to perform a real-time SLAM task. Nice-SLAM [46] extends the possible sizes of the mapped environment. SHINE-Mapping [43] demonstrated implicit mapping of outdoor environments. Currently, works based on Gaussian splattings [17] demonstrate exceptional results [24, 28, 44, 45]. In addition to geometry, implicit maps demonstrate a successful reconstruction of semantic information [42], physical properties [12], and visual features [25].

Integration of language features into implicit representation is an actively researched topic. LERF [18] studies the fusion of language features into Radiance Fields, but is limited to small scenes. LangSplat [32] uses Gaussian Splatting [17] to achieve higher precision and training speed. However, LangSplat [32] 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 [26] were able to generalize the predictions [12], [25], but were constrained by the limited size of the environments. The following works successfully increase training and reconstruction speeds [36], [27]. Gaussian Splatting [17] 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 [40], octree-based feature volumes [39] or combined ones [21].

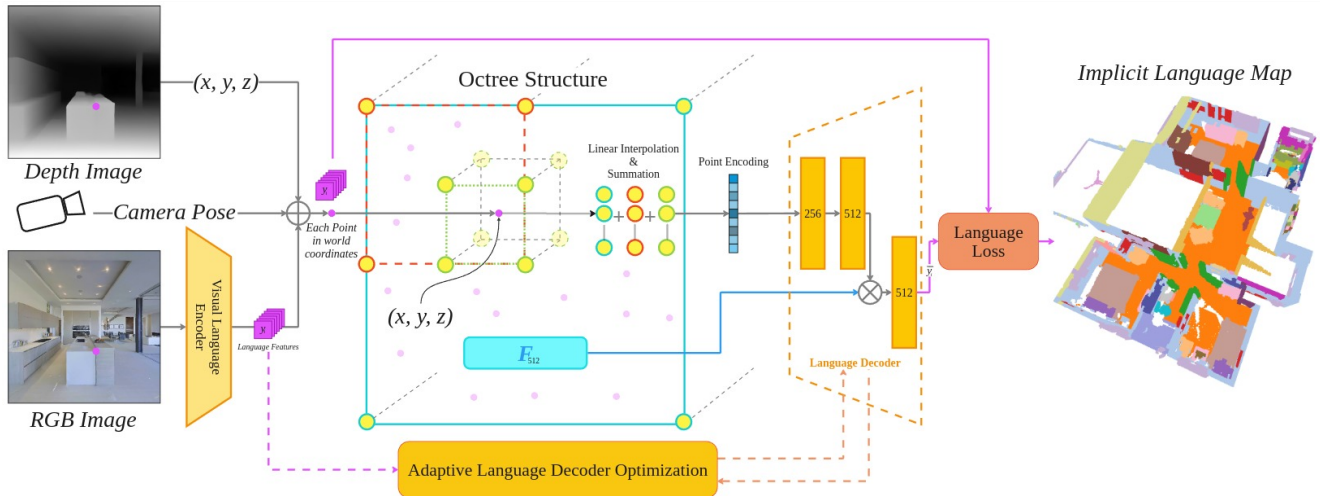Compared to the works discussed above, our approach

Figure 2. Implicit language mapping. Visual-language features y 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 $\bar{y}$ 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).

can build large-scale 3D implicit language maps and can be seamlessly integrated with implicit SLAM methods. In this work, a sparse octree-based representation [39] is utilized to store learnable features, although LiLMaps is not tied to any single representation and can be adapted to others with minimal effort.

3. Method

We investigate the problem of building an implicit visual-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 problem, 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 (Fig. 2) but pixel-wise language features are often inconsistent between frames. We address this problem in Sec. 3.3.

3.1. LilMaps Architecture

Fig. 2 shows the architecture of the proposed approach. Input data for our approach are point clouds associated with CLIP language features (language point clouds), as well as current camera position 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 position. The extraction of language features can be done using per-pixel visual language encoders such as LSeg, OpenSeg, Segment-Anything-CLIP [22], etc. For example, VLMaps [15] 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.

Our goal is to enable implicit visual-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. Such features can be 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 fea-

tures 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 corners features low-dimensional.

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

$$\mathcal{L}_{vl} = -\frac{1}{N}\sum_{i=1}^{N} CosineSimilarity(y_i, \bar{y}_i) \qquad (1)$$

where N is the number of points with visual-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 $\bar{y}$ in spatial coordinates using feature vector F and corners features of the corresponding voxels. Before feeding to the decoder, the corners' 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 $\bar{y}$.

3.2. Adaptive Language Decoder Optimization

Equation (1) uses our MLP-based language decoder to predict visual-language features $\bar{y}$ 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's 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 language features arrives, we perform decoder optimization before the optimization of octree features described in Sec. 3.1.

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 (Fig. 2), the learnable parameters are F vectors (Line 5) and the interpolated and summed point encoding (Line 4). Note that the further described 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

---

Algorithm 1 Adaptive Language Decoder Optimization

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

---

cases, cosine similarity and a predefined threshold Tr 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, 17 and 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 Fig. 2). We initialize only a single F feature vector for all new language features (Line 17) because single 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

Parameter	Symbol	Value
Similarity threshold	Tr	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.

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 visual-language features and list of 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 (Fig. 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. Measurements Update Strategy

During incremental mapping, new observations added to the map should not corrupt previous measurements. However, visual-language features predicted by the visual encoder may be not consistent between frames. VLMaps [15] averaged the language features of the objects received from different views. We note that the averaging can be done in a recursive form:

$$\mathcal{A}_n = \frac{1}{N}\sum_{i=1}^{N} y_i = \mathcal{A}_{n-1}\frac{n-1}{n} + \frac{y_i}{n} \tag{2}$$

In this work we propose to update values of new observations based on the data already stored in the map:

$$y_n = \bar{y}_{n-1}\frac{n-1}{n} + \frac{y_n}{n} \tag{3}$$

It is especially useful for noisy measurements such as visual-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 [15]. However, we observed better results

Figure 3. **Left:** environments reconstructed without active measurements update strategy; **Middle:** Ground Truth; **Right:** environments reconstructed with the measurements update.

when not all previous data are stored and decided to use exponential smoothing instead of the averaging:

$$y_n = \alpha\bar{y}_{n-1} + (1 - \alpha)y_n \tag{4}$$

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

$$\alpha = \frac{CosineSimilarity(y_i, \bar{y}_i)}{0.5 + CosineSimilarity(y_i, \bar{y}_i)} \tag{5}$$

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 [15] through the Habitat simulator using Matterport3D [5]. 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 [33]. 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 Tab. 1.

4.1. Mapping Quality

We evaluate the accuracy (A) and mean recall (mR), precision (mP), intersection over union (mIoU) 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 OpenScene 3D model [31] trained on Matterport3D dataset [5]. The results of random sequences are presented in Tab. 2. To validate the measurements update strategy we also present the results with deactivated measurements update, which are marked by asterisk (LiLMaps*) and activated (LiLMaps).

Seq. Approach	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	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$_{2D}^{mean}$	28	-	-	19	28	-	-	19	28	-	-	19	28	-	-	19	28	-	-	19

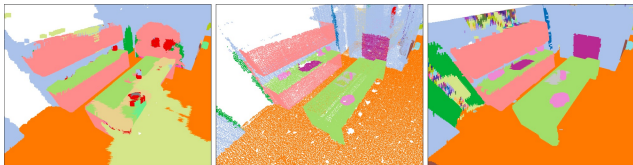Table 2. Language mapping quality evaluation: accuracy (A), recall (mR), precision (mP) and mean IoU (mIoU) in [%].

Figure 4. **Left:** language map produced by OpenScene 3D model [31]; **Middle:** Ground Truth; **Right:** language map created by LiLMaps.

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 until the end.

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 the performance of the method when the input data are close to the ideal ones. 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 them at all.

LiLMaps$_{SEM}$ and LiLMaps*$_{SEM}$ correspond to 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.

As expected, LiLMaps$_{SEM}$ and LilMaps*$_{SEM}$ demonstrate worse results due to the inconsistency of the input data, but both still outperform OpenScene [31]. LilMaps$_{SEM}$ outperforms LiLMaps*$_{SEM}$ due to the proposed Measurements Update technique which addresses potential data inconsistencies. Fig. 3 demonstrates the maps learned with the activated and deactivated Measurements Update procedure. Enabling the Measurements Update results in a cleaner final map, which is crucial for object detection and navigation. 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 have completely missed objects.

Decoders	F1-score				
	$\uparrow$**100%-90%**	$\downarrow$**90%-80%**	$\downarrow\downarrow$**80%-70%**	$\downarrow\downarrow\downarrow$**70%-50%**	$\downarrow\downarrow\downarrow\downarrow$**50%-0%**
$\text{LiLMaps}^{simple}_{Adaptive}$	20	2	1 (picture)	0	0
$\text{LiLMaps}^{simple}_{Pretrained}$	20	2	0	1 (towel)	0
$\text{OpenScene}^{MT}_{HEAD}$	16	4	2	0	1 (objects)
$\text{OpenScene}^{SC}_{HEAD}$	16	5	1	0	1 (objects)
$\text{OpenScene}^{NS}_{HEAD}$	9	8	1	3	2 (picture, shelving)

Table 3. Number of classes falling into different F1-score ranges for compared decoders.

Fig. 4 compares LiLMaps reconstruction with the prediction of the OpenScene 3D model.

For reference purposes, Tab. 2 contains our approach combined with the LSeg model (LiLMaps_{LSeg} and LiLMaps*_{LSeg}) and VLMaps ($\text{VLMaps}^{mean}_{2D}$) with metrics reported in [15]. LSeg could frequently miss objects (e.g., segment a painting as a wall) or provide wrong language features (e.g., detect a bed as a sofa), which significantly influence the final results. However, improving the quality of per-pixel language segmentation is beyond the scope of this research. In all cases, our 3D reconstructed language maps demonstrate better results than the mean results reported in VLMaps [15] 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 [31] 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^{simple}) to match the architecture of OpenScene's head. However, it does not allow us to use the learnable vectors F (Fig. 2) and therefore the results in Tab. 3 are presented when only the learnable corners features with $m = 96$ are optimized.

Tab. 3 summarizes number of classes with distinct F1-score qualities using different optimization types: decoder trained with the proposed adaptive optimization ($\text{LiLMaps}^{simple}_{adaptive}$); decoder pre-trained with the proposed optimization ($\text{LiLMaps}^{simple}_{pretraiend}$); and pre-trained and fixed headings of OpenScene [31] trained on different datasets (Matterport3D [5], nuScenes [4], ScanNet [8]). The first column demonstrates the number of classes reconstructed with F1-score greater than 90%. The method is considered to be better if a larger number of classes are
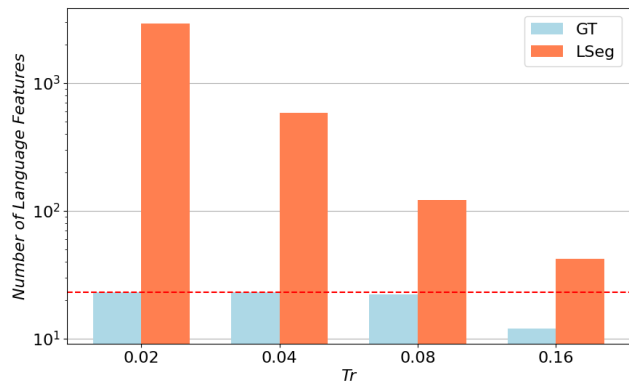

Figure 6. Number of language features stored for the adaptive optimization with varying Tr thresholds. **Blue:** language features are extracted from ground truth (GT) data; **Orange:** language features are extracted by LSeg; **Red line:** total number of different GT classes presented in the scene.

listed in the first column.

To obtain $\text{LiLMaps}^{simple}_{pretraiend}$ 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 ($\text{LiLMaps}^{simple}_{pretraiend}$, $\text{OpenScene}^{MT}_{HEAD}$, $\text{OpenScene}^{SC}_{HEAD}$, $\text{OpenScene}^{NS}_{HEAD}$) during the mapping.

The final results of $\text{LiLMaps}^{simple}_{adaptive}$ are similar to those of $\text{LiLMaps}^{simple}_{pretraiend}$ because every time a new object is observed, the corresponding language features are included in the Adaptive Optimization and the decoder of $\text{LiLMaps}^{simple}_{adaptive}$ 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 Fig. 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.

Tab. 3 demonstrates that the proposed adaptive optimization (both $\text{LiLMaps}_{Adaptive}^{simple}$ and $\text{LiLMaps}_{Pretrained}^{simple}$) 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 Tab. 3), while well-represented features may be irrelevant for the specific scene. This can be seen in the results of $\text{OpenScene}_{HEAD}^{NS}$. $\text{OpenScene}_{HEAD}^{MT}$ is trained on Matterport3D [5] and has better results. $\text{OpenScene}_{HEAD}^{SC}$ is trained on ScanNet [8] which is similar to Matterport3D that explains the similar results. However, $\text{OpenScene}_{HEAD}^{NS}$ pre-trained on NuScenes [4] 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 Tr used to extract unique and unknown features from all input features. Fig. 6 demonstrates the final number of features that were considered distinguished and were involved in the optimization with the corresponding Tr in the end of the mapping. Lower Tr values lead to a larger amount of features, but it is still memory efficient. For comparison, the stored features are collected from hundreds of high-resolution images, but their final amount is less than 0.5% of the number of pixels in a single image with resolution 640x480. 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 present 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 variety of interactions with the environment, for instance, 3D language-based object detection (Fig. 7). Additionally, it can be integrated into existing implicit mapping approaches, introducing only slight overhead.

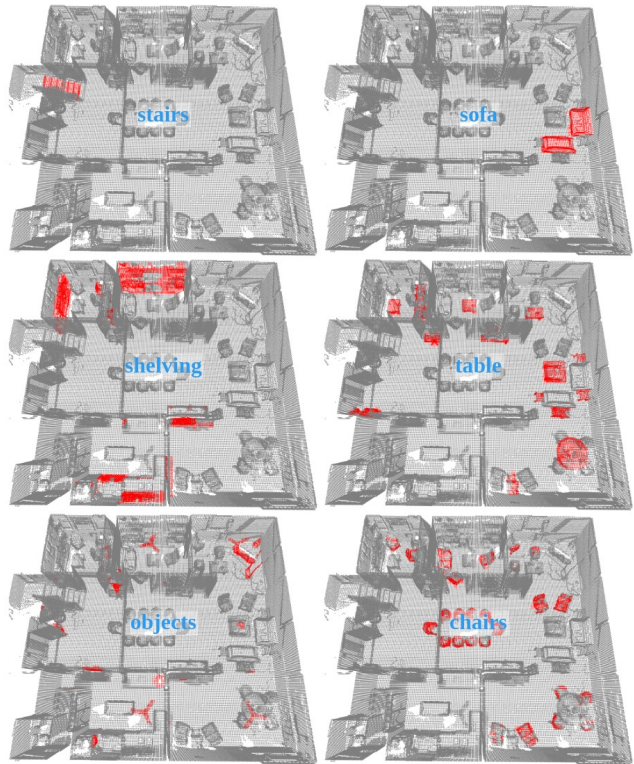We evaluated LiLMaps on the public dataset commonly

Figure 7. 3D language-based object detection performed on our language map. LiLMaps creates 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.

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.

References

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022. 2

[2] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton Van Den Hengel. Vision-and-language navigation: In-

terpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3674–3683, 2018. 2

[3] Michael Bloesch, Jan Czarnowski, Ronald Clark, Stefan Leutenegger, and Andrew J Davison. Codeslam—learning a compact, optimisable representation for dense visual slam. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2560–2568, 2018. 2

[4] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020. 7, 8

[5] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *International Conference on 3D Vision (3DV)*, 2017. 1, 5, 7, 8

[6] Boyuan Chen, Fei Xia, Brian Ichter, Kanishka Rao, Keerthana Gopalakrishnan, Michael S Ryoo, Austin Stone, and Daniel Kappler. Open-vocabulary queryable scene representations for real world planning. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11509–11522. IEEE, 2023. 2

[7] William Chen, Siyi Hu, Rajat Talak, and Luca Carlone. Leveraging large language models for robot 3d scene understanding. *arXiv preprint arXiv:2209.05629*, 2022. 2

[8] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017. 7, 8

[9] Murtaza Dalal, Tarun Chiruvolu, Devendra Chaplot, and Ruslan Salakhutdinov. Plan-seq-learn: Language model guided rl for solving long horizon robotics tasks. *arXiv preprint arXiv:2405.01534*, 2024. 2

[10] Golnaz Ghiasi, Xiuye Gu, Yin Cui, and Tsung-Yi Lin. Scaling open-vocabulary image segmentation with image-level labels. In *European Conference on Computer Vision*, pages 540–557. Springer, 2022. 2

[11] Huy Ha and Shuran Song. Semantic abstraction: Open-world 3d scene understanding from 2d vision-language models. *arXiv preprint arXiv:2207.11514*, 2022. 2

[12] Iain Haughton, Edgar Sucar, Andre Mouton, Edward Johns, and Andrew J Davison. Real-time mapping of physical scene properties with an autonomous robot experimenter. *arXiv preprint arXiv:2210.17325*, 2022. 2

[13] Yicong Hong, Zun Wang, Qi Wu, and Stephen Gould. Bridging the gap between learning in discrete and continuous environments for vision-and-language navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15439–15449, 2022. 2

[14] Yafei Hu, Quanting Xie, Vidhi Jain, Jonathan Francis, Jay Patrikar, Nikhil Keetha, Seungchan Kim, Yaqi Xie, Tianyi Zhang, Zhibo Zhao, et al. Toward general-purpose robots via foundation models: A survey and meta-analysis. *arXiv preprint arXiv:2312.08782*, 2023. 2

[15] Chenguang Huang, Oier Mees, Andy Zeng, and Wolfram Burgard. Visual language maps for robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, London, UK, 2023. 2, 3, 5, 7

[16] Krishna Murthy Jatavallabhula, Alihusein Kuwajerwala, Qiao Gu, Mohd Omama, Tao Chen, Alaa Maalouf, Shuang Li, Ganesh Iyer, Soroush Saryazdi, Nikhil Keetha, et al. Conceptfusion: Open-set multimodal 3d mapping. *arXiv preprint arXiv:2302.07241*, 2023. 2

[17] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023. 2

[18] Justin Kerr, Chung Min Kim, Ken Goldberg, Angjoo Kanazawa, and Matthew Tancik. Lerf: Language embedded radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 19729–19739, 2023. 2

[19] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4015–4026, 2023. 2

[20] Boyi Li, Kilian Q Weinberger, Serge Belongie, Vladlen Koltun, and René Ranftl. Language-driven semantic segmentation. *arXiv preprint arXiv:2201.03546*, 2022. 2

[21] Jiaze Li, Zhengyu Wen, Luo Zhang, Jiangbei Hu, Fei Hou, Zhebin Zhang, and Ying He. Gs-octree: Octree-based 3d gaussian splatting for robust object-level 3d reconstruction under strong lighting. *arXiv preprint arXiv:2406.18199*, 2024. 2

[22] Ming-Feng Li. Per-pixel features: Mating segment-anything with clip, 2023. 2, 3

[23] Corey Lynch, Ayzaan Wahid, Jonathan Tompson, Tianli Ding, James Betker, Robert Baruch, Travis Armstrong, and Pete Florence. Interactive language: Talking to robots in real time. *IEEE Robotics and Automation Letters*, 2023. 2

[24] Hidenobu Matsuki, Riku Murai, Paul HJ Kelly, and Andrew J Davison. Gaussian splatting slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18039–18048, 2024. 2

[25] Kirill Mazur, Edgar Sucar, and Andrew J Davison. Feature-realistic neural fusion for real-time, open set scene understanding. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8201–8207. IEEE, 2023. 2

[26] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. 2

[27] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM transactions on graphics (TOG)*, 41(4):1–15, 2022. 2

[28] Jens Naumann, Binbin Xu, Stefan Leutenegger, and Xingxing Zuo. Nerf-vo: Real-time sparse visual odometry with neural radiance fields. *IEEE Robotics and Automation Letters*, 2024. 2

[29] Yoshiki Obinata, Naoaki Kanazawa, Kento Kawaharazuka, Iori Yanokura, Soonhyo Kim, Kei Okada, and Masayuki Inaba. Foundation model based open vocabulary task planning and executive system for general purpose service robots. *arXiv preprint arXiv:2308.03357*, 2023. 2

[30] Joseph Ortiz, Alexander Clegg, Jing Dong, Edgar Sucar, David Novotny, Michael Zollhoefer, and Mustafa Mukadam. isdf: Real-time neural signed distance fields for robot perception. *arXiv preprint arXiv:2204.02296*, 2022. 2

[31] Songyou Peng, Kyle Genova, Chiyu Jiang, Andrea Tagliasacchi, Marc Pollefeys, Thomas Funkhouser, et al. Openscene: 3d scene understanding with open vocabularies. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 815–824, 2023. 2, 5, 6, 7

[32] Minghan Qin, Wanhua Li, Jiawei Zhou, Haoqian Wang, and Hanspeter Pfister. Langsplat: 3d language gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20051–20060, 2024. 2

[33] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 1, 2, 5

[34] Shreyas Sundara Raman, Vanya Cohen, Eric Rosen, Ifrah Idrees, David Paulius, and Stefanie Tellex. Planning with large language models via corrective re-prompting. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022. 2

[35] Kanchana Ranasinghe, Brandon McKinzie, Sachin Ravi, Yinfei Yang, Alexander T Toshev, and Jonathon Shlens. Perceptual grouping in vision-language models. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2022. 2

[36] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 14335–14345, 2021. 2

[37] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2998–3009, 2023. 2

[38] Edgar Sucar, Shikun Liu, Joseph Ortiz, and Andrew J Davison. imap: Implicit mapping and positioning in real-time. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6229–6238, 2021. 2

[39] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11358–11367, 2021. 2, 3

[40] Jingwen Wang, Tymoteusz Bleja, and Lourdes Agapito. Go-surf: Neural feature grid optimization for fast, high-fidelity rgb-d surface reconstruction. In *2022 International Conference on 3D Vision (3DV)*, pages 433–442. IEEE, 2022. 2

[41] Yunhan Yang, Xiaoyang Wu, Tong He, Hengshuang Zhao, and Xihui Liu. Sam3d: Segment anything in 3d scenes. *arXiv preprint arXiv:2306.03908*, 2023. 2

[42] Shuaifeng Zhi, Edgar Sucar, Andre Mouton, Iain Haughton, Tristan Laidlow, and Andrew J Davison. ilabel: Interactive neural scene labelling. *arXiv preprint arXiv:2111.14637*, 2021. 2

[43] Xingguang Zhong, Yue Pan, Jens Behley, and Cyrill Stachniss. Shine-mapping: Large-scale 3d mapping using sparse hierarchical implicit neural representations. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8371–8377. IEEE, 2023. 2

[44] Liyuan Zhu, Yue Li, Erik Sandström, Konrad Schindler, and Iro Armeni. Loopsplat: Loop closure by registering 3d gaussian splats. *arXiv preprint arXiv:2408.10154*, 2024. 2

[45] Siting Zhu, Renjie Qin, Guangming Wang, Jiuming Liu, and Hesheng Wang. Semgauss-slam: Dense semantic gaussian splatting slam. *arXiv preprint arXiv:2403.07494*, 2024. 2

[46] Zihan Zhu, Songyou Peng, Viktor Larsson, Weiwei Xu, Hujun Bao, Zhaopeng Cui, Martin R Oswald, and Marc Pollefeys. Nice-slam: Neural implicit scalable encoding for slam. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12786–12796, 2022. 2