Self-Improving SLAM in Dynamic Environments: Learning When to Mask - Supplementary Materials

Adrian Bojko
adrian.bojko@cea.fr
Romain Dupont
romain.dupont@cea.fr
Mohamed Tamaazousti
mohamed.tamaazousti@cea.fr
Hervé Le Borgne
herve.le-borgne@cea.fr

Université Paris-Saclay, CEA, List
F-91120,
Palaiseau, France

1 Method details

1.1 SLAM Pipeline

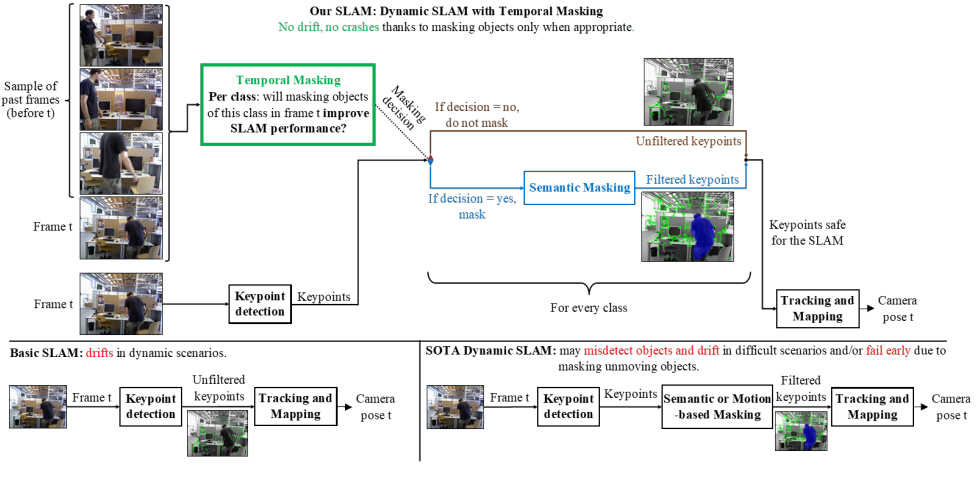


Figure 1: Overview of Dynamic SLAM with Temporal Masking and comparison to other approaches. The key improvement compared to other methods is the per-class choice between masking and not masking objects that does not depend on the SLAM itself, nor priors on object motion or semantics.

We present our SLAM pipeline in Fig. 1. A basic SLAM is composed of two modules: *Keypoint detection* and *Tracking and Mapping*. It takes as input the current frame at time t and outputs the camera pose at time t. A standard Dynamic SLAM adds semantic masking between these two elements, unconditionally filtering keypoints that are on masked objects considered dynamic. Our approach is to add Temporal Masking: a decision module that computes at frame-level and class-level binary masking decisions, *i.e.*, which classes should be masked in the current frame to improve SLAM performance (using a given masking algorithm like Mask R-CNN [4]) and which should not. The decision module uses a video sample as input, which is a set of past frames (before time t) + the current frame at time t. Note that the past frames do not have to be consecutive nor immediately precede the frame at time t: the sample may include frames from the far past.

1.2 LSTM Encoder-Decoder architecture

We propose an architecture for the LSTM Encoder-Decoder in Fig. 2. A ReLU activation and a dropout follow intermediate fully connected layers. The last layer is activated with a sigmoid. We use a binary cross-entropy loss for multi-label classification. The last layer has size $p+1$ as it corresponds to the p semantic classes to mask + a *nothing to mask* class: the latter is a background class and is discarded after inference.

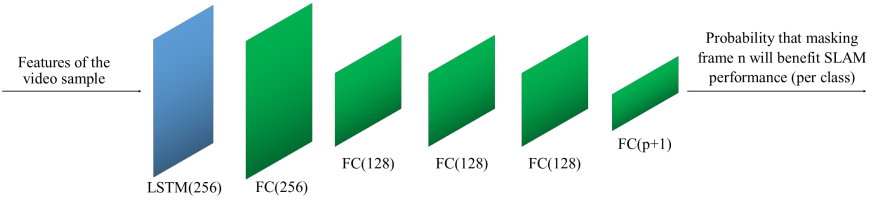


Figure 2: LSTM Encoder-Decoder architecture. When masking p classes, the output has length $p+1$ as it includes a *nothing to mask* class.

1.3 Temporal Annotation: overview and baseline methods

In frame-wise annotations, every frame is separately annotated – manual frame-wise annotations correspond to *fully supervised* training and automatic ones to *self-supervised* training. In sequence-wise annotations, objects of the same class are either masked in all frames or in none – these are weak annotations and correspond to *weakly supervised* training. As data annotation is a costly, SLAM-specific, expert task, we propose to learn temporal masks with self-supervision and compare self-supervision to simpler approaches, full and weak supervision. Figure 3 illustrates the different annotation methods.

Full Supervision Annotations for full supervision are manual and frame-wise: they consist in deciding for every frame and semantic class if masking objects of this class in this frame improves SLAM performance. They require SLAM expertise and become prohibitively costly as the sequence length increases.

Weak Supervision Annotations for weak supervision are manual and sequence-wise: they consist in taking a unique decision for each class and each sequence. The mask of a class

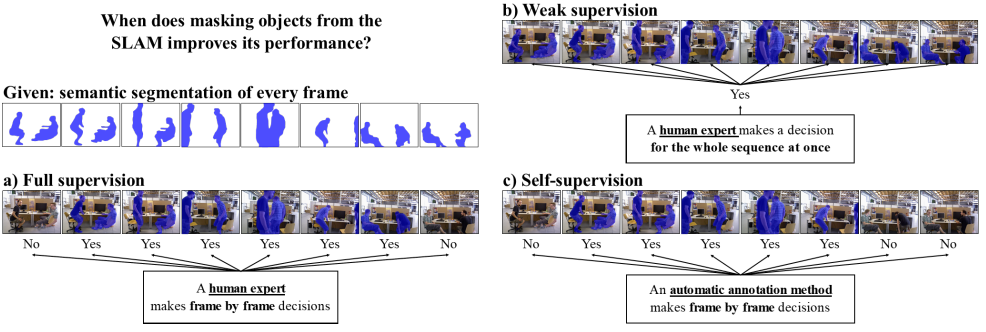


Figure 3: Sequence annotation methods. The annotation consists in deciding, per class, when to mask objects of this class with semantic segmentation. a) In full supervision, a human expert makes decisions for every frame. b) In weak supervision, a human expert makes a single decision that is applied to all frames. c) In self-supervision, an automatic annotation method makes decisions for every frame.

is always active if it improves SLAM performances when at least one frame is masked. Otherwise, the mask is never active for the training sequence. Note that at inference, a model may take different masking decisions within the same sequence even if trained with weak annotations.

1.4 Temporal Annotation: self-supervised method

We first present the annotation method for a single class to mask. Iterating through the full temporal mask space is computationally intractable (up to 2^n masks large, where n is the sequence length). Thus, we compute temporal masks in three steps for every sequence:

1. Random sampling of a subset of all possible temporal masks. The restricted random sampling makes the problem computationally tractable.

2. Benchmarking of the sampled temporal masks using a unified metric. The use of a suitable unified metric makes mask aggregation automatically integrate SLAM failure cases.

3. Performance-weighted aggregation of sampled temporal masks into a unique mask.

The rationale is that samples that perform well tend to mask objects more appropriately, so the result from the aggregation masks objects precisely when appropriate. Step 2 is straightforward: for every sample to test, we run the SLAM applying semantic masks according to the masking decisions in the sample and measure its performance with the unified metric.

1.4.1 Random Sampling of Temporal Masks Subspace

We generate a set of basic temporal masks that we will benchmark to know their impact on SLAM performance. Let l the sequence length, k_0 and k_1 the minimum required length of resp. contiguous blocks of zeros and blocks of ones in a temporal mask (0 = do not mask, 1 = mask). For instance, if we set $l = 7$, $k_0 = 2$ and $k_1 = 3$, then the following temporal masks (0 = do not mask, 1 = mask):

- Respect k_0 and k_1: *1110000* ; *0011100*
- Do not respect k_0 and k_1: *1110111* ; *0011000*

We uniformly sample masks from the space $E(l,k_0,k_1)$ of all temporal masks of length l that respect the min. lengths k_0, k_1. The rationale is that the states of the objects in the scene do not change too quickly: by skipping high-frequency changes (low k_0, k_1), we focus on masks more suited to the scene.

Figure 4: Illustration of sampled temporal masks from a temporal mask space (blue = masked, black = not masked). Conditions k_0, k_1 on masked/unmasked block sizes prevent quick changes between masking states.

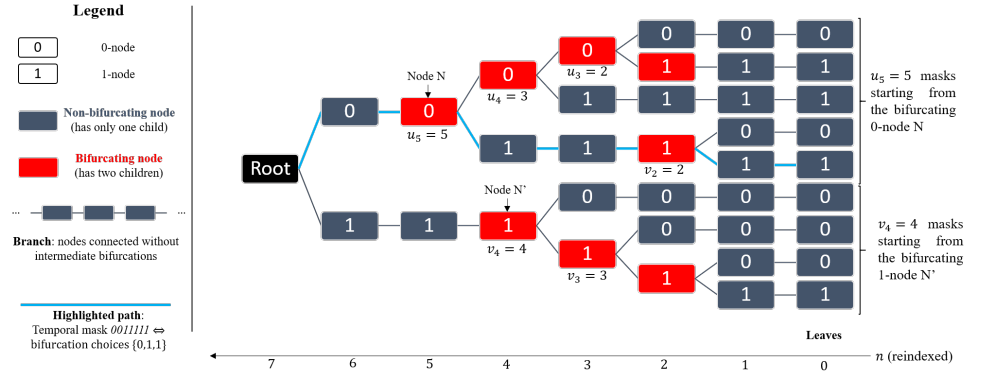
We represent E as a binary tree (see Fig. 5). To keep the problem computationally tractable, the key insight is to use a closed-form expression of the number of possible paths $C(N)$ under any bifurcating node N. Starting from the root, whenever we reach a node that has two children, we randomly pick a child with the odds of each child proportional to the number of possible masks under it. Any mask corresponding to a root-to-leaf path computed in this way has a uniform probability of being sampled from E. Note that if $k_0 = k_1 = 1$ the sampling is trivial as E is the space of all binary strings of length l. Fig. 4 illustrates what sampled masks look like.

Constructing the binary tree representing E. We construct E such that any temporal mask corresponds to the node values of a unique root-to-leaf path in the tree. The differences with an unconstrained binary tree are: 1) All leaves have a depth equal to the sequence length 2) Any temporal mask generated from the tree must respect the minimum block size conditions k_0 and k_1.

Let $i \in 0,1$. We call an *i-node* a node of value i, *i-branch* consecutive i-nodes connected without intermediate bifurcations, and *bifurcating nodes* that have two children. A direct consequence of condition 2) is that we can only add an i-node whose value is different from its parent if it is inside an i-branch at least k_i nodes long. Hence, all temporal masks are defined by a unique set of bifurcation choices: the first branching choice at the tree root + the value of the bifurcating nodes it crosses. Other node values are redundant since they are non-bifurcating and their value determined by their parent.

For instance, let $l = 7, k_0 = 2$, $k_1 = 3$. Fig. 5 illustrates $E(7,2,3)$. If we start with a 0-branch, our mask becomes *00*. Then, if we choose to add a 1-branch (we cannot add less than k_1 1-nodes after a 0-node), our mask becomes *00111*. For the last bifurcation, we can either add a 0-branch (resulting in *0011100*) or a 1-node. In the latter case, we are then forced to add another 1-node as there is no space left for a 0-branch, resulting in *0011111*. Represented as bifurcations choices, *0011111* $\iff \{0,1,1\}$ (highlighted in in Fig. 5).

Uniformly sampling from the masking binary tree. To compute a root-to-leaf path, we make consecutive bifurcation choices. At each step of the traversal, let B the last bifurcation node currently reached. Let $C(N)$ be the number of different masks that can be generated starting from a node N. We define the criterion Q for the next bifurcation choice b: we sample a random number $r \in [0,1[$ then set $b = 0$ if $r < \frac{C(B \cup \{0\})}{C(B)}$ (*i.e.*, we branch towards



- A temporal mask is equivalent to a root-to-leaf path, $l = 7$ nodes long.
- Temporal masks respect the minimum length of consecutive zeros $k_0 = 2$ and ones $k_1 = 3$.
- u_n / v_n: number of masks that can be generated starting at a bifurcating 0-node/1-node at depth n.
  - u_n / v_n depend only on the depth n: bifurcating 0/1-nodes at the same depth have equal subtrees.

Figure 5: Temporal mask space $E(l = 7, k_0 = 2, k_1 = 3)$ as a masking binary tree. A temporal mask is equivalent to a root-to-leaf path.

zero) and $b = 1$ otherwise (*i.e.*, we branch towards one). Hence, the probability $P(b|B)$ of making choice b from node B is $P(b|B) = \frac{C(B \cup \{b\})}{C(B)}$.

Let $M = \{B_0, ..., B_p\}$ an ordered set of bifurcation nodes and $\{b_0, \cdots, b_p\}$ the corresponding branching choices to reach them, made with the criterion Q, defining a mask M in $E(l, k_0, k_1)$. Thus:

$$\begin{aligned} P(B_0, ..., B_p) &= P(b_0) \times P(b_1|b_0) \times ... \times P(b_p|b_0, ..., b_{p-1}) \\ &= \frac{C(B_0)}{|E|} \times \frac{C(B_1)}{C(B_0)} \times ... \times \frac{C(B_p)}{C(B_{p-1})} = \frac{1}{|E|} \end{aligned} \tag{1}$$

Note that $C(B_p) = 1$ (end of tree at the p-th choice) and $C(B_0)$ is the total number of possible masks (*i.e.*, the size of E) since the first bifurcation is the tree root. Thus $P(M) = \frac{1}{|E|}$: the generated mask is uniformly sampled. We only need a closed-form solution for C to be able to uniformly sample temporal masks.

Closed-form solution. We define u_n and v_n as the number of possible masks starting from resp. a bifurcating 0-node and 1-node at depth n. This implies than $C(B) = u_n$ if B is a 0-node and $C(B) = v_n$ otherwise. Given the constraints k_0, k_1 we have the relation:

$$\begin{cases} u_n = u_{n+1} + v_{n+k_1} \\ v_n = u_{n+k_0} + v_{n+1} \end{cases} \tag{2}$$

By re-indexing the depth n from the end of the tree and defining $k := k_0 + k_1$, we have:

$$\begin{cases} u_n = u_{n-1} + v_{n-k_1} \\ v_n = u_{n-k_0} + v_{n-1} \end{cases} \iff \begin{cases} u_n = 2u_{n-1} - u_{n-2} + v_{n-k-1} \\ v_n = 2v_{n-1} - v_{n-2} + v_{n-k-1} \end{cases} \tag{3}$$

Eq. (3) shows that (u_n) and (v_n) are linear recurrence relations with constant coefficients and the same characteristic equation: $x^{k+1} - 2x^k + x^{k-1} - 1 = 0$. [0] gives direct formulas for the initial conditions and approximate solutions. For large n, u_n and v_n have an approximate form $\alpha\rho^n$ with $\rho \gtrapprox 1, \alpha \in]0, 1[$. Finally, we numerically solve the characteristic equation, obtaining a closed-form solution for C. We are now able to directly evaluate criterion Q for

bifurcation choices, *i.e.*, uniformly sample the temporal mask space $E(l,k_0,k_1)$. The total size of E is $|E| = u(l-k_0)+v(l-k_1)$ as the root is not a node per se but only a bifurcation.

1.4.2 Benchmarking of the Sampled Temporal Masks

This step if straightforward: for every sample, we execute the SLAM while applying semantic masks according to the sample being tested. We then evaluate the computed trajectories using a single metric as the USM (sec. 1.5).

1.4.3 Sampled Temporal Mask Aggregation

After measuring the performance (*i.e.*, score) of every temporal mask previously sampled, we aggregate them. To do so, we sum the differences between all pairs of sampled masks (vectors made of -1,0,1) weighted by their score difference. The idea is that a difference in performance is explained by the difference in masking decisions, so the score-weighted sum is a real vector where high values indicate frames that must be masked and low values frames that must not be masked. Let x,y be two sampled temporal masks and s_x,s_y their respective scores. To consider only significant score differences, let σ_a be the absolute noise (below which score differences are meaningless) and σ_r the relative noise (the score difference must be at least σ_r times the first score of the pair). Then we compute the result vector R:

$$R = \sum_{x,y \in \text{Samples}} \begin{matrix} \max(0, \text{sgn}(|s_y - s_x| - \max(\sigma_r |s_x|, \sigma_a))) \\ \times (s_y - s_x)(y - x) \end{matrix} \tag{4}$$

R is a real-valued vector that has high values at indexes where images should be masked and low values where images should not be masked. We normalize R in $[0,1]$ and binarize by applying thresholds in $[0,1]$, generating an arbitrary number of masks. We finally test these masks and select the best one score-wise. If there are equivalent masks (within the noises σ_a and σ_r), we choose the one that masks the most frames. We call this method **Max TM**.

1.4.4 Generalization to multiple classes

Let a sequence of length l and p classes to mask. In the single-class case, we sample q vectors of length l. In the multiclass case, we sample pq vectors that we join in matrices of size $l \times p$. The benchmarking step is unchanged.

The aggregation step is the same up to the computation of R, which is now a real-valued matrix. In the single-class case, R is a vector that we binarize by applying thresholds. In the multiclass case, every class (*i.e.*, column) may have a different optimal threshold. Hence, for every class i: 1) We generate a matrix R_i by zeroing out all columns other than column i. 2) As in the single-class case, we apply thresholds, generating an arbitrary number of temporal masks, evaluate them and select the best one T_i. The rationale is to maximize the relative effect of masking class i. 3) We select the best temporal mask score-wise. 3) We concatenate columns $1,\ldots,p$ of resp. $T_1,\ldots,T_p$, resulting in a temporal mask where all classes are appropriately masked.

1.5 USM: Unified SLAM Metric

ATE RMSE (Absolute Trajectory Error) and sometimes Tracking Rate (% of tracked frames) are two SLAM metrics of interest for our method since they resp. reflect SLAM accuracy

and robustness. As our method maximizes a single scalar value, we propose a metric to unify both ATE RMSE and Tracking Rate (TR),

Regarding the SLAM literature, the performance of a SLAM method is usually reported in terms of ATE RMSE. Sometimes, the authors also report the Tracking Rate. This second metric is nevertheless often ignored although it has important consequences from a practical and scientific point of view. In practice, it is obvious that a method that suddenly fails is not satisfactory and may cause severe issues depending on the application. It also carries a risk to the scientific community since it makes the design of SLAM methods biased towards minimizing the ATE RMSE, regardless of early SLAM failures.

In practice, how can we compare fairly runs with (ATE RMSE=1mm, TR=10%) to (ATE RMSE=1cm, TR=100%)? ATE RMSE is misleading when there are drops in Tracking Rate. One has to compare both at the same time or use the USM. Misleading ATE RMSE/TR typically appear when using the *Full Masks* baseline. This baseline masks everything that might move, so when the solution is not to mask (Fig. 12), it crashes. As the ATE RMSE is computed only on tracked frames, there are less chances for errors compared to a full-sequence run: this means that the ATE RMSE of a run that crashes early is likely lower than one that does not. An extreme case is a crash after the second frame: ATE RMSE would be nearly zero. Thus, it seems important to propose a metric that unifies both ATE RMSE and Tracking Rate.

[8] proposed to use a Area Under Curve approach to take failures into consideration but does not balance ATE RMSE and Tracking Rate into a continuous value, which is necessary to compare different degrees of failure. [7] recently proposed the Penalized ATE RMSE that allows the comparison of different methods in terms of ATE RMSE, which is fixed to a maximal value when the Tracking Rate ρ is below a threshold ρ_c:

$$\text{Penalized ATE RMSE} = \begin{cases} \max(L).(1+\tau), & \text{if } \rho < \rho_c. \\ \text{ATE RMSE}, & \text{otherwise.} \end{cases} \tag{5}$$

Where τ is an arbitrary penalization factor and $\max(L)$ is the highest ATE RMSE over all the compared methods with a Tracking Rate over ρ_c. While going in the proposed direction of metric unification, this metric suffers from several drawbacks. Firstly, it depends on two hyperparameters (τ and ρ_c) that are arbitrarily fixed. More critically, the resulting value depends on the set of methods that is considered. Hence, for the scientific community, the values may be different from one paper to another, making comparison over time difficult.

To address these issues, we propose the Unified SLAM Metric (USM). Let us consider a method that has an ATE RMSE α and a Tracking Rate ρ. We define the function β parametrized by a real ρ_c as:

$$\beta(\alpha,\rho;\rho_c) = \begin{cases} +\infty, & \text{if } \rho < \rho_c. \\ \alpha, & \text{otherwise.} \end{cases} \tag{6}$$

It seems similar to the Penalized ATE RMSE but has one hyperparameter less and, most importantly, avoids any dependence to other methods to get a score. However, in this form β has a major drawback since it attributes a value to the methods that have a Tracking Rate below the threshold ρ_c. Moreover, the fact that this value is infinite makes it difficult to compare methods or to compute an average over several sequences.

Instead, we propose to consider $\exp(-\beta(\alpha,\rho;\rho_c))$. As $e^{-\beta} \sim 1+\beta$ around zero, the resulting metric is quasi linear for small values of ATE RMSE. Inspired by the $mAP^{IoU=.50:0.05:.96}$ defined by the COCO challenge [5] to evaluate object detection, we also propose to integrate

over all possible values of ρ_c, resulting into a remarkably simple expression for our Unified SLAM Metric ς:

$$\varsigma(\alpha,\rho) = \int_0^1 e^{-\beta(\alpha,\rho;\rho_c)} d\rho_c = \rho e^{-\alpha} \tag{7}$$

For a perfect Tracking Rate and a small ATE RMSE, we have $\varsigma(\alpha,\rho) \sim 1-\alpha$, making it consistent with the usual metric used in the SLAM literature. However, when the system fails, the score is penalized proportionally to the Tracking Rate, making the general behavior of the metric correspond to user expectations. However, if one wants to have a different balance between ATE RMSE and Tracking Rate in the final score, it is possible to introduce an hyperparameter λ to control it, resulting into:

$$\varsigma_\lambda(\alpha,\rho) = \rho e^{-\lambda\alpha} \tag{8}$$

λ balances ATE RMSE (*i.e.*, α) and Tracking Rate (*i.e.*, ρ), and ensures dimensional consistency. If $\rho = 100\%$ and $\alpha \ll \frac{1}{\lambda}$, then $\varsigma_\lambda \sim 1-\lambda\alpha$, which is consistent with the usual ATE RMSE metric. If the system fails early (low Tracking Rate), the score is penalized correspondingly. Thus, the general behavior of our metric corresponds to user expectations. We report some scores for several couples α, ρ and λ in Tab. 1.

A practical way to ensure ATE $\ll \frac{1}{\lambda}$ while balancing ATE RMSE/TR is to set $\lambda = \frac{0.1}{\text{Avg. dataset ATE RMSE}}$.

α	ρ	ς	$\varsigma_{\lambda=5}$	$\varsigma_{\lambda=10}$
1 cm	100%	0.99	0.95	0.90
2 cm	100%	0.98	0.90	0.82
3 cm	100%	0.97	0.86	0.74
4 cm	100%	0.96	0.81	0.67
5 cm	100%	0.95	0.77	0.61
10 cm	100%	0.90	0.61	0.37
20 cm	100%	0.82	0.37	0.14
50 cm	100%	0.61	0.08	0.01
1 m	100%	0.36	0.01	0.00
1 cm	90%	0.89	0.86	0.81
1 cm	80%	0.79	0.76	0.72
1 cm	50%	0.50	0.48	0.45
2 cm	90%	0.88	0.81	0.74
2 cm	80%	0.78	0.72	0.65
2 cm	50%	0.49	0.45	0.41

Table 1: Example of score resulting score with the Unified SLAM Metric ς for several values of ATE RMSE α and Tracking Rate ρ. We also port ς_5 and $\varsigma_{\lambda=10}$ that propose a smaller relative importance of low Tracking Rates. **Note**: α is in meter in Eq. (8).

2 Pseudocode

The pseudocode below presents the core steps of our approach.

```
## Hyperparameters: semantic_slam(), slam_metric(), train_split, block sizes
 ↪  (k0, k1), n_samples, n_past_frames

## Compute annotations (i.e., optimal masks)
def binary_tree_sampler():
    u,v = solve_characteristic_equations(n, k0, k1)
    path = []; i=0
    while i < n:
        if random(0,1) < u(i)/(u(i)+v(i)):
            path += [0]; i+= k0
        else:
            path += [1]; i+= k1
    return path

best_masks = []

for sequence in train_split:
    R = 0; mask_set = [binary_tree_sampler() for n in range(n_samples)]

    for mx in mask_set:
        for my in mask_set:
            sx = slam_metric(semantic_slam(sequence, m1))
            sy = slam_metric(semantic_slam(sequence, m2))
            R += (sy-sx)*(my-mx)

    final_masks = binarize_with_thresholds(normalize(R))
    best_masks += [argmax(final_masks, slam_metric)]

## Train network
# First compute the PCA of features from all frames of all sequences
pca_model = compute_pca(resnet50_features(train_seqs))
train_input = []; train_annotations = []

# We do the following for every sequence (length n) in train_split.
for sequence, annotation in zip(train_split, best_masks):
    # Prepare training dataset by creating subsequences. We train the model
     ↪  to infer the correct masking decision for the last frame of the
     ↪  subsequence, a.k.a. the "current frame" at runtime.
    for i in range(n_past_frames, n):
        train_input += [pick_random_frames(n_past_frames, sequence[0:i])] +
         ↪  sequence[i]
        train_annotations += annotation[i]

# Train
spatial_representation = lambda x: pca_model(resnet50_features(x))
model = lambda x: lstm_encoder_decoder(spatial_representation(x))
model_trained = train(model, train_input, train_annotations)
return model_trained
```

Simplified pseudocode of the proposed Self-supervised Temporal Masking approach.

3 Datasets

3.1 Privacy and licensing information

The TUM RGB-D dataset [7] is protected by the Creative Commons 4.0 Attribution License (CC BY 4.0). There are no specific privacy statements but the people appearing in the sequences are clearly willingly recorded.

The KITTI odometry dataset is protected by the Creative Commons Attribution-NonCommercial ShareAlike 3.0 License (CC BY-NC-SA 3.0). The official page[1] specifies that they protected the privacy of people that are involved with the dataset.

The ConsInv dataset is planned for release under similar terms.

3.2 ConsInv Dataset

The ConsInv dataset is made if three subsets: *ConsInv-Indoors*, *ConsInv-Outdoors* and *ConsInv-Extra*, the first two with train/val/test splits. ConsInv-Indoors is designed for single-class experiments on SLAM robustness (at most one object moves in a sequence). ConsInv-Outdoors dataset is designed for multiclass experiments (several objects may move at the same time). ConsInv-Extra includes sequences with the same objects as ConsInv-Indoors but in different environments. We include calibration and raw data, including unused IMU data.

We compute the ground truth using ORB-SLAM 2 [6] without early stopping in stereo mode, which forces frequent bundle adjustments, and with all dynamic objects of the same class masked once any of them moves (we manually annotate the delay). We verified that no excessive masking or motion consensus inversion affected the computation.

Fig. 6 and Tab. 2 illustrate the ConsInv-Indoors dataset. Fig. 7 and Tab. 3 illustrate the ConsInv-Outdoors dataset. Fig. 8 illustrate the ConsInv-Extra dataset (nine sequences in a living room and nine in a meeting room0.

The ConsInv dataset is split in ConsInv-Indoors, ConsInv-Outdoors and ConsInv-Extra, the first two with train/val/test splits. ConsInv-Indoors is designed for single-class experiments on SLAM robustness (at most one object moves in a sequence). ConsInv-Outdoors dataset is designed for multiclass experiments (several objects may move at the same time). ConsInv-Extra includes sequences with the same objects as ConsInv-Indoors but in different environments. We include calibration and raw data, including unused IMU data.

3.2.1 ConsInv-Indoors Dataset.

We built the ConsInv-Indoors dataset (72 seqs: 36 train, 15 val, 21 test; details in supp. mats), made of the subsets ConsInv-Indoors-Dynamic (52 seqs) and ConsInv-Indoors-Static (20 seqs). They include objects moving indoors. We made *ConsInv-Indoors-Static* to test false starts, i.e., incorrect initializations that occur when the camera is static, and the SLAM uses features on moving objects to initialize. Performance is measured in % of prevented false starts. We made *ConsInv-Indoors-Dynamic* to evaluate SLAM robustness to motion consensus inversions and failures due to excessive masking.

[1] http://www.cvlibs.net/datasets/kitti/

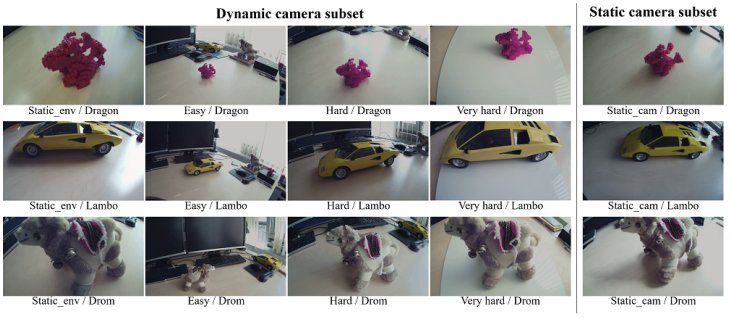


Figure 6: ConsInv-Indoors dataset. It includes three dynamic objects.

Subset	Train	Val	Test
Dynamic	28	12	12
Static	8	3	9
	36	15	21

Table 2: Number of sequences of the ConsInv-Indoors dataset. The camera is mobile in the ConsInv-Indoors-Dynamic subset and fixed in ConsInv-Indoors-Static.

3.2.2 ConsInv-Outdoors Dataset.

We built ConsInv-Outdoors to evaluate SLAM robustness in real outdoor settings. It includes 69 seqs. (44 train, 10 val, 15 test) with cars and pedestrians[2]. They move in some sequences and not in others, sometimes at the same time but not always. Therefore, all-or-nothing strategies (masking all objects or none) are likely to perform poorly.

Figure 7: ConsInv-Outdoors dataset. It includes sequences with vehicles and pedestrians.

Train	Val	Test
44	10	15

Table 3: Number of sequences of the ConsInv-Outdoors dataset.

3.2.3 ConsInv-Extra Dataset.

We made two sets of nine difficult sequences with the dynamic objects of the ConsInv-Indoors dataset, respectively in a meeting room and in a living room.

Figure 8: Context of ConsInv-Indoors (office, left) and contexts of ConsInv-Extra: meeting room (mid) and living room (right).

[2]People appearing in the dataset gave their consent for recording and dataset publication. License and privacy info are in the Github repository.

4 Experiments

4.1 Setup: Spatial Representation Computation

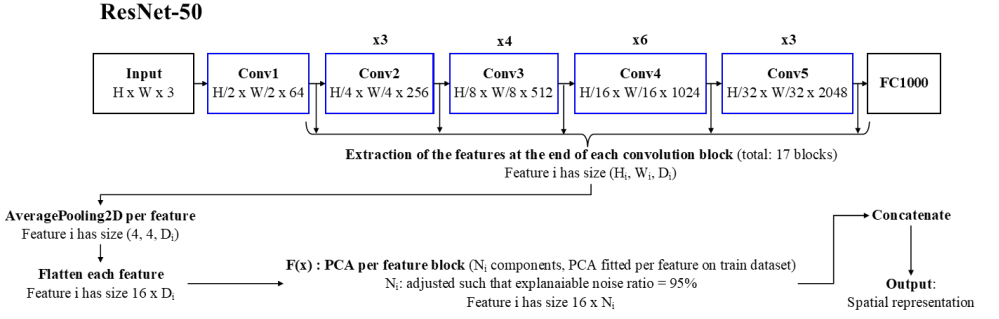


Figure 9: Process to compute spatial representations using ResNet50.

Spatial representation. We use ResNet-50 [3] (TensorFlow 2, pretrained on ImageNet) to compute frame features. We do not fine-tune it. For every frame, we collect the output of each of the 17 convolutional blocks of ResNet-50. For each block, we average-pool-2D its output into size $(4,4,.)$, flatten and apply Principal Component Analysis (computed per block on the training sequences). It results in a feature vector of length about 100 to 1000 per convolutional block, which we concatenate, resulting in the input frame's spatial representation.

Note on the accuracy of temporal masks. The accuracy metric of temporal masks inference is misleading as masking solutions are not unique. An inferred temporal mask could be different from the label yet perform equally SLAM-wise. Accuracy could be low for the same SLAM result; low temporal mask accuracy does **not** imply low SLAM performance. However, there is still a positive correlation between high accuracy and high SLAM performance, which makes early stopping using accuracy possible. But since we are ultimately interested in SLAM performance, mask accuracy is at best unneeded, and at worse misleading, to rely on temporal mask accuracy.

4.2 Interpretation of Inferred Masks

4.2.1 Quantitative interpretation

We showed in the main paper than full supervision – *i.e.*, learning expert manual frame-wise annotations – has a lower performance than self-supervision in addition to the annotation cost. Hence, to better understand why, we directly used manual frame-wise annotations in the SLAM, *without any learning*.

Tab. 4 shows that manual annotations directly input in the SLAM outperforms our method in almost all cases. This means that although our method is better than the state of the art, it can still be further improved. *Manual* gives a new threshold to reach in future works, threshold that can be achieved with better masking decisions.

There are several likely reasons that explain the difference between full supervision (*i.e.*, learned manual annotations) and the direct use of manual annotations. Manual annotations may rely on extremely subtle details to detect motion, like tiny objects reflections on car

Mode	Dataset	Metric	Manual annotations (No learning)	Ours
RGB-D	TUM RGB-D	ATE RMSE (m) ↓ Tracking Rate ↑ USM ↑	**0.019** **96%** **0.80**	**0.019** **96%** **0.80**
Stereo	KITTI	ATE RMSE (m) ↓ Tracking Rate ↑ USM ↑	2.53 **100%** **0.81**	**2.51** **100%** **0.81**
Stereo	ConsInv-Outdoors	USM ↑	**0.93**	0.88
Mono	ConsInv-Indoors-Dynamic	USM ↑	**0.83**	0.75
Mono	ConsInv-Extra-MeetingRoom	USM ↑	**0.74**	0.67
Mono	ConsInv-Extra-LivingRoom	USM ↑	**0.82**	0.74
Mono	ConsInv-Indoors-Static	Prevented false starts ↑	**100%**	**100%**

Table 4: Comparison between our self-supervised approach and manual annotations from a SLAM expert, used directly without learning. All scores are medians over the dataset. For ATE RMSE, lower is better (↓). For Tracking Rate / USM / Prevented false starts, higher is better (↑). We reuse use the model trained on ConsInv-Indoors when evaluating our method on ConsInv-Extra, in order to evaluate how it performs in new contexts.

windows or shadows. Such cues are exceedingly difficult to learn for a neural network – training could require hundreds of sequences, if not more. Another reason is that manual annotations may not be consistent: for instance, if masking an object has no effect on SLAM performance, the annotator might mask it inconsistently in different sequences. Lack of data due to the complexity of label interpretation combined with label inconsistency explains why full supervision performs poorly and further highlights the value of self-supervision.

4.2.2 Qualitative interpretation

Fig. 10 to Fig. 13 show qualitative different results on ConsInv-Outdoors in terms of masking, comparing *No masks* (never masking objects), *Full masks* (always masking objects), *Manual annotations with no learning* and temporal masks inferred with our method, *Self-supervision*. We can observe the following:

1. Manual annotations follow the "mask the bare minimum" approach while self-supervision follows the "mask unless we are sure it is safe" approach.

2. In easy sequences (Fig. 13), where masking has no effect, our approach still prefers to mask objects

3. In hard sequences where excessive masking negatively affects performance (Fig. 10, Fig. 11), our approach masks cars for some time at the start before concluding that "it is not necessary anymore". Fig. 10 shows a difficult case where excessively masking objects leads to SLAM failure.

4. In difficult sequences were objects cause motion consensus inversion (Fig. 12), our approach always masks objects.

5. Our model has, to a certain extent, situational awareness. It learned that a *person* + *car* = *car is not moving*, *i.e.*, masking cars is not necessary when a person is on them

/ very close to them. Likewise, it learned that masking cars is sometimes necessary when a car *has moved in past frames* and not necessarily in the last one. This shows that we do not depend on instantaneous motion detection, and this non-dependence is necessary to solve the motion detection deadlock as explained in the Introduction of the main paper.

6. Masking people is almost always necessary.

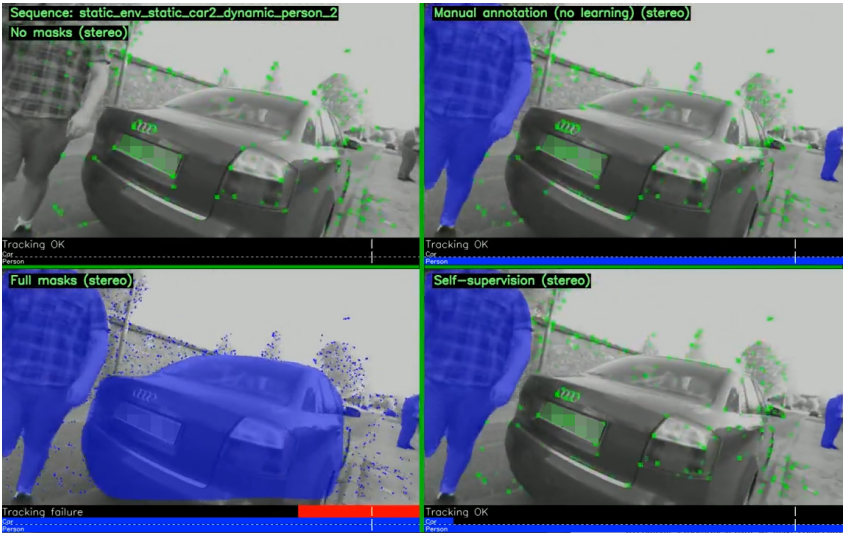


Figure 10: Masking result on a difficult sequence with moving people and static cars. Our approach masks people all the time and almost never cars, making the inferred annotation and the manual annotation very similar. Blue segments indicate masked frames, per class. Red segments indicate tracking failure due to excessive masking.

The general conclusion is that objects are masked when they are *likely to move*, which is a context-dependent definition our model learns. In our experiments, it means an object that is currently moving, an object that was previously seen moving in the current sequence (*e.g.*, a car), or an object known that is often moving (*e.g.*, people).

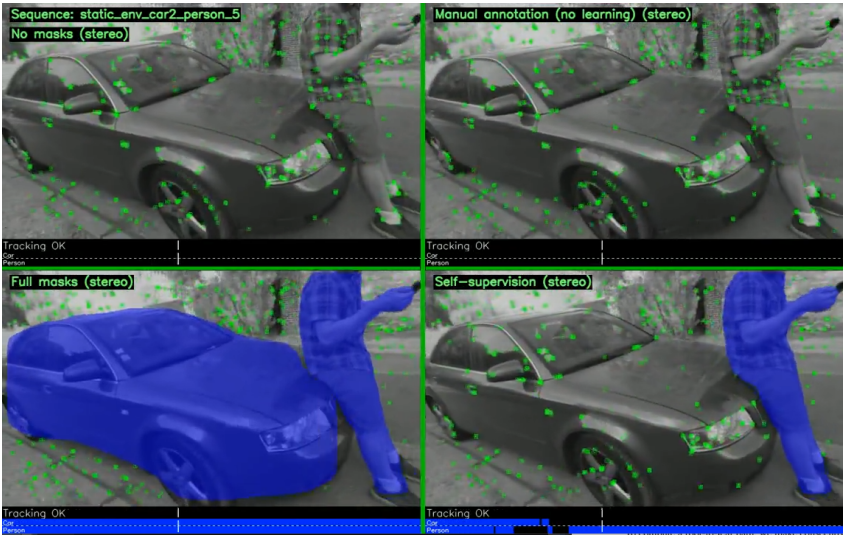


Figure 11: Masking result on a sequence where both people and cars are static. Our approach is more prudent towards masking, but correctly stops masking when needed. Blue segments indicate masked frames, per class.

Figure 12: Masking result on a difficult sequence with moving cars and no people. Our approach says to mask both people and cars all the time out of caution. Blue segments indicate masked frames, per class. Arrows indicate motion direction.

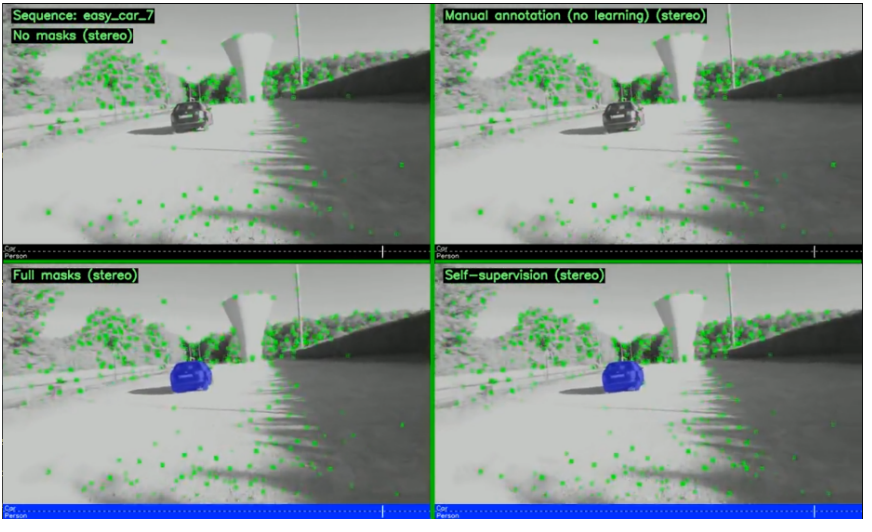


Figure 13: Masking result on an easy sequence with moving cars and no people. Our approach says to mask both people and cars all the time out of caution. Masking objects has no effect in this sequence since the vast majority of features is on the background. Blue segments indicate masked frames, per class.

4.3 Comparison between annotation methods

We manually create weak annotations and full annotations and compare in Tab. 5 the performance of full supervision (using the manual expert annotations on train+val), weak supervision and self-supervision on the ConsInv-Indoors-Dynamic subset, ConsInv-Outdoors (in multiclass mode) and TUM RGB-D dataset. For the ConsInv-Indoors-Dyn. subset (17 train seqs. per object class), the performance of self-supervision is slightly higher. For the TUM RGB-D and ConsInv-Outdoors datasets (resp. 7 and 13 train seqs./class), self-supervision is by far the best approach, followed by weak, then full supervision. Thus, full supervision (i.e., manual frame-wise annotations) and weak supervision (i.e., manual sequence-wise annotations) require more training sequences than self-supervision (i.e., automatic frame-wise annotations).

A first conclusion is that self and weak supervision modes tend to reach the performance of self-supervision with enough data. Additionally, frame-wise *manual* annotations (i.e., full supervision) are surprisingly the most difficult to learn. This is likely because the person annotating sequences relies on cues that are too subtle (*e.g.*, reflections on a car, landmarks that are on the border of the image) for the temporal masking network to learn. Humans tend to over-estimate the learning ability of neural networks, and for this reason machine-generated labels – in a way, closer to the limitations of a computer system – are easier to learn.

The overall conclusion is that self-supervision leads to the best results in addition to removing the need for manual annotations, so we use this method in the rest of the paper.

Dataset	**Full supervision**	**Weak supervision**	**Self-supervision**
ConsInv-Indoors-Dynamic	0.72	0.68	**0.75**
ConsInv-Outdoors	0.74	0.80	**0.88**
TUM RGB-D	0.69	0.70	**0.80**

Table 5: Comparison of supervision modes. Avg. USM on ConsInv/TUM RGB-D.

4.4 Degraded mask quality tests

Segmentation masks can be inaccurate, which may affect the automatic annotations and thus the training. To measure the robustness of our method to this problem, we degraded the quality of the segmentation masks during the annotation process by randomly eroding or dilating every mask by 10px. Tab. 6 shows that our method still has superior performance, albeit slightly lower than before.

Baselines		**Ours**	
No masks	Full masks	Self-sup.	Self-sup. (degraded)
0.57	0.71	**0.75**	0.73

Table 6: Average USM on ConsInv-Indoors-Dynamic to evaluate the robustness to degraded semantic masks.

4.5 Data requirement for training

This section compiles results from the main experiments to estimate how much data is needed to actually train a temporal masking network. The key elements are dataset complexity

Baselines		Ours	
No masks	Full masks	Self-sup.	Self-sup. (degraded)
56%	**100%**	**100%**	**100%**

Table 7: Rate of prevented false starts on ConsInv-Indoors-Static including a degraded mask approach.

– which represents how much the environment/objects change across the dataset and the variety/difficulty of object motion – and dataset size. Of the previously evaluated datasets, TUM RGB-D is the easiest (constant environment, simple object motion), KITTI is easy (similar urban environment, simple object motion), ConsInv-Indoors is hard (same indoors environment, difficult object motions) and ConsInv-Outdoors is very hard (different outdoor environments, difficult object motions). Tab. 8 shows a qualitative appreciation of dataset complexity and size of the datasets previously evaluated.

Dataset	Dataset complexity	Dataset size
TUM RGB-D	+	+
KITTI	+	++
ConsInv-Indoors	++	+++
ConsInv-Outdoors	+++	+++

Table 8: Dataset complexity (variety of environment and object motion) and size of the evaluated datasets.

Empirically, for state-of-the-art performance using self-supervision, we need dataset complexity $\leq$ dataset size. Quantitatively, $\approx$ 10 sequences made of 1000 images per dynamic object class is a good starting point for self-supervision.

4.6 Computation time analysis.

We measured the average time to process a frame on a GTX 1080 Ti GPU once all networks are loaded. We consider full resolution as 1280x720. Results are in Tab. 9. Depending on image resolution, the total time varies from 43ms (23Hz) to 82ms (12Hz), making real-time possible as most of the computation is offloaded to the GPU

Mask inference (DeepLabv3+)	12ms (30% res) or 30ms (60% res)
Image encoding (ResNet50)	22ms (50% res) or 43ms (100% res)
Feature processing (pooling + PCA)	4ms
Masking decision inference	5ms
Total inference time	**43ms** (23Hz) to **82ms** (12Hz)

Table 9: Average inference time on a GTX 1080 Ti, per frame. Full res. is 1280x720.

4.7 Sampling computational tractability.

We give the size of the temporal mask space E in Tab. 10 depending on the number of images n and the minimal block sizes k_0, k_1. The table shows that with block sizes that match real motions ($\approx$1s to start or stop moving, *i.e.*, less than 50 images), the problem becomes intractable after a few seconds of video (at 30Hz). This proves the value of our uniform sampling method since exhaustive exploration of E is computationally intractable.

k	Size of $E(k_0,k_1,n)$						
1	$\lvert E\rvert = 2^n \approx 10^{0.3010n}$						
	$n=$	1	30	100	500	1000	10000
	$\lvert E\rvert =$	2	1.07×10^{9}	1.26×10^{30}	3.27×10^{150}	1.07×10^{301}	1.99×10^{3010}
25	$\lvert E\rvert \approx 0.616\times1.1005^{(n-25)} \approx 0.616\times10^{0.04158(n-25)}$						
	$n=$	1	30	100	500	1000	10000
	$\lvert E\rvert =$	0	2	808	3.53×10^{19}	2.18×10^{40}	3.67×10^{414}
50	$\lvert E\rvert \approx 0.534\times1.0593^{(n-50)} \approx 0.534\times10^{0.0250(n-50)}$						
	$n=$	1	30	100	500	1000	10000
	$\lvert E\rvert =$	0	0	4	9.84×10^{10}	3.24×10^{23}	6.57×10^{248}

Table 10: Approximate size estimation of $E(k_0,k_1,n)$ with block size $k := k_0 = k_1$. The table includes approximate solutions for different sequence lengths n but numerical results were computed with the full solution. It is computationally intractable to fully explore the temporal mask space E if the masked video is longer than a few seconds (at 30Hz).

4.8 Hyperparameter tuning

4.8.1 Choice of dataset annotation methods

In addition to methods used in the main experiments: manual frame-level annotations (for fully supervised training), manual sequence-level annotations (for weakly supervised training) and automatically computed frame-level annotations, *i.e.*, *Max TM* (for self-supervised training), we can consider two other annotation methods: *Min TM* and *Best Random*. *TM* means *Temporal Masking*. *Max TM* refers to the fact that, given temporal masks with the same performance, the automatic annotation method prefers the one masking as many frames as possible (Sec. 1.4.3).

Min TM is the same as Max TM except for the very last step of the sampled temporal mask aggregation: if we obtain several masks that are equivalent in terms of performance, we choose the one that masks the *least* frames instead of the most. This results in masks that are sparse, masking only the bare minimum of frames to reach max SLAM performance. Best Random, on the other hand, consists in completely removing the aggregation step and directly picking the sample that has the best overall score.

We evaluated all annotation methods using the proposed neural network architecture. We used the train split of the ConsInv-Indoors-Dynamic subset for training and evaluated the results on the **val** split of ConsInv-Indoors-Dynamic.

Tab. 11 shows that the automatic annotation method Max TM is the overall best, hence we choose it as our main method in the paper. A possible interpretation is that Min TM masks are too sparse, so the training is very difficult. Best Random masks have no overarching rule (*e.g.*, to mask as little/as much as possible), thus they have no easily identifiable pattern and are very difficult to learn. Overall, automatic methods work better with the Single LSTM architecture.

Between manual and weak annotations, the weak ones perform slightly better, which is a remarkable result given that they cost much less. Another notable result is the fact that manual annotations do not have top performance after learning: the cause is the human bias inherent to manual annotations. A human might use tiny clues like the reflection on a car's window to judge if an object is moving – such clues are exceedingly difficult for the network to learn; additionally, human judgement may be inconsistent, unlike our automatic annotation that always follow the same rules.

Automatic annotations			Manual annotations	
Best Random	**Min TM**	**Self-supervision** (Max TM)	**Weak annotations**	**Full annotations**
0.69	0.68	**0.77**	0.72	0.71

Table 11: Comparison between different annotation methods on the the val split of the ConsInv-Indoors-Dynamic subset.

4.8.2 Automatic annotation parameters

The main parameters of automatic dataset annotation are the minimum block size k_0, k_1 and the number of samples n. Our goal is to compute masks that focus on the key segments of the sequence as much as possible while maximizing SLAM performance. The Min TM method is more practical to evaluate this quality than Max TM as it minimizes the ratio of masked frames: a lower masking ratio at equal performance indicates that the temporal masks fit the sequence better (less useless masking). As the difference between both methods is only at the very last step of mask aggregation, we expect the chosen parameters to also be a good fit for the Max TM method.

We test the annotations directly in the SLAM, without learning them. We first tuned k_0 and k_1 by maximizing the USM. Tab. 12 show that $k_0 = k_1 = 25$ maximize the USM with $USM = 0.91$. Then we tuned n by minimizing the Masking Rate, *i.e.*, the ratio of masked frames within a sequence. Tab. 13 shows that $n = 200$ is a good compromise as the masking rate stagnates from $n = 200$ onwards and its $USM_{n=200} = 0.88$ remains above all other configurations of k_0/k_1. Finally, both Tab. 12 and Tab. 13 show that the proposed automatic annotation methods (Min TM / Max TM) are robust to the choice of $k_0/k_1/n$ (without considering training).

k_0	k_1	USM
1	1	0,88
1	10	0,88
1	25	0,89
1	50	0,88
10	1	0,88
10	10	0,88
10	25	0,88
10	50	0,88
25	1	0,87
25	10	0,88
25	25	**0,91**
25	50	0,87
50	1	0,87
50	10	0,87
50	25	0,87
50	50	0,87

Table 12: Tuning of k_0 and k_1.

n	USM	Masking Rate
50	0,91	0,06
100	0,88	0,05
150	0,89	0,06
200	0,88	**0,04**
250	0,89	0,04
300	0,89	0,04

Table 13: Tuning of n.

References

[1] Richard Austin and Richard Guy. Binary sequences without isolated ones. In *Tfte Fibonacci Quarterly 16.1 (1978):84-86; MR 57 nb. 5778; Zbl*, 1978.

[2] Adrian Bojko, Romain Dupont, Mohamed Tamaazousti, and Hervé Le Borgne. Learning to segment dynamic objects using slam outliers. In *25th International Conference on Pattern Recognition (ICPR)*, 2021.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[4] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-CNN. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[5] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision*, 2014.

[6] R. Mur-Artal and J. D. Tardós. ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-d cameras. *IEEE Transactions on Robotics*, 2017.

[7] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of RGB-d SLAM systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[8] Anqi Joyce Yang, Can Cui, Ioan Andrei Bârsan, Raquel Urtasun, and Shenlong Wang. Asynchronous multi-view slam. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5669–5676, 2021. doi: 10.1109/ICRA48506.2021.9561481.