Self-supervised Dynamic SLAM : Tackling Consensus Inversions

Adrian Bojko

université PARIS-SACLAY

Self-supervised Dynamic SLAM : Tackling Consensus Inversions

SLAM Dynamique Auto-Supervisé : Résolution des Inversions de Consensus

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 580 : Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat : Informatique
Graduate School : Informatique et sciences du numérique
Référent : Faculté des sciences d'Orsay

Thèse préparée au LIST (CEA), sous la direction de Hervé Le Borgne, docteur, le co-encadrement de Romain Dupont, docteur, et le co-encadrement de Mohamed Tamaazousti, docteur.

Thèse soutenue à Paris-Saclay, le 07 avril 2022, par

Adrian BOJKO

Composition du jury

Catherine ACHARD Professeur des universités, Institut National des Sciences Appliquées (INSA) de Rouen	Présidente
Samia AÏNOUZ Professeur des universités, Sorbonne Université	Rapporteur & Examinatrice
David FILLIAT Professeur des universités, École Nationale Supérieure de Techniques Avancées (ENSTA) Paris	Rapporteur & Examinateur
Hervé LE BORGNE Docteur, Commissariat à l'Energie Atomique et aux Energies Alternatives (CEA)	Directeur de thèse

THESE DE DOCTORAT
NNT : 2022UPASG031

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : SLAM Dynamique Auto-Supervisé : Résolution des Inversions de Consensus
Mots clés : slam, dynamique, auto-supervision, inversion de consensus de mouvement, référentiel

Résumé : La capacité d'auto-localisation est essentielle pour les véhicules autonomes, les robots, la réalité mixte et plus généralement les systèmes qui interagissent avec leur environnement. Lorsqu'il n'y a pas de carte disponible, les algorithmes de SLAM (Localisation et Cartographie Simultanées) créent une carte de l'environnement et en même temps y localisent le système. Un capteur populaire est la caméra, qui a l'avantage de fournir passivement une représentation visuelle de l'environnement à bas coût, et donc celui que nous utilisons.

Le SLAM en environnement dynamique, ou SLAM Dynamique, est un défi car l'algorithme doit être capable de percevoir en permanence quelles parties de l'image sont fixes par rapport au référentiel souhaité par l'utilisateur, en général le sol. Des problèmes surviennent lorsque les hypothèses sur lesquelles reposent les algorithmes SLAM deviennent invalides. Un cas remarquable est l'inversion de consensus de mouvement : lorsque la majeure partie d'une image est constituée d'objets en mouvement, l'algorithme n'utilise pas le bon référentiel, et échoue. Un autre est le masquage excessif : certains algorithmes SLAM retirent des images – c'est-à-dire masquent – tous les objets qui pourraient être dynamiques même s'ils ne sont pas en mouvement, et par conséquent échouent si les images deviennent vides.

De façon générale, l'utilisateur peut avoir besoin d'utiliser un algorithme SLAM dans un contexte non supporté. En réalité, l'écart entre ce dont l'utilisateur a besoin et ce que font les algorithmes SLAM est significatif dans la recherche SLAM et la cause de problèmes tels que les inversions de consensus, elles-mêmes rarement présentes dans la littérature. Ainsi, au lieu de proposer un SLAM plus général, nous proposons un algorithme SLAM qui s'adapte à de nouveaux environnements grâce à un apprentissage auto-supervisé automatisé : apprendre automatiquement quelles parties d'une scène peuvent être mobiles par rapport au référentiel souhaité par l'utilisateur et quand il faut les masquer. L'utilisateur fournit des vidéos d'entraînement non annotées et notre algorithme apprend automatiquement quoi en faire.

Nous présentons d'abord l'état de l'art, les bases de données et les métriques SLAM de référence. En particulier, nous détaillons les défis du SLAM Dynamique et de l'évaluation de la robustesse. Les bases de données et métriques SLAM actuelles font partie des points bloquants, nous proposons donc les nôtres.

Dans une deuxième partie, nous explorons les relations entre les points d'intérêt d'une image et les performances du SLAM, et à partir de ce travail, nous présentons un nouvel algorithme de SLAM Dynamique auto-supervisé qui apprend quels objets masquer, en utilisant les outliers SLAM. Les outliers SLAM sont des points d'intérêt rejetés au cours du processus de SLAM : nous avons observé que les outliers sur les objets en mouvement ont des propriétés uniques dans des séquences vidéo faciles et peuvent être utilisés pour apprendre automatiquement à segmenter les objets dynamiques.

Enfin, nous présentons une approche auto-supervisée qui apprend quand masquer des objets : SLAM Dynamique avec Masquage Temporel. A partir d'une méthode donnée de masquage d'objet, on apprend automatiquement quand masquer les objets de certaines classes. On annote automatiquement chaque image des séquences d'entraînement avec des décisions de masquage (s'il faut masquer les objets ou non) puis on apprend les circonstances qui ont mené à ces décisions avec un réseau basé mémoire.

Les résultats de cette thèse montrent que le SLAM Dynamique auto-supervisé est une approche viable pour résoudre les inversions de consensus de mouvement. Plus généralement, l'auto-supervision est la clé pour qu'un SLAM s'adapte aux besoins des utilisateurs. Nous avons dépassé l'Etat de l'Art en termes de robustesse, en plus de clarifier des points aveugles de la littérature en termes d'évaluation de la robustesse des algorithmes de SLAM Dynamique.

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Title : Self-supervised Dynamic SLAM : Tackling Consensus Inversions
Keywords : slam, dynamic, self-supervised, motion consensus inversion, frame of reference

Abstract : The ability of self-localization is essential for autonomous vehicles, robots, mixed reality and more generally to systems that interact with their environment. When maps are not available, SLAM (Simultaneous Localization and Mapping) algorithms create a map of the environment and at the same time locate the system within it. A popular sensor is the camera, which has the benefit of passively providing a visual representation of the environment at a low cost, and for this reason the one we to use for research.

SLAM in Dynamic environments, or Dynamic SLAM, is challenging as the algorithm must be able to continuously perceive what parts of the image are fixed with respect to the frame of reference the user wants, usually the ground. Problems arise when assumptions SLAM algorithms rely on become invalid. A remarkable case is the Motion Consensus Inversion (MCI) : when most of an image is made of moving objects, the SLAM does not use the correct frame of reference and fails. Another one is excessive masking : some SLAM algorithms remove from images – *i.e.*, mask – all objects that might be dynamic even if they are not moving, and consequently fail if images become empty.

More generally, the user may need to use a SLAM algorithm in an unsupported context. In fact, the gap between what the user needs and what SLAM algorithms do is a blind spot in SLAM research and the cause for issues like motion consensus inversions, which are themselves seldom seen in the literature. Hence, instead of proposing a SLAM algorithm that applies to more cases, we propose a SLAM algorithm that adapts to new environments through automated self-supervised training : to automatically learn what objects of a scene may be moving with respect to the user's desired frame of reference, and when they should be masked. The user provides unlabeled training videos and our SLAM automatically learns what to do to with them.

In the first part of this document, we present the State of the Art of algorithms for SLAM and Dynamic SLAM, reference datasets and metrics. We detail the challenges of Dynamic SLAM and robustness evaluation. Current SLAM datasets and metrics are also subject to the user need gap, so we propose our own. Our datasets are the first to explicitly include video sequences with motion consensus inversions or excessive masking and our metric is more general that the usual accuracy metrics, which are misleading in very difficult scenarios.

In the second part, we explore the relation between image features and SLAM performance, and from this work we present a novel self-supervised Dynamic SLAM that learns what objects to mask, using SLAM outliers. Outliers are features rejected during the standard SLAM process : we observed that outliers on objects in motion have unique properties in easy dynamic sequences. Thus, we locate dynamic objects using outliers and learn to segment them, so we can mask dynamic objects in sequences of any difficulty at runtime.

Finally, we present a self-supervised approach that learns when to mask objects : Dynamic SLAM with Temporal Masking. Leveraging an existing method to mask objects, it automatically learns when to mask objects of certain classes. It automatically annotates every frame of training sequences with masking decisions (to mask objects or not), then learns the circumstances that led to these decisions with a memory-based network. We do not make any geometrical assumption, unlike other SLAM algorithms. Using a memory-based approach prevents at runtime motion consensus inversions and excessive masking, which is hardly possible when relying on geometrical methods.

The results of this thesis show that a self-supervised Dynamic SLAM is a viable approach to tackle motion consensus inversions. More generally, self-supervision is the key to have a SLAM adapt to user needs. We surpassed the State of the Art in terms of robustness, in addition to clarifying blind spots of the literature in Dynamic SLAM robustness evaluation.

Remerciements

Maintenant que cette aventure s'est achevée, je souhaite remercier tous ceux qui m'ont aidé au cours de cette thèse : de nombreuses personnes du CEA et ma famille !

Premièrement, un grand merci à mon directeur de thèse, **Hervé Le Borgne**, avec qui j'ai eu l'honneur de travailler. Il a pris le temps qu'il fallait de me guider dans mes recherches. Et il était très présent ! J'ai beaucoup apprécié les exemples très concrets, par exemple "faire comme Columbo" (*i.e.*, commencer par le problème/l'évènement et le résultat/le coupable quand on présente ses recherches). Cela m'a beaucoup aidé à comprendre *comment* faire de la recherche, et non juste ce *qu'est* la recherche.

Puis un grand merci à mes deux co-encadrants, **Romain Dupont** et **Mohamed Tamaazousti**. Ils m'ont donné les premières pistes de recherche. Lors de réunions ou de relectures d'article, leurs remarques étaient toujours très pertinentes. Merci d'être là aux moments les plus importants, entre autres lors des révisions d'articles de dernière minute à l'approche de deadlines de conférences.

Merci au jury, **Samia Aïnouz**, **David Filliat** et **Catherine Achard** qui ont pris le temps de relire mes travaux et ensuite de les juger lors de ma soutenance de thèse. Je suis reconnaissant pour l'attention qu'ils ont prêté à mes recherches.

De même, merci à **Régis Vinciguerra**, chef du CEA LVML (Laboratoire Vision, Modélisation et Localisation – le laboratoire où j'ai effectué ma thèse), et à **Patrick Sayd**, le directeur du SIALV (Service d'Intelligence Artificielle pour le Langage et la Vision – le service incluant le LVML), qui m'ont permis d'effectuer ma thèse au LVML. Ils m'ont soutenu et fait le nécessaire afin que la thèse démarre dans les meilleures conditions. En parallèle de la thèse, participer au Comité de Vie du SIALV fut une excellente expérience qui m'a mené à éditer de nombreuses newsletters. Que de bons souvenirs !

Ensuite, merci aux **membres de l'équipe du CEA LVML** : **Steve, Olivier, Mathieu, Vincent, Laetitia, Richard, Boris...** ainsi que les autres doctorants – mes collègues de bureau – **Mohamed et Jade**. Leurs interventions furent aussi utiles que variées : résolution de problème techniques, obtention de matériel, relecture d'article, discussions d'idées ; tout cela m'a permis d'avancer. Et je n'oublie pas de remercier **Virginie**, qui m'a aidé à faire les procédures admnistratives.

Je remercie aussi les **responsables de FactoryIA**, qui ont mis à disposition leur supercalculateur sans lequel de nombreuses expériences auraient difficilement abouti en temps et en heure.

Enfin, je suis très reconnaissant envers ma famille et je les remercie du fond du cœur pour leur soutien inconditionnel. **Ma mère Ana, mon père André et mon frère Victor** étaient toujours là quand j'en avais besoin. On a passé des moments amusants, comme les fois où on explorait la ville en quête de l'endroit idéal pour filmer des séquences vidéo!

Je ne peux mentionner toutes les personnes avec qui j'ai pu échanger ou travailler lors de ma thèse, mais merci à vous tous !

Acknowledgements

Now that this adventure has come to an end, I would like to thank all those who helped me during this thesis: many people from the CEA and my family!

First, a big thank you to my thesis supervisor, **Hervé Le Borgne**, with whom I had the honor of working. He took the time to guide me in my research. And he was very present! I really appreciated the very concrete examples, for example "to do as Columbo would" (*i.e.*, start with the problem/event and the result/culprit when presenting your research). It helped me a lot to understand *how* to do research, and not just what *is* research.

Then a big thank you to my two co-supervisors, **Romain Dupont** and **Mohamed Tamaazousti**. They gave the first lines of research. During meetings or article proofreading, their remarks were always very relevant. Thank you for being there at the most important times, especially for last-minute paper reviews right before conference deadlines.

Thank you to the jury, **Samia Aïnouz**, **David Filliat** and **Catherine Achard** who took the time to read my work and then judge it during my thesis defense. I am grateful for the attention they gave to my research.

Likewise, thank you to **Régis Vinciguerra**, head of the CEA LVML (Vision, Modeling and Localization Laboratory – the laboratory where I did my thesis), and to **Patrick Sayd**, the director of SIALV (Artificial Intelligence Service for Language and Vision – the service including the LVML), which allowed me to carry out my thesis at the LVML. They supported me and did what was necessary so that the thesis started in the best possible conditions. In parallel with the thesis, participating in the Life Committee of the SIALV was an excellent experience which led me to publish numerous newsletters. What good memories!

Thank you to the **members of the CEA LVML team**: **Steve, Olivier, Mathieu, Vincent, Laetitia, Richard, Boris...** as well as to the other doctoral students – my office colleagues – **Mohamed and Jade**. Their interventions were as useful as they were varied: solving technical problems, obtaining equipment, proofreading articles, discussing ideas; all this allowed me to move forward. And I don't forget to thank **Virginie**, who helped me with the administrative procedures.

I would also like to thank the **FactoryIA managers**, who made their supercomputer available (funded by the Ile-de-France Regional Council) without which many experiments would have been difficult to achieve, and certainly not in time.

Finally, I am very grateful to my family and I thank them from the bottom of my heart for their unconditional support. **My mother Ana, my father André and my brother Victor** were always there when I needed them. We had some fun times, like the times we were exploring the city looking for the perfect place to record video sequences!

I cannot mention all the people with whom I was able to exchange or work during my thesis, but thank you to all of you!

Contents

1 Introduction **19**
- 1.1 Motivation: the SLAM Challenge . . . 19
- 1.2 Context of the thesis . . . 23
- 1.3 Contributions . . . 26
  - 1.3.1 General approach . . . 26
  - 1.3.2 Hypothesis of our thesis . . . 27
  - 1.3.3 Main contributions . . . 28
- 1.4 Structure of the document . . . 29

2 SLAMs and Dynamic SLAMs **31**
- 2.1 SLAM: Simultaneous Localization and Mapping . . . 31
  - 2.1.1 Problem Formulation . . . 31
  - 2.1.2 Sensors . . . 31
  - 2.1.3 Brief history SLAMs . . . 34
  - 2.1.4 Model-based SLAM . . . 35
  - 2.1.5 Learning-Based SLAM . . . 37
  - 2.1.6 Hybrid approaches . . . 39
- 2.2 Dynamic SLAM: SLAM in Dynamic Environments . . . 39
  - 2.2.1 Problem formulation and choice of a feature-based Dynamic SLAM . . . 39
  - 2.2.2 Motion-based, geometrical masking . . . 41
  - 2.2.3 Semantic masking approaches . . . 42
  - 2.2.4 Hybrid approaches . . . 43

3 Understanding the relation between image features and Dynamic SLAM performance **45**
- 3.1 Keypoints as the cornerstone of feature-based SLAMs . . . 46
  - 3.1.1 The use of keypoints in SLAMs . . . 46
  - 3.1.2 Filtering keypoints in Dynamic SLAMs . . . 47
- 3.2 Influence of the number of features on SLAM performance . . . 48
  - 3.2.1 Experimental Setup . . . 48
  - 3.2.2 Evolution of SLAM performance with the number of features . . . 48
- 3.3 Relation between keypoint detector repeatability and SLAM performance . . . 52
  - 3.3.1 The main keypoint detector metric: repeatability . . . 52
  - 3.3.2 Experiments . . . 54
  - 3.3.3 Thoughts on keypoint detector repeatability, SLAM repeatability and self-supervision 59
- 3.4 Relation between outliers and Dynamic SLAM . . . 60
  - 3.4.1 Experimental setup . . . 60
  - 3.4.2 Methods to filter features on moving objects with outliers . . . 61
  - 3.4.3 Methods to densify outliers into masks of moving objects . . . 66
- 3.5 The importance of temporality in keypoint filtering for Dynamic SLAM . . . 70

3.6 Conclusion . . . 71

4 SLAM Robustness: Metrics and Datasets 73
4.1 Introduction . . . 73
4.2 SLAM Robustness . . . 73
4.2.1 General Evaluation Criteria . . . 73
4.2.2 Difficulties of Dynamic Scenarios . . . 74
4.3 Current Metrics, Datasets, and their Limitations . . . 77
4.3.1 Core Metrics . . . 77
4.3.2 Datasets . . . 79
4.3.3 Limitations . . . 80
4.4 Proposed Metrics . . . 83
4.4.1 Penalized ATE RMSE and Success Rate . . . 83
4.4.2 USM: Unified SLAM Metric . . . 83
4.5 Proposed Datasets . . . 85
4.5.1 Ground Truth computation . . . 85
4.5.2 CI Dataset . . . 85
4.5.3 ConsInv Dataset . . . 87
4.6 Conclusion . . . 91

5 From a Robust SLAM to a Dynamic SLAM by Self-Learning of Outliers 93
5.1 Introduction . . . 93
5.2 Learning to Segment Dynamic Objects . . . 94
5.2.1 Overview of the method . . . 94
5.2.2 Outlier and inlier preprocessing . . . 95
5.2.3 Mask creation and network training . . . 97
5.2.4 SLAM Integration . . . 100
5.3 Experiments . . . 100
5.3.1 Experimental setup . . . 100
5.3.2 Results . . . 100
5.3.3 Limitations . . . 106
5.4 Conclusion . . . 106

6 Dynamic SLAM with Temporal Masking 107
6.1 Introduction . . . 107
6.2 SLAM Pipeline . . . 109
6.3 Temporal Masking Network . . . 109
6.4 Temporal Annotation Methods . . . 110
6.4.1 Baseline Methods . . . 110
6.4.2 Self-Supervised Method . . . 111
6.5 Main Experiments . . . 114
6.5.1 Experimental setup . . . 114
6.5.2 Comparison between annotation methods . . . 116
6.5.3 Comparison with the State of the Art . . . 116
6.5.4 Interpretation of Inferred Masks . . . 117
6.5.5 Degraded mask quality tests . . . 122
6.5.6 Computation time analysis and sampling computational tractability. . . . 122
6.5.7 Data requirement for training . . . 122
6.5.8 Hyperparameter tuning . . . 123
6.5.9 Limitations: tests in out-of-context . . . 124
6.6 Complementary work: Dynamic SLAM with Weakly Supervised Temporal Masking . . . 126
6.6.1 Temporal Masking Network for Weak Supervision . . . 126

6.6.2 Experiments 128
6.7 Conclusion 130

7 Conclusion and Perspectives **131**
7.1 Conclusion 131
7.2 Perspectives 132
7.2.1 Further Research 132
7.2.2 Improving Technology Readiness Level for Future Industrialization 133
7.2.3 New Applications 134

A Published SLAM datasets **137**

B Complementary information **141**
B.1 Influence of the number of features on SLAM performance 141

List of Figures

1.1 Example of a static scenario convenient for a SLAM algorithm. 20
1.2 Example of dynamic scenario difficult for a SLAM algorithm, a traffic jam. 21
1.3 Example of very difficult dynamic scenario for a SLAM algorithm. 22
1.4 Example of instance segmentation. 23
1.5 Constrained SLAM. 24
1.6 Application of Augmented Reality to give repair and maintenance instruction in the automotive applications. 24
1.7 Augmented Reality used to visualize custom vehicle furnishing. 25
1.8 Removal of an object across different frames using diminished reality in the presence of specularities. 25
1.9 Feature-based Visual Dynamic SLAM. 27

2.1 Example of stereo + depth camera with an embedded IMU. 32
2.2 Example of LiDARs from the Velodyne family. 33
2.3 Example of RGB+depth camera. 33
2.4 Example of motion capture setup. 34
2.5 General structure of a model-based SLAM pipeline. 35
2.6 Illustration of feature-based and direct SLAMs. 36
2.7 The two main modules of a feature-based SLAM: keypoint generation and Tracking/Mapping. 36
2.8 ORB-SLAM architecture. 38
2.9 ORB-SLAM 2 architecture. 39
2.10 Comparison between model-based SLAMs and a learning-based, end-to-end visual SLAM DeepVO. 40
2.11 The structure feature-based Dynamic SLAM. 41
2.12 An object used in our datasets, a dragon. 42
2.13 Example of optical flow using OpenCV. 42
2.14 Example of optical flow using PWC-Net. 42

3.1 Illustration of matched features in a SLAM. 46
3.2 Benchmark of ORB-SLAM using the GFTT keypoint detector. 49
3.3 Benchmark of ORB-SLAM using the original ORB keypoint detector (with ATE RMSE and Tracking Rate). 49
3.4 Benchmark of ORB-SLAM using the GFTT keypoint detector (with Weighted ATE RMSE and Tracking Rate). 50
3.5 Benchmark of ORB-SLAM using the original ORB keypoint detector (with Weighted ATE RMSE and Tracking Rate). 51
3.6 Benchmark of ORB-SLAM using the original ORB keypoint detector (with Weighted ATE RMSE and Tracking Rate). 51
3.7 Illustration of images from the HPatches dataset for keypoint detector repeatability tests. 52
3.8 Illustration of keypoint detection on the original image and the warped image. 53

3.9 Visual representation of keypoint repeatability computation. . . . 53
3.10 A synthetic image for training or repeatability tests. . . . 54
3.11 Illustration of SuperPoint training process. . . . 54
3.12 Illustration of SuperPoint homographic adaptation. . . . 54
3.13 Examples of images of degraded quality using various levels of JPEG compression. . . . 56
3.14 Illustration of different keypoints detection methods on an image at 50% quality. Original image from the MS COCO dataset. . . . 56
3.15 Illustration of different keypoints detection methods on an image at 90% quality. . . . 57
3.16 Illustration of different keypoints detection methods on an image at 100% quality. . . . 57
3.17 Steps from the ORB-SLAM pipeline from where we can extract outliers. . . . 60
3.18 Typical repeatability profile of ORB-SLAM 2 inliers. Relative threshold = 0.8. . . . 61
3.19 Typical repeatability profile of ORB-SLAM 2 outliers. Relative threshold = 0.2. . . . 62
3.20 Scene before and after a person gets up (10 frames apart). . . . 62
3.21 Image where a person before it starts moving its upper body. . . . 63
3.22 Example of detected outliers after applying different relative thresholds. . . . 63
3.23 Dense masks constructed from outliers collected with the relative threshold $s = 0.2$. . . . 64
3.24 Masks with holes constructed from outliers and inliers. . . . 65
3.25 Computation of superpixels in a simple scene, various sets of parameters (binary). . . . 66
3.26 Computation of superpixels in a simple scene, various sets of parameters (gradient). . . . 67
3.27 Comparison between a SLAM with no masked object and a SLAM with masked objects. 68
3.28 Segmentation of a simple scene using RVOS. . . . 69
3.29 Example of segmentation using COSNet. . . . 69
3.30 Another example of segmentation using COSNet. . . . 69
3.31 Example of possible masking decisions. . . . 70
3.32 Another example of possible masking decisions. . . . 70

4.1 Examples of Motion Consensus Inversions (MCI). . . . 75
4.2 Examples of Excessive Masking leading to SLAM failure. . . . 76
4.3 Illustration of the KITTI Dataset. . . . 81
4.4 Illustration of the TUM RGB-D Dataset. . . . 82
4.5 Miniatures of our CI dataset. . . . 86
4.6 ConsInv-Indoors dataset. . . . 88
4.7 Illustration of the ConsInv-Outdoors Dataset. . . . 89
4.8 Illustration of the ConsInv-Extra-MeetingRoom Dataset. . . . 90
4.9 Illustration of the ConsInv-Extra-LivingRoom Dataset. . . . 90

5.1 Overview of our approach (learning to segment dynamic objects). . . . 94
5.2 Steps of our method (learning to segment images). . . . 95
5.3 Illustration of the sliding window process. . . . 98
5.4 Illustration of the segmentation process. . . . 99
5.5 Illustration of the training process. . . . 99
5.6 Dynamic SLAM at runtime. . . . 100
5.7 Consensus inversion in fr3_walking_xyz (TUM RGB-D). . . . 102
5.8 Failure cases of baselines methods (ORB-SLAM 2 + existing network). . . . 103
5.9 Example of monocular false start. . . . 104
5.10 Method applied to LDSO. . . . 105

6.1 Illustration of our results on the TUM RGB-D dataset. . . . 107
6.2 Temporal Masking Network. . . . 108
6.3 Overview of Dynamic SLAM with Temporal Masking and comparison to other approaches. 109
6.4 LSTM Encoder-Decoder architecture. . . . 110
6.5 Sequence annotation methods. . . . 111

6.6 Illustration of sampled temporal masks from a temporal mask space. 112
6.7 Temporal mask space $E(l = 7, k_0 = 2, k_1 = 3)$ as a masking binary tree. 113
6.8 Process to compute spatial representations using ResNet50. 115
6.9 Results of Temporal Masking on ConsInv-Outdoors, part 1. 119
6.10 Results of Temporal Masking on ConsInv-Outdoors, part 2. 120
6.11 Masking result on an easy sequence with moving cars and no people. 121
6.12 Architectures of the LSTM Encoder-Decoder. 127

B.1 Benchmark of ORB-SLAM using the GFTT keypoint detector (with Weighted ATE RMSE and Tracking Rate). 141
B.2 Benchmark of ORB-SLAM using the Shi keypoint detector (with Weighted ATE RMSE and Tracking Rate). 142
B.3 Benchmark of ORB-SLAM using the Shi-Harris keypoint detector (with Weighted ATE RMSE and Tracking Rate). 142
B.4 Benchmark of ORB-SLAM using the SIFT keypoint detector (with Weighted ATE RMSE and Tracking Rate). 143

List of Tables

3.1 Repeatability of various keypoints detectors (Harris and trained models based on SuperPoint) on various dataset samples of 200 images. . . . 55
3.2 Mean rank of keypoint detectors in single-layer mode on the TUM RGB-D dataset. . . . 58
3.3 Mean rank of keypoint detectors in multi-layer mode on the TUM RGB-D dataset. . . . 58
3.4 Statistics on the ATE RMSE (mm) on 100 SLAM runs on the first 700 images of various sequences of the TUM RGB-D dataset. . . . 64

4.1 Example of score resulting score with the Unified SLAM Metric ς for several values of ATE RMSE α and Tracking Rate ρ. . . . 84
4.2 Number of sequences of the ConsInv dataset. . . . 87

5.1 Average Penalized ATE RMSE (m) of the State-of-the-Art and baselines on Consensus Inversion/Dynamic and TUM RGB-D/Dynamic datasets. N/A indicates that the SLAM mode is not supported. . . . 101
5.2 Success Rate (%) of the State-of-the-Art and baselines on Consensus Inversion/Dynamic and TUM RGB-D/Dynamic datasets. N/A indicates that the SLAM mode is not supported. . . . 101
5.3 Evaluation on Consensus Inversion/Static dataset. We report the ratio of sequences that do not cause initialization fails (false starts). . . . 101
5.4 Average Penalized ATE RMSE (m) and Success Rate (%) of LDSO and our masked version on the Consensus Inversion/Dynamic dataset. . . . 104

6.1 Comparison of supervision modes. Average USM on ConsInv/TUM RGB-D. . . . 116
6.2 Comparison with the State of the Art on various datasets in their preferred mode. . . . 117
6.3 Comparison between our self-supervised approach and manual annotations from a SLAM expert, used directly without learning. . . . 118
6.4 Average USM on ConsInv-Indoors-Dynamic to evaluate the robustness to degraded semantic masks. . . . 122
6.5 Rate of prevented false starts on ConsInv-Indoors-Static including a degraded mask approach. . . . 122
6.6 Average inference time on a GTX 1080 Ti, per frame. Full res. is 1280x720. . . . 122
6.7 Approximate size estimation of $E(k_0, k_1, n)$ with block size $k := k_0 = k_1$. . . . 123
6.8 Dataset complexity (variety of environment and object motion) and size of the evaluated datasets. . . . 123
6.9 Comparison between different annotation methods on the validation split of the ConsInv-Indoors-Dynamic subset. . . . 124
6.10 Tuning of k_0 and k_1. . . . 125
6.11 Tuning of n. . . . 125
6.12 Comparison between different annotation methods and architectures on the validation split of the ConsInv-Indoors-Dynamic subset. . . . 128

6.13 Comparison between supervision modes with different architectures. Average USM on ConsInv/TUM RGB-D. 128
6.14 Comparison with the State of the Art on various datasets in their preferred mode. (with weak supervision). 129
6.15 Dataset complexity (variety of environment and object motion) and size of the evaluated datasets. 129

A.1 List of published SLAM datasets, part 1. 138
A.2 List of published SLAM datasets, part 2. 139

Synthèse en français

La capacité d'auto-localisation est essentielle pour les véhicules autonomes, les robots, la réalité mixte et plus généralement les systèmes qui interagissent avec leur environnement. Lorsqu'il n'y a pas de carte disponible, les algorithmes de SLAM (Localisation et Cartographie Simultanées) créent une carte de l'environnement et en même temps y localisent le système. Un capteur populaire est la caméra, qui a l'avantage de fournir passivement une représentation visuelle de l'environnement à bas coût, et donc celui que nous utilisons.

Le SLAM en environnement dynamique, ou SLAM Dynamique, est un défi car l'algorithme doit être capable de percevoir en permanence quelles parties de l'image sont fixes par rapport au référentiel souhaité par l'utilisateur, en général le sol – mais pas toujours. Des problèmes surviennent lorsque les hypothèses sur lesquelles reposent les algorithmes SLAM deviennent invalides. Un cas remarquable est l'inversion de consensus de mouvement : lorsque la majeure partie d'une image est constituée d'objets en mouvement, l'algorithme n'utilise pas le bon référentiel, et échoue. Un autre est le masquage excessif : certains algorithmes SLAM retirent des images – c'est-à-dire masquent – tous les objets qui pourraient être dynamiques même s'ils ne sont pas en mouvement, et par conséquent échouent si les images deviennent vides.

De façon générale, l'utilisateur peut avoir besoin d'utiliser un algorithme SLAM dans un contexte non supporté. En réalité, l'écart entre ce dont l'utilisateur a besoin et ce que font les algorithmes SLAM est significatif dans la recherche SLAM et la cause de problèmes tels que les inversions de consensus, elles-mêmes rarement présentes dans la littérature. Concrètement, l'utilisateur souhaite localiser la caméra par rapport à un certain référentiel – le sol, la l'intérieur d'un navire, l'intérieur d'un train... – et donc le calcul de trajectoire doit se faire par rapport aux objets dont la position est fixe dans ce référentiel. Toutefois, les algorithmes de SLAM Dynamique ne peuvent pas deviner à l'avance quel est le référentiel attendu par l'utilisateur et c'est pourquoi ils séparent les éléments de l'image qui sont probablement dynamiques des autres: mais les scènes complexes contenant des objets inhabituels et/ou avec une proportion objets statiques/objets dynamiques très déséquilibrée sont mal gérées par les algorithmes existants.

Ainsi, au lieu de proposer un SLAM plus général, supportant une plus grande variété d'environnements, nous proposons un algorithme SLAM qui s'adapte à l'environnement de l'utilisateur. Grâce à un apprentissage auto-supervisé automatisé, notre algorithme apprend automatiquement quelles parties d'une scène peuvent être mobiles par rapport au référentiel souhaité par l'utilisateur et quand il faut les masquer. L'utilisateur fournit des vidéos d'entraînement non annotées et notre algorithme apprend automatiquement quoi en faire.

Nous présentons d'abord l'état de l'art, les bases de données et les métriques SLAM de référence. En particulier, nous détaillons les défis du SLAM Dynamique et de l'évaluation de la robustesse. Les bases de données et métriques SLAM actuelles font partie des points bloquants, nous proposons donc les nôtres, notamment la métrique Unified SLAM Metric (USM).

Dans une deuxième partie, nous explorons les relations entre les points d'intérêt d'une image et les performances du SLAM, et à partir de ce travail, nous présentons un nouvel algorithme de SLAM Dynamique auto-supervisé qui apprend quels objets masquer, en utilisant les outliers SLAM. Les outliers SLAM sont des points d'intérêt rejetés au cours du processus de SLAM : nous avons observé que les outliers sur les objets en mouvement ont des propriétés uniques dans des séquences vidéo faciles

et peuvent être utilisés pour apprendre automatiquement à segmenter les objets dynamiques.

Enfin, nous présentons une approche auto-supervisée qui apprend quand masquer des objets : SLAM Dynamique avec Masquage Temporel. A partir d'une méthode donnée de masquage d'objet, on apprend automatiquement quand masquer les objets de certaines classes. On annote automatiquement chaque image des séquences d'entraînement avec des décisions de masquage (s'il faut masquer les objets ou non) puis on apprend les circonstances qui ont mené à ces décisions avec un réseau basé mémoire. Le Masquage Temporel est un concept nouveau qui est agnostique au choix d'algorithme SLAM et de méthode de masquage, et qui présente des perspectives très intéressantes.

Les résultats de cette thèse montrent que le SLAM Dynamique auto-supervisé est une approche viable pour résoudre les inversions de consensus de mouvement. Plus généralement, l'auto-supervision est la clé pour qu'un SLAM s'adapte aux besoins des utilisateurs. Nous avons dépassé l'Etat de l'Art en termes de robustesse, en plus de clarifier des points aveugles de la littérature en termes d'évaluation de la robustesse des algorithmes de SLAM Dynamique. Enfin, les méthodes serviront à de futures applications industrielles au sein du CEA, où cette thèse a été réalisée.

Chapter 1

Introduction

1.1 Motivation: the SLAM Challenge

Self-localization is a core requirement for many modern applications, as it is necessary, for instance, for path planning and to ensure system safety. It usually relies on a map already available. When planning a path to a specific location, the first step is to know where we are. Self-localization is also necessary to ensure system safety: a typical example is that of an airplane as they have to fly within certain altitude limits and cannot enter no-flight zones; likewise, ships must stay in waters safe for navigation. But in various cases, we need self-localization without having an available map. For example, autonomous vehicles may need to move in an unmapped, rural area. Robots and drones deployed to explore unknown environments cannot plan their motion if they do not know where they are or what is around them. Augmented and mixed reality applications running on portable devices like smartphones may be used anywhere, including private areas, and need to understand their environment and where they are within it. In other words, systems need to self-localize while building a map of the environment *at the same time*. This task is known as Simultaneous Localization and Mapping (SLAM). These algorithms solve a circular problem: *I need to know where I am within the environment to build a map of it, and I need a map of the environment to know where I am within it.*

As SLAM algorithms build a map of its surroundings, they need to acquire information with sensors. The most used ones to observe the environment are cameras, LiDARs and sonars. Cameras capture images representing the environment; LiDARs generate point clouds of the environment using lasers; sonars do the same as LiDARs but use sound instead, typically underwater. Major advantages of cameras over the other sensors include the low cost, the fact that the captured images can be used for high-level interpretations of the scene, and the passive aspect of the sensor. A passive sensor does not emit light nor sound, which can be a hazard in certain contexts (*e.g.*, medical or military). While sonars work better underwater and map creation is easier from LiDAR point clouds than images, cameras remain indispensable sensors in almost all systems that need SLAM capabilities. This explains why cameras are used in most autonomous systems. Common types of cameras include monocular cameras, which have only one sensor, and stereo cameras that have two sensors (*e.g.*, left and right). Cameras are often used with other sensors as Inertial Measurement Units (IMUs, which measure acceleration and rotational speed), but IMUs cannot be used alone for mapping or SLAM as they do not observe the environment. For brevity, we shorten references to SLAM algorithms to just *SLAMs* in the rest of the thesis.

Hence, the ability of camera-based SLAM – also known as Visual SLAM – is essential for modern applications. Cameras project the 3D environment onto a series of 2D images (*i.e.*, a video sequence), and this transformation is defined through a process called calibration. The principle of a Visual SLAM is to compute the 3D points that make the environment – the map – that led to the video sequence, and at the same time compute the 6D pose of the camera (translation + orientation) with respect to the map in every frame.

There are in general three steps in a Visual SLAM: initialization, data association and tracking/mapping. Initialization consists in matching image features that are observed in different frames (*e.g.*, corners of a table) and computing the 3D points and camera poses that led to their observation. This is possible as matched features are projections of the same 3D points, so there is a mathematical relation between features that depends on the position of 3D points and the pose of the camera in both frames. The Visual SLAM algorithm first initializes the map then repeatedly: 1) Matches newly detected features with the existing map (data association) 2) Computes the camera pose (tracking) – solving what is called the Perspective-n-Point problem 3) Updates the map with new 3D points if needed (mapping).

This approach supposes that the environment is static, which is a fundamental, and convenient, hypothesis called the *static world assumption*. It means that the environment does not change with time. For instance, the pose of large structures like buildings rarely changes. The static world assumption is convenient as it implies that all 3D points are fixed in the same frame of reference (the world frame, which is usually the ground), so they can all be used to compute camera poses. Figure 1.1 shows an example of such a static scenario. However, when the world is not static, SLAM becomes much more difficult. Moving objects as cars/trucks/pedestrians/etc. may confuse the SLAM and make it compute erroneous trajectories, if not fail outright, since the static world assumption is not respected.

Figure 1.1 – Example of a static scenario convenient for a SLAM algorithm. It has no objects in motion and the background is clearly visible.

This has led to the creation of SLAMs specifically designed for dynamic environments: Dynamic SLAMs. The general approach consists in identifying the dynamic objects of the scene and masking them, *i.e.*, preventing the SLAM from mapping them. In other words, the data association step is modified to make sure that the map does not include 3D points from dynamic objects as we want to compute camera poses in the world frame. Dynamic SLAM is necessary for applications that need SLAM in highly dynamic contexts, but it is an unsolved problem in general: an example is autonomous

vehicles – safety is still a major challenge which limits the widespread use of autonomous vehicles.

A major challenge of Dynamic SLAMs is to ensure that it uses the frame of reference expected by the user, usually the Earth. In other words, a SLAM algorithm must not solve an ill-posed problem: to compute a pose with respect to a variable or unexpected frame of reference. To do so, most Dynamic SLAMs are designed under the hypothesis that the areas of the image that are part of the *motion consensus* are fixed with respect to the Earth. We can group the 3D points observed in a frame in rigid clusters according to their instantaneous motion: those fixed w.r.t. the Earth, those belonging to a moving car, etc. We define the *dominant motion* of an image as the motion of the largest group. We define *motion consensus* as the areas in a frame that correspond to the 3D points that are part of the group whose motion is the dominant one. A simpler way to formulate the Dynamic SLAM hypothesis, and the one usually seen in the literature, is "most of the image is static", but it is not rigorous.

Still, while the notion of motion consensus makes sense on a larger scale – *e.g.*, building and streets are mostly static under normal circumstances – it may be misleading within the scope of short-term input data. For instance, a camera could be in a traffic jam (fig. 1.2) where the ground is hardly visible: in this case, the SLAM may completely fail and compute nonsensical poses due to what we call a *motion consensus inversion* or MCI. We detail MCIs in section 4.2.2. The problem tackled by the SLAM is ill-posed here: the implicit frame of reference that the SLAM uses to compute poses is not the ground, but one attached to cars.

Figure 1.2 – Example of dynamic scenario difficult for a SLAM algorithm, a traffic jam. The motion consensus (the dominant motion) does not correspond to the ground but to the motion of cars.

In detail, a SLAM maps the environment as it runs; one *assumes* that most mapped parts are fixed with respect to the ground. SLAMs normally do not have a "ground detector" to make sure that the ground is used as frame of reference: this means that if too many mapped points are not fixed with respect to the ground – *e.g.*, when there are many dynamic objects –, the SLAM fails. The implicit

frame of reference in our traffic jam example is an unpredictable barycenter between the frames of all objects respecting the misleading motion consensus... which may be random cars in a traffic jam. A different situation that may lead to variable frames of reference is when objects are deformable. SLAM with deformable objects, even if they are not moving, is extremely difficult and research on this topic is at a very early stage [56].

Figure 1.3 – Example of very difficult dynamic scenario for a SLAM algorithm. The frame of reference that the user wants to use for localization is not obvious: it could be the Earth or the ship itself. There is no clear consensus in the image: the sky, the sea, the Earth, and the ship move differently.

Another complication is that the frame of reference may not even be the ground. Assuming that a SLAM can always locate the user in relation to the ground, it will only be acceptable if our goal is indeed to do so. For instance, in a large transportation as a ship or an airplane, the user's goal may be to locate the camera within the transportation and not with respect to the ground (the Earth). Even if the SLAM is not affected by consensus inversions, it is still a failure if it computes the pose of the camera with respect to the ground. This problem may not appear within a closed environment like a plane, but it is serious in open environments like on of a ship. Figure 1.3 shows such an example. First, the frame of reference that the user wants to use for localization is not obvious: it could be the Earth or the ship itself. Additionally, there is no clear consensus in the image: the sky, the sea, the Earth, and the ship move differently, and that is without counting people moving on the ship itself. A related difficulty is the use of IMUs: they cannot be used for SLAM w.r.t. non-inertial frame of reference – *e.g.*, accelerations would be computed w.r.t. to the Earth and not the ship. Thus, if our goal is to locate the camera within the ship, a SLAM must ignore what is not part of the ship:

the sky, the sea, moving people, etc., which a highly non-trivial task. How could a SLAM guess the correct frame of reference under these conditions?

Therefore, to work properly, SLAM algorithms need to analyze the scene at a higher level of abstraction than the motion consensus so as to identify what should be mapped or not. A possible approach is to use an algorithm to segment objects that might move, and then an heuristic method to decide if this object is in fact moving or not w.r.t. the frame of reference expected by the user. Figure 1.4 shows an example of instance segmentation, which consists in assigning a label to every pixel of image: background, car, person, etc.

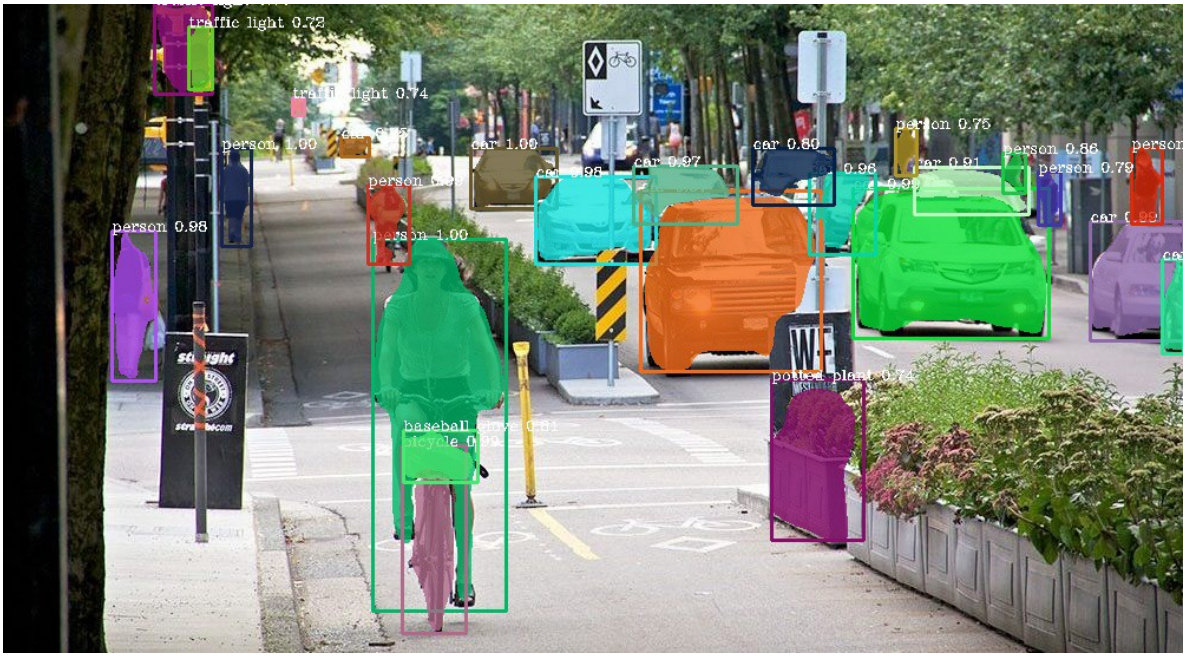


Figure 1.4 – Example of instance segmentation. Instance segmentation identifies every instance of every class separately.

Hence the problem of Dynamic SLAM is still far from solved. Learned algorithms recently enabled major progress but it has not solved all problems: a major drawback is the need for training data, which is costly to obtain. Additionally, the naive use of learning-based segmentation to mask parts of the image that *might* move can have an adverse effect. For instance, if the camera is in a parking, removing all cars from the image may leave it empty depending on where the camera is: this obviously makes the SLAM fail.

1.2 Context of the thesis

We carried out this thesis entirely at CEA LIST, *Laboratory for Integration of Systems and Technology*, from January 2019 to December 2021. The LIST is made of several sub-laboratories, including the LVML, *Laboratory for Vision, Modeling and Localization*, and LASTI, *Laboratory for Semantic Analysis of Text and Images*. The LVML has an history of over ten years of working with SLAM systems and their related technologies, while the LASTI has a likewise long history of working with automated video and image analysis. We did most of the research at LVML, where the advisors of this thesis work, but we had regular interactions with the LASTI, where the supervisor of this thesis works. The current SLAM challenges require both scene understanding and in-depth geometrical SLAM knowledge, so the partnership between both came naturally.

As explained before, SLAM is primarily about locating a system when there is no prior map of the environment. But once we have a base SLAM method, it is valuable to developed specialized methods according to the specific intended use of the SLAM. In particular, the LVML has developed several specialized SLAM technologies for industrial use and has today a multi-camera embedded SLAM able that combines several cameras, IMUs and magnetometers.

In industrial settings, we often have specific information on object geometry, namely CAD (Computer-Aided Design) models. CAD models have the benefit of containing accurate 3D information on object geometry. *Constrained SLAM* is precisely about leveraging this information [95, 96]. The major use case is object tracking, since we already know what to look for. It consists in aligning a 3D model on the image, as illustrated in fig. 1.5. A tracked object can be used as an additional source of information for the SLAM: since we know its exact dimensions, we can use it to improve SLAM accuracy and fix scaling issues. We may also need a precise pose of the object, *e.g.*, for bin-picking applications. And if we are certain that the object is fixed w.r.t. the frame of reference we need, we can use it as a reliable cue to find what parts of the scenario a SLAM algorithm can map without the risk of motion consensus inversions. The main drawback of Constrained SLAM algorithms is the need for 3D models a priori, which is not an issue in specific contexts where we control what objects are present.

Figure 1.5 – Constrained SLAM: the SLAM algorithm aligns CAD models on corresponding objects to improve accuracy [96].

Another specialty of the LVML is Augmented Reality (AR) for industrial use [7, 40, 13, 41], as illustrated in fig. 1.6 and fig. 1.7. AR adds visual information to the objects of interest.

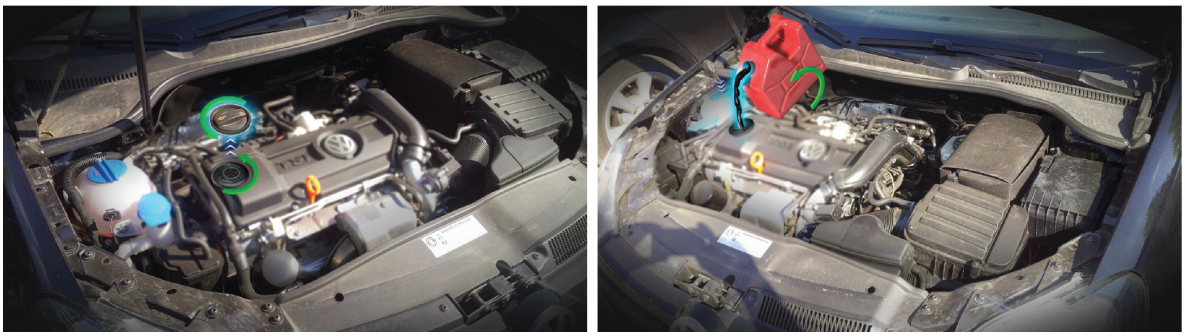


Figure 1.6 – Application of Augmented Reality to give repair and maintenance instruction in the automotive applications [41].

Figure 1.7 – Augmented Reality used to visualize custom vehicle furnishing [41].

The main idea is to replace or superimpose real objects with virtual ones. *E.g.*, Augmented Reality helps workers in industrial maintenance and quality assurance as it makes checking vehicle parts a simple task when combined with Constrained SLAM. The LVML also works on the counterpart of Augmented Reality, Diminished Reality (DR), and a related research topic, specularities – *i.e.*, mirror-like reflections of light sources. Diminished Reality consists in removing parts of an image with inpainting [22, 14], which is the technique of coherently filling a missing region of an image with respect to the rest of it. Inpainting is a challenging task when there are specularities, especially when there are multiple light sources. We illustrate DR in fig. 1.8.

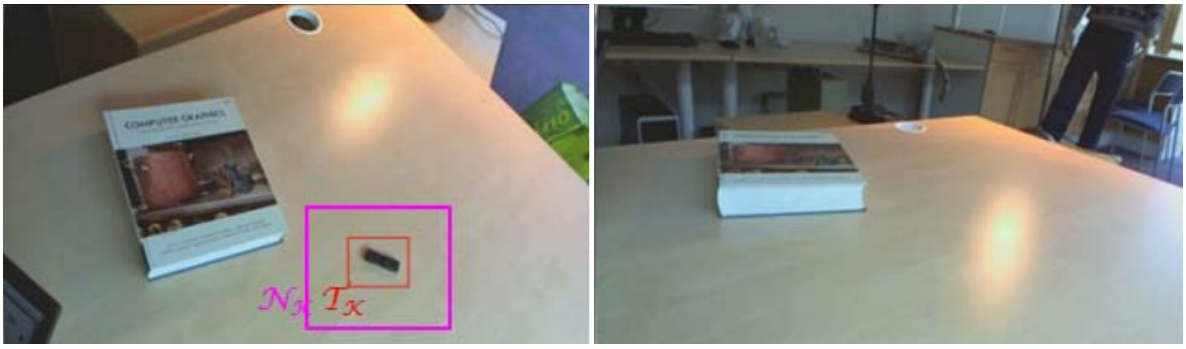


Figure 1.8 – Removal of an object across different frames using diminished reality in the presence of specularities [79].

In the last few years, with the rapid development of Deep Learning, the LVML has started researching learned methods in SLAM contexts. This includes straightforward uses (*e.g.*, object detection) as well as less obvious questions as explainability, which is important for industrial applications.

Having methods to support scenarios with known objects (including humans) and specularities, the technologies at the LVML did not support highly dynamic scenarios when this thesis started. Since a Dynamic SLAM must be robust in dynamic environments to be used in industrial contexts, research on it is necessary. Learned approaches appeared as a potential way to achieve a Robust Dynamic SLAM.

The starting point of this thesis is the idea that to make a robust Dynamic SLAM, we do not need to create whole new SLAMs but instead target the weaknesses of the existing ones, possibly with learning. This would later translate to automatically learning what and when dynamic objects positively or negatively affect SLAMs, which are the core contributions of this thesis. This thesis was the opportunity to construct strong foundations for Dynamic SLAM, which the laboratory will build upon in the future to both continue the research and industrialize it.

1.3 Contributions

Since Dynamic SLAM is a valuable problem to solve, we decided to tackle the problem of creating a robust Dynamic SLAM algorithm – *i.e.*, a SLAM algorithm that always uses the correct frame of reference. Since cameras are present in most systems that need SLAM, we focus our research on Visual SLAM with an emphasis on monocular cameras, as they are low-cost and the most popular ones.

Our goal is to make a Dynamic SLAM that is agnostic with respect to the chosen frame of reference. We detail in this section our approach and the underlying choices, the hypothesis of this thesis, and detail our contributions.

1.3.1 General approach

Our approach can be summarized as continuously identifying all parts of a frame that can be safely mapped and only them, in order to ensure that the SLAM only processes objects that are part of the correct frame of reference. We split this challenge in two objectives: identifying *what* objects to mask and *when* they should be masked, all to maximize SLAM performance.

Rationale of the general approach

For a Dynamic SLAM to be agnostic w.r.t. the choice of frame of reference, we first considered a classical approach where we would identify geometrical criteria to identify parts of the image that belong to the user's expected frame of reference. However, there is a major obstacle: as these criteria are manually defined, it would be very difficult to automatically adjust them to unknown contexts. And this is assuming that it is possible to develop such criteria in the first place: it appears that to handle the challenges previously mentioned – motion consensus inversion and excessive masking – we need long-term understanding of the environment, which is hardly possible with classical algorithms. Thus, we dropped classical approaches.

A second possibility would be to make a Dynamic SLAM algorithm general enough to support any scenario. However, this implies proposing a SLAM approach that is not ill-posed, *i.e.*, a SLAM that can understand *on-the-fly* what frame of reference the user needs and then correctly identify it. This would require a high-level intelligence that is far beyond the scope of Dynamic SLAMs, and extremely difficult to achieve.

A last possibility is automatic two-step adaptation. Instead of creating a general Dynamic SLAM, we automate the adaptation of the Dynamic SLAM to any environment. In a training step, the Dynamic SLAM learns to identify the parts of the image that are fixed w.r.t. the expected frame of reference in the training context – the SLAM is *fine-tuned* to the given context. Later, at runtime, the Dynamic SLAM automatically recognizes what parts of the image should be masked or not, ensuring that the correct frame of reference is used at any time if it runs in a similar context. Thanks to reducing the scope the SLAM, the adaptability approach is within reach of a PhD thesis – unlike the general approach –, so we select it.

Having opted for automatic two-step adaptation, the question now is how can a SLAM algorithm automatically learn what and when to mask objects. There are two methods: unsupervised and self-supervised learning. Supervised training depends on annotated data, which may be costly (both in time and money) and requires a high level of expertise depending on what has to be annotated. *Self*-supervised training consists in supervised training but with automatically annotated data. Unsupervised learning has the benefit of not needing annotated data at the cost of having a complex learning scheme – especially the loss function – and no explicit control over the expected output. Learning what and when to mask objects is already a major challenge, so we preferred to have control over data annotation and to keep the learning scheme as simple as possible. In other words, we decided to tackle the SLAM challenge with self-supervision.

Choice of the type of SLAM

We group SLAMs in two categories: model-based and learned. Model-based SLAM corresponds to geometrical methods and is split in feature-based SLAMs [75], that compute image features (keypoints) then track/map them, and direct SLAMs [39], that process full images at once. Learned SLAMs [106] are neural network that compute camera poses by processing images in a black-box fashion.

We discuss these choices in depth in section 2.2.1, but the main idea is that filtering features in feature-based SLAM is a simple task that can clearly be separated from the rest of the SLAM, with our work being about what and when to filter features. Direct SLAMs expect a dense input (images), so filtering parts of it breaks this basic assumption. Since filtering dynamic objects is significantly more difficult in direct SLAMs and since our challenge is to learn what and when to mask dynamic objects in the first place, there is no reason to use them. Regarding learned SLAMs, the main drawbacks are generalization to unknown environments and the lack of a usable map – it is contained within the SLAM model. For these reasons, we chose to work on feature-based SLAMs. Figure 1.9 shows how a feature-based Dynamic SLAM works. It consists in filtering features right after they are detected based on an external input: masks that indicate the areas of the image that can be safely mapped.

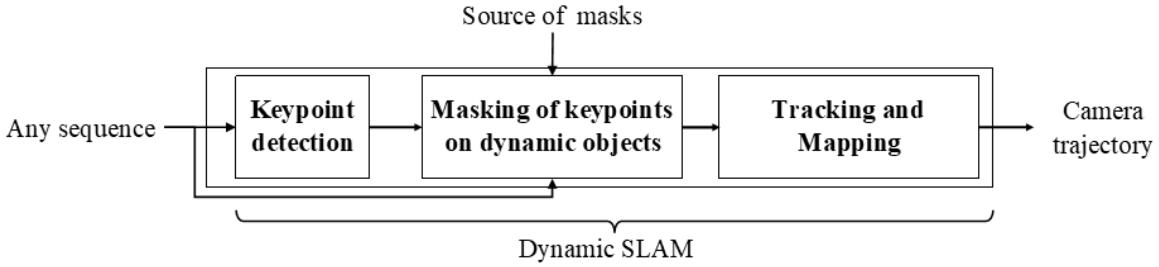


Figure 1.9 – Feature-based Visual Dynamic SLAM. It consists in filtering all features based on masks, which indicate what parts of the image can be safely mapped, *i.e.*, are fixed w.r.t. to the frame of reference expected by the user.

1.3.2 Hypothesis of our thesis

In this thesis we make the following assumptions:

1. We focus only on pure feature-based Visual SLAM. As explained before, Visual SLAM is valuable as cameras are almost omnipresent on systems that need SLAM. Other sensors (LiDARs, sonars, IMUs...) have their own limitations and can hardly replace cameras. Feature-based SLAMs are more convenient than other types (direct/learned). And in any case, we expect SLAMs that fuse output from different sensors to perform better if the output from visual sensors is correctly processed. For simplicity, we will refer to Visual SLAM as simply SLAM unless otherwise noted.

2. We focus on Dynamic SLAM approaches that are generic in terms of context and in terms of underlying SLAM engine. For this reason, we research learned approaches. Self-supervision appears as an efficient way to limit the cost of data annotation.

3. We assume that that the deformability of objects can safely be ignored. Dynamic SLAM with deformable objects generalizes Dynamic SLAM, which is already an unsolved problem. The only deformable objects that appear in our experiments are people and our methods correctly segment them, so for practical purposes we can consider that all objects are rigid.

1.3.3 Main contributions

We can group our works in three major contribution: automatically learning of *what* to mask in dynamic scenarios, automatically learning of *when* to mask dynamic objects, and databases and metrics to evaluate and measure the robustness of Dynamic SLAMs. The overall goal is to have a Robust Dynamic SLAM, so we first ensure that we have proper datasets and metrics, learn what objects to mask, then finally when to mask them to maximize SLAM performance.

Automatically learning of *what* to mask in dynamic scenarios. The main idea is that semantic segmentation is the key to make Dynamic SLAMs robust, in particular to motion consensus inversions caused by moving objects in difficult scenarios. Geometry-based approaches tend to fail under motion consensus. End-to-end learned approaches are limited to their training data and having to learn what objects are dynamic further complicates training. Thus, we propose a Dynamic SLAM approach that automatically learns to segment new dynamic objects based on outliers, *i.e.*, image features that do not fit the static world model. Geometric SLAMs typically discard outliers, while we use them as a source of information to improve the very SLAM that generated them. This makes our approach self-supervised and able to adapt to new environments at a low cost. This contribution corresponds to the publications:

1. *De SLAM Robuste à SLAM Dynamique par Auto-apprentissage d'Outliers* [10] published at RFIAP 2020
2. *Learning to Segment Dynamic Objects using SLAM Outliers* [11] published at ICPR 2021

Automatically learning of *when* to mask dynamic objects. The main idea is that while the ability of masking potentially dynamic objects that may lower SLAM performance is necessary, it is not enough. An object may move or not – *e.g.*, a car may be parked or moving –, so masking it when it is not moving would needlessly lower the performance of the SLAM and even make it fail. A major difficulty is that geometry-based approaches are unreliable in difficult scenarios since they depend on instantaneous motion detection, which cannot always be computed. Thus, we propose a completely different paradigm: self-supervised Temporal Masking. It consists in automatically annotating video sequences with per frame, per semantic class masking decisions then learning the decisions with a memory-based network. The first insight here is to annotate sequences with a method that directly maximizes SLAM performance, without priors on geometry – like the fact that masking dynamic objects improves performance, which is in fact a result that emerges from learning. The second insight is to infer masking decisions independently from the SLAM: this makes our approach independent from the choice of the SLAM algorithm and ensures that the SLAM does not interfere with the computation of masking decisions at runtime. This contribution corresponds to the publications:

1. *Procédé de localisation et cartographie simultanées intégrant un masquage temporel auto-supervisé et modèle d'apprentissage automatique pour générer un tel masquage* [12]. Patent filed in France on December 03, 2021. Number 2112893 – V/Ref.: BD21941 CM – N/Ref.: 073214 FR PHA/BLR
2. *Dynamic SLAM With Self-Supervised Temporal Masking* (title may change), planned for submission.

Databases and metrics to evaluate and measure the robustness of Dynamic SLAMs. In the early phases of this thesis, we observed that SLAM datasets currently available almost never include sequences that cause SLAM failures due to excessive masking or motion consensus inversions. Moreover, popular SLAM metrics that focus on accuracy are biased: in very difficult scenarios, where early SLAM failure are possible, an accurate trajectory does not mean that the SLAM performed well. This makes comparisons complex: what is better, processing 100% of a sequence with an accuracy of 10cm, or 10% of a sequence with an accuracy of 1cm? Therefore, we propose our own datasets and metrics to fill this gap in the current literature. The dataset and metric contributions have evolved over time and are present in all our works [10, 11, 12].

1.4 Structure of the document

The rest of the document is organized as follow:

Chapter 2 presents SLAMs and Dynamic SLAMs, including algorithms from the State of the Art. Section 2.1 presents the general SLAM algorithm and a brief history, sensors, and the most common SLAM paradigms. Section 2.2 focuses on Dynamic SLAM approaches.

Chapter 3 studies the relation between keypoints – SLAM features – and Dynamic SLAM. Section 3.1 introduces the importance of keypoints in feature-based SLAMs and how to filter them. Section 3.2 studies how many features should be used. Section 3.3 clarifies the relation between a keypoint detector performance, its repeatability, and SLAM performance. Section 3.4 focuses on SLAM inliers and outliers and how they can be used for object detection. Section 3.5 exposes how important the temporal aspect is for an efficient temporal masking.

Chapter 4 focuses on measuring SLAM robustness and corresponds to our contribution *Databases and metrics to evaluate and measure the robustness of Dynamic SLAMs*. We first present in section 4.2 the general aspects of SLAM robustness and the specific difficulties of Dynamic SLAMs. Then we present the currently used metrics/datasets and their limitations in section 4.3. After having discussed the limitations of the current landscape of metrics and datasets, we present our own metrics in section 4.4 and our own datasets in section 4.5.

Chapter 5 focuses on the question of what to mask in a Dynamic SLAM and corresponds to our contribution *Automatically learning of what to mask in dynamic scenarios*. We first present a novel method based on SLAM outliers in section 5.2. We then show experiments proving the interest of our method and how it makes learning segmentation masks easier in section 5.3.

Chapter 6 focuses on the question of when to mask objects in a Dynamic SLAM and corresponds to our contribution *Automatically learning of when to mask dynamic objects*. We first present a SLAM pipeline including temporal masking in section 6.2. Then we present the core of our approach, the temporal masking network in section 6.3 and the automatic annotation method in section 6.4. We then present the experiments in section 6.5. **Section 6.6** is a complementary research work on weak supervision. We present a novel network architecture in section 6.6.1 and our experiments in section 6.6.2.

Chapter 7 includes the conclusion and perspectives of this thesis.

Chapter 2

SLAMs and Dynamic SLAMs

2.1 SLAM: Simultaneous Localization and Mapping

2.1.1 Problem Formulation

SLAM – Simultaneous Localization and Mapping – is the computational problem of constructing a map of an environment while locating an agent (*e.g.*, a robot) within it at the same time [17, 74, 40]. This appears as a chicken-egg problem: a map is needed to localize an agent within it, and it is necessary to know the position of an agent to build a map of the environment it perceives. [17] is an extended review on the state of SLAM algorithms in 2016 and is still largely applicable. The critical question – is SLAM solved? – has a clear answer: there are working solutions in specific contexts, but not in the general case. Most importantly, SLAM algorithms are aimed at real-time applications. This means that they are constrained by the available resources, in particular sensors and computational power.

Note that SLAM is distinct from other related problems: SfM (Structure from Motion), odometry, and relocalization. *SfM* [99] is a photogrammetric imaging technique. The main goal of SfM is to estimate three-dimensional structures from two-dimensional image sequences. This has real-world applications as reconstructing monuments in 3D – also known as photogrammetry. However, the position of the camera taking the images is not a required output but only an optional intermediate result. *Odometry* is the use of data from sensors to estimate changes in pose over time. In this case map construction is not necessary. *Relocalization* algorithms locate an agent within a known map. Overall, SLAM algorithms have common points with SfM / odometry / relocalization algorithms but are targeted at real-time use cases where neither localization or mapping information are available but where both are needed.

2.1.2 Sensors

Sensors presents in SLAM systems are classified in three categories: proprioceptive, exteroceptive and absolute positioning systems. Proprioceptive sensors measure the internal state of the agent; exteroceptive sensors observe the environment; absolute positioning systems directly give the agent's pose.

Proprioceptive sensors

Proprioceptive sensors measure the internal state of the agent and measure battery level, wheel positions, joint angles, etc. For SLAM purposes, we focus on sensors related to the pose or motion of the agent. A major advantage of these sensors is that they do not depend on external factors – unlike, for instance, a standard RGB camera that would be useless in a dark environment.

The primary use of such sensors is for dead reckoning: the process of estimating a pose (3D position + 3D orientation) based on the previous pose and the sensor output. However, dead reckoning accumulates errors over time, making it unusable in the long term. These sensors can be used for odometry but cannot be used alone for SLAM: mapping the environment requires first observing it, which is by definition impossible with proprioceptive sensors.

Wheel encoders return wheel angular motion. While obtaining the agent pose is simple rigid body motion, it is not possible to account for slippage or, more generally, any unexpected motion (*e.g.*, hitting an obstacle).

Joint angle encoders return the angles between joints and can be used to compute the pose of articulated parts of the agent with respect to the agent's main body.

Inertial Measurement Units (IMU) are devices that measure linear acceleration using accelerometers and angular acceleration using gyroscopes. By integrating both the linear and angular accelerations, it is possible to obtain the agent trajectory. Due to the double linear acceleration integration, the estimated pose drifts $\propto$ time squared, and can keep sub-meter accuracy only over very short lengths of time – a few seconds for low-cost IMUs. In practice, IMUs are used jointly with other sensors (*e.g.*, cameras). IMUs used in critical applications: airplanes, submarines, missiles, satellites, etc., can remain accurate over longer time spans but are correspondingly more expensive.

Exteroceptive sensors

Exteroceptive sensors are used for the observation of the environment: sonars, lasers, cameras, etc. Compared to proprioceptive sensors, they are not used for dead reckoning but as a source of features that, compared over time, makes it possible to do SLAM. Interpreting these features is the core of SLAM systems.

Vision sensors – *i.e.*, cameras – are passive sensors that capture images of the environment, similarly to a human eye. They work in different wavelengths (visible, infrared, multispectral), can have a single input (monocular) or multiple inputs (stereo, sometimes more) and different lens properties (pinhole, fisheye...). Cameras are systems that project 3D points of the environment on a 2D image; a Visual SLAM computes from 2D images where the camera that took them was. The underlying theory of modern SLAMs was first defined in 2003 under the name Multiple View Geometry [46]. Multiview geometry requires an accurate camera calibration, which quantifies lens distortion, focal length, sensor scale, etc. and is by itself a research topic. Figure 2.1 shows a stereo + depth + IMU camera.

Figure 2.1 – Example of stereo + depth camera with an embedded IMU, a MYNT EYE D1000-120, the camera we used to create our datasets. Image from the MYNT EYE company website.

The main advantages of cameras are that images are convenient for scene understanding/object detection and not only SLAM, and cameras are low-cost and almost omnipresent. They are also useful in contexts were active sensors (LiDARs, Sonars) are not authorized, like medical or military contexts. Drawbacks include poor image quality under lack of light or certain weather conditions (snow, rain, dust) as well as hardware specifications that may be limiting for real-time systems (frame rate, image resolution, shutter speed...).

LiDARs (Light Detection and Ranging) are active sensors that emit laser beams at fixed angular steps. They measure the reflected pulses to estimate the distance between the sensor and the object. Beams often operate in the near-infrared spectrum (thus invisible for humans) and work in ranges

usually from a few meters to a hundred meters. The laser beams may be emitted within a plane, generating a 2D point cloud, or within a cone, generating a 3D point cloud. LiDARs are sensors of choice for autonomous vehicles. However, as for cameras, they may be severely affected by dust, snow and rain. Reflective surfaces are also a problem for LiDARs: they may not be detected at all or generate virtual objects. For instance, LiDARs assume that laser beams move in a linear fashion, so a mirrored image of an object (a virtual object) appears as a real object *behind* the mirror. Figure 2.2 shows examples of LiDARs.

Figure 2.2 – Example of LiDARs from the Velodyne family, used on autonomous vehicles. Image from Wikipedia.

Radars (Radio detection and ranging) are active sensors that emit electromagnetic waves in order to estimate the position of objects with respect to the sensor. Like LiDARs, they generate point clouds. They are used to detect aircrafts, ships, missiles, etc. and rarely for SLAM.

Sonars (sound navigation and ranging) are like radars but use sound waves instead of electromagnetic waves. They are often used to locate objects underwater and more rarely for underwater SLAM.

IR Depth sensors return the distance of objects with respect to the sensor. It may be a simple IR beam (giving the distance of the first object along a line) or an IR emitter / IR camera pair. In this case, it generates a depth map, *i.e.*, a grayscale image in which the color is proportional to the depth. Figure 2.3 shows an RGB+depth camera.

Figure 2.3 – Example of RGB+depth camera, a Kinect, often used in SLAM benchmarks. Image from Wikipedia.

Absolute positioning systems

Absolute positioning systems directly give the pose of the agent. There are global and local systems.

Global navigation satellite systems (GNSS) emit signals to satellites to estimate the global position of the emitter based on signal propagation time and the position of the satellites. The most well-known GNSS system is USA's Global Positioning System (GPS). GPS may be inaccurate and consumer-grade ones (*e.g.*, in a smartphone) have an accuracy rarely better than 1m. Moreover, they work poorly in places where satellites are hard to reach, especially underground and in tunnels. They

are nonetheless an almost ubiquitous sensor in autonomous vehicles and used with other sensors that are more accurate at short ranges, like LiDARs.

Motion capture systems, also known as *mocap* systems, compute the pose of known markers within an environment. A popular application is to capture the motion of the human body for movies/game with IR-reflective markers and IR active cameras – IR cameras coupled with an IR emitter. While mocap systems can be very accurate (<1mm), a major drawback is the need to install the capture system beforehand. Figure 2.4 shows what a mocap setup looks like. Thus, they cannot be used to locate an agent without preparing the environment. In practice, for SLAM, motion capture systems are used to compute accurate ground truths when creating datasets.

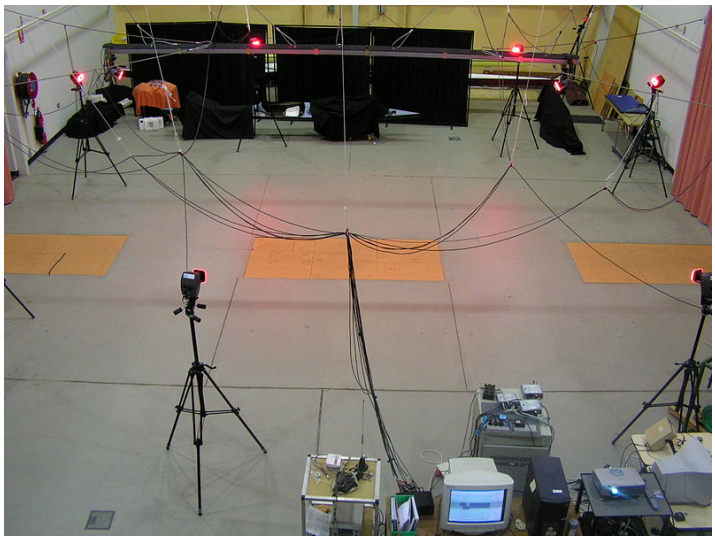


Figure 2.4 – Example of motion capture setup. Image from Wikipedia.

2.1.3 Brief history SLAMs

Cadena *et al*. [17] presents detailed insights on the history of SLAMs until 2016, and we use it as the main source for this section. SLAM systems are not new: Leonard and Durrant-Whyte [58] proposed in 1991 a multi-sonar SLAM. Interestingly, they already talk about the chicken egg problem of localization and mapping, which is still today what defines a SLAM: an accurate location is needed for mapping, and an accurate map is needed for location. The acronym *SLAM* itself was coined in Durrant-Whyte *et al*. in 1996 [33].

SLAM research from the 90s until 2004 has seen the development of probabilistic real-time SLAM approaches based on Extended Kalman Filters (EKF), Rao-Blackwellized particle filters or maximum likelihood estimation. SLAMs became world-famous with the self-driving STANLEY and JUNIOR cars, led by Sebastian Thrun: they won the DARPA Grand Challenge in 2005.

The period from 2004 to 2015 in SLAM research was focused on making SLAMs more efficient, consistent, and better understood from an analytical point of view, *e.g*., with studies on convergence and observability [32]. It also includes the advent of newer and cheaper sensors like the Kinect in 2010 and the corresponding RGB-D SLAM method [51]. Cheaper sensors are also useful for mass-market SLAM, like autonomous robotic vacuum cleaners. The first Roomba to use visual SLAM (seventh generation, 900 series) appeared in 2015.

The period from 2016 until today has seen strong progress in open-sourcing and intelligent SLAMs. The increasing open-sourcing of SLAM engines like ORB-SLAM [75] in 2015 made it possible for other researchers to use known SLAM engines as a backend and focus on more complex questions like SLAM in dynamic environments. It also made repeating experiments and comparing SLAM methods

easier. With the progress of GPUs (Graphical Processing Units) and deep learning methods, SLAMs started to integrate learned modules. Those can be steps like feature generation [30], object masking in Dynamic SLAMs [8] or the whole SLAM, *i.e.*, end-to-end deep SLAMs [106].

There are today a number of open problems. Robustness, both software-wise (recovery after a SLAM failure) and hardware-wise (failing sensors) is challenging to achieve. The dynamic and/or deformable aspect of objects or of the environment are complex issue to tackle. Moreover, the fact that the temporality of changes is variable – seasonal changes occur gradually over months, objects may move suddenly at unpredictable times, object deformations may be temporary or permanent – makes the problem even more difficult. Other challenges include collaborative SLAM (multiples robots working together and sharing maps), computational optimization (on-device intelligent SLAMs) and scene understanding, especially when SLAM is used for path or action planning.

Note that modern SLAM algorithms as ORB-SLAM are in fact not strictly limited to a static world, but rather to an *almost* static world. The meaning of "almost static" depends on the SLAM, but it would typically include environments where moving objects are only seen from afar, covering only a fraction of the image. However, this improved robustness is not enough to handle highly dynamic environments, unlike Dynamic SLAMs.

The key takeaway is that SLAMs have evolved for over thirty years and will continue to do so towards even more intelligent, fast and efficient systems.

2.1.4 Model-based SLAM

Typical model-based SLAMs are constructed in two parts: the frontend and the backend. The frontend interprets incoming sensor data through feature extraction. Newly extracted features are then associated with features previously extracted and processed. The backend uses optimization techniques to estimate the map and the agent pose based on the associated data. In general, short-term data association is used to compute incremental changes while long-term data association is useful for loop closures (global trajectory optimizations) or relocalization after a tracking loss.

The focus of this thesis being Visual SLAMs, we present camera-based SLAMs. Modern SLAMs as ORB-SLAM [75] have three key steps: initialization, tracking and mapping. Initialization consists in computing an initial map – a 3D reconstruction of the environment – and an initial pose from a set of images. SLAM algorithms perform tracking and mapping continuously: they compute in turns respectively the new camera pose and update the map. Tracking consists in computing 2D-3D correspondences between the map and the image in order to estimate camera pose, essentially solving the Perspective-n-Point (PnP) problem. Mapping consists in updating the map once the new pose is known, correcting the position of existing 3D points, or creating new ones that correspond to new 2D features. Figure 2.5 shows the general structure of model-based SLAMs.

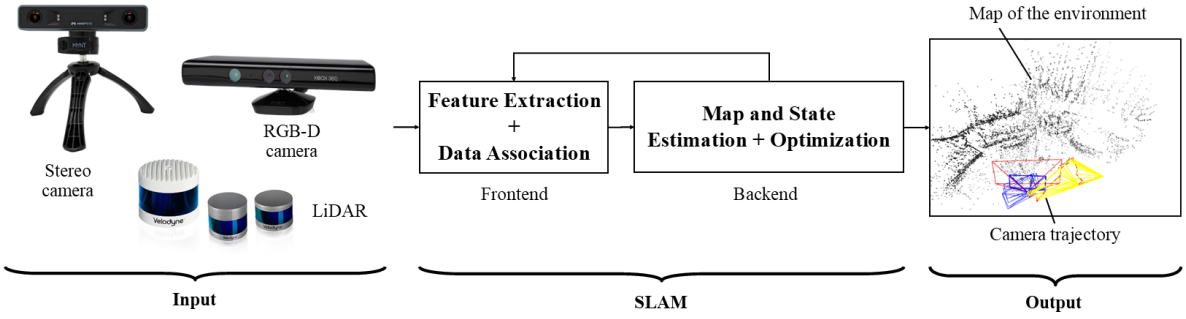


Figure 2.5 – General structure of a model-based SLAM pipeline: sensor input, frontend + backend, map/trajectory output.

There are two schools of thought on the frontend of Visual SLAMs. The first one is to compute

keypoints, or sparse points representing remarkable elements in the image as corners. The second one does not actually extract features from the image: the image *is* a set of features (*i.e.*, pixel values) and used it directly for pose computation. Figure 2.6 illustrates the difference between direct and feature-based SLAM.

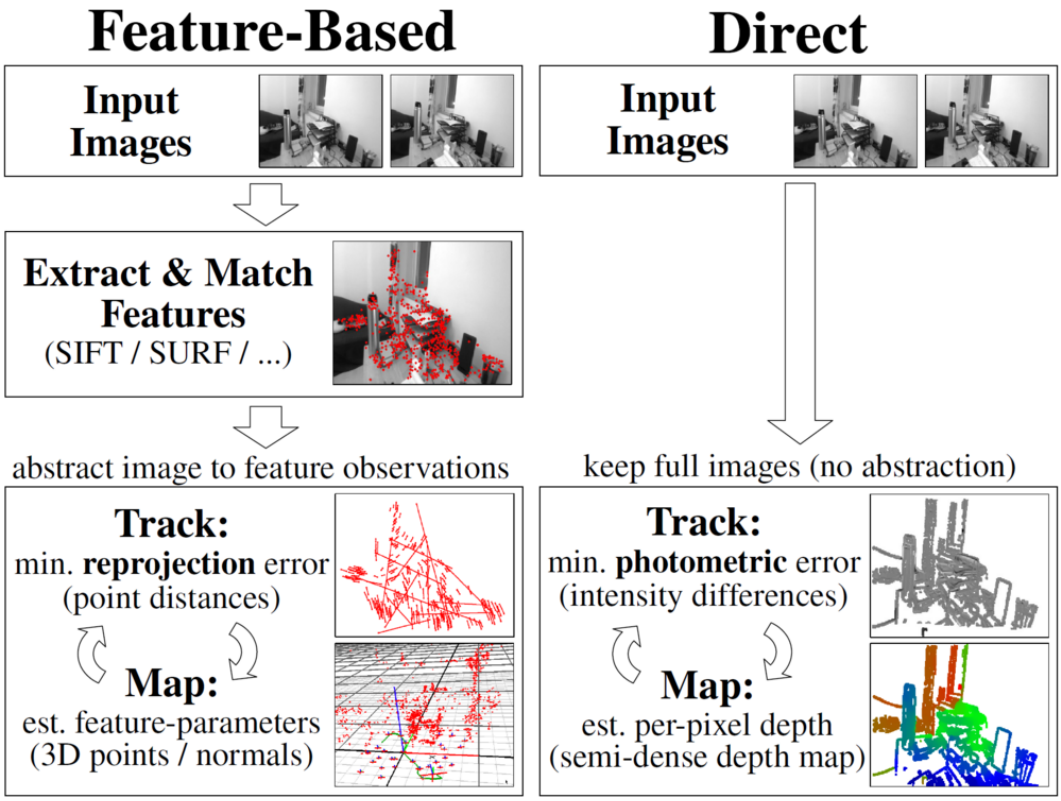


Figure 2.6 – Illustration of feature-based and direct SLAMs [36].

Feature-based methods

Feature-based methods add a feature extraction step after an image is received by the SLAM. The rationale is that the full set of pixels from an image – which can amount to several millions – is difficult to interpret in real-time and contains largely redundant information for SLAM purposes. A major benefit of having a feature extraction step is that it makes the SLAM algorithm lighter at runtime and more modular. Modularity makes the replacement of the feature extraction module by a newer one easier. Figure 2.7 shows the fundamental structure of a feature-based SLAM: the keypoints, *i.e.*, feature, extraction step and the Tracking Mapping step.

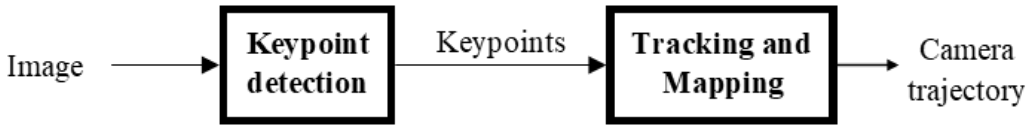


Figure 2.7 – The two main modules of a feature-based SLAM: keypoint generation and Tracking/Mapping.

MonoSLAM [28] (2007) is the first work to consider image features as landmarks to build a cor-

respondingly sparse map. Pose computation then relies on an Extended Kalman Filter (EKF), an iterative process in a Bayesian framework that continuously updates a full state vector containing the agent pose as well as the features position.

Also in 2007, PTAM [54] presented the first Bundle Adjustment (BA) method. Bundle Adjustment consists in minimizing the *reprojection error*: to compute the camera pose and 3D map that minimizes the error between the position of observed 2D features and their estimated 2D position if (re)projected according to the 6D pose and 3D map being optimized. PTAM and later SLAM algorithms successfully tackled this nonlinear optimization problem with Levenberg–Marquardt techniques [54, 75]. As bundle adjustment can be too slow for real-time SLAM, modern SLAMs as ORB-SLAM [75] restrict pose/map optimization to features in *keyframes*, or frames that are heuristically considered to contain novel information compared to previous frames. Note that although bundle adjustment is the *de facto* standard today, research on EKF-based methods is still active [43].

Focus on ORB-SLAM 2. Since our works in later chapters are based on ORB-SLAM 2, we give more details about it. Please note that this is for clarity purposes, since we generally consider SLAMs as black boxes in our works. The monocular SLAM *ORB-SLAM* [75] and especially its extension to stereo / RGB-D sequences *ORB-SLAM 2* [76] are popular in SLAM research as they are open-source, modular and state-of-the-art methods among general SLAM algorithms (Dynamic SLAMs excluded). Figure 2.8 shows the structure of the base ORB-SLAM. Its main modules are **Tracking** – feature extraction, data association, and pose estimation –, **Mapping** – graph-based map creation and optimization –, and optionally **Loop Closing** – long-term global optimization when the camera trajectory loops – and **Relocalization** in case of tracking loss. After initialization, data association consists in matching newly detected features with the 3D map previously built by the SLAM.

Figure 2.9 shows that the main difference between both versions of ORB-SLAM is that the data association step now includes merging data from different sensors, a pair of cameras for stereo and a combined camera + depth sensor for RGB-D, through subprocesses called resp. rectification and registration. Rectification is the projection of left/right RGB images on a common plane, which is necessary to find common features between the images. Registration means matching an RGB image to a depth image in order to know the depth of every RGB pixel. In all cases, it is necessary to calibrate the camera [46].

Direct methods

Direct methods directly use an image as input to the pose/map optimization process. The general approach is to maximize photometric consistency, as in photogrammetry tasks. A key difference with feature-based methods is that the feature extraction step does not exist. In practice, this has the advantage of making the creation of dense reconstructions easier and the SLAM more robust to poorly textured environments, where feature extraction is difficult. On the other hand, direct SLAMs are more sensitive to light changes. The first direct SLAMs appeared in 2007 [88]. Popular approaches today include LSD-SLAM [36], DSO [35] and its evolution LDSO [39]. Note that the concepts of feature-based/direct and map density are separate. Namely, DSO minimizes photometric error, so it is direct, but generates a sparse map.

2.1.5 Learning-Based SLAM

Learning-based SLAM is completely different from model-based SLAM in the sense that the module that transforms a series of images to a trajectory is not crafted but learned from input sequences. In particular, *deep networks* using Convolutional Neural Networks (CNN) or Recurrent Neural Networks have reached the state-of-the-art for certain relocalization or odometry tasks. A major advantage of learning-based systems is that they are end-to-end, removing the need for intermediate steps. However, this makes the design and interpretation of such systems more difficult.

Another major difference of learned SLAMs is that the map is typically not available: the trained model has an internal representation of the environment, but it cannot be easily interpreted. For

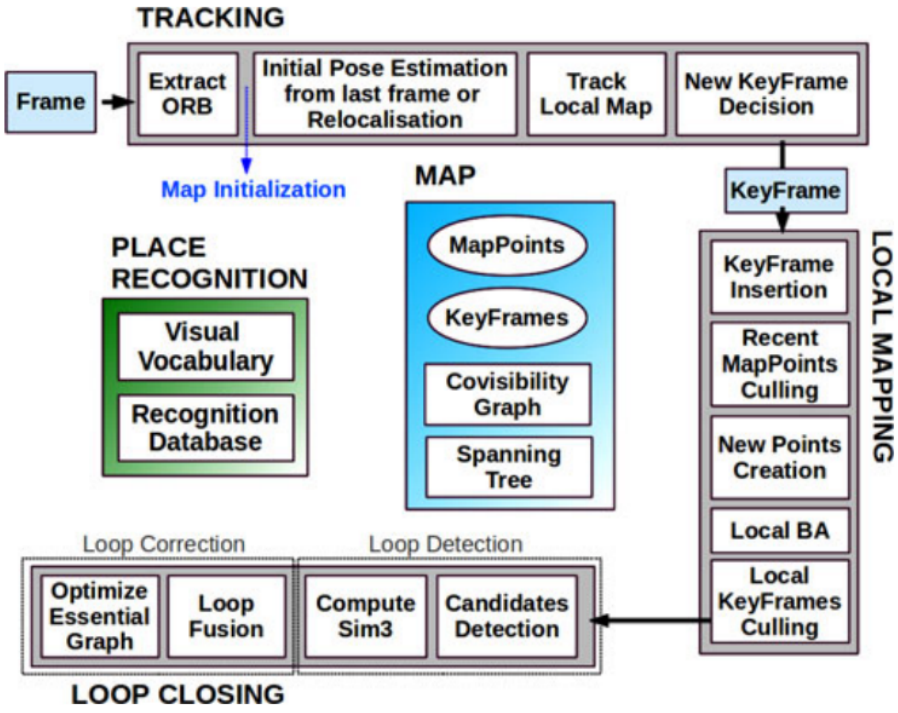


Figure 2.8 – ORB-SLAM architecture [75]. The initialization, feature detection, tracking and mapping steps are clearly separated. It supports only monocular sequences.

this reason, it is an abuse of language to call trained models used for *odometry* as learning-based *SLAM*. On the other hand, these models often also estimate image depth, which can be used for map reconstruction. In practice, these learned models are called both learned odometry and learned SLAM.

Learned Visual Odometry

A milestone in learned visual odometry is DeepVO [106] in 2017, which is a monocular odometry system relying on Recurrent Convolutional Neural Networks – a mix of CNNs and RNNs. Together with the increasing use of GPUs for deep learning and the increasing interest in data-driven algorithms, many other algorithms followed: UndeepVO [60], D3VO [113], DeepTAM [119], DeepSFM [108], ...

In practice, learned odometry does not (yet) perform as well as model-based SLAM and are difficult to generalize to environments not included in the training data. On the other hand, they have made major progress in the last years, and may become the undisputed state-of-the-art in the future. Figure 2.10 compares a learning-based visual odometry, DeepVO, to the typical model-based approach.

Another significant difference with model-based SLAM is the overlap, both in terms of network architecture and training data, with other domains. For instance, learned depth estimation from RGB images is an active research topic [23, 114, 37, 68, 44, 44]. Depth may also be an additional output from a network designed primarily for odometry purposes [113]. Depth necessarily appears in pose computation, explicitly or not – which is why it makes sense that models able to compute a trajectory would also be able to compute the depth of an image. Overall, other domains dealing with pose estimation have strong similarities in terms of network architecture, like object/human pose estimation [59, 90, 27, 6], place recognition [117], feature/keypoint computation [29, 115] and motion prediction [25].

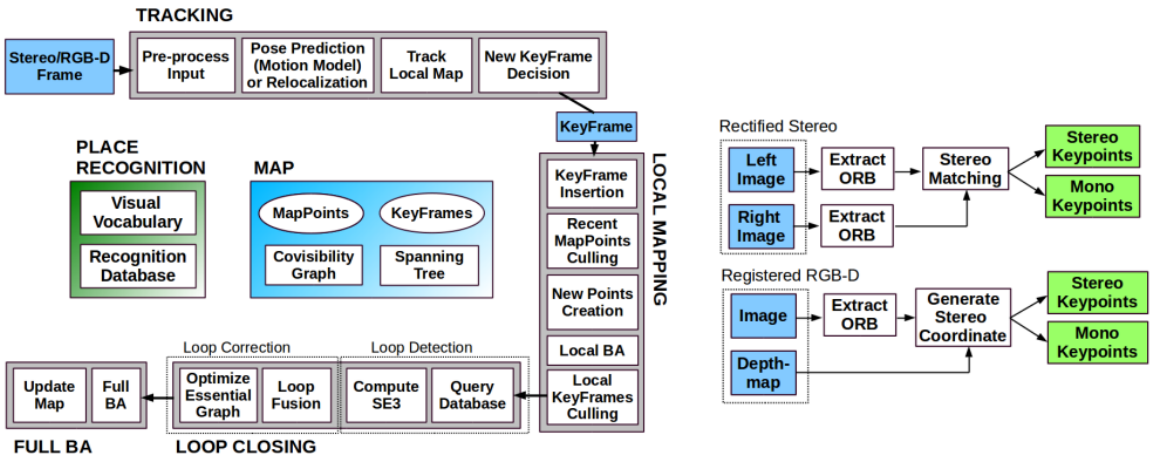


Figure 2.9 – ORB-SLAM 2 architecture [76]. It is an extension of ORB-SLAM to stereo and RGB-D sequences.

2.1.6 Hybrid approaches

Hybrid approaches are a compromise between the power of deep learning and the convenience of model-based approaches. They consist in replacing key modules in the SLAM pipeline (fig. 2.8) as the feature extraction step with learned modules. An example is LIFT-SLAM [16], which uses LIFT [115] for keypoint detection. A similar example is SuperpointVO [30].

Deep learning networks can also work as virtual depth sensors for integration in monocular SLAMs. CNN-SLAM [97] and [67] proposed such approaches. Another major use case is object detection for SLAM in dynamic environments – we talk about this in details in section 2.2 and later chapters.

Today, virtually all modules of a model-based SLAM have their "deep" counterpart [26, 80, 102, 70, 31, 97, 83, 15, 98, 64]. While learning-improved SLAMs have attained unprecedented performance, the usual drawbacks persist: the need for training data, computational complexity, memory requirements, lack of explainability. Considering the fast advances in AI research, this justifies why deep methods are almost omnipresent and at the same time replaced very quickly.

2.2 Dynamic SLAM: SLAM in Dynamic Environments

2.2.1 Problem formulation and choice of a feature-based Dynamic SLAM

Modern Visual SLAMs [76, 39] are mature and used in real-life applications such as Robotics and Autonomous Vehicles. Dynamic SLAMs, or SLAMs designed for dynamic environments, are an active research topic. [81] gives an overview of Dynamic SLAMs: the general principle is to remove image features on dynamic objects so that the SLAM only uses static features. Current methods detect motion, semantically segment dynamic objects, or both.

Given a geometric SLAM algorithm, making it robust to dynamic objects is relevant to real-world applications since it generalizes the SLAM from static to dynamic environments. Keeping the original, non-learned modules of an existing SLAM is valuable as it limits generalization issues encountered by learned algorithms, *e.g.*, end-to-end SLAMs [107]. Since end-to-end SLAMs do not have a map available, as explained in section 2.1.5, end-to-end SLAMs cannot be used for applications requiring an exploitable map.

For these reasons, we focus on Dynamic SLAMs that keep the original SLAM modules. Between a feature-based SLAM (section 2.1.4) and a direct SLAM (section 2.1.4), it is much simpler to filter

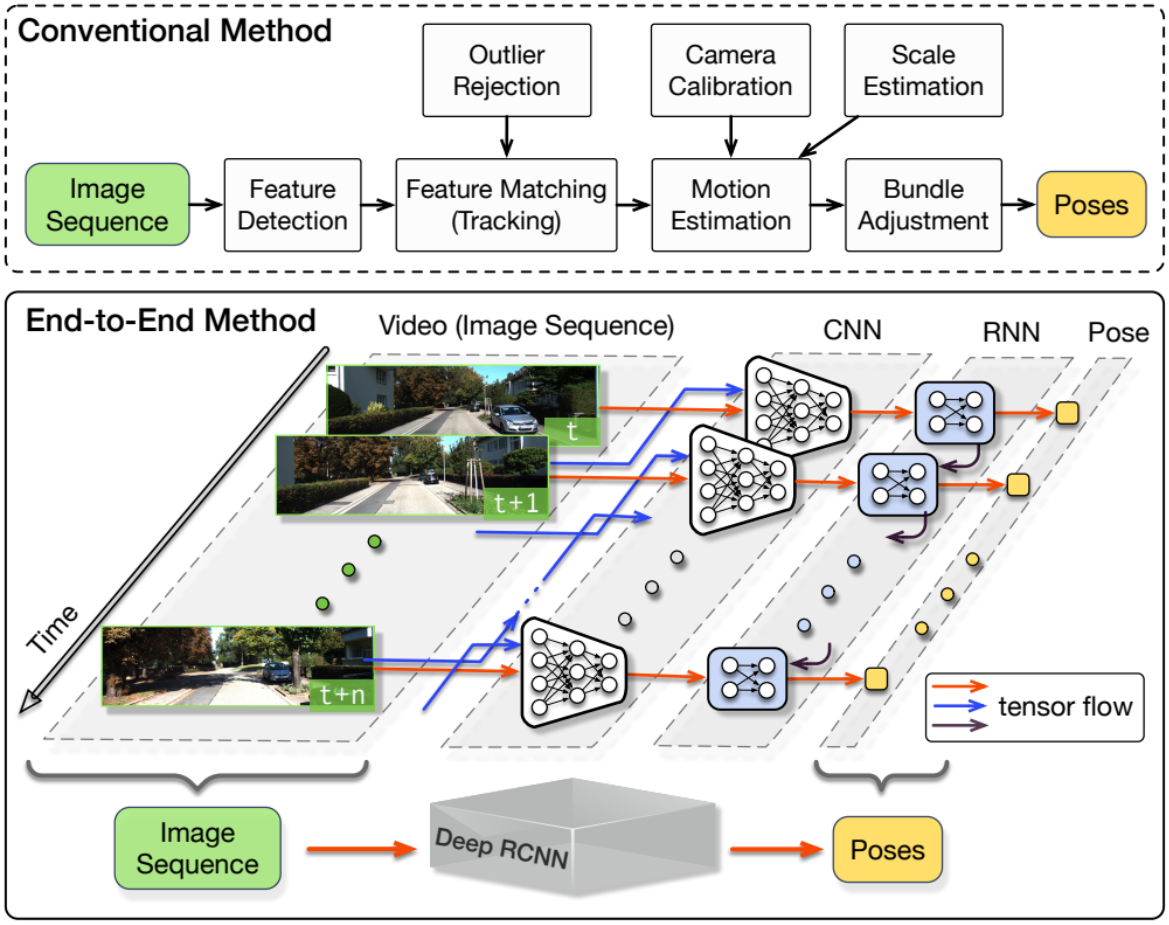


Figure 2.10 – Comparison between model-based SLAMs and a learning-based, end-to-end visual SLAM DeepVO [106].

unwanted features than it is to interfere in a photometric alignment process during SLAM. A direct SLAM expects a dense input (images) and relaxing this hypothesis is not trivial. Moreover, features are sparse compared to images, which makes the manipulation and analysis of features faster and simpler. These reasons explain why the vast majority of Dynamic SLAMs are feature-based, and not learned or direct. **Thus, we focus on feature-based SLAMs in this thesis**.

Figure 2.11 shows the basic structure of a feature-based Dynamic SLAM. The main difference with a feature-based SLAM (fig. 2.7) is the addition of a module to filter image features – *i.e.*, keypoints. From the input image, a new module generates a mask of areas considered dynamic, and then all features in masked areas are removed in the filtering step before proceeding to the normal Tracking and Mapping steps.

Another possibility would be to detect points only areas of the image that are considered static, *i.e.*, pre-filter features. In practice, this would cause edge effects as feature detectors analyze the image by patches, sometimes the full image at once in case of learned detectors. Thus, pre-filtering features is much more likely to cause unexpected issues than post-filtering them. This also explains why post-filtering is the standard approach.

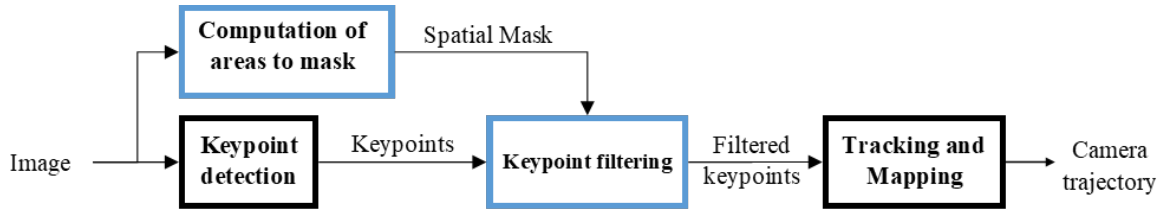


Figure 2.11 – The structure feature-based Dynamic SLAM: keypoint generation, Tracking/Mapping and a keypoint filtering step based on masks computed from the input image.

2.2.2 Motion-based, geometrical masking

Motion-based masking approaches filter features in dynamic regions [57, 21, 110, 84], typically with optical flow. [81] surveys many non-learned Dynamic SLAMs and concludes with "handling missing, noisy, and outlier data remains a future challenge for most of the discussed techniques [...] Most techniques also have difficulty in dealing with degenerate and dependent motion.".

Optical flow approaches [20] compute pixel displacements between frames but may not work if dynamic objects occupy most of the scene or have an erratic motion.

Depth maps approaches [93] use the additional depth information to identify salient objects but are limited by sensor range and resolution.

Clustering/background-foreground approaches [61, 94] identify dynamic objects by grouping and assigning probabilities to points with similar motions but have high computational costs and do not work well with noisy or degenerate motions [81].

A general limit of motion-based masking approaches is the underlying assumption that most of the image corresponds to static objects. The *motion consensus* of an image is often computed through algorithms like RANSAC (Random Sample Consensus), which consist in separating "inliers" from "outliers". Inliers are points that fit a predetermined model and outliers those that do not. The consensus is another name for the set points that fit the model. Therefore, in a SLAM context, the motion consensus is the underlying motion of the inliers with respect to the camera. Thanks to the *mostly* static world assumption, we assume that inliers are fixed with respect to the ground, which means that the relative motion between the camera and the frame of reference corresponding to the inliers is in fact the motion of the camera with respect to the ground. Note that "mostly static" theoretically means here 50% of features are static. This is unlike

In case of motion consensus inversion, the SLAM considers static objects as dynamic and vice-versa – effectively swapping inliers and outliers. This makes the SLAM use dynamic objects as frame of reference: the SLAM consequently drifts when the objects move. Such inversions occur when most of the features are on dynamic objects, which may occur when they are too close to the camera. The SLAM fails because inliers are not fixed anymore with respect to the ground: this implies that the computed motion is not the trajectory of the camera with respect to the ground. This applies to all geometry-based approaches: optical flow, depth maps, clustering and others.

To illustrate the limitations of the optical flow approaches, consider the object, a dragon, in fig. 2.12 (more images in fig. 4.6, section 4.5.3). The difficulty is that the camera and the object may be moving at the same time. Note that some approaches compensate the camera motion, as the Dynamic SLAM based on Lucas-Kanade flow in section 5.3.2, but this supposes that we have a way to estimate the camera motion before the actual SLAM. This estimation is in fact accurate only outside motion consensus inversions, which is the very problem we are trying to solve – so we are deadlocked, unless we use a method that does not rely on geometry.

For illustration purposes, we compute a dense optical flow using OpenCV[1] and PWC-Net [91], a learned optical flow method. Figure 2.13 shows we can see that the difference between a moving object and the background, but we cannot distinguish between a situation where all objects are moving, or

[1] https://docs.opencv.org/4.5.4/d4/dee/tutorial_optical_flow.html

Figure 2.12 – An object used in our datasets, a dragon.

none are. Figure 2.14 shows a different situation: the object is moving in all cases, but it can be identified only in the first case, being almost invisible in the second one.

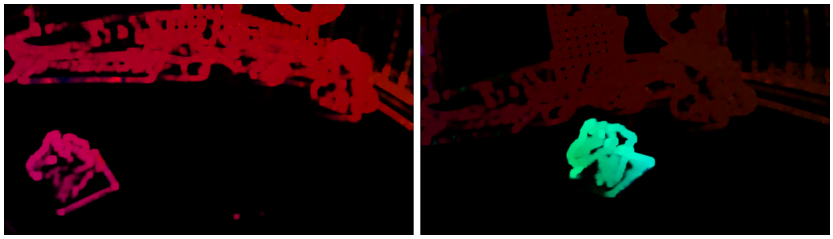


Figure 2.13 – Example of optical flow using OpenCV. The camera is moving in both scenarios. The object is static in the left image and moving in the right image. We cannot distinguish between a situation where all objects are moving, or none are.

Figure 2.14 – Example of optical flow using PWC-Net [91]. The camera and the object are both moving in the two images. The object is far from the camera in the left image and close to it in the right one.

2.2.3 Semantic masking approaches

Semantic masking approaches [53] typically use networks as Mask R-CNN [47] to filter features on objects of specific classes (*e.g.*, cars). Most have the masking module placed right after feature computation: they either remove features on dynamic objects or process them separately. As semantic segmentation does not rely on motion consensus, these approaches are unlikely to suffer MCIs and drift unless there are dynamic objects of unknown classes. However, they tend to mask objects even if they are not moving (*e.g.*, parked cars) and cause early SLAM failure when there are not enough remaining features.

2.2.4 Hybrid approaches

Hybrid approaches combine semantic and motion masking. [8] combines multiview geometry and semantic segmentation to remove features on dynamic objects of known and unknown objects. However, it may remove too many features and fail as it unconditionally masks all detected objects, even if they are not moving. Some hybrid approaches apply semantic masks only when they consider the masked object dynamic. [82] depends on motion detection: it classifies features as static/dynamic according to the semantic class of the object they are on and how many times the corresponding reconstructed map points have been observed at the same position. However, reconstructed points on a dynamic object may be mistakenly considered static under motion consensus inversion, making the method fail. [3] computes an approximate camera motion using features that are considered static by the semantic segmentation, then selects the dynamic features that are on motionless objects using photometric error, and finally inputs both static and selected dynamic features into the SLAM backend. However, if there are features only on objects of dynamic classes, it will be unable to compute the camera pose and fail. [109] is similar to [3] but uses reprojection errors instead of photometric errors.

[5, 118] are two-step approaches: they learn to segment dynamic objects at training time and do only semantic masking at runtime, so they may suffer from early failures. [5] creates a database of dynamic objects by comparing point clouds. However, [5] requires training sequences recording the same location at different times, does not detect objects that do not move between training sequences, and needs a full stereo camera + LIDAR setup. [118] first trains a depth estimation CNN with self-supervision then generates outlier masks – *i.e.*, masks that give the probability of a pixel being photometrically consistent with the previous frame – from depth estimation results at runtime. However, this method requires an existing semantic segmentation network to start training. Moreover, it depends on photometric / consistency losses and those are misleading under an MCI: when a dynamic object covers most of an image, both photometric error and consistency error would be low as the object appears as an inlier instead of an outlier, so outlier mask estimation would not work under an MCI. Finally, both [8, 118] remove detected objects unconditionally even if they are not moving, and may fail if they remove too many of them.

To the best of our knowledge, all current vision-only Dynamic SLAMs have fundamental limitations related to dynamic objects – *e.g.*, motion consensus inversions or early failures due to excessive masking. The key cause is the dependency on instantaneous motion detection, which is not reliable in some cases. Later in this thesis, we first propose the solution to the problem of unknown objects and MCIs in chapter 5. We then propose in chapter 6 a radically different approach as we *predict* the effect of objects on the SLAM. We do so by adding temporal masking to a SLAM: a memory-based decision module that signals the SLAM when to apply given semantic masks.

Chapter 3

Understanding the relation between image features and Dynamic SLAM performance

The goal of this chapter is to reflect the exploratory work that later resulted in our contributions: outlier-based masking, Temporal Masking, datasets, and metrics. We consider the Dynamic SLAM problem as formulated in section 2.2.1, so we focus on feature-based Dynamic SLAM.

Current Dynamic SLAM research is mostly about improving the accuracy of SLAMs when objects move by removing the appropriate features. While image features (*i.e.*, keypoints) [45, 78] and their uses in feature-based SLAMs [75] have been studied for years, the relation between features and Dynamic SLAM performance is unclear. Therefore, we conducted a set of experiments to understand this relation to methodically improve feature-based Dynamic SLAM algorithms. We interchangeably use *keypoint* and *feature* in this chapter. The main results are:

1. Filtering keypoints with external learned tools offers better generalization and performance perspectives for Dynamic SLAMs (section 3.1) than filtering keypoints with geometrical, manually crafted solutions, or using end-to-end learned keypoint detectors.

2. Evaluating SLAM performance is difficult due to SLAM repeatability issues (section 3.3.3) and the need for metrics to account for failures (section 3.2).

3. Feature-based SLAMs are typically configured to use less than 5000 keypoints: in this context, the 2500-3000 range maximizes SLAM performance for the classical keypoint detectors we evaluated (section 3.2).

4. Learned keypoint detectors are, by design, more repeatable than classical ones (section 3.3.2).

5. Superior keypoint detector repeatability does not imply superior SLAM accuracy (section 3.3.2), but self-supervised learning can be used to force this relation (section 3.3.3).

6. Keypoint detection repeatability does not imply SLAM repeatability. But SLAM-repeatable keypoints – both inliers and outliers – can be used to generate objects masks for keypoint filtering. Using these masks improves SLAM performance (section 3.4.2). It is possible to create dense masks by leveraging learned methods (section 3.4.3).

7. Keypoint filtering should consider the temporal aspect of the scene, masking objects when necessary to keep as many useful features as possible (section 3.5).

3.1 Keypoints as the cornerstone of feature-based SLAMs

As explained in section 2.1.4, keypoints are image features crucial for feature-based SLAMs since they are the link between the video sequence and the Tracking/Mapping backend. In this section, we clarify their place in SLAMs and Dynamic SLAMs.

3.1.1 The use of keypoints in SLAMs

Features/keypoints are key elements of an image (*e.g.*, corners) that are matched across frames in order to compute a trajectory. Figure 3.1 illustrates such features. Note that for matching purposes, feature detectors are combined with a *feature descriptor*, which computes a fixed-size, real-valued vector encoding the unique appearance of the neighborhood of the feature. For instance, ORB-SLAM [75] relies on the ORB feature detector [78], which combines the *FAST* detector and *Rotated BRIEF* descriptors. Descriptors are designed to make descriptions unique and easily matched, even if the scene illumination or point of view changes. Keypoint detection is an active research topic.

Feature detection methods can be grouped into end-to-end methods – *i.e.*, fully learned methods – and classical methods, which are geometry-based. Classical methods typically have the advantage of being faster to compute and lightweight [78] compared to end-to-end methods. On the other hand, end-to-end methods [115], thanks to the advances in deep learning, are achieving superior robustness in terms of point matching.

Figure 3.1 – Illustration of matched features (green points) in a SLAM.

3.1.2 Filtering keypoints in Dynamic SLAMs

We previously explained in section 2.2 the principle of feature-based Dynamic SLAMs: to filter features that are on dynamic objects by masking the latter, *i.e.*, dynamic features. However, the concept of what is dynamic is ambiguous, which leads to problems as motion consensus inversions (presented in section 4.2.2). We may have:

1. **No learning**: geometric considerations, like the optical flow, are used to tell which features are dynamic.
2. **External learned tools**: to identify dynamic features, we use an external learned algorithm.
3. **Implicit learning**: in the case of end-to-end features, the feature detector directly computes static features, or alternatively directly classifies its own features in static/dynamic, *e.g.*, [30].

Purely geometrical approaches with no learning are difficult to improve further. In implicit approaches, it is virtually impossible to decouple the keypoint detector from the dynamic point classifier – it is a monolithic trained model. This severely limits the use of different feature detectors, in particular classical ones, which may be a requirement in some industrial or medical settings. Another benefit of decoupling, which appeared much later (in chapter 6), is that dealing with complex scenarios requires a memory-based approach. Integrating memory in a keypoint detector is a very delicate matter as they are normally stateless and designed to operate on a frame-by-frame basis. Moreover, it would make the keypoint detector bloated and exceedingly complex.

For these reasons, we chose the *external learned tools* path: this option offers the best perspectives and generalization. Note that this does **not** prevent using learned feature detectors as [29]. We just need to make our decisions on features being static/dynamic independent from the keypoint detector.

3.2 Influence of the number of features on SLAM performance

The core parameter of most non-learned keypoint detectors is the maximum number of detected features – there may be less at runtime. The configured number of features to use in a SLAM varies significantly but is usually less than 5000. To ensure that our research is not biased by a suboptimal configuration of this setting, we need to determine the number of features lower than 5000 that maximizes SLAM performance for various keypoint detectors. This implies that future SLAM improvements are obtained thanks to our method and not just a better configuration of the SLAM. Unexpectedly, we identified the issue of SLAM repeatability: SLAM results can be unstable, hence SLAMs must be evaluated over several executions of the same dataset.

3.2.1 Experimental Setup

We ran a modified ORB-SLAM 2 [76] on six popular sequences of the TUM RGB-D dataset (presented in section 4.3.2) with various keypoint detectors: ORB-SLAM 2's native detector ORB and OpenCV library's GFTT, Shi, Shi-Harris and SIFT detectors[1]. We increased the number of features from 500 to 6000 in steps of 50, running the SLAM 24 times per step.

The modifications in ORB-SLAM 2 consisted in replacing the original detector by an abstraction layer that calls upon the appropriate keypoint detector. Note that as ORB-SLAM 2 uses image pyramids (8 by default) – *i.e.*, combined feature computation and matching on the same image at different resolutions –, we can run keypoint detectors in two ways: multi-layer or single-layer. In *multi-layer*, we apply the given keypoint detector on every pyramid level (as ORB-SLAM 2 original worked); in *single-layer*, we use the given detector only on the full resolution image. Certain detectors as SIFT *already support* pyramid levels internally, thus we prevent them from doing so, limiting them to one internal pyramid in multi-layer mode; in single-layer mode they are set to use internal pyramids. The native ORB-SLAM 2 detector ORB always runs in multi-layer mode. In this experiment, we use multi-layer mode for all detectors that do not support internal pyramids, and single-layer mode with internal pyramids otherwise.

3.2.2 Evolution of SLAM performance with the number of features

Figure 3.2 shows the results for the GFTT keypoint detector and fig. 3.3 for the ORB keypoint detector. It appears, for all detectors, that the accuracy varies significantly although the median accuracy reaches the expected baseline. We can also see that the Tracking Rate and the ATE RMSE may evolve differently with the number of features, making the interpretation of results difficult.

Therefore, we need a way to compare to compare results even if the ATE RMSE and Tracking Rate behave differently.

[1] Source: https://docs.opencv.org/4.5.4/d5/d51/group__features2d__main.html and https://docs.opencv.org/4.5.4/d3/df6/namespacecv_1_1xfeatures2d.html

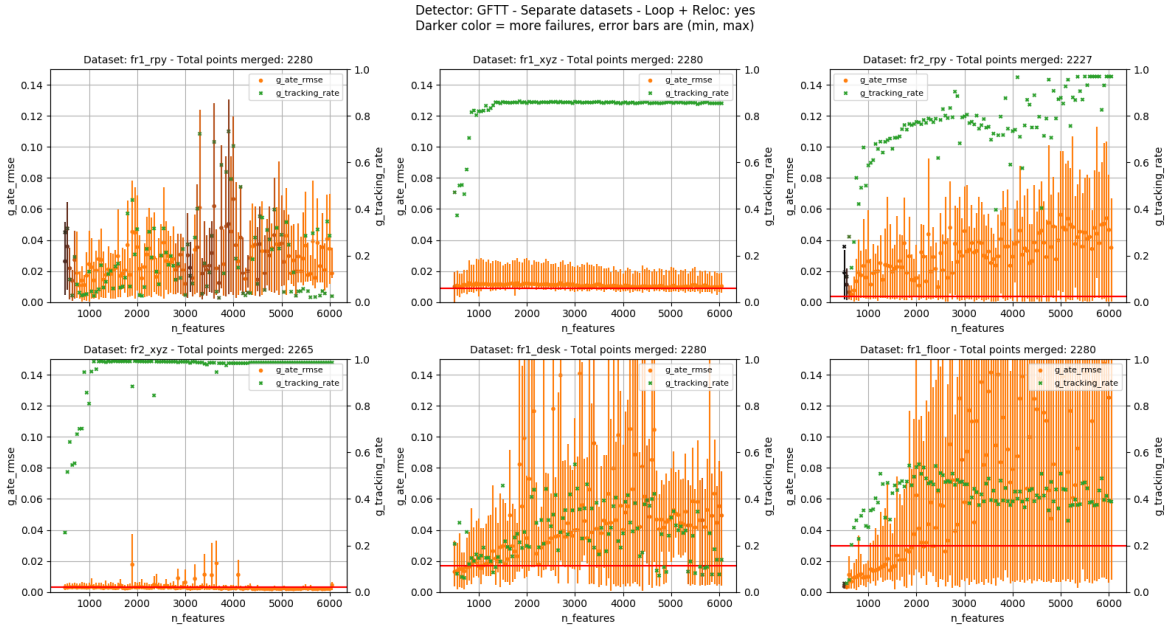


Figure 3.2 – Benchmark of ORB-SLAM using the GFTT keypoint detector. We show the ATE RMSE and Tracking Rate. The red line indicates ORB-SLAM 2's performance in the original paper.

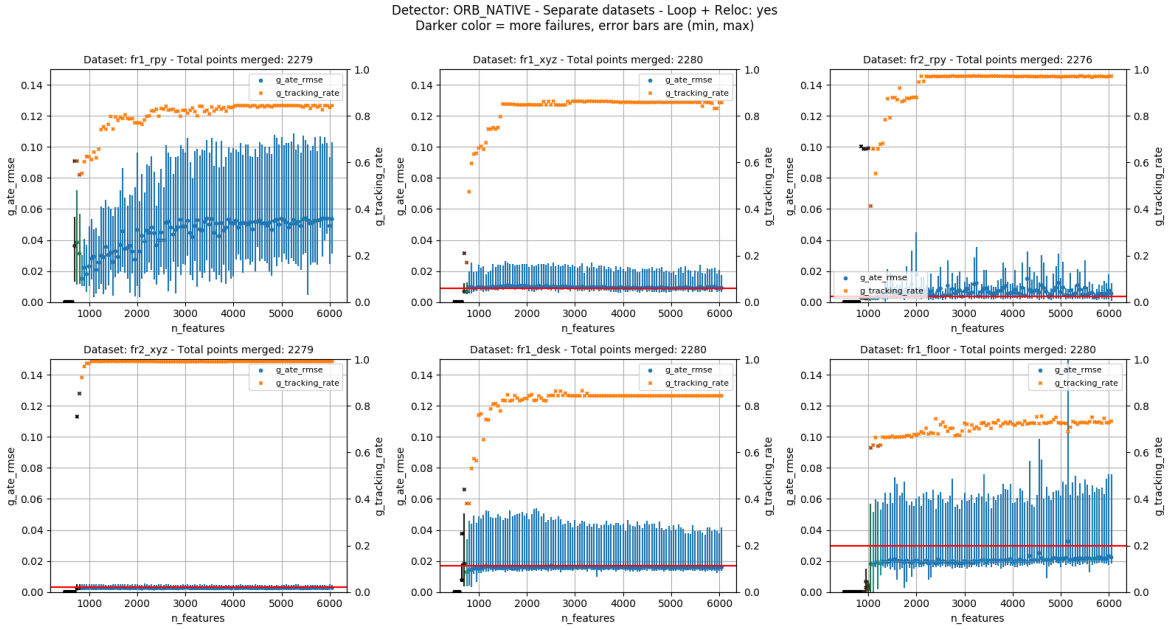


Figure 3.3 – Benchmark of ORB-SLAM using the original ORB keypoint detector. We show the ATE RMSE and Tracking Rate. The red line indicates ORB-SLAM 2's performance in the original paper.

We propose to evaluate SLAM performance with the Weighted ATE RMSE (WATE) instead of the ATE RMSE: $\text{WATE} = \frac{\text{ATE RMSE}}{\text{TR}+\epsilon}$, with $\epsilon = 10^{-7}$. This metric is our first attempt at combining ATE RMSE and Tracking Rate. It will later evolve into new metrics proposed in section 4.4. The analysis with the new metrics is shown in fig. 3.4 and fig. 3.5, without error bars. Results for other keypoint detectors are in appendix B.1. While the results with the ORB detector are smooth, the results on GFTT are unstable for a feature number over 3000. For both detectors, we can clearly see how the SLAM starts working at about 1000 features, with a Tracking Rate and Weighted ATE RMSE that stabilizes. We have comparable results on other detectors (Shi, Shi-Harris, SIFT). For a combined view, fig. 3.6 shows the Weighted ATE RMSE of the ORB detector on all six sequences at the same time.

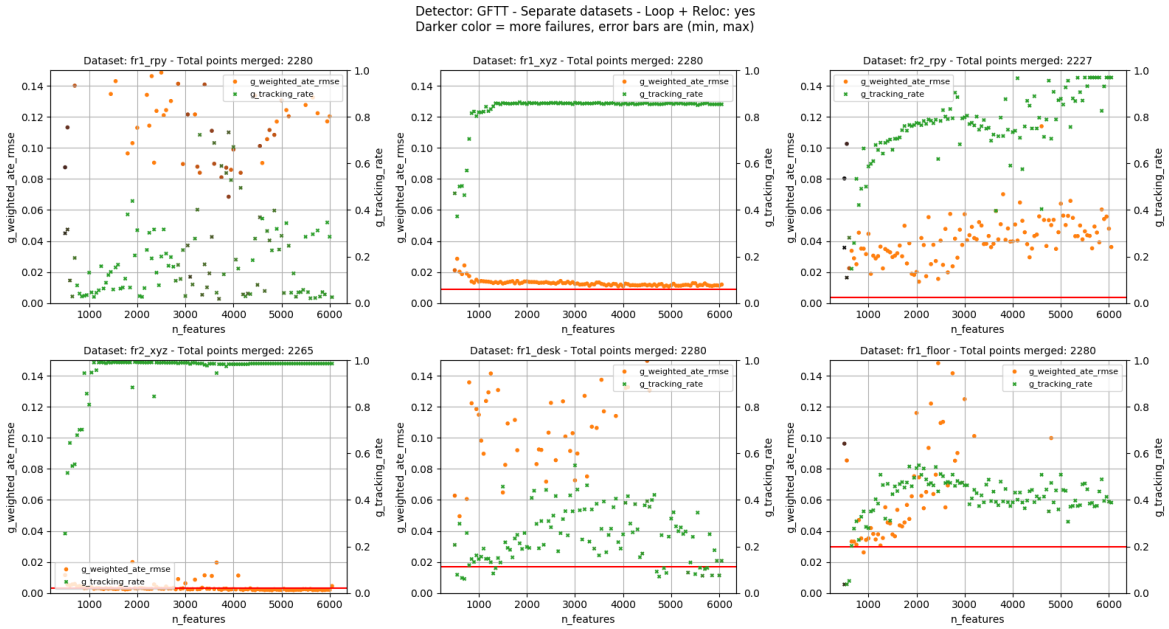


Figure 3.4 – Benchmark of ORB-SLAM using the GFTT keypoint detector. We show the Weighted ATE RMSE and Tracking Rate. The red line indicates ORB-SLAM 2's performance in the original paper.

The conclusion of this experiment is that it is safe to use 2500 to 3000 features for all keypoint detectors and that we have to include the Tracking in a performance analysis. Hence, we use the value of 3000 features for our ORB-SLAM 2 experiments, using the ORB detector, in the next chapters.

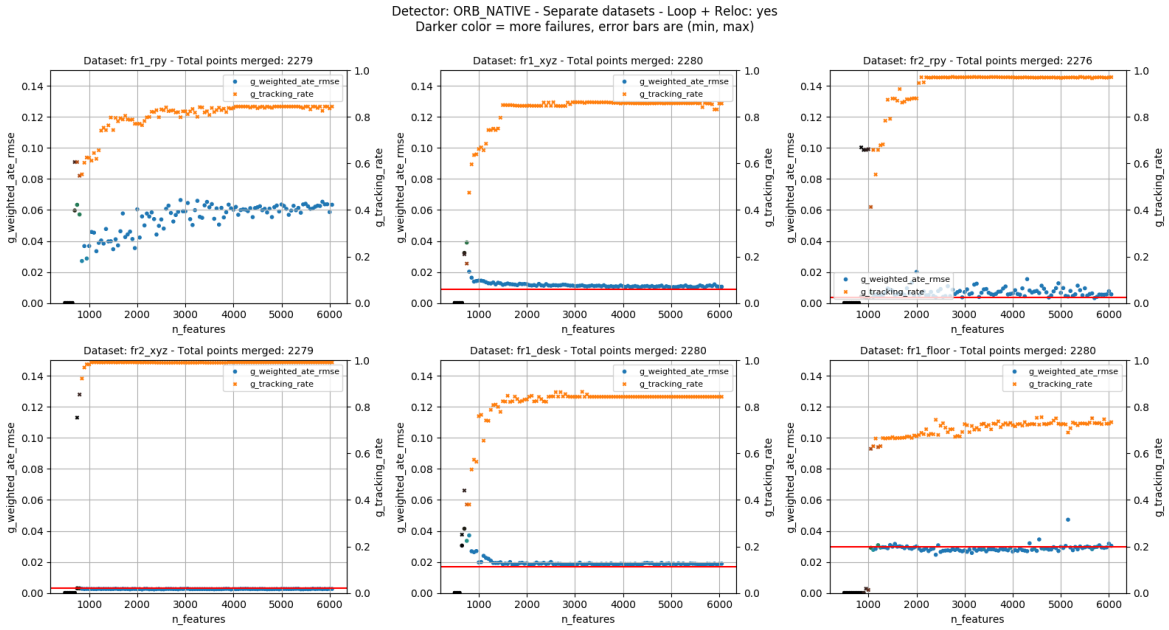


Figure 3.5 – Benchmark of ORB-SLAM using the original ORB keypoint detector. We show the Weighted ATE RMSE and Tracking Rate. The red line indicates ORB-SLAM 2's performance in the original paper.

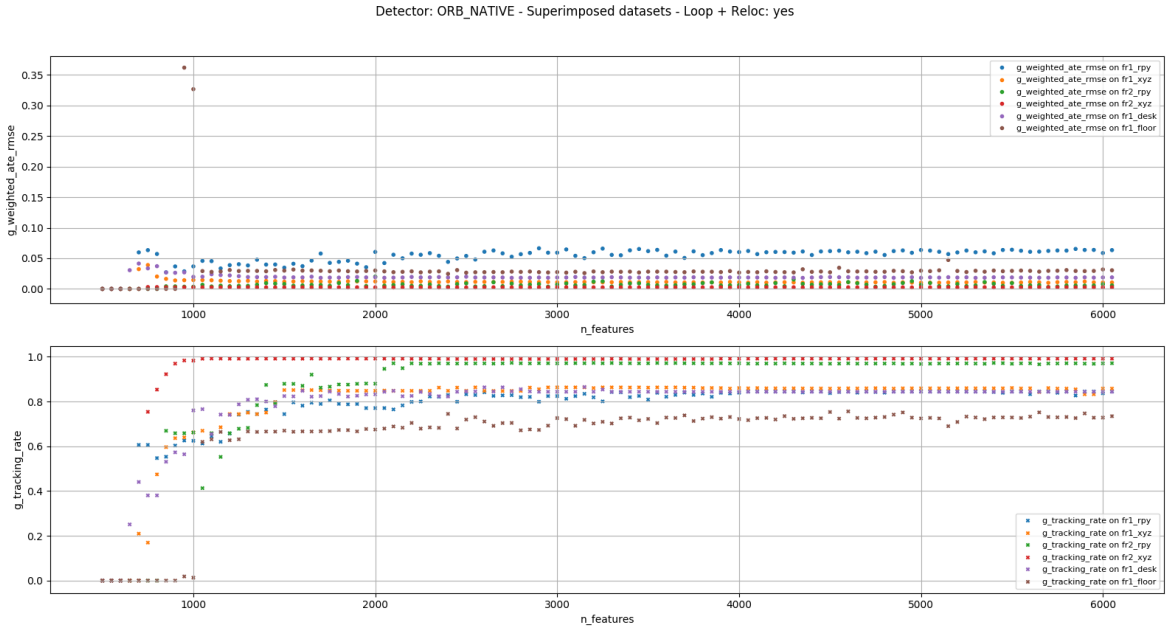


Figure 3.6 – Benchmark of ORB-SLAM using the original ORB keypoint detector. We show the Weighted ATE RMSE and Tracking Rate on six sequences at once.

3.3 Relation between keypoint detector repeatability and SLAM performance

We evaluate in this section both learned and classical, non-learned keypoint detectors. Our goal is to compare both approaches and at the same time understand how much the repeatability of keypoint detectors – the main metric of keypoint detectors – affects SLAM performance. It appears that a keypoint detector being learned or having a higher repeatability does not imply better SLAM performance. A positive outcome of our experiments is that we identified the importance of self-supervision for Dynamic SLAM.

3.3.1 The main keypoint detector metric: repeatability

An essential property of keypoint detectors is repeatability: the fact that keypoints are repeatedly detected on the same physical location in different images. We started by testing how repeatable keypoint detection is. The rationale is that if keypoint detection is repeatable, there may a repeatable property unique to dynamic objects that we could later repeatedly detect.

The standard way to evaluate repeatability is to check if features are physically detected on the same locations, within a certain error margin, between two images of the same scene but from two different points of view. If there are no multiple point of views of the same scene, a fallback approach is to warp images to simulate new points of views. A standard benchmark for this purpose is the HPatches dataset [4]. Figure 3.7 shows a sample from it.

Figure 3.7 – Illustration of images from the HPatches dataset [4] for keypoint detector repeatability tests.

Repeatability computation consists in comparing the keypoints on the common part of an original image and its warped (and if needed, cropped) counterpart. Keypoints on the warped image are projected onto the original one – *i.e.*, unwarped – before comparison, and we check if there is another keypoint within ϵ pixels of the original one. It is necessary to restrict the comparison to the common part as the projection may make some points fall out of bounds. Figure 3.8 illustrates the keypoint detection process on an image and its warped version. Note that keypoint detection may require cropping the original image, especially if using trained models that require an input of fixed size. Figure 3.9 shows the process of matching an image and its warped counterpart for comparing keypoints.

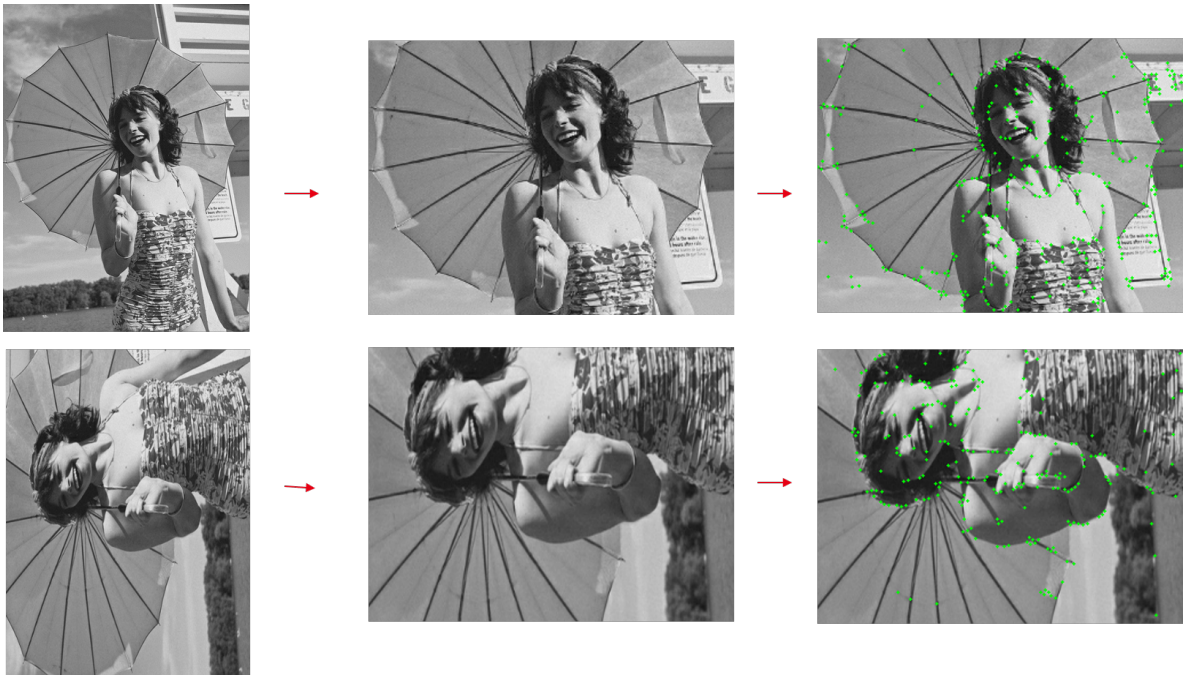


Figure 3.8 – Illustration of keypoint detection on the original image and the warped image. The images are cropped to the same dimensions before detection. Original image from the COCO dataset [65].

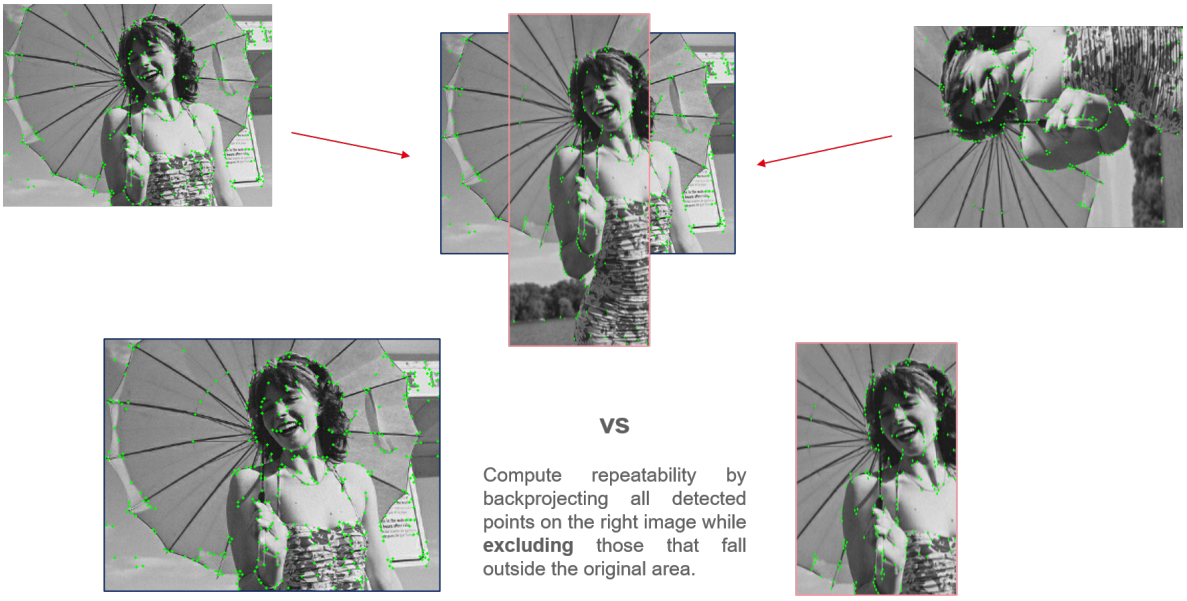


Figure 3.9 – Visual representation of keypoint repeatability computation. Once we have the keypoints from the original image and the warped image, the keypoints of the warped image are projected onto the original image. Repeatability computation evaluates keypoint repeatability only on the common part between the original image and the unwarped image.

3.3.2 Experiments

Experimental Setup

Our first experiments are the qualitative and quantitative evaluation of the repeatability of various keypoint detectors: SuperPoint [29] and variants of it as well as the Harris corner detector [45].

The **Harris corner detector** is a lightweight classical keypoint detector that has been used for a long time for SLAM purposes. It relies on geometrical considerations and consists in the selection of points whose local structure (made of spatial derivatives) have the highest *Harris response*, *i.e.*, the smallest eigenvalue.

SuperPoint is a learned detector that obtained State of the Art results and performance comparable to classical keypoint detectors. The underlying network has a keypoint output (location of keypoints) and description output. It is trained through self-supervision in three steps. a) Pre-training step, where it uses synthetic, automatically generated images and their known corners as ground truth. Figure 3.10 shows such an image. b) Automatic labelling through a process called *homographic adaptation*, which consists in detecting keypoints on warped versions of the same image using the pretrained model, then aggregating them after unwarping the image. c) Joint training of the keypoint detector and descriptor using the automatically labeled images. Figure 3.11 illustrates the training process and fig. 3.12 illustrates the homographic adaptation process.

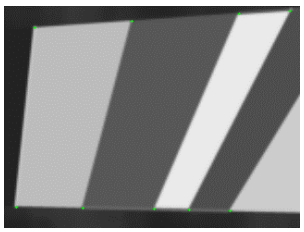


Figure 3.10 – A synthetic image for pretraining or repeatability tests. Green points are labels.

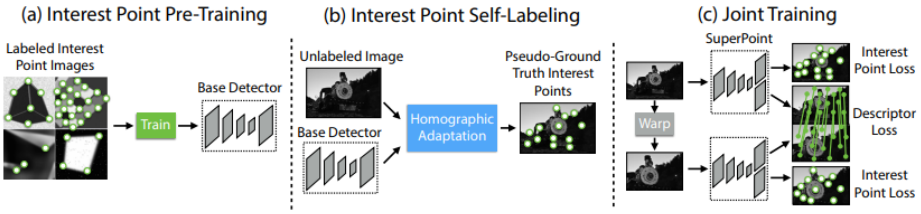


Figure 3.11 – Illustration of SuperPoint training process [29].

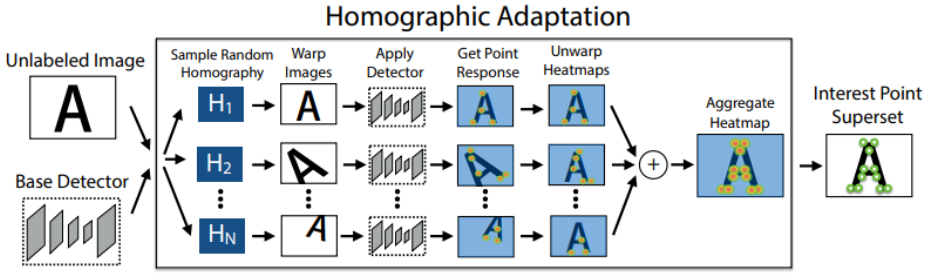


Figure 3.12 – Illustration of SuperPoint homographic adaptation [29], which is used for automatic labelling of images using a pretrained model.

Comparison of detector repeatability between a classic and a learned keypoint detector

We trained the SuperPoint detector ourselves by adapting a public implementation[2]. We trained four variants of it:

1. MagicPoint Shapes: the model resulting from the pretraining step, trained only on synthetic shapes
2. MagicPoint COCO: the model resulting from the full training but without training the descriptor, using COCO images [65] for the last step.
3. SuperPoint COCO: the model resulting from the full training descriptor included, using COCO images.
4. HarrisPoint: the same as SuperPoint COCO but image labels are computed using the Harris corner detector, effectively skipping step 1) and 2) of the SuperPoint pipeline.

We evaluated the repeatability of the Harris corner detector and our trained models on random samples of 200 images from various datasets: HPatches, Synthetic shapes (similar to the ones used for the pretraining of SuperPoint), and COCO. For COCO, we tested degrading the quality of the images through lossy JPEG compression. The rationale is to check detector robustness to the loss of input information. Figure 3.13 illustrates the degradation of image quality.

Qualitatively, we can see the detection result on images of 50%, 90% and 100% quality (no compression) resp. in fig. 3.14, fig. 3.15 and fig. 3.16. We can see that the learned detectors tend to generate less dense keypoints than the Harris corner detector. Additionally, compression artifacts cause the appearance of bogus keypoints that do not correspond to any physical object. Between learned models, MagicPoint Shapes has points that do not correspond to any real object, but those vanish in MagicPoint COCO and SuperPoint COCO. HarrisPoint COCO seems to generate less keypoints than the Harris corner detector. Interestingly, the latter detects less points as image quality decreases, possibly because the local structures are altered to the point that the Harris response is too low.

Table 3.1 shows the quantitative results. SuperPoint COCO appears as the best performer on natural images, especially of high quality. The Harris corner detector performs the best on synthetic shapes or small patches. HarrisPoint has the lowest performance.

The overall conclusion is that the learned keypoint detector SuperPoint is better on high-quality real-life images, MagicPoint Shapes on low-quality real-life images, and the Harris corner detector on synthetic images and HPatches images. This suggests that learned detectors are more repeatable on real-life images than classical detectors, which makes sense as they were trained for that.

Keypoint detector	Synthetic	HPatches	COCO50	COCO80	COCO90	COCO95	COCO99	COCO100
MagicPoint Shapes	0.409	0.711	**0.617**	**0.597**	0.577	0.592	0.619	0.618
MagicPoint COCO	0.454	0.724	0.484	0.494	0.597	0.687	**0.734**	0.705
SuperPoint COCO	0.354	0.530	0.527	0.541	**0.648**	0.694	0.722	**0.716**
HarrisPoint	0.386	0.872	0.461	0.449	0.449	0.463	0.487	0.484
Harris	**0.459**	**0.810**	0.540	0.523	0.489	0.499	0.473	0.469

Table 3.1 – Repeatability of various keypoints detectors (Harris and trained models based on SuperPoint) on various dataset samples of 200 images: from synthetic shapes, HPatches, and COCO dataset. The number after COCO indicates the quality of images in terms of JPEG compression.

[2]https://github.com/rpautrat/SuperPoint

Figure 3.13 – Examples of images of degraded quality using various levels of JPEG compression.

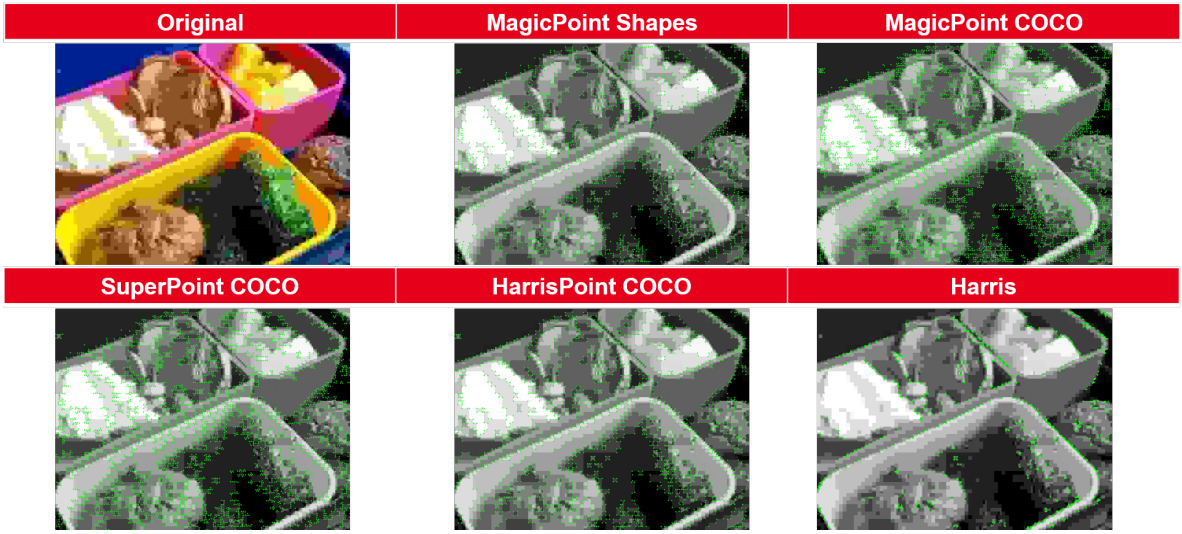


Figure 3.14 – Illustration of different keypoints detection methods on an image at 50% quality. Original image from the MS COCO dataset.

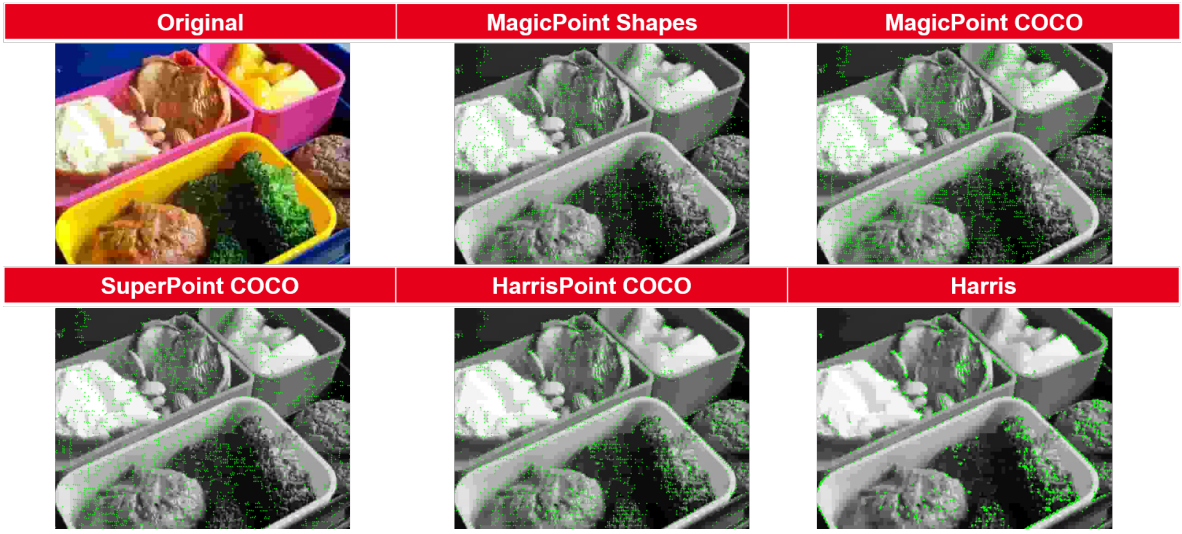


Figure 3.15 – Illustration of different keypoints detection methods on an image at 90% quality.

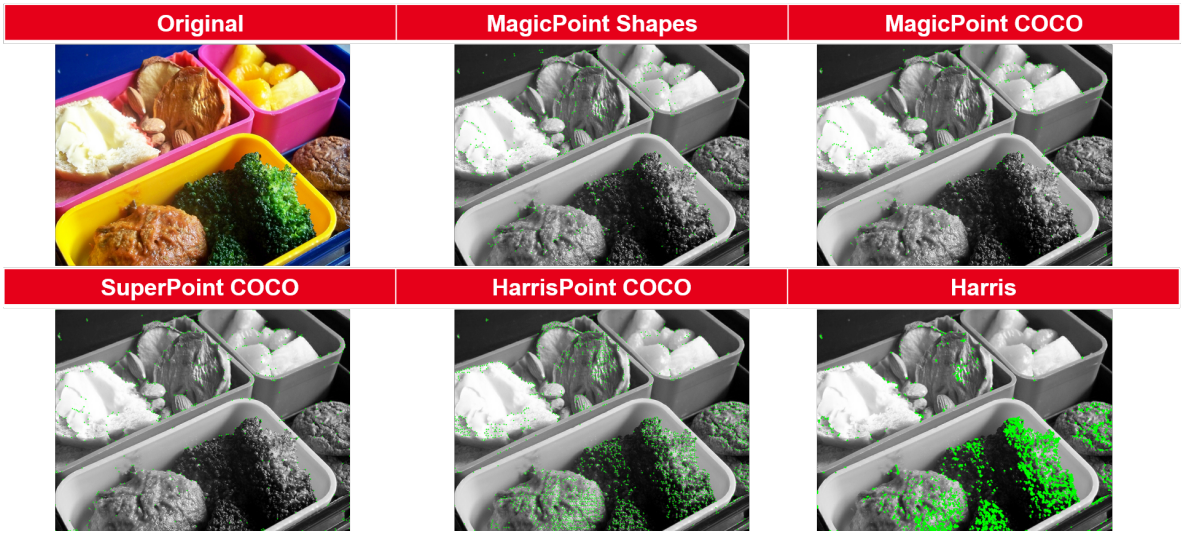


Figure 3.16 – Illustration of different keypoints detection methods on an image at 100% quality.

Comparison of performance between keypoint detectors

The OpenCV library provides a variety of keypoint detectors that can easily be tested. We evaluate the following keypoint detectors: ORB (ORB-SLAM 2 native keypoint detector), BRISK, SIFT, SURF, KAZE, AKAZE, Harris-Laplace, MagicPoint COCO (our model), SuperPoint COCO (authors' pre-trained network), Shi, Shi-Harris, GFTT, FAST, MSD, AGAST and STAR[3]. Note that the keypoint description process used for matching points (Rotated BRIEF in ORB-SLAM 2), is unchanged.

As we defined in section 3.2 the best number of features, 3000, we can compare keypoint detectors and link this result with the experiments on repeatability. The issue is that comparing detectors is tricky even with the Weighted ATE RMSE due to major SLAM failures. Failures cause the SLAM not to output anything, hence the ATE RSME – and the WATE RMSE – cannot be computed. For this reason, we used an alternate solution unique to this experiment.

We run ORB-SLAM 2 24 times on every sequence and compute the miss rate, *i.e.*, how many runs did not produce any output. We consider that the detector failed on a given sequence if the miss rate is over 20%. Then we rank detectors from better to worse according to their WATE RMSE for every sequence – first successful detectors according to their WATE RMSE (lower = better), then failed detectors according to their miss rate (lower = better). Finally, we compute the mean rank of every detector across the full dataset.

Table 3.2 shows the results in single-layer mode and table 3.3 in multi-layer mode. The overall best is the original ORB keypoint detector in both cases. **A first conclusion is that the overall best detectors are the classical ones ORB and SIFT**, which is consistent with their popularity in the research community. **Surprisingly, the learned detectors were not top performers** although they are supposedly the most repeatable. This matches the analysis in the next section, section 3.3.3: we need a direct link between keypoint detector design and SLAM though self-supervision to ensure performance. We can summarize the result as: **better keypoint repeatability through change of point of view does not imply better trajectory accuracy or Tracking Rate**, although there is a positive correlation.

ORB	4.54
SIFT	4.91
AKAZE	4.91
MAGICPOINT_COCO	5.25
SURF	6.39
SUPERPOINT_COCO	6.51
BRISK	6.92
HARRIS_LAPLACE	7.91
AGAST	8.38
FAST	8.60
GFTT	8.87
SHI	8.91
MSD	9.05
STAR_DETECTOR	9.11
KAZE	10.28
SHI_HARRIS	11.08

Table 3.2 – Mean rank of keypoint detectors in single-layer mode on the TUM RGB-D dataset.

ORB	3.76
BRISK	5.16
SIFT	5.38
AKAZE	5.47
SURF	6.10
KAZE	6.80
HARRIS_LAPLACE	7.03
MAGICPOINT_COCO	7.63
SHI	8.29
SHI_HARRIS	8.30
SUPERPOINT_COCO	8.67
GFTT	8.98
FAST	10.21
MSD	10.47
AGAST	10.81
STAR_DETECTOR	11.40

Table 3.3 – Mean rank of keypoint detectors in multi-layer mode on the TUM RGB-D dataset.

In conclusion, the Weighted ATE RMSE and the mean rank were useful to quantify SLAM performance using different keypoint detectors. Still, they are not very suitable for comparison between different SLAM algorithms as the mean rank and the Weighted ATE RMSE are both hard to interpret in terms of accuracy/robustness. For these reasons, we will present better metrics in section 4.4).

[3]Documentation: https://docs.opencv.org/4.5.4/d5/d51/group__features2d__main.html and https://docs.opencv.org/4.5.4/d3/df6/namespacecv_1_1xfeatures2d.html

3.3.3 Thoughts on keypoint detector repeatability, SLAM repeatability and self-supervision

Keypoint detector repeatability vs. self-supervision

We saw that learned approaches benefit from self-supervision, while labels coming from another source, like another keypoint detector, have poorer results. SuperPoint, and its later adaptation in a Dynamic SLAM, SuperPointVO [30], have several common points with our Dynamic SLAM problem. The question of image features is actually an ill-posed problem, like Dynamic SLAM when the frame of reference is not defined (section 1.1). The key issue is that keypoints are used for complex tasks – image stitching, SLAM, motion tracking, etc. –, so measuring the performance of keypoint by themselves is not enough. In practice, we still evaluate a key property of keypoints: their repeatability. However, this property does **not** ensure that a keypoint is "good" for the task they are employed.

To deal with the gap between a keypoint detector being repeatable and keypoints performing well in a SLAM, the authors of SuperPointVO added a third branch to SuperPoint that classifies their own detected points into *dynamic*, *unknown*, and *static*. The labels of this branch are automatically computed by running a lightweight SLAM and checking the stability of the map points corresponding to the detected keypoints – effectively making network training self-supervised. In this way, one increases the chances that the computed keypoints will perform well in a SLAM setting.

As previously explained in section 3.1.2, having the classification as static/dynamic intrinsically tied to the keypoint detector imposes major limitations. Still, the idea of using self-supervision to solve the ill-posedness of a problem is valuable, and eventually inspired our approaches.

Keypoint detector repeatability vs. SLAM repeatability

The experiments on the best number of features showed that ORB-SLAM 2 is not repeatable. We insist on the fact that the randomness is not due to keypoint detectors, which are deterministic, even for the learned SuperPoint. However, the *3D map*, and consequently the trajectory, may change.

We managed to reduce the randomness of ORB-SLAM 2 by doing the following:

1. Deactivating relocalization
2. Deactivating loop closure
3. Setting global random seeds for random functions (especially RANSAC)
4. Deactivating the graphical user interface (GUI)
5. Removing the early stopping during local bundle adjustment. The local bundle adjustment optimizes both map and poses but is a slow operation; for this reason, it is usually stopped, early if needed, after a maximum number of frames.

This still did not completely remove the randomness. A probable reason is that ORB-SLAM 2 is multithreaded and, despite our changes, threads are still not fully synchronized. This experiment will later guide the construction of the ground truth of our datasets (section 4.5.1), which is ORB-SLAM 2 running with the above changes, except the deactivation of relocalization and loop closure. A side-effect is that the SLAM becomes much slower, sometimes not reaching even 1 Hz, as more keyframes (frames used for map optimization) are created.

The conclusion is that making ORB-SLAM 2 – and more generally other SLAMs – perfectly repeatable is difficult. A better approach and good practice is to ensure that benchmarks run the SLAM enough times and compute average/median metrics.

3.4 Relation between outliers and Dynamic SLAM

During our earlier evaluations of ORB-SLAM 2, we observed that in ORB-SLAM 2 keypoints matched with the stored 3D map are clearly separated between inliers and outliers, where inliers are points that are part of the image consensus and can be mapped, and outliers are points that are not part of the consensus and cannot be mapped, thus being rejected. Note that ORB-SLAM 2 supposes that the image consensus corresponds to the static world which the user expects as frame of reference. We asked ourselves: **are inliers and outliers repeatable? Does this repeatability have a relation with dynamic objects?**

3.4.1 Experimental setup

Inliers and outliers are not *strictly* repeatable due to SLAM randomness (*e.g.*, due to multithreading). But how repeatable are they? Figure 3.17 shows the steps from the ORB-SLAM pipeline from where we can extract outliers. They are the *Initial Pose Estimation* step and the *Local Bundle Adjustment* step. The initial pose estimation is done by projecting the last known pose, assuming that the camera moves at a constant velocity.

Within ORB-SLAM 2, outliers are *2D feature - 3D map point matches* that are rejected after the initial match during robust optimization, in other words, *erroneous matches between new features and the existing map*. We store the location of the rejected features and call those outliers. Inliers consist in all matched features that were not rejected – *i.e.*, matched features that are not outliers. It is important to note that what we call outliers are points rejected during optimization that should never have been considered for map integration in the first place.

Features that were *never* matched are simply ignored and are not, in fact, outliers. Likewise, features rejected during initialization or relocalization also are not outliers. These features may not have been matched for many reasons, for instance due to being in areas that do not overlap different points of views: they are not erroneously matched points.

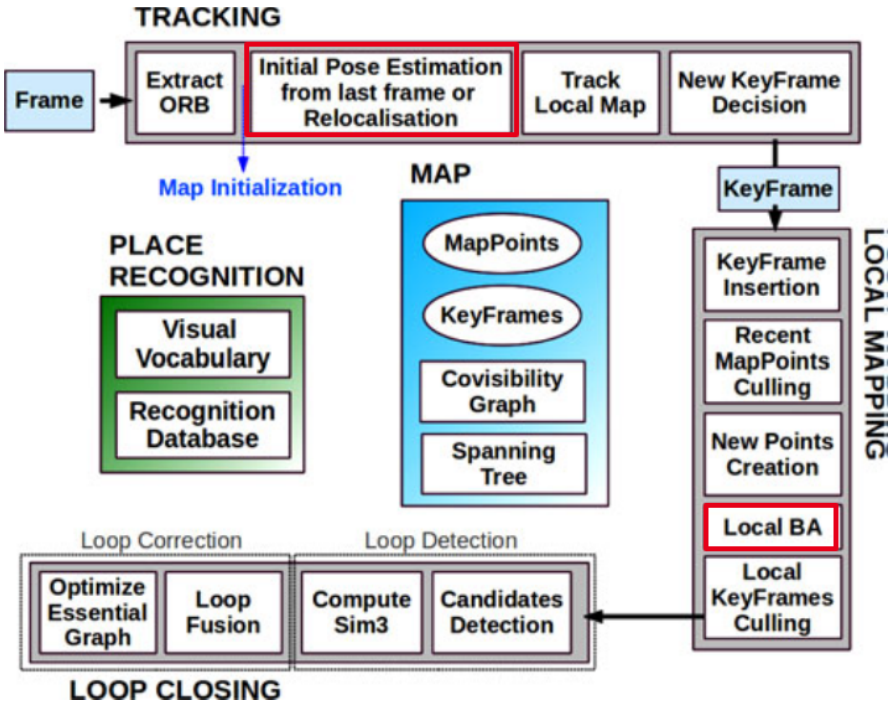


Figure 3.17 – Steps from the ORB-SLAM [75] pipeline (in red) from where we can extract outliers.

We collected inliers and outliers from TUM RGB-D dataset. We ran the SLAM 450 times and accumulated inliers/outliers in the form (x, y, n) where n is the total number of observations of inliers/outliers at the coordinates (x, y). Figure 3.18 and fig. 3.19 show respectively the outlier and inlier profile of the sequence fr3_sitting_halfsphere (788 images) processed with ORB-SLAM 2 in monocular mode. This profile is in fact typical, and other collections resulted in a similar result. We can see that about 90% of all outliers are observed in at most 81 images or less, and 90% of all inliers are in at least 650 images. It is clear that some inliers and outliers are much more repeatable than others.

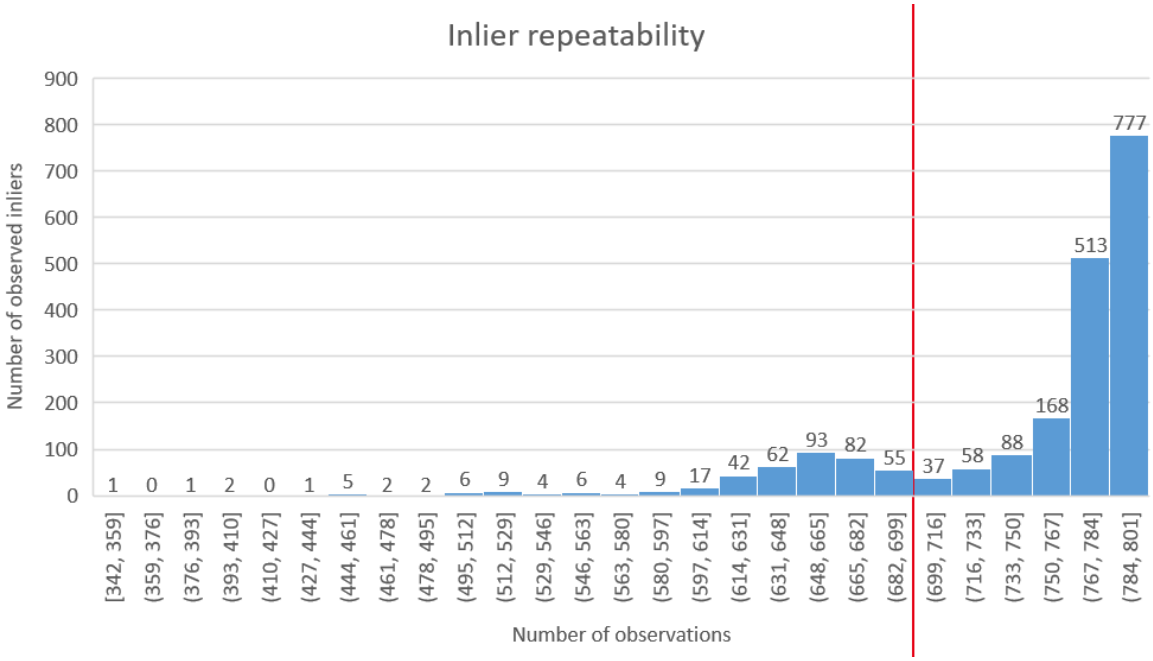


Figure 3.18 – Typical repeatability profile of ORB-SLAM 2 inliers. Relative threshold = 0.8.

The fact that some points are much more repeatable than others is meaningful, so we analyze in detail what this means. A first conclusion is that if we want to understand what these repeatable points represent, we have to remove the noisy points that are not repeatable. To do so, we establish a relative threshold between the left and most repeatable observed point. Let $s \in [0, 1]$. We keep an inlier/outlier i observed n_i times if $n_i \geq \min_i(n_i) + s(\max_i(n_i) - \min_i(n_i))$, and exclude it otherwise. $s = 0.2$ / $s = 0.8$ resp. seem to be good starting points to remove all the "noisy", *i.e.*, non-repeatable, inliers/outliers.

3.4.2 Methods to filter features on moving objects with outliers

We realized by observing the pose of outliers during motion that **outliers are concentrated on objects when they start moving**, and this is the key idea to make masks. We call clusters of outliers **dense outliers**. Figure 3.20 shows a scene right before and after a person gets up. For illustration purposes, we increased the search window used to compute 2D-3D matches from 2px to 20px to force the appearance of more outliers. The reason being that what we see in this figure are outliers from only one SLAM run, while outlier collection occurs over many SLAM runs. It is not recommended to use such large search windows as it lowers the quality of keypoint matching and consequently SLAM accuracy.

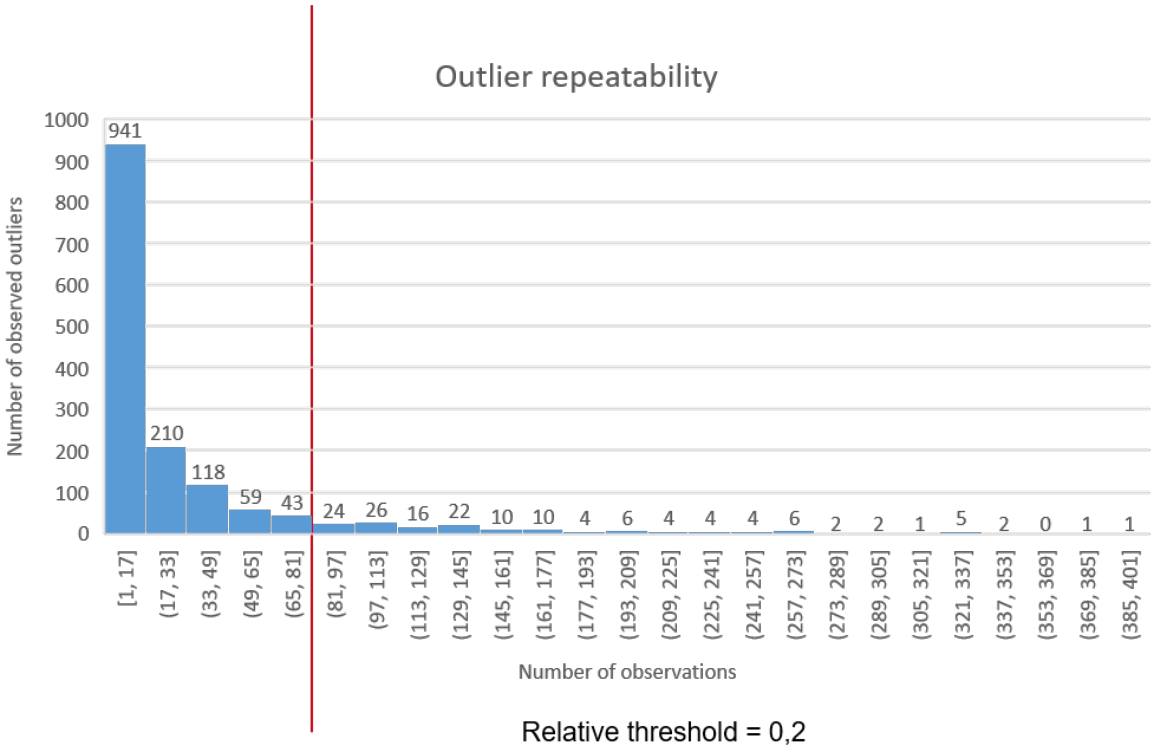


Figure 3.19 – Typical repeatability profile of ORB-SLAM 2 outliers. Relative threshold = 0.2.

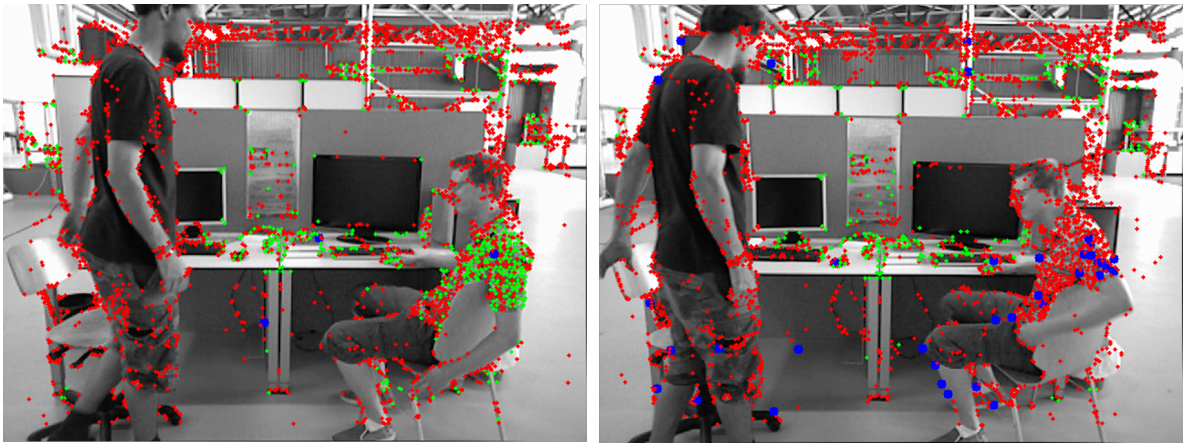


Figure 3.20 – Scene before and after a person gets up (10 frames apart). Red: unmatched features, green: matched features (inliers), blue: outliers. We see that all inliers on the sitting person vanished and were replaced by either unmatched features or outliers.

Figure 3.21 shows an image: in RGB, in grayscale (ORB-SLAM 2 transforms images to grayscale before processing them), and mapped point at instant t. This image is taken right at the moment the person starts moving his upper body. Figure 3.22 shows the corresponding outliers with different thresholds s: 0.0, 0.2 and 0.5.

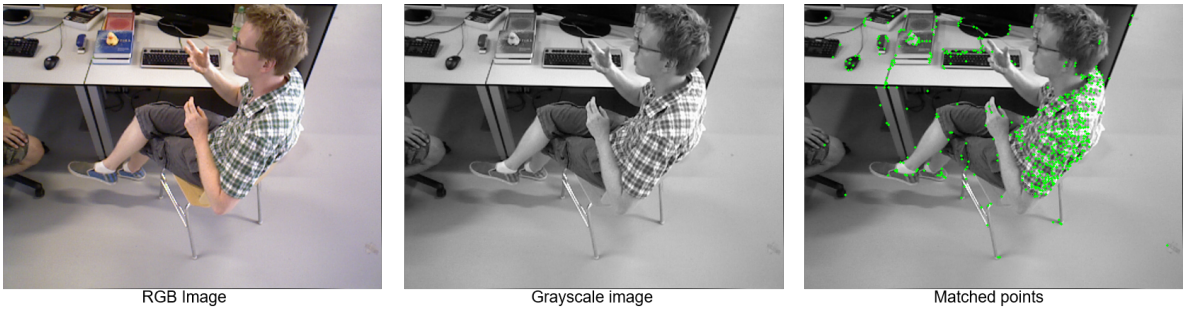


Figure 3.21 – Image where a person before it starts moving its upper body. From left to right: original RGB image, grayscale version, and grayscale version + mapped points.

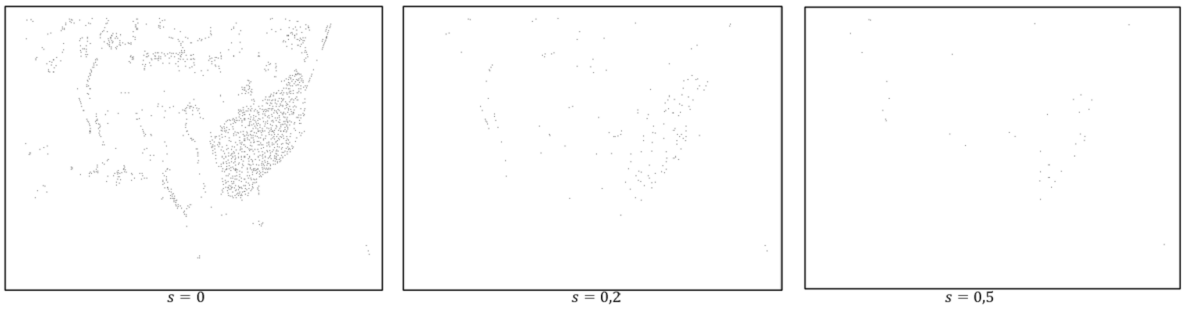


Figure 3.22 – Example of detected outliers after applying different relative thresholds.

We can see that there are very few remaining outliers at or above $s = 0.2$ and they are mostly on the person. **This indicates that there is a strong relation between moving objects and repeatable outliers**. At this point it would be opportune to build a mask from the outliers for future learning, in order to generalize to other sequences. We first evaluate what would happen if in subsequent SLAM executions we filter these repeatable outliers.

Table 3.4 shows extended statistics on tests we did by filtering features found at the same place where repeatable outliers where previously collected. It shows that while direct filtering of outliers does not appear to affect significantly the mean/median accuracy, it has the surprising effect of limiting the max error and reducing the standard deviation. This is a strong indicator that removing features likely to become outliers reduces the randomness of the SLAM and in fact *stabilizes* it, reducing the chance of large errors. During the experience, we noticed that this stabilization effect appears only if we collect outliers from at least 100 SLAM runs.

	Mean		Median		Standard Deviation		Min		Max	
Outlier filtering	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
fr3_long_office_household	12.18	8.07	9.07	7.34	7.46	**2.83**	4.97	4.95	39.01	**23.01**
fr3_nostructure_texture_near_withloop	10.51	11.38	9.54	11.46	3.06	**2.34**	6.26	6.94	19.10	**16.10**
fr3_sitting_halfsphere	33.00	18.21	13.07	14.96	74.32	**11.23**	8.61	9.32	459.76	**81.33**
fr3_sitting_static	2.64	3.55	0.84	3.97	3.11	**2.27**	0.12	0.36	12.42	**10.78**
fr3_sitting_xyz	5.95	9.75	5.26	6.12	**3.90**	14.47	3.46	3.22	26.94	**102.00**
fr3_structure_texture_far	11.03	11.22	10.92	11.22	2.02	**1.55**	6.69	7.90	15.58	**14.85**
fr3_structure_texture_near	18.76	16.25	15.72	15.63	14.04	**4.03**	8.94	10.17	109.26	**26.70**
fr3_teddy	22.75	23.53	20.93	24.10	10.46	**4.57**	2.67	11.52	84.35	**32.92**
fr3_walking_static	2.21	2.04	2.07	2.05	0.74	**0.07**	0.60	1.85	6.09	**2.19**

Table 3.4 – Statistics on the ATE RMSE (mm) on 100 SLAM runs on the first 700 images of various sequences of the TUM RGB-D dataset. We filter features found at the same place where repeatable outliers were previously collected (relative threshold $s = 0.2$). *No* means no outlier filtering and *Yes* means that we filter outliers. Lowest standard deviations and maximums in bold.

Having confirmed that removing outliers has a positive effect on the SLAM, we tried to make a dense mask to evaluate how masking whole areas of the image affects performance. Figure 3.23 shows early tests where we drew disks on the outliers with various radii. We also tried dilating these disks. Unfortunately, as there is a non-negligible number of outliers outside moving objects, increasing R too much caused the removal of too many inliers and caused the SLAM to fail.

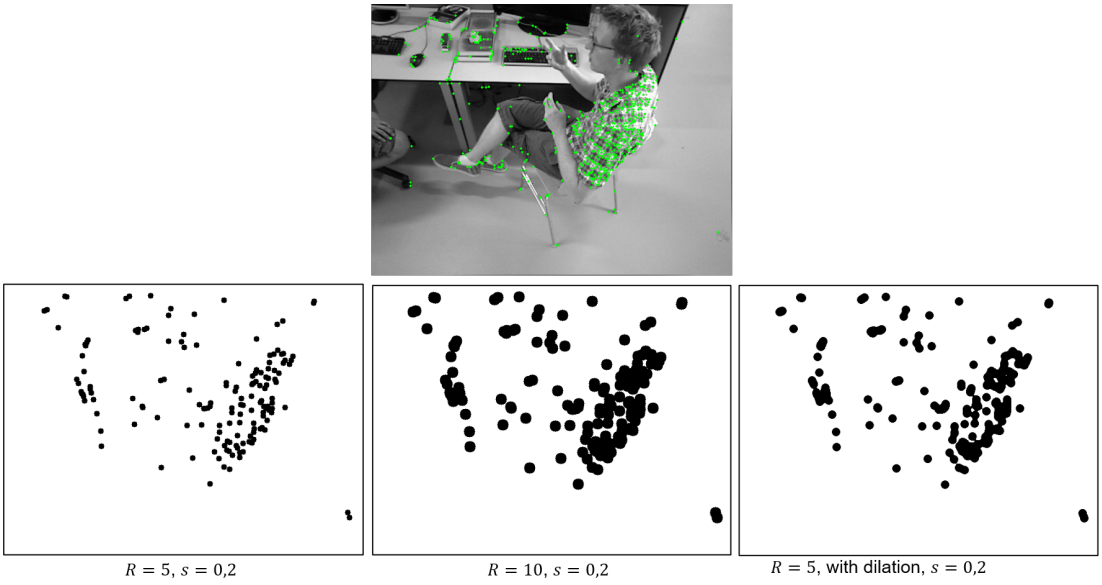


Figure 3.23 – Dense masks constructed from outliers collected with the relative threshold $s = 0.2$. R indicates the radius.

We then identified repeatable inliers and outliers with relative thresholds resp. s_{inlier} and $s_{outlier}$, built masks as previously and then removed the points where there are known inliers. This led to the creation of "perforated masks" as we can see in fig. 3.24. They did improve SLAM performance, but such irregular masks seem exceedingly difficult to learn. We tried tweaking R and s, but it did improve the masks. For this reason, we searched other densification strategies.

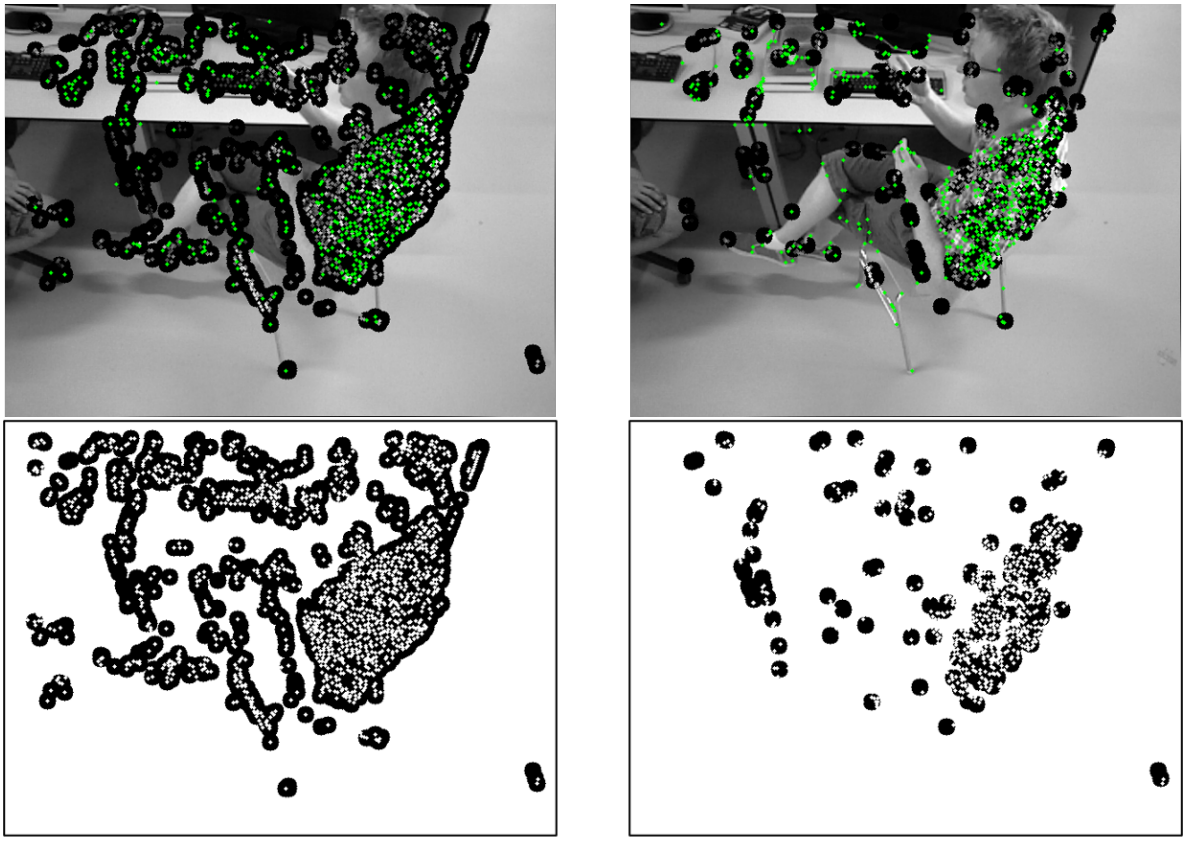


Figure 3.24 – Masks with holes constructed from outliers and inliers. Locations with repeatable inliers are unmasked (in green). The masks are built by collecting inliers and outliers over 100 executions, while the displayed inliers in green correspond to the current execution, so inliers do not perfectly fill the holes.

3.4.3 Methods to densify outliers into masks of moving objects

Superpixels. We first tried using SuperPixels [1] to densify outliers. We tried this on simpler scenes, as illustrated in fig. 3.25. To do so, we compute the ratio of outliers within superpixels and check if it is below a certain threshold. We tried dozens of different thresholds and other ad hoc criteria, but none truly worked. Figure 3.26 shows a trial where superpixels are colored according to the concentration of outliers. Overall, the method does not perform well enough. Areas with many outliers are indeed identified, but the masking rarely masks objects fully and tends to mask areas that naturally generate outliers, as ambiguous/repetitive textures (*e.g.*, chessboards, squared paper).

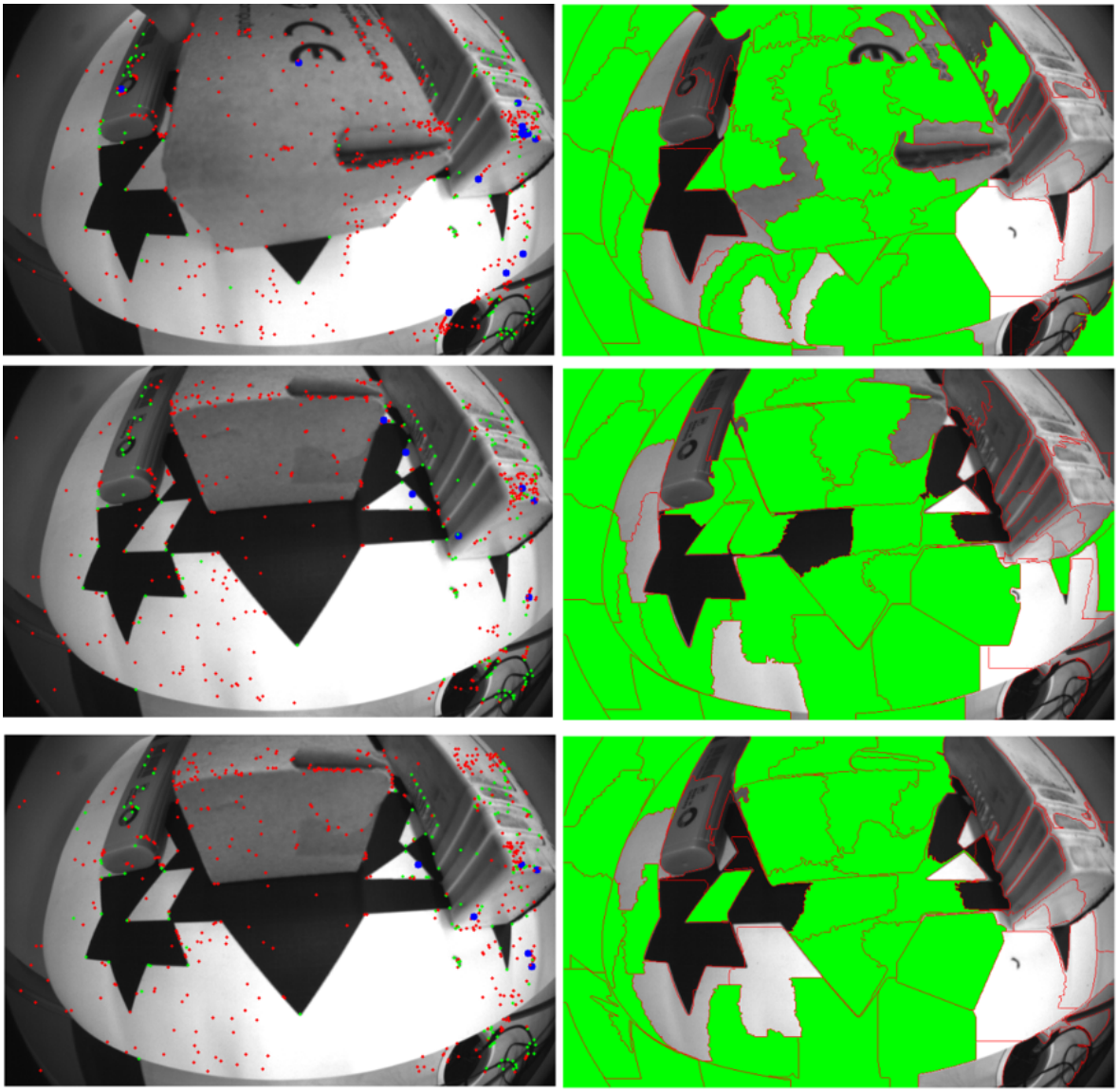


Figure 3.25 – Computation of superpixels in a simple scene, various sets of parameters. The box at the center is moving. Superpixel size = 75px. Areas in green show superpixels marked as dynamic. Red points are unmatched features, green points inliers and blue points outliers.

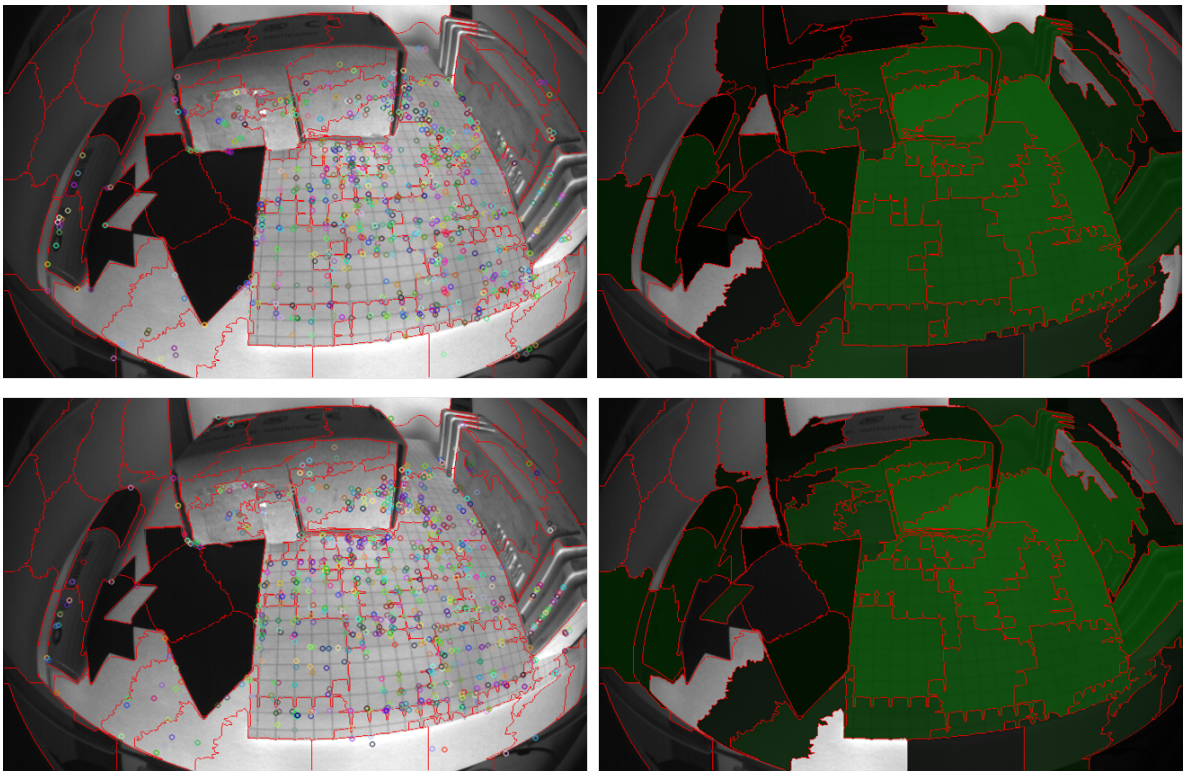


Figure 3.26 – Computation of superpixels in a simple scene, various sets of parameters. The box at the center is moving. Superpixel size = 75px. Areas in green show superpixels marked as dynamic, with a darker green corresponding to a higher ratio of outliers. Red points are unmatched features, green points inliers and blue points outliers.

Learning masks. Being stuck on the densification of outliers, we decided to check if masks could be learned in the first place. To do so, we annotated a toy sequence with masks, as illustrated in fig. 3.27, and trained a DeepLabv3+ model [19] with them. We can see that we can successfully learn such masks. However, we still need a method to automatically compute the annotations. Fortunately, we observed how SLAM performance improved: once problematic objects are masked, the SLAM algorithm does not suffer drifts anymore.

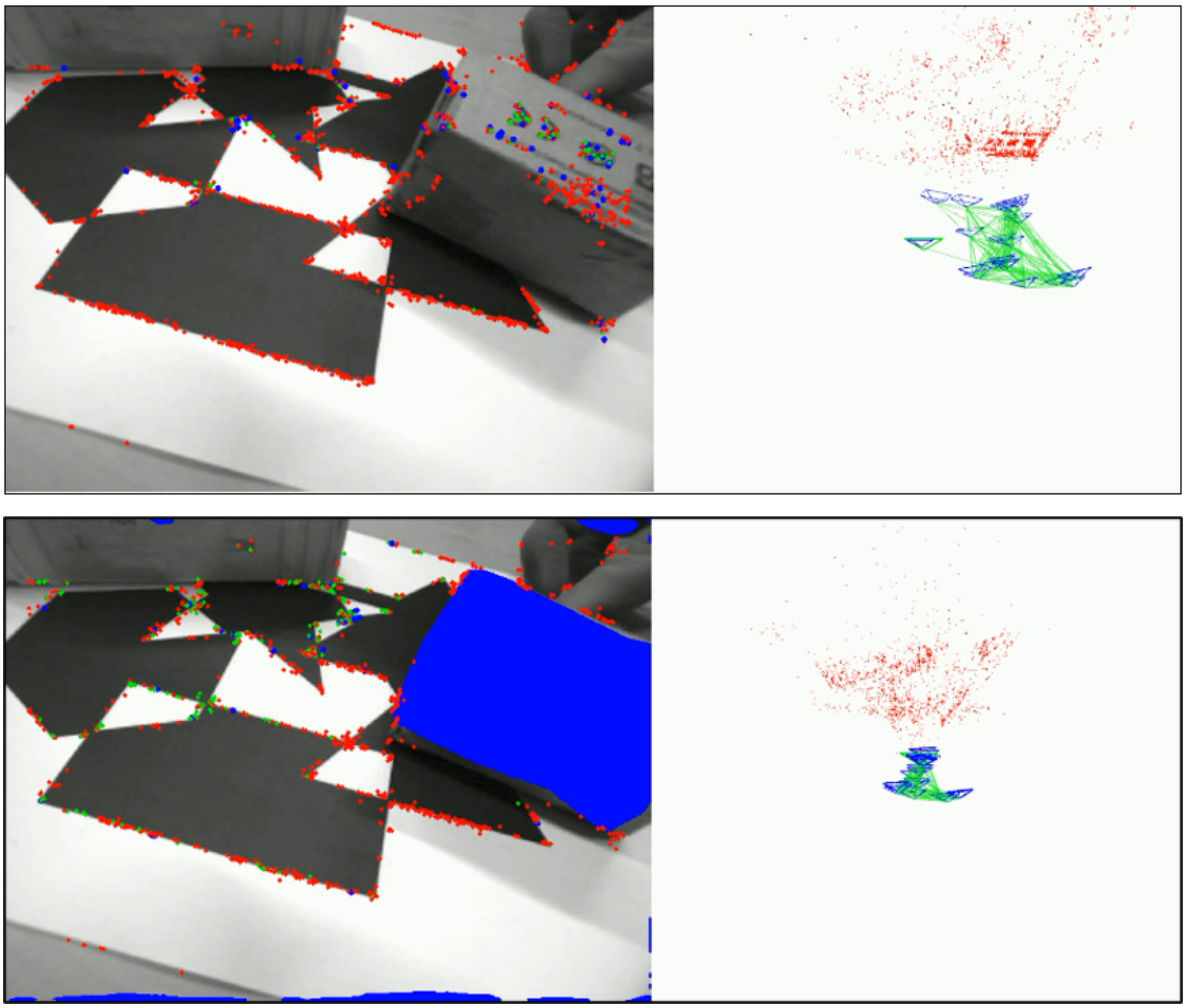


Figure 3.27 – Comparison between a SLAM with no masked object (top) and a SLAM with masked objects (bottom). The mask prevents SLAM failures: the SLAM map (in red) and trajectory (green lines) are correct only if we use a mask. The sequence with no masks shows that the camera (the triangles) repeatedly moves left and right, which is not true. There are some edge effects that were present at this moment due to imperfect training.

RVOS. We evaluated RVOS [100], a video segmentation network. It also masks static objects and sometimes misses dynamic objects, so we abandoned this method.

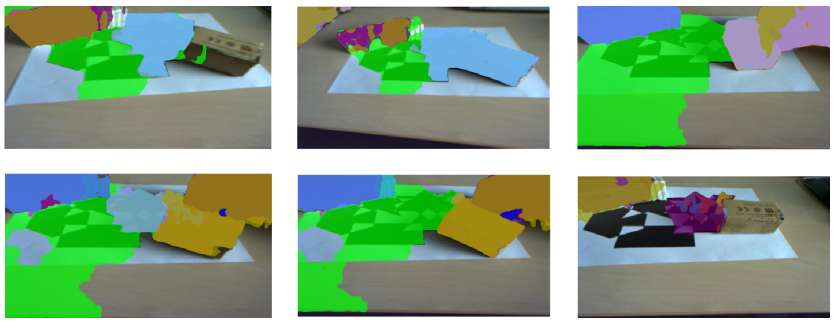


Figure 3.28 – Segmentation of a simple scene using RVOS [100]. The box is moving.

COSNet. We tested another method, COSNet. It has the property of masking "objects of interest", which can be dynamic objects. Figure 3.29 and fig. 3.30 show that the results are accurate. Unfortunately, it does not distinguish between moving and static objects, and loses accuracy when objects are seen from afar or partially occluded, so we cannot directly use it to annotate our sequences.

Figure 3.29 – Example of segmentation using COSNet [66].

Figure 3.30 – Another example of segmentation using COSNet [66].

3.5 The importance of temporality in keypoint filtering for Dynamic SLAM

The experiments on outliers showed that they appear in clusters on objects when they start to move. This highlights the importance of considering the temporal aspect of SLAM for keypoint filtering. In particular, the act of filtering keypoints itself has to consider the temporality of the scene: in other words, it is sometimes useful to filter keypoints, but sometimes it may be counterproductive to do so.

Figure 3.31 and fig. 3.32 show examples of situations where we would mask or not objects. We would mask objects that are moving or when they have a significant risk of moving. Note that the notion of "risk of moving" is subtle as it depends on the spatiotemporal context. In practice, we would mask an object when we know that it can move, *and* we are not sure that we can detect its motion in time – we call this **Temporal Masking**. An example would be when we are behind another car: if we are too close, we cannot be sure if the car in front is moving only from visual information. Temporal Masking could be used to handle both MCIs and excessive masking (we discuss these difficulties in section 4.2.2), which is hardly possible with geometrical methods that rely on instantaneous motion detection. This means that doing a long-term analysis of the scene is necessary to simultaneously overcome motion consensus inversions and excessive masking.

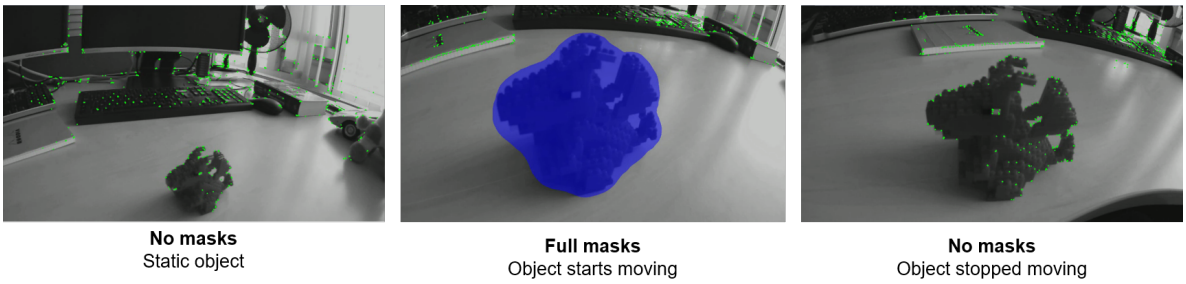


Figure 3.31 – Example of possible masking decisions. We mask an object when it is in motion and not all the time.

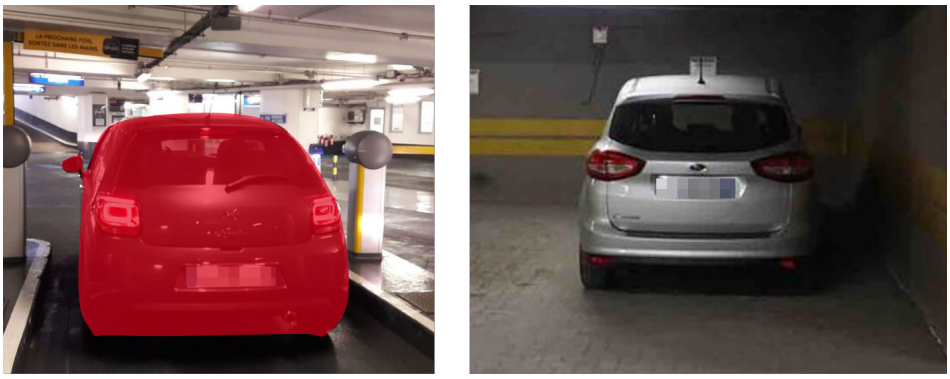


Figure 3.32 – Another example of possible masking decisions. We mask an object when it has a significant risk of motion, even if it has already stopped. In this case, we would mask a car ready to start at a parking payment terminal, but not a car that is parked with nobody inside.

We could have an hyperparameter that controls the risk of SLAM failure, since the baselines *No Masks* (never mask anything) and *Full Masks* (mask all possibly dynamic objects) are polar opposites.

Assuming that all dynamic objects can be segmented, full masking ensures that the SLAM never drifts at the cost of failing early in certain situations. On the other hand, never masking objects would prevent early failures but risks motion consensus inversions. In a situation where the risk of computing an erroneous trajectory is unacceptable – like for autonomous vehicles where a bad pose estimation could cause an accident – we would tend towards full masking. On the opposite, in situations where the top priority is to keep the SLAM running – like for rescue robots – we would tend towards no masking. The risk control could be a runtime setting that controls how much risk of not masking moving objects we accept.

While this idea is promising, it makes the creation of training data too complex. Making the notion of risk implicit to the training data and, consequently, the trained model appears as a better approach. Moreover, it is essential to make our approach self-supervised, since self-supervision explicitly would ensure that the SLAM algorithm knows what frame of reference the user expects.

To summarize, the temporal aspect of keypoint filtering cannot be ignored. The crucial information that leads to the correct decision of what and when to mask objects is available only at specific times.

3.6 Conclusion

From all our experiments in this section, we now better understand the relation between keypoint and Dynamic SLAMs. We first clarified the place of keypoint and concluded that filtering keypoints with external learned tools offers the performance and generalization capabilities. Then we studied the keypoint detectors themselves, defining the general best range for keypoints (2500-3000) and showing that a proper SLAM evaluation requires both several runs and metrics that integrate the Tracking Rate. After that, we studied the repeatability of keypoint detectors: learned detectors are more repeatable, but repeatability is in general weakly related to SLAM performance, although self-supervision can strengthen this relation. In the last part, we saw that outliers can act as source of information for dynamic objects, that their removal improves the SLAM, and tested a variety of methods including learning manually defined masks. Finally, we discussed the importance of the temporal aspect in keypoint filtering.

The key takeaways of this chapter are: we need better Dynamic SLAM metrics, we can improve SLAM performance by modifying the keypoint detection module with self-supervised approaches, and we need to add a temporal aspect to keypoint filtering, especially at runtime.

At this point, we realized that we had the key elements to propose a solution to achieve our goal of making a SLAM always use the correct frame of reference: a self-supervised method to both learn what to mask and when to mask objects.

Regarding what to mask, we use outliers to give only the **approximate** location of a dynamic object. The idea is that we look for the **sudden appearance** of dense outliers – *i.e.*, areas in the image where outliers replace inliers in temporally close images. Doing so, we can leverage COSNet only on the identified area to segment the dynamic object inside, which is the only object of interest now that we focus on it. From that point, we finally use a semi-supervised segmentation tool called SiamMask [105] to propagate the mask across the sequence. Hence, we have a method to automatically segment objects in training sequences using outliers as a starting point, *i.e.*, a way to automatically obtain labelled data to train a semantic segmentation network.

Regarding when to mask, the idea is to consider both the SLAM and the masking method as black boxes. The annotation of video sequences with binary masking decisions (to mask or not) does not depend on a specific SLAM or masking method; at runtime, we infer masking decisions based only on a global analysis of past frames.

The research in this chapter was essential to identify what we need to do to make an improved Dynamic SLAM, and defined a clear direction for our research.

Chapter 4

SLAM Robustness: Metrics and Datasets

4.1 Introduction

Measuring the performance of SLAMs is essential to ensure progress with respect to the State of the Art. There are two essential criteria: having comprehensive datasets and appropriate metrics. However, it appears that datasets used to evaluate Dynamic SLAMs are limited and that the used metrics are inappropriate in some cases.

Certain research domains have a variety of metrics of similar importance – *e.g.*, in image semantic segmentation, we have pixel accuracy, IoU/Jaccard score, Dice coefficient/F1 score. On the other hand, SLAM research relies almost exclusively on *ATE RMSE*, *i.e.*, *Absolute Trajectory Root-Mean-Square-Error* or variants as in [42]. There are additional metrics as the *Relative Pose Error* or *Tracking Rate* [75, 76], but they are usually strongly correlated to the ATE RMSE – hence rarely a source of new insights and often ignored.

We found a limitation in the current Dynamic SLAM literature: datasets used to evaluate Dynamic SLAMs as the TUM RGB-D dataset [89] are challenging due to the *presence* of moving objects. There is hardly any control over the degree of difficulty of object motions, which is not surprising considering that older SLAMs initially could barely handle any object motion. However, this is not the case anymore. In other words: popular SLAM datasets are today *too easy* given current SLAM capabilities – they cannot be used to correctly evaluate the robustness of modern Dynamic SLAMs. Even worse, we identified scenarios in which it is not possible to correctly measure SLAM performance with the ATE RMSE alone.

In this chapter, we will first discuss in section 4.2 how to measure SLAM Robustness and present two issues that – to the best of our knowledge – never have been properly identified: Motion Consensus Inversion and Excessive Masking. We will then present current datasets and metrics in section 4.3, and finally present our own metrics in section 4.4 and datasets in section 4.5.

4.2 SLAM Robustness

4.2.1 General Evaluation Criteria

The robustness of a system, *i.e.*, *the ability of tolerating perturbations that might affect the system's functional body*[1] is vital for it to be used in real-life situations. For SLAM systems, it means that the localization must remain accurate regardless of the quality of the video input. We identified the following perturbations to be overcome as qualitative criteria for SLAM robustness:

[1] Definition from Wikipedia.

1. **Lack of features / ambiguous textures**: the environment may have repetitive textures that confuse the SLAM, like tree leaves or chessboards, or not enough features for accurate localization like a plain corridor.
2. **Illumination changes**: natural or artificial changes that alter the brightness of the image.
3. **Dynamic objects**, *i.e.*, objects that move.
4. **Fast camera movement**: camera motion degrades feature detection and increases requirements on computation time.
5. **Lifelong SLAM**: long-term localization within an environment that slowly changes over time.

The importance of each criterion depends on the actual SLAM application. SLAM for autonomous vehicles would focus on dynamic object support; SLAM in agriculture would focus on robustness to ambiguous textures; underwater SLAM would focus on illumination changes and lack of features. In this thesis, we focus on the most common perturbation: dynamic objects.

4.2.2 Difficulties of Dynamic Scenarios

Dynamic perturbations are fundamentally different from others in the sense that they break a core assumption of SLAM systems: that the environment is static – unlike lifelong SLAM and other SLAMs, major changes easily occur in a few frames. Note that this is a major difference as the environment itself changes: the challenge is not only to understand the environment, but to adapt to it continuously and quickly as changes occur.

The typical effect of dynamic perturbations on the SLAM is that the SLAM integrates features on moving objects while mapping the environment. This causes SLAM drift and lowers the accuracy of the computed trajectory. However, there are two failure cases that are almost non-existent in the current literature: motion consensus inversions and excessive masking. We later show in section 4.3.3 that these scenarios, in addition to being rare in datasets, are not appropriately quantified with the current metrics. These two issues are one of the main challenges that we tackle in our work.

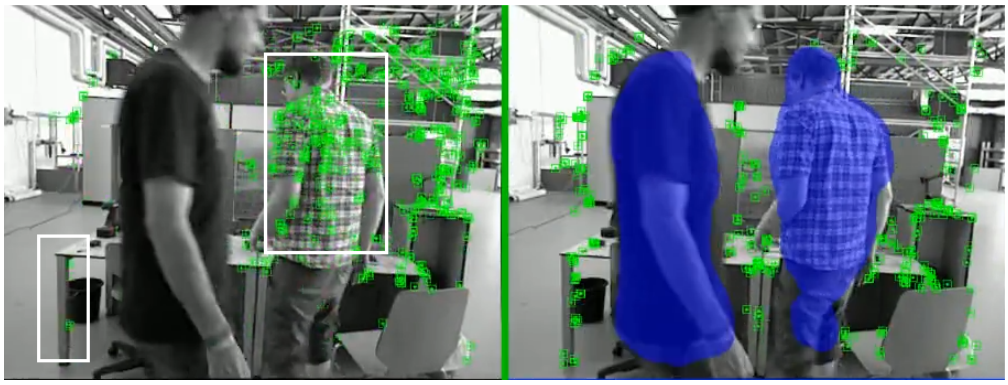
Motion Consensus Inversion (MCI)

The first difficulty is what we call *Motion Consensus Inversion* (MCI). We define an MCI as the event when a SLAM considers a moving object that covers most of the image as part of the background, *i.e.*, everything that is fixed with respect to the frame of reference expected by the user – usually the ground. In other words, under an MCI, the SLAM misidentifies the motion consensus of the background, on which it depends, effectively inverting background and foreground. Consequently, the SLAM will follow the object as it moves and will *ignore* the rest of the image, while it should do the exact opposite. Figure 4.1 shows various examples of motion consensus inversions.

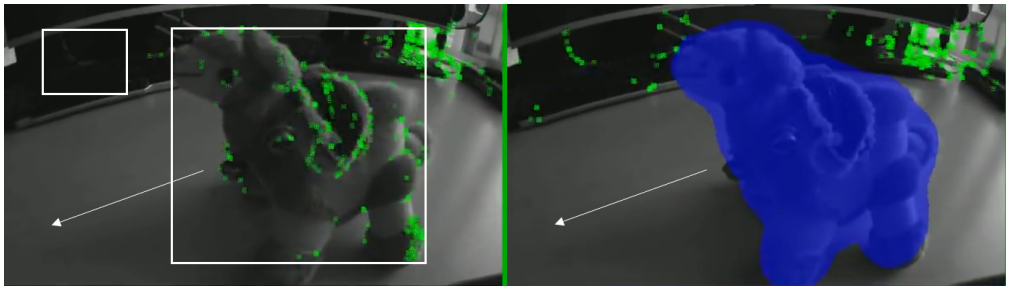
The danger in real-world applications is that the trajectory error is not only very hard to detect as the SLAM does not stop tracking, but also unbounded. For instance, consider a car following another: if the SLAM considers the car in front as part of the background – that is, it suffers from motion consensus inversion –, it may compute that the camera is not moving even if the cars are moving at high speeds, since the *relative* position of the cars does not change. This is, of course, potentially catastrophic.

Note that the consensus inversion is in relation to the frame of reference expected by the user, whatever it is. An extreme example would be a camera moving inside a moving train. If we considered the train as the frame of reference (*i.e.*, the "ground"), then the SLAM would not suffer from MCIs unless the camera looks outside. On the other hand, if we wanted to know the pose of the train, the frame of reference would be the Earth: in this case, if the camera looked inside the train, the SLAM would suffer from an MCI.

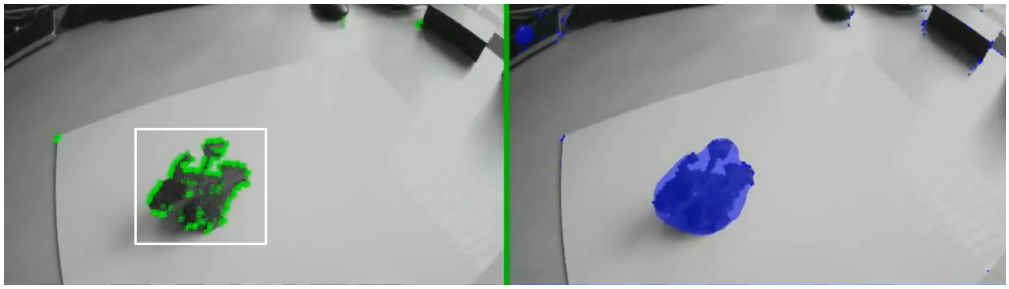
(a) Severe MCI in the ConsInv-Outdoors dataset with a moving car. In the left figure, useful features on the ground are missing while the majority of features is on a moving car, which causes SLAM drift.



(b) Severe MCI in the TUM RGB-D dataset with people walking. In the left figure, useful features on the table are missing while a considerable number of features is on a moving person, which causes SLAM drift.



(c) Severe MCI in the ConsInv-Indoors dataset with a moving object. In the left figure, useful features in the background are missing while the majority number of features is on a moving object, which causes SLAM drift.

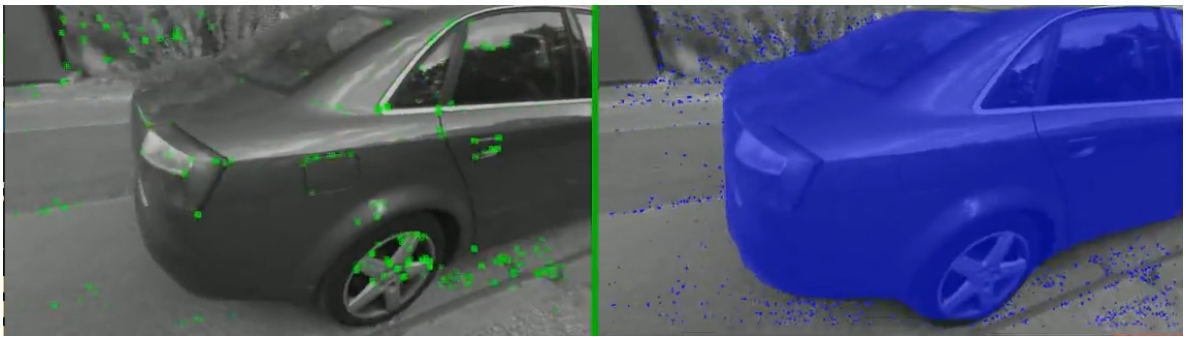


(d) Extreme MCI in the ConsInv-Indoors dataset with a moving object. The plate and dragon are rigidly moved. There are not enough features *not* on the dynamic object for localization. Thus, when the dragon is not masked, the SLAM relies almost exclusively on it for localization, resulting in a SLAM drift. The expected behavior is for the SLAM to stop tracking as long as only the plate or dragon are visible.

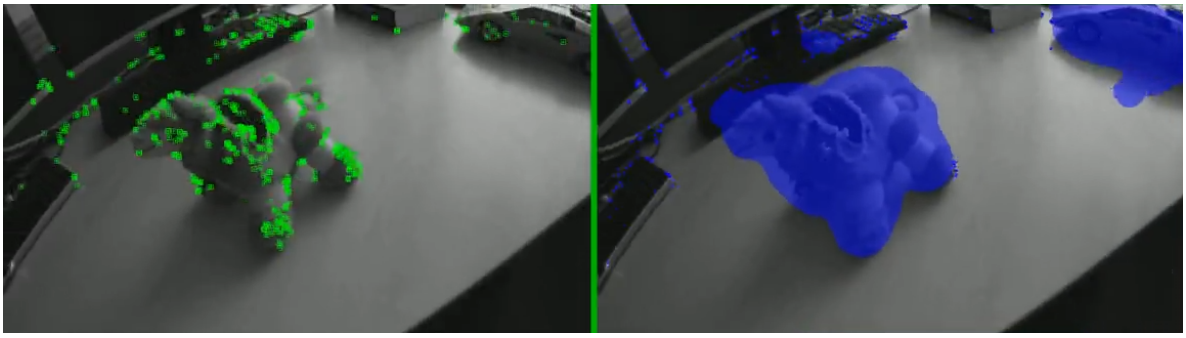
Figure 4.1 – Examples of Motion Consensus Inversions (MCI). Green points are mapped features. Pictures on the left show map points from a SLAM without masks and the right one points from a SLAM with masked dynamic objects.

Excessive Masking

Another difficulty that has not been studied in detail is excessive masking, which is specific to Dynamic SLAMs. A simple Dynamic SLAM strategy is to filter features that are on dynamic objects to prevent them from being used by the SLAM. While this strategy is effective to prevent MCIs (assuming that all dynamic objects are properly masked), it tends to mask all objects that *might* move, even if they are not actually moving. This means that if a dynamic object that is not moving takes 100% of the image, masking it leaves no features whatsoever, which makes the SLAM inevitably fail. In practice, lower ratios (*e.g.*, 70%) may still be high enough to make the SLAM fail due to lack of features. Figure 4.2 shows examples of situations where excessive masking leads to SLAM failures.



(a) Excessive masking in the ConsInv-Outdoors with a static car. In the right figure, most features are removed by masking the car, leaving not enough of them for the SLAM to work and causing it to fail.



(b) Excessive Masking in the ConsInv-Indoors dataset with a static object. In the right figure, most features are removed by masking the object, leaving not enough of them for the SLAM to work and causing it to fail.

Figure 4.2 – Examples of Excessive Masking leading to SLAM failure. Green points are mapped features and blue points are unmapped features due to a SLAM crash. Pictures on the left show points from a SLAM without masks and the right one points from a SLAM with masked dynamic objects.

Dealing with excessive masking is quite tricky in the sense that it appears due to the strategies intended to remove the negative effects of dynamic objects on the SLAM in the first place. Even worse, excessive masking often does *not* lower trajectory accuracy since it makes the SLAM directly fail – it might even improve the accuracy! This counter-intuitive behavior may occur when the SLAM fails early. For instance, if we had a video used for SLAM that is more difficult in its second half, a partial trajectory limited to the first half of the video would be more accurate than a trajectory that spans the whole video. We discuss in depth the limits of current metrics in section 4.3.3.

4.3 Current Metrics, Datasets, and their Limitations

We present in this section the current metrics, datasets and discuss their limitations. The takeaway is that there is not any current metric that can correctly compare SLAM algorithms alone, especially in difficult dynamic scenarios that are rarely present in current datasets.

4.3.1 Core Metrics

The methods for evaluating the performance of SLAM systems have evolved over a long time, as shown in [55]. Two of the main metrics used today, ATE and RPE, were first presented in [89]. We detail them below.

Underlying theory

We represent a trajectory as a timestamped sequence of poses – a transformation from the world to the body frame – in $SE(3)$. [34, 9] give more details on $SE(3)$.

$SE(3)$ is the space of all rigid transformations in $\mathbb{R}^3$, so it has six degrees of freedom: three for the rotations and three for translations. Rotations are defined in SO_3, the group of all rotations in Euclidean geometry, one per axe. This also means that a pose P can be expressed as a pair (R, t) where $R \in SO(3)$ is the corresponding rotation and $t \in \mathbb{R}^3$ the corresponding translation. Note that P can be expressed as a 4×4 matrix:

$$P = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}$$

This means that given the coordinates of a point x_B in the body frame, the coordinates of this same point in the world frame are $x_W = Rx_B + t$. Given a pose P, we define three operators to compute metrics:

$$\text{trans}(P) \coloneqq t$$
$$\text{rot}(P) \coloneqq R$$
$$\angle R \coloneqq \arccos\left(\frac{\text{tr}(R) - 1}{2}\right)$$

We assume that the estimated trajectory is given by a sequence of poses $P_1, ..., P_n \in SE(3)$ and that the ground truth trajectory is given by a sequence of poses $Q_1, ..., Q_n$. In practice, these sequences may not match one-to-one due to different sampling rate, time to initialize, etc. In this case, it is necessary to match and/or interpolate the data beforehand. In the rest of this chapter, we assume that both sequences are time-synchronized and have length n.

ATE: Absolute Trajectory Error

The main SLAM metric is the ATE RMSE, which has the benefit of quantifying the global consistency of the estimated trajectory. The ATE RMSE is the RMS (Root-Mean-Square) of the difference between estimate trajectory points and the corresponding ground truth points. As the frames of references of trajectories are arbitrary, trajectories first need to be aligned. A popular alignment method is the Horn method [49], which finds the rigid-body transformation S that maps the estimated trajectory $P_1, ..., P_n$ onto the ground truth trajectory by minimizing least-square errors.

The mapped estimated trajectory is, for $i \in 1, ..., n$:

$$E_i \coloneqq Q_i^{-1} S P_i$$

The ATE RMSE is defined as:

$$\text{ATE RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \| \text{ trans}(E_i) \|^2}$$

Note that the trajectory alignment includes scaling the trajectory, thus $S \in Sim(3)$ [34] (which has 7 degrees of freedom). Scaling the trajectory is required for monocular SLAM since estimated trajectories have an arbitrary scale, but not for Stereo or RGB-D SLAM. In this case, a SE(3) alignment makes more sense than a Sim(3) alignment since the scales should be the same.

RPE: Relative Pose Error

The relative pose error is the measure of local accuracy of the estimated trajectory over a fixed time interval Δ. Note that Δ is an hyperparameter. Thus, the relative pose error quantifies the drift of a trajectory.

Let $m \coloneqq n - \Delta$. We define the relative pose error matrix F_i for $i \in 1, .., m$:

$$F_i^{\Delta} \coloneqq (Q_i^{-1} Q_{i+\Delta})^{-1} (P_i^{-1} P_{i+\Delta})$$

The RPE is usually divided into translation and rotation components – note that since they are strongly correlated, it is often enough to compute the RPE only with respect to the translation. Thus, we have for the translation component of the RPE:

$$\mathrm{RPE}_{trans}^{\Delta} = \sqrt{\frac{1}{m} \sum_{i=1}^{m} \| \operatorname{trans}(F_i) \|^2}$$

And the rotation component of the RPE:

$$\mathrm{RPE}_{rot}^{\Delta} = \frac{1}{m} \sum_{i=1}^{m} \angle(\operatorname{rot}(F_i^{\Delta}))$$

The choice of Δ needs to be appropriate (a good baseline is $\Delta =$ frame rate). For SLAM systems, averaging over all possible values of Δ makes sense and removes the hyperparameter Δ:

$$\mathrm{RPE}_{trans} = \frac{1}{n} \sum_{\Delta=1}^{n} \mathrm{RPE}_{rot}^{\Delta}$$

$$\mathrm{RPE}_{rot} = \frac{1}{n} \sum_{\Delta=1}^{n} \mathrm{RPE}_{rot}^{\Delta}$$

However, since this makes the computation time quadratic, it may be necessary to limit the number of sampled relative poses in practice.

TR: Tracking Rate

The tracking is the ratio of frames whose camera pose is known, *i.e.*, tracked. It is useful to detect early SLAM failures.

$$\mathrm{TR} = \text{Ratio of tracked frames}$$

Note that the Tracking Rate may vary significantly with the SLAM mode. A typical monocular SLAM can hardly have a Tracking Rate of 100% since it needs at the very least two frames from different points of view to initialize, *i.e.*, it needs significant camera motion to initialize. On the other hand, Stereo/RGB-D SLAM are much more likely to have a 100% Tracking Rate. The reason is that a monocular SLAM does not have access to depth information, so it triangulates features across different frames to initialize, while Stereo and RGB-D SLAM have direct access to this information. As feature triangulation for monocular SLAM requires camera motion, it may take an arbitrarily long time to initialize. And **having a better Tracking Rate does not mean that the SLAM is accurate**, only that it runs longer without stop. These are two separate concepts.

Robustness Metrics

For the sake of completeness, we mention the robustness metrics proposed in [87], which presents the OpenLORIS dataset for lifelong SLAM. The metrics are **Correct Rate of Tracking** – within a trajectory, the ratio of poses that have an accuracy below certain thresholds –, and **Correctness Score of Re-localization** which quantifies the time to re-localize. While these metrics may improve the interpretation of results by distinguishing "poor results" from results that are "good enough", they still do not make it possible to compare SLAMs with a single value in difficult dynamic scenarios.

4.3.2 Datasets

We list the main SLAM datasets in appendix A. These datasets present a wide variety of contexts: drones in forests, cameras underwater, vehicles in urban or rural environments, offices, industrial halls, plantations, etc. and sensors: monocular/stereo/RGB-D cameras, LiDARs, IMUs, barometers and so on.

Recent datasets tend to be targeted at research in scene understanding for autonomous driving rather than SLAM, *e.g.*, being able to correctly segment and classify objects of interest in the scene. Some examples are NuScenes [18], CityScapes [24] and Waymo Dataset [92]. Another trend is the use of simulators as Microsoft AirSim [86] and CARLA Simulator [86] or virtual photo-realistic datasets as Virtual KITTI [38]. The main issue of such datasets is the domain gap with real world data: the performance of a system on simulated data does not guarantee performance in the real world. On the other hand, the advantage is the perfect control over the environment as well as access to perfect ground truth data.

Very few datasets are challenging due to the motion of objects and not due to other factors, as those we mentioned in section 4.2.1. Even fewer contain *real-world* data. To the best of our knowledge, None of these datasets are suitable for learning as they either include too little data and/or do not propose data splits, forcing researchers to make arbitrary choices, which may cause problems comparing results across different works.

Popular datasets in Visual SLAM research are KITTI and TUM RGB-D dataset as they are good baselines to evaluate SLAM systems as a whole.

KITTI Dataset

The KITTI dataset [42] and its later evolutions include various benchmarks: stereo matching, optical flow estimation, 3D visual odometry / SLAM, depth prediction, 3D object detection. The SLAM benchmark, KITTI Visual Odometry, is – of course – the one we are interested in. We illustrate the KITTI odometry dataset in fig. 4.3. The sequences were taken from a station wagon with two color and two grayscale PointGrey Flea2 video cameras at 10 Hz / 1392×512 pixels, a Velodyne HDL-64E 3D laser scanner at 10 Hz (64 laser beams, range: 100 m), and a GPS/IMU localization unit with RTK correction signals. The ground truth is computed from the GPS/IMU output.

The KITTI Odometry benchmark includes 22 sequences, of which 11 (sequences 00-10) have the ground truth. The other 11 (11-20) do not have a public ground truth and to obtain the accuracy of a trajectory it is necessary to submit it to the KITTI website[2]. For this reason, researchers often use only the 11 sequences with known ground truth to evaluate SLAM algorithms. These sequences consist in driving through a calm city and on a highway.

Train and Test sequences for Odometry/SLAM. Methods that do not use KITTI sequences for training usually test all sequences with known ground truth, 00 to 10 [8, 82]. As the dataset creators do not provide a train/test split, there are two unofficial splits that researchers use: training on sequences 00-08 and testing on 09,10,11 [60, 120, 62], or training on sequences 00,01,02,08,09 and testing on 03,04,05,06,07,10 [112, 111, 107]. Note that this does not apply for methods that do only

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

depth prediction, for which the KITTI Depth Prediction dataset is more suitable. To the best of our knowledge, no learned method uses a validation split.

TUM RGB-D Dataset

The TUM RGB-D dataset [89] is a reference benchmark for SLAMs made of 47 sequences in offices or in a industrial hall. Of these, 8 sequences explicitly include moving people: 4 with people walking, and 4 with people sitting but moving theirs arms or their head. There is a validation split made of 33 sequences whose ground truth is not publicly available, the only way of evaluating a trajectory being to submit it to the TUM RGB-D website[3]. For this reason, researchers rarely use these validation sequences. The dataset was recorded with a Kinect sensor: an RGB camera + an Infrared sensor for depth at 640x480 / 30Hz. The ground truth comes from a motion capture system. We illustrate the dynamic sequences of the TUM RGB-D dataset in fig. 4.4.

Train and Test sequences for basic SLAM (non-Dynamic). Methods that do not use TUM RGB-D sequences for training test arbitrarily chosen sequences from the full dataset, not only dynamic sequences [75, 76]. Popular sequences include *fr1_xyz, fr1_desk, fr1_floor, fr2_xyz, fr2_desk, fr3_str_tex_near, fr3_str_tex_far*, but this choice varies between papers. Learned methods use arbitrary train/test sequences. For instance, [112] uses 19 train sequences and 10 test sequences.

Train and Test sequences for Dynamic SLAM. Methods that do not use TUM RGB-D sequences for training (even if they use external learned tools) usually test 5 to 8 sequences among the 8 dynamic sequences [8, 116, 82, 84]. Popular sequences include the 4 *walking* sequences. The choice of other sequences varies. To the best of our knowledge, there is no Dynamic SLAM that learns only from TUM RGB-D sequences, much less only dynamic TUM RGB-D sequences. Some methods test the generalization capabilities of their models on TUM RGB-D sequences. For instance, [111] trains its model on KITTI sequences 00,01,02,08,09 and [62] trains its model on KITTI sequences 00-08.

4.3.3 Limitations

Regarding Dynamic SLAMs, the main limitation of current datasets is that they are biased towards ATE RMSE optimization. The focus is always to evaluate how accurate a SLAM is. Being accurate is a major criterion for SLAM systems, but it is not the only one: we need to take into account the robustness of the system. In particular, current datasets do not include the two difficult cases explained in section 4.2.2, Motion Consensus Inversions and Excessive Masking.

The subtlety here is two-fold. Firstly, these difficult cases can make a SLAM completely fail and, unless one has the appropriate method, essentially impossible to correctly process. Thus, they appear as pointlessly difficult unless the authors of the dataset perceive that these cases are qualitatively different from the usual moving object perturbations – that do not completely crash the SLAM but only make it less accurate within acceptable limits. Sequences that appear needlessly difficult are unlikely to be included in a dataset. Secondly, we need a way to correctly measure the robustness of the SLAM, especially in difficult situations. The issue is that this is not possible with the ATE RMSE, but one would notice this only if they are testing difficult sequence, especially in the Excessive Masking case, *and* comparing several metrics at once – namely an accuracy metric and the Tracking Rate.

Hence, the Motion Consensus Inversion and Excessive have been absent from SLAM datasets. Interestingly, there exist a few sequences that cause MCIs as some dynamic sequences in TUM RGB-D and some in CityScapes [82], but they likely occurred by chance. Certain datasets come close to the MCI problem [71, 89], but none ever properly defined it nor included sequences to evaluate the Excessive Masking issue. Some papers have almost touched upon the MCI/Excessive Masking issues [82, 3, 109], but do not define them. Overall, the SLAM community is aware of the interest of Dynamic SLAM algorithms and the corresponding datasets – however, the MCI / Excessive Masking difficulties are a blind spot both in terms of datasets and metrics.

[3] https://vision.in.tum.de/data/datasets/rgbd-dataset/online_evaluation

Figure 4.3 – Illustration of the KITTI Dataset [42]

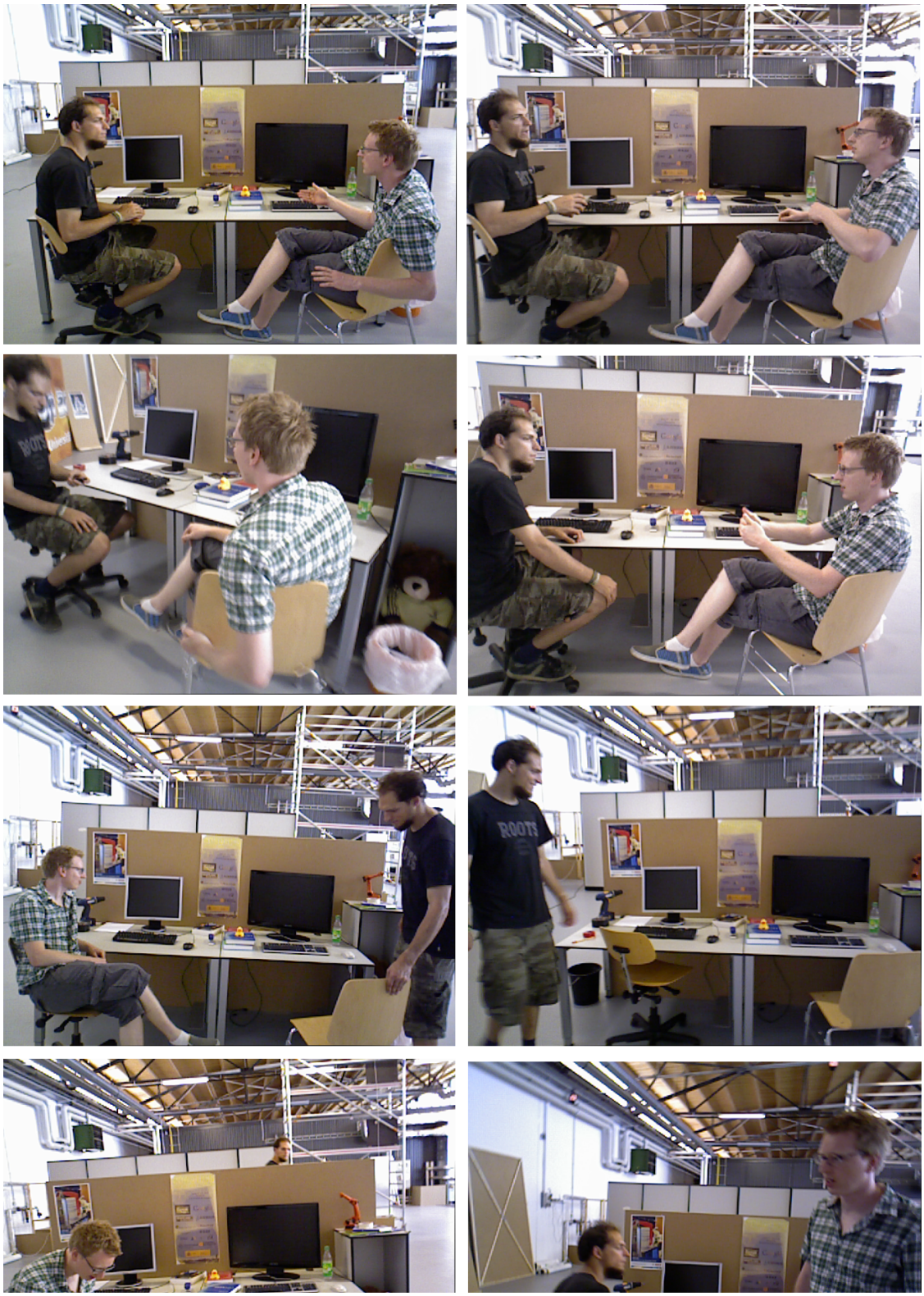


Figure 4.4 – Illustration of the TUM RGB-D Dataset [89]

4.4 Proposed Metrics

As we explained in section 4.3.1, the performance of a SLAM method is usually reported in terms of ATE RMSE, but 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, 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. Thus, it seems important to propose metrics that unify both ATE RMSE and Tracking Rate. We first propose the Penalized ATE RMSE and Success Rate, which we use in chapter 5, then we propose the Unified SLAM Metric, which we use in chapter 6.

4.4.1 Penalized ATE RMSE and Success Rate

If Tracking Rates are too different the comparison of ATE RMSEs is biased: a SLAM that stops early might skip tricky parts of the sequence. Hence, we defined the Penalized ATE RMSE and Success Rate.

We consider that an ATE RMSE is invalid if: 1) it is unknown (*e.g.*, when using reported results) or 2) the Tracking Rate is lower than $\rho - \rho_c$, where ρ_c is a fixed threshold and ρ is the Tracking Rate of the ground truth. The Tracking Rate of the ground truth is the Tracking Rate of the SLAM execution corresponding to the ground truth computation as specified in section 4.5.1, otherwise it must the Tracking Rate we could expect the SLAM in the mode it is run (monocular, stereo, etc.). The idea is to filter out Tracking Rate that are abnormally low while setting a reasonable target.

The Penalized ATE RMSE is computed in relation to other SLAMs in case of invalidation. With τ the penalty factor and L the set of all valid ATE RMSEs computed by other SLAMs on the tested sequence, we define the Penalized ATE RMSE in (4.1):

$$\text{Penalized ATE RMSE} = \begin{cases} \max(L).(1+\tau), & \text{if unknown or } \rho_{gt} < \rho_c \\ \text{ATE RMSE}, & \text{otherwise.} \end{cases} \tag{4.1}$$

The Success Rate of a SLAM on a dataset is the ratio of sequences that the SLAM successfully processes, *i.e.*, whose ATE RMSE is valid.

4.4.2 USM: Unified SLAM Metric

We initially proposed the Penalized ATE RMSE, but 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 value ρ_c as:

$$\beta(\alpha, \rho; \rho_c) = \begin{cases} +\infty, & \text{if } \rho < \rho_c. \\ \alpha, & \text{otherwise.} \end{cases} \tag{4.2}$$

It seems fairly 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. We integrate over all possible values of ρ_c, resulting

into a remarkably simple expression for our Unified SLAM Metric ς:

$$\varsigma(\alpha, \rho) = \int_0^1 e^{-\beta(\alpha,\rho;\rho_c)} d\rho_c = \rho e^{-\alpha} \tag{4.3}$$

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:

$$\varsigma_\lambda(\alpha, \rho) = \rho e^{-\lambda\alpha} \tag{4.4}$$

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 table 4.1.

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 4.1 – Example of score resulting score with the Unified SLAM Metric ς for several values of ATE RMSE α and Tracking Rate ρ. We also report values for ς_5 and $\varsigma_{\lambda=10}$, that propose a smaller relative importance of low Tracking Rates. **Note**: α is in meter in (4.4).

4.5 Proposed Datasets

We present in this section the CI dataset and ConsInv datasets. The goal of both datasets is to test the robustness of Dynamic SLAMs. The CI dataset is the first dataset to explicitly include motion consensus inversions, and it corresponds to our work on self-supervised Dynamic SLAM based on outliers (chapter 5). The ConsInv dataset is a much larger iteration: it includes not only motion consensus inversions, but also sequences to test excessive masking. It corresponds to our work on Temporal Masking (chapter 6). Additionally, ConsInv dataset can be used for learning purposes as it as has train/val/test splits.

We used a MYNT EYE D1000-120 stereo camera at 1280x720 / 30Hz, as illustrated in fig. 2.1 in section 2.1.2. IMU data are recorded but not used. Sequences are about 500 to 1000 images long. Calibration and raw data, including unused IMU data, is available with our dataset.

4.5.1 Ground Truth computation

We compute the ground truth using ORB-SLAM 2 [76] without early stopping in stereo mode and with all dynamic objects masked once any of them moves. We manually annotate the delay. Masking all objects once any moves prevents the SLAM from computing incorrect loop closures due to dynamic object. It also makes it more accurate than if we masked objects only when they move, thanks to the improved bundle adjustment. Note that we prevent the ground truth from being too biased by relying on the stereo mode and double-checking that trajectories computed in both monocular and stereo modes are consistent.

4.5.2 CI Dataset

We created the *CI* dataset (as in Consensus Inversion), illustrated in fig. 4.5, made of two subsets: Dynamic and Static.

CI-Dynamic Dataset

The camera is dynamic in the CI-Dynamic subset. It includes two sequences with static objects (book/notes) and twelve with dynamic objects (dragon/dromedary/car, each object moves in three sequences). This subset covers different situations:

- *Static*: a static object close to the camera. Tests excessive masking, *i.e.*, if the SLAM masks a static object and fails for this reason (as all image features are masked).
- *Easy*: an object moves but does not cause consensus inversion. Tests standard SLAM robustness.
- *Hard*: an object moves and causes motion consensus inversion. Tests SLAM robustness to consensus inversions.
- *Very hard*: an object moves rigidly with the camera while very close to it. Tests the robustness to consensus inversion when detecting object motion is extremely difficult.

CI-Static Dataset

The camera is static in the CI-Static subset. An object close to the camera starts moving and causes a consensus inversion: the SLAM must, however, not compute any motion. We made five sequences per dynamic object.

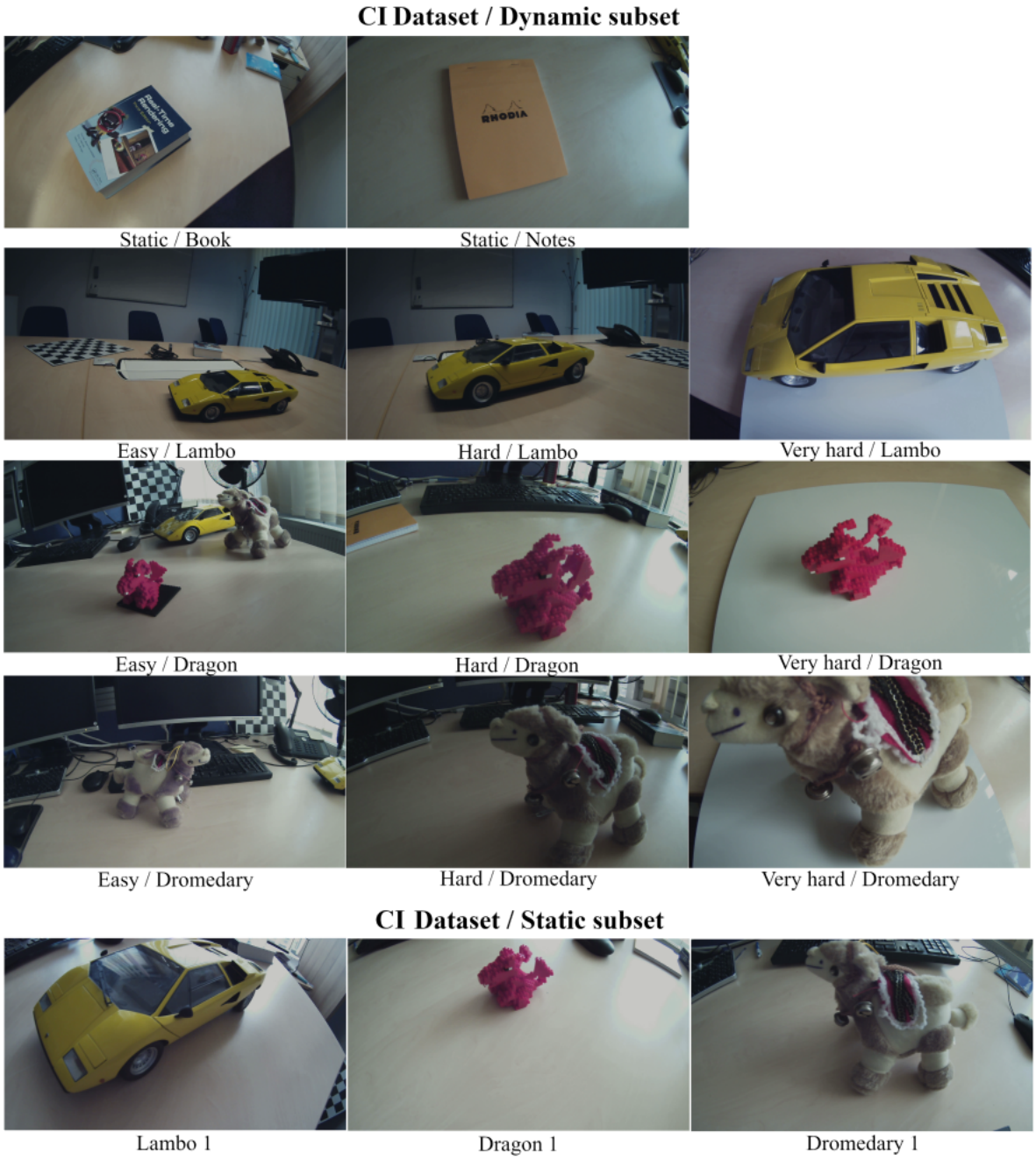


Figure 4.5 – Miniatures of our CI dataset. The camera moves in the Dynamic subset and stays static in the Static subset.

4.5.3 ConsInv Dataset

We present in this subsection the ConsInv dataset (as in Consensus Inversion). It is a major evolution of the CI dataset. 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 experiments on SLAM robustness where at most one object moves at the same time, while the ConsInv-Outdoors dataset is designed for experiments where several objects move at the same time. ConsInv-Extra includes sequences with the same objects as ConsInv-Indoors but in different environments. Camera displacement is limited ($\lessapprox$2m for ConsInv-Indoors/ConsInv-Extra and $\lessapprox 30m$ for ConsInv-Outdoors dataset).

Subset	Train	Val	Test	Total
ConsInv-Indoors-Dynamic	28	12	12	52
ConsInv-Indoors-Static	8	3	9	20
ConsInv-Extra	-	-	18	18
ConsInv-Outdoors	44	10	15	69

Table 4.2 – Number of sequences of the ConsInv dataset.

ConsInv-Indoors Dataset

We built the ConsInv-Indoors dataset, made of the subsets ConsInv-Indoors-Dynamic and ConsInv-Indoors-Static. 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. We include a few sequences with dynamic objects from the CI dataset. We created sequences to evaluate early failures due to masking static objects and to have enough data for training splits.

ConsInv-Outdoors Dataset

We built ConsInv-Outdoors (fig. 4.7) to evaluate SLAM robustness in real outdoor settings. It includes sequences with cars and pedestrians. 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.

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 fig. 4.8 and in a living room (fig. 4.9). These sequences include the same dynamic objects as ConsInv-Indoors. We illustrate these subsets in fig. 4.8 and fig. 4.9. The purpose of ConsInv-Extra is to evaluate generalization capabilities of models trained on ConsInv-Indoors.

Figure 4.6 – ConsInv-Indoors dataset. The camera is mobile in the ConsInv-Indoors-Dynamic subset and fixed in the ConsInv-Indoors-Static subset.

Figure 4.7 – Illustration of the ConsInv-Outdoors Dataset.

Figure 4.8 – Illustration of the ConsInv-Extra-MeetingRoom Dataset.

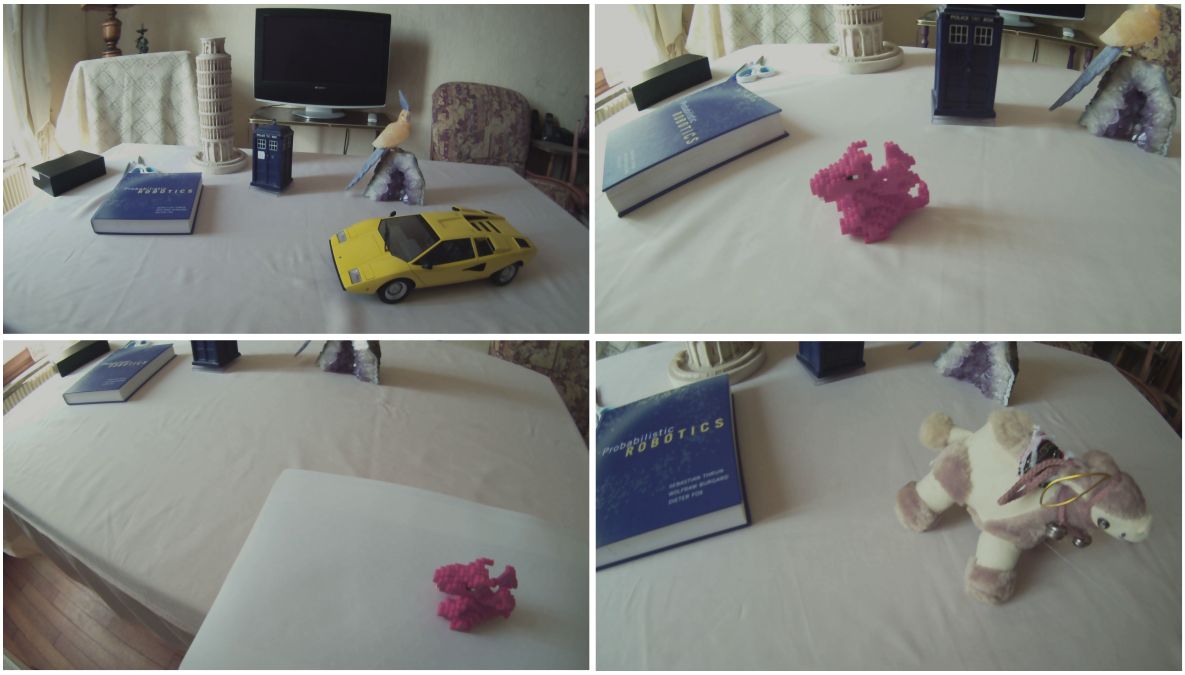


Figure 4.9 – Illustration of the ConsInv-Extra-LivingRoom Dataset.

4.6 Conclusion

We presented in this chapter the difficulties of evaluating SLAMs in dynamic scenarios, detailing two issues that we identified: the Motion Consensus Inversion and the Excessive Masking. We also presented the current SLAM metrics and datasets as well as their limits. Thus, we proposed new SLAM metrics, the Penalized ATE RMSE, the Success Rate, and the USM. We also proposed and new datasets: the CI Dataset and ConsInv Dataset. The key takeaway is that compared to static scenarios, dynamic scenarios have unique challenges that require appropriate – and previously missing – metrics and datasets to be properly tackled.

Chapter 5

From a Robust SLAM to a Dynamic SLAM by Self-Learning of Outliers

We present in this chapter our work on self-supervised Dynamic SLAMs that use outliers as a indirect supervision signal, building upon the work in chapter 3. We detail the resulting method and the corresponding methods experiments. The goal of this chapter is to answer the question: *what* should a Dynamic SLAM mask?

5.1 Introduction

The work in this chapter focuses on two specific difficulties of Dynamic SLAM: the ability to both unknown dynamic objects and to handle motion consensus inversions, which we presented in section 4.2.2. MCIs are rarely studied [82, 5] although it is of high interest in practice. We presented various Dynamic SLAMs approaches in section 2.2. Very few of them can learn automatically to mask unknown objects, namely [5, 118]. [5] requires training sequences recording the same location at different times and needs a full stereo camera + LIDAR setup, while [118] requires an existing semantic segmentation network for bootstrapping and does not work under an MCI. Hence, we propose an approach that at the same time requires little training data and supports unknown objects.

We use geometrical information – SLAM inliers and outliers – to first locate dynamic objects then construct and learn their masks. We build upon the exploratory work in section 3.4: the key idea is that outliers tend to appear in clusters on dynamic objects when they start to move, replacing inliers. Therefore, we use outliers to pinpoint areas of interest, identify the objects inside (which is hard to do without clues on where they are), and create a training database of semantic masks of these objects. We finally use this database to train a semantic segmentation network. We integrate the trained model in an existing SLAM to make it Dynamic. Our approach only needs one monocular sequence per dynamic object, while [5] requires at least two and up to eight in practice. Moreover, unlike [5], we do not need stereo nor depth information, making our approach easier to implement and put into practice. And unlike [118], our approach is not vulnerable to MCIs, especially at runtime.

We use our approach to make ORB-SLAM 2 robust to dynamic objects without any manual annotations. ORB-SLAM 2 is still today one of the best SLAMs and widely used, so improving it is valuable. We also tested transferring the trained masking model into LDSO [39], a feature-based dense SLAM, and the result was a significant improvement in performance showing some inter-SLAM generalization capability.

Note that our approach is different from motion learning [63]. Our approach is about learning to segment objects that generate clusters of outliers, which are *usually* dynamic objects, then masking them. Motion learning consists in learning the motion of objects in a general context: thus, motion learning methods are disconnected from the SLAM context. We discussed in section 1.1 how the gap

between what the user needs and what the SLAM does is the cause for issues as the motion consensus inversion. Widening this gap is likely to make the SLAM perform poorly in difficult scenarios, so we avoid relying on motion learning algorithms to improve SLAM performance.

5.2 Learning to Segment Dynamic Objects

5.2.1 Overview of the method

We outline in this section our approach, illustrated in fig. 5.1, to transform a feature-based SLAM into a Dynamic SLAM by adding the ability to filter dynamic objects. Our goal is to protect the SLAM against the negative effects of dynamic objects in a given environment. We achieve this by training a segmentation network with two classes: *static* and *dynamic*, the latter being masked during the execution of the SLAM. Having detailed classes as *car* or *person* is not needed since we always mask dynamic objects. We show that unconditional masking is more robust than geometrical methods, which fails under consensus inversions. Among mask-based approaches, ours is the only one with automatic annotation and very low data requirements (one sequence per object).

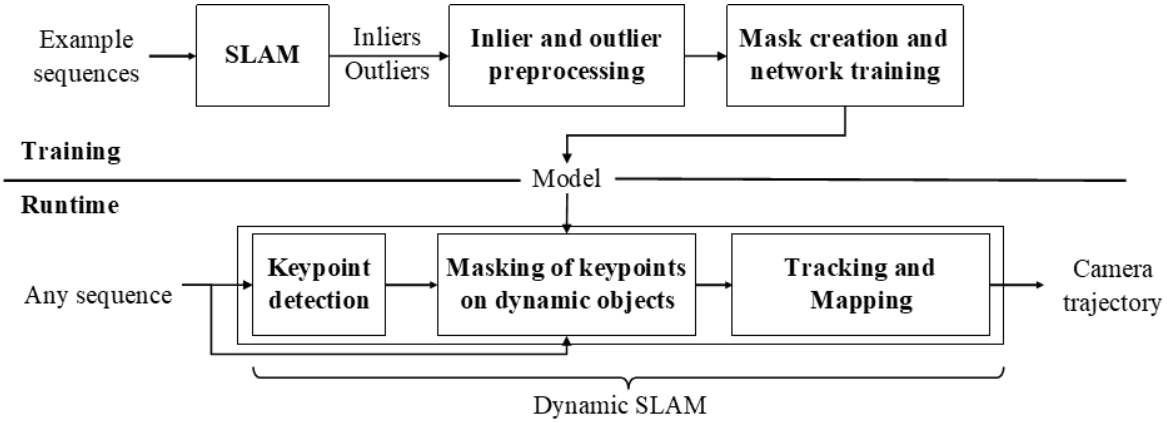


Figure 5.1 – Overview of our approach. We collect inliers and outliers from example sequences and use them to create masks of dynamic objects. We train a semantic segmentation network with the created masks and integrate it in the SLAM after the keypoint detection step. At runtime we infer the masks of any sequence and remove all features on dynamic objects.

SLAMs usually reject non-static features with methods as RANSAC [76]. We make the hypothesis that, if there is no motion consensus inversion, the sudden apparition of dense clusters of outliers characterizes violations of the static world assumption by dynamic objects. Thus, if dense clusters of outliers suddenly replace inliers, it means that the inliers that became outliers were in fact dynamic features, indicating that there is a whole *object* violating the static world assumption rather than just isolated features.

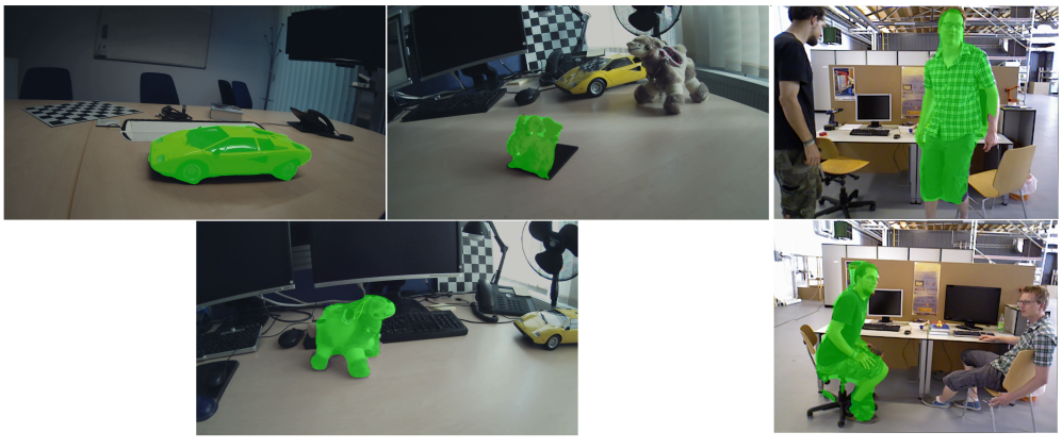
Given a SLAM, we define example sequences as sequences that respect our hypothesis, *i.e.*, make the SLAM generate clusters of outliers on dynamic objects when they move. In practice, example sequences are sequences in which dynamic objects are reconstructed by the SLAM and do not cause motion consensus inversions, *e.g.*, a sitting person that stands up a couple meters away from the camera.

Figure 5.2 illustrates our method. The steps of our approach are:

1. **Outlier and inlier preprocessing**: we use the SLAM to generate outliers and inliers. For non-deterministic SLAMs, we add a filtering step.

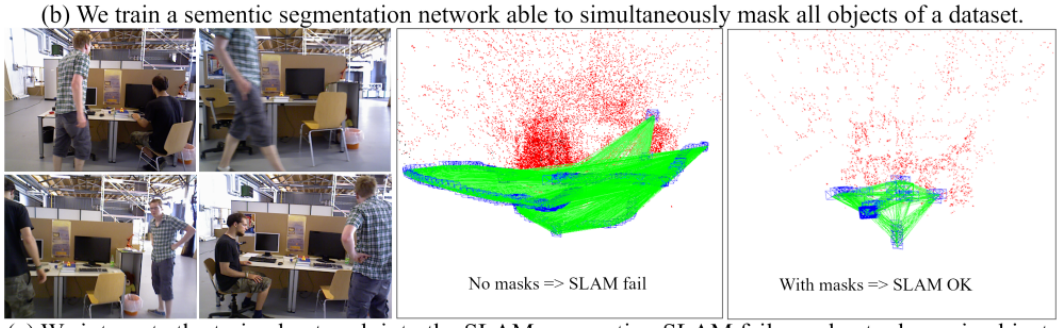
2. **Mask creation and network training**: we use the inliers and outliers to create the masks of dynamic objects in training sequences. This lets us create a complete training database, where the sequences that created inliers/outliers are the network input and the masks the network labels. We the use this database to train a neural network for the image segmentation task.

3. **SLAM Integration and Inference**: we integrate the trained network right after the feature detection step of the SLAM. At runtime, we infer segmentation masks from the input images and use them to filter features in masked areas.



(a) We detect and mask the dynamic objects of monocular sequences (left: our dataset, right: TUM RGB-D).

(b) We train a sementic segmentation network able to simultaneously mask all objects of a dataset.



(c) We integrate the trained network into the SLAM, preventing SLAM failures due to dynamic objects (example with ORB-SLAM 2 in RGB-D mode on *fr3_walking_xyz* of the TUM RGB-D dataset).

Figure 5.2 – Steps of our method. We only need one monocular sequence per dynamic object to make the SLAM robust and prevent major failures due to dynamic objects. We train one network for our CI dataset and one for TUM RGB-D.

5.2.2 Outlier and inlier preprocessing

Once initialized, a feature-based SLAM algorithm computes camera poses for each frame in three major steps:

1. 2D keypoint detection.
2. 2D-3D matching between detected keypoints and known 3D map points + triangulation of new 3D map points.
3. Bundle adjustment: robust optimization of 2D-3D matches and camera poses.

We save the coordinates of outliers and inliers of each frame right after the bundle adjustment: outliers are keypoints whose 2D-3D match was rejected, and inliers those whose 2D-3D match was not rejected. SLAMs may be non-deterministic due to multithreading or random functions (e.g,. RANSAC). We previously discussed this point in section 3.3.3. So, we save inliers and outliers coordinates over several runs and merge them, filtering rarely observed coordinates as they tend to create spurious clusters when merged.

5.2.3 Mask creation and network training

This step is divided in mask creation network training.

Mask creation

this step consists in localizing dynamic objects with sliding windows, refining the windows into masks, and propagating the masks to the whole sequence.

Localizing dynamic objects with sliding windows: on every image of every sequence we use rectangular sliding windows of different sizes, at a fixed stride, to evaluate how the inlier/outlier ratio changes.

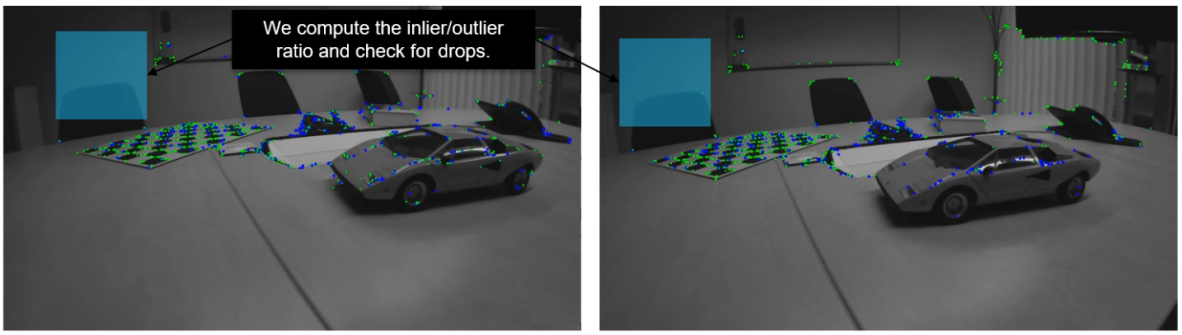
Let w be a window on image p and w' its corresponding window on image p'. Then the outlier score S is:

$$S = \left(\frac{\text{outlier density of } w}{\text{inlier density of } w}\right) / \left(\frac{\text{outlier density of } w'}{\text{inlier density of } w'}\right) \tag{5.1}$$

When a mapped object moves, many inliers become outliers, making the ratio S drop between consecutive frames. We consider that w contains a dynamic object if S is less than a threshold S_{max}, set by the user. Note that this value S_{max} is intrinsic to the SLAM and does not depend on the processed sequence.

We approximately compensate camera motion with the homography $H = K.dR_{p,p'}.K^{-1}$ where K is the camera intrinsic matrix and $dR_{p,p'}$ is the relative rotation between w and w'. We apply H on window w' to have both w and w' match the same physical location. This approximation is easy to compute using a trajectory generated by the SLAM and was accurate enough in our experiments. We illustrate this process in fig. 5.3.

Refining sliding windows into single masks: we merge all bounding boxes that overlap on the same image. Then we project each merged bounding box on the past and future k frames and create image sequences with the content of these projected bounding boxes. The created sequences are a perfect fit for Unsupervised Video Object Segmentation (UVOS, methods that automatically segment salient/dynamic objects in videos) as there is no ambiguity on which object to segment. For each created sequence we apply the author's implementation of COSNet on the central images – a state-of-the-art UVOS network which we already tested in section 3.4.3 –, thus masking the dynamic objects inside the sliding windows.



We look for locations where the inlier/outlier ratio drops between image n and image n+3 with a sliding window.

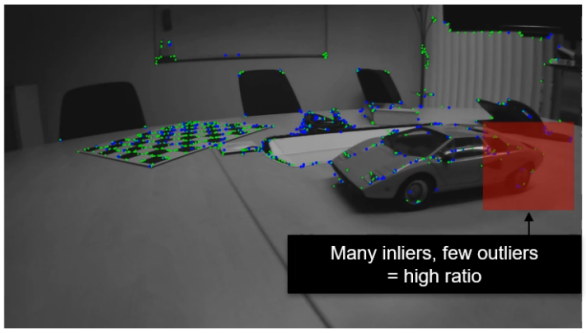


Image n: <u>before</u> the car moves.
(inliers in green, outliers in blue)

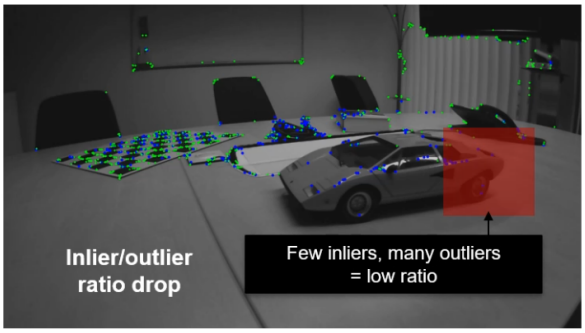
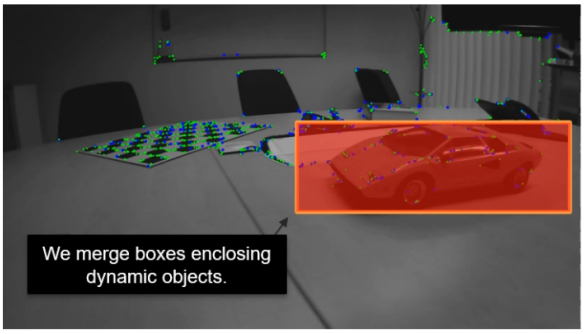


Image $n+3$: <u>after</u> the car moves.



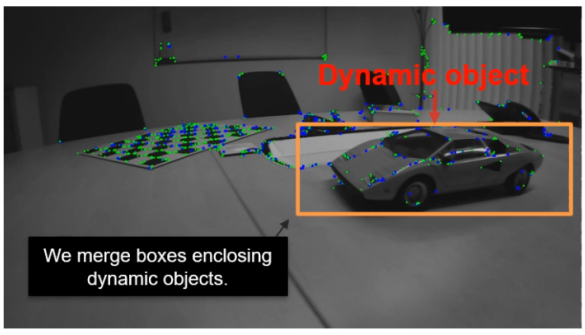


Image n: <u>before</u> the car moves.

Figure 5.3 – Illustration of the sliding window process. The goal is to find areas where the inlier/outlier ratio drops between frame n and frame $n+3$ using a sliding window, and build bounding boxes around these areas. In this scenario, there are ratio drops on the car (the red boxes) since the car starts moving between the two frames. The final result is a bounding box around the areas were ratio drops were found, *i.e.*, the car.

Propagating single masks: now that we have at least a single accurate binary mask for every dynamic object, we can propagate them to past and future frames using semi-supervised video object segmentation. These methods track dynamic objects in videos but require very accurate initial guesses which we have thanks to the previous step. We apply the author's implementation of SiamMask [105], a state-of-the-art network that is both lightweight and class-agnostic, towards past and future frames. The result is a set of binary masks, covering the whole sequence, for each dynamic object. We illustrate this process as well as the previous one in fig. 5.4.

Figure 5.4 – Illustration of the segmentation process. We use COSNet to make to transform the bounding box into a single segmentation mask and SiamMask [105] to propagate segmentation masks across the sequence.

Network training

Our goal is to train a semantic segmentation network able to mask all dynamic objects simultaneously. First, we train one instance of the network for every set of masks generated at the previous step. Then, we infer semantic masks for every sequence and for every trained network. We superimpose the masks inferred on the same sequence and use all superimposed masks to train a final instance of the semantic segmentation network; the computed model can be used to mask all dynamic objects of all sequences simultaneously. Figure 5.5 shows the result: from models able to segment a single object, we train models able to segment all objects at the same time.

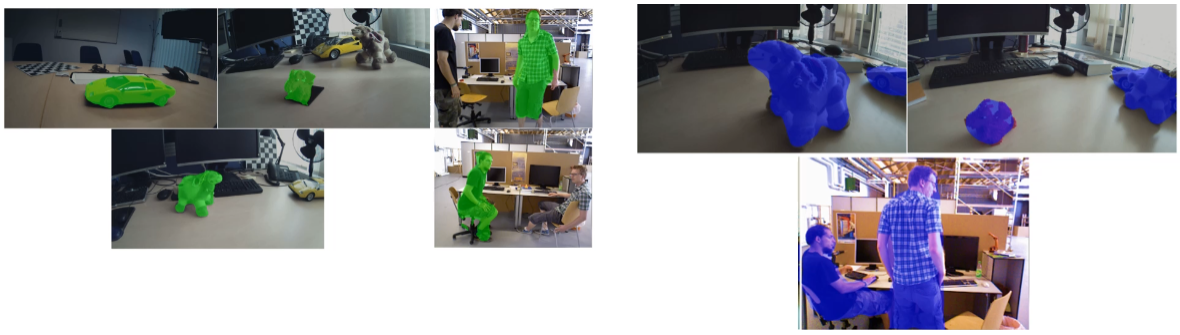


Figure 5.5 – Illustration of the training process. We first train models able to one object class at a time (green masks, left), then combine inferred masks to train models able to mask all of them at the same time (blue masks, right).

5.2.4 SLAM Integration

We integrate the final trained model after the feature detection module in the SLAM. The model will later be used for inference: at runtime, we infer the mask of dynamic objects from the current image, then filter all features whose coordinates are on masked areas. We previously presented the full pipeline in fig. 5.1. The inference part alone is in fig. 5.6.

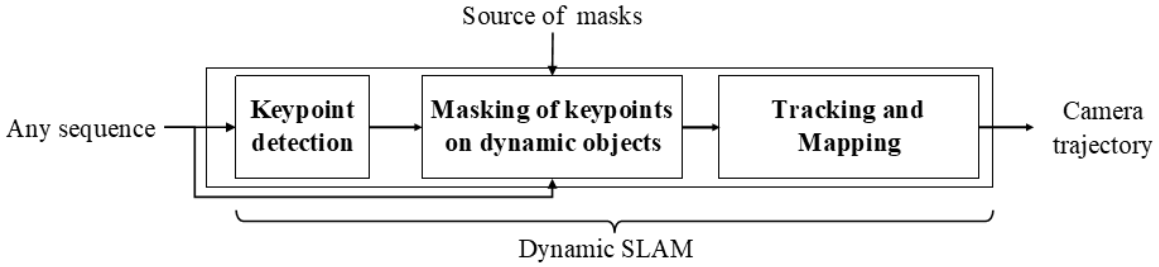


Figure 5.6 – Dynamic SLAM at runtime. It consists in filtering all features on dynamic objects using generated semantic masks.

5.3 Experiments

5.3.1 Experimental setup

We evaluate our method on the TUM RGB-D dataset and on the CI dataset, presented in section 4.5.2.

We use ORB-SLAM 2 as the core SLAM algorithm for the main experiments and LDSO specifically to test the extension to a Direct SLAM. We set the feature number to 3000 following the results from section 3.2 and use default/author settings otherwise. We use our method to train one network on the TUM RGB-D dataset (we use the sequences fr3_sitting_static and fr3_walking_static for training) and one on the CI dataset (we use the *Easy* sequences of the Dynamic subset), presented in section 4.5. For semantic segmentation, we use DeepLabv3+[1] [19], a state-of-the-art semantic segmentation network. We set $\tau = 0.1$ when computing the Penalized ATE RMSE.

While evaluating our method on the easy sequences of the CI dataset we empirically found suitable parameters: sliding windows of size 100x100 / 200x200 / 300x300 / 400x400 with a stride of 50px, a difference of 3 images to compute outliers scores, an interval of at least $k = 30$ images (1s at 30Hz) for the mask propagation step and a max outlier score $S_{max} = 0.15$ to determine if a sliding window contains a dynamic object. We manually adjusted these parameters so as to have qualitatively accurate bounding boxes around dynamic objects.

5.3.2 Results

Comparison with the State-of-the-Art

Dynamic objects, especially when causing consensus inversions, may cause early SLAM failure and decrease in the Tracking Rate of SLAMs, making the comparison of ATE RMSEs biased. To take both the trajectory error and the Tracking Rate into account we use our new metrics: the Penalized ATE RMSE and the Success Rate. The Penalized ATE RMSE integrates failures, so it is directly comparable between SLAMs, and a higher Success Rate expresses that a SLAM is less affected by dynamic objects. We evaluate the methods on the TUM RGB-D and CI datasets.

[1]Source: https://github.com/srihari-humbarwadi/person_segmentation_tf2.0

	State of the Art					ORB-SLAM 2 [76] + ⋯					
						Segmentation baselines					Our seg.
Test dataset	L-K^2[20]	Dyna^3[8]	ST^4[82]	Uni^5[103]	DS^5[116]	No seg.	Mask R-CNN[47]	PWC-Net[91]	RVOS[100]	COSNet[66]	
Consensus Inversion - Mono	0.0547	0.0693	0.0692	N/A	N/A	0.0860	0.0760	0.0237	0.0144	0.0297	**0.0089**
Consensus Inversion - Stereo	N/A	0.0627	0.0699	N/A	N/A	0.0756	0.0630	0.0803	0.0116	0.0148	**0.0094**
TUM RGB-D - Mono	0.0892	0.1108	0.1101	N/A	N/A	0.0252	0.0235	0.0335	0.0331	0.0267	**0.0222**
TUM RGB-D - RGB-D	N/A	0.0206	0.0173	0.0190	0.0802	0.1077	**0.0172**	0.0790	0.0218	0.0245	0.0185

Table 5.1 – Average Penalized ATE RMSE (m) of the State-of-the-Art and baselines on Consensus Inversion/Dynamic and TUM RGB-D/Dynamic datasets. N/A indicates that the SLAM mode is not supported.

	State of the Art					ORB-SLAM 2 [76] + ⋯					
						Segmentation baselines					Our seg.
Test dataset	L-K^2[20]	Dyna^3[8]	ST^4[82]	Uni^5[103]	DS^5[116]	No seg.	Mask R-CNN[47]	PWC-Net[91]	RVOS[100]	COSNet[66]	
Consensus Inversion - Mono	54,5%	63,6%	63,6%	N/A	N/A	45,5%	54,5%	72,7%	72,7%	72,7%	**100,0%**
Consensus Inversion - Stereo	N/A	72,7%	63,6%	N/A	N/A	63,6%	63,6%	63,6%	81,8%	81,8%	**100,0%**
TUM RGB-D - Mono	50,0%	62,5%	62,5%	N/A	N/A	87,5%	87,5%	62,5%	62,5%	**100,0%**	**100,0%**
TUM RGB-D - RGB-D	N/A	**100,0%**	**100,0%**	**100,0%**	87,5%	62,5%	**100,0%**	62,5%	**100,0%**	**100,0%**	**100,0%**

Table 5.2 – Success Rate (%) of the State-of-the-Art and baselines on Consensus Inversion/Dynamic and TUM RGB-D/Dynamic datasets. N/A indicates that the SLAM mode is not supported.

State of the Art			**ORB-SLAM 2** [76] + ⋯					
			Segmentation baselines					**Our seg.**
L-K^2[20]	Dyna^3[8]	ST^4[82]	No seg.	Mask R-CNN[47]	PWC-Net[91]	RVOS[100]	COSNet[66]	
53,3%	60,0%	60,0%	60,0%	60,0%	66,7%	86,7%	80,0%	**100,0%**

Table 5.3 – Evaluation on Consensus Inversion/Static dataset. We report the ratio of sequences that do not cause initialization fails (false starts).

Table 5.1 (cols. 2-6) shows the Penalized ATE RMSE of the State of the Art. Our method performed better than others on our dataset (all modes) and on TUM RGB-D in monocular mode. It is in third place on TUM RGB-D in RGB-D mode.

On TUM RGB-D in monocular mode the results of L-K[2][20], DynaSLAM[3] [8] and Slamantic[4] [82] are explained by the harsh penalty we give to early failures (which happened to the three of them). The original ORB-SLAM 2 already performs well so removing dynamic objects is not really necessary. All Dynamic SLAMs[5] performed well in RGB-D including our method. We reached the standard of other Dynamic SLAMs that rely on manually annotated networks. Figure 5.7 illustrates how the SLAM can output nonsense in presence of consensus inversions (motions that do not exist) and that we prevent it.

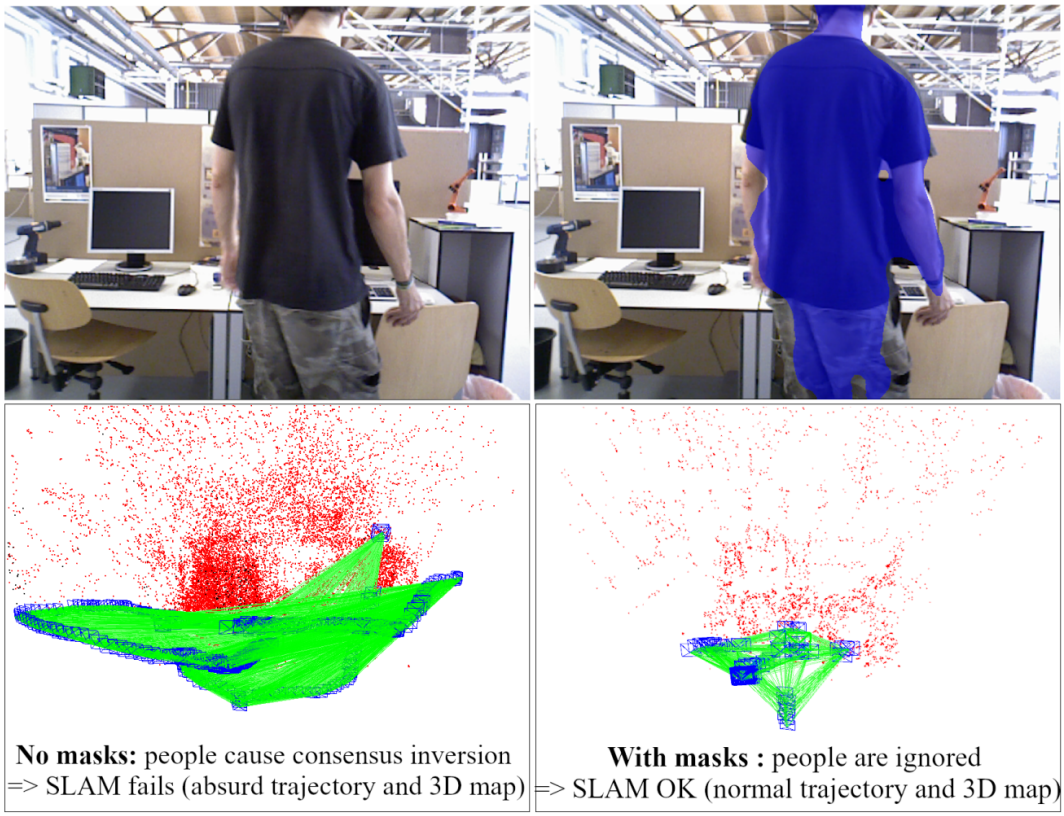


Figure 5.7 – Consensus inversion in fr3_walking_xyz (TUM RGB-D). Camera pose in blue and 3D map in red. Left: no masks, the camera trajectory is nonsensical as the SLAM uses features on moving people. Right: we apply masks using our method, the SLAM trajectory is coherent with the real motion.

On our CI dataset other Dynamic SLAMs performed poorly: they failed in hard / very hard sequences when the object was of an unknown class, *e.g.*, dragon or dromedary. Even Slamantic, that tries not to segment mobile objects that are not moving (*e.g.*, a parked car) also failed in the very hard sequences as the objects are static during most of the sequence.

Table 5.2 (cols. 2-6) shows the Success Rate of the State of the Art. Our approach has the best

[2]We implemented a simplified version of [20] (which uses the Lucas-Kanade optical flow): we warp frames with an homography and we set the optical flow displacement threshold to 2px.

[3]DynaSLAM randomly crashed in RGB-D mode on our system. We refer to the original results in this mode.

[4]The publicly available code of Slamantic does not support monocular mode so we adapted the available stereo code.

[5]We report the results of DS-SLAM and Unified.

performance in all cases. The results are coherent with the Penalized ATE RMSE: a higher Success Rates correspond to a lower Penalized ATE RMSE. Except for TUM RGB-D in RGB-D mode (success rate $\geq 85\%$), all results are below 75%. The results in monocular mode show that it is more difficult to handle monocular dynamic sequences with geometrical approaches, possibly due to the arbitrary scale of the SLAM. The success rates on our dataset (about 60%) shows that both geometrical approaches and hybrid ones combining geometrical/semantic approaches at runtime fail if there are consensus inversions caused by unknown objects.

Our results prove that regarding the robustness to dynamic objects: 1) semantic networks should be fine-tuned to the dynamic objects of the working environment 2) geometrical approaches are unreliable under consensus inversions 3) Hybrid/combined approaches are unreliable under consensus inversion caused by unknown objects.

Comparison with baselines

Unsupervised Video Object Segmentation (UVOS) networks and semantic segmentation networks may appear as trivial solutions to make a SLAM Dynamic as their integration is straightforward.

To test this aspect, we integrated Mask R-CNN (we filter the same semantic classes as DynaSLAM), RVOS [100] (in zero-shot, *i.e.*, unsupervised mode) and COSNet (we only use past frames for inference) in ORB-SLAM 2. We also test a very simple optical flow solution with PWC-Net [91], a learned optical flow network, by masking the area of the image with the 50% most intense optical flow.

Table 5.1 (cols. 7-12) show that we perform better that all baselines. All results are comparable to the State of the Art on TUM RGB-D except for PWC-Net, likely due to its naive integration (if there is no object moving the predicted mask will be wrong). However, on the CI dataset results are quite different: Mask R-CNN and PWC-Net both perform poorly while RVOS and COSNet have very good results. This shows that methods that are class-agnostic perform better than class-aware methods and – considering the other columns of the table – geometrical methods. The main issue with UVOS networks is in fact oversegmentation: RVOS and COSNet failed on the *Static* sequences of the Consensus Inversion / Static subset. They masked the only source of features in the image and made the SLAM fail. The same conclusions come from table 5.2. Figure 5.8 illustrates failure cases.

The conclusion is that UVOS networks are better at making SLAMs robust to dynamic objects than the usual semantic and geometrical approaches. However, they also segment static objects and have an increased risk of failing early in static environments.

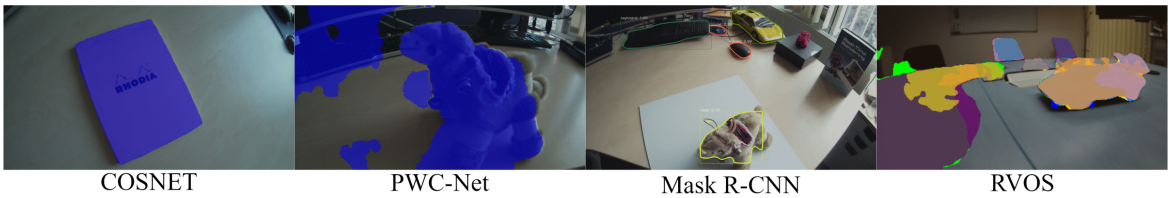


Figure 5.8 – Failure cases of baselines methods (ORB-SLAM 2 + existing network). Mask R-CNN ignores the dragon and considers the dromedary a cake. Other methods segment static objects.

Evaluation of monocular false starts

Monocular SLAMs as ORB-SLAM 2 require the camera to move to initialize, so we evaluate a specific kind of failure: false starts. Figure 5.9 illustrates such a false start: since the camera is static, generated maps or trajectories are necessarily fake. We evaluate the State of the Art and the baselines on the Consensus Inversion / Static subset. We performed best, never initializing incorrectly. All other methods failed, either because the object is unknown (semantic approaches), because the object caused a consensus inversion (geometrical approaches) or because it was not fully segmented

	LDSO [39] + ⋯	
	No segmentation	**Our segmentation**
Avg. Penalized ATE RMSE (m)	0.0833	**0.0581**
Success Rate (%)	36.4%	**63.6%**

Table 5.4 – Average Penalized ATE RMSE (m) and Success Rate (%) of LDSO and our masked version on the Consensus Inversion/Dynamic dataset.

(UVOS approaches). The results show again that it is essential to make a SLAM robust in a specific environment.

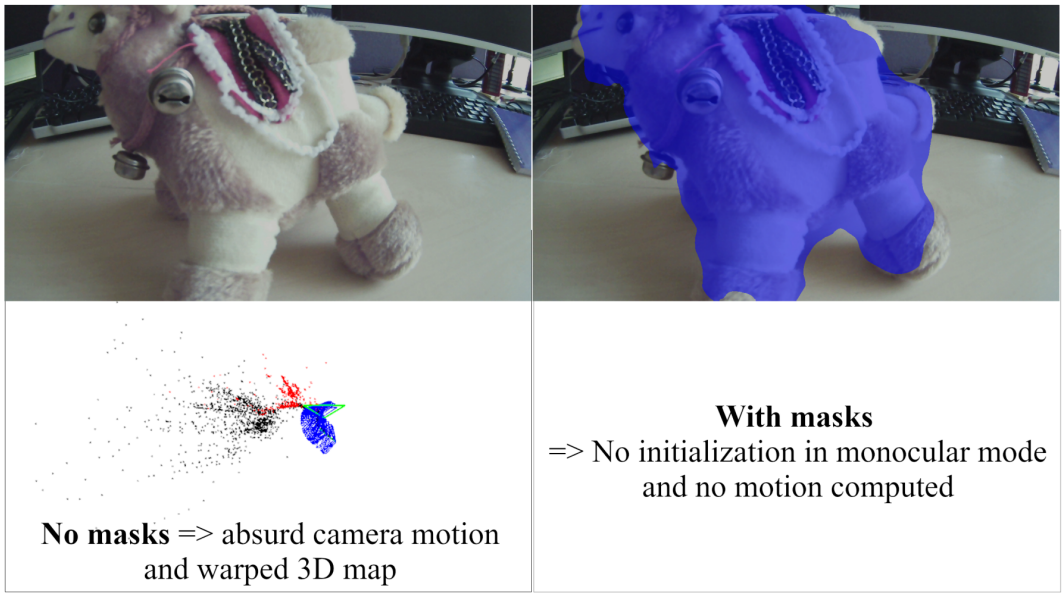


Figure 5.9 – Example of monocular false start. The SLAM cannot initialize as the camera is perfectly static. Yet if the object is not masked the SLAM generates absurd trajectories and 3D maps. Masking dynamic objects prevents such situations.

Extension to a Direct SLAM

We tested integrating the model trained on ORB-SLAM 2 into LDSO, a monocular feature-based, dense SLAM. Since it is feature-based we proceed as for ORB-SLAM 2: we integrate the network right after the feature detector, filtering features that are located on dynamic objects. The input (images) and output (a trajectory) remain unchanged.

Figure 5.10 shows the effect of masking: the 3D reconstruction is incorrect if the object is not masked. However, when masked, the SLAM runs normally. Table 5.4 shows that both the Penalized ATE RMSE and Success Rate improve. While the Success Rate does not reach 100%, the result is very interesting: it is possible to use what we learned from one SLAM algorithm to improve another one.

This shows that our trained model has a certain inter-SLAM generalization capability: it is not limited to a specific SLAM although the training is self-supervised. This makes sense as the model ultimately masks dynamic objects, and dynamic objects are the *raison d'être* of Dynamic SLAMs.

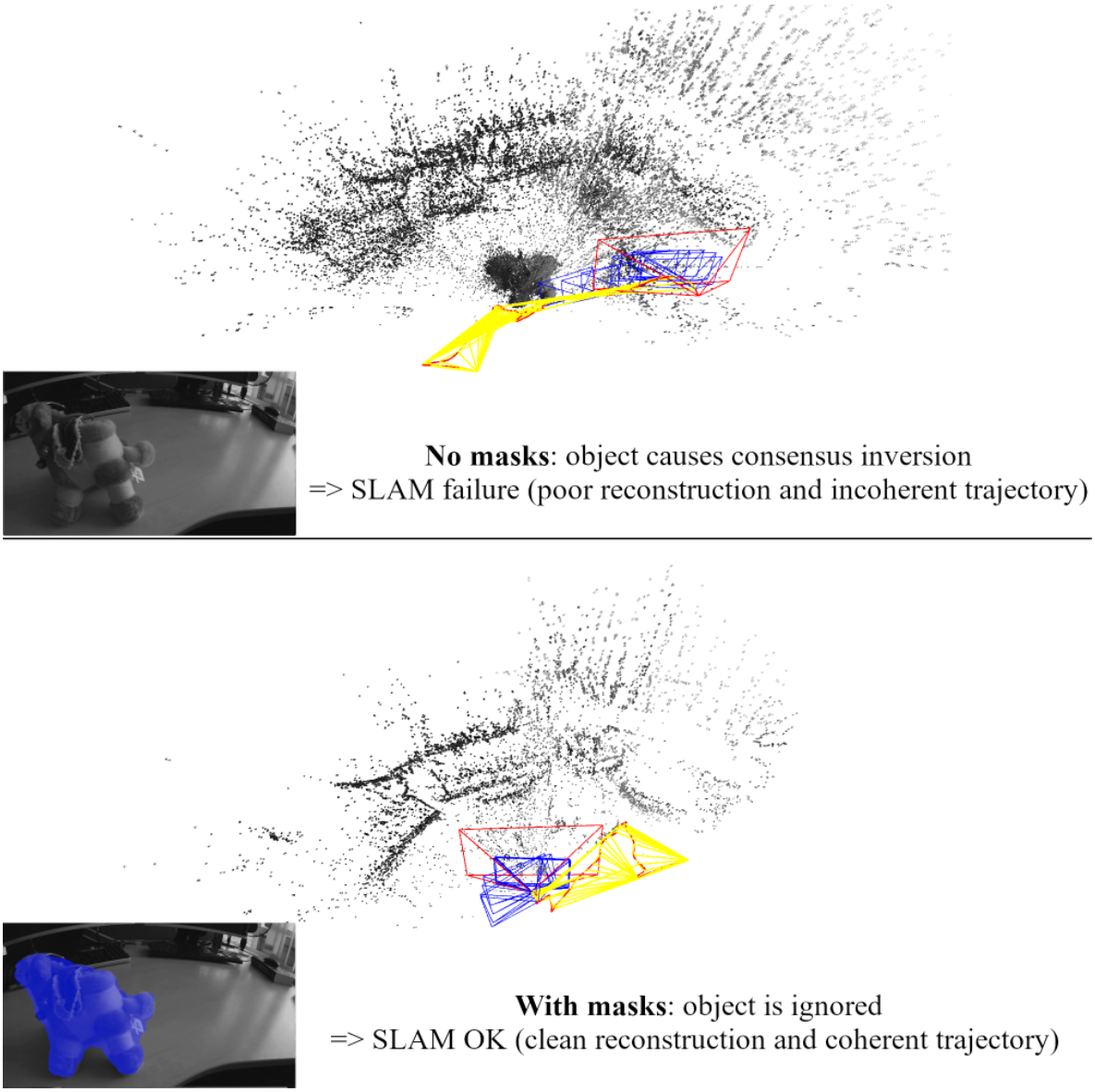


Figure 5.10 – We apply our method on LDSO and evaluate it on a hard sequence. Without masks the SLAM fails, with masks it ignores the object and works correctly.

5.3.3 Limitations

There are some limitations to our method. We rely on video segmentation networks, but they may not work in ambiguous situations, *i.e.*, when an object is not clearly identifiable in an image or not a solid entity. This includes various situations:

1. A dynamic object passes in front of a background similar to the object. This makes boundaries harder to distinguish and complicates image segmentation.

2. An object is (partially) out of bounds or occluded by another object. This makes image segmentation perform poorly and cuts the information flow when propagating masks across a sequence. We can mitigate this issue by splitting sequences where an object stops being visible and considering the rest of the sequence as a different one (the rest of the algorithm in section 5.2.3 is unchanged). But the resulting quality is lowered since per-sequence training datasets become smaller.

3. Non-solid entities like reflections, smoke, liquids, etc. are a poor match to our method. They are very hard to segment in the first place; creating training databases automatically is a major challenge. Our method is unlikely to work with these entities. Regarding deformable objects (*e.g.*, people), our method works if the segmentation algorithm has sufficient generalization capabilities. Seeing these deformable objects in different forms also helps training (*e.g.*, people walking, sitting, running...).

Although it is not really a limitation but more of a design choice, dynamic objects must be reconstructed by the SLAM to generate outliers. This means that it is difficult to mask objects that move all the time. We aim to mask outlier-generating objects: an object that never generates outliers when it moves (outside MCIs) does not need to be masked at all. For practice purposes, it is invisible to the SLAM. This implies that training sequences must be representative of what situations the SLAM algorithm may encounter.

Similarly, we do not handle new dynamic objects at runtime, but this is extremely difficult and a poor idea considering the risk of making the problem we are trying to solve ill-posed, as we discussed in section 1.1. Finally, the improved SLAM stops tracking if a known dynamic object covers the whole image; but we solve this issue by adding a new concept, Temporal Masking, that we present in chapter 6.

5.4 Conclusion

In this chapter we proposed a novel method to learn to segment dynamic objects using only one monocular sequence per dynamic object, which is an advantage in comparison to previous methods. More importantly, we do not need any manual labelling which makes our method much easier to use.

We also evaluated the CI dataset and new metrics to evaluate the robustness of Dynamic SLAMs. We showed that consensus inversions can cause major SLAM failures, even to state-of-the-art Dynamic SLAMs. We improved ORB-SLAM 2 monocular/stereo/RGB-D as well as LDSO at the same time and achieved top results in very challenging scenarios, effectively preventing MCIs.

Finally, another advantage of our method, in addition to preventing SLAM failures and improving motion estimation, is the improvement in map reconstruction. Tasks like relocalization and loop closing need accurate maps and should benefit from our approach.

Chapter 6

Dynamic SLAM with Temporal Masking

We present in this chapter our work on Dynamic SLAM with self-supervised Temporal Masking. We focus on the notion of temporality in Dynamic SLAMs that we first discussed in section 3.5. The goal of this chapter is to propose a solution to the problem: *when* should a Dynamic SLAM mask objects?

6.1 Introduction

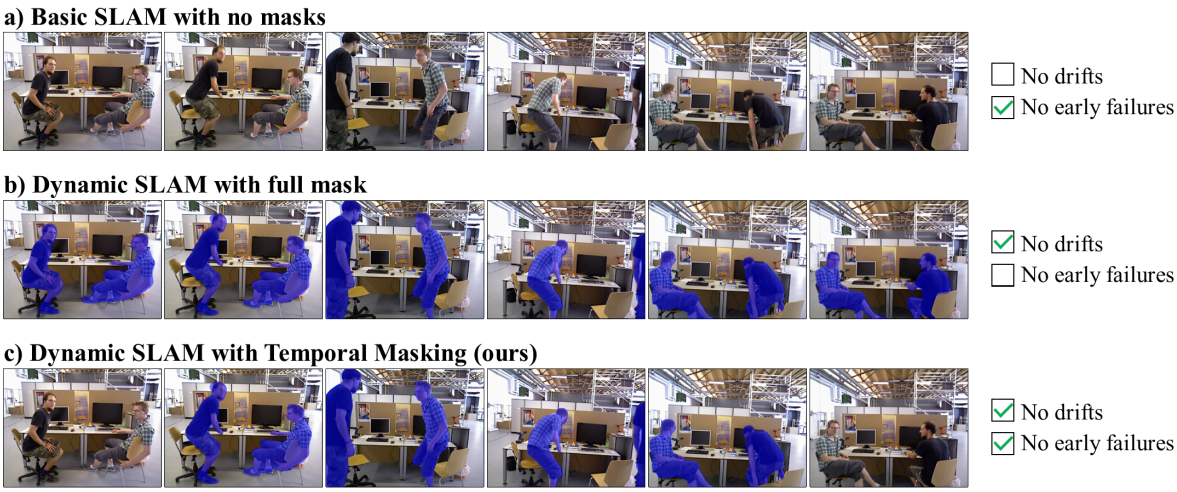


Figure 6.1 – Illustration of our results on the TUM RGB-D dataset. a) A Basic SLAM does not mask dynamic objects and consequently drifts. b) A Dynamic SLAM masks all supposedly dynamic objects, even when they are actually static, and may consequently fail if there are not enough unmasked features. c) Our method, Dynamic SLAM with Temporal Masking, masks dynamic objects when appropriate: when it is predicted to maximize SLAM performance. Our model learns by itself that masking objects in motion is beneficial for the SLAM, while other approaches must make this assumption.

We discussed in the previous chapter how to prevent motion consensus inversions but did not tackle the problem of early failure due to lack of features. To the best of our knowledge, current visual Dynamic SLAMs – as well as the approach presented in the previous chapter – are unable to handle both motion consensus inversions and failures due to excessive masking. The key cause is

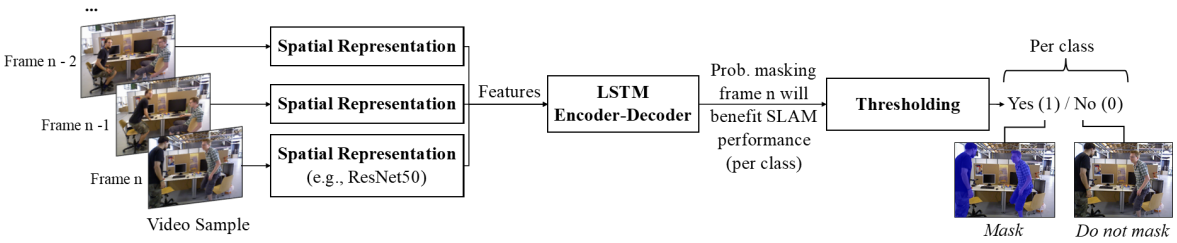


Figure 6.2 – Temporal Masking Network. It takes as input a video sample and outputs, for the last sampled frame, per-class binary masking decisions on which classes should be masked using a semantic mask. Spatial Representation outputs features for every input frame.

the dependency on instantaneous motion detection, unreliable in some cases. This problem is rarely studied. Slamantic proposes computes a "dynamic factor" for each 3D point depending on its semantic class and its detection consistency over time [82], which allows to identify those that should be used for pose estimation. It nevertheless still relies on geometrical considerations to detect instantaneous motion, therefore fails in scenarios where the instantaneous motion of an object is too difficult to detect. Moreover, the computation of dynamic factors depends on heuristic and class-specific thresholds, which makes generalization to new classes or environments a cumbersome process.

We thus propose to decide whether to mask objects or not without any *a priori* on geometry. In practice, we learn a model that provides such a decision frame by frame, depending on the previous frames of the sequence being processed and the data seen during training. Temporal Masking is therefore a new paradigm that differs from earlier approaches by not using any prior to filter 3D points, neither on object motion nor on which semantic classes are dynamic. In place of instantaneous geometric considerations, the masking decision emerges from the memory of training data and the previous frame of the sequence.

Most effective learning-based approaches are supervised, therefore require annotations. To create a temporal mask, a binary decision *mask / no_mask* must be taken for every frame of every training sequence. Beyond the tediousness of the task, it is moreover difficult to make such a choice *a priori* w.r.t. the final SLAM performance, even for an expert. In addition, such a choice may be algorithm-dependent and would have to be done again for any new approach.

Hence, the core of our contribution is a self-supervised approach to automatically create temporal masks for any sequence, without any manual work, and that adapts automatically to any (feature-based) SLAM algorithm. The general idea consists in sampling the space of all possible binary temporal masks, then evaluating and aggregating the samples to reflect the corresponding SLAM performance. A uniform sampling would nevertheless require too many samples, making the method computationally intractable, hence we propose a method to efficiently explore and select samples in this space. Any SLAM performance metric can be used with our method, including the classical trajectory accuracy (ATE RMSE) and robustness to failures (Tracking Rate). Our goal is to maximize SLAM performance holistically, so we choose to use the Unified SLAM Metric presented in section 4.4.2 since it combines accuracy and robustness.

Once the training sequences are automatically annotated, we train a model that takes as input past frames and computes a binary decision for the current one: whether applying a given spatial mask is relevant or not with regard to the final SLAM performance. The underlying architecture is based on recurrent neural networks to take into account possible dependencies with past frames. We show that our approach is on par or better than the State of the Art on TUM RGB-D and KITTI datasets, as well our ConsInv dataset, previously presented in section 4.5.3. We also report the limits of our approach when the test context is different from the training one.

Our approach is different from the State of the Art as we predict the effect of masking objects on SLAM performance: we add temporal masking to the SLAM, a memory-based decision module that

signals when to apply given semantic masks. We consider the SLAM as a black box with respect to masking decisions. As a consequence, Dynamic SLAM algorithms that use internal SLAM data for object masking [3, 8, 82, 109] are not compatible with our approach.

6.2 SLAM Pipeline

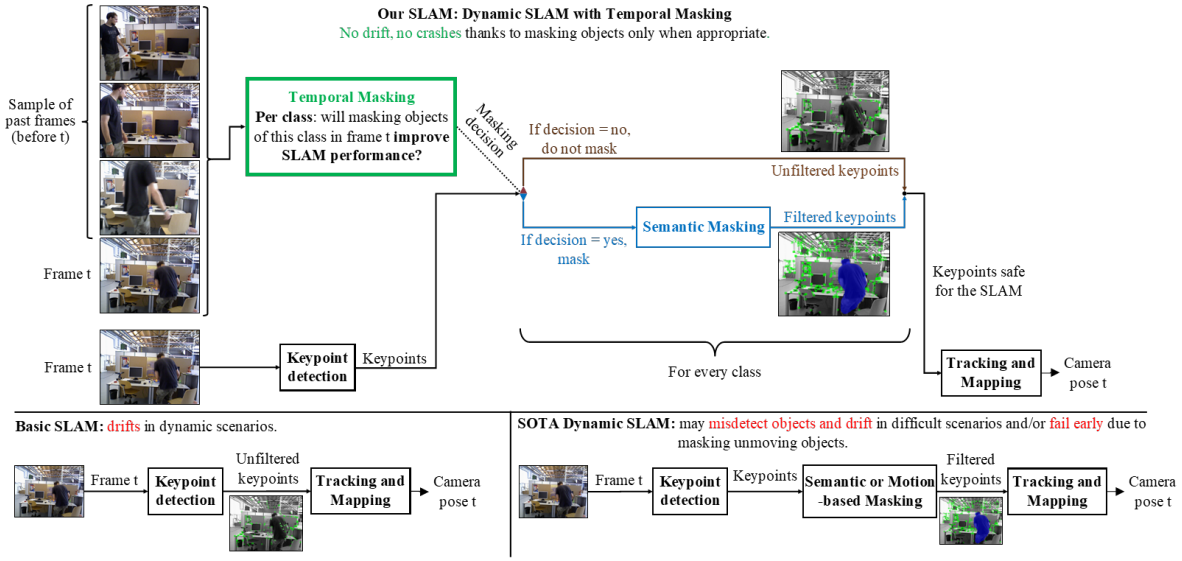


Figure 6.3 – Overview of Dynamic SLAM with Temporal Masking and comparison to other approaches. The key improvement compared to other methods is the per-class choice between masking and not masking objects that does not depend on the SLAM itself, nor priors on object motion or semantics.

We present our SLAM pipeline in fig. 6.3. 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) 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 set of past frames does not have to be consecutive nor immediately precede the frame at time t: the sample may include frames from the far past.

6.3 Temporal Masking Network

Overview. Let a video sequence and a generator of semantic masks segmenting p classes, *e.g.*, Mask R-CNN. Our goal is to predict frame by frame which object classes should be masked (with masks from the given generator) to maximize SLAM performance. Thus, we propose an LSTM-based Temporal Masking Network: it memorizes for every class the circumstances when masking objects pertaining to it is beneficial for the SLAM, to later infer masking decisions for unknown sequences. The reliance on memory instead of geometry-based instantaneous motion detection avoids the deadlocks discussed in section 2.2.2.

LSTM-based network. The Temporal Masking Network, in fig. 6.2, computes which semantic classes in the last frame of a video sample should be masked: it is a multi-label classification problem.

It is composed of a spatial representation module and an LSTM Encoder-Decoder. The network takes as input a video sample, computes a spatial representation of every frame of the sample (*e.g.*, features computed with a CNN encoder), inputs the computed features into an LSTM Encoder-Decoder, and finally applies a threshold to the result to obtain a per-class binary masking decision for the last frame of the sample. We train only the LSTM Encoder-Decoder.

The main reason to compute a spatial representation is that directly inputting tens of images at high resolution – 720p or above – in a LSTM is hardly possible within the memory limits of currently available GPUs. A lightweight spatial representation is far easier to handle and makes training faster. Note that if using deep networks to generate features, they do not need to come from the last layer of the encoder – they can come from intermediate ones.

We propose an architecture for the LSTM Encoder-Decoder in fig. 6.4. 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.

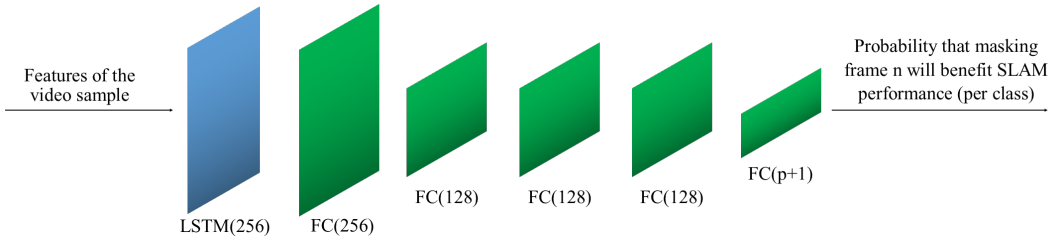


Figure 6.4 – LSTM Encoder-Decoder architecture. When masking p classes, the output has length $p+1$ as it includes a *nothing to mask* class.

6.4 Temporal Annotation Methods

Training a network for temporal masking requires annotating video sequences accordingly. Let us consider a video sequence of length l and a generator of semantic masks (*e.g.*, Mask R-CNN) segmenting p classes: a *temporal mask* is a binary matrix of size $l \times p$ storing masking decisions (*i.e.*, which classes to mask using the given generator) for every frame of the sequence. We make annotations with two approaches: sequence-wise or frame-wise.

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, and expert task, we propose to learn temporal masks with self-supervision in section 6.4.2 and compare this approach to the simpler ones in section 6.4.1. We illustrate the annotation process in fig. 6.5.

6.4.1 Baseline 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.

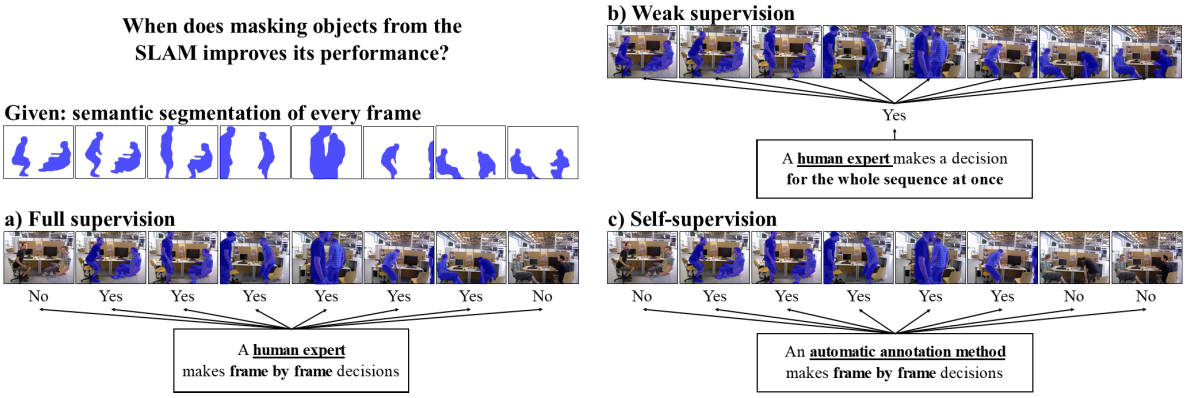


Figure 6.5 – Sequence annotation methods. The annotation consists in deciding, per class, when to mask objects of this class with semantic segmentation. a) In full supervision, a human expert makes decisions for every frame. b) In weak supervision, a human expert makes a single decision that is applied to all frames. c) In self-supervision, an automatic annotation method makes decisions for every frame.

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 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.

6.4.2 Self-Supervised Method

Annotations for self-supervision are automatic and frame-wise. We first present the annotation method for a single class to mask. Note that iterating through the full temporal mask space is computationally intractable (up to 2^n masks large, where n is the sequence length), so we cannot exhaustively evaluate all temporal masks.

We initially considered directly constructing temporal masks with various strategies as sliding masking windows (inspired by our outlier-based masking method), where we would try to find blocks of consecutive frames to mask. This worked in very specific cases, *e.g.*, when a single object moves quickly once in a sequence, but it generalizes poorly. Then we thought about Monte-Carlo methods, that consist in solving mathematical problems through random sampling of known spaces. Monte-Carlo methods are especially useful when one does not need exact answers and exact/exhaustive computation is computationally intractable.

Therefore, 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.

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 6.6 – 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.

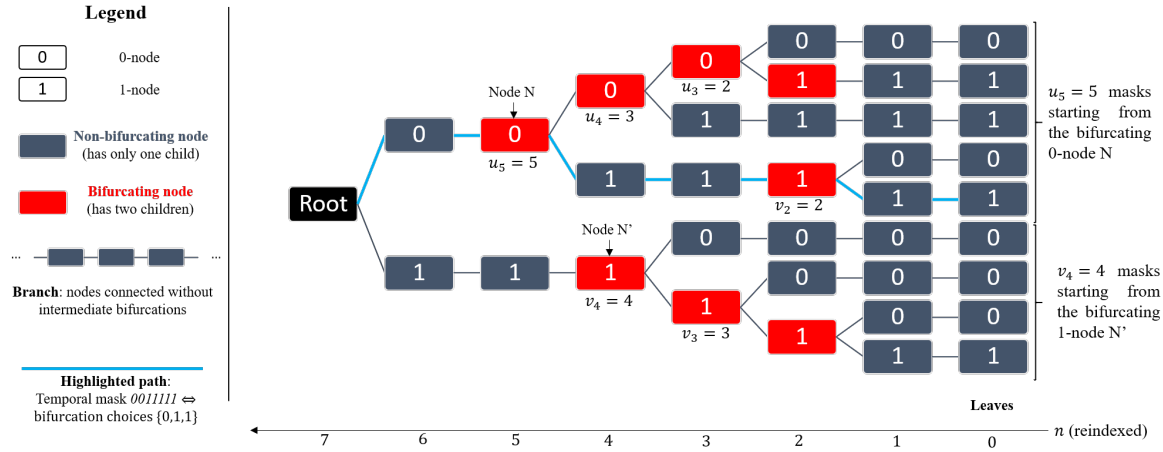
We represent E as a binary tree (see fig. 6.7). To keep the problem computationally tractable, the key insight is to use a closed-form expression of the number of possible paths $C(N)$ under any bifurcating node N. Starting from the root, whenever we reach a node that has two children, we randomly pick 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. Figure 6.6 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 a 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 $l = 7, k_0 = 2, k_1 = 3$. Figure 6.7 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 fig. 6.7).

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



- 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 6.7 – 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.

set $b = 0$ if $r < \frac{C(B\cup\{0\})}{C(B)}$ (*i.e.*, we branch towards 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:

$$\begin{aligned} P(B_0, ..., B_p) &= P(b_0) \times P(b_1|b_0) \times ... \times P(b_p|b_0, ..., b_{p-1}) \\ &= \frac{C(B_0)}{|E|} \times \frac{C(B_1)}{C(B_0)} \times ... \times \frac{C(B_p)}{C(B_{p-1})} = \frac{1}{|E|} \end{aligned} \tag{6.1}$$

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:

$$\begin{cases} u_n = u_{n+1} + v_{n+k_1} \\ v_n = u_{n+k_0} + v_{n+1} \end{cases} \tag{6.2}$$

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

$$\begin{cases} u_n = u_{n-1} + v_{n-k_1} \\ v_n = u_{n-k_0} + v_{n-1} \end{cases} \iff \begin{cases} u_n = 2u_{n-1} - u_{n-2} + v_{n-k-1} \\ v_n = 2v_{n-1} - v_{n-2} + v_{n-k-1} \end{cases} \tag{6.3}$$

(6.3) shows that (u_n) and (v_n) are linear recurrence relations with constant coefficients and the same characteristic equation: $x^{k+1} - 2x^k + x^{k-1} - 1 = 0$. [2] 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.

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, presented in section 4.4.2.

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:

$$R = \sum_{x,y \in \text{Samples}} \begin{matrix} \max(0, \text{sgn}(|s_y - s_x| - \max(\sigma_r |s_x|, \sigma_a))) \\ \times (s_y - s_x)(y - x) \end{matrix} \tag{6.4}$$

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**.

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.

6.5 Main Experiments

6.5.1 Experimental setup

Datasets. We evaluate the 8 dynamic sequences of TUM RGB-D dataset in RGB-D and the first 11 sequences of KITTI odometry dataset, the ConsInv-Indoors dataset in monocular (both the Dynamic and Static subsets are used for training but only the former is of interest for evaluating our Temporal Masking method) and the ConsInv-Outdoors dataset in stereo, all presented in chapter 4. For TUM RGB-D and KITTI, given the small number of sequences, we run our tests in a leave-one-out approach: we run the annotation/train pipeline on all sequences but one and infer/test temporal masks for the remaining one in a round-robin fashion.

Spatial representation. We use ResNet-50 [48] (TensorFlow 2, pretrained on ImageNet) to compute frame features. Figure 6.8 illustrates the spatial representation pipeline. It appears that

naive spatial representations, like using the output of the last layer of ResNet-50 does not work at all: the training of the Temporal Masking Network does not converge. Apparently, the information needed to understand a scene at a high level of abstraction (concepts like *the object just started moving*) is present in lower-level layers. Thus, for every frame, we collect the output of each of the 17 convolutional blocks of ResNet-50. For each block, we apply a 2D average pooling on 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 – a real-valued vector of size 15k to 20k. We considered more advanced PCA strategies [85], but they did not appear as necessary.

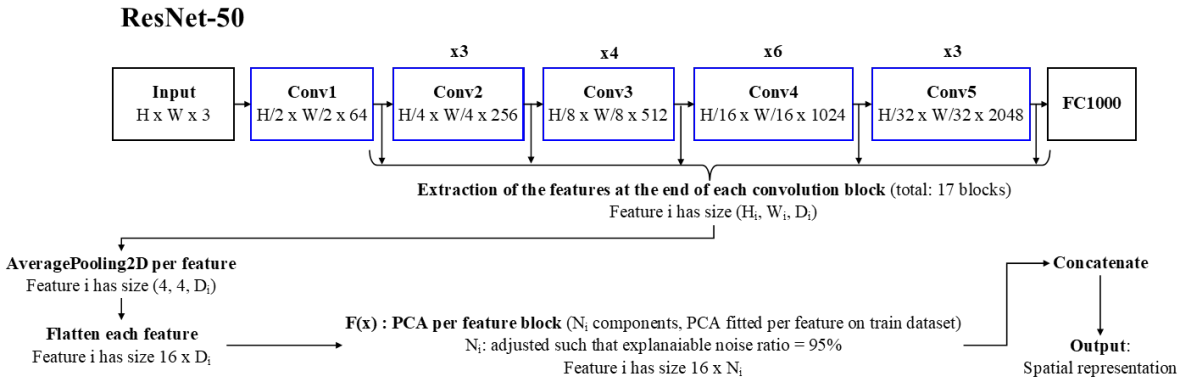


Figure 6.8 – Process to compute spatial representations using ResNet50.

LSTM Encoder-Decoder and Training. Given a frame index k, we generate video samples by randomly sampling 49 frames from frames 0 to $k-1$ and adding frame k. We sample frames from the whole past to avoid depending on instantaneous motion detection. We mask the first 49 frames as we cannot sample them without repetition. We train the LSTM Encoder-Decoder on train+val with early stopping. Training lasted 40 to 60 epochs in our experiments. We augment training sequences with horizontal flipping and process video samples in a random order.

SLAM and Semantic Masking. We use ORB-SLAM 2 (3000 features). As ORB-SLAM 2 is non-deterministic, we run it five times on KITTI dataset and ten times on other datasets for every sequence/temporal mask to evaluate and report the median score. We generate semantic masks for KITTI/ConsInv-Outdoors datasets using Mask R-CNN, trained on the COCO dataset, and for ConsInv-Indoors/TUM RGB-D datasets we used the DeepLabv3+ models from chapter 5. We segment people/vehicles (7 classes) for KITTI, people and cars for ConsInv-Outdoors (2 classes), people for TUM RGB-D (1 class), and drom/lambo/dragon for ConsInv-Indoors (1 class[1]).

Dataset Annotation. For automatic annotations (*i.e.*, self-supervision), we sample 200 temporal masks for each sequence with min. block sizes $k_0 = k_1 = 25$ and use the USM (section 4.4.2) to benchmark/aggregate temporal masks. We manually create weak annotations and full annotations. The latter are verified to prevent all cases of excessive masking and motion consensus inversion. Since the ATE RMSE of sequences in the datasets TUM RGB-D/ConsInv and KITTI is resp ≈1cm and ≈1m with manual expert labels, we set $\lambda = 10m^{-1}$ for TUM RGB-D/ConsInv and $\lambda = 0.1m^{-1}$ for KITTI, which implies $USM \approx TR(1 - \lambda\text{ATE})$ and make comparisons between SLAMs match user expectations.

[1]Note that the model trained on CI dataset segments the dromedary/lambo/dragon in a single *dynamic* class although there are three object classes of interest.

6.5.2 Comparison between annotation methods

We compare in table 6.1 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 and TUM RGB-D dataset.

For the ConsInv-Indoors-Dynamic subset, the performance of self-supervision is slightly higher. For the TUM RGB-D and ConsInv-Outdoors datasets, 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), therefore difficult, for the temporal masking network 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 chapter.

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 6.1 – Comparison of supervision modes. Average USM on ConsInv/TUM RGB-D.

6.5.3 Comparison with the State of the Art

We compare in table 6.2 our method to DynaSLAM[2] [8], Slamantic [82], Lucas-Kanade optical flow [21] (only in monocular), all based on ORB-SLAM 2. We also evaluate StaticFusion [84], that supports only RGB-D. *No masks* refers to the original ORB-SLAM2, *Full masks* refers to ORB-SLAM 2 with semantic masks always applied. All methods use the same semantic masks. Note that since scores are medians over the datasets, median USM results cannot be exactly computed from the median ATE RMSE and median Tracking Rate. We do not show the ATE RMSE and Tracking Rate for the ConsInv dataset since it includes sequences likely to cause failures due to excessive masking. In such sequences the ATE RMSE and Tracking Rate are misleading.

TUM RGB-D dataset. Table 6.2 shows that the major obstacles for the SLAM for this dataset are motion consensus inversions, and our self-supervised model learned to mask objects most of the time to prevent them. All methods performed well, close or equal to the manual annotations, except DynaSLAM that performed barely above the *No masks* baseline and StaticFusion below it. DynaSLAM removes even more features than the Full masks approach, which delays SLAM initialization and causes temporary tracking loss. The Tracking Rate of DynaSLAM is in fact exceedingly low despite a good accuracy (*i.e.*, low ATE RMSE), which proves the importance of measuring the Tracking Rate and the relevance of our metric, the USM. StaticFusion fails in dynamic sequences with fast rotations, as it is unable to filter dynamic objects quickly enough.

KITTI dataset. Table 6.2 shows that our method slightly outperforms others as it makes slightly better masking decisions overall. Still, it appears that the KITTI dataset is unsuitable for Dynamic SLAM testing since no matter when objects are masked, performance hardly changes. In particular, the Tracking Rate is always 100%, so tracking failures are absent from this dataset.

ConsInv-Indoors-Dynamic subset. Table 6.2 shows that our approach outperforms all current methods. DynaSLAM and Slamantic perform worse than the Full masks baseline: thus, masking as

[2]DynaSLAM crashes when processing the TUM RGB-D seq. fr3_walking_xyz. We compute averages on the 7 other sequences in this case.

Mode	Dataset	Metric	Baselines		State of the Art				Ours
			No masks	Full masks	Optical flow [21]	DynaSLAM [8]	Slamantic [82]	StaticFusion [84]	
RGB-D	TUM RGB-D	ATE RMSE (m) ↓	0.105	**0.019**	-	**0.019**	0.028	0.099	**0.019**
		Tracking Rate ↑	**96%**	**96%**	-	69%	**96%**	**96%**	**96%**
		USM ↑	0.55	**0.80**	-	0.57	0.76	0.54	**0.80**
Stereo	KITTI	ATE RMSE (m) ↓	2.59	2.67	-	2.74	2.70	-	**2.51**
		Tracking Rate ↑	**100%**	**100%**	-	**100%**	**100%**	-	**100%**
		USM ↑	0.80	0.80	-	0.79	0.80	-	**0.81**
Stereo	ConsInv-Outdoors	USM ↑	0.61	0.81	-	0.80	0.82	-	**0.88**
Mono	ConsInv-Indoors-Dynamic	USM ↑	0.57	0.71	0.49	0.63	0.68	-	**0.75**
Mono	ConsInv-Extra-MeetingRoom	USM ↑	0.33	0.65	0.34	0.60	0.54	-	**0.67**
Mono	ConsInv-Extra-LivingRoom	USM ↑	0.51	0.73	0.55	0.60	0.69	-	**0.74**
Mono	ConsInv-Indoors-Static	Prevented false starts ↑	56%	**100%**	67%	**100%**	78%	-	**100%**

Table 6.2 – Comparison with the State of the Art on various datasets in their preferred mode. '-' indicates that the mode is not supported by the SLAM algorithm. All scores are medians over the dataset. For ATE RMSE, lower is better (↓). For Tracking Rate / USM / Prevented false starts, higher is better (↑). We use the model trained on ConsInv-Indoors when evaluating our method on ConsInv-Extra, in order to evaluate how it performs in new contexts.

much as possible (DynaSLAM does semantic + motion-based masking) or masking with prior geometric criteria (Slamantic) is suboptimal in difficult scenarios – and masking appropriately is challenging. Lucas-Kanade optical flow has a low score as it does not avoid all consensus inversions and removes features during fast motions, lowering accuracy.

ConsInv-Indoors-Static subset. We also evaluate the ability to prevent false starts (section 4.5.3) on the ConsInv-Indoors-Static subset. Table 6.2 shows that most methods, including ours, prevent all false starts. The underperformance of Slamantic shows that motion-based geometric criteria are vulnerable to motion consensus inversions.

ConsInv-Outdoors dataset. Table 6.2 shows that our multiclass self-supervised method outperforms the State of the Art (USM = 0.88). Surprisingly, almost all other approaches have the same score (USM ≈ 0.80). This means that the masking strategy has to be learned from the environment, or performance may be suboptimal.

Global conclusion. Our method outperforms the State of the Art in all datasets. A remarkable result is that when object motion is difficult to detect as in the ConsInv dataset, the simple semantic approach *Full Masks* is safer than motion-based, geometry-dependent approaches. Since our method does not depend on geometric a priori, it can outperform *Full Masks*, unlike current methods.

6.5.4 Interpretation of Inferred Masks

Quantitative interpretation

We showed previously 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*.

Table 6.3 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 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

Mode	**Dataset**	**Metric**	**Manual annotations** (No learning)	**Ours**
RGB-D	TUM RGB-D	ATE RMSE (m) ↓	**0.019**	**0.019**
		Tracking Rate ↑	**96%**	**96%**
		USM ↑	**0.80**	**0.80**
Stereo	KITTI	ATE RMSE (m) ↓	2.53	**2.51**
		Tracking Rate ↑	**100%**	**100%**
		USM ↑	**0.81**	**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 6.3 – 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.

label inconsistency explains why full supervision performs poorly and further highlights the value of self-supervision.

Qualitative interpretation

Figure 6.9a to fig. 6.11 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 observe the following:

1. Manual annotations follow the "mask the bare minimum" approach while masks inferred from the self-supervised model follow the "mask unless we are sure it is safe" approach.

2. In easy sequences (fig. 6.11), where masking has no effect, our approach still prefers to mask objects.

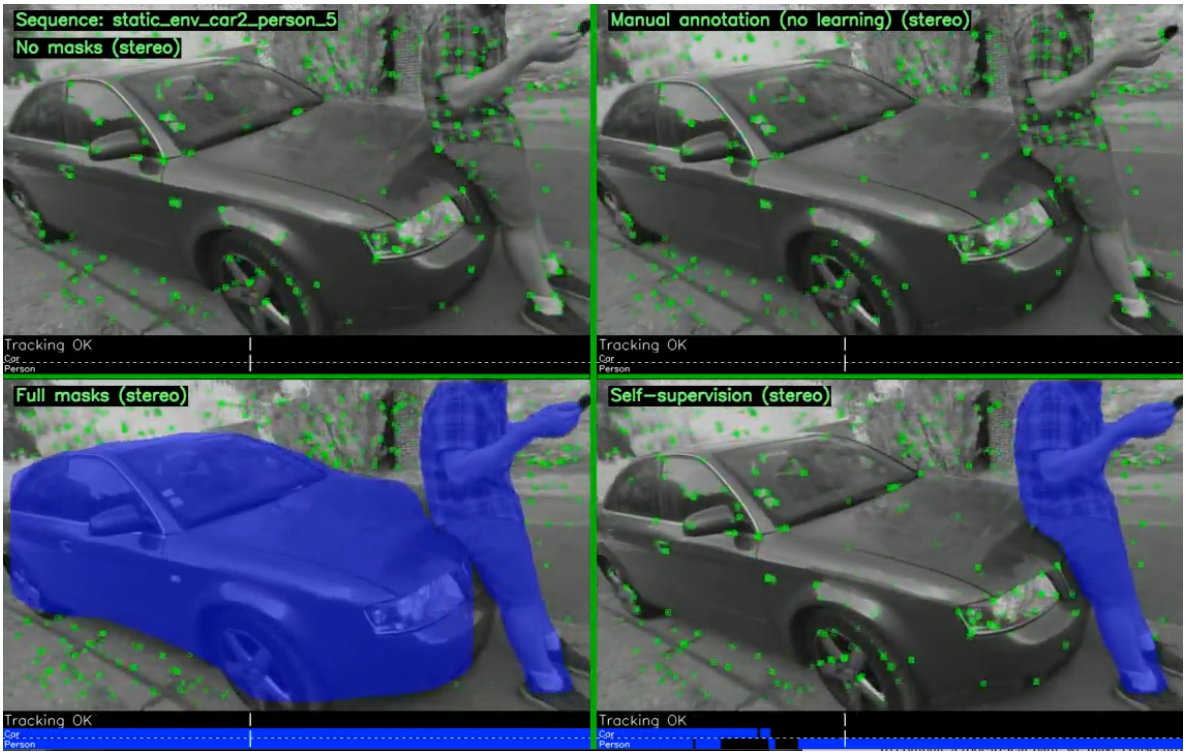
3. In hard sequences where excessive masking negatively affects performance (fig. 6.9a, fig. 6.9b), our approach masks cars for some time at the start before concluding that "it is not necessary anymore". Figure 6.9a shows a difficult case where excessively masking objects leads to SLAM failure.

4. In difficult sequences where objects cause motion consensus inversions (fig. 6.10b), our approach always masks objects.

5. Our model has, to a certain extent, situational awareness. It learned that a *car + person on 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 previously.

6. Masking people is almost always necessary.

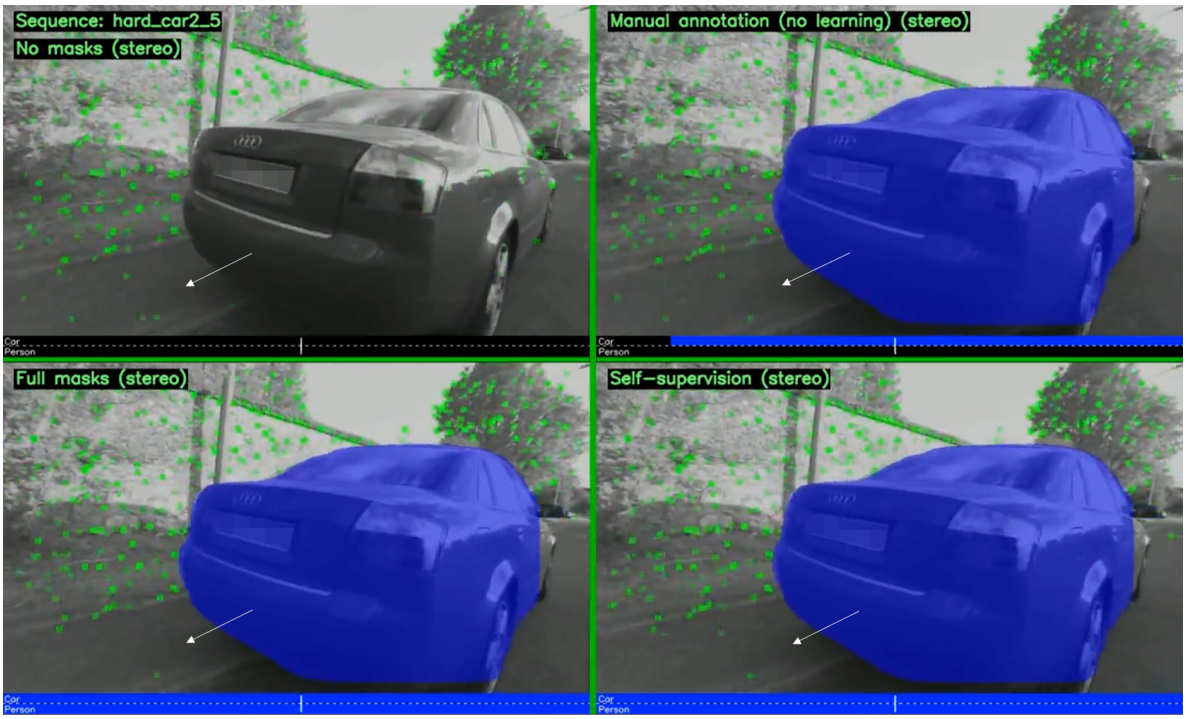
(a) Masking result on a difficult sequence with moving people and static cars. Our approach masks people all the time and almost never cars, making the inferred annotation and the manual annotation very similar.



(b) 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 6.9 – Results of Temporal Masking on ConsInv-Outdoors, part 1. Blue segments indicate masked frames, per class. Red segments indicate tracking failure due to excessive masking.

(a) Another 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.



(b) 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.

Figure 6.10 – Results of Temporal Masking on ConsInv-Outdoors, part 2. Blue segments indicate masked frames, per class. Red segments indicate tracking failure due to excessive masking. Arrows indicate motion direction.

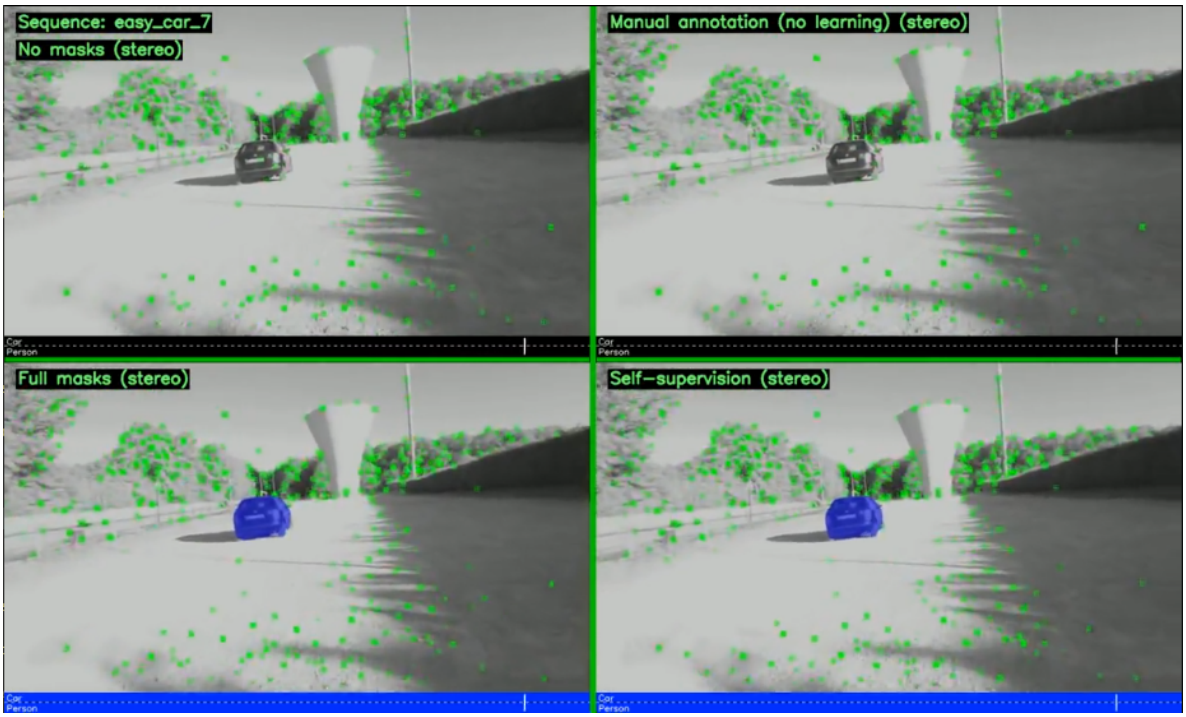


Figure 6.11 – Masking result on an easy sequence with moving cars and no people. Our approach says to mask both people and cars all the time out of caution. Masking objects has no effect in this sequence since the vast majority of features is on the background. Blue segments indicate masked frames, per class.

6.5.5 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. Table 6.4 shows that our method still has superior performance, albeit slightly lower than before.

T_{Ref}	**Baselines**		**Ours**	
	No masks	Full masks	Self-sup.	Self-sup. (degraded)
0.83	0.57	0.71	**0.75**	0.73

Table 6.4 – Average USM on ConsInv-Indoors-Dynamic to evaluate the robustness to degraded semantic masks.

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

Table 6.5 – Rate of prevented false starts on ConsInv-Indoors-Static including a degraded mask approach.

6.5.6 Computation time analysis and sampling computational tractability.

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 table 6.6. 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 6.6 – Average inference time on a GTX 1080 Ti, per frame. Full res. is 1280x720.

We give the size of the temporal mask space E in table 6.7 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 (≈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.

6.5.7 Data requirement for training

This section compiles results from the main experiments to give a rule of thumb to know how much data is needed to actually train a temporal masking network. The key elements are dataset complexity and dataset size.

We define dataset complexity as the number of difficulties present in the dataset among: 1) At least one sequence causes a motion consensus inversion for a *No masks* Basic SLAM 2) At least one sequence causes an early failure due to lack of features to a *Full Masks* Dynamic SLAM 3) Neither the *No masks* nor the *Full Masks* approach prevent all MCIs and early failures.

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	$\lvert E\rvert \approx 0.616 \times 1.1005^{(n-25)} \approx 0.616 \times 10^{0.04158(n-25)}$						
	$n =$	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	$\lvert E\rvert \approx 0.534 \times 1.0593^{(n-50)} \approx 0.534 \times 10^{0.0250(n-50)}$						
	$n =$	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 6.7 – 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).

We define dataset size in tiers based on the number of images: 1) Less than 10000 images 2) From 10000 to 49999 images 3) 50000 images or more.

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). Table 6.8 shows the dataset complexity and size of the datasets previously evaluated.

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

Table 6.8 – 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 lower or equal than dataset size. Quantitatively, about 10 sequences made of 1000 images per dynamic object class is a good starting point for self-supervision.

6.5.8 Hyperparameter tuning

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 (section 6.4.2).

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. Note that we previously raised the idea of controlling the risk of SLAM failure, especially SLAM drifts, through Temporal Masking in section 3.5. In fact, the accepted risk of failure is controlled in the last step of our method during the choice of masks among equivalent ones: *Min TM* would be a high-risk approach and *Max TM* a low-risk one.

We tested all annotation methods using the proposed neural network architectures. We used the **train** split of the ConsInv-Indoors-Dynamic subset for training (not train+val) and evaluated the results on the **val** split of ConsInv-Indoors-Dynamic.

Table 6.9 shows that the automatic annotation method Max TM is the overall best, hence we choose it as our main method in this chapter. 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 exceedingly 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 probably 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 6.9 – Comparison between different annotation methods on the validation split of the ConsInv-Indoors-Dynamic subset.

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 evaluate the annotations directly in the SLAM, without learning them. We first tuned k_0 and k_1 by maximizing the USM. Table 6.10 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. Table 6.11 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 table 6.10 and table 6.11 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).

6.5.9 Limitations: tests in out-of-context

While we designed our method to adapt a SLAM to any context (same object classes/similar environment) at a low cost with self-supervision, we evaluated how the model trained on ConsInv-Indoors performs in the contexts of the ConsInv-Extra dataset – a living room and a meeting room. Table 6.2 shows that we perform slightly better than other SLAMs. Thus, our model works in new contexts

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 6.10 – 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 6.11 – Tuning of n.

similar to the training one and with the same objects. If the change is more drastic, performance would likely decrease and require adaptations that are out of the scope of our proposal.

Another limitation is that the automatic annotation method needs a ground truth trajectory of training sequences. It can be obtained, for instance, with manual expert temporal annotations or with training-time GPS/motion capture data. A ground truth with poor accuracy is likely to lower the quality of automatic annotations.

6.6 Complementary work: Dynamic SLAM with Weakly Supervised Temporal Masking

We present in this section an extension of our work on Dynamic SLAM with Temporal Masking to weak supervision. We propose a new network architecture more suitable for it. We separated this work in a new section to prevent confusion with our main method, which is self-supervised. *Ours* now specifies whether it is the main self-supervised method or the newly improved weakly supervised method.

In the self-supervised case, our proposal is to automatically annotate sequences with temporal masks and learn them with a network. We talked about weak supervision in section 6.4.1 but abandoned this approach in the first experiments (section 6.5.2) as it proved to have a lower performance than self-supervision.

Still, weak supervision remains of interest. Self-supervision has a reduced annotation cost in the sense that it requires little to no human intervention. However, computational costs are not negligible as every sequence has to be evaluated tens, if not hundreds of times. Moreover, if self-supervision cannot be used, we do not have any way out other than manual annotation – which is much more costly. For instance, self-supervision could be unavailable due to limited computational resources or the need for a shorter training pipeline (*e.g.*, if the annotation code cannot be shared).

Therefore, we propose a new network architecture that improves the performance of weak supervision and makes it more useful for practical uses.

6.6.1 Temporal Masking Network for Weak Supervision

We present the new network architecture in fig. 6.12. The architecture designed for self-supervision, first presented in section 6.3, consists in an LSTM followed by a series of fully connected layers. It uses as input the concatenated features of every frame of an input video sample. Our proposal is to input separately features according to their source layer. For instance, if using ResNet50 to generate features (fig. 6.8 in section 6.5), features may come from intermediate layers and not only the last layer. The rationale is that different layers store information at different levels of abstraction, so processing features according to their source layer would make learning this information easier.

Figure 6.12 shows both the previous and new architecture of the Temporal Masking Network. The main difference is that features from different frames are concatenated according to their source layer, then input in separate LSTMs. We name the previous architecture *Single LSTM* and the new architecture *One LSTM Per Layer LSTM Encoder-Decoder*. The different branches are later concatenated to have the same output as before, *i.e.*, a vector of size $p + 1$ where p is the number of classes to mask + a "do not mask anything" class. The latter is discarded during inference.

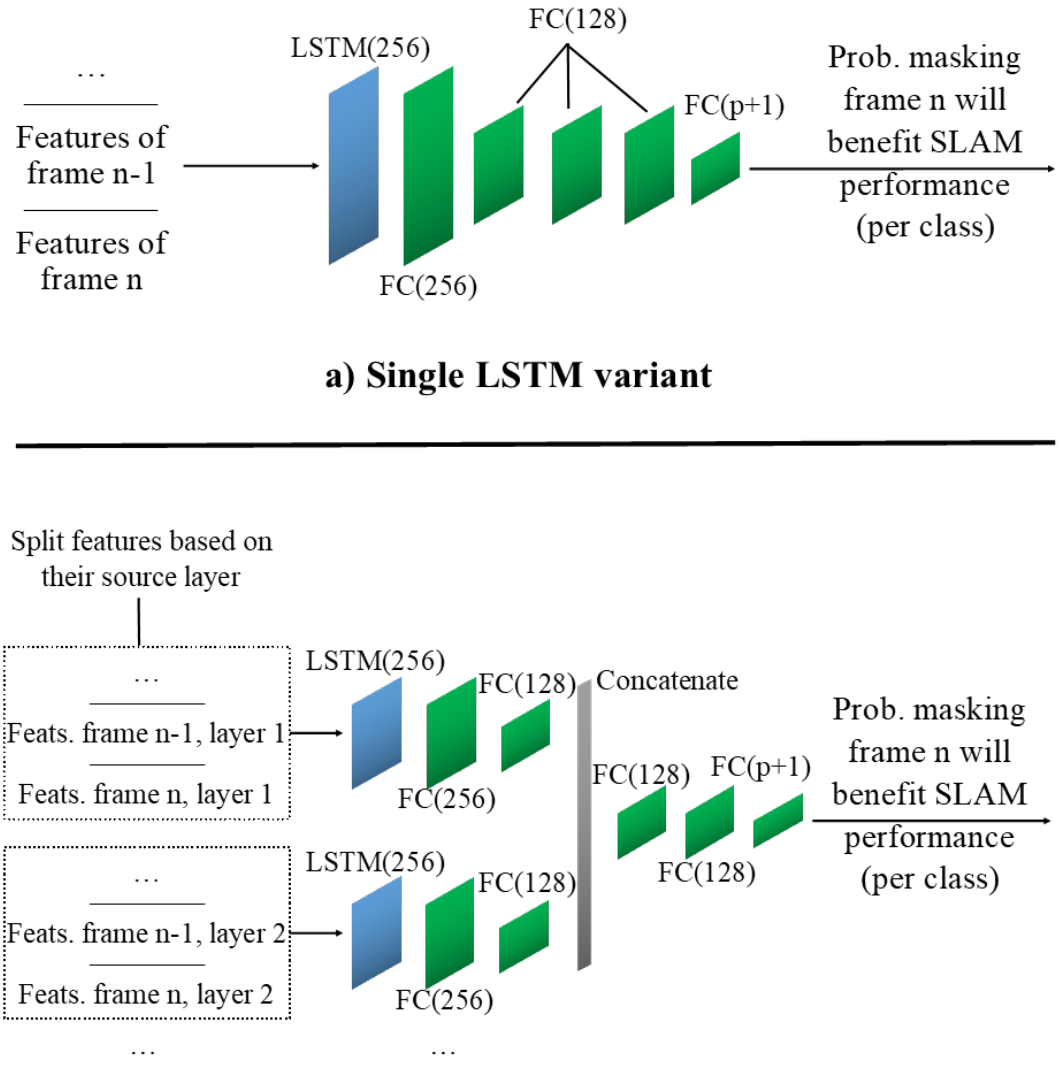


Figure 6.12 – Architectures of the LSTM Encoder-Decoder, a) With a single LSTM and b) With multiple LSTMs. The input are features of every image of a video sample. In a), features are simply concatenated. In b), features are grouped according to their source layer in the neural network used to generate them (see section 6.5.1). When masking p classes, network output has length $p + 1$ as it includes a *nothing to mask* class. Experimentally (section 6.5.1), a) is more suited for full/self-supervision and b) for weak supervision.

6.6.2 Experiments

The experimental setup is the same as in (section 6.5.1), with the exception of how the spatial representation is input in the LSTM Encoder-Decoder. If inputting features into an LSTM Encoder-Decoder using the *One LSTM per Layer* architecture, features are inserted in separate LSTMs according to their source layer in ResNet50.

Choice of architecture according to the annotation method

We follow the notations in section 6.5.8. To determine the most suitable architectures, including the new one (fig. 6.12) for self/weak/full supervision. We trained models on the *train* split of the ConsInv-Indoors dataset and evaluated the results on the *val* split of ConsInv-Indoors-Dynamic.

Table 6.12 shows that the *Single LSTM* architecture is more suitable for full/self-supervision training and *One LSTM/Layer* for weak supervision. The automatic annotation method Max TM remains the overall best. Between manual and weak annotations, the weak ones perform slightly better, which is a remarkable result given that they cost much less. Weak supervision has a slightly better performance with the One LSTM/Layer architecture.

	Automatic annotations			**Manual annotations**	
Architecture	**Min TM**	**Max TM** (Self-supervision)	**Best Random**	**Weak annotations** (Weak supervision)	**Full annotations**
Single LSTM	0.68	**0.77**	0.69	0.72	0.71
One LSTM/Layer	0.66	0.75	0.70	**0.73**	0.70

Table 6.12 – Comparison between different annotation methods and architectures on the validation split of the ConsInv-Indoors-Dynamic subset. Best automatic / manual methods in bold.

While the Single LSTM architectures is better for self-supervision, it is not as much for weak supervision. Thus, we decided to extend the comparison between supervision modes (table 6.1 in section 6.5.2). We compare the performance of every mode on *test* sequences after training on train+val sequences.

Table 6.13 shows that the difference between both architectures is sharp for weak supervision. While self-supervision stays the overall best method, weak supervision performs significantly better, sometimes with a negligible difference compared to self-supervision. Between manual and weak annotations, the weak ones perform slightly better, which is a remarkable result given that they cost much less. Weak annotations work better with the One LSTM/Layer architecture and full annotations with a single LSTM, although the difference is low – it may even become negligible with enough data. For practical purposes, when the use of weak annotations is a requirement, it might be useful to try both architectures.

Dataset	**Full supervision**	**Weak supervision** Single LSTM arch.	**Weak supervision** One LSTM/Layer arch.	**Self-supervision**
ConsInv-Indoors-Dynamic	0.72	0.68	0.74	**0.75**
ConsInv-Outdoors	0.74	0.80	0.79	**0.88**
TUM RGB-D	0.69	0.70	0.74	**0.80**

Table 6.13 – Comparison between supervision modes with different architectures. Average USM on ConsInv/TUM RGB-D. Includes the number of sequences per object class.

For the rest of this chapter, unless otherwise noted, we use the One LSTM/Layer architecture for weak supervision.

Comparison with the State of the Art

Having improved the performance of weak supervision, we extended the comparison to the State of the Art (table 6.2 in section 6.5.3) to include weak supervision.

Table 6.14 shows that in most cases our weakly supervised method is second place only to our own self-supervised method. Thus, weak supervision appears as a viable solution in settings where self-supervision is unavailable.

Mode	Dataset	Metric	Baselines		State of the Art				Ours	
			No masks	Full masks	Optical flow [21]	DynaSLAM [8]	Slamantic [82]	StaticFusion [84]	Self-Supervision	Weak Supervision
RGB-D	TUM RGB-D	ATE RMSE (m) ↓	0.105	**0.019**	-	**0.019**	0.028	0.099	**0.019**	0.032
		Tracking Rate ↑	**96%**	**96%**	-	69%	**96%**	**96%**	**96%**	**96%**
		USM ↑	0.55	**0.80**	-	0.57	0.76	0.54	**0.80**	0.74
Stereo	KITTI	ATE RMSE (m) ↓	2.59	2.67	-	2.74	2.70	-	**2.51**	2.63
		Tracking Rate ↑	**100%**	**100%**	-	**100%**	**100%**	-	**100%**	**100%**
		USM ↑	0.80	0.80	-	0.79	0.80	-	**0.81**	0.80
Stereo	ConsInv-Outdoors	USM ↑	0.61	0.81	-	0.80	0.82	-	**0.88**	0.79
Mono	ConsInv-Indoors-Dynamic	USM ↑	0.57	0.71	0.49	0.63	0.68	-	**0.75**	0.74
Mono	ConsInv-Extra-MeetingRoom	USM ↑	0.33	0.65	0.34	0.60	0.54	-	**0.67**	0.53
Mono	ConsInv-Extra-LivingRoom	USM ↑	0.51	0.73	0.55	0.60	0.69	-	**0.74**	0.73
Mono	ConsInv-Indoors-Static	Prevented false starts ↑	56%	**100%**	67%	**100%**	78%	-	**100%**	**100%**

Table 6.14 – Comparison with the State of the Art on various datasets in their preferred mode. We include results from weakly supervised temporal masking. '-' indicates that the mode is not supported by the SLAM algorithm. 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.

Data requirements for training

To complete the study on weakly supervised Temporal Masking, we complete the conclusions on training data requirements (table 6.8 in section 6.5.7). For readability, as the original table is short, we copied it here.

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

Table 6.15 – 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 lower or equal than dataset size; for weak supervision, we need dataset complexity lower than dataset size. Quantitatively, ≈ 10 sequences made of 1000 images per dynamic object class is a good starting point for self-supervision. For weak supervision, this requirement increases to ≈ 15 sequences. This explains why the performance of weak supervision is average on TUM RGB-D/ConsInv-Outdoors datasets, and state-of-the-art on ConsInv-Indoors/KITTI datasets.

6.7 Conclusion

In this work, we introduced the concept of Temporal Masking to overcome the limits of current Dynamic SLAMs. The general idea is to mask objects of certain classes, frame by frame, in order to maximize a chosen SLAM metric. In detail, we proposed a neural network architecture that we train in a self-supervised way thanks to the proposed automatic annotation method. We compared the obtained model with baselines and with the State of the Art [8, 82, 21, 84] and outperformed them all on real data: on the TUM RGB-D, KITTI and ConsInv datasets

We showed that one of the baselines, weakly supervised Temporal Masking, is not as good as self-supervised Temporal Masking. Still, it remains of interest, so we did a complementary research work and proposed a novel network architecture better suited for weak supervision. The performance of the resulting model is above the State of the Art and below our self-supervised model, so it is valuable for contexts where self-supervision is unavailable.

Chapter 7

Conclusion and Perspectives

7.1 Conclusion

Simultaneous Localization and Mapping is an ability that is necessary for many applications as autonomous vehicles and robotics, in particular its extension specialized for dynamic environments, Dynamic SLAM. This ability is what enables navigation and interaction with the environment.

However, Dynamic SLAM is a significantly more difficult challenge than SLAM. The presence of dynamic objects makes it impossible to rely on the convenient assumption that the world is static. The purpose of running a Dynamic SLAM in dynamic environments is to ensure that, at all times, poses are computed with respect to the correct frame of reference; we showed that several factors complicate this task, namely situations known as motion consensus inversions and excessive masking.

A major SLAM challenge is to develop a method that works in any environment, including dynamic ones. Designing a Dynamic SLAM that works is not trivial, but it has already been done; designing one that works in *any* scenario is, however, extremely difficult. Generalization to arbitrary scenarios requires the ability of understanding on-the-fly which frame of reference the user expects to be used no matter what is happening in the scene – *e.g.*, a drone flying in a forest, a busy road or a ship exploring the seas. Such generalization capabilities require a level of intelligence far above what AIs can do today.

This thesis proposes to overcome the SLAM challenge with a self-supervised approach. Instead of trying to develop increasingly complex solutions to make a SLAM more general, our philosophy is to learn what the SLAM has to do to improve itself from data provided by the user – data that are representative of the scenarios that the SLAM algorithm has to support. Our Dynamic SLAM automatically learns what dynamic objects it should mask (*i.e.*, filter) and when it should do so to maximize SLAM performance, all from unlabeled training sequences provided by the user.

The first part of our work consisted in creating a setup for evaluation. The existing datasets were quite limited, seldom having very difficult scenarios like motion consensus inversions. Hence, they are not representative of difficult contexts that exist in real life. Moreover, the existing metrics – the omnipresent accuracy metric ATE RMSE and the less popular robustness metric Tracking Rate – tend to be misleading in these scenarios. A proper performance analysis needs to compare these two metrics at the same time, making comparisons complex. For these reasons, we proposed our own datasets, containing difficult dynamic scenarios, in particular scenarios with motion consensus inversions or risk of excessive masking. We also proposed the Penalized ATE RMSE, Success Rate and Unified SLAM Metric, which combine ATE RMSE and Tracking Rate in order to better reflect user needs and make scalar comparisons simple and meaningful.

Once we clarified the difficulties of Dynamic SLAMs and had ways to evaluate the robustness of Dynamic SLAMs with respect to them, we tackled the problem of what objects to mask in a given scenario. Instead of making educated guesses, which tend to be biased, we developed a self-supervised approach that automatically learns from SLAM outliers what should be masked. Outliers

are features in an image that the SLAM rejects for not fitting the static world model, and they tend to be concentrated on dynamic objects when they start moving. We do not assume the nature of objects to mask: the SLAM learns to segment them from outliers. This means that objects that are naturally ignored by the chosen SLAM, *e.g.*, textureless objects, may very well never need to be masked. Thanks to our method, we made our Dynamic SLAM safe from motion consensus inversions in any scenario from which we have easy sequences to learn from, surpassing the State of the Art.

Our last contribution is to add the temporal aspect to the act of masking objects to our Dynamic SLAM, *i.e.*, Temporal Masking. It appears that never masking objects or masking them all the time is suboptimal. Hence, we propose to learn the masking decisions that directly maximize SLAM performance through self-supervision. Without any assumption on the motion of objects, we developed a method that automatically annotates video sequences with masking decisions, annotations that we learn with our Temporal Masking Network. Our memory-based network can go beyond the capabilities of the usual approaches based on instantaneous motion detection, making our method perform beyond the State of the Art on current SLAM datasets as well as on our own dataset. In particular, our approach handles motion consensus inversions while preventing excessive masking.

7.2 Perspectives

After this thesis, there are three main tasks that may be considered: research to further improve the proposed methods, improving the technology readiness level for future industrialization, and new applications of the developed concepts.

7.2.1 Further Research

Regarding further research, we are considering several improvements in future works.

Further Research on Outlier-based Masking

In our outlier-based masking method, we analyze outliers and compute bounding boxes with geometry alone. Motion learning approaches [50] could be used as an extra source of information to identify more accurately the outlier-generating objects in the scene. They might also prove useful to shorten the computation pipeline, reducing the first steps to a single module. Another difficulty of object masking is dealing with object occlusions – *i.e.*, when an object is only partially visible –, which complicate the creation of the training database. Learned motion estimation [52] and occluded keypoint localization [77] would improve the quality and consistence of the training database as they may be able to segment objects even when they are barely visible, which our proposed approach has difficulty doing so.

Further Research on Temporal Masking

Instance-level Masking. The proposed approach makes masking decisions at the class level – to mask or not all objects of a certain class. It cannot make different decisions, within the same image, for different instances of the same class: *e.g.*, to mask a moving car while not masking a parked car. Since networks as Mask R-CNN compute instance-level masks, it seems valuable to explore a strategy that considers class instances separately. Note that computing temporal masking decisions for every instance separately is a major challenge that will require significant research. It increases the dimension of the temporal masking space: for a single frame, we do not have one binary decision per class but a potentially higher, *variable* (and unknown) number of binary decisions per class. Likewise, the Temporal Masking network has to be modified to be able to learn masking decisions for variable number of objects. This means that the output of the Temporal Masking (for a single image as input) would not be a vector of fixed size n – where n is the number of classes – but n sets of binary values of variable size. Moreover, we cannot separate the spatial aspect from the temporal one in the network: assuming that the neural network can handle instance-level inference, each inferred temporal masking

decision must then be matched to a specific detected object instance within the image, and not just to segmentation classes that apply to the whole image. This further complicates the question.

Combining Semantic and Temporal Masking. It would be useful to jointly learn to segment objects and learn when to mask them. Both could synergize and improve each other, in the same sense that depth and ego-motion estimation learning are often joined. A single network could take as input series of images and output semantic masks only when appropriate, without separating binary decisions and semantic networks. There would be no need to output binary masking decisions: objects simply would not be segmented when it is not appropriate.

End-to-end SLAMs. Temporal Masking is also of interest for end-to-end SLAMs. Our approach is a feature-based SLAM aided by learned algorithms, so we can clearly separate Temporal Masking from the rest of the SLAM. End-to-end SLAMs are single networks that take as input images and output trajectories. They do have features and maps, but they are encoded within the layers of the network and in practice too complex to extract, much less filter. Therefore, we preferred to rely on feature-based SLAM: we can create a general approach that applies to any feature-based SLAM without worrying about AI interpretability issues. Nonetheless, if in the future we understand how networks for end-to-end SLAM work and where the key features for the implicit mapping task are, a Temporal Masking module could be integrated in the neural network to boost its performance.

Better performance. Temporal Masking method can still be improved: we noticed during our benchmarks, especially on the ConsInv-Outdoors and ConsInv-Indoors dataset, that objects are not always masked optimally compared to manual annotations. Adding other types of sensors as input (*e.g.*, depth sensors, an IMU) could help making appropriate masking decisions when the main camera provides ambiguous information, for instance when it is exceedingly close to an object. For now, the Temporal Masking Network is split in two steps: the computation of image features and the computation of masking decisions from these features, mostly due to hardware limitations (memory and computation time). Making the network truly end-to-end would ensure that its performance is not limited by an imperfect match between the information contained in the features and the task we are trying to train the network for.

Finally, we could have the possibility of controlling the acceptable risk of SLAM drift at runtime. When annotating sequences, we have to find the right balance between masking too much out of caution and masking not enough and risking a drift, which is unacceptable in some cases: *e.g.*, a SLAM for autonomous cars that drifts could cause an accident. In our approach, this balance is implicit in the annotation process. It would be convenient to have a configurable network hyperparameter at runtime that controls this balance: this would make the model more flexible, able to adapt to situations of different risk. The difficulty is that the annotation process would need to generate data for diverse levels risk and the network architecture modified to integrate the notion of controllable risk.

7.2.2 Improving Technology Readiness Level for Future Industrialization

Regarding future industrialization, this thesis was carried out at the LVML whose primary purpose is to provide localization and modelling solutions for industrial partners. The ability of automatically adapting the SLAM to arbitrary dynamic scenarios is, therefore, valuable. Nonetheless, the Technology Readiness Level (TRL) is still low: in other words, the proposed methods need to be further tested and improved before being used in industrial applications. Once the TRL is increased enough, our work will allow LVML's SLAM algorithms to become more robust to unknown dynamic objects in industrial contexts.

Integration into the Existing SLAM Base

Increasing the technology readiness level implies integrating the proposed methods into the LVML tech stack. We expect self-supervised object masking to synergize well with the existing constrained SLAM technology. Constrained SLAM [95] relies on known 3D models (*e.g.*, cars, industrial parts, etc.) to identify objects and improve SLAM accuracy. The ability to identify outlier-generating objects could

be paired with constrained SLAM to label 3D models as static/dynamic. Another idea is to make model alignment easier as outliers indicate the pose of dynamic objects. The LVML also works on Augmented Reality and Diminished Reality: in these, correctly identifying objects to process or to remove – including outliers – is necessary, and our method could help doing so.

The automation of both outlier-based masking and Temporal Masking is not trivial and requires a significant work on the engineering aspect of our implemented pipelines. They first must be adapted to the LVML's own SLAM: we did not use it since it contains proprietary code and using ORB-SLAM makes result comparison simpler w.r.t. the State of the Art, but LVML's SLAM is designed for industrial applications. In addition, as our pipelines includes data collection and network training steps, it is necessary to industrialize them in a data-aware fashion.

Combining Outlier-based Masking and Temporal Masking into a Single Pipeline

We presented our two major contributions separately. However, they synergize well: the former is a solution to learn how to mask unknown objects, and the latter is a solution to learn when to apply these masks. Since they are complementary, it would be convenient to integrate both methods into a single, robust pipeline. This pipeline would take as input sequences representative of the environment we want the SLAM algorithm to process and would output one model to segment dynamic objects and one model to compute the corresponding masking decisions.

Furthermore, the idea of combining the contribution in a single pipeline is also applicable to the idea of combining semantic and Temporal Masking. In this case, the input of the pipeline would be unchanged but the output would be a unique model that segments objects only when needed.

7.2.3 New Applications

Regarding new applications, outlier-based masking could be extended to entities other than SLAM keypoints and the Temporal Masking paradigm could be used in new contexts.

Outlier-based masking is at its core a method to identify problematic entities in easy sequences and transfer this knowledge to identify the same kind of entity in difficult sequences where such entities would normally not be identified, preventing known issues. Like vaccination, an algorithm learns what to do in a harmless setting to survive dangerous situations. But these entities can virtually be anything in an algorithm. We can imagine using lines in a line-based SLAM [104] or image patches in direct SLAMs. Outside SLAMs, we could apply this concept to robustify data processing pipelines and reduce the chances of false positives/negatives. For instance, in electrocardiography (ECG) analysis we could learn to detect false positives – *e.g.*, abnormal but harmless readings due to temporary anxiety. Another possibility is to use the vaccination concept for feature engineering: this process is crucial for reliable performance of machine learning systems, and it could be partially learned on easy datasets and then applied on difficult datasets. As we can see, the concept of "outlier-based vaccination" is very general – we only need to find where to apply it.

Certain scenarios include reflections that may confuse the SLAM. Specularities (mirror-like reflections of light sources) tend to generate misleading features and consequently lower SLAM accuracy. Most SLAM algorithms do not explicitly process reflections as their effect on accuracy is usually negligible. However, as for motion consensus inversions, this means that the SLAM is likely to fail in very difficult scenarios. On the other hand, there is significant research on specularities [101], including within the LVML itself: research on the structural properties of specularities applied to their propagation [79] and prediction [72, 73] for Augmented Reality (*e.g.*, object retexturing) and Diminished Reality (removal of objects that takes overlapping specularities into account). Temporal Masking could be applied to specularity detection instead of object detection, improving SLAM performance in these circumstances. The idea is to decide when to remove or not specularities – note that this question is not trivial, as specularity detection is trickier than object detection due to being non-physical and prone to false positives that Temporal Masking could potentially reduce. More generally, Temporal Masking could be applied to any future elements that a user can segment and needs to know if they

should be masked. The interest of Temporal Masking is the only thing the user needs to change is the module computing segmentation masks.

Finally, we can consider that Temporal Masking is a method that maximizes the performance of an algorithm that takes as input a time series by triggering a specific action when appropriate. This concept could potentially be applied to contexts different from series of images and masking choices. For instance, in path planning algorithms [69], the action could be to block certain directions that are expected to lead to dead ends based on the last image inputs or on the local map. For instance, if a robot is exploring a cave system, a trained model could know what patterns of caves indicate danger, even if the sensors themselves cannot detect anything due to lack of light, or any other reason. In a way, we would be giving an "instinct" to the robot. Interestingly, there is a parallel between instincts and Temporal Masking. Instincts guide the behavior of animals for their own benefit without rational conscious thoughts from them[1]; similarly, Temporal Masking signals an algorithm when to mask objects or not to maximize SLAM performance, even if we do not understand the underlying model. More generally, the input could potentially be any kind of time series, like the heart rate input for sports coaching. The model could learn when to increase/decrease the effort so as to improve training results. The key takeaway is: Temporal Masking is not about detecting anomalies; it is about finding out what (not) to do to maximize the chances of reaching a certain goal.

We showed that the idea of learning from outliers and the concept of Temporal Masking both have a lot of potential, especially when applied in contexts out of SLAM applications. Hopefully, these ideas will inspire future researchers for novel applications in their own field!

[1] From https://en.wiktionary.org/wiki/instinct

Appendix A

Published SLAM datasets

The SLAM landscape has seen a variety of SLAM datasets in recent years. We give links and essential info about currently available datasets in table A.1 and table A.2[1]. The links in the table are all available as of the publication of this thesis. The datasets used in this thesis are in chapter 4.

[1]We used the site https://github.com/youngguncho/awesome-slam-datasets as a source for part of the table.

Name	Link	Affiliation	Year	Environment	Platform
VIODE	https://github.com/kminoda/VIODE	EPFL, U. Tokyo	2021	Urban	Veh
	Minoda, Koji *et al.* "VIODE: A Simulated Dataset to Address the Challenges of Visual-Inertial Odometry in Dynamic Environments". IEEE Robotics and Automation Letters 2021,				
Virtual KITTI 2	https://europe.naverlabs.com/research/computer-vision/proxy-virtual-worlds-vkitti-2/	Never Labs	2020	Urban	Veh
	Cabon, Yohann ; Murray, Naila ; Humenberger, Martin. "Virtual KITTI 2". Arxiv:2001.10773, 2020.				
Waymo	https://waymo.com/open	Waymo	2020	Urban	Veh
	Sun, Pei *et al.* "Scalability in perception for autonomous driving: Waymo open dataset". Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020.				
NuScenes	https://www.nuscenes.org/	nuScenes	2020	Urban	Veh
	H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan and O. Beijbom. "nuScenes: A multimodal dataset for autonomous driving". CVPR 2020.				
4Seasons	https://www.4seasons-dataset.com/	TUM	2020	Urban	Veh
	P. Wenzel ; *et al.* "4Seasons: A Cross-Season Dataset for Multi-Weather SLAM in Autonomous Driving". Proceedings of the German Conference on Pattern Recognition (GCPR) 2020.				
OpenLORIS	https://lifelong-robotic-vision.github.io/dataset/scene.html	Intel	2020	Indoor	Mob
	Xuesong Shi *et al.* "Are We Ready for Service Robots? The OpenLORIS-Scene Datasets for Lifelong SLAM". 2020 International Conference on Robotics and Automation (ICRA).				
ICL-Dataset	https://peringlab.org/lmdata/	Imperial College	2019	Indoor	Hand, MAV
	Sajad Saeedi *et al.* "Characterizing Visual Localization and Mapping Datasets". International Conference on Robotics and Automation (ICRA), 2019,				
FMDataset	https://github.com/zhuzunjie17/FastFusion	Hangzhou Dianzi / Tsinghua	2019	Indoor	Hand
	Shan, Zeyong, Ruijian Li, and Sören Schwertfeger. "RGBD-Inertial Trajectory Estimation and Mapping for Ground Robots." Sensors 19.10 (2019): 2251.				
UZH-FPV Drone Racing	https://fpv.ifi.uzh.ch/	UZH, ETH	2019	Indoor, Outdoor	UAV
	J. Delmerico *et al.* "Are We Ready for Autonomous Drone Racing? The UZH-FPV Drone Racing Dataset" IEEE International Conference on Robotics and Automation (ICRA), 2019.				
Syncity (Simulator)	https://www.cvedia.com/simulation/	CVEDIA	2019	Urban, Terrain	Veh, UAV
Bonn RGB-D	https://www.ipb.uni-bonn.de/data/rgbd-dynamic-dataset/	U. Bonn	2019	Indoor	Hand
	Palazzolo *et al.* "ReFusion: 3D Reconstruction in Dynamic Environments for RGB-D Cameras Exploiting Residuals". IROS 2019.				
ETH3D	https://www.eth3d.net/slam_overview	ETH	2019	Indoors	Hand
	T. Schöps, T. Sattler, M. Pollefeys, "BAD SLAM: Bundle Adjusted Direct RGB-D SLAM", Conference on Computer Vision and Pattern Recognition (CVPR), 2019.				
Segway DRIVE	http://drive.segwayrobotics.com/	Segway	2019	Urban	Mob
	Jianzhu Huai, Yusen Qin, Fumin Pang, Zichong Che. "Segway DRIVE Benchmark: Place Recognition and SLAM Data Collected by A Fleet of Delivery Robots".				
ADVIO Dataset	https://github.com/AaltoVision/ADVIO	Aalto U	2018	Urban	Hand
	Saraee, Elham, Mona Jalal, and Margrit Betke. "SAVOIAS: A Diverse, Multi-Category Visual Complexity Dataset." arXiv preprint arXiv:1810.01771 (2018).				
DeepIO Dataset	http://deepio.cs.ox.ac.uk/	Oxford	2018	Indoor	Hand
	Chen, Changhao, *et al.* "OxIOD: The Dataset for Deep Inertial Odometry." arXiv preprint arXiv:1809.07491 (2018).				
Aqualoc Dataset	http://www.lirmm.fr/aqualoc/	ONERA-DTIS	2018	Underwater	ROV
	Ferrera, Maxime, *et al.* "The Aqualoc Dataset: Towards Real-Time Underwater Localization from a Visual-Inertial-Pressure Acquisition System."arXiv:1809.07076.				
Rosario Dataset	http://www.cifasis-conicet.gov.ar/robot/doku.php	CONICET-UNR	2018	Terrain	Mob
	Taihú Pire *et al.* "The Rosario Dataset: Multisensor Data for Localization and Mapping in Agricultural Environments". In: International Journal of Research Robotics, 2018.				
InteriorNet	https://interiornet.org/	Imperial College	2018	Indoor	Hand
	Li, Wenbin, *et al.* "InteriorNet: Mega-scale multi-sensor photo-realistic indoor scenes dataset." arXiv preprint arXiv:1809.00716 (2018).				
SPO Dataset	https://www.h-ka.de/isrg/publications/vod	TUM, Karlsruhe	2018	Urban	Hand
	N. Zeller, F. Quint, U. Stilla (2018): A Synchronized Stereo and Plenoptic Visual Odometry Dataset, arXiv:1807.09372.				
KAIST Day/Night	https://sites.google.com/view/multispectral/home	KAIST-RCV	2018	Urban	Veh
	Choi, Yukyung, *et al.* "KAIST Multi-Spectral Day/Night Data Set for Autonomous and Assisted Driving." IEEE Transactions on Intelligent Transportation Systems 19.3 (2018): 934-948.				
TUM-Visual-Inertial	https://vision.in.tum.de/data/datasets/visual-inertial-dataset	TUM	2018	Indoor, Urban	Hand
	Schubert, David, *et al.* "The TUM VI Benchmark for Evaluating Visual-Inertial Odometry." arXiv preprint arXiv:1804.06120 (2018).				
Complex Urban	https://sites.google.com/view/complex-urban-dataset	KAIST-IRAP	2018	Urban	Veh
	Jinyong Jeong ; Younggun Cho ; Young-Sik Shin ; Hyunchul Roh and Ayoung Kim. "Complex Urban Dataset with Multi-level Sensors from Highly Diverse Urban Environments". IJRR, 2019,				
Multi Vech Event	https://daniilidis-group.github.io/mvsec/	Upenn	2018	Urban	Veh
	Zhu, Alex Zihao, *et al.* "The Multi Vehicle Stereo Event Camera Dataset: An Event Camera Dataset for 3D Perception." IEEE Robotics and Automation Letters (2018).				
VI Canoe	https://databank.illinois.edu/datasets/IDB-9342111	UIUC	2018	Terrain	USV
	Miller, Martin, Soon-Jo Chung, and Seth Hutchinson. "The Visual–Inertial Canoe Dataset." The International Journal of Robotics Research 37.1 (2018): 13-20.				
RPG-event	http://rpg.ifi.uzh.ch/davis_data.html	ETH-RPG	2017	Indoor	UAV / Hand
	Mueggler, Elias, *et al.* "The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM." The International Journal of Robotics Research, 2017				
Underwater Cave	http://cirs.udg.edu/caves-dataset/	UDG	2017	Underwater	AUV
	Mallios, Angelos, *et al.* "Underwater caves sonar data set." The International Journal of Robotics Research (2017): 0278364917732838.				
Robot @ Home	http://mapir.isa.uma.es/mapirwebsite/index.php/mapir-downloads/203-robot-at-home-dataset.html	MRPT	2017	Indoor	Mob
	Ruiz-Sarmiento *et al.* "Robot@ home, a robotic dataset for semantic mapping of home environments." The International Journal of Robotics Research (2017)				
Zurich Urban MAV	http://rpg.ifi.uzh.ch/zurichmavdataset.html	ETH-RPG	2017	Urban	UAV
	Andras L. Majdik, Charles Till, Davide Scaramuzza. "The Zurich Urban Micro Aerial Vehicle Dataset". International Journal of Robotics Research, April 2017.				
SceneNet RGB-D	https://robotvault.bitbucket.io/scenenet-rgbd.html	Imperial	2017	Indoor	Hand
	John McCormac *et al.* "Scenenet rgb-d: Can 5m synthetic images beat generic imagenet pre-training on indoor segmentation." IEEE International Conference on Computer Vision (ICCV), 2017.				
Symphony Lake	http://dream.georgiatech-metz.fr/?q=node/79	Georgia Tech	2017	Terrain (Lake)	USV
	Griffith, Shane, Georges Chahine, and Cédric Pradalier. "Symphony Lake Dataset." The International Journal of Robotics Research 36.11 (2017): 1151-1158.				
Agricultural robot	http://www.ipb.uni-bonn.de/data/sugarbeets2016/	Bonn	2017	Terrain	Mob
	Chebrolu, Nived, *et al.* "Agricultural robot dataset for plant classification, localization and mapping on sugar beet fields." The International Journal of Robotics Research 36.10 (2017)				
Beach Rover	https://robotics.estec.esa.int/datasets/katwijk-beach-11-2015/	TEC-MMA	2017	Terrain	Mob
	Hewitt, Robert A., *et al.* "The Katwijk beach planetary rover dataset." The International Journal of Robotics Research (2017): 0278364917737153.				
CARLA (Simulator)	https://carla.org/	Intel	2017	Urban	Veh
	Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez,Vladlen Koltun. "CARLA: An Open Urban Driving Simulator". Proceedings of the 1st Annual Conference on Robot Learning, 2017.				
Airsim (Simulator)	https://microsoft.github.io/AirSim/	Microsoft	2017	Urban	Veh, UAV
	Shital Shah, Debadeepta Dey, Chris Lovett, Ashish Kapoor. "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles". arXiv:1705.05065				
EuRoc	http://projects.asl.ethz.ch/datasets/doku.php?id=kmavvisualinertialdatasets	ETH-ASL	2016	Indoor	UAV
	M. Burri *et al.* "The EuRoC Micro Aerial Vehicle Datasets". The International Journal of Robotics Research (IJRR), 2016.				
TUM-MONO	https://vision.in.tum.de/data/datasets/mono-dataset	TUM	2016	Indoor, Urban	Hand
	Engel, Jakob, Vladyslav Usenko, and Daniel Cremers. "A photometrically calibrated benchmark for monocular visual odometry." arXiv preprint arXiv:1607.02555 (2016).				
Cityscape	https://www.cityscapes-dataset.com/	Daimler AG	2016	Urban	Veh
	M. Cordts *et al.* The Cityscapes Dataset for Semantic Urban Scene Understanding. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016				

Table A.1 – List of published SLAM datasets, part 1.

Name	Link	Affiliation	Year	Environment	Platform
Solar-UAV	http://projects.asl.ethz.ch/datasets/doku.php?id=fsr2015	ETHZ	2016	Terrain	UAV
	P. Oettershagen *et al.* "Long-Endurance Sensing and Mapping Using a Hand-Launchable Solar-Powered UAV". International Conference on Field and Service Robotics (FSR), 2016				
Oxford-robotcar	http://robotcar-dataset.robots.ox.ac.uk	Oxford	2016	Urban	Veh
	Maddern, Will, *et al.* "1 year, 1000 km: The Oxford RobotCar dataset." The International Journal of Robotics Research (2016): 0278364916679498.				
NCLT	http://robots.engin.umich.edu/nclt/	UMich	2016	Urban	Mob
	Carlevaris-Bianco *et al.* "University of Michigan North Campus long-term vision and lidar dataset." The International Journal of Robotics Research (2016)				
MPO-Japan	http://robotics.ait.kyushu-u.ac.jp/kyushu_datasets/outdoor_dense.html	Kyushu U	2016	Urban, Terrain	Veh
	Jung, Hojung, *et al.* "Multi-modal panoramic 3D outdoor datasets for place categorization." Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on. IEEE, 2016.				
Cartographer	https://google-cartographer-ros.readthedocs.io/en/latest/data.html	Google	2016	Indoor	Hand
	W. Hess, D. Kohler, H. Rapp, and D. Andor, Real-Time Loop Closure in 2D LIDAR SLAM, in Robotics and Automation (ICRA), 2016 IEEE International Conference on. IEEE, 2016. pp. 1271–1278.				
Virtual KITTI	https://europe.naverlabs.com/research/computer-vision/proxy-virtual-worlds-vkitti-1/	Never Labs	2016	Urban	Veh
	Adrien Gaidon *et al.* "Virtual worlds as proxy for multi-object tracking analysis". Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, 2016.				
CCSAD	https://aplicaciones.cimat.mx/Personal/jbhayet/research	CIMAT	2015	Urban	Veh
	R. Guzman *et al.* "Towards Ubiquitous Autonomous Driving: The CCSAD Dataset". In Conference on Computer Analysis of Images and Patterns (CAIP), 2015.				
TUM-Omni	https://vision.in.tum.de/data/datasets/omni-lsdslam	TUM	2015	Indoor, Urban	Hand
	David Caruso *et al.* "Large-scale direct slam for omnidirectional cameras." Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference 2015.				
Augmented ICL-NUIM	http://redwood-data.org/indoor/index.html	Redwood	2015	Indoor	Hand
	Sungjoon Choi *et al.* "Robust reconstruction of indoor scenes." Computer Vision and Pattern Recognition (CVPR), 2015				
Cambridge Landmark	http://mi.eng.cam.ac.uk/projects/relocalisation/	Cambridge	2015	Urban	Hand
	A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," in ICCV, 2015.				
SFU	http://autonomy.cs.sfu.ca/sfu-mountain-dataset/	QUT	2015	Outdoor	Veh
	Jake Bruce *et al.* "The SFU Mountain Dataset: Semi-Structured Woodland Trails Under Changing Environmental Conditions," IEEE Int. Conf. on Robotics and Automation Workshop, 2015.				
ICL-NUIM	https://www.doc.ic.ac.uk/∼ahanda/VaFRIC/iclnuim.html	Imperial	2014	Indoor	Hand
	A. Handa *et al.* "A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM". IEEE International Conference on Robotics and Automation (ICRA), 2014				
MRPT-Malaga	https://www.mrpt.org/robotics_datasets	MRPT	2014	Urban	Veh
	J.L. Blanco-Claraco *et al.* "The Málaga Urban Dataset: High-rate Stereo and Lidars in a realistic urban scenario", The International Journal of Robotics Research (IJRR), 2014				
KITTI	http://www.cvlibs.net/datasets/kitti/index.php	KIT	2013	Urban	Veh
	A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision Meets Robotics: The KITTI Dataset. The International Journal of Robotics Research (IJRR), 32(11):1231–1237, 2013				
Canadian Planetary	http://asrl.utias.utoronto.ca/datasets/3dmap/#Datasets	UToronto	2013	Terrain	Mob
	Tong, C., Gingras, D., Larose, K., Barfoot, T. D., and Dupuis, E. "The Canadian Planetary Emulation Terrain 3D Mapping Dataset." International Journal of Robotics Research, 2013				
Microsoft 7 scenes	https://www.doc.ic.ac.uk/∼ahanda/VaFRIC/iclnuim.html	Microsoft	2013	Indoor	Hand
	J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. Fitzgibbon, "Scene coordinate regression forests for camera relocalization in rgb-d images," in CVPR, June 2013.				
SeqSLAM	http://www.tu-chemnitz.de/etit/proaut/datasets/nordland/64x32-grayscale-1fps.tar	QUT	2012	Urban	Veh
	M. J. Milford and G. F. Wyeth. "SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights". IEEE International Conference on Robotics and Automation (ICRA), 2012				
ETH-challenging	http://projects.asl.ethz.ch/datasets/doku.php?id=laserregistration:laserregistration	ETH-ASL	2012	Urban, Terrain	Hand
	F. Pomerleau, M. Liu, F. Colas, and R. Siegwart. Challenging data sets for point cloud registration algorithms. The International Journal of Robotics Research (IJRR), 31(14):1705–1711, 2012				
TUM-RGBD	https://vision.in.tum.de/data/datasets/rgbd-dataset	TUM	2012	Indoor	Hand / Mob
	J. Sturm *et al.* "A Benchmark for the Evaluation of RGB-D SLAM Systems". IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2012				
ASRL-Kagara-airborn	https://michaelwarren.info/docs/datasets/kagaru-airborne-stereo/	UToronto	2012	Terrain	UAV
	M. Warren *et al.* "Large Scale Monocular Vision-only Mapping from a Fixed-Wing sUAS". International Conference on Field and Service Robotics (FSR), 2012				
Devon Island Rover	http://asrl.utias.utoronto.ca/datasets/devon-island-rover-navigation/	UToronto	2012	Terrain	Mob
	P. T. Furgale *et al.* "The Devon Island Rover Navigation Dataset." International Journal of Robotics Research, 2012				
ACFR Marine	http://marine.acfr.usyd.edu.au/datasets/	ACFR	2012	Underwater	AUV
UTIAS Multi-Robot	http://asrl.utias.utoronto.ca/datasets/mrclam/	UT-IAS	2011	Urban	Mob
	K. Y. K. Leung *et al.* "The UTIAS multi-robot cooperative localization and mapping dataset". The International Journal of Robotics Research (IJRR), 2011				
Ford Campus	http://robots.engin.umich.edu/SoftwareData/Ford	UMich	2011	Urban	Veh
	G. Pandey, J. R. McBride, and R. M. Eustice. Ford Campus vision and LIDAR data set. The International Journal of Robotics Research (IJRR), 2011				
San Francisco	https://sites.google.com/site/chenmodavid/datasets	Stanford	2011	Urban	Veh
	Chen, David M., *et al.* "City-scale landmark identification on mobile devices." Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on. IEEE, 2011.				
MIT-DARPA-Urban	http://grandchallenge.mit.edu/wiki/index.php?title=PublicData	MIT	2010	Urban	Veh
	A. S. Huang *et al.* "A High-rate, Heterogeneous Data Set From The DARPA Urban Challenge". The International Journal of Robotics Research (IJRR), 2010				
St Lucia Stereo	http://asrl.utias.utoronto.ca/∼mdw/uqstluciadataset.html	UToronto	2010	Urban	Veh
	M. Warren *et al.* "Unaided Stereo Vision based Pose Estimation". Australasian Conference on Robotics and Automation (ACRA), 2010.				
Marulan	http://sdi.acfr.usyd.edu.au/	ACFR	2010	Terrain	Mob
	Thierry Peynot *et al.* "The marulan data sets: Multi-sensor perception in a natural environment with challenging conditions." International Journal of Robotics Research, 2010.				
COLD	https://www.pronobis.pro/#data	KTH	2009	Indoor	Hand
	A. Pronobis and B. Caputo. COLD: The CoSy Localization Database. The International Journal of Robotics Research (IJRR), 28(5):588–594, 2009.				
NewCollege	http://www.robots.ox.ac.uk/NewCollegeData/	Oxford-Robot	2009	Urban	Mob
	M. Smith *et al.* "The New College Vision and Laser Data Set". International Journal of Robotics Research (IJRR), 2009.				
Rawseeds-indoor	http://www.rawseeds.org/home/category/benchmarking-toolkit/datasets/	Milano	2009	Indoor	Mob
	A. Bonarini *et al.* "Rawseeds: Robotics advancement through web-publishing of sensorial and elaborated extensive data sets". IROS workshop, 2006				
Rawseeds-outdoor	http://www.rawseeds.org/home/category/benchmarking-toolkit/datasets/	Milano	2009	Urban	Mob
	A. Bonarini *et al.* "Rawseeds: Robotics advancement through web-publishing of sensorial and elaborated extensive data sets". IROS workshop, 2006				
FABMAP	http://www.robots.ox.ac.uk/∼mobile/IJRR_2008_Dataset/	Oxford-Robot	2008	Urban	Veh
	Cummins, Mark, and Paul Newman. "FAB-MAP: Probabilistic localization and mapping in the space of appearance." International Journal of Robotics Research, 2008				
Older SLAM Benchmarks	http://ais.informatik.uni-freiburg.de/slamevaluation/datasets.php	Intel, MIT	2008		

Table A.2 – List of published SLAM datasets, part 2.

Appendix B

Complementary information

B.1 Influence of the number of features on SLAM performance

This section presents the full results corresponding to the experiments with Weighted ATE RMSE and Tracking Rate in section 3.2.2.

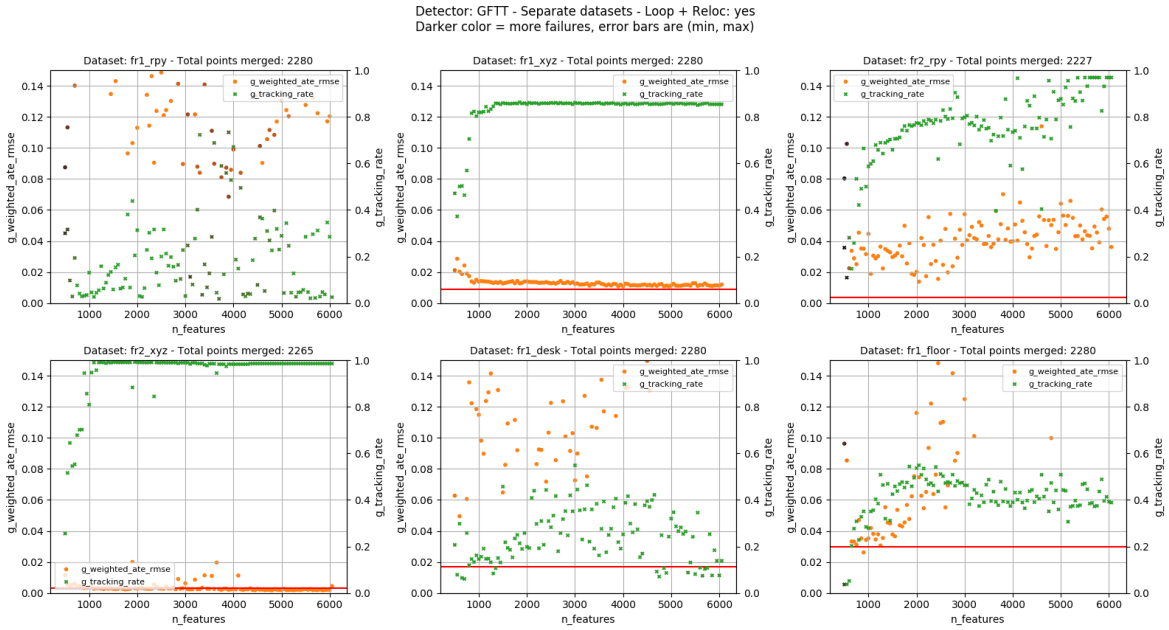


Figure B.1 – Benchmark of ORB-SLAM using the GFTT keypoint detector. We show the Weighted ATE RMSE and Tracking Rate. The red line indicates ORB-SLAM 2's performance in the original paper.

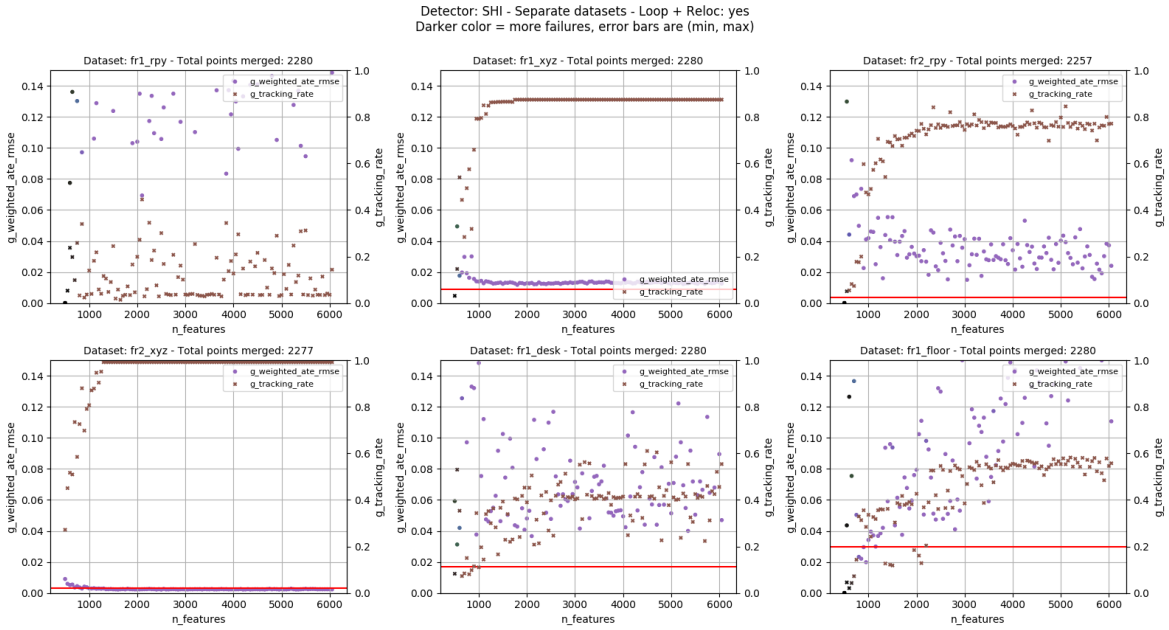


Figure B.2 – Benchmark of ORB-SLAM using the Shi keypoint detector. We show the Weighted ATE RMSE and Tracking Rate. The red line indicates ORB-SLAM 2's performance in the original paper.

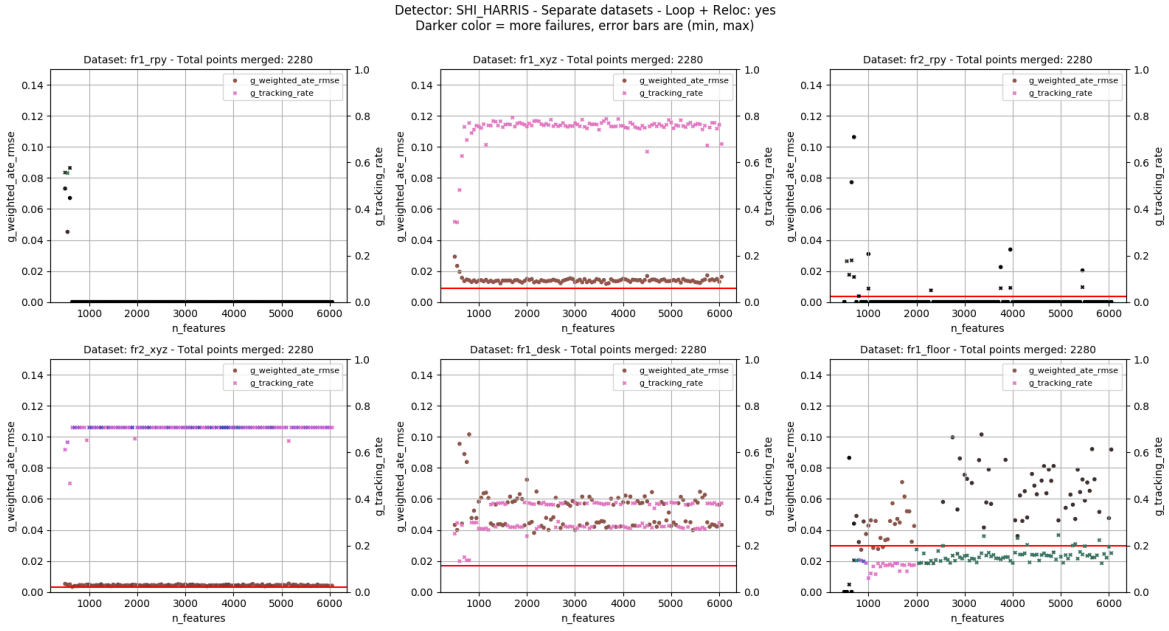


Figure B.3 – Benchmark of ORB-SLAM using the Shi-Harris keypoint detector. We show the Weighted ATE RMSE and Tracking Rate. The red line indicates ORB-SLAM 2's performance in the original paper.

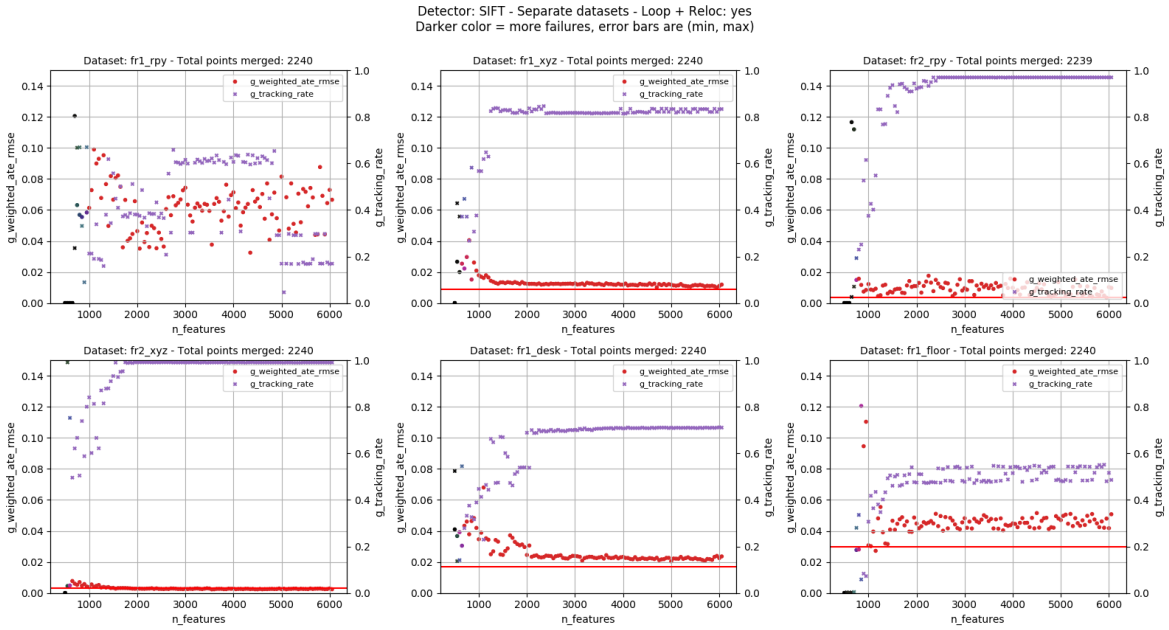


Figure B.4 – Benchmark of ORB-SLAM using the SIFT keypoint detector. We show the Weighted ATE RMSE and Tracking Rate. The red line indicates ORB-SLAM 2's performance in the original paper.

Bibliography

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274–2282, 2012.

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

[3] I. Ballester, A. Fontan, J. Civera, K. H. Strobl, and R. Triebel. DOT: Dynamic object tracking for visual SLAM. *CoRR (Arxiv)*, 2020.

[4] V. Balntas, K. Lenc, A. Vedaldi, and K. Mikolajczyk. Hpatches: A benchmark and evaluation of handcrafted and learned local descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[5] D. Barnes, W. Maddern, G. Pascoe, and I. Posner. Driven to distraction: Self-supervised distractor learning for robust monocular visual odometry in urban environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[6] A. Benzine, B. Luvison, Q. C. Pham, and C. Achard. Single-shot 3d multi-person pose estimation in complex images. *Pattern Recognition*, 112:107534, 2021.

[7] B. Besbes, S. N. Collette, M. Tamaazousti, S. Bourgeois, and V. Gay-Bellile. An interactive augmented reality system: a prototype for industrial maintenance training applications. In *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 269–270, 2012.

[8] B. Bescos, J. M. Fácil, J. Civera, and J. Neira. DynaSLAM: Tracking, mapping, and inpainting in dynamic scenes. *IEEE Robotics and Automation Letters*, 2018.

[9] J. L. Blanco-Claraco. A tutorial on **SE**(3) transformation parameterizations and on-manifold optimization, 2021.

[10] A. Bojko, R. Dupont, M. Tamaazousti, and H. Le Borgne. De slam robuste à slam dynamique par auto-apprentissage d'outliers. In *Reconnaissance des Formes, Image, Apprentissage et Perception (RFIAP)*, 2020.

[11] A. Bojko, R. Dupont, M. Tamaazousti, and H. Le Borgne. Learning to segment dynamic objects using slam outliers. In *25th International Conference on Pattern Recognition (ICPR)*, 2021.

[12] A. Bojko, H. Le Borgne, R. Dupont, and M. Tamaazousti. Procédé de localisation et cartographie simultanées intégrant un masquage temporel auto-supervisé et modèle d'apprentissage automatique pour générer un tel masquage., 2021. FR patent 2112893.

[13] S. Bourgeois, B. Meden, V. Gay-Bellile, M. Tamaazousti, and S. Knodel. Modeling, tracking, annotating and augmenting a 3d object in less than 5 minutes. *IEEE ISMAR Joint Workshop on Tracking Methods & Applications and TrakMark*, 2013.

[14] F. Bousefsaf, M. Tamaazousti, S. H. Said, and R. Michel. Image completion using multispectral imaging. *IET Image Processing*, 12(7):1164–1174, 2018.

[15] E. Brachmann, A. Krull, S. Nowozin, J. Shotton, F. Michel, S. Gumhold, and C. Rother. DSAC — Differentiable RANSAC for Camera Localization. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2492–2500, July 2017.

[16] H. M. S. Bruno and E. L. Colombini. LIFT-SLAM: a deep-learning feature-based monocular visual SLAM method. *arXiv:2104.00099 [cs]*, Mar. 2021. arXiv: 2104.00099.

[17] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 2016.

[18] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

[19] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *European Conference on Computer Vision (ECCV)*, 2018.

[20] J. Cheng, Y. Sun, W. Chi, C. Wang, H. Cheng, and M. Q. Meng. An accurate localization scheme for mobile robots using optical flow in dynamic environments. In *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2018.

[21] J. Cheng, Y. Sun, and M. Q.-H. Meng. Improving monocular visual SLAM in dynamic environments: an optical-flow-based approach. *Advanced Robotics*, 2019.

[22] G. Chican and M. Tamaazousti. Constrained patchmatch for image completion. In *Proceedings of the 10th International Symposium on Visual Computing*, pages 560–568. Springer, 2014.

[23] R. Clark, M. Bloesch, J. Czarnowski, S. Leutenegger, and A. J. Davison. Learning to Solve Nonlinear Least Squares for Monocular Stereo. In V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, editors, *Computer Vision – ECCV 2018*, Lecture Notes in Computer Science, pages 291–306. Springer International Publishing, 2018.

[24] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[25] E. Corona, A. Pumarola, G. Alenyà, and F. Moreno-Noguer. Context-aware Human Motion Prediction. *arXiv:1904.03419 [cs]*, Mar. 2020. arXiv: 1904.03419.

[26] J. Czarnowski, T. Laidlow, R. Clark, and A. J. Davison. DeepFactors: Real-Time Probabilistic Dense Monocular SLAM. *IEEE Robotics and Automation Letters*, 5(2):721–728, Apr. 2020. arXiv: 2001.05049.

[27] Q. Dai, V. Patil, S. Hecker, D. Dai, L. Van Gool, and K. Schindler. Self-supervised Object Motion and Depth Estimation from Video. *arXiv:1912.04250 [cs]*, May 2020. arXiv: 1912.04250.

[28] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067, 2007.

[29] D. DeTone, T. Malisiewicz, and A. Rabinovich. SuperPoint: Self-Supervised Interest Point Detection and Description. *arXiv:1712.07629 [cs]*, Dec. 2017. arXiv: 1712.07629.

[30] D. DeTone, T. Malisiewicz, and A. Rabinovich. Self-improving visual odometry. *arXiv:1812.03245 [cs]*, 2018.

[31] L. Ding and C. Feng. DeepMapping: Unsupervised Map Estimation From Multiple Point Clouds. *arXiv:1811.11397 [cs]*, Nov. 2018. arXiv: 1811.11397.

[32] G. Dissanayake, S. Huang, Z. Wang, and R. Ranasinghe. A review of recent developments in simultaneous localization and mapping. *2011 6th International Conference on Industrial and Information Systems*, pages 477–482, 2011.

[33] H. Durrant-Whyte, D. Rye, and E. Nebot. Localization of autonomous guided vehicles. In G. Giralt and G. Hirzinger, editors, *Robotics Research*, pages 613–625, London, 1996. Springer London.

[34] E. Eade. Lie groups for 2d and 3d transformations., 2013.

[35] J. Engel, V. Koltun, and D. Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40:611–625, 2018.

[36] J. Engel, T. Schoeps, and D. Cremers. Lsd-slam: large-scale direct monocular slam. In *Eur. Conf. Comput. Vis.*, volume 8690, pages 1–16, 09 2014.

[37] Y. Feng, Z. Liang, and H. Liu. Efficient deep learning for stereo matching with larger image patches. In *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 1–5, Oct. 2017.

[38] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig. Virtual worlds as proxy for multi-object tracking analysis. In *CVPR*, 2016.

[39] X. Gao, R. Wang, N. Demmel, and D. Cremers. LDSO: Direct sparse odometry with loop closure. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[40] G. Gay-Bellile, S. Bourgeois, M. Tamaazousti, S. Naudet-Collette, and S. Knodel. A mobile markerless augmented reality system for the automotive field. In *IEEE ISMAR Workshop on Tracking Methods and Applications (TMA)*, 2012.

[41] V. Gay-Bellile, S. Bourgeois, D. Larnaout, and M. Tamaazousti. Applications of augmented reality for the automobile industry. *Fundamentals of Wearable Computers and Augmented Reality, Second Edition*, pages 433–456, 2015.

[42] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[43] P. Geneva, J. Maley, and G. Huang. An Efficient Schmidt-EKF for 3D Visual-Inertial SLAM. *arXiv:1903.08636 [cs]*, Mar. 2019. arXiv: 1903.08636.

[44] B. A. Griffin and J. J. Corso. Learning Object Depth from Camera Motion and Video Object Segmentation. In A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, editors, *Computer Vision – ECCV 2020*, Lecture Notes in Computer Science, pages 295–312, Cham, 2020. Springer International Publishing.

[45] C. G. Harris and M. J. Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, 1988.

[46] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2 edition, 2003.

[47] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-CNN. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[48] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[49] B. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society A*, 4:629–642, 04 1987.

[50] Z. Huang, S. Zhang, J. Jiang, M. Tang, R. Jin, and M. H. Ang. Self-supervised motion learning from static images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1276–1285, June 2021.

[51] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *In Proc. UIST*, pages 559–568, 2011.

[52] S. Jiang, D. Campbell, Y. Lu, H. Li, and R. Hartley. Learning to estimate hidden motions with global motion aggregation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9772–9781, October 2021.

[53] M. Kaneko, K. Iwami, T. Ogawa, T. Yamasaki, and K. Aizawa. Mask-SLAM: Robust feature-based monocular SLAM by masking using semantic segmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2018.

[54] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007.

[55] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner. On measuring the accuracy of slam algorithms. *Autonomous Robots*, 27:387–407, 2009.

[56] J. Lamarca and J. M. M. Montiel. Camera tracking for slam in deformable maps. In *ECCV Workshops*, 2018.

[57] P. Lenz, J. Ziegler, A. Geiger, and M. Roser. Sparse scene flow segmentation for moving object detection in urban environments. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, 2011.

[58] J. J. Leonard and H. F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. *Proceedings IROS '91:IEEE/RSJ International Workshop on Intelligent Robots and Systems '91*, pages 1442–1447 vol.3, 1991.

[59] P. Li, X. Chen, and S. Shen. Stereo R-CNN based 3D Object Detection for Autonomous Driving. *arXiv:1902.09738 [cs]*, Feb. 2019. arXiv: 1902.09738.

[60] R. Li, S. Wang, Z. Long, and D. Gu. UnDeepVO: Monocular Visual Odometry Through Unsupervised Deep Learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7286–7291, May 2018.

[61] S. Li and D. Lee. RGB-d SLAM in dynamic environments using static point weighting. *IEEE Robotics and Automation Letters*, 2017.

[62] S. Li, X. Wang, Y. Cao, F. Xue, Z. Yan, and H. Zha. Self-supervised deep visual odometry with online adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[63] Z. Li, T. Dekel, F. Cole, R. Tucker, N. Snavely, C. Liu, and W. T. Freeman. Learning the depths of moving people by watching frozen people. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[64] K.-N. Lianos, J. L. Schönberger, M. Pollefeys, and T. Sattler. VSO: Visual Semantic Odometry. In V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, editors, *Computer Vision – ECCV 2018*, Lecture Notes in Computer Science, pages 246–263. Springer International Publishing, 2018.

[65] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *Computer Vision*, 2014.

[66] X. Lu, W. Wang, C. Ma, J. Shen, L. Shao, and F. Porikli. See more, know more: Unsupervised video object segmentation with co-attention siamese networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[67] H. Luo, Y. Gao, Y. Wu, C. Liao, X. Yang, and K. Cheng. Real-time Dense Monocular SLAM with Online Adapted Depth Prediction Network. *IEEE Transactions on Multimedia*, pages 1–1, 2018.

[68] W. Luo, A. G. Schwing, and R. Urtasun. Efficient Deep Learning for Stereo Matching. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5695–5703, Las Vegas, NV, USA, June 2016. IEEE.

[69] J. M. Mendes Filho, E. Lucet, and D. Filliat. Real-time distributed receding horizon motion planning and control for mobile multi-robot dynamic systems. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 657–663, 2017.

[70] Z. Min and E. Dunn. VOLDOR-SLAM: For the Times When Feature-Based or Direct Methods Are Not Good Enough. *arXiv:2104.06800 [cs]*, Apr. 2021. arXiv: 2104.06800.

[71] K. Minoda, F. Schilling, V. Wüest, D. Floreano, and T. Yairi. VIODE: A simulated dataset to address the challenges of visual-inertial odometry in dynamic environments. *IEEE Robotics and Automation Letters*, 2021.

[72] A. Morgand, M. Tamaazousti, and A. Bartoli. An empirical model for specularity prediction with application to dynamic retexturing. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 44–53. IEEE, 2016.

[73] A. Morgand, M. Tamaazousti, and A. Bartoli. A geometric model for specularity prediction on planar surfaces with multiple light sources. *Transactions on Visualization and Computer Graphics*, 24(5):1691–1704, 2017.

[74] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd. Real time localization and 3d reconstruction. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 363–370. IEEE, 2006.

[75] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. ORB-SLAM: A versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 2015.

[76] 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.

[77] N. D. Reddy, M. Vo, and S. G. Narasimhan. Occlusion-net: 2d/3d occluded keypoint localization using graph networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7318–7327, 2019.

[78] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.

[79] S. H. Said, M. Tamaazousti, and A. Bartoli. Image-based models for specularity propagation in diminished reality. *IEEE transactions on visualization and computer graphics*, 24(7):2140–2152, 2017.

[80] M. R. U. Saputra, P. P. B. de Gusmao, C. X. Lu, Y. Almalioglu, S. Rosa, C. Chen, J. Wahlström, W. Wang, A. Markham, and N. Trigoni. DeepTIO: A Deep Thermal-Inertial Odometry with Visual Hallucination. *arXiv:1909.07231 [cs]*, Jan. 2020. arXiv: 1909.07231 version: 2.

[81] M. R. U. Saputra, A. Markham, and N. Trigoni. Visual SLAM and structure from motion in dynamic environments: A survey. *ACM Comput. Surv.*, 2018.

[82] M. Schorghuber, D. Steininger, Y. Cabon, M. Humenberger, and M. Gelautz. SLAMANTIC - leveraging semantics to improve VSLAM in dynamic environments. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, 2019.

[83] J. L. Schönberger, M. Pollefeys, A. Geiger, and T. Sattler. Semantic Visual Localization. *arXiv:1712.05773 [cs]*, Dec. 2017. arXiv: 1712.05773.

[84] R. Scona, M. Jaimez, Y. R. Petillot, M. Fallon, and D. Cremers. StaticFusion: Background reconstruction for dense RGB-d SLAM in dynamic environments. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[85] M. E. A. Seddik, M. Tamaazousti, and R. Couillet. A kernel random matrix-based approach for sparse pca. In *International Conference on Learning Representations (ICLR)*, 2019.

[86] S. Shah, D. Dey, C. Lovett, and A. Kapoor. *AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles*, pages 621–635. Springer, 2018.

[87] X. Shi, D. Li, P. Zhao, Q. Tian, Y. Tian, Q. Long, C. Zhu, J. Song, F. Qiao, L. Song, Y. Guo, Z. Wang, Y. Zhang, B. Qin, W. Yang, F. Wang, R. Chan, and Q. She. Are we ready for service robots? the openloris-scene datasets for lifelong slam. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020.

[88] G. Silveira, E. Malis, and P. Rives. An efficient direct method for improving visual slam. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 4090–4095, 2007.

[89] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-d SLAM systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[90] K. Su, D. Yu, Z. Xu, X. Geng, and C. Wang. Multi-Person Pose Estimation with Enhanced Channel-wise and Spatial Information. *arXiv:1905.03466 [cs]*, May 2019. arXiv: 1905.03466.

[91] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz. PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[92] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2446–2454, 2020.

[93] Y. Sun, M. Liu, and M. Q. H. Meng. Improving RGB-d SLAM in dynamic environments: A motion removal approach. *Robotics and Autonomous Systems*, 2017.

[94] Y. Sun, M. Liu, and M. Q. H. Meng. Motion removal for reliable RGB-d SLAM in dynamic environments. *Robotics and Autonomous Systems*, 2018.

[95] M. Tamaazousti. *L'ajustement de faisceaux contraint comme cadre d'unification des méthodes de localisation: application à la réalité augmentée sur des objets 3D*. PhD thesis, PhD thesis, Université Blaise Pascal-Clermont-Ferrand II, 2013.

[96] M. Tamaazousti, S. Naudet-Collette, V. Gay-Bellile, S. Bourgeois, B. Besbes, and M. Dhome. The constrained slam framework for non-instrumented augmented reality. *Multimedia Tools and Applications*, 75(16):9511–9547, 2016.

[97] K. Tateno, F. Tombari, I. Laina, and N. Navab. CNN-SLAM: Real-Time Dense Monocular SLAM with Learned Depth Prediction. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6565–6574, July 2017.

[98] L. Tiwari, P. Ji, Q.-H. Tran, B. Zhuang, S. Anand, and M. Chandraker. Pseudo RGB-D for Self-Improving Monocular SLAM and Depth Prediction. *arXiv:2004.10681 [cs]*, Aug. 2020. arXiv: 2004.10681.

[99] S. Ullman. The interpretation of structure from motion. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 203:405 – 426, 1979.

[100] C. Ventura, M. Bellver, A. Girbau, A. Salvador, F. Marques, and X. Giro-i Nieto. Rvos: End-to-end recurrent network for video object segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[101] F. Wang, S. Ainouz, C. Petitjean, and A. Bensrhair. Specularity removal: A global energy minimization approach based on polarization imaging. *Computer Vision and Image Understanding*, 158:31–39, 2017.

[102] H. Wang, Y. Sun, and M. Liu. Self-Supervised Drivable Area and Road Anomaly Segmentation Using RGB-D Data For Robotic Wheelchairs. *IEEE Robotics and Automation Letters*, 4(4):4386–4393, Oct. 2019.

[103] K. Wang, Y. Lin, L. Wang, L. Han, M. Hua, X. Wang, S. Lian, and B. Huang. A unified framework for mutual improvement of SLAM and semantic segmentation. In *2019 International Conference on Robotics and Automation (ICRA)*, 2019.

[104] Q. Wang, Z. Yan, J. Wang, F. Xue, W. Ma, and H. Zha. Line flow based simultaneous localization and mapping. *IEEE Transactions on Robotics*, 37(5):1416–1432, 2021.

[105] Q. Wang, L. Zhang, L. Bertinetto, W. Hu, and P. H. S. Torr. Fast online object tracking and segmentation: A unifying approach. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[106] S. Wang, R. Clark, H. Wen, and N. Trigoni. DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2043–2050, Singapore, Singapore, May 2017. IEEE.

[107] S. Wang, R. Clark, H. Wen, and N. Trigoni. End-to-end, sequence-to-sequence probabilistic visual odometry through deep neural networks. *The International Journal of Robotics Research*, 2018.

[108] X. Wei, Y. Zhang, Z. Li, Y. Fu, and X. Xue. DeepSFM: Structure From Motion Via Deep Bundle Adjustment. *arXiv:1912.09697 [cs]*, Aug. 2020. arXiv: 1912.09697.

[109] L. Xiao, J. Wang, X. Qiu, Z. Rong, and X. Zou. Dynamic-SLAM: Semantic monocular visual localization and mapping based on deep learning in dynamic environment. *Robotics and Autonomous Systems*, 2019.

[110] B. Xu, W. Li, D. Tzoumanikas, M. Bloesch, A. Davison, and S. Leutenegger. MID-fusion: Octree-based object-level multi-instance dynamic SLAM. In *International Conference on Robotics and Automation (ICRA)*, 2019.

[111] F. Xue, X. Wang, S. Li, Q. Wang, J. Wang, and H. Zha. Beyond Tracking: Selecting Memory and Refining Poses for Deep Visual Odometry. *arXiv:1904.01892 [cs]*, Apr. 2019. arXiv: 1904.01892.

[112] F. Xue, X. Wang, J. Wang, and H. Zha. Deep Visual Odometry with Adaptive Memory. *arXiv:2008.01655 [cs]*, Aug. 2020. arXiv: 2008.01655.

[113] N. Yang, L. von Stumberg, R. Wang, and D. Cremers. D3VO: Deep Depth, Deep Pose and Deep Uncertainty for Monocular Visual Odometry. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1278–1289, Seattle, WA, USA, June 2020. IEEE.

[114] X. Yang, H. Luo, Y. Wu, Y. Gao, C. Liao, and K.-T. Cheng. Reactive obstacle avoidance of monocular quadrotors with online adapted depth prediction network. *Neurocomputing*, 325:142–158, Jan. 2019.

[115] K. M. Yi, E. Trulls, V. Lepetit, and P. Fua. LIFT: Learned Invariant Feature Transform. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision – ECCV 2016*, pages 467–483, Cham, 2016. Springer International Publishing.

[116] C. Yu, Z. Liu, X. Liu, F. Xie, Y. Yang, Q. Wei, and Q. Fei. DS-SLAM: A semantic visual SLAM towards dynamic environments. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[117] X. Zhang, L. Wang, Y. Zhao, and Y. Su. Graph-Based Place Recognition in Image Sequences with CNN Features. *Journal of Intelligent & Robotic Systems*, Aug. 2018.

[118] Y. Zhang and J. J. Leonard. A front-end for dense monocular SLAM using a learned outlier mask prior. *CoRR (Arxiv)*, 2021.

[119] H. Zhou, B. Ummenhofer, and T. Brox. DeepTAM: Deep Tracking and Mapping. In V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, editors, *Computer Vision – ECCV 2018*, Lecture Notes in Computer Science, pages 851–868. Springer International Publishing, 2018. ECCV 2018.

[120] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe. Unsupervised Learning of Depth and Ego-Motion from Video. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6612–6619, July 2017.