A Mixed Method for Monocular Simultaneous Localization and Mapping

Liza Belos

Université Gustave Eiffel

École doctorale n°532
Mathématiques, Sciences et Technologies de l'Information et de la Communication

THESIS MANUSCRIPT

to obtain the doctorate degree issued by the University Gustave Eiffel

A Mixed Method for Monocular Simultaneous Localization and Mapping

Liza Belos

February 2023

Jury

M. Pascal Monasse Professor HDR	École des Ponts ParisTech	Supervisor
Mme. Eva Dokladalova Professor HDR	ESIEE Paris	Supervisor
M. Cédric Demonceaux Professor HDR	Université Bourgogne Franche Comté	Reviewer
M. Antoine Manzanera Professor HDR	ENSTA Paris	Reviewer
Mme. Beatriz Marcotegui Professor HDR	Mines Paris	Examiner
M. Raouf Ben Jemaa Engineering Director	Trimble	Examiner
M. Jean-Philippe Tarel Senior Researcher	Université Gustave Eiffel	Examiner
M. Guillaume Bourmaud Associate Professor	Université de Bordeaux	Examiner

LIGM, Université Gustave Eiffel
IMAGINE, ENPC, CNRS, F-77454 Marne-la-Vallée

Abstract From the continuous image flow of a camera, Visual Simultaneous Localization and Mapping (SLAM) keeps track of the pose of this camera while making a 3D reconstruction, both in real-time. For maximal generality, this thesis focuses on SLAM methods that use only a monocular camera and no other sensors for augmented reality, autonomous drones, or autonomous driving. Recent SLAM pipelines significantly reduced the overall error. However, they are not robust and accurate enough for many applications. The produced drift is still high, and the existing algorithms do not succeed in tracking the camera in some videos. Two main categories of SLAM methods exist: direct and indirect. Direct methods use the optical flow of the video to estimate the pose and do the 3D reconstruction. They can reconstruct 3D point clouds which are highly accurate. However, they produce a lot of drift on high optical flow videos. Indirect methods use feature matching to estimate the pose and do the 3D reconstruction. Unlike direct methods, indirect methods can accurately track high optical flow video. On the other hand, these methods are sensitive to repetitive texture or high light change.

This manuscript presents a new SLAM method, which mixes indirect and direct methodologies, to benefit from the advantages of both. It relies on the generalized map, a data structure with a set of operators ensuring consistency and coherence of information between indirect and direct algorithms. The operators implement efficient algorithms that do not slow the SLAM's execution. MOD SLAM is a mixed direct and indirect SLAM made on top of the generalized map. At each step, MOD SLAM predicts which direct or indirect pipeline method will produce the lowest error. MOD SLAM does not execute both methods at each step but instead chooses which to execute. Consequently, MOD SLAM presents a competitive running time compared to the state-of-the-art methods, while increasing robustness and accuracy.

This manuscript presents in detail the state-of-the-art of Monocular SLAM. We develop our contribution, starting with the generalized map. We detail the MOD SLAM method, and present an analysis of the impact of the MOD SLAM decision system. To conclude, we show that MOD SLAM can run on 100% of the KITTI and TUM sequences. It increases the precision compared to the start-of-the-art methods while running at 20fps on high-end Android phones.

Key-words Image Processing; SLAM; Localization; Mapping; 3D; Augmented Reality; Autonomous Driving; Hybrid; Mixed; ORB-SLAM; DSO; MOD SLAM; Generalized Map; Optimisation; Gauss-Newton; Levenberg-Marquardt;

Résumé Les algorithmes de localisation et cartographie visuelles simultanées (SLAM) estiment en temps réel la pose d'une caméra à partir de son flux vidéo continu, tout en faisant une reconstruction 3D de la scène. Nous nous intéressons aux méthodes de SLAM qui n'utilisent qu'une caméra monoculaire et aucun autre capteur. Ces méthodes de SLAM sont utiles à des applications de réalité augmentée, ou de drones autonomes. Les recherches récentes dans le domaine du SLAM monoculaire ont réussi à considérablement réduire l'erreur globale. Cependant, les méthodes SLAM actuelles ne sont pas assez robustes et assez précises pour des applications telles que celles citées plus haut. En effet, les méthodes actuelles échouent sur certains types de vidéos, et produisent une dérive de l'estimation de la pose. Il existe deux catégories principales de méthodes SLAM : directe et indirecte. Les méthodes directes utilisent le flux optique de la vidéo pour estimer la pose de la caméra et la reconstruction 3D. Elles produisent des nuages de points 3D précis, mais beaucoup de dérive quand le flux optique entre les images consécutives de la vidéo est élevé. Les méthodes indirectes font correspondre des points 2D sur les images de la vidéo pour estimer la pose de la caméra et effectuer la reconstruction 3D. Ainsi, les méthodes indirectes ne sont pas sensibles au flux optique élevé. Néanmoins, elles sont sensibles aux textures répétitives, qui peuvent faire échouer le SLAM.

MOD SLAM est une nouvelle méthode de SLAM qui combine les méthodes indirectes et directes afin de bénéficier des avantages des deux. Elle s'appuie sur la carte généralisée, une structure de données munie d'un ensemble d'opérateurs. Ces opérateurs assurent la cohérence des informations indirectes et directes, et utilisent des algorithmes efficaces réduisant le coût d'implémentation du SLAM. À chaque étape, MOD SLAM exécute la méthode qu'il estime, produira le moins de dérive. Comme MOD SLAM n'exécute pas les deux méthodes à chaque étape, elle est compétitive en temps de calcul avec l'état de l'art, et peut fonctionner à 20 images par seconde sur les téléphones Android haute gamme.

Ce manuscrit présente en détail les méthodes de SLAM monoculaire de la littérature. Nous développons la carte généralisée et expliquons MOD SLAM en détail. Une analyse fine de l'impact du système de décision de MOD SLAM est présentée, afin de démontrer l'impact de celui-ci. Nous concluons en montrant que nous avons créé une méthode SLAM combinant les principes des algorithmes directs et indirects, réussissant à finir 100% des séquences des bases de données populaires, tout en augmentant la précision des méthodes de pointe.

Mots-clés Traitement d'image et de vidéo ; SLAM ; Cartographie ; Localisation ; 3D ; Réalité augmentée ; Conduite autonome ; ORB-SLAM ; DSO ; MOD SLAM ; Carte généralisée ;

I'm extremely grateful to Eva and Pascal for all your time and patience. I would like to thank my defense committee, who generously provided knowledge and expertise. I had the pleasure of working with Abdou (Abderahmane Bedouhene), who has been my office colleague for 3 years and was always here to help. Many thanks to Azelle (Azelle Courtial, PhD Student at ENSG), Johanna (Johanna Rivas, PhD Student at University of Rennes), and Timothé (Timothé Chauvet, Student at Efrei Paris) for all their support. I'd like to mention Pierre-Alain Langlois, Thomas Daumain, and Eric Llorens. Finally, I'd like to thank all the Imagine Team.

This thesis has been realized in collaboration with the Urban Vision project funded by the National Research Agency via the i-site FUTURE [2]. Thank you to all the Urban Vision Teams.

Contents

Résumé en Français

Un algorithme de cartographie et localisation simultanée (SLAM) a pour but de localiser un agent, tout en faisant une reconstruction 3D de l'environnement de celui-ci (appelée cartographie). Les informations calculées sont stockées dans une structure de données appelée "carte". Ces algorithmes doivent s'exécuter en temps réel. Au sein d'un programme de SLAM visuel, l'agent est une caméra, pouvant être monoculaire, stéréo, RGB-D (image couleur et carte de profondeur) ou un LiDAR (détection et estimation de la distance par la lumière). L'algorithme de SLAM visuel déduit la position de la caméra et calcule la reconstruction 3D en exploitant le flux d'image continu de la caméra [3]. Afin de répondre aux besoins de la réalité augmentée [4] ou des drones autonomes [5], ce manuscrit se concentre sur les méthodes de SLAM monoculaire.

Ces dernières années, la recherche a perfectionné les algorithmes de SLAM monoculaire, réduisant considérablement leur erreur globale. Cependant, la robustesse et la précision de ces méthodes ne sont toujours pas suffisantes pour les besoins cités. Il existe deux types de méthode de SLAM monoculaire. Les SLAM monoculaires directs se basent sur des optimisations photométriques afin d'estimer la pose de la caméra, ainsi que le nuage de points 3D. Les SLAM monoculaires indirects calculent des correspondances de points 2D entre les différentes images de la vidéo. Ces correspondances servent à exécuter des optimisations minimisant une erreur de reprojection, permettant d'estimer les poses et points 3D. Étant donné que les SLAM directs se basent sur des optimisations photométriques, ces méthodes produisent des dérives de la pose lorsque le flux optique entre les images consécutives de la vidéo est élevé. Les SLAM indirects se basant sur des correspondances de points, ces méthodes peuvent échouer lorsque les images manquent de points caractéristiques, ou ont des textures répétitives.

Afin de résoudre le problème de robustesse et d'augmenter la précision, une solution serait de faire une méthode hybride, c'est-à-dire à la fois directe et indirecte, afin qu'elle bénéficie des avantages des deux. Il existe peu de méthodes hybrides dans la littérature. La méthode "Loosely Coupled SLAM" exécute deux méthodes, une directe et une indirecte, l'une à côté de l'autre. Cette méthode souffre des inconvénients des deux méthodes, et met en évidence le problème que les structures de carte existantes ne sont pas adaptées aujourd'hui pour les méthodes hybrides. Les auteurs du papier "A unified formulation for visual odometry" [6] présentent une méthode d'optimisation hybride, et mettent en évidence le problème d'utiliser des points 3D directes et indirectes dans la même méthode.

Pour répondre à ces deux problèmes, nous proposons deux contributions principales. Premièrement, la carte généralisée, une structure de données munie d'un ensemble d'opérateurs. Ces opérateurs assurent la cohérence des informations indirectes et directes, et utilisent des algorithmes efficaces réduisant le coût d'implémentation du SLAM. Deuxièmement, MOD SLAM est une nouvelle méthode de SLAM, s'appuyant sur la carte généralisée, qui combine les méthodes indirectes et directes afin de bénéficier des avantages des deux.

La section suivante énonce les critères technologiques et algorithmiques des algorithmes de SLAM. En partant de ces critères, nous développons l'état de l'art. Nous développons ensuite la carte généralisée, ainsi que la méthode MOD SLAM.

1 Les méthodes de SLAM

Correspondant au Chapitre 2

Cette section a pour but de démontrer les enjeux des méthodes de SLAM. Nous présentons les critères et contraintes matérielles qu'ont ces méthodes de SLAM, pour répondre au besoin des applications d'aujourd'hui. Puis, nous expliquerons les différentes familles de méthodes de la littérature et leur problème.

1.1 Les différentes caméras et capteurs

Les algorithmes de SLAM peuvent utiliser le flux de plusieurs types de caméras en entrée : les caméras monoculaires, stéréo, RGB-D ou des LiDAR. Parmi les quatre types de caméras, les capteurs monoculaires consomment le moins d'énergie, mais fournissent moins d'information pour les algorithmes de SLAM ; les caméras stéréos sont composées de deux capteurs monoculaires, à une distance fixe l'une de l'autre et pointant dans la même direction. Grâce à la deuxième caméra, les algorithmes de SLAM déduisent des informations de profondeur supplémentaire. Cependant, utiliser une caméra stéréo demande deux fois plus d'énergie et deux fois plus de ressources de calcul par rapport à une caméra monoculaire ; les caméras RGB-D envoient deux flux vidéos : celui d'une caméra monoculaire, et sa carte de profondeur associée. Pour cela, une plateforme de calcul déduit une carte de profondeur en utilisant un projecteur et une caméra infrarouge. Les caméras RGB-D consomment plus d'énergie que les caméras stéréo. De plus, les caméras RGB-D sont très sensibles aux conditions de lumière, et ne fonctionnent pas pour des distances de profondeur très élevées ; les LiDAR utilisent un laser pour fournir une information de profondeur très précise. Ce laser va envoyer un faisceau de lumière qui va

être réfléchi sur les objets de la scène et revenir vers le LiDAR. Le temps qui a été mis par le faisceau de lumière pour revenir au LiDAR permet de calculer la distance de l'objet. À cause de leur fonctionnement, les LiDAR consomment énormément d'énergie. Ainsi, les différents types de caméras peuvent améliorer la précision du SLAM au détriment de la consommation d'énergie.

En plus des caméras, les méthodes de SLAM peuvent utiliser des capteurs donnant des informations sur la position dans l'espace de la caméra, tels que les centrales inertielles ou les GPS. Nous avons choisi de travailler les méthodes de SLAM monoculaire, sans aucun autre capteur. Comme l'état de l'art, dont ORB-SLAM et DSO, ont été évalués sans aucun autre capteur, cela nous permettra de nous comparer à ces méthodes. De plus, les algorithmes de SLAM peuvent intégrer aisément les capteurs donnant les informations spatiales.

1.2 Les différentes familles de SLAM

Chaque méthode de SLAM répond à une structure commune. Tous sont composés d'une partie d'estimation de pose, d'une partie de triangulation de points 3D, et d'une partie de raffinement : l'estimation de pose repose sur la mise en correspondance de points 3D de coordonnées connues, avec des points 2D de l'image. Ces correspondances permettent de calculer la pose de l'image ; après chaque estimation de pose, la méthode de SLAM estime si elle a besoin d'une trame clé. Une trame clé sert de référence pour les futures estimations de pose. À chaque fois qu'une trame clé est créé, une triangulation de points 3D et optionnellement un raffinement sont exécutés ; La partie de triangulation de points 3D consiste à estimer la position de nouveau point 3D, à partir de correspondances entre les différentes trames clé ; La partie de raffinement corrige les imperfections d'estimation de pose, et de points 3D.

Ce qui différencie ces méthodes est la manière dont ils exécutent ces trois parties. En effet, certains des algorithmes sont directs, et exécutes une optimisation directement depuis les pixels de l'image, alors que d'autres sont indirects, et vont d'abord mettre en correspondances des points clés. Ces méthodes sont évaluées sur deux jeux de données principales : le jeu de données KITTI est composée de vidéos filmées depuis une voiture en circulation à environ 50 km/h à 10 trames par secondes ; le jeu de données TUM contient des vidéos d'intérieur à 30 trames par secondes.

Les méthodes directes utilisent le flux optique des images consécutives de la vidéo pour garder la trace des pixels. Cette trace permet de calculer la pose de la caméra,

et de faire la reconstruction 3D. Par exemple, DTAM [7] est une méthode optimisant photométriquement tous les pixels des images en utilisant le GPU, et qui ainsi permet une estimation de pose extrêmement précise et une reconstruction 3D extrêmement dense. Cependant, DTAM ne peut fonctionner que pour de très petits environnements (par exemple, dans leur papier, les auteurs limitent la reconstruction à un bureau de travail). LSD-SLAM [8] par Engel et al. est la première méthode de SLAM direct qui est éparse. Contrairement à une méthode dense, une méthode éparse ne fait l'estimation de points 3D que pour un sous ensemble de pixels de l'image, augmentant la vitesse de calcul, et permettant de se passer de GPU. Elle a été publiée en 2014 et fonctionne avec seulement une caméra monoculaire et un processeur peu puissant. DSO [9], publié en 2017, est une version améliorée et plus précise de LSD SLAM. Notamment, ce SLAM est capable d'optimiser dans un raffinement local les paramètres internes à la caméra, en plus de la pose et des points 3D. Sur des images calibrées avec un faible flux optique, tels que les vidéos de TUM, DSO donne d'excellents résultats. Cependant, les méthodes directes donnent de mauvais résultats sur les vidéos avec un grand flux optique. C'est le cas de KITTI, qui filme une voiture roulant à 50km/h, à seulement 10 trames par seconde. **Les méthodes indirectes** tentent de mettre en correspondance des groupes de pixels ayant un modèle spécifique, sur plusieurs images proches dans le temps de la vidéo. Soulignons que la plupart des méthodes de SLAM indirect sont basées sur les techniques de géométrie épipolaire et sur les techniques de "Structure acquise à partir d'un mouvement" par Hartley and Zisserman [10]. Parallel Tracking and Mapping [11] par Klein et al. est une méthode de SLAM publiée en 2007 créée pour reconstruire des pièces de taille limitées, mais fonctionne aussi sur des plus grands environnements. En se servant du parallélisme du processeur, ils arrivent à garder en mémoire une grande densité de point 3D, améliorant en même temps la qualité de l'estimation de pose. ORB-SLAM [12] par Mur-Artal et al. est une version améliorée de PTAM publiée en 2015, couplée avec un système de fermeture de boucle développé originalement par Starsdat et al. ORB-SLAM reste aujourd'hui la méthode monoculaire indirecte rendant les meilleurs résultats. En 2022, une méthode de Ni et al. [13] améliore ORB-SLAM pour fonctionner dans des environnements dynamiques (avec beaucoup d'objets en mouvement). Les méthodes indirectes arrivent à mettre en correspondance des points 2D sur des images distantes, d'où leur robustesse sur des vidéos telles que KITTI. Par contre, ces méthodes sont sensibles aux textures répétitives (arbre, route), ou en manque de texture (provoquées par exemple par du flou). Ainsi, les méthodes indirectes ont beaucoup de mal à traiter jusqu'au bout les vidéos de TUM.

Les méthodes directes échouent sur des vidéos avec des grands mouvements, tels que les vidéos de la base de données KITTI, tandis que les méthodes indirectes échouent

sur les vidéos manquant de texture ou avec des schémas répétitifs (tel que le feuillage des arbres). Ces deux familles de méthodes sont complémentaires. Une idée serait de créer une méthode hybride, bénéficiant à la fois des avantages des méthodes directes et indirectes.

Les structures de cartes existantes ne sont adaptées que pour contenir des informations directes ou indirectes, mais pas les deux en même temps. C'est pourquoi dans ce manuscrit, nous développons une "carte généralisée", c'est-à-dire une base de développement pour les méthodes directes et indirectes. À partir de cette carte généralisée, nous créons une méthode hybride bénéficiant du meilleur des méthodes directes et indirectes.

2 Carte généralisée

Correspondant au Chapitre 3

Dans le domaine du SLAM, *la carte* est une structure de données contenant toutes les informations calculées par un algorithme SLAM. Ces informations contiennent les poses des caméras, les coordonnées 3D des points, les correspondances 2D-3D, etc. L'interface de la carte contient des opérations permettant de lire et d'écrire efficacement les informations qui y sont contenues. Par conséquent, l'implémentation des opérations a un impact énorme sur les besoins en mémoire et le temps d'exécution. Les cartes d'aujourd'hui limitent les informations qu'elles peuvent contenir aux algorithmes soit directs, soit indirects. Ainsi, les implémentations de carte d'aujourd'hui sont inadaptées à la création d'un SLAM hybride. Pour toutes ces raisons, nous introduisons le concept de *carte généralisée*. Une carte généralisée a pour but d'être une base de développement pour les types de SLAM indirects, directs et hybrides. Elle doit respecter trois conditions principales.

(i) La carte généralisée doit être compatible les familles de méthodes directe et indirecte. La carte généralisée est représentée sous la forme d'un graphe. Chaque nœud du graphe représente une trame. Chaque trame est associée à un ou plusieurs points 3D de la carte. Chaque paire de trames étant associée à des points 3D en commun est connectée avec un poids égal au nombre de points communs aux deux nœuds. Chaque trame et chaque point 3D contiennent des coordonnées. Ces coordonnées peuvent être représentées de différentes manières. Pour rendre la carte compatible avec le plus d'algorithmes possible, la carte généralisée convertit à la volée les informations, et met en cache les conversions. Les algorithmes des SLAMS doivent s'exécuter sur des sous-groupes de trames et/ou de points spécifiques. Pour optimiser l'exécution d'algorithmes sur des sous-groupes, nous introduisons le système d'étiquette. Chaque point et chaque nœud

peut être étiqueté une ou plusieurs fois. Par exemple, un point peut être étiqueté en tant que direct ou en tant qu'indirect.

(ii) La carte hybride doit s'assurer de la cohérence des informations entre toutes ces familles d'algorithme. Même si la carte généralisée peut contenir tout type d'information, elle doit maintenair la cohérence entre ces informations. Pour n'importe quelle trame de la carte généralisée. La pose de cette trame peut être estimée en utilisant des points 3D directs, ou des points 3D indirects. Plus ces deux approximations se ressemblent, plus la carte est cohérente. Il peut arriver qu'un algorithme raffine les points d'un certain type (direct ou indirect), en modifiant la pose de caméras, sans prendre en compte les points du type opposé. Dans ce cas, la carte perd sa cohérence, puisque deux estimations de poses à partir des points directs et indirects ne donneront plus le même résultat. Il y a deux cas possibles. Le premier cas est celui d'un algorithme direct modifiant seulement des points directs. Étant donné que les points indirects sont liés par des correspondances 2D-3D, il est très facile de les optimiser à nouveau. Ainsi, lorsqu'un algorithme direct modifie les points directs, les points indirects sont optimisés dans un raffinement rapide de la carte. Le deuxième cas est celui d'un algorithme indirect modifiant seulement les points indirects. Il n'est pas possible de re-optimiser les points directs, au risque qu'ils convergent dans des minima locaux. Avant de re-optimiser les points directs, ceux-ci sont déformés de sorte à rapprocher leur ancienne reprojection de leur nouvelle. Après ceci, ces points ont une chance accrue de re-converger vers un minimum global, c'est pourquoi une optimisation est exécutée.

(iii) La carte généralisée doit être suffisamment efficace. Le graphe est codé sous forme de "graphe de hachage". Les arêtes des trames et des points 3D sont stockées dans des hash-map et des hash-set. Ces structures de hachage sont spécialement optimisées pour la carte généralisée. Dans un SLAM, les actions les plus souvent exécutées sont l'estimation de pose, qui lit principalement la carte. L'écriture de la carte est plus rare. C'est pourquoi, nous développons des algorithmes, optimisant la vitesse de lecture puis la vitesse d'écriture dans la carte, au détriment d'une consommation plus élevée de mémoire. Ainsi, presque tous les algorithmes de lecture ont une complexité constante. Les algorithmes d'écriture, par contre, doivent écrire plus de choses en mémoire, pour permettre aux algorithmes de lecture de retrouver plus rapidement l'information.

Ainsi, la carte généralisée est capable de faire fonctionner différents types d'algorithmes ensemble, tout en s'assurant de la cohérence et de la consistance des informations.

3 MOD SLAM

Correspondant au Chapitre 4

MOD SLAM est une méthode de SLAM hybride basée sur la carte généralisée, combinant ORB-SLAM et DSO. Le but de MOD SLAM est de bénéficier des avantages des méthodes directes et indirectes.

MOD SLAM est composé de trois blocs principaux : l'estimation de pose, la triangulation de nouveaux points 3D et le raffinement de la carte. Pour chacun de ces blocs, MOD SLAM choisi entre plusieurs chemins, exécutant des méthodes différentes. MOD SLAM prédit la méthode qui va fournir le plus de précision et de robustesse. Seulement la méthode évaluée la plus apte à donner les meilleurs résultats est exécutée. Ce système de décision se base sur le comportement des algorithmes d'optimisation sur les trames précédentes.

3.1 Estimation de la pose

Durant l'estimation de pose, MOD SLAM a le choix entre deux chemins : (i) Le premier chemin est composé d'une estimation de pose basée sur une optimisation photométrique. Pour cela, l'algorithme teste une quantité importante de poses basée sur un modèle de vélocité constant. Il optimise toutes ces poses de manière photométrique, et retient la pose qui a obtenu la plus basse erreur photométrique. Il est préférable que MOD SLAM prenne ce chemin lorsque le flux optique entre l'image courante et l'image précédente est petit. Cependant, il peut arriver que le système de prédiction se trompe, et que cette optimisation de pose donne un très mauvais résultat. Dans ce cas, MOD SLAM exécute un raffinement avec les points indirects. (ii) Le deuxième chemin est composé d'une estimation de pose basée sur une minimisation d'erreurs de reprojection suivie d'un raffinement photométrique. L'algorithme met en correspondances des points 2D de la trame courante avec la projection de points 3D indirects récents. L'algorithme optimise la pose de la trame de sorte à minimiser la distance entre la projection des points 3D et la position des points 2D. Cette méthode est très robuste au flux optique élevée mais ne donne pas une pose précise. C'est pourquoi l'estimation de pose indirecte est suivie d'une optimisation directe (qui n'est plus gênée par le flux optique élevé). Il peut arriver dans de rare cas que l'optimisation de pose basée sur l'erreur de reprojection échoue. Dans ce cas, MOD SLAM retourne au premier chemin, pour faire une estimation de pose de manière photométrique.

Pour choisir entre ces deux chemins, MOD SLAM utilise un système de décision basé sur deux critères. Lorsque le nombre de points 3D indirects est trop faible, cela

engendre de l'imprécision voir des pertes de robustesse dans les algorithmes d'estimation de pose indirecte. Dans ce cas, MOD SLAM choisit de prendre le premier chemin direct. Puis, MOD SLAM compare les incertitudes des anciennes optimisations de pose directes et indirectes. Cette incertitude est calculée à partir des Hessiennes des poses lors des optimisations directes et indirectes. MOD SAM choisit d'utiliser la méthode qui a donné l'incertitude la plus faibe.

Une fois la pose estimée, MOD SLAM projette les points 3D indirects récents sur la trame courante pour établir le plus de correspondances 2D-3D possible. Ces correspondances servent à raffiner la pose, et à supprimer les points 3D qui ne sont pas assez robustes.

3.2 Triangulation et raffinement

Dans un SLAM, une trame clé est une trame de référence pour les futures estimations de pose. Lorsqu'une trame clé est créée, de nouveaux points 3D sont estimés, et un raffinement est exécuté. Pour chaque trame, MOD SLAM a la possibilité de créer une nouvelle trame clé directe, et une nouvelle trame clé indirecte. Pour cela, il prend deux décisions, inspirées de celles de ORB-SLAM et de DSO. MOD SLAM crée une trame clé directe si le flux optique entre la dernière trame clé est élevée. MOD SLAM crée une trame clé indirecte lorsque le nombre de points 3D indirects qui a été mis en correspondances avec des points 2D sur la dernière image est inférieur à 95% (tel que défini dans le papier ORB-SLAM [14]) par rapport au nombre de correspondances de la dernière trame clé.

À la création d'une trame clé directe ou indirecte, de nouveaux points 3D sont créés. Afin d'initialiser les points 3D directs, MOD SLAM suit des points 2D sur plusieurs trames. Cela permet de réduire un intervalle correspondant à la profondeur du point 3D par rapport à sa trame de référence. MOD SLAM crée les points indirects en mettant en correspondances des points 2D sur des trames distantes.

Lorsqu'une trame clé directe est créée, MOD SLAM peut exécuter un raffinement direct. De même pour une trame clé indirecte. Cependant, pour une même itération, MOD SLAM ne peut pas exécuter un raffinement à la fois direct et indirect. Le raffinement de la carte ne peut s'exécuter que pour les points directs, indirects, aucun, mais jamais les deux en même temps. C'est pourquoi MOD SLAM prend une autre décision permettant de choisir lequel des raffinements il va exécuter. Cette décision est basée sur le nombre de points directs et indirects. Un nombre important de points signifie plus de données

pour l'algorithme d'optimisation et donc plus de précision. La décision est aussi basée sur la proportion de valeurs aberrantes à la fois des points directs et indirects.

4 Résultats de MOD SLAM

Correspondant au Chapitre 5

Les paramètres de MOD SLAM étant interdépendants, nous utilisons un algorithme de recherche pour régler automatiquement les paramètres de MOD SLAM. Pour chaque paramètre de MOD SLAM, l'algorithme teste des valeurs possibles pour ce paramètre, en exécutant MOD SLAM cinq fois avec du bruit. Il compare les résultats avec une vérité terrain, et retient la valeur du paramètre ayant donné le meilleur résultat. Deux versions de cet algorithme existent : la version rapide raffine un paramètre à chaque itération et envoie les modifications au prochain paramètre ; la version lente modifie les paramètres d'un coup après avoir testé l'ensemble des paramètres.

Nous avons évalué et comparé MOD SLAM sur trois ensembles de données : KITTI, TUM, et notre propre base de données basée sur les vidéos d'un smartphone. Sur la base de données KITTI, MOD SLAM arrive à traiter tous les vidéos dans leur intégralité, contrairement aux méthodes de l'état de l'art qui échouent sur la seconde vidéo de KITTI. MOD SLAM obtient une erreur moyenne de 27 mètres contre 34 mètres pour ORB-SLAM, et 55 mètres pour DSO. Cela montre que MOD SLAM est plus robuste et plus précis que ORB-SLAM et DSO. MOD SLAM donne les mêmes résultats que DSO sur les vidéos TUM. ORB-SLAM échoue sur 50% des vidéos TUM. Enfin, MOD SLAM donne les meilleurs résultats sur nos propres vidéos, sur lequel ORB-SLAM échoue.

MOD SLAM est capable de tourner en temps réel sur un téléphone Android haut de gamme à 20 trames par seconde, prouvant ainsi sa capacité temps réel. La carte généralisée faisant office de base de code pour les SLAM, mais aussi pour les applications, nous avons pu développer rapidement une application de réalité augmentée, Chapitre 6.3, page 118.

Les Figures 6.4, 6.5, et 6.6 de la page 122 montrent l'impact du système de décision sur le résultat de MOD SLAM sur certaines vidéos KITTI. Plus précisément, les paramètres du système de décision sont testés, afin d'évaluer leur importance. Les figures montrent clairement une baisse de l'erreur lorsque la recherche automatique des paramètres est activée.

5 Conclusion

Correspondant au Chapitre 6

Les applications telles que les drones autonomes ou la réalité augmentée ont besoin de méthode de SLAM monoculaire robuste et précise. Cependant, les méthodes courantes ne sont pas suffisamment robustes et précises. En effet, les SLAM monoculaires directs sont sensibles au flux optique élevé, tandis que les SLAM monoculaires indirects sont sensibles à l'absence ou l'ambiguïté des textures sur les images. Pour pallier ce problème, une solution serait de créer un SLAM combinant les deux types de méthodes. Néanmoins, les implémentations de carte actuelle ne permettent pas de contenir à la fois les informations directes et indirectes. Ainsi, la création de MOD SLAM s'est décomposée en 3 étapes : la création d'une carte généralisée, la création de la méthode MOD SLAM, et l'évaluation.

Pour rappel, dans un programme de SLAM, la carte est un graphe contenant toutes les informations calculées par le SLAM (pose de caméra, nuage de points, correspondances 2D-3D, ...). Une carte généralisée doit répondre à trois critères principaux. Premièrement, elle doit pouvoir contenir les informations des méthodes directes et indirectes. Pour cela, nous avons défini un formalisme unifié permettant d'implémenter une carte qui puisse contenir les informations directes et indirectes. Deuxièmement, elle doit assurer la cohérence des informations. En effet, un algorithme pourrait ne modifier que les points directs, sans prendre en compte les points indirects. Dans ce cas, un algorithme déforme automatiquement la carte, pour la rendre plus cohérente. Troisièmement, la carte généralisée ne doit pas ralentir le SLAM. Pour cela, nous avons travaillé sur des algorithmes efficaces, réduisant au maximum le temps d'exécution des fonctions de la carte les plus utilisées.

MOD SLAM a été construit en se basant sur la carte généralisée. C'est un SLAM, mélangeant les techniques ORB-SLAM et DSO. MOD SLAM est décomposé en trois blocs principaux : l'estimation de pose, la triangulation et le raffinement. Pour chacune de ces trois étapes, deux chemins sont possibles, indirects ou directs. MOD SLAM utilise un système de décision permettant de prédire la méthode qui donnera le plus de robustesse et de précision, afin de choisir le chemin à prendre en avance. De ce fait, MOD SLAM parvient à augmenter la robustesse et la précision sur les bases de données populaires. De plus, grâce au système de décision, MOD SLAM n'est pas ralenti par rapport aux méthodes de l'état de l'art. Cela lui permet de s'exécuter à 20 trames par seconde sur un téléphone Android haut de gamme.

Le code source de la carte généralisée ainsi que de MOD SLAM est disponible sur GitHub à l'adresse `http://github.com/belosthomas/libCML`.

1 Introduction

1.1 Motivation and Problem Statement

In the last 15 years, Simultaneous Localization and Mapping (SLAM) methods have evolved to enable the creation of autonomous vehicle [15]. SLAM algorithms provide the vehicle with a real-time estimation of its pose simultaneously with a 3D reconstruction of its environment (commonly called “mapping”). They populate a “map” with the resulting poses and 3D points. They base their estimate on a series of observations by different sensors embedded in the vehicle and from the previous estimates of the map (Figure 1.1). For example, in Visual SLAM algorithms, used for autonomous cars [5], the estimation is based on the continuous image flow of a camera, either monocular, stereo, or a LiDAR.

Among the visual SLAM algorithms, the monocular methods are used in applications with limited energy consumption, such as autonomous drones [16], and augmented reality devices [4]. The particularity of monocular SLAM is its complexity, as no sensors

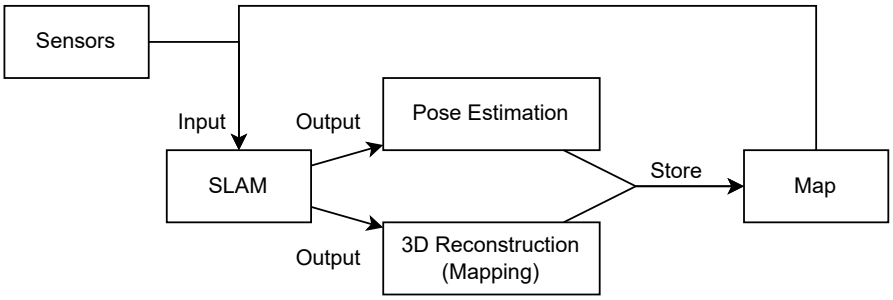


Figure 1.1: Simultaneous Localization and Mapping. SLAM algorithms continually build and refine a map with real-time pose estimation and 3D reconstruction. The algorithms compute the pose estimation and mapping by adopting a "recursive" approach, leveraging the information contained within the map in conjunction with sensor data.

provide 3D information about the environment. The pose estimation relies on the 3D reconstruction, and the 3D reconstruction depends on the pose estimation.

Boosted by robotics applications, monocular visual SLAM methods have significantly improved their robustness and accuracy. However, this progress is still not sufficient for the cited applications. The accumulated drift among the different processes of the algorithms is still too high. Additionally, in particular cases, the SLAM program "lose the tracking", meaning it cannot provide any result.

In recent research, two prominent families of monocular SLAM methods emerged [17]. Direct methods estimate the camera motion and 3D reconstruction by photometrically optimizing errors using the pixel intensities on the image, such as an optical flow. DTAM [7] for Dense Tracking and Mapping is a dense direct method, as it photometrically optimizes the coordinates of each image's pixel. On the contrary, LSD-SLAM [8] for Large-Scale Direct Monocular SLAM is a sparse direct method, as it photometrically optimizes on a subset of pixels of the images. Recent studies [18] have shown that Direct Sparse Odometry [9] (and its loop closure variant LDSO [19]) is the direct method producing the lowest drift. Indirect methods make correspondences between 2D features of consecutive images. They run a re-projection minimization of the camera motion and triangulate 3D points. For example, Parallel Tracking and Mapping for Small Augmented Reality Workspaces [11] has inspired numerous indirect SLAM methods. Recent studies [18] have shown that ORB-SLAM [12] produces the most robust and accurate indirect method on popular datasets.

Direct and indirect methods have been evaluated on two main datasets: KITTI and TUM. The KITTI [20] dataset is composed of sequences filmed inside a moving car at ten frames per second. It allows testing SLAM robustness for the purpose of autonomous car applications. The optical flow between the images of the KITTI video is high. ORB-SLAM produces an average drift of 35 meters on the KITTI dataset and loses the tracking during one video. Because DSO is sensible to drift, it produces an average of 55 meters of drift on the KITTI dataset and loses the tracking in the same video as ORB-SLAM. The TUM [21] dataset is composed of sequences at 30 frames per second filmed indoors by hand. It allows testing SLAM robustness for augmented reality or autonomous drones. The optical flow of the TUM sequences is low. Some part of the videos contains few features and repetitive patterns. Due to the lack of features, ORB-SLAM loses the tracking on half of the TUM sequences. DSO produces an average drift of 3 meters on TUM.

Direct and indirect methods are complementary. While DSO works well on the TUM datasets, it produces a high amount of drift on the KITTI datasets. Conversely, ORB-SLAM produces a lower drift on the KITTI dataset but has a lot of failures on the TUM dataset. A SLAM using both direct and indirect principles (that we denote as a "hybrid SLAM") is prone to increase the robustness and lower the drift. Nonetheless, very few hybrid methods exist in the literature:

- "Loosely Coupled SLAM" [22] merges ORB-SLAM and DSO by running them side by side. This method inherits both the disadvantages of ORB-SLAM and DSO and simultaneously needs the computation resources (in terms of CPU and memory) of ORB and DSO.
- "A unified formulation for visual odometry" [6] merges photometric and feature-based optimization using a multi-objective optimization called Weighted Sum Scalarization [23]. On the TUM dataset, this method results in good accuracy. We could not test this method on the KITTI dataset, as no code is available.

The two presented methods highlight three main bottlenecks.

Bottleneck 1 : The implementation of the map

"Loosely Coupled SLAM" method [22] highlights that the current map implementation does not allow to mix indirect and direct data. Current map implementation does not allow storing data of different SLAM families. They are not optimized to store and retrieve such a high quantity of information in enough low time to allow real-time running. Moreover, by using the same map for direct and indirect algorithms, the direct and indirect data in the map need to accord themselves (we denote the "coherence of the information").

Bottleneck 2 : Mixing photometric and re-projection optimization

"A unified formulation for visual odometry" method [6] highlights the problem of mixing photometric and re-projection errors in the same optimization processes. The former is measured as an aggregated color difference, whereas the latter is a normalized distance in 3D space. Both need to be correctly weighted for each frame of the video.

1.2 Thesis Contributions

Solution 1 : The Implementation of the Map

The first issue is the current SLAM implementation does not allow to mix indirect and direct data. We developed a "generalized map" that mixes direct and indirect information. The generalized map ensures consistency and coherence of information between direct and indirect algorithms. Here is a list of our contributions:

1.a We propose a unified formalism for the map of direct and indirect families.

1.b We propose a generalized map that ensures the coherence and consistency of the information.

1.c We propose fast algorithms for the generalized map and demonstrate their efficiency.

Solution 2 : Mixing Direct and Indirect Algorithms

To solve the problem of creating a method mixing direct and indirect algorithms, we developed MOD SLAM [1]. MOD SLAM is a mixed method that aims to produce the minimal drift of ORB-SLAM and DSO. MOD SLAM uses a prediction system to decide which direct or indirect method will produce minimal drift.

2.a We define a novel SLAM architecture that is ORB-SLAM-based [14] and DSO-based based on the generalized map.

2.b We formulate and evaluate a decision method, allowing MOD SLAM to predict which ORB-SLAM or DSO will be more robust at each step of the SLAM.

Evaluation and Study of the Solutions

We made an automatic parameter-tuning algorithm for MOD SLAM, particularly for the parameters of the decision system. We evaluated MOD SLAM on different datasets and plotted the impact of numerous parameters of the decision system of MOD SLAM to prove their impact.

3.a We propose an algorithm for SLAM parameter optimization and automatic plotting.

3.b We show that by mixing ORB-SLAM and DSO, MOD SLAM can handle 100 percent of the videos on the KITTI dataset, while decreasing the drift. We show that MOD SLAM can run in real-time on an Android phone.

3.c With these algorithms, we studied the decision parameter of MOD SLAM.

To conclude, this thesis issued three main contributions: the development of a generalized map, the creation of a mixed SLAM method named MOD SLAM, and a study of MOD SLAM.

1.3 Manuscript Structure

After the state-of-the-art presentation in Chapter 2, these three contributions are developed in Chapters 3, 4, and 5.

Chapter 2 - State-of-the-art Presentation

The second chapter presents the SLAM methods technological criteria (computing platform camera sensors, spatial sensors) and the SLAM methods algorithms criteria (the common pipeline of each family of methods and the differences between direct and indirect methods). We classify state-of-the-art algorithms according to the technological and algorithms criteria. We explain their issues and the challenges of improving these methods for these state-of-the-art.

Chapter 3 - Complete Explanation of ORB-SLAM and DSO

Our solutions are based on the indirect ORB-SLAM [12] method and the direct DSO [9] method. The third chapter is a complete explanation of these two state-of-the-art methods.

Chapter 4 - Generalized Map (Solution 1)

The fourth chapter presents our generalized map contribution. This generalized map solves the technical problem described above, permitting a mixed-method creation in chapter 4. The generalized map is a SLAM framework that ensures the communication and consistency of information between direct and indirect methods. It permits mixing two methods of different families with minimal modification to the algorithms of the methods.

Chapter 5 - Mixed ORB and DSO SLAM (Solution 2)

The fifth chapter presents our main contribution: a hybrid SLAM method, mixing ORB-SLAM and DSO. We present a novel decision system, allowing the execution of either ORB-SLAM algorithms or DSO at each iteration. Consequently, the computational resources used by our mixed method do not increase compared to ORB-SLAM and DSO. Our result shows that MOD SLAM is significantly more robust than ORB-SLAM and DSO.

Chapter 6 - MOD SLAM Results and Parameters Study

During the sixth chapter, we introduce an automatic parameter tuning algorithm for MOD SLAM. The final result shows that the tuned version of MOD SLAM significantly increases the accuracy. We show that MOD SLAM can run in real-time on Android, and we show an augmented reality application of MOD SLAM. Finally, using the parameters tuning algorithm, we plotted the impact of different parameters of the MOD SLAM decision system to prove its efficiency.

1.4 Conclusion

Monocular SLAM is the problem of constructing a map containing the localization of an agent and a 3D reconstruction of the surrounding of the agent. The estimations have to be computed in real-time based on the image flow of a monocular camera embedded in the agent. Monocular SLAM algorithms pave the way in augmented reality and autonomous drone applications. However, current methods are not robust enough and accurate for these applications. Two main families of monocular SLAM exist: direct methods are based on optical flow principles, while indirect methods are based on feature correspondences. DSO, the state-of-the-art of direct method, produces high drift when the optical flow between the video images is high. ORB-SLAM produces drift and loses tracking when the image lacks texture or contains repetitive patterns. The complementarity of these methods would allow the creation of a hybrid method, which would benefit from the advantages of ORB-SLAM and DSO. Few hybrid methods have been created. “Loosely Coupled SLAM” method [22] runs ORB-SLAM and DSO side by side, and highlights the problem of mixing direct and indirect information in the same map. While “A unified formulation for visual odometry” [6] method use optimization techniques mixing direct and indirect points and highlights the problem of mixing photometric and feature-based optimization processes in the same pipeline. To solve these problems, we created two

solutions: a generalized map, and a mixed SLAM method. This manuscript presents our generalized map contribution, allowing for mixing direct and indirect methods in the same map, and MOD SLAM, a mixed SLAM method using a decision system predicting at each step the best way to use. In the last chapter, we demonstrate the efficiency of our solutions.

2

Visual SLAM Overview

Simultaneous Localization and Mapping programs aim to continuously estimate the 3D poses of a mobile agent within a scene while constructing a 3D map of its environment (Figure 2.1). In a Visual Simultaneous Localization and Mapping algorithm, the agent is the camera, which continuously feeds the algorithms with images. During the previous Chapter 1, we introduced that the main difficulty of SLAM is that the 3D pose estimation and the 3D reconstruction algorithms are dependent: pose estimation algorithms need precise reconstructions, and mapping algorithms need accurate pose estimations. Moreover, the tight dependency between both may increase uncertainty and inaccuracy in the pipeline. The SLAM problem is even more difficult, as augmented reality and autonomous drone applications require real-time estimation while restricting computing resources.

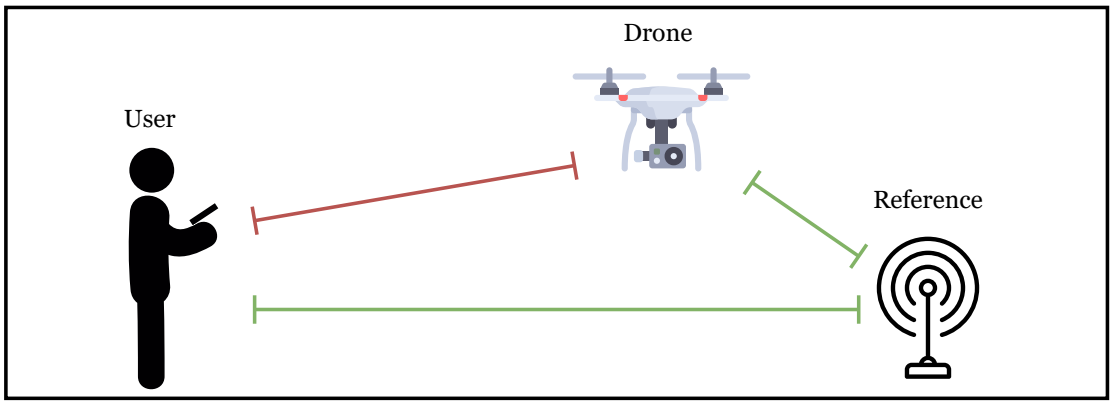


Figure 2.1: The problem of autonomous drone development at Parrot. The drone has to calculate its pose relative to a user for automatic landing. To achieve this goal, the integrated algorithm uses the images of an embedded camera, an inertial measurement unit, and the knowledge of the absolute distance between the drone and a reference, and between the user and the reference.

This chapter aims to better identify the difficulties and challenges of SLAM by providing an overview of the different methods. This chapter classifies SLAM algorithms and explains the functioning of the two main families of SLAM methods. It situates our contributions among the existing methods.

1. To begin, we explain the benefits and drawbacks of using different image sensors, spatial sensors, and computational platforms. We show the different technological constraints of the different applications of the SLAM. By enumerating the constraints of current applications, we conclude the importance and choice of the contribution of embedded monocular-only SLAM.
2. We present two popular datasets and our own dataset that we use to evaluate and compare monocular SLAM algorithms.
3. Then, we present the different families of SLAM. We show the similarity and the difference between the methods of these families.
4. We use the presented technological constraints and SLAM families to classify SLAM algorithms. We chose these SLAM algorithms based on their impact and influence in the field. We describe the improvement of these methods over time.
5. We explain the benefits and drawbacks of these SLAM families and demonstrate that they are complementary. We enumerate the current challenges and situate where our contributions lie in current research.

To be able to classify SLAM algorithms, we begin by showing the different technology that these SLAM can use.

2.1 Technological Criteria for SLAM Classification

This section discusses the need for the SLAM application to be embeddable while maintaining enough precision and robustness. For that, we describe the different technological constraints emerging from these applications.

Among these applications: autonomous robot navigation [24], augmented reality [4], and urban analysis [25]. They require SLAM programs to have a certain amount of precision with limited resources [26]. For example, autonomous drones and augmented reality devices have a limitation in energy consumption [27], which means that they also have limitations in the technologies on which to operate. In contrast, urban analysis applications, or autonomous robots in the industry, can use larger computing structures [28, 29].

SLAM programs use the following technologies: the platform on which they are running (including the processor and memory), image sensors that provide a continuous image stream, and additional sensors that provide spatial information. Each of these technologies influences the accuracy and energy consumption of the SLAM. The computing platform limits the computing resources of the SLAM, whereas the image sensors and spatial sensors can provide more information but can also bring noises to the program [30].

The applications limit the technologies that SLAM programs can use. Next, we develop all these technologies, starting with the vision sensors that the SLAM device can embed. Then, we present common spatial sensors SLAM devices can embed upon. Finally, we present each target of the computational platform associated with its applications, from powerful servers to minimal platforms.

2.1.1 The different cameras

The type of image sensor used by a visual SLAM method influences the quality of the result [31], the speed of the program, and the energy consumption [32] of the device running the SLAM. Among these sensors:

- Monocular Sensors[1] provide one image that contains information about the light intensity. However, we will show in Section 3.2 that this light intensity does not correspond to the real-life light intensity because the camera can only capture a limited range of light intensity [33].
- Stereo Cameras are composed of two monocular sensors [34], which allow algorithms to deduce the depth of each object in the image [35].
- RGB-D Cameras use the help of a stereo camera to provide images along with matrices containing information about depth [36] (the distance of the object appearing in the image relative to the camera).
- LiDAR for "light detection and ranging" use a laser to determine the depth of pixels and output matrices containing depth information [37].

For each of these sensors, we will show the advantages and drawbacks of the quality of the SLAM, the speed, and the energy consumption of the device running the SLAM.

[1] We consider all sensors of the monocular camera array as one unique sensor.

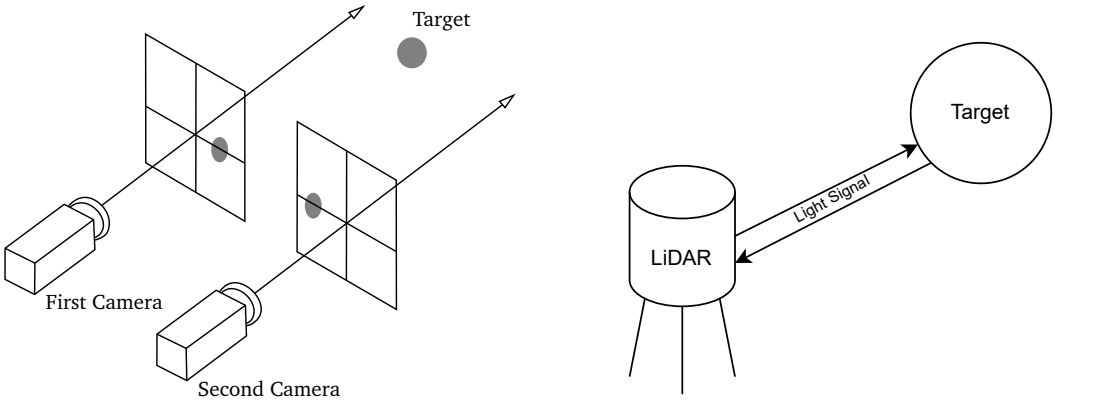
Monocular Sensors

Compared to the four types of sensors cited, monocular sensors provide only one video stream for SLAMs programs. This makes the SLAM methods less accurate and less robust [18]. As the algorithms have only one image to process, the SLAM programs are faster (for example, feature extraction is a slow process. It would be twice as slow if it had two images to process). As monocular SLAM pipelines use lighter algorithms and fewer devices on their board, they are less energy-consuming. However, not all monocular cameras are equivalent. There are two types of monocular cameras. A *rolling shutter* camera scans the sensor measurement line by line [38]. Because there is a delay between each consecutive scanline read, each line of pixels of the image has a slightly different capture time, during which the platform may have moved. This induces the rolling shutter effect, as shown in the figure 2.2. A *global shutter* image sensor uses an electronic shutter to read all scanlines instantly [39]. This fixes the rolling shutter camera problem of image lines that have different times. Today's smartphones and cameras use rolling shutter sensors due to their lower cost [40].

(a) This image from Adobe shows the difference between a global shutter camera (left) and a rolling shutter camera (right). The two camera capture a running plane propeller.

(b) Photo captured inside a moving train in movement with a smartphone. The background is far from the camera, so it has a low apparent motion. The sign is close to the camera, so it has a high apparent motion.

Figure 2.2: The rolling shutter effect. The rolling shutter camera scans the sensor data from the camera line by line, from top to bottom, resulting in the deformation of objects with high apparent motion (the propeller appears deformed and the sign bar appears tilted).



(a) Model of parallel stereo camera. This figure has been inspired by [41].

(b) Concept of LiDAR. LiDAR devices use a laser to estimate the depth of target objects.

Figure 2.3: Stereo Camera versus versus LiDAR.

Stereo Cameras

Stereo cameras are made up of two monocular sensors. As shown in Figure 2.3a, the two monocular sensors can be placed in parallel or oriented as long as they point to a common target [34]. With two cameras side-by-side, SLAM programs can deduce the depth of characteristic points in the images [35]. This reduces pose estimation and mapping dependencies and makes the SLAM more accurate and robust. Moreover, stereo SLAM pipelines need to process two times more data than monocular SLAM pipelines and need to run depth-map estimation algorithms, which induce costs, increasing the need for computational resources and energy.

RGB-D Cameras

RGB-D Cameras, such as Intel RealSense camera [42], use a computing platform, an infrared projector, and an infrared monocular sensor to estimate the depth of the environment [43]. The computing platform sends the image from the RGB monocular sensor and the depth map to the SLAM device [36]. A SLAM method using RGB-D sensors has the advantage of being faster than a SLAM based on stereo cameras. However, the device that runs the SLAM will have a higher energy consumption than a device that runs with stereo cameras. Note that the depth information provided by the RGB-D sensors can be inaccurate [44] depending on the light condition.

LiDAR

LiDAR, for "light detection and ranging", uses a laser to determine its distance to its surrounding objects and to compute a depth map. [45] (Figure 2.3b). There are two kinds of LiDAR: rotating LiDAR results in an accurate 360-depth map. Solid-state LiDARs are fixed and, therefore, equivalent to RGB-D cameras. LiDAR has better accuracy than RGB-D sensors and is not influenced by light conditions, but results in a lower-resolution image when the capture is given limited time [37]. They are much heavier, costly, and consume much more energy than a standard RGB-D camera.

We showed that SLAM algorithms work with monocular, stereo, RGB-D, and LiDAR. Monocular sensors are inexpensive and suitable for embedded devices. Stereo, RGB-D, and LiDAR make SLAM more accurate but are heavy, expensive, and energy-wasting. We focus our work on monocular SLAM, the most challenging trending setup that current research has struggled to improve. We also want our SLAM to work on embedded devices with restricted energy consumption and computation resources, such as cars or drones.

The following section talks about supplementary sensors to the cameras that a SLAM can benefit from: the spatial sensors.

2.1.2 Supplementary Spatial Sensors

The data provided by the image sensor can lead to gauge freedom in the optimization processes of the SLAM [46]. This gauge freedom can be reduced or neutralized by spatial sensors such as GPS and IMU, which provide spatial information about the coordinates of the device and the orientation of the device.

Inertial Measurement Unit

An Inertial Measurement Unit (a.k.a. IMU) provides the orientation of a device. It is made up of a gyroscope, an accelerometer, and optionally a magnetometer [47]. The gyroscope measures the orientation of the device with latency, whereas the accelerometer is sensitive to in the acceleration of the orientation, with very low latency [47]. A magnetometer measures the orientation to the north with latency. With a sensor fusion algorithm [48], an IMU is able to combine the information of these three sensors to provide a precise

orientation with low drift and a correct latency [49]. SLAM using IMU needs to use the Kalman filter, which is either expensive or innacurate [50][2].

Most Visual SLAM algorithms generally develop core methods without IMU, as it is always possible to add them later [51] (for example, ORB-SLAM2 [14] to ORB-SLAM3 [52], DSO [9] to VI-DSO [53]).

GPS

GPS for "Global Positioning System", or more generally, any GNS for "Global Navigation Satellite System", provides absolute translation information. Public GPS has a precision of 5 to 15 meters but can be reduced to 1 meter by combining multiple distant GPS [54]. Although this precision is not sufficient for indoor SLAM (the precision is even lower indoors because of the loss of signal), the GPS information outside, such as inside a car, could improve accuracy [55], at the cost of increased energy consumption [56].

We choose not to use any complementary spatial sensor in our contributions because IMU would need too many computational resources and can be added later, and GPS does not give enough benefits while consuming too much energy.

We presented the different sensors that can be input into SLAM programs. We saw that the more sensors with higher accuracy a SLAM used, the more the SLAM needs computational resources and energy. What limits the computational resources is the computational platform.

2.1.3 Computational Platforms

SLAM applications run on various computational platforms, from small devices as portable platforms to high-performance computing platforms. Each has processing, energy, sensors, cost limitations, and, above all, real-time constraints. Most applications, such as augmented reality, need to process the data online, which means that the applications process the results quick enough. In the following, we present a list of the main platforms and their associated applications.

[2] because the Hessian that is unnecessary without IMU needs to be computed for the Kalman filter [50].

Handheld platforms

Handheld platforms encompass smartphones [57] and augmented reality glasses [58]. According to the website gsmarena.com referencing all the smartphones, they are composed on average of quad-core ARM CPU clocked at 1,5 GHz and of 8 GB of RAM. On these devices, SLAM can serve to guide the user or autonomous steering, or to display 3D objects on an image.

Mobile platforms

Mobile platforms such as autonomous cars, robots inside factories [59], or drones [5] are most often composed of an ARM or Intel Atom CPU, with a powerful APU[3], such as the Jetson by NVIDIA. SLAM on mobile platforms can serve Advanced Driver Assistance Systems. The energy consumption of these devices is equivalent to the consumption of a laptop computer [60, 27].

High-performance computing platforms

High-performance computing platforms are big infrastructures composed of powerful computers where SLAM can run in the cloud [61]. This is useful for urban analysis [25] or multi-drone reconstruction [62], and military task tutoring [61].

Our goal is to develop SLAM for portable platforms to desktop computers: our algorithm must work on smartphones and autonomous drones. For the simplicity of development, we did our test on desktop computers. Consequently, we discard the possibility of GPU usage.

Monocular videos are needed to evaluate the robustness and accuracy of SLAM methods. The next section introduces two popular monocular datasets and our own dataset used to evaluate monocular SLAM algorithms. Then, we classify each monocular SLAM method and evaluate them on these datasets.

[3] APU for Accelerated Processing Unit are specialized unit integrated inside CPU, mostly behaving as small GPU, such as Intel HD Graphics.

2.2 The Datasets

2.2.1 KITTI Odometry Dataset

The KITTI Odometry Dataset [20] is composed of 11 videos at 10 frames per second. These videos have been captured inside a moving car with a global shutter camera. Each video contains between 1000 and 4000 frames. The cars move at a speed of around 50km/h. The videos include many ambiguous textures, moving objects, and acceleration, which can make most methods struggle.

Figure 2.4: **The KITTI Dataset.**

2.2.2 TUM Dataset

The difficulties of the TUM dataset are the fast and unpredictable movement and the varying light condition. This is what makes ORB-SLAM lose track and fail in 50% of the videos of the dataset. On the contrary, the direct method has no problem handling the TUM video. The low optical flow between the frames helps any direct method estimate the pose accurately. By handling the light condition, the state-of-the-art direct method has lowered the drift to less than 3 meters.

Figure 2.5: **The TUM Datasets.**

2.2.3 The IMAGINE P3 Dataset

Using a Google Pixel 3a camera, we have filmed perfect loops inside a room at a resolution of 1920x1080, downsampled to 640x360. Some images of these videos are shown in Figure 2.6. The camera is a low-quality rolling shutter camera, giving a lot of compression artifacts. The videos include very fast rotation while walking around a meeting table.

Figure 2.6: The IMAGINE P3 Dataset.

2.2.4 Stereopolis

Stereopolis is a dataset made by Urban Vision, a project funded by the National Research Agency via the i-site FUTURE [2]. It is a dataset of videos filmed in a car driving around the University Gustave Eiffel, embedding five calibrated global shutter cameras at different angles, a LiDAR, and a GPS. The ten frames per second video streams are composed of 2058x2458 colored images with 10 bits per component. These sequences are associated with LiDAR raw data and a trajectory ground truth from the GPS.

Figure 2.7: The Stereopolis Dataset.

In the following sections, we classify SLAM methods of the literature and evaluate them using these four datasets.

2.3 Classification of SLAM algorithms

SLAM methods in the literature follow a *general pipeline* [6], composed of algorithms with the same objective: tracking, pose estimation, mapping, etc. We name a set of algorithms with the same objective a "*block* of a SLAM pipeline". SLAM methods use two different types of algorithms inside the blocks: the photometric algorithms and the feature-based algorithm. According to the type of algorithms a SLAM uses inside each block, it is classified into one of the three SLAM families: direct, semi-direct, or indirect.

This section aims to classify SLAM algorithms of the literature according to their families. We start by presenting the SLAM's general pipeline by describing the role of each block. Then, we describe the differences between these algorithms. Finally, we classify numerous SLAM methods.

2.3.1 SLAM General Pipeline

Figure 2.8 representing each main block of the general pipeline shows that SLAM programs alternate between estimating the pose, mapping new 3D points, and refining the map's data. In the following, we explain each block of the general pipeline.

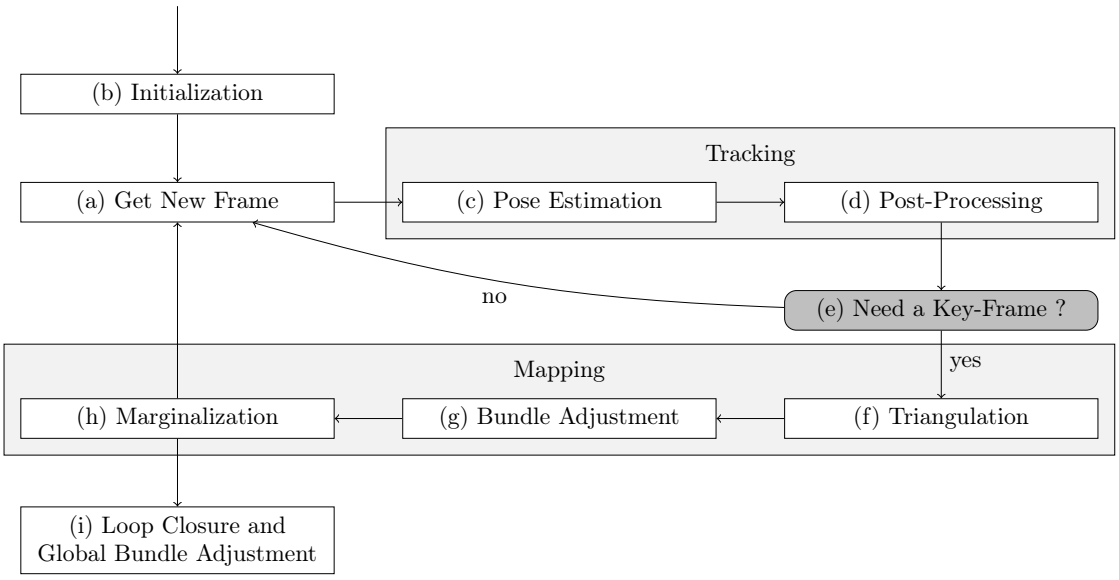


Figure 2.8: **General SLAM Pipeline.**

(a) Get New Frame

The pipeline loop starts by parsing an image sequence. These images are optionally pre-processed: by undistorting the image, scaling the image, and extracting 2D features in the image.

(b) Initialization

Pose estimation block of the SLAM pipelines needs a 3D reconstruction to estimate poses. SLAM pipelines mapping block needs a pose estimation to make a 3D reconstruction. That is why an initialization is required to bootstrap the pose estimation/3D reconstruction loop. The initialization aims to estimate the first video frames' pose and a 3D map-points cloud. The initialization algorithms run an optimization of the pose of the video's first frames and of the 3D map-points cloud. Since it is a demanding process, the initialization is either very slow or inaccurate. However, the accuracy is regained later inside the bundle adjustment block.

(c) Pose Estimation

The pose estimation block is executed for every frame of the video. The pose is estimated with the association of known 3D point coordinates with 2D features in the image.

(d) Post-Processing

After the pose estimation, most SLAM methods perform some post-processing, which refines the pose and the 2D-3D correspondences.

(e) Key-frame Decision

For the proper functioning of pose estimation, the SLAM needs a minimum of 3D points to associate with 2D features. Therefore, the algorithm needs to triangulate new 3D map-points regularly. The SLAM runs the triangulation algorithms of new 3D map-points when it estimates a frame needs to be tagged as a key-frame. At each iteration, based on a key-frame decision system, the algorithm chooses if it creates a new key-frame. A *key-frame* is a reference frame for future pose estimation. Key-frame-based SLAM reduces computational resources by reducing the number of times the SLAM executes the mapping and by reducing the number of data the SLAM optimizes during the bundle

adjustment by optimizing a sub-graph containing only the key-frames. SLAM methods generally insert a key-frame when the camera has moved a certain distance since the last key-frame. When the camera is not moving for a long time, only one key-frame is tagged during the period.

(f) Triangulation

The mapping block has the role of estimating new 3D map-points. From 2D-2D matches between the recent key-frames, the mapping algorithm estimates the 3D coordinates of new map-points.

(g) Bundle Adjustment

The bundle adjustment block consists of refining the map (the frame poses, the map-points coordinates, and optionally the camera calibration). SLAM methods can execute a refinement *globally*, for a large part of the key-frames, or *locally*, in a small window of the last key-frame.

(h) Marginalization

The bundle adjustment optimization process has to invert a Hessian matrix with $\eta_m \times 3 + \eta_f \times 7$ rows and columns, with η_m the number of map-points to optimize, and η_f the number of points to optimize. The number of map-points and frames can quickly grow to high values, making the Hessian slow to invert. The marginalization consists of removing some map-points and some frames from the Hessian computation while keeping the problem coherent.

(i) Loop Closure and Global Bundle Adjustment

Some methods have a loop closure process, trying to match points from two distant frames, allowing the SLAM programs to make a global bundle adjustment, reducing overall drift.

The general pipeline is made up of three main blocks: the pose estimation block, the mapping block, and the bundle adjustment block. Each of these blocks can use algorithms of different types. The following section defines these different types of algorithms to then classify SLAM into families according to the algorithms types they use.

2.3.2 Optimization Methods

Inside SLAM pipeline, multiple algorithms optimize different parameters with different criteria. Pose estimation consists of an optimization of the pose based on the map-points. The mapping consists of map-points optimization based on the poses. And bundle adjustment consists of poses and 3D points optimization. There are two main visual SLAM optimization algorithms: *feature-based optimization* and *photometric optimization*.

Feature-Based Optimizations Feature-based optimization methods proceed by extracting salient image feature sets and matching them using feature-specific descriptors. The algorithms use these correspondences to minimize re-projection errors and optimize either frame poses or 3D point coordinates [63]. A Feature-based optimization will converge to the global minimum, whatever at which distance of the global minimum it starts. However, it has a very slow convergence the more it is close to the global minimum [6]. Most indirect SLAM optimization methods are based on techniques detailed in the monograph by Hartley and Zisserman. Hartley and Zisserman have gathered the best work on indirect multiple view geometry [10]. Also, most indirect SLAM methods use existing algorithms in their pipeline. Among the pose estimation algorithms, we can cite the Perspective-n-Point algorithm [64], and more especially, the Efficient PnP (EPnP) algorithm by Lepetit et al. [65], or the Lambda Twist algorithm by Persson et al. [66]. Using the solution to the five-point relative pose problem by Nister et al. [67] indirect SLAM can be initialized efficiently. We can also cite Hartley and Sturm for their work on the triangulation algorithm [68].

Photometric Optimizations Photometric optimization does not depend on the extraction and matching of any feature. Algorithms use pixel intensities directly to optimize frame poses and 3D points on the image in an optical flow manner [21]. Photometric optimization error functions are highly non-convex. During a photometric optimization, if the current states are far from the global minimum, it has a high probability of converging to a local minimum. On the contrary, photometric optimization gives high convergence with high precision when the current states are already close solution [6].

Pose estimation and bundle adjustment algorithms are based on one or both of these methods. A SLAM pipeline is indirect, direct, or semi-direct, depending on the optimization method it uses.

2.4 The different families of SLAM

During the last years, many SLAM methods emerged, each with its optimization methods, and adapted for different devices. This part aims to review the major monocular SLAM algorithms. These algorithms are listed in Table 2.1. Notice that the recent literature already provides detailed state-of-the-art SLAM overviews [69, 17].

The SLAM pipelines we will present have been chosen because of the impact factor on monocular visual SLAM research. For recent research, the impact factor was not well determined. We chose the recent paper by hand, with what we thought were good quality and interesting improvements.

Reference	Name	Year	Type	Monocular	Stereo	RGB-D	GPU	IMU
[63]	MonoSLAM	2007	Indirect	X				
[11]	PTAM	2007	Indirect	X				
[7]	DTAM	2011	Direct Dense	X			X	
[70]	SVO	2014	Semi-Direct	X				
[8]	LSD	2014	Direct Semi-dense	X				
[12]	ORB-SLAM	2015	Indirect	X				
[14]	ORB-SLAM2	2017	Indirect	X	X	X		
[71]	CNN-SLAM	2017	Direct Semi-dense	X			X	
[9]	DSO	2017	Direct Sparse	X				
[19]	LDSO	2018	Direct Sparse	X				
[53]	VI-DSO	2018	Direct Filtering	X				X
[22]	Loosely-coupled SLAM	2018	Semi-Direct	X				
[72]	CNN-SVO	2019	Semi-Direct	X			X	
[52]	ORB-SLAM3	2021	Indirect Filtering	X	X	X		X
[1]	ModSLAM (Ours)	2021	Semi-Direct	X				

Table 2.1: Classification of main SLAMs methods of literature. Table classifying each SLAM according to the above criteria. The methods in blue are indirect. The methods in red are direct. And the methods in pink are semi-direct.

2.4.1 Indirect family of SLAM

SLAM methods of the direct families use only the feature-based optimization described in the previous section. Because they are based on feature matching across images, they have no difficulties to match features in distant images. Because of that, SLAM pipelines composing this family have the ability to close loops [12]. Still, indirect methods require textured images to work correctly and are disturbed by repetitive patterns [11].

MonoSLAM [63] by Davison et al. is a SLAM method proposed in 2007 whose goal is to have "longterm, repeatable localization within restricted volumes". With only one monocular camera (and optionally an IMU), it successfully estimates in real-time the pose of a camera, as long as the camera stays in the same room. For that, it uses a probabilistic 3D map composed of a few map-points (around 100) and their uncertainty. It optimizes the poses and the map-points with an extended Kalman filter.

Parallel Tracking and Mapping [11] by Klein et al. is a SLAM method published in 2007 for "for small AR workspaces", but can succeed in wider environments. It uses two threads, one for the tracking and the pose estimation and one for the mapping and the bundle adjustment. This parallelism increases the density and, therefore, the quality of MonoSLAM mapping.

Scale drift-aware large scale monocular SLAM [73] by Strasdat et al. improved the previous methods by developing optimization methods based on Lie Algebra. Particularly, this work developed the loop closure system by presenting "a new pose-graph optimization technique which allows for the efficient correction of rotation, translation and scale drift". In 2011, the same authors pushed their loop closure system further by applying them to large-scale loop in a paper named: *Double-window optimization for constant time visual SLAM* [74].

ORB-SLAM [12] by Mur-Artal et al. is an improved version of PTAM published in 2015, coupled with the loop system of Starsdat et .al. It has generally improved each pipeline block by making it more robust. We explain this method in detail in the next section. In 2022, Ni et al. have improved ORB-SLAM to work in dynamic environments (with many moving objects) [13].

In the methods above, we saw SLAM pipelines that work by tracking corners. However, SLAM pipelines can work by tracking segments and planes. During the writing of this manuscript in 2022, a new SLAM paper just got out, which seems to be a new state of the art. Structure PLP-SLAM [75] by Shu et al. is a SLAM tracking both corners, segments, and planes.

2.4.2 Direct family of SLAM

SLAM methods of the direct family are exclusively based on photometric optimization algorithms. They are decomposed into two variants. Direct SLAM programs can map in a *dense* way, using all image pixels [7], or in a *sparse* way, using a subset of points of interest in the images [9]. The advantage is that direct SLAM works well in a poorly textured environment and is not disturbed by repetitive patterns. Nevertheless, because they are based on the optical flow method, direct SLAM needs low optical flow, as high pixel movement makes it impossible to optimize.

DTAM [7] is a method optimizing photometrically all the pixels of the image using photometric optimization. To accomplish this at full frame rate, they need to use a highly parallelizable algorithm on GPU. Their models allow for extremely precise pose estimation and tracking in very restricted environments (in their paper, they limit the 3D reconstruction to one work desk).

LSD-SLAM [8] by Engel et al. was the first full direct sparse SLAM in 2014, running with only one monocular SLAM and small CPUs. It uses the same principle as DTAM but in a sparse manner. Thanks to its sparsity, it can run on the CPU only, without GPU. We explain this method below.

DSO [9], published in 2017, is an improved version of LSD-SLAM, which is much more accurate. It is able to "jointly optimize the full likelihood for all involved model parameters, including camera poses, camera intrinsic, and geometry parameters (inverse depth values)" in a window of multiple frames while taking into account the photometric calibration. We explain this method below.

2.4.3 Semi-Direct Family of SLAM

Semi-Direct family of SLAM is composed of SLAM methods that use photometric and feature-based optimization. The pipeline is either split in two [70], or *mixed* [22]. By combining Indirect and Direct SLAM, some work has tried to implement the advantage of one optimization method on another. For example, loop closure on a direct pipeline with a loosely-coupled method [22].

SVO [70] by Forster et al. is a SLAM method published in 2014 based on PTAM, introducing direct algorithms similar to the DTAM [7] in the pipeline. SVO tracking algorithms do not match corners indirectly with descriptors but keep track of the corners directly with an optical flow. This allows SVO to have 2D-2D matching with sub-pixel

accuracy. CNN-SVO [72] inserts an image depth prediction algorithm based on a neural network to SVO in 2019.

Loosely-coupled SLAM [22] is a semi-direct method running ORB-SLAM and DSO in parallel. The marginalized frames of DSO are sent to ORB-SLAM for feature matching and bundle adjustment. This allows DSO to benefit from loop closure. It has been inspired by SVO and LSD-SLAM. LDSO [19] added Loop Closure to DSO, and VI-DSO [53] added inertial unit.

2.5 Issues of Current SLAM Algorithms

During Chapter 1, we have shown that direct and indirect methods have accuracy and robustness problems. Indeed, the SLAM methods presented above have different issues with different types of videos. Some of them fail to continue the tracking in the middle of a video, stopping giving any results. When this problem happens, we say that the SLAM "lose the track". Some of them constantly produce inaccuracy, producing a shift between the estimated pose and the ground-truth. This shift is called the drift.

This section showcases current SLAM methods' issues by showing these SLAM perform on standard datasets.

2.5.1 KITTI Odometry Dataset

As the KITTI videos were captured at high speed with a low number of frames per second inside a car, making the optical flow high between the video frames. Moreover, direct methods struggle to compute high optical flow, inducing a lot of imprecision during the execution of these methods on KITTI videos. On the contrary, the indirect methods do not have any problems with high optical flow video.

But, these videos contain many repetitive patterns on the tree and on the road. As indirect methods are based on feature matching, repetitive patterns make them create tons of outliers, which they sometimes do not detect. These repetitive patterns are even more difficult to detect due to the low resolution of the image of KITTI.

Indirect methods have been proven to work better on the KITTI dataset than the direct method in the literature. The poor result of the direct methods is explained by the optical flow problem, which causes a lot of drift.

We observe a particular case in the video n°01 of the KITTI dataset. To our knowledge, no monocular method that does not use the GPU exists that succeeds in KITTI 01. Methods lost track during different parts of the video. As shown in Figure 2.9, during this video, the DSO [9] pipeline, state-of-the-art of direct methods, fails to estimate the pose while crossing a bridge at high speed. The combination of a huge light change, high speed, and moving objects make direct algorithms incapable of optimizing the pose. Later in this video, the combination of repetitive texture, low texture, and moving object makes ORB-SLAM2 [14], the state-of-the-art of indirect algorithms, unable to estimate the pose.

2.5.2 TUM Dataset

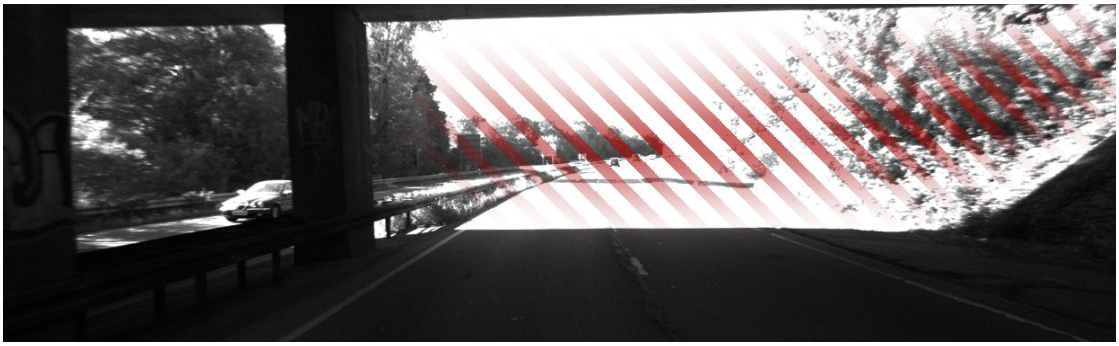
The difficulties of the TUM dataset are the fast and unpredictable movement and the varying light condition. This is what makes ORB-SLAM lose track and fail in 50% of the videos of the dataset. On the contrary, the direct method has no problem handling the TUM video. The low optical flow between the frames helps any direct method estimate the pose accurately. By handling the light condition, the state-of-the-art direct method has lowered the drift to less than 3 meters.

2.5.3 The IMAGINE P3 Dataset

First of all, most methods, direct or indirect, will produce an inaccurate result because of the rolling shutter. However, some recent methods of state-of-the-art start handling the rolling shutter perfectly. Most indirect methods fail in these videos. The movements are too fast and too weird for any indirect method, while the light condition change, and sometimes the image is low-textured. Otherwise, the direct method succeeds almost perfectly for the same reason as the TUM dataset.

2.6 Conclusion

We have seen that SLAM is mostly useful for its real-time online application on small devices. These devices have technological limitations. The energy consumption and computational resources are limited. Today's technologies must run SLAM with one monocular camera on small embedded devices.



(a) **Issues of Direct Algorithms.** Direct methods are sensitive to high light change.

(b) **Issues of Direct Algorithms.** Direct methods are sensitive to high optical flow.

(c) **Issues of Indirect Algorithms.** Indirect methods are sensitive to moving objects.

(d) **Issues of Indirect Algorithms.** Indirect methods cannot track images with too many repetitive textures and low-textured images.

Figure 2.9: The issues of direct and indirect algorithms. These four images from the sequence n°01 of the KITTI dataset show where direct and indirect algorithms fail. KITTI 01 is a video known not to work with state-of-the-art SLAM methods.

SLAM methods of the literature do not meet the requirements. They are not sufficiently accurate, robust, or fast for the current application. Indeed, direct methods struggle with high movement, while indirect methods struggle with low-textured images.

Here is a list of today's challenges for monocular SLAM pipelines:

- Robustness to any environment, such as dynamic environments, outdoor video with a lot of light change, or repetitive texture pattern and high optical flow.
- A processing time sufficiently low to run on handheld devices such as smartphones, as this is a requirement for augmented reality.
- Accurate mapping (meaning sufficiently dense mapping with sufficiently accurate map-points) for application to augmented reality. Particularly, we show an application of augmented reality in Chapter 6.3 that needs dense reconstruction to estimate a depth map and to estimate the occlusion of a virtual object.

This defines the goal of our research: we want to achieve online SLAM for embedded devices with one monocular camera, without IMU and GPU. As it is online, the SLAM needs to run without loop closure and global bundle adjustment. For that:

- Our contribution MOD-SLAM is a semi-direct SLAM, which is askin to the Loosely-coupled SLAM [22] in principle but does not run ORB-SLAM and DSO side by side, instead mixing them at a low level, so it can benefit from the advantages of both ORB-SLAM and DSO. In particular, we show that MOD SLAM can process the whole video KITTI 01 shown as problematic in Figure 2.9.
- Current SLAM map implementations cannot handle direct and indirect information in terms of structure, speed, and coherence. To develop a hybrid SLAM pipeline, we had to create a new framework, mainly a new map. This map keeps the coherence and consistency of the information in the map while being fast.
- Because MOD SLAM has twice the parameters as a standard SLAM, we also had to develop new algorithms for automatic parameter optimization.

As our contributions are based on ORB-SLAM and DSO, the next chapter intends to develop these two methods. These two chapters will show their advantages and drawbacks to better understand our contribution in the next part.

3

The ORB-SLAM and DSO Pipelines

ORB-SLAM [12, 14] became the state-of-the-art indirect method for monocular SLAM in 2015 due to its clever use of ORB points. To be more precise, ORB-SLAM introduced a novel initialization system that manages both flat and non-flat scenes. It intelligently combines ORB features with a bag of words to quickly and precisely match points among the images. Its usage of three threads for localization, local refinement, and global refinement makes the SLAM real-time in any situation. The integration of the loop closure system into the global refinement thread drastically reduces the drift on video with loops. Direct Sparse Odometry [9] is a sparse direct SLAM method that uses only the CPU. Using photometric optimization, DSO method has a subpixel precision, resulting in an accurate 3D points cloud denser than any indirect method.

As the following chapters present our contributions based on ORB-SLAM and DSO, this chapter explains the ORB-SLAM indirect method and the DSO direct method.

3.1 ORB-SLAM

The ORB-SLAM presentation is organized as follows: First, we present ORB features and descriptors and the process used to match corners. Thanks to the features and the map, we show how the ORB-SLAM pipeline allows the algorithms to estimate the pose and map simultaneously. Then, we show how the map is organized. Notice that we only present the monocular version of ORB-SLAM. We skip the variants of the algorithm that adapt it to Stereo and RGB-D cameras [14].

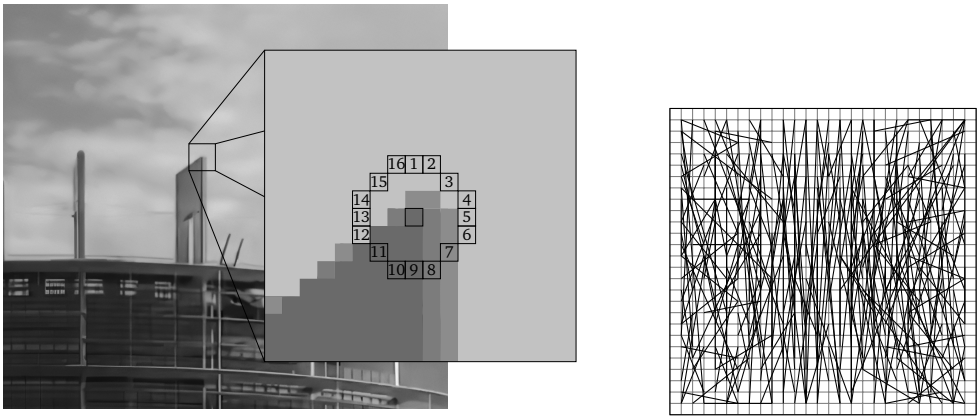
3.1.1 Tracking Corners

Indirect SLAM, such as ORB-SLAM, estimates 3D points by triangulating corners among consecutive images of the input video. It needs to extract repeatable corners (meaning that the algorithm always detects the corner in the same physical place) in the images and match these points between two images. The matching process compares the characteristics of the pixels of the image surrounding the corners. These characteristics are called descriptors. The process of matching the corners among the consecutive images of a video is called tracking.

There are multiple algorithms to extract corners from images, such as the one from Harris et al. [76], or Shi et al. [77]. Gil et al. have compared interest points and descriptor methods [78], and Gauglitz et al. compared these methods for the application of visual tracking [79]. The one we are going to detail is ORB [80], which is based on two other methods named *FAST* [81] (for "Features from accelerated segment test") and *BRIEF* [82, 83] (for "Binary Robust Independent Elementary Features").

FAST Corner detector

FAST [81] is a corner detector algorithm. It computes a score for each image pixel, representing the probability of the pixel being a corner. To calculate this score, it compares the neighbor pixels (Figure 3.1a). FAST has the advantage of being rotation invariant. For each corner, it computes the angle of the feature. Though FAST has two



(a) FAST Corner **(b)** BRIEF Descriptor Pattern

Figure 3.1: ORB Features. ORB is the combination of a scale invariant FAST and rotation invariant BRIEF.

problems: it is not scale-invariant and does not extract a uniform amount of corners among images (parts of the image might be more textured than others).

ORB [80] is a corner detector based on FAST that solves these two problems. ORB makes the FAST corner extractor scale invariant by computing it on multiple scales of the image. The ORB algorithm computes FAST corners on small image sub-blocks and ensures that the number of corners between the blocks is around the same.

BRIEF Descriptor

Then, for each extracted ORB corner, descriptors need to be computed. Descriptors allow comparing corners to evaluate if they are the same. Floating descriptors such as SIFT [84] (for "Scale-invariant feature transform") are vectors of reals numbers, which use L2 distance for comparison. Binary descriptors are vectors of binary and use Hamming distance. Binary descriptors are more suitable for SLAM pipelines, as they are much faster to compute and compare than floating descriptors: Let $n \in \mathbb{N}$ be the size of the descriptors, the L2 distance of SIFT computes n squared distances, whereas the Hamming distance uses only $n/32$ processor cycle[1].

BRIEF [82, 83] is a method that aims to compute the binary descriptor of a corner. The descriptor is computed by looking at the intensity of the neighbor pixels of the corners. Each binary of the descriptor is the result of the comparison of the intensity of two pixels at a specific position relative to the position of the corner. The segment endpoints of the figure 3.1b are the pixel positions where the image intensity is compared, relatively to the position of the corner. Depending on the result of the comparison, a 1 is put in the binary descriptor if the first intensity is lower than the second intensity, and a 0 otherwise. *ORB* [80] uses a rotation invariant adaptation of the BRIEF descriptors, obtained by un-rotating each feature with its FAST angle before computing its descriptors.

To conclude, *ORB* [80] is a method combining a well-distributed scale-invariant FAST with orientation-invariant BRIEF. ORB corner extraction method and descriptor allows for tracking corner among the consecutive frames of a video. To track these corners, algorithms are needed. We see these algorithms in the next section.

[1] by combining the AND operator, and the special *popcount* ASM instruction which computes the number of 1 in a vector of 8 bytes in one CPU cycles.

3.1.2 Matching Algorithms

Matching algorithms aim to match the 2D corners of one image with other 2D corners of a second image or 2D corners with a list of 3D map-points.

A 2D-2D brute force matching method compares all the descriptors of one image with all the descriptors of a second image. A match is validated when: (i) the distance between the two descriptors is below a threshold; (ii) the distance between the two descriptors multiplied by a ratio is below the second-best distance. Let $n_a \in \mathbb{N}, n_b \in \mathbb{N}$ the number of corners in the two images. A brute force matching has a complexity of $O(n_a \times n_b)$.

A 2D-2D locality-sensitive matching [85] (LSH matching) algorithm hashes similar descriptors into the same "buckets". One bucket contains descriptors that are highly probable to be similar. LSH algorithms have a complexity of $O(n_a log(n_b))$ complexity. Among the LSH algorithms, a notable one is matching supported by *Bag of Words* [86].

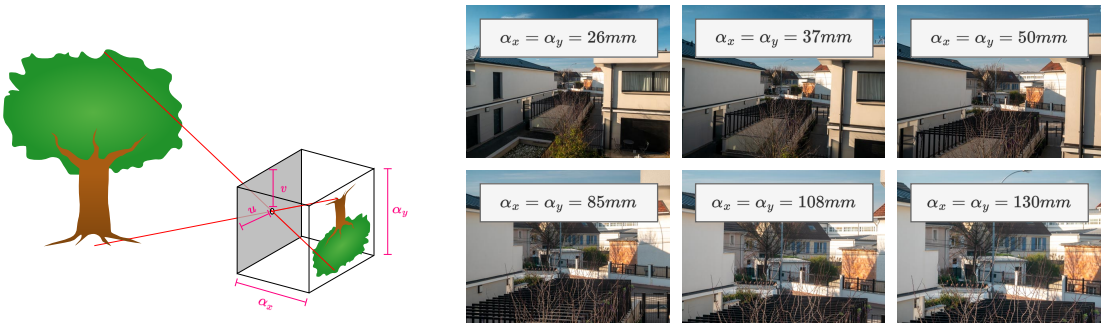
Notice that some algorithms allow filtering the matching by using geometric consistency, such as the work of Choi et al. [87].

Let's a frame associated with a pose estimation, and a list of existing 3D map-points. To find 2D-3D correspondences between interest points of the frame and the existing 3D map-points, a method consists of re-projecting these 3D map-points into the image. The algorithm compares the descriptor of each re-projected 3D point with the descriptors of the corners in the vicinity. The descriptor of a 3D point is the median[2] of all its associated 2D interest point descriptors.

Thanks to the matching algorithms, ORB-SLAM can track corners among the consecutive image of the video and make 2D-3D correspondences. This gives data for ORB-SLAM to optimize 3D points and pose and therefore run its SLAM pipeline.

3.1.3 Re-projection Optimization

ORB-SLAM pose estimation, mapping, and bundle adjustment algorithms are based on a process of optimization. This process aims to minimize a *re-projection error*.



(a) The intrinsic parameters. $\{\alpha_x, \alpha_y\}$ is the focal lenght. $\{u, v\}$ is the principal point. Figure modified from [88].

(b) Impact of different focal lengths α on the image.

Figure 3.2: The pinhole model.

The Pinhole Model

The pinhole model (Figure 3.2) approximate the production of image by a pinhole camera. A pinhole camera is an "ideal camera" with no distortion. The intrinsic matrix produces a 2D position in pixel coordinate from a homogeneous 2D position in the camera coordinates. The intrinsic matrix is defined by Hartley and Zisserman [10] as:

$$\mathbf{K} = \begin{bmatrix} \alpha_x & \gamma & u & 0 \\ 0 & \alpha_y & v & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{3.1}$$

This matrix is the composition of a scaling transformation by the focal length α_x and α_y and a translation by the principal point offset u, v. γ is the skew coefficient between the horizontal and the vertical axis, and is equal to 0 for most lenses.

The Re-projection Error

Let $\mathbf{t}_k \in \mathbb{R}^3$ be a map-point, $\mathbf{T}_i \in \mathbb{SE}(3)$ be the pose of a frame[3], and $\Pi^{\mathbf{T}_i}(\mathbf{t}_k)$ be the projection of $\mathbf{t}_k$ through $\mathbf{T}_i$. The re-projection error is the norm between the projection of a 3D point and the point $\mathbf{F}_{i,h} \in \mathbb{R}^2$ where it should appear on the frame:

$$E^{\text{proj}}(\mathbf{F}_{i,h}, \mathbf{t}_k, \mathbf{T}_i) = \left\| \mathbf{K}\Pi^{\mathbf{T}_i}(\mathbf{t}_k) - \mathbf{F}_{i,h} \right\|_2 \tag{3.2}$$

[2]Given a set of descriptors, the n^{th} bit of the median descriptor of this set is equal to the median of the n^{th} bit of all descriptors.
[3]$\mathbb{SE}(3)$ is the set of pose

The reprojection error is computed thanks to the 2D-2D and/or 3D-2D matchings that have been previously computed.

To minimize the sum of squared re-projection errors, ORB-SLAM uses the Levenberg–Marquardt algorithm explained in the Appendix A. ORB-SLAM uses a library named G2O [89] which contains algorithms for re-projection error minimization.

This re-projection error minimization is the most essential part of all indirect SLAM algorithms. It is used in all the essential stages of the pipeline, that we are going to describe.

3.1.4 ORB-SLAM Pipeline

The ORB-SLAM pipeline follows the previous general pipeline. Like most pipelines, it is composed of three main blocks, which are the tracking and pose estimation, the mapping and bundle adjustment, and the loop closure. ORB-SLAM has the particularity to execute each of the three processes in a separate thread. The three following sections explain each main stage of the pipeline.

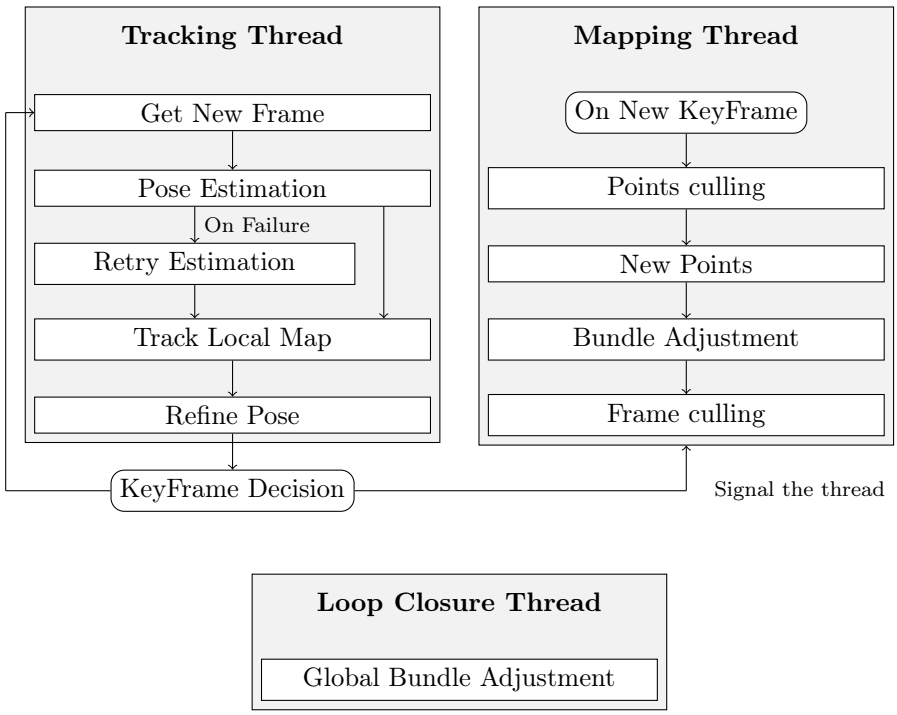


Figure 3.3: ORB-SLAM Pipeline. ORB-SLAM uses three threads to track, map, and close a loop.

Tracking Thread

The tracking thread's goal is to continuously estimate the pose of incoming frames of the video. The pose estimation is based on the 3D mapped points in the mapping thread, explained in the following mapping thread part.

1. ORB Corner extraction. First, the tracking thread receives a new image from a video, coming from a camera or the dataset. ORB-SLAM computes ORB corners for the new frame as described in Section 3.1.1. Then ORB-SLAM starts a pose estimation of the frame.
2. Pose Estimation from the previous frame. As a first attempt, ORB-SLAM assumes that the camera moved at a constant velocity in the same direction. It detects 3D-2D matches by re-projecting the local 3D map-points and uses these matches to estimate a pose. If the number of inliers during the optimization is below a threshold $\psi^{\text{orb}}_{\text{inlierNumThreshold}}$, ORB-SLAM rejects the hypothesis, as it violates the constant velocity motion model, and starts a second attempt.
3. Pose Estimation from the reference frame. As a second attempt, ORB-SLAM computes 2D-2D matches from the current frame with the last key-frame. It uses these matches to estimate a pose. If the number of inliers during the optimization is below the threshold $\psi^{\text{orb}}_{\text{inlierNumThreshold}}$, ORB-SLAM refuses the solution. If ORB-SLAM has rejected both attempts, it considers itself lost and waits for the loop closure thread to recover the tracking.
4. After each iteration of the pose optimization of the previous two attempts: (i) the matches with an error above a threshold are removed from the optimization and marked as outliers; (ii) the outliers whose re-projection falls below the threshold are considered again as inliers. This makes the optimization more robust in the case of dynamic environments or repetitive textures.
5. Local Map Tracking. ORB-SLAM stores the recent key-frames and map-points inside two list named *active key-frames* and *active map-points*. These two lists are used for the further pose estimation and triangulation. After the pose estimation, ORB-SLAM updates the active key-frames and the active map-points. Then, it tries to re-project the active map-points on the current frame to obtain more 2D-3D matches. Because most of the time, ORB-SLAM successfully brings new 2D-3D matches during the local map tracking, it can refine the pose to obtain better accuracy.
6. New keyframe Decision. The mapping thread has a system to cull useless key-frames. Consequently, the tracking thread can create numerous key-frames without slowing

the bundle adjustment. Allowing the tracking thread to create key-frames with almost no limit makes the SLAM more robust to challenging camera movements. ORB-SLAM tracks the number of 3D map-points inside the local map, and saves this number during the key-frame creation. If the current number of 3D map-points inside the local map falls below the initial number of map-points multiplied by a fixed ratio of 90% (as defined in by the original paper of ORB-SLAM [14]), then the tracking thread creates a new key-frame. The key-frame is sent to the mapping thread so that it can create new 3D map-points.

Mapping Thread

The goal of the mapping thread is to create new 3D map-points for the pose estimation, and to refine the map. The creation of new 3D map-points needs to be executed quickly, therefore, the process of refinement of the map can be interrupted at any time, to start the most rapidly possible the creation of new 3D map-points. The interruption occurs when the tracking thread estimates that is starts to have too few map-points for the pose estimation.

1. Point culling. The algorithm may have created new 3D points in the previous frame. Some of these new 3D map-points may be outliers. Therefore, the point culling algorithm checks that the newly created map-points have been matched through enough frames.
2. Triangulation. A 3D map-point can be produced by the triangulation of two 2D-2D matches, which involves the singular value decomposition of a 6×4 matrix [10]. ORB-SLAM accept the new 3D map-points only if they have a certain amount of parallax. New 3D points go to a list of newly triangulated points, which the algorithm deems uncertain. The point needs to be successfully tracked across multiple images (more than 25% of the frame after three key-frames and five frames have passed since the map-point has been created) to be classified as an inlier.
3. Local Bundle Adjustment. The bundle adjustment refines the local map through a Levenberg-Marquardt minimization of the pose and the 3D points simultaneously.
4. Frame culling. When two key-frames are too close and share almost the same points, ORB-SLAM can remove one redundant key-frame. This improves the bundle adjustment speed.

Loop Closure Thread

ORB-SLAM uses a thread to match the points across distant frames of a sequence, by using Bag of Words. If a loop is found inside the map, ORB-SLAM tries to merge the 3D points, and do a global bundle adjustment of all the key-frames and the points inside the loop. The loop closure thread is decomposed into 4 steps: the detection of candidates, the computation of similarity transformation, the map-points fusion, and the global bundle adjustment. We do not detail further this stage as it is not used in our framework.

Initialization

To calculate a pose, ORB-SLAM needs 3D map-points. While to compute 3D map-points, ORB-SLAM needs several pose estimations. The goal of the initialization part of ORB SLAM is to compute, at the same time, the pose of two frames at the beginning of the video and 3D points, which can serve to estimate the pose of new frames. At first, the initialization process extracts and matches ORB points among images. Then, two cases are possible:

- When the scene has sufficient parallax, fundamental matrices need to be computed and decomposed into a rotation matrix and a translation.
- When the scene is plane, a fundamental matrix will not work; therefore, a homography matrix needs to be computed and transformed into a transformation matrix.

Because ORB-SLAM cannot predict whether the scene is plane, it computes both the fundamental matrix and the homography matrix. These matrices are calculated based on the previously computed ORB correspondences inside a RANSAC scheme [64]. ORB-SLAM computes errors for the two matrices with a symmetric transfer error, and retains the matrix with the lowest error.

- If selected, the fundamental matrix is decomposed into four possible candidate poses using singular value decomposition.
- If the homography deemed better, it is decomposed into eight possible candidate poses using the method of Faugeras et. al [90].

Each of these candidate poses are tested by triangulating points for each. To validate a solution: the points need a certain parallax and needs to be in front of the camera. If no clear solution appears, the algorithm is executed for the next frame. Otherwise, a bundle adjustment is performed, and the pipeline is started.

3.1.5 The Map of ORB-SLAM

ORB-SLAM stores its map using the following representation:

- For the poses, it uses translation vectors and rotation matrices;
- For the 3D point, it uses 3D euclidean coordinates.

Local Map, Active Frames, and Active Points To accelerate the pose estimation process, ORB-SLAM stores a map subgraph named **local map**. The local map contains the **active frames** and the **active points**. The active frames contain frames adjacent to the latest frame. The active points are all the 3D points visible in at least one active frame. The local map is updated during the local map tracking. Only active points are tracked during pose estimation, and only the local map undergoes a bundle adjustment during key-frame creation.

Co-visibility Graph, Essential Graph, Spanning Tree The co-visibility graph (Figure 3.4a) is a graph containing the edges between frames which share the same points. Essential graph (Figure 3.4b) and Spanning Tree (Figure 3.4c) are two minimal versions of this graph. When a loop closure is detected, it is more efficient to optimize over the spanning tree than the whole key-frame of the loop.

(a) Co-visibility Graph (b) Essential Graph (c) Spanning Tree

Figure 3.4: ORB-SLAM Map Graphs. Figure extracted from ORB-SLAM Paper: Co-visibility Graph, Spanning Tree, and Essential Graph [12].

3.1.6 Results

Table 3.1 lists the essential parameters of ORB-SLAM. These parameters are going to be re-used during the explanation of our hybrid method in Chapter 5.

The parameter $\psi^{\text{orb}}_{\text{numCorner}}$ corresponds to the number of ORB corners that are extracted from each image. 2000 is a high number, but at most 5% of these corners are used for the optimization processes because ORB-SLAM has a strict outlier rejection algorithm. The parameter $\psi^{\text{orb}}_{\text{inlierNumThreshold}}$ corresponds to the number of inliers required for a pose estimation to be accepted. By maintaining a relatively low threshold, the algorithm is provided with an opportunity to continue the tracking even under challenging conditions. The parameter $\psi^{\text{orb}}_{\text{KFRatioThreshold}}$ is a threshold of the number of tracked map-points since the last key-frame, used to decide the creation of a new key-frame. The parameter $\psi^{\text{orb}}_{\text{baNumIteration}}$ is the maximal number of iterations in the bundle adjustment.

The original ORB-SLAM paper evaluated ORB-SLAM with loop closure. This evaluation is not relevant as we are developing SLAM without loop closure. With the parameters of Table 3.1, we evaluated ORB-SLAM on the KITTI dataset, the TUM dataset, and the IMAGINE P3 dataset. For each sequence, we averaged the results over ten runs. We validated our evaluation with those from the CNN-SVO Paper [72], which also evaluated ORB-SLAM without loop closure.

ORB-SLAM obtains the results presented in Table 3.2. The method succeeds on the KITTI dataset but loses track during most TUM sequences and all the IMAGINE P3 datasets. Chapter 6.3 compares the results of ORB-SLAM with those of DSO (presented in the following section) and our hybrid method.

The next method we are going to describe, DSO, a direct method based on optical flow and photometric optimization, succeeds on the TUM sequences but struggles on the KITTI dataset.

Notation	Description	ORB-SLAM Default Values
$\psi^{orb}_{numCorner}$	Number of ORB points per image	2000
$\psi^{orb}_{inlierNumThreshold}$	Pose Inlier Threshold	10
$\psi^{orb}_{KFRatioThreshold}$	Keyframe Point Ratio	0.9
$\psi^{orb}_{BaNumIteration}$	Bundle Adjustment Iterations	0 to 15

Table 3.1: ORB-SLAM Parameters Summary.

	ORB-SLAM w/o loop	Evaluation from CNN-SVO [72]
Absolute Trajectory Error		
Average*	**34.79**	**35.82**
KITTI 00	67	78
KITTI 01	x	x
KITTI 02	43	41.0
KITTI 03	1	1
KITTI 04	0.9	0.9
KITTI 05	43	40
KITTI 06	49	52
KITTI 07	17	17
KITTI 08	58	52
KITTI 09	60	58
KITTI 10	9	18
Success Ratio		
TUM	56%	
Imagine P3	0%	

Table 3.2: Absolute Trajectory Error on KITTI datasets [20], and Success Ratio on TUM and Imagine P3. *The average has been made without KITTI 01, because most methods fail on this video.

3.2 Direct Sparse Odometry (DSO)

The previously presented ORB-SLAM became the state-of-the-art indirect method. On the contrary, DSO [9] became the state-of-the-art direct method for CPU-only monocular SLAM in 2017. This is due to its clever use of photometric optimization. The presentation of the DSO method is organized as follows: first, we describe how to optimize directly on the image pixels, then, we develop the pipeline of DSO.

3.2.1 Photometric Error

DSO pose estimation, mapping, and bundle adjustment explained in the following, are based on a process of optimization. However, contrary to the previous ORB-SLAM algorithms, these direct algorithms optimize poses and map-points by minimizing a *photometric error*. The photometric error is a function expressed from the pixel intensities of successive images, similar to an optical flow approach [91].

Let $\mathbf{t}_k \in \mathbb{R}^3$ be a map-point, $\mathbf{T}_i \in \mathbb{SE}(3)$ be the pose of a frame, and $\Pi^{\mathbf{T}_i}(\mathbf{t}_k)$ be the projection of $\mathbf{t}_k$ through $\mathbf{T}_i$. The photometric error is the huber norm [92], writted as $\|\cdot\|_{\text{huber}}$, between the color intensity on the image where the 3D map-point project and the color intensity of the corner $c_{i,h} \in \mathbb{R}$ where it appears first:

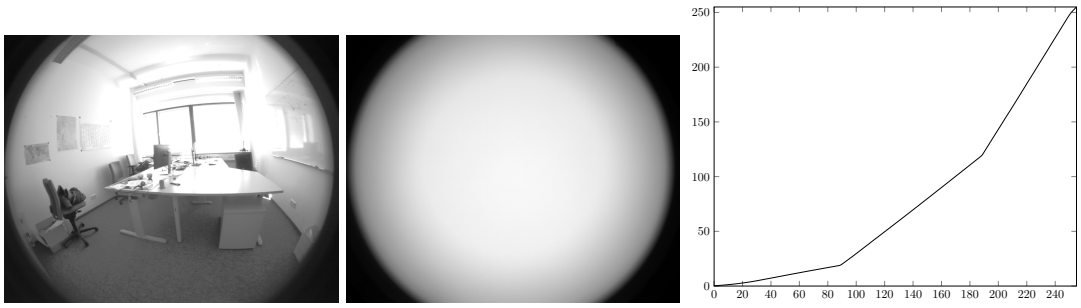
$$E^{\text{photometric}}(\mathbf{T}_i, \mathbf{t}_k, c_{i,h}) = \left\| c_{i,h} - I_i(\Pi^{\mathbf{T}_i}(\mathbf{t}_k)) \right\|_{\text{huber}}. \tag{3.3}$$

The DSO method parameterizes the map-points with an inverse depth [93]. Therefore, when optimizing the map-points coordinates, DSO optimizes only one parameter, the inverse depth. This constrains the optimization algorithm to make each direct map-points stay on an epipolar line.

To minimize the error of Equation 3.3, DSO must compute the gradient of the images $\{I_i, i \in \mathbb{N}\}$. Moreover, DSO does not represent the images I_i as discrete pixel arrays but rather as continuous 2D functions (for that DSO interpolates the intensity at a given position).

$$I_i : \mathbb{R}^2 \to \mathbb{R}. \tag{3.4}$$

As shown in Figure 3.5, in real-life cases, the colors of captured images may not correspond to real-life colors. The edges of the images can be darker due to the vignetting



(a) Image from TUM Dataset **(b)** TUM Camera Vignetting **(c)** TUM Camera Non-linear response

Figure 3.5: Photometric Calibration of TUM 01 [21].

effect, and the colors might be distorted due to photometric distortion. DSO needs a photometric calibration to correct the image's pixels colors:

$$I_i(\mathbf{o}) = \text{photometricCorrection}(\mathbf{o}, U_i), \tag{3.5}$$

with U the distorted image [21], and $o \in \mathbb{R}^2$. DSO is able to work without photometric calibration. In that case, it produces less accuracy and robustness.

Furthermore, the light conditions might change during video capture. DSO introduces a model of light, composed of two parameters per frame: $\mathrm{l}_i^a \in \mathbb{R}$ and $\mathrm{l}_i^b \in \mathbb{R}$. We denote these parameters as light parameters. DSO optimize these parameters at the same time as the poses. The photometric error formula $I_i(\Pi^{\mathbf{T}_i}(\mathbf{t}_k))$ become $\mathrm{l}_i^a \times I_i(\Pi^{\mathbf{T}_i}(\mathbf{t}_k)) + \mathrm{l}_i^b$.

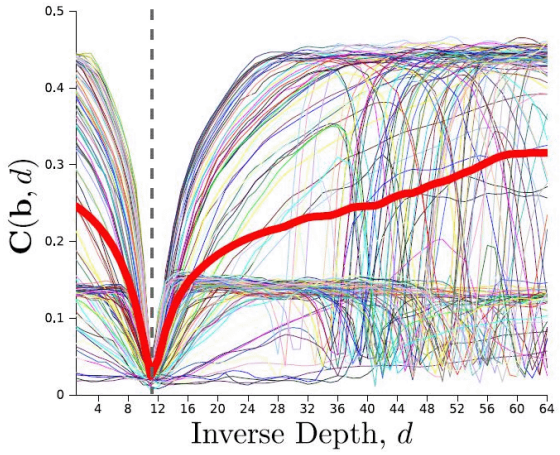


Figure 3.6: Inverse depth error has a lot of local minimum. This plot comes from the paper DTAM [7].

To minimize the photometric error, DSO uses a Gauss-Newton minimization. As shown in Figure 3.6, the photometric error has many global minimums when optimizing a single map-point. Consequently, if the optimization of a single map-point is not well initialized, it can result in an incorrect minimum. Therefore, DSO must take precautions:

- Each map-point is associated with an inverse depth interval, initially from 0 to infinite. This interval is reduced among the video images. The optimization is initialized at the center of the interval when it is small enough. If DSO cannot reduce this interval, it rejects the associated map-point.
- DSO algorithms optimize not only with the projection of the map-points on the frame $\Pi^{\mathbf{T}_i}(\mathbf{t}_k)$, but also with the neighborhood pixels of the projection $\Pi^{\mathbf{T}_i}(\mathbf{t}_k) + s$ with $s \in \mathbb{R}^2$ a pixel shift. The DSO method defines a pattern that contains multiple shifts. By considering the neighborhood pixels into account, DSO increases the amount of data for optimizing a single point. Hence DSO increases the accuracy of the optimization. DSO uses seven shifts, an informed choice, as it is amenable to optimization with vector SSE instructions. Therefore, DSO has to minimize the squared sum of 8 residuals to optimize a single map-point.
- To reduce the number of local minimums in the image, DSO runs its optimization first on smaller images, then on larger images.

3.2.2 Pipeline

Figure 3.7 shows that the DSO pipeline contains two main blocks: the pose estimation block and the mapping block. The following is a description of these two blocks.

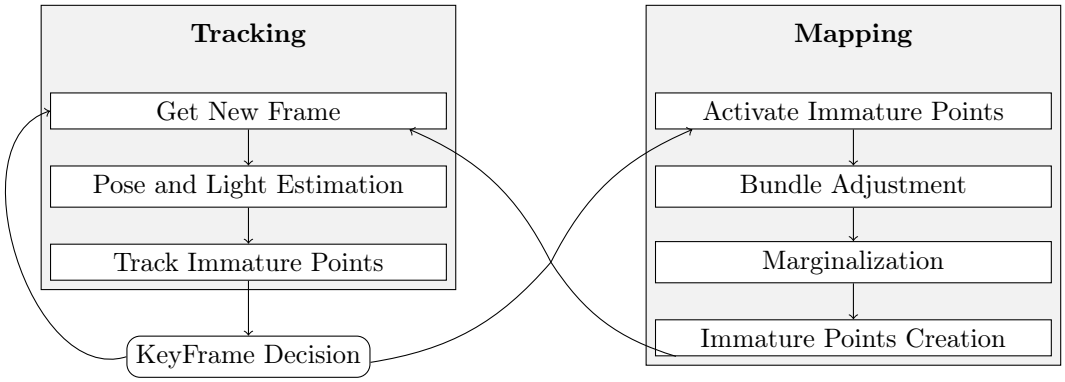


Figure 3.7: DSO pipeline.

Pose Estimation

1. Get New Frame. DSO pre-processes frames by undistorting the image and making a Gaussian pyramid [94] of the image by downscaling the image by two at multiple levels. These down-scaled image versions are needed for the optimization process explained above. DSO needs at least five scales of the image, but can work with more levels depending on the resolution of the video.
2. Pose Estimation. The DSO pose estimate is carried out by optimizing multiple cameras pose candidates, using a photometric minimization of recent points. The pose estimation algorithm creates candidates using a constant-velocity motion model. If it finds a good camera, it stops trying other candidates and returns directly. Otherwise, it returns the optimized camera, which produces a lower residual. To make the photometric optimization more robust to high optical flow, it is executed multiple times at each scale of the image pyramid. First, with the lower scale, to optimize on a smaller pixel shift when the camera moves a lot. Then, the size of the image increases until it reaches its original size.
3. Track immature points. DSO tracks photometrically new potential 3D points. This tracking is done on each consecutive image of the video to avoid problems due to high optical flow.
4. Key-frame decision. DSO chooses to produce a key-frame when: There has been a significant camera movement since the last key-frame, or there has been a high light change since the previous key-frame, or the pose estimation residual suddenly became high.

Mapping

1. Activate Immature Points. DSO counts points to achieve a target density of $\psi^{\text{dso}}_{\text{desiredPointDensity}}$. It classifies the map-points from immature to mature. The map-points to classify as mature are chosen by their quality, but also so that all the active points reprojection on the frame are well distributed on all the images. A map-point is considered to have a good quality when its depth range becomes small and if all its residuals to each frame are low enough.
2. Frame management DSO wants to hold a maximum of $\psi^{\text{dso}}_{\text{baMaxFrames}}$ frames to optimize inside the bundle adjustment. These frames are called the active key-frames. A newly created key-frame will necessarily become an active key-frame.

The $\psi^{\text{dso}}_{\text{baMaxFrames}} - 1$ other active key-frames are chosen so that they "overlaid with the points hosted in them" [9].

3. Marginalization of points During the bundle adjustment, each map-point must be seen by a sufficient amount of active key-frame to be optimized. When this is not the case, the most accurate points are fixed and added to a marginalized Hessian. The remaining are deleted.
4. Bundle Adjustment. DSO uses a Levenberg-Marquadt optimization algorithm to minimize the photometric error.
5. New Point Creation. DSO creates $\psi^{\text{dso}}_{\text{immatureDensity}}$ new immature points. These are candidate points, which means that DSO reserves its decision whether they are inliers or outliers. DSO tracks the immature map-points among the images of the videos, on which DSO reduces the range of the immature map-points inverse depth.

Initialization

We have described the pose estimation and the mapping processes of DSO. However, the pose estimation depends on the result of the mapping. And the mapping depends on the result of the pose estimation.

The DSO initialization aims to initialize the map so that both the pose estimation and the mapping have data to base on their estimation. The DSO initialization optimizes the map-points and the pose during the first frames of the videos. Feature point are extracted on the first frame of the video. Map-points are created at a distance of 1 from the first. As described previously, the photometric error has a lot of local minima (Figure 3.6). As the optimization does not converge easily, a regularization loss is added to promote the neighbor points with similar depth. This initialization of DSO is a slow process and not constrained to be real-time.

3.2.3 Parameters Summary

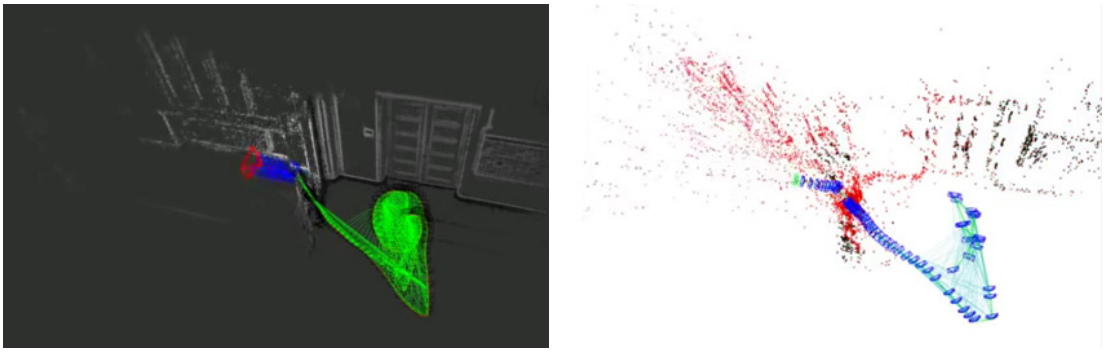
Table 3.3 summarizes the important parameters of the DSO method that are going to be re-used during the description of our method in Chapter 5. The DSO method extracts on each key-frame $\psi^{\text{dso}}_{\text{immatureDensity}}$ corners. At each key-frame creation, it tries to hold $\psi^{\text{dso}}_{\text{immatureDensity}}$ map-points. $\psi^{\text{dso}}_{\text{baMaxFrames}}$ corresponds to the number of frames (with their map-points) the DSO method will refine at each new key-frame. Finally, $\psi^{\text{dso}}_{\text{baNumIterations}}$ is the number of refinement iterations that the DSO methods will do at each iteration.

Notation	Description	DSO Value
$\psi^{dso}_{immatureDensity}$	Density of immature points to create	1500
$\psi^{dso}_{desiredPointDensity}$	Density of immature points to hold	2000
$\psi^{dso}_{baMaxFrames}$	Bundle Adjustment Number of Frames	7
$\psi^{dso}_{baNumIterations}$	Bundle Adjustment Iterations	6

Table 3.3: DSO Parameters Summary.

	DSO
Absolute Trajectory Error	
Average*	**55.56**
KITTI 00	114
KITTI 01	x
KITTI 02	120
KITTI 03	2.1
KITTI 04	1.5
KITTI 05	52
KITTI 06	59
KITTI 07	17
KITTI 08	111
KITTI 09	63
KITTI 10	16
Success Ratio	
TUM	100%
Imagine P3	100%

Table 3.4: Absolute Trajectory Error on KITTI datasets [20].



(a) Density and Precision of DSO Map-Points. **(b) Density and Precision of ORB-SLAM Map-Points.**

Figure 3.8: Comparison of DSO and ORB-SLAM Map-Points Density and Precision. These images comes from the presentation video of DSO [9]

With the set of parameters of Table 3.3, DSO obtains the results of Table 6.1. Contrary to ORB-SLAM, DSO succeeds on the TUM and Imagine P3 sequences. However, it produces higher drift on the KITTI sequences.

3.3 Conclusion

The two main families of monocular SLAM method, direct and indirect, have complementary weaknesses. ORB-SLAM, a state-of-the-art indirect method, is sensible for low or repetitive textured images due to its usage of descriptors. DSO, a state-of-the-art direct method, is sensitive to high optical flow, because of its usage of photometric minimization. The next chapters focus on the creation of a hybrid SLAM method, that combines the advantages of both families of methods. In the following, we develop the generalized map, an essential implementation of the map for the development of hybrid SLAM methods.

4

Generalized Map

The map is a crucial element of SLAM methods. Its role is to efficiently store and retrieve information, such as camera pose, 3D point coordinates, 2D features, 2D-3D correspondences, etc. During the execution of a SLAM pipeline, the algorithm constantly updates the map. Consequently, its implementation significantly impacts memory usage and execution time.

We have shown in the previous chapter that current map designs limit the development of hybrid SLAM pipelines. The present method proposed in the literature, such as joint optimization and decision systems, uses algorithms that share different types of information. The various algorithms need a system to share these pieces of data efficiently.

For these reasons, we propose a *generalized map*. The generalized map is a code basis for developing hybrid, direct, and indirect SLAMs. It aims to be compatible with any algorithm. It makes all these algorithms of different families work together by doing the communication between these different algorithms, which was not intended to work together at first. It has the duty to maintain the coherence of the information between these different algorithms. The generalized map we propose allows us to create a hybrid SLAM method, explained in the next chapter.

In Figure 4.1, algorithms of direct family and indirect family use the different functions of the interface of the generalized map to retrieve information and store their estimation. The generalized map is able to signal these algorithms that one algorithm has modified information it uses, through the function of the interface (that we call observer in the following). In the figure, the arrow link in both way the generalized map and the algorithms, as they communicate through the interface function of the generalized map.

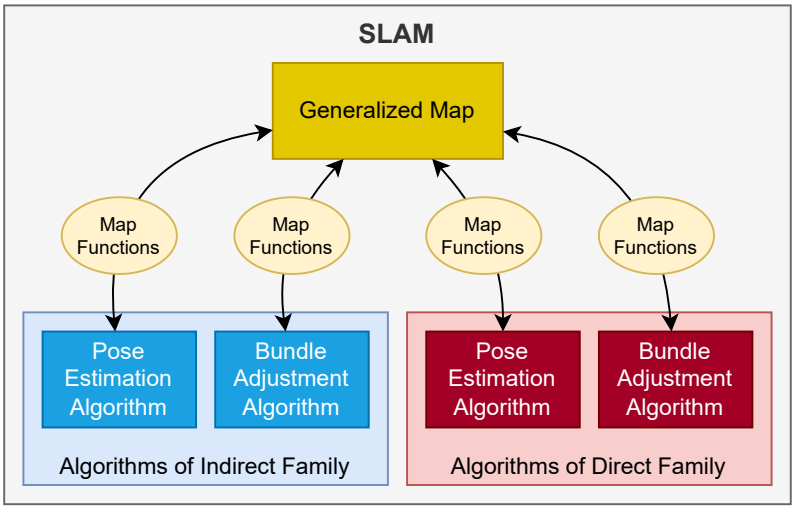


Figure 4.1: The generalized map makes the communication between direct and indirect algorithms. Each algorithm of the different families does not communicate between each other or through the SLAM, but instead through the map.

A generalized map is a map that follows these criteria:

1. The Generalized Map must be compatible with any family of methods. The map must allow any algorithm to store its specific information.
2. The Generalized Map must ensure the coherence of the shared information between different algorithms families. The map definition must be independent of the information representation that the algorithms need to use. Also, algorithms of different families can update the map at different times. Therefore, the map must guarantee the propagation of these modifications.
3. The mechanism that allows the Generalized Map to support the generalization may not impact the efficiency of implementation. A Generalized Map has a higher number of data to process than a standard map implementation. It has to use efficient algorithms to limit its computational resources.
4. The Generalized Map must remain consistent. The Generalized Map must be made to make work algorithm that was not intended to work together at first. They are not made to remain consistent together. For example, one algorithm may remove a map-point while another uses it.

To deal with these properties, we propose a unified formalism for the map and provide its formalization based on graph theory.

4.1 Graph Structure of the Map

In our work, we propose to consider a map as a weighted non-oriented graph. In the graph, the vertices represent the frames. The edges link the co-visible frames, according to [74]. We call this graph *generalized map* $G = (V, E, w)$, where V is the set of vertices, E is the set of edges and w is an application from $V \times V$ to N resulting in the weight of an edge.

The figure 4.2 is a summary of the generalized map graph we define below.

The set of vertices V is defined as follows:

$$\begin{aligned} V &= \{v_1, ..., v_n\}, n \in \mathbb{N} \\ v_i &= \{\mathbf{T}_i \in \mathbb{SE}(3), \\ &\quad I_i : \mathbb{R}^2 \times \mathbb{N} \longrightarrow \mathbb{R}, \\ &\quad \mathbf{F}_i = \{\mathbf{f}_{i,0}, ..., \mathbf{f}_{i,h}\}, \mathbf{F}_i \in \mathbb{R}^{2\times h} \\ &\quad C_i^v \subset \mathbb{N}\}. \end{aligned} \tag{4.1}$$

$\mathbf{T}_i$ is the transformation of the pose estimated by the algorithm. It can take multiple representations, as explained below. I_i is a multi-scale image representation (pyramid). We present the tags below. $\mathbf{F}_i$ is a matrix of 2D feature point coordinates.

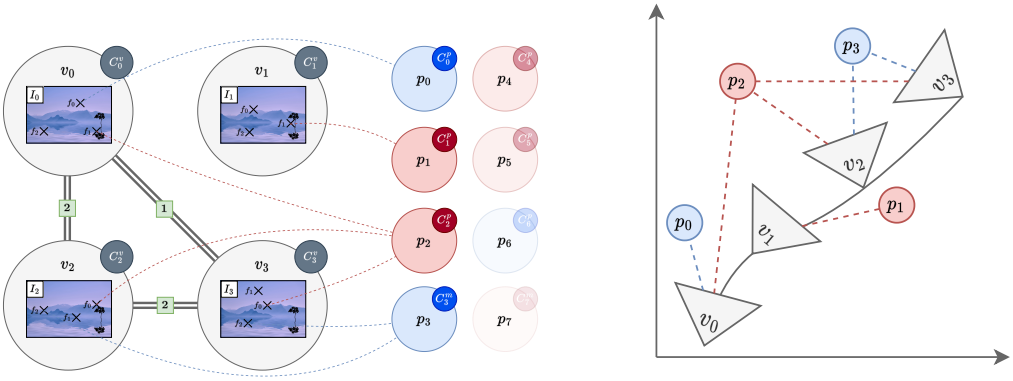


Figure 4.2: Map graph structure illustration. Red circles represent the edges, containing images and feature points. The red circles represent the direct map-points, and the blue circles represent the indirect map-points. Blue and red dotted edges represent point visibility. Thick gray edges are the co-visibility edges.

Before defining the edges of the graph, we introduce the map-points. The set of map-points P is defined as:

$$\begin{aligned} P &= \{p_1, ..., p_l\} \\ p_k &= \{\mathbf{t}_k \in V \times \mathbb{R}^3, \\ &\quad C_k^p \subset \mathbb{N}\} \end{aligned} \tag{4.2}$$

with $\mathbf{t}_k \in V \times \mathbb{R}^3$ the 3D coordinates of the map-point p_k relative to a frame. C_k^p are set of tags attached to the map-point p_k. We explain the tags below. The set S defines the visible map-points on a frame:

$$\begin{aligned} S &\subset V \times P \\ S_i &= \{p_k \in P | (v_i, p_k) \in S\} \end{aligned} \tag{4.3}$$

S_i defines the visible map-points of the frame v_i, with $i \in \mathbb{N}$. We also define A the association between a feature point of a frame and a map-point:

$$A: \quad V \times \mathbb{N} \quad \longrightarrow \quad P \tag{4.4}$$

In the following chapters, A is also used such that:

$$A: \quad V \quad \longrightarrow \quad \mathbb{N} \times P \tag{4.5}$$

The weight between two vertices v_i, v_j is the number of map-points in common between v_i and v_j:

$$w_{i,j} = card(S_i \cap S_j). \tag{4.6}$$

The set of edges E is then defined as:

$$\begin{aligned} E &\subset V \times V \\ E &= \{(v_i, v_j) | w_{i,j} \geqslant 1, i \neq j\}. \end{aligned} \tag{4.7}$$

Several algorithmic steps need to run on specific frames and points of the map. In a mixed SLAM, a direct algorithm only uses direct points and an indirect algorithm only uses indirect points. For this reason, we introduce the concept of *tags*. SLAM pipelines can create named tags of frames or points. Each frame and point can carry one or more tags. For instance, a map-point can be tagged as “Direct” or “Indirect". C_i^v is the set of tags that a frame v_i carries. C_k^p is the set of tags a point p_k carries. Thanks to the tag

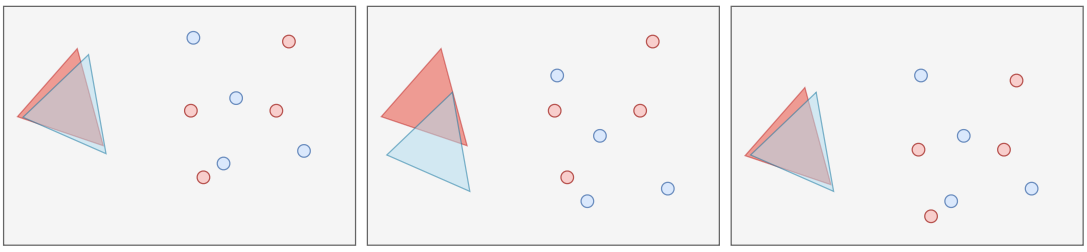
system, the generalized map allows algorithms to manipulate the subgraphs of objects with specific tags.

4.2 Graph Coherence

The previous subsection presented a generalized map structure that is suitable for all families of algorithms. This generalized map has the role of keeping the coherence of information between direct and indirect families of algorithms. This subsection aims to show how to keep the generalized map coherent when algorithms of different families manipulate it.

Let $G = (V, E, w)$ be a generalized map. G contains two types of information that direct and indirect families share: the poses $\{\mathbf{T}_i, i \in \mathbb{N}\}$ of the frames $\{t_i\}$ and the coordinates $\{\mathbf{t}_k, k \in \mathbb{N}\}$ of map-points $\{p_k\}$. Let M_{direct} be the set of map-points carrying a direct tag (direct $\in C_k^p$), and M_{indirect} be the set of map-points carrying an indirect tag (indirect $\in C_k^p$).

Let v_i be a frame of G. Let $\mathbf{T}_i^{\text{direct}}$ be an estimation of $\mathbf{T}_i$ from the direct points M_{direct}. Let $\mathbf{T}_i^{\text{indirect}}$ be an estimation of $\mathbf{T}_i$ from the indirect points M_{indirect}. A map is coherent if $\mathbf{T}_i^{\text{direct}} \approx \mathbf{T}_i^{\text{indirect}}$. This is shown in Figure 4.3. At first, in Subfigure 4.3b, the map is coherent, meaning that the direct pose estimation and the indirect pose estimation are almost equal. In Subfigure 4.3b, when an indirect algorithm refines the indirect pose and points, it can move them. However, an indirect algorithm does not move the direct point. The map became incoherent. In Subfigure 4.3c, a deformation is applied to keep the map coherent.



(a) Initial Map. **(b) After an indirect bundle adjustment.** **(c) After a deformation.**

Figure 4.3: The coherence of the map. The pose in red is computed from the direct points in red. The pose in blue is computed from the indirect points in blue.

The generalized map must ensure coherence between direct and indirect map-points. When an algorithm modifies direct or indirect map-points, the generalized map must adapt the points of the other family.

When a direct algorithm optimizes the pose of a frame using the direct map-points, the generalized map corrects the indirect map-points using a re-projection error minimization. The optimization of a indirect map-points is fast, when fixing the poses of the camera [1].

When an indirect algorithm moves frames with indirect points, the direct points cannot be photometrically optimized, as they risk converging to a local minimum, as explained in the previous Chapter 3.2 page 62. We propose to "deform" the map when an indirect algorithm moves the indirect frames by moving direct points proportionally to their frame pose change (Figure 4.3). Let p_k be the map-point to be corrected. Let $\Pi^{\mathbf{T}_i}(p_k)$ be the projection of map-point p_k in pose $\mathbf{T}_i$ and $\Omega^{\mathbf{T}_i}(\mathbf{F}_{i,h}, d)$ be the unprojected feature $\mathbf{F}_{i,h} \in \mathbb{R}^2$ from $\mathbf{T}_i \in V$ at a distance $d \in \mathbb{R}$. Let v_i be a frame, $\mathbf{T}_i$ be its old pose, and $\mathbf{T}_i^{\text{new}}$ be its new pose. The generalized map deform the 3D map-points using the following $\text{deform} : P, V \to \mathbb{R}^3$ function:

$$\text{deform}(p_k, v_i) = \Omega^{\mathbf{T}_i^{\text{new}}}(\Pi^{\mathbf{T}_i}(\mathbf{t}_k)_{x,y}, \Pi^{\mathbf{T}_i}(\mathbf{t}_k)_z). \tag{4.8}$$

Let $M_k = \{v_i | (v_i, p_k) \in S\}$ be the list of frames that see a point p_k. Then, the new coordinate of the point $\mathbf{t}_k^{\text{new}}$ is:

$$\mathbf{t}_k^{\text{new}} = \frac{\sum^{M_k} \text{deform}(p_k, v_i)}{\text{card}(M_k)}. \tag{4.9}$$

The deform function results the new position of a 3D point p_k, when its reference camera v_i move, according to the distance between the 3D point and the reference point, in the way of an inverse depth parameterization [93]. Equation 4.9 moves the position of a 3D point p_k, when one camera $p_k \in M_k$ move, according to the distance between the 3D point and all the camera which see this 3D point, by averaging the function deform.

When an indirect algorithm moves the poses, this transformation makes each projection (on each frame it appears) of a direct point closer to its old re-projection. The generalized map photometrically optimizes each map-points to ensure their accuracy after the deformation.

The generalized map has a way of keeping its map-points coherent. The following section talks about efficiently implementing these theoretical notions.

[1] In that case, the Hessian is a block diagonal matrix

4.3 Efficient Algorithms for the Generalized Map

The previous chapter showed how SLAM pipelines of the literature are composed of three main blocks: the pose estimation block, the mapping block, and the bundle adjustment block. Each block must constantly manipulate the map using its operations to retrieve and store information. It is important to reduce the complexity of those operations. Among these operations:

- The pose estimation block, executed for every frame of the video, has to track corners among the image by making correspondences, and for that, retrieve and store features in $\mathbf{F}_i$, retrieve the association between features and map-points S_i from a frame v_i, and retrieve the map-points with the active tag. It needs to add and remove tags on a few frames and map-points.
- The mapping block, executed for a small subset of frames, has to retrieve all the features in $\mathbf{F}_i$, associate map-points with feature points, and list the map-points with the immature tag.
- The bundle adjustment block, executed for a small subset of frames, lists all the co-visible frames of a frame v_k and all map-points seen by these frames.

Thus, pose estimation strongly impacts the performance of the SLAM, whereas bundle adjustment has less impact on performance. The pose estimation part mostly reads the map and rarely writes on it. The mapping part reads and writes the map. During the execution of the method MOD SLAM (explained in the following chapter) on KITTI and TUM, we measured an average of 9×10^8 read operations and 3×10^7 write operations. The reading operation of the map impacts more the execution time of a method than the writing operation. That is why the generalized map is optimized for the read operation rather than for the write operation.

This section aims to present efficient algorithms for the generalized map algorithms. We demonstrate that these algorithms minimize the impact of the map on the SLAM execution time, particularly on ORB-SLAM and DSO. We start by showing a method to store the map graph in RAM (Random-access memory) to minimize its impact on the execution time of the generalized map algorithms. Then we develop each operation on the map of each block and demonstrate their low impact on the execution time.

We define an *object* as a combination of variables, functions, and data structures equivalent to an instance of a class in oriented programming languages such as C++ and Java. In the following, we consider the defined generalized map, frame, and map-

points as classes that can be instantiated as objects. We define the complexity of a hash-structure as the complexity of finding the bucket place (defined in the following section) corresponding to an element, existing or not. We define the efficiency of an algorithm as the quality of this algorithm being fast (in terms of execution time).

4.3.1 Efficient Graph Structure

The generalized map algorithms' complexity depends on the structure used to store the generalized map graph. That is why we first present how to store the map efficiently.

The most common way to store a graph is by using an adjacency matrix. This would mean an adjacency containing millions of elements squared, which is too large. Hence, this method is not possible.

To store the graph, the generalized map uses hash-set and hash-map. A hash-set structure allows fast knowledge of whether an element belongs to it. A hash-map allows the storage of information within the edges (the hash-map value is the additional information). Frame and map-points structures contain hash-set to store edges and link to other elements and hash-map for edges with weights.

Hash-structures store their values in an array. The location of the value in the array is computed using a hashing function hash, which strongly impacts its speed. Most hash-structures use a function such as $\mathrm{index}(x) = \mathrm{hash}(x) \bmod n$ to choose where to store a value within the hash-structure. When two objects x and y have the same index ($\mathrm{index}(x) = \mathrm{index}(y)$), this induces a collision inside the hash-map. For an algorithm to retrieve an object x in the hash-structures, it has to iterate and compare all objects that collide with x. Hence, collisions increase the computation time during hash-structure manipulation. The default hash of the C++ STL (Standard Template Library) computes the hashes of objects with their pointer as a "random function" [95]. The STL hash functions cause two problems. First, our generalized map implementation allocates the frames and map-points inside a *pool allocator*. The list of pointers is a list of incremental numbers, where the increment is the size of the structure to hash. This leads to inefficiency of the hashing function index, and to randomness within the program[2]. Therefore, we need to define hash-structures and hash functions for the different types of objects stored within the map.

[2] In each execution, the values of the pointers differ. Consequently, the hash-map store the object in different order inside the hash.

The generalized map comprises two types of edges: frame-to-frame edges and frame-to-map-point edges. We propose two hash-structures and hash functions for these two types of edges.

Naive Approach for Hash Containing few Frames

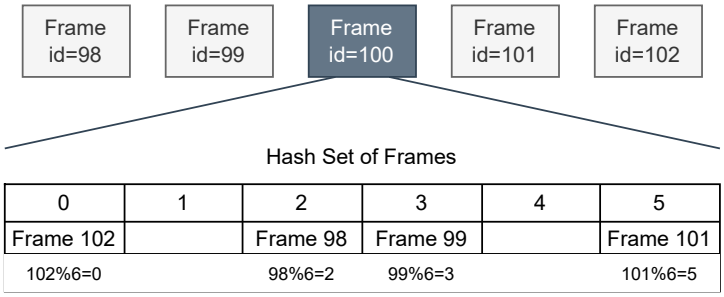


Figure 4.4: Naive Hash Frame Structure. Inside a naïve hash frame structure, it is highly improbable that two frames collide because a frame is linked to its neighborhood. Therefore, the complexity of this hash-map is $O(1)$.

A *naïve dense hash-structure* is made up of a large array of size l, named *bucket*. Each element e is stored in the case $hash(e) \bmod l$ of the bucket. When two elements collide, the hash-structure stores these two elements in the same case of the bucket array. Let η be the number of elements within a dense hash-structure. A bad hash function can cause the hash-structure to store all its elements in the same case, making the hash-structure $O(\eta)$. However, a good hash function makes a dense hash-map $O(\frac{\eta}{l})$.

Each frame and map-point have an associated ID: $v_{\text{id}} \in \mathbb{N}$ and $p_{\text{id}} \in \mathbb{N}$. This ID is a unique incremental. We define the hash function of any object a as $hash(a)$. A frame is always linked to other close frames in time and rarely linked to distance frames in time. Consequently, the figure 4.4 shows that the index function is efficient for the frames. The plot of Figure 4.5 shows an almost constant time, whatever the number of elements inside the hash-map.

Hash-Structures of Map-points

Map-points are more likely to produce random index than frames. That is why the generalized map uses a different hash-structure for the map-points. Hash-structures of the literature work by creating a tree with each branch on or multiple bits of the hash value. They work faster when the hash of each object has the most different bits. To achieve

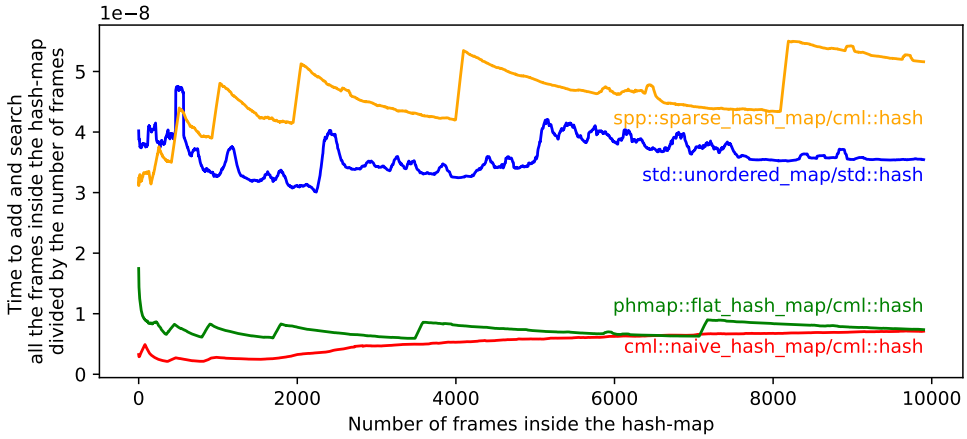


Figure 4.5: Performance of multiple hash-maps to store frames. This graph compares the performance of the Phmap hash-structure [96], the Sparse Spp hash-structure [97], the C++ hash-structure [98], and our own naïve hash-structure, for storing frame. We did our tests on a single core of an i9.

this, we use a method named *Integer hashing* [99, 100]. Each map-point has a unique incremental ID. We define the hash of a map-point as $\text{hash}(m_k) = \text{integerHashing}(k)$ where integerHashing is a function from $\mathbb{N}$ to $\mathbb{Z}_2^*$, such that each bit of the result of $integerHashing(k+1)$ has a 50% chance of being different from $\text{integerHashing}(k)$.

The performance of the hash-structures of the literature with map-points as keys was equivalent to their performance with frames as keys. The flat hash map implementation of Phmap [96] performed the best in our test for the map-points.

From these two hash-structures, the following section introduces efficient algorithms and demonstrates the low complexity of our generalized map implementation.

4.3.2 Efficient Algorithms

This section aims to introduce efficient algorithms for the generalized map. The complexity described in Table 4.1, emerges directly from the complexity of hash-structures of frames and map-points. Next, we present a detailed description of the main algorithms of the map. Inside the text, "(a), (b), (c), ..." letters refer to multiple summaries of the algorithm complexities put in Table 4.2.

Our generalized map implementation is based on an observer pattern [101]. The observer pattern allows one object (named the listener) to trigger a method when another object is modified.

Algorithms	Complexity
Test if $v_i \in A$ with $v_i \in V$ and $A \subseteq V$	$O(1)$
Test if $p_k \in A$ with $p_k \in P$, $A \subseteq P$, and $\eta = \text{card}(A)$	$O(\log \eta)$
Retrieve all the features $\mathbf{f}_{i,j}$ in $\mathbf{F}_i$, and their associated map-points and descriptors. The features are stored inside arrays. Their optional descriptors are stored inside arrays in the private data structure.	$O(1)$

Table 4.1: Algorithm directly coming from the complexity of the hash-structure.

	Algorithms	Complexity
(a)	**List all the frames and map-points belonging to tag.**	$O(1)$
(b)	**List all the map-points associated with a frame and belonging to tag.**	$O(1)$
(c)	**Add and remove tag from a map-point.**	$O(\eta_i \log \eta_j)$
(c)	**Add and remove a frame from a tag.**	$O(1)$
(d)	**Retrieve the associated features of a map-point in a frame** S_i. A hash-map of map-points has a read and write complexity of $O(\log \eta)$ where η is the number of map-points inside the hash-map.	$O(\log \eta)$
(d)	**Get the frames and the features associated with a map-point, or know if a map-point appears on a frame**. The map-points stores a list of frames that see it, using a hash-map, where the key is the frame, and the value is the feature on the frame.	$O(1)$
(d)	**Retrieve the associated feature of a map-point.**	$O(1)$
(e)	**Retrieve the co-visibility frames of a frame** v_k. Where η is the number of co-visible frames of the frame v_k.	$O(\eta \ \log \ \eta)$
(f)	**To associate a map-point with a feature point**, this point needs to be stored inside the existing lists in $O(1)$ and hash-structure in $O(\log \ \eta_i)$ with η_i the number of map-points inside the frame. After storing the association, the generalized map must update the co-visibility graph. This co-visibility graph is updated in $O(\eta_j)$, where η_j is the number of frames that see the map-point.	$O(log \ \eta_i + \eta_j)$
(g)	**To list all the co-visible map-points of a certain group between two frames**, the generalized map needs to test if all the map-points of the first frame belong to the second frame. This is done in $O(\eta_i log \ \eta_j)$, where η_i and η_j are the numbers of points of the specified group in the two frames.	$O(\eta_i log \ \eta_j)$

Table 4.2: Complexity of the main algorithms of the generalized map.

(a) The map object listens to frame objects and map-points objects tag change. It stores one hash-set for each tag and updates them each time a frame object or a map-point object changes its tag. Therefore, the complexity of listing all the frames and map-points having a specific tag is $O(1)$.

(b) Frame objects listen to multiple map-point objects and add their reference to a set when their tags are changed. Hence, the generalized map precomputes the sets of map-points. The method allowing to obtain all the map points of a frame belonging to a group has a complexity of $O(1)$.

(c) Each time a tag is added or removed from a frame or map-point, the algorithm needs to update hash-structures. The complexity of modifying one object tag is the complexity of modifying a hash-structure. Hence, the complexity of adding or removing a tag from a frame is $O(1)$. The complexity of adding or removing a tag from a map-point is $O(\eta_i \log \eta_j)$, with η_i the number of frames associated with the map-point and η_j the average number of map-points inside the hash-structures to update.

(d) When associating a map-point to a feature in a frame, the reference of the map-point is stored into two hash-structures, from map-points to features, and from features to the map-point. Moreover, the map-points class contains a hash-set of frames with which it is associated. Therefore, the complexity of knowing if a map-point belongs to a frame or the associated features of a map-point is $O(\log \eta)$. The complexity of knowing if a frame is associated with a map-point is $O(1)$, and the associated map-point of a feature is $O(1)$.

(e) Using the observer pattern, our generalized map updates the co-visibility graph each time the map is modified. Each frame contains a hash-map, with frames as keys and numbers of points as values. As it is a hash-map between frames, the writing complexity on this hash-map is $O(1)$. The update must be executed on all the co-visible frames, making the update complexity $O(\eta)$ where η is the number of co-visible frames. Since the generalized map stores the entire co-vsibility graph, the read complexity is $O(1)$. However, most of the time, the co-visible frames must be sorted and filtered, making the complexity $O(\eta \ \log \ \eta)$.

(f) Each time a map-point is associated with a frame, methods are triggered to update the co-visibility graph. Thus, the complexity of associating a map-point with a frame is $O(log\ \eta_i + \eta_j)$, with η_i the number of map-points inside the frame, and η_j the number of apparitions in frames of the map-point to associate.

(g) To find the co-visible map-points between 2 frames, the generalized map lists all the points of one frame and looks if it belongs to the other frame (we do this in the other

direction by looking if the frame belongs to the point in the code). We showed that the complexity of knowing if a frame belongs to a hash-set of hash-map was $O(1)$. Therefore, the complexity is the number of points in the frame multiplied by the looking complexity inside each point.

As shown in Table 4.2, the techniques above use more computational resources when the map is modified and uses more memory but has a low read complexity. Duplicating some hash-map for each tag permits all classes of the generalized map to have functions restricted to one tag with decreased complexity compared with the original function. This allows algorithms to avoid useless iterations and reduce their complexity.

In real execution on the KITTI dataset, we measured the average execution time of each operation of the generalized map. The plot in Figure 4.6 compares the average execution time of each operation of the generalized map with the number of calls of these operations by the SLAM. It shows that the more a function is called, the more efficient it is.

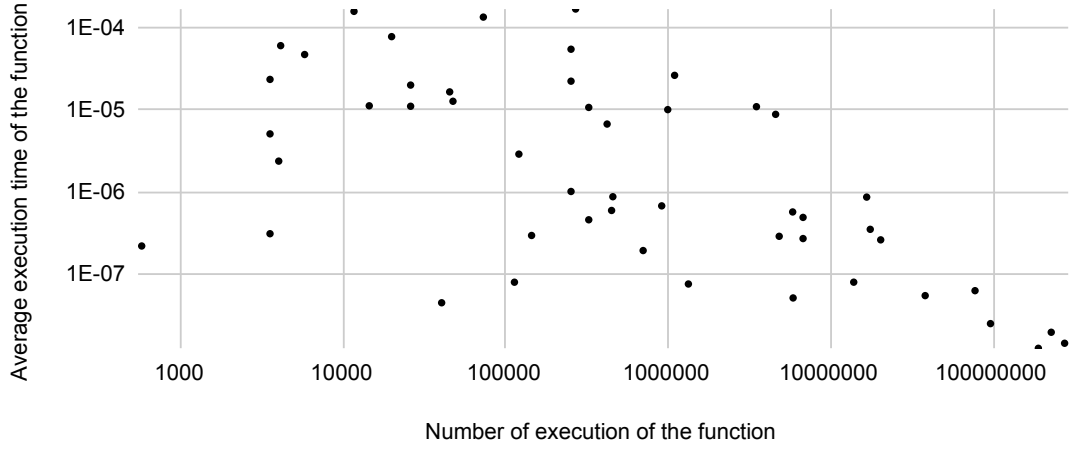


Figure 4.6: Number of calls of the methods of the map versus their execution time in second (Log scale). The most a map function is executed, the more efficient it is.

4.3.3 Discussion

Comparaison of the generalized map It is difficult to compare the generalized map with the map of the literature. No metrics have been made to compare the map. Moreover, the generalized map has to deal with the tag system, and higher amount of data, compared to standard map. Compared to the map of ORB-SLAM and DSO, the complexity of the reading operation of the generalized map are equal or lower than those of ORB-SLAM and DSO. This is especially the case for the reading of the co-visibility graph. On the contrary, some writing operations have higher complexity. For example, when associating a map-point with a frame, the generalized map has to update and cache the co-visibility graph. We will show in Chapter 6.3 that the generalized map is efficient enough to allow SLAM method with a high amount of data to run in real-time on smartphone.

Computation of the complexities The complexities have been computed on a model of computation (multitape Turing machine), which does not define all the operations that today's processors can execute. This model does not consider the different mechanisms of the current machine, such as the processor cache. That is why, algorithms with higher complexity can run faster than algorithms with lower complexity. This is the case of the phmap[96] hash-structures. This hash-structure uses AVX (Advanced Vector Extensions) and SSE (Streaming SIMD Extensions) instructions from the processor. Consequently, its complexity is $O(log\ \eta)$ in the multitape Turing machine model but appears $O(1)$ in the tests.

We optimized the generalized map to benefit from the cache and the AVX/SSE instruction set of the processor. We tested the ability of our algorithm to benefit from the cache on multiple processors.

4.4 Conclusion

We created a generalized map responding to four main criteria:

1. We defined a unified formalism for the generalized map, allowing to mix direct and indirect information in the same graph. We created a "tag" system, allowing each algorithm to work on a sub-graph of the generalized map. The generalized map is able to make algorithms of different families work together through function of an interface that permits to manipulate the map safely.

2. These functions contain mechanisms ensuring the coherence of the information. The generalized map deforms the map to ensure the global convergence of the direct algorithm: when a indirect algorithm modifies the pose of the camera, the direct point re-projection stays in the same place.
3. As the generalized map has to handle a lot of information, we defined hash structure and function to store the generalized map. We introduced a function that permits the efficient manipulation of the map.

The generalized map act as a framework. It allows the implementation of SLAM methods that are indirect, direct, or both. It enables the implementation of "dataset driver", which allows the program to read a dataset and camera device, and send its data to the SLAM. It allows the creation of applications that can be plugged into any SLAM and dataset, as we are going to see on page 118 of Chapter 6.3. The next chapter talks about the creation of a hybrid SLAM based on the generalized map.

The code source of the generalized map is available in our GitHub: `https://github.com/belosthomas/libCML`. Compilation options allow switching hash-map implementation.

5

MOD SLAM: Mixed SLAM Pipeline

We observed that the advantages and drawbacks of direct and indirect methods are complementary. Direct methods are robust to untextured images and give high-precision results rapidly on low-optical flow videos but struggle with high-optical flow videos. In contrast, indirect methods work well on high-optical-flow videos but struggle with untextured images. On this basis, we created and implemented a method named MOD SLAM for Mixed ORB and DSO SLAM, which benefits from both direct and indirect worlds. We define a *mixed SLAM pipeline* as a hybrid method that uses both direct and indirect approaches on a single thread. MOD SLAM is implemented on top of the generalized map (see Chapter 4), which handles the coherence and consistency of the information between ORB algorithms and DSO algorithms. Moreover, to mix direct and indirect algorithms, MOD SLAM integrates multiple decision systems, each making MOD SLAM choose a direct or indirect path. Each decision on the system was designed to provide the best results at the different steps. The MOD SLAM method is presented in the following order:

1. We develop the MOD SLAM pipeline in detail. We explain how the MOD SLAM pipelines alternate direct and indirect algorithms, whether in the tracking block, or in the mapping and bundle adjustment block.
2. We decompose the decision system. MOD SLAM uses a decision system to predict which of the direct or indirect algorithms is more likely to give better results.
3. In the next chapter, we show the result we obtained on both a desktop computer and an Android phone, and we study the impact of the decision system.

The MOD SLAM pipeline follows the general pipeline presented in Chapter 2. In a separate thread, MOD SLAM reads, parses, and pre-processes the video stored on the hard drive or issued by the camera. Each image is sent to the main thread via a queue system. The main thread receives the pre-processed image, extracts ORB feature points from the image, and executes pose estimation. The pose estimation starts with a decision

to use either a direct or indirect algorithm. The right pose estimation algorithm is run according to the decision and estimates the pose of the image. Then, it decides whether to create a direct keyframe or an indirect keyframe, and which bundle adjustment to execute. MOD SLAM runs the keyframe creation process and, optionally, the bundle adjustment according to the three decisions. In the following, we develop each step of the MOD SLAM pipeline and the decision system.

Many algorithms presented below come originally from the source codes of ORB-SLAM [14] and DSO [9]. We adapted them for the generalized map, and modified them for the MOD SLAM method.

5.1 Mixed Pose Estimation

The pose estimation pipeline is decomposed into two sub-pipelines, presented in Figure 5.1. The path at the top of the figure is an indirect-based pose estimation refined with direct points. The path at the bottom of the figure is a direct-based pose estimation, optionally refined with an indirect algorithm in the local map tracking block. The pipeline ends by tracking the local map. Next, we present the direct-based pose estimation, then the indirect-based pose estimation, followed by the local map tracking.

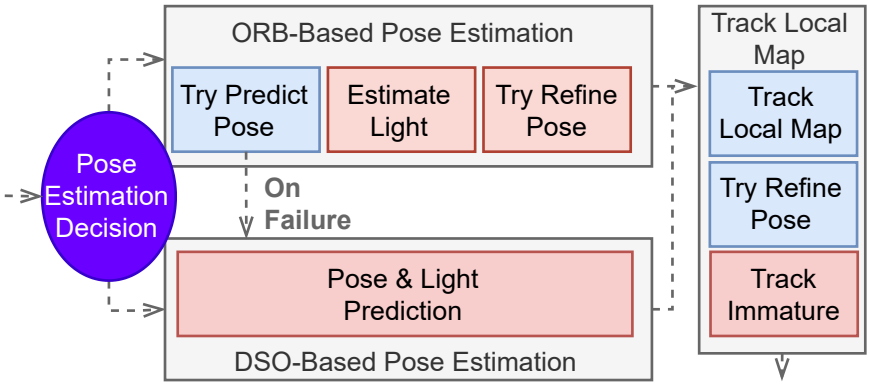


Figure 5.1: MOD SLAM Pose Estimation Pipeline. The pose estimation algorithm can take two paths: estimate with a feature-based algorithm refined with photometric algorithm (top path), or estimate with a photometric algorithm refined with feature-based algorithm (bottom path).

5.1.1 Direct-Based Pose Estimation

The direct-based pose estimation presented in Algorithm 1 runs a photometric optimization following the procedure of DSO [9]. It initializes the state of the optimization $\boldsymbol{\beta} \in \mathbb{R}^8$ with a given pose guess of $\mathbf{T}_i$ and light parameters guess of $\mathbf{l}_i$. The COMPUTEJACOBIANANDRESIDUALS function computes the gradient vector $\mathbf{g}$ and the residual r of the sum of the photometric error of all active direct map-points. MOD SLAM classifies as outliers map-points with a photometric error higher than a threshold τ. τ is computed at line 6. It is computed such that at least 60% [1] of the map-points are classified inliers. The COMPUTEHESSIAN function computes the Hessian $\mathbf{H}$ of the photometric error function.

Photometric optimization is extremely sensitive to high optical flow. The direct-based pose estimation takes two precautions to reduce the risk of local convergence. Firstly, the optimization of Algorithm 1 is executed multiple times at different scales of the image. The width and height of the image are divided by $2^{\text{imageLevel}}$. The algorithm starts with a higher image level to make the image smaller and, consequently, smaller optical flow distance in pixels. At line 3, the algorithm starts at image level 4. It progressively decreases the image level until the original image size is reached (at level 0). Secondly, the direct pose estimation algorithm generates 50 guesses from the previously estimated motion by perturbating the rotation and the translation of this motion. The last motion is computed as $\text{lastMotion} = \mathbf{T}_{i-1} - \mathbf{T}_{i-2}$. The guesses are generated following this model: $\{\text{lastMotion}, \text{lastMotion} \div 2, \text{lastMotion} \times 2, \text{lastMotion} + \text{randomRotations}\}$ where randomRotation is a set of a small rotation transformations. The Algorithm 1 is executed a maximum of 50 times with all the guesses. The algorithm returns the guess with the lower photometric error. If one guess has an error lower than the last photometric error multiplied by 1.5, the latter is accepted without testing the other guesses.

At the end, the last Hessian is saved at line 26 to compute the uncertainty of the pose. MOD SLAM uses this uncertainty during the pose estimation decision.

The photometric Algorithm 1 is extremely fast. The Hessian and Jacobian computation highly benefits from vectorized CPU instructions. 50 photometric trials are faster than one feature-based optimization, that we expain in the coming section.

[1] Some parameters values are hard-coded inside the original code of ORB-SLAM and DSO. As these parameters have optimal values, MOD SLAM does not modify them.

Algorithm 1 Photometric Minimization of the pose and the light

Input:	v_i	A frame v_i.
	$\mathbf{T}_i$	A guess for $\mathbf{T}_i$ for the pose of v_i.
	$\mathbf{l}_i$	A guess for $\mathbf{l}_i$ for the light of v_i.
Output:	$\mathbf{T}_i$	The optimized pose $\mathbf{T}_i$
	$\mathbf{l}_i$	The optimized light $\mathbf{l}_i$
	$\mathbf{H}_i^{\text{direct}}$	The last computed Hessian

$\boldsymbol{\beta} \leftarrow \{\mathbf{T}_i, \mathbf{l}_i\}$
$\varrho \leftarrow \{p_k, p_k \in S_{v_i}, \text{direct} \in C_k^p\}$
for imageLevel $\in \{4, 3, 2, 1, 0\}$ **do**
  $\tau \leftarrow 1$
  $\mathbf{g}, \mathbf{e} \leftarrow$ COMPUTEJACOBIANANDRESIDUALS$(v_i, \varrho, \boldsymbol{\beta}, \text{imageLevel}, \tau \times 20)$
  while $\frac{\text{card}(\mathbf{e})}{\text{card}(\varrho)} < 0.6 \wedge \tau < 50$ **do** ▷ $\frac{\text{card}(\mathbf{e})}{\text{card}(\varrho)}$ is the inliers ratio
    $\tau \leftarrow \tau \times 2$
    $\mathbf{g}, \mathbf{e} \leftarrow$ COMPUTEJACOBIANANDRESIDUALS$(\boldsymbol{\beta}, \text{imageLevel}, \tau \times 20)$
  end while
  $\mathbf{H} \leftarrow$ COMPUTEHESSIAN$(\boldsymbol{\beta}, \mathbf{g}, \mathbf{e})$
  $\lambda \leftarrow 0.01$
  for 20 iterations **do**
    Solve $\delta\boldsymbol{\beta}$ such $(\mathbf{H} + \lambda I)(\boldsymbol{\beta} + \delta\boldsymbol{\beta}) = \mathbf{g}$
    $\boldsymbol{\beta}_{\text{new}} = \boldsymbol{\beta} + \delta\boldsymbol{\beta}$
    $\mathbf{g}_{\text{new}}, \mathbf{e}_{\text{new}} \leftarrow$ COMPUTEJACOBIANANDRESIDUALS$(\boldsymbol{\beta}_{\text{new}}, \text{imageLevel}, \tau \times 20)$
    if $\mathbf{e}_{\text{new}} < \mathbf{e}$ **then**
      $\boldsymbol{\beta}, \mathbf{g}, \mathbf{e} \leftarrow \boldsymbol{\beta}_{\text{new}}, \mathbf{g}_{\text{new}}, \mathbf{e}_{\text{new}}$
      $\lambda \leftarrow \lambda \div 2$
      $\mathbf{H} \leftarrow$ COMPUTEHESSIAN$(\boldsymbol{\beta}, \mathbf{g}, \mathbf{e})$
    else
      $\lambda \leftarrow \lambda \times 4$
    end if
  end for
end for
$\{\mathbf{T}_i, \mathbf{l}_i\} \leftarrow \boldsymbol{\beta}$
$\mathbf{H}_i^{\text{direct}} \leftarrow \mathbf{H}$

5.1.2 Indirect-Based Pose Estimation

The Indirect-Based Pose Estimation (top path of Figure 5.1) is an ORB-SLAM-like pose estimation refined with direct points. In case of failure, it switches to a DSO-Based Pose Estimation (corresponding to the arrow from the top path to the bottom path). When the video has a high optical flow (such as a low FPS video or high-speed movement), it is better to use a feature-based optimization, as photometric optimization has an increased risk of converging to a local minimum. Since the feature-based optimization does not converge entirely to the solution[2] MOD SLAM makes a photometric refinement using Algorithm 1. This photometric refinement permits MOD SLAM to have an estimate of the light ($\mathbf{l}_i^a$ and $\mathbf{l}_i^b$).

The indirect-based pose estimation of MOD SLAM is similar to the pose estimation of the ORB-SLAM pipeline. MOD SLAM executes two trials: first, assuming that the camera moves at a uniform velocity; secondly, by matching 2D features with a reference key-frame.

First trial: Constant Velocity Motion Model

Most of the time, the camera moves with a uniform velocity in the same direction. That is why MOD SLAM tries to estimate the pose by reprojecting 3D active indirect points on an assumed pose. MOD SLAM initializes the pose $\mathbf{T}_i$ of the frame v_i from the previous poses by assuming a constant velocity: $\mathbf{T}_i = \mathbf{T}_{i-1} + (\mathbf{T}_{i-1} - \mathbf{T}_{i-2})$, where $\mathbf{T}_{i-1} - \mathbf{T}_{i-2}$ is the last velocity.

First trial: Creation of 2D-3D matches

From the hypothetical $\mathbf{T}_i$, MOD SLAM finds 2D-3D matches and stores them in $A(v_i)$ by re-projecting the previous tracked points of v_{i-1} into the assumed pose $\mathbf{T}_i$ (Algorithm 2). To be more precise, each 3D indirect map-points matched with the previous frame v_{i-1} is reprojected to the frame v_i. MOD SLAM makes candidates by looking for features around the reprojection. If the descriptor of one feature is near the descriptor of the 3D point, a 2D-3D matching is added in $A(v_i)$.

At line 2, the NEARESTNEIGHBOR algorithm computes the nearest neighbor using FLANN library [102], and a pre-computed KD-Tree of the features by the generalized map. This function takes three arguments: a 2D point, the set of 2D candidates, and a

[2] due to the slow convergence properties of optimization using features distance

radius τ. During the ORB explanation, Chapter 2 showed that ORB features are FAST features extracted at different image scales. The level of features varies proportionally with the feature distance to the camera. At line 5, the function LVL returns a feature level. Comparing the level of the features reduces the number of candidates, the ambiguity, and increases the number of correspondences.

The DESCRIPTORDISTANCE function computes the Hamming distance between the BRIEF-like descriptors of the ORB features. By using the special `popcount` processor instruction, the Hamming distance between two features is executed in 8 processor cycles: `__builtin_popcountll(binary64_a[i / 8] ^ binary64_b[i / 8])`.

The threshold τ of the algorithm is set to 7. However, if the number of found matches is lower than 20, the algorithm is re-executed with $\tau = 14$. If the matching number is still lower than 20, the first trial is considered as a failure, and the algorithm switches to the second trial.

Algorithm 2 Matching by reprojection

Input:	$\mathbf{F}_i$	The set of features $\mathbf{F}_i$ associated with its descriptors.
	$\mathbf{F}_{i-1}$	The set of features $\mathbf{F}_{i-1}$ associated with its descriptors.
	$\mathbf{A}_{i-1}$	A set of 2D-3D matchings for v_{i-1}
	τ	Distance in pixels for the nearest neighbor.
Output:	$\mathbf{A}_i$	A set of 2D-3D matchings for v_i

1: **for each** $\mathbf{t}_k, \exists \mathbf{t}_k \in A(i-1, g), g \in \mathbb{N}$ **do** ▷ For each map-point on the previous frame
2: nns $\leftarrow$ NEARESTNEIGHBOR$(\Pi^{\mathbf{T}_i}(\mathbf{t}_k), \mathbf{F}_i, \tau)$
3: $h_{\text{best}}, d_{\text{best}} \leftarrow \emptyset, \infty$
4: **for each** $h \in$ nns, $\nexists A(i, h)$ **do**
5: **if** LVL$(\mathbf{F}_{i,h}) =$ LVL$(\mathbf{F}_{i-1,g})$ and DESCRIPTORDISTANCE$(\mathbf{F}_{i,h}, \mathbf{F}_{i-1,g}) < d_{\text{best}}$ **then**
6: $h_{\text{best}} \leftarrow h$
7: $d_{\text{best}} \leftarrow$ DESCRIPTORDISTANCE$(\mathbf{F}_{i,h}, \mathbf{F}_{i-1,g})$
8: **end if**
9: **end for**
10: **if** $d_{\text{best}} < 50$ **then**
11: $A(i, h) \leftarrow \mathbf{p}_k$
12: **end if**
13: **end for**

First trial: Optimization of the pose

Thanks to the previous algorithm, MOD SLAM has 2D-3D matching, allowing to estimate $\mathbf{T}_i$. This optimization Algorithm 3 needs to take into account that these 2D-3D matching may contain many outliers. The optimization algorithm starts by marking all the 2D-3D matches as inliers. It starts a loop of four iterations, in which:

1. Ten Levenberg-Marquardt steps are executed with only the inliers. The updates are computed using the G^2O library [89], which computes only the Jacobian, and approximates the Hessian as $\mathbf{J}^\top\mathbf{J}$, with $\mathbf{J}$ the Jacobian.
2. After Levenberg-Marquardt iterations, the 2D-3D matches having a high error (when the distance between the re-projection and the corner is higher than 3 pixels) are marked as outliers, and the ones having a low error are marked as inliers (a match can become inlier again after having been marked as an outlier).
3. If, after the inliers/outliers classification, less than ten map-points remain, the optimization stops and signals a failure. In that case, MOD SLAM goes to the second trial.

The feature-based optimization algorithm of the pose returns an estimation of the pose, and the inliers classification of the 2D-3D matches. This classification is used for the quality measurement of the estimation, which tells whether the solution is accepted.

At the end of the algorithm, the last Hessian is stored for the computation of the uncertainty during the pose estimation decision.

First trial: Quality mesasurement of the estimation

During the indirect pose estimation pipeline, MOD SLAM has to decide if the indirect pose is acceptable. ORB-SLAM decides that a pose estimation is acceptable if there is more inliers than $\psi^{\text{orb}}_{\text{inlierNumThreshold}}$. MOD SLAM adds a second condition, which tests whether the ratio of inliers versus outliers is sufficiently high, using a threshold $\psi^{\text{orb}}_{\text{orbInlierRatioThreshold}}$. The parameters of ORB-SLAM judging the quality of the pose estimation are made more stringent, to decide if the indirect pose estimation passed. The original test has been made so that the SLAM can continue even if there is a failure. On the contrary, MOD SLAM has to switch to the direct-based pose estimation backup path in the case of a high uncertainty.

Algorithm 3 Feature-Based Optimization of the pose

Input:	$\mathbf{v}_i$	The frame to optimize
	$\mathbf{A}_i$	A set of 2D-3D matchings for v_i
Output:	$\mathbf{T}_i$	An estimate of $\mathbf{T}_i$
	$\mathbf{H}_i^{\text{indirect}}$	The last Hessian to compute the uncertainty of the pose

inliers $\leftarrow h, \exists A(v_i, h)$
for i from 0 to 2 **do**
    $\boldsymbol{\beta} \leftarrow \mathbf{T}_{i-1} + (\mathbf{T}_{i-1} - \mathbf{T}_{i-2})$
    OPTIMIZEWITHHUBER($\boldsymbol{\beta}, \{\{A(v_i, h)\}, h \in inlier\}$)
    inliers $\leftarrow h, \exists A(v_i, h)$, RESIDUAL($\boldsymbol{\beta}, A(v_i, h)) < \psi^{\text{orb}}_{\text{optimthreshold}}$
end for
for i from 0 to 2 **do**
    $\boldsymbol{\beta} \leftarrow \mathbf{T}_{i-1} + (\mathbf{T}_{i-1} - \mathbf{T}_{i-2})$
    OPTIMIZEWITHOUTHUBER($\boldsymbol{\beta}, \{\{A(v_i, h)\}, h \in inlier\}$)
    inliers $\leftarrow h, \exists A(v_i, h)$, RESIDUAL($\boldsymbol{\beta}, A(v_i, h)) < \psi^{\text{orb}}_{\text{optimthreshold}}$
end for
$\mathbf{T}_i \leftarrow \boldsymbol{\beta}$
$\mathbf{J} \leftarrow$ COMPUTEJACOBIAN($\boldsymbol{\beta}, \{\{A(v_i, h)\}, h \in inlier\}$)
$\mathbf{H}_i^{\text{indirect}} \leftarrow \mathbf{J}^\top \mathbf{J}$

Second trial: Pose Estimation with Reference Key-Frame

For a second trial, MOD SLAM tries a pose estimation algorithm based on the last key-frame. MOD SLAM computes 2D matches between features of the frame v_i and features of the last reference key-frame, that we denote v_r. The algorithm tries to obtain as many matches as possible, even if some outliers appear.

This matching algorithm shown in Algorithm 4 is similar to Algorithm 2, but is based on an LSH (Locality sensitive hashing) method supported by a bag of words. Due to its complexity, we choose to simplify the description of this algorithm.

This matching is followed by an estimation of the pose and a quality measurement, which are exactly the same as on the first trial.

Photometric Refinement

The re-projection optimization converges slowly near the solution. To speed up MOD SLAM, and to obtain a more precise pose estimate, the number of feature-based optimization iterations is limited. A photometric refinement is rather executed, which results in a precise pose. Algorithm 1 is executed to refine the pose optimized pose by the indirect algorithm. As the initial pose is near the solution, the optical flow problem is compensated, and the optimization is extremely fast.

Filtering

At the end of Algorithms 4 and 2, MOD SLAM executes an optional outlier filtering using the orientation of the extracted ORB features, as described by Algorithm 5. This algorithm states that, between two frames, the difference in rotation between the pair of features of the matches must remain the same. MOD SLAM uses a histogram to determine three valid rotations so it can filter the matches.

5.1.3 Pose Estimation Decision

Due to computation time constraints, it is not feasible to test both methods. The prediction needs to be made in advance from what has been computed from the previous frames.

The pose estimation decision comprises two conditions. First, if the number of ORB map-points tracked in the previous frame is below a threshold $\psi^{\text{pe}}_{\text{minorb}}$, MOD SAM decides to use a pose estimation refined with ORB. On a low-textured image, an indirect pose

Algorithm 4 Matching by Bag of Words

Input: $\mathbf{F}_i$ The set of features $\mathbf{F}_i$, each associated with an descriptor.
$\mathbf{F}_r$ The set of features $\mathbf{F}_r$, each associated with an descriptor.
$\mathbf{A}_r$ A set of 2D-3D matchings for v_r
Output: $\mathbf{A}_i$ A set of 2D-3D matchings for v_i

for each $\mathbf{t}_k, \exists \mathbf{t}_k \in A(r,g), g \in \mathbb{N}$ **do**
  Candidates $\leftarrow$ GETCANDIDATESWITHBOW($\mathbf{F}_i$)
  for each $h \in$ Candidates, $\nexists A(i,h)$ **do** ▷ Compute best candidate
    if DESCRIPTORDISTANCE($\mathbf{F}_{i,h}, \mathbf{F}_{i-1,g}$) $< d_{\text{best}}$ **then**
      $h_{\text{best}} \leftarrow h$
      $d_{\text{best}} \leftarrow$ DESCRIPTORDISTANCE($\mathbf{F}_{i,h}, \mathbf{F}_{i-1,g}$)
    end if
  end for
  if $d_{\text{best}} < 50$ **then** ▷ If best candidate has a good score
    $A(i,h) \leftarrow \mathbf{p}_k$ ▷ Set the matching
  end if
end for

Algorithm 5 Matching Filtering with ORB rotation

Input: matchings A set of 2D-2D ORB matchings.
Output: filtered The filtered set of 2D-2D matchings.

histogram $\leftarrow \{0, 0, 0, ...\} \in \mathbb{R}^{12}$
for each $\mathbf{f}_a, \mathbf{f}_b \in$ matchings **do**
  rotation $\leftarrow$ ANGLEOF($\mathbf{f}_a$) $-$ ANGLEOF($\mathbf{f}_b$)
  if rotation < 0 **then**
    rotation $\leftarrow$ rotation $+ 360$
  end if
  bin $\leftarrow$ round(rotation $\div 30$)
  histogram$_{\text{bin}}$ = histogram$_{\text{bin}} + 1$
end for
threeMaxima $\leftarrow$ COMPUTETHREEMAXIMA(histogram)
filtered $\leftarrow \emptyset$
for each $\mathbf{f}_a, \mathbf{f}_b \in$ matchings **do**
  rotation $\leftarrow$ ANGLEOF($\mathbf{f}_a$) $-$ ANGLEOF($\mathbf{f}_b$)
  if rotation < 0 **then**
    rotation $\leftarrow$ rotation $+ 360$
  end if
  bin $\leftarrow$ round(rotation $\div 30$)
  if bin $\in$ threeMaxima **then**
    filtered $\leftarrow$ {filtered, $\{\mathbf{f}_a, \mathbf{f}_b\}$}
  end if
end for

estimation probably fails or takes time. As feature-based pose estimation is unadapted for low textured images, MOD SLAM rather uses a photometric pose estimation.

Secondly, MOD SLAM compares the covariance obtained by indirect and direct pose estimation algorithms. The covariance is a measure of the uncertainty during an optimization process [103]. The covariance matrices are the inverse Hessian of direct and indirect pose estimation algorithms, which are computed for each frame v_i in Algorithm 1 and 3: $(\mathbf{H}_{i-1}^{\text{direct}})^{-1}$ and $(\mathbf{H}_{i-1}^{\text{indirect}})^{-1}$. As indirect and direct methods have different representations of rotation, MOD SLAM crops the Hessian to use only the covariance of the poses translations. In the above formula, we assume that the Hessian matrices have already been cropped and are therefore 3×3 matrices. The uncertainties $u_{i-1}^{\text{direct}} \in \mathbb{R}$ and $u_{i-1}^{\text{indirect}} \in \mathbb{R}$ are defined as:

$$u^{\text{direct}} = trace((\mathbf{H}_{i-1}^{\text{direct}})^{-1}), \qquad u^{\text{indirect}} = trace((\mathbf{H}_{i-1}^{\text{indirect}})^{-1}). \tag{5.1}$$

MOD SLAM executes the direct and indirect pose estimations on the same map. As this map is coherent, an estimation from direct points should approximately be the same as with indirect points (as explained in Section 4.2). MOD SLAM computes both Hessian for a common pose v_i from the map-points of the same coherent map. Consequently, both Hessian are in the same scale, and uncertainties u^{direct} and u^{indirect} are in the scale map unit. Hence, it is mathematically meaningful to compare both uncertainties.

Let $\psi_{\text{coweight}}^{\text{pe}}$ be a weight that influences the decision. MOD SLAM chooses a direct or indirect pose estimation by comparing u^{direct} and u^{indirect}.

$$decision = \begin{cases} \text{indirect}, & \text{if } \|u^{\text{indirect}}\| < \psi_{\text{coweight}}^{\text{pe}} \times \|u^{\text{direct}}\| \\ \text{direct}, & \text{otherwise} \end{cases} \tag{5.2}$$

A value for $\psi_{\text{coweight}}^{\text{pe}}$ below 1 favors the direct pipeline decision, whereas a value above favors the indirect pipeline decision.

5.1.4 Local Map Tracking

After the pose estimation, MOD SLAM tracks the direct and indirect local map. Firstly, It chooses some recent map-points to tag as active. It re-projects these map-points into the current frame to add more 2D-3D correspondences in the set A. Then, it optionally runs an indirect refinement, including the new correspondences. Finally, MOD SLAM runs a

direct algorithm that tracks the newly created direct points. We develop each of these steps in the following.

Update Active Indirect Key-Frames and Map-Points

The first step of the local map tracking is to update the active indirect key-frames and map-points. MOD SLAM tags as active each frame that shares a map-point with the current frame v_i. Then, it tags as active each map-point seen by an active key-frame.

As a reminder, C_i^v contains the tags of a frame v_i, and C_j^p contains the tags of a map-point p_j.

Algorithm 6 Computation of Active Key-Frames

Input:	v_i	The current tracked frame.
Output:	$\{v_j, \text{active indirect} \in C_j^v\}$	New active indirect key-frames.
Output:	$\{p_k, \text{active indirect} \in C_k^p\}$	New active indirect map-points.

$C_i^v \leftarrow \{C_i^v, \text{active indirect}\}$
$\eta \leftarrow 1$
for each $p_k, \exists h, A(v_i, h) = p_k$ **do**
    for each $v_j, \exists m, A(v_j, m) = p_k$ **do**
        $C_j^v \leftarrow \{C_j^v, \text{active indirect}\}$
        $\eta \leftarrow \eta + 1$
    end for
end for
for each $v_j, \text{active indirect} \in C_j^v$ **do**
    if $\eta > 30$ **then**
        break
    end if
    for each $v_k, \exists \{v_j, v_k\} \in E, \text{active indirect} \notin C_k^v$ **do**
        $C_k^v \leftarrow \{C_k^v, \text{active indirect}\}$
        $\eta \leftarrow \eta + 1$
    end for
end for

Reprojection of Active Map-Points

MOD SLAM adds to S_i each active indirect 3D map-points in the viewing frustum (meaning MOD SLAM knows the point is visible on the frame, does not have any correspondence yet). Then, it re-projects each active indirect 3D map-points, as shown in Algorithm 7. The function COMPUTEVIEWCOSANDSCALE indicates whether a 3D map-points appears on the frame. It predicts at which ORB level the map-point should have been extracted. This predicted level is used by MOD SLAM to remove most 2D candidates for each point and therefore maximize the number of 2D-3D matchings.

Indirect Refinement

From the newly created 2D-3D matches, MOD SLAM optionally executes a feature-based optimization, from the Algorithm 3. This feature-based optimization can have two roles:

- In some rare cases, the photometric optimization may converge to a local minimum. To evaluate that, MOD SLAM looks at the number of points marked as outliers inside the photometric optimization. If the ratio is higher than a threshold $\psi^{\text{dso}}_{\text{poseRatioThreshold}}$, MOD SLAM launches a refinement of the estimated pose with an indirect algorithm.
- If, for the frame v_i, MOD SLAM did not run any feature-based optimization algorithm (for a pose estimation or a refinement), it executes a one iteration optimization to compute the Hessian $\mathbf{H}_i^{\text{indirect}}$ of the indirect pose based on indirect points.

Algorithm 7 Matching by reprojection

Input: $\mathbf{F}_i$ The set of features $\mathbf{F}_i$, each associated with an descriptor.
Output: $\mathbf{A}_i$ A set of 2D-3D matchings for v_i

for each $\mathbf{t}_k$, active indirect $\in C_k^p$ **do**
  result, viewCos, predLevel = COMPUTEVIEWCOSANDSCALE($\mathbf{t}_k$)
  if result is invalid **then**
    continue
  end if
  if $1 - \text{viewCos} \leq 0.002$ **then**
    $\tau \leftarrow 2.5$
  else
    $\tau \leftarrow 4.0$
  end if
  nn $\leftarrow$ nearestNeighbor($\Pi^{\mathbf{T}_i}(\mathbf{t}_k), \mathbf{F}_i, \tau$) ▷ All points within a radius τ
  $h_{\text{best}}, d_{\text{best}} \leftarrow \emptyset, \infty$
  $d_{\text{best2}}, h_{\text{best2}} \leftarrow \emptyset, \infty$
  for each $h \in \text{nn}, \nexists A(i,h)$ **do** ▷ Compute the best candidate
    if LVL($\mathbf{F}_{i,h}$) $\neq$ predLevel **then**
      continue
    end if
    if DESCRIPTORDISTANCE($\mathbf{F}_{i,h}, \mathbf{t}_k$) $< d_{\text{best}}$ **then**
      $h_{\text{best2}}, d_{\text{best2}} \leftarrow h_{\text{best}}, d_{\text{best}}$
      $h_{\text{best}} \leftarrow h$
      $d_{\text{best}} \leftarrow$ DESCRIPTORDISTANCE($\mathbf{F}_{i,h}, \mathbf{t}_k$)
    end if
  end for
  if $d_{\text{best}} < 50$ **then** ▷ If the best candidate has an distance lower than 50
    if LVL($\mathbf{F}_{i,h_{\text{best}}}$) = LVL($\mathbf{F}_{i,h_{\text{best2}}}$) and $d_{\text{best}} > \psi^{\text{orb}}_{\text{reprojRatio}} \times d_{\text{best2}}$ **then**
      continue
    end if
    $A(i,h) \leftarrow \mathbf{p}_k$ ▷ Set the matching
  end if
end for

Direct Immature Points Management: Initialization of 3D Points

We have shown in 3.2 that a photometric error function has many local minima.

Contrary to indirect map-point optimization algorithms, direct map-point optimization algorithms are sensitive to poorly initialized 3D positions. When a direct map-point is not initialized correctly, a photometric optimization of this map-point converges to a local minimum. We define an algorithm named "trace algorithm", by reusing the term inside the DSO source code. The goal of the trace algorithm is, for each newly created direct map-point, to find an interval (real interval for the inverse depth of the map-point) to initialize it. This allows the photometric optimization of these map-points to converge to local minima.

As the direct 3D map-points are expressed using an inverse depth relative to their reference pose, the trace algorithm has to reduce an interval of the inverse depth.

For that, the trace Algorithm 8 tracks the immature direct points among each image of the video to gradually reduce the depth interval of these points. Let $\mathbf{d}_k = \mathbf{d}_k^{\text{low}}, \mathbf{d}_k^{\text{high}}$ be the inverse depth interval of the point p_k. Firstly, this algorithm runs some prior checking on the immature point. The function OUTOFBOUND checks that the 2D projection of each 3D point on the interval in the frame v_i belongs to the image. If not, MOD SLAM is unable to deduce whether the map-point is on the image or not. In that case, it is safer to invalidate the map-point. The function INTERVALTOOSHORT checks that the distance between the projection of the minimum and maximum interval is not too short. In that case, it is useless to reduce the interval, as it is already short enough for further photometric optimization.

Next, the trace Algorithm 8 tries 100 inverse depth values in the interval of v_k. For each try, MOD SLAM computes a photometric error of v_k. It stores the global minimum in bestI (line 14), and the nearest local minimum around the global minimum in secondBestI (line 28).

The function REDUCEINTERVAL reduces the interval $\mathbf{d}_k$ around the found local minimum bestI.

Algorithm 8 Photometric Trace Algorithm

Input:	$\mathbf{v}_i$	The current tracked frames.
	p_k, direct immature $\in C_k^p$	All the direct immature map-points
Output:	p_k, direct immature $\in C_k^p$	The tracked immature points

for each p_k, direct immature $\in C_k^p$ **do**
  if OUTOFBOUND(p_k) **then**
    p_k is invalid
    continue
  end if
  if INTERVALTOOSHORT(p_k) **then**
    p_k is still valid
    continue
  end if
  bestEnergy, bestI $\leftarrow \infty, \infty$
  for $i \in 0..100$ **do**
    energy $\leftarrow$ PHOTOMETRICERRORIMMATURE(p_k, i)
    if energy $<$ bestEnergy **then**
      bestEnergy, bestI $\leftarrow$ energy, i
    end if
  end for
  if bestEnergy $> \psi^{\text{dso}}_{\text{energyThreshold}}$ **then**
    p_k is an outlier
    continue
  end if
  secondBestEnergy, secondBestI $\leftarrow \infty, \infty$
  for $i \in 0..100$ **do**
    if $i \geq$ bestI $- 2$ or $i \leq$ bestI $+ 2$ **then**
      continue
    end if
    energy $\leftarrow$ PHOTOMETRICERRORIMMATURE(p_k, i)
    if energy $<$ secondBestEnergy **then**
      secondBestEnergy, secondBestI $\leftarrow$ energy, i
    end if
  end for
  quality $\leftarrow$ secondBestEnergy $\div$ bestEnergy
  REDUCEINTERVAL($p_k, bestI$)
end for

5.2 Mapping

The mapping part of a SLAM has three functions: make key-frames, create new map-points required for future pose estimation, and refine the map. Figure 5.2 shows that MOD SLAM performs indirect and direct mapping separately, as an algorithm of one family cannot triangulate or refine the map points of the other family. Direct and indirect key-frames are also separated and have their own tags.

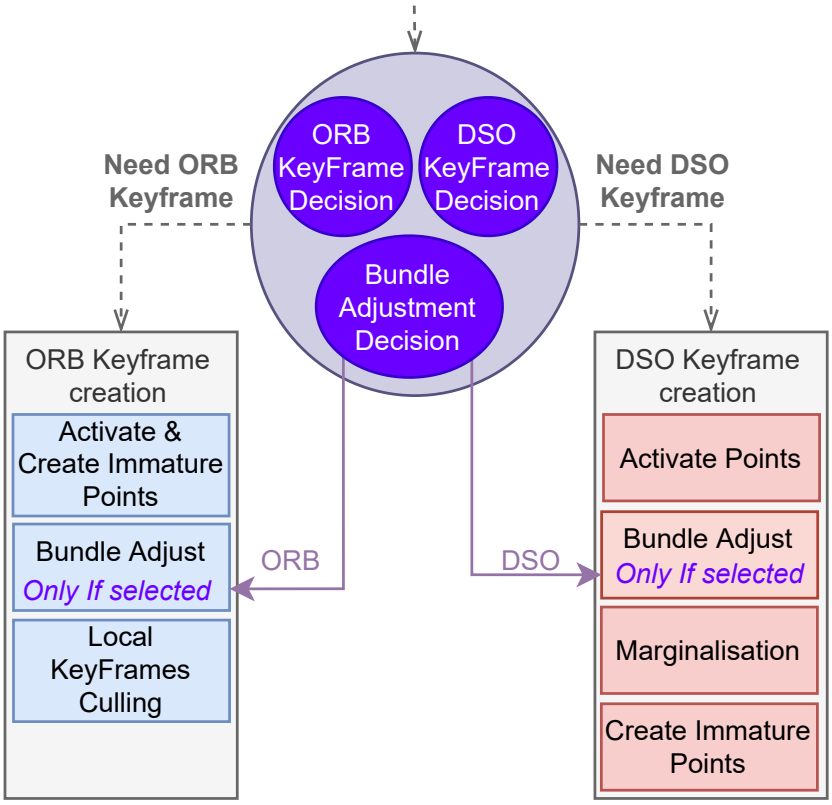


Figure 5.2: MOD SLAM Key-frame creation. At each frame, MOD SLAM can create an ORB key-frame and/or a DSO key-frame, and can only execute a direct bundle adjustment, an indirect bundle adjustment or no bundle adjustment.

5.2.1 Direct Key-Frame Creation

Direct Key-Frame Decision

The DSO Key-Frame Decision tells if MOD SLAM needs a new DSO key-frame. Two DSO processes are dependent on this decision. The direct pose estimation needs a reference frame not too far from the frame to track; otherwise, the optimization may diverge. The immature DSO points need to be activated not too late, or Algorithm 8 will invalidate them (line 3). Therefore, MOD SLAM needs to create direct key-frames when the optical flow from the last image to the last key-image becomes too high or when the direct pose optimization starts to give a high error:

- MOD SLAM estimates an optical flow from the last pose estimation, during Algorithm 1. If the optical flow is too high, it creates a direct key-frame.
- When the light change is too large, MOD SLAM creates a direct key-frame.
- If the residual of the photometric pose optimizer (Algorithm 1) suddenly becomes higher than the residual of the reference key-frame multiplied by a parameter $\psi^{\text{dso}}_{\text{kfResidualRatio}}$, MOD SLAM creates a new direct key-frame.

Direct Immature Points Management: Activation

The first step of the direct key-frame creation block is to activate new immature points, as shown in Algorithm 9.

Algorithm 9 Activation of Direct Immature Point

Input:	$\mathbf{v}_i$	The current tracked frames.
	p_k, direct immature $\in C_k^p$	All the direct immature map-points
Output:	p_k, active $\in C_k^p$	Activated direct map-points

1: allProjections $\leftarrow \Pi^{\mathbf{T}_i}(\mathbf{t}_k)$, active direct $\in C_k^p$
2: **for each** p_k, direct immature $\in C_k^p$ **do**
3: | **if** not ISINLIER(p_k) **then**
4: | | REMOVE(p_k)
5: | **end if**
6: | **if** NEARESETPOINTDISTANCE(allProjections, $\Pi^{\mathbf{T}_i}(\mathbf{t}_k)) < \psi^{\text{dso}}_{\text{tracerMinimumDistance}}$ **then**
7: | | allProjections $\leftarrow$ {allProjections, $\Pi^{\mathbf{T}_i}(\mathbf{t}_k)$}
8: | | $C_k^p \leftarrow$ to activate
9: | **end if**
10: **end for**
11: **for each** p_k, direct immature $\in C_k^p$ **do**
12: | OPTIMIZEDIRECTPOINT(p_k)
13: | **if** STILLINLIER(p_k) **then**
14: | | $C_k^p \leftarrow$ active direct
15: | **else**
16: | | REMOVE(p_k)
17: | **end if**
18: **end for**

MOD SLAM ensures that the reprojections of the active map-points on the current frame are well distributed (line 6). After their activation, the map-points are photometrically optimized (line 12). In rare cases, MOD SLAM deletes map-points whose optimization resulted in a high error.

Bundle Adjustment

After the immature point activation, MOD SLAM runs a direct bundle adjustment. The direct bundle adjustment refines seven frames with their associated points. It works using a Gauss-Newton minimization. Consider the Gauss-Newton system $\mathbf{H}\delta\boldsymbol{\beta} = \mathbf{g}$. The Hessian $\mathbf{H}$ is a square matrix. It can be decomposed into multiple parts. Poses $\{\mathbf{v}_i, ...\}$ and points $\mathbf{p}_i$ can be grouped:

$$\mathbf{H} = \begin{pmatrix} \mathbf{H}_{\mathbf{v},\mathbf{v}} & \mathbf{H}_{\mathbf{v},\mathbf{p}} \\ \mathbf{H}_{\mathbf{p},\mathbf{v}} & \mathbf{H}_{\mathbf{p},\mathbf{p}} \end{pmatrix}. \tag{5.3}$$

During the bundle adjustment of DSO, the size of $\mathbf{H}$ is:

$$\mathbf{H}^{\text{dso}}_{\text{row}} = \mathbf{H}^{\text{dso}}_{\text{col}} = \text{number of poses} \times 6 + \text{number of points} \times 1. \tag{5.4}$$

Map-points greatly populate the Hessian, therefore slowing its inversion. The **Schur Complement** is a method allowing the marginalization of these map-points contribution to the Hessian, hence, accelerating the SLAM:

$$\mathbf{H}_m = \mathbf{H}_{\mathbf{v},\mathbf{v}} - \mathbf{H}_{\mathbf{v},\mathbf{p}}\mathbf{H}^{-1}_{\mathbf{p},\mathbf{p}}\mathbf{H}_{\mathbf{p},\mathbf{v}} \tag{5.5}$$

$$\mathbf{g}_m = \mathbf{g}_P - \mathbf{H}_{\mathbf{v},\mathbf{p}}\mathbf{H}^{-1}_{\mathbf{p},\mathbf{p}}\mathbf{g}_{\mathbf{p}}. \tag{5.6}$$

A typical DSO [9] bundle adjustment optimizes at least 7 poses and 3000 points. Without the Schur Complement, DSO has to invert a 3042×3042 Hessian. By using the Schur Complement, DSO is able to compute the Hessian only for the poses, and reduce it to a size of 42×42. Consequently, the Hessian allows computing updates for the poses only. MOD SLAM expresses each direct map-points as inverse depth based on their reference pose. Therefore, the Jacobian and Hessian of each direct map-points is a single value (a matrix of 1 by 1). The cost of computing an update for the 3000 points is small. Moreover, by inverting the matrice of the direct map-points, MOD SLAM obtains an uncertainty for each of these points.

As explained in Chapter 4.2 at page 73, to maintain the coherence of the map, MOD SLAM runs a refinement of the indirect points after the direct bundle adjustment. Each indirect point that is seen by one of the direct active key-frame is optimized.

Management of the actives frames

Each frame that does not contain enough points is set as not active. If more than seven frames are active, one should be removed. A score is computed for each active frame. The frame with the lowest score is removed.

$$\text{dist}(v_a, v_b) = \|(\mathbf{T}_a - \mathbf{T}_b)_{xyz}\| \tag{5.7}$$

$$\text{distScore}(v_a, v_b) = 1 \div (\text{dist}(v_a, v_b) + 0.0001) \tag{5.8}$$

$$\text{distScoreBack}(v_a) = \sqrt{\text{dist}(v_a, v_i)} \tag{5.9}$$

$$\text{totalScore}(v_a) = -\,\text{distScoreBack}(v_a) \times \sum^{v_b, a \neq b, \text{active direct} \in C_b^v} \text{distScore}(v_a, v_b) \tag{5.10}$$

The function totalScore computes the score of a frame. The frame with the smallest score is marginalized. The function distScore increases the score of two frames the closest they are. The function distScoreBack decreases the score of a frame if it is too close to the current frame. The result is shown in Figure 5.3. Most active key-frames are near the active frame, as stated by distScore. But, far key-frames from the current frame still remain active as stated by distScoreBack.

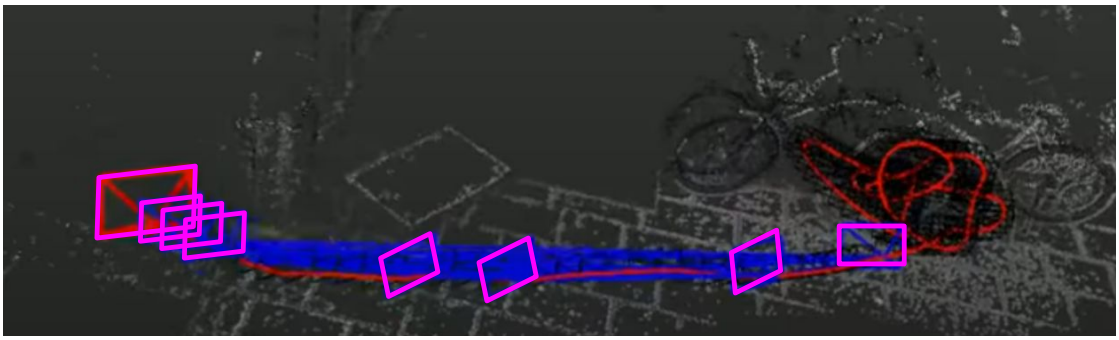


Figure 5.3: Screenshot from the DSO Presentation [9]. We highlighted the key-frames in pink.

Immature points management: creation

The DSO algorithm extracts $\psi^{\text{dso}}_{\text{immatureDensity}}$ points on the last frame. For each of these 2D points, MOD SLAM creates an immature indirect map-point. Each of these points has an associated inverse depth interval, which is initialized to $[0; 1000]$. Immature points are tracked along frames of the video in Algorithm 8.

5.2.2 Indirect Key-Frame Creation

Indirect Key-Frame Decision

MOD SLAM needs to create an indirect key-frame when the indirect algorithm does not have enough map-points to estimate the pose. Let v_r be the reference key-frame, and v_i be the current frame. Let a_r and a_i be the number of indirect map-points tracked on the reference key-frame and on the current frame. A key-frame is created if $a_i < \min(a_r, \psi^{\text{orb}}_{\text{kfRefLimit}}) \times \psi^{\text{orb}}_{\text{kfRatio}}$. The $\psi^{\text{orb}}_{\text{kfRefLimit}}$ limits the number of created key-frames when the number of indirect map-points is very high. The $\psi^{\text{orb}}_{\text{kfRatio}}$ is a ratio for MOD SLAM to compare the number of map-point in the reference key-frame and in the current frame.

Immature point management

The main goal of the indirect key-frame creation is to create new indirect map-points. Before creating new 3D map-points, MOD SLAM classifies immature map-points as inliers, outliers, or still immature. For each indirect immature point:

1. If the immature map-points have been created too recently, nothing is done for this immature point. The point is processed at least when five frames and three key-frames have been created since the creation of the immature map-point.
2. The "found ratio" of an indirect map-point is defined as the ratio between the number of times MOD SLAM has found a 2D-3D correspondance for that map-point, and the number of times MOD SLAM known this map-point is in the frame. The set S contains the point known to be in the frame. The set A contains 2D-3D correspondances. A map-point in S_{v_i} does not necessarily have a corresponding $A(v_i, j), j \in \mathbb{N}$. MOD SLAM computes the found ratio of a map-point as: $\frac{card(\{A(v_i,j),j\in\mathbb{N}\})}{card(S_{v_i})}$. If the found ratio is lower than 25%, MOD SLAM considers that the map-point is an outlier.
3. If the map-point has correspondences on at least three key-frames, it is considered as an inlier. In that case, it is optimized. An optional covariance is computed for that map-point.[3]
4. MOD SLAM removes the indirect immature map-points that last more than ten key-frames.

[3] On the first MOD SLAM paper, immature map-points were also removed by looking at their covariance. This was dropped in subsequent versions as it was not improving the accuracy enough and because the computation of the covariances was slowing the pipeline.

After the immature map-points tracking, MOD SLAM triangulates new indirect immature map-points with the 2D-2D matching feature map-points from the recent key-frames. For each covisible key-frame v_c of the current frame v_i:

1. If the distance between $\mathbf{T}_c$ and $\mathbf{T}_i$ is too small, it is not possible to triangulate 3D map-points from these two. In MOD SLAM, the scale of the map is not fixed. The distance between $\mathbf{T}_c$ and $\mathbf{T}_i$ is divided by the median depth of the point of v_i to have an invariant scale between the two poses.
2. 2D correspondences are made between v_c and v_i. For each correspondence, MOD SLAM tries to triangulate a 3D point. The map-point is accepted if it has enough parallax. In that case, it is tagged as immature: MOD SLAM does not know yet if the point is an inlier.

Bundle Adjustment

The indirect bundle adjustment can refine many poses and map-points. It converges rapidly even when the state is initialized further away from the global minimum but with less precision than a direct bundle adjustment.

The first step of the bundle adjustment algorithm is to list the frames to refine. MOD SLAM chooses to optimize the poses of the frames that share the most map-points with the current frame. The generalized map allows retrieving the frames that share at least one map-point with the current frame in constant time. The algorithm sorts these frames so that only the frames that share the most map-points with the current frame are retained. The number of frames to optimize is limited by a parameter $\psi^{\text{orb}}_{\text{maxOptimKf}}$.

In a second step, MOD SLAM lists the map-points to optimize by listing each map-points belonging to the frames to optimize.

Then, the algorithm needs to fix some poses, to make the optimization invariant to scale and translation. It fixes some key-frames poses that see a map-point to optimize.

MOD SLAM computes ten Levenberg-Marquardt iterations with the G^2O library[89], which result in updates for each frame and map-points to optimize.

Before applying these updates to the poses and the map-points, the generalized map deformation algorithm presented in Chapter 4.2 at page 73 moves the direct frame and direct map-points with indirect frames and map-points so that the reprojection of direct points remains mostly in the same position.

After the deformation, the updates are applied. To maintain the accuracy of the deformed map-points, MOD SLAM executes a photometric optimization on these map-points. This optimization is fast, as each Jacobian and Hessian contains only one value.

5.2.3 Bundle Adjustment Decision

The bundle adjustment decision is made up of 3 conditions:

- **Minimum ORB points.** If the number of active ORB points is below a threshold, MOD SLAM chooses a direct bundle adjustment, as an indirect bundle adjustment cannot do anything. MOD SLAM uses the parameter $\psi^{\text{ba}}_{\text{minorb}}$ as a threshold for this condition.
- **Saturated Ratio.** When the saturated ratio of the DSO pose estimation is high, it probably converges to local minima. Convergence of the direct bundle adjustment to a local minimum must still be avoided. Let the indirect bundle adjustment get out of this local minima first. MOD SLAM uses the parameter $\psi^{\text{ba}}_{\text{maxsatur}}$ as a threshold for this condition.
- **Points comparaison.** After the condition above, we know that both indirect and direct algorithms can potentially give a good bundle adjustment. MOD SLAM compares the number of ORB and DSO active points to choose the right bundle adjustment. The comparison is influenced by a weight $\psi^{\text{ba}}_{\text{pweight}}$. MOD SLAM runs the bundle adjustment with the method involving the higher number of points.

If MOD SLAM decides to run an indirect bundle adjustment, it will first run all the indirect key-frame creation, then all the direct key-frame creation. Conversely, if MOD SLAM chooses to run a direct bundle adjustment, it will first run all the direct key-frame creation, then indirect key-frame creation.

5.3 Initialization

In SLAM algorithms, the pose estimation depends on the result of the mapping, and the mapping depends on the result of the pose estimation. Consequently, MOD SLAM needs a process of initialization that creates both pose and mapping results at the same time. The initialization process in MOD SLAM is more complex than other methods because MOD SLAM has to initialize both direct and indirect map-points.

We tested both ORB-SLAM and DSO methods on KITTI, TUM and our own P3 dataset. We observed that ORB-SLAM struggled a lot to initialize the SLAM, and sometimes would start very late on the video. On the contrary, DSO would initialize on the first ten frames, but would be extremely slow for these ten frames. Moreover, from an ORB-SLAM initialization, the creation of photometric point is hard, due to high optical flow and inprecise pose estimation. On the contrary, it is possible to initialize the 3D indirect map-points directly when the DSO initialization algorithm finishes. For such reasons, we decided to re-use the DSO initialization algorithm, after which the indirect map-points are created.

In the video's first image, the DSO initializer extracts corners at each scale level of the image. The density of features point to extract at each level is defined as $\psi^{\text{dso}}_{\text{densityFactor}}(0.03\ 0.05\ 0.15\ 0.5\ 1.0)$. A graph is constructed such that each corner is linked to its neighbor on the same level, and on the upper and lower image level.

For 5 to 20 frames, the DSO initializer performs a Gauss-Newton optimization of the pose and the depths of the points. The optimization process is performed first from the smaller image of the pyramid to the largest image. Each time the initializer start to process a lower image of the pyramid, the depths of the point are propagated from the higher level through the lower level. Moreover, at each iteration, the depth of the points are regularized, by averaging the depth of each point with their nearest neighbor.

Initialization with DSO helped with Deep Learning

By mixing the DSO initializer with the deep learning initializer, we improve the DSO initializer. For that, we estimate the depth of the first frame of the video with a deep learning method named MonoDepth [104]. Then, we add a new regularization weight inside the DSO initializer optimization, which makes the point's depth proportional to the estimate of MonoDepth. This reduces the optical flow robustness problem of the DSO initializer, makes the DSO initializer faster, and takes fewer frames.

MOD SLAM works using the initialization of DSO, optionally helped with Deep Learning (activated with a parameter passed at the execution).

5.4 MOD SLAM version 2

After the publication of MOD SLAM [1], the implementation still had some problems.

Real-time problems.

MOD SLAM v1 includes a reimplementation of ORB-SLAM and DSO. These reimplementations were not real-time. Consequently, MOD SLAM v1 was not real-time. We focused only on the methods and not on the performance of the implementation.

MOD SLAM v2 uses an optimized version of the generalized map, presented in the previous chapter. MOD SLAM v2 uses AVX/SSE instructions for Intel processors and Neon instructions for ARM processors to speed up calculations, particularly image processing.

For example, the ORB feature extractor needs to blur the image to compute BRIEF descriptor (so it avoids noise). The execution time of this blur function has been reduced by a factor of 100.

The data structure was reorganized to optimize the cache usage and reduce the number of memory allocations, by using pooled allocators. This is particularly the case for the direct optimizer.

Reproductibility problems.

MOD SLAM v1 had randomness in the result. This happens because some calculations were numerically unstable (some were producing “NaN”), and some processes had slight randomness (accumulating over time).

The instability came first from the map incoherence. On MOD SLAM v1, the map deformation was not enabled. MOD SLAM v2 uses the automatic deformation of the map and disables marginalized points within the DSO optimization.

Randomness came from the fact that the hash-structure was using pointer addresses, and therefore was returning data in random order. Some algorithms work better by iterating on the most recent frame first.

Parameters empirically tuned.

On MOD SLAM v1, we empirically adjusted the parameters, whereas on MOD SLAM v2, the parameters were tuned as described in the next chapter.

5.5 Conclusion

MOD SLAM is a hybrid SLAM pipeline that mixes ORB-SLAM and DSO. To make the information between the different types of algorithms coherent, MOD SLAM is based on the generalized map. For each step of the pipeline, MOD SLAM decides if it will run a feature-based algorithm, similar to those of ORB-SLAM, or a photometric algorithm, similar to those of DSO. At each iteration, MOD SLAM is able to choose between a photometric based pose estimation algorithm robust to low textured image, or a feature-based pose estimation algorithm robust to high optical flow. We will show in the next chapter that this makes MOD SLAM benefit the robustness of ORB-SLAM and DSO at the same time. The next chapter also presents an automatic parameter tuning method for MOD SLAM, the result of MOD SLAM on the popular dataset, and a study of the impact of the parameter of the decision system.

The code source of MOD SLAM is on the folder "src/cml/slam/modslam" of our Github: `https://github.com/belosthomas/libCML`.

6

MOD SLAM Results

The last chapter explained MOD SLAM, a SLAM method mixing the algorithms of ORB-SLAM and DSO on the same pipeline. Each of these algorithms has many parameters. All these parameters are co-dependent. The parameters of the first version of MOD SLAM have been tuned empirically. The process of tuning the parameters of MOD SLAM is tedious and time-consuming. That is why the first part of this chapter presents an automatic procedure to tune the parameters of MOD SLAM. From these tuned parameters, this chapter presents the results of MOD SLAM. We show that MOD SLAM surpasses ORB-SLAM and DSO in terms of accuracy and robustness.

MOD SLAM is based on the generalized map, presented during Chapter 4. We show that the generalized map is efficient enough to run MOD SLAM on smartphones in real-time. We also show that the extensibility of the generalized map allows us to rapidly create an augmented reality application in a few lines of code.

The last part of this chapter presents the impact of the decision system of MOD SLAM. We show that the decision system has a non-negligible impact on the overall results.

This chapter is organized as follows.

1. We present a grid search algorithm for automatic parameter tuning.
2. We show the result obtained on MOD SLAM. We demonstrate the robustness and accuracy of MOD SLAM compared to ORB-SLAM and DSO.
3. We briefly introduce applications based on MOD SLAM. We show that MOD SLAM can run on Android in real-time.
4. We demonstrate the impact of the decision system of MOD SLAM by plotting the parameters of the decision system.

6.1 Grid search Algorithm

Our grid search algorithm is a naive approach for optimizing the parameters of MOD SLAM. A separate file contains the name and possible values of all parameters of the SLAM. The grid search algorithm optimizes each parameter consecutively. As shown in Algorithm 10, it tests several possible values for each parameter. The parameter which has made the SLAM result in the lower absolute trajectory error average is kept, and used for the next optimization of the parameters.

Algorithm 10 Automatic Parameter Finding Algorithm.

```
1: Ψ ← The parameters of MOD SLAM
2: while the user do not stop the program do
3:     for each ψ ∈ Ψ do
4:         bestE, bestV ← ∞, ∅
5:         for each v ∈ possibleValuesFor(ψ) do
6:             e ← runModslamNTimesWithNoise(ψ, v, 5)
7:             if e < bestE then
8:                 bestE, bestV ← e, v
9:             end if
10:        end for
11:        setDefaultParameterOfModslam(ψ, bestV)
12:    end for
13: end while
```

We define two versions of the algorithm:

- The fast algorithm optimizes the current parameter and passes the modification to the following parameter
- The slow algorithm tests all parameters and applies all the parameter modification that causes the most significant improvement only after having tested all the parameters. Contrary to the fast algorithm, the slow algorithm will produce the same exact result, whatever in which order the parameters are tested.

As MOD SLAM is deterministic, each error is based on the sum of averages of five executions perturbed with random noise (noise on the image or small variation of the parameters) over the different datasets. This allows the algorithm to check that the SLAM is stable and robust with a given set of parameters.

6.2 Results

This section aims to show the result obtained by MOD SLAM on the datasets presented in Chapter 2.

Results on KITTI Dataset

Table 6.1 shows the result on the KITTI dataset on MOD SLAM v1, MOD SLAM v2, ORB-SLAM2, and DSO. Unlike any other method, our SLAM is deterministic and reproducible. Our SLAM always gives the same result when the same entry is fed to the algorithm. That is why we tested MOD SLAM v2 without noise and by adding random noise.

In most cases, we get a better error than ORB-SLAM2 and DSO. We observe in some videos a significant drift reduction. Only MOD SLAM succeeds in processing the KITTI 01 video. We have shown in the state-of-the-art chapter that KITTI 01 is a video that contains parts that DSO cannot handle but ORB-SLAM can, and another part that ORB-SLAM cannot handle but DSO can. That is why both fail on this video. As our SLAM can choose between both methods, it can overcome all hurdles.

	MOD SLAM v2	with noise	ORB-SLAM2	DSO	MOD SLAM v1
Average*	**27.13**	35.39	34.79	55.56	35.48
KITTI 00	**39**	56	67	114	105
KITTI 01	**17**	12	x	x	12
KITTI 02	80	94	**43**	120	43
KITTI 03	2.7	8	**1**	2.1	1.8
KITTI 04	**0.6**	0.9	0.9	1.5	1.1
KITTI 05	**21**	18	43	52	40
KITTI 06	**46**	55	49	59	44
KITTI 07	**11**	11	17	17	16
KITTI 08	**36**	52	58	111	54
KITTI 09	**24**	51	60	63	39
KITTI 10	11	8	**9**	16	11

Table 6.1: Absolute Trajectory Error on KITTI datasets [20].

Number of successes on TUM and KITTI

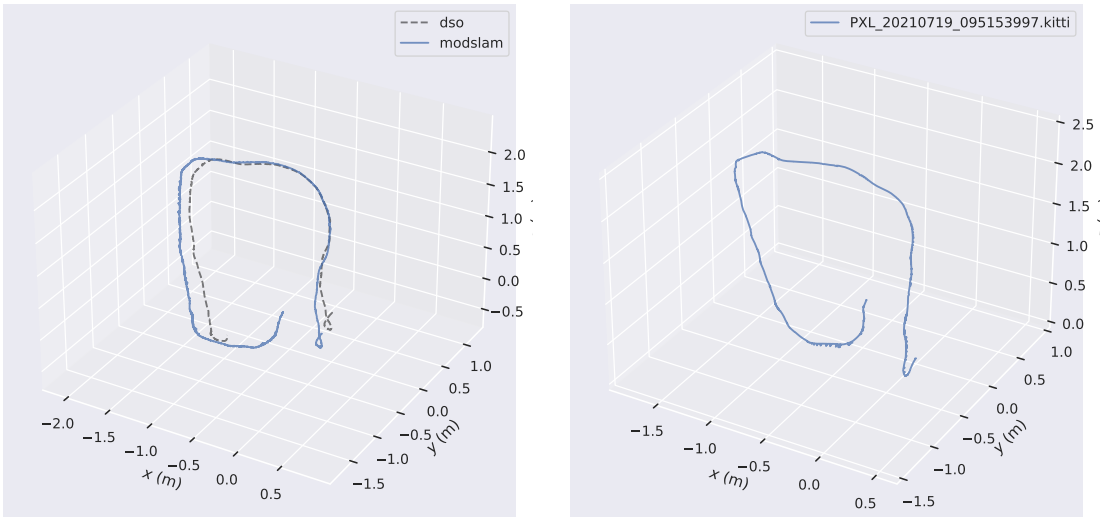
Table 6.2 represents the number of successes in the three datasets. ORB-SLAM2 has been specialized on KITTI but is faulty on most TUM sequences. DSO is specialized for the TUM videos but produces high drift on KITTI sequences. MOD SLAM can run on all videos without changing any parameter while still maintaining the precision of ORB on the KITTI sequences and the density of the generated point cloud of DSO on the TUM dataset.

	MOD SLAM	ORB-SLAM2	DSO
TUM	100%	56%	100%
KITTI	100%	91%	91%
P3	100%	0%	100%
Total	100%	58%	98%

Table 6.2: **Success ratio on multiple datasets.** We tested on TUM Monocular Dataset [21], KITTI dataset [20], and our own smartphone videos

Results on Imagine P3

In our paper MOD SLAM [1], we tested our method on the video of a Google Pixel 3a. The results are plotted in Figure 6.1. ORB-SLAM2 got lost on all the videos. Observe how the loop closes: MOD SLAM and DSO get almost the same result, except for some videos where we can observe a slight improvement for MOD SLAM.



(a) Estimated trajectory of MOD SLAM v1 and DSO. This figure comes from our paper MOD SLAM [1].

(b) Estimated trajectory of MOD SLAM v2.

Figure 6.1: Estimated trajectories of a video taken with a Google Pixel 3a. The video ends at the same place it started. The closer the two trajectory ends, the lower the drift. ORB-SLAM2 got lost on all the videos.

MOD SLAM results on Stereopolis

Stereopolis on MOD SLAM is a work in progress. Stereopolis sequences regroup the difficulties of both direct and indirect methods. The optical flow between the images is higher than any KITTI videos. The greater part of the images is repetitive textures, and some moving object appears. Moreover, the images of the Stereopolis dataset are not photometrically calibrated.

We used a specific configuration file to execute MOD SLAM on the front camera of the Stereopolis dataset. We cropped the images to hide the hosting car and a part of the sky. Figure 6.2 shows screenshots of MOD SLAM during its execution. They show the high drift produced by MOD SLAM.

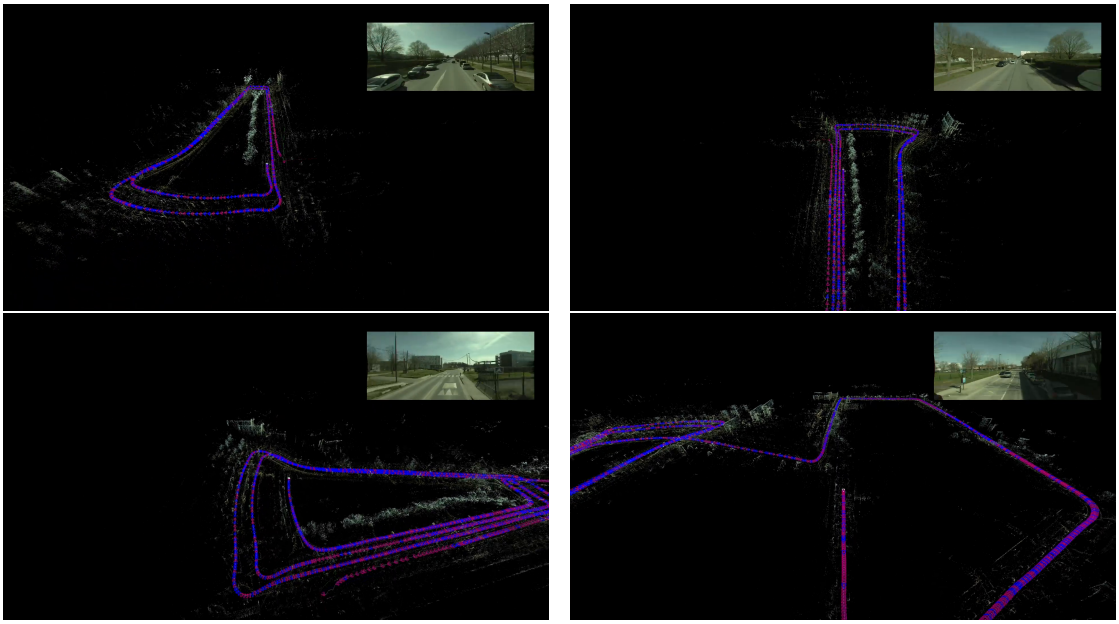


Figure 6.2: Screenshot MOD SLAM execution on Stereopolis.. The car hosting the camera makes loops. The pose estimation is shifted each time the car goes on the same road loop.

This section showed the result of MOD SLAM on the different standard literature datasets. The following section shows the result of MOD SLAM in a real application.

6.3 An Augmented Reality Application of MOD SLAM

The generalized map is a framework allowing the creation of SLAM of different families but also applications. Applications can connect to any SLAM through the generalized map. This chapter aims to present an application coded on top of the generalized map for augmented reality, taking into account the depth obstruction and the lightning.

SLAM on Android Smartphone

MOD SLAM has been developed and tested on desktop computers to work on smaller devices like smartphones or drones. That explains why MOD SLAM depends on no libraries except Qt for the user interface to make MOD SLAM compatible with numerous devices. Therefore, MOD SLAM can easily be built as an Android APK.

Computation of a depth map using Guided Filtering

Using the generalized map, we implemented in fewer than 100 lines of code a program similar to guided filtering [105], that computes a real-time depth map for each video frame using the point cloud estimated by the SLAM.

For each video frame, the augmented reality algorithm projects direct and indirect map-points of the generalized map into the image. The depth of the projected points is reported on a depth map. Because each pixel of the depth map does not have a corresponding point to reproject, the depth map is dilated. Each time the depth map is dilated, the dilatation takes into account the image. To be more precise, the algorithm is decomposed into five steps:

- Let a v_i be a frame. Let $\mathbf{D}$ be the depth map of v_i. Each pixel of $\mathbf{D}$ will be computed with a weighted average of multiple depths. Therefore, we also introduce $\mathbf{E}$, which is the sum of the depth, and $\mathbf{W}$, which is the total weight. Thus, $\mathbf{D}_{x,y} = \frac{\mathbf{E}_{x,y}}{\mathbf{W}_{x,y}}$.
- For each map-point m_j of the generalized map, if the projection of m_j into $\mathbf{T}_i$ $\mathbf{o} = \Pi^{\mathbf{T}_i}(\mathbf{t}_j)$ is a 2D point inside the image I_i of v_i, then the algorithm adds $\mathbf{o}_z \times c_k$ to $\mathbf{E_o}$, and c_k to $\mathbf{W_o}$, with $c_k \in \mathbb{R}$ the certainty of m_j.
- For each pixel $\mathbf{o}$ of the map-point, and for each neighborhood of $\mathbf{o}$ as $\mathbf{n}$, we add to $\mathbf{E_n}$ the value $0.01 \times \mathbf{E_o} \times \|I_\mathbf{o} - I_\mathbf{n}\|_1$, and we add to $\mathbf{W_n}$ the value $0.01 \times \mathbf{W_o} \times \|I_\mathbf{o} - I_\mathbf{n}\|_1$.
- The previous action is repeated 1000 times.
- The final depth map is computed so that $\mathbf{D}_{x,y} = \frac{\mathbf{E}_{x,y}}{\mathbf{W}_{x,y}}$.

Draw a 3D object, by taking into account the light and the depth obstruction

Thanks to the estimated depth maps, and the estimated light condition of DSO, it is possible to draw 3D objects into the real scene, taking into account the obstruction and the light.

- A 3D model is loaded in the OBJ format. To have more ease in the next step, we convert the 3D model composed of polygons into a list of triangles.
- The 3D model position is set at execution.
- We project and draw each triangle of the 3D model into the image of the video. Before drawing each pixel, we check that the depth corresponding to the pixel is in front by comparing it with the pixel of the depth map.
- The drawn color of the pixel is modified by using the DSO light model and the estimated light of v_i.

Figure 6.3 shows the results we obtained with this algorithm. We added a Pikachu in one of the videos of the TUM dataset. Look at how the Pikachu appears when the camera moves behind the door.

In Figure on 6.3b, the right side of the Pikachu is cropped due to the wall. However, we can observe that the depth map is not perfect and that there is some "latency".

(a) A Pikachu is hidden behind that door... can you see it ?

(b) And now... can you see it ?

(c) Let's get closer.

Figure 6.3: A Pikachu is added in augmented reality, taking into account the occlusion and the lighning. The top left image is a depth map computed in real-time from the 3D map-points estimated by MOD SLAM v2.

6.4 Parameters Study

This subsection aims to study different conditions for the decision system, mainly in videos 0, 2, and 9 of the KITTI dataset. We choose these videos because they are the longest and have a variety of difficulties (fast turning, light change, texture repetition, etc.) The study is done with our automatic parameter optimizer and plotting.

We structured each analysis into three parts. Initially, we assess the impact of the parameter variation on the decision-making process. Then, we explore the effects of the parameters on the overall error. Lastly, we examine the parameter's ability to generalize across multiple video sources.

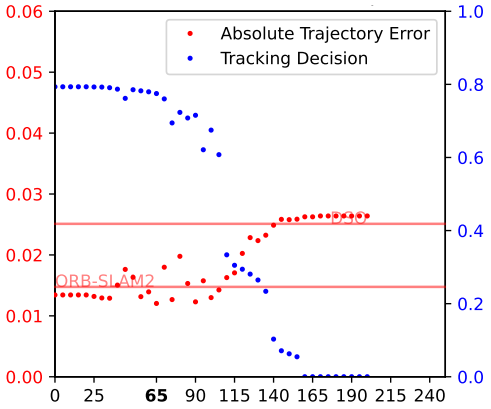
6.4.1 Pose Estimation Decision

Minimum number of indirect active points (Figure 6.4)

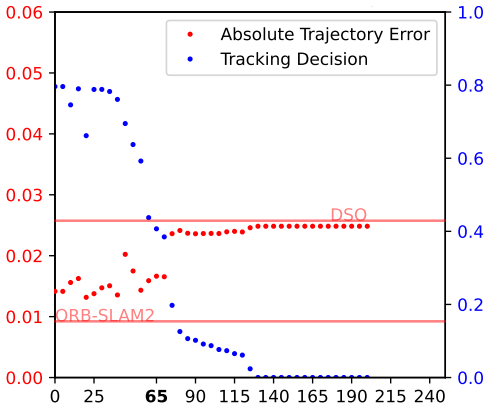
At each iteration, MOD SLAM tags a certain number of indirect map-points as "Active map-points". MOD SLAM uses these map-points for the indirect pose estimation. A too-low number of indirect active map-points may create weakness inside the indirect algorithm. In that case, MOD SLAM should not estimate the pose with an indirect algorithm. On that account, MOD SLAM uses the quantity of active indirect map-points to estimate the pose of the next image in the video. If the number of indirect active map-points is lower than a threshold $\psi^{\text{pe}}_{\text{minorb}}$, MOD SLAM uses a direct algorithm to estimate the pose of the next frame. Otherwise, it goes to the following covariance condition. The graphs of Figure 6.4 show a significant impact on the decision in all cases. The higher we increase $\psi^{\text{pe}}_{\text{minorb}}$, the more often the pose estimation decision chooses direct algorithms. We observe a correlation between the direct/indirect decision and the accuracy of the SLAM. A value of $\psi^{\text{pe}}_{\text{minorb}}$ higher than 0 decreases the error. However, a too-high value for $\psi^{\text{pe}}_{\text{minorb}}$ increases the error. We found the optimal value for $\psi^{\text{pe}}_{\text{minorb}}$ to be 65 for most videos.

Covariance Comparison (Figure 6.5)

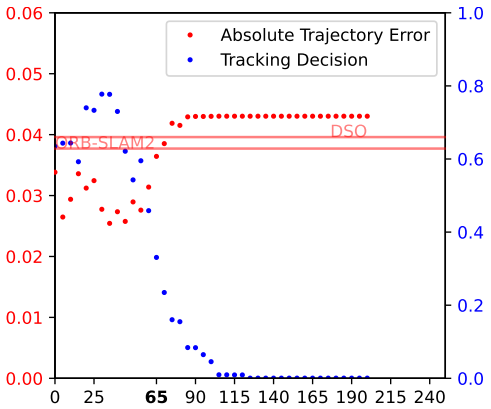
We shown in the previous chapter, at page 93, that, at each iteration of the MOD SLAM pipelines, direct and indirect algorithms compute the uncertainty u_{direct} and u_{indirect} of the last pose estimation. These uncertainties may indicate that the direct or the indirect method could give a better result. Based on the uncertainties of the previous iterations, the covariance condition decides between an indirect base pose estimation or a direct base pose estimation. $\psi^{\text{pe}}_{\text{coweight}}$ is a weight that influences this decision. With a lower weight, the condition chooses indirect algorithm, and with a higher weight, the decision will choose direct algorithm. We optimized and plotted the parameter $\psi^{\text{pe}}_{\text{coweight}}$ to study the impact of this condition. On the plots of Figure 6.5, we observe almost no impact on KITTI 00 et KITTI 09 because this condition takes place after the minimum number of indirect active map-points condition. The previous decision decided to use direct pose estimation before this decision (and so this decision has not to be made). We added the plots KITTI 04 on Figure 6.5, since on this video, the previous condition was never triggered. During the execution of MOD SLAM of this video, we observe a high impact of the covariance comparison condition. During our test, we observed that this condition is not universal: the optimal value differs from on TUM dataset and KITTI dataset.



(a) KITTI 00



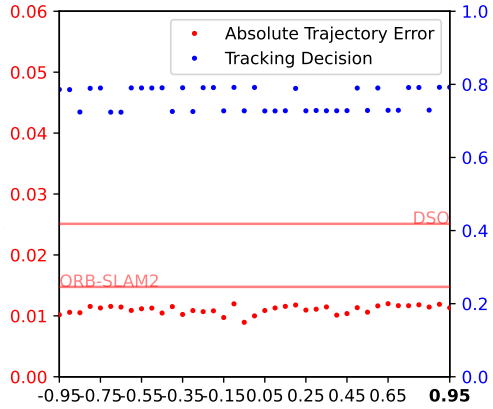
(b) KITTI 02



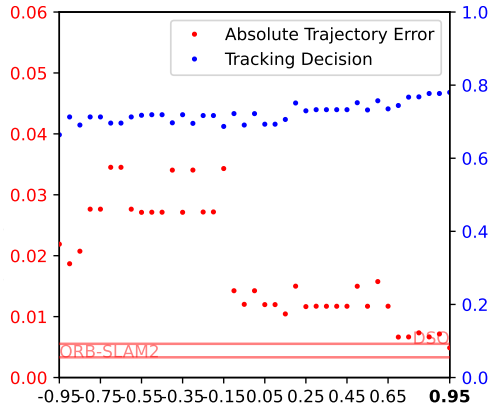
(c) KITTI 09

Figure 6.4

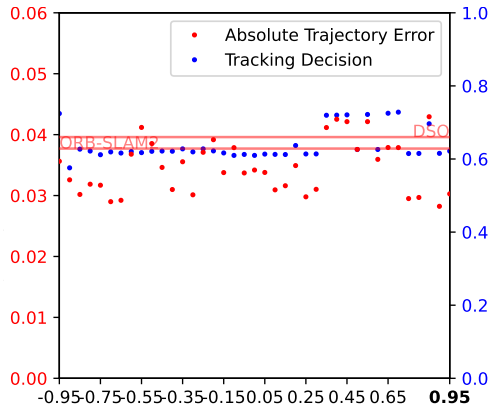
Pose Estimation Decision: Minimum number of ORB active points.



(a) KITTI 00



(b) KITTI 04



(c) KITTI 09

Figure 6.5

Pose Estimation Decision: Covariance Comparison. This condition happens after the "Minimum number of ORB active points" condition.

6.4.2 Bundle Adjustment Decision

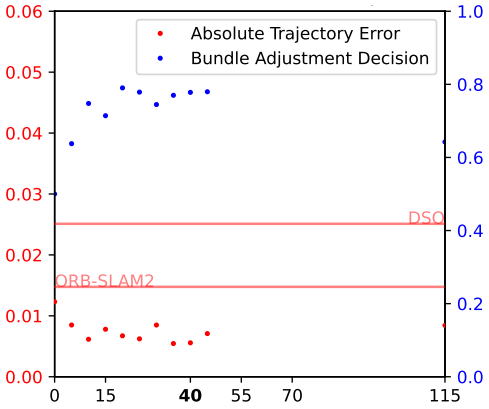
Minimum number of ORB inliers (Figure 6.6)

The number of optimized points is the number of active points that have been successfully reprojected on the current frame and are estimated as inliers. On low-textured images, the indirect part has few 3D points. A condition using the number of indirect points helps to reject feature-based optimization when there is insufficient indirect robustness. If the number of indirect map-points inliers is lower than $\psi^{\text{ba}}_{\text{minorb}}$, MOD SLAM chooses a direct bundle adjustment. The graph 6.6 shows the impact of a minimum number of indirect points on the estimation of the pose. We can observe that the algorithm tends to choose more DSO when the minimum number increases. With a too-high value of $\psi^{\text{ba}}_{\text{minorb}}$, the SLAM is not able to finish the videos. This parameter allows a small decrease in the overall error. The optimal value seems to generalize.

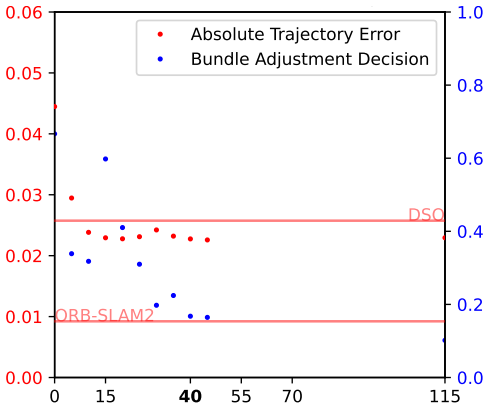
6.5 Conclusion

The grid search has allowed us to tune the SLAM algorithms to the point where MOD SLAM outperforms ORB SLAM and DSO in terms of accuracy and robustness. Furthermore, as MOD SLAM is based on the mixed map, which is optimized to be fast, MOD SLAM can run on smartphones in real-time. Thanks to the mixed map, we have been able to rapidly create an augmented reality application in a few lines of code. In the last section of the chapter, we proved that the decision system has a non-negligible impact on the overall results.

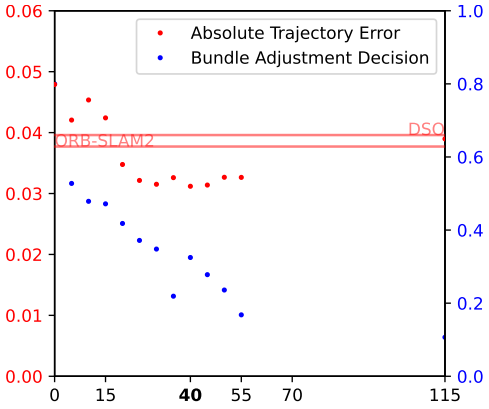
The grid search algorithm, the script to evaluate the SLAM, and the script to create the plot, are available in the “evaluation” folder of our Github: `https://github.com/belosthomas/libCML`.



(a) KITTI 00



(b) KITTI 02



(c) KITTI 09

Figure 6.6: Bundle Adjustment Decision: Minimum number of ORB inliers.

7

Conclusion and Perspectives

7.1 Conclusion

Monocular Simultaneous Localization and Mapping method allows a robot to localize itself and build a map of its surroundings using only a single monocular camera. We have shown that it is important to use lightweight sensors in applications such as autonomous drones or augmented reality. Using a single monocular camera is a more affordable and lightweight option than using other sources such as RGB-D Camera or LiDAR.

Monocular algorithms work by tracking features on the image over time, to calculate the robot's pose relative to those features, and to update the map. They follow a general pipeline composed of three main blocks: The tracking block estimates the pose from the existing 3D map-points. The mapping block creates new 3D map-points. The bundle adjustment block refines the map.

We have shown that two types of techniques exist to implement these methods. Photometric techniques optimize the pixel intensities of the image in an optical flow manner. Feature-based techniques make 2D-3D features correspond to execute an optimization of the re-projections. A SLAM using photometric techniques is categorized as a direct SLAM. A SLAM using feature-based techniques is categorized as an indirect SLAM. A SLAM using both techniques (without mixing the two techniques in the same block) is called a semi-direct SLAM.

The main challenge of these monocular methods is that they use a single camera as their input, which provides a limited amount of information about the environment. Hence, the three categories of SLAM have difficulty in obtaining accurate estimates of the pose and the structure of the environment. Direct Sparse Odometry [9], the state-of-the-art of direct method, is sensitive to high optical flow. ORB-SLAM [12], the state-of-the-art of indirect method, is sensitive to low-textured images. The Semi-direct pipelines suffer from both the problems of direct and indirect methods.

To solve these problems, we developed the MOD SLAM method, which, for each block, mixes DSO and ORB-SLAM techniques. At each iteration, MOD SLAM decides direct or indirect method according to specific criteria, such as the uncertainty of the pose or the number of indirect inliers. In order to create MOD SLAM method, we started by developing a generalized map.

Generalized Map

The generalized map is a map implementation compatible with direct and indirect families of algorithms. It allows them to store their specific information. For this purpose, we presented a unified formalism of the map, which generalize to the different type of SLAM methods. Based on this formalism, we have been able to write indirect and direct algorithms in a unified way, during Chapter 5.

The generalized map ensures the coherence of information between the direct and indirect algorithms. The map definition is independent of the information representation. Moreover, the map comprises an automatic deformation algorithm, which allows the map to maintain its coherence. The deformation algorithms move coherently the frame and the 3D points, which are not optimized by one algorithm. Therefore these not-optimized frames and 3D points do not become outliers in other algorithms.

The generalized map must be efficient enough not to impact the method's execution time, even with a large amount of information in it. From the unified formalism, we introduced an implementation of the generalized map, which is efficient enough to make mixed SLAM method real-time. This implementation is based on the usage of hash-structures specifically made for the generalized map.

Thanks to the generalized map, algorithms of different families can communicate through the map without modifying their implementation. This is the case of the ORB-SLAM and DSO algorithms inside MOD SLAM.

MOD SLAM

MOD SLAM is a mixed ORB and DSO SLAM method based on the generalized map. During chapter 5, we developed the central part of MOD SLAM based on the general pipeline. The three main blocks of MOD SLAM, which are the tracking block, the mapping block, and the bundle adjustment block, use a decision system that permits MOD SLAM to choose if it will execute more direct or indirect algorithms.

The tracking block uses a decision based on the number of inliers of the indirect and direct methods. MOD SLAM chooses an indirect algorithm in case of high optical flow. The indirect algorithm robustly optimizes the pose, while the following direct algorithm makes this pose estimation precise. A direct algorithm is chosen when MOD SLAM assesses that the direct algorithm can estimate the pose without struggling.

MOD SLAM uses the keyframe decision of ORB-SLAM [12] and DSO [9] to decide if the direct and the indirect mapping blocks need to be executed. Direct and indirect algorithms may both need to create new map-points at any time. However, only one of the two bundle adjustments can be executed, or none. One bundle adjustment is sufficient to refine the map. Two bundle adjustments would make the keyframe creation very slow. The execution of only one bundle adjustment without losing the coherence of the map is made possible thanks to the deformation algorithm of the generalized map.

Parameters tuning

The parameters of MOD SLAM are linked and depend on the other. To tune these parameters, we presented automatic parameter-finding algorithms in Chapter 6.3.

We have shown the increased robustness of MOD SLAM, compared to ORB-SLAM and DSO. Especially, MOD SLAM is the only method that can handle the video KITTI 01. We have shown that the tuned version of MOD SLAM is much more accurate than ORB-SLAM and DSO. However, we also show that MOD SLAM struggles on the Stereopolis dataset.

We have determined that MOD SLAM could run in real-time on Android thanks to the generalized map and the automatic parameters tuner. We presented an augmented reality application based on the generalized map, and using MOD SLAM.

From the automatic parameter tuning algorithm, we plotted the impact of the main parameters of the decision system. Thanks to these plots, we have proven the impact of the decision system on the accuracy and robustness.

7.2 Perspectives

We have shown that MOD SLAM improved the result of ORB-SLAM and DSO, while being able to work on an Android phone. However, numerous problems still need to be solved, and multiple axes must be worked.

- **Stereopolis.** One of the main perspectives is to convert the Stereopolis Dataset into a standard dataset with photometrically calibrated images. Photometrically calibrated images packed in a standard format would allow us to run MOD SLAM more on this dataset and to compare MOD SLAM with other methods on this dataset. Notably, this would allow us to run Stereopolis with DSO, which is particularly sensitive to the bad photometric calibration of the Stereopolis dataset, and with the ORB-SLAM method, which is less sensitive, as the BRIEF descriptors are almost invariant to the application of the non-linear inverse response on the images.
- **Mixed Optimization.** The decision system and the generalized map deformation system are solutions to the problems of mixing photometric and feature-based optimization. However, a mixed bundle adjustment would perform better. No solution has been addressed to this problem, as no method exists to optimize two objective functions which express different metering (the two meterings are: the distance between color; and the distance between two points in a scale-variant space).
- **Mixed Corner Detector.** The code of MOD SLAM can still be optimized. Particularly, a mixed corner extractor for ORB-SLAM and DSO would decrease the computation time.
- **Loop Closure.** MOD SLAM does not contain any loop closure system. But the code already contains the necessary algorithms to implement this loop closure system. This would improve results on numerous videos containing loops.
- **Marginalization Hessian in DSO Optimization.** The marginalization Hessian of DSO has been disabled due to performance impact. It decreases the accuracy of DSO by disabling the marginalization Hessian. We need to find a way to re-enable this marginalization Hessian.
- **Rolling shutter handling.** MOD SLAM does not natively handle rolling shutter. It is sufficiently robust to give a result with the rolling shutter, but these results would be much more accurate by handling the rolling shutter.

7.3 Code availability

Our code is available on GitHub, including the generalized map, MOD SLAM, and the automatic parameter-finding algorithms.

Bibliography

[1] Liza Belos, Pascal Monasse, and Eva Dokladalova. “MOD SLAM: Mixed Method for a More Robust SLAM without Loop Closing”. In: VISAPP 2022, 2022, pp. 691–701 (cit. on pp. 22, 41, 110, 117).

[2] i-site FUTURE. *Urban Vision*. ANR-16-IDEX-0003. 2019 (cit. on pp. 3, 36).

[3] Mathieu Gonzalez, Eric Marchand, Amine Kacete, and Jérôme Royan. “S3LAM: SLAM à Scène Structurée”. In: *GRETSI 2022 - XXVIIIème Colloque Francophone de Traitement du Signal et des Images*. Nancy, France, Sept. 2022, pp. 1–4 (cit. on p. 9).

[4] Li Jinyu, Yang Bangbang, Chen Danpeng, et al. “Survey and evaluation of monocular visual-inertial SLAM algorithms for augmented reality”. In: *Virtual Reality & Intelligent Hardware* 1 (4 Aug. 2019), pp. 386–410 (cit. on pp. 9, 19, 28).

[5] Vijay Kumar and Nathan Michael. “Opportunities and challenges with autonomous micro aerial vehicles”. In: *The International Journal of Robotics Research* 31 (11 Sept. 2012), pp. 1279–1291 (cit. on pp. 9, 19, 34).

[6] Georges Younes, Daniel Asmar, and John Zelek. “A Unified Formulation for Visual Odometry”. In: IEEE, Nov. 2019, pp. 6237–6244 (cit. on pp. 9, 21, 24, 37, 40).

[7] Richard A. Newcombe, Steven J. Lovegrove, and Andrew J. Davison. “DTAM: Dense tracking and mapping in real-time”. In: IEEE, Nov. 2011, pp. 2320–2327 (cit. on pp. 12, 20, 41, 43, 62).

[8] Jakob Engel, Thomas Schöps, and Daniel Cremers. *LSD-SLAM: Large-Scale Direct Monocular SLAM*. 2014 (cit. on pp. 12, 20, 41, 43).

[9] Jakob Engel, Vladlen Koltun, and Daniel Cremers. “Direct Sparse Odometry”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40 (3 Mar. 2018), pp. 611–625 (cit. on pp. 12, 20, 23, 33, 41, 43, 45, 49, 61, 65, 66, 86, 87, 104, 105, 125, 127).

[10] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, Mar. 2004 (cit. on pp. 12, 40, 53, 56).

[11] Georg Klein and David Murray. “Parallel Tracking and Mapping for Small AR Workspaces”. In: IEEE, Nov. 2007, pp. 1–10 (cit. on pp. 12, 20, 41, 42).

[12] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos. “ORB-SLAM: A Versatile and Accurate Monocular SLAM System”. In: *IEEE Transactions on Robotics* 31 (5 Oct. 2015), pp. 1147–1163 (cit. on pp. 12, 20, 23, 41, 42, 49, 58, 125, 127).

[13] Jianjun Ni, Xiaotian Wang, Tao Gong, and Yingjuan Xie. “An improved adaptive ORB-SLAM method for monocular vision robot under dynamic environments”. In: *International Journal of Machine Learning and Cybernetics* (Aug. 2022) (cit. on pp. 12, 42).

[14] Raul Mur-Artal and Juan D. Tardos. “ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras”. In: *IEEE Transactions on Robotics* 33 (5 Oct. 2017), pp. 1255–1262 (cit. on pp. 16, 22, 33, 41, 45, 49, 56, 86).

[15] Emanuele Menegatti, Andrea Zanella, Stefano Zilli, Francesco Zorzi, and Enrico Pagello. “Range-only SLAM with a mobile robot and a Wireless Sensor Networks”. In: IEEE, May 2009, pp. 8–14 (cit. on p. 19).

[16] Lukas von Stumberg, Vladyslav Usenko, Jakob Engel, Jorg Stuckler, and Daniel Cremers. “From monocular SLAM to autonomous drone exploration”. In: IEEE, Sept. 2017, pp. 1–8 (cit. on p. 19).

[17] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. “Visual SLAM algorithms: a survey from 2010 to 2016”. In: *IPSJ Transactions on Computer Vision and Applications* 9 (1 Dec. 2017), p. 16 (cit. on pp. 20, 41).

[18] Maksim Filipenko and Ilya Afanasyev. “Comparison of Various SLAM Systems for Mobile Robot in an Indoor Environment”. In: IEEE, Sept. 2018, pp. 400–407 (cit. on pp. 20, 30).

[19] Xiang Gao, Rui Wang, Nikolaus Demmel, and Daniel Cremers. “LDSO: Direct Sparse Odometry with Loop Closure”. In: IEEE, Oct. 2018, pp. 2198–2204 (cit. on pp. 20, 41, 44).

[20] A. Geiger, P. Lenz, and R. Urtasun. “Are we ready for autonomous driving? The KITTI vision benchmark suite”. In: IEEE, June 2012, pp. 3354–3361 (cit. on pp. 20, 35, 60, 66, 115, 116).

[21] Jakob Engel, Vladyslav Usenko, and Daniel Cremers. “A Photometrically Calibrated Benchmark For Monocular Visual Odometry”. In: (July 2016) (cit. on pp. 20, 40, 62, 116).

[22] Seong Hun Lee and Javier Civera. “Loosely-Coupled Semi-Direct Monocular SLAM”. In: *IEEE Robotics and Automation Letters* 4 (2 Apr. 2019), pp. 399–406 (cit. on pp. 21, 24, 41, 43, 44, 47).

[23] R. Timothy Marler and Jasbir S. Arora. “The weighted sum method for multi-objective optimization: new insights”. In: *Structural and Multidisciplinary Optimization* 41 (6 June 2010), pp. 853–862 (cit. on p. 21).

[24] Ashutosh Singandhupe and Hung Manh La. “A Review of SLAM Techniques and Security in Autonomous Driving”. In: IEEE, Feb. 2019, pp. 602–607 (cit. on p. 28).

[25] Hyunchul Roh, Jinyong Jeong, Younggun Cho, and Ayoung Kim. “Accurate Mobile Urban Mapping via Digital Map-Based SLAM”. In: *Sensors* 16 (8 Aug. 2016), p. 1315 (cit. on pp. 28, 34).

[26] Thrilochan Sharma P., P. Sankalprajan, Ashish Joel Muppidi, and Prithvi Sekhar Pagala. “Analysis of Computational Need of 2D-SLAM Algorithms for Unmanned Ground Vehicle”. In: IEEE, May 2020, pp. 230–235 (cit. on p. 28).

[27] Juan Zhang, James F. Campbell, Donald C. Sweeney II, and Andrea C. Hupman. “Energy consumption models for delivery drones: A comparison and assessment”. In: *Transportation Research Part D: Transport and Environment* 90 (Jan. 2021), p. 102668 (cit. on pp. 28, 34).

[28] Patrick Benavidez, Mohan Muppidi, Paul Rad, et al. “Cloud-based realtime robotic Visual SLAM”. In: IEEE, Apr. 2015, pp. 773–777 (cit. on p. 28).

[29] Supun Kamburugamuve, Hengjing He, Geoffrey Fox, and David Crandall. “Cloud-based Parallel Implementation of SLAM for Mobile Robots”. In: ACM, Mar. 2016, pp. 1–7 (cit. on p. 28).

[30] Josep Aulinas, Yvan Petillot, Joaquim Salvi, and Xavier Lladó. “The SLAM problem: a survey”. In: *Artificial Intelligence Research and Development* (Jan. 2008), pp. 363–371 (cit. on p. 29).

[31] Olivia Christie, Joshua Rego, and Suren Jayasuriya. “Analyzing Sensor Quantization Of Raw Images For Visual Slam”. In: IEEE, Oct. 2020, pp. 246–250 (cit. on p. 29).

[32] Mubariz Zaffar, Shoaib Ehsan, Rustam Stolkin, and Klaus McDonald Maier. “Sensors, SLAM and Long-term Autonomy: A Review”. In: IEEE, Aug. 2018, pp. 285–290 (cit. on p. 29).

[33] Navid Nourani-Vatani and Jonathan Roberts. “Automatic camera exposure control”. In: 2007, pp. 1–6 (cit. on p. 29).

[34] Kusworo Adi and Catur Edi Widodo. “Distance measurement with a stereo camera”. In: *International Journal of Innovative Research in Advanced Engineering (IJIRAE)* 4 (11 Nov. 2017), pp. 24–27 (cit. on pp. 29, 31).

[35] Jakob Engel, Jorg Stuckler, and Daniel Cremers. “Large-scale direct SLAM with stereo cameras”. In: IEEE, Sept. 2015, pp. 1935–1942 (cit. on pp. 29, 31).

[36] Felix Endres, Jurgen Hess, Jurgen Sturm, Daniel Cremers, and Wolfram Burgard. “3-D Mapping With an RGB-D Camera”. In: *IEEE Transactions on Robotics* 30 (1 Feb. 2014), pp. 177–187 (cit. on pp. 29, 31).

[37] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. “Real-time loop closure in 2D LIDAR SLAM”. In: IEEE, May 2016, pp. 1271–1278 (cit. on pp. 29, 32).

[38] Cenek Albl, Zuzana Kukelova, Viktor Larsson, et al. “From Two Rolling Shutters to One Global Shutter”. In: IEEE, June 2020, pp. 2502–2510 (cit. on p. 30).

[39] Stefan Lauxtermann, Adam Lee, John Stevens, and Atul Joshi. “Comparison of global shutter pixels for CMOS image sensors”. In: Jan. 2007, p. 8 (cit. on p. 30).

[40] Per-Erik Forssen and Erik Ringaby. “Rectifying rolling shutter video from hand-held devices”. In: IEEE, June 2010, pp. 507–514 (cit. on p. 30).

[41] Minoru Asada, Takamaro Tanaka, and Koh Hosoda. “Visual tracking of unknown moving object by adaptive binocular visual servoing”. In: IEEE, Feb. 1999, pp. 249–254 (cit. on p. 31).

[42] Leonid Keselman, John Iselin Woodfill, Anders Grunnet-Jepsen, and Achintya Bhowmik. “Intel(R) RealSense(TM) Stereoscopic Depth Cameras”. In: IEEE, July 2017, pp. 1267–1276 (cit. on p. 31).

[43] Faraj Alhwarin, Alexander Ferrein, and Ingrid Scholl. *IR Stereo Kinect: Improving Depth Images by Combining Structured Light with IR Stereo*. 2014 (cit. on p. 31).

[44] Hussein Haggag, Mohammed Hossny, Despina Filippidis, et al. “Measuring depth accuracy in RGBD cameras”. In: IEEE, Dec. 2013, pp. 1–7 (cit. on p. 31).

[45] Behnam Behroozpour, Phillip A. M. Sandborn, Ming C. Wu, and Bernhard E. Boser. “Lidar System Architectures and Circuits”. In: *IEEE Communications Magazine* 55 (10 Oct. 2017), pp. 135–142 (cit. on p. 32).

[46] Hauke Strasdat, J.M.M. Montiel, and Andrew J. Davison. “Visual SLAM: Why filter?” In: *Image and Vision Computing* 30 (2 Feb. 2012), pp. 65–77 (cit. on p. 32).

[47] Ilham Arun Faisal, Tito Waluyo Purboyo, and Anton Siswo Raharjo Ansori. “A Review of Accelerometer Sensor and Gyroscope Sensor in IMU Sensors on Motion Capture”. In: *Journal of Engineering and Applied Sciences* 15 (3 Nov. 2019), pp. 826–829 (cit. on p. 32).

[48] Wangyan Li, Zidong Wang, Guoliang Wei, et al. “A Survey on Multisensor Fusion and Consensus Filtering for Sensor Networks”. In: *Discrete Dynamics in Nature and Society* 2015 (2015), pp. 1–12 (cit. on p. 32).

[49] Chaithanya Mummadi, Frederic Leo, Keshav Verma, et al. “Real-Time and Embedded Detection of Hand Gestures with an IMU-Based Glove”. In: *Informatics* 5 (2 June 2018), p. 28 (cit. on p. 33).

[50] Teresa Vidal-Calleja, Juan Andrade-Cetto, and Alberto Sanfeliu. “Conditions for suboptimal filter stability in SLAM”. In: IEEE, Jan. 2004, pp. 27–32 (cit. on p. 33).

[51] Gabriel Nützi, Stephan Weiss, Davide Scaramuzza, and Roland Siegwart. “Fusion of IMU and Vision for Absolute Scale Estimation in Monocular SLAM”. In: *Journal of Intelligent & Robotic Systems* 61 (1-4 Jan. 2011), pp. 287–299 (cit. on p. 33).

[52] Carlos Campos, Richard Elvira, Juan J. Gomez Rodriguez, Jose M. M. Montiel, and Juan D. Tardos. “ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM”. In: *IEEE Transactions on Robotics* 37 (6 Dec. 2021), pp. 1874–1890 (cit. on pp. 33, 41).

[53] Lukas Von Stumberg, Vladyslav Usenko, and Daniel Cremers. “Direct Sparse Visual-Inertial Odometry Using Dynamic Marginalization”. In: IEEE, May 2018, pp. 2510–2517 (cit. on pp. 33, 41, 44).

[54] Nelson Acosta and Juan Toloza. “Techniques to improve the GPS precision”. In: *International Journal of Advanced Computer Science and Applications* 3 (8 2012) (cit. on p. 33).

[55] Dániel Kiss-Illés, Cristina Barrado, and Esther Salamí. “GPS-SLAM: An Augmentation of the ORB-SLAM Algorithm”. In: *Sensors* 19 (22 Nov. 2019), p. 4973 (cit. on p. 33).

[56] Thomas Olutoyin Oshin, Stefan Poslad, and Athen Ma. “Improving the Energy-Efficiency of GPS Based Location Sensing Smartphone Applications”. In: IEEE, June 2012, pp. 1698–1705 (cit. on p. 33).

[57] Junichi Ido, Yoshinao Shimizu, Yoshio Matsumoto, and Tsukasa Ogasawara. “Indoor Navigation for a Humanoid Robot Using a View Sequence”. In: *The International Journal of Robotics Research* 28 (2 Feb. 2009), pp. 315–325 (cit. on p. 34).

[58] Haomin Liu, Guofeng Zhang, and Hujun Bao. “Robust Keyframe-based Monocular SLAM for Augmented Reality”. In: IEEE, Sept. 2016, pp. 1–10 (cit. on p. 34).

[59] Khalid Yousif, Alireza Bab-Hadiashar, and Reza Hoseinnezhad. “An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics”. In: *Intelligent Industrial Systems* 1 (4 Dec. 2015), pp. 289–311 (cit. on p. 34).

[60] Tao Peng, Dingnan Zhang, Ruixu Liu, Vijayan K. Asari, and John S. Loomis. “Evaluating the Power Efficiency of Visual SLAM on Embedded GPU Systems”. In: IEEE, July 2019, pp. 117–121 (cit. on p. 34).

[61] J LaViola, B Williamson, Conner Brooks, et al. “Using augmented reality to tutor military tasks in the wild”. In: Dec. 2015 (cit. on p. 34).

[62] Patrik Schmuck and Margarita Chli. “Multi-UAV collaborative monocular SLAM”. In: IEEE, May 2017, pp. 3863–3870 (cit. on p. 34).

[63] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. “MonoSLAM: Real-Time Single Camera SLAM”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29 (6 June 2007), pp. 1052–1067 (cit. on pp. 40–42).

[64] Martin A. Fischler and Robert C. Bolles. “Random sample consensus”. In: *Communications of the ACM* 24 (6 June 1981), pp. 381–395 (cit. on pp. 40, 57).

[65] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. “EPnP: An Accurate O(n) Solution to the PnP Problem”. In: *International Journal of Computer Vision* 81 (2 Feb. 2009), pp. 155–166 (cit. on p. 40).

[66] Mikael Persson and Klas Nordberg. *Lambda Twist: An Accurate Fast Robust Perspective Three Point (P3P) Solver*. 2018 (cit. on p. 40).

[67] David Nistér. “An efficient solution to the five-point relative pose problem”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26 (6 June 2004), pp. 756–770 (cit. on p. 40).

[68] Richard I. Hartley and Peter Sturm. “Triangulation”. In: *Computer Vision and Image Understanding* 68 (2 Nov. 1997), pp. 146–157 (cit. on p. 40).

[69] Bo Jin and Feng Yang. “An overview of SLAM”. In: ed. by Yueguang Lv. SPIE, Aug. 2018, p. 26 (cit. on p. 41).

[70] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. “SVO: Fast semi-direct monocular visual odometry”. In: IEEE, May 2014, pp. 15–22 (cit. on pp. 41, 43).

[71] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassir Navab. “CNN-SLAM: Real-Time Dense Monocular SLAM with Learned Depth Prediction”. In: IEEE, July 2017, pp. 6565–6574 (cit. on p. 41).

[72] Shing Yan Loo, Ali Jahani Amiri, Syamsiah Mashohor, Sai Hong Tang, and Hong Zhang. “CNN-SVO: Improving the Mapping in Semi-Direct Visual Odometry Using Single-Image Depth Prediction”. In: (Oct. 2018) (cit. on pp. 41, 44, 59, 60).

[73] H. Strasdat, J. M. M. Montiel, and A. Davison. “Scale Drift-Aware Large Scale Monocular SLAM”. In: Robotics: Science and Systems Foundation, June 2010 (cit. on p. 42).

[74] Hauke Strasdat, Andrew J. Davison, J.M.M. Montiel, and Kurt Konolige. “Double window optimisation for constant time visual SLAM”. In: IEEE, Nov. 2011, pp. 2352–2359 (cit. on pp. 42, 71).

[75] Fangwen Shu, Jiaxuan Wang, Alain Pagani, and Didier Stricker. “Structure PLP-SLAM: Efficient Sparse Mapping and Localization using Point, Line and Plane for Monocular, RGB-D and Stereo Cameras”. In: (July 2022) (cit. on p. 42).

[76] C. Harris and M. Stephens. “A Combined Corner and Edge Detector”. In: Alvey Vision Club, 1988, pp. 23.1–23.6 (cit. on p. 50).

[77] Jianbo Shi and Tomasi. “Good features to track”. In: IEEE Comput. Soc. Press, 1994, pp. 593–600 (cit. on p. 50).

[78] Arturo Gil, Oscar Martinez Mozos, Monica Ballesta, and Oscar Reinoso. “A comparative evaluation of interest point detectors and local descriptors for visual SLAM”. In: *Machine Vision and Applications* 21 (6 Oct. 2010), pp. 905–920 (cit. on p. 50).

[79] Steffen Gauglitz, Tobias Höllerer, and Matthew Turk. “Evaluation of Interest Point Detectors and Feature Descriptors for Visual Tracking”. In: *International Journal of Computer Vision* 94 (3 Sept. 2011), pp. 335–360 (cit. on p. 50).

[80] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. “ORB: An efficient alternative to SIFT or SURF”. In: IEEE, Nov. 2011, pp. 2564–2571 (cit. on pp. 50, 51).

[81] Yenewondim Biadgie and Kyung-Ah Sohn. “Feature Detector Using Adaptive Accelerated Segment Test”. In: IEEE, May 2014, pp. 1–4 (cit. on p. 50).

[82] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. *BRIEF: Binary Robust Independent Elementary Features*. 2010 (cit. on pp. 50, 51).

[83] Michael Calonder, Vincent Lepetit, Mustafa Ozuysal, et al. “BRIEF: Computing a Local Binary Descriptor Very Fast”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34 (7 July 2012), pp. 1281–1298 (cit. on pp. 50, 51).

[84] Yan Ke and Rahul Sukthankar. “PCA-SIFT: a more distinctive representation for local image descriptors”. In: IEEE, May 2004, pp. 506–513 (cit. on p. 51).

[85] Marius Muja and David G. Lowe. “Fast Matching of Binary Features”. In: IEEE, May 2012, pp. 404–410 (cit. on p. 52).

[86] D. Galvez-López and J. D. Tardos. “Bags of Binary Words for Fast Place Recognition in Image Sequences”. In: *IEEE Transactions on Robotics* 28 (5 Oct. 2012), pp. 1188–1197 (cit. on p. 52).

[87] Ouk Choi and In So Kweon. “Robust feature point matching by preserving local geometric consistency”. In: *Computer Vision and Image Understanding* 113 (6 June 2009), pp. 726–742 (cit. on p. 52).

[88] Larry D. Kirkpatrick, Gregory E. Francis, and Robert Powell. *Physics: A World View (6th Edition)*. 2006 (cit. on p. 53).

[89] Rainer Kummerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. “G^{2}o: A general framework for graph optimization”. In: IEEE, May 2011, pp. 3607–3613 (cit. on pp. 54, 91, 107).

[90] Olivier Faugeras and F. Lustman. “Motion and Structure from Motion in a Piecewise Planar Environment”. In: *International Journal of Pattern Recognition and Artificial Intelligence* 02 (03 Sept. 1988), pp. 485–508 (cit. on p. 57).

[91] Bruce D Lucas and Takeo Kanade. *An iterative image registration technique with an application to stereo vision*. Vol. 81. Vancouver, 1981 (cit. on p. 61).

[92] Oleksandr Zadorozhnyi, Gunthard Benecke, Stephan Mandt, Tobias Scheffer, and Marius Kloft. *Huber-Norm Regularization for Linear Prediction Models*. 2016 (cit. on p. 61).

[93] J. Civera, A.J. Davison, and J. Montiel. “Inverse Depth Parametrization for Monocular SLAM”. In: *IEEE Transactions on Robotics* 24 (5 Oct. 2008), pp. 932–945 (cit. on pp. 61, 74).

[94] Edward H Adelson, Charles H Anderson, James R Bergen, Peter J Burt, and Joan M Ogden. “Pyramid methods in image processing”. In: *RCA engineer* 29 (6 Nov. 1984), pp. 33–41 (cit. on p. 64).

[95] Alan Parker. *Algorithms and Data Structures in C++*. Routledge, May 2018 (cit. on p. 76).

[96] Team Parallel HashMap Development. *The Parallel Hashmap*. 2019 (cit. on pp. 78, 82).

[97] Team Sparsepp Development. *Sparsepp: A fast, memory efficient hash map for c++*. 2018 (cit. on p. 78).

[98] ISO. *ISO IEC 14882:2017 Information technology — Programming languages — C++*. Fifth. Dec. 2017, p. 1605 (cit. on p. 78).

[99] Thomas Wang. *Webpage on "Integer Hash Function" (https://gist.github.com/badboy/6267743)* (cit. on p. 78).

[100] Bob Jenkins. *Webpage on "Integer Hashing" (https://burtleburtle.net/bob/hash/integer.html)* (cit. on p. 78).

[101] James E. McDonough. *Observer Design Pattern*. 2017 (cit. on p. 78).

[102] Marius Muja and David G. Lowe. “Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration”. In: *International Conference on Computer Vision Theory and Application VISSAPP’09*. INSTICC Press, 2009, pp. 331–340 (cit. on p. 89).

[103] Hauke Strasdat, J.M.M. Montiel, and Andrew J. Davison. “Visual SLAM: Why filter?” In: *Image and Vision Computing* 30 (2 Feb. 2012), pp. 65–77 (cit. on p. 95).

[104] Clement Godard, Oisin Mac Aodha, Michael Firman, and Gabriel Brostow. “Digging Into Self-Supervised Monocular Depth Estimation”. In: IEEE, Oct. 2019, pp. 3827–3837 (cit. on p. 109).

[105] Kaiming He, Jian Sun, and Xiaoou Tang. “Guided Image Filtering”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35 (6 June 2013), pp. 1397–1409 (cit. on p. 119).

[106] José Luis Blanco-Claraco. “A tutorial on SE(3) transformation parameterizations and on-manifold optimization”. In: (Mar. 2021) (cit. on p. 142).

A

Theoretical Optimization Background

At each iteration, SLAM programs need to estimate poses and map-points coordinates. These estimations are done by minimizing a sum of error functions, one for each frame-point pair to optimize. These error functions describe the difference between what is seen and what is estimated. For example, between the projection of a 3D point, and where it should have been projected.

Let $\boldsymbol{\beta}$ be the current estimation of any poses or points, $\mathbf{e}$ be a vector containing each residual of each error functions, $\mathbf{J}$ and $\mathbf{H}$ be the Jacobian and the Hessian of $\mathbf{e}$ according to $\boldsymbol{\beta}$.

The Gauss-Newton method is an iterative process to decrease the sum of errors. At each iteration, it computes a step $\delta\boldsymbol{\beta}$ that is added to $\boldsymbol{\beta}$ to decrease the sum of errors.

$$\begin{aligned} \mathbf{H} &= (\mathbf{J}^\top \mathbf{J}), \\ \mathbf{g} &= -\mathbf{J}^\top \mathbf{e}, \\ \mathbf{H}\delta\boldsymbol{\beta} &= \mathbf{g}. \end{aligned} \tag{A.1}$$

To compute the Hessians and Jacobians, the derivative of the pose of the camera is needed. These derivatives are explained in the next notasection.

$\mathbb{SE}(3)$ and $\mathfrak{se}(3)$ group

The $\mathbb{SE}(3)$ **group** contains all the 3D rotations. These rotations can be represented as 3×3 orthogonal matrix, angle-axis, or quaternions. The $\mathbb{SE}(3)$ **group** contains any 3D transformation composed of 3D rotations and translations. These transformations can be represented as the composition of an element of $\mathbb{SE}(3)$ and a translation, or by a 4×4 matrix:

$$\mathbf{v} = \begin{pmatrix} \mathbf{R}_{3\times3} & \mathbf{t}_{1\times3} \\ 0 & 1 \end{pmatrix} \quad \mathbf{R} \in \mathbb{SE}(3), \mathbf{t} \in^3 \tag{A.2}$$

A problem of $\mathbb{SE}(3)$ derivative, is the over-parametrization (a quaternion is composed of 4 values, but $\mathbb{SE}(3)$ elements have only 3 degrees of freedom), making it difficult to optimize. To solve this problem, poses are not optimized in the $\mathbb{SE}(3)$ group, but in the tangential space $\mathfrak{se}(3)$.

The tangential space of $\mathbb{SE}(3)$ is denoted by $\mathfrak{so}(3)$ and the tangential space of $\mathbb{SE}(3)$ is denoted by $\mathfrak{se}(3)$. $\mathfrak{so}(3)$ is an isomorphic lie group containing all skew-symmetric 3×3 matrices. The exponential map is a function converting $\mathfrak{se}(3)$ to $\mathbb{SE}(3)$, and the logarithmic map is a function converting $\mathbb{SE}(3)$ to $\mathfrak{se}(3)$.

$$exp : \mathfrak{se}(3) \to \mathbb{SE}(3) \qquad\qquad log : \mathbb{SE}(3) \to \mathfrak{se}(3) \tag{A.3}$$

To avoid over-parametrization, most algorithms optimize with $\mathfrak{se}(3)$ parametrized pose.

Optimization in $\mathbb{SE}(3)$

To simplify further derivatives, it would be convenient to continue to use $\mathbb{SE}(3)$ parametrization.

Let $\mathbf{v} \in \mathfrak{se}(3)$, $\mathbf{v} = exp(\mathfrak{v})$, and $f : \mathbb{SE}(3) \to$. The derivative of $exp(\mathfrak{v}) = \mathbf{v}$ according to $\mathfrak{v}$ is known and written in "A tutorial on $\mathfrak{se}(3)$ transformation parameterizations and on-manifold optimization" by Blanco and Jose-Luis [106]. Therefore, any derivable function $f : \mathbb{SE}(3) \to$ can be used to optimize an $\mathfrak{v} \in \mathfrak{se}(3)$:

$$\frac{\mathrm{d}f(\mathbf{v})}{\mathrm{d}\mathfrak{v}} = \frac{\mathrm{d}f(\mathbf{v})}{\mathrm{d}\mathbf{v}} \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}\mathfrak{v}} \tag{A.4}$$

In all the algorithms of the manuscrit, the optimization processes are done in the $\mathfrak{se}(3)$ space. However, the derivatives are computed in the $\mathbb{SE}(3)$ space because it is more convenient.