DCSLAM : un SLAM temps réel à contraintes dynamiques

D. Ramadasan[1] M. Chevaldonné[2] T. Chateau[1]

[1] Institut Pascal
[2] ISIT

24 avenue des Landais, Aubière
datta.ramadasan@univ-bpclermont.fr

Résumé

La localisation d'une caméra vidéo en temps réel dans un environnement inconnu ou partiellement connu est un problème abordé par les algorithmes de type CSLAM (Constrained Simultaneous Localization And Mapping). Ceux-ci introduisent au SLAM classique de nouvelles contraintes pour déterminer avec plus de précision la pose de la caméra et la structure 3D de l'environnement. Toutefois, les difficultés d'implémentation restreignent ces approches à l'utilisation d'une ou deux contraintes. Afin de dépasser ce problème, nous proposons un nouvel algorithme de CSLAM temps réel, nommé DCLSAM (SLAM à contraintes dynamiques), conçu pour s'adapter dynamiquement, à chaque optimisation, au nombre variable de familles de paramètres et de contraintes hétérogènes. Nous utilisons pour cela une méthode permettant de générer automatiquement, à partir d'une liste exhaustive de contraintes, un algorithme d'optimisation spécialisé au problème. C'est, à notre connaissance, la seule implémentation qui allie à la fois flexibilité et performance. Les expérimentations proposées montrent la pertinence de notre approche en terme de précision et de temps d'exécution par rapport à l'état de l'art sur plusieurs benchmarks publics de complexités différentes. Une application de réalité augmentée mixant des objets et des contraintes hétérogènes est également proposée.

Mots Clef

SLAM, temps réel, ajustement de faisceaux, réalité augmentée, méta-programmation.

Abstract

The real-time localization of a camera in an unknown or partially known environment is a problem addressed by CSLAM algorithms (it Constrained Simultaneous Localization And Mapping). They introduce new constraints to conventional SLAM to determine more accurately the pose of the camera and the 3D structure of the environment. However, implementations difficulties restrict these approaches to the use of one or two constraints. To overcome these difficulties, we propose a new real-time CSLAM algorithm (Constrained Simultaneous Localization And Mapping), named DCSLAM, designed to dynamically adapt each optimization to the variable number of parameters families and heterogeneous constraints. An automatic method is used to generate, from an exhaustive list of constraints, a dedicated optimization algorithm. This is, to our knowledge, the only implementation that combines flexibility and performance. The proposed experiments show the effectiveness of our approach in terms of accuracy and execution time compared to the state of the art on several public benchmarks of varying complexity. An augmented reality application mixing heterogeneous objects and constraints is also available.

Keywords

SLAM, real-time, bundle adjustment, augmented reality, meta-programming.

1 Introduction

Les recherches récentes dans le domaine des méthodes de type SLAM sont très actives. Ces algorithmes sont utiles dans de nombreuses applications comme la robotique mobile ou la réalité augmentée. Ils se classent en deux principales catégories : les méthodes probabilistes utilisant des filtres Bayésien récursifs (filtres de Kalmans, filtres à particules,...) et les méthodes d'optimisation de type ajustement de faisceaux (AF). Toutes ces méthodes estiment la pose de la caméra et la structure 3D de la scène, à partir de contraintes de reprojection. Lorsque l'on ajoute des contraintes issues d'objets connus ou de mouvement de caméra en plus de celles issues de l'environnement, on parle alors de SLAM contraint ou CSLAM. Nous proposons, dans cet article, une nouvelle implémentation, nommée DCSLAM, utilisant un nombre variable de contraintes pour optimiser différentes familles de paramètres comme la pose de la caméra, des points 3D, des segments 3D, la dimension et la pose de différentes objets présents dans la scène. Les contraintes appliquées aux paramètres peuvent correspondre à des distances de points à points, de points à surface, de contours à surfaces, ou des contraintes liées à la géométrie épipolaire. Le nombre de contraintes et de

familles de paramètres n'étant pas forcément constant au cours du SLAM, il est indispensable que l'algorithme d'optimisation évolue en conséquence. Les bibliothèques d'optimisation de l'état de l'art, les plus connues étant Ceres [1] et g2o [2], permettent de gérer plusieurs types de contraintes mais cela se fait au détriment des performances tout en contraignant la modélisation du problème. C'est pourquoi, nous proposons également une nouvelle bibliothèque d'optimisation, nommée LMA [3], basée sur la génération automatique de code à la compilation. LMA est capable de générer un solveur fortement spécialisé pour la résolution de problèmes de moindre carré non-linéaire de type Levenberg-Marquardt. L'implémentation proposée se présente sous la forme d'une librairie C++, librement distribuée, permettant une grande liberté de modélisation tout en supportant un nombre variable de familles de paramètres et de contraintes.

Après un état de l'art des travaux connexes (section 2), la bibliothèque d'optimisation proposée est présentée (section 3). Le DCSLAM est détaillé par la suite section (4) et un *benchmark* d'optimisation ainsi que l'utilisation du DCSLAM dans le cadre de la réalité augmentée seront exposés dans la section 5.1.

2 Positionnement bibliographique

Cette section dresse un panorama des travaux récents autour du CSLAM par vision monoculaire ainsi que les différentes bibliothèques utilisables pour l'ajustement de faisceaux (AF).

2.1 Les bibliothèques d'optimisation pour l'ajustement de faisceaux

L'AF est un processus d'optimisation basé sur la minimisation d'une fonction de coût correspondant le plus souvent à l'erreur de re-projection. Implémenter un AF de manière efficace est un problème difficile. Lourakis *et al.* [15] expliquent ce problème en détaillant la structure interne de la matrice hessienne (approximée par $J^T J$, J étant la jacobienne de la fonction de coût) ainsi que la résolution des équations normales en utilisant les propriétés éparses du problème. Ils montrent qu'il est possible d'éluder les valeurs nulles et de voir la hessienne comme un ensemble de matrices de petite taille afin de limiter l'espace mémoire requis et d'éviter les calculs inutiles. Ils détaillent également l'utilisation du complément de Schur pour réduire la complexité des problèmes de petites et moyennes dimensions. Toutefois, le complément de Schur est le résultat d'un produit matrice par matrice ce qui devient pénalisant pour les problèmes de grande taille. Dans le comparatif [2] sur des problèmes de grande taille (jusqu'à 4500 poses caméras), Agarwal *et al.* utilisent la méthode du complément de Schur implicite. Cette méthode consiste à approximer la résolution des équations normales avec l'algorithme du Gradient Conjugué Pré-conditionné (PCG). Il est alors possible d'évaluer le complément de Schur implicitement à chaque itération du PCG sous la forme de produits matrice par vecteur.

Plusieurs bibliothèques *opensource* ont été écrites avec les langages C ou C++ qui offrent de bonnes performances. En 2004, Lourakis *et al.* ont développé la première bibliothèque populaire dédiée à l'AF : sba. Celle-ci offre une résolution partiellement éparse du problème. L'astuce du complément de Schur est utilisée de manière éparse afin de réduire la complexité algorithmique du problème. Puis une méthode dense de Cholesky issue du paquet LAPACK est utilisée pour résoudre les équations normales ce qui limite son utilisation aux problèmes de petites et moyennes dimensions.

La seconde bibliothèque populaire pour l'AF est un *framework* nommé g2o développé par Kummerle *et al.* [13] et dédié aux méthodes de SLAM par optimisation de graphe. Pour les problèmes à deux familles de paramètres (poses et points 3D), ce framework peut utiliser une structure éparse basée sur de petites matrices de taille statique. De plus, afin de résoudre différents types de problème d'optimisation, g2o utilise un mécanisme d'extension basé sur l'héritage pour gérer de nouveaux paramètres et de nouvelles fonctions d'erreurs. Ceci permet de pouvoir optimiser n'importe quel type d'objet tant que la paramétrisation associée est définie, mais nécessite tout de même une recopie des données à optimiser. Selon [13], cette bibliothèque surpasse sba en terme de performances et constitue une solution viable pour les problèmes de petites et moyennes dimensions.

La bibliothèque open-source qui est actuellement la plus populaire et la plus complète pour résoudre des problèmes de moindre carré non-linéaires se nomme Ceres. Elle est développée par Agarwal *et al.* [1], supportée par Google, et fait appel à des bibliothèques matricielles denses et éparses fortement optimisées comme Eigen [10] ou SuiteSparse [7]. Elle contient de nombreux algorithmes d'optimisation comme la minimisation par région de confiance et permet l'utilisation du complément de Schur implicite. De plus, Ceres peut gérer des fonctions de coût à plusieurs familles de paramètres et plusieurs contraintes avec une API simple et tout en offrant des bonnes performances sur les problèmes de petites, moyennes et grandes dimensions. Toutefois, les paramètres à optimiser doivent suivre un certain formalisme ce qui peut nécessiter des conversions de données.

La bibliothèque qui est actuellement la plus performante pour les problèmes d'AF de grande taille est *pba*. Elle est développée par Wu *et al.* [25] avec pour objectif d'exploiter les architectures parallèles à travers le SIMD, le multiprocessus et le GPU. Même si elle est plus performante sur les problèmes de moyennes et grandes tailles utilisées pour le comparatif 3, cette bibliothèque est conçue pour ne résoudre que l'AF à deux familles de paramètres. Elle ne sera donc pas utilisée dans le comparatif.

1. ceres-solver.org/
2. github.com/RainerKuemmerle/g2o
3. https://github.com/bezout/LMA

	g2o	Ceres	LMA
Type matrice générique			×
Type de paramètre générique	×		×
Fonction non virtuelle			×
Paramètre non virtuel		×	×
Multi-blocs statiques			×
Interface simple		×	×

TABLE 1 – Caractéristique des implémentations.

	g2o	Ceres	LMA
Dérivée automatique	×	×	×
Fonction robuste	×	×	×
Dense	×	×	×
Eparse		×	×
Schur Dense	×	×	×
Schur Eparse	×	×	×
Schur Implicite		×	×
Région de confiance		×	

TABLE 2 – Algorithmes pour l'optimisation.

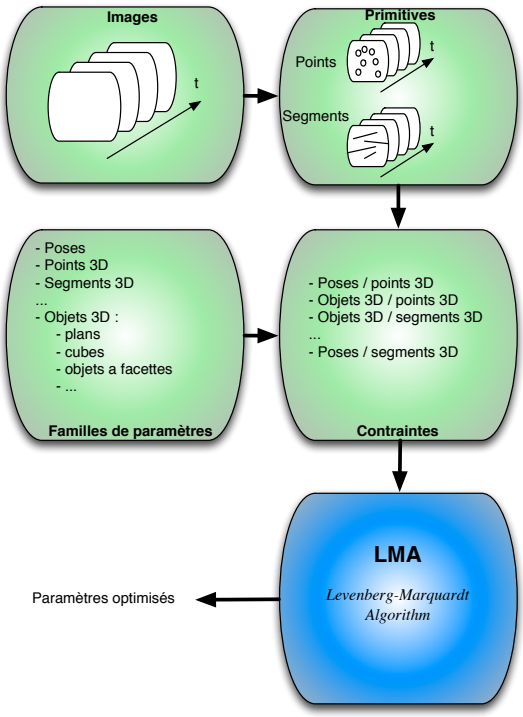

FIGURE 1 – Principe de fonction du DCSLAM : les primitives sont détectées dans les images et génèrent des contraintes avec les poses et les objets. La liste de contraintes est utilisée pour générer de manière automatique un AF avec la bibliothèque LMA.

Toutes les bibliothèques citées nécessitent une ré-écriture du problème à optimiser pour satisfaire un certain formalisme. De plus, g2o et Ceres utilisent le mécanisme d'héritage du langage pour assurer un niveau suffisant de généricité. Mais cela se fait au détriment des performances car l'utilisation du polymorphisme crée un niveau d'indirection supplémentaire dans les appels de fonctions et limite les possibilités d'optimisation du code par le compilateur. Pour dépasser les limitations de ces bibliothèques, nous avons développé une bibliothèque nommée *LMA* pour *Levenberg-Marquardt Algorithm* qui a pour principale caractéristique de facilement s'adapter à une grande variété de problèmes à travers une API simple sans utiliser de polymorphisme afin de conserver de bonnes performances. La table 1 résume les différences entre les bibliothèques g2o, Ceres et LMA, au niveau implémentation, et la table 2 liste les fonctionnalités disponibles.

2.2 Positionnement sur les méthodes de SLAM

De manière classique, le critère minimisé dans les méthodes de type SLAM [17, 11] est une erreur pixelique entre les points 3D projetés dans les images caméra et les observations correspondantes. D'autres approches utilisent des segments 3D seuls [8] pour les environnements faiblement texturés, ou en complément des points 3D [12] pour apporter de la robustesse au processus de localisation lors de mouvement rapide de la caméra. Toutefois, ces méthodes sont soumises à la dérive du facteur d'échelle et à l'accumulation d'erreurs. Pour améliorer la méthode, des travaux récents en vision monoculaire [9, 23, 19] utilisent des objets présents dans l'environnement pour contraindre les paramètres de reconstruction et augmenter la robustesse. Dans [23], Tamaazousti *et al.* ajoutent, dans l'AF introduit par [17], des contraintes provenant d'un modèle précis d'objet observé sur les points 3D associés au modèle. Toutefois, la pose de l'objet n'étant pas optimisée, cette méthode n'est pas adaptée à une application à plusieurs objets modélisés et initialisés approximativement. D'autres travaux utilisent des systèmes stéréo multi-vues [3] ou des capteurs de profondeur [24, 20, 5, 22] pour reconstruire des formes 3D. Dans [6], Dame *et al.* utilisent un SLAM dense pour construire une carte de profondeur en utilisant des *a priori* de formes 3D, ce qui nécessite beaucoup de calculs supplémentaires comparativement aux approches éparses. La figure 2 résume le positionnement de notre approche DCSLAM dans l'état de l'art. Dans ce travail, on propose d'inclure une quantité variable d'objets et de contraintes dans le DCSLAM en utilisant uniquement une caméra vidéo. A notre connaissance, ce travail est le premier à proposer d'inclure, dynamiquement au cours du SLAM, des contraintes et des objets dans un AF temps réel optimisant l'ensemble des paramètres (poses caméra, primitives 3D et objets 3D) en même temps. Le résultat est un suivi d'objet stable sans aucun tremblement visible sur la projection virtuelle des objets dans l'image. Le principe de fonctionnement du DCSLAM est illustré par la figure 1.

	RT	monoculaire	C-Pose	C-Objet
[3, 5]				×
[24]	×			×
[22]			×	×
[19, 21]		×		×
[4]	×	×		×
[20]	×		×	×
[6]		×	×	×
[23]	×	×	×	
DCSLAM	×	×	×	×

FIGURE 2 – DCSLAM par rapport à l'état de l'art. RT = temps réel, C-Pose = contrainte sur la pose caméra, C-Objet = contrainte sur l'objet.

Pour résumer, les principales contributions que nous proposons sont : 1) une nouvelle implémentation de l'algorithme de Levenberg-Marquardt dont les codes sources sont disponibles en *opensource*, 2) une évaluation quantitative des performances en terme de temps d'exécution et de précision de la librairie LMA par rapport à l'état de l'art, 3) une nouvelle approche pour la génération automatique de l'AF à partir d'une liste de contraintes pour une gestion dynamique des paramètres et contraintes.

3 LMA : une nouvelle implémentation de Levenberg-Marquardt

L'algorithme de Levenberg-Marquardt [14][16] est une méthode très populaire de moindre carrés non-linéaire pour résoudre des problèmes de vision par ordinateur. L'idée principale de notre implémentation est de fournir une interface simple avec un processus non intrusif d'adaptation au problème tout en conservant de bonnes performances d'exécution. LMA résout les équations normales soit avec l'algorithme de Cholesky dense (issue de Eigen [4]), soit avec une implémentation éparse du gradient conjugué préconditionné (PCG). Elle implémente également les astuces d'optimisation classiques liées à l'AF (comme le complément de Schur et le complément de Schur implicite) afin d'être efficace sur les problèmes de petites, moyennes et grandes dimensions. LMA agit comme un méta-programme (un programme qui en écrit un autre) utilisant les templates C++ pour analyser à la compilation le problème à optimiser à partir d'une liste de foncteurs C++ (un foncteur modélise une contrainte). Les paramètres sont déduits à partir des arguments des foncteurs et les degrés de liberté *ddl* sont définis par l'utilisateur. Puis LMA génère une structure de données pour stocker les foncteurs et les paramètres dans un container hétérogène (nommé tuple) adapté au nombre de familles de paramètres et de contraintes. La résolution des équations normales est générée de manière efficace en utilisant une représentation éparse composée d'ensembles de petites matrices de tailles statiques (*ie.* connues à la compilation). La dimension des matrices dépend des *ddl* des paramètres correspondant, et celles-ci sont stockées dans des tuples. Dans [18], Polok *et al.* montrent que l'implémentation des équations normales basée sur l'utilisation de matrices de tailles statiques offre de meilleures performances pour des problèmes de moindre carrés comparativement aux autres approches éparses utilisées par exemple dans Ceres. L'utilisation des tuples permet de ne pas avoir recours au polymorphisme ce qui permet la génération d'un programme plus efficace par le compilateur. De plus, LMA analyse à la compilation les dérivées croisées utiles à partir des liens entre les arguments des foncteurs afin d'éviter la génération de code correspondant à des dérivées nulles. De manière similaire, les parties symétriques et diagonales de la hessienne sont détectées à la compilation ce qui permet de spécialiser les stratégies de stockage et d'inversion. A l'exécution, les observations ainsi que les paramètres correspondant sont ajoutés au solveur. Un graphe de données est utilisé pour conserver les relations entre chaque paramètre (par exemple, le point d'intérêt i est lié au point 3D j vu dans la caméra k dont l'erreur est calculée par la fonction d'erreur). La politique de minimisation est gérée par l'algorithme de Levenberg-Marquardt qui met à jour chaque paramètre du problème lorsque l'erreur globale diminue.

4. eigen.tuxfamily.org

4 DCSLAM

Dans cette section, notre méthode de DCSLAM est illustrée à travers un exemple de SLAM faisant intervenir 8 familles de paramètres : poses caméra, primitives 3D (points 3D et segments 3D en utilisant la formalisation de Klein *et al.* [12]) et des objets 3D de différentes sortes (plan, sphère, parallélépipède, cylindre ou modèle à facette préalablement photo-modélisé avec le logiciel Acute3D). Dans le cadre de ces travaux, nous ne disposons pas des modèles exacts ni des poses précises des objets observés. Trois catégories de contraintes sont utilisées : reprojection entre poses et primitives 3D, contrainte épipolaire sur les poses, et distance 3D entre objets 3D et primitives 3D. Chaque famille de paramètres possède une paramétrisation propre facilitant l'application des contraintes associées. Par exemple, la contrainte planaire est paramétrée par le vecteur normal unitaire du plan. La paramétrisation des orientations des poses caméra, segments 3D et objets 3D utilise l'exponentiel map [11]. Le fonctionnement du DCSLAM est basé sur l'utilisation d'une liste exhaustive de contraintes et de paramètres organisée dans un graphe de dépendances (figure 3). Chaque contrainte possède des méthodes de création et suppression, et d'association aux paramètres. Les contraintes associées aux poses de la caméra et aux primitives 3D sont créées, associées, et supprimées selon le schéma classique reconstruction incrémentale [17]. Dans ce qui suit, nous nous intéresserons aux méthodes de création et de suppression des contraintes (section 4.1) puis à la méthode d'association des primitives 3D

associées aux objets (section 4.2). Enfin, la résolution du graphe de dépendances sera détaillée sans la section 4.3.

4.1 Création et suppression d'objets 3D

La propriété dynamique du DCSLAM provient des méthodes de création et de suppression associées à chaque contrainte. Lorsqu'un objet 3D est observé, une contrainte associant l'objet et ses primitives 3D est créée et ajoutée au processus d'optimisation. Pour les objets 3D de l'environnement, la position, l'orientation, le facteur d'échelle et la forme sont initialisés grossièrement de manière semi-automatique et optimisés en ligne pendant le DCSLAM. L'initialisation consiste à faire correspondre, de manière grossière, un modèle 3D d'objet à un sous-ensemble de primitives 3D issus de la reconstruction. Au cours du DCSLAM, l'utilisateur de l'application peut utiliser une interface pour cliquer une enveloppe convexe dans l'image autour de l'objet. Les primitives 2D qui sont à l'intérieur de l'enveloppe convexe sont alors utilisées pour initialiser l'objet à partir des associations 2D−3D. Chaque classe d'objet (de la liste exhaustive de paramètres) est testée et celle qui donne le meilleur recalage 3D est conservée. Si aucune primitive 3D n'est associée à un objet 3D, celui-ci n'est plus contraint, il est donc supprimé du processus d'optimisation.

4.2 Association d'objets et de primitives 3D

L'objectif de DCSLAM consiste à intégrer dynamiquement dans le processus d'optimisation des contraintes apportées par les objets partiellement connus de la scène (*i.e* dont le modèle géométrique est grossièrement connu). Toutefois, les primitives 3D reconstruites peuvent correspondre soit à la partie inconnue de l'environnement, soit aux objets partiellement connus. Afin d'assurer la convergence du DCSLAM, une étape d'association des primitives 3D aux objets correspondants est nécessaire à chaque optimisation. Dans ces travaux, une primitive 3D P est associée à un objet π si la distance d_P entre P et la plus proche surface de l'objet π est inférieure à un seuil σ_π . Afin de choisir une valeur de σ_π invariante au facteur d'échelle de la reconstruction, celle-ci est choisie en fonction de la dimension D_π de l'objet π tel que $\sigma_\pi = D_\pi \times \lambda$ avec λ l'intensité de la contrainte fixée à 0.005. Si une association est créée, les paramètres de la primitives 3D P ainsi que les paramètres de l'objet π correspondant vont être optimisés afin de minimiser la distance d_P. Le processus d"optimisation doit donc combiner des erreurs pixeliques (erreur de reprojection) et des erreurs métriques (distance 3D entre les primitives et les objets 3D). L'homogénéisation de ces deux erreurs est assurée en pondérant les erreurs 3D par le coefficient σ_π .

4.3 Résolution du graphe de dépendances

Afin de conserver des performances temps réel, on étend le schéma d'AF incrémental décrit par [17] à l'utilisation de contraintes. Ainsi, les contraintes ne sont appliquées que sur les paramètres de reconstruction correspondants à la fin de la trajectoire du SLAM. La modularité du DCSLAM

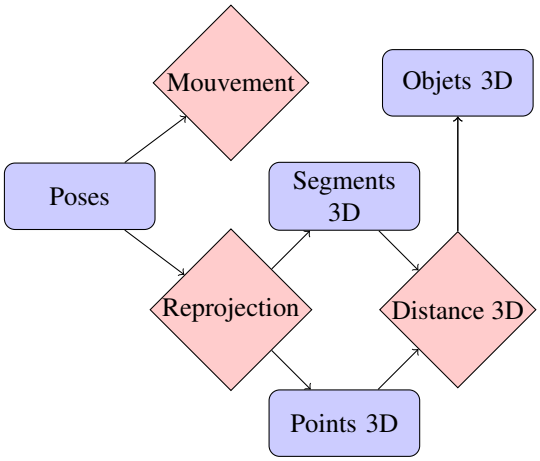

FIGURE 3 – Graphe de dépendances entre les contraintes et les paramètres.

provient de la formalisation du problème d'optimisation sous la forme d'un graphe de dépendances entre contraintes et paramètres. Ce graphe sert à générer un code où chaque contrainte résout ses dépendances avant de s'évaluer elle même. Les liens entre les paramètres et les contraintes sont résumés par la figure 3 : les objets 3D sont liés aux primitives 3D à travers la contrainte de minimisation de la distance 3D (entre la primitive et l'objet). Celle-ci s'applique sur les primitives qui ont été préalablement sélectionnées par la contrainte d'erreur de reprojection liée aux dernières poses du SLAM incrémental. Le solveur est généré par analyse, à la compilation, de la liste de contraintes composant le graphe de dépendances (à chaque contrainte correspondant un foncteur) ainsi que les paramètres associés. Il en résulte un solveur spécialisé au problème donné dont la fonction de coût à minimiser correspond à la somme des contraintes dynamiquement ajoutées.

5 Expérimentation

Les expérimentations sont effectuées sur un ordinateur de bureau équipé d'un processur i7 3.4GHz. La section qui suit 5.1 est une comparaison entre les 3 bibliothèques d'optimisation : g2o, Ceres et LMA. Les autres expérimentations illustrent le DCSLAM dans la section 5.1.

5.1 Benchmark

Trois comparatifs sont effectués sur des problèmes de différentes tailles en utilisant les solveurs g2o, Ceres-1.10 et LMA. Le premier comparatif est un problème simple visant à optimiser l'équation d'un cercle afin de minimiser la distance d'observations 2D à ce cercle. Le problème se compose de 3 *ddl.* et de 2000 observations. Chaque solveur itère 3 fois et le calcul des dérivées est numérique et centré. Après 3 itérations, l'erreur finale est la même pour les 3 solveurs. Le temps de calcul est de 2.2 ms pour g2o , de 2.4 ms pour Ceres, et de 0.51 ms pour LMA. Sur ce comparatif, LMA est 4.8 fois plus rapide que Ceres, et 4.3 plus

	g2o	Ceres	LMA
Eparse	∅	2.63403	**1.0668**
Schur dense	3.6029	2.36683	**1.01193**
Schur éparse	3.4901	2.4247	**1.10304**
Schur implicite	∅	2.89774	**1.1224**

TABLE 3 – Temps d'exécution (sec) sur un problème à 16 poses, 22106 points 3D, 83718 observations.

rapide que g2o. L'algorithme de résolution dense étant le même pour les 3 solveurs, la différence de performance est le résultat de l'absence d'héritage dans LMA.

La seconde expérimentation est effectuée sur une instance du jeu de données utilisé dans le comparatif de Ceres [2]. Chaque solveur est limité à 10 itérations de Levenberg-Marquardt, les calculs de dérivés sont numériques et centrés, et l'algorithme du PCG est limité à 20 itérations lorsqu'il est utilisé. Le tableau 3 montre les temps d'exécution des 3 solveurs sur l'instance à 16 poses, 22k points 3D et 83k du jeu de données. Sur cette instance, LMA est 2 à 3 fois plus rapide que Ceres et g2o, avec une précision semblable, sur les 4 algorithmes testés : Eparse (PCG éparse sans complément de Schur), Schur dense (Cholesky dense et complément de Schur), Schur éparse (PCG éparse et complément de Schur), Schur implicite (PCG éparse et complément de Schur implicite). g2o n'implémente ni la résolution éparse sans complément de Schur, ni la méthode implicite.

La troisième expérimentation s'effectue sur l'ensemble des 35 instances du jeu de données [2] (en utilisant la configuration décrite précédemment) et montrent que LMA est moyenne 2.5 plus rapide que Ceres, et plus précis 21 fois sur 35. Comparativement à g2o, LMA est 3.5 plus rapide avec une précision équivalente.

Ces résultats sont à pondérés : g2o est une bibliothèque dédiée à l'optimisation de graphe, et Ceres demeure une solution plus généraliste et plus riche au niveau des méthodes d'optimisation. Même s'il est possible de résoudre des problèmes denses avec LMA, son implémentation est spécialisée pour la résolution éparse. De plus, en utilisant le calcul de dérivées analytiques des jacobiennes, les différences de temps d'exécution entre les solveurs restent les mêmes mais Ceres montre une meilleur convergence grâce à la minimisation par région de confiance.

5.2 Application à la réalité augmentée

L'objectif de l'expérimentation qui suit est d'évaluer la sensibilité de notre méthode face aux initialisations grossières d'objets. Trois méthodes sont comparées : SLAM classique, CSLAM sans optimisation des objets, DCSLAM avec optimisation des objets. Pour ces expérimentations, le capteur utilisé est une caméra vidéo *global shutter* cadencée à 60 images par seconde avec une résolution de 640×480.

L'expérimentation, illustrée par la figure 4, compare les

FIGURE 4 – Comparaison visuelle entre le SLAM (à gauche), le CSLAM (au centre), et le DCSLAM proposé (à droite).

méthodes SLAM, CSLAM et DCSLAM sur une application de réalité augmentée où des modèles d'objets sont projetés dans les images. Cela permet d'évaluer visuellement la qualité du recalage, donc la bonne estimation des objets. Chaque méthode est évaluée sur la même séquence vidéo et les objets sont initialisés de manière identique. On remarque que l'erreur de recalage des objets avec le SLAM classique (images de gauche) est importante car aucune contrainte ne vient limiter l'accumulation d'erreur. Le résultat de l'approche CSLAM (images du centre) est meilleur que l'approche SLAM, mais l'erreur de projection des objets reste trop importante pour que l'approche soit exploitable. Si un seul objet était visible, la contrainte associée aurait modifié la trajectoire de la caméra et le recalage aurait été correct. Mais dans une configuration à plusieurs objets, il n'est pas possible de corriger la trajectoire pour satisfaire l'ensemble des contraintes objets. C'est pourquoi il est nécessaire de libérer les degrés de liberté de chaque objet afin que la reconstruction globale soit meilleure. Avec l'approche DCSLAM (images de droite), l'optimisation des objets a permis une meilleure reconstruction et l'ensemble des objets se projettent correctement dans l'image.

5.3 Performances de l'application

L'implémentation de l'algorithme de DCSLAM donne des performances temps réel. Sur une séquence d'images de 10k images, en utilisant des points 3D, des segments 3D, et 8 objets 3D de 5 classes différentes, tous visibles sur la séquence , les temps obtenus sont de 5 ms pour la détection (Harris+Canny) et l'appariemment, 1 ms pour le calcul de pose (combinant points et segments 3D) et 70 ms pour la mise à jour de la carte (processus effectué en parallèle). Une perspective d'amélioration des temps de calculs serait de calculer toutes les dérivées analytiquement dans l'AF incrémental (pour l'instant les dérivées sur les segments 3D sont numériques). NB : la bibliothèque LMA est utilisée pour l'AF lors de la mise à jour de la carte, mais également pour le calcul de pose combinant points et segments 3D, et

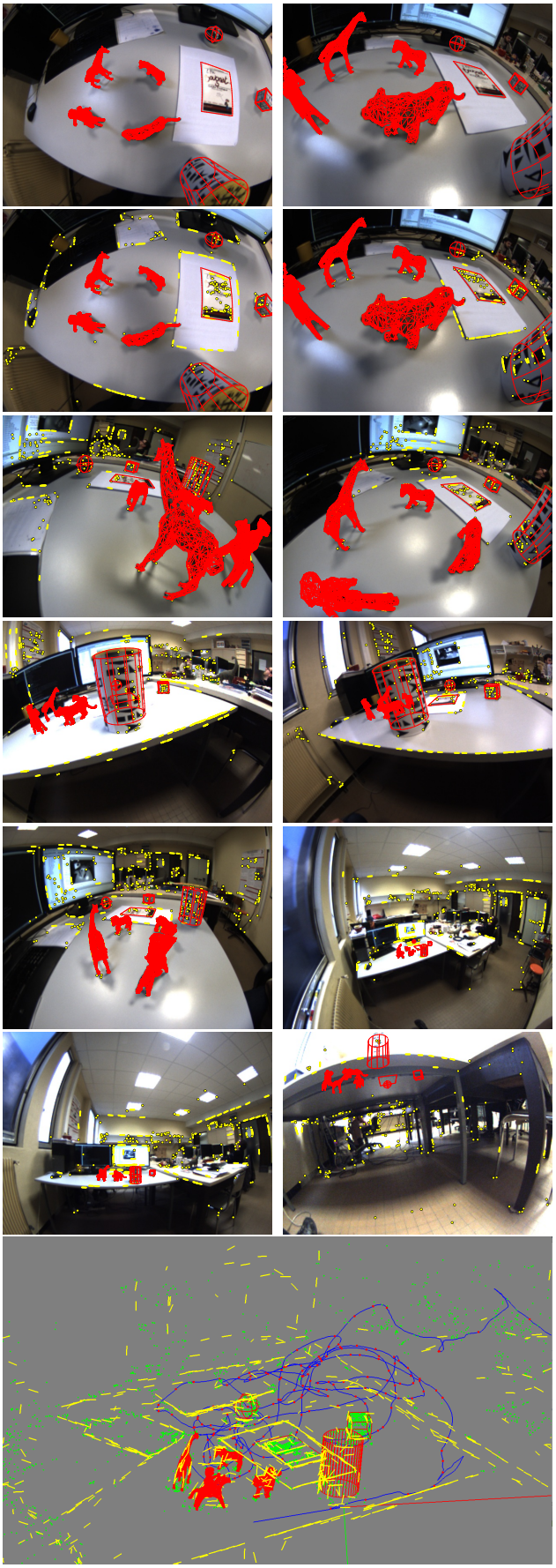

FIGURE 5 – Le DCSLAM pour la réalité augmentée.

pour l'initialisation segments et des objets 3D. Des vidéos illustrant le DCSLAM ont été enregistrées en temps réel et sont disponibles en ligne [5].

5.4 Perspectives d'améliorations

Sur la figure 5, on observe que la surface inférieure du cylindre n'est pas observée, donc pas contrainte, par conséquent mal recalée. Pour résoudre ce problème, il serait possible d'ajouter un objet de type *sol* dans le processus d'optimisation tel qu'aucun objet ne puisse être en dessous du sol ou en lévitation, mais ceci dépend fortement de l'application visée. Un autre cas d'échec est dû à la dérive du *SLAM* monoculaire. Lorsque que les objets ne sont pas observés, la reconstruction est faiblement contrainte et l'accumulation d'erreurs peut provoquer une dérive de la reconstruction. Ceci provoque un décalage, entre la position réelle des objets et leur position estimée, qui s'observe lorsque les objets sont de nouveaux dans le champ de la caméra. Toutefois, si la dérive n'est pas trop importante et que l'association entre primitives 3D et objets reste correcte, notre méthode permet aux objets d'être ré-optimisés en tenant compte des nouvelles observations. De plus, les segments 3D sont inutilisables sur certains objets courbes qui mettent en défaut l'appariement de contour. Pour finir, la détection d'un objet connu dans l'image pourrait être automatique en utilisant un détecteur d'objet multi-classe s'exécutant en parallèle du SLAM et proposant l'ajout de nouveaux objets.

6 Conclusion

L'approche DCSLAM proposée permet d'implémenter facilement des problèmes de SLAM avec un nombre variables de familles de paramètres et de contraintes. L'implémentation temps réel du DCSLAM est basée sur l'utilisation de la bibliothèque d'optimisation LMA dont les performances sont supérieures à l'état de l'art notamment sur des problèmes d'AF. La gestion des contraintes étant dynamique, la méthode peut s'adapter à différent types de scénarios, faisant intervenir différents types d'objets 3D. L'approche permet l'utilisation simultanée de contrainte de mouvement, contraintes géométriques et contrainte de reprojection sur des points et segments 3D. L'utilisation d'un nombre important de contraintes de nature différente apporte au processus de reconstruction une précision et une stabilité importante. L'application du DCSLAM à la réalité augmentée faisant intervenir plusieurs types d'objets et plusieurs types de contraintes a également illustré la précision et les performances temps réel de la méthode.

Remerciement

Ce travail est cofinancé par l'Union européenne. L'europe s'engage en Auvergne avec le Fonds européens de développement régional.

5. http://ip.univ-bpclermont.fr/index.php/fr/axes-de-recherche/ispr/18-comsee/18-comsee-videos

Références

[1] S. Agarwal and K. Mierle. Ceres solver : Tutorial & reference. *Google Inc*, 2, 2012.

[2] S. Agarwal, N. Snavely, S. M. Seitz, and R. Szeliski. Bundle adjustment in the large. In *Computer Vision–ECCV 2010*, pages 29–42. Springer, 2010.

[3] S. Y. Bao, M. Chandraker, Y. Lin, and S. Savarese. Dense object reconstruction with semantic priors. In *CVPR*, pages 1264–1271. IEEE, 2013.

[4] J. Bastian, B. Ward, R. Hill, A. van den Hengel, and A. Dick. Interactive modelling for ar applications. In *Mixed and Augmented Reality (ISMAR), 2010 9th IEEE International Symposium on*, pages 199–205. IEEE, 2010.

[5] Y. Chen and G. Medioni. Object modelling by registration of multiple range images. *Image and vision computing*, 10(3) :145–155, 1992.

[6] A. Dame, V. A. Prisacariu, C. Y. Ren, and I. Reid. Dense reconstruction using 3d object shape priors. In *CVPR*, pages 1288–1295. IEEE, 2013.

[7] T. Davis, W. Hager, and I. Duff. Suitesparse, 2013.

[8] E. Eade and T. Drummond. Edge landmarks in monocular slam. In *BMVC*, pages 7–16, 2006.

[9] A. P. Gee, D. Chekhlov, W. W. Mayol-Cuevas, and A. Calway. Discovering planes and collapsing the state space in visual slam. In *BMVC*, pages 1–10, 2007.

[10] B. Jacob and G. Guennebaud. Eigen is a c++ template library for linear algebra : Matrices, vectors, numerical solvers, and related algorithms, 2012.

[11] G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007.

[12] G. Klein and D. Murray. Improving the agility of keyframe-based slam. In *Computer Vision–ECCV 2008*, pages 802–815. Springer, 2008.

[13] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g 2 o : A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3607–3613. IEEE, 2011.

[14] K. Levenberg. A method for the solution of certain problems in least squares. *Quarterly of applied mathematics*, 2 :164–168, 1944.

[15] M. A. Lourakis and A. Argyros. SBA : A Software Package for Generic Sparse Bundle Adjustment. *ACM Trans. Math. Software*, 36(1) :1–30, 2009.

[16] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics*, 11(2) :431–441, 1963.

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

[18] L. Polok, S. V. Ila, and P. Smrž. Cache efficient implementation for block matrix operations. In *Proceedings of the 21st High Performance Computing Symposia*. Association for Computing Machinery, 2013.

[19] G. Reitmayr, E. Eade, and T. W. Drummond. Semiautomatic annotations in unknown environments. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 67–70. IEEE, 2007.

[20] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. Kelly, and A. J. Davison. Slam++ : simultaneous localisation and mapping at the level of objects. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 1352–1359. IEEE, 2013.

[21] G. Silveira, E. Malis, and P. Rives. The efficient e-3d visual servoing. *International Journal of Optomechatronics*, 2(3) :166–184, 2008.

[22] Y. Taguchi, Y.-D. Jian, S. Ramalingam, and C. Feng. Point-plane slam for hand-held 3d sensors. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5182–5189. IEEE, 2013.

[23] M. Tamaazousti, V. Gay-Bellile, S. Collette, S. Bourgeois, and M. Dhome. Nonlinear refinement of structure from motion reconstruction by taking advantage of a partial knowledge of the environment. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3073–3080. IEEE, 2011.

[24] T. Tykkala, A. I. Comport, and J.-K. Kamarainen. Photorealistic 3d mapping of indoors by rgb-d scanning process. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1050–1055. IEEE, 2013.

[25] C. Wu, S. Agarwal, B. Curless, and S. M. Seitz. Multicore bundle adjustment. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3057–3064. IEEE, 2011.