7H-06

Visual SLAM ソフトウェア高速化検討のための処理時間分析

後村 胤樹 † 大川 猛 †† 大津 金光 †† 馬場 敬信 ‡ 横田 隆史 ††
† 宇都宮大学工学部情報工学科 †† 宇都宮大学大学院工学研究科情報システム科学専攻
‡ 宇都宮大学オプティクス教育研究センター

1 はじめに

自律移動ロボット，AR，自動運転などの技術で重要な，環境地図の構築と自己位置の推定を同時に行う SLAM(Simultaneous Localization and Mapping) 技術が研究されている．SLAM は測域センサや USB カメラなどの外部センサと，ジャイロなどの内部センサから得られるデータを用いて処理を行うが，この外界センサとしてカメラのみを使用し，得た画像データのみを用いて自己位置推定と地図構築を行う SLAM を Visual SLAM と呼ぶ．Visual SLAM にカメラを用いる利点は，小型，人体への影響が小さい，安価で容易に手に入る等が挙げられる．Visual SLAM は一般的に，画像入力，特徴点抽出・記述，特徴追跡，自己位置推定，地図構築，地図修正の順に処理を行っていく．

Visual SLAM のうち ORB(Oriented FAST and Rotated BRIEF) 特徴量を用いた SLAM ソフトウェアに ORB-SLAM2 がある [1]．ORB 特徴量を用いる利点は，ORB 特徴量のデータサイズが小さいので，他の特徴量よりも処理時間を短くできることである．また，ORB-SLAM2 は Visual SLAM の中でも比較的新しいオープンソースの SLAM ソフトウェアであり，ある程度のリアルタイム性を実現している．しかし，リアルタイム性を持たせるために環境地図の精度を犠牲にしていることが課題となっている．

本研究では，ORB-SLAM2 において環境地図の高精度化とスループットの向上の両立を実現するために高速化を実現する．本稿では，ORB-SLAM2 の高速化のための前段階として，ORB-SLAM2 の処理時間を計測しボトルネックを明らかにする．

2 ORB-SLAM2 の内部構造

ORB-SLAM2 の主要なスレッドとその主な関数を図 1 に示す．ORB-SLAM2 は，図 1 中に示すように，処理を大きく 3 つに分け，それぞれ別のスレッドに実装し，パイプライン動作させることで，処理時間の削減を行っている．各スレッドが行う処理から，それぞれを Tracking，Mapping，Closing と呼ぶ．

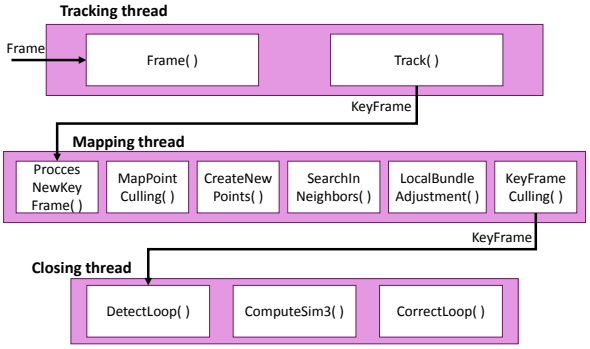

図 1: ORB-SLAM2 の主要スレッドとその実行関数

2.1 Tracking スレッド

このスレッドは，センサから得た入力画像 (フレーム) を用いて特徴点を抽出し，カメラ姿勢の推定を行い，キーフレームを作成する．他のスレッドとは異なり，全てのフレームに対して次の 2 つの関数を実行する．Frame() は ORB 特徴量の抽出を行う．Track() はカメラ姿勢を推定し，現在のマップと現在のフレームを照会して特徴点の対応 (マップポイント) を探す．次に，地図上に記録していない特徴点が，現在のフレーム上にある一定量存在していた場合，それをキーフレームとして Mapping スレッドに送る．

2.2 Mapping スレッド

このスレッドは，Tracking スレッドで作成したキーフレームを受け取って処理を行う．新しいキーフレームを取得したら，ProcessNewKeyFrame() でキーフレームの挿入，MapPointCulling() でロバストではないマップポイントを取り除き，CreateNewMapPoints() で新しいマップポイントを作成する．SearchInNeighbors() は隣接キーフレームとマップポイントからより多くの対応点を見つける．LocalBundleAdjustment() は今のフレームと隣接フレームのマップポイントとカメラの姿勢を補正する．KeyFrameCulling() では冗長なキーフレームを除去する．各関数は基本的に全てのキーフレームに対して実行される．

2.3 Closing スレッド

このスレッドは，Mapping スレッドで作成した地図上で一度通った道を再度訪れた場合 (ループと呼ぶ)，その時点までのカメラの軌跡とマップポイントを修正する．DetectLoop() はループが発生しているかを確認する．ループが検出されたら ComputeSim3() で

Processing time analysis for accelerating Visual SLAM software

†Kazuki Ushiromura, ††Takeshi Ohkawa,
††Kanemitsu Ootsu, ‡Takanobu Baba
††Takashi Yokota
Department of Information Science, Faculty of Engineering, Utsunomiya University (†)
Department of Information Systems Science, Graduate School of Engineering, Utsunomiya University (††)
Center for Optical Research and Education, Utsunomiya University (‡)

表 1: Tracking スレッドの処理時間

	呼出回数 (回)	総和 (sec)	平均 (sec)	最大 (sec)	最小 (sec)
フレーム 1 枚の処理	4541	185.677	0.041	1.502	0.027
Frame()	4541	132.589	0.029	0.055	0.021
Track()	4541	53.079	0.012	1.473	0.001

表 2: Mapping スレッドの処理時間

	呼出回数 (回)	総和 (sec)	平均 (sec)	最大 (sec)	最小 (sec)
キーフレーム 1 枚の処理	1528	314.375	0.206	0.870	0.020
ProcessNewKeyFrame()	1528	25.420	0.017	0.039	0.000
MapPointCulling()	1528	0.207	0.000	0.004	0.000
CreateNewMapPoints()	1528	63.204	0.041	0.087	0.015
SerachInNeighbors()	1527	72.282	0.047	0.178	0.004
LocalBundleAdjustment()	1525	147.747	0.097	0.583	0.013
KeyFrameCulling()	1526	5.512	0.004	0.028	0.000

表 3: Closing スレッドの処理時間

	呼出回数 (回)	総和 (sec)	平均 (sec)	最大 (sec)	最小 (sec)
キーフレーム 1 枚の処理	1527	18.668	0.012	2.259	0.000
DetectLoop()	1527	11.984	0.008	0.027	0.000
ComputeSim3()	244	0.272	0.001	0.018	0.000
CorrectLoop()	4	6.411	1.603	2.237	1.002

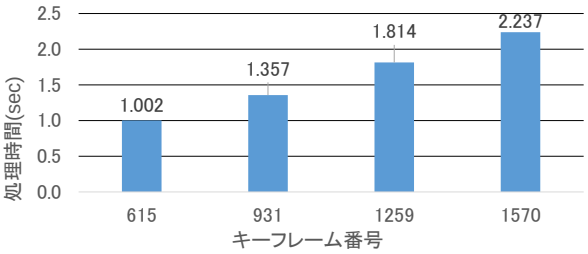

図 2: CorrectLoop() 実行時の処理時間の推移

隣接するキーフレーム間の対応する点を検出し両者の相似度を算出する．相似度が一定値以上であれば，CorrectLoop() でカメラの姿勢とマップポイントの修正を行う．CorrectLoop() はその実行条件によって，実行回数は他の関数よりも少なくなっている．

3 ORB-SLAM2 の処理時間計測

3.1 計測結果

ORB-SLAM2 を 1 台の PC 上で動作させ処理時間を計測した．実験に使用したマシンは，CPU : Intel Core i7-870@2.39GHz，メモリ : 12GB，OS : Ubuntu16.04 LTS(64 ビット) を用いた．また，入力データは，住宅街を車載カメラで約 10 分間走行した動画 (KITTI のサンプルデータセットのシーケンス 00) を使用した [2]．フレーム 1 枚のサイズは 376*1241(275.8 KB)，総フレーム数は 4541 枚である．

計測したのは，図 1 中において，フレーム (あるいはキーフレーム)1 枚に対する各スレッドの処理時間，及びその内部関数の処理時間である．

得られた結果から，各関数の呼び出し回数，1 つの動画の総処理時間，フレーム (あるいはキーフレーム)1 枚あたりの平均処理時間，最大･最小値を算出して表 1〜3 にまとめた．また，Closing スレッドの CorrectLoop() の処理時間に関しては図 2 に詳細を示す．

3.2 評価・考察

スレッド単位でフレーム (キーフレーム)1 枚の処理時間の総和を比較すると，Mapping スレッドが約 314sec であり，処理に一番時間が掛かっていることが分かる．また，Closing スレッドの CorrectLoop() の処理は，呼出回数は少ないが，他の関数と比較して時間がかかりすぎているので，高速化の効果が大きいと予想される．これらから，スレッド単位では Mapping スレッドが，関数単位では CorrectLoop() がボトルネックであることが分かる．この 2 つの部分を高速化することで ORB-SLAM2 の処理速度が向上すると考えられる．

ボトルネックのうち，特に CorrectLoop() については，図 2 から実行回数が増えるにつれて 1 回の処理時間が増加していることが分かる．このようになるのは，SLAM の実行時間が増加するにつれてカメラ姿勢の軌跡とマップポイントが増加していくため，修正する際に比較するデータサイズが増加していくからだと考えられる．このことから，SLAM を長時間実行させた場合は SLAM の実行時間につれて CorrectLoop() の処理時間は上昇していくことが予想される．従って，CorrectLoop() の処理時間を削減する為には，GPU を用いて並列化するだけではなく，関数のアルゴリズムを改善する必要があると考えられる．

4 おわりに

本稿では，ORB-SLAM2 ソフトウェアに対して，作成する地図精度と処理時間の両立するために高速化を検討する際のボトルネックを調べるために，スレッド毎に処理時間を計測した．今後は発見したボトルネックを対象に，GPU や FPGA 等を用いた高速化を行う予定である．

謝辞

本研究は，一部日本学術振興会科学研究費補助金 (基盤研究 (C)15K00068，同 (C)16K00068，同 (C)17K00072，同 (C)17K00265) の援助による．

参考文献

[1] Ral Mur-Artal and Juan D. Tards: “ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras”, IEEE Transactions on Robotics, Vol.33, No.5, pp.1255-1262, 2017.

[2] “The KITTI Vision Benchmark Suite”, http://www.cvlibs.net/datasets/kitti/eval_odometry.php, (2017/12/19 アクセス).