Proceedings of the 4th International Conference on Autonomous Robots and Agents, Feb 10-12, 2009, Wellington, New Zealand

自動キウイフルーツ収穫ロボットの開発

Development of An Autonomous Kiwifruit Picking Robot

A. J. Scarfe, R. C. Flemmer, H. H. Bakker and C. L. Flemmer
発表者：22S1023 高見俊介

The design concept and development status of an autonomous kiwifruit-picking robot is presented. The robot has an intelligent vision system that ensures that only 'good' fruit is picked. The robot receives instruction by radio link and operates autonomously as it navigates through the orchard, picking fruit,unloading full bins of fruit, fetching empty bins and protecting the picked fruit from rain. The robot has four picking arms, each of which will pick one fruit per second. To extend the useful annual work period of the robot, it is envisaged that it will also be used to pollinate kiwifruit flowers.

Key Words: autonomous, agricultural, robotic, kiwifruit, picker, picking robot, end effector, navigation, vision system, pick-rate, pollination, New Zealand

1　緒言

産業用ロボットの導入は広まっているが，農業へのロボットの導入事例は少ない．産業用ロボットが働く環境は比較的，清潔で乾燥しており不確定要素が少ない．一方，農場では照明や天候，地形など多様な変化が存在している．また，産業用ロボットが扱う対象物は十分堅牢で色，大きさ，形はすべて一様である．一方，農場で扱う農作物の種類によっては，色，大きさ，形が多様で，果実部分が葉に覆われていることが多く，ロボットが把持すると傷みやすい．

対して，手作業による農作物の収穫作業は身体的な負担が大きく，農作業に従事する労働力を確保し，維持することは慢性的な問題である．特に，農場で働く従業員に支払われる給与が比較的高い国々 (アメリカ，イタリア，イスラエル，オーストラリア，ニュージランド) では，農業に自動化ロボットを導入する経済的誘因政策がある．

本稿では，ニュージーランドで開発中の自動キウイフルーツ収穫ロボットの進捗ついて示す．同様に，イタリアで開発中の自動オレンジ収穫ロボットや，ベルギーで開発中の自動りんご収穫ロボットなどの農業用ロボットについても触れる．どちらも現在，商業的に実現可能なものではなく，収穫ロボットの位置決めには人間のオペレータが必要である．

2　関連研究

Mingas らは [3] でグローバルマップとローカルスキャンの間で ICP アルゴリズムを実行し，その結果を利用する SLAM を FPGA 上に実装している．Ratter らは [4] で GraphSLAM を並列に GPU 上に ICP ベースで実装し，計算を高速化している．

3　アプローチ方法

3.1　GraphSLAM

GraphSLAM は，環境をグラフとして表す SLAM アルゴリズムである．操作可能な GraphSLAM アルゴリズムは，

1. 状態ベクトルと情報行列からなる線形方程式系の作成
2. システム内のループクロージャを探す
3. ループクロージャの線形方程式のシステムへの組み込み
4. 方程式の線形システムを解き，状態ベクトルを更新
5. 環境の地図を作成

の手順で構築する．

ステップ 1，3，4 で表される誤差の収束に焦点を当てるため，ステップ 2 は手作業で行う．マッピングはステップ 5 で実行されるが，GraphSLAM の状態推定には含まれない．今回設計するシステムを用いて姿勢を推定することで姿勢が既知の状態でマッピングすることができる．

GraphSLAM アルゴリズムのステップ 1 と 3 では，ループクロージャによって誤差を収束するための線形システムを式 (1) として構築する．$\Delta\vec{x}$ に現在の状態ベクトル $\vec{x}$ に適用される状態変化を含む．状態ベクトル内の状態間の関係は情報行列 $\mathbf{H}$ で，見つかる誤差は誤差ベクトル $\vec{b}$ で表す．

$$\mathbf{H}\Delta\vec{x} = -\vec{b} \tag{1}$$

誤差の合計は，式 (1) の線形システムを反復的に解くことによって最小にすることができる．

3.2　FPGA への実装

まず数学的記述から始め，次にこれらの記述を関数型言語で実装する．CλaSH の文法や構文は関数型言語である Haskell を元に構成されており，組み合せ回路と同期回路の両方に対して構造的な設計アプローチをとることができる．設計プロセスは，

A. 数学的解法/アルゴリズムを選択
B. Haskell で数学的記述を指定して CλaSH に変換
C. ハードウェア制約を決定
D. 面積-時間トレードオフを作成
E. シミュレーションと合成

の 5 つの手順に分かれている．ステップ A では問題の解について数学的に定義する．情報行列 $\mathbf{H}$ には通常多くのゼロが含まれているため，スパース表現にするとメモリ使用量を減らすことができる．FPGA 上で GraphSLAM を効率的に実装するには，アルゴリズムが決定論的である必要がある．再利用しやすい規則的な構造となることが望ましい．ステップ B では Haskell で数学的記述を指定した後，Haskell インタプリタを使用して記述をシミュレートする．FPGA の仕様上の限界に達しておらず，完全な並列設計が可能な場合は，直接ステップ E に進み，CλaSH コンパイラで HDL を生成する．ステップ C ではシステムのどの部分がボトルネックになるかを決定する．FPGA は面積と時間によって制約を受ける．面積によって FPGA が論理要素またはデジタル信号プ

ロセッサ (DSP) で作成された演算子で並列に実行できる操作の数が決定される．時間は FPGA が動作するクロック周波数によって定義され，組合せパスの長さによって定義される．できる限り多くの計算資源を並列に使用するような機能設計が要求されるが，FPGA で並列計算可能なリソースより多い場合には並列に処理できない分を逐次的に計算することで解決する．ステップ D では使用するハードウェアの使用率を高めるため，すべてのアルゴリズムの全体的な構造を見て，時間の経過とともに再利用されるハードウェアの規則的な部分を決定する．ステップ E では CλaSH コンパイラが FPGA 上で直接合成可能な HDL を自動的に生成する．以上の 5 ステップをハードウェアに GraphSLAM を記述するために用いる．

4　評価

設計したシステムのシミュレーションを行い，GraphSLAM でループを閉じる前と閉じた後を比較した．さらに設計したシステムを実機に実装し，ハードウェアリソースの利用率を確認した．

4.1　シミュレーションとシミュレーション結果

状態ベクトルを入力し，GraphSLAM によって状態ベクトルを修正する．入力の状態ベクトルは，インテル研究所の SLAM データセットの最初の 336 ポーズで構成されており，建物の廊下を通る最初のループを含む．図 1 の左に補正されていない状態ベクトル（経路）を，右にループを閉じた後に GraphSLAM によって修正された状態ベクトルを示す．

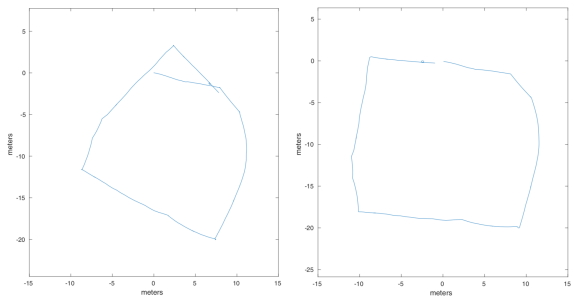

Fig.1: Example input (left) and output state vector (right) of the GraphSLAM algorithm

この画像には地図が描かれていないが，ロボットが移動した軌跡を示す．右側は正方形の図形を示しており，実際の建物の構造に一致している．データセットのオドメトリデータは ICP アルゴリズムで修正されており，GraphSLAM でループを閉じた結果，オドメトリのみでの場合と比較して状態推定が良好になった．一方，修正前のオドメトリは，ループを閉じた後も状態推定において誤差が収束することはなかった．ICP アルゴリズムで修正したオドメトリの状態ベクトルのループを閉じることは，修正されていない状態ベクトルよりも容易である．

4.2　実機への実装

3 章にて示した手法を用いて実際に Altera 社の FPGA である Cyclone V 上に CλaSH を用いて GraphSLAM を決定論的に構築した．このアーキテクチャに対応する合成結果を表 1 に示す．

Table 1: FPGA SYNTHESIS RESULTS

	Used	Available	Utilization
ALMs	2611	41,910	6%
BlockRAM(bits)	412,784	5,662,720	7%
DSPs	8	112	7%

システムは 15bit の整数ビットと 12bit の小数ビットで構成される 27bit 符号付き固定小数点データ型を使用してシミュレートされ，合成された．ベクトル ALU は，8 つの計算を並列に実行する算術演算ユニットとしてシミュレートされ，合成された．表 1 の合成結果が示す 27bitDSP の数に等しい．

5　結言

FPGA 上に GraphSLAM アルゴリズムを実装するために使用する方法論を示した．CλaSH を用いて数学的記述を低級なハードウェア記述に変換し，FPGA 上に GraphSLAM を実装した．設計時間を定量化することは難しい問題であるが，従来の方法ではハードウェアアーキテクチャの開発に膨大な時間がかかる SLAM のような大規模なアプリケーションに対して CλaSH による抽象化で開発を大幅に簡略化することができた．

5.1　今後の展望

面積-時間トレードオフを手作業で行ったが，これは時間のかかるプロセスである．リソースと並列化性能の観点からハードウェアの性能限界とアルゴリズムをパラメータ化することにより，設計がパラメータ化可能であれば，面積-時間トレードオフを自動的に行うことができ，さらなる設計時間の短縮が可能である．

参考文献

[1] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard: “A Tutorial on Graph-Based SLAM,” IEEE Intelligent Transportation Systems Magazine, Vol. 2, No. 4, pp. 31–43, 2010.

[2] C. Baaij, M. Kooijman, J. Kuper, A. Boeijink, and M. Gerards: “CλaSH: Structural Descriptions of Synchronous Hardware Using Haskell,” 2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools, pp. 714–721, 2010.

[3] Grigorios Mingas, Emmanouil Tsardoulias, and Loukas Petrou: “An FPGA implementation of the SMG-SLAM algorithm,” Microprocessors and Microsystems, Vol. 36, No. 3, pp. 190 – 204, 2012.

[4] A. Ratter, C. Sammut, and M. McGill: “GPU accelerated graph SLAM and occupancy voxel based ICP for encoder-free mobile robots,” 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 540–547, 2013.