An FPGA-based Real-time Simultaneous Localization and Mapping System *

Mengyuan Gu [1], Kaiyuan Guo [1], Wenqiang Wang [2], Yu Wang [1], Huazhong Yang [1]
[1] Electronic Engineering Department, TNLIST, Tsinghua University, Beijing, China
[2] Microsoft Research Asia, Beijing, China

Abstract—**Simultaneous localization and mapping (SLAM) is a key algorithm in localization tasks. Considering the limited payload and power on mobile robots, FPGA-based SLAM is a promising onboard solution. This paper presents an FPGA-based SLAM system, which can recover the indoor moving trajectory of the stereo cameras in real-time. We propose a low computational complexity VO-SLAM (Visual Odometry based SLAM) algorithm, and implement the algorithm on a matrix processor based on DE3 develop board. Dedicated matrix accelerators are designed to support application requirements, and a hierarchical matrix computing mechanism is proposed. The algorithm accuracy in the real scenario test is comparable to more complex EKF-SLAM algorithm. Onboard experiments demonstrate the system achieves a processing speed of 31 fps with 30000 features in the global map, which outperforms designs in other publications. We compare the onboard implementation with Intel i7 and achieve 10x energy saving for each frame.**

I. Introduction

In recent years, simultaneous localization and mapping (SLAM) has been an active research field. It is a process of constructing a representation about the environment while keeping track of the robot's location at the same time. Traditional SLAM algorithm based on Extended Kalman Filter (EKF-SLAM) suffers from a quadratic computational complexity with the number of feature points in the global map [1], thus the size of the map is constrained, and real-time processing becomes a challenge.

Another challenge lies in the choice of computation platform. Indoor mobile robots cannot accommodate powerful battery considering payload limitations. Therefore, traditional CPUs and GPUs are not suitable for onboard computation due to their high power consumption. FPGA has advantages of parallel computing, energy-efficiency, and flexibility with reconfigurable logic, thus becomes a promising platform for onboard localization problem. However, mapping SLAM algorithm onto FPGA also brings some difficulties. First, the computation of traditional EKF-SLAM is complex and needs to be simplified for onboard implementation. Another difficulty is that the complex work flow in SLAM algorithm is difficult to be implemented on FPGA through a finite state machine (FSM). In this work, a Visual Odometry based SLAM algorithm (VO-SLAM) is proposed and implemented on FPGA. The main contributions are as follows:

1) We propose a low computational complexity VO-SLAM algorithm suitable for FPGA implementation, which performs drift-free pose estimation, and gives localization results accurate to 1$\sim$2 cm in the real scenario test.

2) We implement the VO-SLAM algorithm on a DE3 develop board. We adopt an embedded system design consisting of a master processor and multiple matrix accelerators. Dedicated matrix accelerators and hardware-software interface are designed to support our application. Onboard test suggests that the system can process a global map of 30000 feature points at 31 frames per second, and achieves 10x energy saving in processing each frame compared with Intel i7.

II. Related Work

SLAM is a widely researched technology for localization task. However, maintaining the covariance matrix in EKF-SLAM is computational expensive. In [2], the processing speed of the SLAM system is limited to 3$\sim$4 fps, thus it cannot be used for real-time control. In [3], the number of features in the global map is bounded to 100, which is insufficient in large indoor scenarios.

FPGA is suitable for onboard localization tasks due to the high power efficiency. In [1], an FPGA implementation of the update step in EKF-SLAM is presented, but it cannot perform a complete localization work flow onboard. In [4], an embedded stereo vision module is proposed with only the feature detection implemented on FPGA. In this work, different from accelerating certain steps in EKF-SLAM with FPGA, we propose a low computational complexity VO-SLAM algorithm, and map it onto FPGA.

III. Proposed VO-SLAM ALgorithm

Traditional EKF-SLAM is computational intensive, thus it is difficult to be implemented on a resource-limited FPGA board with real-time processing speed. We put forward a simplified VO-SLAM algorithm as shown in Fig. 1. The work flow is introduced in this section.

Image Capture: The stereo image capture system is composed of two cameras aligned in parallel in fixed relative position. They are mounted on the mobile robot and record a sequence of images when the robot is moving.

Image Rectification: The stereo cameras are calibrated with the Stereo Camera Calibrator provided by MATLAB Toolbox. The acquired stereo image pairs are rectified with the camera parameters to get images without optical distortion and only have a disparity in the horizontal direction.

* This work was supported by 973 project 2013CB329000, National Natural Science Foundation of China (No. 61373026), the Importation and Development of High-Caliber Talents Project of Beijing Municipal Institutions, and Tsinghua University Initiative Scientific Research Program.

Feature Extraction: SIFT feature detection method is applied to stereo images to extract the 2D pixel positions and descriptors of feature points. SIFT method is invariant to translation, rotation, thus ensures the same feature points can be extracted from different shooting angles.

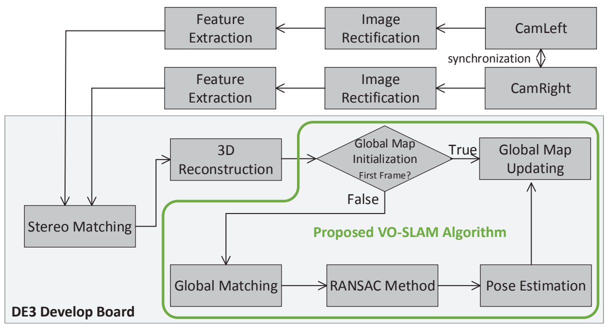


Fig. 1. Work flow of the VO-SLAM based localization system.

Stereo Matching: The acquired feature descriptors are used to build matching relationships between the feature points extracted from stereo image pairs. This procedure is similar to global matching, which is further discussed.

3D Reconstruction: The disparity value (i.e., the horizontal displacement between matched feature points in stereo images) can be used to compute the depth of feature points, then the 3D coordinates of feature points in the camera coordinate system are computed with the pinhole camera model [3].

Global Matching: Assuming the feature descriptors of feature points in the current frame are saved in matrix A ($[DscpC_1, \ldots, DscpC_m]^{\mathrm{T}}$), and those in the global map are saved in matrix B ($[DscpG_1, \ldots, DscpG_n]^{\mathrm{T}}$).

1) Multiply B by $DscpC_i$. The result is a $n * 1$ column vector, with each element representing the vector angle cosine between $DscpC_i$ and $DscpG_j$ ($j = 1 \ldots n$). Then the vector angles are obtained through arc-cosine function and saved in a $n * 1$ column vector Res_i. The global matching step is accelerated by matrix operations in this work. Multiplying B and A^{T} ($B * A^{\mathrm{T}}$), the vector angle cosines between every current feature descriptor and every global feature descriptor are obtained by one single instruction.

2) Find the smallest and second-smallest element in Res_i, denoted by $Res_i(l_1)$ and $Res_i(l_2)$. A match between the ith feature point in the current frame and the l_1th feature point in the global map is considered valid if $Res_i(l_2)$ is 0.8 times larger than $Res_i(l_1)$ [5].

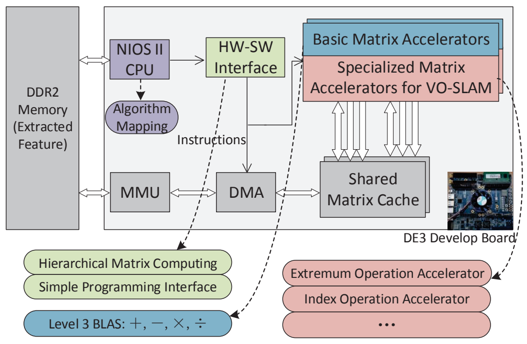
RANSAC Method: RANdom Sample Consensus method is adopted to eliminate the mismatched feature points. Multi-sampling technique in RANSAC finds the pose estimation model with the largest number of inliers (correct matching features), and outliers with wrong information are excluded.

Pose Estimation: Assuming the global coordinates of matched feature points are saved in $CoorGlobal$ and current coordinates are saved in $CoorCamera$, the coordinate transformation is represented by a $3 * 3$ rotation matrix R and a $3 * 1$ translation vector T. The pose estimation is solved by estimating the rotation and translation matrices that minimize the reprojection error shown in Eq. (1) [4].

$$\min_{R,T} \sum_{inliers} \|CoorCamera - T - R * CoorGlobal\|^2 \qquad (1)$$

Global Map Initialization and Updating: Coordinates and descriptors of feature points in the first frame are added to the global map as an initialization step. Then for successfully matched feature points, their current coordinates in the camera coordinate system are back projected to the world coordinate system with transformation matrices R and T, and are denoted by new observations. The coordinates of feature points in the global map are updated with the mean value of every observation, thus randomly distributed measurement errors are decreased by the error compensation method. Unmatched feature points are back projected to the world coordinate system and are directly added to the global map.

Computation Complexity Analysis: Assuming the global map is composed of n feature points, each feature is represented by a p-dimensional descriptor, and m_c feature points in the current frame are matched with the global map. The global matching step suffers from a computational complexity of $O(m_c * n * p)$. In the pose estimation step, the computational complexity is $O(3 * m_c * k)$, where k represents the number of iterations in RANSAC. Considering the RANSAC iterations k and the number of matched feature points m_c are much smaller than the size of the global map n, the computational complexity has been greatly decreased in VO-SLAM compared with EKF-SLAM ($O(n^2)$) [1].



DDR2 Memory (Extracted Feature)
NIOS II CPU
HW-SW Interface
Basic Matrix Accelerators
Specialized Matrix Accelerators for VO-SLAM
Algorithm Mapping
Instructions
MMU
DMA
Shared Matrix Cache
DE3 Develop Board
Hierarchical Matrix Computing
Simple Programming Interface
Level 3 BLAS: +, −, ×, ÷
Extremum Operation Accelerator
Index Operation Accelerator
...

Fig. 2. Overall architecture of the localization system.

IV. Hardware Implementation

The overall architecture of the localization system is shown in Fig. 2. In this work, we focus on implementing an accurate real-time SLAM on FPGA, suitable for mobile systems. To deal with both the complex task scheduling and large amount of matrix operations in the proposed VO-SLAM algorithm, we adopt a universal matrix processor consisting of a soft-core processor and multiple matrix accelerators [6]. The master processor sends instructions to accelerators, and accelerators perform specific matrix operations. We extend the matrix processor in this work. Firstly, we design specialized matrix accelerators to meet specific application requirements. Secondly, the hardware-software interface is redesigned to support matrices operation stored in the external memory. Meanwhile, we encapsulate the low-level schedule programs and provide easy-to-use programming interface. Finally, we map the VO-

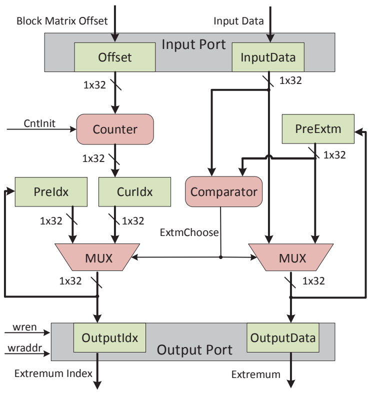


Fig. 3. Extremum operation accelerator.

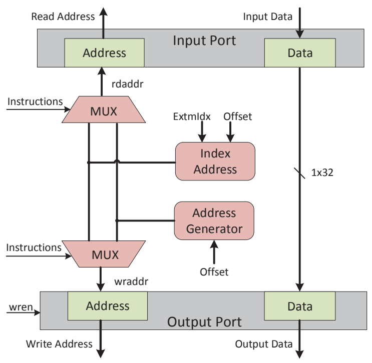


Fig. 4. Index operation accelerator.

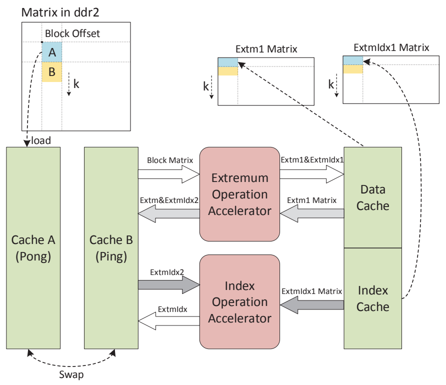


Fig. 5. The hiearchical matrix computing mechanism.

SLAM algorithm onto the matrix processor and achieve a complete localization system on a DE3 develop board.

A. Specialized Accelerators for VO-SLAM Algorithm

The proposed VO-SLAM algorithm is composed of a series of matrix operations. Some basic operations, such as the matrix addition, subtraction, dot multiplication ($.*$), dot division ($./$), and matrix multiplication ($*$) are supported by the universal matrix processor. We further design specialized matrix accelerators to support the proposed VO-SLAM algorithm. In this subsection, we introduce the hardware structure of the designed accelerators.

1) Extremum Operation Accelerator: In the global matching step, we need to get the smallest element in each column of a matrix. We design the extremum operation accelerator shown in Fig. 3. The input matrix in the shared matrix cache can be a block of a large matrix stored in external memory. Thus we introduce the block matrix offset (32-bit integer), representing the position of the input matrix in the large matrix saved in DDR2 memory. The counter is set to the input offset for initialization. As matrix elements are scanned along the column direction, every input data is compared with the previous extremum. When the column scan is over, the extremum is picked out. Currently, we insert 8 extremum operation accelerators in our design, which enables to process 8 columns in parallel.

2) Index Operation Accelerator: In the global matching step, we need to compare the smallest and second-smallest element in each column. The second-smallest element is obtained by substituting the column's smallest element with infinity, and sending that column into the extremum operation accelerator once again. Thus, we design an accelerator to write data (e.g. infinity) to a specified address provided by the index; meanwhile, reading out data from a specified address is achieved in the same module. The structure of the index operation accelerator is shown in Fig. 4.

B. Hierarchical Matrix Computing and Algorithm Mapping

Partitioning large matrices saved in the external memory into small blocks enables matrix accelerators to process matrices larger than the cache capacity. However, finding the extremum in large matrices is not an independent process between different blocks. We propose a hierarchical matrix computing mechanism for extremum operation, which consists of two stages: finding column extremum in each block and merging results in different blocks, as shown in Fig. 5.

1) Load a block from the external memory into *pong* buffer. Find column extremum in the block saved in *ping* buffer. Since we insert 8 extremum operation accelerators, 8 columns can be processed in parallel. The block extremum (Extm1) and the index (ExtmIdx1) are written to the data cache (Extm1 Matrix) and index cache (ExtmIdx1 Matrix) in one row. Swap *ping* and *pong* buffer.

2) Merge the column extrema saved in data cache. The column extrema of different blocks are saved in the same column in the data cache. Send each column in Extm1 Matrix into the extremum operation accelerator to pick out the final extremum (Extm) and its position (ExtmIdx2) in Extm1 Matrix. Then we read out the position of the final extremum (ExtmIdx) from ExtmIdx1 Matrix according to the index ExtmIdx2.

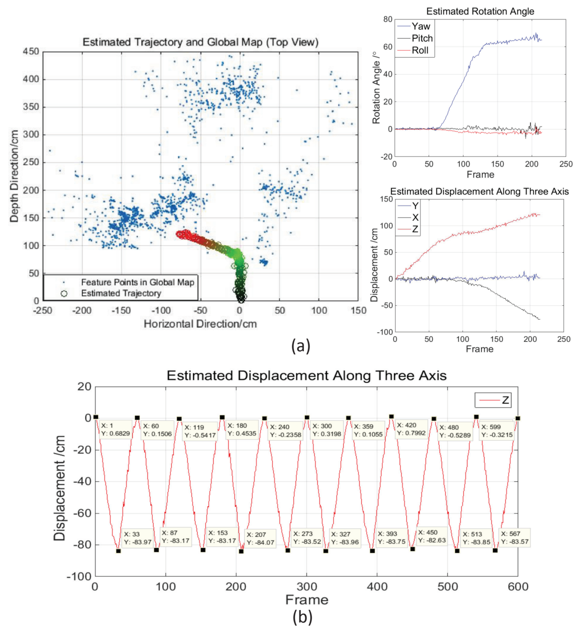


Fig. 6. (a): Estimated trajectory and global map in real scenario test. (b): The displacement along depth direction in the drift-free test.

We further encapsulate the low-level schedule programs and provide easy-to-use C++ classes and class functions to

TABLE I
MOTION ESTIMATION ACCURACY TEST.

Ground Truth						Estimated					
X (cm)	Y (cm)	Z (cm)	Pitch (°)	Yaw (°)	Roll (°)	X (cm)	Y (cm)	Z (cm)	Pitch (°)	Yaw (°)	Roll (°)
0	0	0	0	0	0	0±0.42	0±0.46	0±0.48	0±0.07	0±0.08	0±0.04
0	0	-80.00	0	0	0	1.05	0.23	-78.29	0.16	0.13	0.14
50.00	0	0	0	0	0	49.32	0.43	0.39	0.31	-0.11	0.12
0	0	0	0	90	0	2.4	-0.8	3.3	0.46	91.3	-0.17

manipulate matrix stored in the shared matrix cache or DDR2 memory. The SLAM algorithm is mapped to a DE3 develop board by programming with the C++ interface on the NIOS II processor. The system is flexible and scalable since it can be modified through software programming.

V. EXPERIMENTAL RESULTS

We implement the proposed VO-SLAM algorithm on a DE3 develop board, which has an Altera Stratix III EP3SL340 FPGA, and we take a NIOS II/f soft-core processor as the master processor. In this section, the localization accuracy of the VO-SLAM algorithm and the performance of the onboard system will be shown.

The image capture system moves along a level track to different distances and rotates certain degrees in the experiment. Table. I shows the ground truth and averaged estimated positions from the VO-SLAM algorithm. When the system is locally static, the ± variation values indicate the standard deviations of the estimated positions; while in other cases, the estimated positions are averages of ten measurements. The proposed VO-SLAM typically gives localization results accurate to 1.23 cm and 0.17° in the experiment.

In Fig. 6(a), the platform moves forward for about 80 cm at a speed of 10 cm/s, then rotates anticlockwise for about 65° and moves along that direction. The moving trajectory is recovered with a global map, representing the positions and spatial relations of objects in the environment. In Fig. 6(b), the platform moves forward and backward along the same path for 20 times, the estimated displacement along depth direction is visualized. No apparent drift occurs in the experiment, which demonstrates the VO-SLAM algorithm can deal with loop closure trajectory with bounded accumulative error.

When parameters are set to typical values ($r = 512, m_c = 70$), the onboard system reaches a frame rate at 31 fps with a global map of over 30000 feature points. In real scenario test, The global map has about 2000 feature points in a 20 m^2 space, thus we believe a map of 30000 feature points is enough for SLAM task in large indoor space.

We compare our FPGA implementation with a software version on an Intel i7 3770K CPU running at 3.4GHz. The VO-SLAM algorithm running on the CPU shows better performance (41 fps vs 31 fps) due to the fully optimized matrix operations in Matlab. However, the power consumption of Intel i7 is also huge, and we achieve 10x energy saving in processing each frame (1.88 J/Frame vs 0.19 J/Frame). We further compare our work with other publications, as shown in Table. II. The proposed VO-SLAM system achieves the highest frame rate and biggest global map size, with comparable localization accuracy with traditional EKF-SLAM algorithm. Considering the low power feature of our computing system, we believe our work achieves the best energy efficiency.

TABLE II
COMPARISON WITH OTHER PUBLICATIONS.

Work	Method	Platform	Accuracy	Framerate Map Size
Schmid et al. [7]	Key-frame odometry Inertial mesurements	CPU 1.8GHz	38 cm Outdoor	14.6 fps 300
Achtelik et al. [8]	Visual SLAM Inertial mesurements	CPU 1.6GHz	6.9 cm Indoor	10 fps 300
Spamp et al. [4]	EKF-SLAM	CPU FPGA	1 cm Indoor	14 fps <40
Stasse et al. [3]	EKF-SLAM	CPU 1.6GHz	1∼2 cm Indoor	30 fps 100
Heng et al. [9]	Key-frame odometry	CPU 3.3GHz	4.57 cm Indoor	12 fps –
Proposed	**VO-SLAM**	**Softcore FPGA**	**1∼2 cm Indoor**	**31 fps 30000**

VI. CONCLUSION

In this paper, we implement a simultaneous localization and mapping system (SLAM) system on FPGA. A novel localization algorithm named VO-SLAM is proposed to decrease the computational complexity while ensure the localization accuracy. The algorithm accuracy is comparable to complex EKF-SLAM algorithm. Furthermore, we implement the VO-SLAM on a DE3 develop board. Dedicated matrix accelerators and sophisticated hardware-software interface are designed to support our application. The proposed system achieves a processing speed of 31 fps for a global map of over 30000 feature points, which outperforms designs in other publications.

REFERENCES

[1] S. Cruz *et al*, "Fpga implementation of a sequential extended kalman filter algorithm applied to mobile robotics localization problem," in *LASCAS, 2013*, pp. 1–4.

[2] C. L. Wang *et al*, "Bearing-only visual slam for small unmanned aerial vehicles in gps-denied environments," *International Journal of Automation and Computing, 2013*, vol. 10, no. 5, pp. 387–396.

[3] A. J. Davison *et al*, "Monoslam: Real-time single camera slam," *TPAMI*, vol. 29, no. 6, pp. 1052–1067, 2007.

[4] G. Spampinato *et al*, "An embedded stereo vision module for 6d pose estimation and mapping," in *IROS, 2011*, pp. 1626–1631.

[5] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[6] W. Wang *et al*, "A universal fpga-based floating-point matrix processor for mobile systems," in *FPT, 2014*, pp. 139–146.

[7] K. Schmid *et al*, "Autonomous vision-based micro air vehicle for indoor and outdoor navigation," *Journal of Field Robotics*, vol. 31, no. 4, pp. 537–570, 2014.

[8] M. Achtelik *et al*, "Onboard imu and monocular vision based control for mavs in unknown in-and outdoor environments," in *ICRA, 2011*, pp. 3056–3063.

[9] L. Heng *et al*, "Self-calibration and visual slam with a multi-camera system on a micro aerial vehicle," in *RSS*, 2014.