sensors

MDPI

Article

A Reconfigurable Visual–Inertial Odometry Accelerated Core with High Area and Energy Efficiency for Autonomous Mobile Robots

Yonghao Tan [1,†], Mengying Sun [1,†], Huanshihong Deng [1], Haihan Wu [1], Minghao Zhou [1], Yifei Chen [1], Zhuo Yu [1], Qinghan Zeng [2,*], Ping Li [3], Lei Chen [1] and Fengwei An [1,*]

1 School of Microelectronics, Southern University of Science and Technology, Shenzhen 518055, China
2 Scientific and Technical Center for Innovation, Beijing 100080, China
3 Department of Computing and School of Design, The Hong Kong Polytechnic University, Hong Kong, China
* Correspondence: cqh_zgy@163.com (Q.Z.); anfw@sustech.edu.cn (F.A.)
† These authors contributed equally to this work.

Citation: Tan, Y.; Sun, M.; Deng, H.; Wu, H.; Zhou, M.; Chen, Y.; Yu, Z.; Zeng, Q.; Li, P.; Chen, L.; et al. A Reconfigurable Visual–Inertial Odometry Accelerated Core with High Area and Energy Efficiency for Autonomous Mobile Robots. *Sensors* **2022**, *22*, 7669. https://doi.org/10.3390/s22197669

Academic Editor: Gianni D'Angelo

Received: 2 September 2022
Accepted: 6 October 2022
Published: 9 October 2022

Abstract: With the wide application of autonomous mobile robots (AMRs), the visual inertial odometer (VIO) system that realizes the positioning function through the integration of a camera and inertial measurement unit (IMU) has developed rapidly, but it is still limited by the high complexity of the algorithm, the long development cycle of the dedicated accelerator, and the low power supply capacity of AMRs. This work designs a reconfigurable accelerated core that supports different VIO algorithms and has high area and energy efficiency, precision, and speed processing characteristics. Experimental results show that the loss of accuracy of the proposed accelerator is negligible on the most authoritative dataset. The on-chip memory usage of 70 KB is at least 10× smaller than the state-of-the-art works. Thus, the FPGA implementation's hardware-resource consumption, power dissipation, and synthesis in the 28 nm CMOS outperform the previous works with the same platform.

Keywords: SLAM; VIO; accelerator; reconfigurable; AMRs

1. Introduction

In recent years, AMRs have achieved rapid development driven by practical application demands. As the core technology of AMRs, simultaneous localization and mapping (SLAM) include five parts as Figure 1 shows: sensor data reading, front-end visual odometry, backend optimization, loop closure, and mapping. As a single sensor cannot cope with all situations, the most effective approach is to fuse data from both the camera and IMU, an algorithm called VIO. Aside from the computational complexity of loop closure and global optimization, the VIO system can estimate the position and trajectory of a moving object in real time with lower power consumption, which is very important for lightweight AMRs with high processing speed requirements.

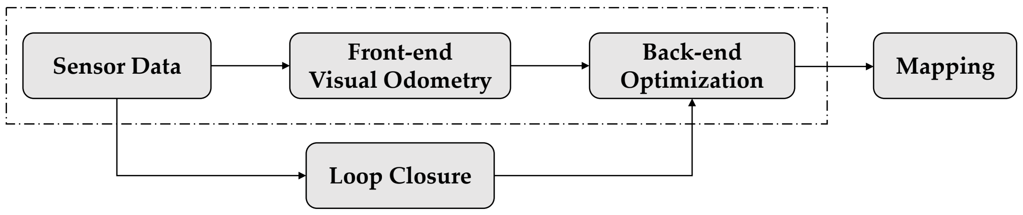

Figure 1. Process Diagram of SLAM.

The visual front-end of VIO mainly estimates the robot's motion through the direct method or feature point extraction. In 2008, the direct method proposed by Silveira G. et al. [1] used the difference in light intensity of each pixel in adjacent frame images to estimate the camera's motion. At present, representative open-source projects are DTAM [2], LSD-SLAM [3], and DSO [4], etc. Compared with feature point extraction, the direct method saves time for calculating features and can maintain certain robustness when the texture is scarce, but it is difficult to work in scenes with drastic changes in light. Feature point extraction uses image features instead of image intensity, which overcomes the shortcomings of the direct method but relies on the feature extraction results. In recent years, classic features have included SIFT [5], SURF [6], FAST [7], and so on.

Backend optimization is to optimize the motion pose estimated by the front end to minimize accumulated errors. The main backend algorithms are divided into filtering methods using extended Kalman filter (EKF) or other filters and optimization methods using bundle adjustment (BA) or graph optimization methods. In the case of limited computing resources and relatively simple quantities to be estimated, the filtering method represented by EKF is very effective; however, because the storage capacity and state quantity are in a quadratic growth relationship, there are many feature data, and the filtering method is less efficient. In addition to KF [8] and EKF [9], there are also information filters [10,11] and particle filters [12–14]. Contrary to the filtering method, the optimization method no longer relies on the information at a specific moment but obtains the optimal global estimation of all landmarks by optimizing the joint error function of all poses and landmarks. Existing optimization methods include that by Di, K. et al. [15], who studied an extended BA algorithm that can utilize both 2D and 3D information and is suitable for RGB-D cameras. Alismail, H. et al. [16] abandoned the minimization of reprojection error and achieved a significant improvement in accuracy with a photometric BA algorithm based on maximizing photometric continuity. The incremental, consistent, and efficient BA proposed by Liu, H. et al. [17] adopts incremental technology, which consumes only about 1/10 of the computing resources of traditional BA under the premise of ensuring accuracy.

In addition to improving algorithms, using SLAM systems also relies on hardware acceleration. Compared with CPU, GPU has powerful floating-point computing capability, so GPU is generally used in the early stage of hardware acceleration design research, such as feature extraction accelerator based on SIFT [18] and SURF [19] features. However, the GPU system consumes a lot of power. With the development of field programmable gate arrays (FPGAs), hardware acceleration relies more on FPGA. Yum, J. et al. [20] designed a complete SIFT hardware accelerator, Wilson, C. et al. [21] proposed a complete FPGA implementation of the SURF accelerator, and Ulusel, O. [22] showed that feature extraction on the FPGA platform has great advantages over CPU and GPU.

Moreover, accelerators suitable for backend optimization have also been proposed recently. For example, Tertei, D. T. R. et al. [23] proposed an efficient FPGA SoC hardware structure using systolic array matrix multiplication to accelerate the EKF-SLAM algorithm. Wang, J. et al. [24] designed a reconfigurable matrix multiplication coprocessor for accelerating matrix multiplication in visual navigation algorithms. However, the hardware accelerators can only accelerate specific algorithms, or they are incomplete, only accelerating the feature extraction or matrix multiplication part and lacking pose estimation and trajectory output.

Currently, the relatively complete acceleration design includes Liu, R. et al. [25] implementing a complete ORB-SLAM system on FPGA. The feature extraction and feature matching parts are accelerated by FPGA, but the trajectory estimation and pose optimization are completed by ARM. Li, Z. et al. [26] presented an accurate, low-power, real-time CNN-SLAM processor that implements full-visual SLAM on a single chip. Suleiman, A. et al. designed a chip that can perform VIO [27], reducing on-chip storage to 1/4 the size through image compression techniques. Zhang, Z. et al. designed a VIO system that uses Kintex-7 XC7K355T (an FPGA) [28], and Wang, C. et al. designed a visual SLAM system that uses UltraScale+ XCZU7EV (an FPGA) [29].

This paper proposes a reconfigurable, real-time, low-area, and energy-efficient VIO accelerator implemented on an FPGA with the following major design features: (i) a reconfigurable accelerator architecture that adapts to different Kalman-filter-based VIO algorithms; (ii) an optimized instruction-based structure supporting the simultaneous workflow of fixed-point and floating-point units to accelerate VIO algorithms for real-time usage. This design can support the post estimation and trajectory output of real-time > 60 Hz frame input and > 200 Hz of IMU input; and (iii) a computing core with shared memory and the memory reuse strategy. This works only consumes 70 KB of on-chip memory, which is at least 10× lower than the previous works. To our knowledge, this is the first integrated reconfigurable architecture that supports multiple VIO algorithms implemented in FPGA.

2. Algorithms

2.1. Overall Procedure of the Visual–Inertial Odometry

Figure 2 shows the overall procedure of the visual–inertial odometry, consisting of IMU pre-integration, feature coordinates optimization, and extended Kalman filter (EKF). IMU pre-integration computes the prior estimate of pose and the position and other state variables, such as velocity and feature location of the AMR concerning the IMU. Feature coordinates optimization optimizes the pre-estimated pixel location of the stored features by minimizing the 2D reprojection error. Then, measured state variables are estimated through the location difference between the optimized results and the prior estimated ones. At last, EKF combines the prior estimation of state variables and the measured state variables according to the covariance matrix and outputs the post estimation of the state variables, where pose and position represent the trajectory. Each floating-point function circuit is assigned a priority level to ensure that each floating-point computation circuit can be accessed by only one floating-point function circuit at the same clock.

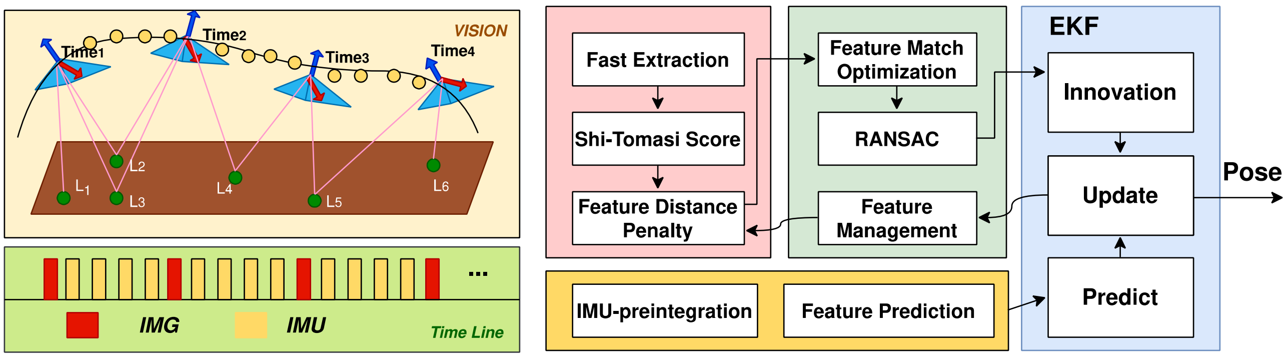

Figure 2. The overall procedure of the visual–inertial odometry.

2.2. IMU Pre-Integration

IMU can measure acceleration and angular velocity through the accelerometer and gyroscope. Through integration, the rotation and displacement between two frames of images can be obtained. That is to say, if the position, velocity, and rotation at time k are known, these values at time $k + 1$ can be obtained. However, in the optimization algorithm, these values at each moment are estimated. When optimizing them, the data between two moments must be re-integrated, which makes the calculation requirements large. Lupton T. et al. [30] proposed IMU pre-integration to avoid repeated integration, and Forster, C. [31] made it better.

A schematic diagram of the IMU model is shown in Figure 3, and its formula is shown in (1). The measured values of linear acceleration and angular velocity are represented by $\hat{(\cdot)}$. Linear acceleration is the resultant vector of gravitational acceleration and object acceleration, b_{a_t} and b_{ω_t} are offsets, n_a and n_ω are Gaussian noise.

$$\begin{gathered}\hat{a}_t = a_t + b_{a_t} + R_w^t g^w + n_a, \\ \hat{\omega}_t = \omega_t + b_{\omega_t} + n_\omega\end{gathered} \tag{1}$$

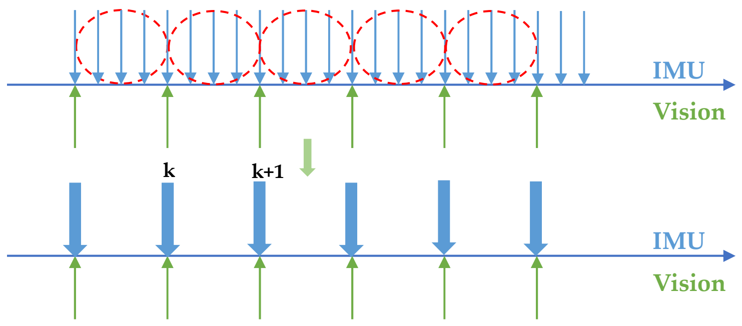

Figure 3. Diagram of IMU model.

The representation of the position, velocity, and rotation (quaternion form) of $k+1$ frame can be obtained from k frame, as shown in (2). Here, b represents the body coordinate system, w represents the world coordinate system, R_t^w represents the rotation matrix from the body coordinate system to the world coordinate system at time t, $q_t^{b_k}$ represents the rotation of the body coordinate system at time t relative to the body coordinate system at the time b_k, and $q_{b_k}^w$ represents the rotation of the body coordinate system relative to the world coordinate system at the time b_k.

$$\begin{aligned} p_{b_{k+1}}^w &= p_{b_k}^w + v_{b_k}^w \Delta t_k + \iint_{t\in[k,k+1]} [R_t^w(\hat{a}_t - b_{\alpha_t}) - g^w] dt^2, \\ v_{b_{k+1}}^w &= v_{b_k}^w + \int_{t\in[k,k+1]} [R_t^w(\hat{a}_t - b_{\alpha_t}) - g^w] dt, \\ q_{b_{k+1}}^w &= q_{b_k}^w \otimes \int_{t\in[k,k+1]} \tfrac{1}{2}\Omega(\hat{\omega}_t - b_{\omega_t}) q_t^{b_k} dt \end{aligned} \tag{2}$$

where

$$\Omega(\omega) = \begin{bmatrix} -\lfloor w \rfloor_\times & \omega \\ -\omega^T & 0 \end{bmatrix}, \ \lfloor w \rfloor_\times = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}. \tag{3}$$

In (2), PVQ of the body coordinate system at time $k+1$ depends on b_k. If the PVQ of the body coordinate system is directly used as a variable to optimize and iteratively update, it will lead to a large amount of calculation. The idea of IMU pre-integration is to adjust the reference coordinate system from the world coordinate system w to the body coordinate system b_k of k frame so that the integration result becomes the relative change of b_{k+1} to b_k. It is realized by multiplying $R_w^{b_k}$, as shown in (4). The detailed derivation process is omitted [32].

$$\begin{aligned} R_w^{b_k} p_{b_{k+1}}^w &= R_w^{b_k}(p_{b_k}^w + v_{b_k}^w \Delta t_k - \tfrac{1}{2} g^w \Delta t_k^2) + \alpha_{b_{k+1}}^{b_k}, \\ R_w^{b_k} v_{b_{k+1}}^w &= R_w^{b_k}(v_{b_k}^w - g^w \Delta t_k) + \beta_{b_{k+1}}^{b_k}, \\ q_w^{b_k} \otimes q_{b_{k+1}}^w &= \gamma_{b_{k+1}}^{b_k}. \\ \alpha_{b_{k+1}}^{b_k} &= \iint_{t\in[k,k+1]} [R_t^{b_k}(\hat{a}_t - b_{\alpha_t})] dt^2, \\ \beta_{b_{k+1}}^{b_k} &= \int_{t\in[k,k+1]} [R_t^{b_k}(\hat{a}_t - b_{\alpha_t})] dt, \\ \gamma_{b_{k+1}}^{b_k} &= \int_{t\in[k,k+1]} \tfrac{1}{2}\Omega(\hat{\omega}_t - b_{\omega_t}) \gamma_t^{b_k} dt \end{aligned} \tag{4}$$

2.3. Algorithm of Feature Extraction

2.3.1. FAST-9 Feature Detection

In the feature extraction technology, in addition to FAST [7], there are SIFT [5], SURF [6], and other algorithms. The features they extracted have strong invariance, but the time consumption is relatively large. In a system, feature extraction is only a part, and subsequent algorithms such as registration, purification, and fusion are also performed. This makes the real-time performance not suitable and reduces the system performance. The Fast detector is very fast because it does not involve complex operations such as scale and gradient. It uses the gray value of the pixel in a specific neighborhood to compare the size with the center point to determine whether it is a corner point.

The proposed accelerated core adapts the FAST-9 feature extraction [7] in the vision pipeline, a trade-off between accuracy and efficiency since the proposed accelerated core targets high-performance and low-power design.

$$S_{c\to p}\begin{cases} darker, P < C - T \\ similar, C - T \leq P \leq C + T \\ brighter, C + T < P \end{cases} \tag{5}$$

Here, S stands for the intensity relationship of the pixels P with the center C, and T is the user-defined threshold. If there are consecutive nine pixels darker or brighter, the center pixel is a FAST feature in the accelerated core.

This paper uses pipeline stages for hardware implementation since the FAST algorithm requires counters, which should be accumulated by clock cycles. However, it is not enough to implement it in 16 clock cycles, as shown in Figure 4, because a true circle path to access all pixels in the FAST algorithm should be half a circle. For example, if pixels P_1, P_2, and P_{10}–P_{16} are brighter while others are not, the center pixel should be a FAST feature. However, the first round only considers seven brighter pixels (P_{10}–P_{16}) which makes an incorrect judgment. The one-and-a-half circles for detecting solves this problem by redundantly calculating eight more pixels.

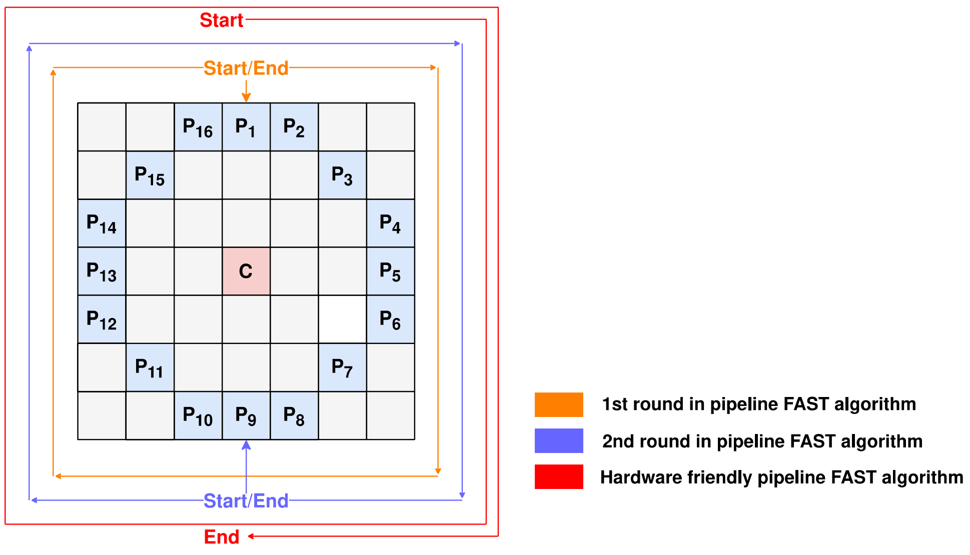
Start
Start/End
P16
P1
P2
P15
P3
P14
P4
P13
C
P5
P12
P6
P11
P7
P10
P9
P8
Start/End
End
1st round in pipeline FAST algorithm
2nd round in pipeline FAST algorithm
Hardware friendly pipeline FAST algorithm

Figure 4. The hardware-friendly pipeline FAST algorithm.

2.3.2. Gradient-Based Score Calculation

This paper adapts a gradient-based score (Grad score in short) to indicate the quality of a feature after FAST feature detection, as shown below:

$$Grad\ score = \sum_{i=1}^{6}\sum_{j=1}^{6}\left(P_{i,j+1} - P_{i,j-1}\right)^2/64 + \left(P_{i+1,j} - P_{i-1,j}\right)^2/64 \tag{6}$$

Here, the input patch is size 8 × 8, and P stands for the pixel values. Therefore, the Grad score is calculated with a 6 × 6 window inside the patch. The score will be used in the rest process in the core.

2.4. Algorithm of Feature Coordinates Alignment

Feature coordinates calculated from IMU pre-integration and feature prediction are corrected by photometric error, which is based on the difference in pixel data between the old feature patches in the last frame and the new feature patches extracted in the current frame. The photometric error is defined by:

$$e_{l,j} = P_l\left(\boldsymbol{p}_j\right) - I_l\left(\boldsymbol{p}s_l + \boldsymbol{W}\boldsymbol{p}_j\right) - m \tag{7}$$

where the scalar factor $s_l = 0.5^l$ stands for the down-sampling number of layers of the image pyramid. $\boldsymbol{W}$ stands for the warping transform matrix in [33], $\boldsymbol{p}_j$ is the patch pixel in the patch P_l, and $\boldsymbol{p}$ is the feature coordinate.

This paper adapts the QR-decomposition method in [33], which stacks all error terms together for given estimated coordinates $\hat{p}$:

$$b(\hat{p}) = \overline{A}(\hat{p})\delta p \tag{8}$$

Here, $\overline{A}(\hat{p})$ is based on the patch intensity gradients along the X and Y axes. The QR-decomposition of $\overline{A}(\hat{p})$ obtains an equivalent reduced linear equation system:

$$b(\hat{p}) = A(\hat{p})\delta p \tag{9}$$

The iteration will be user-defined and ten times in the proposed core. If the photometric error is still higher than a certain threshold, the feature will be marked as bad and abandoned, while others could be used in the EKF update.

2.5. Supported Operations in SLAM Algorithms

Most SLAM algorithms require not only basic operations, such as the addition or multiplication of scalars, but also operations of rotation dynamics, including rotation matrix, quaternion, and lie algebra. The proposed VIO accelerated core provides full functionality to support the abovementioned operations, shown in Table 1.

Table 1. Supported operations by the accelerated core.

Operation	Description	Time Consumption
Scalar add/sub	Scalar addition/subtraction	5 clock cycles
Scalar mul	Scalar multiplication	2 clock cycles
Scalar reci	Scalar reciprocal	10 clock cycles
Scalar sqrt_slow	Scalar square root with high accuracy	27 clock cycles
Scalar sqrt_fast	Scalar square root with low latency	10 clock cycles
Sin_Cos	Sine and cosine function for input in radians	52 clock cycles
M_inv	Matrix inversion	31 clock cycles
Li2R	Transform lie algebra to a rotation matrix	73 clock cycles
Li2Q	Transform lie algebra to quaternion	65 clock cycles
R2Q	Transform rotation matrix to quaternion	53 clock cycles
Q2R	Transform quaternion to a rotation matrix	14 clock cycles
Q_q	Quaternion multiplication	14 clock cycles

3. Hardware Design

This section introduces the overall hardware architecture of the proposed VIO accelerated core and details four important submodules and techniques adapted in the design.

3.1. Overall Hardware Architecture

Figure 5 shows the proposed accelerator's overall architecture constructed by four sub-modules, including a fixed-point vision pipeline, a memory interface, a programmable computation core, and layers for the EKF engine.

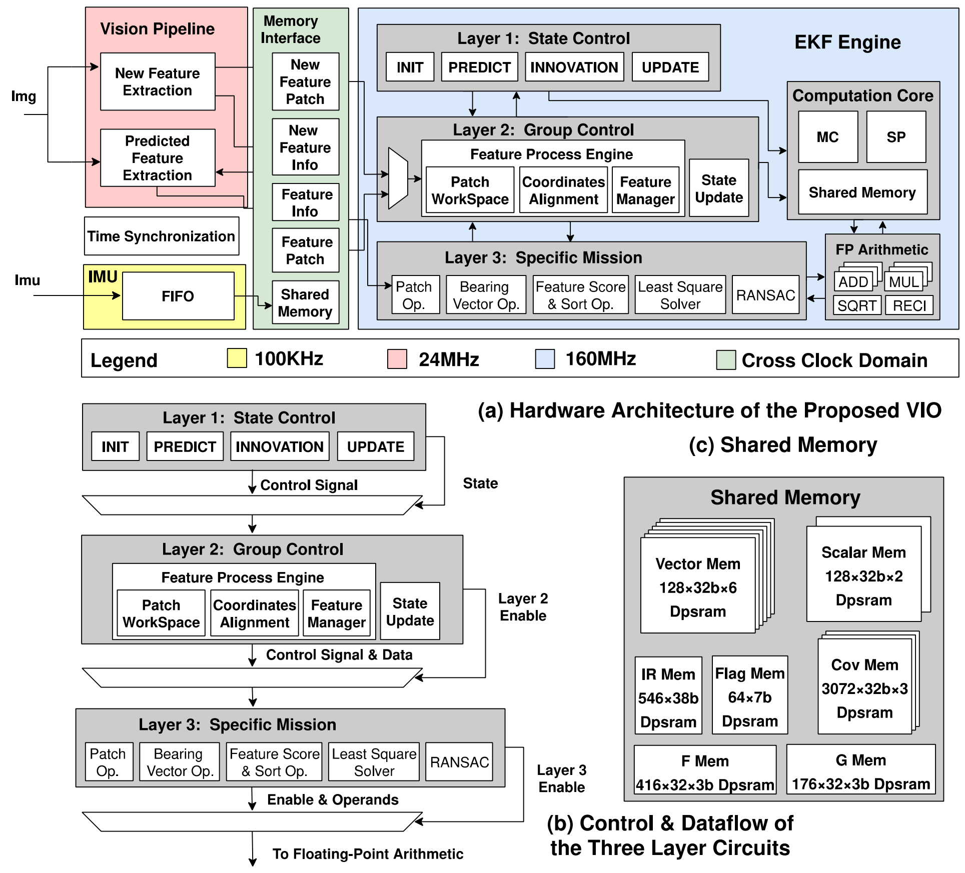

Figure 5. Overall hardware architecture of the proposed accelerated core. (**a**) Hardware architecture of the proposed VIO accelerated core; (**b**) control and dataflow of the three-layer circuits; (**c**) shared memory for the storage of vectors and matrices.

The vision pipeline receives input data from the image sensor to perform new and predicted feature extraction. The memory interface uses a shared memory strategy to store intermediate values and output data. The programmable computation core pre-loads three programs to perform the computation of IMU pre-integration and the computation of the Jacobian matrix. This core can satisfy the pose estimation of a sample class of Kalman-filter-based SLAM algorithms. In the EKF engine, layer 1 contains a finite state machine in charge of the whole process. The layer of the feature processing engine scores and sorts the features received from the feature extraction engine then compares the best 25 of them with the features stored in the feature manager and replaces the old ones with them if the new features are better. The feature processing engine also computes the 2D location difference of the same feature in the last and current frames. Then, the EKF engine completes the update process of the EKF process with the help of a specific mission layer and FP arithmetic. The specific mission layer shares the floating-point arithmetic with the computation core for higher hardware utilization.

3.2. Fixed-Point Vision Pipeline

Figure 6 details the structure of the fixed-point vision pipeline, including a Gaussian pyramid-image half sample engine, a FAST feature identifier, a Grad score-based patch extractor, and a patch controller.

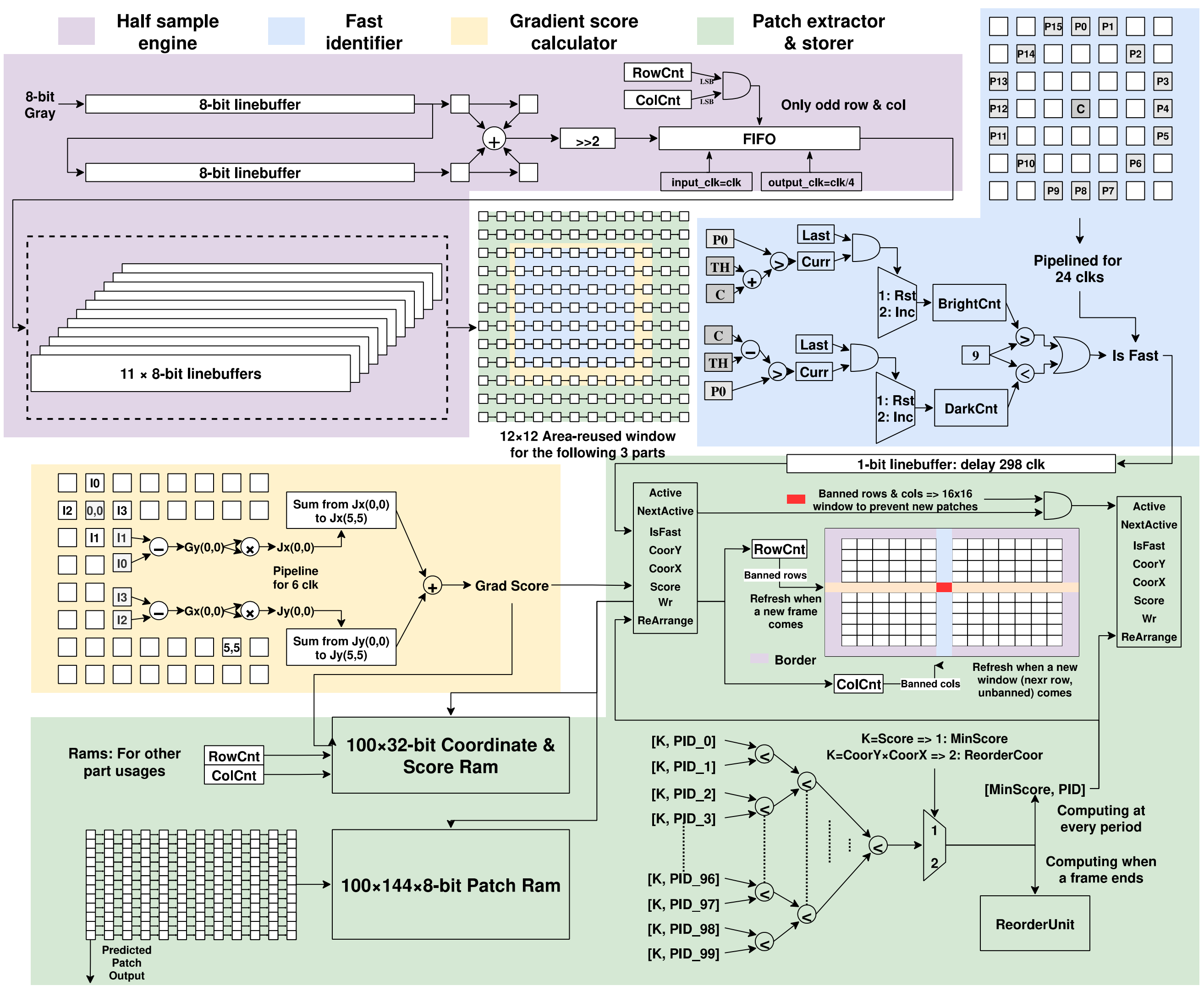
Half sample engine
Fast identifier
Gradient score calculator
Patch extractor & storer
8-bit Gray
8-bit linebuffer
8-bit linebuffer
RowCnt
ColCnt
Only odd row & col
>>2
FIFO
input_clk=clk
output_clk=clk/4
11 × 8-bit linebuffers
12×12 Area-reused window for the following 3 parts
Pipelined for 24 clks
Last
Curr
1: Rst 2: Inc
BrightCnt
DarkCnt
Is Fast
1-bit linebuffer: delay 298 clk
Sum from Jx(0,0) to Jx(5,5)
Sum from Jy(0,0) to Jy(5,5)
Pipeline for 6 clk
Grad Score
Active NextActive IsFast CoorY CoorX Score Wr ReArrange
Banned rows & cols => 16x16 window to prevent new patches
Banned rows
Refresh when a new frame comes
Border
Banned cols
Refresh when a new window (nexr row, unbanned) comes
Rams: For other part usages
100×32-bit Coordinate & Score Ram
100×144×8-bit Patch Ram
Predicted Patch Output
K=Score => 1: MinScore
K=CoorY×CoorX => 2: ReorderCoor
[MinScore, PID]
Computing at every period
Computing when a frame ends
ReorderUnit

Figure 6. Detailed structure of the fixed-point vision pipeline.

The architecture workflow is pipelined with line buffers, thus reducing memory consumption and increasing processing throughput. The operation provides a foundation for the following process by interpreting images at multiple resolutions and different scales with a Gaussian sampling pyramid of the input grayscale image. The half-sampled pixels are transferred through an asynchronous FIFO, followed by a group of 11 line buffers. The rest processes are based on an area-reused 12 $\times$ 12 sliding window where pixel data can be available for three layers.

The FAST identifier is a feature extractor adapting the FAST-16 algorithm where if a pixel differs greatly from enough pixels in its surrounding neighborhood, the pixel may be a corner. Adapting the FAST algorithm for every pixel indicates a continually analyzed 16 pixels, i.e., whether the pixel is brighter/darker than the center pixel as the last pixel does. In the hardware implementation, the algorithm is processed in a pipeline which takes 24 periods. At every stage, the circuit will detect the brightness for both the current and last pixels and increase or reset the counters. In Figure 6, BrightCnt means the number of continuing pixels with a grey value more significant than the grey value of the center plus threshold, while DarkCnt stands for the opposite. If BrightCnt or DarkCnt is larger than 9, the pixel is identified as a FAST feature, and the result will be passed to the patch controller by a line buffer.

The FAST identifier is optional, considering the patch extractor has already been attached with a Grad score comparator which adapts only the front-computing. Input pixel data from the 8 × 8 window generates their gradient values which are used to calculate the Grad score according to (6).

The patch controller temperately stores the information of a maximum of 100 patches in sub-patch controllers, where a patch is defined by the feature pixel and its surrounding 12 × 12 window of pixels. In order to avoid the extracted feature points being too concentrated, this paper proposed a patch extract strategy, which constructs banned rows and banned columns. When a new feature/patch is extracted, its surrounding 16 rows and columns will be banned for the subsequent extraction unless the new feature has a higher Grad score. The 100 sub-patch controllers are awakened one by one to reduce power consumption and send write to enable signal to the patch controller for patch storing, which is implemented by dual-port rams. If the maximum feature limit (100) is reached, a special sub-patch controller will be used to cover the patch with the least Grad score if the new feature has a higher score, ensuring the extracted patches are high quality for the rest process. At the end of a frame, the vision pipeline sends a signal to the top state machine, indicating the available feature patches.

3.3. Programmable Computation Core

Figure 7 details the structure of the programmable computation core realized by instruction control, which is an extension of the previous work [34]. A matrix computation core performs the matrix operation and controls the program workflow, and a special computation core performs quaternion, angle-axis, and scalar operations. The computation core is controlled by a proposed instruction set architecture (ISA). Operation instructions store the offset address of operands, source, and target address, transpose or generate antisymmetric matrix, and the operation types. The PC reads instructions from IRMEM and sends them to the corresponding instruction decoder for operation instructions. Each core has its own 38-bit instruction set and an 8-bit address instruction memory but shares a 4 × 7-bit address, 32-bit per data, memory for operators. Each core has and only has access to its own nine adders, nine multipliers, one floating sqrt, and one reciprocal computation circuit. Compared with the matrix core, the special core has some unique computation circuits, such as cosine and sine. Two instructions for matrix core, bubble and sync, and one for special core, bubble, are used to synchronize the cores and secure the computation order. An interface and matched instructions for 16-bit per address, 38-bit per data memory are kept for the potential memory. The operations required by different Kalman-filter-based algorithms can be programmed into the instruction and operator memory blocks. What is more, Table 2 shows the supported operations by the computation core, including normal scalar operations and special operations required by most SLAM algorithms.

Table 2. Description and time consumption of operations supported by the computation core.

Operation	Description	Time Consumption
Scalar add/sub	Scalar addition/subtraction	5 clock cycles
Scalar mul	Scalar multiplication	2 clock cycles
Scalar reci	Scalar reciprocal	10 clock cycles
Scalar sqrt_slow	Scalar square root with high accuracy	27 clock cycles
Scalar sqrt_fast	Scalar square root with low latency	10 clock cycles
Sin_Cos	Sine and cosine function for input in radians	52 clock cycles
M_inv	Matrix inversion	31 clock cycles
Li2R	Transform lie algebra to a rotation matrix	73 clock cycles
Li2Q	Transform lie algebra to quaternion	65 clock cycles
R2Q	Transform rotation matrix to quaternion	53 clock cycles
Q2R	Transform quaternion to a rotation matrix	14 clock cycles
Q_q	Quaternion multiplication	14 clock cycles

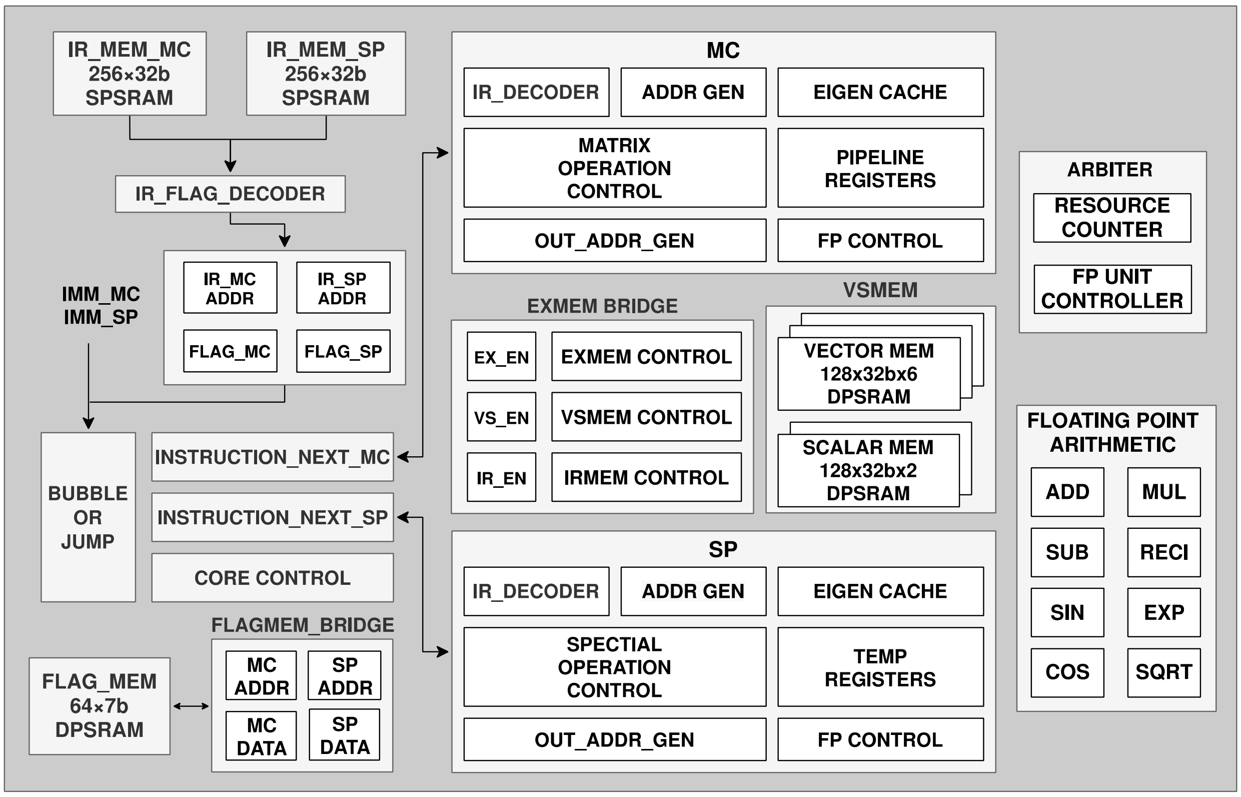

Figure 7. Modules and processing flow of the programmable computation core.

3.4. Feature Processing Engine

Figure 8 shows the structure of the feature processing engine, which includes a memory interface, a feature manager, a floating-point patch workspace, a coordinates alignment circuit, and interaction with the EKF engine. During the process of VIO, after IMU pre-integration, the pose of the feature concerning the last frame is exported out of the operator memory.

The patch workspace requests the 12×12 feature patch in a fixed-point format whose center locates where the feature is in the last frame from the feature extraction engine in the fixed-point vision pipeline, converts it to the floating-point format, and stores it temporarily into the dual-port rams. A bilinear interpolation is performed to sample an 8×8 patch from the original 12×12 patches according to the warping matrix and the updated patch coordinates [33]. Then, the patch workspace measures the difference between the old and the new patches, which are the gray value difference and grad difference along the X and Y axes. The information will be output to the coordinate alignment circuit.

The coordinates alignment circuit generates matrix A and vector b according to (7) and (9). The least-square equation optimizes the feature location and output to patch the workspace for resampling. The process ends when the iteration reaches the time limit or the 2D reprojection error is lower than a specific threshold. If the reprojection error is acceptable, the feature processing engine computes and stores the Jacobian matrix of measured results and noise in memory. EKF engine performs the update of the Kalman filter. After finishing the update, the feature processing engine sorts and compares the newly extracted features with the old ones. The feature processing engine replaces the old feature information and patches with the new one if the new feature is better.

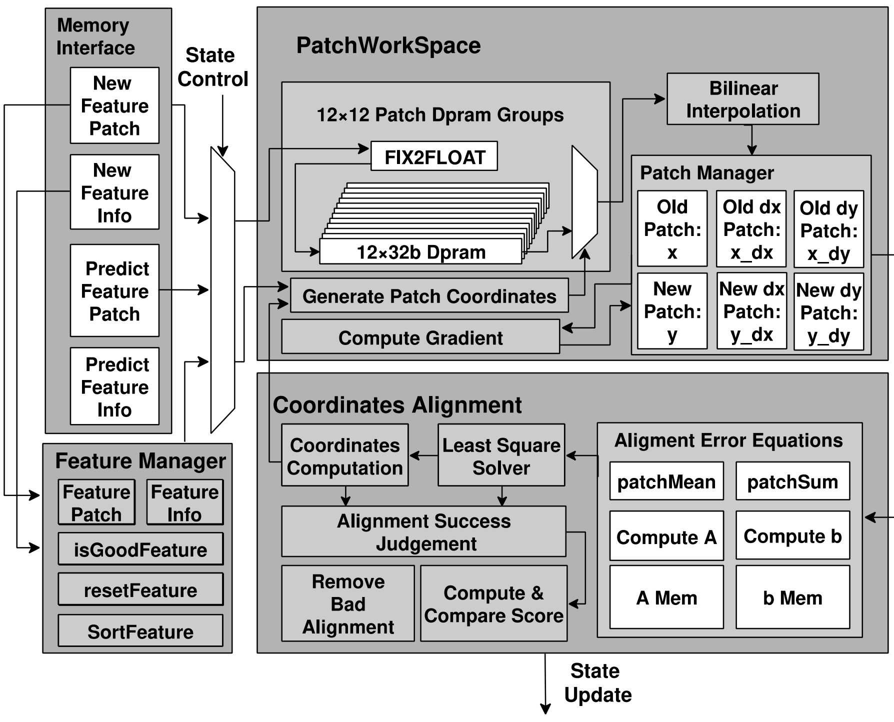

Figure 8. Detailed structure of the feature processing engine.

In order to save resources and reduce power consumption, 3 × 3 operation cores are used for matrix-vector operations of any size. To this end, this paper innovatively adopts the idea of vectorization, which is in matrix multiplication, the left matrix is input to the column from left to right, and the right matrix is input to the row from top to bottom. Each set of operations completes an M33 matrix, calculates a new M33 matrix every three rows and three columns, as shown in Figure 9, and finally, calculates the entire large matrix with low resource and power consumption.

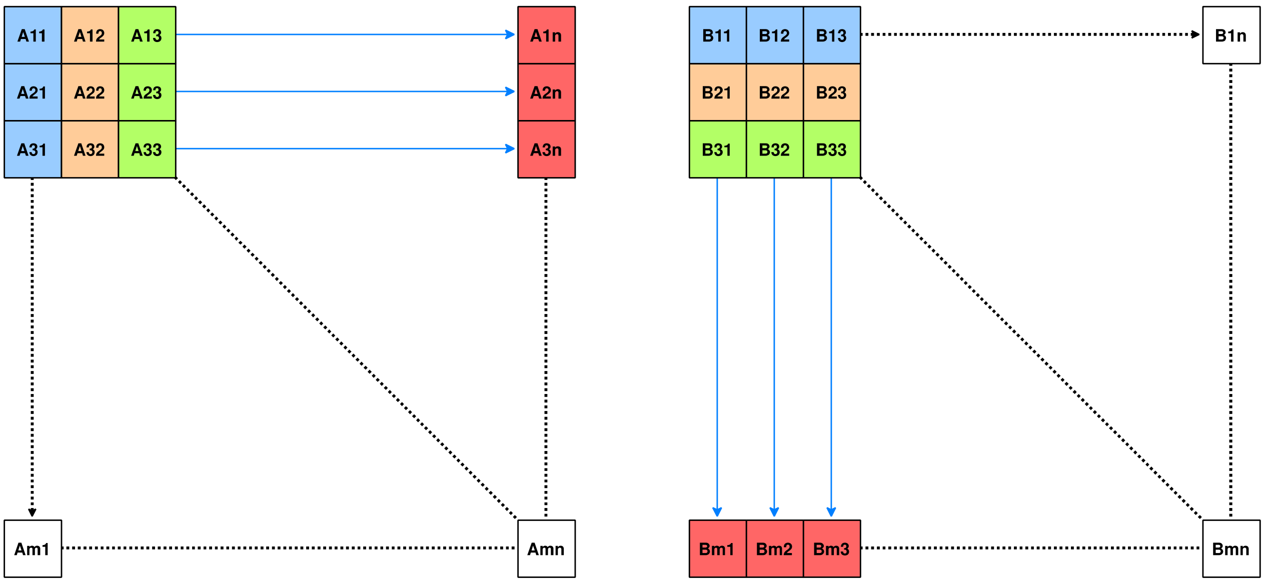

Figure 9. Schematic diagram of vectorized matrix multiplication strategy.

4. Implementation Results

4.1. Accuracy of Datasets Compared with Software Platform

A comparison to the software platform on a dataset is performed to verify the functionality and accuracy of the proposed VIO accelerator. The following were the main parameters of the PC: Ubuntu 18.04 (64-bit), Intel(R) Core (TM) i5-8265U CPU with 8G RAM. Moreover, we translate the ROVIO [33] algorithm into the ISA proposed by this paper. The dataset used in the evaluation is EuRoC, which is one of the most authoritative and challenging datasets. We translate the EKF process of ROVIO into the proposed ISA and evaluate them on EuRoC. As shown in Table 3, the accuracy loss is neglectable, indicating the high accuracy of the proposed accelerated core.

Table 3. Evaluation of accuracy.

Dataset	ROVIO (Software)	The Proposed Core
MH_1	0.19%	0.19%
MH_2	0.23%	0.23%
MH_3	0.47%	0.49%
MH_4	0.55%	0.52%
MH_5	0.78%	0.79%
V1_1	0.28%	0.26%
V1_2	0.35%	0.34%
V1_3	0.27%	0.25%
V2_1	0.26%	0.26%
V2_2	0.37%	0.40%
V2_3	0.61%	0.61%

4.2. Evaluation Platform and Experiments

The evaluation platform is constructed by a four-wheeled platform that can move smoothly, carrying the Xilinx XCVU 440 FPGA evaluation board with monocular camera mt9v034 and MPU9250, as shown in Figure 10. Figure 11 illustrates the measured experience map over frames during a typical SLAM operation in a flat corridor, which is about 70 m in total. The evaluation platform starts from one of the corners and moves smoothly at a speed of 0.35 m/s on average. After a loop of testing, we find that the trajectory is only slightly shifted, indicating the robust functionality and relatively high accuracy of the proposed accelerated core.

4.3. Discussions

Table 4 illustrates the implementation results of the proposed reconfigurable VIO accelerator. ASIC synthesis is performed to better compare with state-of-the-art ASIC implementation results. This work operates in the highest frequency both in FPGA and ASIC synthesis. Nevertheless, the on-chip memory usage is the lowest among these works for occupying only 70 KB when [29] is lower but with the usage of build-in SoC. Compared with FPGA implementations, slice LUTs and FFs consumed by the accelerator are 2.05× and 2.35× less than [28]. DSP consumption in this work is 8.03× and 1.8× less than [28] and [29]. As illustrated in Table 4, the core area in 28 nm CMOS technology is only 3 mm^2, which is 6.67× and 3.64× smaller in comparison to [27] and [26]. This work supports reconfigurable SLAM architecture, while others do not.

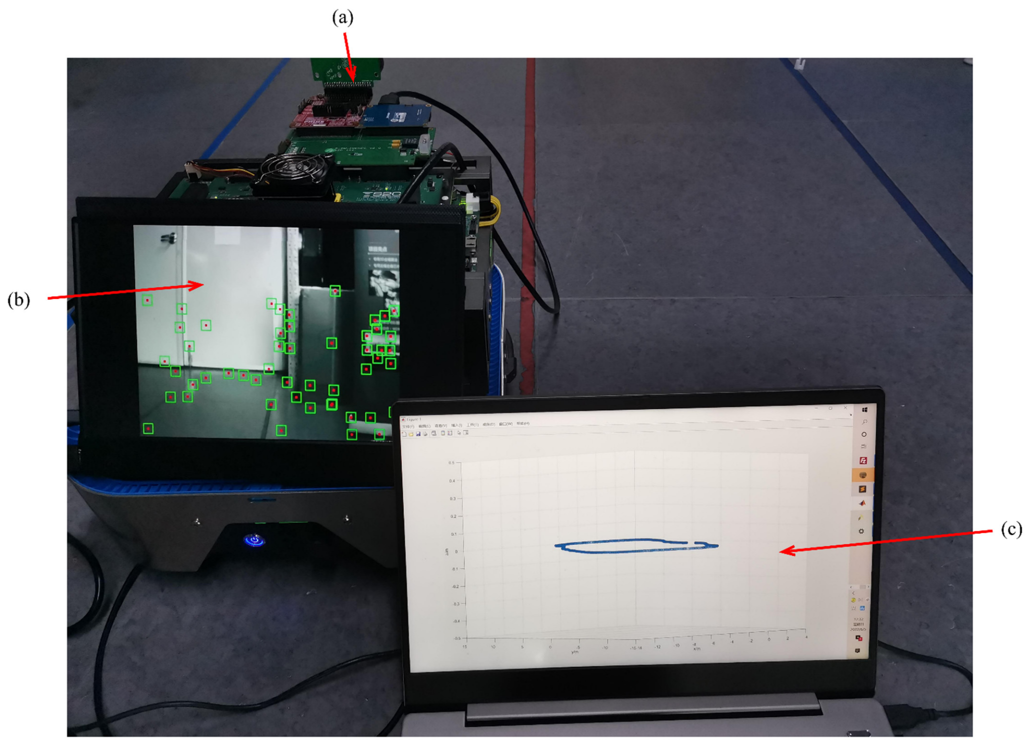

Figure 10. Demo evaluation platform. (**a**) VU440 FPGA board with mt9v034 image sensor and MPU9250 IMU. (**b**) New features are shown through the HDMI display. (**c**) Trajectory output in the x, y, and z axes (three dimensions).

Table 4. Hardware implementation results. The outstanding work in the comparison is bold.

	MIT 2017 [28]	**ICFPT 2021 [29]**	**JSSC 2019 [27]**	**ISSCC 2019 [26]**	**This Work**
Type	VIO	SLAM	VIO	SLAM	VIO
Odometry	IMU	Visual	IMU	Visual	IMU
FPGA Platform	Kintex-7 XC7K355T	UltraScale + XCZU7EV	N/A	N/A	UltraScale + XCVU440
Technology	N/A	N/A	65 nm	28 nm	28 nm *
Resolution	N/A	640×480	752×480	640×480	640×480
Speed	20 fps	15.5 fps	**171 fps**	80 fps	160 fps
Frequency	100 MHz	100 MHz	62.5 MHz/ 83.3 MHz	240 MHz	**160 MHz (250 MHz *)**
SoC	No	Yes	No	No	No
On-chip Memory	2048 KB	**61 KB**	854 KB	1126 KB	70 KB
LUTs	192,000	146,572	N/A	N/A	**91,802**
FFs	144,000	74,166	N/A	N/A	**61,107**
DSPs	771	173	N/A	N/A	**96**
Power	1.46 W	Not Given	24 mW	243.6 mW	**0.683 W** (179.52 mW)
Area	N/A	N/A	20 mm^2	10.92 mm^2	**3 mm^2**
Reconfigurable	No	No	No	No	Yes
Application	Nano and pico robots	Autonomous navigation	AR, VR and UAVs		AMRs

*: Synthesis results in 28 nm CMOS technology.

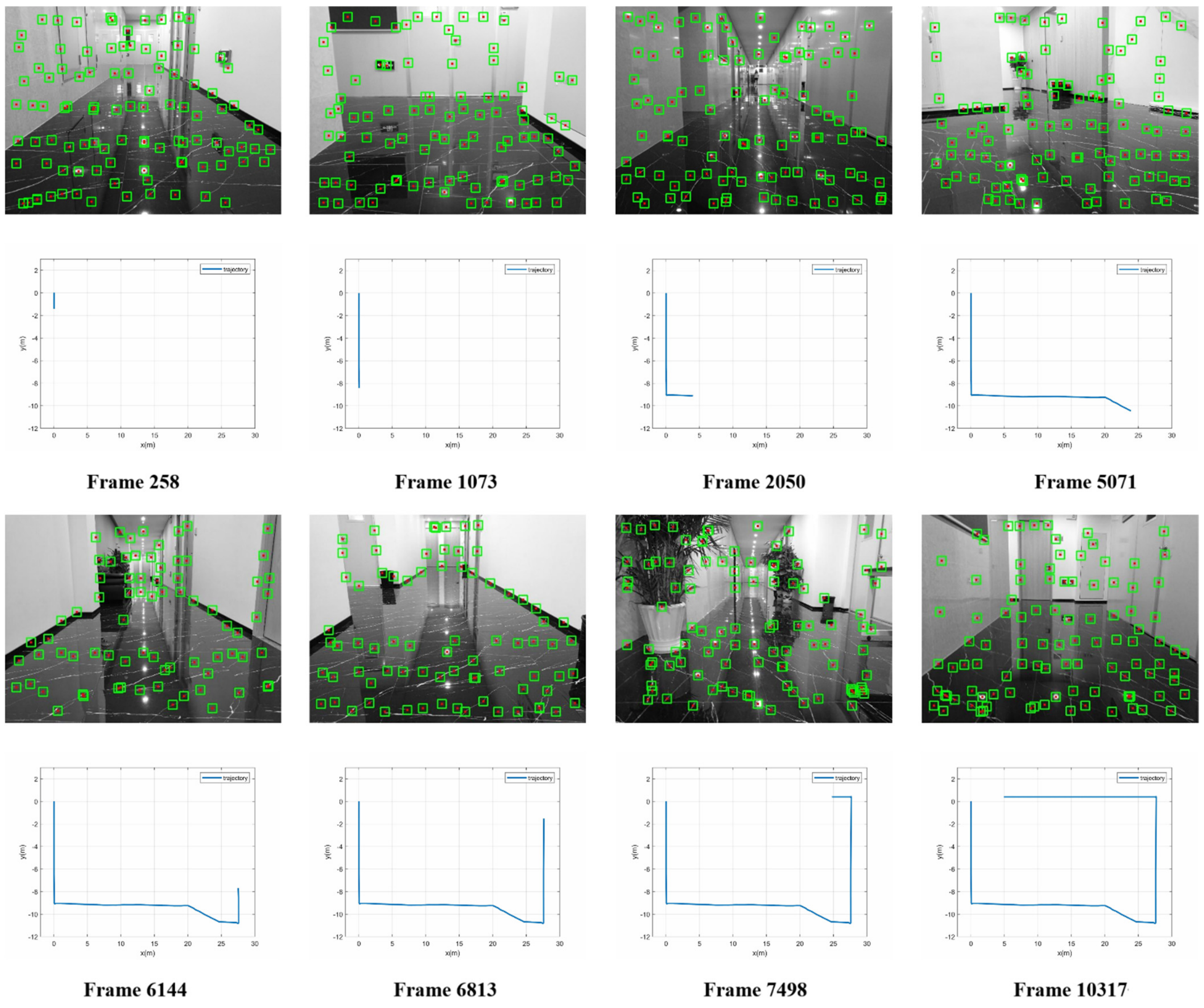

Figure 11. Measured experience map over frames during SLAM process.

The power consumption on the FPGA platform is about 0.683 W, which is lower than the 1.46 W on the Kintex-7 XC7K355T [28]. The power proportion of each module is shown in Figure 12. Based on the synthesis results in the 28 nm CMOS process, the power consumption is only 179.52 mW, which is much better than that in [26] with the same 28 nm CMOS technology, but still worse than that in [27] with 65 nm CMOS technology.

Due to the lightweight characteristic of the EKF optimization and FAST-based features, the proposed accelerated core essentially reduces the consumption of LUTs, FFs, and DSPs. Secondly, compared with other backend optimization techniques, such as bundle adjustment, EKF does not require storing the entire image, significantly reducing a large amount of on-chip memory. Finally, the delicate design improves the peak performance of the proposed accelerated core.

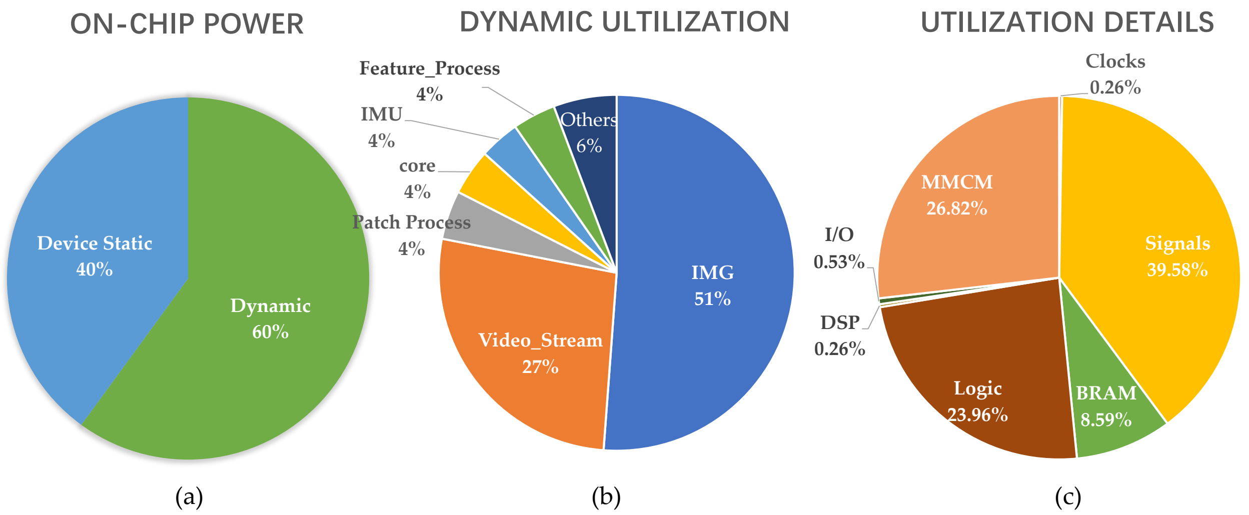

Figure 12. Power proportion of each module. (**a**) on-chip power; (**b**) dynamic utilization; (**c**) utilization details.

5. Conclusions

This paper proposed a reconfigurable visual inertial odometer (VIO) for the simultaneous localization and mapping (SLAM) application in autonomous mobile robots. The proposed accelerated core leveraged a lightweight feature extraction algorithm with the extended Kalman filter (EKF) update algorithm to provide high energy efficiency while achieving appropriate SLAM results. Furthermore, the reconfigurability of the accelerated core brings more potential downstream exploration for AMR applications.

The main hardware architecture of the VIO accelerator consisted of four sub-modules: (1) fixed-point vision pipeline, (2) memory interface, (3) programmable computation core, and (4) layers for the EKF engine. We first evaluated the accuracy both on benchmark datasets and real experimental tests. As shown in the implementation results, the on-chip memory usage of 70 KB was the lowest among the standalone works for SLAM. Meanwhile, the hardware-resource usage and power dissipation on the FPGA implementation and the synthesis of 28 nm CMOS technology also outperformed the state-of-the-art works on the same condition.

Author Contributions: Conceptualization, F.A. and Y.T.; methodology, H.D.; validation, Y.T. and H.W.; formal analysis, H.D.; investigation, Y.T., M.S. and Y.C.; data curation, Y.T., M.Z. and Z.Y.; writing—original draft preparation, Y.T. and M.S.; writing—review and editing, F.A., Q.Z., Y.T. and M.S.; visualization, Y.T. and H.D.; supervision, P.L., L.C. and F.A.; project administration, F.A.; funding acquisition, F.A. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially funded by the Science, Technology, and Innovation Commission of Shenzhen Municipality under grants JSGG20200102162401765, K2021390006, and K2021390007. This work was partially funded by the Special field pre-research foundation under grant 30121603, Guangdong College Students' Scientific and Technological Innovation pdjh2022b0456, and College Students' Innovative Entrepreneurial Training Plan Program 2022S20.

Institutional Review Board Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Silveira, G.; Malis, E.; Rives, P. An Efficient Direct Approach to Visual SLAM. *IEEE Trans. Robot.* **2008**, *24*, 969–979. [CrossRef]
2. Newcombe, R.A.; Lovegrove, S.J.; Davison, A.J. DTAM: Dense tracking and mapping in real-time. In Proceedings of the IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, 6–13 November 2011.
3. Engel, J.; Stuckler, J.; Cremers, D. Large-scale direct SLAM with stereo cameras. In Proceedings of the 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Hamburg, Germany, 28 September–2 October 2015.

4. Engel, J.; Koltun, V.; Cremers, D. Direct Sparse Odometry. *IEEE Trans. Pattern Anal. Mach. Intell.* **2016**, *40*, 611–625. [CrossRef] [PubMed]
5. Lowe, D.G. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vis.* **2004**, *60*, 91–110. [CrossRef]
6. Herbert, B.; Andreas, E.; Tinne, T.; LucVan, G. Speeded-Up Robust Features (SURF). *Comput. Vis. Image Underst.* **2008**, *110*, 346–359.
7. Rosten, E.; Porter, R.; Drummond, T. Faster and better: A machine learning approach to corner detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **2008**, *32*, 105–119. [CrossRef]
8. Castellanos, J.A.; Montiel, J.; Neira, J.; Tardós, J.D. The SPmap: A probabilistic framework for simultaneous localization and map building. *IEEE Trans. Robot. Autom.* **1999**, *15*, 948–952. [CrossRef]
9. Huang, G.P.; Mourikis, A.I.; Roumeliotis, S.I. Analysis and improvement of the consistency of extended Kalman filter based SLAM. In Proceedings of the 2008 IEEE International Conference on Robotics and Automation, Pasadena, CA, USA, 19–23 May 2008.
10. Eustice, R.M.; Singh, H.; Leonard, J.J. Exactly Sparse Delayed-State Filters for View-Based SLAM. *IEEE Trans. Robot.* **2007**, *22*, 1100–1114. [CrossRef]
11. Thrun, S.; Liu, Y.; Koller, D.; Ng, A.Y.; Ghahramani, Z.; Durrant-Whyte, H. Simultaneous Localization and Mapping With Sparse Extended Information Filters. *Int. J. Robot. Res.* **2004**, *23*, 693–716. [CrossRef]
12. Montemarlo, M. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In Proceedings of the Theaaai National Conference on Artificial Intelligence, Edmonton, AB, Canada, 28 July–1 August 2002.
13. Hähnel, D.; Burgard, W.; Fox, D.; Thrun, S. An Efficient {FastSLAM} Algorithm for Generating Maps of Large-Scale Cyclic Environments from Raw Laser Range Measurements. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2003, Las Vegas, NV, USA, 27–31 October 2003.
14. Grisetti, G.; Stachniss, C.; Burgard, W. Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters. *IEEE Trans. Robot.* **2007**, *23*, 34–46. [CrossRef]
15. Di, K.; Zhao, Q.; Wan, W.; Wang, Y.; Gao, Y. RGB-D SLAM Based on Extended Bundle Adjustment with 2D and 3D Information. *Sensors* **2016**, *16*, 1285. [CrossRef] [PubMed]
16. Alismail, H.; Browning, B.; Lucey, S. Photometric Bundle Adjustment for Vision-Based SLAM. In Proceedings of the Asian Conference on Computer Vision, Taipei, Taiwan, 20–24 November 2016.
17. Liu, H.; Chen, M.; Zhang, G.; Bao, H.; Bao, Y. ICE-BA: Incremental, Consistent and Efficient Bundle Adjustment for Visual-Inertial SLAM. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–23 June 2018.
18. Chao, J.; Geng, Z.X.; Wei, X.F.; Shen, C. SIFT implementation based on GPU. In Proceedings of the ISPDI 2013—Fifth International Symposium on Photoelectronic Detection and Imaging, Beijing, China, 25–27 June 2013.
19. Cornelis, N.; Gool, L.V. Fast scale invariant feature detection and matching on programmable graphics hardware. In Proceedings of the IEEE International Conference on Technologies for Practical Robot Applications, Woburn, MA, USA, 9–10 November 2009.
20. Yum, J.; Lee, C.H.; Kim, J.S.; Lee, H.J. A Novel Hardware Architecture with Reduced Internal Memory for Real-Time Extraction of SIFT in an HD Video. *IEEE Trans. Circuits Syst. Video Technol.* **2016**, *26*, 1. [CrossRef]
21. Wilson, C.; Zicari, P.; Craciun, S.; Gauvin, P.; Lam, H. A power-efficient real-time architecture for SURF feature extraction. In Proceedings of the International Conference on ReConFigurable mputing and FPGAs (ReConFig14), Riviera Maya, Mexico, 7–9 December 2015.
22. Ulusel, O.; Picardo, C.; Harris, C.B.; Reda, S.; Bahar, R.I. Hardware acceleration of feature detection and description algorithms on low-power embedded platforms. In Proceedings of the International Conference on Field Programmable Logic & Applications, FPL 2016, Lausanne, Switzerland, 29 August–2 September 2016.
23. Tertei, D.T.R.; Piat, J.; Devy, M. FPGA design of EKF block accelerator for 3D visual SLAM. *Comput. Electr. Eng.* **2016**, *55*, 123–137. [CrossRef]
24. Wang, J.; Zhan, Y.; Wang, Z.; Peng, Z.; Xu, J.; Liu, B.; Yu, G.; An, F.; Wang, C.; Zou, X. A Reconfigurable Matrix Multiplication Coprocessor with High Area and Energy Efficiency for Visual Intelligent and Autonomous Mobile Robots. In Proceedings of the 2021 IEEE Asian Solid-State Circuits Conference, A-SSCC 2021, Busan, Korea, 7–10 November 2021.
25. Liu, R.; Yang, J.; Chen, Y.; Zhao, W. eSLAM: An Energy-Efficient Accelerator for Real-Time ORB-SLAM on FPGA Platform. In Proceedings of the 56th Annual Design Automation Conference, Las Vegas, NV, USA, 2–6 June 2019.
26. Li, Z.; Chen, Y.; Gong, L.; Liu, L.; Kim, H.S. An 879GOPS 243mW 80fps VGA Fully Visual CNN-SLAM Processor for Wide-Range Autonomous Exploration. In Proceedings of the 2019 IEEE International Solid-State Circuits Conference—(ISSCC), San Francisco, CA, USA, 17–21 February 2019.
27. Suleiman, A.; Zhengdong, Z.; Carlone, L.; Karaman, S.; Sze, V. Navion: A 2-mW Fully Integrated Real-Time Visual-Inertial Odometry Accelerator for Autonomous Navigation of Nano Drones. *IEEE J. Solid-State Circuits* **2019**, *54*, 1106–1119. [CrossRef]
28. Zhang, Z.; Suleiman, A.Z.; Carlone, L.; Sze, V.; Karaman, S. Visual-Inertial Odometry on Chip: An Algorithm-and-Hardware Co-design Approach. In Proceedings of the Robotics: Science and Systems, Cambridge, MA, USA, 12–16 July 2017.
29. Wang, C.; Liu, Y.; Zuo, K.; Tong, J.; Ding, Y.; Ren, P. ac2SLAM: FPGA Accelerated High-Accuracy SLAM with Heapsort and Parallel Keypoint Extractor. In Proceedings of the 2021 International Conference on Field-Programmable Technology (ICFPT), Auckland, New Zealand, 6–10 December 2021.

30. Lupton, T.; Sukkarieh, S. Visual-Inertial-Aided Navigation for High-Dynamic Motion in Built Environments Without Initial Conditions. *IEEE Trans. Robot.* **2012**, *28*, 61–76. [CrossRef]
31. Forster, C.; Carlone, L.; Dellaert, F.; Scaramuzza, D. IMU Preintegration on Manifold for Efficient Visual-Inertial Maximum-a-Posteriori Estimation. In Proceedings of the Robotics: Science and Systems XI, Rome, Italy, 13–17 July 2015.
32. Tong, Q.; Li, P.; Shen, S. VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. *IEEE Trans. Robot.* **2017**, *34*, 1–17.
33. Bloesch, M.; Omari, S.; Hutter, M.; Siegwart, R. Robust visual inertial odometry using a direct EKF-based approach. In Proceedings of the 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Hamburg, Germany, 28 September–2 October 2015.
34. Tan, Y.; Deng, H.; Sun, M.; Zhou, M.; Chen, Y.; Chen, L.; Wang, C.; An, F. A Reconfigurable Coprocessor for Simultaneous Localization and Mapping Algorithms in FPGA. *IEEE Trans. Circuits Syst. II Express Briefs* 2022, *in press*.