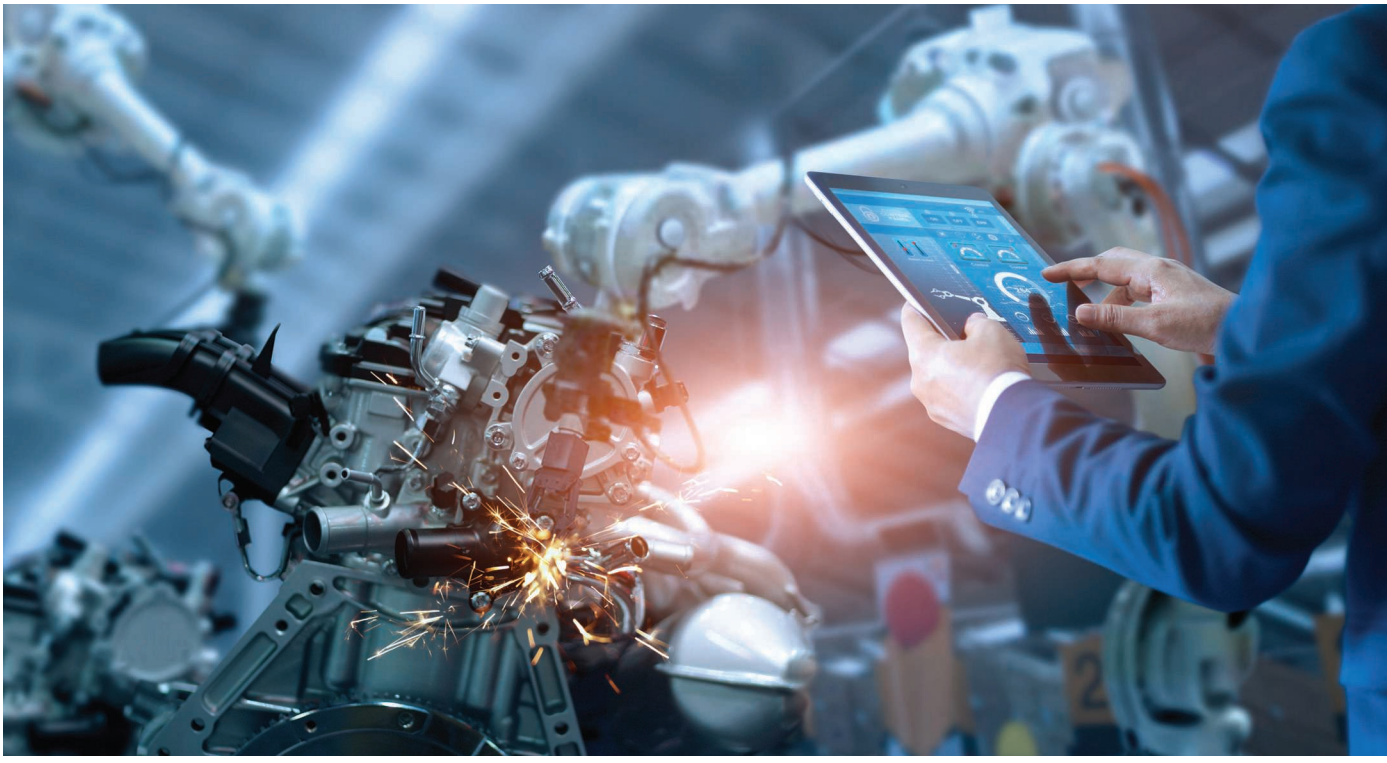
# A Survey of FPGA-Based Robotic Computing

Zishen Wan,* Bo Yu,* Thomas Yuang Li, Jie Tang, Yuhao Zhu,
Yu Wang, Arijit Raychowdhury, and Shaoshan Liu



## Abstract

Recent researches on robotics have shown significant improvement, spanning from algorithms, mechanics to hardware architectures. Robotics, including manipulators, legged robots, drones, and autonomous vehicles, are now widely applied in diverse scenarios. However, the high computation and data complexity of robotic algorithms pose great challenges to its applications. On the one hand, CPU platform is flexible to handle multiple robotic tasks. GPU platform has higher computational capacities and easy-to-use development frameworks, so they have been widely adopted in several applications. On the other hand, FPGA-based robotic accelerators are becoming increasingly competitive alternatives, especially in latency-critical and power-limited scenarios. With specialized designed hardware logic and algorithm kernels, FPGA-based accelerators can surpass CPU and GPU in performance and energy efficiency. In this paper, we give an overview of previous work on FPGA-based robotic accelerators covering different stages of the robotic system pipeline. An analysis of software and hardware optimization techniques and main technical issues is presented, along with some commercial and space applications, to serve as a guide for future work.

## I. Introduction

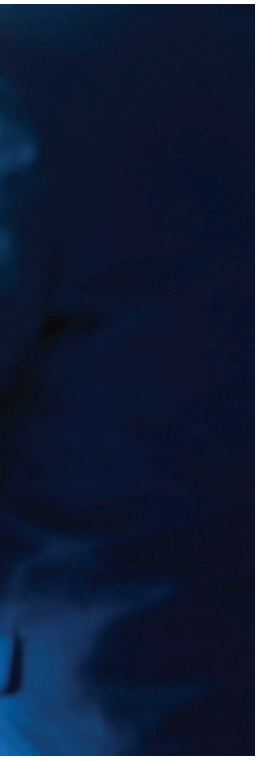
Over the last decade, we have seen significant progress in the development of robotics, spanning from algorithms, mechanics to hardware platforms. Various robotic systems, like manipulators, legged robots, unmanned aerial vehicles, self-driving cars

*Digital Object Identifier 10.1109/MCAS.2021.3071609*
*Date of current version: 24 May 2021*

* *These authors contributed equally to this work.*
*Corresponding author: Shaoshan Liu (email: shaoshan.liu@perceptin.io).*

have been designed for search and rescue [1], [2], exploration [3], [4], package delivery [5], entertainment [6], [7] and more applications and scenarios. These robots are on the rise of demonstrating their full potential. Take drones, a type of aerial robots, as an example, the number of drones has grown by 2.83x between 2015 and 2019 based on the U.S. Federal Aviation Administration (FAA) report [8]. The registered number has reached 1.32 million in 2019, and the FFA expects this number will come to 1.59 billion by 2024.

However, robotic systems are pretty complicated [9]–[11]. They tightly integrate many technologies and algorithms, including sensing, perception, mapping, localization, decision making, control, etc. This complexity poses many challenges for the design of robotic edge computing systems [12], [13]. On the one hand, the robotic system needs to process an enormous amount of data in real-time. The incoming data often comes from multiple sensors and is highly heterogeneous. However, the robotic system usually has limited onboard resources, such as memory storage, bandwidth, and compute capabilities, making it hard to meet the real-time requirements. On the other hand, the current state-of-the-art robotic system usually has strict power constraints on the edge that cannot support the amount of computation required for performing tasks, such as 3D sensing, localization, navigation, and path planning. Therefore, the computation and storage complexity, as well as real-time and power constraints of the robotic system, hinder its wide application in latency-critical or power-limited scenarios [14].




Therefore, it is essential to choose a proper compute platform for the robotic system. CPU and GPU are two widely used commercial compute platforms. CPU is designed to handle a wide range of tasks quickly and is often used to develop novel algorithms. A typical CPU can achieve 10-100 GFLOPS with below 1GOP/J power efficiency [15]. In contrast, GPU is designed with thousands of processor cores running simultaneously, which enable massive parallelism. A typical GPU can perform up to 10 TOPS performance and become a good candidate for high-performance scenarios. Recently, benefiting in part from the better accessibility provided by CUDA/OpenCL, GPU has been predominantly used in many robotic applications. However, conventional CPU and GPUs usually consume 10 W to 100 W of power, which are orders of magnitude higher than what is available on the resource-limited robotic system.

Besides CPU and GPU, FPGAs are attracting attention and becoming a platform candidate to achieve energy-efficient robotics tasks processing. FPGAs require little power and are often built into small systems with less memory. They have the ability to parallel computations massively and makes use of the properties of perception (e.g., stereo matching), localization (e.g., SLAM), and planning (e.g., graph search) kernels to remove additional logic and simplify the implementation. Taking into account hardware characteristics, several algorithms are proposed which can be run in a hardware-friendly way and achieve similar software performance. Therefore, FPGAs are possible to meet real-time requirements while achieving high energy efficiency compared to CPUs and GPUs.

Unlike the ASIC counterparts, FPGA technology provides the flexibility of on-site programming and re-programming without going through re-fabrication with a modified design. Partial Reconfiguration (PR) takes this flexibility one step further, allowing the modification of an operating FPGA design by loading a partial configuration file. Using PR, part of the FPGA can be reconfigured at runtime without compromising the integrity of the applications running on those parts of the device that are not being reconfigured. As a result, PR can allow different robotic applications to time-share part of an FPGA, leading to energy and performance efficiency, and making FPGA a suitable computing platform for dynamic and complex robotic workloads.

FPGAs have been successfully utilized in commercial autonomous vehicles. Particularly, over the past three years, PerceptIn has built and commercialized autonomous vehicles for micromobility, and PerceptIn's products have been deployed in China, US, Japan and Switzerland. In this paper, we review how PerceptIn developed its computing system by relying heavily on FPGAs, which perform not only heterogeneous sensor synchronizations, but also the acceleration of software components on the critical path. In addition, FPGAs are used heavily in space robotic applications, for FPGAs offered unprecedented flexibility and significantly reduced the design

*Zishen Wan,* School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA and John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138 USA. Bo Yu,* Thomas Yuang Li and Shaoshan Liu, PerceptIn Inc, Fremont, CA 94539 USA. Jie Tang, School of Computer Science and Engineering, South China University of Technology, Guangzhou, Guangdong, China. Yuhao Zhu, Department of Computer Science, University of Rochester, Rochester, NY 14627 USA. Yu Wang, Department of Electronic Engineering, Tsinghua University, Beijing, China. Arijit Raychowdhury, School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA.*

cycle and development cost. In this paper, we also delve into space-grade FPGAs for robotic applications.

The rest of paper is organized as follows: Section II introduces the basic workloads of the robotic system. Section III, IV and V reviews the various perception, localization and motion planning algorithms and their implementations on FPGA platforms. In section VI, we discuss about FPGA partial reconfiguration techniques. Section VII and VIII present robotics FPGA applications in commercial and space areas. Section IX concludes the paper.

## II. Overview of Robotics workloads

### *A. Overview*

Robotics is not one technology but rather an integration of many technologies. As shown in Fig 1, the stack of the robotic system consists of three major components: application workloads, including sensing, perception, localization, motion planning, and control; a software edge subsystem, including operating system and runtime layer; and computing hardware, including both microcontrollers and companion computers.

We focus on the robotic application workloads in this section. The application subsystem contains multiple algorithms that are used by the robot to extract meaningful information from raw sensor data to understand the environment and dynamically make decisions about its actions.

### *B. Sensing*

The sensing stage is responsible for extracting meaningful information from the sensor raw data. To enable intelligent actions and improve reliability, the robot platform usually supports a wide range of sensors. The number and type of sensors are heavily dependent on the specifications of the workload and the capability of the onboard compute platform. The sensors can include the following:

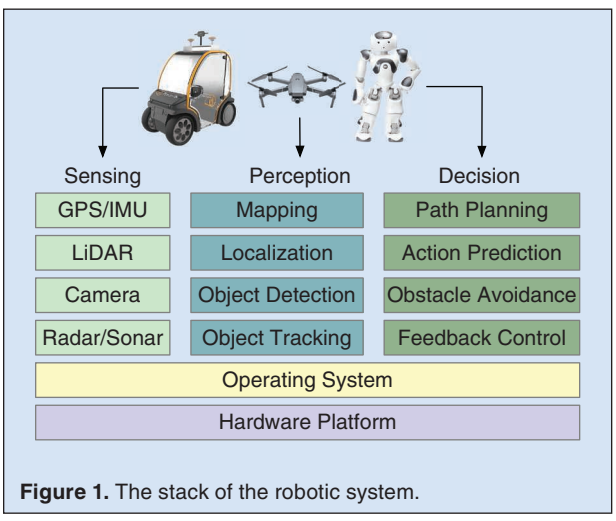



**Figure 1.** The stack of the robotic system.

**Cameras.** Cameras are usually used for object recognition and object tracking, such as lane detection in autonomous vehicles and obstacle detection in drones, etc. RGB-D camera can also be utilized to determine object distances and positions. Take autonomous vehicle as an example, the current system usually mounts eight or more 1080p cameras around the vehicle to detect, recognize and track objects in different directions, which can greatly improve safety. Usually, these cameras run at 60 Hz, which will process multiple gigabytes of raw data per second when combined.

**GNSS/IMU.** The global navigation satellite system (GNSS) and inertial measurement unit (IMU) system help the robot localize itself by reporting both inertial updates and an estimate of the global location at a high rate. Different robots have different requirements for localization sensing. For instance, 10 Hz may be enough for low-speed mobile robots, but high-speed autonomous vehicles usually demand 30 Hz or higher for localization, and high-speed drones may need 100 Hz or more for localization, thus we are facing a broad spectrum of sensing speeds. Fortunately, different sensors have their own advantages and drawbacks. GNSS can enable fairly accurate localization, while it runs at only 10 Hz, thus unable to provide real-time updates. By contrast, both accelerometer and gyroscope in IMU can run at 100–200 Hz, which can satisfy the real-time requirement. However, IMU suffers bias wandering over time or perturbation by some thermo-mechanical noise, which may lead to an accuracy degradation in the position estimates. By combining GNSS and IMU, we can get accurate and real-time updates for robots.

**LiDAR.** Light detection and ranging (LiDAR) is used for evaluating distance by illuminating the obstacles with laser light and measuring the reflection time. These pulses, along with other recorded data, can generate precise and three-dimensional information about the surrounding characteristics. LiDAR plays an important role in localization, obstacle detection and avoidance. As indicated in [16], the choice of sensors dictates the algorithm and hardware design. Take autonomous driving as an instance, almost all the autonomous vehicle companies use LiDAR at the core of their technologies. Examples include Uber, Waymo, and Baidu. PerceptIn and Tesla are among the very few that do not use LiDAR and, instead, rely on cameras and vision-based systems, and in particular PerceptIn's data demonstrated that for the low-speed autonomous driving scenario, LiDAR processing is slower than camera-based vision processing, but increases the power consumption and cost.

**Radar and Sonar.** The Radio Detection and Ranging (Radar) and Sound Navigation and Ranging (Sonar)

system is used to determine the distance and speed to a certain object, which usually serves as the last line of defense to avoid obstacles. Take autonomous vehicle as an example, a danger of collision may occur when near obstacles are detected, then the vehicle will apply brakes or turn to avoid obstacles. Compared to LiDAR, the Radar and Sonar system is cheaper and smaller, and their raw data is usually fed to the control processor directly without going through the main compute pipeline, which can be used to implement some urgent functions as swerving or applying the brakes.

### *C. Perception*

The sensor data is then fed into the perception layer to sense the static and dynamic objects, and build a reliable and detailed representation of the robot's environment using computer vision techniques (including deep learning).

The perception layer is responsible for object detection, segmentation and tracking. There are obstacles, lane dividers and other objects to detect. Traditionally, a detection pipeline starts with image pre-processing, followed by a region of interest detector and then a classifier that outputs detected objects. In 2005, Dalal and Triggs [17] proposed an algorithm based on histogram of orientation (HOG) and support vector machine (SVM) to model both the appearance and shape of the object under various condition. The goal of segmentation is to give the robot a structured understanding of its environment. Semantic segmentation is usually formulated as a graph labeling problem with vertices of the graph being pixels or super-pixels. Inference algorithms on graphical models such as conditional random field (CRF) [18], [19] are used. The goal of tracking is to estimate the trajectory of moving obstacles. Tracking can be formulated as a sequential Bayesian filtering problem by recursively running the prediction step and correction step. Tracking can also be formulated by tracking-by-detection handling with Markovian decision process (MDP) [20], where an object detector is applied to consecutive frames and detected objects are linked across frames.

In recent years, deep neural networks (DNN), also known as deep learning, have greatly affected computer vision and made significant progress in solving robot perception problems. Most state-of-the-art algorithms now apply one type of neural network based on convolution operation. Fast R-CNN [21], Faster R-CNN [22], SSD [23], YOLO [24], and YOLO9000 [25] were used to get much better speed and accuracy in object detection. Most CNN-based semantic segmentation work is based on Fully Convolutional Networks (FCN) [26], and there are some recent work in spatial pyramid pooling network [27] and pyramid scene parsing network (PSPNet) [28] to combine global image-level information with the locally extracted feature. By using auxiliary natural images, a stacked autoencoder model can be trained offline to learn generic image features and then applied for online object tracking [29].

### *D. Localization*

The localization layer is responsible for aggregating data from various sensors to locate the robot in the environment model.

GNSS/IMU system is used for localization. The GNSS consist of several satellite systems, such as GPS, Galileo and BeiDou, which can provide accurate localization results but with a slow update rate. In comparison, IMU can provide a fast update with less accurate rotation and acceleration results. A mathematical filter, such as Kalman Filter, can be used to combine the advantages of the two and minimize the localization error and latency. However, this sole system has some problems, such as the signal may bounce off obstacles, introduce more noise, and fail to work in closed environments.

LiDAR and High-Definition (HD) maps are used for localization. LiDAR can generate point clouds and provide a shape description of the environment, while it is hard to differentiate individual points. HD map has a higher resolution compared to digital maps and makes the route familiar to the robot, where the key is to fuse different sensor information to minimize the errors in each grid cell. Once the HD map is built, a particle filter method can be applied to localize the robot in real-time correlated with LiDAR measurement. However, the LiDAR performance may be severely affected by weather conditions (e.g., rain, snow) and bring localization error.

Cameras are used for localization as well. The pipeline of vision-based localization is simplified as follows: 1) by triangulating stereo image pairs, a disparity map is obtained and used to derive depth information for each point; 2) by matching salient features between successive stereo image frames in order to establish correlations between feature points in different frames, the motion between the past two frames is estimated; and 3) by comparing the salient features against those in the known map, the current position of the robot is derived [30].

Apart from these techniques, sensor fusion strategy is also often utilized to combine multiple sensors for localization, which can improve the reliability and robustness of robot [31], [32].

### *E. Planning and Control*

The planning and control layer is responsible for generating trajectory plans and passing the control commands

based on the original and destination of the robot. Broadly, prediction and routing modules are also included here, where their outputs are fed into downstream planning and control layers as input. The prediction module is responsible for predicting the future behavior of surrounding objects identified by the perception layer. The routing module can be a lane-level routing based on lane segmentation of the HD maps for autonomous vehicles.

Planning and Control layers usually include behavioral decision, motion planning and feedback control. The mission of the behavioral decision module is to make effective and safe decisions by leveraging all various input data sources. Bayesian models are becoming more and more popular and have been applied in recent works [33], [34]. Among the Bayesian models, Markov Decision Process (MDP) and Partially Observable Markov Decision Process (POMDP) are the widely applied methods in modeling robot behavior. The task of motion planning is to generate a trajectory and send it to the feedback control for execution. The planned trajectory is usually specified and represented as a sequence of planned trajectory points, and each of these points contains attributes like location, time, speed, etc. Low-dimensional motion planning problems can be solved with grid-based algorithms (such as Dijkstra [35] or A* [36]) or geometric algorithms. High-dimensional motion planning problems can be dealt with sampling-based algorithms, such as Rapidly-exploring Random Tree (RRT) [37] and Probabilistic Roadmap (PRM) [38], which can avoid the problem of local minima. Reward-based algorithms, such as the Markov decision process (MDP), can also generate the optimal path by maximizing cumulative future rewards. The goal of feedback control is to track the difference between the actual pose and the pose on the predefined trajectory by continuous feedback. The most typical and widely used algorithm in robot feedback control is PID.

While optimization-based approaches enjoy mainstream appeal in solving motion planning and control problems, learning-based approaches [39]–[43] are becoming increasingly popular with recent developments in artificial intelligence. Learning-based methods, such as reinforcement learning, can naturally make full use of historical data and iteratively interact with the environment through actions to deal with complex scenarios. Some model the behavioral level decisions via reinforcement learning [41], [43], while other approaches directly work on motion planning trajectory output or even direct feedback control signals [40]. Q-learning [44], Actor-Critic learning [45], policy gradient [38] are some popular algorithms in reinforcement learning.

## III. Perception on FPGA

### *A. Overview*

Perception is related to many robotic applications where sensory data and artificial intelligence techniques are involved. Examples of such applications include stereo matching, object detection, scene understanding, semantic classification, etc. The recent developments in machine learning, especially deep learning, have exposed robotic perception systems to more tasks. In this section, we will focus on the recent algorithms and FPGA implementations in the stereo vision system, which is one of the key components in the robotic perception stage.

Real-time and robust stereo vision systems are increasingly popular and widely used in many perception applications, e.g., robotics navigation, obstacle avoidance [46] and scene reconstruction [47]–[49]. The purpose of stereo vision systems is to obtain 3 D structure information of the scene using stereoscopic ranging techniques. The system usually has two cameras to capture images from two points of view within the same scenario. The disparities between the corresponding pixels in two stereo images are searched using stereo matching algorithms. Then the depth information can be calculated from the inverse of this disparity.

Throughout the whole pipeline, stereo matching is the bottleneck and time-consuming stage. The stereo matching algorithms can be mainly classified into two categories: local algorithms [50]–[56] and global algorithms [57]–[61]. Local methods compute the disparities by only processing and matching the pixels around the points of interest within windows. They are fast and computationally-cheap, and the lack of pixel dependencies makes them suitable for parallel acceleration. However, they may suffer in textureless areas and occluded regions, which will result in incorrect disparities estimation.

In contrast, global methods compute the disparities by matching all other pixels and minimizing a global cost function. They can achieve much higher accuracy than local methods. However, they tend to come at high computation cost and require much more resources due to their large and irregular memory access as well as the sequential nature of algorithms, thus not suitable for real-time and low-power applications. Many research works in stereo systems focus on the speed and accuracy improvement of stereo matching algorithms, and some of the implementations are summarized in Tab. I

### *B. Local Stereo Matching on FPGA*

Local algorithms are usually based on correlation, where the process involves finding matching pixels in the left and right image patches by aggregating costs within a

specific region. There are many ways for cost aggregation, such as the sum of absolute differences (SAD) [62], the sum of squared differences (SSD) [63], normalized cross-correlation (NCC) [64], and census transform (CT) [65]. Many FPGA implementations are based on these methods. Jin et al. [66] develop a real-time stereo vision system based on census rank transformation matching cost for 640 × 480 resolution images. Zhang et al. [67] propose a real-time high definition stereo matching design on FPGA based on mini-census transform and cross-based cost aggregation, which achieves 60 fps at 1024 × 768 pixel stereo images. The implementation of Honegger et al. [68] achieves 127 fps at 376 × 240 pixel resolution with 32 disparity levels based on block matching. Jin et al. [69] further achieve 507.9 fps for 640 × 480 resolution images by applying fast local consistent dense stereo functions and cost aggregation.

### *C. Global Stereo Matching on FPGA*

Global algorithms can provide state-of-the-art accuracy and disparity map quality, however, they are usually processed through high computational-intensive optimization techniques or massive convolutional neural networks, making them difficult to be deployed on resource-limited embedded systems for real-time applications. However, some works have attempted to implement global algorithms on FPGA for better performance. Park et al. [70] present a trellis-based stereo matching system on FPGA with a low error rate and achieved 30 fps at 320 × 240 resolution with 128 disparity levels. Sabihuddin et al. [71]

**Table I.**
**Comparison of Stereo Vision Systems on FPGA platforms, across local stereo matching, global stereo matching, semi-global stereo matching (SGM) and efficient large-scale stereo matching (ELAS) algorithms. The results reported in each design are evaluated by frame rate (fps), image resolution (width × height), disparity levels, million disparity estimations per second (MDE/s), power (W), resource utilization (logic% and BRAM%) and hardware platforms, where MDE/s = width × height × fps × disparity.**

| Algorithm | Reference | Frame Rate (FPS) | Image Resolution (Width × Height) | Disparity Level | MDE/s | Power (W) | Resource(%) Logic/BRAM | FPGA Platform |
|---|---|---|---|---|---|---|---|---|
| Local Stereo Matching | Jin et al. [66] | 230 | 640 × 480 | 64 | 4522 | – | 34.0/95.0 | Xilinx Virtex-4 |
| | Zhang et al. [67] | 60 | 1024 × 768 | 64 | 3020 | 1.56 | 61.8/67.0 | XC4VLX200-10 |
| | Honegger et al. [68] | 127 | 376 × 240 | 32 | 367 | 2.8 | 49.0/68.0 | Altera EP3SL150 |
| | Jin et al. [69] | 507.9 | 640 × 480 | 60 | 9362 | 3.35 | 81.0/39.7 | Altera Cyclone III EP3C80 Xilinx Vertex-6 |
| Global Stereo Matching | Park et al. [70] | 30 | 320 × 240 | 128 | 295 | – | –/– | Xilinx Virtex II pro-100 |
| | Sabihuddin et al. [71] | 63.54 | 640 × 480 | 128 | 2498 | – | 23.0/58.0 | Xilinx XC2VP100 |
| | Jin et al. [72] | 32 | 640 × 480 | 60 | 590 | 1.40 | 72.0/46.0 | Xilinx XC4VLX160 |
| | Zha et al. [59] | 30 | 1920 × 1680 | 60 | 5806 | – | 84.8/91.9 | Xilinx Kintex 7 |
| | Puglia et al. [60] | 30 | 1024 × 768 | 64 | 1510 | 0.17 | 57.0/53.0 | Xilinx Virtex-7 XC7Z020CLG484-1 |
| Semi-Global Stereo Matching | Banz et al. [74] | 37 | 640 × 480 | 128 | 1455 | 2.31 | 51.2/43.2 | Xilinx Virtex-5 |
| | Wang et al. [75] | 42.61 | 1600 × 1200 | 128 | 10472 | 2.79 | 93.9/97.3 | Altera 5SGSMD5K2 |
| | Cambuim et al. [76] | 127 | 1024 × 768 | 128 | 12784 | – | –/– | Altera Cyclone IV |
| | Rahnama et al. [77] | 72 | 1242 × 375 | 128 | 4292 | 3.94 | 75.7/30.7 | Xilinx ZC706 |
| | Cambuim et al. [78] | 25 | 1024 × 768 | 256 | 5033 | 6.5 | 50.0/38.0 | Altera Cyclone IV GX, Stratix IV GX |
| | Zhao et al. [79] | 147 | 1242 × 375 | 64 | 4382 | 9.8 | 68.7/38.7 | Xilinx Ultrascale + ZCU102 |
| Efficient Large-Scale Stereo Matching | Rahnama et al. [80] | 47 | 1242 × 375 | – | – | 2.91 | 11.9/15.7 | Xilinx ZC706 |
| | Rahnama et al. [81] | 50 | 1242 × 375 | – | – | 5 | 70.7/8.7 | Xilinx ZCU104 |

implement a dynamic programming maximum likelihood (DPML) based hardware architecture for dense binocular disparity estimation and achieved 63.54 fps at 640 × 480 pixel resolution with 128 disparity levels. The design in Jin et al. [72] uses a tree-structured dynamic programming method, and achieves 58.7 fps at 640 × 480 resolution as well as a low error rate. Recently, some other adaptations of global approaches for FPGA-implementation have been proposed, such as cross-trees [59], dynamic programming for DNA sequence alignment [60], and graph cuts [73], where all of these implementations achieve real-time processing.

### *D. Semi-Global Matching on FPGA*

Semi-global matching (SGM) [82] bridges the gap between local and global methods, and achieves a notable improvement in accuracy. SGM calculates the initial matching disparities by comparing local pixels, and then approximates an image-wide smoothness constraint with global optimization, which can obtain more robust disparity maps through this combination. There are several critical challenges for implementing SGM on hardware, e.g., data dependence, high complexity, and large storage, so this is an active research field with recent works proposing FPGA-friendly variants of SGM [74], [75], [83]–[85].

Banz et al. [74] propose a systolic-array based hardware architecture for SGM disparity estimation along with a two-dimensional parallelization concept for SGM. This design achieves 30 fps performance at 640 × 480 pixel images with a 128-disparity range on the Xilinx Virtex-5 FPGA platform. Wang et al. [75] implement a complete real-time FPGA-based hardware system that supports both absolute difference-census cost initialization, cross-based cost aggregation and semi-global optimization. The system achieves 67 fps at 1024 × 768 resolution with 96 disparity levels on the Altera Stratix-IV FPGA platform, and 42 fps at 1600 × 1200 resolution with 128 disparity levels on the Altera Stratix-V FPGA platform. The design in Cambuim et al. [76] uses a scalable systolic-array based architecture for SGM based on the Cyclone IV FPGA platform, and it achieves a 127 fps image delivering rate in 1024 × 768 pixel HD resolution with 128 disparity levels. The key point of this design is the combination of disparity and multi-level parallelisms such as image line processing to deal with data dependency and irregular data access pattern problems in SGM. Later, to improve the robustness of SGM and achieve a more accurate stereo matching, Cambuim et al. [78] combine the sampling-insensitive absolute difference in the pre-processing phase, and propose a novel streaming architecture to detect noisy and occluded regions in the post-processing phase. The design is evaluated in a full stereo vision system using two heterogeneous platforms, DE2i-150 and DE4, and achieves a 25 fps processing rate in 1024 × 768 HD maps with 256 disparity levels.

While most existing SGM designs on FPGA are implemented using the register-transfer level (RTL), some works leveraged the high-level synthesis (HLS) approach. Rahnama et al. [77] implement an SGM variation on FPGA using HLS, which achieves 72 fps speed at 1242 × 375 pixel size with 128 disparity levels. To reduce the design effort and achieve an appropriate balance among speed, accuracy and hardware cost, Zhao et al. [79] recently propose FP-Stereo for building high-performance SGM pipelines on FPGAs automatically. A series of optimization techniques are applied in this system to exploit parallelism and reduce resource consumption. Compared to GPU designs [86], it achieves the same accuracy at a competitive speed while consuming much less energy.

To compare these implementations, the depth quality of are evaluated on Middlebury Benchmark [87], with four image pairs *Tsukuba*, *Venus*, *Teddy*, *Cones*. As shown in Tab. II, there is a general trade-off between accuracy and processing speed. The stereo vision system designs in Tab. I are drawn as points in Fig. 2 (if both power and speed number are reported), using $log_{10}$ (*power*) as *x*-coordinate and $log_{10}$ (*speed*) as *y*-coordinate ($y - x = log_{10}$ (*energy_efficiency*)). Besides FPGA-based implementations, we also plot GPU and CPU experimental results as a comparison to FPGA designs' performance. In general, local and semi-global stereo matching designs have achieved higher performance and energy efficiency than global stereo matching designs. As introduced in section III-C, global stereo matching algorithms usually involve massive computational-intensive optimization techniques. Even for the same design, varying design parameters (e.g., window size) may result in a 10x difference in energy efficiency. Compared to GPU and CPU-based designs, FPGA-based designs have achieved higher energy efficiency, and the speed of many FPGA implementations have surpassed general-purpose processors.

### *E. Efficient Large-Scale Stereo Matching on FPGA*

Another popular stereo matching algorithm that offers a good trade-off between speed and accuracy is Efficient Large-Scale Stereo Matching (ELAS) [90], which is currently one of the fastest and accurate CPU algorithms concerning the resolution on Middlebury dataset. ELAS implements a slanted plane prior very effectively while its dense estimation of depth is completely decomposable over all pixels, which make it attractive for easily parallelized.

Rahnama et al. [80] first implement and evaluate an FPGA accelerated adaptation of the ELAS algorithm, which achieved a frame rate of 47 fps (up to 30× compared high-end CPU) while consuming under 4 W of power. By taking advantage of different components on the SoC, several elaboration blocks such as feature extraction and dense matching are executed on FPGA, while I/O and other conditional/sequential blocks are executed on ARM-core CPU. The authors also reveal the strategy to accelerate complex and computationally diverse algorithms for low power and real-time systems by collaboratively utilizing different compute components. Later, by leveraging and combining the best features of SGM and ELAS-based methods, Rahnama et al. [81] propose a sophisticated stereo approach and achieve an 8.7% error rate on the challenging KITTI 2015 dataset at over 50 fps, with a power consumption of only 4.5 W.

### *F. CNN-Based Stereo Vision System on FPGA*

Convolutional neural networks (CNNs) have been demonstrated to perform very well on many vision tasks such as image classification, object detection, and semantic segmentation. Recently, CNN has also been utilized in stereo estimation [91], [92] and stereo matching [93]. CNN is applied to determine SGM penalties [94], estimate real-time optical flow disparity [95] and predict cost volume computation and aggregation [96].

CNN has been deployed on FPGA platforms in several works [97]–[100], with an example of lightweight YOLOv2 for object detection [101]. Nakahara et al. implement a pipelined-based architecture for lightweight YOLOv2 with a binarized CNN on Xilinx ZCU102 FPGA platform. This design achieves a 40.81 fps object detection speed, which is 177.4× faster than ARM Cortex-A57 and 27.5× faster than NVIDIA Pascal embedded GPU. Many FPGA-based CNN accelerator implementations have been summarized in [15].

## IV. Localization on FPGA

### *A. Overview*

For robots, one of the most critical tasks is localization and mapping. Simultaneous Localization and Mapping (SLAM) is an advanced robot navigation algorithm for constructing or updating a map of unknown surroundings while simultaneously keeping tracking the robot's location. Localization and mapping are two concurrent tasks and cannot be solved independently from each other. Localizing a robot requires a sufficiently detailed map, and constructing or updating or a map

**Table II.**
**A comparison between different designs on performance (MDE/s) and accuracy results on Middlebury Benchmark. (The lower of average bad pixel rate means the better stereo matching performance.)**

| Reference | MDE/s | Tsukuba | | | Venus | | | Teddy | | | Cones | | | Average Bad Pixel Rate |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | nonocc[1] | all[2] | disc[3] | nonocc | all | disc | nonocc | **all** | disc | nonocc | all | disc | |
| Shan et al. [88] | 15437 | — | 24.5 | — | — | 15.7 | — | — | 15.1 | — | — | 14.1 | — | all = 17.3 |
| Shan et al. [89] | 13076 | 3.62 | 4.15 | 14.0 | 0.48 | 0.87 | 2.79 | 7.54 | 14.7 | 19.4 | 3.51 | 11.1 | 9.64 | 7.65 |
| Wang et al. [75] | 10472 | 2.39 | 3.27 | 8.87 | 0.38 | 0.89 | 1.92 | 6.08 | 12.1 | 15.4 | 2.12 | 7.74 | 6.19 | 5.61 |
| Jin et al. [69] | 9362 | 1.66 | 2.17 | 7.64 | 0.4 | 0.6 | 1.95 | 6.79 | 12.4 | 17.1 | 3.34 | 8.97 | 9.62 | 6.05 |
| Jin et al. [66] | 4522 | 9.79 | 11.6 | 20.3 | 3.59 | 5.27 | 36.8 | 12.5 | 21.5 | 30.6 | 7.34 | 17.6 | 21.0 | 17.2 |
| Zhang et al. [67] | 3020 | 3.84 | 4.34 | 14.2 | 1.2 | 1.68 | 5.62 | 7.17 | 12.6 | 17.4 | 5.41 | 11.0 | 13.9 | 8.2 |
| Banz et al. [74] | 1455 | 4.1 | — | — | 2.7 | — | — | 11.4 | — | — | 8.4 | — | — | nonocc = 6.7 |
| Jin et al. [72] | 590 | 1.43 | 2.51 | 6.6 | 2.37 | 2.97 | 13.1 | 8.11 | 13.6 | 15.5 | 8.12 | 13.8 | 16.4 | 8.71 |

[1]nonocc: average percentage of bad pixels in non-occluded regions.
[2]all: average percentage of bad pixels in all regions.
[3]disc: average percentage of bad pixels in discontinuous regions.

requires accurate landmarks or pose estimates from known positions.

Many SLAM algorithms have been developed in the last decades to improve the accuracy and robustness, and its implementation comes in a diverse set of sizes and shapes. One end of the spectrum is dense SLAM algorithms [102]–[105], which can generate high-quality maps of the environment with complex computations. Dense SLAM algorithms usually are executed on powerful and high-performance machines to ensure real-time performance. At the same time, the intensive computation characteristic makes dense SLAM hard to deploy on edge devices. The other end of the spectrum is sparse SLAM [106]–[109], which is computationally light by only selecting limited numbers of landmarks or features.

To form a compromise in terms of compute intensity and accuracy quality between these two extremes, a family of works described as semi-dense SLAM has emerged [110], [111]. They aim to achieve better computational efficiency compared to dense methods by only processing a subset of high-quality sensory information while providing a more dense and informative map compared to sparse methods.

A typical SLAM system includes two components: the front-end and the back-end, which are with different computational characteristics. The front-end associates sensory measurements in consecutive frames to physical landmarks. It incrementally deduces the robot motion by applying geometry constraints on the associated sensory observations. The back-end tries to minimize errors introduced from sensory measurement noises by performing optimizations on a batch of observed landmarks and tracked poses. Filter based (e.g., Extended Kalman Filter) and numerical optimization based (e.g., bundle adjustment) algorithms are two prevalent methods for SLAM back-end.

A critical challenge to mobile robot localization is accuracy and efficiency under stringent power and resource constraints. To avoid losing tracked features due to large motions between consecutive frames, SLAM systems need to process sensory data at a high frame rate. For example, open data sets for evaluating localization algorithms [112], [113] for drones and vehicles provide images at 10 to 20 fps. Low power computing systems are always required to extend the battery life of mobile robots. Most SLAM algorithms are developed on CPU or GPU platforms, of which power consumption is hundreds of Watts. To execute SLAM efficiently on mobile robots and meet real-time and power constraints, specialized chips and accelerators have been developed. FPGA SoCs provide rich sensor interfaces, dedicated hardware logic and programmability, hence they have been explored in diverse ways in recent years.

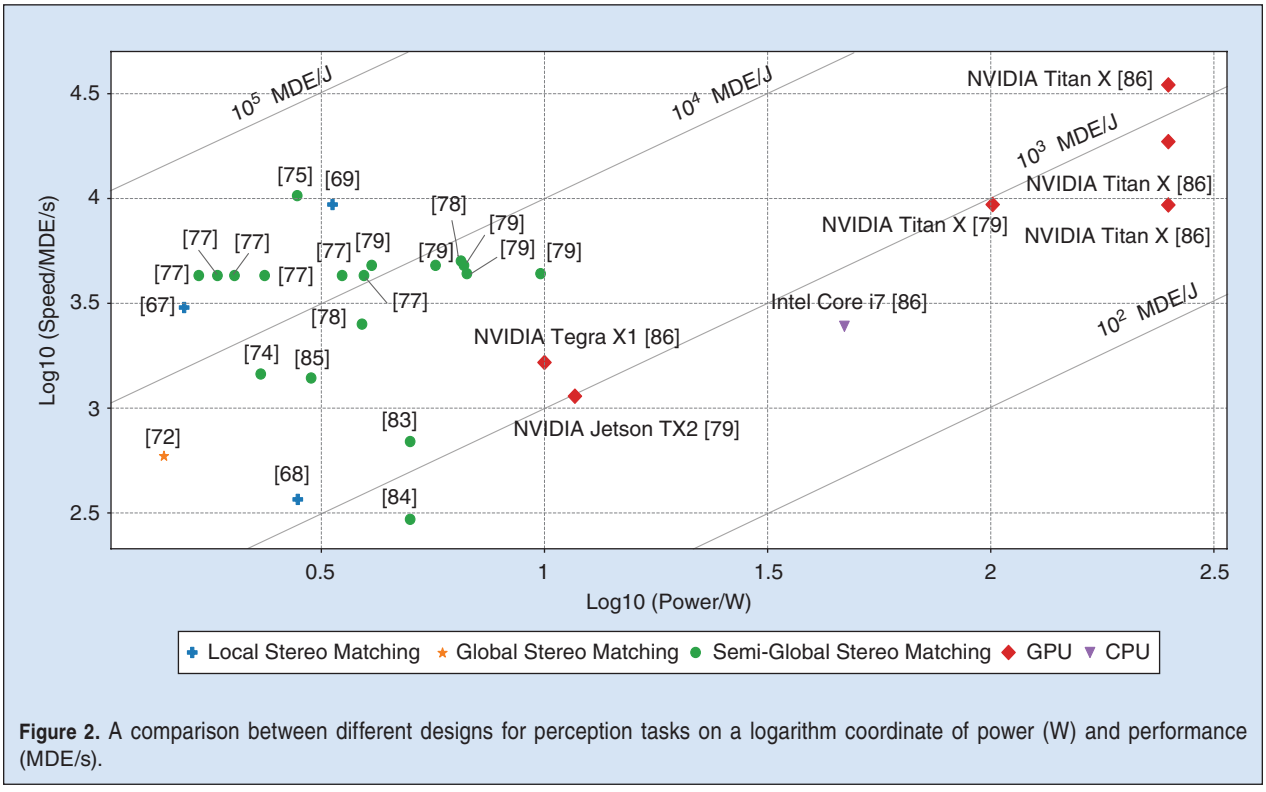



**Figure 2.** A comparison between different designs for perception tasks on a logarithm coordinate of power (W) and performance (MDE/s).

We summarize and discuss FPGA-based accelerators for SLAMs in the following sections.

### *B. Dense SLAM on FPGA*

Dense SLAM can construct high quality and complete models of the environment, and most of them are running in high-end hardware platforms (especially GPU). One of the representative real-time dense SLAM algorithms is KinectFusion [114], which was released by Microsoft in 2011. As a scene reconstruction algorithm, it continuously updates the global 3D map and tracks the location of depth cameras within the surrounding environment. KinectFusion is generally composed of three algorithms: ray-casting algorithm for generating graphics from surface information, iterative closest point (ICP) algorithm for camera-tracking and volumetric integration (VI) algorithm for integrating depth streams into the 3D surface. Several works have attempted to implement real-time dense SLAM algorithms on a heterogeneous system with FPGA embedded.

Several works implement computationally intensive components of dense SLAMs, such as ICP and VI, on FPGA to accelerate the critical path. Belshaw [102] presents an FPGA implementation of the ICP algorithm, which achieves over 200 fps tracking speed with low tracking errors. This design divides the ICP algorithm into filtering, nearest neighbor, transform recovery and transform application stages. It leverages fixed-point arithmetic and the power of two data points to utilize FPFA logic efficiently. Williams [103] notices that the nearest neighbor search takes up the majority of ICP runtime, and then proposes two hybrid CPU-FPGA architectures to accelerate the bottleneck of the ICP-SLAM algorithm. The implementation is performed with Vivado HLS, a high-level synthesis tool from Xilinx, and achieves a maximum 17.22× speedup over the ARM software implementation. Hoorick [104] presents an FPGA-based heterogeneous framework using a similar HLS method to accelerate the KinectFusion algorithm and explored various ways of dataflow and data management patterns. Gautier et al. [105] embed both ICP and VI algorithms on an Altera Stratix V FPGA by using the OpenCL language and the Altera OpenCL SDK. This design was a heterogeneous system with NVIDIA GTX 760 GPU and Altera Stratix V FPGA. By distributing different workloads on different parts of SoC, the entire system achieves up to 28 fps real-time speed.

### *C. Sparse SLAM on FPGA*

Sparse SLAM algorithms usually use a small set of features to track and maintain a sparse map of surrounding environments. These algorithms exhibit lower power consumption but are limited to the localization accuracy.

#### 1) EKF-SLAM

EKF-SLAM [106] is a class of algorithms that utilizes the extended Kalman Filter (EKF) for SLAM. EKF-SLAM algorithms are typically feature-based and use the maximum likelihood algorithm for data association. Several heterogeneous architectures using multi-core CPUs, GPUs, DSPs, and FPGAs are proposed to accelerate the complex computation in EKF-SLAM algorithms. Bonato et al. [115] presents the first FPGA-based architecture for the EKF-SLAM based algorithm that is capable of processing 2 D maps at up to 1800 features at real-time with a frequency of 14 Hz, compared to 572 features with Pentium CPU and 131 features with ARM. They analyze the computational complexity and memory bandwidth requirements for FPGA-based EKF-SLAM, and then propose an architecture with a parallel memory access pattern to accelerate the matrix multiplication. This design achieves two orders of magnitude more power-efficient than a general-purpose processor.

Similarly, Tertei et al. [116] propose an efficient FPGA-SoC hardware architecture for matrix multiplication with systolic arrays to accelerate EKF-SLAM algorithms. The setup of this design is a PLB peripheral to PPC440 hardcore embedded processor on a Virtex5 FPGA, and it achieves a 7.3× speedup with a processing frequency of 44 Hz compared to the pure software implementation. Later, taking into account the symmetry in cross-covariance matrix-related computations, Tertei et al. [117] improve the previous implementation to further reduce the computational time and on-chip memory storage on Zynq-7020 FPGA.

DSP is also leveraged in some works to accelerate EKF-SLAM algorithms. Vincke et al. [118] implement an efficient implementation of EKF-SLAM on a low-cost heterogeneous architecture system consisting of a single-core ARM processor with a SIMD coprocessor and a DSP core. The EKF-SLAM program is partitioned into different functional blocks based on the profiling characteristics results. Compared to a non-optimized ARM implementation, this design achieved 4.7× speed up from 12 fps to 57 fps. In a later work, Vincke et al. [119] replace the single-core ARM with a double-core ARM to optimize the non-optimized blocks using the OpenMP library. This design achieves a 2.75× speedup compared to non-optimized implementation.

#### 2) ORB-SLAM

ORB-SLAM [107] is an accurate and widely-used sparse SLAM algorithm for monocular, stereo, and RGB-D cameras. Its framework usually consists of five main procedures: feature extraction, feature matching, pose estimation, pose optimization and map updating. Based on

the profiling results on a quad-core ARM v8 mobile SoC, feature extraction is the most computation-intensive stage in the ORB-SLAM system, which consumes more than half of CPU resources and energy budget [120].

ORB based feature extraction algorithm usually consists of two parts, namely Oriented Feature from Accelerated Segment Test (oFAST) [121] based feature detection and Binary Robust Independent Elementary (BRIEF) [122] based feature descriptors computation. To accelerate this bottleneck, Fang et al. [120] design and implement a hardware ORB feature extractor and achieved a great balance between performance and energy consumption, which outperforms ARM Krait by 51% and Intel Core i5 by 41% in computation latency as well as outperforms ARM Krait by 10% and Intel Core i5 by 83% in energy consumption. Liu et al. [123] propose an energy-efficient FPGA implementation eSLAM to accelerate both feature extraction and feature matching stages. This design achieves up to 3× and 31× speedup in framerate, as well as up to 71× and 25× in energy efficiency improvement compared to Intel i7 and ARM Cortex-A9 CPUs, respectively. This eSLAM design utilizes a rotationally symmetric ORB descriptor pattern to make the algorithm more hardware-friendly, resulting in a 39% less latency compared to [120]. Rescheduling and parallelizing optimization techniques are also exploited to improve the computation throughput in eSLAM design.

Scale-invariant feature transform (SIFT) and Harris corner detector are also commonly-used feature extraction methods. SIFT is invariant to rotation and translation. Gu et al. [109] implement SIFT-feature based SLAM algorithm on FPGA and accelerate the matrix computation part to achieve speedup. Harris corner detector is used to extract corners and features of an image, and Schulz et al. [124] propose an implementation of Harris and Stephen corner detector optimized for an embedded SoC platform that integrates a multicore ARM processor with Zynq-7000 FPGA. Taking into account I/O requirements and the advantage of parallelization and pipeline, this design achieves a speedup of 1.77 compared to dual-core ARM processors.

**3) Fast-SLAM**

One of the key limitations of EKF-SLAM is its computational complexity since EKF-SLAM requires time quadratic in the number of landmarks to incorporate each sensor update. In 2002, Montemerlo et al. [108] propose an efficient SLAM algorithm called Fast-SLAM. Fast-SLAM decomposes the SLAM problem into a robot localization problem and a landmark estimation problem. It recursively estimates the full posterior distribution over landmark positions and robot path with a logarithmic scale.

Abouzahir et al. [125] implement Fast-SLAM 2.0 on a CPU-GPGPU-based SoC architecture. The algorithm is partitioned into function blocks, and each of them is implemented on the CPU or GPU accordingly. This optimized and efficient CPU-GPGPU partitioning enables accurate localization and a 37× execution speedup compared to non-optimized implementation on a single-core CPU. Further, Abouzahir et al. [126] perform a complete study of the processing time of different SLAM algorithms under popular embedded devices, and demonstrate that Fast-SLAM2.0 allowed a compromise between the consistency of localization results and computation time. This algorithm is then optimized and implemented on GPU and FPGA using HLS and parallel computing frameworks OpenCL and OpenGL. It is observed that the global processing time of FastSLAM2.0 on FPGA implementations achieves 7.5× acceleration compared to high-end GPU. The processing frequency achieves 102 fps and meets the real-time performance constraints of an operated robot.

**4) VO-SLAM**

The visual odometry based SLAM algorithm (VO-SLAM) also belongs to the Sparse SLAM class with low computational complexity. Gu et al. [109] implement the VO-SLAM algorithm on a DE3 board (Altera Stratix III) to perform drift-free pose estimation, resulting in localization results accurate to 1-2cm. A Nios II soft-core is used as a master processor. The authors design a dedicated matrix accelerator and propose a hierarchical matrix computing mechanism to support application requirements. This design achieves a processing speed of 31 fps with 30000 global map features, and 10× energy saving for each frame processing compared to Intel i7 CPU.

***D Semi-Dense SLAM on FPGA***

Semi-dense SLAM algorithms have emerged to provide a compromise between sparse SLAM and dense SLAM algorithms, which attempt to achieve improved efficiency and dense point clouds. However, they are still usually computationally intensive and require multicore CPUs for real-time processing.

Large-Scale Direct Monocular SLAM (LSD-SLAM) is one of the state-of-the-art and widely-used semi-dense SLAM algorithms, and it directly operates on image intensities for both tracking and mapping problems. The camera is tracked by direct image alignment, while geometry is estimated from semi-dense depth maps acquired by filtering over multiple stereo pixel-wise comparisons.

Several works have explored LSD-SLAM FPGA-SoC implementation. Boikos et al. [127] investigate the

performance and acceleration opportunities for LSD-SLAM in the SoC system. This design achieves an average framerate of more than 4 fps for a resolution of 320 × 240 with an estimated power of less than 1 W, which is a 2× acceleration and more than 4.3× energy efficiency compared to a software version running on embedded CPUs. The author also notes that the communication between two accelerators is via DDR memory since the produced intermediate data is too large to be fully cached on the FPGA. Hence, it is important to optimize the memory architecture (e.g., data movement and caching techniques) to ensure the scalability and compatibility of the design.

To further improve the performance of [127], Boikos et al. [128] re-implement the design using a dataflow architecture and distributed asynchronous blocks to allow the memory system and the custom hardware pipelines to function at peak efficiency. This implementation can process and track more than 22 fps with an embedded power budget and achieves a 5× speedup over [127].

Furthermore, Boikos et al. [129] combine a scalable depth estimation with direct semi-dense SLAM architecture and propose a complete accelerator for semi-dense SLAM on FPGA. This architecture achieved more than 60 fps at the resolution of 640 × 480 and an order of magnitude power consumption improvement compared to Intel i7-4770 CPU. This implementation leverages multi-rate and multi-modal units to deal with LSD-SLAM's complex control flow. A new dataflow paradigm is also proposed where the kernel is linked with a single consumer and a single producer to achieve high efficiency.

### *E. CNN-Based SLAM on FPGA*

Recently, CNNs have made significant progress in the perception and localization ability of the robots compared to handcrafted methods. Take one of the main SLAM components, feature extraction, as an example, the CNN-based approach SuperPoint [130] can achieve 10%-30% higher matching accuracy compared to handcrafted ORB. Other CNN-based methods, such as DeepDesc [131] and GeM [132], also present significant improvements in feature extraction and descriptor generation stage. However, CNN has a much higher computational complexity and requires more memory footprint.

Several works have explored to deploy CNN on FPGAs. Xilinx DPU [133] is one of the state-of-the-art programmable dedicated to CNN, which has a specialized instruction set and works efficiently across various CNN topologies. Xu et al. [134] propose a hardware architecture to accelerate CNN-based feature extraction SuperPoint on the Xilinx ZCU102 platform and achieve 20 fps in a real-time SLAM system. The key point of this design is an optimized software dataflow to deal with the extra post-processing operations within CNN-based feature extraction networks. 8-bit fixed-point numerics are leveraged in the post-processing operations and CNN backbone. Similar hardware-oriented model compression techniques (e.g., data quantization and weight reduction) have been widely adopted in robotics and CNN related designs [135]–[142].

Yu et al. [143] build a CNN-based monocular decentralized-SLAM (DSLAM) on the Xilinx ZCU102 MPSoC platform with DPU. DSLAM is usually used in multi-robot applications that can share environment information and locations between agents. To accelerate the main components in DSLAM, namely visual odometry (VO) and decentralized place recognition (DPR), the authors adopt CNN-based Depth-VO-Feat [144] and NetVLAD [145] to replace handcrafted approaches and propose a cross-component pipeline scheduling algorithm to improve the performance.

To enable multi-tasking processing in embedded robots on CNN accelerators, Yu et al. [146] further propose an INterruptible CNN accelerator (INCA) with a novel virtual-instruction-based interrupt method. Feature extraction and place recognition of DSLAM are deployed and accelerated on the same CNN accelerator of the embedded FPGA system, and the interrupt response latency is reduced by 1%.

### *F. Bundle Adjustment on FPGA*

Besides the hardware implementation of the frontend of the SLAM system, several works investigate to accelerate the backend of the SLAM system, mainly Bundle Adjustment (BA). BA is heavily used in robot localization [107], [147], autonomous driving [148], space exploration missions [149] and some commercial products [150], where it is usually employed in the last stage of the processing pipeline to refine camera trajectories and 3D structures further.

Essentially, BA is a massive joint non-linear optimization problem that usually consumes a significant amount of power and processing time in both offline visual reconstruction and real-time localization applications.

Several works aim to accelerate BA on multi-core CPUs or GPUs using parallel or distributed computing techniques. Jeong et al. [151] exploit efficient memory handling and fast block-based linear solving, and propose a novel embedded point iterations method, which substantially improves the BA performance on CPU. Wu et al. [152] present a multi-core parallel processing solution running on CPUs and GPUs. The matrix-vector product is carefully restructured in this design to reduce memory requirements and compute latency substantially. Eriksson et al. [153] propose a distributed approach for very large scale global bundle adjustment

computation to achieve BA performance improvement. The authors present a consensus framework using the proximal splitting method to reduce the computational cost. Similarly, Zhang et al. [154] propose a distributed formulation to accelerate the global BA computation without much distributed computing communication overhead.

To better deploy BA in embedded systems with strict power and real-time constraints, recent works explore BA algorithm acceleration using specialized hardware. The design in [155] implements both the image frontend and BA backend of a VIO algorithm on a single-chip for nano-drone scale applications. Liu et al. [156] propose a hardware-software co-designed BA hardware accelerator and its implementation on an embedded FPGA-SoC to achieve higher performance and power efficiency simultaneously. Especially, a co-observation optimization technique and a hardware-friendly differentiation method are proposed to accelerate BA operations with optimized usage of memory and computation resources. Sun et al. [157] present a hardware architecture running local BA on FPGAs, which works without external memory access and refines both cameras poses and 3D map points simultaneously.

### *G. Discussion*

We summarize FPGA based SLAM systems in Tab. III. It only includes works that implement the whole SLAM on an FPGA and provide overall performance and power evaluation. The works in the table adopt a similar FPGA-SoC architecture that accelerates computationally intensive components by FPGA fabrics and offloads others works to embedded processors on FPGAs. Compared with sparse method, the semi-dense implementation has lower frame rate, which is mainly due to the high resolution data processed in the pipeline. Due to the high frame rates and low power consumption, sparse SLAM FPGA have been used in drones and autonomous vehicles [16]. The two sparse SLAM implementations achieve similar performance in terms of frame rate. Compared with the ORB design, the VO SLAM design includes pre-processing and outliers removal hardware, such as image rectification and RANSAC, which lead to a more accurate but power inefficient implementation.

**Table III.**
**Comparison of FPGA SLAM Systems.**

| | Method | Platform | Frame Rate | Power | Indoor Error |
|---|---|---|---|---|---|
| Boikos et al. [127] | Semi Dense | Xilinx Zynq 7020 SoC | 4.5 fps | 2.5 W | na |
| Liu et al. [123] | ORB | Xilinx Zynq 7000 SoC | 31 fps | 1.9 W | 4.5 cm |
| Gu et al. [109] | VO | Altera Stratix III | 31 fps | 5.9 W | 2 cm |

## V. Planning and Control on FPGA

### *A. Overview*

Planning and control are the modules that compute how the robot should maneuver itself. They usually include behavioral decision, motion planning and feedback control kernels. Without loss of generality, we focus on the motion planning algorithms and their FPGA implementations in this section.

As a fundamental problem in the robotic system, motion planning aims to find the optimal collision-free path from the current position to a goal position for a robot in complex surroundings. Generally, motion planning contains three steps, namely roadmap construction, collision detection and graph search [38], [158]. Motion planning will become a relatively complicated problem when robots work with a high degree of freedom (DOF) configurations since the search space will be exponentially increased. Typically, state-of-the-art CPU-based approaches take a few seconds to find a collision-free trajectory [159]–[161], making the existing motion planning algorithms too slow to meet the real-time requirement for complex robot tasks and environments. Several works have investigated approaches to speed up motion planning, either for each stage or whole pipeline.

### *B. Roadmap Construction*

In the roadmap construction step, the planner generates a set of states in the robot's configuration space and then connects them with edges to construct a general-purpose roadmap in the obstacle-free space. Each state represents a robot's configuration, and each edge represents a possible robot movement. Conventional algorithms build the roadmap by randomly sampling poses from configuration space at runtime to navigate around the obstacles present at that time.

Several works explore roadmap construction acceleration. Yershova et al. [162] improve the nearest neighbor search to accelerate roadmap construction by orders of magnitude compared to the naive nearest-neighbor searching. Wang et al. [163] reduce the computation workload by trimming roadmap edges and keeping the roadmap to a reasonable size to achieve speedup. Different from online runtime approaches, Murray et al. [164] completely remove the runtime latency by conducting the roadmap construction only once at the design time. A more general and much larger roadmap is precomputed and allows for fast and successive

queries in complex environments without reprogramming the accelerator during runtime.

### *C. Collision Detection*

In the collision detection step, the planner determines whether there are potential collisions with the environment or the robot itself during movement. Specifically, collision detection is the primary challenge in motion planning, which often comprises 90% of the processing time [165].

Several works leverage data parallelization computing on GPUs to achieve speedup [165]–[167]. For example, Bialkowski et al. [165] divide the RRT* algorithm of collision detection tasks into three parallel dimensions and construct thread block grids to execute collision computations simultaneously. However, GPU can only provide a constant speedup factor due to the core limitations, which is still hard to achieve the real-time requirement.

Recently, [168]–[170] develop high-efficiency custom hardware implementations based on the FPGA system. Atay and Bayazit [168] focus on directly accelerating the PRM algorithm on FPGA by creating functional units to perform the random sampling, nearest neighbor search and parallelizing triangle-triangle testing. However, this design cannot be reconfigured at runtime, and the huge resource demands make it fail to support a large roadmap. Murray et al. [169] present a novel microarchitecture for an FPGA-based accelerator to speed up collision detection by creating a specialized circuit for each motion in the roadmap. This solution achieves sub-millisecond speed for motion planning query and improves the power consumption by more than one order of magnitude, which is sufficient to enable real-time robotics applications.

Besides real-time constraint, motion planning algorithms also have flexibility requirements to make the robots adapt to dynamic environments. Dadu-P [170] build a scalable motion planning accelerator to attain both high efficiency and flexibility, where a motion plan can be solved in around 300 microseconds in a dynamic environment. A hardware-friendly data structure representing roadmap edges is adopted to achieve flexibility, and a batched processing as well as a priority-rating method are proposed to achieve high efficiency. But this design comprises a 25× latency increase to make it retargetable to different robots and scenarios due to the external memory access. Murray et al. [164] develop a fully retargetable microarchitecture of collision detection and graph search accelerator that can perform motion planning in less than 3 ms with a modest power consumption of 35 W. This design divides the collision detection workflow into two stages. The collision detection results for the discretized roadmap are precomputed in the first stage before runtime, and then the collision detection accelerator streams in the voxels of obstacles and the edges of flags which are in collision at runtime.

### *D. Graph Search*

After collision detection, the planner will try to find the shortest and safe path from the start position to the target position based on the obtained collision-free roadmap through graph search. Several works explore graph search accelerations. Bondhugula et al. [171] employ a parallel FPGA-based design using a blocked algorithm to solve large instances of All-Pairs Shortest-Paths (APSP) problem, which achieves a 15× speedup over an optimized CPU-based implementation. Sridharan et al. [172] present an architecture-efficient solution based on Dijkstra's algorithm to accelerate the shortest path search, and Takei et al. [173] extend this for a high degree of parallelism and large-scale graph search. Recently, Murray et al. [164] accelerate graph search with the Bellman-Ford algorithm. By leveraging a precomputed roadmap and bounding specific robot quantities, this design enables a more compact and efficient storage structure, dataflows and a low-cost interconnection network.

## VI. Partial Reconfiguration

FPGA technology provides the flexibility of on-site programming and re-programming without going through re-fabrication with a modified design. Partial Reconfiguration (PR) takes this flexibility one step further, allowing the modification of an operating FPGA design by loading a partial configuration file, usually a partial BIT file [174]. Using PR, after a full BIT file configures the FPGA, partial BIT files can be downloaded to modify reconfigurable regions in the FPGA without compromising the integrity of the applications running on those parts of the device that are not being reconfigured.

A major performance bottleneck for PR is the configuration overhead, which seriously limits the usefulness of PR. To address this problem, in [175], the authors propose a combination of two techniques to minimize the overhead. First, the authors design and implement fully streaming DMA engines to saturate the configuration throughput. Second, the authors exploit a simple form of data redundancy to compress the configuration bitstreams, and implement an intelligent internal configuration access port (ICAP) controller to perform decompression at runtime. This design achieves an effective configuration data transfer throughput of up to 1.2 Gbytes/s, which actually well surpasses the theoretical

upper bound of the data transfer throughput, 400 Mbytes/s. Specifically, the proposed fully streaming DMA engines reduce the configuration time from the range of seconds to the range of milliseconds, a more than 1000-fold improvement. In addition, the proposed compression scheme achieves up to a 75% reduction in bitstream size and results in a decompression circuit with negligible hardware overhead.

Another problem of PR is that it may incur additional energy consumption. In [176], the authors investigate whether PR can be used to reduce FPGA energy consumption. The core idea is that there are a number of independent circuits within a hardware design, and some can be idle for long periods of time. Idle circuits still consume power though, especially through clock oscillation and static leakage. Using PR, one can replace these circuits during their idle time with others that consume much less power. Since the reconfiguration process itself introduces energy overhead, it is unclear whether this approach actually leads to an overall energy saving or to a loss. This study identifies the precise conditions under which partial reconfiguration reduces the total energy consumption, and proposes solutions to minimize the configuration energy overhead. In this study, PR is compared against clock gating to evaluate its effectiveness. The authors apply these techniques to an existing embedded microprocessor design, and successfully demonstrate that FPGAs can be used to accelerate application performance while also reducing overall energy consumption.

Further, PerceptIn demonstrate in their commercial product that Runtime partial reconfiguration (RPR) is useful for robotic computing, especially computing for autonomous vehicles, because many on-vehicle tasks usually have multiple versions where each is used in a particular scenario [16]. For instance, in PerceptIn's design, the localization algorithm relies on salient features; features in key frames are extracted by a feature extraction algorithm (based on ORB features [177]), whereas features in non-key frames are tracked from previous frames (using optical flow [178]); the latter executes in 10 ms, 50% faster than the former. Spatially sharing the FPGA is not only area-inefficient, but also power-inefficient as the unused portion of the FPGA consumes non-trivial static power. In order to temporally share the FPGA and "hot-swap" different algorithms, PerceptIn develop a partial reconfiguration engine (PRE) that dynamically reconfigures part of the FPGA at runtime. The PRE achieves a 400 MB/sec reconfiguration throughput (i.e., bitstream programming rate). Both the feature extraction and tracking bitstreams are less than 4 MB. Thus, the reconfiguration delay is less than 1 ms.

## VII. Commercial Applications of FPGAs in Autonomous Vehicles

Over the past three years, PerceptIn has built and commercialized autonomous vehicles for micromobility. Our products have been deployed in China, US, Japan and Switzerland. We summarize system design constraints, workloads and their performance characteristics from the real products. A custom computing system is developed by taking into account the inherent task-level parallisim, cost, safety and programmability [16], [179]. FPGA plays a critical role in our system, which synchronizes various sensors and accelerates the component on the critical path.

### *A. Computing system*

**Software pipeline.** Fig. 3 shows the block diagram of the processing pipeline in our vehicle, which consists of three parts: sensing, perception and planning. The sensing module bridges sensors and computing system. It synchronizes various sensor samples for the downstream perception module, which performs two fundamental tasks: 1) locating the vehicle itself in a global map and 2) understanding the surroundings through depth estimation and object detection. The planning module uses the perception results to devise a driveable route, and then converts the planed path into a sequence of control commands, which will drive the vehicle along the path. The control commands are sent to the vehicle's Engine Control Unit (ECU) via the CAN bus interface.

Sensing, perception and planning are serialized. They are all on the critical path of the end-to-end latency. We pipeline the three modules to improve the throughput. Within perception, localization and scene understanding are independent and could execute in parallel. While there are multiple tasks within scene understanding, they are mostly independent with the only exception that object tracking must be serialized with object detection. The task-level parallelisms influence how the tasks are mapped to the hardware platform.

**Algorithm.** Our localization module is based on Visual Inertial Odometry algorithms [180], [181], which fuses camera images, IMU and GPS samples to estimate the vehicle pose in the global map. The depth estimation employs traditional stereo vision algorithms, which calculates depths according to the principal of triangulation [182]. In particular, our method is based on the classic ELAS algorithm, which uses handcrafted features [183]. While DNN models for depth estimation exist, they are orders of magnitude more compute-intensive than non-DNN algorithms [184] while providing only marginal accuracy improvements to our use-cases. We detect objects using DNN models,

such as YOLO [24]. We use the Kernelized Correlation Filter (KCF) [185] to track detected objects. The planning algorithm is formulated as Model Predictive Control (MPC) [186].

**Hardware architecture.** Fig. 4 is the hardware system designed for our autonomous vehicles. The sensing hardware consists of stereo cameras, IMU and GPS. In particular, our system uses stereo cameras for depth estimation. One of the cameras is also used for semantic tasks such as object detection. The cameras along with the IMU and the GPS drive the VIO-based localization task.

Considering the cost, compute requirements and power budget, our computing platform is composed of a Xilinx Zynq Ultrascale+ FPGA and an on-vehicle PC equipped with an Intel Coffe Lake CPU and an Nvidia GTX 1060 GPU. The PC is the main computing platform, while the FPGA plays a critical role, which bridges sensors and the PC, and provides an acceleration platform. To optimize the end-to-end latency, explore the task level parallelism and ease practical development and deployment, planning and scene understanding are mapped onto the CPU and the GPU respectively, and sensing and localization are implemented on the FPGA platform.

### *B. Sensing on FPGA*

We map sensing on the Zynq FPGA platform. The FPGA processes sensor data and transfer sensor data to the PC for subsequent processing. The reason that sensing is mapped to FPGA is threefold. First, embedded FPGA platforms today are built with rich sensor interface (e.g. standard MIPI Camera Serial Interface) and sensor pre-processing hardware (e.g. ISP). Second, by having the FPGA directly process sensor data in situ, we allow accelerators on the FPGA to directly process sensor data without involving the power-hungry CPU for data movement and task coordination. Finally, processing sensor data on the FPGA naturally leads to a design of hardware-assisted multiple sensor synchronization mechanism.

**Sensor Synchronization** Sensor synchronization is critical to perception algorithms that fuse multiple sensors. Sensor fusion algorithms assume sensor samples have been well synchronized. For example, widely adopted datasets, such as KITTI, provide synchronized data so that researchers could focus on algorithmic development.

An ideal synchronization ensures that 1) various sensor samples have a unified timing system, and 2) timestamps of samples precisely record the time of events triggering the sensors. GPS synchronization is

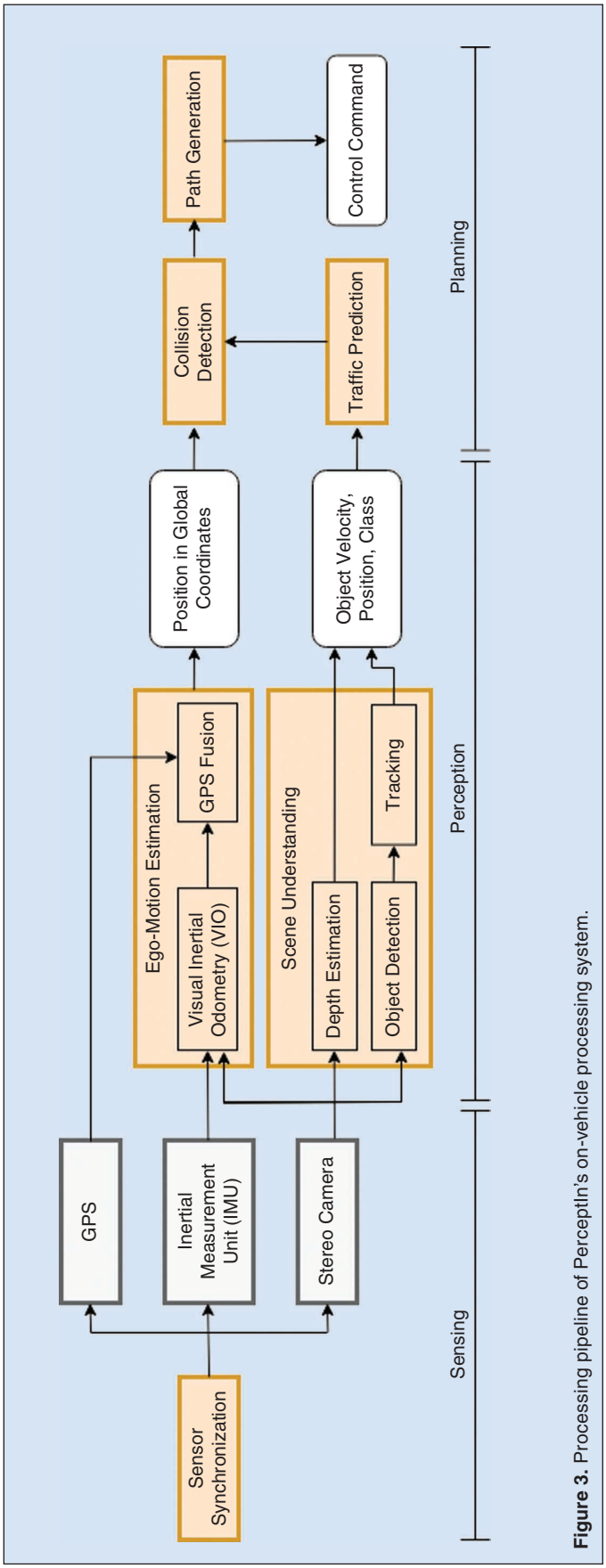



**Figure 3.** Processing pipeline of PerceptIn's on-vehicle processing system.

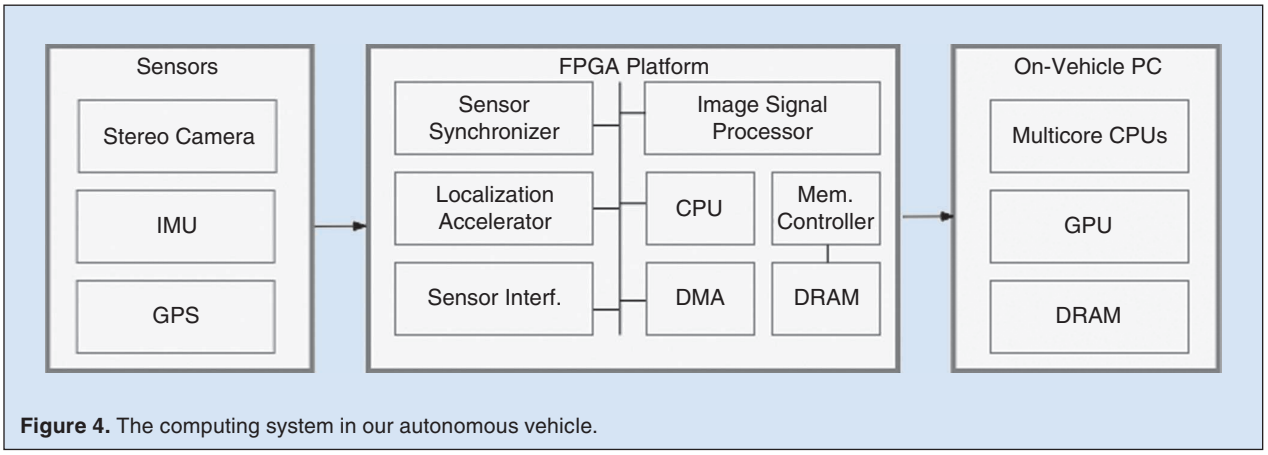



**Figure 4.** The computing system in our autonomous vehicle.

now wildly adopted to unify various measurements in a global timing domain. Software-based synchronization associates samples with timestamps at the application or the driver layer. This approach is inaccurate due to the software processing before the timestamp stage. The software processing introduces variable latency that is non-deterministic.

To obtain more precise synchronization, we uses a hardware synchronizer implemented by FPGA fabrics. The hardware synchronizer triggers the camera sensors and the IMU using a common timer initialized by the satellite atomic time provided by the GPS device. It records the triggering time of each sensor sample, and then pack the timestamp with the corresponding sensor data. In terms of costs, the synchronizer is extremely lightweight in design with only 1,443 LUTs and 1,587 registers and consumes 5mW of power.

### *C. Perception on FPGA*

For our autonomous vehicles, the perception tasks includes scene understanding (depth estimation and objection detection) and localization, which are independent. The slower one dictates the overall perception latency.

We evaluate our perception algorithms on the CPU, GPU and Zynq FPGA platform. Fig. 5 compares the latency of each perception tasks on the FPGA platform with the GPU. Due to the available resources, the FPGA platform is faster than the GPU only for localization, which is more lightweight than other tasks. We offload localization to the FPGA while leaving other perception task on the GPU. This partitioning frees more GPU resources for depth estimation and object detection, which is benefit for reducing the perception pipeline's latency.

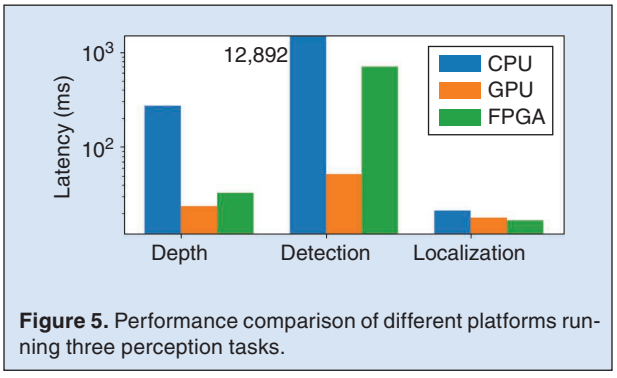



**Figure 5.** Performance comparison of different platforms running three perception tasks.

As with classic SLAM algorithms, our localization algorithm consists of a front-end and a back-end. The front-end uses the ORB features and descriptors for detecting and tracking key points [120], [187]. The back-end uses Levenberg-Marquardt's (LM) algorithm, a non-linear optimization algorithm, to optimize the position of 3 D key points and the pose of the camera [156], [188].

The ORB feature extraction/matching and the LM optimizer are the most time-consuming parts of our SLAM algorithm, which take up nearly all the execution time. We accelerate ORB feature extraction/matching and the non-linear optimizer on FPGA fabrics. The rest lightweight parts are implemented on the ARM core of the Zyqn platform. We use independent hardware for each camera to extract features and compute descriptors. Hamming distance and Sum of Absoluated Difference (SAD) matching are implemented to obtain stable matching results. Compared with the CPU implementation, our FPGA implementation achieves a 2.2× speedup and 44 fps, and saves 83% energy.

We use LM algorithm to optimize features and poses over a fixe-size sliding window. To solve the non-linear optimization problem, the LM algorithm iteratively use Jacobbian to linearize the problem and solve the linear equation at each iteration. Schur elimination is used to reduce the dimension of the linear equation, thus reduce the complexity of solving the equation. Cholesky factorization is employed to solve the linear equation. For sliding-window based vSLAM, the Jacobian

and Schur elimination are the most time-consuming parts. By profiling our algorithm on datasets [189], Schur and Jacobian computations account for 29.8% and 48.27% of total time. We implemented Schur elimination and Jacobian updates on FGPA fabrics [156]. Compared with the CPU implementation, the FPGA achieves 4× and 27× speedup for Schur and Jacobbian, and saves 76% energy.

## VIII. Application of FPGAs in Space Robotics

In the 1980s, field-programmable gate arrays (FPGA) emerged as a result of increasing integration in electronics. Before the use of FPGA, glue-logic designs were based on individual boards with fixed components interconnected via a shared standard bus, which has various drawbacks, such as hindrance of high volume data processing and higher susceptibility to radiation-induced errors, in addition to inflexibility. The utilization of FPGAs in space applications began in 1992, for FPGAs offered unprecedented flexibility and significantly reduced the design cycle and development cost [190].

FPGAs can be categorized by the type of their programmable interconnection switches: antifuse, SRAM, and Flash. Each of the three technologies comes with trade-offs. Antifuse FPGAs are non-volatile and have minimal delay due to routing, resulting in a faster speed and lower power consumption. The drawback is evident as they have a relatively more complicated fabrication process and are only one time programmable. SRAM-based FPGAs are the most common type employed in space missions. They are field reprogrammable and use the standard fabrication process that foundries put in significant effort in optimizing, resulting in a faster rate of performance increase. However, based on SRAM, these FPGAs are volatile and may not hold configuration if a power glitch occurs. Also, they have more substantial routing delay, require more power, and have a higher susceptibility to bit errors. Flash-based FPGAs are non-volatile and reprogrammable, and also have low power consumption and route delay. The major drawback is that in-flight reconfiguration is not recommended for flash-based FPGAs due to the potentially destructive results if radiation effects occur during the reconfiguration process [191]. Also, the stability of stored charge on the floating gate is of concern: it is a function including factors such as operating temperature, the electric fields that might disturb the charge. As a result, flash-based FPGAs are not as frequently used in space missions [192].

### *A. Radiation Tolerance for Space Computing*

For electronics intended to operate in space, the harsh space radiation present is an essential factor to consider. Radiation has various effects on electronics, but the commonly focused two are total ionizing dose effect (TID) and single event effects (SEE). TID results from the accumulation of ionizing radiation over time, which causes permanent damage by creating electron-hole pairs in the silicon dioxide layers of MOS devices. The effect of TID is that electronics gradually degrade in their performance parameters and eventually fail to function. Electronics intended for application in space are tested for the total amount of radiation, measured in kRads, they can endure before failure. Usually, electronics that can withstand 100 kRads are sufficient for low earth orbit missions to use for several years [191].

SEE occurs when high-energy particles from space radiation strike electronics and leave behind an ionized trail. The results are various types of SEEs [193], which can be categorized as either soft errors, which usually do not cause permanent damage, or hard errors, which often cause permanent damage. Examples of soft error include single event upset (SEU), and single event transient (SET). In SEU, a radiation particle struck a memory element, causing a bit flip. Noteworthy is that as the cell density and clock rate of modern devices increases, multiple cell upset (MCU), corruption of two or more memory cells in a single particle strike, is increasingly becoming a concern. A special type of SEU is single event functional interrupt (SEFI), where the upset leads to loss of normal function of the device by affecting control registers or the clock. In SET, a radiation particle passes through a sensitive node, which generates a transient voltage pulse, causing wrong logic state at the combinatorial logic output. Depending on whether the impact occurs during an active clock edge or not, the error may or may not propagate. Some examples of hard error include single event latch-up (SEL), in which energized particle activates parasitic transistor and then cause a short across the device, and single event burnout (SEB), in which radiation induces high local power dissipation, leading to device failure. In these hard error cases, radiation effects may cause the failure of an entire space mission.

Space-grade FPGAs can withstand considerable levels of TID and have been designed against most destructive SEEs [194]. However, SEU susceptibility is pervasive. For the most part, radiation effects on FPGA are not different from those of other CMOS based ICs. The primary anomaly stems from FPGAs' unique structure, involving programmable interconnections. Depending on their type, FPGAs have different susceptibility toward SEU in their configuration. SRAM FPGAs are designated by NASA as the most susceptible ones due to their volatile nature. Even after the radiation hardening

process, the configuration of SRAM FPGAs is only designated as "hardened" or simply having embedded SEE mitigation techniques rather than "hard,"which means close to immune [191]. Configuration SRAM is not used in the same way as the traditional SRAM. A bit flip in configuration causes an instantaneous effect without the need for a read-write cycle. Moreover, instead of producing one single error in the output, the bit flip shifts the user logic directly, changing the device's behavior. Scrubbing is needed to rectify SRAM configuration. Antifuse and flash FPGAs are less susceptible to effects in configuration and are designated "hard" against SEEs in their configuration without applying radiation hardening techniques [191].

Design based SEU/fault mitigation techniques are commonly used, for, in contrast to fabrication level radiation hardening techniques, they can be readily applied to commercial off the shelf (COTS) FPGAs. These techniques can be classified into static and dynamic. Static techniques rely on fault-masking, toleration of error without requiring active fixing. One such example is passive redundancy with voting mechanisms. Dynamic techniques, in contrast, detect faults and act to correct them. The common SEU Mitigation Methods include [195], [196]:

1) Hardware Redundancy: functional blocks are replicated to detect/tolerate faults. Triple modular redundancy (TMR) is perhaps the most widely used mitigation technique. It can be applied to entire processors or parts of circuits. At a circuit level, registers are implemented using three or more flip flops or latches. Then, voters compare the values and output the majority, reducing the likelihood of error due to SEU. As internal voters are also susceptible to SEU, they are sometimes triplicated also. For mission-critical applications, global signals may be triplicated to mitigate SEUs further. TMR can be implemented at ease with the help supporting HDLs [197]. It is important to note that a limitation of TMR is that one fault, at most, can be tolerated per voter stage. As a result, TMR is often used with other techniques, such as scrubbing, to prevent error accumulation.
2) Scrubbing: The vast majority of memory cells in reprogrammable FPGAs contain configuration information. As discussed earlier, configuration memory upset may lead to alteration routing network, loss of function, and other critical effects. Scrubbing, refreshing and restoration of configuration memory to a known-good state, is therefore needed [196]. The reference configuration memory is usually stored in radiation-hardened memory cells either off or on the device. Scrubbers, processors or configuration controllers, carry out scrubbing. Some advanced SRAM FPGAs, including ones made by Xilinx, support partial reconfiguration, which allows memory repairs to be made without interrupting the operation of the whole device. Scrubbing can be done in frame-level (partial) or device-level (full), which will inevitably lead to some downtime; some devices may not be able to tolerate such an interruption. Blind scrubbing is the most straightforward way of implementation: individual frames are scrubbed periodically without error detection. Blind scrubbing avoids the complexity required in error detection, but extra scrubbing may increase vulnerability to SEUs as errors may be written into frames during the scrubbing process. An alternative to blind scrubbing is readback scrubbing, where scrubbers actively detect errors in configuration through error-correcting code or cyclic redundancy check [195]. If an error is found, scrubber initiates frame-level scrubbing.

Currently, the majority of space-grade FPGA comes from Xilinx and Microsemi. Xilinx offers the Virtex family and Kintex. Both are SRAM based, which have high flexibility. Microsemi offers antifuse based RTAX and Flash-based RTG4, RT PolarFire, which have lower susceptibility against SEE and power consumption. 20 nm Kintex and 28 nm RT PolarFire are the latest generations. The European market is offered with Atmel devices and NanoXplore space-grade FPGAs [198]. Table IV shows the specifications of the above devices.

### *B. FPGAs in Space Missions*

For space robotics, processing power is of particular importance, given the range of information required to accurately and efficiently process. Many of the current and previous space missions are packed with sophisticated algorithms that are mostly static. They serve to increase the efficiency of data transmission; nevertheless, data processing is done mainly on the ground. As the travel distance of missions increases, transmitting all data to, and processing it on the ground is no longer an efficient or even viable option due to transmission delay. As a result, space robots need to become more adaptable and autonomous. They will also need to preprocess on-board a large amount of data collected and compress it before sending it back to Earth [199].

The rapid development of new generation FPGAs may fill the need in space robotics. FPGAs enable robotic systems to be reconfigurable in real-time, making the systems more adaptable by allowing them to respond more efficiently to changes in environment and data. As a result, autonomous reconfiguration and performance

optimization can be achieved. Also, the FPGAs have a high capability for parallel processing, which is useful in boosting processing performance. The use of FPGA is present in various space robots. Some of the most prominent examples of the application are the NASA Mars rovers. Since the first pair of rovers were launched in 2003, the presence of FPGAs have steadily increased in the later rovers.

**1) Mars Exploration Rover Missions**

Beginning in the early 2000s, NASA have been using FPGAs in exploration rover control and lander control. In Opportunity and Spirit, the two Mars rovers launched in 2003, two Xilinx Virtex XQVR1000s were in the motor control board [200], which operates motors on instruments as well as rover wheels. In addition, an Actel RT 1280 FPGA was used in each of the 20 cameras on the rovers to receive and dispatch hardware commands. The camera electronics consist of clock driver that provides timing pulses through the charge-coupled device (CCD), an IC containing an array of linked or coupled capacitors. Also, there are signal chains that amplify the CCD output and convert it from analog to digital. The Actel FPGA provides the timing, logic, and control functions in the CCD signal chain and inserts a camera ID into camera telemetry to simplify processing [201].

Selected electronic parts have to undergo a multi-step flight consideration process before utilized in any space exploration mission [200], [202]. The first step is the general flight approval, during which the manufacturers perform additional space-grade verification tests beyond the normal commercial evaluation, and NASA meticulously examines the results. Additional device parameters, such as temperature considerations and semiconductor characteristics are verified in these tests. What follows is flight-specific approval. In this step, NASA engineers examine the device compatibility with the mission. For instance, considerations of the operating environment including factors like temperature and radiation. Also included are a variety of mission-specific situations that the robot may encounter and the associated risk assessment. Depending on the specific application of the device, whether mission critical or not, and the expected mission lifetime, the risk standards varies. Finally, parts go through specific design consideration to ensure all the design requirements have been met. Parts are examined for their designs addressing issues such as SEL, SEU, SEFI. The Xilinx FPGAs used addressed some of the SEE through the following methods [201]:

1) Fabrication processes largely prevents SEL
2) TMR reduces SEU frequency
3) Scrubbing allows device recovery from single event functional interrupts

MER went successful and despite being designed for only 90 Martian days (1 Martian day = 24.6 hours), continued until 2019. The implementation of mitigation techniques was also proven to be effective as the observed error rate was very similar to that predicted [200].

**2) Mars Science Laboratory Mission**

Launched in 2011, Mars Science Lab (MSL) was the new Rover sent on to Mars. FPGAs were heavily used in its key components, mainly responsible for scientific instrument control, image processing, and communications.

Curiosity has 17 cameras on board: four navigation cameras, eight hazard cameras, the Mars Hand Lens

**Table IV.**
**Specifications of Space-Grade FPGAs.**

| Device | Logic | Memory | DSPs | Technology | Rad. Tolerance |
|---|---|---|---|---|---|
| Xilinx Virtex-5QV | 81.9 K LUT6 | 12.3 Mb | 320 | 65 nm SRAM | SEE immune up to LET > 100 MeV/(mg · cm$^2$) and 1 Mrad TID |
| Xilinx RT Kintex UltraScale | 331 K LUT6 | 38 Mb | 2760 | 20 nm SRAM | SEE immune up to LET > 80 MeV/(mg · cm$^2$) and 100-120 Krads TID |
| Microsemi RTG4 | 150 K LE | 5 Mb | 462 | 65 nm Flash | SEE immune up to LET > 37 MeV(mg · cm$^2$) and TID > 100 Krads |
| Microsemi RT PolarFire | 481 K LE | 33 Mb | 1480 | 28 nm Flash | SEE immune up to LET > 63 MeV(mg · cm$^2$) and 300 Krads |
| Microsemi RTAX | 4 M gates | 0.5 Mb | 120 | 150 nm antifuse | SEE immune up to LET > 37 MeV(mg · cm$^2$) and 300 Krads TID |
| Atmel ATFEE560 | 560 K gates | 0.23 Mb | – | 180 nm SRAM | SEL immune up to 95 MeV(mg · cm$^2$) and 60 Krads TID |
| NanoXplore NG-LARGE | 137 K LUT4 | 9.2 Mb | 384 | 65 nm SRAM | SEL immune up to 60 MeV(mg · cm$^2$) and 100 Krads TID |

Imager (MAHLI), two Mast Cameras, the Mars Descent Imager (MARDI), and the ChemCam Remote Microscopic Imager [203]. MAHLI, the mast cameras, and MARDI share the same electronics design. Similar to the system used on MER, an Actel FPGA provides the timing, logic, and control functions in the CCD signal chain and transmits pixels to the digital electronics assembly (DEA), which interfaces the camera heads with the rover electronics, transmitting command to the camera heads and data back to the rover. There is one DEA dedicated to each of the imagers above. Each is has a Virtex-II FPGA that contains a Microblaze soft-processor core. All of the core functionalities of the DEA, including timing, interface, and compression, are implemented in the FPGA as logic peripherals of the Microblaze. Specifically, the DEA provides an image processing pipeline that includes 12 to 8-bit commanding of input pixels, horizontal subframing, and lossless or JPEG image compression [203]. What runs on the Microblaze is the DEA flight software, which coordinates DEA hardware functions such as camera movements. It receives and executes commands, and transmits command from the Earth. The flight software also implements image acquisition algorithms, including autofocus and autoexposure, performs error correction of flash memory, and mechanism control fault protection [203]. In total, the flight software consists of 10,000 lines of ANSI C code, all implemented on the FPGA. Additionally, FPGAs power communication boxes (Electra-Lite) to provide critical communication to Earth from the rovers through a Mars relay network [204]. They are responsible for a variety of high speed bulk signal processing.

#### 3) Mars 2020 Mission

Perseverance is NASA's latest launched Mars rover. The presence of FPGA continued and increased. FPGA was used in the autonomous driving system as a coprocessor for algorithm acceleration for the first time in NASA's planetary rovers. Perseverance runs on the GESTALT (grid-based estimation of surface traversability applied to local terrain) AutoNav algorithm same as Curiosity [205]. Added was the FPGA based accelerator, called Vision Compute Element (VCE). During landing, VCE serves to provide sufficient computing power for the Lander Vision System (LVS), which performs an intensive task of estimates the landing location in 10 seconds by fusing data from the designed landing location, IMU, and landmark matches. After landing, the connection between VCE and LVS is severed. Instead, VCE is repurposed for the GESTALT driving algorithm. The VCE has three cards plugged into a PCI backplane: a CPU card with BAE RAD750 processor, a Compute Element Power Conditioning Unit (CEPCU), and a Computer Vision Acceleration Card (CVAC). While the former two parts were inherited from the MLS mission, the CVAC is new. It has two FPGAs. One is called the Vision Processor–a Xilinx Virtex 5QV that contains image processing modules for matching landmarks to estimate position. The other is called the Housekeeping FPGA–a Microsemi RTAX 2000 antifuse FPGA that handles tasks such as synchronization with the spacecraft, power management, Vision Processor configuration.

Through more than two decades of use in space, FPGAs have shown their reliability and applicability for space robotic missions. The properties of FPGAs make them good onboard processors, ones that have high reliability, adaptability, processing power, and power efficiency: FPGAs have been used for space robotic missions for decades and are proven in reliability; they have unrivaled adaptability and can even be reconfigured in run time; their capability for high degree parallel processing allow significant acceleration in executing many complex algorithms; hardware/software co-design method makes them potentially more power-efficient. They may finally help us close the two-decade performance gap between commercial processors and space-grade ASICs. As a direct result, the achievements that the world has made in fields such as deep learning and computer vision, which were often too computationally intense for space-grade processors to be used, may become applicable for robots in space in the near future. The implementation of those new technologies will be of great benefit for space robots, boosting their autonomy and capabilities and allowing us to explore farther and faster.

## IX. Conclusion

In this paper, we review the state-of-the-art FPGA-based robotic computing accelerator designs and summarize their adopted optimized techniques. According to the results shown in Section III, IV and V, by co-designing both the software and hardware, FPGA can achieve more than 10× better performance and energy efficiency compared to the CPU and GPU implementations. We also review the partial reconfiguration methodology in FPGA implementation to further improve the design flexibility and reduce the overhead. Finally, by presenting some recent FPGA-based robotics applications in commercial and space areas, we demonstrate that FPGA has excellent potential and is a promising candidate for robotic computing acceleration due to its high reliability, adaptability and power efficiency.

The authors believe that FPGAs are the best compute substrate for robotic applications for several reasons: first, robotic algorithms are still evolving rapidly,

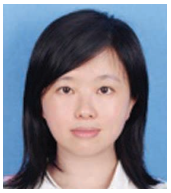
and thus any ASIC-based accelerators will be months or even years behind the state-of-the-art algorithms; on the other hand, FPGAs can be dynamically updated as needed. Second, robotic workloads are highly diverse, thus it is difficult for any ASIC-based robotic computing accelerator to reach economies of scale in the near future; on the other hand, FPGAs are a cost-effective and energy-effective alternative before one type of accelerator reaches economies of scale. Third, compared to SoCs that have reached economies of scale, e.g., mobile SoCs, FPGAs deliver a significant performance advantage. Fourth, partial reconfiguration allows multiple robotic workloads to time-share an FPGA, thus allowing one chip to serve multiple applications, leading to overall cost and energy reduction.

However, FPGAs are still not the mainstream computing substrate for robotic workloads, for several reasons: first, FPGA programming is still much more challenging than regular software programming, and the supply of FPGA engineers is still limited. Second, although there is significant progress in the past few years in the FPGA High-Level Synthesis (HLS) automation, such as [206], HLS is still not able to produce optimized code, and IP supports for robotic workloads are still extremely limited. Third, commercial software support for robotic workloads on FPGAs is still missing. For instance, there is no official ROS support on any commercial FPGA platform today. For robotic companies to fully exploit the power of FPGAs, these problems need to be first addressed, and the authors use these problems to motivate our future research work.

**Zishen Wan** (Student Member, IEEE) is currently a Ph.D. student in Electrical and Computer Engineering at Georgia Institute of Technology, Atlanta, GA, U.S.A. He received the M.S. degree from Harvard University, Cambridge, MA, in 2020, and the B.S. degree from Harbin Institute of Technology, Harbin, China, in 2018, both in Electrical Engineering. He has a broad research interest in VLSI design, computer architecture, and edge intelligence, with a focus on energy-efficient and robust hardware and system design for autonomous machines. He has received the Best Paper Award in DAC 2020 and CAL 2020.

**Bo Yu** (Senior Member, IEEE) received the B.S. degree in electronic technology and science from Tianjin University, Tianjin, China, in 2006, and the Ph.D. degree from the Institute of Microelectronics, Tsinghua University, Beijing, China, in 2013. He is currently the CTO of PerceptIn, Fremont, CA, U.S.A., a company focusing on providing visual perception solutions for robotics and autonomous driving. His current research interests include algorithm and systems for robotics and autonomous vehicles. Dr. Yu is also a Founding Member of the IEEE Special Technical Community on Autonomous Driving.

**Thomas Yuang Li** (Student Member, IEEE) is a research intern at PerceptIn, U.S.A. and a student member of the IEEE. His research interests include building autonomous space explorers for future commercial robotic space-exploration missions as well as space robotics and computing related topics.



**Jie Tang** (Senior Member, IEEE) is currently an associate professor in School of Computer Science and Engineering of South China University of Technology, Guangzhou, China. She received her B.E. degree From University of Defense Technology and Ph.D. degree from the Beijing Institute of Technology, both in Computer Science. She was previously a visiting researcher at the Embedded Systems Center at University of California, Irvine, USA, and a research scientist at Intel China Runtime Technology Lab. Dr. Tang is mainly doing research on Computing Systems for Autonomous Machines. She is a founding member and secretary of the IEEE Computer Society Special Technical Community on Autonomous Driving Technologies.

**Yuhao Zhu** (Member, IEEE) is an Assistant Professor of Computer Science at University of Rochester, U.S.A. His research group focuses on applications and computer systems for visual computing. His work is recognized by the Honorable Mention of the 2018 ACM SIGARCH/IEEE-CS TCCA Outstanding Dissertation Award and multiple IEEE Micro Top Picks designations. He is a recipient of the NSF CAREER Award in 2020.

**Yu Wang** (Senior Member, IEEE) received his B.S. degree in 2002 and Ph.D. degree (with honor) in 2007 from Tsinghua University, Beijing, China. He is currently a Tenured Professor and Chair with the Department of Electronic Engineering, Tsinghua University. His research interests include application specific hardware computing, parallel circuit analysis, and power/reliability aware system

design methodology. Dr. Wang has authored and coauthored over 250 papers in refereed journals and conferences. He has received Best Paper Award in ASPDAC 2019, FPGA 2017, NVMSA17, ISVLSI 2012, and Best Poster Award in HEART 2012 with 9 Best Paper Nominations. He is a recipient of DAC Under-40 Innovator Award in 2018. He served as TPC chair for ICFPT 2019, ISVLSI 2018, ICFPT 2011 and Finance Chair of ISLPED 2012-2016, and served as the program committee member for leading conferences in EDA/FPGA area. Currently he serves as Associate Editor for IEEE Trans on CAS for Video Technology, IEEE Transactions on CAD, and ACM TECS. He is an IEEE/ACM senior member. He is the co-founder of Deephi Tech (acquired by Xilinx in 2018), which is a leading deep learning computing platform provider

**Arijit Raychowdhury** (Senior Member, IEEE) is currently a Professor in the School of Electrical and Computer Engineering at the Georgia Institute of Technology, U.S.A., where he joined in January 2013. From 2013 to July 2019 he was an Associate Professor and held the ON Semiconductor Junior Professorship in the department. He received his Ph.D. degree in Electrical and Computer Engineering from Purdue University (2007) and his B.E. in Electrical and Telecommunication Engineering from Jadavpur University, India (2001). His industry experience includes five years as a Staff Scientist in the Circuits Research Lab, Intel Corporation, and a year as an Analog Circuit Researcher with Texas Instruments Inc. His research interests include low power digital and mixed-signal circuit design, design of power converters, sensors and exploring interactions of circuits with device technologies. Dr. Raychowdhury holds more than 25 U.S. and international patents and has published over 200 articles in journals and refereed conferences. He currently serves on the Technical Program Committees of ISSCC, VLSI Circuit Symposium, CICC, and DAC. He was the Associate Editor of the IEEE Transactions on Computer Aided Design from 2013-2018 and the Editor of the Microelectronics Journal, Elsevier Press from 2013 to 2017. He is the winner of Qualcomm Faculty Award, 2020; IEEE/ACM Innovator under 40 award; the NSF CISE Research Initiation Initiative Award (CRII), 2015; Intel Labs Technical Contribution Award, 2011; Dimitris N. Chorafas Award for outstanding doctoral research, 2007; the Best Thesis Award, College of Engineering, Purdue University, 2007; SRC Technical Excellence Award, 2005; Intel Foundation Fellowship, 2006; NASA INAC Fellowship, 2004; the Meissner Fellowship 2002. He and his students have won several fellowships and eleven best paper awards over the years. Dr. Raychowdhury is a Senior Member of the IEEE.

**Shaoshan Liu** (Senior Member, IEEE) is Founder and CEO of PerceptIn (www.perceptin.io) U.S.A., a company focusing on providing visual perception solutions for autonomous robots and vehicles. Dr. Shaoshan Liu received his Ph.D. in Computer Engineering from University of California, Irvine and M.P.A. from Harvard University. His research focuses on Computing Systems for Autonomous Machines. Dr. Shaoshan Liu has published over 80 research papers and holds over 150 U.S. international patents on autonomous machines. Dr. Shaoshan Liu is an ACM Distinguished Speaker, and an IEEE Computer Society Distinguished Speaker.

## References


[1] A. Qiantori, A. B. Sutiono, H. Hariyanto, H. Suwa, and T. Ohta, "An emergency medical communications system by low altitude platform at the early stages of a natural disaster in Indonesia," *J. Med. Syst.*, vol. 36, no. 1, pp. 41–52, 2012. doi: 10.1007/s10916-010-9444-9.

[2] A. Ryan and J. K. Hedrick, "A mode-switching path planner for uav-assisted search and rescue," in *Proc. 44th IEEE Conf. Decision and Control*, 2005, pp. 1471–1476.

[3] N. Smolyanskiy, A. Kamenev, J. Smith, and S. Birchfield, "Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots and Syst. (IROS)*, 2017, pp. 4241–4247.

[4] A. Giusti et al., "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robot. Automat. Lett.*, vol. 1, no. 2, pp. 661–667, 2015. doi: 10.1109/LRA.2015.2509024.

[5] J. K. Stolaroff, C. Samaras, E. R. O'Neill, A. Lubers, A. S. Mitchell, and D. Ceperley, "Energy use and life cycle greenhouse gas emissions of drones for commercial package delivery," *Nature Commun.*, vol. 9, no. 1, pp. 1–13, 2018. doi: 10.1038/s41467-017-02411-5.

[6] S. J. Kim, Y. Jeong, S. Park, K. Ryu, and G. Oh, "A survey of drone use for entertainment and AVR (augmented and virtual reality)," in *Augmented Reality and Virtual Reality*. Springer-Verlag, 2018, pp. 339–352.

[7] S. Jung, S. Cho, D. Lee, H. Lee, and D. H. Shim, "A direct visual serving-based framework for the 2016 IROS autonomous drone racing challenge," *J. Field Robot.*, vol. 35, no. 1, pp. 146–166, 2018. doi: 10.1002/rob.21743.

[8] "Fact sheet—The Federal Aviation Administration (FAA) aerospace forecast fiscal years (FY) 2020–2040," 2020. https://www.faa.gov/news/fact_sheets/news_story.cfm?newsId=24756

[9] S. Liu, L. Li, J. Tang, S. Wu, and J.-L. Gaudiot, "Creating autonomous vehicle systems," *Synthesis Lectures Comput. Sci.*, vol. 6, no. 1, pp. 1–186, 2017. doi: 10.2200/S00787ED1V01Y201707CSL009.

[10] S. Krishnan et al., "The sky is not the limit: A visual performance model for cyber-physical co-design in autonomous machines," *IEEE Comput. Arch. Lett.*, vol. 19, no. 1, pp. 38–42, 2020. doi: 10.1109/LCA.2020.2981022.

[11] S. Krishnan et al., "Machine learning-based automated design space exploration for autonomous aerial robots," 2021, arXiv:2102.02988.

[12] S. Liu and J.-L. Gaudiot, "Autonomous vehicles lite self-driving technologies should start small, go slow," *IEEE Spectr.*, vol. 57, no. 3, pp. 36–49, 2020. doi: 10.1109/MSPEC.2020.9014458.

[13] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," *Proc. IEEE*, vol. 107, no. 8, pp. 1697–1716, 2019. doi: 10.1109/JPROC.2019.2915983.

[14] S. Liu, J. Tang, Z. Zhang, and J.-L. Gaudiot, "Computer architectures for autonomous driving," *Computer*, vol. 50, no. 8, pp. 18–25, 2017. doi: 10.1109/MC.2017.3001256.

[15] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "[DL] a survey of FPGA-based neural network inference accelerators," *ACM Trans. Reconfigurable Technol. Syst. (TRETS)*, vol. 12, no. 1, pp. 1–26, 2019. doi: 10.1145/3289185.
[16] B. Yu, W. Hu, L. Xu, J. Tang, S. Liu, and Y. Zhu, "Building the computing system for autonomous micromobility vehicles: Design constraints and architectural optimizations," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarch. (MICRO)*, 2020. doi: 10.1109/MICRO50266.2020.00089.
[17] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recogn. (CVPR'05)*, 2005, vol. 1, pp. 886–893.
[18] X He, R. S. Zemel, and M. A. Carreira-Perpinan, "Multiscale conditional random fields for image labeling," in *Proc. IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recogn. (CVPR 2004*), 2004, vol. 2, p. II.
[19] X. He, R. S. Zemel, and D. Ray, "Learning and incorporating top-down cues in image segmentation," in *Proc. Comput. Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 338–351.
[20] Y. Xiang, A. Alahi, and S. Savarese, "Learning to track: Online multi-object tracking by decision making," in *Proc. IEEE Int. Conf. Comput. Vision (ICCV)*, 2015, pp. 4705–4713. doi: 10.1109/ICCV.2015.534.
[21] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vision (ICCV)*, Dec. 2015. doi: 10.1109/ICCV.2015.169.
[22] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.
[23] W. Liu et al., "SSD: Single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015.
[24] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015.
[25] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016.
[26] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1411.4038, 2014.
[27] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *CoRR*, vol. abs/1406.4729, 2014.
[28] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," *CoRR*, vol. abs/1612.01105, 2016.
[29] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. S. Torr, "Fully-convolutional Siamese networks for object tracking," *CoRR*, vol. abs/1606.09549, 2016.
[30] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part I," *IEEE Robot. Automat. Mag.*, vol. 13, no. 2, pp. 99–110, 2006. doi: 10.1109/MRA.2006.1638022.
[31] M. Montemerlo et al., "Junior: The Stanford entry in the urban challenge," *J. Field Robot.*, vol. 25, no. 9, pp. 569–597, 2008. doi: 10.1002/rob.20258.
[32] J. Ziegler et al., "Making bertha drive—an autonomous journey on a historic route," *IEEE Intell. Transp. Syst. Mag.*, vol. 6, no. 2, pp. 8–20, 2014. doi: 10.1109/MITS.2014.2306552.
[33] C. Katrakazas, M. Quddus, W.-H. Chen, and L. Deka, "Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions," *Transp. Res. C, Emerg. Technol.*, vol. 60, pp. 416–442, 2015. doi: 10.1016/j.trc.2015.09.011.
[34] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Trans. Intell. Veh.*, vol. 1, no. 1, pp. 33–55, 2016. doi: 10.1109/TIV.2016.2578706.
[35] Y. Deng, Y. Chen, Y. Zhang, and S. Mahadevan, "Fuzzy dijkstra algorithm for shortest path problem under uncertain environment," *Appl. Soft Comput.*, vol. 12, no. 3, pp. 1231–1237, 2012. doi: 10.1016/j.asoc.2011.11.011.
[36] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, 1968. doi: 10.1109/TSSC.1968.300136.
[37] S. M. LaValle and J. J. Kuffner Jr., "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, 2001. doi: 10.1177/02783640122067453.
[38] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom. (1989–June 2004)*, vol. 12, no. 4, pp. 566–580, 1996. doi: 10.1109/70.508439.
[39] S. Shalev-Shwartz, N. Ben-Zrihem, A. Cohen, and A. Shashua, "Long-term planning by short-term prediction," 2016, arXiv:1602.01580.
[40] M. Gómez, R. González, T. Martínez-Marín, D. Meziat, and S. Sánchez, "Optimal motion planning by reinforcement learning in autonomous mobile vehicles," *Robotica*, vol. 30, no. 2, pp. 159, 2012. doi: 10.1017/S0263574711000452.
[41] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," 2016, arXiv:1610.03295.
[42] M. Bojarski et al., "End to end learning for self-driving cars," 2016, arXiv:1604.07316.
[43] X. Geng, H. Liang, B. Yu, P. Zhao, L. He, and R. Huang, "A scenario-adaptive driving behavior prediction approach to urban autonomous driving," *Appl. Sci.*, vol. 7, no. 4, p. 426, 2017. doi: 10.3390/app7040426.
[44] C. J. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3-4, pp. 279–292, 1992. doi: 10.1023/A:1022676722315.
[45] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances Neural Inf. Process. Syst.*, 2000, pp. 1008–1014.
[46] S. L. Hicks, I. Wilson, L. Muhammed, J. Worsfold, S. M. Downes, and C. Kennard, "A depth-based head-mounted visual display to aid navigation in partially sighted individuals," *PloS One*, vol. 8, no. 7, p. e67695, 2013. doi: 10.1371/journal.pone.0067695.
[47] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, "Elasticfusion: Real-time dense slam and light source estimation," *Int. J. Robot. Res.*, vol. 35, no. 14, pp. 1697–1716, 2016. doi: 10.1177/0278364916669237.
[48] V. A. Prisacariu et al., "Infinitam v3: A framework for large-scale 3d reconstruction with loop closure," 2017, arXiv:1708.00783.
[49] S. Golodetz, T. Cavallari, N. A. Lord, V. A. Prisacariu, D. W. Murray, and P. H. Torr, "Collaborative large-scale dense 3d reconstruction with online inter-agent pose optimisation," *IEEE Trans. Vis. Comput. Graphics*, vol. 24, no. 11, pp. 2895–2905, 2018. doi: 10.1109/TVCG.2018.2868533.
[50] M. Pérez-Patricio and A. Aguilar-González, "FPGA implementation of an efficient similarity-based adaptive window algorithm for real-time stereo matching," *J. Real-Time Image Process.*, vol. 16, no. 2, pp. 271–287, 2019. doi: 10.1007/s11554-015-0530-6.
[51] D.-W. Yang, L.-C. Chu, C.-W. Chen, J. Wang, and M.-D. Shieh, "Depth-reliability-based stereo-matching algorithm and its VLSI architecture design," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 6, pp. 1038–1050, 2014. doi: 10.1109/TCSVT.2014.2361419.
[52] A. Aguilar-González and M. Arias-Estrada, "An FPGA stereo matching processor based on the sum of hamming distances," in *Proc. Int. Symp. Appl. Reconfigurable Comput.*, 2016, pp. 66–77.
[53] M. Pérez-Patricio, A. Aguilar-González, M. Arias-Estrada, H.-R. Hernandez-de Leon, J.-L. Camas-Anzueto, and J. de Jesús Osuna-Coutiño, "An FPGA stereo matching unit based on fuzzy logic," *Microprocessors Microsyst.*, vol. 42, pp. 87–99, 2016. doi: 10.1016/j.micpro.2015.10.011.
[54] G. Cocorullo, P. Corsonello, F. Frustaci, and S. Perri, "An efficient hardware-oriented stereo matching algorithm," *Microprocessors Microsyst.*, vol. 46, pp. 21–33, 2016. doi: 10.1016/j.micpro.2016.09.010.
[55] P. M. Santos, J. C. Ferreira, and J. S. Matos, "Scalable hardware architecture for disparity map computation and object location in real-time," *J. Real-Time Image Process.*, vol. 11, no. 3, pp. 473–485, 2016. doi: 10.1007/s11554-013-0338-1.
[56] K. M. Ali, R. B. Atitallah, N. Fakhfakh, and J.-L. Dekeyser, "Exploring HLS optimizations for efficient stereo matching hardware implementation," in *Proc. Int. Symp. Appl. Reconfigurable Comput.*, 2017, pp. 168–176.
[57] B. McCullagh, "Real-time disparity map computation using the cell broadband engine," *J. Real-Time Image Process.*, vol. 7, no. 2, pp. 87–93, 2012. doi: 10.1007/s11554-010-0155-8.
[58] L. Li, X. Yu, S. Zhang, X. Zhao, and L. Zhang, "3d cost aggregation with multiple minimum spanning trees for stereo matching," *Appl. Opt.*, vol. 56, no. 12, pp. 3411–3420, 2017. doi: 10.1364/AO.56.003411.
[59] D. Zha, X. Jin, and T. Xiang, "A real-time global stereo-matching on FPGA," *Microprocessors Microsyst.*, vol. 47, pp. 419–428, 2016. doi: 10.1016/j.micpro.2016.08.005.
[60] L. Puglia, M. Vigliar, and G. Raiconi, "Real-time low-power FPGA architecture for stereo vision," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 64, no. 11, pp. 1307–1311, 2017. doi: 10.1109/TCSII.2017.2691675.
[61] A. Kjær-Nielsen et al., "A two-level real-time vision machine combining coarse-and fine-grained parallelism," *J. Real-Time Image Process.*, vol. 5, no. 4, pp. 291–304, 2010. doi: 10.1007/s11554-010-0159-4.
[62] S. Wong, S. Vassiliadis, and S. Cotofana, "A sum of absolute differences implementation in FPGA hardware," in *Proc. 28th Euromicro Conf.*, 2002, pp. 183–188.

[63] M. Hisham, S. N. Yaakob, R. A. Raof, A. A. Nazren, and N. W. Embedded, "Template matching using sum of squared difference and normalized cross correlation," in *Proc. IEEE Student Conf. Res. Development (SCOReD)*, 2015, pp. 100–104. doi: 10.1109/SCORED.2015.7449303.
[64] J.-C. Yoo and T. H. Han, "Fast normalized cross-correlation," *Circuits Syst. Signal Process.*, vol. 28, no. 6, p. 819, 2009. doi: 10.1007/s00034-009-9130-7.
[65] B. Froba and A. Ernst, "Face detection with the modified census transform," in *Proc. 6th IEEE Int. Conf. Automatic Face and Gesture Recogn.*, 2004, pp. 91–96.
[66] S. Jin et al., "FPGA design and implementation of a real-time stereo vision system," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 1, pp. 15–26, 2009.
[67] L. Zhang, K. Zhang, T. S. Chang, G. Lafruit, G. K. Kuzmanov, and D. Verkest, "Real-time high-definition stereo matching on FPGA," in *Proc. 19th ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, 2011, pp. 55–64. doi: 10.1145/1950413.1950428.
[68] D. Honegger, P. Greisen, L. Meier, P. Tanskanen, and M. Pollefeys, "Real-time velocity estimation based on optical flow and disparity matching," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, 2012, pp. 5177–5182.
[69] M. Jin and T. Maruyama, "Fast and accurate stereo vision system on FPGA," *ACM Trans. Reconfigurable Technol. Syst. (TRETS)*, vol. 7, no. 1, pp. 1–24, 2014. doi: 10.1145/2567659.
[70] S. Park and H. Jeong, "Real-time stereo vision FPGA chip with low error rate," in *Proc. Int. Conf. Multimedia and Ubiquitous Eng. (MUE'07)*, 2007, pp. 751–756.
[71] S. Sabihuddin, J. Islam, and W. J. MacLean, "Dynamic programming approach to high frame-rate stereo correspondence: A pipelined architecture implemented on a field programmable gate array," in *Proc. Canadian Conf. Electr. Comput. Eng.*, pp. 1461–1466, 2008.
[72] M. Jin and T. Maruyama, "A real-time stereo vision system using a tree-structured dynamic programming on FPGA," in *Proc.ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, 2012, pp. 21–24. doi: 10.1145/2145694.2145698.
[73] R. Kamasaka, Y. Shibata, and K. Oguri, "An FPGA-oriented graph cut algorithm for accelerating stereo vision," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, 2018, pp. 1–6. doi: 10.1109/RECONFIG.2018.8641737.
[74] C. Banz, S. Hesselbarth, H. Flatt, H. Blume, and P. Pirsch, "Real-time stereo vision system using semi-global matching disparity estimation: Architecture and FPGA-implementation," in *Proc. Int. Conf. Embedded Comput. Syst., Arch., Model. Simulation*, 2010, pp. 93–101.
[75] W. Wang, J. Yan, N. Xu, Y. Wang, and F.-H. Hsu, "Real-time high-quality stereo vision system in FPGA," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 10, pp. 1696–1708, 2015. doi: 10.1109/TCSVT.2015.2397196.
[76] L. F. Cambuim, J. P. Barbosa, and E. N. Barros, "Hardware module for low-resource and real-time stereo vision engine using semi-global matching approach," in *Proc.30th Symp. Integrated Circuits Syst. Des., Chip Sands*. 2017, pp. 53–58. doi: 10.1145/3109984.3109992.
[77] O. Rahnama, T. Cavalleri, S. Golodetz, S. Walker, and P. Torr, "R3sgm: Real-time raster-respecting semi-global matching for power-constrained systems," in *Proc. Int. Conf. Field-Programmable Technol. (FPT)*, 2018, pp. 102–109. doi: 10.1109/FPT.2018.00025.
[78] L. F. Cambuim, L. A. Oliveira, E. N. Barros, and A. P. Ferreira, "An FPGA-based real-time occlusion robust stereo vision system using semi-global matching," *J. Real-Time Image Process.*, vol. 17, no. 5, pp. 1–22, 2019. doi: 10.1007/s11554-019-00902-w.
[79] J. Zhao et al., "FP-stereo: Hardware-efficient stereo vision for embedded applications," 2020, arXiv:2006.03250.
[80] O. Rahnama, D. Frost, O. Miksik, and P. H. Torr, "Real-time dense stereo matching with ELAS on FPGA-accelerated embedded devices," *IEEE Robot. Automat. Lett.*, vol. 3, no. 3, pp. 2008–2015, 2018. doi: 10.1109/LRA.2018.2800786.
[81] O. Rahnama et al., "Real-time highly accurate dense depth on a power budget using an FPGA-CPU hybrid soc," *IEEE Trans. Circuits and Syst. II: Express Briefs*, vol. 66, no. 5, pp. 773–777, 2019. doi: 10.1109/TCSII.2019.2909169.
[82] H. Hirschmuller, "Accurate and efficient stereo processing by semi-global matching and mutual information," in *Proc. IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recogn. (CVPR'05)*, 2005, vol. 2, pp. 807–814.
[83] D. Honegger, H. Oleynikova, and M. Pollefeys, "Real-time and low latency embedded computer vision hardware based on a combination of FPGA and mobile CPU," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, 2014, pp. 4930–4935.
[84] S. Mattoccia and M. Poggi, "A passive RGBD sensor for accurate and real-time depth sensing self-contained into an FPGA," in *Proc. 9th Int. Conf. Distrib. Smart Cameras*, 2015, pp. 146–151. doi: 10.1145/2789116.2789148.
[85] S. K. Gehrig, F. Eberli, and T. Meyer, "A real-time low-power stereo vision engine using semi-global matching," in *Proc. Int. Conf. Comput. Vision Syst.*, 2009, pp. 134–143.
[86] D. Hernandez-Juarez, A. Chacón, A. Espinosa, D. Vázquez, J. C. Moure, and A. M. López, "Embedded real-time stereo estimation via semi-global matching on the GPU," *Procedia Comput. Sci.*, vol. 80, pp. 143–153, 2016. doi: 10.1016/j.procs.2016.05.305.
[87] H. Hirschmuller and D. Scharstein, "Evaluation of cost functions for stereo matching," in *Proc. IEEE Conf. Comput. Vision and Pattern Recogn.*, 2007, pp. 1–8. doi: 10.1109/CVPR.2007.383248.
[88] Y. Shan et al., "FPGA based memory efficient high resolution stereo vision system for video tolling," in *Proc. Int. Conf. Field-Programmable Technol.*, 2012, pp. 29–32.
[89] Y. Shan et al., "Hardware acceleration for an accurate stereo vision system using mini-census adaptive support region," *ACM Trans. Embedded Comput. Syst. (TECS)*, vol. 13, no. 4s, pp. 1–24, 2014. doi: 10.1145/2584659.
[90] A. Geiger, M. Roser, and R. Urtasun, "Efficient large-scale stereo matching," in *Proc. Asian Conf. Comput. Vision*, 2010, pp. 25–38.
[91] S. Zagoruyko and N. Komodakis, "Learning to compare image patches via convolutional neural networks," in *Proc. IEEE Conf. Comput. Vision and Pattern Recogn.*, 2015, pp. 4353–4361.
[92] J. Žbontar and Y. LeCun, "Stereo matching by training a convolutional neural network to compare image patches," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 2287–2318, 2016.
[93] W. Luo, A. G. Schwing, and R. Urtasun, "Efficient deep learning for stereo matching," in *Proc. IEEE Conf. Comput. Vision and Pattern Recogn.*, 2016, pp. 5695–5703.
[94] A. Seki and M. Pollefeys, "SGM-Nets: Semi-global matching with neural networks," in *Proc. IEEE Conf. Comput. Vision and Pattern Recogn.*, 2017, pp. 231–240.
[95] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *Proc. IEEE Conf. Comput. Vision and Pattern Recogn.*, 2016, pp. 4040–4048.
[96] A. Kuzmin, D. Mikushin, and V. Lempitsky, "End-to-end learning of cost-volume aggregation for real-time dense stereo," in *Proc. IEEE 27th Int. Workshop on Mach. Learn. Signal Process. (MLSP)*, 2017, pp. 1–6. doi: 10.1109/MLSP.2017.8168183.
[97] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, and L. Wang, "A high performance FPGA-based accelerator for large-scale convolutional neural networks," in *Proc. 26th Int. Conf. Field Programmable Logic and Appl. (FPL)*, 2016, pp. 1–9.
[98] J. Qiu et al., "Going deeper with embedded FPGA platform for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2016, pp. 26–35. doi: 10.1145/2847263.2847265.
[99] K. Guo et al., "Angel-eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 1, pp. 35–47, 2017. doi: 10.1109/TCAD.2017.2705069.
[100] J. Yu et al., "Instruction driven cross-layer CNN accelerator for fast detection on FPGA," *ACM Trans. Reconfigurable Technol. Syst. (TRETS*, ), vol. 11, no. 3, pp. 1–23, 2018. doi: 10.1145/3283452.
[101] H. Nakahara, H. Yonekawa, T. Fujii, and S. Sato, "A lightweight YOLOv2: A binarized CNN with a parallel support vector regression for an FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2018, pp. 31–40.
[102] M. S. Belshaw, "A high-speed iterative closest point tracker on an FPGA platform," PhD thesis, 2008.
[103] B. Williams, "Evaluation of a soc for real-time 3d slam," 2017.
[104] B. Van Hoorick, "FPGA-based simultaneous localization and mapping (slam) using high-level synthesis," 2019.
[105] Q. Gautier et al., "Real-time 3d reconstruction for FPGAs: A case study for evaluating the performance, area, and programmability trade-offs of the Altera OpenCL SDK," in *Proc. Int. Conf. Field-Programmable Technol. (FPT)*, 2014, pp. 326–329. doi: 10.1109/FPT.2014.7082810.

[106] T. Bailey, J. Nieto, J. Guivant, M. Stevens, and E. Nebot, "Consistency of the EKF-SLAM algorithm," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, 2006, pp. 3562–3568.
[107] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: A versatile and accurate monocular slam system," *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1147–1163, 2015. doi: 10.1109/TRO.2015.2463671.
[108] M. Montemerlo et al., "FastSLAM: A factored solution to the simultaneous localization and mapping problem," *AAAI/IAAI*, vol. 593598, 2002.
[109] M. Gu, K. Guo, W. Wang, Y. Wang, and H. Yang, "An FPGA-based real-time simultaneous localization and mapping system," in *Proc. Int. Conf. Field Programmable Technol. (FPT)*, 2015, pp. 200–203. doi: 10.1109/FPT.2015.7393150.
[110] C. Cadena et al., "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Trans. Robot.*, vol. 32, no. 6, pp. 1309–1332, 2016. doi: 10.1109/TRO.2016.2624754.
[111] J. Engel, J. Sturm, and D. Cremers, "Semi-dense visual odometry for a monocular camera," in *Proc. IEEE Int. Conf. Comput. Vision*, 2013, pp. 1449–1456.
[112] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *Int. J. Robot. Res.*, vol. 32, no. 11, pp. 1231–1237, 2013. doi: 10.1177/0278364913491297.
[113] M. Burri et al., "The EuRoC micro aerial vehicle datasets," *Int. J. Robot. Res.*, vol. 35, no. 10, pp. 1157–1163, 2016. doi: 10.1177/0278364915620033.
[114] R. A. Newcombe et al., "KinectFusion: Real-time dense surface mapping and tracking," in *Proc. 10th IEEE Int. Symp. Mixed and Augmented Reality*, 2011, pp. 127–136.
[115] V. Bonato, E. Marques, and G. A. Constantinides, "A floating-point extended Kalman filter implementation for autonomous mobile robots," *J. Signal Process. Syst.*, vol. 56, no. 1, pp. 41–50, 2009. doi: 10.1007/s11265-008-0257-8.
[116] D. T. Tertei, J. Piat, and M. Devy, "FPGA design and implementation of a matrix multiplier based accelerator for 3d EKF SLAM," in *Proc. Int. Conf. ReConFigurable Comput. and FPGAs (ReConFig14)*, 2014, pp. 1–6.
[117] D. T. Tertei, J. Piat, and M. Devy, "FPGA design of EKF block accelerator for 3d visual slam," *Comput. Electr. Eng.*, vol. 55, pp. 123–137, 2016. doi: 10.1016/j.compeleceng.2016.05.003.
[118] B. Vincke, A. Elouardi, and A. Lambert, "Real time simultaneous localization and mapping: towards low-cost multiprocessor embedded systems," *EURASIP J. Embedded Syst.*, vol. 2012, no. 1, p. 5, 2012. doi: 10.1186/1687-3963-2012-5.
[119] B. Vincke, A. Elouardi, A. Lambert, and A. Dine, "SIMD and OpenMP optimization of EKF-SLAM," in *Proc. Int. Conf. Multimedia Computi. Syst. (ICMCS)*, 2014, pp. 712–716. doi: 10.1109/ICMCS.2014.6911157.
[120] W. Fang, Y. Zhang, B. Yu, and S. Liu, "FPGA-based ORB feature extraction for real-time visual slam," in *Proc. Int. Conf. Field Programmable Technol. (ICFPT)*, 2017, pp. 275–278. doi: 10.1109/FPT.2017.8280159.
[121] Y. Biadgie and K.-A. Sohn, "Feature detector using adaptive accelerated segment test," in *Proc. Int. Conf. Inf. Sci. Appl. (ICISA)*, 2014, pp. 1–4.
[122] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Proc. European Conf. Comput. Vision*, 2010, pp. 778–792.
[123] R. Liu, J. Yang, Y. Chen, and W. Zhao, "eSLAM: An energy-efficient accelerator for real-time ORB-slam on FPGA platform," in *Proc. 56th Annu. Des. Automat. Conf.*, 2019, pp. 1–6.
[124] V. H. Schulz, F. G. Bombardelli, and E. Todt, "A Harris corner detector implementation in SoC-FPGA for visual slam," in *Robotics*. Springer-Verlag, 2016, pp. 57–71.
[125] M. Abouzahir, A. Elouardi, S. Bouaziz, R. Latif, and A. Tajer, "Large-scale monocular FastSLAM2. 0 acceleration on an embedded heterogeneous architecture," *EURASIP J. Adv. Signal Process.*, vol. 2016, no. 1, p. 88, 2016. doi: 10.1186/s13634-016-0386-3.
[126] M. Abouzahir, A. Elouardi, R. Latif, S. Bouaziz, and A. Tajer, "Embedding SLAM algorithms: Has it come of age?" *Robot. Autonom. Syst.*, vol. 100, pp. 14–26, 2018. doi: 10.1016/j.robot.2017.10.019.
[127] K. Boikos and C.-S. Bouganis, "Semi-dense SLAM on an FPGA SoC," in *Proc. 26th Int. Conf. Field Programmable Logic Appl. (FPL)*, 2016, pp. 1–4. doi: 10.1109/FPL.2016.7577365.
[128] K. Boikos and C.-S. Bouganis, "A high-performance system-on-chip architecture for direct tracking for slam," in *Proc. 27th Int. Conf. Field Programmable Logic Appl. (FPL)*, 2017, pp. 1–7. doi: 10.23919/FPL.2017.8056831.
[129] K. Boikos and C.-S. Bouganis, "A scalable FPGA-based architecture for depth estimation in SLAM," in *Proc. Int. Symp. Appl. Reconfigurable Comput.*, 2019, pp. 181–196.
[130] D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superpoint: Self-supervised interest point detection and description," in *Proc. IEEE Conf. Comput. Vision and Pattern Recogn. Workshops*, 2018, pp. 224–236.
[131] E. Simo-Serra, E. Trulls, L. Ferraz, I. Kokkinos, P. Fua, and F. Moreno-Noguer, "Discriminative learning of deep convolutional feature point descriptors," in *Proc. IEEE Int. Conf. Comput. Vision*, 2015, pp. 118–126.
[132] F. Radenović, G. Tolias, and O. Chum, "Fine-tuning CNN image retrieval with no human annotation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 7, pp. 1655–1668, 2018. doi: 10.1109/TPAMI.2018.2846566.
[133] Xilinx. "DPU for convolutional neural network."
[134] Z. Xu, J. Yu, C. Yu, H. Shen, Y. Wang, and H. Yang, "CNN-based feature-point extraction for real-time visual slam on embedded FPGA," in *Proc. IEEE 28th Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, 2020, pp. 33–37. doi: 10.1109/FCCM48280.2020.00014.
[135] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, arXiv:1510.00149.
[136] S. Krishnan, S. Chitlangia, M. Lam, Z. Wan, A. Faust, and V. J. Reddi, "Quantized reinforcement learning (QUARL)," 2019, arXiv:1910.01055.
[137] H. F. Langroudi, V. Karia, J. L. Gustafson, and D. Kudithipudi, "Adaptive posit: Parameter aware numerical format for deep learning inference on the edge," in *Proc. IEEE/CVF Conf. Comput. Vision and Pattern Recogn. Workshops*, 2020, pp. 726–727.
[138] T. Tambe et al., "Algorithm-hardware co-design of adaptive floating-point encodings for resilient deep learning inference," in *Proc. 57th ACM/IEEE Des. Automat. Conf. (DAC)*, 2020, pp. 1–6. doi: 10.1109/DAC18072.2020.9218516.
[139] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," 2016, arXiv:1605.04711.
[140] J. Choi, S. Venkataramani, V. Srinivasan, K. Gopalakrishnan, Z. Wang, and P. Chuang, "Accurate and efficient 2-bit quantized neural networks," in *Proc. 2nd SysML Conf.*, 2019, vol. 2019.
[141] J. Kim, K. Yoo, and N. Kwak, "Position-based scaled gradient for model quantization and pruning," *Adv. Neural Inform. Process. Syst.*, vol. 33, 2020.
[142] T. Tambe et al., "Adaptivfloat: A floating-point based data type for resilient deep learning inference," 2019, arXiv:1909.13271.
[143] J. Yu et al., "CNN-based monocular decentralized SLAM on embedded FPGA," 2020.
[144] H. Zhan, R. Garg, C. Saroj Weerasekera, K. Li, H. Agarwal, and I. Reid, "Unsupervised learning of monocular depth estimation and visual odometry with deep feature reconstruction," in *Proc. IEEE Conf. Comput. Vision and Pattern Recogn.*, 2018, pp. 340–349.
[145] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, "NetVLAD: CNN architecture for weakly supervised place recognition," in *Proc. IEEE Conf. Comput. Vision and Pattern Recogn.*, 2016, pp. 5297–5307.
[146] J. Yu et al., "INCA: Interruptible CNN accelerator for multi-tasking in embedded robots," in *Proc. 57th ACM/ESDA/IEEE Des. Automat. Conf. (DAC)*, 2020.
[147] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source slam system for monocular, stereo, and RGB-D cameras," *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, 2017. doi: 10.1109/TRO.2017.2705103.
[148] S. Liu, *Engineering Autonomous Vehicles and Robots: The DragonFly Modular-based Approach*, 1st ed. Wiley-IEEE Press, Mar. 2020.
[149] M. Maimone, Y. Cheng, and L. Matthies, "Two years of visual odometry on the mars exploration rovers," *J. Field Robot.*, vol. 24, no. 3, pp. 169–186, 2007. doi: 10.1002/rob.20184.
[150] B. Klingner, D. Martin, and J. Roseborough, "Street view motion-from-structure-from-motion," in *Proc. IEEE Int. Conf. Comput. Vision*, 2013, pp. 953–960.
[151] Y. Jeong, D. Nister, D. Steedly, R. Szeliski, and I.-S. Kweon, "Pushing the envelope of modern methods for bundle adjustment," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 8, pp. 1605–1617, 2011. doi: 10.1109/TPAMI.2011.256.
[152] C. Wu, S. Agarwal, B. Curless, and S. M. Seitz, "Multicore bundle adjustment," in *Proc. CVPR 2011*, 2011, pp. 3057–3064.
[153] A. Eriksson, J. Bastian, T.-J. Chin, and M. Isaksson, "A consensus-based framework for distributed bundle adjustment," in *Proc. IEEE Conf. Comput. Vision and Pattern Recogn.*, 2016, pp. 1754–1762.
[154] R. Zhang, S. Zhu, T. Fang, and L. Quan, "Distributed very large scale bundle adjustment by global camera consensus," in *Proc. IEEE Int. Conf. Comput. Vision*, 2017, pp. 29–38.
[155] A. Suleiman, Z. Zhang, L. Carlone, S. Karaman, and V. Sze, "Navion: A 2-mw fully integrated real-time visual-inertial odometry accelerator for autonomous navigation of nano drones," *IEEE J. Solid-State Circuits*, vol. 54, no. 4, pp. 1106–1119, 2019. doi: 10.1109/JSSC.2018.2886342.
[156] Q. Liu, S. Qin, B. Yu, J. Tang, and S. Liu, "π-ba: Bundle adjustment hardware accelerator based on distribution of 3d-point observations," *IEEE Trans. Comput.*, 2020.

[157] R. Sun, P. Liu, J. Xue, S. Yang, J. Qian, and R. Ying, "BAX: A bundle adjustment accelerator with decoupled access/execute architecture for visual odometry," *IEEE Access*, vol. 8, pp. 75,530–75,542, 2020. doi: 10.1109/ACCESS.2020.2988527.
[158] P. Leven and S. Hutchinson, "A framework for real-time path planning in changing environments," *Int. J. Robot. Res.*, vol. 21, no. 12, pp. 999–1030, 2002. doi: 10.1177/027836490202101200l.
[159] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, 2011. doi: 10.1177/0278364911406761.
[160] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, 2015, pp. 3067–3074.
[161] K. Hauser, "Lazy collision checking in asymptotically-optimal motion planning," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, 2015, pp. 2951–2957.
[162] A. Yershova and S. M. LaValle, "Improving motion-planning algorithms by efficient nearest-neighbor searching," *IEEE Trans. Robot.*, vol. 23, no. 1, pp. 151–157, 2007. doi: 10.1109/TRO.2006.886840.
[163] W. Wang, D. Balkcom, and A. Chakrabarti, "A fast online spanner for roadmap construction," *Int. J. Robot. Res.*, vol. 34, no. 11, pp. 1418–1432, 2015. doi: 10.1177/0278364915576491.
[164] S. Murray, W. Floyd-Jones, G. Konidaris, and D. J. Sorin, "A programmable architecture for robot motion planning acceleration," in *Proc. IEEE 30th Int. Conf. Appl.-Specific Syst., Arch. Process. (ASAP)*, 2019, vol. 2160, pp. 185–188.
[165] J. Bialkowski, S. Karaman, and E. Frazzoli, "Massively parallelizing the RRT and the RRT," in *Proc. IEEE/RSJ Int, Conf. Intell. Robots and Syst.*, 2011, pp. 3513–3518.
[166] J. Pan and D. Manocha, "GPU-based parallel collision detection for fast motion planning," *Int. J. Robot. Res.*, vol. 31, no. 2, pp. 187–200, 2012. doi: 10.1177/0278364911429335.
[167] J. Pan, C. Lauterbach, and D. Manocha, "G-planner: Real-time motion planning and global navigation using GPUs," in *AAAI*, 2010.
**[168] N. Atay and B. Bayazit, "A motion planning processor on reconfigurable hardware," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA 2006)*, 2006, pp. 125–132.
[169] S. Murray, W. Floyd-Jones, Y. Qi, G. Konidaris, and D. J. Sorin, "The microarchitecture of a real-time robot motion planning accelerator," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarch. (MICRO)*, 2016, pp. 1–12. doi: 10.1109/MICRO.2016.7783748.
[170] S. Lian, Y. Han, X. Chen, Y. Wang, and H. Xiao, "DADU-P: A scalable accelerator for robot motion planning in a dynamic environment," in *Proc. 55th ACM/ESDA/IEEE Design Automat. Conf. (DAC)*, 2018, pp. 1–6. doi: 10.1109/DAC.2018.8465785.
[171] U. Bondhugula et al., "Hardware/software integration for FPGA-based all-pairs shortest-paths," in *Proc. 14th Annu. IEEE Symp. Field-Programmable Custom Comput. Mach.*, 2006, pp. 152–164. doi: 10.1109/FCCM.2006.48.
[172] K. Sridharan, T. Priya, and P. R. Kumar, "Hardware architecture for finding shortest paths," in *Proc. TENCON 2009 IEEE Region 10 Conf.*, pp. 1–5.
[173] Y. Takei, M. Hariyama, and M. Kameyama, "Evaluation of an FPGA-based shortest-path-search accelerator," in *Proc. Int. Conf. Parallel Distrib. Process. Techn. Appl. (PDPTA)*, The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2015 p. 613.
[174] K. Vipin and S. A. Fahmy, "FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications," *ACM Comput. Surveys (CSUR)*, vol. 51, no. 4, pp. 1–39, 2018. doi: 10.1145/3193827.
[175] S. Liu, R. N. Pittman, and A. Forin, "Minimizing partial reconfiguration overhead with fully streaming DMA engines and intelligent ICAP controller," in *FPGA*, 2010, p. 292.
[176] S. Liu, R. N. Pittman, A. Forin, and J.-L. Gaudiot, "Achieving energy efficiency through runtime partial reconfiguration on reconfigurable systems," *ACM Trans. Embedded Comput. Syst. (TECS)*, vol. 12, no. 3, p. 72, 2013. doi: 10.1145/2442116.2442122.
[177] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski, "Orb: An efficient alternative to sift or surf," in *ICCV*, vol. 11, p. 2, 2011.
[178] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proc. 7th Int. Joint Conf. Artif. Intell.*, 1981.
[179] W. Fang, Y. Zhang, B. Yu, and S. Liu, "Dragonfly+: FPGA-based quad-camera visual slam system for autonomous vehicles," *Proc. IEEE HotChips*, p. 1, 2018.
[180] T. Qin, P. Li, and, and S. Shen, "VINS-MONO: A robust and versatile monocular visual-inertial state estimator," *IEEE Trans. Robot.*, vol. 34, no. 4, pp. 1004–1020, 2018. doi: 10.1109/TRO.2018.2853729.
[181] K. Sun et al., "Robust stereo visual inertial odometry for fast autonomous flight," *IEEE Robot. Automat. Lett.*, vol. 3, no. 2, pp. 965–972, 2018. doi: 10.1109/LRA.2018.2793349.
[182] R. Szeliski, "Computer vision: Algorithms and applications," in *Texts in Computer Science*. London: Springer-Verlag, 2010.
[183] A. Geiger, M. Roser, and R. Urtasun, "Efficient large-scale stereo matching," in *Proc. 10th Asian Conf. Comput. Vision*, 2010.
[184] Y. Feng, P. Whatmough, and Y. Zhu, "ASV: Accelerated stereo vision system," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarch. (MICRO '52)*, 2019, p. 643–656.
[185] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 3, pp. 583–596, Mar. 2015. doi: 10.1109/TPAMI.2014.2345390.
[186] A. Kelly, *Mobile Robotics: Mathematics, Models, and Methods*. Cambridge Univ. Press, 2013.
[187] J. Tang, B. Yu, S. Liu, Z. Zhang, W. Fang, and Y. Zhang, "π-soc: Heterogeneous soc architecture for visual inertial slam applications," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots and Syst. (IROS)*, 2018, pp. 8302–8307. doi: 10.1109/IROS.2018.8594181.
[188] S. Qin, Q. Liu, B. Yu, and S. Liu, "π-ba: Bundle adjustment acceleration on embedded FPGAs with co-observation optimization," in *Proc. 27th IEEE Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM 2019)*, San Diego, CA, Apr. 28–May 1, 2019, pp. 100–108.
[189] https://grail.cs.washington.edu/projects/bal/
[190] P. L. Mckerracher, R. P. Cain, J. C. Barnett, W. S. Green, and J. D. Kinnison, "Design and test of field programmable gate arrays in space applications," 1992.
[191] M. Berg, "FPGA mitigation strategies for critical applications," 2019.
[192] D. Sheldon, "Flash-based FPGA NEPP FY12 summary report,"
[193] R. Gaillard, "Single event effects: Mechanisms and classification," in *Soft Errors in Modern Electronic Systems*. Springer-Verlag, 2011, pp. 27–54.
[194] M. Wirthlin, "FPGAs operating in a radiation environment: lessons learned from FPGAs in space," *J. Instrumentation*, vol. 8, no. 2, p. C02020, 2013. doi: 10.1088/1748-0221/8/02/C02020.
[195] F. Brosser and E. Milh, "SEU mitigation techniques for advanced reprogrammable FPGA in space," Master's thesis, 2014.
[196] B. Ahmed and C. Basha, "Fault mitigation strategies for reliable FPGA architectures," Ph.D. thesis, Rennes 1, 2016.
[197] S. Habinc, "Suitability of reprogrammable FPGAs in space applications," Gaisler Research, Feasibility Rep., 2002.
[198] G. Lentaris et al., "High-performance embedded computing in space: Evaluation of platforms for vision-based navigation," *J. Aerospace Inform. Syst.*, vol. 15, no. 4, pp. 178–192, 2018. doi: 10.2514/1.I010555.
[199] T. Y. Li and S. Liu, "Enabling commercial autonomous robotic space explorers," *IEEE Potentials.*, vol. 39, no. 1, pp. 29–36, 2019. doi: 10.1109/MPOT.2019.2935338.
[200] D. Ratter, "FPGAs on Mars," *Xcell J.*, vol. 50, pp. 8–11, 2004.
[201] J. F. Bell III et al., "Mars exploration rover athena panoramic camera (pancam) investigation," *J. Geophys. Res. Planets*, vol. 108, 2003. doi: 10.1029/2003JE002070.
[202] "Space flight system design and environmental test." https://www.nasa.gov/sites/default/files/atoms/files/std8070.1.pdf (accessed Sept. 1, 2020)
[203] M. C. Malin et al., "The Mars Science Laboratory (MSL) mast cameras and descent imager: Investigation and instrument descriptions," *Earth Space Sci.*, vol. 4, no. 8, pp. 506–539, 2017. doi: 10.1002/2016EA000252.
[204] C. D. Edwards, T. C. Jedrey, A. Devereaux, R. DePaula, and M. Dapore, "The electra proximity link payload for Mars relay telecommunications and navigation," 2003. doi: 10.2514/6.IAC-03-Q.3.a.06.
[205] A. Johnson et al., "The lander vision system for Mars 2020 entry descent and landing," 2017.
[206] "Vivado high-level synthesis." https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html (accessed Sept. 10, 2020)