PAPER Special Section on Picture Coding and Image Media Processing

Algorithm-Hardware Co-Design of Real-Time Edge Detection for Deep-Space Autonomous Optical Navigation

Hao XIAO[†,††a)], **Yanming FAN**[†††], ***Nonmembers***, **Fen GE**[††,†††], ***Member***, **Zhang ZHANG**[†], ***and*** **Xin CHENG**[†], ***Nonmembers***

SUMMARY Optical navigation (OPNAV) is the use of the on-board imaging data to provide a direct measurement of the image coordinates of the target as navigation information. Among the optical observables in deep-space, the edge of the celestial body is an important feature that can be utilized for locating the planet centroid. However, traditional edge detection algorithms like Canny algorithm cannot be applied directly for OPNAV due to the noise edges caused by surface markings. Moreover, due to the constrained computation and energy capacity on-board, light-weight image-processing algorithms with less computational complexity are desirable for real-time processing. Thus, to fast and accurately extract the edge of the celestial body from high-resolution satellite imageries, this paper presents an algorithm-hardware co-design of real-time edge detection for OPNAV. First, a light-weight edge detection algorithm is proposed to efficiently detect the edge of the celestial body while suppressing the noise edges caused by surface markings. Then, we further present an FPGA implementation of the proposed algorithm with an optimized real-time performance and resource efficiency. Experimental results show that, compared with the traditional edge detection algorithms, our proposed one enables more accurate celestial body edge detection, while simplifying the hardware implementation.

key words: *edge detection, autonomous optical navigation, star centroid estimation, real-time processing*

1. Introduction

As future missions ambitiously take us farther into the Solar system, there will be a need for autonomous operations with minimal Earth contact. One key enabler for autonomy is on-board optical navigation (OPNAV) that uses imaging data to aid in spacecraft navigation. It has the advantages of independence, low cost, high reliability, high accuracy, and real-time performance, and has become a key technology in deep space exploration [1], [2]. In the typical case, OPNAV uses the optical sensor on-board to take pictures of some target body, extracts the optical observables from the picture, and then, measures the image coordinates of the target as navigation information [3]. Among the available optical observables in deep-space, the edge of the celestial body is an important feature for locating the planet centroid [4], [5]. As a result, accurate and fast edge extraction of the celestial body from satellite imageries is crucially important for high-precision and real-time OPNAV.

Among the existing edge detection algorithms, the Canny edge detector [6] has remained a standard for many years and has best performance [7]. However, the computationally intensive nature of the Canny algorithm makes it costly on hardware resources and processing speed. Especially for high-resolution satellite imageries, the computational load of Canny algorithm may increase exponentially, and hence, make it challenging to real-time process under the constrained computation and energy resources on-board. More importantly, there are surface markings covered on the celestial body, such as craters and atmosphere, which may incur false edges using conventional edge detectors. As shown in Fig. 1, the traditional Canny algorithm cannot isolate the noise edges caused by surface markings, which result in significant accuracy loss for the measurement of the planet centroid. Therefore, fast and robustness celestial body edge detection algorithms with a high degree of hardware efficiency are indispensable for the success of OPNAV.

Many recent works have presented hardware implementation for real-time Canny edge detection. In [7], [8], constant threshold is used regardless of the image characteristics to reduce the computational complexity, where the performance of edge detection is relatively degraded compared with adaptive threshold approaches. He *et al.* [9] present a pipelined implementation of an improved Canny edge detector with a self-adapt threshold mechanism. In [10], a parallel design of Canny implementation is proposed for high-throughput real-time edge detection. Xu *et al.* [11] explore the parallelism of Canny edge detection at block level and propose an implementation of distributed Canny edge detection algorithm to reduce processing latency. To support a variable input resolution, Possa *et al.* [12] present a multi-resolution FPGA-based architecture for flexible and real-time Canny edge detection. In [13], a fast and resource-

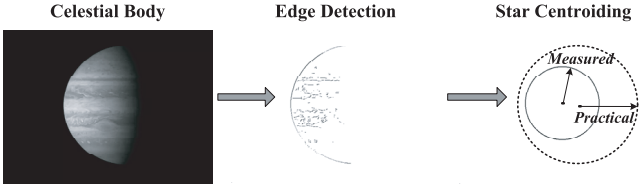

Fig. 1 Problems of using Canny algorithm to extract the planet centroid

Manuscript received December 20, 2019.
Manuscript revised March 19, 2020.
Manuscript publicized June 15, 2020.

[†]The authors are with the School of Microelectronics, HeFei University of Technology, Hefei, China.
[††]The authors are with the Science and Technology on Electronic Information Control Laboratory, Chengdu, China.
[†††]The authors are with the College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing, China.
a) E-mail: xiaohao@hfut.edu.cn

DOI: 10.1587/transinf.2020PCP0002

efficient hardware implementation of Canny algorithm that works in stream-in-stream-out manner is proposed for real-time line detection. In a more recent paper [14], the authors focus on the energy-efficient Canny edge detection for advanced mobile vision applications by proposing low complexity architectures for the major computation element of Canny algorithm, including an image filter, a gradient magnitude and direction calculator, and an adaptive threshold value selection module. The above designs have shown effectiveness in real-time edge detection using FPGAs. However, these designs target the traditional Canny algorithm, which cannot be applied directly for OPNAV because of the noise edges caused by surface markings.

There are also literature on the extraction of celestial body edges for OPNAV. Li *et al.* [15] use image preprocessing technology to enhance the contours and suppress the rest of the image, and then, adopt Canny algorithm to extract edges. Du *et al.* [16] use threshold and morphological method to initially remove the noise. The edge points of a celestial body are then detected by Prewitt-Zernike moment operator. In [17], Richardson extrapolation is used to achieve image differentiation, and then, a small gradient box and RANSAC algorithm are used to extract edge points. Jiang and Wang *et al.* [18], [19] try to combine the functions of a star tracker and a navigation with a single camera by proposing a feature extraction algorithm to synchronously extract the star centroid and celestial body edges in an image. The accuracy achieved by these edge extraction algorithms are able to meet the requirements of OPNAV. However, none of them considers the practical hardware implementation of these algorithms, and less computation in the image-processing algorithm is necessary for real-time OPNAV due to the constrained computation and energy capacity on-board. To enable fast and accurate extraction of celestial body edges from high resolution images with less computational complexity, we presents an optimized edge detection algorithm for real-time deep-space autonomous OPNAV [20]. This paper extends our previous work [20] in several aspects: (1) we describe in further detail the optimized algorithm, hardware architecture and implementation results; (2) experimental comparisons to the state of the art are elaborated in the algorithm and hardware performance.

In this paper, we present an algorithm-hardware co-design of real-time edge detection for deep-space autonomous optical navigation. To fast and accurately extract the edge of the celestial body from high-resolution satellite imageries, we propose a light-weight edge detection algorithm as well as an FPGA-based real-time processing architecture. First, an optimized non-maximum suppression (NMS) mechanism is proposed to ease the hardware implementation without sacrificing the accuracy. Second, an adaptive threshold selection approach is proposed to suppress the noise edges caused by craters and atmosphere covered on celestial bodies. Moreover, unlike the conventional double-thresholding method, the proposed scheme adopts a single threshold method, thus costing much less computational complexity and facilitating the hardware implementation. According to the optimized algorithm, we further implement a completely pipelined hardware architecture for real-time edge detection. Optimized storage and computation structures are proposed for improving the real-time performance and the resource efficiency. We have implemented and verified the proposed solution using Virtex-4 FPGA. Experimental results show that, compared with the traditional Canny edge detector, the proposed algorithm enables more accurate celestial body edge detection, while simplifying the hardware implementation.

The rest of this paper is arranged as follows: Sect. 2 gives a brief introduction of the traditional Canny edge detector. Section 3 presents our proposed edge detection algorithm, which is followed by the hardware architecture in Sect. 4. Implementation results and conclusions are given in Sect. 5 and 6, respectively.

2. Canny Edge Detection Algorithm

Figure 2 shows the block diagram and the basic procedures of classic Canny edge detection algorithm [6]. It consists of the following five steps: (1) applying Gaussian filtering to the input images to remove the noise; (2) computing the image derivatives and calculating the gradient magnitude and orientation for finding the intensity gradients of the image; (3) NMS process to get rid of spurious response to edge detection, which is also referred as edge thinning process; (4) extracting high and low threshold from the gradient magnitude histogram of entire image; and (5) hysteresis thresholding to apply high and low thresholds with connectivity analysis. The upper threshold is used to mark edges that are definitely edges. The lower threshold is to find faint pixels that are actually a part of an edge.

Performing Gaussian blur is used for removing noise that presents in images, and smoothing the images. As shown in Eq. (1), it is carried out using one 2D Gaussian filter $g(x, y)$, which can be further represented by two 1D Gaussian filters $g(x)$ and $g(y)$. Here, x is the distance from the origin in the horizontal axis, y is the distance from the origin in the vertical axis, and σ is the standard deviation of the Gaussian distribution.

$$\begin{aligned} g(x,y) &= \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \\ &= \left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}\right)\left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}}\right) \\ &= g(x) \cdot g(y) \end{aligned} \tag{1}$$

After Gaussian filtering, gradient magnitudes and directions are calculated at every single point in the image. The gradient magnitude of a certain point determines if it

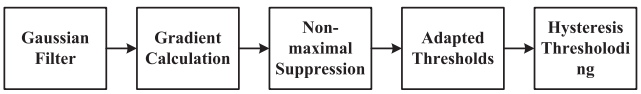

Fig. 2 Block diagram and the basic procedures of Canny edge detection algorithm

possibly lies on an edge or not. And the gradient direction shows how the edge is oriented. The gradient can be calculated by using the Sobel operator shown in Eq. (2), where H_x and H_y are the horizontal and vertical derivatives at the point being considered. G_x and G_y are the horizontal and the vertical derivative of image, respectively. And the gradient magnitude $|G|$ and the gradient direction θ can be represented by Eq. (3) and Eq. (4), respectively.

Next, NMS is an edge-thinning algorithm that eliminates non-ridge pixels giving a one-pixel-wide aspect at the edges. It works by finding the pixel with the maximum gradient magnitude in the gradient direction. The gradient magnitude of this pixel is compared with two of its immediate neighbors along the gradient direction and the gradient magnitude is set to zero if it does not correspond to a local maximum. When the pixel's gradient direction is one of eight possible main directions (0°, 45°, 90°, 135°, 180°, 225°, 275°, 315°), the neighbor pixels are decided. For the gradient directions that do not coincide with one of the eight possible main directions (ex. M_1 and M_2 shown in Fig. 3), an interpolation is necessary to infer the neighboring gradients using Eq. (5)–(7). Here, d is the tangent function of the gradients direction θ, and M_1 and M_2 are the gradient magnitudes of neighbor pixels along the direction.

$$H_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, H_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{2}$$

$$|G| = \sqrt{(G_x)^2 + (G_y)^2} \tag{3}$$

$$\theta = \arctan\left(\frac{G_y}{G_x}\right) \tag{4}$$

$$d = \frac{G_y}{G_x} \tag{5}$$

$$M_1 = d \cdot M(x+1, y+1) + (1-d) \cdot M(x+1, y) \tag{6}$$

$$M_2 = d \cdot M(x-1, y-1) + (1-d) \cdot M(x-1, y) \tag{7}$$

After NMS, there still exist noise edge points. Thus, double thresholding is adopted to further determine whether an image pixel is on an edge or not. It works by setting two thresholds, a high and a low threshold, based on the histogram of the gradient magnitude for the entire image. The high threshold T_{high} is computed such that a percentage P_1 of the total pixels in the image would be classified as strong edges. The low threshold T_{low} is computed as a percentage P_2 of the high threshold.

Then, in the hysteresis thresholding stage, two thresholds are used, T_{low} and T_{high}. All pixel with a magnitude higher than T_{high} are considered true edges. Pixels with magnitudes between T_{low} and T_{high} are considered as edge candidates. Pixels that do not satisfy these two criteria are suppressed. Edge candidates become true edges if they are connected to true edges directly or through other candidates. For more information regarding methodologies to determine the threshold values, we refer the readers to [21], [22].

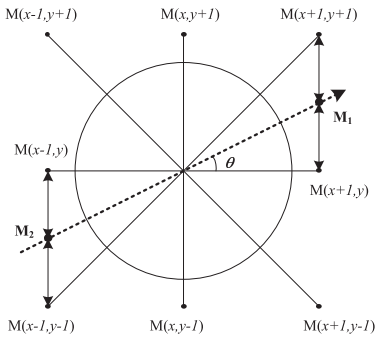

Fig. 3 Interpolation for computing the neighboring gradients

3. Proposed Algorithm

In the scenario of OPNAV, it is required to isolate the edge of the celestial body from the satellite imagery for extracting the planet centroid. Although the classic Canny algorithm have shown effectiveness in detecting edges, it is not a viable option for OPNAV because of its particularity. First, the Canny algorithm cannot distinguish the edges caused by the contour of the celestial body and the ones caused by the craters and atmosphere covered on the celestial body. For measuring the planet centroid, the edges caused by the texture on the celestial body are noises, which may interfere the accuracy of the measurement significantly. Second, considering the constrained hardware and energy resources onboard, the original NMS algorithm defined in Eq. (3)–(7) is quite expensive to implement on hardware. Even though using hard macros, e.g., CORDIC IP, provided by FPGA tools can facilitate the implementation, it still requires lots of hardware resources that may exceed the limits of the FPGA hardware resources. Moreover, the above mentioned T_{low} and T_{high} for hysteresis thresholding are selected based on the gradient magnitude histogram, which needs a large computational complexity and storage resources, especially for high-resolution satellite imageries. Thus, to suppress the noise edges caused by textures and enable the algorithm easier for real-time processing, this paper proposes a lightweight edge detection algorithm for OPNAV. Generally, it follows the basic steps of the classic Canny edge detection shown in Fig. 2. However, the NMS mechanism and the adaptive threshold selection approach are optimized for ensuring the algorithm viable for real-time OPNAV applications.

3.1 Optimized Non-Maximun Suppression

NMS is applied on the pixels to remove the weaker edges.

Table 1 Proposed NMS method

Region	Gradient		NMS
	Magnitude	Sign	
1	$\|G_x\| > 2\|G_y\|$	/	$\|G_{(x,y)}\|$ vs. $\{\|G_{(x-1,y)}\|, \|G_{(x+1,y)}\|\}$
2	$\|G_y\| < \|G_x\| < 2\|G_y\|$	XOR(S_{G_x}, S_{G_y})=1	$\|G_{(x,y)}\|$ vs. $\{\|G_{(x-1,y)}\|, \|G_{(x+1,y)}\|, \|G_{(x-1,y+1)}\|, \|G_{(x+1,y-1)}\|\}$
3	$\|G_y\| < \|G_x\| < 2\|G_y\|$	XOR(S_{G_x}, S_{G_y})=0	$\|G_{(x,y)}\|$ vs. $\{\|G_{(x-1,y)}\|, \|G_{(x+1,y)}\|, \|G_{(x+1,y+1)}\|, \|G_{(x-1,y-1)}\|\}$
4	$\|G_y\|/2 < \|G_x\| < \|G_y\|$	XOR(S_{G_x}, S_{G_y})=1	$\|G_{(x,y)}\|$ vs. $\{\|G_{(x,y+1)}\|, \|G_{(x,y-1)}\|, \|G_{(x-1,y+1)}\|, \|G_{(x+1,y-1)}\|\}$
5	$\|G_y\|/2 < \|G_x\| < \|G_y\|$	XOR(S_{G_x}, S_{G_y})=0	$\|G_{(x,y)}\|$ vs. $\{\|G_{(x,y+1)}\|, \|G_{(x,y-1)}\|, \|G_{(x+1,y+1)}\|, \|G_{(x-1,y-1)}\|\}$
6	$\|G_x\| < \|G_y\|/2$	/	$\|G_{(x,y)}\|$ vs. $\{\|G_{(x,y+1)}\|, \|G_{(x,y-1)}\|\}$

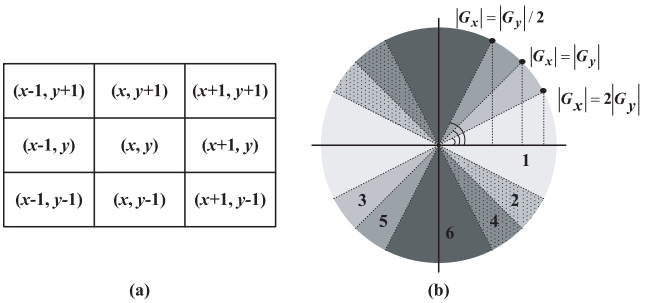

Fig. 4 Proposed low complexity NMS method. (a) Location of neighborhood pixels for NMS. (b) Determination of diagonal directions

However, the direct calculation of the gradient magnitude and direction needs large computational complexity. Furthermore, using interpolation to estimate the gradient magnitude of the neighbor pixels along the gradient direction is also expensive for real-time processing. Thus, approximation techniques are non-trivial while guaranteeing the accuracy of the detected edges. For the gradient magnitude, one possible approximation to the Euclidean norm is by using the sum of absolute values shown in Eq. (8). For the gradient direction, instead of calculating the angle, this paper proposes to infer the direction by comparing the magnitude of the horizontal (G_x) and vertical (G_y) gradient component. As shown in Fig. 4, our approach simplifies the gradient direction to one of the six sections. With the rotation invariant characteristic of the gradient direction, these regions can be identified by 5 critical angles which are 0°, 26.58°($\approx \arctan\frac{1}{2}$), 45°, 71.58°, 90°, respectively. And to assign each pixel to one out of the six possible directions, we utilize the magnitude and the sign of the horizontal and vertical gradient components (G_x and G_y).

$$\sqrt{G_x^2 + G_y^2} \approx |G_x| + |G_y| \tag{8}$$

As an illustration, Table 1 lists the detailed direction partitions, as well as the NMS operation carried out within each given region. When $|G_x| > 2\left|G_y\right|$ is selected, the gradient direction is located in regions 0, while $\left|G_y\right| > 2\,|G_x|$ selection refers to region 6. If $|G_y| < |G_x| < 2|G_y|$ is selected, the gradient direction can be within region 2 or 3, and this needs a further decision by checking the sign bit of G_x and G_y with an XOR gate. When G_x and G_y have different signs, the gradient direction should be within region 2. Otherwise, it should be within region 3. Similarly, when $|G_y|/2 < |G_x| < |G_y|$ is selected, the gradient direction can be within region 4 or 5. In case of region 4, G_x and G_y have different signs, and in case of region 5, G_x and G_y have the same sign. After region selection, the actual NMS is performed, where the magnitude of the edge pixel is compared with the corresponding neighborhood pixels. For the edge pixel located in region 1 and 6, it is compared with adjacent pixels in the horizontal and vertical directions, respectively. For region 2 and 3, the edge pixel is compared with adjacent pixels not only in the horizontal direction but also the diagonal direction (45°). And for region 4 and 5, the edge pixel is compared with adjacent pixels in both horizontal and diagonal (135°) directions. Only the pixel whose gradient magnitude is larger than both of its neighbors in the gradient direction is accepted as the candidate for further processing.

Table 2 Performance comparison for the NMS algorithms.

	Original Canny	[12]	[14]	Proposed
Average P_{fa}	0	6.70%	4.73%	0.95%

To evaluate the proposed NMS approach, the simulation is carried out with more than 100 published deep-space images. For comparison, the original NMS algorithm and another two approximation schemes [12] and [14] are also employed in our simulation. The original NMS algorithm uses Eq. (3)–(7), where Euclidean norm is adopted for calculating the gradient magnitude and the interpolation scheme is adopted for calculating the neighbors' gradient. Since this work uses the detected edges to infer the planet centroid, the noise edges incurred by the approximation are of crucial importance to the accuracy of the estimated centroid. Thus, the percentage of the detected edge pixels by the proposed algorithm that are not detected by the original Canny edge detector, also referred to false positives P_{fa}, is used to represent the performance of the proposed scheme. As shown in Table 2, the proposed scheme shows only 0.95% error compared with the ideal one. However, considering the computational complexity, the proposed NMS consists of hardware-friendly operations, e.g., addition, ×2 and ÷2, which eases the hardware implementation significantly. Comparing with the other two approximation schemes, the proposed one also shows the lowest P_{fa}. This is because [12] and [14] simplify the gradient directions into 2 and 4 sections respec-

tively, while our proposed one uses 6 sections to approximate, which is more closed to the ideal one.

3.2 Simplified Adaptive Threshold Selection

Unlike general images whose pattern are diverse, the background and object in the deep-space environment is monotony. It is observed that well-exposed celestial objects always have cleared contour with smoothed background. However, the craters, texture, and atmosphere covered on the celestial body may also incur edges, which interfere the further calculation of the planet centroid significantly. Thus, the purpose of the threshold in this work is to identify the true edges of the celestial body while suppressing the noise edges as much as possible. To do this, we observe that, due to the darkness of the deep-space background, the intensity of the image changes much more sharply at the real edge when compared with the noise edge.

Exploiting this idea, this paper proposes an adaptive single-threshold segmentation algorithm, which allow isolating the real edge of the rest of the image, and also being convenient for hardware implementation. Eq. (9) shows the proposed adaptive threshold, whose computation steps are as follows: (1) assuming the pixels of the entire image to be $p(x, y)$, and calculating the mean value of the gradient magnitudes $\overline{G_{p(x,y)}}$ for $p(x, y)$; (2) removing pixels whose gradient magnitude is smaller than $\overline{G_{(x,y)}}$, resulting in the subset of original image $p_0(x, y)$, and then calculate the mean value of gradient magnitude $\overline{G_{p_0(x,y)}}$ for $p_0(x, y)$; 3) Calculating the standard deviation of the gradient magnitude $\sigma_{p_0(x,y)}$ for $p_0(x, y)$, and get the threshold via Eq. (9).

$$T = \overline{G_{p_0(x,y)}} + \sigma_{p_0(x,y)} \times 2 \tag{9}$$

Figure 5 shows the simulation results of the real deep-space images published by NASA (Jupiter PIA04866, Mars PIA04591, Mercury PIA10398 and Pluto PIA09113). In more details, Fig. 5(b)-(d) shows the binary images according to the steps above. At the 1st step, $\overline{G_{p(x,y)}}$ is used to remove the background, leaving the celestial body containing both true edges and noise edges (as shown in Fig. 5(b)). Thus, using $\overline{G_{p_0(x,y)}}$ may further remove part of the noise edges, which is shown in Fig. 5(c). Finally, the standard deviation ($\sigma_{p_0(x,y)}$) of the gradient magnitude are adopted to filter the rest noise edges.

4. Hardware Architecture

In this section, we shall show how to implement the proposed edge detection algorithm on FPGA so that it can meet the needs of real-time autonomous optical navigation. In the rest of this section, we provide a high-level overview of the proposed hardware architecture followed by details of each functional component.

4.1 Architecture Overview

The block diagram of the proposed hardware architecture for edge detection is shown in Fig. 6. According to the algorithm above, it consists of five functional blocks: Gaussian filter, gradient calculator, NMS unit, adaptive threshold calculation, and thresholding. Getting the data stream from the

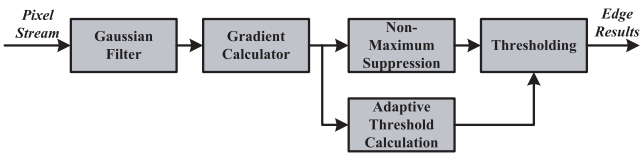

Fig. 6 Overview of proposed hardware architecture

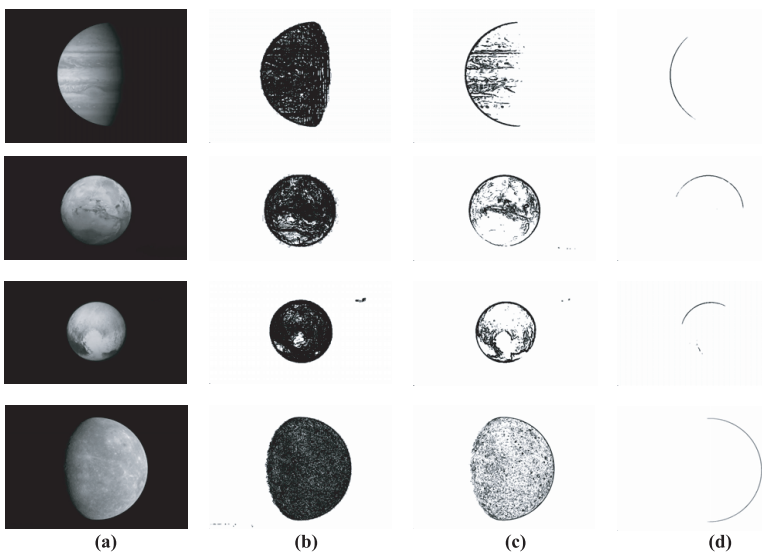

Fig. 5 Simulation results of the proposed adaptive threshold selection method. (a) Real deep-space images. (b) Binary images after filtering by $\overline{G_{p(x,y)}}$. (c) Binary images after filtering by $\overline{G_{p_0(x,y)}}$. (d) Binary images after threshold segmentation.

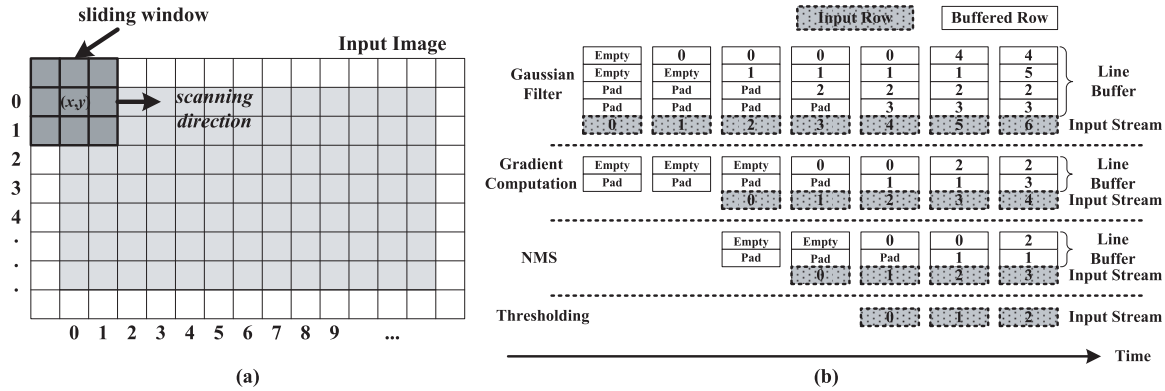

Fig. 7 Pipelined image processing in stream-in-stream-out manner. (a) A 3×3 sliding window scanning from the first column of the first row until the last column of the last row. (b) In/Out stream of each pipelined functional stage.

image sensor, the system first applies a Gaussian filter on the whole image. Then, the gradient at each pixel is calculated for getting the gradient magnitude and direction of a certain pixel. Furthermore, non-maximum suppression is carried out as described in Sect. 3.2. Meanwhile, the adaptive thresholds are calculated according to Eq. (9). Finally, the threshold is used to select the edge results after NMS.

The proposed architecture works in a completely pipelined stream-in-stream-out manner. The image pixels come in as a stream from the first point in the upper left to the last one in the lower right. As shown in Fig. 7(a), the detection is carried out by a window sliding the image from the first column of the first row until the last column of the last row to process every pixel with its neighborhood to conduct the edge results. The pipelined processing, as shown in Fig. 7(b), has four major stages, which are Gaussian smoothing, gradient computation, NMS and edge thresholding, respectively. In Fig. 7(b), the numbers (0 to 6) inside the boxes that denote the buffered row correspond to the column index of Fig. 7(a). And the grey boxes with dashed line are the incoming data of each pipeline stage. The detailed implementation of each pipeline stage is explained in the following subsections.

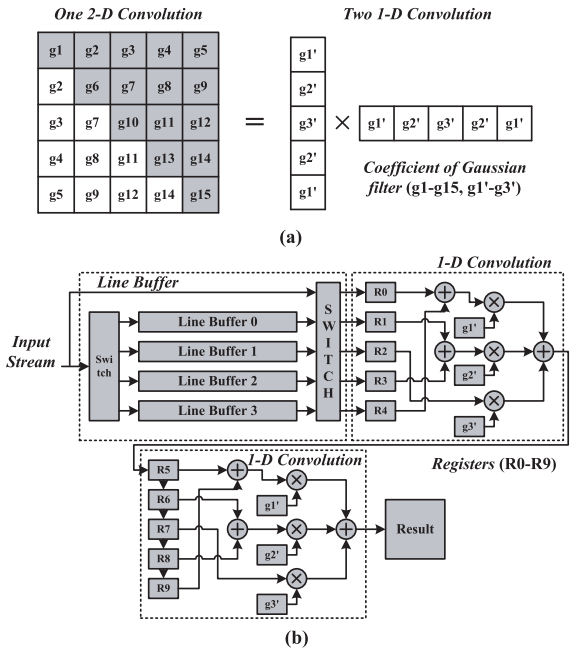

Fig. 8 HW Architecture for Gaussian filter. (a) Gaussian filter template. (b) Hardware structure of Gaussian filter.

4.2 Gaussian Filter Architecture

In the hardware implementation, a sigma (σ) of 1.5 and a kernel size of 5 are adopted for the Gaussian filtering. The hardware structure consists of a rotating line buffer, convolution buffers and two 1D convolution units. The rotating line buffer that consists of four static random-access memory (SRAM) banks is used for the temporary store of the image lines necessary for a 5×5 convolution. As shown in Fig. 8(b), the input image lines are stored in these buffers alternately. The output of the line buffers, along with the incoming image line, are inputted to the respective register (R0-R9) of the convolution buffer through a switching network. The convolution units deal with the arithmetic processing needed to calculate the Gaussian filtering with the 5×5 kernel. As shown in Fig. 8(a), to reduce the occupied logic resources of the FPGA, we utilize the symmetry of the Gaussian filter template, whose coefficients are denoted by g1-g15, to divide the one-step 2D convolution process into two 1D ones, where a pattern is used to convolute the columns first and then the rows. Since the symmetrical columns and rows must be multiplied by the same coefficient (g1'-g3'), they are added together first and then the sum is multiplied by the coefficient. As shown in Fig. 8(b), the number of the multipliers necessary for pipelined processing is thus reduced to 6 from the original 25.

Figure 7(b) illustrates the timing graph of the proposed Gaussian filter. Since the size of the Gaussian kernel is 5×5, it is necessary to store four image lines before the first convolution starts. It is noted that an additional two lines padded with zero are extended to the first and the last

lines respectively to ensure the 5×5 sliding window cover all the image lines. And thus, the first Gaussian filtering starts when the 3rd line (indexed by 2) comes, and the results are passed to the next stage, gradient computation, as input. The incoming pixel, on the one hand, is feed to the convolution buffer through the switching network, and on the other hand, is fetched to replace the unused data in the line buffer. The same column of the four line buffers are fetched to the convolution buffer simultaneously for the 1D convolution.

4.3 Gradient Calculation Architecture

Figure 9 shows the hardware structure for calculating the gradient of each image pixel. It consists of a rotating line buffer, a window buffer and an arithmetic unit for 3×3 Sobel operator. In the rotating line buffer, there are two SRAM banks for the temporary store of the image lines necessary for the 3×3 convolution. The input image lines are stored in these line buffers alternately. The output of the two line buffers, along with the incoming image line, are inputted to the respective row of the window buffer through the switching network. The window buffer, which is a 3×3 shift register array, stores the minimum size of image data necessary for one convolution. It works as a buffer between the rotating SRAM banks and the arithmetic unit. The arithmetic unit is a combinational logic block dealing with the 3×3 Sobel operator.

As an illustration, Fig. 7(b) details the pipelined manner of the proposed gradient calculation architecture. As the size of the Sobel kernel is 3×3, an additional line padded with zero is extended to the first and the last lines respectively to ensure the sliding window cover all the image lines. Then, every clock cycle, the incoming pixel is feed to the window buffer through the switching network, and on the other hand, it is fetched to replace the used data in the line buffer. Meanwhile, the same column of the adjacent two lines that are stored in the line buffers are read and fetched to the window buffer. In this manner, the entire image is scanned by the 3×3 Soble kernel pixel by pixel, resulting in the gradient of each pixel, G_x and G_y, passing to the following stage for NMS and adaptive threshold calculation.

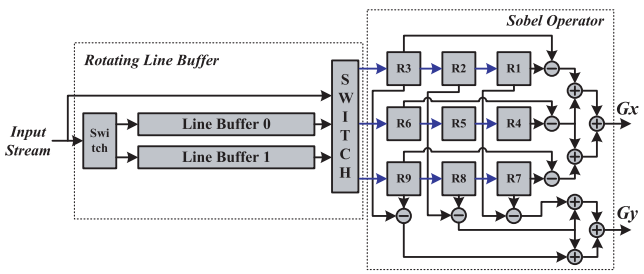

Fig. 9 HW architecture for gradient calculation

4.4 Non-Maximun Suppression Architecture

Figure 10 shows the hardware architecture of the directional NMS, where G_x and G_y are the horizontal and vertical gradient of the input pixel, respectively. According to the proposed algorithm in Sect. 3, the directional NMS unit consists of three components including the gradient magnitude calculator, the direction selector and the gradient magnitude comparator. In the the gradient magnitude calculator, the computation of square root is approximate by the absolute operation in Eq. (8). The direction selector uses the input gradient components G_x and G_y to assign each pixel to one out of the six possible gradient direction. According to the sign and value of the G_x and the G_y, the region of the direction can be determined; and thus the gradient magnitudes of the neighbors (G_1 to G_4 and G_6 to G_9) along the direction are selected and compared with the gradient magnitude of the center pixel (G_5). The pixels whose gradient correspond to a local maximum gradient magnitude are considered to be candidate edge points. Otherwise, the gradient of the pixel that does not correspond to a local maximum gradient magnitude is set to zero.

It is noteworthy that the proposed architecture significantly simplifies the hardware implementation of NMS. First, it uses one addition operation to compute the gradient magnitude, getting rid of the resource-inefficient square root operation. Furthermore, deploying the arctangent operation to calculate the gradient angle costs too much time and resources, while the proposed method uses the magnitude of G_x and G_y to quantize the gradient directions. As shown in

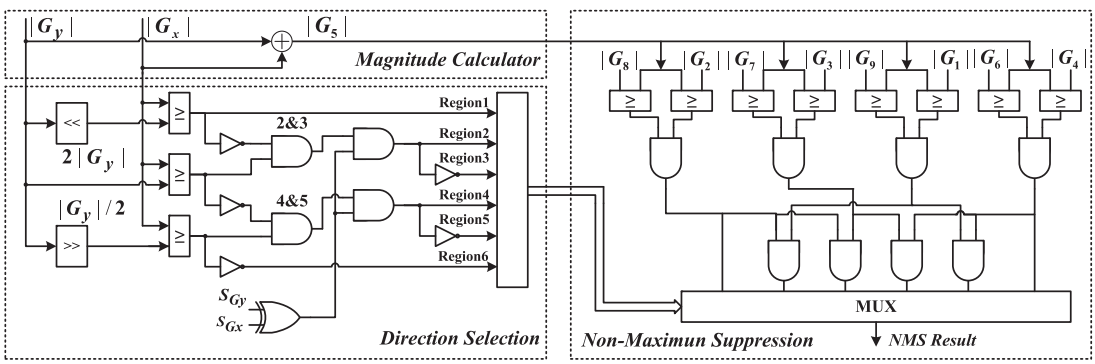

Fig. 10 HW architecture for NMS

Table 3 Hardware resources and latency costed by various computation in NMS algorithms.

	Slices	Slice Reg	LUTs	DSP	Latency
Magnitude (Equ.3)	131	293	269	2	14
Arctan (Equ.4)	586	985	924	0	20
Division (Equ.5)	165	453	251	0	16
Proposed NMS	66	25	239	0	2

Note: The results are gotten by using Xilinx ISE 14.7.

Fig. 10, hardware-friendly operations, such as shift, comparison and addition, are enough, avoiding the arctangent operation for achieving better utilization and speed performance on FPGA. To quantitatively evaluate the advantages of the proposed method from the viewpoint of hardware efficiency, Table 3 compares the hardware resources and latency of the proposed NMS implementation and the hard macros necessary for directly implementing the original NMS algorithm in Eq. (3)–(5). It is obvious that the advantages of the proposed method are twofold: (1) it costs much less hardware resources; (2) it performs much faster.

Figure 7(b) also illustrates the detailed timing of the NMS stage. It works similarly with the gradient calculation stage. There are two buffers for the temporary store of the data lines. Every clock cycle, the incoming data is feed to a 3×3 window buffer through the switching network, and on the other hand, it is fetched to replace the used data in the line buffer. The 3×3 window buffer is adopted for storing neighbors necessary for a NMS operation. Then, the results after directional NMS are passed to the following stage for the final thresholding.

4.5 HW Architecture for Adaptive Threshold Selection

The threshold calculation unit uses the gradient components (G_x and G_y) to generate the threshold for the edge selection. According to the algorithm in Eq. (9), the mean value, $\overline{G_{p(x,y)}}$ and $\overline{G_{p_0(x,y)}}$, and the standard deviation ($\sigma_{p_0(x,y)}$) of the gradient magnitude are necessary for the calculation of the threshold. It is noted that the herein $p(x, y)$ and $p_0(x, y)$ represents different sets of pixels, where the former is the original image and the latter is a subset of $p(x, y)$ filtered by $\overline{G_{p(x,y)}}$. This module works in parallel with the directional NMS unit. And the generated threshold is adopted to filter the contour of the celestial body from the candidates after NMS.

For hardware implementation, the direct calculation of $\overline{G_{p(x,y)}}$, $\overline{G_{p_0(x,y)}}$ and $\sigma_{p_0(x,y)}$ needs at least three iterations, which cost huge storage resources and processing time. Since the image sensor exposes every 200 *ms* and the target planets are quite far from it, the successive images change little. Thus, to avoid the iteration and ease the hardware implementation, we assume the three adjacent images are the same. Accordingly we propose to distribute the computation of the threshold into multiple successive expose period in a pipelined manner. As shown in Fig. 11(a), $\overline{G_{p(x,y)}}$ is first calculated by accumulating the incoming gradient components of the 1st picture, and it is updated once a new picture comes. Starting from the 2nd picture, the resulting $\overline{G_{p(x,y)}}$ of the previous picture is used to filter the current incoming gradient components, and the results after filtering are further accumulated to calculate $\overline{G_{p_0(x,y)}}$. At the 3rd picture, the $\overline{G_{p_0(x,y)}}$ getting in the previous period is further used to calculate the standard deviation $\sigma_{p_0(x,y)}$, and then the 1st threshold can be gotten by using Eq. (9). The circuits for calculating $\overline{G_{p(x,y)}}$, $\overline{G_{p_0(x,y)}}$ and $\sigma_{p_0(x,y)}$ are shown in Fig. 11(b). And starting from the 3rd picture, a new threshold is generated every expose period.

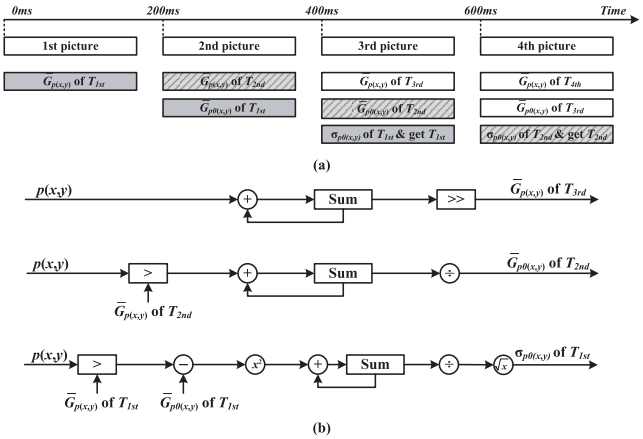

Fig. 11 HW for the adaptive threshold selection. (a) Timing for the calculation of threshold. (b) Circuit for the calculation of threshold.

5. Experimental Results

The effectiveness and performance of the proposed algorithm are validated twofold. First, experiments are performed to validate the effectiveness and performance of the proposed algorithm by utilizing both synthetic and real satellite imageries. Second, the hardware implementation is evaluated in terms of the occupied resource and the real-time processing efficiency.

5.1 Edge Extraction Performance

In order to validate the effectiveness and robustness of the proposed algorithm, we use real images obtained from deep-space flight missions and synthetic simulated images. The real raw images used here include a Jupiter image (PIA04866), a Mars image (PIA04591), a Mercury image (PIA10398) and a Pluto image (PIA09113). The synthetic image of the celestial body is also generated with fixed Gaussian noise added. The edge points of the real and synthetic images are extracted by using Canny edge extraction algorithm adopted in [15] and our proposed algorithm. Figure 12 shows the edge extraction results of the real image. The results indicate that the detected edges of celestial bodies using the proposed is thinner than those of the traditional Canny method. Moreover, the noise edges caused by craters and atmosphere covered on celestial bodies are also sup-

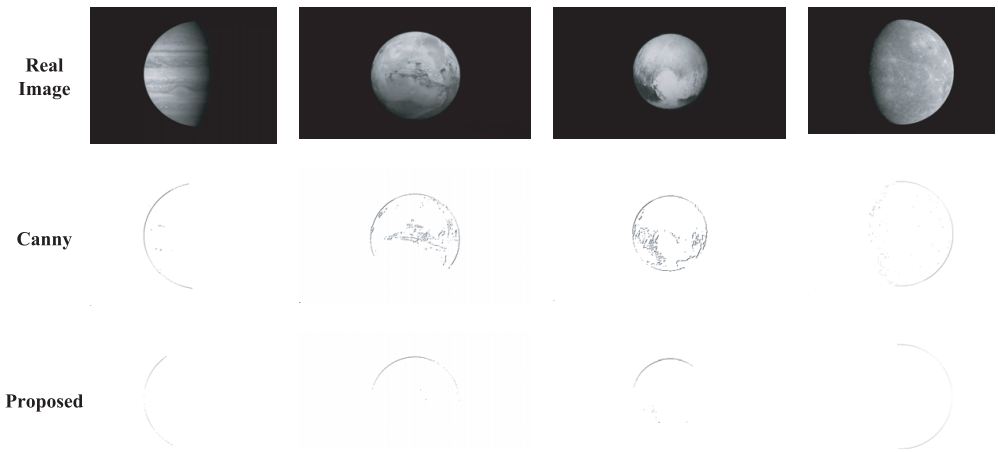

Fig. 12 Simulation results of the proposed algorithm and the typical Canny algorithm

pressed using the proposed method.

To quantitatively measure the performance of the proposed algorithms, we adopt the following measures: (1) Noise-to-Signal Ratio (NSR); (2) Edge extraction error; (3) Edge width in pixel. The NSR is defined as the ratio of the number of detected edge points that do not coincide with the ideal edge to those that coincide with the ideal edge. In the simulated images, the actual centroid position and apparent radius of the planet are accurately known in advance. In the real images, the outline of the planet image is assumed as round. And thus, the centroid position and apparent radius of the planet of the planet can be fitted by Hough Transform and the Least Square Circle fitting algorithm. The ideal edges of the real images are assumed to be the edge points within the width of ±1 *pixels* from the fitted circle. For the edge extraction error, we adopt the root mean square error (RMSE) proposed in [18]. After manually removing the false edges (the edge points beyond the width of ±5 *pixels* from the fitted circle), the edge extraction accuracy of the proposed algorithm is evaluated by calculating the absolute deviation of the distance between the extracted edge point and the planet centroid with respect to the theoretical apparent radius.

Table 4 and Table 5 shows the overall performance of the two algorithms applied to the four real images and the synthetic image. The proposed algorithm achieves a significantly better edge extraction performance. First, the NSR of the proposed algorithm is over one order of magnitude better than the other algorithm. This indicates the proposed algorithm can efficiently suppress the noise inside the celestial body. After removing the false edges inside the celestial body, the RMSE further shows the detected edges of the proposed algorithm is over two times more accurate than those of the conventional algorithm. Moreover, the proposed algorithm can extract continuous one-pixel wide edges from the real images, whereas the edges extracted by the Canny algorithm are multi-pixel wide and numerous false edges are also extracted. Therefore, the proposed algorithm can extract edge points with high accuracy and achieve better feature extraction performance in terms of qualitative and quantitative measurements.

Table 4 Overall performance evaluation results of Canny algorithm.

	Jupiter	**Mars**	**Mercury**	**Pluto**	**Synthetic**
NSR	12.10%	61.51%	34.38%	62.14%	4.69%
Edge width	Multi	Multi	Multi	Multi	Multi
RMSE	1.561	1.507	0.752	2.134	0.5165

Table 5 Overall performance evaluation results of proposed algorithm.

	Jupiter	**Mars**	**Mercury**	**Pluto**	**Synthetic**
NSR	0%	3.02%	0.35%	9.29%	0.60%
Edge width	One	One	One	One	One
RMSE	0.458	0.438	0.470	1.068	0.3228

It is noted that, as shown in Fig. 12, the proposed algorithm may remove part of the real edges and degrade the connectivity of the edges when compared with the original Canny algorithm. This is mainly caused by the different threshold segmentation methods. In the original Canny algorithm, the purpose of the threshold is to remove fake edges. However, in our proposed method, the threshold is required to identify the edge points exactly on the *ring* of the celestial body while suppressing the other edges as much as possible. Thus, to get thinner edges and suppress noise edges inside the celestial body, it loses a part of the real edges during the threshold segmentation. However, this penalty is acceptable when extracting the planet centroid using star centroiding algorithms, e.g., ellipse fitting [4] and Gaussian fitting [5], and the effectiveness of the proposed method is also confirmed with the performance evaluation above.

Unlike pixel-level edge detectors used in this work and [15], sub-pixel edge detection algorithms are adopted in [16], [18], [19] for more accurate OPNAV. Specifically, [16] uses Prewitt-Zernike moment and [18], [19] use the second-order Taylor expansion [23] to determine the sub-pixel location of the edge points. They are certainly more accurate in detecting edges, while this in turn becomes several orders

Table 6 Comparisons of various FPGA implementation results.

Xilinx	FPGA Resources: Occupied Slices	Slice Reg.	Slice LUTs	DSP	**Max. Freq.** (MHz)	**Time***
[11]	2,988	5,080	10,312	28	250	3.325 ms / 185.48 μs
[13]	/	4,810	4,437	8	100	2.647 ms / 25.76 μs
Proposed	2,511	4,270	3,813	26	250	2.642 ms / 20.60 μs
Altera	**FPGA Resources: Adaptive Logic Modules**	**Registers**	**DSP**		**Max. Freq.** (MHz)	**Time***
[12]	3,406	6,608	28		242	2.663 ms / 41.33 μs
Proposed	3,167	2,194	1		250	2.642 ms / 20.60 μs

Note: PL / IL of processing a 512×512 image at 100MHz.

of magnitude more complex than the proposed one. Considering the constrained computation and energy capacity on-board, light-weight algorithms with less computational complexity are desirable for real-time OPNAV. Moreover, it has been found pushing more accuracy to select better pixels is unnecessary, because the final accuracy is provided by the results of nonlinear least-squares estimation [17]. Thus, to avoid computationally-intensive approaches, pixel-level edge detector is here adopted to select the pixels of interest.

Table 7 Number of used line buffers.

	Proposed	[12]	[13]
Gaussian filter	4	5	4
Gradient computation	2	3	2
NMS	2	3	2
Thresholding	0	3	2
Total line buffer*	8	14	10

Note: depth of each line buffer equals the width of the image.

5.2 Hardware Performance

To evaluate the proposed hardware architecture, we have implemented a complete edge detection system using Xilinx Virtex-4 FPGA and ISE design suite 14.7. The optical sensor adopted is CMOSIS CMV20000 with a resolution of 5120×3840. This sensor generates 12-bit grayscale images which are processed by the proposed hardware in a completely stream-in-stream-out manner.

Table 6 lists the hardware resources and execution time of various Canny edge detectors implemented with both Xilinx and Altera FPGAs. For the fair comparison, the execution time is measured using a 512×512 image working at 100 MHz, although the proposed implementation can support image resolutions of up to 5120×3840. Compared with the previous work, the proposed Canny edge detector uses a smaller number of logic elements with less execution time. In more detail, [11] adopts distributed processing manner that allows parallel processing with multiple computing engines. However, to support block level edge detection, it suffers more hardware resources and execution time. Both [12] and [13] use the completely pipelined architecture. However, thanks to the hardware-friendly NMS and thresholding method, the proposed design shows significant area efficiency compared to [12]. Comparing with [13], our proposed design costs more DSP blocks, since [13] uses fixed threshold approach, while the proposed one uses adaptive threshold approach, which complicates the computation but enables better performance in edge detection. In more detail, the DSP blocks adopted in our design are all consumed by calculating the adaptive threshold illustrated in Sect. 4.5. In Xilinx FPGA, 26 DSP blocks are used, one for square operation and the other 25 ones for the divider. In Altera FPGA, due to its divider IP does not cost DSP resource, only one DSP block is used for the square operation. Thus, ignoring the DSP and logic resources costed for adaptive threshold calculation, the proposed implementation should be much more resource-efficient than [13]. Moreover, considering the latency, our design calculates the threshold while doing the NMS, which hides the processing latency. And thus, it shows performance improvements in execution time compared with [12] and [13], although all of them work in the same pipelined stream-in-stream-out manner.

In the context of high-resolution satellite imageries, block random-access memories (BRAMs) used to buffer the image lines are the most critical resource in FPGA devices, especially for completely pipelined architectures. Since the data width and buffer depth vary among different applications, we compare the number of line buffers adopted in the architectures. As listed in Table 7, the proposed design shows BRAM reduction of 43% and 20%, respectively, compared with [12] and [13], making the remaining memory have enough space for further processing. In general, the streaming calculation of threshold requires no line buffer, making our method resource-efficient.

Using Xilinx Virtex-4 FPGA, the proposed architecture can work with a maximum frequency of 250 MHz. In the practical implementation, it works at 120 MHz which is the rate of the pixel stream coming from the image sensor. In terms of real-time performance, we use the Initial Latency (IL) and the Processing Latency (PL) proposed in [12] to measure. The IL is defined here as the amount of time between when a pixel arriving at the system and when it coming out after all the processing finished. It also corresponds to the time expended to fill the pipeline, and Fig. 13 shows the IL of the proposed pipelined architecture. The PL is defined here as the sum of the IL and the time to process all the pixels of an image. It indicates the longest period of the

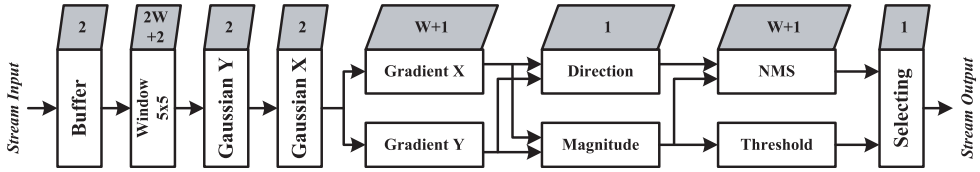

Fig. 13 Processing latency of each pipelined stage

Table 8 Comparison of timing performance for the edge detection implementations

Image resolution	GPU [7]	Proposed FPGA
512×512	3.40 *ms*	1.06 *ms*
1024×1024	10.92 *ms*	4.21 *ms*
2048×2048	31.46 *ms*	16.81 *ms*
3936×3936	96.62 *ms*	62.03 *ms*
5120×3840	/	78.73 *ms*

*FPGA working at 250 MHz.

whole algorithm. Since the pixel stream has a constant rate, we can express IL and IP in terms of pixels using Eq. (10) and (11), where W and H denote the width and height of the images, respectively.

$$IL = 12 + 4 \times W \tag{10}$$

$$PL = W \times H + (12 + 4W) \tag{11}$$

Table 8 shows that the proposed FPGA also has an evident advantage over the GPU implementation proposed in [7]. The FPGA speedup factor for the GPU implementation varies from 1.55 to 3.21. It is noted that, for larger images however, the FPGA speedup decreases in function of the image resolution. This is due to the high number of CUDA cores of GPU, which enables a massively parallel processing with high definition images. Regarding the target image (5120×3840) of this work, a single edge detection takes 78.73 *ms* at 250 MHz, satisfying the real-time image processing speed requirement for OPNAV.

6. Conclusion

In this paper, an algorithm-hardware co-design of real-time edge detection is proposed for deep-space autonomous optical navigation. In terms of the algorithm, we modify the traditional Canny algorithm by proposing an optimized adaptive thresholding approach to efficiently detect the edge of the celestial body while suppressing the noise edges caused by surface markings. Moreover, hardware-oriented optimization is also carried out to the NMS and threshold selection for easing the hardware implementation without sacrificing the accuracy. Based on the proposed algorithm, we further propose a completely pipelined hardware architecture with optimized storage and computation structure for improving its real-time performance and the resource efficiency. Finally, the proposed solution has been implemented and verified on FPGA. Experimental results confirm our proposed solution performs effectively with high accuracy, low processing latency and efficient on-chip resources, making it a viable option for real-time edge detection in OPNAV.

Acknowledgments

This work is supported in part by National Natural Science Foundation of China 61974039, 61834006, 61504059, 61674049, U19A2053, the Aeronautical Science Foundation of China 2018ZCP4, the Fundamental Research Funds for Central Universities JZ2019HGTB0092, and the Project of Science and Technology on Electronic Information Control Laboratory.

References

[1] T. Teil and H. Schaub, "Software architecture for close-loop autonomous optical navigation scenarios," Proc. 1st Annual RPI Workshop on Image-Based Modeling and Navigation for Space Applications, New York, USA, pp.1–10, June 2018.
[2] X. Ma, J. Fang, and X. Ning, "An overview of the autonomous navigation for a gravity-assist interplanetary spacecraft," Prog. Aerospace Sciences, vol.63, no.11, pp.56–66, 2013.
[3] M. William and J. Owen, "Methods of optical navigation," Advances in the Astronautical Sciences, no.140, pp.1635–1653, 2011.
[4] J.A. Christian and E.G. Lightsey, "Onboard image-processing algorithm for a spacecraft optical navigation sensor system," J. Spacecraft and Rockets, vol.49, no.2, pp.337–352, 2012.
[5] T. Delabie, J.D. Schutter, and B. Vandenbussche, "An Accurate and Efficient Gaussian Fit Centroiding Algorithm for Star Trackers," J. Astronautical Sciences, vol.61, no.1, pp.60–84, 2014.
[6] J.F. Canny, "A computational approach to edge detection," IEEE Trans. Pattern Anal. Mach. Intell., vol.8, no.6, pp.679–698, Nov. 1986.
[7] Y. Luo and R. Duraiswami, "Canny edge detection on NVIDIA CUDA," Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Anchorage, USA, pp.1–8, June 2008.
[8] A. Alaghi, C. Li, and J.P. Hayes, "Stochastic circuits for real-time image-processing applications," Proc. ACM/EDAC/IEEE Design Automation Conf. (DAC), Austin, USA, no.136, June 2013.
[9] W. He and K. Yuan, "An improved Canny edge detector and its realization on FPGA," Proc. 7th World Congress on Intelligent Control and Automation, Chongqing, China, pp.6561–6564, June 2008.
[10] C. Gentsos, C. Sotiropoulou, S. Nikolaidis, and N. Vassiliadis, "Real-time canny edge detection parallel implementation for FPGAs," Proc. 17th IEEE Int. Conf. Electronics, Circuits and Systems, Athens, Greece, pp.499–502, Dec. 2010.
[11] Q. Xu, S. Varadarajan, C. Chakrabarti, and L.J. Karam, "A distributed Canny edge detector: algorithm and FPGA implementation," IEEE Trans. Image Process., vol.23, no.7, pp.2944–2960, July 2014.
[12] P.R. Possa, S.A. Mahmoudi, N. Harb, C. Valderrama, and P. Manneback, "A multi-resolution FPGA-based architecture for real-time edge and corner detection," IEEE Trans. Comput., vol.63,

no.10, pp.2376–2388, Oct. 2014.
[13] F. Zhou, Y. Cao, and X. Wang, "Fast and resource-efficient hardware implementation of modified line segment detector," IEEE Trans. Circuits Syst. Video Technol., vol.28, no.11, pp.3262–3273, Nov. 2018.
[14] J. Lee, H. Tang, and J. Park, "Energy efficient Canny edge detector for advanced mobile vision applications," IEEE Trans. Circuits Syst. Video Technol., vol.28, no.4, pp.1037–1046, April 2018.
[15] S. Li, R. Lu, L. Zhang, and Y. Peng, "Image processing algorithms for deep-space autonomous optical navigation," J. Navigation, vol.66, no.4, pp.605–623, July 2013.
[16] S. Du, M. Wang, X. Chen, and Shenghui Fang, "A high-accuracy extraction algorithm of planet centroid image in deep-space autonomous optical navigation," J. Navigation, vol.69, no.4, pp.828–844, July 2016.
[17] D. Mortari, F. Dilectis, and R. Zanetti, "Position estimation using the image derivative," MDPI Aerospace, vol.2, no.3, pp.435–460, 2015.
[18] J. Jiang, H. Wang, and G. Zhang, "High-accuracy synchronous extraction algorithm of star and celestial body features for optical navigation sensor," IEEE Sensors J., vol.18, no.2, pp.713–723, 2018.
[19] H. Wang, J. Jiang, and G. Zhang, "Celestial object imaging model and parameter optimization for an optical navigation sensor based on the well capacity adjusting scheme," MDPI Sensors, vol.17, no.4, pp.1–23, 2017.
[20] X. Hao, F. Yanming, Z. Zhang, and X. Cheng, "A Fast and Accurate Edge Detection Algorithm for Real-Time Deep-Space Autonomous Optical Navigation," Proc. 10th IEEE Int. Conf. Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Metz, France, pp.601–604, Sept. 2019.
[21] R. Szeliski, Computer Vision: Algorithms and Applications, Springer, 2011.
[22] T.B. Moeslund, Introduction to Video and Image Processing, Springer, 2012.
[23] C. Steger, Unbiased Extraction of Curvilinear Structures from 2D and 3D Images, Ph.D. Dissertation, Technischen Universität München, Munich, Bavaria, Germany, 1998.

Hao Xiao received the B.E. degree from Zhejiang University, Hangzhou, China, in 2005, the M.S. degree from Fudan University, Shanghai, China, in 2009, and the Ph.D. degree from Tokyo Institute of Technology, Tokyo, Japan, in 2012. From 2012 to 2014, he worked at Tokyo Institute of Technology as a research fellow. He was a lecturer with Nanjing University of Aeronautics and Astronautics from 2014 to 2017. Since November 2017, he has been a Full Professor with the School of Microelectronics in HeFei University of Technology. His research interests include energy-efficient VLSI system design, computer vision, and hardware security.

Yanming Fan received his B.E. degree from Nanjing University of Aeronautics and Astronautics in 2017. Since 2017, he has been studying for his M.S. degree in the College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics. His research interests include image signal processing, deep neural network acceleration and processing-in-memory architectures.

Fen Ge received the B.E., M.E. and Ph.D. degrees from Nanjing University of Aeronautics and Astronautics in 2003, 2006 and 2010, respectively. She is currently an Associate Professor at College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics. Her research interests include network-on-chip and reconfigurable computing.

Zhang Zhang received the B.S. degree in electronic science and technology from Hefei University of Technology, Hefei, China, in 2004, and the Ph.D. degree in microelectronics from Fudan University, Shanghai, China, in 2010. He was a Visiting Scholar in Georgia Institute of Technology from 2016 to 2017, and he has been a Full Professor in the School of Electronic Science and Applied Physics, Hefei University of Technology since 2019. His current research interests include mixed signal/power management circuits design and ultra-low power biomedical circuits design. He is an IC design professional member of China Computer Federation.

Xin Cheng received the B.S. and M.S. degrees in microelectronics from Hefei University of Technology, Hefei, China, in 2006 and 2009 respectively, and the Ph.D. degree in microelectronics from the Institute of Electronics, Chinese Academy of Sciences, Beijing, China, in 2012. She is currently an Associate Professor in the School of Electronic Science and Applied Physics, Hefei University of Technology. Her research interests include integrated circuit design and power electronics.