J Real-Time Image Proc (2014) 9:127–140
DOI 10.1007/s11554-012-0285-2

SPECIAL ISSUE

VLSI implementation of star detection and centroid calculation algorithms for star tracking applications

Mohsen Azizabadi · Alireza Behrad · M. B. Ghaznavi-Ghoushchi

Received: 27 January 2012 / Accepted: 10 October 2012 / Published online: 27 October 2012

Abstract Nowadays, hardware implementation of image and video processing algorithms on application specific integrated circuit (ASIC) has become a viable target in many applications. Star tracking algorithm is commonly used in space missions to recover the attitude of the satellite or spaceship. The algorithm matches stars of the satellite camera with the stars in a catalog to calculate the camera orientation (attitude). The number of stars in the catalog has the major impact on the accuracy of the star tracking algorithm. However, the higher number of stars in the catalog increases the computation burden and decreases the update rate of the algorithm. Hardware implementation of the star tracking algorithm using parallel and pipelined architecture is a proper solution to ensure higher accuracy as well as higher update rate. Noise filtering and also the detection of stars and their centroids in the camera image are the main stages in most of the star tracking algorithms. In this paper, we propose a new hardware architecture for star detection and centroid calculation in star tracking applications. The method contains several stages, including noise smoothing with fast Gaussian and median filters, connected component labeling, and centroid calculation. We introduce a new and fast algorithm for star labeling and centroid calculation that needs only one scan of the input image.

M. Azizabadi · A. Behrad (✉) · M. B. Ghaznavi-Ghoushchi
Faculty of Engineering, Shahed University, Tehran, Iran
e-mail: behrad@shahed.ac.ir

M. Azizabadi
e-mail: m.azizabadi@shahed.ac.ir

M. B. Ghaznavi-Ghoushchi
e-mail: ghaznavi@shahed.ac.ir

Keywords Star detection and labeling · Centroid calculation · Star tracking · Application specific integrated circuit (ASIC) · Hardware implementation

1 Introduction

Star trackers are widely used systems in many space-based applications and satellites. Star tracking algorithms determine the attitude of satellite or spaceship by matching stars of the satellite camera with the stars in a catalog. Although other sensors like GPS, sun tracker, and magnetometer may be used for attitude determination, star trackers have the advantage of producing higher accuracy. They also have the capability of attitude determination without prior information, which is known as the Lost-In-Space (LIS) capability.

Different algorithms have been proposed for the task of star tracking. Graph matching algorithm is the most commonly used approach for star tracking. In this approach, stars are considered as vertices of an undirected graph G, which the angular separation between each pair of stars is the edge weights. Triplet algorithm [1, 2] is the well-known algorithm of this group, where the three angular separations between three stars or triplet vertices are used for star identification. In [3] the angular separations of more than three stars were used for star matching or star identification. The algorithm generates match groups through star pair matching. Then smaller match groups are removed and the remaining groups are validated by checking distances between all stars in the group, and finally a group with the largest number of matches is selected as the true match. The algorithm of Kolomenkin et al. [4] is based on a geometric voting scheme. In this approach, a pair of stars in the camera image votes for star pairs in the catalog if the angular distance between the stars of both pairs is similar. Then the star pair with the most votes is selected as the matched pair. Finally, the attitude of the camera is recovered using a quaternion-based method.

The grid-based or pattern recognition approaches are another group of star tracking algorithms, which employ the information of entire stars in the field for star identification [5, 6]. These methods employ a rotation-compensated bitmap of all stars in the neighborhood to form a pattern for identification.

The star pattern recognition method of [7] was based on the singular value decomposition of a measured unit column vector matrix in a measurement frame and the corresponding cataloged vector matrix in a reference frame. It was shown that singular values are invariant with respect to coordinate transformation; therefore, rotation compensation stage was not required.

Star tracking algorithms generally include two stages: 1-star detection and centroiding and 2-star matching. In LIS mode of operation, stars of input image are matched with stars of entire sky catalog. However, in normal mode of operation, a restricted number of catalog stars are used for matching. The number of stars in input image and the catalog of stars have a major effect on the accuracy of star tracking algorithms. However, the increase in the number of stars increases the computation overhead of the algorithm. Consequently, the update rate and accuracy of the algorithm are decreased especially in normal mode of operation.

Although guide star selection is used in some of the star tracking algorithms to overcome the problem of high number of stars in the catalog, the efficiency of the algorithm is reduced [8, 9]. Hardware implementation of the star tracking algorithm is a proper solution to ensure both speed and accuracy of the algorithm. The speed of star detection and centroiding algorithm has a major impact on the update rate of the star tracking algorithm, especially in the normal mode of operation.

In this paper, we present a new hardware architecture for star detection and centroid calculation in star tracking applications. The proposed algorithm detects stars and calculates their centroids in only one image scan. Therefore, the memory usage and the latency of the algorithm are considerably reduced.

The organization of the paper is as follows: in Sect. 2, we briefly present some of the research literature related to hardware implementation of image filtering, connected component labeling, and star tracking algorithms. Section 3 introduces the proposed algorithm for stars labeling and centroid calculation. Section 4 presents the results of simulation and synthesis. Finally, the conclusions appear in Sect. 5.

2 Related work

Recently, hardware implementation of image processing algorithms has emerged as the most viable solution for improving the performance and speed of the different image processing algorithms [10, 11]. Hardware implementation of the star tracking algorithm using parallel and pipelined architecture is also a proper solution to ensure higher accuracy and higher update rate.

The first step in star tracking algorithms is the detection of stars in the camera image. The main challenge in this stage is to remove spurious spikes, which may be due to proton-induced effects on the image sensor [12] or other objects in the sky like other satellites. To handle the problem of spurious spikes, it is necessary to employ proper filtering stage before star detection. Generally, the filtering stage includes a median filter to remove outliers and a low-pass filter like Gaussian filter to smooth the input image.

Different architectures have been proposed for hardware implementation of image filtering, most of which focused on the implementation of median filtering. In [13] a hardware architecture was proposed for the real-time implementation of a sliding 3×3 median filter. Hu and Ji [14] proposed two different architectures for hardware implementation of median filtering including standard and multi-level median filters. The pipeline architecture of multi-level median filter had four clock cycles of latency in comparison with the six clock cycle of latency for the standard median filter. However, multi-level implementation did not ensure the exact calculation of median value for all intensity values. In [15] an optimized systolic array was utilized for hardware implementation of median filtering. The pipeline structure of this architecture included seven stages, where each stage was executed in one clock cycle.

After applying the necessary filtering stages, the input image is converted to binary image by employing a thresholding algorithm. Subsequently, stars in the image are extracted using a connected component labeling method. To match stars in the input image with the stars in the catalog, it is also necessary to calculate the star centers using a centroiding algorithm.

Several software implementations have been proposed for labeling of the connected components in an image. These approaches are divided into five main classes of one-scan, two-scan, multi-scan, hybrid and tracing-type algorithms [16]. In one-scan algorithm, pixels of every object are determined concurrently with image scan and a unique label is assigned to every object. In [17] a one-scan algorithm was proposed for connected component labeling. The algorithm scanned the image to find an unlabeled pixel, and then the neighborhood of the pixel was recursively searched to find all pixels of the object. The recursive search of the algorithm made the algorithm unsuitable for hardware implementation.

In two-scan algorithms, image is scanned twice. In first scan provisional labels are assigned to objects and in second scan equivalent labels are merged. In multi-scan algorithms, image is scanned several times in the forward and backward directions to extract labels and merge equivalent labels. Hybrid algorithms scan the image several times in the forward and backward directions like multiple-scan algorithm. However, similar to two-scan algorithms a table is employed to resolve the labeling equivalency. Tracing-type algorithms generally use iterative and recursive operations to trace connected components or contours of objects and avoid the analysis of label merging.

Although multi-scan approaches are generally efficient for connected component labeling using software implementation, the hardware implementation of multi-scan algorithms increases the memory usage and pipeline latency extremely. Therefore, connected component labeling with less image scan is preferable for hardware implementation. Existing approaches for labeling connected components using hardware architectures generally use two-scan approaches [18, 19]. In these architectures, provisional labels are assigned to various pixels in first scan and simultaneously, a lookup table is generated for label equivalency. In second scan, equivalent labels are merged using the table.

Application of two-scan labeling approaches for the star detection and centroid calculation needs three image scans. Two image scans are required for the star detection and labeling and one image scan is used for centroid calculation. This approach increases memory usage and pipeline latency for the hardware implementation.

In [20], a combined hardware and software implementation of a star tracking algorithm was proposed to increase the update rate of the star sensor. The algorithm implemented only the star segmentation and centroiding with FPGA. The star segmentation algorithm [21] used four-connectivity to detect stars, which made the label merging simpler with the cost of losing generality.

In [22] anterior image acquisition and pre-processing hardware system for a star sensor were described. The system is based on CMOS image sensor and FPGA technology. The paper implemented only pre-processing stage for the star sensor.

In this paper, we present a new hardware architecture for star detection and centroid calculation in star tracking applications. We use a new algorithm to detect stars and calculate centroid, where eight-connectivity is used for the segmentation. The proposed algorithm is implemented in two stages. The first stage is used for pre-processing, which includes median and Gaussian filters. The second stage of the algorithm detects stars and calculates stars' centers. The proposed algorithm detects stars and calculates centroids in only one image scan. Therefore, the memory usage of the algorithm is considerably reduced. To implement the proposed one-scan algorithm, we employ tables to store star labels and the necessary information for centroid calculation. By merging star labels, all the tables are updated concurrently with the image scan.

3 Proposed architecture for star detection and centroiding

The proposed algorithm for star detection and centroid calculation is a one-scan algorithm that segments stars and calculates centroids. Figure 1 shows the general block scheme of the proposed architecture for one-scan star labeling, equivalent label merging, and centroid calculation. The architecture contains several units, including (1) filtering unit, (2) labeling and memory unit, (3) control unit, and (4) centroid calculation unit. Filtering unit applies median and Gaussian filters to remove spike noise and image smoothing. Labeling and memory unit receives image pixels after applying filtering stages. The unit applies a threshold to detect stars' pixels and assigns labels. In star tracking applications, it is only necessary to calculate star centers; therefore the proposed architecture merges star labeling and the centroid calculation to obtain a one-scan algorithm. For this purpose, labeling and memory unit stores the necessary information to calculate stars' centers concurrently with star labeling. When a star pixel is detected, its neighboring pixels are checked to assign a previously defined or a new label to the detected pixel. It may be necessary to merge some previously defined labels as well. When a pixel is assigned to a label or two labels are merged, the information to calculate the star center and necessary tables' entries are updated simultaneously. After receiving all the image pixels, outputs of the memory unit are made valid and the centroid calculation module is activated to calculate stars' centroids.

3.1 Filtering unit

The filtering unit is employed to remove spurious spikes and smooth the input image. The proposed filtering stage includes median filter to remove outliers and subsequently, a Gaussian filter is applied to smooth the input image and increase the centroiding accuracy.

To implement the Gaussian filter, a set of registers and First In First Out (FIFO) blocks are used to provide image data for convolution in a pipelined manner. Registers are used to store the image data for convolution and FIFOs are employed to generate required delays.

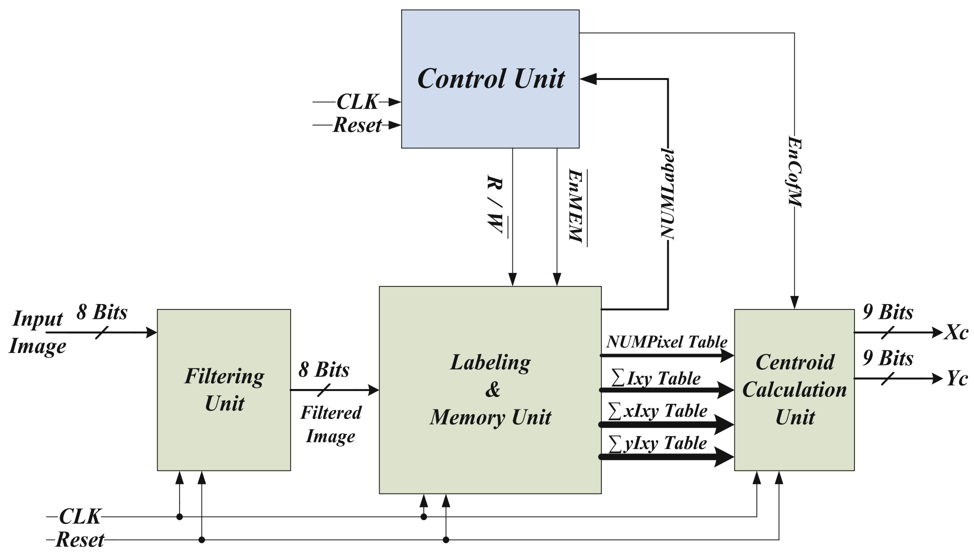


Fig. 1 General block scheme of the proposed architecture for star detection and centroid calculation

Hardware implementation of floating point division is expensive in terms of silicon and needs more clock cycles. Therefore, it is necessary to convert floating point coefficients to fixed point for hardware implementation of Gaussian filter. Two different approaches may be employed for this purpose. In first approach, all filter coefficients are divided by the smallest coefficient of the kernel and results are rounded to integer numbers. In this approach, after applying the convolution operator, the output value is divided by the sum of integer coefficients. Different algorithms may be employed for hardware implementation of fixed point division [23, 24]. Figure 2 illustrates the implemented algorithm for fixed point division. For the implemented Gaussian filtering, the dividend and divisor are 15- and 7-bit numbers, respectively. The division algorithm is completed in nine stages. In each stage, one bit of the quotient is determined by comparing the selected bits of the dividend (m_i) and the divisor. One or several stages of the algorithm may be conducted in one clock cycle based on the critical path analysis.

In [25], filter coefficients were normalized to make their sum as a power of two numbers. Therefore, the division stage is converted to a simple shift operation and hardware cost is reduced. In the section of experimental results, the impacts of two methods on image signal integrity are compared.

The implemented algorithm for hardware realization of the median filtering is based on the method of Maheshwari et al. [13]; however, we use different architecture. The algorithm to apply 3 × 3 median filtering includes the following stages:

- Each column of image data is sorted in the ascending order.
- Each row of the resultant image data is sorted in the ascending order.

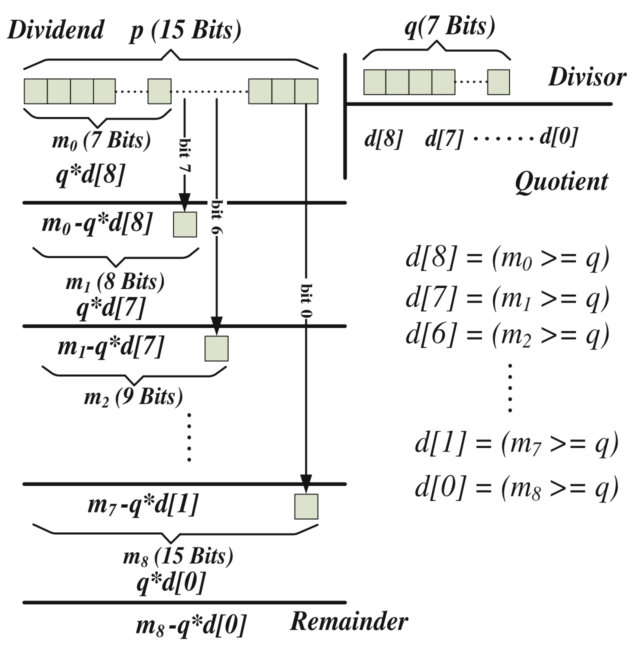


Fig. 2 Different stages of the implemented fixed point division

- Median of resultant diagonal elements is considered as the median value.

Figure 3 illustrates the results of the algorithm in each step to calculate the median value. Figure 4 shows the block scheme of the proposed algorithm for hardware implementation of 3 × 3 median filter. The architecture employs two FIFO blocks and six registers to store the necessary data for filtering. To calculate the median value, four 3 → 1 and one 3 → 3 sorter blocks are also employed to calculate the median value.

The proposed architecture calculates the median value of a 3 × 3 image window in three clock cycles in comparison

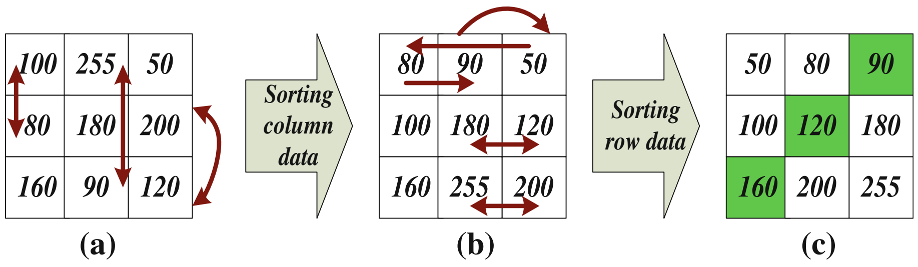


Fig. 3 Different steps for applying median filtering, **a** input image data for 3 × 3 window **b** results after sorting columns data **c** results after sorting rows data. The median of diagonal elements is considered as the median value

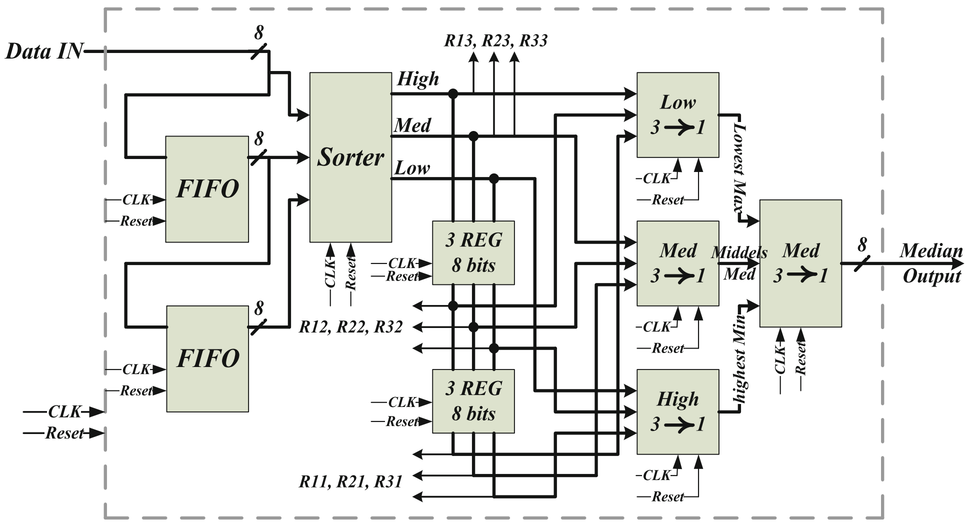


Fig. 4 Block scheme of the proposed algorithm for pipeline implementation of median filtering

with architectures in references [14] and [15], which need six and seven clock cycles, respectively. Furthermore, the proposed architecture in [13] used 3 SRAM modules to implement 3 × 3 median filtering, which is not efficient.

3.2 Labeling and memory unit

The unit receives image pixels from filtering unit and after applying a threshold, segments star pixels to calculate centroid. Figure 5 shows the algorithmic flow diagram of labeling and memory unit. One of the simplest and most used methods to calculate a star centroid is the center of mass for the star, which is defined as follows:

$$x_{cj} = \frac{\sum_{i=1}^{N_j} x_i I(x_i, y_i)}{\sum_{i=1}^{N_j} I(x_i, y_i)} \tag{1}$$

$$y_{cj} = \frac{\sum_{i=1}^{N_j} y_i I(x_i, y_i)}{\sum_{i=1}^{N_j} I(x_i, y_i)} \tag{2}$$

Here N_j is the total number of pixels for jth star, $I(x_i,y_i)$ are the intensity values for different star pixels and (x_{cj}, y_{cj}) is the coordinate of jth star centroid.

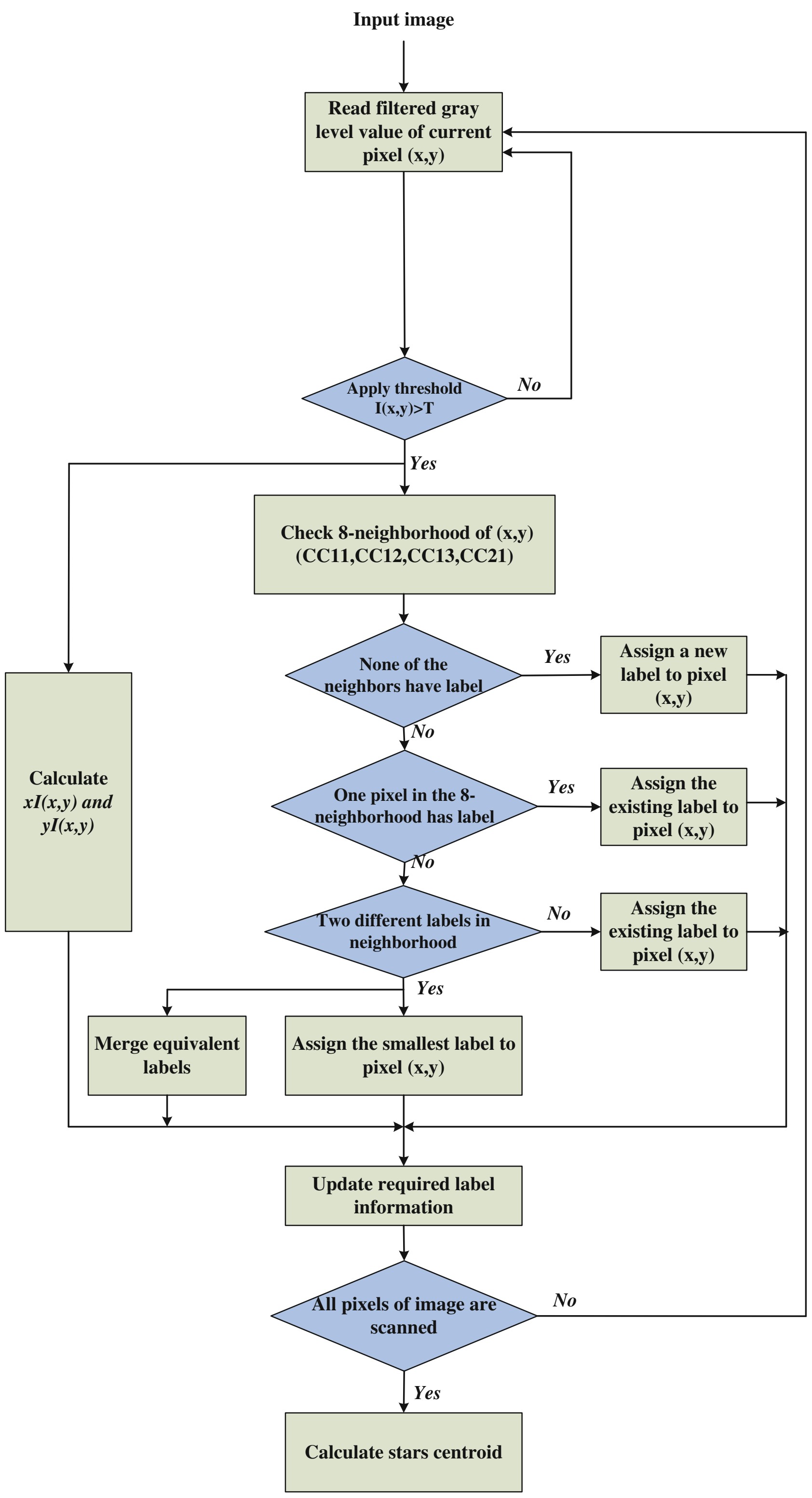
A commercial star tracker like CT601 has the image resolution of 480 × 480 pixels with field of view (FOV) of 8 × 8 degree [26]. The average number of detectable stars in a sky image is given by [27]:

$$N_{\text{Star}} = 6.57 e^{1.08M} \frac{1 - \cos\left(\frac{A}{2}\right)}{2} \tag{3}$$

where M is the visual magnitude (M_v) of detectable stars and A is the image FOV. CT601 sensor is nominally sensitive to stars with instrument magnitude up to 6.0. Therefore, the average number of detectable stars is six for the CT601. For more sensitive star tracker as well as star trackers with larger FOV, the average number of detectable stars may increase. Maximum number of detectable stars is generally considered as a multiple of the average detectable stars number. Based on this information and without loss of generality, in all parts of the proposed architectures in this paper, we assume the standard image size of 320 × 480 pixels with the maximum number of 255 generated labels in an image. The maximum number of pixels for each label is determined based on the largest star in the image. We experimentally use the maximum number of 4,096 pixels for each label.

Figure 6 illustrates the hardware architecture for the pipeline implementation of the labeling and memory unit. Two-scan labeling approaches need two image scans for the star detection and labeling. Furthermore, the calculation of stars' centroids adds another image scan to the algorithm. To handle this problem and obtain an efficient and one-scan algorithm for star detection and centroid calculation, the proposed architecture removes unnecessary stages and utilizes four memory blocks to save necessary

Fig. 5 Algorithmic flow diagram for labeling and memory unit


Input image
Read filtered gray level value of current pixel (x,y)
Apply threshold I(x,y)>T
No
Yes
Check 8-neighborhood of (x,y) (CC11,CC12,CC13,CC21)
None of the neighbors have label
Yes
Assign a new label to pixel (x,y)
No
One pixel in the 8-neighborhood has label
Yes
Assign the existing label to pixel (x,y)
No
Two different labels in neighborhood
No
Assign the existing label to pixel (x,y)
Yes
Calculate xI(x,y) and yI(x,y)
Merge equivalent labels
Assign the smallest label to pixel (x,y)
Update required label information
All pixels of image are scanned
No
Yes
Calculate stars centroid

information for centroid calculation during the image scan. As Eqs. (1) and (2) show, we should calculate the terms $\sum_{i=1}^{N_j} x_i I(x_i, y_i)$, $\sum_{i=1}^{N_j} y_i I(x_i, y_i)$, and $\sum_{i=1}^{N_j} I(x_i, y_i)$, for the centroid estimation. Additionally, most of the star tracking algorithms need the number of pixels for each star. Consequently, four memory blocks are employed to save the terms $\sum_{i=1}^{N_j} x_i I(x_i, y_i), \sum_{i=1}^{N_j} y_i I(x_i, y_i), \sum_{i=1}^{N_j} I(x_i, y_i)$, and the number of pixels for stars.

To reduce the memory usage of the proposed architecture, we do not utilize an additional memory block to store

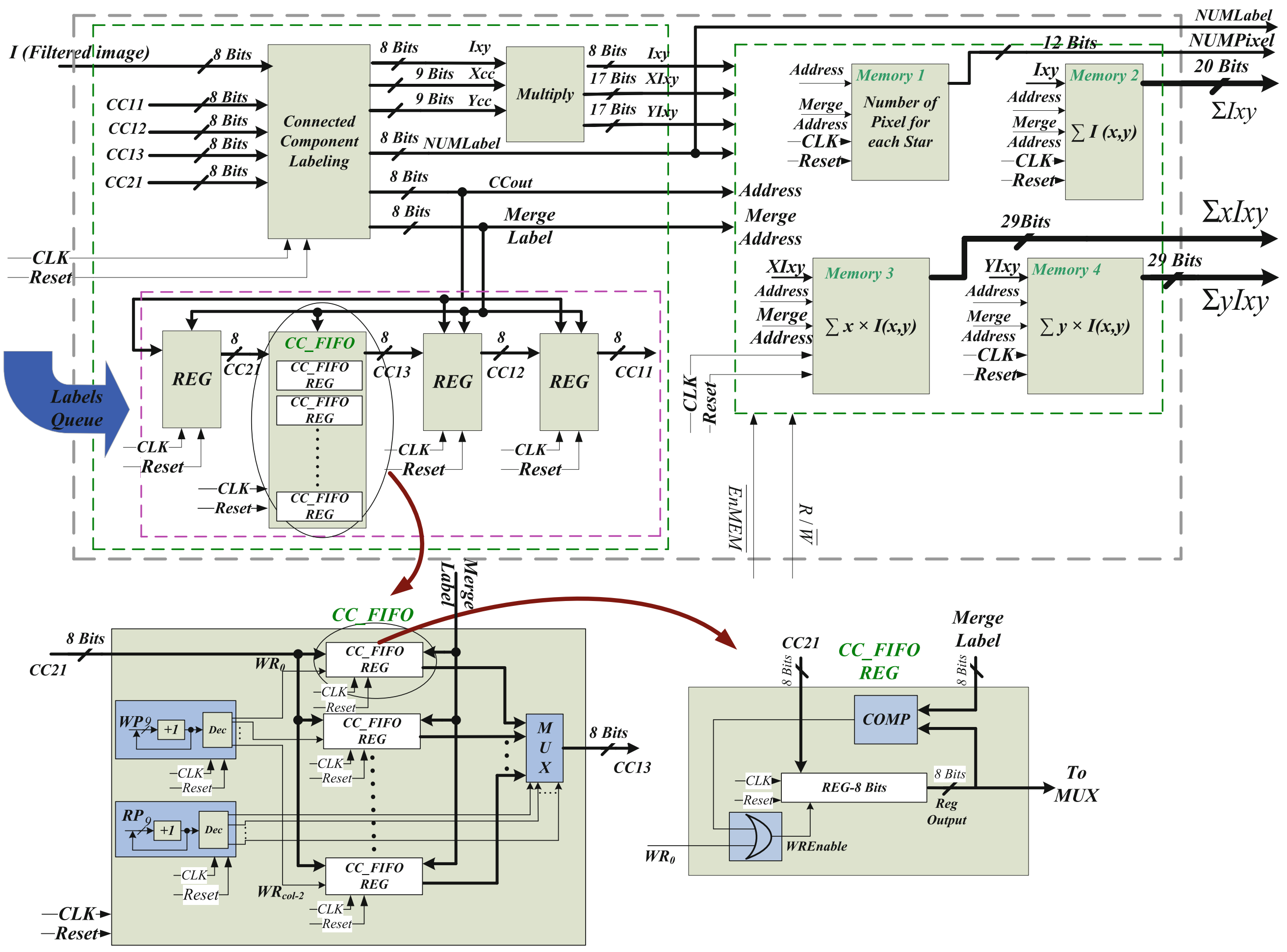


Fig. 6 Hardware architecture for the pipeline implementation of the labeling and memory unit

labels. In this architecture, stars' labels are the addresses of memory blocks for storing centroid data. In our implementation, we utilize the number of pixels for stars as the flag to determine the validity of a memory entry.

To assign a label to a star pixel, its neighboring pixels are checked as shown in Figs. 5 and 7. We merge the equivalent labels when a label equivalency is detected. Therefore, the following three states are expectable in the proposed architecture:

1. There are no predefined labels (all labels are zero) in the neighborhood of the current star pixel. In this case, we assign a new label to the current star pixel, i.e. "*CCout*" and update the memory blocks and number of generated labels, i.e. "*NUMlabel*". "*NUMlabel*" shows the number of generated labels. With the detection of a new label, we increase "*NUMlabel*" value by 1. In this situation, four memory blocks to store the terms $\sum_{i=1}^{N_j} x_i I(x_i, y_i)$, $\sum_{i=1}^{N_j} y_i I(x_i, y_i)$, $\sum_{i=1}^{N_j} I(x_i, y_i)$, and the number of pixels, are initialized by $xI(x,y)$, $yI(x,y)$, $I(x,y)$, and '1' values, respectively. Where (x,y) is the coordinate of current star pixel and $I(x,y)$ is its intensity value. As we mentioned before the label value determines the address of memory entries to store centroid data.
2. There is only one predefined label in the neighborhood of current star pixel. In this case, we assign the same label to the current pixel ("*CCout*"). Then $xI(x,y)$, $yI(x,y)$, $I(x,y)$, and '1' values are added to the content of memory blocks entries with the address of the label.
3. There are two different labels in the neighborhood of the current star pixel. In this case, we assign the smallest label to the current pixel ("*CCout*") and the "*Merge Address*" signal represents the largest label for merging.

Merging two equivalent labels in the proposed algorithm is performed by completing the following actions:

1. Memory blocks' entries with the address of "*Merge Address*" and the current memory inputs are added to entries with the address of "*Address*" *or* "*CCout*".
2. Number of pixels for the memory blocks' entry with the address of "*Merge Address*" is set to zero

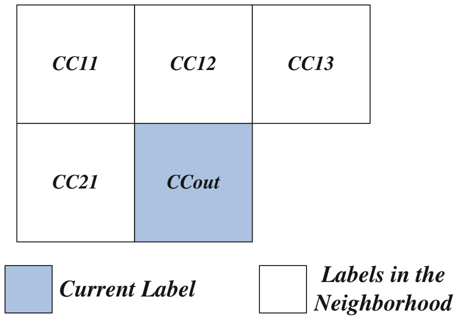


Fig. 7 Labels in the neighborhoods of a star pixel for star labeling

to make the label invalid. To use the memory block more efficiently, we also save "*Merge Address*" in a variable to assign it to the next detected label.

3. All entries in labels queue (registers and the FIFO to generate *CC11* to *CC21*) with the value of "*Merge Address*" are replaced with the "*CCout*" value.

All the actions for merging equivalent labels are performed in only one clock. For this purpose, the hardware architecture of Fig. 8 is utilized to implement memory blocks. The figure illustrates the architecture for the memory block to store $\sum_{i=1}^{N_j} I(x_i, y_i)$ term. As the figure shows, the memory block includes two address and data ports for reading from two separate memory addresses, an address port for clearing a memory content and an address and data port for writing. When no label equivalency is detected ("*Merge Address*" $= 0$ and "*Address*" $\neq 0$), the current input data (*Ixy*) are added to the memory content with the address of "*Address*". In this state, different address ports are set to

$$WRAdd = RDAdd1 = Address \quad (4)$$

$$RDAdd2 = CLRAdd = 0 \quad (5)$$

In the case of label merging ("*Merge Address*" $\neq 0$ and "*Address*" $\neq 0$), two memory contents with the addresses of "*Address*" and "*Merge Address*" are read and added to the current data, i.e. *Ixy*. The summation result is stored in the memory entry with the address of "*Address*". In this state, the memory content with the address of "*Merge Address*" is set to zero. Consequently, different address ports in this state are initialized as follows:

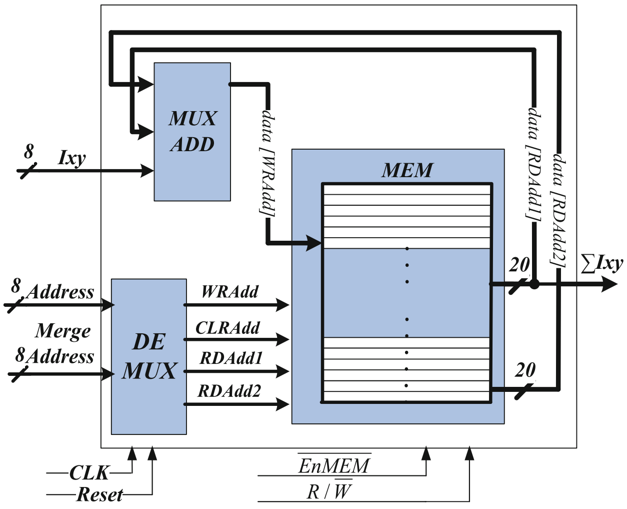


Fig. 8 Hardware architecture for memory blocks with the capability of merging label information in one clock cycle

$$WRAdd = RDAdd1 = Address \quad (6)$$

$$RDAdd2 = CLRAdd = Merge\,Address \quad (7)$$

Tables 1 and 2 summarize the functionality of DEMUX and MUXADD blocks in Fig. 8 for updating memory contents and label merging.

To merge labels all entries in labels queue (registers and the FIFO to generate *CC11* to *CC21*) with the value of "*Merge Address*" are replaced with the "*CCout*" value. As shown in Fig. 6, the FIFO is implemented using register bank and circular write and read pointers. With this structure, it is possible to merge labels in one clock cycle. Figure 9 shows the pseudo-code to implement FIFO structure.

3.3 Control unit

This unit has the responsibility of generating required timing and controlling signals to guarantee the correct operation of different units in the proposed architecture. With the start of image scan, the filtering and labeling and memory units are activated to detect stars' labels and save the required information in memory blocks. This state of operation is enabled by setting controlling signals to $R/\overline{W} = 0$, $\overline{EnMEM} = 0$, $EnCofM = 0$. In this state, the outputs for the memory blocks are invalid and centroid calculation unit is deactivated.

Table 1 Functionality of DEMUX block

Mode of operation	Input signal		Output			
	Merge Address	*Address*	*WRAdd*	*RDAdd1*	*RDAdd2*	*CLRAdd*
Normal	0	$\neq 0$	*Address*	*Address*	0	0
Label merging	$\neq 0$	$\neq 0$	*Address*	*Address*	*Merge Address*	*Merge Address*

When all the image pixels are received by labeling and memory unit, outputs of memory blocks are made valid. In this state, the filtering and labeling units are deactivated temporarily and the centroid calculation module is activated to calculate stars' centers. Controlling signals are set to $R/\overline{W} = 1$, $\overline{EnMEM} = 0$ and $EnCofM = 1$ in this state. When the calculation of stars' centers finished, the memory blocks are reset to calculate stars' centers for the next frames. In this state, controlling signals are set to $R/\overline{W} = 0$, $\overline{EnMEM} = 1$ and $EnCofM = 0$.

Figure 10 shows timing diagram and number of clock cycles for different operational state of the proposed architecture. Image size of $M \times N$ pixels is assumed in this figure.

3.4 Centroid calculation unit

This module firstly removes small stars based on the number of pixels and calculates the centroids for remaining stars using the information of memory blocks. As mentioned before, the unit is enabled by setting controlling signals to $R/\overline{W} = 1$, $\overline{EnMEM} = 0$, and $EnCofM = 1$. The unit calculates centers of stars and saves the results in a table to be used by the star tracking algorithm.

Table 2 Functionality of MUXADD block

Mode of operation	Input signal		Output signal
	Merge Address	*Address*	*Data* (*WRAdd*)
Normal	0	$\neq 0$	*Ixy* + *data* (*RDAdd1*)
Label merging	$\neq 0$	$\neq 0$	*Ixy* + *data* (*RDAdd1*) + *data* (*RDAdd2*)

4 Experimental results

The proposed hardware architectures were implemented using Verilog HDL and validated through a number of simulations using Modelsim simulator. In the next step, the HDL codes were synthesized in an ASIC Digital Design Flow (ADDF) using 65 and 180 nm CMOS technologies.

The star detection and centroid calculation algorithms are implemented and verified as a general model in software environments like Matlab and openCV. Therefore, the first step to examine the accuracy of the proposed architecture is to compare the results of hardware implementation with the software results. Truncation error is a general source of error that occurs typically in transforming floating point software models into fixed point HDL implementations. To obtain truncation error, several simulations were carried out using Modelsim simulator and the software implementation using a Matlab program. The experimental results showed that the maximum Mean Absolute Error (MAE) between Matlab and Verilog implementations is 0.66 pixels for star centroid calculation. Furthermore, the Standard Deviation of Absolute Errors (SDAE) is less than 0.36 pixels.

Table 3 shows the MAE and SDAE between hardware and software implementations of Gaussian and median

Fig. 9 Pseudo-code to implement the FIFO structure

```
Inputs:
$CC_I$: FIFO input(input label)
Merge_Label: Label for merging

Outputs:
$CC_O$: FIFO output(output label)

Parameters:
col: Image width

Variables:
WP: Write Pointer (circular pointer)
RP: Read Pointer (circular pointer)

Start
  WP = 0;
  RP = 1;
  With positive edge clock
      CC_FIFO_REG[WP] = $CC_I$;
      $CC_O$ = CC_FIFO_REG[RP];
      do in all registers in the FIFO
             if (CC_FIFO_REG == Merge_Label)
                 CC_FIFO_REG = $CC_I$;
             endif
      end do

       WP = WP+1 mod (col-2);
       RP = RP+1 mod (col-2);
end
```

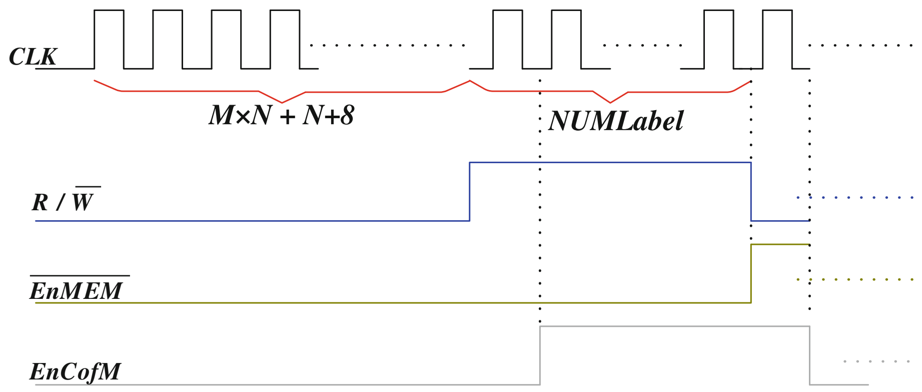


Fig. 10 Timing diagram of different controlling signals for various operational states of the proposed architecture. Image size of $M \times N$ pixels is assumed in this figure

Table 3 MAE and SDAE between hardware and software implementations of Gaussian and median filtering for different test images

Filtering algorithm	Error criteria	Test images				
		Cameraman	Coins	Pout	Lena	Sky Image1
Gaussian (6 bit rounding method)	MAE	0.3288	0.3208	0.3141	0.32	0.2940
	SDAE	0.2433	0.2315	0.2247	0.23	0.2015
Gaussian (6, −6) [25]	MAE	0.329	0.3237	0.322	0.3252	0.2938
	SDAE	0.2406	0.2337	0.2319	0.2349	0.2037
Gaussian (7, −7) [25]	MAE	0.2993	0.2862	0.2804	0.2852	0.2716
	SDAE	0.217	0.197	0.1897	0.1949	0.1797
Gaussian (8, −8) [25]	MAE	0.2964	0.2907	0.2861	0.2898	0.2732
	SDAE	0.2099	0.2017	0.1962	0.2	0.1803
Median	MAE	0	0	0	0	0
	SDAE	0	0	0	0	0

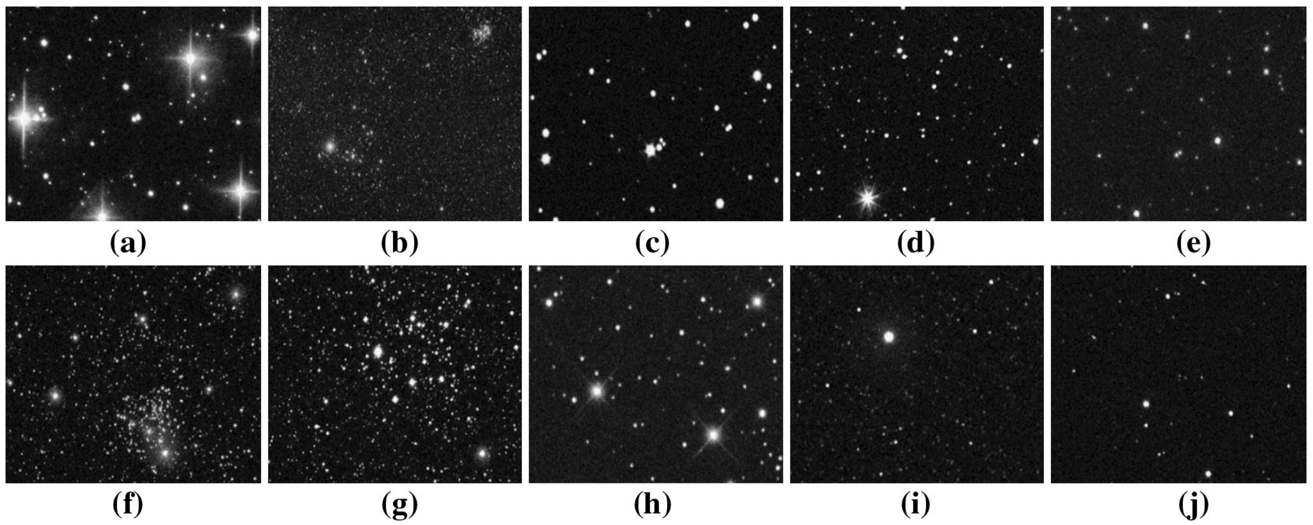


Fig. 11 Sky test images to examine and compare results of hardware and software implementations, **a** Sky Image1, **b** Sky Image2, **c** Sky Image3, **d** Sky Image4, **e** Sky Image5, **f** Sky Image6, **g** Sky Image7, **h** Sky Image8, **i** Sky Image9, **j** Sky Image10

filtering for different test images. As the table shows, the truncation error between software and hardware implementations of Gaussian filtering using 6-bit rounding method is less than 0.33 units. The table reveals that MAE for rounding method using 6-bit coefficients is marginally less than the normalized coefficient method of Khorbotly et al. [25]. Also the change in number of bits from 6 to 8 does not change the MAE considerably. The table also shows that the results of hardware and software implementations for median filtering are the same.

We also implemented star detection and centroid calculation algorithms using a Matlab program. We tested both software and hardware implementations using several test images. Figure 11 depicts several test images that were used to examine and compare results of hardware and software implementations. In all the experiments, the software and hardware implementation detected the same stars. However, the calculated centroids are slightly different in subpixel order because of the truncation error. Table 4 shows MAE and SDAE between calculated centroids using software and hardware implementations. As the table shows, the maximum MAE between Matlab and Verilog implementation is 0.66 pixels and the standard deviation of the absolute errors is less than 0.36 pixels.

Table 4 MAE and SDAE for the calculated centroids using software and hardware implementation

Test images	MAE		SDAE	
	Xc	*Yc*	*Xc*	*Yc*
Sky Image1	0.466	0.4721	0.2993	0.3071
Sky Image2	0.5095	0.5039	0.3342	0.3364
Sky Image3	0.5288	0.5452	0.3347	0.2839
Sky Image4	0.4651	0.4947	0.2906	0.2941
Sky Image5	0.465	0.5824	0.3298	0.3334
Sky Image6	0.6193	0.5652	0.2879	0.275
Sky Image7	0.5038	0.5135	0.3129	0.296
Sky Image8	0.4686	0.5515	0.259	0.2753
Sky Image9	0.6570	0.544	0.2429	0.2303
Sky Image10	0.5112	0.5654	0.2876	0.3581

The proposed hardware architecture and HDL codes were also synthesized in an ASIC digital design flow (ADDF) using 65 and 180 nm CMOS technologies. ASIC technology has the advantage of higher speed as well as lower chip area and power consumption compared with digital signal processors (DSP) and FPGA [28].

The design was optimized to obtain maximum allowable clock, lower area and power consumption. Figure 12 shows the ASIC layout for the proposed hardware architecture in 180 nm CMOS technology. Timing, power consumption, and area parameters of the designed chip are shown in Table 5.

Table 6 compares the operational speed of the proposed hardware architecture with the software implementation. As shown in the table, in comparison with software

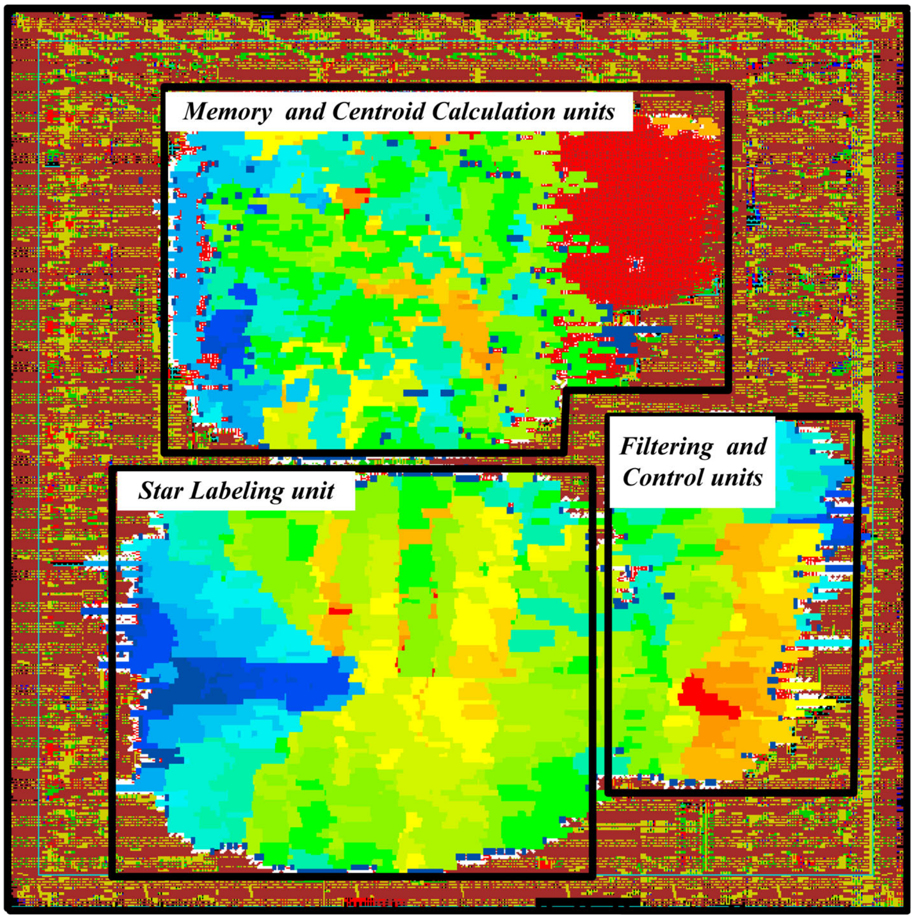

Memory and Centroid Calculation units
Filtering and Control units
Star Labeling unit

Fig. 12 ASIC layout for the proposed hardware architecture in 180 nm CMOS technology

implementation, the hardware architecture increases the processing frame rate by 57.5 and 19.4 times in 65 and 180 nm technologies, respectively. To measure the processing frame rate, we used a laptop whose specifications are shown in Table 7. Commercial star trackers like CT601 have the processing rate of about 5 and 10 Hz for LIS and normal modes of operation, respectively.

We also conducted critical path analysis for the proposed hardware architecture. Figure 13 and Table 8 show results of critical path analysis. As shown in the figure, the critical path occurs between the outputs of memory and centroid calculation units.

Table 5 Different ASIC parameters for the designed chip

ASIC parameter	Technology	
	65 nm	180 nm
Maximum clock frequency (MHz)	133.333	45
Minimum clock period (ns)	7.5	22.22
Total area (mm^2)	16.4067	115.9455
Power consumption (mW)	121.5063	203.0714

Table 6 Operational speed of star detection and centroid calculation algorithm using software and hardware implementations in different CMOS technologies for frame size of 320 × 480 pixels

Implementation	Processing frame rate (frame/s)
Software	15
Hardware with CMOS 65 nm technology	862.24
Hardware with CMOS 180 nm technology	291.1

Table 7 Specification of the utilized laptop for software implementation

Component	Specification
CPU	Intel core due 2
CPU frequency	2.4 GHz
Hard capacity	150 GB
RAM	2 GB
OS	Windows XP

4.1 Discussion

As mentioned before, star tracking algorithms generally include two stages, which are (1) star detection and centroiding and (2) star matching. In normal mode of operation, a restricted number of catalog stars are used for matching; therefore, the update rate of star detection and centroiding algorithm has a major impact on the update rate of the star tracking algorithm.

Commercial star trackers like CT601 have a processing rate of about 5 and 10 Hz for LIS and normal modes of operation, respectively. Consequently, we expect a major enhancement on the update rate of the star tracking algorithm by using the proposed hardware architecture, especially in normal mode of operation.

The cross boresight noise equivalent angle (NEA) error for a star tracker is approximately calculated by [27]:

$$\mathrm{NEA} = \frac{A \times E_{\mathrm{Centroid}}}{N_{\mathrm{Pixel}} \times \sqrt{N_{\mathrm{Star}}}} \quad (8)$$

where A is the FOV of the star tracker, E_{Centroid} is the average centroiding error, N_{Pixel} is the number of pixels across the focal plane, and N_{Star} is the number stars for matching. The results of Table 4 show that E_{Centroid} for the hardware implementation is about 0.5 pixels; however, for software implementation it is estimated at 0.1 pixels. To handle this problem, the number of stars for matching or image resolution could be increased, but at the expense of computation burden, which is easily solved using hardware implementation.

Table 8 Results of critical path analysis

Parameter description	Result
Critical path length	21.89 (ns)
Critical path slack	0.01 (ns)
Critical path Clk period	22.00 (ns)
Total negative slack	0.00 (ns)
No. of violating paths	0.00

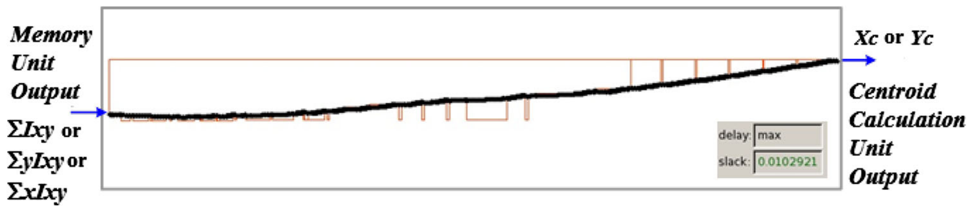


Fig. 13 Results of critical path analysis. Critical path occurs in the output of centroid calculation unit

5 Conclusions

In this paper, we proposed a new hardware architecture for star detection and centroid calculation in star tracking applications. The star detection, labeling, and centroid calculation algorithms need only one scan of the input image, which considerably reduces the memory usage of the algorithm and increases the operational speed. The proposed hardware architecture was implemented using Verilog HDL and validated through a number of simulations using Modelsim simulator. We also compared the results of hardware implementation with the results of the software program. The comparisons showed that the maximum MAE between software and hardware implementation is 0.66 pixels for star centroid calculation. Furthermore, the standard deviation of the absolute errors is less than 0.36 pixels. The proposed hardware architecture codes were also synthesized in an ADDF using 65 and 180 nm CMOS technologies. The results showed that hardware architecture increases the update rate by 57.5 and 19.4 times in 65 and 180 nm technologies, respectively.

References

1. Accardo, D., Rufino, G.: Brightness-independent start-up routine for star trackers. IEEE Trans Aerosp Electron Syst **38**(3), 813–823 (2002)
2. Lamy, Au, Rousseau, G., Bostel, J., Mazari, B.: Star recognition algorithm for APS star tracker: oriented triangles. IEEE Aerosp Electron Syst Mag **20**(2), 27–31 (2005)
3. Steyn, W., Jacobs, M., Oosthuizen, P.: A high performance star sensor system for full attitude determination on a microsatellite. In: Workshop on Control of Small Spacecraft at the 1997 Annual AAS Guidance and Control Conference, Breckenridge, CO, USA (1997)
4. Kolomenkin, M., Pollak, S., Shimshoni, I., Lindenbaum, M.: Geometric voting algorithm for star trackers. IEEE Trans Aerosp Electron Syst **44**(2), 441–456 (2008)
5. Clouse, D.S., Padgett, C.W.: Small field-of-view star identification using bayesian decision theory. IEEE Trans Aerosp Electron Syst **36**(3), 773–783 (2000)
6. Lee, H., Oh, C.S., Bang, H.: Modified grid algorithm for star pattern identification by using star trackers. In: IEEE International Conference on Recent Advances in Space Technologies (RAST '03), Daejon, South Korea, pp. 385–391. (2003)
7. Juang, J.N., Kim, H.Y., Junkins, J.L.: An efficient and robust singular value method for star pattern recognition and attitude determination. J Astronaut Sci **52**(1), 211–220 (2004)
8. Kim, H.Y., Junkins, J.L.: Self-organizing guide star selection algorithm for star trackers: thinning method. In: IEEE Aerospace Conference Proceedings, TX, USA, pp. 2275–2283 (2002)
9. Zhang, C., Chen, C., Shen, X.: A new guide star selection algorithm for star tracker. In: World, Fifth (ed.) Congress on Intelligent Control and Automation (WCICA 2004), pp. 5445–5449. China, Wuhan (2004)
10. Mahalingam, V., Bhattacharya, K., Ranganathan, N., Chakravarthula, H., Murphy, R.R., Pratt, K.S.: A VLSI architecture and algorithm for Lucas–Kanade-Based optical flow computation. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **18**(1), 29–38 (2010)
11. Wahid, K., Martuza, M., Das, M., McCrosky, C.: Efficient hardware implementation of 8 × 8 integer cosine transforms for multiple video codecs. J. Real-Time Image Process, 1–8 (2011). doi:10.1007/s11554-011-0209-6
12. Hopkinson, G., Dale, C., Marshall, P.: Proton effects in charge-coupled devices. IEEE Trans Nucl Sci **43**(2), 614–627 (1996)
13. Maheshwari, R., Rao, S.S.S.P., Poonacha, P.G.: FPGA implementation of median filter. In: IEEE Tenth International Conference on VLSI Design, Hyderabad, India, 4–7 Jan 1997, pp. 523–524 (1997)
14. Hu, Y., Ji, H.: Research on image median filtering algorithm and its FPGA implementation. In: IEEE WRI Global Congress on Intelligent Systems (GCIS '09) Shanghai, China 2009, pp. 226–230
15. Vega-Rodríguez, M.A., Sánchez-Pérez, J.M., Gómez-Pulido, J.A.: An FPGA-based implementation for median filter meeting the real-time requirements of automated visual inspection systems. In: Proceedings of the 10th Mediterranean Conference on Control and Automation, Lisbon, Portugal, Citeseer (2002)
16. He, L., Chao, Y., Suzuki, K., Wu, K.: Fast connected-component labeling. Pattern Recogn **42**(9), 1977–1987 (2009)
17. AbuBaker, A., Qahwaji, R., Ipson, S., Saleh, M.: One scan connected component labeling technique. In: IEEE International Conference on Signal Processing and Communications (ICSPC 2007), Dubai, pp. 1283–1286 (2007)
18. Flatt, H., Blume, S., Hesselbarth, S., Schunemann, T., Pirsch, P.: A parallel hardware architecture for connected component labeling based on fast label merging. In: IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP 2008) Hannover, Appelstr, pp. 144–149 (2008)
19. Ito, Y., Nakano, K.: Optimized component labeling algorithm for using in medium sized FPGAs. In: Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2008), Higashi-Hiroshima, pp. 171–176 (2008)
20. Jiang, J., Zhang, G., Wei, X., Li, X.: Rapid star tracking algorithm for star sensor. IEEE Aerosp Electron Syst Mag **24**(9), 23–33 (2009)
21. Cohen, H.A.: Parallel algorithm for gray-scale image segmentation. In: IEEE Australian and New Zealand Conference on Ligent Information Systems, Adelaide, SA, 18–20 November 1996, pp. 143–146 (1996)
22. Hao, X., Jiang, J., Zhang, G.: Star sensor image acquisition and preprocessing hardware system based on CMOS image sensor and FGPA. Proc. SPIE **5253**, 207–210 (2003)
23. Obermann, S.F., Flynn, M.J.: Division algorithms and implementations. IEEE Trans Comput **46**(8), 833–854 (1997)
24. Sorokin, N.: Implementation of high-speed fixed-point dividers on FPGA. J Comput Sci Technol **6**(1), 8–11 (2006)
25. Khorbotly, S., Hassan, F.: A modified approximation of 2D Gaussian smoothing filters for fixed-point platforms. In: IEEE 43rd Southeastern Symposium on System Theory (SSST), Auburn, USA, March 14-17, pp. 151–159 (2011)
26. Laher, R., Catanzarite, J., Conrow, T., Correll, T., Chen, R., Everett, D., Shupe, D., Lonsdale, C., Hacking, P., Gautier, N., Lebsock, K.: Attitude control system and star-tracker performance of the Wide-field Infrared Explorer spacecraft. In: Paper

AAS 00-146, Proceedings of the 2000 AAS/AIAA Spaceflight Mechanics Meeting Clearwater, FL, January 23–26 (2000)
27. Liebe, C.C.: Accuracy performance of star trackers-a tutorial. IEEE Trans Aerosp Electron Syst **38**(2), 587–599 (2002)
28. Gokhale, M., Graham, P.S.: Reconfigurable computing: accelerating computation with field-programmable gate arrays, p. 93. Springer, Berlin (2005)

Author Biographies

Mohsen Azizabadi was born in Tehran, Iran, in 1987. He received his B.Sc. in Electronic Engineering from Shahed University in 2009. He is currently a M.Sc. student in Faculty of Engineering, Shahed University, Tehran, Iran. His current research interests are hardware implementation of image processing algorithms on FPGA and ASIC.

Alireza Behard was born in Iran in 1973. He received his B.Sc. degree in electronic engineering from Electrical Engineering Faculty, Tabriz University, Tabriz, Iran, in 1995, M.Sc. degree in digital electronic from Electrical Engineering Faculty, Sharif University of Technology, Tehran, Iran, in 1998, and Ph.D. degree in electronic from Electrical Engineering Faculty, Amirkabir University of Technology, Tehran, Iran, in 2004. Currently, he is the assistant professor of Engineering Faculty, Shahed University, Tehran, Iran. His research fields are image and video processing and machine vision with special attention to visual target tracking.

Mohammad Bagher Ghaznavi-Ghoushchi received his B.Sc. degree from the Shiraz University, Shiraz, Iran, in 1993, M.Sc. and Ph.D. degrees both from the Tarbiat Modares University, Tehran, Iran, in 1997, and 2003, respectively. During 2003–2004, he was a researcher in TMU Institute of Information Technology. He is currently an Assistant Professor with Shahed University, Tehran, Iran. His interests include VLSI Design, Low-Power and Energy-Efficient circuit and systems, Computer Aided Design Automation for Mixed-Signal, and UML-based designs for SOC and Mixed-Signal.