Single-scan Run-length Algorithm for Real-time Fiducial Marker Detection Based on FPGA Devices

Farhana, B., Sufi

Department of Electrical and Electronic Engineering, University of Rajshahi, Bangladesh, fsufi.apee@ru.ac.bd

Fernando, R., Calle

Department of Technologies and Information systems, University of Castilla-La Mancha, Spain, fernando.rincon@uclm.es

Jesus, B., Romero

Department of Technologies and Information systems, University of Castilla-La Mancha, Spain, jesus.barba@uclm.es

Juan Carlos, L., López

Department of Technologies and Information systems, University of Castilla-La Mancha, Spain, juancarlos.lopez@uclm.es

Image processing pipelines involved in detecting markers from real-time video employ contour detection algorithms. Field Programmable Gate Arrays (FPGA) can provide greater computing power due to parallel processing and pipelining of the different stages in these algorithms. However, FPGA implementation of different contour detection methods remains challenging due to resource limitation and timing constraints. In this work a novel approach for a complete pipeline has been presented based on FPGA. The method used for the detection of fiducial markers from real-time video streams is based on a single-scan approach. Implementation of such pipeline into reconfigurable technology is specially challenging compared to software-oriented implementation as pointers, dynamic structures, efficient use of resources, etc. need to be considered. A driver is the need to fit the design into low-cost reconfigurable devices. To help face the related challenges of a hardware implementation High Level Synthesis (HLS) technology has been adopted, which implies reengineering of the reference algorithms. The results show a successful detection of markers over a dataset acquired in a controlled indoor environment with different resolutions at different distances. The proposal has been prototyped and tested on a Zedboard with Xilinx Zynq®-7000 SoC.

CCS CONCEPTS • Hardware→ High-level and register-transfer level synthesis • Reconfigurable logic applications • Computing methodologies→ Computer vision problems.

Additional Keywords and Phrases: Contour detection algorithm, Single-scan run length algorithm, Stream processing, FPGA, HLS, Fiducial markers, Edge computer vision.

ACM Reference Format:

Farhana, B., Sufi, Fernando, R., Calle, Jesus, B., Romero, Juan Carlos, L., López. 2022. Single-scan Run-length Algorithm for Real-time Fiducial Marker Detection Based on FPGA Devices. In *Proceedings of the 6th International Conference on Video and Image Processing, December 23-26, 2022 in Shanghai, China, 10 pages.*

1 INTRODUCTION

Square fiducial markers are becoming increasingly popular for augmented reality and virtual reality applications, mobile robot localization, unmanned aerial vehicle (UAV), unmanned ground vehicle (UGV), simultaneous localization and mapping (SLAM) techniques, pose estimation, applications for assisted living (AAL), underwater detection, i.e. computer vision in general [1-11].

Most widely used are ArUco, AprilTag, which define black square markers normally with a white border, composed of a specific pattern of white blocks inside. These types of markers are also known as binary markers. The basic technique to identify these markers includes a sequential process of image binarization with thresholding, contour detection, and pattern recognition in the respective order [12-14].

One of the most demanding computing stages in marker detection is contour detection. Connected Component analysis or blob detection is an important part of contour detection in image processing, especially for pattern recognition in computer vision. Over the years many algorithms have been developed for this purpose, most of which are based on labeling pixels and sorting them according to label equivalence tables and finally extracting the contour or blob data. After the first step of segmentation, each pixel in a binary image becomes either an object pixel, or a background pixel, and through a Connected Component Labeling (CCL) algorithm, the object pixels are assigned provisional labels, the labels of the connected pixels that form one blob are merged in the following steps. This labeling and merging procedure can take up multiple scans of an image, which require significant amount of memory, and buffering the image. As a result, they are more time consuming, and use more resources like memory, hence hardware implementation of such algorithms becomes difficult as well [15-16]. A review of connected component labeling algorithms is offered by He [15].

Based on the different approaches to label a connected component the algorithms can be broadly classified as: Contour Tracing Labeling (CTL), Hybrid Object Labeling (HOL), and Run length Based Labeling (RCL) algorithms. Also, based on the number of scans the algorithms can be of the types: Multi-scan, Two-scan, or Single-scan algorithm. Multiple scan connected component labeling algorithms require repeated image scans which makes them inefficient in terms of time and memory usage.

A look into the current state of the art shows that some of these algorithms have been implemented on Field Programmable Gate Array (FPGA) devices [17-19]. Some of the two-scan algorithms have also been implemented on FPGAs in recent works [20-22]. Single-scan algorithms ensure real-time processing speed and do not require buffering; hence they can be suitable for hardware implementation [23-30]. Single Scan algorithms with run-length based techniques have been adapted in some of the works [20, 25, 27, 28, 30] some of which have been implemented onto FPGAs, a two-scan run-length based approach is used in [22], and [19] used a multi-scan method with run-length based technique for contour detection. However, all of these are based on label assignment process and the use of equivalence tables. Also, none of the recent work proposes a complete hardware pipeline from acquisition of video frames up till possible detection of markers, but focus on the acceleration of some of the steps in the process. On the other hand, the recent work on fiducial marker detection from video-stream are software based techniques [12-14], and not based on reconfigurable hardware.

The main contribution of the present work is the design of an architecture for a complete pipeline for detecting fiducial markers from real-time video based on FPGA devices. It processes a video-frame pixel-by-pixel on a single scan. For this work Bailey's single scan crack run length algorithm [28] has been selected for contour detection, as this particular algorithm has been conceived as FPGA friendly, aiming efficient implementation. It also allows a single-scan approach taking the run lengths of the contour pixels into account. Moreover, as it does not use labeling of pixels, it requires no label resolving equivalence tables, hence eliminating the need for buffering the frames, multiple scans for the contour detection, and reduces the use of memory resources.

The design has been implemented and tested on a Xilinx Zedboard with Xilinx Zynq®-7000 SoC, and the cores are designed and synthesized using High Level Synthesis (HLS). The complete pipeline allows the parallelization of all the three important stages, namely: contour detection, corner and pattern recognition. The remainder of the paper is organized as follows: Section 2 presents the principle of the contour detection algorithm used [28], Section 3 discusses the architecture of the solution, Section 4 mentions the experimental results of the different cores in the pipeline, and finally Section 5 concludes the discussion with future scopes.

2 CONTOUR RECOGNITION

The labeling procedure, and the merging of the labels that are required for contour detection methods, usually require multiple scanning of the image frame. This makes hardware implementation of such algorithms more difficult. The single scan contour detection algorithm by Bailey [28] is based on chain coding through a crack-run-length encoding that enables an FPGA implementation. The basic principle of chain coding all the objects within an image frame is to construct fragments of chain codes as boundaries are encountered during the scan. With the scanning of each row, the existing fragments are extended on each end, thus building all the chains in parallel.

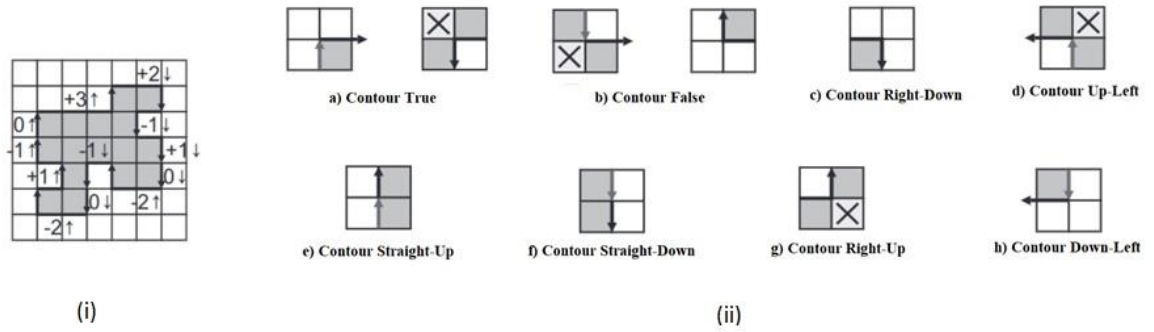

Figure 1: (i) Crack run length encoding (reproduced from [28]). (ii) 2x2 patterns of interest, where × are don't care situations, each consisting of a block of specific operations.

In the crack-run-length encoding approach adapted by Bailey a simple linking mechanism is introduced which indicates whether the right hand end of each run links up to the previous row, or down to the next row, as shown in figure 1.i (reproduced from [28]). In traversing the boundary in a clockwise direction, runs along the top have positive lengths, and runs along a bottom boundary have negative lengths indicating the direction of the corresponding links.

Several data structures are used to create the crack run length codes. As each possible contour in a row extends either to the left or to the right, a list of links is created. Each link structure consists of – a run length, the direction at the end of that run, and a pointer to the next link. Since multiple chains are being built simultaneously within an image frame, a list of segments is also created. Each segment structure consists of the co-ordinates of the start of a chain, and a pointer to the next link. The start and the end x-positions of each chain fragment are also tracked to merge the links to their corresponding segments. For a more detailed explanation of the algorithm, please refer to [28]. The 2x2 patterns of interest that need to be considered for the links and the segments are shown in fig. 1.ii (the patterns have been renamed from the originals in [28] for better understanding). The pseudo codes of each of these operations are detailed in [28], the operations within each block are executed in parallel.

However, the simulation of the original algorithm in [28] unveiled some issues that made it hang and enter in endless loops. For example, single pixel holes create such problems especially when they are diagonally connected (see Fig. 2.i.a-c). In these cases the new object pixel may be connected to a previously found contour on the left, or belong to a new contour on the right. In the case of Fig. 2.i.a, the algorithm fails to recognize the image as a completed blob. The presence of single object (dark) pixels (Fig. 2.i.d) also create unnecessary segments by detecting them as individual blobs which in this case of detecting fiducial markers are insignificant and use up resources unnecessarily. For the same reason single pixel holes (Fig. 2.i.e) can also be filled in, as well as contours with similarities of single pixel vertical columns of a few pixel lengths (in fig. 2.i.f). In this work a solution has been provided to filter out or fill in such single pixel critical cases (Fig. 2.ii) by introducing some simple filtering in the pre-process core (sub-section 3.2).

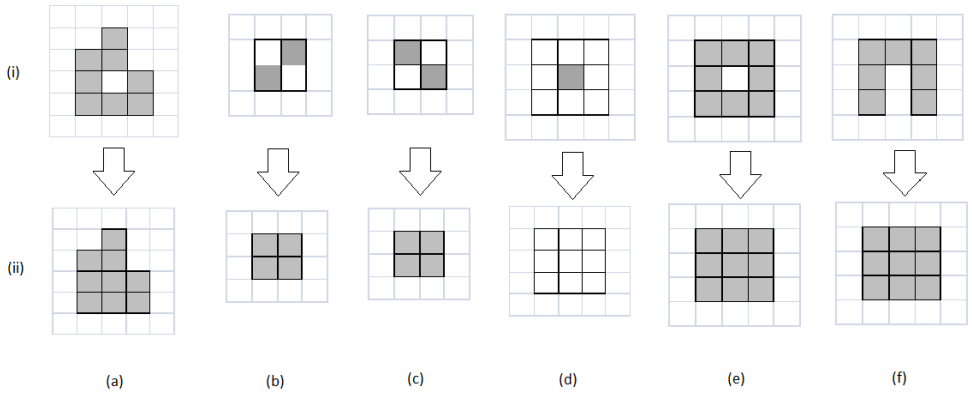

Figure 2: (i) Some examples of critical cases for the Bailey [28] proposed crack run length algorithm -- (a) Failure of completed blob (a closed contour). (b-c) Diagonally connected single object or blank (hole) pixels that create contour linking problems, these may take up unnecessary resources. (d) Single object pixel. (e) Single empty pixel (hole). (f) An empty vertical column of a few pixels. (ii) Pre-process core filtration results of the single pixel critical cases.

3 ARCHITECTURE OF THE SOLUTION

In the present work, fiducial marker detection from a video source involves a sequence of operations to acquire and filter the source video frames, detect the candidate contours, and compare them against a number of templates. Such a process can be implemented as a stream pipeline that receives a frame, provides a positive or negative detection of markers, and also their co-ordinates. More concretely, the pipeline

includes five cores: thresholding, pre-process, contour detection – based on the algorithm proposed by Bailey [28], corner detection, and pattern matching. Figure 3 shows the different stages involved in the pipeline processing. All the cores have been modeled using a high-level description written in C which will later be synthesized into RTL and deployed on real hardware (Xilinx Zynq®-7000 SoC), as described in section 4.

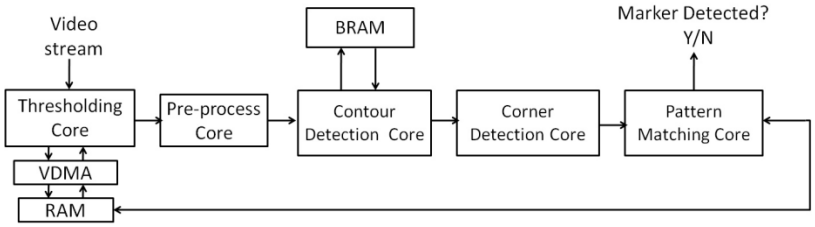

Figure 3: The complete pipeline of the design architecture.

The input to the pipeline is a stream of pixels corresponding to the video-frame containing the markers. During the processing, each stage will generate a new stream of higher abstraction level features, finally providing the list of detected markers and their co-ordinates in the video-frame. The details of every stage are provided below.

There are several concerns that have been taken into account for the design of the described pipeline. First a single-scan approach has been selected for the cores working at the pixel level to reduce the computing time for each frame and thus increase the final FPS of the algorithm. Next, the stages before contour detection have been specifically tuned for the detection of fiducial markers. These makers are closed square shaped contours which enclose other black and white rectangles. The color and form factor are used to aggressively filter and discard background data, contours that don't fit the marker type, and contours below a minimum size. This has a severe impact in both the computation time and local memory need as discussed in the experimental results of section 4.

3.1 Thresholding Core

The purpose of this first stage is the segmentation of a colored video frame based on a threshold value. The core receives an input stream of 24-bits 3 channel RGB pixels which are processed on the fly so just a very limited amount of internal memory is required for each of them. The result is an 8-bits single channel binary image where the values above the threshold are considered to be black while the rest become white. The threshold parameter is configurable and can be tuned according to ambient conditions of location of the video footage.

Since segmentation is a classic image processing library, the implementation of this core is based on the Threshold operator already available in the Xilinx Vitis HLS vision library.

3.2 Pre-process core

The purpose of the preprocessing stage is to prepare the image for the contour detection stage. Such preparation involves the creation of a single pixel white border around the image to simplify the start of the contour detection.

The other important task handled by this core is to solve the critical conditions mentioned in section 2 while describing the contour detection algorithm of [28], which arises when two blank or object pixels are diagonally connected while tracing the contours. Diagonally connected single object pixels can originate from two different contours, or from the same contour with a single hole in their midst (Fig. 2.i.a). The diagonally connected pixel scenarios (Fig. 2.i.a-c) give rise to unnecessary number of links which need to be processed to find corresponding segments, and may not be resolved correctly in detecting a blob in a few cases as has been found in case of fig. 2.i.a. Filtering out these single object pixels and filling in single pixel holes (empty pixels), simplifies the computation of the contours and reduces the size of the data structures. The filtered solutions are shown in fig.2.ii.

Figure 4.i shows the coding template used in the HLS description of the core. It is based on the concept of a 3x3 sliding window (Fig. 2.ii) applied through all the pixels streamed into the core. Therefore, it is modeled as a while-loop whose body describes the tasks to be performed for each pixel: read the pixel, shift the window, and apply a filter operation to the window. That filter depends on the concrete functionality of the core, which in this case corresponds to the cases described in fig. 2. Figure 4.ii describes the template for the RTL equivalent hardware generated from the HLS description in the loop. Additionally a 2 row line buffer is required to hold the neighborhood pixels of the 3x3 sliding window temporarily.

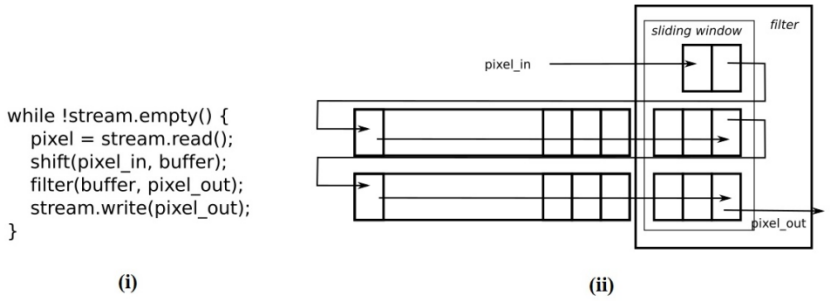

Figure 4: (i) HLS template for pixel stream processing. (ii) RTL equivalent architecture.

3.3 Contour Detection Core

Once the video frame has been preprocessed, the pixel stream is fed into the contour detection core. This core has been designed according to the chain coding through crack run length encoding proposed in [28]. The core provides a list of possible contour segments, and a list of links associated with run-lengths (refer to Section 2 for details).

The architecture and therefore the HLS model of the core is similar to the one described in the previous stage. However, the sliding window required is smaller (2x2) and thus are the row buffering needs (just a single row). Also the filter operations are different since they follow the patterns described in figure 2. Additionally a block of static RAM is required to keep both the list of contour segments and run-lengths detected in the frame. The RAM is required since the data structures used for the run-length and segment storage, are based on a dynamic linked list.

3.4 Corner Detection Core

Once all contour segments and their run-lengths have been identified, the detection stage will track every run-length to find four possible marker corners. All run-lengths not belonging to a closed contour or a quadrilateral are discarded. Then the sequence of run-lengths is analyzed in order to find sudden variations in the gradient of the contour (horizontal vs. vertical variations). This is an aggressive strategy adapted for obtaining the sharp corners of a quadrilateral since that is the shape of a possible fiducial marker. The four co-ordinates of the rectangular shapes are then streamed to the next core. The output of the contour detection core is a list of links and a list of corresponding segments.

The implementation of the corner detection core is based on a Finite State Machine (FSM) that manages the extraction of the run-lengths from the RAM, the evaluation of the gradients, the storage of the possible corners into a local FIFO, and the checking of the closed contour once the last run-length of the segment is obtained.

The gradient analysis is based on saturated counters for both horizontal and vertical relative offsets from the last detected corner. Once a corner is detected, the FSM records the last direction of the contour (up, down, left or right). Then for each new run-length extracted, each counter is updated and trimmed with respect to the saturated value if necessary. If the offset for the opposite direction (e.g. vertical when direction was up or down) gets to a threshold (e.g. 5 pixels) then the end point of that run-length is considered a possible marker corner, and the corresponding counters are reset and the direction updated. The process is repeated until four corners in a closed contour are found.

3.5 Pattern Matching Core

The pattern matching core is the final one in the pipeline, it receives the four co-ordinates of each possible marker from the corner detection core and also the original video frame from the thresholding core. The pattern matching core then creates a unique grid system for each rectangular shape by – calculating the slope (*m*) between the upper left point $A(x_1,y_1)$ and upper right point $B(x_2,y_2)$ with equation (1), and using a simple line equation (2). The discretization is calculated using equation (3), where *D* is the number of divisions (in this case 16 in order to sample the mid-points of an 8x8 grid). The process is iterated using equation (4) for *n* lines (see Fig.5). Equation (5) is used to find the pixel values of the mid-points of all the grid-boxes.

$$m = \frac{y2-y1}{x2-x1} = \frac{\Delta y}{\Delta x} \quad (1)$$

$$y = mx + b \quad (2)$$

$$\delta x = \frac{\Delta y}{D}, \ \delta y = \frac{\Delta x}{D} \quad (3)$$

$$x_n = x_{n-1} + \delta x, \quad y_n = y_{n-1} + \delta y \quad (4)$$

$$(x, y) = A(x_1, y_1) + \left(\frac{\delta x}{2}, \frac{\delta y}{2}\right) \quad (5)$$

These values are then compared against the mid-point pixel values of the grids in the templates. The core also rotates the rectangles in 90º, 180º, and 270º angles and compares with respective rotated template values. Finally, it gives an output confirming whether that particular fiducial marker is present in the video frame or not.

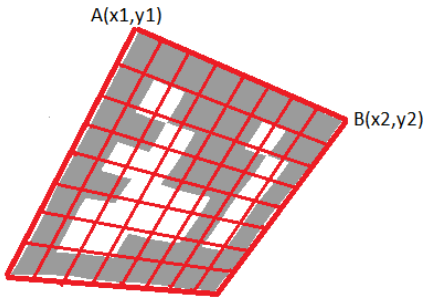

Figure 5: Example of a 8x8 grid system used for checking the pattern of a possible marker rectangle. The pixel values in each grid square mid-point is checked against the template values.

4 EXPERIMENTAL DATA AND RESULTS

In the present work, the architectural solution designed around the algorithm of [28] – the contour core and the corner detection core, make it possible to detect regions of interest with a single scan of a video frame through pixel-by-pixel processing, without creating any labels, or using any label resolving techniques. This eliminates: the need for buffering the frames, and also multiple scans for the contour detection, thus reducing the use of memory resources. For the current work a dataset in a controlled indoor environment with different resolutions at different distances was created and tested on a Zedboard with Xilinx Zynq®-7000 SoC. The cores are designed and synthesized using Xilinx Vitis HLS tools.

Table1: Performance and Resource Estimates of the Different Cores (640×480 resolution)

Core	FFs	LUTs
Thresholding	1959	2394
Pre-process	2354	5141
Contour Detection	3358	7930
Corner Detection	1161	2074
Pattern Matching	14	115

[a]FF – Flip Flops, LUT – Look up Table.

Both the resources and latency of the proposed solution depend very much on the resolution and type of images to be considered. Table 1 shows the FPGA resources in terms of LUTs and FFs required for the video-frame in fig. 6.i of the resolution 640x480 pixels. Additionally, enough DRAM space is required for the storage of the frame under analysis. Such storage is used during the pattern matching stage once the possible marker contours have been detected and the maker patterns are to be checked against the marker positions in the original video-frame.

An increase in resolution will have some, although limited, impact on the FPGA resources, since most of the pixel stream processing is performed on the fly. A resolution increase will mostly imply the extension of the row buffers used in the thresholding and pre-process cores, while the contour and marker detection are not very much affected. The impact, however, will be higher with respect to the usage of memory. Extra DRAM will be required for higher resolution video-frames, although DRAM size is not normally an issue in this kind of systems. It may also affect static RAM size, since bigger images can also imply more possible contours and then extra run-lengths to be stored in the dynamic list.

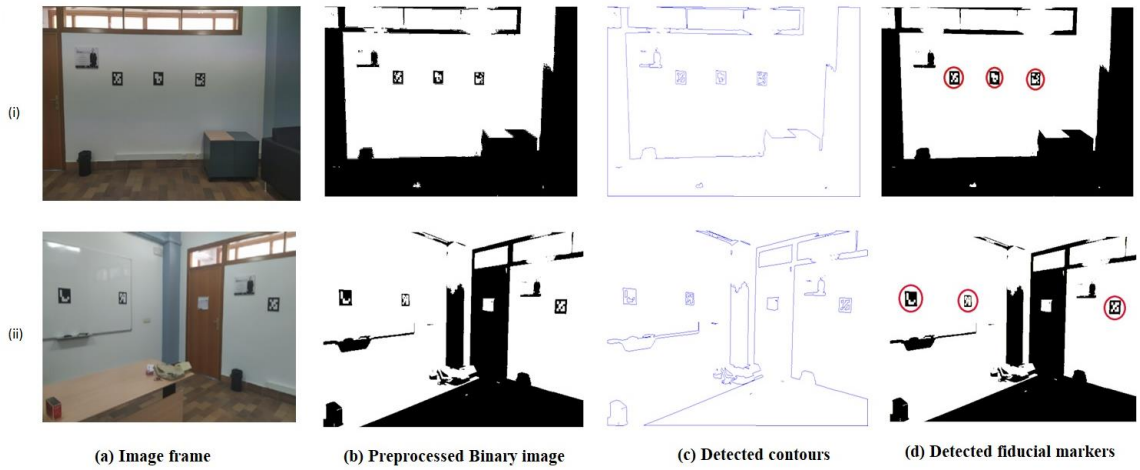

Figure 6: (i-ii) Sample image frames of 640×480 resolution --- (a) Original images. (b) Binarized images (pre-process core output). (c) Detected contours. (d) Correctly identified fiducial markers (circled) from the templates provided (pattern matching core output).

The main contributors to the latency are the contour detection and pattern matching cores. Additionally, in the current implementation, and in order to keep static memory requirements as low as possible, no double buffering has been implemented which causes the corner detection stage to block until all possible contours have been extracted. Latency contribution can then be divided into two terms: detection of contours, and detection of markers. Before and during the detection of contours, the thresholding and pre-process cores perform pixel-by-pixel stream processing with an Initiation Interval (II) of 1 cycle, while the contour detection core has an II of 6. That means that every new cycle a new pixel is pre-processed, while the contour detection requires 5 more cycles. Since the speed of processing in the pipeline is determined by the slowest stage, contour detection latency is the result of the multiplication of the II=6 by the size of the video-frame in pixels.

The overall latency for the marker detection can be approximated by the following formula in the current implementation of the pipeline:

$$latency = (\#\ pixels \times 6) + \ MAX\left((\#\ segment + \#\ runlengths) \times 2\right), pattern\ matching\ latency) \quad (6)$$

The second term in equation (6) is concerning to the marker detection, here the II is 2, due to memory access latency. In this stage the RAM will be accessed as many times as segments and run-lengths have been detected. The pattern-matching latency has to be considered along with this. In this case, each possible pattern implies (8x8=) 64 DRAM accesses, in order to compare the expected and real values in the picture. DRAM memory has 10 times more latency than the static one, and therefore matching a possible pattern implies around 700 cycles. Additionally, each pattern can be in fact matched against the 4 possible 90 degrees rotations, which would require 2800 cycles. However, the pattern matching process can be started as soon as a possible marker has been detected, which implies overlapping the processes of corner detection and marker matching. Since the number of run-lengths is normally expected to be orders of magnitude higher than the number of patterns to match, in general only the higher of the two terms will contribute to the final value of the latency.

While the increase in resolution of the frames has a direct impact on the overall latency of the algorithm, this is not the main issue to consider for the implementation of the pipeline architecture. The most important factor is the number of possible contours, and more concretely, the number of existing run-lengths in the frame. This has a proportional impact in the memory requirements which can be very high. In the worst case scenario for a row of N pixels it can be possible to find up to N/3 links. However, this would correspond to a frame resembling a checkerboard which is far from real world situations. In our experiments the number of links detected is in the range of a few thousands.

For the example picture of figure 6.i, the number of links and run-lengths detected were 2970 and 70, respectively. This led to a measured overall latency of 18.5 ns for a 100 MHz clock cycle for a single frame. Additionally, 62 Kbits of static memory were required: 2.4 Kbits for the storage of the segments, and 59.4 Kbits for the run-lengths.

5 CONCLUSION

In this work we have presented a novel approach for a complete imaging pipeline to allow processing of input video-stream up to the fiducial marker detection. The pipeline is based on a single-pass run length based algorithm for contour detection, and the contour and corner detection cores are optimized for fiducial marker detection in particular. The crack run-length algorithm proposed for contour detection based on FPGAs by Bailey [28] has been experimentally tested in the contour detection core and adjusted with the pre-process core. None of the previous work mentioned earlier that used a single-scan approach for contour detection used a complete pipeline up to the final marker detection. Also, while there exist different approaches for fiducial marker detections based on software [12-14], there are none with hardware based approach. The single-scan pipeline proposed in this work reduces the use of resources on hardware like an FPGA, and provides a marker detection from video stream with about 35-70 fps for 640x480 resolution, depending on the image frame resulting after the thresholding. The pipeline has also been tested for HD and other resolutions resulting in similar values.

Future improvement will involve the calculation of the run lengths during the pre-process stage, so that image pixels will not feed the contour core, following a contour and corner detection only from the run lengths. This will reduce the number of inputs fed into the contour detection core by two orders of magnitude.

ACKNOWLEDGMENTS

This work has been partially funded by the Spanish Ministry of Economy and Competitiveness (MINECO) under the project TALENT-BELIEF, Grant no: PID2020-116417RB-C44, and by the European Commission under the Horizon 2020 project SHAPES, Grant no: 857159.

REFERENCES

[1] Schoun, B., Oagaz, H., Choi, M.H. 2021. Corner-based square fiducial marker detection for hand-manipulated AR objects. In Proceedings of the IEEE International Conference on Intelligent Reality (ICIR). IEEE. doi: 10.1109/ICIR51845.2021.00014

[2] Oliveira J. A., Piardi, L., Giometti, E. B., Leitao, P. 2021. Improving the mobile robots indoor localization system by combining SLAM with fiducial markers. In: 2021 Latin American Robotic Symposium (LARS), 2021 Brazilian Symposium on Robotics (SBR). IEEE. doi: 10.1109/LARS/SBR/WRE54079.2021.9605456

[3] Annusewicz, A., Zwierzchowski, J. 2020. Marker detection algorithm for the navigation of a mobile robot. In Proceedings of the 27th International Conference - Mixed Design of Integrated Circuits and Systems 2020, Poland, pp. 223-226. IEEE Xplore. doi: 10.23919/mixdes49814.2020.9155658

[4] Liu, Y., Schofield, H., Shan, J. 2021. Navigation of a self-driving vehicle using one fiducial marker. In: 2021 IEEE International

Conference on Multisensor Fusion and Integration for Intelligent Systems. IEEE. doi: 10.1109/MFI52462.2021.9591194

[5] Attard, L., Debono, C.J., Valentino, G., Castro, M., Tambutti, M.L.B. 2018. An RGB-D video-based wire detection tool to aid robotic arms during machine alignment measurement. In: Proceedings of the IST 2018 - IEEE International Conference on Imaging Systems and Techniques. IEEE. doi: 10.1109/IST.2018.8577199

[6] Khazetdinov, A., Zakiev, A., Tsoy, T., Svinin, M., Magid, E. 2021. Embedded ArUco: a novel approach for high precision UAV landing. In: 2021 International Siberian Conference on Control and Communications (SIBCON). IEEE.

[7] Zhang, X., Jiang, J., Fang, Y et al. 2019. Enhanced fiducial marker based precise landing for quadrotors. In Proceedings of the 2019 IEEE-ASME International Conference on Advanced Intelligent Mechatronics, Hong Kong, China. IEEE. doi: 10.1109/AIM.2019.8868532

[8] Acuna, R., Li, Z., Volker, W. 2018. MOMA: Visual Mobile Marker Odometry. In: 2018 International Conference on Indoor Positioning and Indoor Navigation (IPIN), France, IEEE. doi: 10.1109/IPIN.2018.8533685

[9] Truong, V.T., Lao, J.S., Huang, C.C. 2020. Multi-camera marker-based real-time head pose estimation system. In: 2020 International Conference on Multimedia Analysis and Pattern Recognition, MAPR. IEEE. doi: 10.1109/MAPR49794.2020.9237775

[10] Romero-Ramirez, F. J., Muñoz-Salinas, R., Medina-Carnicer, R. 2019. Fractal Markers: a new approach for long-range marker pose estimation under occlusion. IEEE Access, vol. 7, IEEE. doi: 10.1109/ACCESS.2019.2951204

[11] Wang, Y., Ji, Y. Liu, D. Tamura, Y. et al. 2020. ACMarker: acoustic camera-based fiducial marker system in underwater environment. In: IEEE Robotics and Automation Letters, vol. 5 (6), pp. 5018-5025. IEEE. doi: 10.1109/LRA.2020.3005375

[12] Romero-Ramirez, F. J., Muñoz-Salinas, R., Medina-Carnicer, R. 2018.Speeded up detection of squared fiducial markers.In: Image and Vision Computing, vol. 76, pp. 38–47, Aug. 2018. doi: 10.1016/j.imavis.2018.05.004

[13] Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F. J., Marin-Jimenez, M. J. 2014. Automatic generation and detection of highly reliable fiducial markers under occlusion. In: Pattern Recognition, vol. 47 (6), pp. 2280–2292. Elsevier. doi: https://doi.org/10.1016/j.patcog.2014.01.005

[14] Hajjami, J., Caracotte, J., Caron, G., Napoleon, T. 2020. ArUcOmni: detection of highly reliable fiducial markers in panoramic images. In: 2020 IEEE-CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). IEEE. doi:10.1109/CVPRW50498.2020.00325

[15] He, L., Ren, X., Gao, Q., Zhao, X., Yao, B., Chao, Y. 2017. The connected-component labeling problem: A review of state-of-the-art algorithms. In: Journal of Pattern Recognition, Vol. 70, pp. 25–43. Elsevier. doi:10.1016/j.patcog.2017.04.018

[16] Cabaret, L., Lacassagne, L., Oudni, L. 2015. A review of world's fastest connected component labeling algorithms: Speed and energy estimation. In: Conference on Design and Architectures for Signal and Image Processing (DASIP). IEEE. doi:10.1109/DASIP.2014.7115641

[17] Benkrid, K., Sukhsawas, S., Crookes, D. 2003. An FPGA-based image connected component labeler. In: Lecture Notes in Computer Science, vol. 2778, pp. 1012-1015. Springer. doi: 10.1007/978-3-540-45234-8_108

[18] Jablonski, M., Gorgon, M. 2004. Handel-C implementation of classical component labelling algorithm. In: Euromicro Symposium on Digital System Design,DSD, IEEE. doi: 10.1109/DSD.2004.1333301

[19] Appiah, K., Hunter, A., Dickinson, P., Owens, J. 2008. A run-length based connected component algorithm for FPGA implementation. In Proceedings of the 2008 International Conference on Field-Programmable Technology, ICFPT 2008, pp. 177-184. IEEE. doi: 10.1109/FPT.2008.4762381

[20] Zhou, H., Dou, R., Cheng, L., Liu, J., Wu, N. 2022. A provisional labels-reduced, real-time connected component labeling algorithm for edge hardware. In: IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 69 (6), pp. 2997-3001. IEEE. doi:10.1109/tcsii.2022.3152783

[21] Schwenk, K., Huber, F. 2015. Connected component labeling algorithm for very complex and high-resolution images on an FPGA platform. In Proceedings of the SPIE High-Performance Computing in Remote Sensing, vol. 964603, pp. 964603-02, SPIE. doi:10.1117/12.2194101

[22] Appiah, K., Hunter, A., Dickinson, Meng, H. 2010. Accelerated hardware video object segmentation: From foreground detection to connected components labeling. In: Computer Vision and Image Understanding, vol. 114, pp. 1282–1291. Elsevier. doi: 10.1016/j.cviu.2010.03.021

[23] Zhao, C., Gao, W., Nie, Fe. 2020.A memory-efficient hardware architecture for connected component labeling in embedded system. In: IEEE Transactions on Circuits and Systems for Video Technology, vol. 30 (9), IEEE. doi: 10.1109/TCSVT.2019.2937189

[24] Bailey, D. G., Klaiber, M. J. 2019.Zig-zag based single-pass connected components analysis. In: Journal of Imaging, vol. 5 (4). MDPI. doi: 10.3390/jimaging5040045

[25] Tang, J.W., Shaikh-Husin, N., Sheikh, U. U., Marsono, M. N. 2018. A linked list run-length-based single-pass connected component analysis for real-time embedded hardware, In: ACM Journal of Real-Time Image Processing, vol. 15 (1), pp. 197-215. Springer. doi: 10.1007/s11554-016-0590-2

[26] Klaiber, M. J. Bailey, D. G. Baroud, Y. O. Simon, S. 2016. A resource-efficient hardware architecture for connected component analysis, In: IEEE Transactions on Circuits and Systems for Video Technology, vol. 26 (7), pp. 1334-1349. IEEE. doi: 10.1109/TCSVT.2015.2450371

[27] Zhao, F., Zhang L. H., Zhi, Y. Z.. 2013. Real-time single-pass connected components analysis algorithm, In: EURASIP Journal on Image and Video Processing, vol. 2013, pp. 1-10. Springer. doi: 10.1186/1687-5281-2013-21

[28] Bailey, D. G. 2010. Chain coding streamed images through crack run-length encoding, In: 25th International Conference Image and

Vision Computing New Zealand, IEEE. doi: 10.1109/IVCNZ.2010.6148812

[29] Johnston, C.T. Bailey, D. G. 2008. FPGA implementation of a single pass connected components algorithm, In: Proceedings - 4th IEEE International Symposium on Electronic Design, Test and Applications, DELTA 2008, pp. 228-231. IEEE. doi: 10.1109/DELTA.2008.21

[30] Trein, J., Schwarzbacher, A. Th., Hoppe, B., Noffz, K.-H., Trenschel, T. 2007. Development of a FPGA based real-time blob analysis circuit. In: Irish Signals and Systems Conference (ISSC), Derry, UK (2007), pp. 121–126.

Authors' background

Your Name	**Title***	**Research Field**	**Personal website （URL）**
Farhana Binte Sufi	Assistant Professor	Embedded Heterogenous Systems, Computer Vision, Biomedical Instrumentation	http://www.ru.ac.bd/eee/
Fernando Rincon Calle	Professor	Embedded Heterogenous Systems, Computer Vision, IoT	https://arcoresearch.com
Jesus Barba Romero	Professor	Embedded Heterogenous Systems, Computer Vision, IoT	https://arcoresearch.com
Juan Carlos López López	Professor	Embedded Heterogenous Systems, Computer Vision, IoT	https://arcoresearch.com