VOL. 10, NO. 17, SEPTEMBER 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences
©2006-2015 Asian Research Publishing Network (ARPN). All rights reserved.
www.arpnjournals.com

IMPLEMENTATION AND OPTIMIZATION OF CONNECTED COMPONENT LABELING IN RASPBERRY PI

Nirmal T. M.[1], K. R. Joy[1] and Rajeev K.[2]
[1]Department of ECE, Sahrdaya College of Engineering & Technology, Kodakara, Kerala, India
[2]Strategic Services Group, QuEST Global, Technopark, Trivandrum, Kerala, India
E-Mail: nirmaltm@ieee.org

ABSTRACT

Computer vision is the method of acquiring, processing, analyzing and understanding high-dimensional data images in order to produce numerical or symbolic information. These concepts are being pervasively applied in all fields of life, but due to the requirements of high processing power and high degree of parallelism, real-time computer vision applications are restricted to FPGA's and high end processing GPU's.

The Raspberry Pi is a low cost medium processing power embedded development board. Raspberry's capability in performing computer vision functions has been demonstrated with OpenCV support. But optimization of OpenCV functions in Raspberry Pi still considered a challenge. There is no direct OpenCV function for performing Connected component Labelling. The Connected Component Labeling is commonly used for identifying objects and marking fields for majority of computer vision application. This work discuses about the implementation and optimization of connected component labeling algorithms on Raspberry Pi. Apart from algorithm level adaptations for better hardware utilization, code level optimization is also explored. CCL is implemented with fastest algorithm known as LSL. In addition to LSL, Rosenfeld and contour based labeling are discussed for reference. The final implementation consists of real time connected component labeling with component numbering and estimation of area bounded by each component.

Keywords: labeling, opencv, raspberry pi, ARM, visual studio, rosenfeld, contour labeling.

INTRODUCTION

Image processing works in early 90's was challenging due to less processing power and storage space. Useful image processing algorithms still stay as documents due to this. With the development of modern processors like GPU, APU the problem of processing power was significantly reduced. Trend area of today is to implement and enhance the image processing on embedded platform.

Generally the image processing requires processing power of a high end computer for getting the real time performance. Applications like medical imaging, automotive, robotic application, military and remote sensing are exception for using high end computer for the computation of image processing algorithms. In this scenario the development of embedded systems comes. Currently for such application the FPGA or small embedded board networked with high end computer is used. This is not considered as cost efficient and portable. One of the main areas of image processing that is emerging today is computer vision. Computer vision includes methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information, e.g., in the forms of decisions.

An embedded board is described as a single-board computer (SBC) with complete computer built on a single circuit board; consist of microprocessor(s), memory, input/output (I/O) and other features required of a functional computer. Single-board computers were made as demonstration or development systems, for educational systems or for use as embedded computer controllers.

The component detection from an image is the basic step in computer vision. It can be applied to many fields such as medical image processing, automotive industry etc. Enhancing the processing speed of connected component labeling algorithm in an embedded platform is an advantage to real time and portable image processing applications. The algorithm helps to find out the number of components in an image as well as to find the area of a region in an image.

The scope of this work is to implement and optimize the connected component labeling algorithm in an embedded platform. The optimization is done to improve the number of frames per second and better processing with memory utilization. The rest of this article is organized as follows. Section 2 describes the literature review performed for this work. Section 3 describes the development of the. Section 4 consists of implementation and performance measurement. Section 5 discussion about the result obtained. Section 6 includes the conclusion.

LITERATURE REVIEW

The real time embedded processing boards are very common in the industrial and military applications. But due to low performance these complex computer vision is not usually implemented on this types of boards. There are number of works done in the area of raspberry pi robotic applications. But referring to computer vision area the number of works are coming down. The raspberry Pi is enriched with the support of OpenCV and OpenGL. But computer vision application development is very less in Raspberry Pi. This section described with the previous work done on Raspberry Pi in the area of computer vision and also the connected component labeling algorithm.

The raspberry Pi based color identification lead to the design and implementation of the image processing based color speaking system using Raspberry Pi and USB Camera. This design is a minimized electronic gadget which identifies the color of an object image and speaks out the corresponding color. The model uses hardware components such as Raspberry Pi (Model B) and USB webcam [1]. MATLAB Simulink tool boxes are used to implement the project. The proposed gadget works in standalone mode without the necessity of PC when once programmed. The work used rapid prototype technique approach of image processing for real-time application using MATLAB Simulink support package meant for Raspberry PI. The real time color identification is done using the help of thresholding and identification of major color component in an image using OpenCV support.

Many of the OpenCV functions are explained in Real-Time Cartoonization Using Raspberry Pi [2]. Cartoonization is a method of image stylization in which input moreover looks like a sketch or cartoon like image. It includes making edges bolder, making colors brighter and lively, accompanied with smoothening operation so as to filter out the high frequency details. Furthermore, reduction in the luminance contents is carried out, which helps to quantize the image for obtaining cartoon like effect. Different image processing techniques such as contrast stretching, bilateral filtering, luminance quantization, and edge detection are being implemented forachieving the desired outcome. OpenCV programming is used for implementing image processing techniques. Many of OpenCV real time functions are used for getting this cartoonized image.

The character reorganization can be done with the help of Raspberry Pi, the work on detecting the Tamil characters using Raspberry Pi [3] deals with the Object Identification of OpenCV functionality. The input is taken by a camera in the hand-held device and the output is given as speech through microphone using the above hardware interface. The software implementation and the hardware interfacing were done using Embedded Linux.

These are the main works done in the Raspberry Pi on computer vision application. The optimization of the application is yet not to done in an efficiently. The optimization in a real time embedded board is done in various platforms. Due to Raspberry Pi is new to common market the optimization on this is not be announced in a successful manner. Some of the optimization work done in similar platforms are discussed in this section.

The optimization of face detection in an embedded processor is described as Implementation and Optimization of Embedded Face Detection System [4]. To attain real time performance particularly on mobile platforms needs to apply optimized algorithms. This performance optimization of general computer vision algorithms such as Viola Jones Face Detection on embedded systems with limited resources. The Viola Jones algorithm which is popular for face detection, can be implemented on mobile platforms. The algorithms are benchmarked on the Intel processor and Beagle Board xM, which is a new low-cost low-power platform based on the Texas Instruments (TI) DM 3730 processor architecture. The DM 3730 processor is characterized by the presence of an asymmetric dual-core architecture, which includes an ARM and a DSP along with a shared memory between them. OpenCV, which is a famous open source computer vision library developed by Intel corporation was utilized for some of the algorithms. Comparative results for the different platforms are introduced and analyzed with an emphasis on real-time Application.

Connected component Labeling is one of the core computer vision algorithm, the labeling is done on the binary image. The best method to perform the CCL is Light Speed Labeling [5]. The real utilization of CCL – that implies to extract some features – we performed an in-depth analysis of the labeling parts duration to explain why run-length based algorithms outperform pixel-based ones. The experimentation shows that all pixel-based algorithms have a very similar behavior (labeling, transitive closure and relabeling parts) both on random images and images from data-bases. Moreover, the features computation is a significant time-consuming part for pixel-based algorithms whereas for LSL, features computation is very fast and efficient (for image granularity ≥4) run-length encoding. Future work will consider the parallelization of these algorithms as modern processors are multicore.

From the above discussion the majority of computer vision application done on Raspberry Pi is with the help of a PC interfaced to it. The image processing can be directly done in Raspberry Pi. This work aims to a fastest implementation of connected component labeling which is a core computer Vision Algorithmic application in Raspberry Pi. The algorithm is used for counting the number of components and the area bounded by each component in the image. The work focused to detect the CCL in faster method and also aims to find the number of component and its area.

DESIGN AND DEVELOPMENT

This section describes the relevance of Connected Component Labeling and how it can be found out. The algorithms used for finding the CCL and the detailed description about Light Speed Labeling Algorithm is also included.

Connected component labeling is also called as connected component analysis, blob extraction, region labeling, blob discovery, or region extraction. It is an algorithmic application of graph theory, where subsets of connected components are uniquely labeled based on a given heuristic. Connected component labeling [6] is not to be confused with segmentation. Connected component labeling is used in computer vision to detect connected regions in binary digital images, although color images and data with higher dimensionality can also be processed. When integrated into an image recognition system or human computer interaction interface, connected component labeling can operate on a variety of information. Blob extraction is generally performed on the resulting binary image from a thresholding step. Blobs may be counted, filtered and tracked.

A graph, containing vertices and connecting edges, is constructed from relevant input data. The vertices contain information required by the comparison heuristic, while the edges indicate connected 'neighbors'. An algorithm traverses the graph, labeling the vertices based on the connectivity and relative values of their neighbors. Connectivity is determined by the medium image graphs, for example, can be 4-connected or 8-connected.

Rosenfeld Connected Component Labeling

The Rosenfeld is a historical based approach algorithm. It consists of two pass for making the final labeled image. The first scan create the pre labeled image which consist of one left and three upper connected pixel labels and also it creates the equivalence table during the first scan. The next step is to resolve the equivalence table [7]. After resolving the equivalence table the result will be of an idea regarding the number of labels in the image. This is same as the number of components or elements in the image. All these operations are performed on the threshold binary image. The second pass is performed on the labeled image, during the second scan the labeled image entry is replaced according to the equivalence table entry.

Haralick Connected Component Labeling

In 1981 Haralick introduced an iterative algorithm which does not require any auxiliary storage for label equivalences [8]. This technique involves multiple forward and backward raster scan passes through the image until no label change occurs. All the label collisions are solved on the local neighborhood basis according to the equation (1), however ET does not apply here. After the first scan through the binary image B, all the pixels will be assigned with the preliminary labels similarly as in the classical algorithm, however all the label ambiguities will be resolved on the local neighborhood basis during the following multiple forward and backward scans with alternating scan mask Ms = Mf and Ms = Mb respectively according to:

$$g(x,y) = \begin{cases} F_b & if\ g(x,y) = F_b \\ g_{min} & Otherwise \end{cases} \quad (1)$$

This algorithm was designed for systems with limited memory resources processing low resolution images. The performance of this algorithm is related to the size and the complexity of the binary image, thus it is not recommended for higher resolution images. Recent implementations improved the processing time by introducing local equivalence tables, however the number of scans through the image frame is still dependent on the image complexity and is hard to predict. Hence these algorithms are not suitable for real-time video processing and will not be taken into consideration in further discussion.

Light Speed Labeling

The Light speed Labeling focused on architectural algorithm adequacy, which means it can be adopted for RISC architecture mechanism such as ARM core processors. LSL algorithm defines in a segment based labeling format. It introduces a new-line labeling that help to reduce the conditional statements. Which reduces the number of consecutive comparison check and hence the time for execution is also reduced. The next technique used in LSL is the introduction of Run Length Coding mechanism for the label identification. The RLC helps to reduce the number of passes throw the actual image.

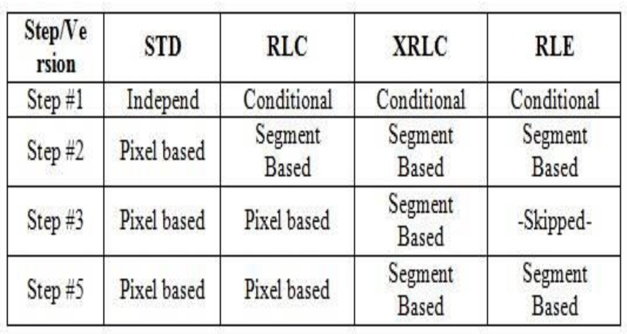
The LSL algorithm is designed for using in RISC architecture machines. The pipeline mechanism in ARM architecture will help LSL to perform in faster manner. Comparing to classical and historical algorithms LSL is not a single pass or two pass algorithms, it’s actually a 3-pass labeling algorithm. In first pass also call as pre-pass the algorithm form a relative labeling. The relative labeling is based on RLC which help to speeded up the total performance of the process and reduces the number of calculation in the next pass. The LSL help to remove the main drawback of segment based algorithms fusion sorting mechanism. Consider the case where LSL is fixed as a part complex processing algorithms or process. The base underling LSL can find out number of adjacent segment and the labels for the same. The LSL consist of five basic steps. They are described as follows.

Step#1: First Labeling (Relative Segment Labeling)
Step#2: Equivalence Building
Step#3: Second Labeling (First Absolute Labeling)
Step#4: Equivalence Resolution
Step#5: Third Labeling (Final Labeling)

The above mentioned are the basic steps for an LSL algorithm to perform its operation.

The LSL algorithm can be implemented in four methods [9]. They are STD, RLC, XRLC, and RLE. Only differences between these are the use of basic five steps. The first step will be a data independent or conditional to pixel value. Steps 2, 3 and 5 can be either pixel based or segment based. In RLE the Step#3 will be skipped. The specification of 4 versions is described in Table-1.

Table-1. LSL version specification.



Step/Version	STD	RLC	XRLC	RLE
Step #1	Independ	Conditional	Conditional	Conditional
Step #2	Pixel based	Segment Based	Segment Based	Segment Based
Step #3	Pixel based	Pixel based	Segment Based	-Skipped-
Step #5	Pixel based	Pixel based	Segment Based	Segment Based

LSL Algorithm

The LSL algorithm parameter [10] defined as follows.

- er, a relative label,
- er, a relative label,
- ea, an absolute label,
- a, an ancestor label
- X, a binary image of size $h\ x\ w$, Xi the current line of X, and X_{i-1} the previous line.
- EA, an image of size $h\ x\ w$ of absolute labels ea before equivalence resolution
- L, an image of size $h\ x\ w$ of absolute labels ea after equivalences resolution
- ER_i, an associative table of size w holding the relative labels er associated to X_i
- ner, the number of segments of ER_i – black + white.
- RLC_i, a table holding the run length coding of segments of the line X_i, RLC_{i-1} is the similar memorization of the previous line.
- ERA_i, an associative table holding the association between er and ea: $ea = ERA_i[er]$
- EQ, the table holding the equivalence classes, before transitive closure
- A, the table of equivalence classes after their resolution
- RLC, a 2D table of size $h\ x\ 2w$ holding all segments of every line, used along LSL evolutions
- LEA, 2D list of absolute labels of every line, used in LSL evolutions

Relative segment labeling: Step#1

Step#1 performs a relative labeling of each line. For each line X_i the ER_itable holds the associated relative label er of each segment. Relative refers to that a same numbering (restarting from zero) is performed for every line. As segments are separated by slices of background pixels, an efficient numbering trick consists in assigning odd numbers to segments and even numbers to background (Figure-1). While labeling segments, their run length code (begins and end $[j_0;\ j_1]$ of each segment) is also stored into the RLC_i table.

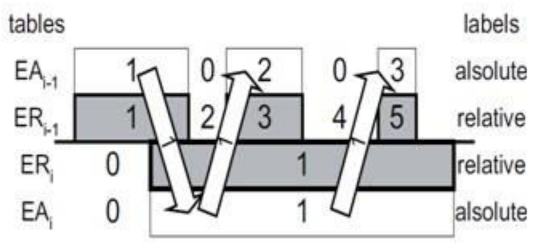


Figure-1. Tables ER_{i-1},$ER\ _i$,RLC_{i-1} and RLC_i.

In algorithm 7, f represents the front detection of asegment and is computed with a XOR (noted $\oplus$) while $b$performs a correction of the end of segment. Indeed thisend of segment is detected one pixel after the real segmentend. Memory accesses are also optimized through the introduction of two registers: $x_0 \leftarrow X_i[j]$ the current pixel and$x_1 \leftarrow X_i[j-1]$ the previous one. Such a register rotation(line 13), saves one memory access on four, that is 25%.Note that this algorithm is not data independent but hoped to be faster. The result of its execution is given in figure-3. The algorithm 7 called segment labeling RLC. Such kind of a numbering (Alg. 1 line 10) is known in the field of parallel computing as referring to the "scan"concept [11].

Algorithm 1: LSL segment detection RLC

Input: X_i a binary line of width w
Result: ER_i, RLC_i and ner

```
1. $x_1 \leftarrow 0$ previous value of $X$
2. $f \leftarrow 0$ front detection
3. $b \leftarrow 0$ right border compensation
4. $er \leftarrow 0$
5. for$j = 0$to$w - 1$do up tostep 13
6. $x_0 \leftarrow X_i[j]$
7. $f \leftarrow x_0 \oplus x_1$
8. if$f \neq 0$then do up to step 11
9. $RLC_i[er] \leftarrow j - b$
10. $b \leftarrow b - 1$
11. $er \leftarrow er + 1$
12. $ER_i[j] \leftarrow er$
13. $x_1 \leftarrow x_0$
14. $x_0 \leftarrow 0$
15. $f \leftarrow x_0 \oplus x_1$
16. $RLC_i[er] \leftarrow w - b$
17. $er \leftarrow er + f$
18. $ner \leftarrow er$
19. return $ner$
```

Algorithm 2: LSL equivalence construction

Input: $ER_i - 1, RLC_i, EQ, ERA_{i-1}, ERA_i, ner$
Result: nea the current number of absolute labels, update of EQ and ERA_i

```
1. for er = 1 to ner step 2 do up to step 30
2. $j_0 \leftarrow RLC_i[er-1]$
3. $j_1 \leftarrow RLC_i[er]$
4. [check extension in case of 8-connect algorithm]
5. if $j_0 > 0$ then $j_0 \leftarrow j_0 - 1$
6. if $j_1 < n - 1$ then $j_1 - j_1 + 1$
7. $er_0 \leftarrow ER_{i-1}[j_0]$
8. $er_1 \leftarrow ER_{i-1}[j_1]$
9. [check label parity: segments are odd]
10. if $er_0$ is even then $er_0 \leftarrow er_0 + 1$
11. if $er_1$ is even then $er_1 \leftarrow er_1 - 1$
12. if $er_1 \geq er_0$ then do up to step 24
13. $ea \leftarrow ERA_{i-1}[er_0]$
14. $a \leftarrow EQ[ea]$
15. for $e_{rk} = er_0 + 2$ to $er_1$ do up to step 24
16. $ea_k \leftarrow ERA_{i-1}[er_k]$
17. $a_k \leftarrow EQ[ea_k]$
18. [min extraction and propagation]
19. if $a < a_k$ then do step 20
20. $EQ[ea_k] \leftarrow a$
21. else do up to step 24
22. $a \leftarrow a_k$
23. $EQ[ea] \leftarrow a$
24. $ea \leftarrow ea_k$
25. $ERA_i[er] \leftarrow a$ the global min
26. else do up to step 30
27. [new label]
28. $nea \leftarrow nea + 1$
29. $EQ[nea] \leftarrow nea$
30. $ERA_i[er] \leftarrow nea$
```

Equivalence construction: Step#2

Step #2 is the equivalence construction (Algo. 2).For each segmenter, its boundaries $[j_0; j_1]$ are read fromRLC_i to directly obtain the relative labels of every adjacent segmentin the previous line: er_0 is the label of the first segmentand $er_1$the label of the last segment. As background slicesare labeled with even numbers, a paritycheck is applied to er_0 and er_1 (lines 11,12). The number ofadjacent segments istrivially $(er_1/er_0) = 2 + 1$. If there is anadjacency, the absolute label ea of the first segment is readfrom the associative table ERA_{i-1} that holds the bijectionbetween relative and absolute labels. The ancestor a, which isthe smallest label of the equivalence class is initialized withthe label that is equivalent toea. The propagation of algorithm 2 in $EA_i$and ER_i is shown in Figure-2.

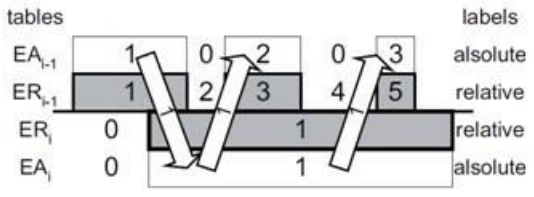


Figure-2. Propagation of absolute labels through to relative labels.

First absolute labeling: Step#3

In first absolute labeling the Absolute label is created with taking values from associative labels with reference as relative labels. Simply the Step#3 consists in replacing the relative label of every segment by its absolute label (Algo. 3). This step is straight forward: ERA_i can be interpreted as a Look up Table to be applied to $ER_i$to create EA_i

Algorithm 3: LSL segment first absolute labeling

```
1. for $i = 0$ to $h - 1$ do step 2
2. for $j = 0$ to $w - 1$ do step
3.   $EA_i[j] \leftarrow ERA_i[ER_i[j]]$o
```

Equivalence Resolution: Step#4

Step #4 is the resolution of the equivalence classes. The Selkow's algorithm [12] is preferred to Rosenfeld's for building equivalences as it is more stable (ie runtime predictable). Yet, both resort to the same algorithm in computing the transitive closures (Algo. 4) .The EQ table is solved and packed into the associative table of ancestors A.

Algorithm 4: Equivalence resolution& pack

```
1. for $e \in [1 : n_e]$ do
2. if $T[e] \neq e$ then
3. $T[e] = T[T[e]]$
4. else
5. $n_a = n_a + 1$
6. $T[e] = n_a$
```

Second Absolute Labeling: Step#5

Step #5 is identical to step#3: every absolute label ea is replaced by its ancestor a (Algo. 5).

Algorithm 5: LSL second absolute labeling.

```
1. for $i = 0$ to $h - 1$ do step 2
2. for $j = 0$ to $w - 1$ do step 3
3. $EA_i[j] \leftarrow A[EA_i[j]]$
```

The above mentioned LSL algorithm is suitable for Connected Component Labeling in Raspberry Pi, it is designed for obtain the fastest real time CCL in RISC architecture.

VOL. 10, NO. 17, SEPTEMBER 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

SIMULATION AND IMPLEMENTATION

This section describes about the LSL algorithms simulation, implementation and performance analysis. The simulation and implementation was done in C++ using the OpenCV image access libraries as well as without it and the Raspberry Pi implementation performance analysis was taken. The Rosenfeld and LSL interface of OpenCV is only for reading, storing and displaying the image.

The Rosenfeld CCL algorithm is a historical algorithm [13]. This method is only for a reference purpose. It is also used to finalize which image interface is better among OpenCV and CImg [14]. The work focused on simulation of Rosenfeld in Visual Studio 2010, and implementation on Raspberry Pi. The input will be an image directly from memory, a .jpeg or .bmp image.
The implementation in raspberry Pi is done in three different ways. One is image loaded form memory. Second is image captured form camera interface and the third is camera interfaced real time image capture. For camera interface the Pi cam is used. It is directly connected to Raspberry Pi B+ CSI (Camera Serial Interface). The output obtained for a normal image reading, form memory is shown in Figure-3.

The LSL algorithm is used to detect the number of components from an image taken from camera. First the image was directly captured from Raspberry Pi inbuilt library and converted to Mat Data type of OpenCV. The result is same as Figure-5. The performance of LSL camera is better than Rosenfeld. It takes only 1.15 seconds for processing a single frame (1980x960). It can be reduced further by a huge factor to obtain a real time manner. The first thing is to reduce the camera access time. For this we adopt a new library RaspiCam [15]. It is much efficient to give almost 30FPS for camera. So the LSL can achieve real-time results. After that the area of each object is calculated in real-time manner.The real-time implementation of LSL is also included with the area calculation (no: of Pixel bounded) and the number of component detection. The performance analysis is taken with 700Mhz Raspberry Pi B+ and obtained 23.26 FPS for QQVGA image capturing and almost 3FPS for VGA basic image capturing. It can improve up to 5 FPS in the case of over clocking (1GHz Turbo mode) the Raspberry Pi. The real time capturing and displaying is shown in Figure-5. The command line result is shown in Figure-6.

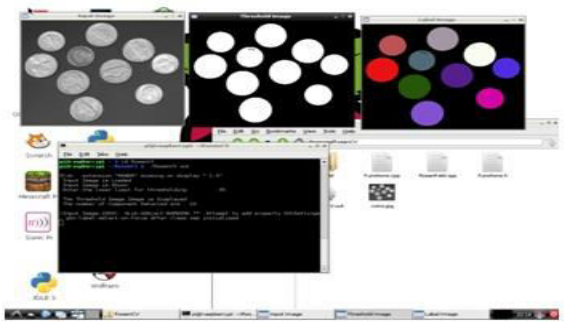


Figure-3. Rosenfeld CCL implementation on Raspberry Pi B+.

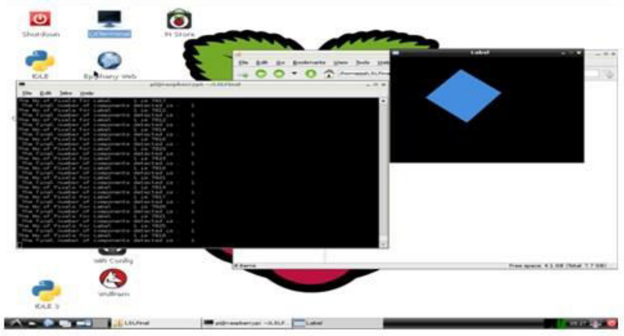
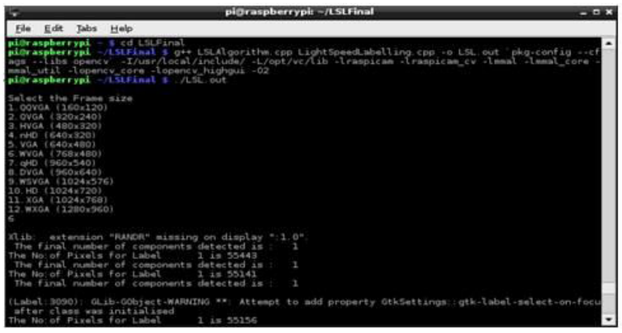


Figure-4. LSL Real Time output.



```
pi@raspberrypi ~ $ cd LSLFinal
pi@raspberrypi ~/LSLFinal $ g++ LSLAlgorithm.cpp LightSpeedLabelling.cpp -o LSL.out `pkg-config --cfl
ags --libs opencv` -I/usr/local/include/ -L/opt/vc/lib -lraspicam -lraspicam_cv -lmmal -lmmal_core -
mmal_util -lopencv_core -lopencv_highgui -O2
pi@raspberrypi ~/LSLFinal $ ./LSL.out

Select the Frame size
1. QQVGA (160x120)
2. QVGA (320x240)
3. HVGA (480x320)
4. nHD (640x320)
5. VGA (640x480)
6. WVGA (768x480)
7. qHD (960x540)
8. DVGA (960x640)
9. WSVGA (1024x576)
10. HD (1024x720)
11. XGA (1024x768)
12. WXGA (1280x960)
6

Xlib:  extension "RANDR" missing on display ":1.0".
 The final number of components detected is :   1
The No:of Pixels for Label       1 is 55443
 The final number of components detected is :   1
The No:of Pixels for Label       1 is 55141
 The final number of components detected is :   1

(Label:3090): GLib-GObject-WARNING **: Attempt to add property GtkSettings::gtk-label-select-on-focu
 after class was initialised
The No:of Pixels for Label       1 is 55156
```

Figure-5. LSL realtime command line.

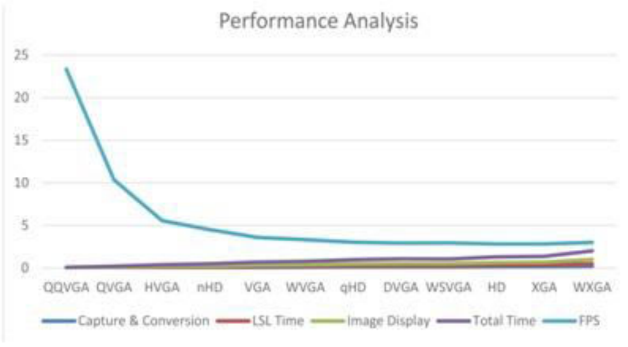


Figure-6. LSL performance comparison.

RESULTS AND DISCUSSIONS

Rosenfeld and Light Speed Labeling are the two algorithms used for the implementation of Connected Component Labelling. Frame pixel comparison in Rosenfeld amounts twice the number of comparison required in LSL. Because of this the Light Speed Labeling is faster and was chosen for final implementation in this work, which was on the Raspberry Pi. Redundant comparison present in Rosenfeld algorithm where optimized for the making of LSL.

The final LSL on Raspberry Pi consist of total comparison less than or equal to number of pixels in a frame. The total time required for LSL to obtain the labeled image for a VGA image is 0.23 Seconds, which is much smaller compared to Rosenfeld. The image interface to the algorithm is performed by OpenCV. The Mat data type is used to store and display the image.

VOL. 10, NO. 17, SEPTEMBER 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

The performance of Connected Component Labelling was measured. For LSL algorithm the main time consuming is Absolut Labeling (Step 3) and Final Labeling (Step 5). Both of these step mainly consist of memory swapping function. Because of this for processing small sized frames required little amount of time. And for large frame size required huge amount of time. The Frame per Second analysis of different size of frames is given in Figure-6. The X-axis gives the size of frames and Y-axis gives the FPS value. From the analysis it is clear that when the size of frame increases the FPS reduces. The time performance analysis is given in Figure-6 similar to FPS when the frame size increases the memory swapping in LSL algorithm, Image capture and Labeled image display time is increases. Because of this for large frame size the execution time is higher.

The real time labeling was implemented using the Raspberry Pi. The camera capturing was done in an enhanced way by adopting RaspiCam library. The final result of LSL in real time implementation shows a 0.345 second for one frame of VGA and an FPS score of 2.8 was obtained. The number of labeled objects are calculated and the pixel area is found in real time execution. The raspberry Pi fails to give real-time result for HD frame. It gives only 1.5 FPS. In the case of QQVGA sized image the performance is 23.2 FPS. This is due to the lower processing power of Raspberry Pi with 700MHz. The over clocking increase the performance of VGA up to 6.2 FPS but results in heating up of the processor.

CONCLUSIONS

Implementation and optimization of Connected Component labeling is successfully done in Raspberry Pi model B+ board. The code level optimization of LSL was done. The real time implementation on raspberry Pi with label counting and area calculation was performed.

The result obtained with LSL is compared with Rosenfeld classical algorithm. The execution time of Rosenfeld real-time with VGA (640x480) requires 1.242824850 seconds but LSL algorithm requires only 0.345444350 seconds. There is 80% percentage of performance improvement for LSL when compared with Rosenfeld. The RAM utilization for both LSL and Rosenfeld are comparable.

The real-time performance of LSL with QQVGA comes around 23.28FPS and for VGA size of image it comes around 3FPS in 700 MHz of Raspberry Pi. The Raspberry Pi B+ board provides real-time performance for frame size less than or equal to VGA and for HD size of frames it shows nearly real-time.

The real time performance can be increased by adopting the new version of Raspberry Pi. Which is Raspberry Pi 2 Model B, it can deliver the performance six time better than the Raspberry Pi B+ version. Which means for the FPS VGA image in real-time will be almost 16-19 FPS. This is equal to a high end computer result.

REFERENCES

[1] S.A.K.Jilani and G.R.S.Manasa. 2014. “Raspberry Pi Based Color Speaker” SSRG International Journal of Electronics and Communication Engineering (SSRG-IJECE) – Vol. 1, No.7.

[2] Parimal Sikchi, Neha Beknalkar and Swapnil Rane. 2014. ‘Real-Time Cartoonization Using Raspberry Pi’ IJCAT International Journal of Computing and Technology, Vol. 1, No. 6.

[3] V. Ajantha Devi and S. Santhosh Baboo. 2014. ‘Embedded Optical Character Recognition On Tamil Text Image’ International Journal of Computer Science Trends and Technology (IJCST) – Vol. 2, No.4, Jul-Aug, pp. 127-131.

[4] Aby P.K, Anumol Jose, Bibin Jose, Dinu L.D, Jomon John and Sabarinath G. 2011. “Implementation and Optimization of Embedded Face Detection System” Proceedings of International Conference on Signal Processing, Communication, Computing and Networking Technologies.

[5] Laurent Cabaret and Lionel Lacassagne. 2014. ‘What Is the World's Fastest Connected Component Labeling Algorithm?’. SiPS: IEEE International Workshop on Signal Processing Systems, Oct. Belfast, United Kingdom, pp.6.

[6] H. Samet and M. Tamminen. "Efficient Component Labeling of Images of Arbitrary Dimension Represented by Linear Bintrees". IEEE Transactions on Pattern Analysis and Machine Intelligence (TIEEE Trans. Pattern Anal. Mach. Intell.) Vol. 10, p. 579.

[7] Rosenfeld, J.L. Platz. Sequential operator in digital pictures processing, Journal of ACM, Vol. 13,4, pp 471-494, 1966.

[8] R. Haralick. 1981. “Some neighborhood operations,” Real Time/Parallel Computing Image Analysis, pp. 11–35.

[9] Lacassagne, Lionel and Bertrand Zavidovique. 2011. "Light speed labeling: efficient connected component labeling on RISC architectures." Journal of Real-Time Image Processing 6, No. 2, pp. 117-135.

[10] Lacassagne, Lionel and Bertrand Zavidovique. 2009. "Light Speed Labeling for RISC architectures." In ICIP, pp. 3245-3248.

[11] G.E. Blelloch. 1990. Vector Models for Data-Parallel Computing. The MIT Press, Cambridge Massachusetts.

[12] S.M. Selkow. 1972. One pass complexity analysis of digital pictures properties, Journal of ACM, Vol. 19, No. 2, pp. 283-295.

[13] Nirmal T M, Rajeev K. and K. R. Joy. 2015. ‘Implementation and Optimization of Core Computer Vision Function in Raspberry Pi’ IJAER, Special Issue, Vol. 10 No.20 PP 15932-15937.

[14] http://cimg.sourceforge.net/reference/.

[15] https://github.com/cedricve/raspicam.