IEEE Instrumentation and Measurement
Technology Conference
Anchorage, AK, USA, 21-23 May 2002

Size/Position Identification in Real-Time Image Processing using Run Length Encoding

C. H. Messom[1], S. Demidenko[2], K. Subramaniam[2] and G. Sen Gupta[3]
[1]II&MS, Massey University, Albany, New Zealand
[2]IIS&T, Massey University, Palmerston North, New Zealand
[3]School EEE, Singapore Polytechnic, Singapore
Email: C.H.Messom@massey.ac.nz, S.Demidenko@massey.ac.nz, Karthik@xtra.co.nz, G.SenGupta@sp.edu.sg

Abstract *– This paper presents the use of Run Length Encoding (RLE) for real-time image processing. RLE compresses the image and allows the image processing algorithm to efficiently identify the size and position of the objects in the image. The algorithm is application to a robotic vision system enable to reliably identify 11 objects in the 16.67ms sample time of an interlaced NTSC video image. It is shown that the time complexity of the compression phase is linear in the size of the image while the image processing phase is linear in the number of object in the image and the number of lines occupied by the objects in the image.*

Keywords *– Real-time Image processing, Compressed Image processing, Robotics Vision System*

I. INTRODUCTION

Vision systems are the primary sensor input to advanced robotic applications such as mobile robotics, agricultural robotics and inspection systems [1]. However, vision causes a significant processing burden in intelligent robotics since the majority of image processing algorithms are computationally extremely expensive. For this reason any improvement in the vision algorithm leads to significant benefits in terms of achieving real time performance.

Till now reliable real-time performance has only been possible using dedicated hardware implementations. Vision systems implemented on the basis of the dedicated hardware (for example, using special full custom design ASICs) are fast and reliable however, they are inflexible and expensive.

Intelligent robot control systems that make use of commodity hardware (readily available, or programmable systems) can be flexible and cheap. However they normally suffer from limited processing speed. Most commodity vision systems use video signals as input to the image capture subsystems and so provide frame rates of approximately 30Hz and field rates of 60Hz for interlaced images. Processing these images with useful resolution (240 x 320 and above) in the 33.3 ms and 16.67 ms sample times respectively represents a significant challenge. The sample time cannot be devoted completely to vision since the intelligent control algorithms will not be able to complete within the real time constraints.

In general, image processing algorithms are applied to uncompressed images. The reason for this is that most of the popular image compression techniques distort the color, size and position of the objects in the image that can only be recovered by decompressing the image. Processing of compressed images in real time also is problematic when the compression algorithm is computationally expensive.

Paper [2] reported a real-time vision system using incremental tracking, whereby a window around each object is created. Once the location of an object has been determined, its movements are tracked only within the window. The position of this window is adjusted each frame after identifying the new position of the object.

This technique is used mainly due to the savings it offers in processing time, as the whole image of the playing field does not need to be analyzed. Instead, only the tracking windows for each object need to be analyzed. The technique analyzes a very small proportion of the field, leading to the intermittent problem of losing objects that stray outside their tracking windows. This problem is particularly hard to solve for fast moving unpredictable objects. Nevertheless, this method was used in order to achieve real-time image processing.

This paper examines the processing of compressed images in real-time using the Run Length Encoding (RLE) algorithm. The algorithm is not computationally expensive. It allows an image to be encoded with just a single access to each pixel. The RLE algorithm was applied to the problem of finding the difference between two binary images in [3]. It was shown that an algorithm could be efficiently implemented using systolic architecture. Such an implementation has the advantage that the computational complexity (and the execution time) increases just linearly with the size of the image.

The RLE algorithm can be implemented on commodity hardware in the robotic application domain (for example, in robot soccer visual systems) [4, 5]. This paper examines the *full* image processing of Run Length Encoded 24-bit RGB images using commodity hardware. It shows that once the compression phase is completed, the algorithm is essentially linear in the number of objects and size of the image. The

implementation discussed in section V is more than four times faster than that reported in [5].

The RLE encoding algorithm preserves the position, size and color of the objects in the image. The compressed image is processed to identify the size and positions of the objects in the image in real-time. RLE is a loss-less compression technique if the exact pixel values are preserved. The RLE compression used in this paper produces significant losses since a range of RGB values are represented by a single color value. The fact that the original image cannot be recovered by the RLE compression is not a problem in this application since, the compressed image is not required for any post-processing other than object recognition. It is shown that the performance of the image processing on the compressed image is significantly faster than standard threshold based techniques when used for robotic vision applications.

II. RUN LENGTH ENCODING

The RLE algorithm takes a line of pixels in the image and identifies the matching color identifier for each pixel. The color identifiers represent a set of RGB values that form a convex subspace of the 24-bit RGB color space. A convex partition of the RGB color space must be provided for each color identifier before this can be achieved (see Figure 1). This convex partition is specified by a range of RGB values namely, R_{min} - R_{max}, G_{min} - G_{max}, B_{min} - B_{max}.

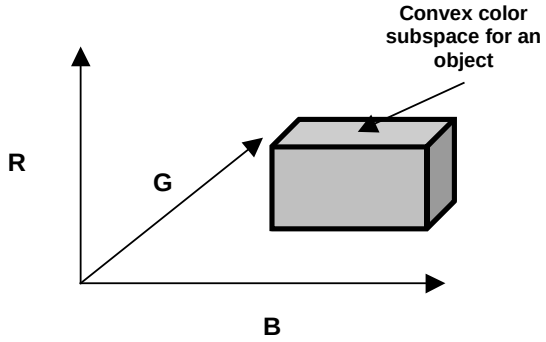

Figure 1. Convex color subspace

The matching color partition for each color identifier is created based on the expected color of the object in the image, which is dependent on the actual color of the object and the lighting conditions. Given a fixed set of objects and controlled uniform lighting, a constant set of convex color partitions can be defined.

Once each pixel has been classified as being part of a particular object's color partition, a line of pixels is run length encoded by identifying the start and end position of each object (Figure 2). The start and end position of the line of pixels that are from a single object and the color identifier of the object forms the components of the run length element given in Equation 1.

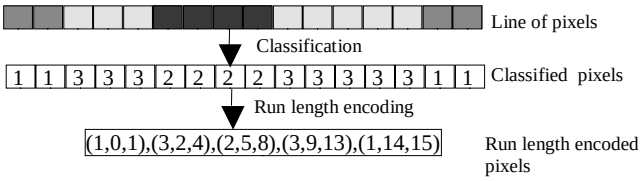

Figure 2. Run Length Encoding of a line of pixels

Run Length Element = (ColorID, StartIndex, EndIndex) (1)

For objects of less than 3 pixels in size the compression algorithm is very inefficient. This is because the RLE algorithm requires 3 integers to code each object. However the existence of small objects does not necessarily make the algorithm inefficient for the whole image since large objects (larger than 20 pixels in width), including the background, have compression ratios in excess of 75%. If the image has only small objects the compression ratio of the objects' pixels will be small, however the background will be very large in such images producing a large overall compression ratio. If the background is not of interest for the image processing algorithm then the compression phase can skip over the background pixels increasing the compression ratio still further.

The compressed image is then processed to identify the objects in the image, their sizes and their positions. The run length encoded lines are compared to identify the adjacent elements with identical color classifications (Figure 3). These run length elements are tagged with object identifiers (Equation 2) that are unique for each element.

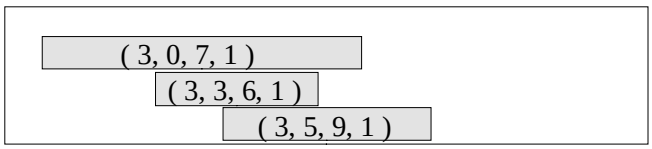

Figure 3. Grouping of objects in compressed image

Tagged Run Length Element =

(ColorID, StartIndex, EndIndex, ObjectID) (2)

The run length elements that have no neighbors of the same color are allocated a new object identifier (ObjectID). If the run length element is adjacent to a run length element of the same color it is assigned the same object identifier as the neighbor. This approach guarantees that each object of a particular color is tagged with a unique object identifier.

Starting from the top of the image the object identifiers can be consistently assigned. However when a run length element has more than one adjacent element in the previous row (Figure 4) two object identifiers can exist for a single object. This is resolved by re-labeling the second neighboring element with the correct object identifier.

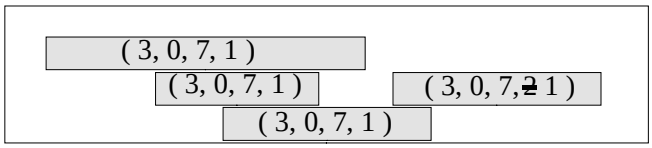

Figure 4. Reassigning object identifiers

Once the elements have been correctly classified with unique object identifiers, they are grouped together by totaling up the run length elements that share the same object identifier. This provides the size of each object. The position of each object is determined using a center of gravity calculation based on the size and position of the run length elements. The size and position of the run length elements is given by the start and end position of the run lengths and the line in the image in which they occur.

III. THE ALGORITHM

The image analysis algorithm has been implemented using two data structures, 'RLE_element' and 'Neighbors'. Each instance of the first structure, 'RLE_element', holds the information for a single sequence of pixels of the same color. If a run of pixels of a particular color occurs, an instance of the 'RLE_element' structure stores the location and length of the run of pixels, the object ID number, the color partition the object falls into, and whether it has any neighbors of the same classification (Figure 5).

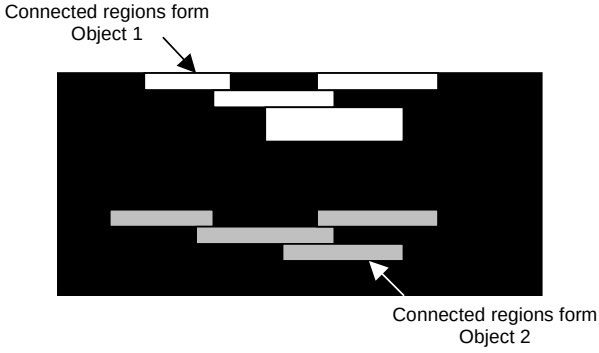

Figure 5. Neighboring RLE elements forming objects

An instance of the second structure, 'Neighbors', is used to store for each RLE_element, the locations of all neighboring elements falling into the same color partition. Thus, when a run of pixels is recorded in a RLE_element structure, the region around the run of pixels is also checked for other runs of pixels, with the same color categorization. Every run that falls into the same color category is stored in the current object's corresponding 'Neighbors' structure. The structure holds all the neighboring objects' locations in an array (Figure 6).

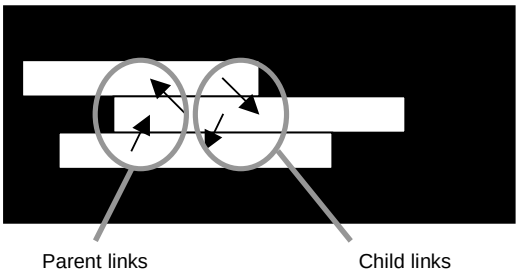

Figure 6. Links to neighboring RLE elements of identical color

The whole program passes through the image twice: firstly compressing the image and secondly - processing the encoded image. The first pass analyses each row of pixels, until the end of the captured image is reached. In each row, the runs of pixels are classified into their respective color partitions.

When an object other than the background is encountered, two processes are executed. Firstly, the object is assigned the next available object ID. Next, the surrounding region of the current object is checked for other objects of the same color category. If any exist, they are considered connected, and belong to be the same object. The connected region is recorded as a neighbor of the current object, and vice versa. The background, or playing field, is not recorded with the 'neighbors' field.

The second pass through the image selects each object, and uses it as the root of a tree. From this root, a recursive depth-first traversal is performed on all the connected regions that have been recorded as 'Neighbors' in the first pass. In this process, all connected regions will be assigned the same object ID, as the tree root. If the connected regions already have been assigned an object ID, the tree overwrites its object ID. The result of this second pass through the image is that all connected portions are identified with the same object. The flow chart in Figure 7 illustrates the processes in the algorithm.

IV. ALGORITHM OPTIMIZATION

The RLE compression phase examines each pixel and identifies the corresponding object color based on the predefined convex color space (Figure 1). This calculation can be achieved by 6 comparisons of the pixel's RGB value with the defined range of the convex color space. An optimization that has been applied [5] is to use a lookup table to extract the

required color ID. The tradeoff of this approach is that the improved processing speed comes at the expense of memory required. For a 24 bit RGB system a lookup table of size 16MB is required (256*256*256 bytes). This lookup table must be created off-line as the time taken to create such a large lookup table is significant. A further disadvantage with the lookup table method is that an adaptive object color partition is difficult to implement. (An adaptive object color partition may be required in systems where the lighting conditions change over time).

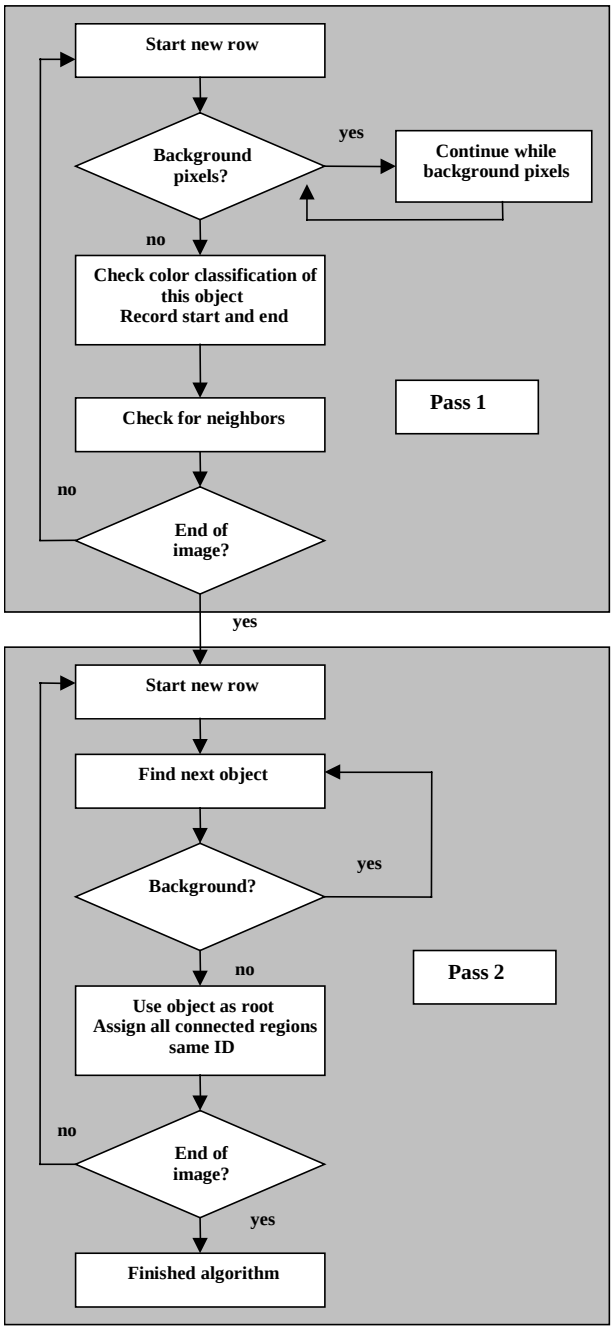

Figure 7. Image processing flow chart

A method to overcome the large space requirement of an RGB lookup table was originally proposed by [5]. This required three lookup tables, one for each R, G and B component. These three lookup tables are projections of the 3 dimensional RGB lookup table into the R, G, and B axes. Given a particular pixel value, each lookup table will return the set of color identifiers that match that pixel value. The three returned sets can then be intersected to identify which actual object color corresponds to the given RGB value.

The sets can be implemented by representing each object color with a binary identifier (1, 2, 4 etc) and returning a set of possible object colors by bit-wise OR of the required object IDs. For example, if the R component value of 55 corresponded to object color 1 and 4 then the R lookup table will return 5 (which is 1 bit-wise OR 4).

The set intersections can then be implemented with the bit-wise AND of the results from the three (R, G and B) lookup tables (Equation 3).

Color ID = R[r] bit-wise AND G[g] bit-wise AND B[b] (3)

Continuing the example from above, if a G component value of 100 corresponds to object color 1 and 2 while, a B component value of 125 corresponds to object 1 and 8, then the RGB value of (55,100,125) will correspond to object 1.

The size of each R, G and B lookup table depends on the number of object colors required. One byte can accommodate 8 object colors. A system with 8 object colors will require each lookup table to be 256 bytes, a total of 768 bytes for the three R, G and B lookup tables. The 16MB RGB lookup table and the three 256 byte R, G and B lookup tables were tested in software on the current system. As expected there was not any significant difference in time performance between the two methods, but there was a significant improvement in the memory space required.

V. PROTOTYPE APPLICATION

Robotic soccer has been chosen as the test bed for the proposed algorithm and its software implementation using the commodity hardware. In brief, the robot team plays against the opponent team on a specified size field. All the team robots are equipped with wireless communication facilities to receive the instructions from the central computer. The computer controls the robots by transmitting information to them regarding the location of the ball, where the robot must go next, where its opponents are, and where its teammates are. The robots operate using a common vision system, which views the playing field from above (Figure 8). The identified colored objects are grouped together to produce positions and orientation of the robots and obstacles in the image.

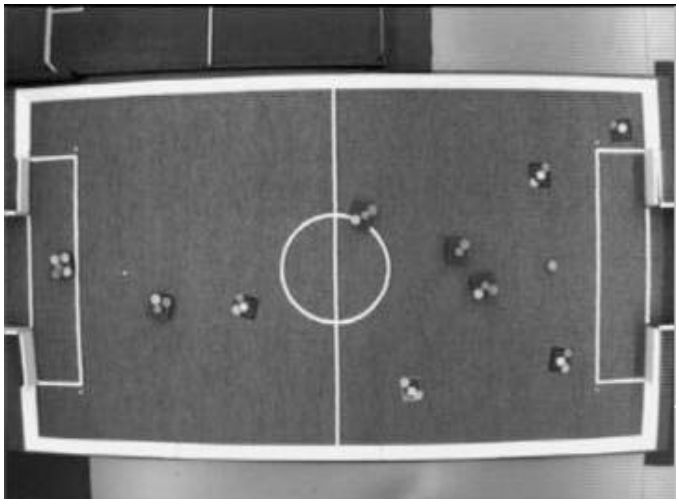

Figure 8. Robot Soccer Image

Classifying the objects in the image into opponent robots, own robots, or the ball is done by recognizing colored patterns on the upper faces of the robots. This is the final process in the overall image processing algorithm, following the first phase – the object location. The robot controller then makes decisions on its game strategy based on the positions of all the players and the ball and transmits the required information to the mobile robots.

VI. RESULTS

The proposed algorithm was tested on commodity hardware: a Flashbus Pro image capture card [6], 1GHz Pentium III with 256MB RAM. The signal was an interlaced NTSC video, each half frame delivered at 60Hz. Using an image of 240 x 320 pixels the system was used to identify the size and position of 32 colored objects. The processing consumed 5 ms of CPU time a significant improvement on the 33 ms reported in [5].

The technique presented here compares favorably to the color threshold based approaches discussed in [7] which, take more than 33ms on the hardware above. The time performance of the RLE image processing algorithm compares favorably with the incremental tracking, real-time approach presented in [2]. However, the main advantage over incremental tracking is that RLE image processing is a full image scanning technique and therefore does not suffer the problem of losing track of fast moving and unpredictable objects.

The RLE algorithm's time complexity is linear in height and linear in width of the image since the each pixel is processed once. The image processing of the compressed image is linear in number of lines occupied by each object and the number of objects in view since the background is skipped over and the grouping process acts on the run length elements of the objects in view. The image processing phase of the algorithm scales extremely well. If the number of objects in the image is fixed, increasing the resolution of the image will only incur a linear cost in image processing time.

For a small number of small objects in a high-resolution image the RLE phase represents the most time consuming part of the algorithm. This can be seen from the example of 11 objects each occupying 16 lines of a 960 x 1280 pixel image. The 1,228,800 pixels are processed by the RLE phase while only 176 (which is 11 objects * 16 lines) run length elements are processed by the grouping phase of the image processing algorithm. Since each operation in the grouping phase is not significantly more computationally expensive that the RLE phase the difference in the number of operations means that the RLE phase dominates the processing time as compared to the image processing phase.

VII. CONCLUSIONS

An application of run length encoding (RLE) to real time image processing was presented. It was shown that the time complexity of this algorithm is very good, particularly in the case of robotics applications where there are a small number of small objects. The algorithm can process a 240 x 320 image within 5 ms which, with the linear increase in processing time with respect to the size of image, we expect the algorithm to process a 480 x 640 resolution image within the 16.67 ms sample time of an interlaced NTSC system.

Further performance improvements can be achieved with the RLE algorithm by modifying the image capture process (as discussed in section IV). If the image's RGB components are fed through look-up-tables that classify the component into the possible objects in view then the RLE algorithm is reduced to just two bit-wise AND operators for each pixel. Implementing this operation in hardware is expected to contribute significantly to the scalability of the RLE image processing system, which will be investigated in future work.

REFERENCES

[1] R. Templer, H. Nicholls and T. Nicolle, "Robotics for meat processing - from research to commercialisation", Industrial Robot, Vol 26, Number 4, 1999.

[2] C.H.Messom, G.S. Gupta and H.L. Sng, "Distributed Real-time Image Processing for a Dual Camera System", CIRAS 2001, Singapore, 2001 pp 53-59.

[3] F. Ercal, M. Allen, and F. Hao, "A Systolic Image Difference Algorithmfor RLE-Compressed Images", IEEE Transactions on Parallel and Distributed Systems, Vol. 11, No. 5, May 2000.

[4] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa and H. Matsubara, "RoboCup: A Challenge Problem for AI", RoboCup-97: Robot Soccer World Cup I, Springer Verlag, London, 1998.

[5] J. Bruce, T. Balch and M. Veloso, "Fast and Inexpensive Color Image Segmentation for Interactive Robots", IROS 2000, San Francisco, 2000.

[6] http://www.integraltech.com/Products/FrameGrabbers.html

[7] J. Baltes. "Practical camera and colour calibration for large rooms". In Manuela Veloso, Enrico Pagello, and Hiroaki Kitano, editors, RoboCup-99: Robot Soccer World Cup III, pages 148-161, 2000. Springer.