Fast Image Capture and Vision Processing For Robotic Applications

Gourab Sen Gupta, Donald Bailey

School of Engineering and Advanced Technology
Massey University, Palmerston North
New Zealand

Abstract This chapter details a technique to significantly increase the speed of image processing for robot identification in a global-vision based system, targeted at real-time applications. Of major significance are the proposed discrete and small look-up tables for Y, U and V color thresholds. A new YUV color space has been proposed which significantly improves the speed of color classification. The look-up tables can be easily updated in real-time and are thus suitable for adaptive thresholding. The experimental results confirm that the proposed algorithm greatly improves the performance of the image processing system. The results are compared with other commonly used methods such as a composite look-up table which is indexed using RGB pixel values.

Keywords global vision, colour segmentation, YUV colour space, incremental tracking

1 Introduction

Vision systems are widely used in the industry for object tracking, intrusion detection, vehicle and mobile robot guidance, inspection automation, etc. [1]. The majority of the commodity vision systems use video signals, most often from a CCD camera, as input to the image capture and analysis subsystems. Typically, such vision systems provide frame rates of 30 Hz

or field rates of 60 Hz for interlaced images. Processing these images with useful resolution of 320 x 240 and above in the 33.3 ms and 16.67 ms sample times respectively can pose a significant challenge especially when other processing tasks such as strategy and task allocation, low-level control and communication with the robots, are also to be completed. A lot of research efforts have been spent on improving the speed of image processing for robotic applications [2] and it continues to attract a lot of attention of researchers. Faster vision processing algorithms result in better motion control and hence better coordination between agents to accomplish a collaborative task.

A computationally inexpensive vision processing algorithm using Run Length Encoding (RLE) has been discussed in papers [3] and [4]. RLE is an image compression technique that preserves the topological features of an image, allowing it to be used for object identification and location [5]. Though the RLE algorithm can be implemented on commodity hardware for multi-agent collaborative systems such as the robot soccer vision system [6, 7], its significant processing speed advantage is when there are a large number of objects to track. For smaller systems with a limited number of agents, say two to five, the commonly used blob identification techniques together with the incremental tracking algorithm is adequate and equally efficient.

Interlaced images introduce the 'image scattering' problem for a moving object because of the time delay between the two fields of the image. To overcome this problem, the odd and even scan fields have to be processed separately. However, because of quantization errors in each field, a stationary object may appear to be in two different locations. Filtering techniques are required to minimize such a negative effect. The other sources of errors in the vision system are due to variation of light intensity and inherent sensor noise.

The cumulative effect of the errors in the vision system is very significant, especially in a highly dynamic collaborative system where the agents are moving very fast. Figure 1 shows the software hierarchy of a multi-agent robotic controller using global vision. The image processing software identifies the position and orientation of the robots and other objects of interest in the robots' workspace. The vision data is filtered to reduce the effect of noise and passed on to the strategy/task allocation layer. At this layer of the software architecture, the behavior required of an agent is determined. The errors in the vision system percolate down the hierarchy of the robotic controller and have profound effect on the robot behavior and hence the overall performance of the collaborative system. It is thus imperative that careful considerations are given to eliminate or at least minimize the vision processing errors.

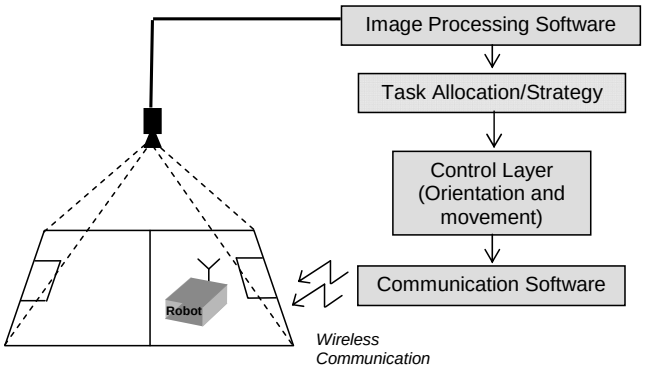

Fig. 1. Software hierarchy of a multi-agent robotic controller using global vision

2 Global Vision – Sources of Error

A global vision system uses a single or multiple cameras to detect and track several objects. The main sources of errors in the vision system are described in the following sub-sections.

2.1 Separate processing of odd and even scan fields of an interlaced bit mapped image

A stationary object may be reported at different locations in each frame due to different quantization errors. This is explained using a bit mapped image of 16x16 pixel resolution as shown in Figure 2. The object of interest, the centre position coordinates of which are required to be calculated, is of a square shape.

To calculate the position, the well known zero-order moment and centre-of-gravity equations (1) are used. For an m x n binary image, the coordinates are given by:

$$\text{Area, } A = \sum_{i=1}^{n}\sum_{j=1}^{m} B[i,j]$$

$$\text{COG, } \bar{x} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} jB[i,j]}{A} \qquad \bar{y} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} iB[i,j]}{A} \tag{1}$$

B[i, j] is the value of the bit (0 or 1) in the image at location [i, j].
i is the row number (i.e. Y-coordinate)
j is the column number (i.e. X-coordinate)

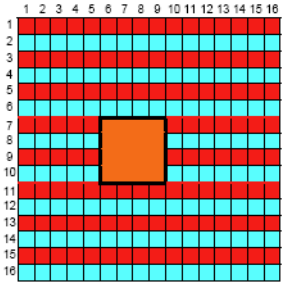

Fig. 2. Bit-map of an interlaced image

For the illustrated example, the coordinates of the object in the Odd scan is (7.5, 8.0) as shown in the calculations below:

$$A = 8$$

$$\bar{x} = \frac{6\times 2 + 7\times 2 + 8\times 2 + 9\times 2}{8} = 7.5$$

$$\bar{y} = \frac{7\times 4 + 9\times 4}{8} = 8$$

The coordinates of the object in the Even scan is (7.5, 9.0) as shown in the calculations below:

$$\bar{x} = \frac{6\times 2 + 7\times 2 + 8\times 2 + 9\times 2}{8} = 7.5$$

$$\bar{y} = \frac{8\times 4 + 10\times 4}{8} = 9$$

The separate processing of odd and even scan fields do not affect the X Coordinate. Only the Y Coordinate has a shift of 1 pixel. Looking at a physical area of 170 cm x 150 cm and working with an image resolution of 320x240 pixels, 1 pixel translates into a shift of ~0.62 cm in the Y Coordinate of the object. Moreover, this offset will not be constant for a moving

object since the shift can occur both in a positive or negative direction. Filtering techniques are often employed to minimize the quantization error.

2.2 Variation of light intensity

While errors due to separate processing of odd and even scans could be pre-dominant under controlled light conditions, nonetheless, these errors are further compounded by variation in light intensity from one frame to another. In the real-world applications of collaborative robotics, it is often not possible to create ideal (or at least stable) light conditions. To partially overcome this problem, YUV color thresholding can be employed with the Y (intensity) range extended to the maximum limits. However, extending the color boundaries too much result in the threshold values of two or more colors overlapping each other. Also extending the threshold values wider will make the vision system error prone as stray pixels from the background will be picked up too. This limits the color tolerance.

2.3 Inherent Sensor Noise

Certain errors are inherent in the system. These originate from the camera, the frame grabber card, connecting cables, etc.

Errors in vision-generated data have a significant impact on targeting accuracy even when intercepting or striking a stationary object. The tests carried out on moving targets, however, are more significant as interception accuracy suffers when the target is moving. This is due to the fact that actions are initiated based on predicted future positions which are different from the current position and are calculated based on velocity measures which are very noisy.

3 Experimental Hardware Setup

The experimental hardware setup consists of a Pulnix 7EX NTSC camera (www.pulnix.com) with analog composite video output and a FlashBus MV Pro image capture (frame grabber) card with PCI interface (www.integraltech.com). The image is captured at a resolution of 320x480 at a sampling rate of 30Hz. The odd and even fields are processed separately; hence the effective image resolution is 320x240 delivered at a sampling rate of 60 Hz. The captured image is processed on a 1.8 GHz Pentium 4 PC with 512 MB RAM. The image capture card was configured for

off-screen capture, as shown in Figure 3. Off-screen capture mode facilitates fast processing of the image from the system RAM. The image is transferred to the VGA RAM only when a live image is required to be seen on the screen, such as during the setting and testing of color thresholds. Once the color tuning is done, the transfer of image to the VGA RAM is switched off.

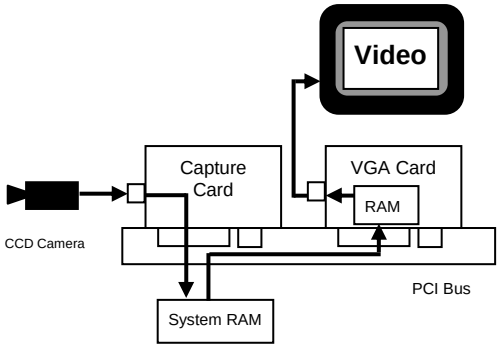

Fig. 3. Image capture card in off-screen capture mode

An important hardware feature of the FlashBus MV Pro frame grabber card is that it generates a Vertical Sync for every odd and even field. The interrupt can be detected in the software. This facilitates implementation of an interrupt based system and fixes the sample time of the controller.

4 Colour segmentation, area thresholding, blob merging

The color image sequence is processed at three levels: pixel level, blob level, and object level. To facilitate identification and separation of individual objects, a color jacket comprising two color patches can be used on each robot, as shown in Figure 4. In applications where several groups or teams of robots are involved, one of the color patches is used to identify the group (team color patch) and the other is used to identify which robot it is within the group (robot color patch). The centers of the two color patches, Cr and Ct, are first calculated from the image. The inclination of the line joining the two centers gives the orientation of the robot while the coordinates of the centre of the line give its position. For some target objects such as a ball in a robot soccer system, only the centre of the color patch is calculated as it does not have an orientation. The velocity of the target is used to add a direction vector to its position.

The accuracy of the angle calculation depends on the accuracy with which the centers of the color patches are detected and how far apart these centers are. If the centers are closer to each other, any small variation in the calculation of Cr and Ct will result in a large variation in angle. In order to improve the accuracy of angle calculation, experiments with different color jackets were performed. The centers of the color patches in Figure 5 are further apart than those in Figure 4, thus improving the accuracy of the angle calculation.

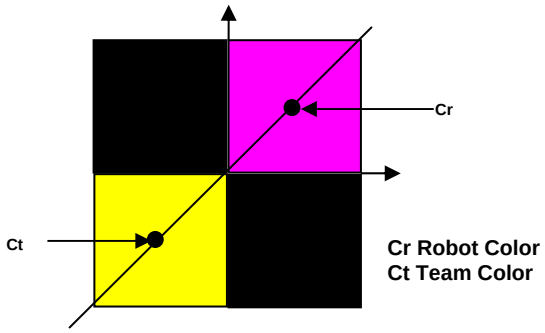

Fig. 4. Color jacket for Identification of Robot

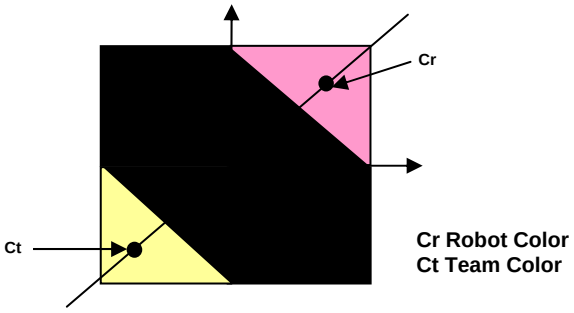

Fig. 5. Color jacket for improved accuracy of orientation

The object detection algorithm starts with color segmentation. It searches the image to determine if the pixels belong to one of the calibrated color classes. The pixels are then grouped to create color patches using a 'sequential component labeling algorithm'. This algorithm uses a two-pass labeling technique [8] with identifiers (labels) that increment from the value of 1. Ideally, the number of labels is equal to the number of desired color patches on the objects in the entire image.
The procedure for checking the membership and grouping of pixels consists of 5 steps split into two passes.

A. FIRST PASS (Steps 1 to 4)

1. Process the image in the tracking window from left to right, top to bottom, analyzing each pixel.
2. If the pixel in the image is within the YUV threshold values of the color of interest, then
   (a) If only one of its upper and left neighbors has a label, copy the label.
   (b) If both upper and left neighbors have the same label, copy that label.
   (c) If both upper and left neighbors have different labels, copy the upper pixel's label and enter the labels in an equivalence table as equivalent labels.
   (d) If not (a), (b) or (c) assign a new label to this pixel and enter it in the equivalence table.
3. If there are more pixels to consider, repeat step 2 for additional pixels, otherwise proceed to step 4.
4. Find the lowest label for each equivalent set in the equivalence table and add to the equivalence table.

B. SECOND PASS (Step 5)

5. Process the picture by replacing each label with the lowest label in its equivalent set.

To illustrate the algorithm, a binary image (rather than a YUV image) is used in the following example. Figure 6 shows a representation of the binary image before the first pass of the algorithm. The 0's represent the background of the image and the 1's represent the objects of interest. It can be seen that there are two objects of interest in the image, one in the left and the other in the right.

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	1	1	1	1	0	0	0	0	1	1	0	1	0
0	1	1	1	1	1	1	0	0	1	1	1	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 6. The binary image

Figure 7 shows the image after the first pass of the algorithm. The objects now have multiple labels as the first pass of the algorithm was not able to correctly label all shapes (i.e. the object on the left has labels 3 and 1 and the object on the right has labels 2 and 4). Figure 8 shows the image after the second pass of the algorithm, which resolves the problem of multiple labels for single objects.

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0	0	0	2	0
0	3	1	1	1	0	0	0	0	4	4	0	2	0
0	3	1	1	1	1	1	0	0	4	4	4	2	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 7. The image after the first pass

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0	0	0	2	0
0	1	1	1	1	0	0	0	0	2	2	0	2	0
0	1	1	1	1	1	1	0	0	2	2	2	2	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 8. Image after the second pass

The algorithm described above is often called the '2-neighbour' algorithm in which the upper and left neighboring pixels of the pixel under test, are considered to evaluate the membership (label) of a pixel. Another well know and very similar algorithm is called the '4-neighbour' algorithm in which the four neighboring pixels – upper, left, bottom and right – are considered for evaluating the label of a pixel. This algorithm is more time consuming and gives only marginal improvement in the accuracy of color segmentation. Thus the '2-neighbour' algorithm is often preferred over the '4-neighbour' algorithm, especially in real-time applications where the color patches are relatively large in pixel area.

Once the colors have been segmented, two separate processing steps follow – filtering based on area threshold and blob merging. In order to reduce noise, very small color blobs are discarded if they fall below a certain

area threshold. For example, it is usually safe to discard patches which are only 2 or 3 pixels in area as these would generally be noise from the background. If patches of the same color are very close to each other, then these are merged to form one patch. This blob merging technique, with a distance tolerance, is widely used in practice and reported in literature [9, 10].

Having identified the separate objects, the centre of each object is calculated using the centre of gravity calculation described earlier (Equation 1).

5 Interrupt based multi-buffered image capture

In order to fix the sample time and delay of the vision control loop to a constant value, an interrupt driven approach is adopted. In this method the vision processing and strategy functions are placed in an interrupt service routine. The routine is serviced on each occurrence of the Vertical Sync signal on the frame grabber card which generates a Vertical Sync for every odd and even field. For an image resolution of up to 640x480, the card can capture the image at 30 frames per second. Hence the interrupt service routine of the interlaced image is executed every 16.67 ms.

This poses a challenge - all the vision processing, strategy calculations, calculation of motion control data and transmission of RF packets to all the robots, must be completed within 16.67 ms. This is achieved by segregating the process of capturing the image and processing it. This is implemented using a four-stage ring buffer to capture and process the image. If the image processing is guaranteed to complete in the specified sample time only two buffers are required. Since this is not the case for the color segmentation and blob-identification algorithm presented in section 4, a four buffer system is required. The buffer organization is shown in Figure 9. When an image is captured in a buffer, the image from the previous buffer is processed. This helps to avoid buffer contention during image capture and processing [11].

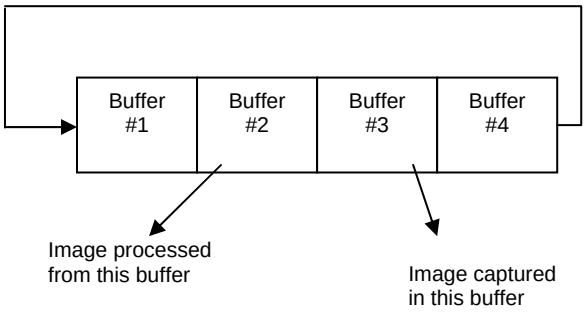

Fig. 9. Multi-buffered image captures

6 Full Tracking vs. Incremental Tracking

The full tracking algorithm searches through the whole image, testing each pixel whether it is a member of one of the calibrated object color classes. The full tracking process is very inefficient when the objects are small and only represent a small percentage of the whole image. For increased efficiency of image processing, incremental tracking is generally employed [12]. Incremental tracking is the approach whereby a small region around the last known position of the object (Figure 10) or its predicted position (Figure 11) is processed rather than the whole image, thereby decreasing the processing time or computational resources required. By limiting the objects to be tracked, it is possible to increase the incremental tracking window size and yet keep the vision processing time within the sample period of 16.67 ms. Using larger incremental tracking window sizes, the object being tracked is nearly always identified. In the event that the object is "lost" (i.e. it is not inside the predicted tracking window), the fault tolerant software reverts to the full-tracking mode, whereby the whole image is analyzed to 'recover' the object's position.

To locate the object positions the first time, the image must initially be scanned fully for one frame. This will identify the starting position of all the objects within the field of view. Then the incremental tracking algorithm can continue.

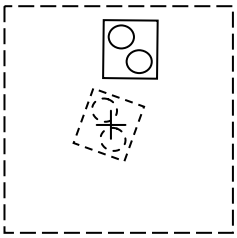

Fig. 10. Tracking window centered on last known position of object

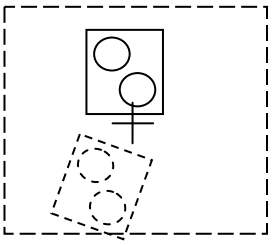

Fig. 11. Tracking window centered on predicted position of object

The number of pixels that must be processed is related to the size of the tracking window and the number of objects being tracked. This technique is significantly more efficient than full tracking if the sizes of the tracking windows are much smaller than the size of the image divided by the number of objects.

Reducing the sample period of the system can drastically reduce the required size of the tracking window, since the objects would have moved a shorter distance in the shorter sample time. In a sample case, halving the sample period would have the objects move half the distance, so the length of the tracking window can be halved. This gives an area reduction of a factor of four. Therefore doubling the frame rate (i.e. halving the sample period) would reduce the execution time of this algorithm by a factor of four. Thus, paradoxically, this algorithm is more likely to complete in the required sample time if the frame rate is higher.

In a system that has a reliable frame rate, the last known velocity of the object, rather than the last known position, can be used to centre the tracking window on the predicted position of the object. Using this method, the size of the tracking window can be reduced further as the error in the predicted position will depend only on the uncertainty in the measured velocity. The only risk associated with reducing the tracking window size relates to collisions, which can alter the velocity of the objects significantly.

In robot soccer, the ball is the lightest and fastest object and so is at the greatest risk of being 'lost'.

Several techniques to overcome this problem have been proposed in the literature [13] based on predicting collisions of fast moving light objects. Figure 12 shows the movement of the tracking window as the robot moves.

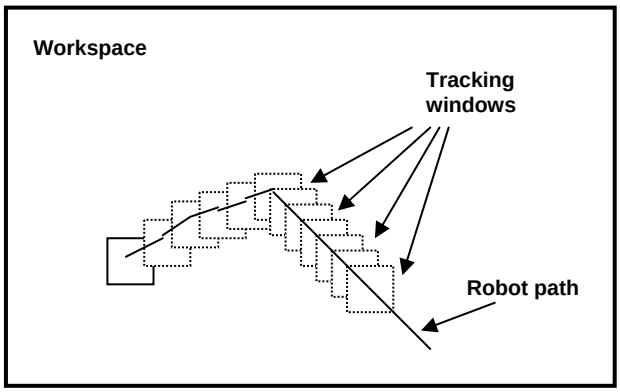

Fig. 12. Robot Path and the incremental Tracking window

Adaptive Tracking Window Size

Enlarging the tracking window for the faster and lighter objects can produce reliable tracking results. A minimum tracking window size may be defined for a stationery object. As the object starts to move, the size of the tracking window will increase proportionately and adapt to the object's velocity. This will ensure that larger the velocity, bigger the tracking window size and lesser the possibility of 'losing' it. This will greatly increases the reliability of tracking. However, the downside is that even for a fixed number of objects, the number of pixels that need to be processed is not constant and can vary a lot depending on the individual object velocities. The variable number of pixels that will need to be processed creates an uncertainty in the total processing time per frame. This variability will have detrimental effect on the control of the robot motion as the sample time for error correction will no more be fixed.

7 A Fast Access Color Look-Up-Table (LUT)

7.1 Limitations of using RGB color space

The blob detection algorithm can be implemented on any commodity vision system. The image digitization can be done using the FlashBus MV Pro frame grabber card which provides pixel color information in RGB (Red, Green and Blue). A convex partition of the RGB color space can be created for each color identifier, as shown in Figure 13. This convex partition (color subspace cube) is specified by a range of RGB values namely, MinR-MaxR, MinG-MaxG, and MinB-MaxB.

Blob identification based on RGB color space is not reliable in the face of varying light intensities as the luminance cannot be separated from chrominance [4]. In order to cater to a wide variation of light intensity, the volume of the color cube has to be extended. The drawback of doing this is that the color cubes of different colors will overlap and encroach into each other's boundaries making it very difficult to segregate colors and at the same time detect them reliably. Instead a convex color subspace, defined in the YUV color space was implemented with greatly enhanced robustness (in respect of reliability of detection). The Y component independently corresponds to light intensity of the color and a wider threshold span can be set for it to cater to varying light intensities.

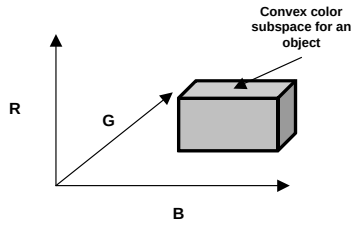

Fig. 13. Convex RGB color subspace

7.2 Defining YUV thresholds

To define the YUV color subspace, a sample of the image is captured and the color of interest is zoomed in. In the zoomed image a rectangular region is defined, within which, each pixel is processed to calculate its YUV

value using the color space transformation matrix which is shown below (Equation 2).

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.5 \\ 0.5 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

or alternatively as

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ U &= 0.565(B - Y) \\ V &= 0.713(R - Y) \end{aligned} \tag{2}$$

The relative position and orientation of the RGB color cube in the YUV color space is shown in Figure 14. From the computed YUV values, the MinY, MaxY, MinU, MaxU, MinV and MaxV are set. A user may manually fine-tune these thresholds using the application's GUI. Usually a wider range of Y values are desirable giving it more bandwidth to account for varying light intensity.

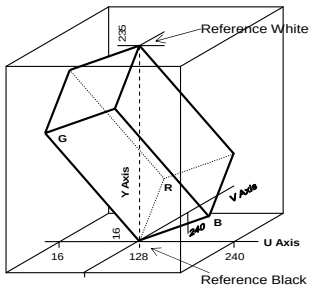

Fig. 14. Relative positioning of YUV and RGB color space

7.3 Membership Testing

For the two-pass sequential algorithm for blob-detection, each RGB color pixel of an image needs to be tested to determine its color sub-space membership. Since the color boundaries are defined in YUV color space, each pixel value would have to be converted from RGB to YUV, using equation

(2), before testing membership using equation (3). This is a computationally intensive process and the overall performance could be quite low. Hence the mechanism used for thresholding warrants close scrutiny and requires careful efficiency consideration.

```
IF ( (Y >= MinY) AND (Y <= MaxY) AND
     (U >= MinU) AND (U <= MaxU) AND
    (V >= MinV) AND (V <= MaxV) )
THEN pixel_of_interest = TRUE                    (3)
```

Equation (2) requires to perform 5 multiplications and the whole implementation of equation (3) may require up to 6 logical ANDing operations to determine whether a pixel belongs to a color subspace and is thus of interest. To improve the computational efficiency, implementations using Boolean valued decomposition of the multidimensional threshold have been tested [4] on static images resulting in substantial reduction in processing time. This, however, still requires the color space to be transformed from RGB to YUV. This method had not been tested previously by any researcher on live images in real time.

7.4 A One-dimensional Color Look-Up-Table

An initial implementation in this research work used a large one-dimensional color look-up-table (LUT) and an indexing technique based on the RGB value of the pixel. The index is created, using the equation (4) below, which is used to access the LUT.

$$index = R*65536 + G*256 + B \quad (4)$$

For a 24-bit RGB color output from the frame grabber card, the maximum value of R, G or B is 255. Thus the size of the LUT is 256x256x256 bytes (16MB). For each RGB value, a unique index is created by the equation (4).

7.5 Posting the Look-Up-Table

Once the YUV thresholds have been defined for each color, the LUT is posted with color identities (IDs) for the entire RGB color space as shown in the code segment in Figure 15.

The time it takes to update the LUT is not of any consequence as the update is done during the color tuning phase. It is important that during the

inspection time, the processing should not take unduly long and hence repeated multiplications and logical ANDing must be avoided.

```
for (r=0; r<256; r++)
 for (g=0; g<256; g++)
  for (b=0; b<256; b++)
   {
    y=(299*r+587*g+114*b+500)/1000;
    u=(565*(b-y) + 128000)/1000;
    v=(713*(r-y) + 128000)/1000;
    index = r*65536 + g*256 + b;

    //-- initialise on update --
    LUT[index] = NoCOL;

    //-- Reference Colour range --
    if ( (MinY<=y && y<=MaxY) &&
         (MinU<=u && u<=MaxU) &&
         (MinV<=v && v<=MaxV))
    {
          LUT[index] = RefCOL;
    }
   }
```

Fig. 15. Posting the LUT with color ID

7.6 Inspecting the Look-Up-Table

To test whether a pixel is in the YUV sub-space, given its RGB value, the index is calculated using equation (4) and the LUT content at that indexed location is tested as shown in the code segment in Figure 16.

To further improve the processing speed, the multiplications in equation (4) were replaced by shift-left operations as in Equation (5) below.

$$\text{index} = R<<16 + G<<8 + B \tag{5}$$

```
index = r<<16 + g<<8 + b;
if ( LUT[index] == RefCol )
//-- it is a desired pixel
{
    //-- process the pixel
}
```

Fig. 16. Inspecting the LUT

To classify each pixel in an image into one of a discrete number of color classes, the index is created from the pixel's RGB values and the LUT is queried. The returned value indicates color class membership.

The advantage of this method is that it does not require the RGB value of each pixel to be converted to YUV, which otherwise would take considerable processing time. However, this method has the following drawbacks-

- It takes very long to update the LUT. Using the experimental hardware setup detailed in section 3, it took 909 ms to update the LUT. This eliminates the possibility of updating the LUT in a real-time processing environment and hence the thresholds defined for each discrete color class cannot be adapted to variations in light intensity.
- Because of the huge size of the LUT (16 Mbytes), the algorithm runs slower on a computer with a small cache memory, as there is frequent memory swapping. Nonetheless, in an image where the majority of the pixels belong to the background color class, this is not a significant problem as the same part of the LUT will be accessed most of the time.

8 Discrete YUV Look-Up-Table

Recent work has focused on efficiency issues so that effective classification can be provided in real-time. For simplicity of tuning, each color is classified with a pair of thresholds on each of the Y, U, and V axes as illustrated in Figure 17.

Since each axis is independent, the large LUT may be decomposed into separate Y, U, and V LUTs of 256 elements each. The total size of all the discrete LUTs put together is only 768 bytes, greatly reducing the memory requirements compared to the LUT described in section 7.4. For each array element, one bit is used to represent each color class as shown in Figure 18.

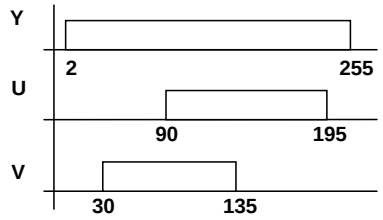

Fig. 17. YUV Thresholds for Color 3

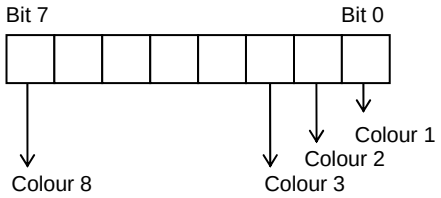

Fig. 18. Color representations in the LUT element

In programming terms, the color IDs are defined as-

```
#define colour1_ID 0x01
#define colour2_ID 0x02
#define colour3_ID 0x04
#define colour4_ID 0x08
#define colour5_ID 0x10
```

Using arrays of bytes, 9 different color classes can be represented (including 8 of interest and the background). To cater to more colors, the system can easily be scaled up to implement arrays of 16-bit or 32-bit integers.

Once the color thresholds have been defined, the YUV LUTs are then posted with the color IDs as shown in Figure 20. The shaded cells store a value of 1.

8.1 Populating the discrete YUV Look-Up-Table

After defining the YUV thresholds for every color class, each LUT is posted individually with color IDs. The code segment shown in Figure 19 updates the Y-LUT. The U- and V-LUTs are similarly posted.

```
for (y=0; y<=255; y++)
{
   if ((y >= Col1_MinY) &&
       (y <= Col1_MaxY))
     Y_LUT[y] |= Colour1_ID;
 //-- repeat for other colours
}
```

Fig. 19. Posting the LUT with color ID

To update the three LUTs it takes only 8.2 μs using the same experimental hardware setup. This enables the LUTs to be updated in real-time and hence may be used in implementing adaptive color thresholding.

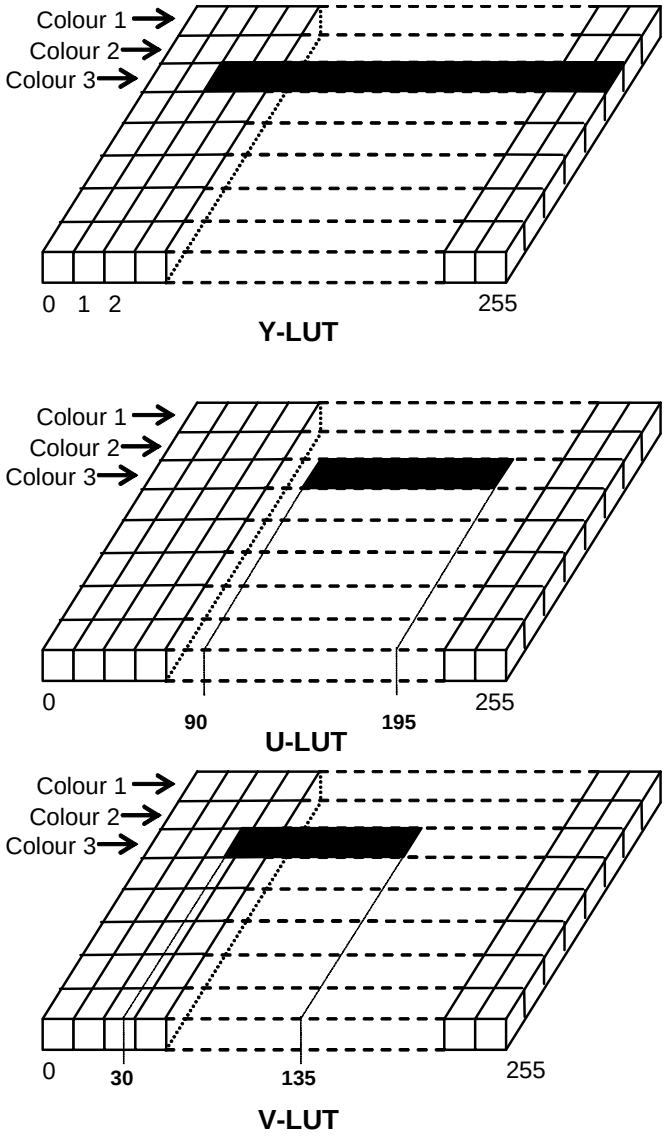

Fig. 20. YUV LUT populated for color 3 (not to exact scale)

8.2 Testing Color Class Membership

A pixel belongs to a color class only if it is within all three Y, U, and V ranges. The color class membership can therefore be computed as a bitwise AND of the elements of each component array. This is shown in the code segment in Figure 21.

```
if (Y_LUT[Y] & U_LUT[U] & V_LUT[V] &
   Colour1_ID)
{
  //the pixel belongs to Color1 class
}
```

Fig. 21. Testing color class membership

The membership testing is very fast as the bitwise AND operation is computationally inexpensive. This method, however, requires the YUV values of each pixel to be available, which is often not the case for commodity hardware. Thus the advantage gained by using a smaller LUT is partly offset by the additional computation time required to map a pixel from RGB color space to YUV color space. In Section 8.3 two methods of speeding up this mapping are presented and the performance evaluation is detailed in Section 8.5.

8.3 Color Space Transformation

The standard transformation matrix of equation (2) for mapping RGB to YUV involves several floating point multiplications, which are potentially very time-consuming. The floating point arithmetic may however be replaced by integer operations as in equations (6) below.

$$\begin{aligned} Y &= (299R + 587G + 114B)/1000 \\ U &= 565(B - Y)/1000 + 128 \\ V &= 713(R - Y)/1000 + 128 \end{aligned} \tag{6}$$

The U and V components are offset by 128 to bring them into positive range to facilitate array indexing. Equations (6) still use division operations. The division operations may be eliminated by scaling the coefficients by powers of 2 rather than powers of 10, allowing the use of a computationally less expensive shift-right operation as shown in Equations (7) below.

$$
\begin{aligned}
Y &= (9798R + 19235G + 3736B) >> 15 \\
U &= 18514(B - Y) >> 15 + 128 \\
V &= 23364(R - Y) >> 15 + 128
\end{aligned}
\tag{7}
$$

8.4 A New Color Space

This research makes a significant contribution by proposing a novel Y'U'V' color space. This new color space not only retains all the colors of the RGB color cube, it actually increases the volume of the YUV color cube, thereby enhancing the resolution of spatial color separation. The proposed transformation is governed by equation (8):

$$
\begin{bmatrix} Y' \\ U' \\ V' \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}
\tag{8}
$$

The new Y'U'V' color space has several advantages over the standard YUV space represented by equation (2).

- Computationally it is very inexpensive. Only integer additions and subtractions are involved; all floating point operations and logical shift operations have been completely eliminated.
- The white point corresponds to equal quantities of R, G, and B.
- The new Y'U'V' color space provides better resolution. Without scaling, the Y' range is 0 to 765, U' range is from -510 to 510 and V' range is from -255 to 255. This enables colors that are closer together to be detected reliably.
- The Y', U' and V' axes are orthogonal, making the transformation back to RGB similarly simple. It also gives better decorrelation of the color space for many images.

This larger LUTs (2298 elements as compared to 768) increase the LUT update time to 19.2 μs. This is still extremely fast and causes no concern for real-time update of the LUT.

8.5 Experimental Results and discussion

The proposed improvements to the methods of transforming from RGB to YUV color space and the new Y'U'V' color space were evaluated using the robot soccer system, which offered the possibility of testing the algorithms for real-time processing with differing number of objects. A very high precision counter was implemented in the software which enabled measurement of time with an accuracy of a hundredth of a milli-second. Tests were done with 4 objects (3 home robots and ball), 7 objects (3 home robots, 3 opponent robots and ball) and 11 objects (5 home robots, 5 opponent robots and ball) for incremental and full tracking. To evaluate the robustness, the tests were done for incremental as well as for full tracking modes. The test results are summarized in Table 1, and compared in Figures 22 and 23. The tracking times were measured for 1000 frames and averaged.

Table 1: Summary of test results

System	#Objects	Average Tracking Time (ms)		LUT update time
		Incremental	Full	
#1	4	5.27	15.54	909 ms
	7	5.62	20.97	
	11	5.95	28.34	
#2	4	5.38	21.89	8.2 µs
	7	5.64	30.37	
	11	5.99	41.72	
#3	4	5.34	17.22	8.2 µs
	7	5.56	23.41	
	11	5.85	31.68	
#4	4	5.15	14.34	19.2 µs
	7	5.38	19.22	
	11	5.68	25.76	

With respect to the data presented in Table 1, the various systems are as follows:

System #1: Large composite LUT indexed using RGB - equation (5)
System #2: Separate YUV LUTs, with integer division used to map RBG to YUV – equation (6)

System #3: Separate YUV LUTs using the >> operation to map RBG to YUV – equation (7)

System #4: New color space (Y'U'V') and separate Y'U'V' LUTs – equation (8)

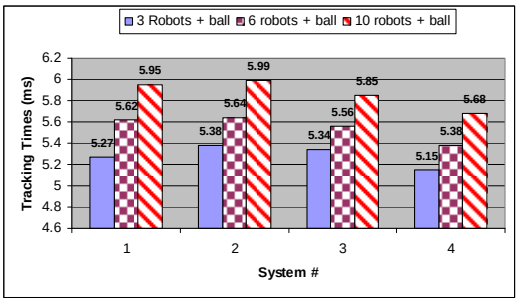

Fig. 22. Comparison of Incremental Tracking Time

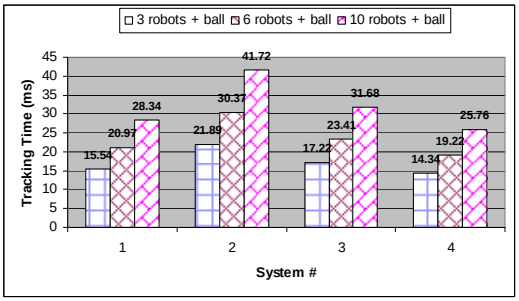

Fig. 23. Comparison of Full Tracking Time

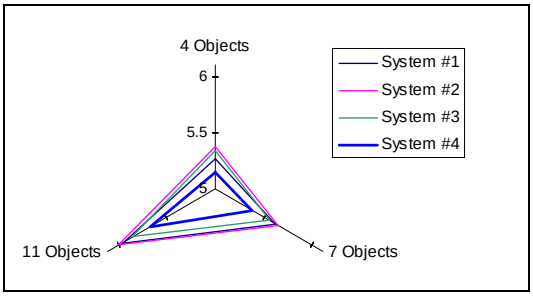

Fig. 24. Chart showing the performance of the four systems for Incremental Tracking

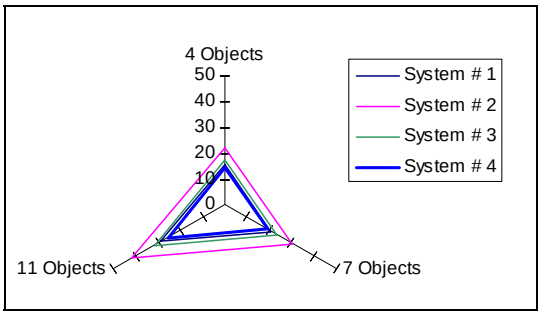

Fig. 25. Chart showing the performance of the four systems for Full Tracking

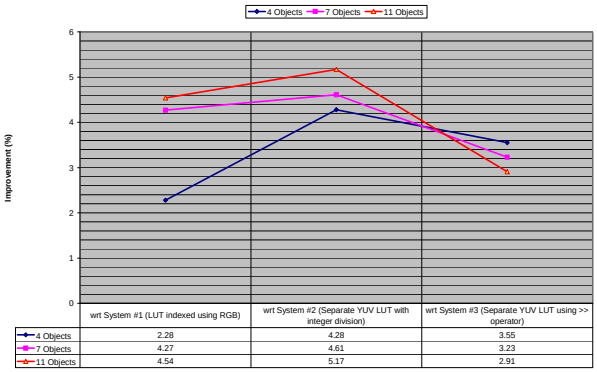

Fig. 26. Summary of improvements in processing speed (in %) achieved by Y'U'V' for incremental tracking

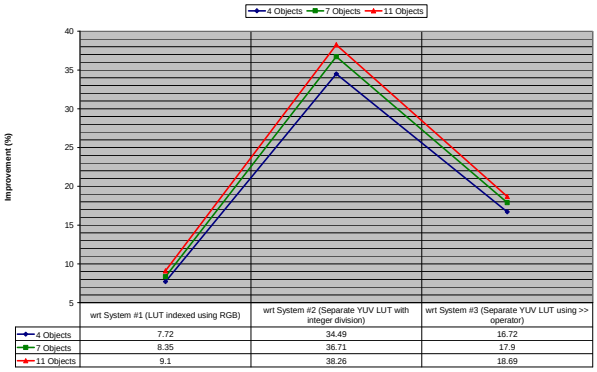

Fig. 27. Summary of improvements in processing speed (in %) achieved by Y'U'V' for full tracking

The radar charts in Figures 24 and 25 show the vision processing time taken by the different systems for incremental and full tracking respectively for 4, 7 and 11 objects. The system using Y'U'V' color space and discrete look-up-tables takes the least amount of time.

Figures 26 and 27 highlight the relative improvements achieved in the vision processing time by using the new Y'U'V' color space for different number of objects. The improvements reported are with respect to the other three systems. As can be seen, the improvements are substantial.

To summarize the experimental results, it can be said that an efficient arrangement of discrete Y, U and V LUTs for fast color segmentation has been proposed and tested for real-time vision processing applications. A significant reduction in LUT size (and hence the memory requirement) has been achieved. This translates into large increase in program execution speed for incremental and full tracking of multiple objects, especially on processors with small cache. The time to update the discrete LUT is negligible (8.2 μs for YUV and 19.2 μs for Y'U'V') and hence is very suitable for real-time update. This is a vast improvement over the method that employs a large LUT with indexing using RGB and takes 909 ms to update the LUT. This has laid the foundation which will enable further work to be done to make the LUT 'adaptive' in order to cater to variation of light intensities and color distortions, possibly due to reflections from nearby objects.

To enhance the gains derived from the discrete YUV LUTs, the proposed new color space, Y'U'V', further simplifies the transformation from RGB to a YUV-like color space. The time to fully track 7 objects reduces from 30.37 ms (using YUV LUT) to 19.22 ms (using Y'U'V' LUT), which is an improvement of 36.71%.

Furthermore, the Y'U'V' color space allows better color resolution, thereby increasing the robustness of color classification. The results compare favorably to the color threshold based approaches discussed in [7].

8 References

1. Berthold K. P. Horn, Robot Vision (MIT Electrical Engineering and Computer Science), MIT Press, 1986.
2. M. Jamzad, B.S. Sadjad, V.S. Mirrokni, M. Kazemi, H. Chitsaz, A. Heydarnoori, M.T. Hajiaghai, and E. Chiniforoosh, "A Fast Vision System for Middle Size Robots in RoboCup", A. Birk, S. Coradeschi, S. Tadokoro (Eds.): RoboCup 2001: Robot Soccer World Cup V, Spinger Verlag 2002, pp. 71 – 80.

3. C.H.Messom, S. Demidenko, K. Subramaniam and G. Sen Gupta, “Size/Position Identification in Real-Time Image processing using Run Length Encoding”, IMTC, Alaska, USA, 2002, pp 1055-1059
4. J. Bruce, T. Balch and M. Veloso, “Fast and Inexpensive Color Image Segmentation for Interactive Robots”, IROS 2000, San Francisco, 2000, pp. 2061 – 2066.
5. F. Ercal, M. Allen, and F. Hao, "A Systolic Image Difference Algorithm for RLE-Compressed Images”, IEEE Transactions on Parallel and Distributed Systems, Vol. 11, No. 5, May 2000, pp. 433 – 443.
6. 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, pp. 1 – 19.
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, New York, 2000. Springer, pp. 148 – 161.
8. M. Ramesh Jain, R. Kasturi, and B.G. Schunck, Machine Vision, McGraw-Hill International Editions, Computer Science Series, International Edition, 1995.
9. Sangho Park, Aggarwal, J.K., “Segmentation and tracking of interacting human body parts under occlusion and shadowing”, Proceedings of the workshop on Motion and Video Computing, 5-6 Dec. 2002, pp. 105-111, ISBN: 0-7695-1860-5
10. Garcia, C. Apostolidis, X., “Text detection and segmentation in complex color images”, Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '00, 06/05/2000 - 06/09/2000, Istanbul, Turkey, pp. 2326-2329, ISBN: 0-7803-6293-4
11. M. Shand, "Flexible image acquisition using reconfigurable hardware", IEEE Symposium on FPGA's for Custom Computing Machines (FCCM '95), 1995, pp. 0125
12. C.H .Messom, G. Sen Gupta and H.L. Sng, “Distributed Real-time Image Processing for a Dual Camera System”, CIRAS 2001, Singapore, 2001, pp. 53-59.
13. Chakravarthy, A. Ghose, D., “Obstacle avoidance in a dynamic environment: a collision cone approach”, IEEE Transactions on Systems, Man and Cybernetics, Part A, Sep 1998, Vol. 28, Issue: 5, pp. 562-574, ISSN: 1083-4427