A New Colour Space for Efficient and Robust Segmentation

Gourab Sen Gupta[*,+], Donald Bailey[*], Chris Messom[#]
[*]IIS&T, Massey University, Palmerston North, New Zealand
[#]IIMS, Massey University, Albany, New Zealand
Email: {g.sengupta, d.g.bailey, c.h.messom@massey.ac.nz}
[+]School of Electrical and Electronic Engineering, Singapore Polytechnic, Singapore

Abstract

In many applications, real-time object identification and tracking is of paramount importance. Traditionally, such systems, employing real-time colour-based segmentation, are implemented in hardware to realise fast processing speed. Most of the inexpensive commodity frame grabber cards provide pixel colour information in RGB. However, using RGB colour space, it is difficult to separate the chrominance from luminance. For the classification algorithm to be robust in the face of variation of brightness of illumination, the HSI or YUV colour model is usually used. In this paper we evaluate the efficiency and reliability of using the classical methods to transform the RGB colour space to YUV colour space coupled with techniques to create and use efficient look-up-tables (LUTs) for fast colour classification of pixels. A new YUV colour space is proposed and its usefulness depicted by exhaustive data analysis using robot soccer system as the test platform.

Keywords: object identification, colour threshold, LUT, YUV colour space

1 Introduction

In real-time interactive mobile robot applications, such as multi-agent collaborative systems, vision plays a very important role [1,2]. Local vision systems of mobile robots have significant limitations. Due to space constraints, dedicated hardware has to be used, which may be fast and reliable, but expensive and inflexible. In contrast, global vision systems, based on mass-produced off-the-shelf tools, are inexpensive and flexible. With modern, fast computing machines, processing speed is only a limited constraint. Improvement in the image processing algorithms has brought real-time vision processing into the realms of commodity hardware. However, when the number of objects that need to be detected and reliably tracked increases, the real-time processing capabilities of the vision system are seriously challenged.

2 Vision System Description

2.1 The Need for Fast Vision Processing

In the context of robot soccer, the robots easily attain speeds of 1.5 m/s. At times the ball travels at a speed in excess of 2 m/s. In such a highly dynamic environment, the robots have to exhibit basic actions like positioning at a designated location, moving to the ball or blocking an opponent, turning to a desired angle, circling around the ball and shooting the ball into opponent's goal. These basic functions rely on being fed and updated with current position and orientation of the targets (ball and robots) at a very fast rate. Among other factors, the strategy and path planning in a robot soccer game are dependent on the speed of image processing.

2.2 Vision Capture Hardware Configuration

The vision system consists of a Pulnix 7EX NTSC colour CCD camera and a FlashBus MV Pro frame grabber card which is plugged into a PC. The image is captured at a resolution of 640x480 pixels at a sampling rate of 30 Hz. The odd and even fields of the interlaced image are processed separately. Hence the effective image resolution is 320 x 240 delivered at a sampling rate of 60 Hz. The captured image is processed using a Pentium 4 PC (1.8 GHz, 512 MB RAM). To derive maximum speed of video feed, the frame grabber card was configured for off-screen image capture, as shown in Figure 1. The image is sent to the system RAM and processed from there. Only when the live image is required to be seen on the computer monitor, such as while tuning the colours, the image is loaded in the VGA memory.

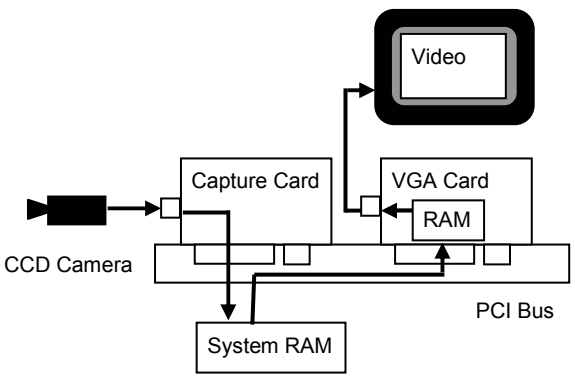

Figure 1: Image capture in off-screen capture mode.

2.3 Interrupt Driven Multi-buffered Video 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. The FlashBus MV Pro vision card 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 vision processing and strategy calculations 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 then process the image. If the image processing is guaranteed to complete in the specified sample time only two buffers are required. Since this may not always be the case, a four buffer system, as shown in figure 2, is implemented which offers some fault-tolerance.

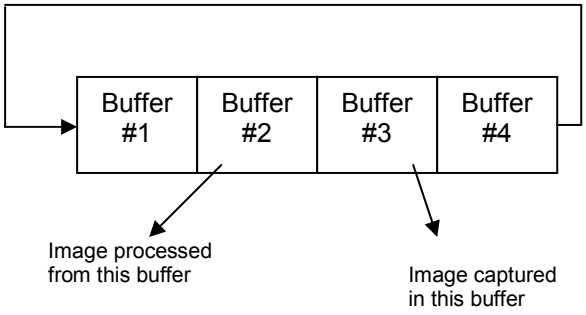

Figure 2: Multi-buffered image capture.

2.4 Full Tracking v/s Incremental Tracking

The full tracking algorithm searches through the whole image, testing if each pixel is a member of one of the calibrated object colours [3]. The full tracking process is very inefficient when the objects are small and only represent a small percentage of the whole image. To overcome this problem an incremental tracking approach is adopted. In this approach a small part of the image around the last known position of the object is analysed as shown in figure 3. To initialize the object positions the image must initially be scanned fully for 1 frame. This will identify the starting position of all the objects within the field of view. The incremental tracking algorithm can then be used for subsequent images.

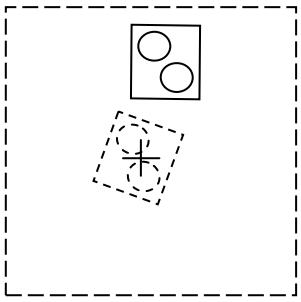

Figure 3: Tracking window centred on last known position of object.

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.

It can be seen that reducing the sample time 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. Halving the sample time would have the objects move half the distance, so the sides of the tracking window can be halved, but this gives an area reduction of a factor of four. Therefore doubling the frame rate reduces the time of this algorithm by four, so paradoxically an incremental 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 can be used to centre the tracking window on the expected position of the object, rather than the last known position. This is illustrated in figure 4. In this manner the size of the tracking window can be reduced further as the error in predicted position will depend only on the uncertainty in the measured velocity. Ball speeds of 2.5 ms^{-1} can be reliably tracked with a tracking window of 40x40 pixels.

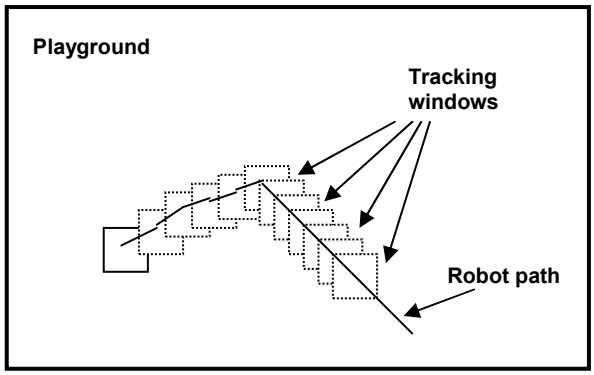

Figure 4: Robot path and the incremental tracking window.

3 Colour Segmentation

When independent thresholds are used for each colour component, the RGB colour space gives poor segmentation results. Any change in the intensity of a colour results in a diagonal movement within the RGB space, with a significant effect on all of the components. Several standard colour space transformations overcome this problem, enabling efficient and robust colour segmentation.

The YUV colour space is widely used in video and broadcasting today. It is very different from RGB colour space; instead of three large colour channels, it deals with one brightness or luminance channel (Y) and two colour or chrominance channels (U-blue and V-red). The transformation from RGB to YUV that retains the same number of colours in both spaces is

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (1)$$

or alternatively as

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

The relative position and orientation of the RGB colour cube in the YUV colour space is shown in figure 5.

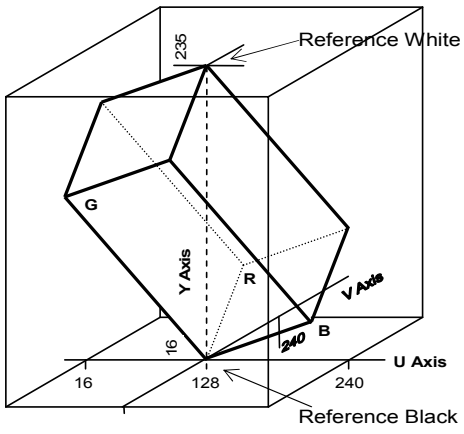

Figure 5: Relative positioning of YUV and RGB colour space.

A blob detection technique using colour thresholding, nearest neighbour classification and an efficient method to form and access a colour look-up-table (LUT) has been detailed in [4, 5].

A set of constant thresholds are used to define a colour class in the YUV colour space. This method essentially creates a huge composite LUT with 16777216 (256x256x256) entries to store colour IDs. For each pixel in the RGB colour space, a unique index is created using equation (3) to access and update the table.

$$\text{index} = R*65536 + G*256 + B \quad (3)$$

To classify each pixel in an image into one of a discrete number of colour classes, the index is created from the pixel's RGB values and the LUT is queried. The returned value indicates colour 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 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 colour class cannot be adapted to variations in light intensity.
- Because of the huge LUT size (16 Mbytes), the algorithm runs slowly on a computer with small cache, as there is frequent memory swapping. Nonetheless, in an image where the majority of the pixels belong to the background colour class, this is not a significant problem.

3.1 Structure of the YUV LUT

Recent work has focused on efficiency issues so that effective classification can be provided in real-time. For simplicity of tuning, each colour is classified with a pair of thresholds on each of the Y, U, and V axes as illustrated in figure 6. Since each axis is independent, the large LUT may be decomposed into separate Y, U, and V LUTs of 256 elements each, greatly reducing the memory requirements from that in [4]. For each array element, one bit is used to represent each colour class as shown in figure 7.

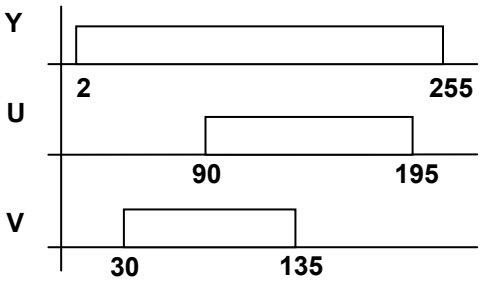

Figure 6: YUV Thresholds for Colour 3.

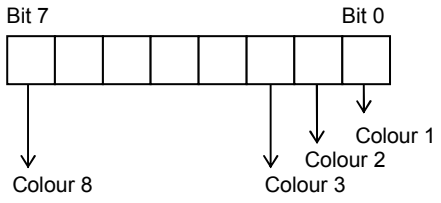

Figure 7: Colour representation in the LUT element.

In programming terms, the colour IDs are defined as-

```
#define colour1_ID 0x01
#define colour2_ID 0x02
#define colour5_ID 0x10
```

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

Once the colour thresholds have been defined, the YUV LUTs are then posted with the colour IDs as shown in figure 8. The shaded cells store a value of 1.

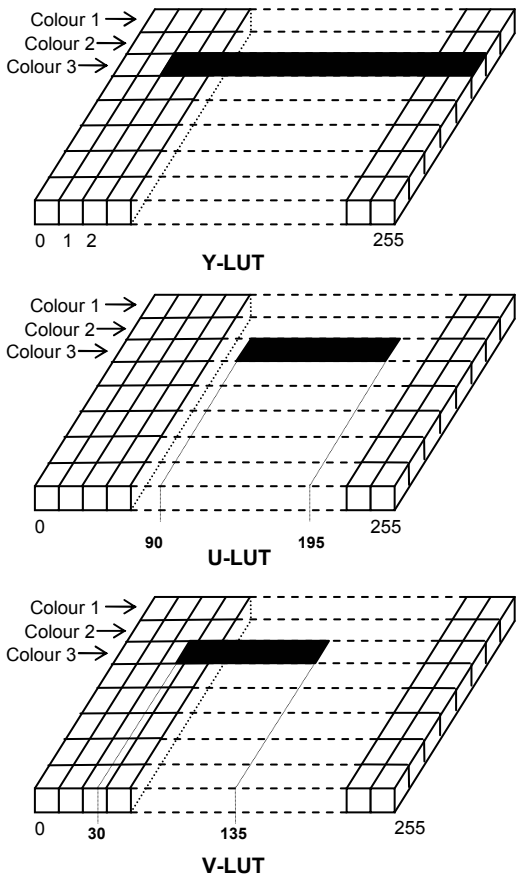

Figure 8: YUV LUT for colour 3 (not to exact scale).

3.2 Populating the YUV LUT

After defining the YUV thresholds for every colour class, each LUT is posted individually with colour IDs. The code segment in figure 9 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
}
```

Figure 9: Posting the LUT with colour ID.

To update the 3 LUTs it takes only 8.2 μS. This enables the LUTs to be updated in real-time and hence may be used in implementing adaptive colour thresholding.

3.3 Testing Colour Class Membership

A pixel belongs to a colour class only if it is within all three Y, U, and V ranges. The colour 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 10.

```
if (Y_LUT[Y] & U_LUT[U] & V_LUT[V] &
     Colour1_ID)
{
  //the pixel belongs to Colour1 class
}
```

Figure 10: Testing colour 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 colour space to YUV colour space. In Section 4 we present two methods of speeding up this mapping and evaluate the performance in Section 5.

4 Colour Space Transformation

The standard transformation matrix of equation (1) 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 equation (4).

$$\begin{aligned} Y &= (299R + 587G + 114B)/1000 \\ U &= 565(B - Y)/1000 + 128 \\ V &= 713(R - Y)/1000 + 128 \end{aligned} \quad (4)$$

The U and V components are offset by 128 to bring them into positive range to facilitate array indexing. Equation (4) still uses 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.

$$\begin{aligned} Y &= (9798R + 19235G + 3736B) >> 15 \\ U &= 18514(B - Y) >> 15 + 128 \\ V &= 23364(R - Y) >> 15 + 128 \end{aligned} \quad (5)$$

4.1 A New Colour Space

We propose a new ***Y'U'V'*** colour space, which not only retains all the colours of the RGB colour cube; it actually increases the volume of the YUV colour cube, thereby enhancing the resolution of spatial

colour separation. The proposed transformation is governed by equation (6):

$$\begin{bmatrix} Y' \\ U' \\ V' \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (6)$$

The ***Y'U'V'*** colour space has several advantages over the standard YUV space represented by equation (1).

- 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.
- There is better resolution in the ***Y'U'V'*** colour space. 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 colours 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 colour 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.

5 Experimental Results

The robot soccer system was used as the test bed for evaluating the system. This is a good example of a system where real-time image processing is absolutely essential for good performance. Moreover, there are several objects on the field – home and opponent robots, ball etc, and thus the system could be evaluated for different number of objects. Tests were done with 4 objects (3 home robots and ball) and 7 objects (3 home robots, 3 opponent robots and ball) for incremental and full tracking. The test results are summarized in Table 1, and compared in figures 11 and 12.

6 Conclusions

The paper presented an efficient arrangement of separate Y, U and V LUTs for fast colour segmentation. A significant reduction in LUT size (and hence the memory requirement) is achieved, which translates into large increase in program execution speed for full image tracking, especially on processors with small cache. The time to update the LUT is negligible (8.2 µs for YUV and 19.2 µs for ***Y'U'V'***) and hence is suitable for real-time update. This is a vast improvement over the earlier method, which took 909 ms to update the LUT. This allows the LUT to be made 'adaptive' to cater to variation of light intensities and colour distortions, possibly due to reflections from nearby objects.

Table 1: Summary of test results.

System	#Objects	Tracking Time (in ms)	
		Incremental	Full
#1	4	5.27	15.54
	7	5.62	20.97
#2	4	5.38	21.89
	7	5.64	30.37
#3	4	5.34	17.22
	7	5.56	23.41
#4	4	5.15	14.34
	7	5.38	19.22

System #1: Large LUT indexed using RGB - equation (3)

System #2: Separate YUV, with integer division used to map RBG to YUV – equation (4)

System #3: Separate YUV using the >> operation – equation (5)

System #4: New colour space (***Y'U'V'***) – equation (6)

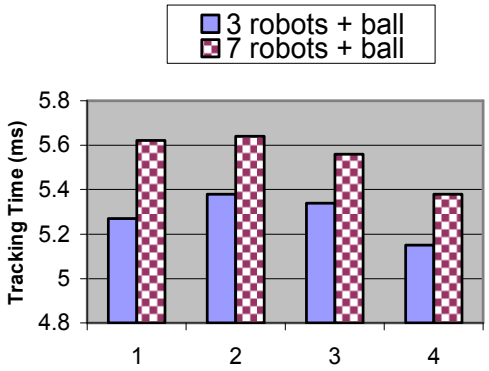

Figure 11: Comparison of Incremental Tracking Time.

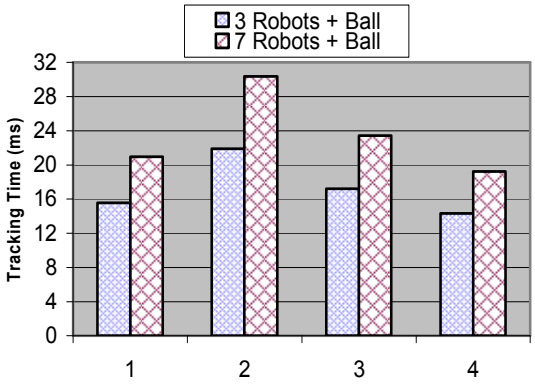

Figure 12: Comparison of Full Tracking Time.

To enhance the gains derived from the YUV LUT, we have proposed a new colour space, ***Y'U'V',*** which further simplifies the transformation from RGB to a YUV-like colour 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 36.7%.

Furthermore, the ***Y'U'V'*** colour space allows better colour resolution, thereby increasing the robustness of colour classification. The results compare favourably to the colour threshold based approaches discussed in [6].

7 References

[1] 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, pp. 71 – 80. (2002)

[2] Pieter Jonker, Jurjen Caarls, and Wouter Bokhove, "Fast and Accurate Robot Vision for Vision Based Motion", P. Stone, T. Balch, G. Kraetzschmar (Eds.): *RoboCup 2000: Robot Soccer. World Cup IV* Spinger Verlag 2001, pp. 149 -158 (2001)

[3] C.H.Messom, G. Sen Gupta and Sng H.L, "Distributed Real-Time Image Processing for a Dual Camera System", *International conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS 2001), Singapore*, pp 53-59 (2001)

[4] G. Sen Gupta, Tin Aung Win, Chris Messom, Serge Demidenko and Subhas Mukhopadhyay, "Defect analysis of grit-blasted or spray painted surface using Vision Sensing Techniques", *Image and Vision Computing New Zealand (IVCNZ 2003)*, *New Zealand*, pp 18-23 (2003).

[5] M. Ramesh Jain, R. Kasturi, and B.G. Schunck, *Machine Vision*, McGraw-Hill International Editions, Computer Science Series, International Edition (1995).

[6] J.Baltes. "Practical camera and colour calibration for large rooms", *RoboCup-99: Robot Soccer World Cup III*, New York, pp 148-161 (2000)