Finding Wally

MSc. Dissertation Report

Cian Booth

Contents

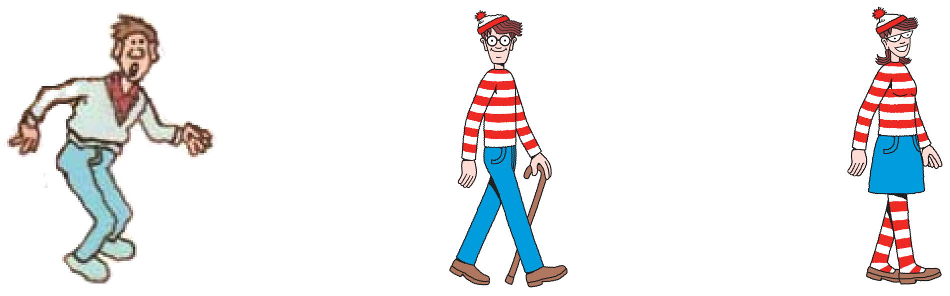
1 Introduction

Computer vision is a field concerned with using computers to process and analyze images. It is an attempt to replace the need for humans to perceive visual information, and perhaps to improve upon innate human ability in some areas.

One of the most important things that human vision is able to do, is to recognise an object. This means that the brain is able to decipher the basic information received from the eyes, and deduce any objects described within it. This is a complicated task, and not easily replicated programmatically.

Just as there are several ways to describe an object, there are also several different techniques that can be used to locate an object. Each technique has varying requirements and reliability. Some use specific descriptions (e.g. a sample image) and produce very reliable results. Others do not require such precise descriptions, but do not create as dependable results. By combining these techniques together, the reliability and robustness of object recognition can be increased. Unfortunately, using all available techniques will considerably reduce the speed that objects can be found at. This can be mitigated by computing the techniques in parallel, with a task farm. This means that general tasks (in this case, each method of identifying an object) will be run concurrently. Task farms are especially useful in this case, because it allows the use of libraries that would otherwise be complicated to parallelise. Parallelism allows computer vision programs to produce results both quickly and reliably.

A good way of testing the task farm method is with Where's Wally puzzles. These are simple cartoons, normally a large image filled with various characters, who wear simply coloured clothing, see Figure 1(a). One of these characters is the eponymous Wally, who is dressed distinctly from the others, see Figure 1(b). Similarly dressed characters exist, Figure 1(c), adding complexity to specifically finding Wally. The cartoon nature of the characters means that shapes are boldly coloured, and often bordered by a black line. As Where's Wally is puzzle, sometimes Wally will be hard to find; he is often obscured, camoflauged or simply small. This provides a non-trivial problem, that is still simple enough to provide solutions for.



(a) A normal person **(b)** Wally **(c)** Wenda

Figure 1: Characters from Where's Wally

1.1 Shape and Colour Analysis

One obvious way of analysing an image with a computer, is to break the image down into shapes and colours. In terms of language, it is easiest to depict an object by describing it's shape and colour, i.e. "the red box" or "the green hand". This is conceptually simple to explain and understand, and thus is more intuitive to program. Most images are saved as raster images (e.g. PNG, BMP, GIF), which is a 2D array of colours that directly map to pixels on a screen. This is opposed to vector images that store the location and colour of geometric primitives (squares, circles, triangles, etc.). Vector images, rather than raster images, are directly easier to perform shape and colour analysis on. However, input devices such as webcams and scanners almost exclusively produce images in raster formats.

Regardless of format, methods of colour analysis are simple to implement programatically. This is because colour is instrinsic to all methods of storing the data of an image. Shape analysis is less simple for raster images. Shape boundaries must be found, which is normally done through edge detection. These boundaries are then analysed to find points of intersection, and the curvature between them. The curves, in turn, must then be examined to find what shapes exist.

These methods allow for a descriptional, heuristic method for locating characters. No previous image of Wally is required, only a description, such as "red and white stripes" or "black glasses". This allows users to extend the solution past Wally, and towards other characters, who do not have to be seen prior. This form of analysis can lead to false positivies, and should be combined with other results.

An example of the usefulness of this, beyond Where's Wally, could be found in augmented reality technology, such as Google Glass. A common problem people experience is losing their keys. Users with access to AR devices could use colour and shape analysis to enhance their searching (i.e. if they are visible, but in a cluttered area). As keys generally have a few well defined shapes and colour schemes, the device would not have to store what the keys look like in advance. Assuming that parallelism is available, this could potentially be done faster than the human eye can search, helping the user significantly.

1.2 Feature Analysis

Some of the most reliable computer vision libraries (such as SIFT [1]), were developed while considering the neuroscience of human vision. Tanaka[2] and Perrett and Oram[3] found that human object recognition identifies objects with features that are invariant to brightness, scale and position. These results have been used as inspiration for feature analysis. This finds 'feature's, regions of an image which are scale, rotation and illumination invariant. These features are most immediately useful when compared with the features of another image. For example, the features from an image of just Wally can be used to locate Wally in a normal puzzle image.

This method generally requires an existing image to find Wally, which leaves However, this method is very reliable, and, as long as Wally is not obscured, will likely locate him. Normally, Wally is obscured, so this method should be combined with other techniques.

Feature analysis benefits from task farming when there is an image needs to be searched for a large number of sub-images. A beyond Wally example would be in detecting employees going into work on a flexitime basis. This would use photos of each employee combined with CCTV to note the time that workers enter and leave their workplace. Users would not need any form of ID other than their own faces, and this can be combined with existing security systems.

2 Literature Review

2.1 Shape and Colour Analysis

2.2 Feature Recognition

Detecting features within an image is an important technique that should be used in object recognition. One of the most robust algorithms available for this is SIFT (Scale-Invariant Feature Transform), developed by David Lowe[1]. This algorithm uses various techniques to find scale, rotationally and translationally invariant keys, which are also partially invariant under affine transforms and changes in illumination. The keys contain feature vectors, which describe the area around them. This algorithm is very reliable at finding a given sub-image within a larger image. However, some of the techniques required to produce reliable results require unexpected amounts of memory. To create a scale-space version of the image, four versions of the image must be produced, one of which is scaled to be twice the size of the original. This is not a problem when analysing a single image, or analysing multiple images. If the algorithm is used for analysing multiple images concurrently, memory constraints would limit it's efficiency. This can be seen with Zhang's parallel implementation of SIFT [4], where the scale-space creation is one of the areas the program spends most time on. This paper shows reasonable parallel efficiency, but only has results up to 32 cores. The paper also only solves 5 images at a time, which gives a 7 times speed-up over an optimised version of SIFT. This number of images is not large enough to show the true limits of shared memory parallelism.

SURF (Speeded-Up Robust Features), developed by Bay et. al.[5], offers a similarly robust solution, but with greater efficiency. This algorithm replaces Laplacian of Gaussian filters used in SIFT with box filters, which calculate in constant time, once an integral image has been produced. It greater potential for parallelism than SIFT; calculating versions of the image for the scale space are independent of the previous level and can be done in parallel.

3 Colour Analysis

Colour analysis is arguably the easiest of the three main techniques to implement. It only requires that an image be accessible in the normal way; an array of pixels. As this problem considers only Where's Wally images, the pixels can be accessed in the RGB format. This means that each pixel contains information on how much red, green and blue to show. This is as opposed to HSV, or hue, saturation and value. Hue represents the pixel's position on a colour wheel, saturation represents it's vibrancy and value is a measure of it's brightness. RGB maps easily to HSV, but in cartoon images, there isn't generally a large change in an objects lightness level. Thus it is easier to work with RGB images, but the methods described below also work for HSV.

3.1 Function: `get_colour_in_image`

usage `get_colour_in_image(image, colourA, colourB, rg,gr,rb,br,bg,gb)`

image The input image, as loaded by OpenCV.

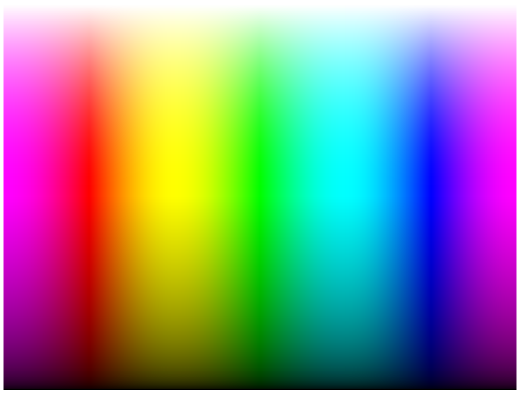
colourA,colourB Strings describing the limits of colours to allow, in the form "#RRGGBB"

rg,gr,... The allowed ratio of each colour. For example, $R >= rg \times G$, and $G >= gr \times R$.

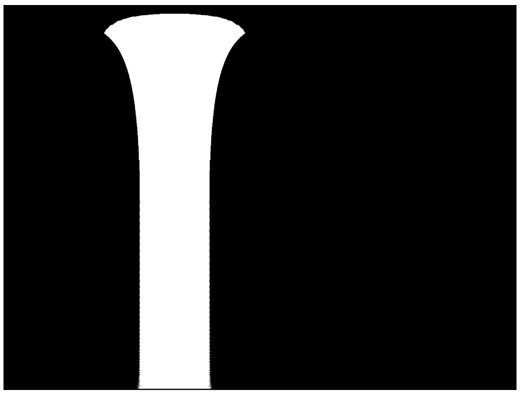
returns Binary mask the same size as the input image, with values of 255 if pixel was within the required values, and 0 otherwise.

This function allows a user to search the image for a specific range of colours. Moreover, the user is able to restrict the results based on the relationships between the pixels colour values. For example, a user can search for colours that are yellow by searching the entire colourspace and ensuring that R and G have similar values, and that they are both greater than B;

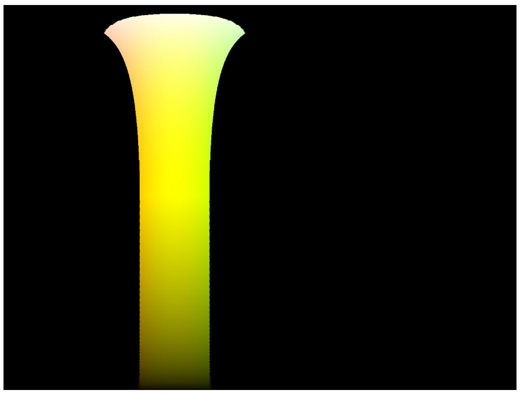
```
yellow = get_colour_in_image(input,"#000000","#FFFFFF",0.8,0.8,1.1,0,1.1,0)
```



(a) A typical colour spectrum



(b) The mask produced by the function. White represents a pixel that was considered to be yellow.



(c) The mask overlayed on the original image, revealing "yellow" colours.

Figure 2: Isolating yellow from an image using `get_colour_in_image`

The output of that function can be seen in Figure 2. The description "yellow" is not clearly defined; there is no single colour code for yellow. However, it is fair to describe yellow as some combination of red and green, with little to no blue. Without describing the specific colour code being searched for, yellow colours can be found, and results can be produced. This function allows the user to search for objects in a more robust fashion, if just for some colour.

3.2 Function: `get_greyscale_in_image`

usage `get_greyscale_in_image(image, low, high, tolerance)`

image The input image, as loaded by OpenCV.

low,high The range of values that should be allowed on the greyscale.

tolerance This is a finite tolerance, with non-zero values allowing nearly greyscale colours to be included.

returns Binary mask the same size as the input image, with values of 255 if pixel was within the required values, and 0 otherwise.

Although greyscale is a special case of colour, slightly different requirements were discerned for greyscale searches. Users are able to defince a tolerance, which allows for colours that are not strictly grey to be included in results. For example;

```
black = get_greyscale_in_image(input, 0, 40, 20)
```

will include blacks and very dark greys, as well as very dark reds, greens and blues.

3.3 Function: `find_regions_from_mask`

usage `find_regions_from_mask(mask)`

image The input mask, a binary single channel matrix.

returns A list of structs, containing information about the size of the region, the average position and bounding box.

It is important to be able to discretise the results of the above functions. Something is needed that will convert the binary matrix into a list of results. This can be done by counting and collecting data about the distinct regions available in the matrix. A region is defined as a group of pixels that are connected through non-zero nearest neighbours.

A naive way to do this computationally is shown in Figure 3. Each non-zero pixel is assigned a unique integer value. Each pixel of the matrix is looped over, and it's value becomes the maximum of itself and it's four nearest neighbours. This is repeated until no pixels change value. This method takes the maximum value of neighbouring pixels, and ignores non-zero pixels, so maximal values can not be spread outside of a region's boundary. Within a region, it is evident that every pixel will have the value of the maximum pixel within that region. As each pixel has a unique value, it follows that each regional maximum will also have a unique value. By counting the unique values in the matrix, the number of regions can be found. Similarly, properties of the region can be calculated by analysing specific unique values. For example, the mean position of the pixels within the region, the size of the region, and the bounding box around the region.

More computationally efficient methods exist. Lifeng [6] describes the method of connected-component labelling. The removes the need for iteration until the matrix stabilises, and also reduces the memory need per pixel. A pixel $p_{i,j}$ that has zero-valued or non-existant neighbours $p_{i-1,j}$ and $p_{i,j-1}$ is assigned a new temporary label. Otherwise, if the upper pixel $p_{i,j-1}$ is non-zero, it has a label (thanks to the ordering of the algorithm). The pixel $p_{i,j}$ is then assigned with the label of $p_{i,j-1}$. If the label of $p_{i,j}$ has not been set, then it gets the label of the non-zero left pixel $p_{i-1,j}$. If $p_{i-1,j} = p_{i,j-1}$ but they do not have the same label, then a label equivalence is noted, for later use. Equivalent labels are then merged, and the regions have been developed. This method, is both simple and memory efficient enough to be suitable for parallelism; temporary labels can be calculated concurrently and the final labels only need a small amount of halo data to be obtained.

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

(a) Input binary matrix, A, with dimensions X, Y

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

(b) Unique value mask, B, such that $B_{ij} = iY + j$

1	2	0	0	0
6	0	0	9	10
0	0	13	14	15
0	17	18	19	20

(c) $C^0_{ij} = A_{ij} \cdot B_{ij}$

6	2	0	0	0
6	0	0	14	15
0	0	18	19	20
0	18	18	19	20

(d) $C^{t+1}_{ij} = max\left(C^t_{ij}, nbr(C^t_{ij})\right)$ for $C^t_{ij} \neq 0$

6	6	0	0	0
6	0	0	20	20
0	0	20	20	20
0	20	20	20	20

(e) The final result, when $C^{t+1} = C^t$

Figure 3: A simplistic region detection algorithm in action

3.4 Pattern: Red and White Stripes

This algorithm seeks to find Wally by one of his most identifiable features; his red and white jumper. The algorithm works by producing a binary mask of all the red areas in an image, using `get_colour_in_image`, and a binary mask of all the white areas in an image, using `get_greyscale_in_image`.

3.4.1 Weaknesses

This pattern does have some weaknesses associated with it. Prioritisation of results is hard to get correct. Assuming that Wally is the largest region of red and white stripes fails in any situation where that is not true. Attempting to describe the size of Wally's jumper specifically stop the pattern from scaling to images of different sizes. Furthermore, the

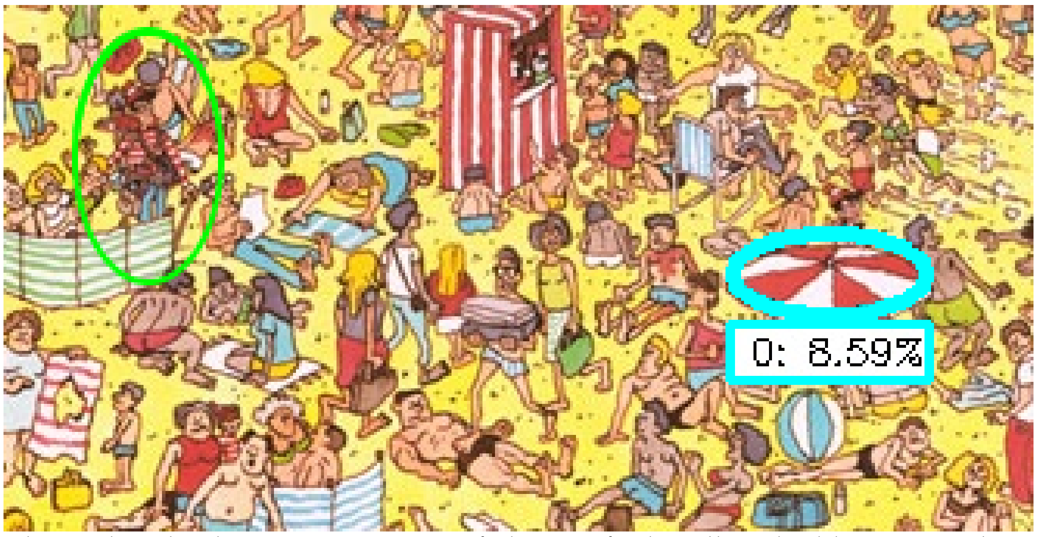


Figure 4: The Red and White Stripes pattern failing to find Wally. The blue ring indicates the top result (an umbrella). The green ring indicates Wally's actual location, which is not even ranked in the top 100 results.

image itself may be flawed to limits in image size. Figure 4 shows an example where Wally's stripes are so small they begin to overlap. This leaves the white stripes being stained pink by the neighbouring red stripes, and so the true Wally's jumper is ignored by the pattern. Meanwhile, larger and clearer stripes show up, giving a result set of entirely false positives.

References

[1] D. G. Lowe, “Object recognition from local scale-invariant features,” in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2, pp. 1150–1157, Ieee, 1999.

[2] K. Tanaka, “Mechanisms of visual object recognition: monkey and human studies,” *Current opinion in neurobiology*, vol. 7, no. 4, pp. 523–529, 1997.

[3] D. I. Perrett and M. W. Oram, “Visual recognition based on temporal cortex cells: Viewer-centred processing of pattern configuration,” *Zeitschrift fur Naturforschung C-Journal of Biosciences*, vol. 53, no. 7, pp. 518–541, 1998.

[4] Q. Zhang, Y. Chen, Y. Zhang, and Y. Xu, “Sift implementation and optimization for multi-core systems,”

[5] H. Bay, T. Tuytelaars, and L. Van Gool, “Surf: Speeded up robust features,” in *Computer Vision–ECCV 2006*, pp. 404–417, Springer, 2006.

[6] L. He, Y. Chao, K. Suzuki, and K. Wu, “Fast connected-component labeling,” *Pattern Recognition*, vol. 42, no. 9, pp. 1977–1987, 2009.