Universe Scientific Publishing

Object Tracking Using Image Segmentation

Tong Lin

Mercer Island High School, WA 98040, USA.

Abstract: Tracking objects in real time is an increasingly common application in computer vision. In this work, the author analyses different methods for tracking an object using conventional image processing algorithms. Different image segmentation methods were developed and their efficacies in detecting objects were analysed on sample images. The image segmentation method selected was subsequently used in video analysis for tracking a particular object across frames. To improve the detection processing, a windowing technique was used to reduce the segmentation computation time. The method developed was able to detect and track objects in real time without significant loss of frame rate.

Keywords: Image Segmentation; Object Tracking; Windowing

1. Objective

The aim of this work was to develop a conventional image processing algorithm for tracking an object across frames in video. It was also the objective of this work to ensure that the tracking computation does not cause noticeable delay in the rendering of the video.

2. Background

In recent times, deep learning based computer vision has become the favourite tool in the computer vision domain for image analysis and tracking. However, training deep learning algorithms requires extensive computational resources which are out of the reach of many individuals. Also, for many applications, it is observed that conventional image processing methods deliver comparable results without appreciable loss of accuracy and or efficiency.

In this work, the author uses MATLAB (*MATLAB*, n.d.) as the computing tool for developing algorithms. While MATLAB provides a comprehensive array of image processing methods in its Image Processing Toolbox, the author decided to use the Toolbox only for basic input and output functions. The core algorithms were developed as MATLAB functions

3. Theory

In computer vision, images are digitally represented using a rectangular array of pixels. With a typical 8 bit representation, each pixel value can vary from 0 to 255. Grayscale images are thus represented by the MxN matrix where M represents the number of rows and N represents the number of columns. Thus there are MxN pixels, each of which varies from 0 to 255. However for colour images, 3 dimensional matrices with 3 channels, one each for R,G,B i.e. MxNx3 pixels are used. A value of (0,0,0) for a certain pixel location represents the colour black whereas (225, 0, 0) represents full red, (0, 255, 0) represents full green, (0,0, 255) represents full blue, (255, 255, 255) represents white with the other colours and saturations being represented by different combination of the pixel values as shown in the colour wheel diagram shown in Fig 1.a

An alternate representation of the same colour image can be realised using the HSV format where H stands for Hue, S for Saturation and V for Value. Hue represents the colour spectrum, saturation represents the intensity of the colour whereas value represents the brightness of the pixel (Cheng et al., 2001).

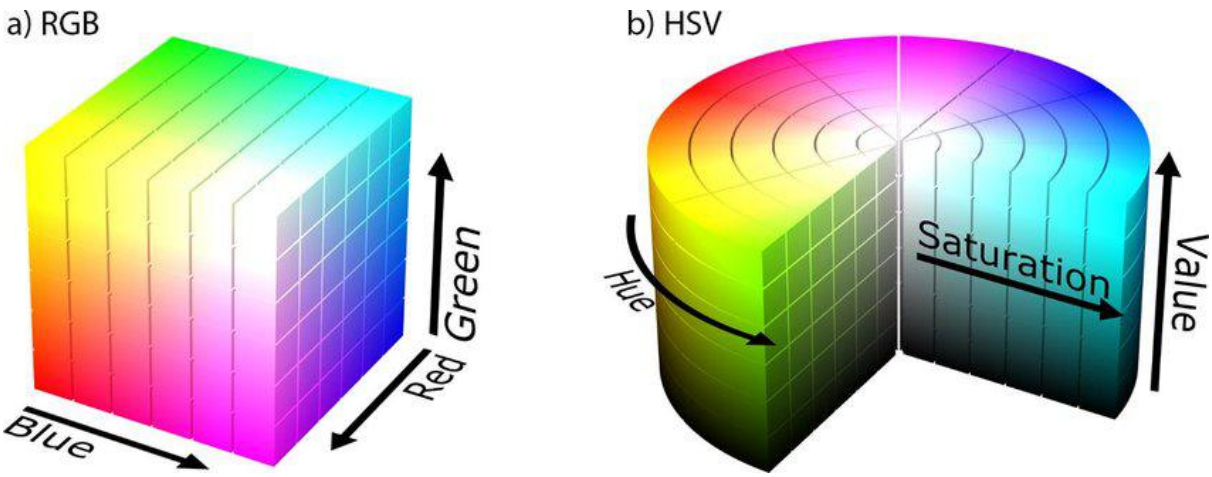

Fig. 1 Image Representation in RGB and HSV color spaces

4. Methods

The author developed different MATLAB functions to track an object. There is the script, ball_tracking_in_video, which iterates through each frame of a video and calls another function called detect_object to try and detect an object inside of each frame of the video. There is also another script, detect_object, which contains the detect_object function.

In the ball_tracking_in_video script, it first reads a video using the VideoReader function. It then creates a counter that keeps track of the number of frames and uses a while loop to iterate through each frame (with the counter being incremented within this loop) of the video. Also, within this loop, the frame, which is in RGB, is converted into HSV. This is an important step because the HSV colour-space separates the intensity of the colour from the actual colour information while the RGB colour-space does not. Using the HSV colour-space makes it easier to disregard potential lighting differences or other inconsistencies, which is ideal for this application. After the frame is converted into HSV, if the counter is equal to one, the detect_object function searches the whole frame to find pixels with a certain hue and saturation, however if the counter is not one, the detect_object function searches in a certain window.

The detect_object function takes the following inputs: an image as a matrix of HSV values, a lower threshold for the intended hue, an upper threshold for the intended hue, a lower threshold for the intended saturation, an upper threshold for the intended saturation, an image as a matrix of RGB values, a boolean indicating whether or not to use a specific window, the previous minimum row for the window, the previous maximum row for the window, the previous minimum column for the window, and the previous maximum column for the window. It outputs the mask of image with only the detected object as a matrix of RGB values, a mask of the image with only the detected object in it, the minimum row where it found the object, the maximum row where it found the object, the minimum column where it found the object, and the maximum column where it found the object. To create the mask with only the detected object highlighted, it first initialises an empty matrix which has the same size as the image as a matrix of HSV values. The function then does different actions based on whether or not a window should be used (indicated in the inputs) and whether or not the function can find the object. If a window should be used, a window is created that is 20 pixels wider on each side than the previous maximum and minimum row and column numbers. However, if the values of the window goes beyond the boundaries of the image, the image boundaries are used instead. Otherwise if a window should not be used or if the function could not find an object in the previous frame, the window is set to be the boundaries of the whole image. After the window is set, the function iterates through the window to find any pixels within the hue and saturation thresholds, changing the colour in those areas to be white in both the RGB and non-RGB mask of the original image. Lastly it finds the minimum and maximum row and column in the non-RGB mask where the values are not black and outputs them.

For initial hypothesis testing, we tested the detection of blue, red and yellow coloured ball in Fig 2.a using HSV and RGB methods. As can be seen, both the methods perform reasonably well for blue colour, with the RGB result show in Fig 2.b and HSV result shown in Fig 2.c. However, as the colour composition values in Fig 3 indicate, RGB based image segmentation would fail for the red ball as the orange ball in Fig 3.b has a red channel value of 254 which is higher than that for the red ball at 219. Even the yellow ball has a red channel value of 237, higher than that of the red ball. RGB method also cannot detect composite colours like orange and yellow which are made of mixed proportions of red and green. However, HSV method is successful at detecting composite colours such as yellow as the HSV colour wheel can be used to isolate the yellow hue in the Hue channel of the image. It is also able to segment the red ball as shown in Fig 4.a

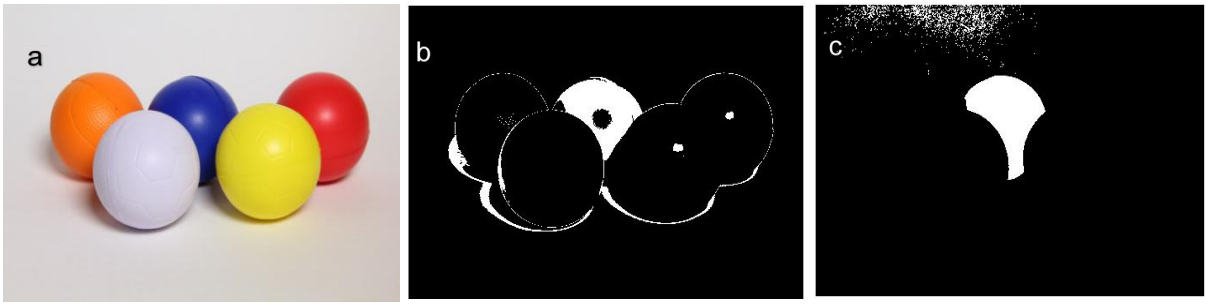

Fig. 2 Blue Ball Detection a) Original Image b) RGB based segmentation and c) HSV based segmentation

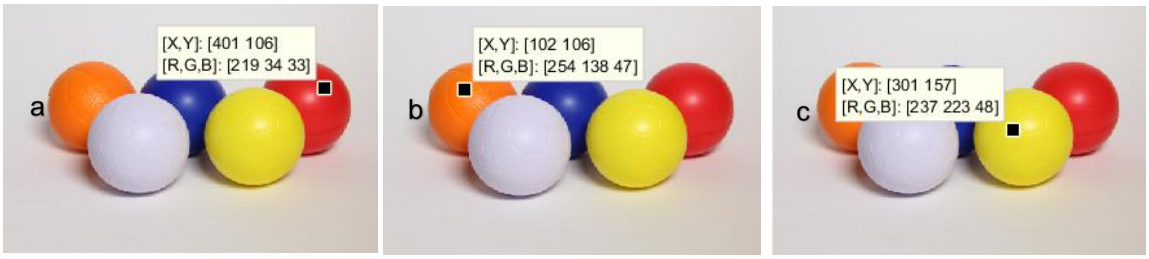

Fig. 3 RGB Values of a) Red Ball, b) Orange Ball and c) Yellow Ball

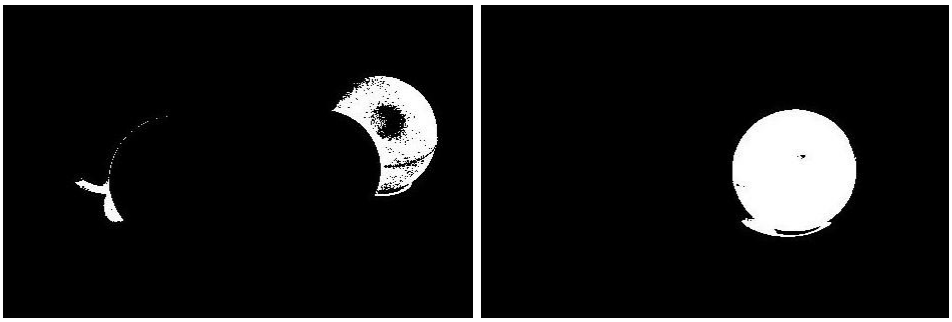

Fig. 4 HSV Based Segmentation of a) Red Ball and b) Yellow Ball

Once the HSV based segmentation method was finalised for the object detection, it was then used to track a green ball across a video by reading frames, processing each frame using the object detection function and then republishing the frame. However, the processing time for detecting the ball in the whole image caused a significant lag in the video playback. Hence, we added a feature for searching for the green ball in the neighbourhood of its last known location. For this, we implemented a windowing function, where we find the top and bottom rows and left and right columns for the detected ball in the current image, as shown by the orange rectangle in Fig 5. These values are then passed to the detection function in the next frame,

and then a window with a certain number of pixels larger than the current window on all sides is used to search for the ball. The window to be checked for the ball is shown with the red rectangle in Fig 5. Thus the number of pixels to be checked for the presence of the object reduces from the whole image to a small subset of it. When the current frame cannot detect the object, then the detection algorithm uses the whole image again in the next frame to search for it. Thus when occlusions happen or the object goes out of the frame view, the algorithm is able to resume the tracking of the object.

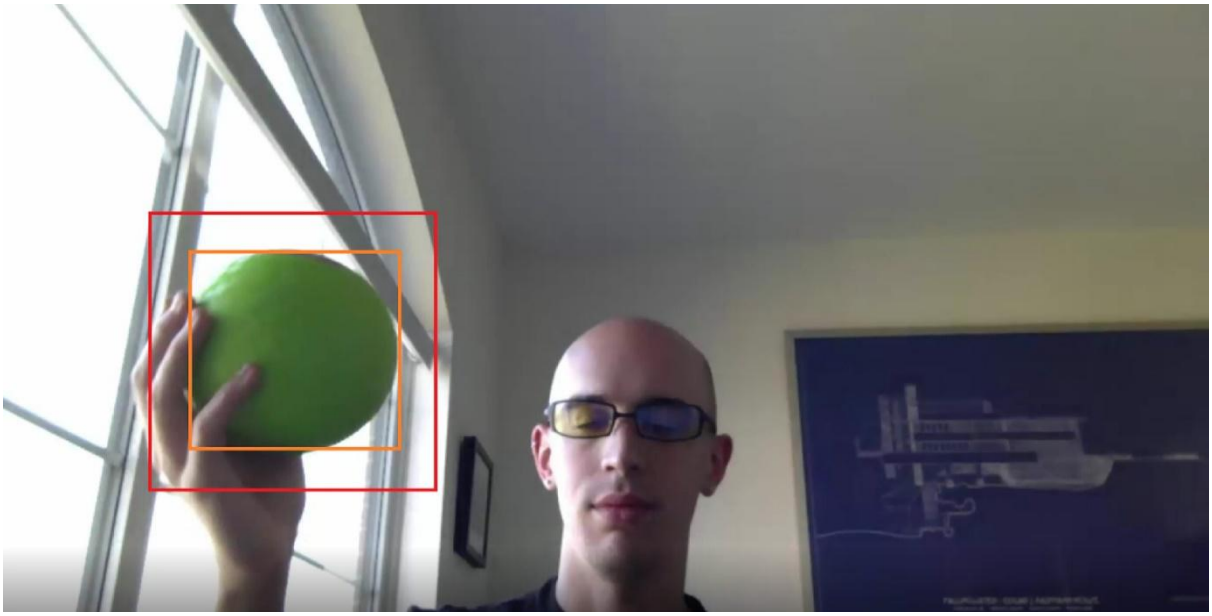

Fig. 5 Detection of Green Ball with Windowing

5. Results and Observations:

In order to validate our approach, we carried out a series of experiments to check the suitability of our approach. I applied the ball tracking algorithm to detect the green ball in the video found at (Rosebrock, 2015). The detection with windowing and without windowing technique was executed on a MacBook Pro Intel Dual Core 2.9 GHz, 8GB DDR3 RAM machine. As can be seen from Table 2, tracking without windowing took significantly longer time than tracking with windowing. The mean time for tracking with windowing was approximately one-third of the time taken for tracking without windowing.

Table 1: Comparison of time taken for tracking with and without windowing

	Tracking without windowing (secs)	**Tracking with windowing (secs)**
	175.660	57.791
	153.551	70.621
	149.201	46.665
Mean	159.471 ∓ 14.881	58.3590 ∓ 11.998

Conclusion

In this work, the author initially investigated different methods for segmenting objects based on their colour. While RGB based segmentation worked to some extent for primary colours, it failed for composite colours. To mitigate the shortcomings of RGB based method, HSV based segmentation was used. It performed consistently well for both primary and composite colours. The HSV based segmentation method developed for colour images was then applied for tracking a ball in a video by extracting each frame and passing it through the segmentation method. The windowless segmentation method increased the latency beyond acceptable limits as it was searching for the ball in the entire image in each frame. A windowing technique where the ball is searched in the neighbourhood of its previous known location was used to reduce the latency to one-third, thereby ensuring real time tracking.

References

[1] Cheng, HD., Jiang, XH., Sun, Y., & Wang, JL., Color image segmentation: advances and prospects. Pattern Recognition, 34(12), 2259-2281. https://doi.org/10.1016/S0031-3203(00)00149-7. December 2021.

[2] MATLAB. (n.d.). MathWorks. Retrieved January 9, 2022, from https://www. mathworks. com/products/matlab.html. January 9, 2022.

[3] Rosebrock, A. OpenCV Track Object Movement. PyImageSearch. Retrieved January 9, 2022, from https://www.pyimagesearch.com/2015/09/21/opencv-track-object-movement/ September 21, 2015.