1. Task One: Count and Marking Eggs

The source image is read in and converted to an HSV image. HSV was used because it is robust to different lighting conditions, as well as the shiny surface of the eggs. For each colour (blue, red, green, orange, yellow, white), a range of HSV values were created as parameters. These are summarised in table 1 and were sampled using the photoshop colour tool.

Table 1: HSV Ranges for egg colours

Colour	H (Min)	S (Min)	V (Min)	H (Max)	S (Max)	V (Max)
Red	0	100	100	5	255	255
Blue	75	0	0	125	255	255
Green	55	100	100	80	255	255
Yellow	16	100	100	30	255	255
Orange	11	88	100	15	255	255
White	0	0	100	25	150	255

A helper function called “createMask” was implemented to create a clean edge for each colour. This function calls the “inRange” function to create a mask subject to each HSV range specified in the table above. This mask is then blurred using a median blur with a kernel size of 15. A canny edge detector is then applied to create an edge. These edges are then dilated to connect any edges that weren’t connected, and then dilated to reduce the edge thickness. These steps are illustrated for the blue eggs in Figures 1 to 4.

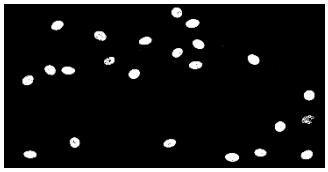

Figure 1: inRange function applied to blue eggs

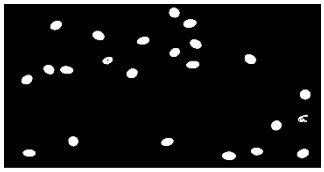

Figure 2: Median Blur applied

Figure 3: Canny edge detection

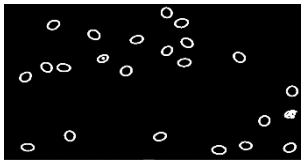

Figure 4: Dilation

A “contourImage” function was then implemented for each corresponding colour mask. This function calls the “findContours” function which creates a vector of arrays of each contour in the image. This vector is looped through in a for loop, and if the area of the contour is greater than a threshold area, the contour is considered a valid egg, and the contour is drawn on the original image using the “drawContours” function. This logic is summarised in Figure 5 and is repeated for each egg colour.

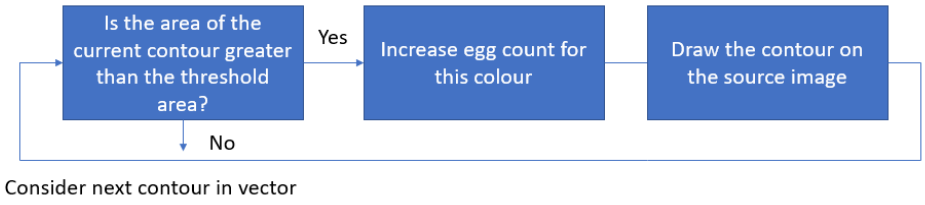

Figure 5: Logic for counting and drawing eggs

A total egg count of each egg is then displayed to the command prompt, and the final image with marked eggs is saved using ”imwrite”.

2. Task Two

The source image is read in and converted to an HSV image. It is then passed through the "createMask" helper function described in task 1. The image is then passed through the Canny edge detection process. This edge is dilated to connect edges that aren't already, and then eroded to decrease the edge size. These steps are illustrated in Figures 6 through to 8.

Table 2: HSV range for eggs in task 2

Colour	H (Min)	S (Min)	V (Min)	H (Max)	S (Max)	V (Max)
Red	0	100	100	12	255	255

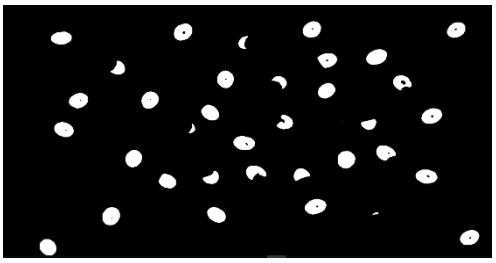

Figure 6: inRange & blurred

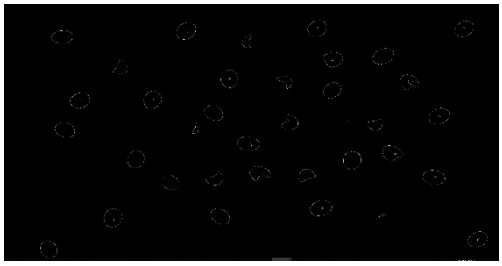

Figure 7: Canny edge detection

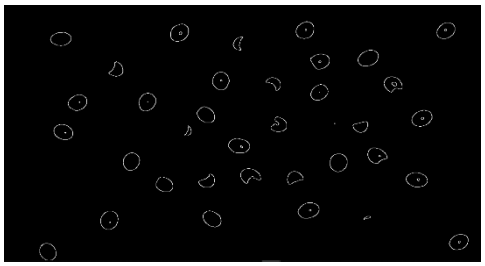

Figure 8: Dilation then erosion

The "findContours" function is then used to create a vector of arrays of the contours of the egg pieces. At this stage, both valid and broken eggs are contoured. The "convexHull" function is then used to create a vector of arrays of convex hulls for all the contours that were found.

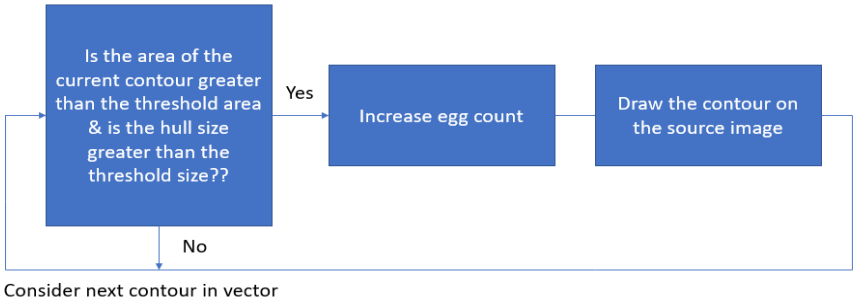

Figure 9: Logic to determine if a contour is a valid egg

A helper function "contourImage" is then called to loop through the vectors of corresponding contours and hulls, if the contour area size and the size of the hull (the length of array) are greater than a threshold value, then the contour is considered a valid egg and the surrounding hull is drawn on the original source image using "drawContour". This logic is summarised in Figure 9. The contour area size is useful as valid eggs generally have a greater area than broken pieces of egg. However, for some eggs that are only slightly broken, the area size of a contour is still similar to valid eggs. Therefore the number of points in the represented hull (hull.size()) is needed as a second condition. A long flat edge (as present in some of the broken eggs) is represented in a small number of points in convex hull form, therefore eggs that are broken tend to have a smaller hull.size() than those that aren't.

Note that there is one egg near the top left-hand corner that may possibly be broken, but is considered whole by the algorithm. This is because the flat edge is so small relative to other flat edges in other broken edges, that the hull.size() value is still similar to a whole egg.

3. Task 3: Counting and Classifying Chocolate Pieces

Pre Processing

The image is processed into Grey Scale thresholded using “threshold”, blurred using “medianBlur”, and then passed through a canny edge detector, eroded and then dilated.

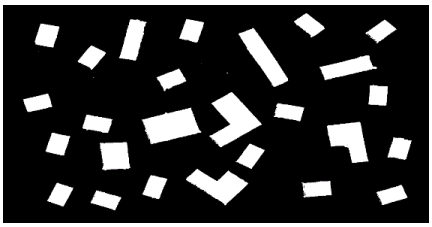

Figure 10: Thresholded image

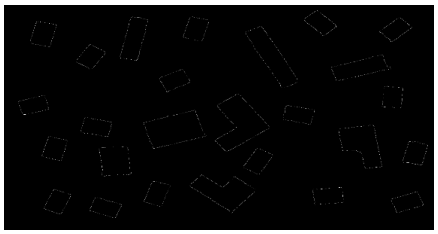

Figure 11: Canny edge detection

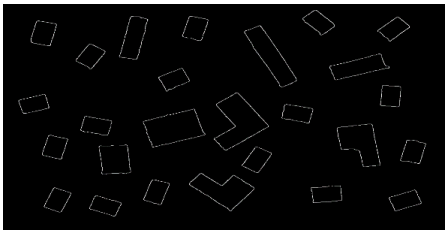

Figure 12: Dilated and eroded

Processing

The “findContours” function is called to contour each piece of chocolate. The “minAreaRect” function is used to bound chocolate contour with the minimum rotated rectangle. The area of an average single piece of chocolate is divided through all the contour area’s (the result is rounded) to calculate the number of pieces of chocolate per block.

Classification

By convention, the height of a single piece of chocolate was considered as the shortest edge – this was found on average to be a value of 180 pixels. Comparatively the width was found to be 120 pixels. The range between the biggest height and smallest height for individual cubes was 19 pixels. These values are used in the classification algorithm.

Figure 13: individual chocolate piece

There are four possible numbers of chocolate per block: one, two, three, four. Each contour is looped through subjected to the decision tree criteria in Figure 14. Because there are only two different configurations for block numbers of 2,3 and 4, the algorithm simplifies the task by searching for one of those configurations, and if it doesn’t find that one, by default it is the other. Because there is a possibility that

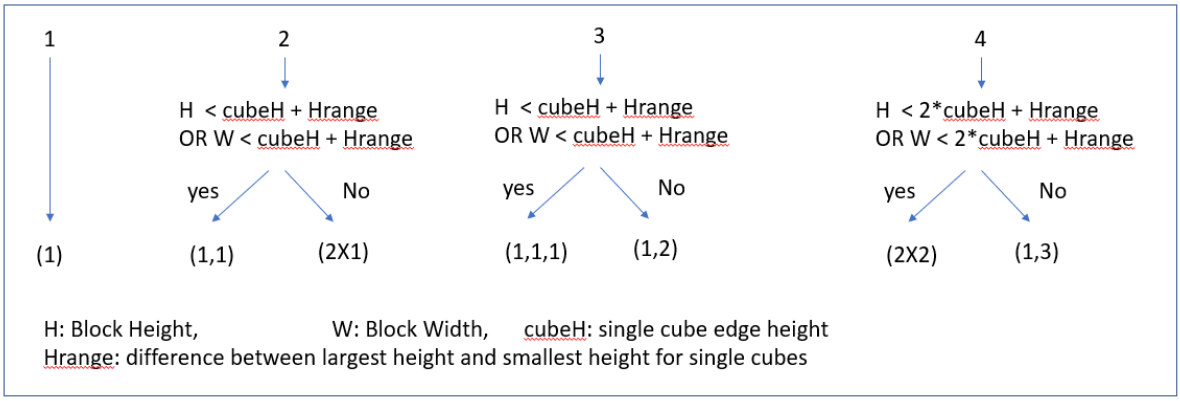

Figure 14: Decision tree for classifying configuration

minAreaRect may miss-classify a rectangle width as an edge or vice versa, the decision criteria for 2, 3 and 4 accounts for this possibility with an OR statement.

The “moment” function is then used to find the centre of each piece of chocolate. Each time a condition for a configuration is met, a counter for that configuration is increased, and the “putText” function is used to draw the configuration text on the corresponding contour on the output image.

4. Object Detection and Categorisation

This task is achieved by processing the source image lolly by lolly.

Chocolate

First, the "createMask" function is called to create a mask for chocolate pieces. In this function, the Mat object is put through a canny edge detector, then dilated and eroded. "FindContours" is then used on the mask to create a vector of potential chocolate contours. The vector is then looped through and if the contour is greater than a threshold area, it is considered a piece of chocolate.

Gummy Egg

The gummy egg is found by using the "createMask" function for white, then using the canny edge detector, dilation and erosion, "findContours" and then finally by checking the contour area against a threshold area.

Block Lollies

There are four different colours of block lolly – red, green, yellow and brown. Four individual masks were created for each colour using the "inRange" function. The "bitwise_or" function was then used to combine the masks into one mask with all block lollies. This mask was then dilated then eroded to connect the block lolly edges. The "findContour" function was then used to find potential block lolly contours. Because the heart lolly HSV values fall within the range of the red block lolly, the heart lolly contours were capture in this vector of contours. The "convexHull" function is used to create a vector of hull points for each contour. This is because the size of the heart hulls is greater than the size of the block hulls. Therefore the vector of hulls is looped through, and if the hull size is within the threshold range, a block lolly is counted.

Table 3: HSV Range values for lollies

Lolly	Min Hue	Min Saturation	Min Value	Max Hue	Max Saturation	Max Value
Chocolate	6.5	100	100	15	255	255
Gummy Egg	20	58.65	100	23	255	255
Block Lolly Green	33	102	114.75	39	178.5	137.7
Block Lolly Brown	9.5	117.3	114.75	13.5	170.85	204
Block Lolly Red	1.5	117.5	183.6	7	175.95	234.6
Block Lolly Yellow	23	176	199	25	235	214.2

A similar process was used for hearts, gummy bears and coke bottles. Because this task only specifies classifying "chocolate, gummy eggs and block lollies", the output for the remaining lollies is suppressed.