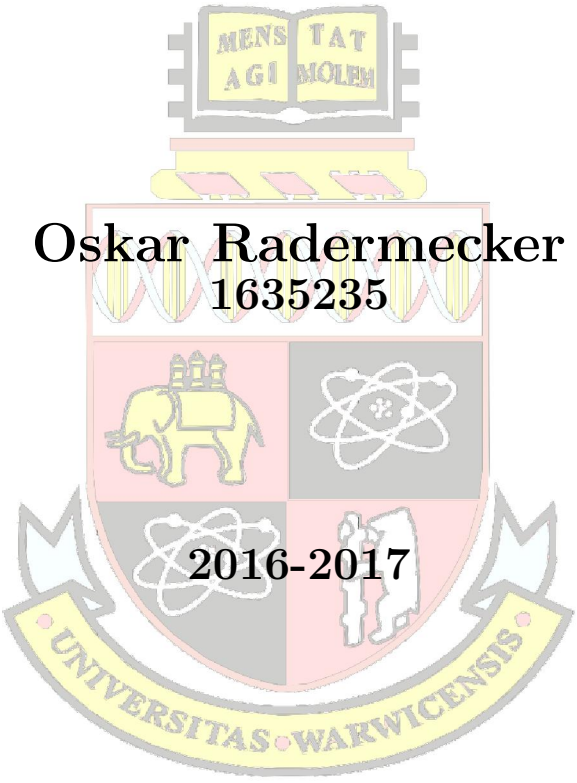
THE UNIVERSITY OF WARWICK

University of Warwick

CS413

Coursework Assignment 2016 in MATLAB

Problem 2 - Image Segmentation



Oskar Radermecker
1635235

2016-2017

Contents

List of Figures

1. Introduction

The aim of this assignment was to solve one of the three problems given in class. Problem 2 concerning image segmentation was chosen. The aim of this task was to develop a colour segmentation algorithm capable of detecting Lego bricks of different colours against both a plain and cluttered background. The algorithm was developed in Matlab. The assignment is split into four tasks addressed in four different chapters:

1. Code, explain and describe the colour segmentation algorithm

2. Investigate the effect of a Gaussian distributed random noise on the performances of your algorithm

3. Extract the Lego bricks from the original image and order them according to their size in a new window

4. Test your algorithm on a new image and discuss its performances

2. Colour Image Segmentation Algorithm

This algorithm can be divided into two parts:

1. Separation of the different colours

2. Object identification

The images presented in Chapters 2, 3 and 4 were extracted from `lego-bricks-1.jpg`, as this picture has the most plain background and is thus the easiest one. The algorithm was also run on the two other pictures, providing mitigated results, especially for yellow and black, in `lego-bricks-3.jpg`. These will not be presented here to keep some consistency.

2.1 Separation Part

This is the separation of the different colours in the original picture into clusters. Several algorithms were investigated for this purpose including k-means algorithm, histogram based methods, edge detection and region growing. In the end, two solutions were chosen: a k-means algorithm in the L*a*b* space and a thresholding method in the HSV space. As thresholding provided better results and was much quicker to run, only its results were used in Chapters 3 and 4. Some results of the k-means clustering are nevertheless presented in the Appendix. Sobel edge detection was also implemented, but the results were quite poor, and it was not included in the final version of the algorithm. The code is presented in `Sobel.m`.

2.1.1 K-means Algorithm

This is a method of dividing an image into k-clusters while minimising a function (s.a standard Euclidean distance). The method is quite simple to implement and guarantees a result, even if the solution may not be optimal. RGB, HSV, YCbCr and L*a*b* spaces were investigated but the best clustering was obtained in the L*a*b* space. Only planes a* and b* were used as L* represents lightning, not crucial for this application.
This is realised by the built-in MATLAB function : *kmean*[1].
The number of clusters needs to be set manually. An optimal value was empirically chosen : 5 clusters for `lego-bricks-1.jpg`, 6 for `lego-bricks-2.jpg` and 7 for `lego-bricks-3.jpg`.

2.1.2 Thresholding Method

Thresholding has the advantage of isolating a single colour while being simple to implement. It is also much quicker to run than the 1^{st} method. However, the hue, saturation and value thresholds are optimised for specific images (`lego-bricks-1`, `lego-bricks-2` and `lego-bricks-3`) and might not work as well on other images for which these thresholds might be different.
Inspired from [7], this second algorithm has pre-entered threshold values in a `struct` which are then applied to the image as masks. Six colours are separated: Red, Green, Yellow, Dark Blue, Cyan and Black as it was not possible to isolate White.
Note: Red is separated differently than the other colours, improving clustering (figures 2.1 and 2.2). This was achieved by keeping both very high and low hue values.

[1]This function is ran three times to be sure to find an as optimal as possible solution.

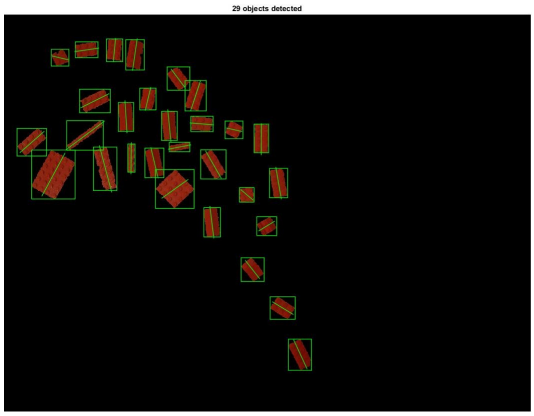


Figure 2.1: Red colour clustering with the improved method

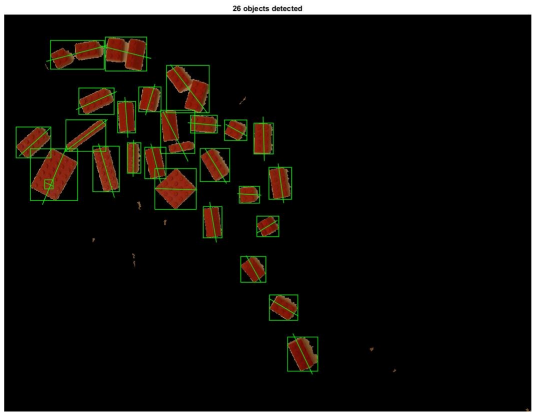


Figure 2.2: Red colour clustering with the standard method

2.2 Identification Part

After having been isolated, each r/g/b plane from each cluster is processed and binarized. The binarization threshold is calculated using Otsu's method which minimises the intra-class variance of two classes of pixels. Then, a built-in MATLAB function (*bwconncomp*) identifies the different objects in the cluster. This method works well on objects which are not too close to each other, but when bricks touch or overlap, they can be recognised as one single element and not be separated. Other errors such as multiple detections or no detection of an object (white bricks) can also occur.

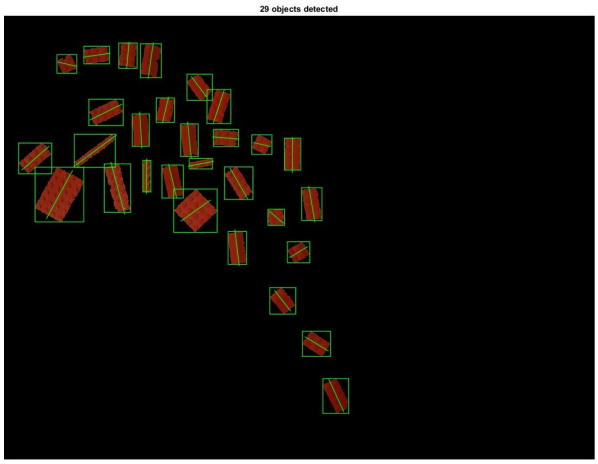


Figure 2.3: Red colour clustering by the thresholding algorithm in the HSV Colour Space

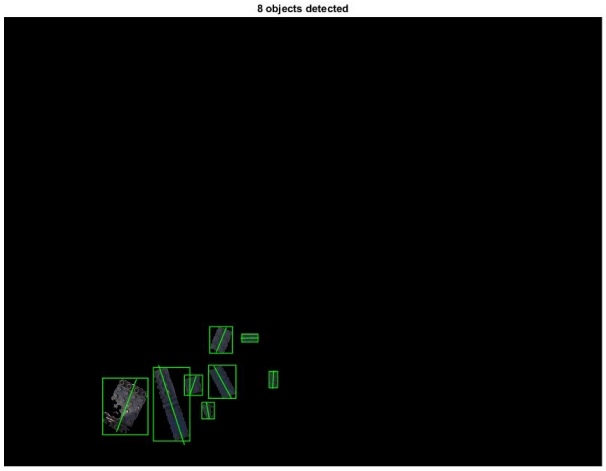


Figure 2.4: Dark blue colour clustering by the thresholding algorithm in the HSV Colour Space

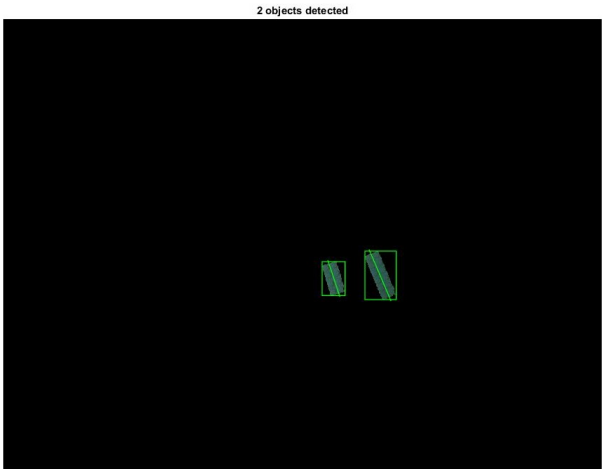


Figure 2.5: Cyan colour clustering by the thresholding algorithm in the HSV Colour Space

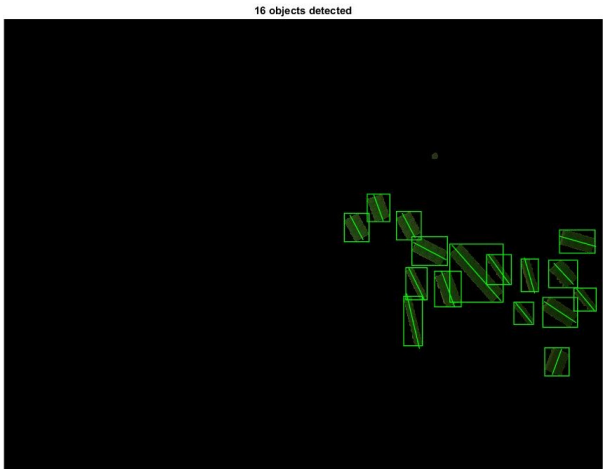


Figure 2.6: Green colour clustering by the thresholding algorithm in the HSV Colour Space

Figure 2.7: Yellow colour clustering by the thresholding algorithm in the HSV Colour Space

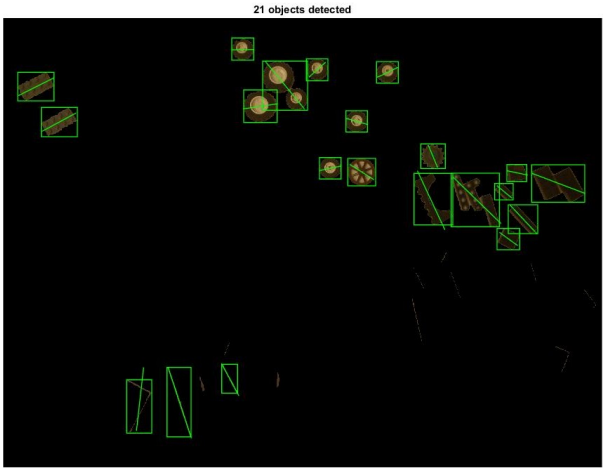


Figure 2.8: Black colour clustering by the thresholding algorithm in the HSV Colour Space

Table 2.1 summarises results and errors of the algorithm.

End result : 97 detected and isolated objects while 116 bricks were counted manually $\Rightarrow$ correctness of 83.6%. Detected bricks = $\frac{97+8}{116} = 90.5\%$.

These values were optimised for the **three** Lego-bricks pictures, leading to the formation of artefacts such as the blue bricks in the black cluster (figure 2.8). Increasing the Low Hue threshold removes them but prevents the recognition of wheels in the second image. Also, the black cluster with `lego-bricks-3.jpg` presents many artifacts due to the background. They can be eliminated be increasing the minimum size of objects in

Colour	Real Total Number	Detected and Isolated	Not Detected	Detected but not Isolated	Other Remarks
Red	29	29	0	-	-
Green	16	16	0	-	-
Dark Blue	8	11	0	-	Three detected on two different clusters
Cyan	2	2	0	-	-
Yellow	24	21	0	2×2 together and 1×3 together	One doubly detected
Black	10	7	0	1×2 together and 1×3 together	One only partially
Wheels	8	7	0	1×2 together	-
Gray	3	3	0	-	-
Brown	1	1	0	-	-
Beige	1	0	1	-	-
White	14	0	14	-	-
Total	116 =	97	+ 15	+ 8	- 4

Table 2.1: Error Assessment Table for manual threshold separation

identify_objects but can also remove some correctly detected bricks.

3. Effect of Gaussian distributed random noise

Gaussian Distributed Random Noise was added to achieve an SNR of 1, 10 and 20. The definition of the SNR used is given in the Appendix.

As noise is random and created at the beginning of each trial, results may be different from trial to trial.

Note: it was tried to filter the noisy image using a Gaussian filter before processing but this decreased the quality of the results, and the idea was abandoned (`Code.m`, line 21).

3.1 SNR = 1

Optimised for `lego-bricks-1.jpg` by decreasing *HueLow* for yellow from 0.095 to 0.05.

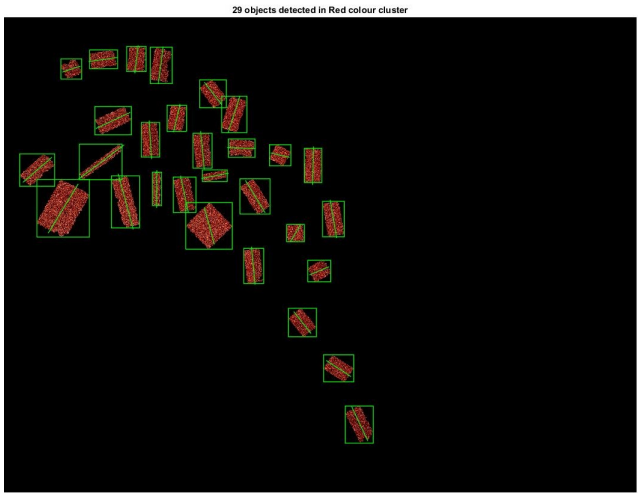


Figure 3.1: Red colour clustering by thresholding with Gaussian noise, SNR = 1

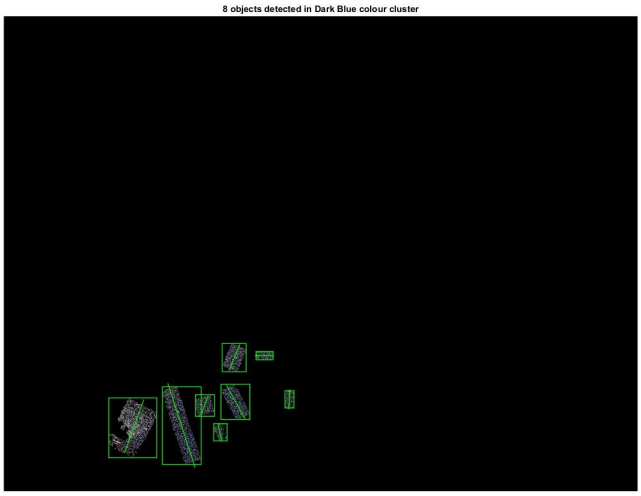


Figure 3.2: Blue colour clustering by thresholding with Gaussian noise, SNR = 1

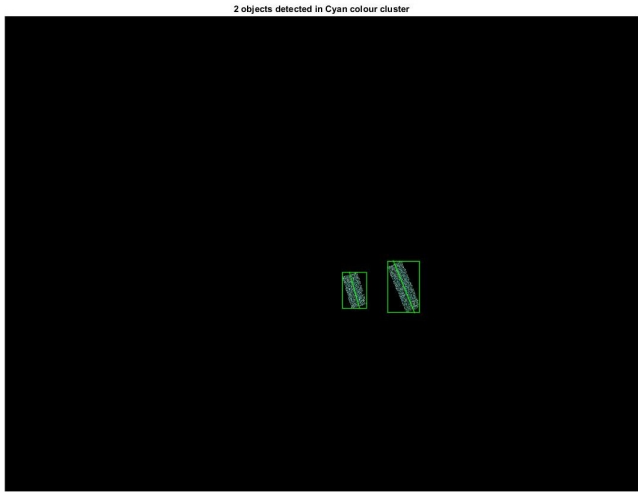


Figure 3.3: Cyan colour clustering by thresholding with Gaussian noise, SNR = 1

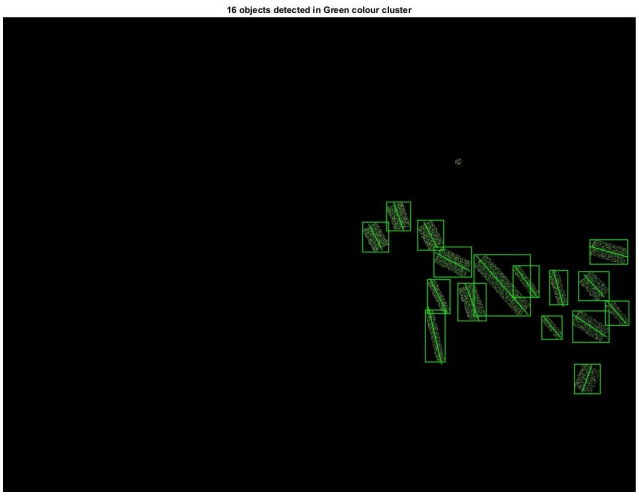


Figure 3.4: Green colour clustering by thresholding with Gaussian noise, SNR = 1

Figure 3.5: Yellow colour clustering by thresholding with Gaussian noise, SNR = 1

Figure 3.6: Black colour clustering by thresholding with Gaussian noise, SNR = 1

Colour	Real Total Number	Detected and Isolated	Not Detected	Detected but not Isolated	Other Remarks
Red	29	29	0	-	-
Green	16	16	0	-	-
Dark Blue	8	9	0	-	One detected on two different clusters
Cyan	2	2	0	-	-
Yellow	24	22	3	1×2 together	Two doubly detected
Black	10	9	0	1×3 together	One doubly detected
Wheels	8	6	0	1×3 together	-
Gray	3	3	0	-	-
Brown	1	1	0	-	-
Beige	1	0	1	-	-
White	14	0	14	-	-
Total	116 =	97	+ 18	+ 5	- 4

Table 3.1: Error Assessment Table for manual threshold separation with SNR = 1

3.2 SNR = 10

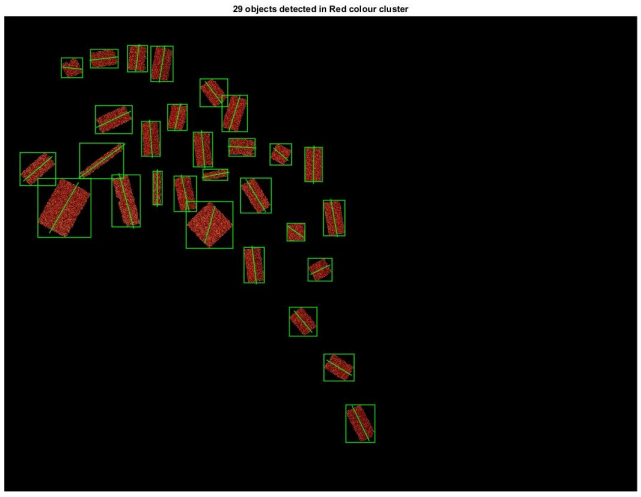


Figure 3.7: Red colour clustering by thresholding with Gaussian noise, SNR = 10

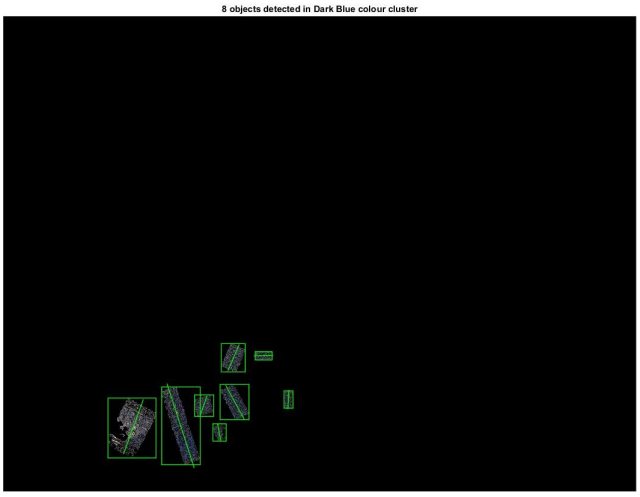


Figure 3.8: Blue colour clustering by thresholding with Gaussian noise, SNR = 10

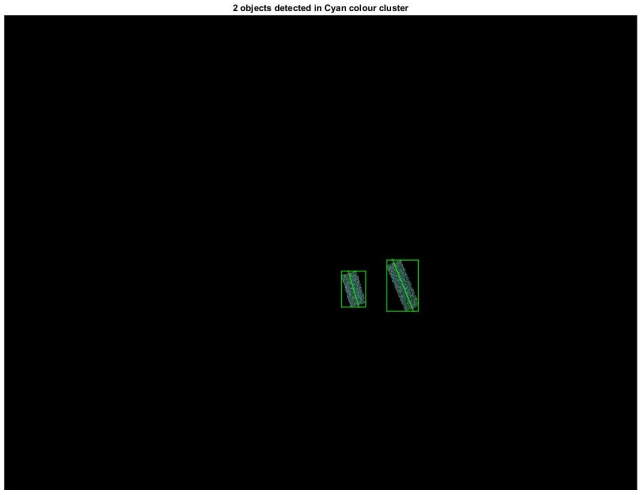


Figure 3.9: Cyan colour clustering by thresholding with Gaussian noise, SNR = 10

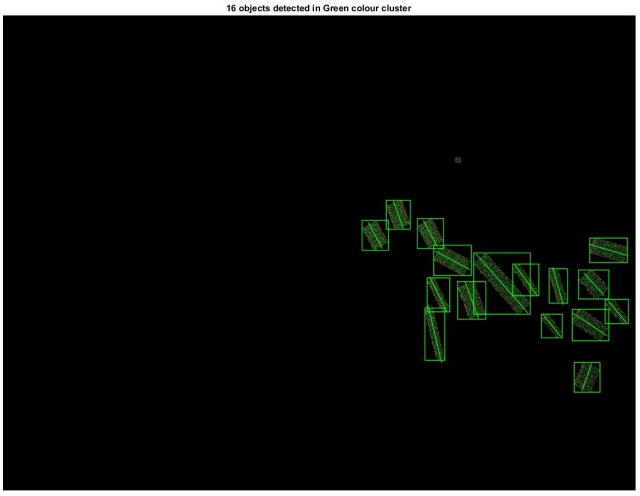


Figure 3.10: Green colour clustering by thresholding with Gaussian noise, SNR = 10

Figure 3.11: Yellow colour clustering by thresholding with Gaussian noise, SNR = 10

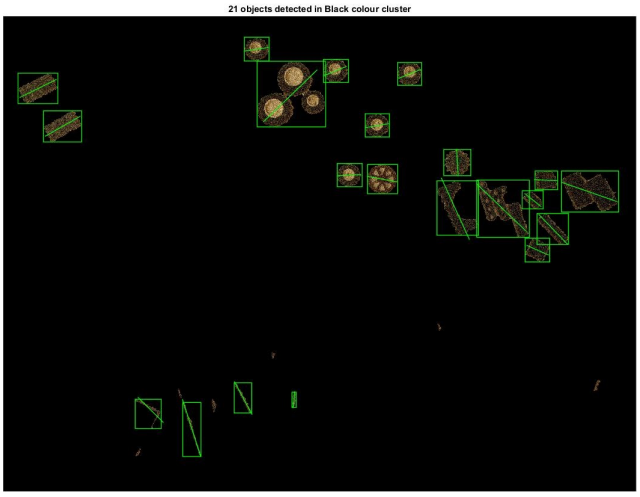


Figure 3.12: Black colour clustering by thresholding with Gaussian noise, SNR = 10

Colour	Real Total Number	Detected and Isolated	Not Detected	Detected but not Isolated	Other Remarks
Red	29	29	0	-	-
Green	16	16	0	-	-
Dark Blue	8	12	0	-	Four detected on two different clusters
Cyan	2	2	0	-	-
Yellow	24	22	2	1×2 together	One doubly detected
Black	10	7	0	1×2 together and 1×3 together	-
Wheels	8	6	0	1×3 together	-
Gray	3	3	0	-	-
Brown	1	1	0	-	-
Beige	1	0	1	-	-
White	14	0	14	-	-
Total	116 =	98	+ 17	+ 6	- 5

Table 3.2: Error Assessment Table for manual threshold separation with SNR = 10

3.3 SNR = 20

Optimised for `lego-bricks-1.jpg` by increasing *HueLow* for black from 0.02 to 0.085. Only for .

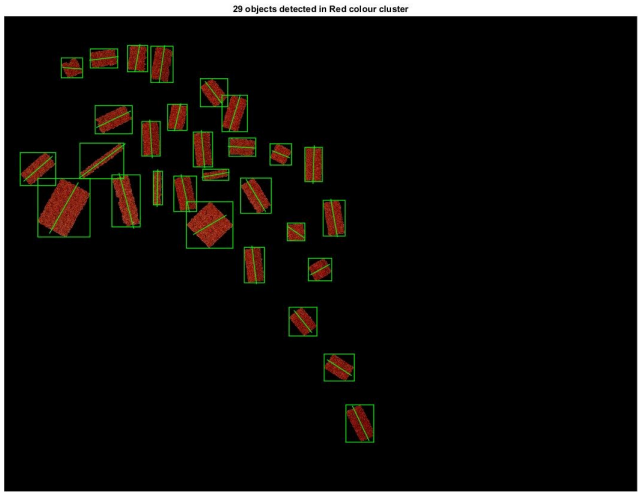


Figure 3.13: Red colour clustering by thresholding with Gaussian noise, SNR = 20

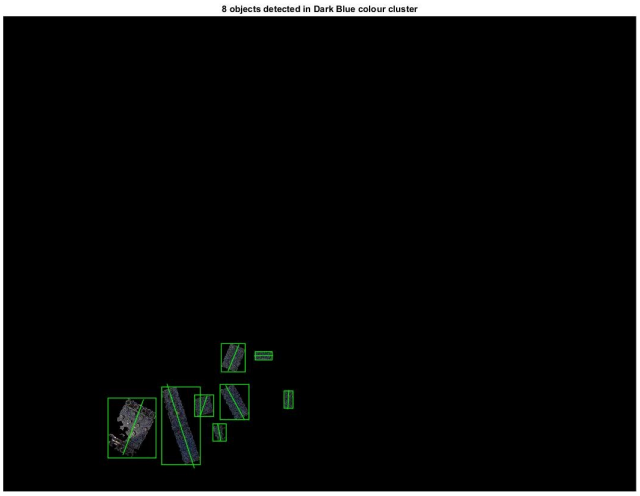


Figure 3.14: Blue colour clustering by thresholding with Gaussian noise, SNR = 20

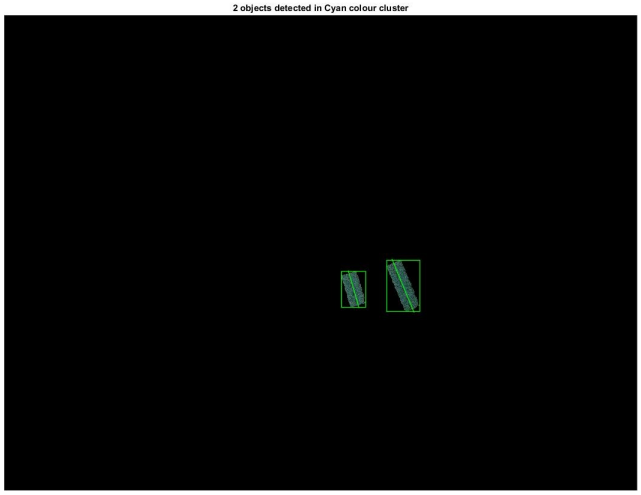


Figure 3.15: Cyan colour clustering by thresholding with Gaussian noise, SNR = 20

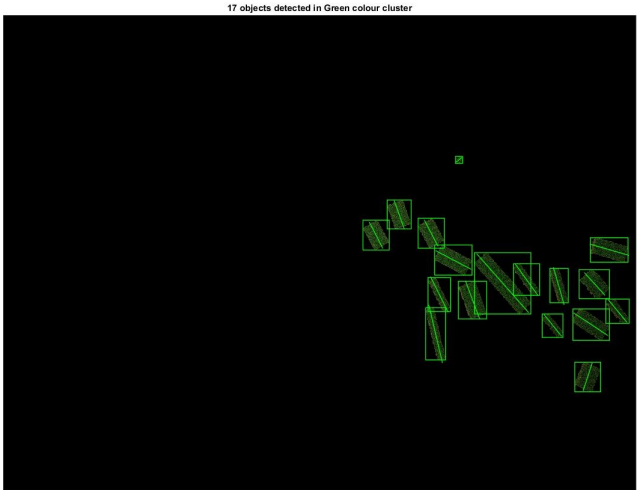


Figure 3.16: Green colour clustering by thresholding with Gaussian noise, SNR = 20

Figure 3.17: Yellow colour clustering by thresholding with Gaussian noise, SNR = 20

Figure 3.18: Black colour clustering by thresholding with Gaussian noise, SNR = 20

Colour	Real Total Number	Detected and Isolated	Not Detected	Detected but not Isolated	Other Remarks
Red	29	29	0	-	-
Green	16	17	0	-	Green Tip on gray brick detected
Dark Blue	8	8	0	-	-
Cyan	2	2	0	-	-
Yellow	24	22	0	2×2 together	-
Black	10	7	0	1×2 together and 1×3 together	-
Wheels	8	7	0	1×2 together	-
Gray	3	3	0	-	-
Brown	1	0	1	-	-
Beige	1	0	1	-	-
White	14	0	14	-	-
Total	116 =	95	+ 16	+ 6	- 1

Table 3.3: Error Assessment Table for manual threshold separation with SNR = 20

4. Extraction of bricks

Bricks of the same colour were extracted from the cluster, orientated upwards, sorted by area and pasted in a new image. This part was tricky as "SubarrayIdx", from *regionprops*, extracted bricks in black and white. A new matrix multiplication was defined to achieve a colour extraction.

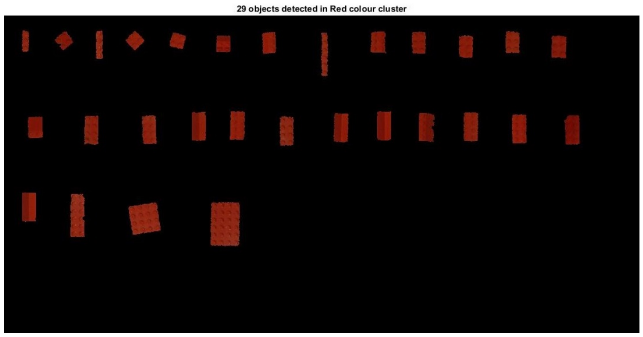


Figure 4.1: Red bricks extraction

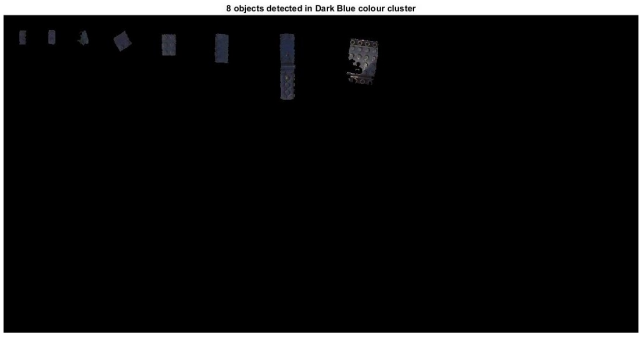


Figure 4.2: Blue bricks extraction

Figure 4.3: Cyan bricks extraction

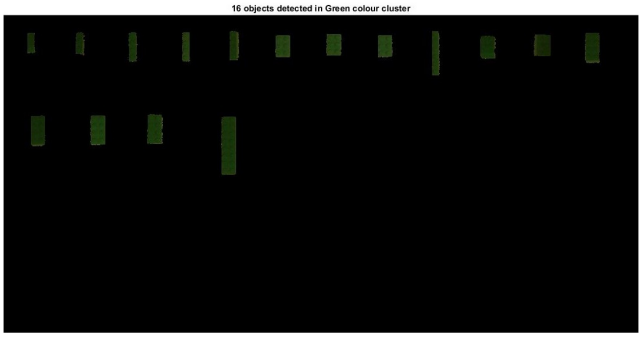


Figure 4.4: Green bricks extraction

Figure 4.5: Yellow bricks extraction

Figure 4.6: Black bricks extraction

5. Test on chosen image

For this test, two images of M&M's were taken.

Figure 5.1: Plain M&Ms image

Figure 5.2: Cluttered M&Ms image

To make the algorithm work on these pictures, they were two possible changes:

1. Change the thresholds
2. K-means Clustering

5.1 Change of Thresholds

One possible change was to modify, adapt and optimise the thresholds to the new image. This was realised manually for the plain background image using the *Colour Thresholder*

App in MATLAB.

5.1.1 Plain Background

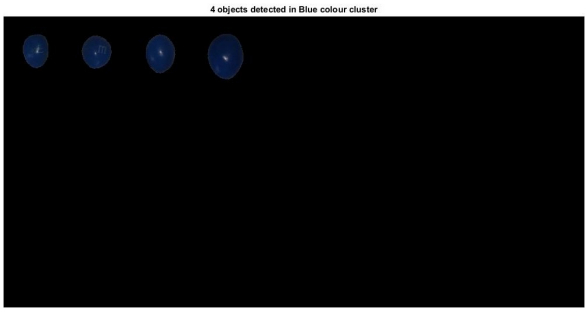


Figure 5.3: Blue M&M's extraction with HSV tresholding with plain background

Figure 5.4: Green M&M's extraction with HSV tresholding with plain background

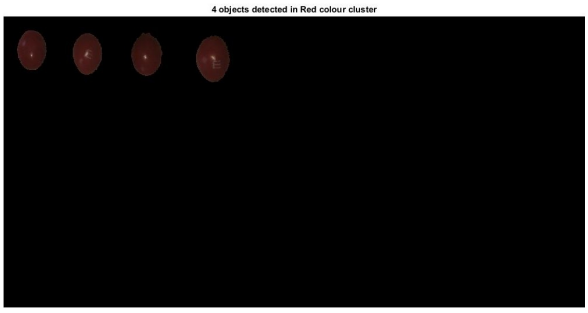


Figure 5.5: Red M&M's extraction with HSV tresholding with plain background

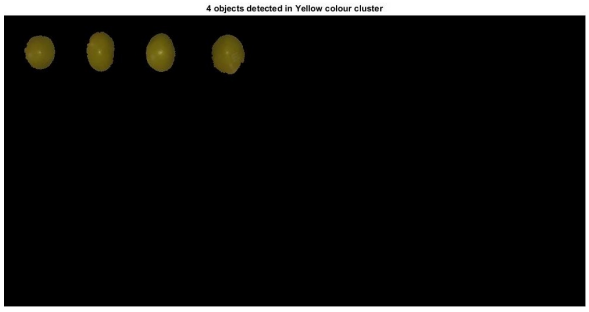


Figure 5.6: Yellow M&M's extraction with HSV tresholding with plain background

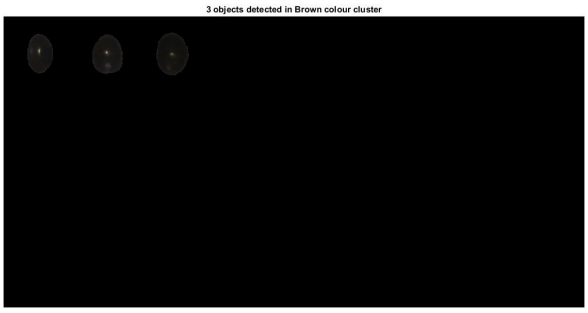


Figure 5.7: Brown M&M's extraction with HSV tresholding with plain background

Figure 5.8: Orange M&M's extraction with HSV tresholding with plain background

5.1.2 Cluttered Background

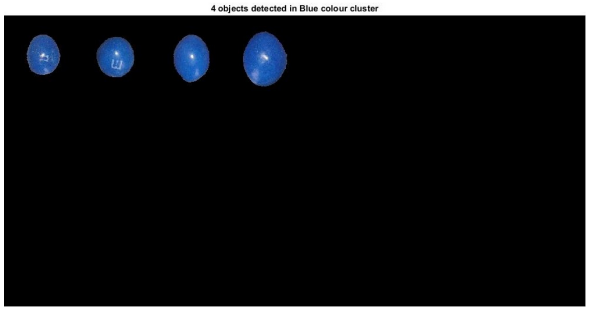


Figure 5.9: Blue M&M's extraction with HSV tresholding with cluttered background

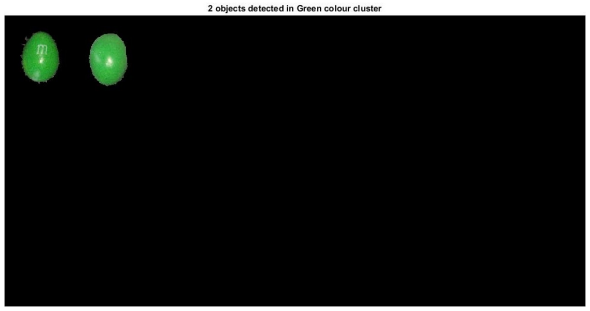


Figure 5.10: Green M&M's extraction with HSV tresholding with cluttered background

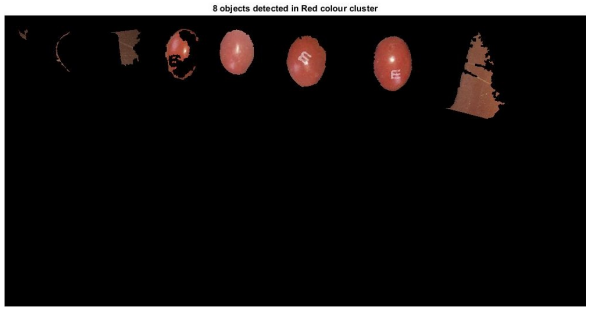


Figure 5.11: Red M&M's extraction with HSV tresholding with cluttered background

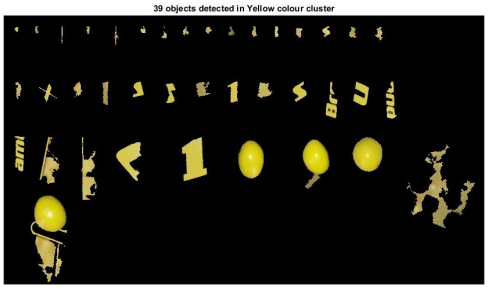


Figure 5.12: Yellow M&M's extraction with HSV tresholding with cluttered background

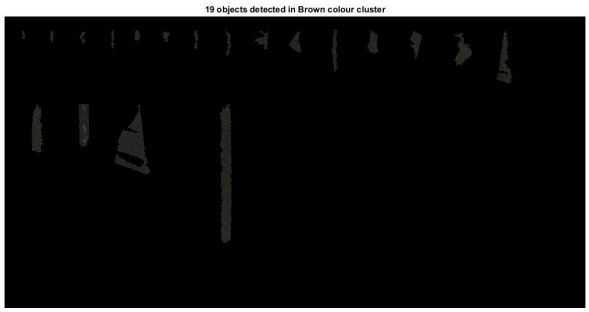


Figure 5.13: Brown M&M's extraction with HSV tresholding with cluttered background

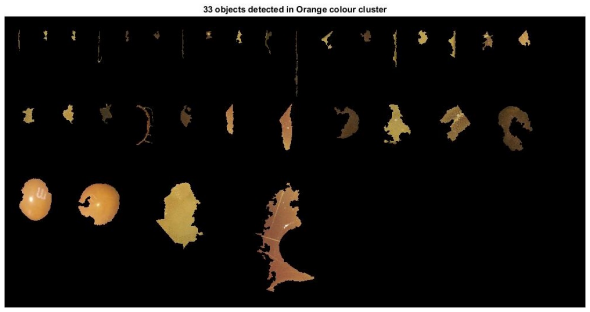


Figure 5.14: Orange M&M's extraction with HSV tresholding with cluttered background

Although the results for the plain background are excellent, this method performs much worse with a cluttered background. This is due to the similarity between the colours of

the background and the objects. It was tried to narrow down the thresholds, but this decreased the quality of extraction of the objects.

5.2 K-means Clustering

The k-means clustering could also be used as clustering adapts automatically. The best results were obtained with 5 clusters: blue, green, yellow, red and orange M&Ms were identified. Dark Brown M&Ms were not correctly identified with this technique.

5.2.1 Plain Background

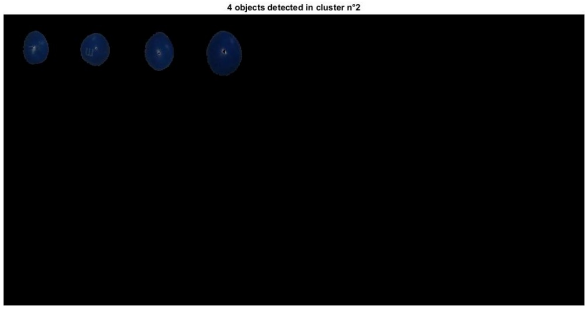


Figure 5.15: Blue M&M's extraction with k-means clustering with plain background

Figure 5.16: Green M&M's extraction with k-means clustering with plain background

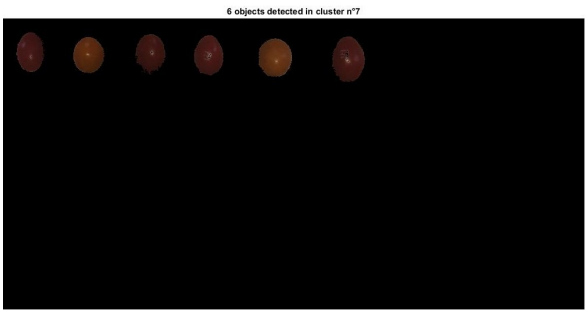


Figure 5.17: Red and orange M&M's extraction with k-means clustering with plain background

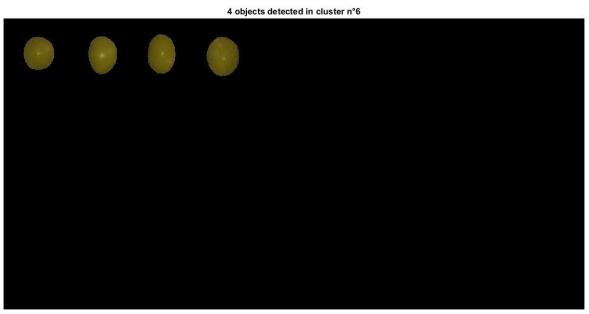


Figure 5.18: Yellow M&M's extraction with k-means clustering with plain background

5.2.2 Cluttered Background

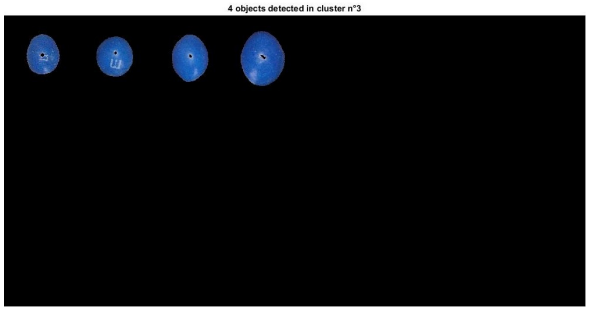


Figure 5.19: Blue M&M's extraction with k-means clustering with cluttered background

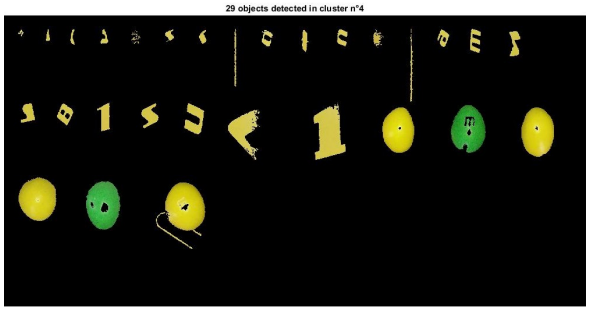


Figure 5.20: Yellow and green M&M's extraction with k-means clustering with cluttered background

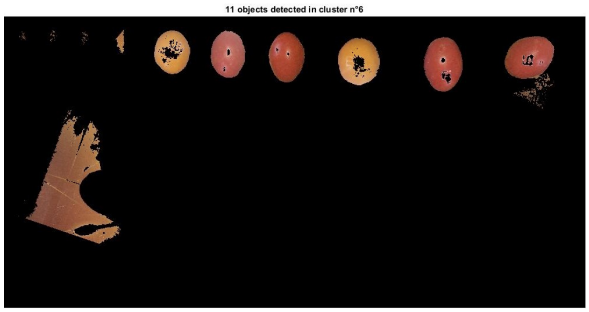


Figure 5.21: Red and orange M&M's extraction with k-means clustering with cluttered background

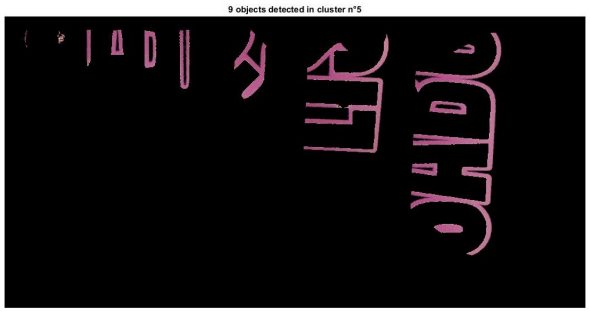


Figure 5.22: Purple M&M's extraction with k-means clustering with cluttered background

6. Conclusion

The difficulty of separating colours lies in the fact that, often, the background and objects are composed of similar colours, making clustering difficult. Thresholding can be quite effective with a plain background but is optimised for an individual image and might not work for another one. On the other hand, K-means algorithm has the advantage of adapting automatically while being less efficient and requiring much more computer power. Further improvements could be achieved by combining filtering on shape or area. Objects smaller than 500 pixels are already filtered out, but a deeper analysis of their sizes could lead to an improved filter. Also, filtering the shape (such as non-linear edges

for Lego bricks) could also be implemented. This requires, however, a certain knowledge of the objects which is not always the case.

Word Count : 1246.

Bibliography

[1] MATLAB. *Documentation* [Online]. Available at : <http://tinyurl.com/hst6n9n>.

[2] STEVE EDDINS AND FARUK. (2010, December 24th) *What color is green? + Comment*. [Online]. Available at : <http://tinyurl.com/znvwtp9>.

[3] MATLAB. *Detecting A Cell Using Image Segmentation* [Online]. Available at : <http://tinyurl.com/hj4hakg>.

[4] MATLAB. *Color-Based Segmentation Using K-Means Clustering* [Online]. Available at : <http://tinyurl.com/h7gpwrb>.

[5] WIKIPEDIA. (2016, November 21). *K-moyennes* [Online]. Last viewed : 27 December 2016. Available at : <http://tinyurl.com/jhug4op>.

[6] WIKIPEDIA. (2016, December 9). *Image segmentation* [Online]. Last viewed : 20 December 2016. Available at : <http://tinyurl.com/hebjc7n>.

[7] IMAGE ANALYST. (2015, December 19). *SimpleColorDetectionByHue()* [Online]. Available at : <http://tinyurl.com/zu3mwuw>.

Appendix

K-means Algorithm : Separation Part

Similarly for the thresholding algorithm, each r, g, b plane from each cluster is processed, binarized, and objects are identified.

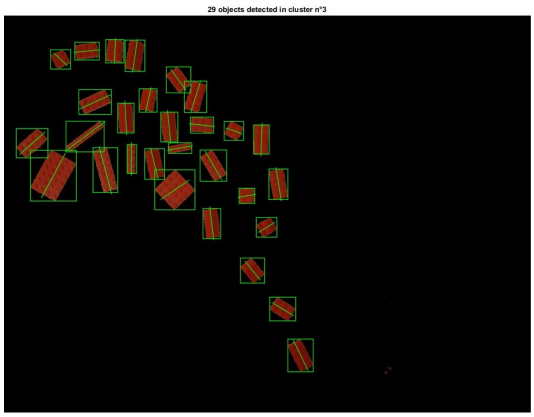


Figure 6.1: Red colour clustering with the k-means algorithm in the Lab Colour Space

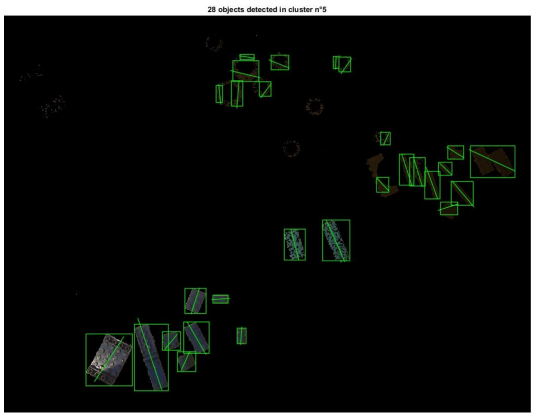


Figure 6.2: Blue and black colour clustering with the k-means algorithm in the Lab Colour Space

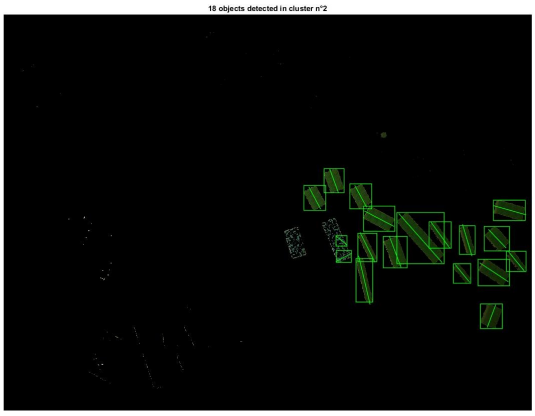


Figure 6.3: Green colour clustering with the k-means algorithm in the Lab Colour Space

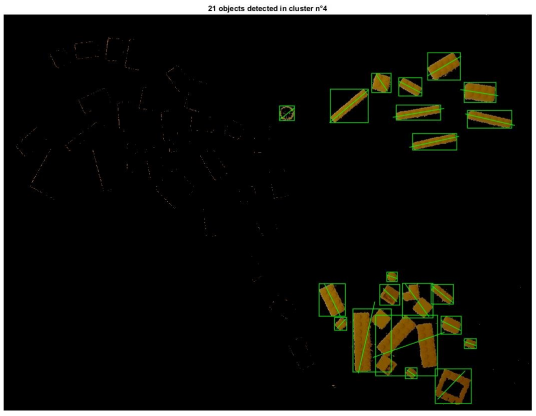


Figure 6.4: Yellow colour clustering with the k-means algorithm in the Lab Colour Space

Table 6.1 shows the number of bricks detected with k-means clustering. The final results are quite similar to manual thresholding : whites are never detected, intense colours as

red, green or blue are always perfectly detected. The main difference is that the dark colour objects are worse detected and isolated.

Colour	Real Total Number	Detected and Isolated	Not Detected	Detected but not Isolated	Other Remarks
Red	29	29	0	-	-
Green	16	16	0	-	-
Dark Blue	8	8	0	-	-
Cyan	2	4	0	-	Two detected on two different clusters
Yellow	24	20	0	2 × 2 together and 1 × 3 together	-
Black	10	9	0	1 × 2 together	One only partially
Wheels	8	8	3	-	Not accurately detected
Gray	3	1	2	-	-
Brown	1	0	1	-	-
Beige	1	1	0	-	-
White	14	0	14	-	-
Total	116 =	96	+ 20	+ 5	- 5

Table 6.1: Error Assessment Table for kmean clustering

Gaussian Noise

Definition of noise, taken from the notes given in class, used in the MATLAB code.
Signal-to-noise ratio :

$$SNR_{dB} = 10\log_{10}\left[\frac{VAR(\mathbf{f})}{RSS(\mathbf{f},\mathbf{g})}\right] \tag{6.1}$$

where $VAR(\mathbf{f}) = E[f^2]$ is the **variance** of the signal, also called average sum of squares, and $RSS(\mathbf{f},\mathbf{g})$ is the **residual sum of squares**, also called mean squared error, which represents the pixel-by-pixel difference in two images $\mathbf{f}$ and $\mathbf{g}$:

$$RSS(\mathbf{f},\mathbf{g}) = \frac{1}{N \times M}\sum_i (f_i - g_i)^2$$

.

How to run code

`Kmeans.m` : enter the name of image to be treated and the number of clusters and run. To add noise to original image, uncomment single commented lines in the "Noise" section (lines 12-17).

`Threshold.m` : enter the name of image to be treated and uncomment the right "col_prop" structure : for Lego-bricks or M&M's. Run. To add noise to original image, uncomment the single commented lines in the "Noise" section (lines 11-16).