Improving color extraction in automotive telltales

Betina Miroslavova Borisova

Visteon Corporation, Sofia, Bulgaria

February 11, 2022

Abstract

During product development, one of the goals is to meet the client's requirements. In the automotive industry especially, everything should be validated strictly. A lot of requirements come from safety regulations, aiming to achieve high product quality. One of the essential things to accomplish that, is to validate the colors of the telltales (or simply referred to as objects in this document). Their colors carry additional information for the user. It might indicate ON/OFF state or that something is not working properly. It is important to find an automated way to confirm that the product is behaving as expected. The purpose of this research is to find an algorithm that can solve this. The input should be image of an object and the produced output - the detected colors and their coverage area. The algorithm can be used with less or without any prior knowledge of the telltales. It should allow for custom configuration when such information is present.

Keywords: *Automation, Automotive, Clustering, Color detection, Color extraction, Computer vision, Grayscale, HSV, K-means, Machine vision, RGB, Telltales, Validation, Vision recognition*

1 Problem Statement

There are a lot of different approaches that can be used to extract the colors of an object. One notable example is to get the mean value of all the relevant pixels. The benefit is that it is quite fast and the only required pre-processing is to separate the object from the background. However, an important downside to this approach is that if there is more than one color, the reported mean value is not going to be correct. This makes it impossible to easily separate similar types of objects as the ones shown in Figure 1.

Figure 1: White and black text telltale

Using the average of the pixels' values, the first example will produce a color that is closer to the darker shade of green and the one on the right is going to be lighter. In perfect conditions it is possible to differentiate them, but in reality, the quality of the pictures is not going to be the same. There might be different variations of brightness that will affect the result. Even if it is assumed that the pictures of both objects will always be taken under the same conditions, it will only allow for a differentiation between the objects themselves. This is not applicable in situations where the knowledge for one of them is missing. An example of this problem is when only one of the images is provided to the algorithm. The reported color will be a variation of the green, but not knowing what to compare it to will create an ambiguity. In that case it will be impossible to say if the text inside is black or white because the reported color might be affected by the overall brightness of the image.

To be able to validate the colors of objects

that fall in such cases, it is needed to gain additional information that will allow for easier object identification. The proposed algorithm trades speed performance and in return provides the data required for more accurate analysis.

2 Algorithm analysis

There are a lot of methods used for color segmentation of images. Usually, different clustering approaches are used for that purpose – for example algorithms like K-means [9], Fuzzy C-means [7] (a comparison between those two can be found here [2]), and their different variations with additional improvements introduced to them, such as Superpixel-Based Fast Fuzzy C-Means [3] and Color Image Quick Fuzzy C-Means [4]. There are research articles that explain and compare the methods in depth [5] and there are different suggestions on how to approach the color segmentation of images [1]. Unlike the algorithms mentioned, the proposed algorithm allows for more customizability. It iterates only once over the whole image and afterwards different thresholds are used to adjust the clusters. The solution in this document in its essence is inspired by the K-means clustering algorithm. With the modifications introduced it is more suitable for the problem area that is targeted.

When analyzing the K-means algorithm, there are a few problems that occur while trying to achieve the required results:

1. Selection of centroids
   The approach here is to use the knowledge that the clustering algorithm is working with colors. This allows for usage of predefined attractors. They are specified as the primary - red, green, blue and the secondary colors - cyan, magenta, and yellow, with the gray color as an addition.

2. The algorithm returns as many colors as expected
   Sometimes this number might not be accurate because it cannot be provided. This requires further reduction of the clusters. However, even if a relatively big number is specified it is still hard to apply the algorithm.

3. The different variations in the colors are easily picked up
   For example, the darker and brighter red are detected separately, and it makes it hard to determine how many centroids are needed to capture the significant ones. In Figure 2 it is illustrated how instead of the black inside the telltale, it turns red because of the clustering. An important characteristic of the algorithm is to treat such deviations in a single color as one cluster.

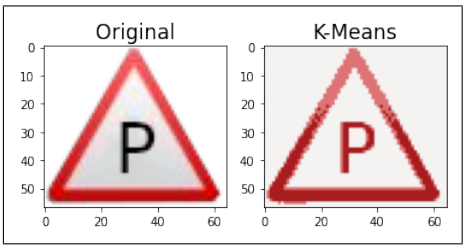

Figure 2: K-means telltale visualization results

3 Algorithm description

3.1 Object extraction

An important goal of the algorithm is to split the telltale from the background. There are two methods of mask generation that complement each other and can be used to produce satisfactory results. It is important to note that in most cases the background is going to be homogeneous, which makes it significantly easier for the extraction of the object.

The first method of mask generation (Figure 3) is simply using a threshold. The parameter is calculated based on the average of the minimum and maximum values of the image when it is grayscale. Everything that is above the threshold is set to the highest possible value and everything that is below it is set to zero. The knowledge of the predominant color of the image, if it is mostly white or black after the applied threshold, is useful for mask correction. If one of the colors represents more than 50 percent of all the pixels, it can be said that it is dominant. If the image is mostly white the chances are that the background is captured as the telltale. This requires an inversion of the mask to have the correct result. Having this additional information – if a bitwise not is applied to the mask, it can be used to further optimize the second method.

Sometimes only this pre-processing is not going to be enough. Some parts of the telltale might be lost due to the used threshold. Therefore, the other way to generate a mask is to use

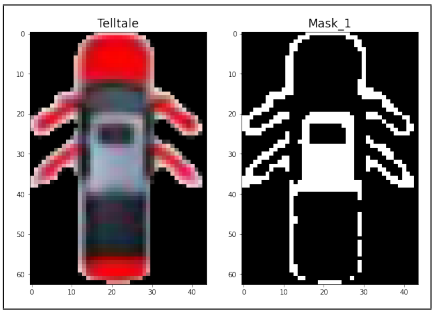

Figure 3: Generated mask - first method

algorithms to determine which are the edges of the telltale and then to extract the contours of it. Filling them will produce the second mask (Figure 4). For better detection of the contours the images can be extended with a few pixels on each side. The information gained from the previous step will determine the color of the added padding. It will be white or black if an inversion is applied to the mask or not.

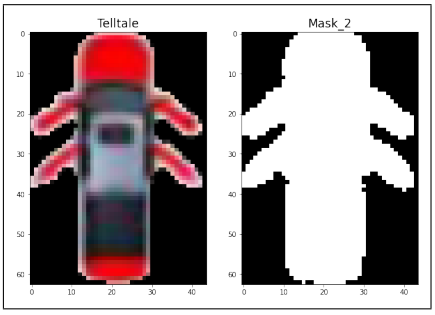

Figure 4: Generated mask - second method

When both masks are combined into one the information gained about the telltale's shape can be used to know which pixels are relevant and which are not. In this case the final mask looks similar to the second one (Figure 4). To incorporate it into the algorithm, the image of the telltale can be converted to RGBA. The alpha channel will be the mask where the white parts of it represent the telltale, and the black parts are the background making it transparent.

3.2 Algorithm

The designed algorithm consists of two main steps - color detection and color reduction.
The purpose of the first one is to detect the colors present in the image by clustering its pixels to preselected centroids. In case of the RGB representation of the telltale, the red, green, blue, cyan, magenta, yellow and grey, are initially chosen as the attractors. Having more centroids can be both beneficial and detrimental. In some cases, the accuracy of the result is increased, but in other cases it leads to more noise being captured.

When it comes to the decision of which pixel will be part of which cluster, it is important to note that the RGB colors are represented as a cube in the 3D color space (Figure 5) with the values red, green, and blue as its axes. This allows for the usage of the Euclidean distance (1). Each pixel is assigned to the cluster with minimum distance between the pixel's values and the cluster's centroid.

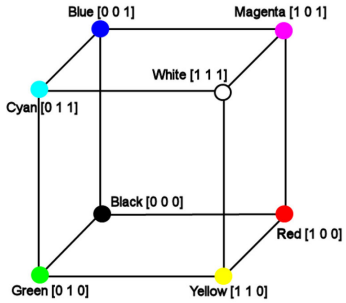

Figure 5: RGB color space - cube [8]

$$\begin{aligned} x &= (x_1, x_2, x_3) \\ y &= (y_1, y_2, y_3) \\ d(x, y) &= \sqrt{\sum_{i=1}^{3} (x_i - y_i)^2} \end{aligned} \tag{1}$$

Where:

- x represents the RGB values of the centroid
- y represents the RGB values of the pixel

In its essence, the first step represents the K-means clustering, but with the difference that the proposed algorithm is doing this for only one iteration and the centers are selected based on the knowledge that the algorithm is working with colors.

It is also possible to use the HSV representation of the image. Two modifications are needed to make the algorithm suitable for this color model. The first one is to convert the RGB centroids to HSV and the second one is to change the way the distance is calculated, the formula from (1) can still be used but after the coordinates are converted as shown in (2). This has to be done because the 3D color space is no longer a cube, the HSV's representation is a cylinder (Figure 6).

$$
\begin{aligned}
x &= \rho \cos \varphi \\
y &= \rho \sin \varphi \\
z &= z
\end{aligned} \tag{2}
$$

Where:

- φ is the Hue in radians
- ρ is the Saturation
- z is the Value

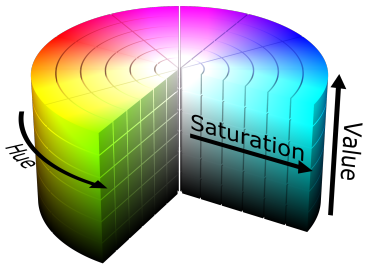

Figure 6: HSV color space - cylinder [6]

The second step of the algorithm is the additional color reduction. The purpose of it is to remove the captured noise and to evaluate the confidence (3) of the detected clusters. If the expected number of colors is explicitly selected, the confidence-based color reduction is skipped.

$$
\begin{aligned}
d_{RGB} &= \max_{x,y \in RGB}(d(x,y)) \\
c(x,y) &= 100 - (100/d_{RGB})d(x,y)
\end{aligned} \tag{3}
$$

Where:

- d_{RGB} is the maximum distance between two points in the 3D color space
- $d(x,y)$ is the distance between the centroid and the pixel's value
- $c(x,y)$ is the confidence

3.3 Settings

The settings that can be applied are:

1. Expected number of colors
2. Coverage area threshold
3. Confidence deviation
4. Minimum confidence gain
5. Parameter *configured* - indicating if the expected number of colors is specified by the user

They can be used for fine-tuning, depending on the situation. If the expected number of colors is specified by the user explicitly then the algorithm is in non-automatic mode (the parameter *configured* is set to True). It performs only the first step and then removes the colors that do not satisfy the coverage area threshold. They are treated as noise. The coverage area is calculated based on the number of pixels that are part of the object and not the whole image. The background is not considered in any of the calculations. It is important to note that the background in this case refers to everything that is not part of a telltale. After the initial filtering, the additional cluster reduction is initiated, until the expected number of colors is reached.

In the case where that number is not explicitly specified (the parameter *configured* is set to False), the algorithm is in automatic mode. It uses the default values of the settings to produce the results in which it is confident the most. The first part in this mode is the same as in the non-automatic mode. The process is then further extended and considers the confidence deviation and the minimum confidence gain parameters. The purpose of it is to reduce, if needed, the detected colors even more. To understand when further reduction is required, the average confidence of all the clusters is calculated. With the allowed deviation subtracted from it, the minimum threshold of confidence is determined. Before finalizing the removal of the cluster that does not meet the requirements one additional check is applied. If after the modification, the minimum confidence gain is not reached then the change is not applied. In that case the algorithm stops. On the other hand, if this threshold is passed, the color is removed, and the algorithm iteratively proceeds until the exit condition is met.

3.4 Algorithm steps

Input:

- Image of the telltale in RGB/HSV mode
- Boolean – if configured or not
- Number of expected colors
- Coverage area threshold

- Confidence deviation
- Minimum confidence gain

Color extraction:

1. For every relevant pixel in the image:
   (1.1) Calculate the distance between it and the centroids
   (1.2) Assign it to the cluster with the minimum distance between their values
   (1.3) Store information about the current state of the cluster - the mean color of all the pixels that are part of it, the coverage area, and the confidence
2. For every cluster:
   (2.1) If the coverage area is above the specified threshold do nothing
   (2.2) If the requirement is not met, remove the cluster, and save the values of the unclassified pixels
3. For every unclassified pixel:
   (3.1) Find the centroid with the least distance to the pixel's values that has coverage area above the specified threshold and assign the pixel to the cluster
   (3.2) Recalculate the mean value, the coverage area, and the confidence of the clusters
4. Every time that the colors exceed the expected number:
   (4.1) Find the color with the minimum confidence and merge the pixels with the one between which the mean colors of the clusters have minimum distance
   (4.2) Recalculate the mean value, the coverage area, and the confidence of the clusters
5. If the number of expected colors is explicitly specified stop the algorithm
6. If the number of expected colors is not explicitly specified:
   (6.1) Calculate the average confidence of all the clusters and subtract the allowed confidence deviation
   (6.2) If the cluster with minimum confidence is below that threshold remove it and assign its pixels to the centroid with the minimum distance between their mean values
   (6.3) Recalculate the mean value, the coverage area and the confidence of the clusters
   (6.4) Find the new average confidence and what is the confidence gain
      6.4.1. If the confidence gain is above the specified threshold apply the modification permanently and go to 6.1
      6.4.2. If the confidence gain is not above the specified threshold revert the color removal and stop the algorithm

Output: The mean values of the clusters and the coverage area.

4 Performance evaluation

Default settings:

- Not configured explicitly
- Number of expected colors - 3
- Coverage area threshold – 1.0%
- Confidence deviation – 5.0%
- Minimum confidence gain – 5.0%

4.1 Performance on templates

Image visualization description:
The results from the algorithm are visualized with several images.
Examples such as Figure 7 consist of four inner pictures:

- Top-left: the original image
- Top-right: the result with applied alpha channel
- Bottom-left: the telltale colored with the initial centroids
- Bottom-right: the telltale colored with the mean values of the clusters

In similar images as Figure 8, the color values output from the algorithm can be seen for the different tests.
The masks are displayed in pictures such as Figure 9. Mask_1 is generated using only thresholding, Mask_2 is created using the telltale's contours and the last image is the final mask that combines both Mask_1 and Mask_2.

4.1.1 Single-colored templates

Single-colored telltales have one predominant color.

Test 1

- Results in fully automated mode with default settings

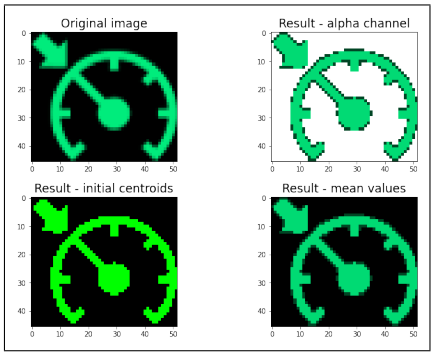

Figure 7: Single-colored telltale visualization results

```
Centroid: (0, 0, 0)
Color: (0, 68, 35)  -> #004423
Confidence: 82.49
Coverage area: 23.4

Centroid: (0, 255, 0)
Color: (0, 221, 116)  -> #00dd74
Confidence: 72.52
Coverage area: 73.84

Centroid: (127, 127, 127)
Color: (0, 119, 68)  -> #007744
Confidence: 68.33
Coverage area: 2.75
```

Figure 8: Single-colored telltale results

As shown in Figure 7 and Figure 8 the present colors in the telltale are correctly detected, including the outlier in darker green. That is the expected output. In this case using the previous algorithm that returns only the mean color would still produce accurate results. It is important that the proposed algorithm improves the functionality and it should have similar performance on examples where the previous one is already achieving satisfactory results.

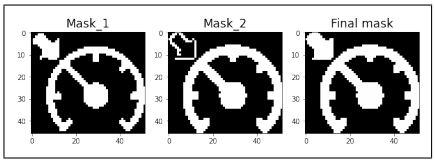

Figure 9: Generated masks

In Figure 9 it is captured how the method that uses the contours sometimes can fail and the result can be partially correct. In such cases the first mask that is generated completes the object and the algorithm produces accurate results. There are cases where the opposite happens – the second one amplifies the first.

- Results in manual mode – number of expected colors set to 1

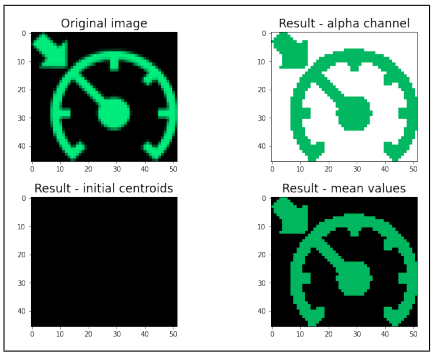

Figure 10: Configured single-colored telltale visualization results

```
Centroid: (0, 0, 0)
Color: (0, 183, 96)  -> #00b760
Confidence: 72.35
Coverage area: 100.0
```

Figure 11: Configured single-colored telltale results

Usually, to the naked eye, on the first image the border of the telltale is not seen. Because of that, the number of colors in that case can be set to one. As seen in Figure 10 and Figure 11 the output of the algorithm is the correct result.

Test 2

- Results in fully automated mode with default settings

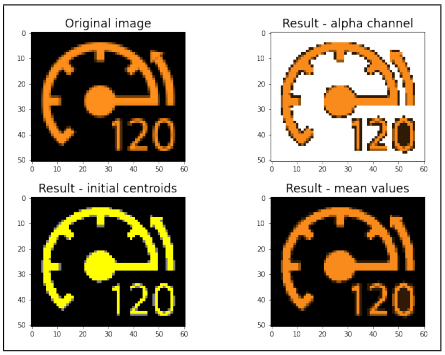

Figure 12: Single-colored telltale visualization results

```
Centroid: (0, 0, 0)
Color: (46, 26, 5)  -> #2e1a05
Confidence: 87.83
Coverage area: 26.94

Centroid: (127, 127, 127)
Color: (152, 86, 17)  -> #985611
Confidence: 72.86
Coverage area: 12.82

Centroid: (255, 255, 0)
Color: (248, 139, 27)  -> #f88b1b
Confidence: 98.99
Coverage area: 60.24
```

Figure 13: Single-colored telltale results

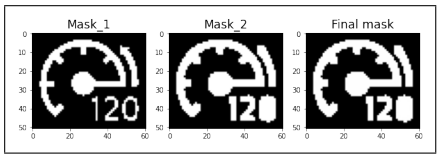

Figure 14: Generated masks

In this case, the algorithm not only detects the border, but it also captures the background inside the zero (Figure 12). Analyzing the used mask for the telltale in Figure 14, the one generated by the contours negatively impacts the result. The first image is the correct representation of the object, but the second one is with captured noise. However, in most cases using both masks, results in better detections of the colors rather than using only one.

- Results in manual mode – number of expected colors set to 1

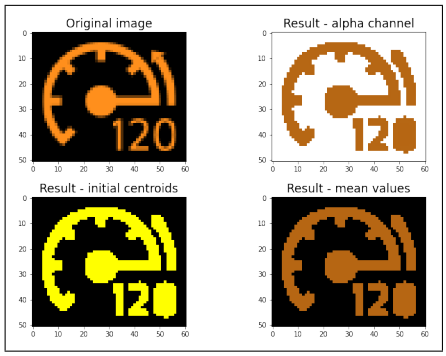

Figure 15: Configured single-colored telltale visualization results

```
Centroid: (255, 255, 0)
Color: (182, 102, 19)  -> #b66613
Confidence: 86.95
Coverage area: 100.0
```

Figure 16: Configured single-colored telltale results

Manually setting the number of expected colors produced the correct result (Figure 15 and Figure 16) although the color appears darker because of the noise captured by the mask.

Test 3

- Results in fully automated mode with default settings

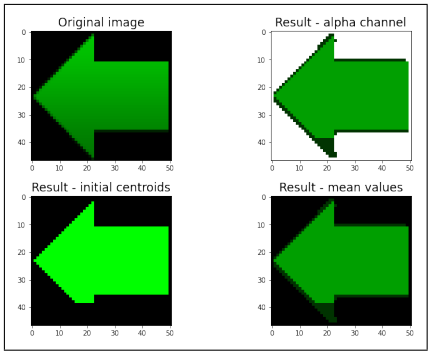

Figure 17: Single-colored gradient telltale visualization results

As shown in Figure 17 and Figure 18, in cases where there is a gradient, the algorithm also works well. The darker parts of the telltale are captured without any explicit information from the user.

```
Centroid: (0, 0, 0)
Color: (0, 51, 0)   -> #003300
Confidence: 88.42
Coverage area: 11.51

Centroid: (0, 255, 0)
Color: (0, 159, 0)   -> #009f00
Confidence: 78.47
Coverage area: 88.49
```

Figure 18: Single-colored gradient telltale results

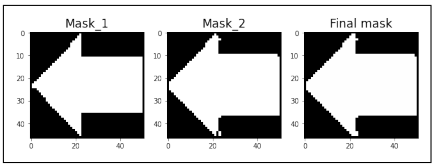

Figure 19: Generated masks

The masks also correctly represent the shape of the object (Figure 19).

- Results in manual mode – number of expected colors set to 1

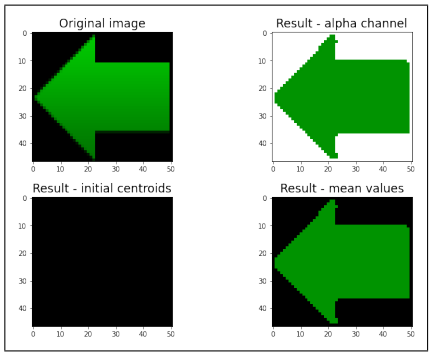

Figure 20: Configured single-colored gradient telltale visualization results

```
Centroid: (0, 0, 0)
Color: (0, 147, 0)   -> #009300
Confidence: 78.26
Coverage area: 100.0
```

Figure 21: Configured single-colored gradient telltale results

As seen in Figure 20 and Figure 21, manually selecting how many colors are expected to be in the telltale further improves the result.

4.1.2 Two-colored templates

Two-colored telltales have two predominant colors.

Test 1

- Results in fully automated mode with default settings

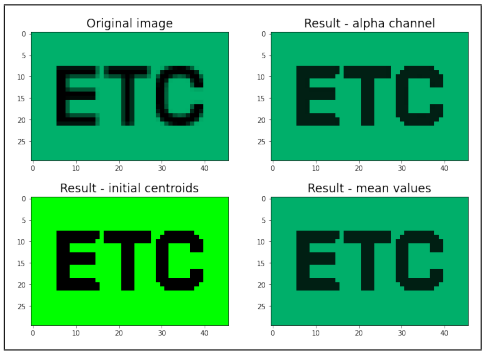

Figure 22: Two-colored telltale visualization results

```
Centroid: (0, 255, 0)
Color: (0, 175, 106)   -> #00af6a
Confidence: 69.85
Coverage area: 81.45

Centroid: (0, 0, 0)
Color: (0, 31, 19)   -> #001f13
Confidence: 91.68
Coverage area: 18.55
```

Figure 23: Two-colored telltale results

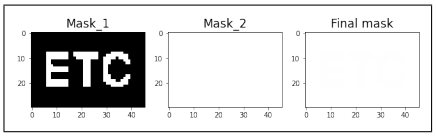

Figure 24: Generated masks

Coming back to the case described in the "Problem statement" section in Figure 22 can be seen how the algorithm successfully solves it. In Figure 23 the two colors that are detected are green and black allowing for easy classification of the telltale.

In Figure 24 it is illustrated how using only the first mask is going to produce incorrect results. The telltale consists of background and text inside it, and both should be considered when detecting the colors.

- Results in manual mode – number of expected colors set to 2

The results in the manual mode do not differ from the automatic mode in this case (Figure 22 and Figure 23).

Test 2

- Results in fully automated mode with default settings

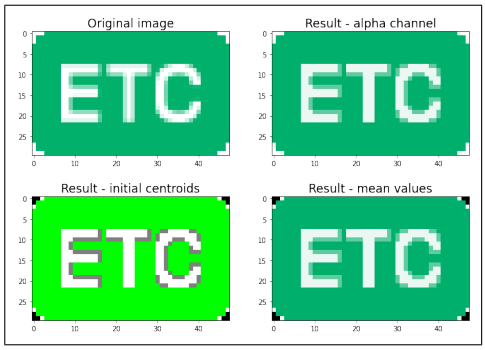

Figure 25: Two-colored telltale visualization results

```
Centroid: (255, 255, 255)
Color: (234, 247, 242)  -> #eaf7f2
Confidence: 94.24
Coverage area: 14.36

Centroid: (0, 255, 0)
Color: (0, 175, 107)  -> #00af6b
Confidence: 69.86
Coverage area: 80.04

Centroid: (127, 127, 127)
Color: (96, 202, 160)  -> #60caa0
Confidence: 80.03
Coverage area: 5.6
```

Figure 26: Two-colored telltale results

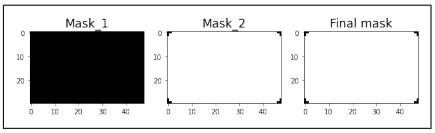

Figure 27: Generated masks

For the template of the telltale with white text inside, the algorithm performs similarly, and the result is correct (Figure 25, Figure 26 and Figure 27).

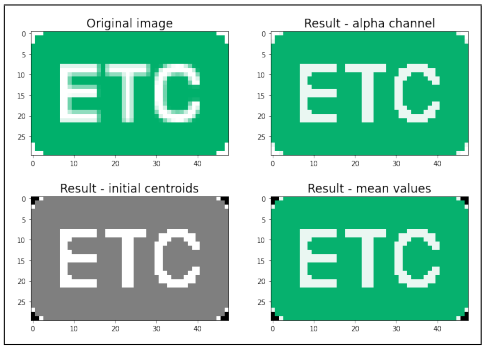

Figure 28: Configured two-colored telltale visualization results

```
Centroid: (255, 255, 255)
Color: (234, 247, 242)  -> #eaf7f2
Confidence: 94.24
Coverage area: 14.36

Centroid: (127, 127, 127)
Color: (6, 177, 110)  -> #06b16e
Confidence: 76.01
Coverage area: 85.64
```

Figure 29: Configured two-colored telltale results

- Results in manual mode – number of expected colors set to 2

Manually selecting the number of expected colors to two resulted in merging of the two green colors. The output is cleaner, and the coverage area is also accurate (Figure 28 and Figure 29).

4.1.3 Multi-colored templates

Multi-colored telltales have more than two predominant colors.

Test 1

- Results in fully automated mode with default settings

Because of the white background of the image an inversion is applied to Mask_1 and produces the result that is seen in Figure 32. Also, it is visible that the second mask completes the first one in this case. The algorithm produces correct results even with more colors.

- Results in manual mode – number of expected colors set to 3

The results in the manual mode do not differ from the automatic mode in this case (Figure 31 and Figure 32).

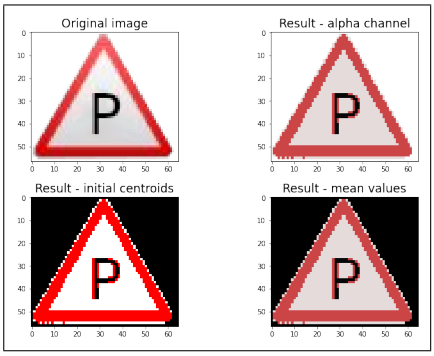

Figure 30: Multi-colored telltale visualization results

```
Centroid: (255, 255, 255)
Color: (229, 219, 219)   -> #e5dbdb
Confidence: 87.22
Coverage area: 54.64

Centroid: (0, 0, 0)
Color: (6, 6, 6)   -> #060606
Confidence: 97.48
Coverage area: 3.16

Centroid: (255, 0, 0)
Color: (203, 68, 70)   -> #cb4446
Confidence: 89.57
Coverage area: 42.2
```

Figure 31: Multi-colored telltale results

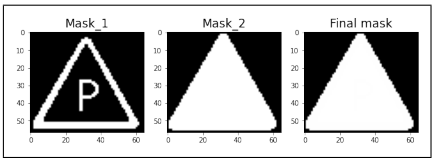

Figure 32: Generated masks

Test 2

- Results in fully automated mode with default settings

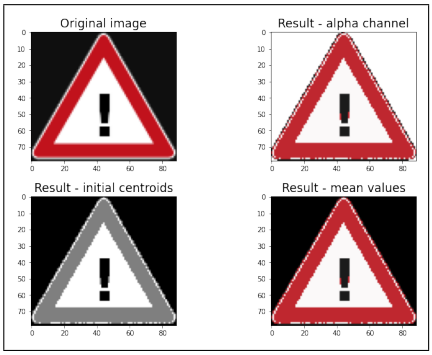

Figure 33: Multi-colored telltale visualization results

```
Centroid: (0, 0, 0)
Color: (27, 27, 27)   -> #1b1b1b
Confidence: 89.14
Coverage area: 7.11

Centroid: (255, 255, 255)
Color: (251, 249, 249)   -> #fbf9f9
Confidence: 98.01
Coverage area: 40.1

Centroid: (127, 127, 127)
Color: (191, 39, 47)   -> #bf272f
Confidence: 75.41
Coverage area: 52.79
```

Figure 34: Multi-colored telltale results

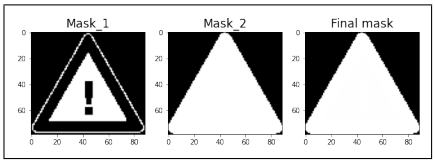

Figure 35: Generated masks

For this telltale, the algorithm detects all the colors as expected (Figure 33 and Figure 34). Because of the shape the masks (Figure 35) are easily generated with no mistakes.

- Results in manual mode – number of expected colors set to 3

The results in the manual mode do not differ from the automatic mode in this case (Figure 33 and Figure 34).

4.2 Performance on real telltales

4.2.1 Single-colored telltales

Test 1

- Results in fully automated mode with default settings

This is another example of how the algorithm picks up the background as part of the object because of the specific shape (Figure 38) of the telltale. The orange is correctly extracted (Figure 36 and Figure 37), but the black color is detected as well.

- Results in manual mode – number of expected colors set to 1

When the expected color is set to one, the detected orange gets darker because all the clusters are merged including the black background (Figure 39 and Figure 40).

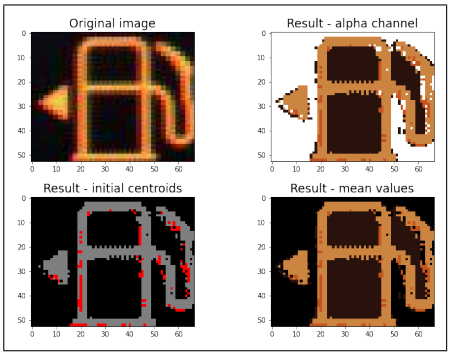

Figure 36: Single-colored telltale visualization results

```
Centroid: (0, 0, 0)
Color: (40, 18, 11)  -> #28120b
Confidence: 89.62
Coverage area: 50.27

Centroid: (255, 0, 0)
Color: (197, 83, 31)  -> #c5531f
Confidence: 75.98
Coverage area: 5.39

Centroid: (127, 127, 127)
Color: (204, 132, 65)  -> #cc8441
Confidence: 97.1
Coverage area: 44.34
```

Figure 37: Single-colored telltale results

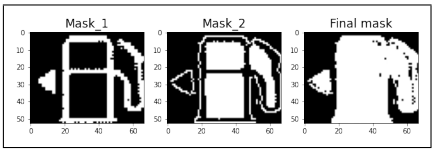

Figure 38: Generated masks

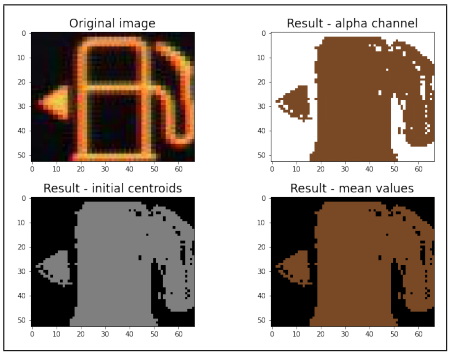

Figure 39: Configured single-colored telltale visualization results

Test 2

- Results in fully automated mode with default settings

```
Centroid: (127, 127, 127)
Color: (121, 72, 36)  -> #794824
Confidence: 77.0
Coverage area: 100.0
```

Figure 40: Configured single-colored telltale results

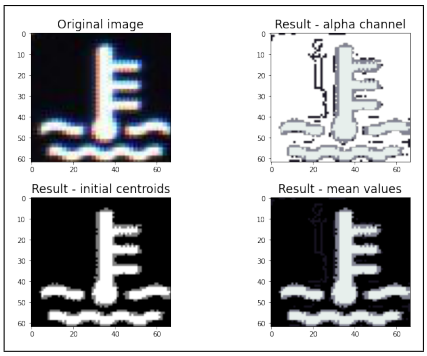

Figure 41: Single-colored telltale visualization results

```
Centroid: (0, 0, 0)
Color: (25, 21, 36)  -> #191524
Confidence: 88.83
Coverage area: 14.55

Centroid: (127, 127, 127)
Color: (130, 132, 147)  -> #828493
Confidence: 95.18
Coverage area: 29.52

Centroid: (255, 255, 255)
Color: (231, 239, 236)  -> #e7efec
Confidence: 92.37
Coverage area: 55.93
```

Figure 42: Single-colored telltale results

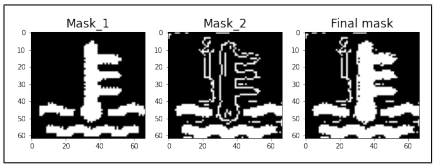

Figure 43: Generated masks

- Results in manual mode – number of expected colors set to 1

Comparing the results from Figure 41, Figure 42 and Figure 44, Figure 45, the fully automatic mode in this case is behaving better than manually selecting the expected number of colors. The latter results, because of the reduction, are more grayish rather than white as in the first output. The mask (Figure 43) detects noise as

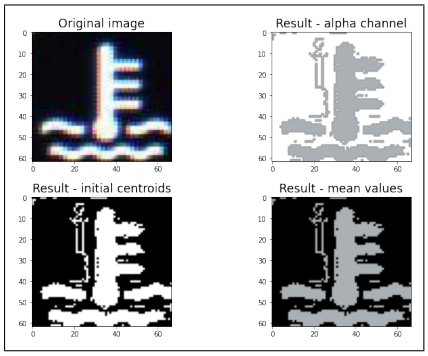

Figure 44: Configured single-colored telltale visualization results

```
Centroid: (255, 255, 255)
Color: (171, 176, 181)  -> #abb0b5
Confidence: 76.7
Coverage area: 100.0
```

Figure 45: Configured single-colored telltale results

part of the telltale's shape, but it is low and is not affecting the output a lot.

4.2.2 Two-colored telltales

Test 1

- Results in fully automated mode with default settings

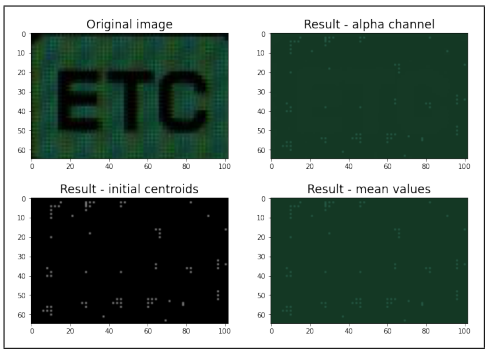

Figure 46: Two-colored telltale visualization results

For this telltale, because of the noise the results are worse (Figure 46 and Figure 47). The final mask (Figure 48) is correctly representing the object, but the black text inside it is lost. Different optimizations can be used, for example brightening the image to improve the color extraction.

- Results in manual mode – number of expected colors set to 2

```
Centroid: (0, 0, 0)
Color: (20, 56, 36)  -> #143824
Confidence: 84.17
Coverage area: 98.93

Centroid: (127, 127, 127)
Color: (38, 89, 69)  -> #265945
Confidence: 74.75
Coverage area: 1.07
```

Figure 47: Two-colored telltale results

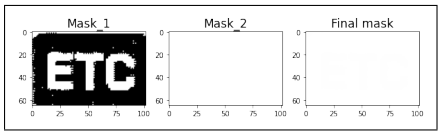

Figure 48: Generated masks

The results in the manual mode do not differ from the automatic mode in this case (Figure 46 and Figure 47).

Test 2

- Results in fully automated mode with default settings

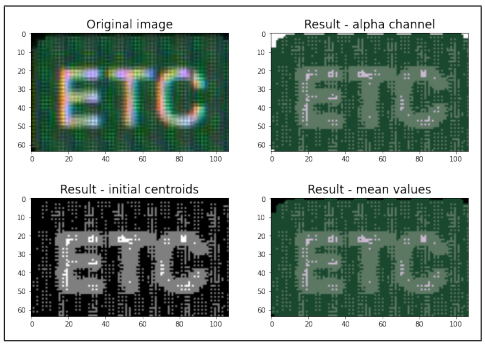

Figure 49: Two-colored telltale visualization results

- Results in manual mode – number of expected colors set to 2

This is a similar case as with the previous telltale. The colors because of the noise are distorting the final output (Figure 49 and Figure 50). When the expected number of colors is specified, the text is not lost, but it is no longer white – it appears greenish (Figure 52 and Figure 53). The mask (Figure 51) is accurate so only additional optimization should be applied, like the suggested in the previous example, to improve the result.

```
Centroid: (0, 0, 0)
Color: (26, 72, 46)  -> #1a482e
Confidence: 79.62
Coverage area: 59.4

Centroid: (127, 127, 127)
Color: (92, 120, 99)  -> #5c7863
Confidence: 89.98
Coverage area: 38.65

Centroid: (255, 255, 255)
Color: (203, 185, 210)  -> #cbb9d2
Confidence: 77.93
Coverage area: 1.96
```

Figure 50: Two-colored telltale results

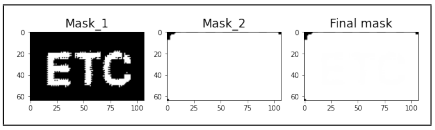

Figure 51: Generated masks

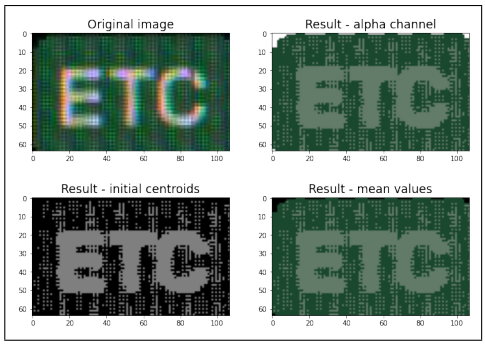

Figure 52: Configured two-colored telltale visualization results

```
Centroid: (0, 0, 0)
Color: (26, 72, 46)  -> #1a482e
Confidence: 79.62
Coverage area: 59.4

Centroid: (127, 127, 127)
Color: (98, 123, 105)  -> #627b69
Confidence: 97.96
Coverage area: 40.6
```

Figure 53: Configured two-colored telltale results

4.2.3 Multi-colored telltales

Test 1

- Results in fully automated mode with default settings

The algorithm also performs good on more

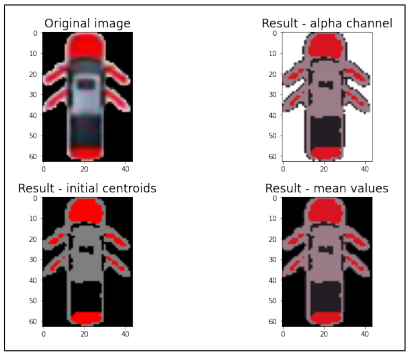

Figure 54: Multi-colored telltale visualization results

```
Centroid: (0, 0, 0)
Color: (33, 29, 35)  -> #211d23
Confidence: 87.17
Coverage area: 28.25

Centroid: (255, 0, 0)
Color: (217, 20, 33)  -> #d91421
Confidence: 87.68
Coverage area: 18.71

Centroid: (127, 127, 127)
Color: (154, 124, 135)  -> #9a7c87
Confidence: 96.7
Coverage area: 53.04
```

Figure 55: Multi-colored telltale results

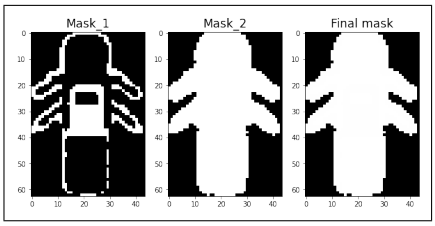

Figure 56: Generated masks

complex objects such as the one in Figure 54. The purple color that is detected is caused because of a lot of different variations in the values of the pixels where they are predominantly pink. In this case the second method used for mask generation is performing better than the first one (Figure 56).

- Results in manual mode – number of expected colors set to 3

The results in the manual mode do not differ from the automatic mode in this case Figure 54 and Figure 55.

5 Optimizations

5.1 Resizing

The goal of this modification is to reduce the time it takes to produce the output. By resizing the input image, the algorithm will have to loop over less pixels which will result in speedup of the process. This method impacts the detected values of the colors and their coverage area, but the variation in them is not drastic. In Figure 57 it is illustrated how that modification removes noise that is detected in one of the telltales' tests (Figure 46) As seen in Figure 47, the coverage area of the second color that is detected barely passes the threshold while in the case when the size of the image is reduced by 50% the noise is missing (Figure 58).

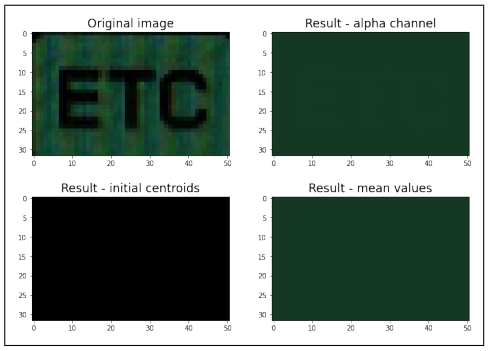

Figure 57: Two-colored resized telltale visualization results

```
Centroid: (0, 0, 0)
Color: (20, 56, 36)  -> #143824
Confidence: 84.09
Coverage area: 100.0
```

Figure 58: Two-colored resized telltale results

5.2 Blurring

Because of the noise that can be captured with a camera, the algorithm can detect a lot of meaningless colors. The purpose of the blurring is to solve that issue. Reducing the noise in the image is expected to produce more accurate output. Different methods can be used to blur the image. In some cases, this modification alone will not be enough, but it can be used in conjunction with other optimizations, for example brightness adjustment.

5.3 Brightness and contrast adjustment

As seen in some cases, for example Figure 46, the image brightness is low. This negatively affects the colors extraction. It only detects the noise and the black text inside it is merged with the background. One way to solve this problem is to increase the brightness. However, a more general solution is required. The problem will not always be caused because of the low brightness, it might be caused also because it is too high. The best solution is to use an automatic way to adjust the illumination.

In Figure 59, it can be seen how with the applied modification, the colors (Figure 60) are correctly extracted from the image.

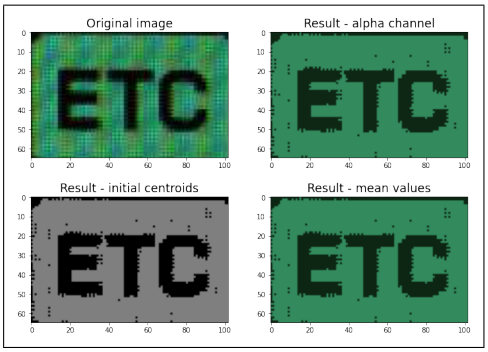

Figure 59: Two-colored adjusted telltale visualization results

```
Centroid: (0, 0, 0)
Color: (13, 38, 20)  -> #0d2614
Confidence: 89.72
Coverage area: 25.44

Centroid: (127, 127, 127)
Color: (51, 139, 93)  -> #338b5d
Confidence: 81.13
Coverage area: 74.56
```

Figure 60: Two-colored adjusted telltale results

6 Conclusion

In this document the suggested algorithm for color extraction was described and analyzed. Its performance showed that it can successfully provide the additional information needed to have a more accurate analysis of the telltales. Further in-depth research can be conducted to find ways to optimize the execution time of the algorithm. In comparison to the method that outputs the mean color value, it is trading speed performance

in exchange for more detailed information. That additional knowledge allows for easier and more accurate analysis.

7 Acknowledgements

Special thanks to Filip Borisov[1], Martin Vachovski[1] and Stiliyan Georgiev[1] for their contributions and support. The discussions and the shared ideas played a huge part in the success of this research.

[1] *Visteon Corporation, Sofia, Bulgaria*

References

[1] Zhensong Chen et al. "Image Segmentation via Improving Clustering Algorithms with Density and Distance". In: *Procedia Computer Science* 55 (2015). 3rd International Conference on Information Technology and Quantitative Management, ITQM 2015, pp. 1015–1022. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2015.07.096. URL: https://www.sciencedirect.com/science/article/pii/S1877050915015719.

[2] Soumi Ghosh and Sanjay Kumar. "Comparative Analysis of K-Means and Fuzzy C-Means Algorithms". In: *International Journal of Advanced Computer Science and Applications* 4 (May 2013). DOI: 10.14569/IJACSA.2013.040406.

[3] Tao Lei et al. "Superpixel-Based Fast Fuzzy C-Means Clustering for Color Image Segmentation". In: *IEEE Transactions on Fuzzy Systems* 27.9 (2019), pp. 1753–1766. DOI: 10.1109/TFUZZ.2018.2889018.

[4] Zun-yang Liu et al. "Background dominant colors extraction method based on color image quick fuzzy c-means clustering algorithm". In: *Defence Technology* 17.5 (2021), pp. 1782–1790. ISSN: 2214-9147. DOI: https://doi.org/10.1016/j.dt.2020.10.002. URL: https://www.sciencedirect.com/science/article/pii/S2214914720304608.

[5] George Seif. *The 5 Clustering Algorithms Data Scientists Need to Know*. Feb. 2018. URL: https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68.

[6] SharkD. *The HSV color model mapped to a cylinder*. [Online; accessed 29-January-2022]. 2015. URL: https://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder.png.

[7] R Suganya and R Shanthi. "Fuzzy c-means algorithm-a review". In: *International Journal of Scientific and Research Publications* 2.11 (2012), p. 1.

[8] Stephen Westland and Vien Cheung. "RGB Systems". In: Jan. 2015, pp. 1–6. DOI: 10.1007/978-3-642-35947-7_12-2.

[9] Soner Yıldırım. *K-Means Clustering - Explained*. Detailed theoretical explanation and scikit-learn implementation. Mar. 2020. URL: https://towardsdatascience.com/k-means-clustering-explained-4528df86a120.