Automated Resistor Classification

Group Thesis

Pascal Niklaus, Gian Ulli

pniklaus@student.ethz.ch, ug@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Tobias Langner, Jochen Seidel
Prof. Dr. Roger Wattenhofer

January 16, 2015

Abstract

In this group thesis, we develop an algorithm that allows the automated classification of resistors using image recognition. The algorithm is implemented in an Android app and therefore usable on every Android smartphone.

The problem is split into parts and the result is obtained step by step beginning with the localization of the resistor in the captured image and cropping the image such that only the resistor's body remains. In the following analysis, the color rings are separated from the background and finally assigned a specific color. In the end, it is possible to determine the resistance of the resistor.

Our evaluation of the algorithm shows that the resistor localization works with nearly every image and also under different conditions. The detection of the color rings on the body of the resistor also performs well on most of the images. However, our approach to the color assignment turned out to be unreliable. The task is quite difficult because every camera has different saturation and brightness settings. Also, external conditions like shadows and the background have a significant impact on the colors in an image.

Contents

CHAPTER 1

Introduction

1.1 Motivation

While working in the lab, one often faces the problem of having a bunch of different resistors which might be used in a circuit. Unfortunately, these parts are all too often neither sorted nor labeled according to their values. However, the resistance of a resistor is encoded by a color combination. The combination either consists of four or five color rings, which allows the determination of the resistor's resistance. The number of possible colors is restricted and every resistor has in theory a unique color ring assignment. Therefore, we thought that this problem could be solved automatically using image recognition, which is more comfortable than using an ohmmeter or looking up the value in a table. In particular, it would be useful to have an app which does all that work for you. Since we are interested in image recognition and smartphone app development, this project is a good combination of both topics.

This group project focuses on the development of an algorithm that can be used for the automated classification of resistors. The algorithm systematically analyzes an image, searches the resistors and then calculates the resistances according to their color code. Additionally, this algorithm is implemented in an Android app.

In addition to the improvement of comfort, this app can also be useful for people who have difficulties to identify the small color rings or who suffer from color blindness. Another advantage is that no additional hardware is needed, since one can just measure resistors on the go with a smartphone.

1.2 Related Work

There exist several previous approaches to this topic. Most of those implementations are experimental. A notable project is described in [1] and was primarily thought to be an Android app. Later on, it was optimized for a specific webcam with manual focus and an additional light source with code running on a regular computer. The main reason for the platform change was the poor focus of the smartphone camera at low distances, which made it nearly impossible to detect any color rings. The algorithm searches the resistor as a line between its leads. From there, it locates the rings perpendicular to the alignment of the resistor's body and finally determines their colors. Due to the fact that the camera is fixed and an additional light source is provided, the algorithm could be calibrated accurately. Therefore, it was possible to obtain reliable values.

Another realization can be found in [2]. This app was developed for iOS, which has the advantage that there exist fewer devices and cameras, which simplifies the development. Further restrictions are made about the number of color rings and the orientation of the resistor. Namely, the algorithm requires the exact positioning of the resistor in the captured image, realized by a tiny box seen on the camera preview window. The user has to assure that the resistor is inside this box. Moreover, only resistors with four rings are allowed. As the hardware is basically restricted to one device, the colors can be determined pretty accurately by having calibrated values for each possible color. Due to incompatibilities with a new iOS version, further development was abandoned.

Chapter 2

Preliminaries

2.1 Resistor Decoding

As mentioned before, the resistance of a resistor is encoded by a color combination. As there exists no standard that defines the direction the color combination has to be read, most combinations have two corresponding resistances. Our algorithm solves this inconsistency by returning two possible values if neither of those can be rejected due to further restrictions explained below.

Color	Ring 1	Ring 2	Ring 3	Ring 4	Ring 5
	Ring 1	*Ring 2*		*Ring 3*	*Ring 4*
black		0	0	10^0	
brown	1	1	1	10^1	1%
red	2	2	2	10^2	2%
orange	3	3	3	10^3	
yellow	4	4	4	10^4	
green	5	5	5	10^5	0.5%
blue	6	6	6	10^6	0.25%
violet	7	7	7	10^7	0.1%
gray	8	8	8		0.05%
white	9	9	9		
gold				10^{-1}	5%
silver				10^{-2}	10%

Table 2.1: The resistance of a five ring resistor or a *four ring resistor* is based on the color rings.

Within this thesis, we define a resistor as a set R_n of color rings where n denotes the total amount of rings (either four or five) and c_i is the i-th color ring of the resistor.

$$R_n = \{c_1,\ c_2,\ \ldots,\ c_n\}, \qquad n = 4, 5 \tag{2.1}$$

Furthermore, $\mathrm{val}(c_i)$ denotes the value and $\mathrm{tol}(c_i)$ the tolerance of the color belonging to color ring i according to Table 2.1. The corresponding resistance $\mathrm{res}(R_n)$ can then be calculated as follows:

$$\mathrm{res}(R_4) = \big(10 \cdot \mathrm{val}(c_1) + \mathrm{val}(c_2)\big) \cdot 10^{\mathrm{val}(c_3)} \pm \mathrm{tol}(c_4) \tag{2.2}$$

$$\mathrm{res}(R_5) = \big(100 \cdot \mathrm{val}(c_1) + 10 \cdot \mathrm{val}(c_2) + \mathrm{val}(c_3)\big) \cdot 10^{\mathrm{val}(c_4)} \pm \mathrm{tol}(c_5) \tag{2.3}$$

The full decoding scheme can be seen in Table 2.1. A blank cell means that this specific color is not possible for that ring. In consequence, there are some color combinations that yield an invalid resistance. If detected, the algorithm can reject such a combination.

2.2 Lab Color Space

A normal image captured with a camera is stored in the RGB color space. There exist several alternative color spaces, like the Lab color space that has some advantages especially useful for our purposes. The Lab space consists of three channels. L measures the lightness, A the amount of green or red and B the amount of blue or yellow. Since green and red as well as blue and yellow are color-opponents, they cannot occur at the same time and each pair can therefore be described by one coordinate. In [3], one can find a comprehensive overview of the Lab space.

One advantage of Lab is that all information about color is stored in only two channels. For us, this is helpful because the color is an important criteria for the way we localize the resistor in the image. The biggest benefit, however, is that the measured color differences in the Lab color space correspond to the ones perceived by our eyes. This helps us to find the color changes that the human eye would also notice. For example, we don't weight aberrations in the background, like variations of blue, that much but rather color changes from blue to a different color like brown. To calculate color distances, the CIE ΔE^*_{94} color difference formula is used (see [3] for further information). This function compares a reference color $(L_1,\ a_1,\ b_1)$ to a color $(L_2,\ a_2,\ b_2)$. The metric is calculated as follows:

$$\Delta E^*_{94} = \sqrt{\left(\frac{\Delta L}{k_L \cdot S_L}\right)^2 + \left(\frac{\Delta C_{ab}}{k_C \cdot S_C}\right)^2 + \left(\frac{\Delta H_{ab}}{k_H \cdot S_H}\right)^2} \tag{2.4}$$

where

$$\Delta L = L_1 - L_2$$

$$\Delta C_{ab} = \sqrt{a_1^2 + b_1^2} - \sqrt{a_2^2 + b_2^2}$$

$$\Delta H_{ab} = \sqrt{(a_1 - a_2)^2 + (b_1 - b_2)^2 - \Delta C_{ab}^2}$$

$$S_L = 1, \quad S_C = 1 + 0.045 \cdot \sqrt{a_1^2 + b_1^2}, \quad S_H = 1 + 0.015 \cdot \sqrt{a_1^2 + b_1^2}$$

$$k_L = 1, \quad k_C = 1, \quad k_H = 1$$

It is important to note that this metric is not commutative, since some weighting factors only depend on the reference color. This means that the reference color is taken into account stronger than the color it is compared to.

2.3 Software and Hardware for the Implementation

The algorithm is implemented in Java for the Android platform (API version ≥ 14). For the image processing, we use OpenCV 2.4.9 [4]. The Android app was tested on three devices: HTC One, Samsung S3 and LG Google Nexus 5.

Chapter 3

Problems and Methods

In this chapter, we want to explain our resistor detection algorithm. The algorithm takes an image of one or multiple resistors as input and returns the value and a cropped picture of each resistor it found. The resistors should be placed on a uniform gray background. The algorithm consists of three consecutive stages. The first stage scales the captured image to a predefined width of 1920 pixels and locates the resistors in it. Afterwards, the algorithm cuts out each resistor properly and rotates the image, if necessary. The second stage locates the color rings on the resistor and returns a list of color values, one for each ring. The third and final stage is the color classification. For all rings, the closest color from the color code explained above has to be identified. Having determined the color of each ring, it is now possible to calculate the corresponding resistance.

Due to technological limits, it is not always possible to detect the tolerance ring, especially when it is golden. The only difference between a golden ring and a beige background is the glossy surface of the ring, which often cannot be seen in an image of the resistor. Thus, it is necessary to decide whether a four ring or a five ring resistor should be detected before the algorithm can be executed.

3.1 Resistor Localization

The first stage of the algorithm is the localization of resistors in an image. This problem is solved using a line detection algorithm and statistical analysis of the image.

3.1.1 Noise Filtering

To improve the results from the line detection, several noise filters get applied to the image obtained from the camera. First, a median and a Gaussian blur are used. These filters ensure that most camera artifacts are removed and noise is reduced considerably. Then, the morphological operation *Opening* is applied, which improves the separation of the resistors against the neutral background.

Detailed information about the three filters can be found in [5, pp. 109–121]. The resulting image after all three filters will be called *simplified image*. Figure 3.1 shows the effect of the filters on an image with a single resistor.

In our implementation, we use an aperture size of 3 for both blur filters and a kernel size of 5 for the morphological *Opening*.

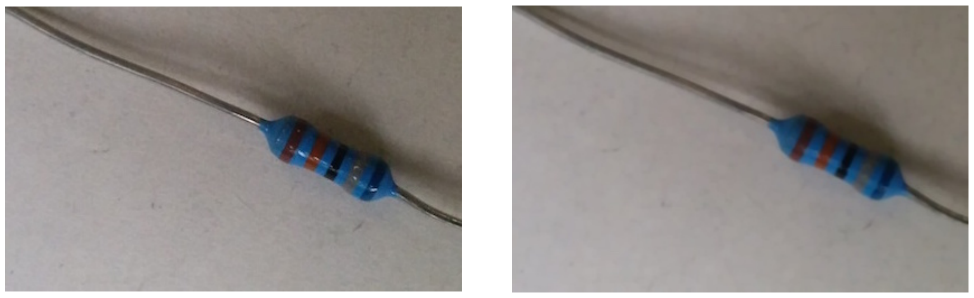

(a) Original image (b) Simplified image

Figure 3.1: The image on the right shows the effect the noise filters described above have on the original image captured by the camera.

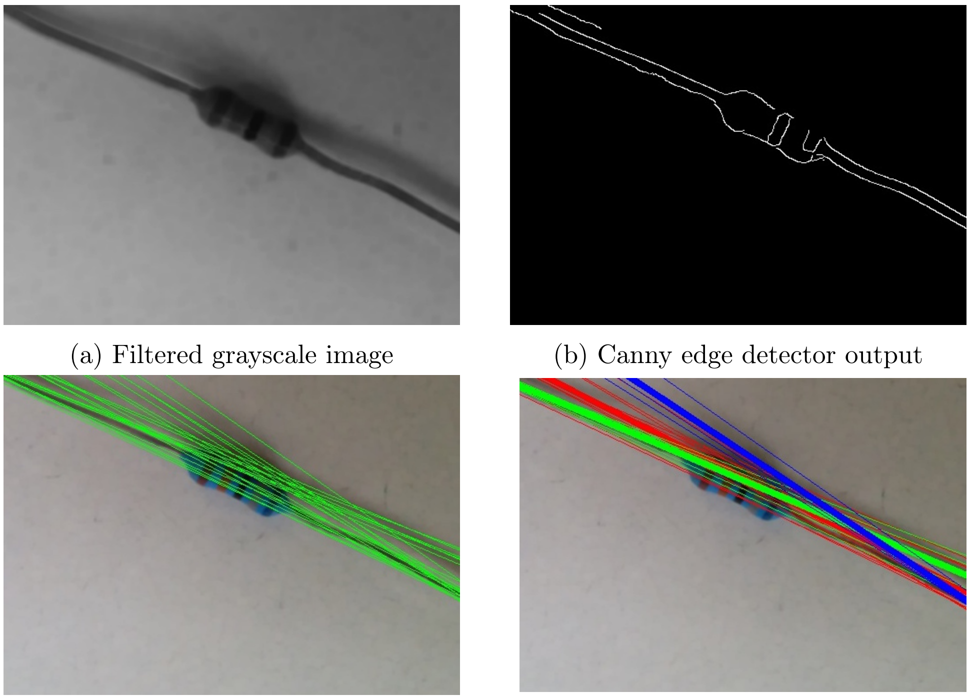
3.1.2 Hough Line Detection

The resistors are located using a line detection algorithm. Before we can apply this algorithm, we have to find edges in the given image. As the edge detection algorithm neglects color information, the simplified image gets converted to grayscale. The monochrome image is filtered with OpenCV's standard blur (see [5, p. 110] for more information) and the morphological operation *Erode* (see [5, pp. 115–120]), which improves the edge detection's results. We use a kernel size of 5 for both filters.

The Canny edge detector then can find the edges of the resistors in the image using directional derivatives. This detector was first described in [6] by J. Canny. A detailed explanation of OpenCV's implementation of the algorithm can be found in [5, pp. 151–153]. We use a low threshold of 15, a high threshold of 60, and an aperture size of 3.

As described by Bradski and Kaehler in [5, pp. 153–158], Hough line transform can be used to find lines in the binary image obtained from the Canny edge detector. We use a resolution of 1 pixel by $\frac{\pi}{180}$ radians and a threshold of 80. OpenCV's implementation of Hough line transform returns the lines parameterized with the angle θ of their normal with respect to the x-axis and their distance ρ from the origin that is located in the image's bottom left corner.

As can be seen in Figure 3.2c, the Hough line transform returns a lot of lines which are more or less parallel to the resistor. To simplify further analysis, these lines have to be grouped in *line groups* – groups of "similar" lines. Since the difference of θ between two nearly parallel lines is very small, the lines are grouped according to their angle θ in our implementation. For this, an empty list of line groups is created. For each line found with Hough line transforms, the angle θ is compared to the average angle $\bar{\theta}$ of every line group in the list. If the difference between the two angles is smaller than 0.1 radians and the respective difference between the distances ρ and $\bar{\rho}$ is smaller than 100, the line gets added to the line group. If the parameters of a found line are not within those ranges, a new line group is created. At the end of this process, all the lines are grouped in very few line groups. In Figure 3.2d the line groups are drawn in different colors, with the average line of each group drawn thicker. Although the described line grouping algorithm is sensitive to the lines' order, our tests with different types of images show that it has a good overall performance. There is a line group whose average line is parallel to the resistor in virtually every image.


(a) Filtered grayscale image (b) Canny edge detector output

(c) Hough line transform output (d) Line groups

Figure 3.2: These four images illustrate the individual steps of the line detection.

3.1.3 Statistical Analysis

The line detection algorithm explained above returns lines that are parallel to exactly one resistor. Each of these lines is now analyzed separately to determine the position of the resistor on the line and cut it out accordingly.

The simplified image is first rotated in such a way that the line is horizontal. Then, the image is cropped to a strip with a height of 30% of the original image's width around the line, called *resistor strip*. As Figure 3.3 shows, the resistor is – in the ideal case – vertically centered in this resistor strip.

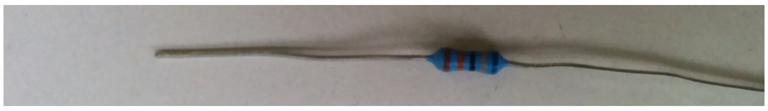


Figure 3.3: The simplified image gets rotated and cropped such that the resistor is vertically centered in the resulting resistor strip.

The next step is to locate the resistor horizontally. This can be done by using the fact that the body of a resistor is the only colorful part of the resistor strip.

To enhance this effect, a median blur filter with a relatively big aperture size (11 in our implementation) is applied and the saturation value of each pixel gets shifted such that the average saturation of the whole resistor strip is 60%. The image is then converted to Lab color space.

The color information in the Lab color space is stored in channels A and B. Besides the information about the brightness, this is taken into account in our definition of the *color deviation* σ which measures the amount of color variation. For a given array of pixels, the color deviation is defined as follows:

$$\sigma := k_1 \cdot \sigma_L + k_2 \cdot (\sigma_A + \sigma_B) \tag{3.1}$$

σ_L, σ_A and σ_B are the standard deviations of the corresponding color channels of all pixels. k_1 and k_2 are scaling factors that are set to 1 and 10 respectively in our implementation.

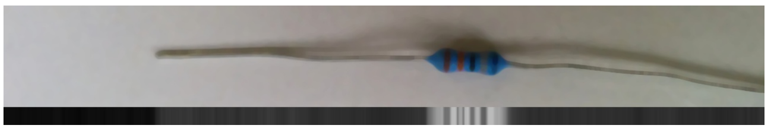


Figure 3.4: Each pixel column below the resistor strip is colored according to the color deviation of the column above it: the higher the color deviation, the brighter the column.

As can be seen in Figure 3.4, the color deviation allows us to locate the resistor with a high precision. For this, the color deviation of each column is calculated. If it exceeds a limit σ_0, the column is marked as *belonging to the resistor*. To allow a restrictive selection of σ_0, the marked sections of the resistor strip are defragmented: If the distance between two marked columns is less than d_0 pixels, all the columns in between are also marked as *belonging to the resistor*. At the end of this process, the whole resistor strip is divided into regions *belonging to the resistor* (*resistor regions*) and regions that do not. If a resistor region is longer than l_0 pixels, it is regarded as potential resistor and the corresponding part of the resistor strip gets analyzed vertically.

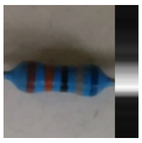


Figure 3.5: Each pixel row to the right of the resistor strip is colored according to the color deviation of the row left of it: the higher the color deviation, the brighter the row.

The vertical analysis works similar to the horizontal one. At the end, the biggest resistor region gets selected. If it is more than h_0 pixels high, it is cut out and added to the list of resistors.

In our implementation, we use the following parameters: $\sigma_0 = 70$, $d_0 = 20$ (horizontal analysis) or $d_0 = 10$ (vertical analysis), $l_0 = h_0 = 21$.

After all lines have been analyzed, we are left with a list of resistor images and their location in the original camera image.

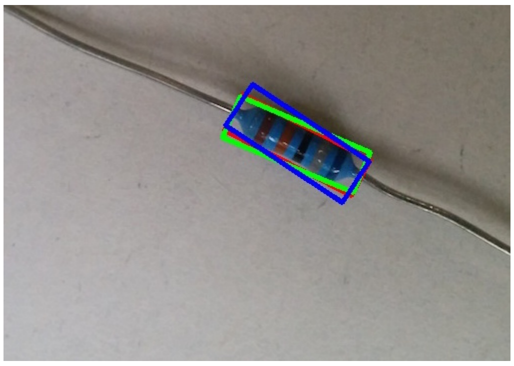


Figure 3.6: The algorithm found a resistor for each line group in 3.2d – here marked by a box in the respective color.

In Figure 3.6, we can see an example image in which the algorithm found three versions of the same resistor, one for each line group. In this case (two or more possible resistor boundaries intersect), the algorithm chooses the resistor with the smallest area and removes the others from the list. The reason for this is to avoid analyzing the same resistor multiple times in the following (computation-expensive) stages.

3.2 Color Ring Detection

The second stage of our algorithm detects all color rings of the resistor in the cropped image from the first stage. This stage is split in two parts. In the first part, the algorithm tries to find the edges of the color rings. In the second part, a penalty system is used to decide which edges belong to which color rings – aiming at having a list of all color rings.

3.2.1 Edge Detection

The edges of the color rings are detected by observing changes in color along the vertical axis. We assume that the image is rotated in such a way that the rings

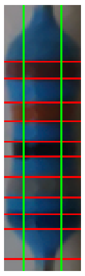


Figure 3.7: This illustrates the result of the edge detection. The green vertical lines highlight the column range of which we calculate the RSMs while the red horizontal lines highlight the detected edges

are aligned horizontally. A color ring's edge is characterized by an abrupt change in color between very few rows. We use the CIE ΔE^*_{94} function (Equation 2.4) to measure the change in color. As this function is designed for the Lab color space, the image has to be converted before the edge detection.

Assuming minor color changes along the horizontal axis, it is a valid simplification to calculate the mean color of a specific column range in one row, hereinafter called *row section mean* (RSM). The restriction to a smaller range of columns reduces the influence of the background in further analysis. For our selected resolution, we use a 21 pixel wide column range in the center of the image. In Figure 3.7, the RSM is calculated from the sections between the green vertical lines.

To decide whether a row is the edge of a color ring, the RSM of this row can be compared to the RSM of the previous row using the CIE ΔE^*_{94} function. As color changes can happen over multiple rows, the comparison to only the previous row can be misleading. This especially holds for high resolution images, where color changes are smoother. To counter this effect, our algorithm calculates the mean of several previous rows' RSMs and compares it to the RSM of the current row. To ensure that the color changes of previous edges have no impact on the current potential edge, only the last $H = N/20$ RSMs are compared to the RSM of the current row (for an image with N rows in total).

Concretely, the edge detection works as follows: First, the RSM of each row is calculated and saved in a list. Then, every RSM in this list gets compared to the average of the last H RSMs. If the distance between these two is greater than c_0, the corresponding row is added to the list of color ring edges. To avoid detecting one edge multiple times (for instance when the change in color stretches across multiple rows), there has to be one row that is not considered to be an edge before another edge can be added to the list. We use a value of 6.0 for c_0 in our implementation.

3.2.2 Penalty System

In the second part of the color ring detection, the list of edges from the first part is used to find the color rings. For each resistor segment between a pair of consecutive edges, the algorithm has to decide whether the section belongs to an *actual color ring* (ACR) or to the background. The results for all the resistor sections can be encoded by a bit string: 1 means that the corresponding section belongs to an ACR, 0 that it is part of the background. It is possible that more than one edge is recognized inside one ACR or background ring. Therefore, an ACR is represented by a sequence of ones enclosed by zeros and a background ring vice versa. The bit string has a length of $k - 1$, where k is the length of the list of edges. So, there are 2^{k-1} different ACR-background assignments if we consider every possible combination.

(a) 101010100 (b) 101101010

Figure 3.8: Subfigures (a) and (b) show two different ACR-background assignments with the corresponding bit string. (b) is the correct assignment.

To find the correct ACR-background assignment, our algorithm uses a penalty system. Each possible bit string is rated based on multiple criteria. The bit string with the lowest penalty is returned. The main advantages of such a penalty system are that every criteria can be rated dynamically based on the deviation from a mean value and that new criteria can easily be added. A drawback is, however, that each combination is in principle possible. Thus, well defined criteria and penalties are required to avoid invalid results. Our algorithm evaluates five criteria:

1. *Number of rings*

   If the number of ACRs is too small, a very large penalty p_1 is added. If the tolerance ring was not found, a medium penalty p_2 is added.

2. *Width of color rings*

   The width of all ACRs should be more or less equal. So, if the width w of an ACR is smaller than 75% of the mean width $\bar{w}$ of all resistor sections, a proportional penalty of $p_3 \cdot |w - \bar{w}|$ is added. If w is more than 25% greater than $\bar{w}$, a slightly higher penalty of $p_4 \cdot |w - \bar{w}|$ is added. Too wide rings get penalized stronger because it is less likely to have a wider ACR than the mean than to have a narrower one.

3. *Color distances of the ACRs*

   The third criteria evaluates the color distance between each ring and the overall color mean of the resistor. For each ACR, a penalty inversely proportional to the color distance is assigned (p_5 divided by the color distance). The idea behind this is that the ACRs' color is different from the overall mean while the background rings' color is similar.

4. *Color distances of the background rings*

   All background rings should have a similar color. For each presumed background ring, a penalty of p_7 times the distance to the color mean of the whole resistor is assigned.

5. *Color ring order*

   This criteria penalizes a combination if two or more consecutive zeros or ones occur, because in the ideal case the combination should have alternating ones and zeros. For each pair of ones or zeros, a small penalty p_6 is added.

After careful tuning, we chose the following parameters for our implementation:

$$p_1 = 10000 \quad p_2 = 50 \quad p_3 = 10 \quad p_4 = 15 \quad p_5 = 300 \quad p_6 = 5 \quad p_7 = 2$$

3.3 Color Classification

In this section, we describe the third and final stage of our algorithm. It is a major challenge to assign the correct color to each detected ACR. It leads to 10 comparisons of the ACR mean color to predefined reference colors. This is again done with the ΔE^*_{94} metric (Equation 2.4).

The algorithm compares the ACR color mean to each reference color. Instead of having just one reference value for each identifiable color, it is a better practice to have a list of reference values for each color. Each of the 10 selectable colors are rated according to the color distances between their reference colors and the ACR color mean. The lowest three color distances are summed up for each selectable color to finally choose the color with the lowest sum of those three minimal distances. This method covers variations of colors, since for example the color red is not the same on each type of resistor. Furthermore, it helps to detect the correct color even if the lighting conditions are not the same in every image by including darker and brighter shades of each color in the respective reference list.

Further improvement can be obtained by excluding impossible color assignments. According to the calculation formula for the resistance described in 2.1, there exist some color combinations that lead to to an invalid resistance. The most important exclusion condition is that the first ring cannot be black because that would mean that the resistance is $0\,\Omega$. This especially helps to guess the first ring correctly, since a major problem turned out to be the distinction between black and brown. A further restriction belongs to the multiplier ring (either the third or the fourth ring, depending on the total number of rings). This ring can neither be gray nor white, since multipliers of 10^8 and 10^9 are not allowed.

A problem with the color classification is that both gray and brown are found within a wide range of values in the Lab color space. This makes it very hard to define those colors via the reference lists. To mitigate this error, the algorithm does not only look at the most likely but also at the second most likely color. We figured out that often, when gray is the most likely color, the ring is in fact brown, which is found as the second most likely color. This does not happen the other way around: for a gray ring the second most likely color is usually white. Based on those observations, we decided that the algorithm chooses brown if the most likely color is gray and the second most likely is brown. This manipulation is valid especially when keeping in mind that brown is one of the most occurring colors and therefore should be found more often than gray.

Having classified the color of each ACR, it is now straightforward to calculate the resistance according to Equations 2.2 and 2.3.

Chapter 4

Implementation

Besides developing an algorithm, we also wanted to have a finished implementation of it in form of an Android app. An important factor is the usability. The user interface should be as simple and as intuitive as possible.

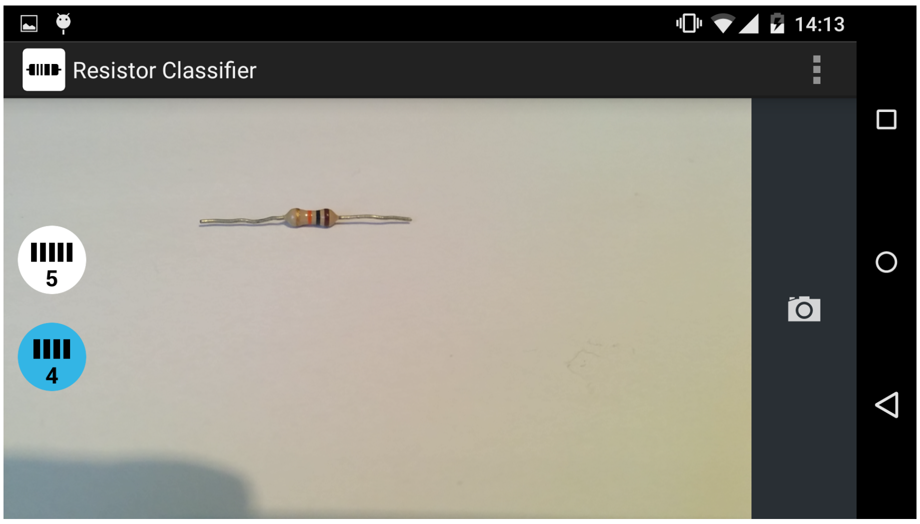


Figure 4.1: The camera preview is in the center of the apps' main window as well as all buttons needed for a scan.

The main window of our app can be seen in Figure 4.1. The biggest part is the camera preview. With the two buttons on the left side, the user can select whether a four ring or a five ring resistor should be detected. The big button on the right allows to take a picture, which will then be analyzed. Additionally, the camera preview allows manual focus by touching the selected region. Figure 4.2 shows the result screen that is displayed after the image has been processed. On the top, the user sees the cropped image with the original resistor. The green lines mark the detected edges and the green boxes the chosen ACRs. Below the cropped image, there is a schematic drawing of the detected resistor with its color rings. As mentioned before, there can exist two valid resistances for a detected color combination. Both are listed below the drawing. The result screen is actually a slider gallery and each detected resistor is on a separate page (denoted by the blue dot on the bottom of the screen). In this example, there is only one resistor in the original image, which is classified correctly (10 kΩ).

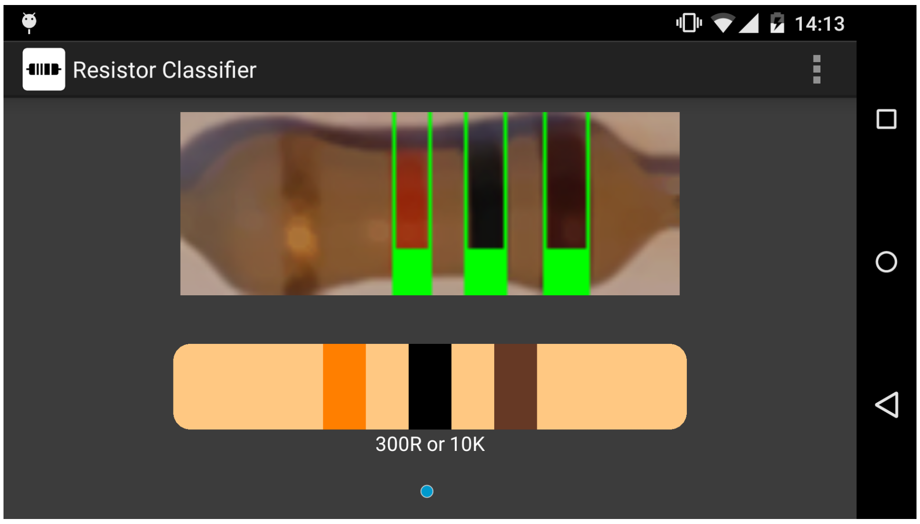


Figure 4.2: On the top of the result screen that is displayed after the algorithm has analyzed the image, there is a cropped image with the original resistor. In addition, there is a schematic drawing of the detected resistor for better visibility.

However, it may happen that a color ring gets assigned the wrong color. Then, the user has the possibility to manually change this specific color by tapping on the respective color ring in the schematic drawing. A dialog shows up with a list of all possible colors. For each color in the list, a blue bar indicates the probability that this color is correct according to the algorithm. Colors that are more likely to match the ring are listed first. The dialog can be seen in Figure 4.3.

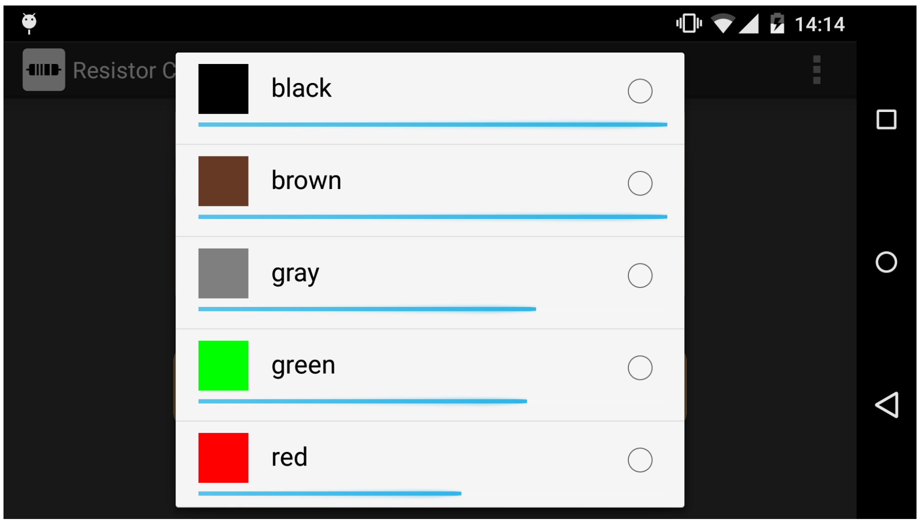


Figure 4.3: After tapping a ring on the schematic drawing, the color selection dialog shows up and the user can correct an assigned color.

Chapter 5

Evaluation and Discussion

In this chapter, we describe how we evaluate our algorithm. We developed a separate evaluation routine which analyzes pictures of 29 different resistors. Three different devices were used: HTC One, LG Google Nexus 5 and Samsung S3. Every resistor was photographed in three ways with every device (horizontal, tilt left and tilt right). This results in 261 different images.

The evaluation system allows to check all stages described in chapter 3 one after the other. The first two stages (resistor localization and color ring detection) have to be evaluated manually, while the results of the third stage are compared automatically to a predefined list of the real color rings. The two parts of the second stage (edge detection and the penalty system) are assessed individually.

5.1 Evaluation Results

The results of the evaluation can be seen in Figure 5.1 and Table 5.1. The algorithm could only classify 36 of the 261 resistors correctly. However, a detailed analysis of the numbers for the individual stages shows that the resistor localization in particular and the color ring detection in general yield good results. The most problematic stage is clearly the color classification because it only assigned correct colors to 25% of the images that passed the previous stages.

One's attention has to be turned to the last stage. Across all three tested devices, the color detection yields poor results. One reason for this is missing calibration of the algorithm in favor of usability. This means that the characteristics of a specific camera and environmental conditions are not taken into account. This leaves room for improvement. Some suggestions can be found in Chapter 6.

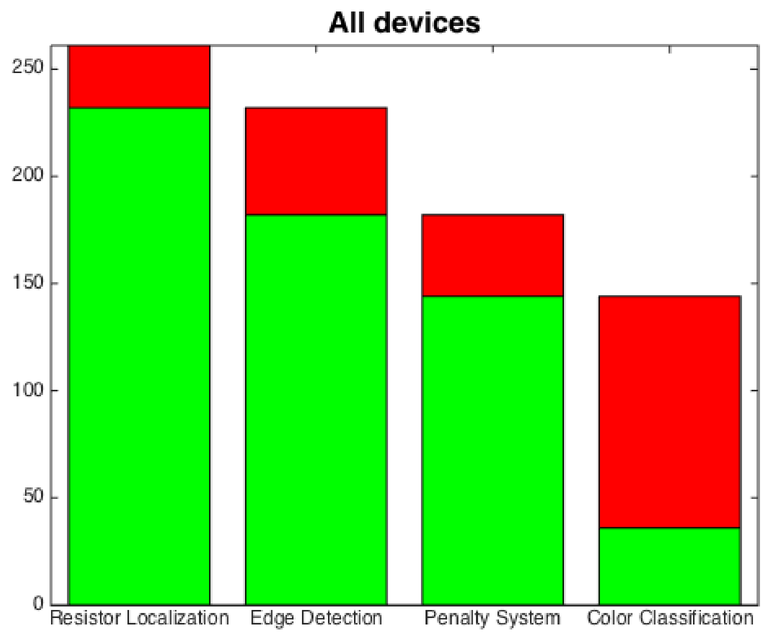


Figure 5.1: The four bars indicate the amount of resistors that passed the corresponding algorithm stage successfully (green) or failed at it (red).

	All devices	Samsung S3	Nexus 5	HTC One
Resistor localization	**88.9%**	82.8%	89.7%	94.3%
Edge detection	**78.4%**	83.3%	76.9%	75.6%
Penalty system	**79.1%**	80.0%	70.0%	87.1%
Color Classification	**25.0%**	33.3%	26.2%	16.7%

Table 5.1: This table shows the percentages of resistors that passed the corresponding algorithm stage successfully whereby only the resistors that passed all previous stages are taken into account.

5.1.1 Color Classification

A more detailed analysis of the color classification can be seen in Figure 5.2 and Table 5.2. The colors brown, orange, yellow and violet are recognized very unreliably. We explain this by the following reasons:

– In the Lab space, there are many triples that lead to a brown-like color. Since our color detection chooses the reference color that is located closest to the measured color, it happens that – in the worst case – nearly every other color can be chosen instead of brown. Moreover, a dark brown is often recognized as black. Any adjustment to the reference list of brown would invert the problem because black rings would be chosen as brown. Unfortunately, brown is along with black the most occurrent color. This error explains a great amount of the wrongly classified resistors.

– Orange and red are described by very similar triples in the Lab space. This leads to wrong assignments. Many orange rings are rather dark and they are therefore often classified as red or brown. However, a change in the reference list of orange would again invert the problem as many red and brown rings would be classified as orange.

– The poor results in the detection of yellow come from the fact that yellow rings look more like green ones on a blue resistor. On a beige resistor, they are sometimes not even recognized because their edges are not detectable.

– Since violet and blue are also described by similar triples, it is likely that these two colors get interchanged. Violet rings are usually dark (that means not a high red portion) and therefore seem to be blue. Just as in the cases above, any adjustment to the reference list of violet would invert the problem.

As white is probably the rarest color, we did not have any resistors with white rings on it. However, white is – similar to black – relatively good recognizable because it is located close to a boundary of the brightness coordinate L. There exists no color that is brighter than white and, similarly, no color that is darker than black.

Black	Brown	Red	Orange	Yellow
90.0%	50.0%	68.0%	21.1%	5.3%

Green	Blue	Violet	Gray	White
69.7%	80.0%	35.7%	62.5%	-

Table 5.2: This table specifies the percentage of correct classification for every identifiable color.

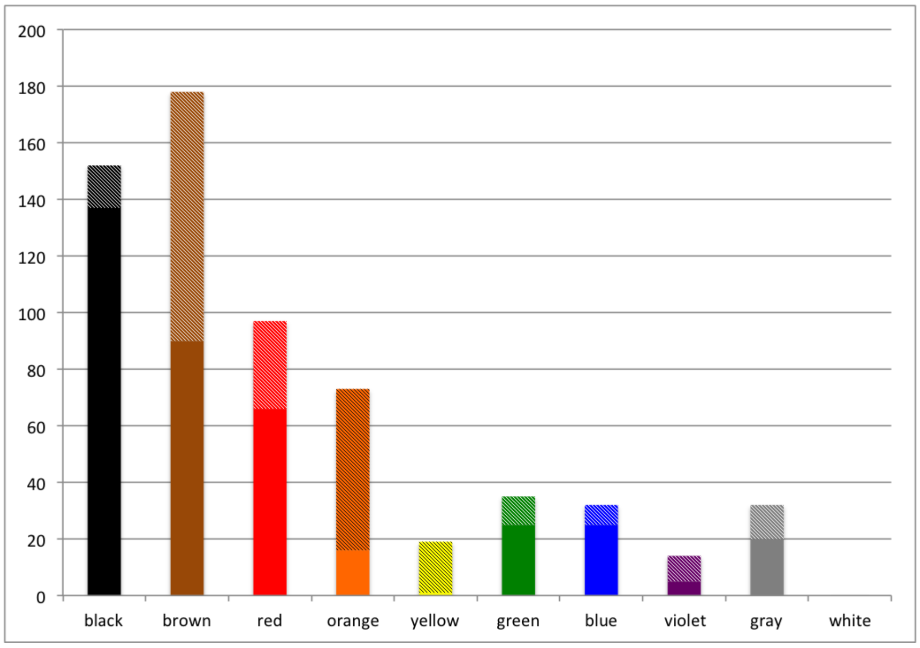


Figure 5.2: The height of each bar denotes the total amount of rings with the respective color. The filled part shows the number of correctly classified rings and the hatched part the number of rings that are misleadingly assigned another color.

Chapter 6

Future Work

As explained in the previous chapter, the detection of the colors turned out to be a problem. It is important to have an algorithm which is able to distinguish different colors on different backgrounds. A possible improvement could be obtained by considering the background, so that the color of a pixel region also depended on the color of the regions next to it. The human eye distinguishes colors in a similar way; it continuously adapts itself to the background color.

Instead of comparing each color to a list of reference colors, it could be a better approach to calibrate the reference list for every image based on the lighting conditions and the background. This would allow a reliable color detection under different circumstances. One could even go so far as to require a card with all 10 colors printed on it that has to be in the picture as well. This would allow the algorithm to get a proper reference value for each color.

An entirely different approach to the problem of determining the resistance of a resistor with a smartphone could be the measuring of the resistor with an external hardware device. It would contain a voltage divider and a little amplifier and could be connected via the headphone/microphone jack. Through the headphone output, a small voltage can be applied on the additional device, whose output voltage then is measured with the microphone input. Based on the ratio of those two voltages, the value of the connected resistor can be calculated.

Acknowledgements

Finally, we would like to thank Tobias Langner and Jochen Seidel for the (usually) weekly meetings, their constant support and useful inputs.

Bibliography

[1] ResCan. http://armageddon421.de/?p=279 (February 2012) Acessed: 2014-01-11.

[2] Nothing Labs: Resistor Photo ID. http://www.nothinglabs.com/resistorphotoid/ (before September 2013) Acessed: 2014-01-11.

[3] Luo, M.R.: Development of colour-difference formulae. Review of Progress in Coloration and Related Topics **32** (June 2002) 28–39

[4] OpenCV. http://opencv.org/

[5] Bradski, G., Kaehler, A.: Learning OpenCV. Computer Vision with the OpenCV Library. O'Reilly (2008)

[6] Canny, J.: A computational approach to edge detection. Pattern Analysis and Machine Intelligence, IEEE Transactions on **PAMI-8**(6) (November 1986) 679–698