Computer Vision Laboratories

Project Final - Resistor Recognition

EE/CPE 428 - 03

Computer Vision

Winter 2023

Group 7

Students: Eitan Klass, Roey Mevorach, Brandon Pragasa, and

Nathan Jaggers

Project Due: 03/24/22

Instructor: Dr. Zhang

Table of Contents

Introduction

In today's day and age, electronics are everywhere. They make your coffee, measure the temperature, are the heart of all your computers, and so on and so forth. Part of the designs for these electronics is carefully selected resistors that restrict current flow and adjust a signal to allow the device to do its job. If for whatever reason a resistor other than the intended one was to take its place then this could cause the device to not work or be damaged. That's why it is important to verify the nominal value of a resistor before using it. When done manually, however, this becomes a very monotonous task that is prone to error. This is where computer vision can help. With the aid of computer vision technology, it can be leveraged to automate the resistor value detection process and enable us to be more efficient.

Problem

Resistors are commonly used in electronic circuits to regulate voltage and limit current. The bands on a resistor indicate its resistance, tolerance, and sometimes temperature coefficient. Accurately reading these bands can be challenging, particularly when the bands are small or obscured. Although humans can see colors very well and the resistor color bands are a reliable way to decode the nominal resistance value of a resistor, there are a few pitfalls here.

First is that not all humans see color the same way, some people are color blind which makes trying to read a resistor difficult. Even in the case where someone isn't color blind and they can accurately identify resistors, as stated before, it is a monotonous task that could lead to the person becoming less focused, in turn increasing the risk of human error. Finally, it is a time-consuming task. It doesn't take much time to decode a few resistors but if you have to sort through a large quantity of them, that could take a significant portion of time that could be better

used elsewhere. Imagine trying to debug a circuit on a breadboard or PCB where there are 100 resistors or more. It would be a serious undertaking to decode and keep track of them all.

Proposal

Resistors are one of the most commonly used electrical components in electronic circuits. Identifying and measuring the value of a resistor is a crucial step in circuit design and testing. In this project, we aim to recognize resistors from an image and determine their resistance value by leveraging several computer vision operations and concepts. The goal of our project is to create a program capable of accurately identifying the resistance values of resistors in images by analyzing the colored bands present on the resistor and determining their corresponding values.

For this project, a few assumptions are made about the resistors and the given images. The resistors to be identified will be 4 band resistors, oriented in the correct direction when reading bands from left to right, and there will only be one resistor per image to identify. The images will be well lit and the resistors will be a pretty close distance. *Figure 1* serves as a prime example of the expected quality and content of the image.

Figure 1: Carbon Film Resistor

Background

A resistor is an electrical component that reduces the flow of current. There are many types of resistors like fixed resistors or potentiometers with variable resistance. Fixed resistors also have subsets in themselves like the popular carbon film resistor as seen in *Figure 1* in the previous section.

Many resistors look like this with a cylindrical body and leads coming out opposite ends. Note the color of the casing, the color of the bands, and the reflective shine of the resistor. The color bands are important because that is the color code identifying the nominal value of the resistor. The bands can be decoded using *Figure 2* below. Note how different amounts of bands follow slightly different rules for decoding. It is also worth noting that the color bands have different meanings depending on where they are located on the resistor.

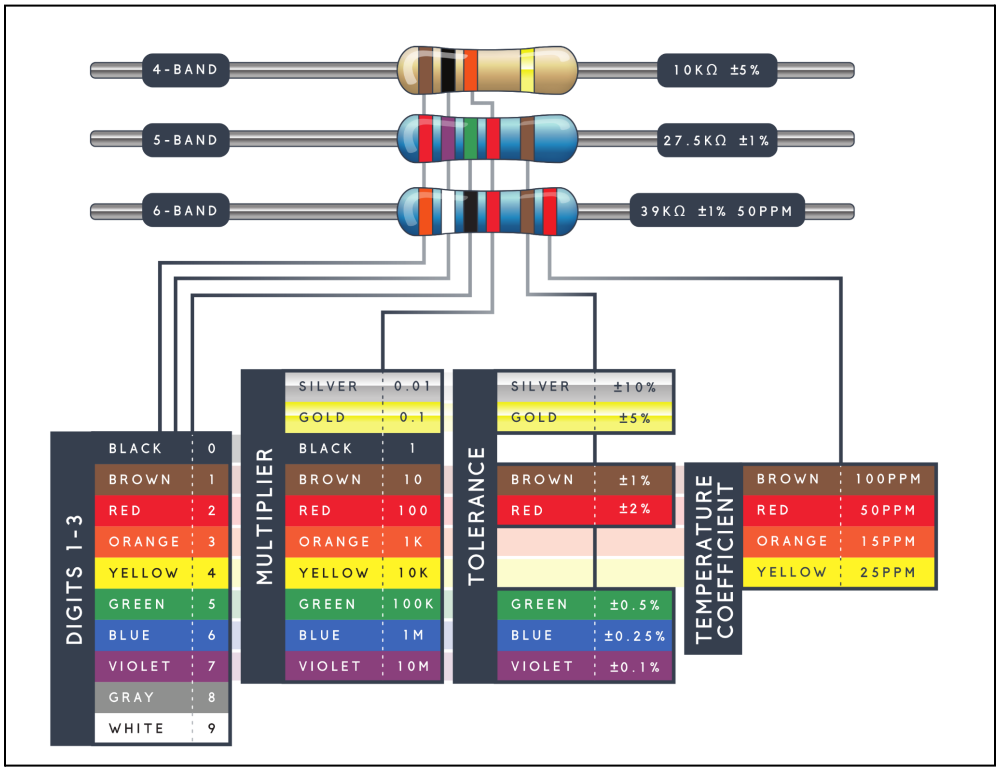

Figure 2: Resistor Color Band Decoder

There have been many previous attempts and implementations for using computer vision to identify a resistor and its resistance. One example is “Automatic Segmentation and Classification of Resistors in Digital Images.” This group proposed an automated method for detecting and classifying resistors in digital images. The approach involved several stages of preprocessing the input image to detect the resistor's boundaries, separate the resistor from the background, and extract features to classify the resistor color bands and determine the nominal value. They implemented their algorithm using OpenCV and used computer vision techniques like image segmentation, masking, and thresholding as well as a Support Vector Machine (SVM) to aid in the classification of resistor bands.

Unlike the previous group, "A new method of resistor's color rings detection based on machine vision" uses a method for detecting the color rings on a resistor using machine learning techniques. The authors had developed this method with industrial applications for quality control purposes in mind where they would use industrial black and white cameras. The method involves capturing an image of the resistor using a camera and then processing the image to identify the colors of the rings even though they are in grayscale. The authors developed a color segmentation algorithm that can accurately segment the rings from the rest of the resistor. Then a neural network to classify the colors of the rings.

The last paper explored in preparation for this project was “Real-Time Resistor Color Code Recognition using Image Processing in Mobile Devices” which has many commonalities with the first paper [1]. The difference here is that the authors present a real-time application for android devices that uses a video feed instead of still images. This presents them with different challenges and solutions compared to the first group. They employed computer vision techniques like nonlinear filtering, template matching, and photometric invariance (RGB to HSV) to allow

for the recognition of the resistor in the video feed and extraction of the color bands. To classify the features they developed a Euclidean distance-based clustering strategy.

From these papers, we learned that in this project we should work in the HSV color space to avoid problems with inconsistent lighting conditions. There were also some recommendations from the authors of the first paper [1] to avoid statistical analysis for the location of resistor bands. Instead, they recommended developing an edge detection algorithm for band position estimation or other band detectors to know the location of the band prior to image preprocessing.

Procedure

After conducting a thorough review of previous research on similar projects, we drew inspiration from the approach presented in "Automatic Segmentation and Classification of Resistors in Digital Images." Our approach closely follows theirs but with some modifications. As depicted in *Figure 3*, we have developed an algorithm that accurately detects and identifies the colored bands on a resistor to recognize its value.

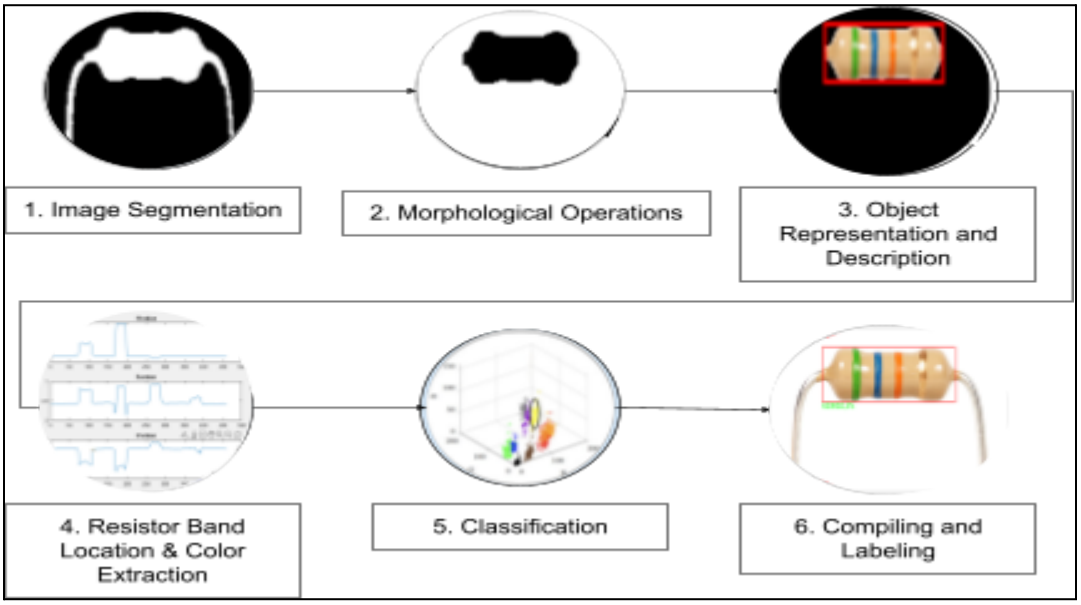

Figure 3: Resistor Recognition Algorithm Outline

The first step in our resistor recognition algorithm involves segmenting the resistor body from the rest of the image. To accomplish this we break it up into several stages. Before extracting any information from the image, a few pre-processing techniques need to be applied. To ensure consistent image segmentation, we first employ the 'rgb2gray' function to convert the image to grayscale. Next, we utilize the 'medfilt2' function to smooth the image with a 9x9

median filter, minimizing illumination and reducing noise. This step is crucial in preserving the edges of the resistor body and its bands while achieving a consistent background for accurate image segmentation.

Selecting the appropriate segmentation method is critical to the success of the project. To achieve this, we tested two different approaches and compared their outcomes. Specifically, we compared edge-based segmentation and thresholding segmentation on two different images with varying backgrounds, as seen in *Figures 4 and 5*: one with high contrast and the other with a noisy background.

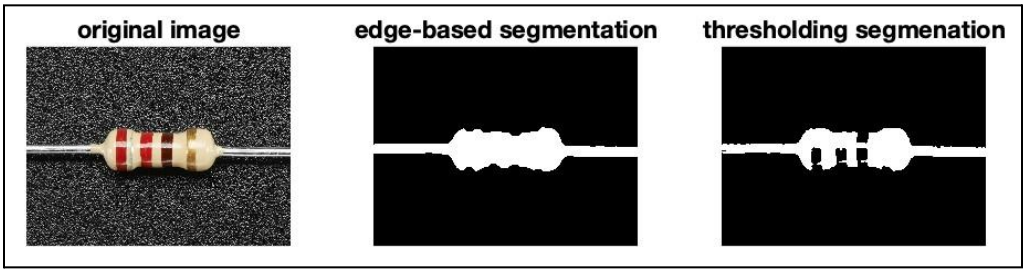

Figure 4: Image Segmentation on Images with Noisy Backgrounds

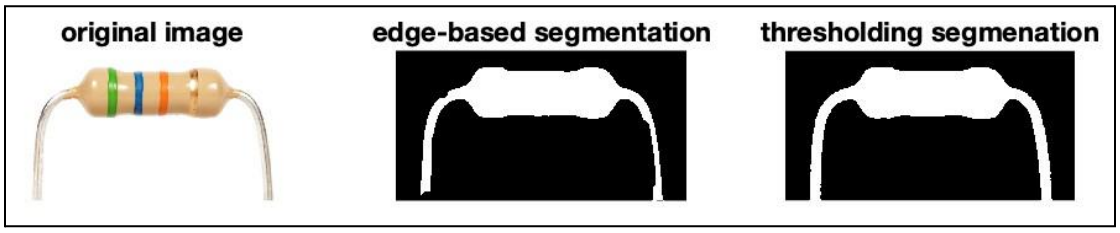

Figure 5: Image Segmentation on Images with High-Contrast Backgrounds

The comparison results, as shown in *Figure 5*, demonstrate that both methods are effective on images with high-contrast backgrounds. However, when the image has a noisy or non-cohesive background, edge-based segmentation significantly outperforms thresholding segmentation, as

depicted in *Figure 4*. Based on these findings, we opted to utilize edge detection to identify the resistor body outline.

In our edge-based segmentation function, we employ several built-in functions to obtain the desired outcome. First, we perform edge detection with the Prewitt operator to extract the edges in the image. We then utilize the 'bwareopen' function to remove all connected objects having fewer than 20 pixels. Afterward, the 'imclose' function closes the gaps between lines as only resistor pixels should be present at this stage. Next, we fill in the lines using the 'imfill' function to create one large object, and finally, we return the largest object using the 'bwpropfiltfill' function to extract the object of interest. The segmentation process is illustrated in *Figure 6* below.

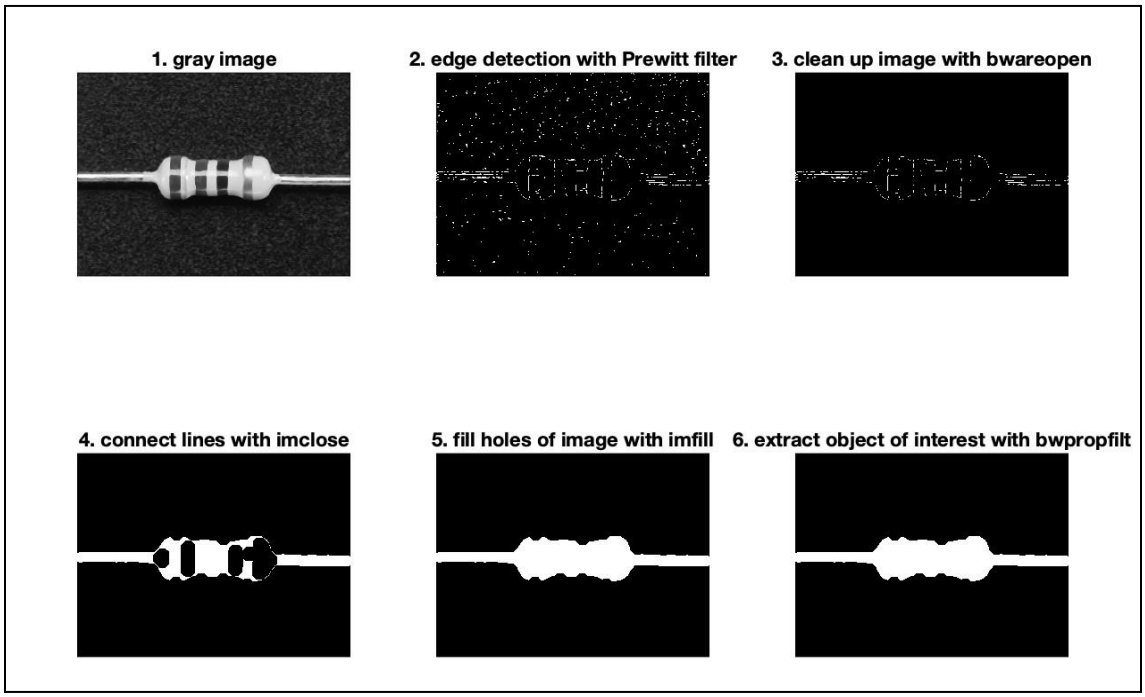

Figure 6: Edge-Based Segmentation Process

After obtaining the mask of the resistor and its leads, we utilized morphological operations, specifically erosion, and dilation, to eliminate the leads of the resistor. We use the ‘imerode’ function to erode the segmentation result and then use the ‘bwpropfiltfill’ function to extract the largest object in the image. This removes any residual lead pixels and leaves only the resistor. Finally, we utilize the ‘imdilate’ function to dilate the image and return the final resistor mask. The results of each morphological operation are displayed in *Figure 7*.

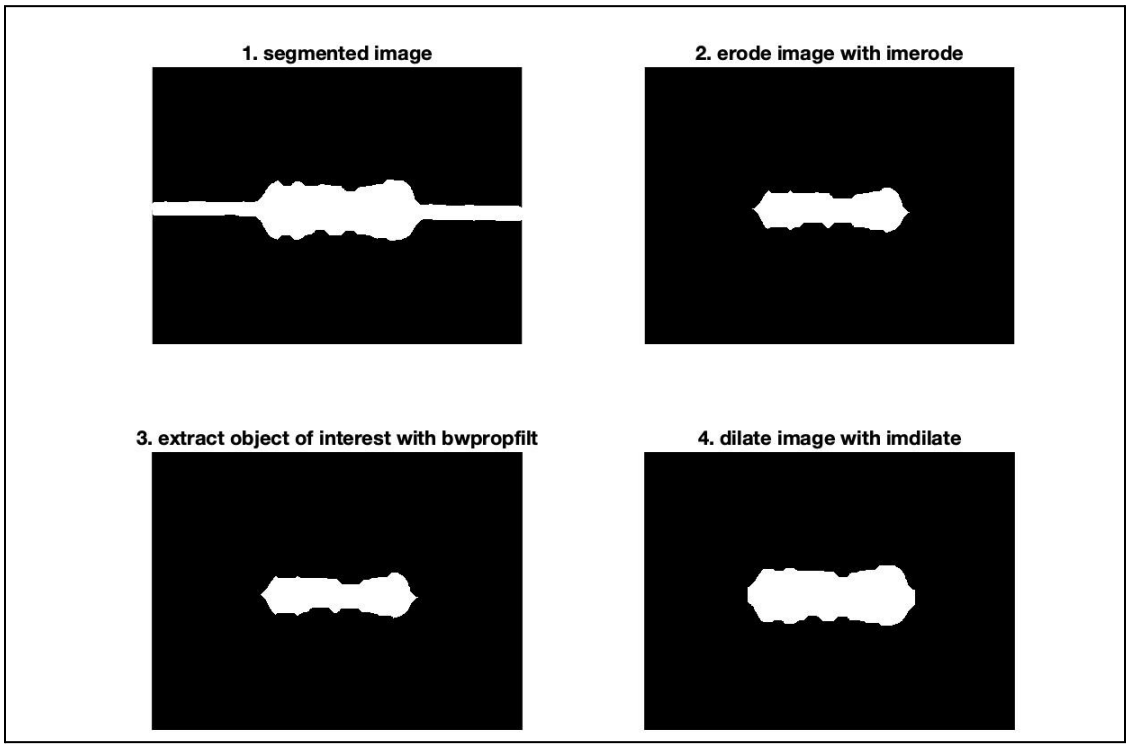

Figure 7: Edge-Based Segmentation Process

Code Snippet 1 illustrates the morphological operations used to acquire the resistor mask and how the mask is employed to extract the resistor from the original image. It is important to note that the resistor mask must be inverted to be properly utilized. Another significant aspect of the code is the creation of the 'strel' object. The dimensions and shape of the ‘strel’ object are

parameters that need to be adjusted by the user for the erosion and dilation functions to work correctly. We discovered that a rectangle with the dimensions of 25x40 works best for the majority of our training images.

```
% Dilate and Erode leads of resistors
se = strel("rectangle", [40 25]);
eroded_img = imerode(res_seg, se);
eroded_img = bwpropfilt(eroded_img, 'Area', 1, 'Largest');
dilated_img = ~imdilate(eroded_img, se);
subplot(612); imshow(dilated_img); title('Dilated Image');

% Mask out resistor body from background
res_body = imoverlay(res1, dilated_img, 'black');
```

Code Snippet 1: Morphological Operations and Masking Original Image

After getting the masked RGB image after the morphological operations, the next step was the extraction of the location and color of the resistor body. To optimize this process, we used “regionprops” on the post-morphological operation image to obtain a bounding box to get the area of the resistor body from the original image to then work on the cropped image. From further analysis of the dimensions of the bands and ends of the resistor body, we obtained another cropped image from the resistor body of a slice of the resistor as shown in *Figure 8* and coded in *Code Snippet 2*.

```
% Get slice of resistor body for further processing
slice = imcrop(res_body,[box(1) + box(3)*0.1, box(2) + box(4)/2, box(3) -
box(3)*.2, box(4)/4]);
subplot(615); imshow(slice); title('Slice Resistor')
% Segment the slice for getting bounding boxes around bands
gray_slice = rgb2gray(slice);
mask2 = gray_slice < 160;
subplot(616); imshow(mask2); title(' Gray Slice Resistor')
```

Code Snippet 2: Cropping Resistor Body

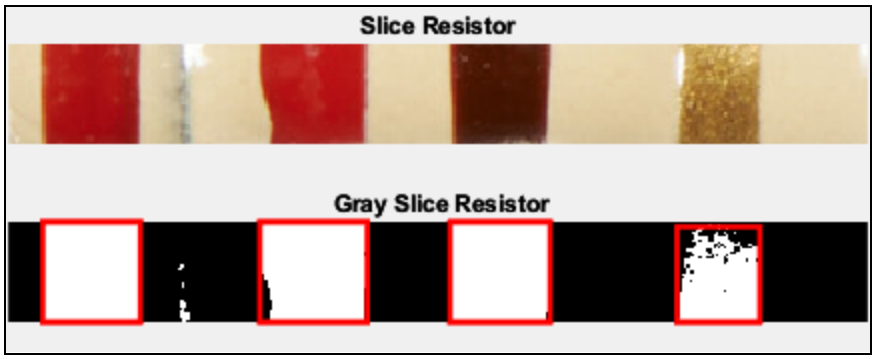

Figure 8: Slice of Resistor Body

Next, we performed segmentation with thresholding on the slice of the resistor and used the "regionprops" function to get bounding boxes around the bands of the resistor. However, there would be regions of various sizes detected from using the function, so we filtered those out by thresholding based on the area of the box in order to get rid of smaller regions and only work on the band regions when compiling the feature array for the resistor. From each bounding box that surrounded a band, the start and end points along the x-axis were extracted from the attributes of each band's bounding box as shown in *Code Snippet 3*.

```
% Iterate through objects and filter
band_num = 1;
for i=1:size(props2, 1)
   box2 = props2.BoundingBox(i,:);
   % Use box area as threshold for detecting bands
   box2_area = box2(3) * box2(4);
   if (box2_area > 100)
       subplot(616); rectangle('Position', [box2(1), box2(2), box2(3), box2(4)],
'EdgeColor', 'r', 'LineWidth', 2);
       % Assign the start & end x-axis values to feature array
       feature_arr(4,band_num) = round(box2(1));
       feature_arr(5,band_num) = round(box2(1)+box2(3));
       band_num = band_num + 1;
   end
end
```

Code Snippet 3: Filtering for Band BoundingBox objects

After obtaining the bandwidths, we created histograms for each channel of the HSV color space that examined each value across the x-axis of the image. For each channel, we reduced the matrix to a vector of the averages of the columns and put the corresponding hue, saturation, and intensity values into their correct bands. For each band's HSV values, we simply averaged the bandwidth regions of the HSV vectors and used this finalized feature array for classification.

To set up the classification algorithm, we set up HSV arrays, a color index array, a multiplier vector, and a tolerance vector. Then, we inputted the feature array along with the color index array into the "hsvToIdx" function to obtain a reduced feature array that contained four elements with corresponding indexes. Within this function, we had to iterate through the original feature array and determine if the HSV values were within a certain error percentage for each color's HSV in the colorHSV matrix shown in *Code Snippet 4*. At this point, we realized that for each different image, we had to adjust the error percentage in order to correctly classify the colors of the bands because of the influence that illumination had.

```
% HSV values for the colors of a
resistor
blackHSV = [0 0 0];
brownHSV = [0.0472 0.8296 0.3554];
redHSV = [0.4067 0.8393 0.6278];
orangeHSV = [0.0690 0.8502 0.9385];
yellowHSV = [0.17 1 1];
greenHSV = [0.2604 0.7459 0.6522];
blueHSV = [0.5037 0.6880 0.6035];
violetHSV = [0.83 0.5 1];
greyHSV = [0 0 0.5];
whiteHSV = [0 0 1];
goldHSV = [0.0898 0.5608 0.8463];
silverHSV = [0 0.75 0.75];
```

```
function feat_arr = hsvToIdx(featVec, color)
   tmp_arr = zeros(1,4);
   errorRange = 0.25;
   numBands = size(featVec, 2);
   numColors = size(color, 1);
   for i = 1:numBands
       for j = 1:numColors
           thisColor = color(j,:);
           hueInRange = inrange(featVec(1, i),
thisColor(1), errorRange);
           satInRange = inrange(featVec(2, i),
thisColor(2), errorRange);
           valInRange = inrange(featVec(3, i),
thisColor(3), errorRange);
           % disp(thisColor(1))
           if (hueInRange && valInRange)
               tmp_arr(i) = j;
               break;
           end
       end
   end
   feat_arr = tmp_arr;
end
```

Code Snippet 4: Filtering for Band BoundingBox objects

Finally, after obtaining the reduced feature array/class vector for the bands, we inputted this into our “determine_res” function in order to get strings of the resistor’s ohm value and tolerance value needed for labeling the resistor in the original image. In the last step, we showed the detection of the resistor by drawing a rectangle around the resistor body in the original image and displaying a text of the resistor’s nominal value and tolerance next to the rectangle in the image.

Results and Analysis

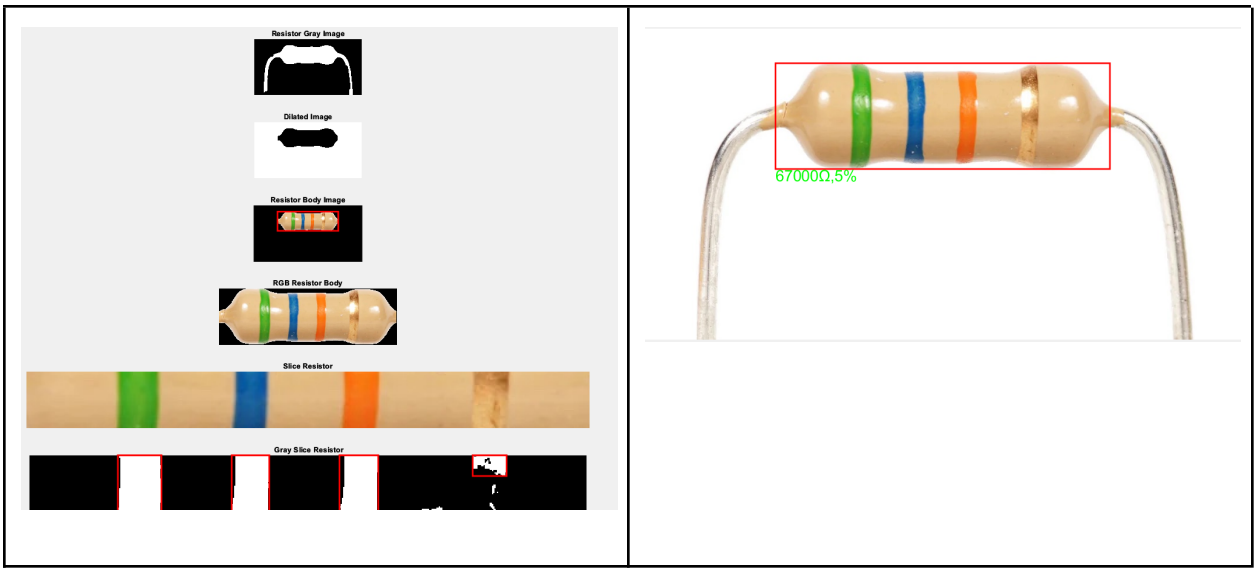

Figure 9: Result of Test Image 1

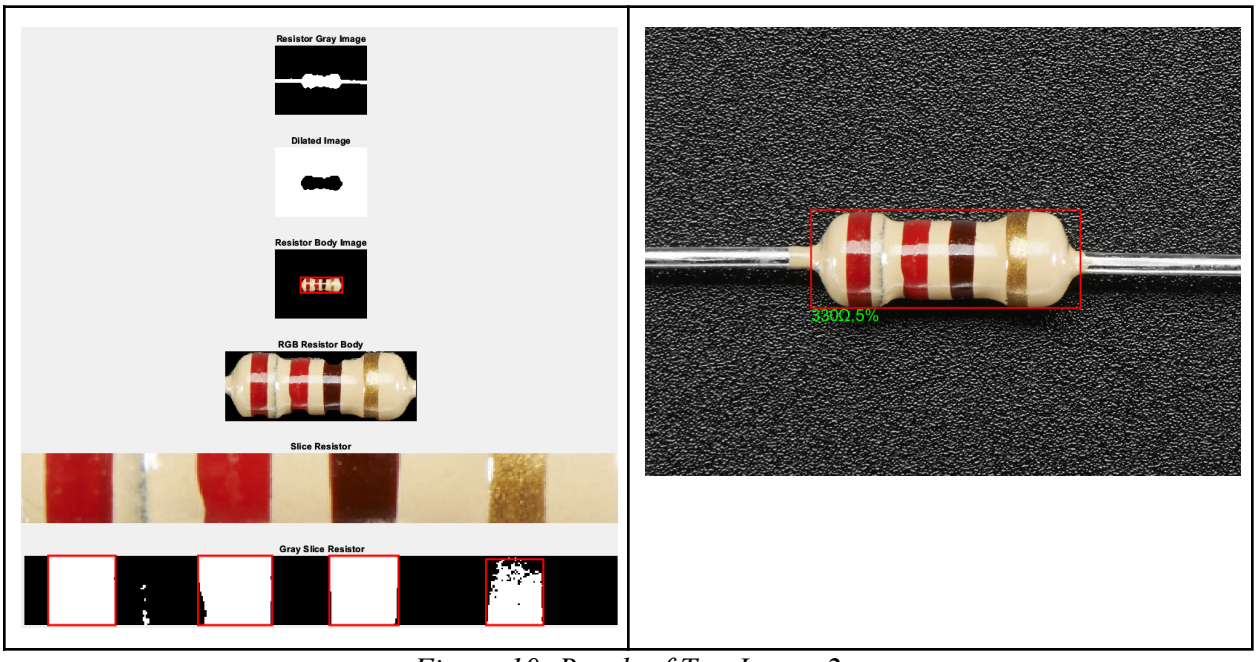

Figure 10: Result of Test Image 2

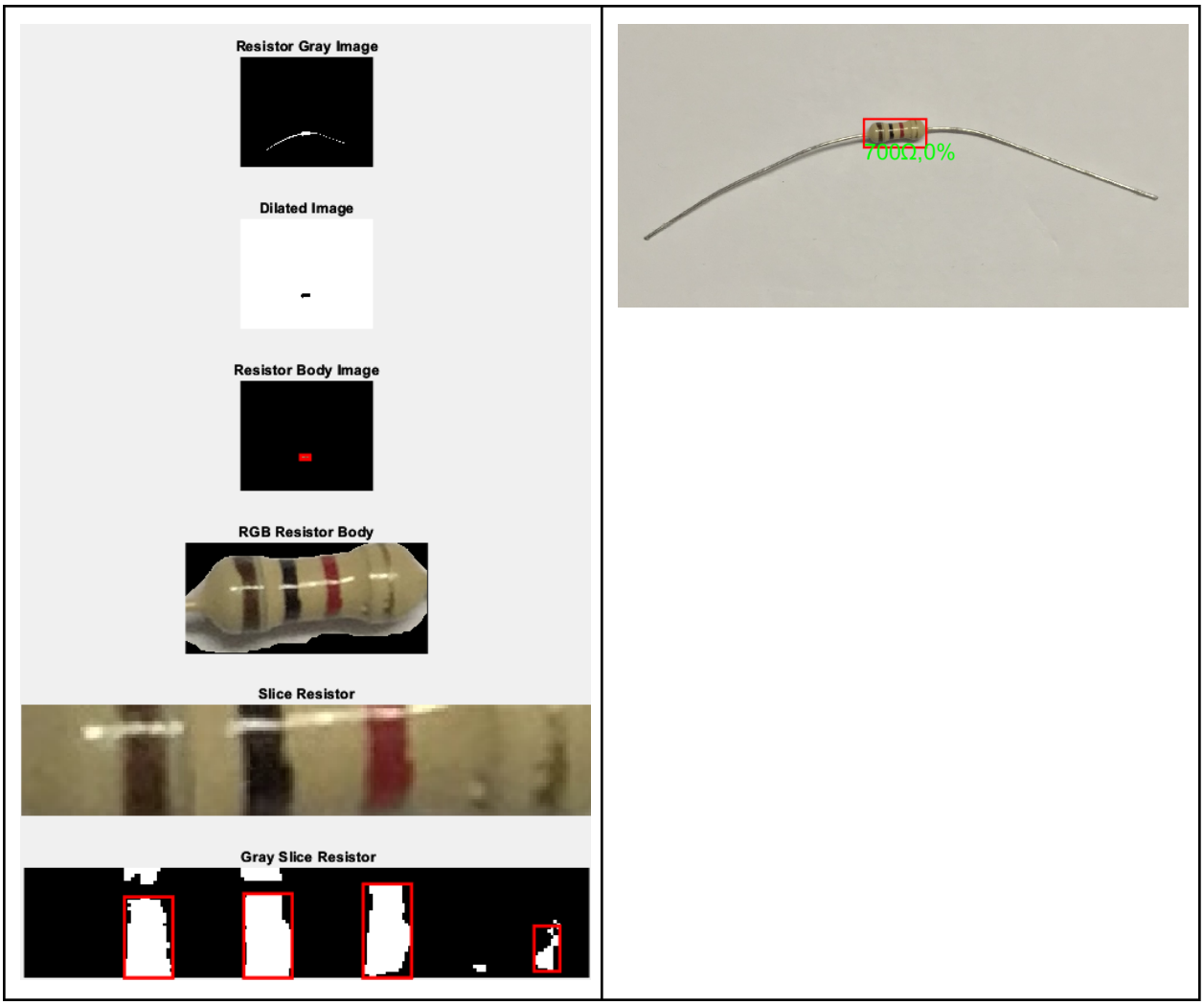

Figure 11: Result of Test Image 3

Figure 12: Result of Test Image 4

The results shown above display the successes and limitations of our algorithm. For resistor images similar to those in *Figure 9* and *Figure 10*, which are correctly oriented and have thick bands, we were able to easily detect and classify the resistor. For resistor images similar to those in *Figure 11* and *Figure 12*, which are correctly oriented, but are distant or have thin bands, we had issues with color extraction. It is important to note that for each image, threshold values for masks, dimensions of cropped images, and the "errorRange" value in our "hsvToIdx" function had to be varied for the best results. Even after further refining, the illumination creating glare, shadows, and the similarity of colors on the resistor in the image greatly impacted the performance of our classification algorithm. This is due to the fact that the HSV values for some of the colors are already very close, for example, brown, yellow, and gold's HSV values are very similar which makes it difficult to vary the "errorRange" value needed for differentiating the colors in the bands such as in *Figures 11* and *Figure 12*. The shadows and glares on the resistor also made it very difficult to segment the image to get the color bands. Overall, our implementation works for "easy" images but does not support difficult images like the ones shown above.

Originally, we were planning on only working with resistors with images with high-contrast backgrounds, but surprisingly this wasn't an issue when using edge-based segmentation. This segmentation method allowed us to identify the edges of the resistors in the image with more complex backgrounds, such as in *Figure 10*. This is because the algorithm relies on detecting changes in contrast, rather than color or texture. As a result, we were able to test our resistors on a wider range of images, which is important for real-world applications where images are not always perfectly controlled. This finding not only expands the scope of our

testing but also makes our results more applicable to real-world scenarios. We are excited about the potential impact of this approach and plan to continue exploring its use in future projects.

Conclusion

In conclusion, the use of computer vision technology can significantly improve the efficiency and accuracy of resistor value detection. As demonstrated by the papers discussed, computer vision techniques such as image segmentation, masking, thresholding, and machine learning can be used to identify the nominal resistance value of a resistor by analyzing its color bands. This is especially useful when dealing with a large number of resistors or when the task is monotonous, time-consuming, and prone to human error.

Our approach aims to leverage these techniques to create a program that can accurately identify and measure the value of resistors in images. While our assumptions limit the scope of our project to only four-band resistors and well-lit images, we believe that our program can be expanded to include more complex resistors and varied image qualities with additional development and training.

Our algorithm has shown promising results in accurately detecting and classifying resistors in images with constrained attributes. However, we have also identified the limitations of our algorithm, particularly in dealing with images that have distant or thin bands, and with illumination that creates glares and shadows. Our findings suggest that further refinements and improvements are needed to make the algorithm more robust and reliable. Nevertheless, the edge-based segmentation method we employed proved to be a valuable technique in expanding the scope of our testing and improving the applicability of our results to real-world scenarios.

Overall, computer vision has proven to be a powerful tool in the field of electrical engineering, and it is exciting to see how it can continue to improve the accuracy and efficiency of tasks like resistor value detection. As technology continues to advance, we can only imagine how computer vision can be further leveraged to revolutionize the field.

Teamwork

Eitan Klass

Every member did a significant amount of work and had a positive impact on the final project. I was in charge of creating the functions to handle the classification of the resistors as well as adding to the procedure for the final report.

Roey Mevorach

For this project I worked on image segmentation, morphological operations, and using the mask to extract the resistor from the original image. I wrote the sections for those topics on the report, as well as helping throughout the report. I also helped with the classification algorithm.

Brandon Pragasa

I worked on the resistor color and location extraction, compiling and labeling, as well as combining all the steps together for one cohesive algorithm. I also worked on the results, analysis, and conclusion of the report.

Nathan Jaggers

I worked on the introduction and background of the report as well as histogram creation for color and location extraction. I also worked on a rough draft of functions for compiling and labeling.

References

[1]M. Muminovic and E. Sokic, “Automatic Segmentation and Classification of Resistors in Digital Images,” *2019 XXVII International Conference on Information, Communication and Automation Technologies (ICAT)*, Oct. 2019, doi: https://doi.org/10.1109/icat47117.2019.8939034.

[2]X. Li, Z. Zeng, M. Chen, and S. Che, “A new method of resistor’s color rings detection based on machine vision,” *IEEE Xplore*, Oct. 01, 2017. https://ieeexplore.ieee.org/document/8242770/ (accessed Mar. 16, 2023).

[3]M. F. Demir, A. Cankirli, B. Karabatak, A. Yavariabdi, E. Mendi, and H. Kusetogullari, “Real-Time Resistor Color Code Recognition using Image Processing in Mobile Devices,” *2018 International Conference on Intelligent Systems (IS)*, Sep. 2018, doi: https://doi.org/10.1109/is.2018.8710533.

[4]“Resistors - learn.sparkfun.com,” *Sparkfun.com*, 2019. https://learn.sparkfun.com/tutorials/resistors/all

Appendix

```
%% Final Project - Resistor Recognition Algorithm
% Read in image
res1 = imread("easy1.jpg");
res_gray = rgb2gray(res1);

% Preprocessing smoothing for illumination minimization
res_gray = medfilt2(res_gray, [9 9]);

% Perform Edge-Based Segmentation
res_seg = edge_based_seg(res_gray);
subplot(611); imshow(res_seg); title('Resistor Gray Image');

% Dilate and Erode leads of resistors
se = strel("rectangle", [40 25]);
eroded_img = imerode(res_seg, se);
eroded_img = bwpropfilt(eroded_img, 'Area', 1, 'Largest');
dilated_img = ~imdilate(eroded_img, se);
subplot(612); imshow(dilated_img); title('Dilated Image');

% Mask out resistor body from background
res_body = imoverlay(res1, dilated_img, 'black');
subplot(613); imshow(res_body); title('Resistor Body Image');

% Get Bounding Box for resistor body
props = regionprops('table', ~dilated_img, 'BoundingBox');
box = props.BoundingBox(1,:);
subplot(613); rectangle('Position', [box(1), box(2), box(3), box(4)], 'EdgeColor',
'r', 'LineWidth', 2);

% Crop to only look at resistor body
new_res = imcrop(res_body, [box(1), box(2), box(3), box(4)]);
subplot(614); imshow(new_res); title('RGB Resistor Body')

% Get slice of resistor body for further processing
slice = imcrop(res_body,[box(1) + box(3)*0.1, box(2) + box(4)/2, box(3) -
box(3)*.2, box(4)/4]);
subplot(615); imshow(slice); title('Slice Resistor')

% Segment the slice for getting bounding boxes around bands
gray_slice = rgb2gray(slice);
mask2 = gray_slice < 160;
subplot(616); imshow(mask2); title(' Gray Slice Resistor')

% Initialize Feature Array
feature_arr = zeros(5,4);

% Obtain BoundingBox objects from image
props2 = regionprops('table', mask2, 'BoundingBox');
```

```
% Iterate through objects and filter
band_num = 1;
for i=1:size(props2, 1)
   box2 = props2.BoundingBox(i,:);
   % Use box area as threshold for detecting bands
   box2_area = box2(3) * box2(4);
   if (box2_area > 100)
       subplot(616); rectangle('Position', [box2(1), box2(2), box2(3), box2(4)],
'EdgeColor', 'r', 'LineWidth', 2);
       % Assign the start & end x-axis values to feature array
       feature_arr(4,band_num) = round(box2(1));
       feature_arr(5,band_num) = round(box2(1)+box2(3));
       band_num = band_num + 1;
   end
end

% Setup histograms to analyze HSV color space along the x-axis of image
% HSV Channels
figure;
hsv_slice = rgb2hsv(slice);
x = linspace(1, size(slice,2), size(slice,2));
h = hsv_slice(:,:,1);
h = mean(h);
s = hsv_slice(:,:,2);
s = mean(s);
v = hsv_slice(:,:,3);
v = mean(v);
subplot(311); plot(x,h); title('H-value')
subplot(312); plot(x,s); title('S-value')
subplot(313); plot(x,v); title('V-value')

% Extract Colors from hsv vectors using start/end x-axis values from boxes
for i=1:4
   feature_arr(1,i) = mean(h(feature_arr(4,i):feature_arr(5,i)));
   feature_arr(2,i) = mean(s(feature_arr(4,i):feature_arr(5,i)));
   feature_arr(3,i) = mean(v(feature_arr(4,i):feature_arr(5,i)));
end

% HSV values for the colors of a resistor
blackHSV = [0 0 0];
brownHSV = [0.0472 0.8296 0.3554];
redHSV = [0.4067 0.8393 0.6278];
orangeHSV = [0.0690 0.8502 0.9385];
yellowHSV = [0.17 1 1];
greenHSV = [0.2604 0.7459 0.6522];
blueHSV = [0.5037 0.6880 0.6035];
violetHSV = [0.83 0.5 1];
greyHSV = [0 0 0.5];
whiteHSV = [0 0 1];
goldHSV = [0.0898 0.5608 0.8463];
silverHSV = [0 0.75 0.75];

% Index values
blackIdx = 1;
```

```
brownIdx = 2;
redIdx = 3;
orangeIdx = 4;
yellowIdx = 5;
greenIdx = 6;
blueIdx = 7;
violetIdx = 8;
greyIdx = 9;
whiteIdx = 10;
goldIdx = 11;
silverIdx = 12;
colorHSV = [blackHSV; brownHSV; redHSV; orangeHSV; yellowHSV; greenHSV; blueHSV;
violetHSV; greyHSV; whiteHSV; goldHSV; silverHSV];

% Multiplier and Tolerance Matrices for determining resistor value
multiplier_matrix = [1, 10, 100, 1000, 10000, 100000, 1000000, 10000000, 100000000,
1000000000];
tol_matrix = [0, 1, 2, 0, 0, 0.5, 0.25, 0.1, 0.05, 0, 5, 10, 20];

% Vector containing the correct color index values
class_vector = hsvToIdx(feature_arr, colorHSV);

% Determines the ohm and tolerance value of the resistor based off class
% vector
[ohms, tolerance] = determine_res(class_vector, multiplier_matrix, tol_matrix);

% Displaying the resistor attributes and bounding box around resistor body
final_text = ohms + "," + tolerance;
figure;
imshow(res1), rectangle('Position', [box(1), box(2), box(3), box(4)], 'EdgeColor',
'r', 'LineWidth', 2);
text(box(1), box(2) + box(4) + 10, final_text, 'Color', 'g', 'FontSize', 20);
```

```
%% Function Appendix
function inRange = inrange(x, value, error)
  % Input: x - the value to check if it's within the range
  %        value - the central value of the range
  %        error - the percentage of error allowed from the central value
  % Output: inRange - a boolean value indicating if x is within the range
  % Check if x is within the range defined by value and error
  inRange = (x >= value - (value * error)) & (x <= value + (value * error));
end



function feat_arr = hsvToIdx(featVec, color)
 % The function takes in a feature vector and a color array as inputs and outputs
an array of color indices for each feature.
 % Input: featVec - matrix of HSV feature vectors with size (3, numBands)
 %        color - matrix of HSV values representing the different color classes
 %                with size (numColors, 3)
 % Output: feat_arr - array of color indices for each band

 % Initialize temporary array for color indices
 tmp_arr = zeros(1,4);
 % Set the allowed range for error
 errorRange = 0.25;
 % Determine the number of bands in the feature vector
 numBands = size(featVec, 2);
 % Determine the number of color classes
 numColors = size(color, 1);
 % Loop through each band and compare its HSV values to each color class
 for i = 1:numBands
     for j = 1:numColors
         % Select the j-th color class
         thisColor = color(j,:);
         % Check if the hue value is in range
         hueInRange = inrange(featVec(1, i), thisColor(1), errorRange);
         % Check if the saturation value is in range
         satInRange = inrange(featVec(2, i), thisColor(2), errorRange);
         % Check if the value (brightness) is in range
         valInRange = inrange(featVec(3, i), thisColor(3), errorRange);

         % If the HSV values are within the range of the current color class
         if (hueInRange && valInRange && satInRange)
             % Store the color class index in the temporary array
             tmp_arr(i) = j;
             break;
         end
     end
 end
 % Assign the temporary array to the output array
 feat_arr = tmp_arr;
end
```

```
function [res_val, tol] = determine_res(feat_vector, multiplier_matrix, tol_matrix)
  % This function takes in a feature vector, a multiplier matrix, and a tolerance
matrix and outputs a string representing the resistance value and tolerance of a
resistor.
  % Input: feat_vector - a 4-element vector containing the values of the colored
bands
  %        multiplier_matrix - a matrix containing the multiplier values for each
colored band
  %        tol_matrix - a matrix containing the tolerance values for each colored
band
  % Output: res_val - a string representation of the resistor value in ohms
  %         tol - a string representation of the tolerance value in percent
   % Calculate the resistance value in ohms
  % the first two bands represent the significant digits of the value
  ohms = feat_vector(1) * 10 + feat_vector(2);
  % the third band represents the multiplier for the value
  ohms = ohms * multiplier_matrix(feat_vector(3));
  % convert the value to a string
  res_val = int2str(ohms);
  % add the ohm symbol to the end of the value
  res_val = res_val + "Ω";
  % Calculate the tolerance value in percent
  % the fourth band represents the tolerance value
  tol = int2str(tol_matrix(feat_vector(4)));
  % add the percent symbol to the end of the tolerance value
  tol = tol + "%";
end


function object = edge_based_seg(gray_img)
   %% Edge-based segmentation using the Prewitt edge detector
   % Input: gray_img - grayscale image
   % Output: object - binary image containing the extracted object
   % Apply the Prewitt edge detection filter
   edge_img = edge(gray_img, 'prewitt');
   % Clean up the edge image
   clean_img = bwareaopen(edge_img, 20);
   % Convert the edge image to a binary image (if it's not binary already)
   if ~islogical(clean_img)
       bin_img = imbinarize(clean_img);
   else
       bin_img = clean_img;
   end
   % Close the gaps between the lines
   closed_img = imclose(bin_img, strel('disk', 25));
   % Fill image in
   fill_img = imfill(closed_img, 'holes');
   % Extract the object of interest
   object = bwpropfilt(fill_img, 'Area', 1, 'Largest');
end
```