ANKARA UNIVERSITY

FACULTY OF ENGINEERING

ELECTRIC-ELECTRONIC ENGINEERING DEPARTMENT

EEE 481 PROJECT REPORT

SMART TRAFFIC SYSTEM

FURKAN ÖZKAN 16290608

Süleyman Cevdet ÖZBEY 16290617

DR. ÖĞR.ÜYESİ GÖKHAN SOYSAL

Contents

INTRODUCTION

1.1 INTRODUCTION

Today, the traffic has become mixed with the proliferation of vehicles. The most effective way to regulate traffic is traffic police. However, it is not possible to have traffic police everywhere, so the traffic lights automatically regulate. Traffic lights now turn red, yellow and green color cycles for certain periods. The duration of this cycle varies according to the number of traffic and pedestrians. However, this period does not change into the day, depending on the region.

Therefore, we decided to work to adjust the time of the traffic lights change according to the traffic density. It is necessary to know the number of vehicles, the number of pedestrians, and the intensity of the traffic lights to adjust the changeover times.

We decided to move forward in this project based on the definition of the number of vehicles, we will work on finding the number of vehicles with image processing. Instead of identifying directly on the carts for image processing, we will first work on circles, rectangles and triangles, simplifying the problem. For example, flats cars, squares trucks, triangles will be considered in this direction as a motorcycle will progress. Because of the variety of tools, we have also put certain limitations on simple shapes. For example, we will introduce a red object constraint, which means that we will eliminate objects other than red or near red colors using filters.

Our analyzes on the images in the project, will be done in **MATLAB** environment.

1.2 MATLAB

MATLAB is a multi-paradigm numerical computing software and a fourth-generation programming language. Developed by MathWorks. Matlab allows matrix operations, function and data drawing, algorithm development, user interface, as well as working with programs written in other languages such as C, C ++ and Java.

MATLAB was first accepted by researchers and developers in control engineering. In addition, linear algebra is a popular language for teaching numerical analysis and image processing. With MATLAB you can call functions in the C programming language or Fortran and write subprograms.

MATLAB performs many mathematical calculations effectively and quickly, such as linear algebra, statistics, optimization, numerical analysis, optimization, and fourier analysis.

The MATLAB programming language is also used for 2D and 3D graphics drawing.

With MATLAB, even very complex mathematical calculations are completed in a short time.

Two and three dimensional graphs of basic mathematical functions can be drawn with MATLAB. You can easily draw any kind of two and three dimensional mathematical graph MATLAB, especially polynomials, parabolas, sine, cosine waves.

1.3 IMAGE PROCESSING

People need to transfer images to a computer environment to make things easier, or for human eyes to find it difficult to perceive.

To describe image processing simply: It is a method that can be identified with different techniques in order to obtain useful information according to the related needs through the images that have been digitized. The image processing method is used to alienate or improve certain recorded images by manipulating and modifying existing images and graphics. For example, it is possible to see the quality degradation experienced when scanning photos and documents to digital media. It is during this quality reduction that image processing comes into play. We use the image processing method to minimize the degraded image quality and visual distortions. This and many other places that can be used in image processing is among the rapidly developing technologies.

Objectives of image processing:

• Image enhancement

• Image repair

• Image compression

• Image analysis

• Image recognition

So where is image processing used;

- Face recognition and security systems.

- Demographic information analysis.

- Traffic, astronomy, radar and photo industry applications.

- Determination of population density by using application images.

- Radiology field (Tomography, Ultrasound, etc.).

- Understanding of both underwater and satellite images in the field of Military Industry.

- The field of medicine to distinguish various organ structures.

We can apply image processing in 3 steps;

• Transfer an image to a computer: At first, we need to transfer the image to the computer. There are various applications for transferring the image to a computer, such as MATLAB, Phtyon, C ++ and so on. These are applications. In addition, we can transfer real-time images to the computer.

• Analyze the image: After transferring the image to the computer, we can do the operations we want. For example, we can examine the alignment of the face on which we transfer a face picture, or we can identify the model, license plate, color of a car that violates the rule.

• Output: After making the analysis we want from the last image we can print and use. If this is a real-time image, we can quickly process and print it out on the computer.

General concepts in image processing;

Pixel: It is the combination of the words “Picture element, it is the unit element of the image.

Brightness: Shows the brightness value of a pixel in the x and y coordinates.

Discrimination: In order to be able to express the analog image in the digital system, it must first be divided into a finite number of spatial dimensions (sampling, sampling), and then the analog brightness value in each part must be expressed in one of a number of discrete digital levels (quantization, quantization).

Resolution: The number of pixels per inch or cm. (1 inch. = 2.54 cm) Indicates how many pixels the image is divided, ie how many pixels it is represented. The higher the resolution, the higher the frequency the sample is sampled, and the details in the image become clearer.

Spatial Frequencies: Spatial frequencies are the frequency of change in brightness value over a distance.

Input image: is considered a two-dimensional, MxN-length matrix, and the pixel value (1,1) in the upper left corner is numbered as the starting point.

Spatial Frequencies (fx, fy): Typically, the sampling period is assumed to be 1 pixel and spatial frequencies and periods are expressed in pixels.

Image Types:

Binary image; It consists of “0” and “1” s. These values correspond to “black” and “white. 1 pixel, 1-bit footprint

Gray scale; It takes values between “0-255. It contains shades of gray. It gets black as it gets closer to 0 and it gets whitened as it gets closer to 1. 8 pixels are usually reserved for 1 pixel.

Color Image; It is a picture of shades of red, green and blue. 8 bits are allocated for 1 pixel.

2 GENERAL INFORMATIONS

2.1 PROJECT OBJECTIVE

Count the number of vehicles on the roads with the camera at traffic intersections and change the green light setting. In this way, the green light time will be very short on empty roads and the number of vehicles will be much more green light time. In this way, the traffic density and fuel waste and environmental pollution caused by continuous vehicles stop and go will be reduced. In the first phase of the project, we can provide some convenience in distinguishing between circle, circle, rectangle and triangles. For example, when the correct color filters are applied by distinguishing circle-like shapes, an X-ray result can be used to determine whether tumor-like objects are applied. Or, if it is known in the field of technology that capacitors are rectangular in the card of any device, it can be verified by checking the top images of the cards produced by image processing to check if the rectangular shape is below a certain number, and that it is a manufacturing error.

2.2 PROGRESS STEPS OF THE PROJECT

Starting from the foundation for intelligent traffic control, we will first try to distinguish the shapes of circles, rectangles and triangles from a picture. In this section, we will briefly talk about what we do in the steps. In the first step, we will transfer the picture to the matlab environment as we explained the features of matlab. As we mentioned, the image will look like a matrix in a matlab environment. By applying color filters in the matrix of the image, we will try to distinguish the desired color, for example, red from other colors. To do this, we will apply various operations to the image. In the second step, to handle the objects in the image separately, we assign each object to a variable that we store and apply the operations we want to each object in the image using a loop. In the next step we will perform the operations in the loop to extract the characteristics of the shape of the objects, such as the center of the object, the edge points, the length of the longest line passing through the center, the area, the ratio of the area of the object to the area of the surrounding box. Each step will be described under headings.

3 USING MATLAB

In this section we will talk about how we transfer the image to the printer and the image formats.

3.1 IMAGE UPLOAD

To upload an image to Matlab, we first need the directory and name where an image is stored. To find out, you can right-click on the image to find the directory and name it is saved in. Then in the matlab, command windowa image = imread ('Directory \ imagename.format'); By typing the command, we transfer the image to the matlab environment. Here, the imread function defines the desired image file from the specified directory for matlab. The image is transferred to matlab as a matrix format is uint8.

When we transfer the image and then transfer it to the 'image' variable to do other operations, in the workspace, the window where the workspace processed variables appear appears as shown.

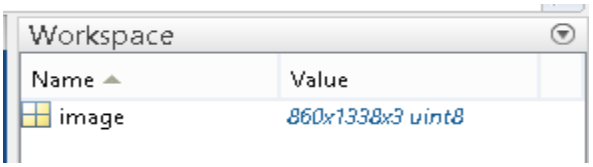


Figure 3.1

The uint8 format we see here consists of three variables, which we call row (x), column (y), and "RGB". For example, the image in figure 3.1 is a matrix with values of 860x1338x3. Here, '860' represents the number of lines, 38 1338 'represents the number of stuns, and 3 represents the color space, i.e. "RGB" (red, green, blue).

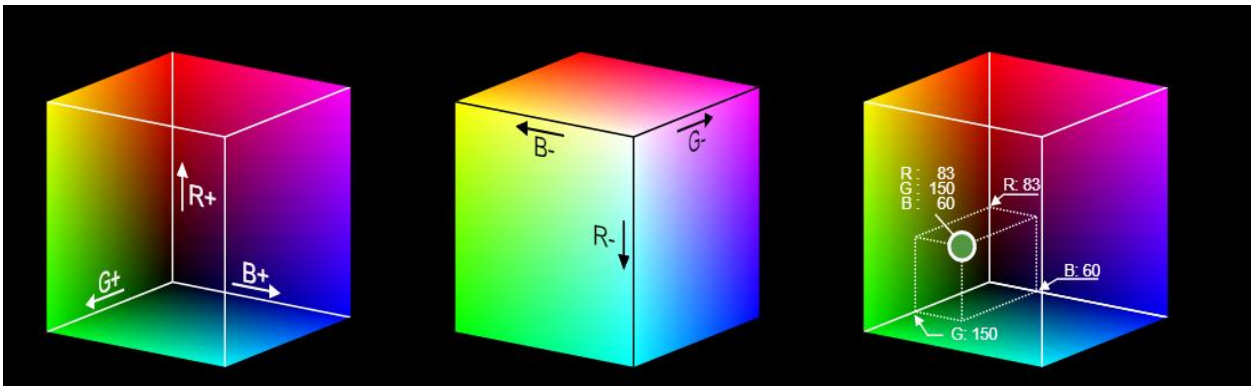
4 COLOR FILTERS RED OBJECT RECOGNATION

In the previous section, we talked about loading the image on the printer. In this section, we will focus on color separation, which is one of the parameters we will use to distinguish objects. Later in the project, we will match objects of different shapes and colors with vehicles of different sizes. As a result, in this section, we will talk about the filters and methods we apply for color separation and recognition of the separated color.

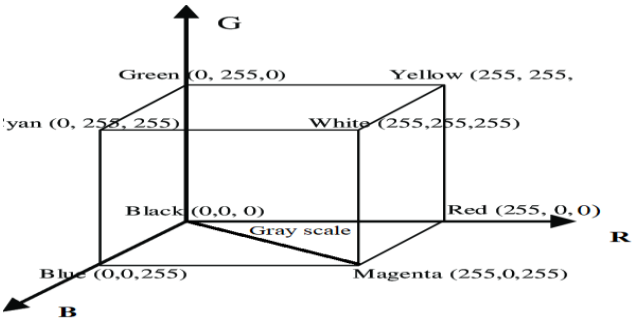
4.1 COLOR FILTERS

As mentioned in the image upload section, we can upload images to the printer in uint8-like formats. Talking about color space and formats in more detail in this section will make it easier for us to understand color filters.

There are 3 basic colors in Matlab color space (RGB). These are respectively “Red, Green, Blue” and “Red, Green, Blue”. The RGB image consists of a 3D matrix. Each color is represented by 8 bits. 1 pixel is represented by a total of 24 color bits. Thus, (2 ^ 8) ^ 3 = 16,777,216 different colors can be formed. RGB color space is like a Cartesian coordinate system. The RGB picture can be “uint8”, “uint16” or “double”. As can be seen from the following image, any pixel takes a value from inside the cube in this coordinate system.



You can see where the points like black, white, red, blue and green coincide from the coordinate system. In addition, each diagonal is gray scale, i.e. it contains shades of gray.



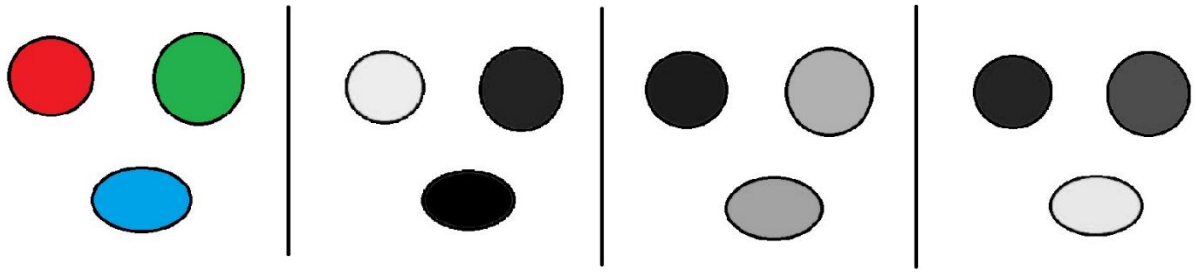
The closer the image in the image variable is to the filtered color, the more white it appears, the more black it appears to be, the farther away the filtered color is

To apply a color filter to an image, we apply the following commands;

imshow (image (:,:, 1)) for the red color filter,

imshow (image (:,:, 2)) for green color filter,

imshow (image (:,:, 3)) for the blue color filter,



Resmin aslı Kırmızı filtre Yeşil filtre Mavi filtre

As can be seen, the red colored object is closer to the white when the red filter is applied, the green colored object is closer to the white when the green filter is applied, and the blue object is closer to the white when the blue filter is applied than the other colored objects. After applying these color filters, we will explain why we will need the gray format of the image to get a clearer image.

We use 'rgb2gray' function to convert the picture to gray format. Before we know what the "rgb2gray yar command is, let's say "gray". In Matlab, RGB color space is a 3-dimensional matrix and a grayscale vector. Grayscale can range from 0 (black) to 255 (white). Values closer to 0 are black, and they become whitening when approaching 255. The "rgb2gray" command provides the transition from the "RGB" color space to the grayscale space.

RGB renk uzayı Grayscale uzayı

We got the gray image and the red filter. The red-filtered parts of the picture are red filtered, with values closer to 1 (white) in the matrix, others closer to 0 (black). In our gray image, the red areas are closer to 0 (black) than the other colored areas. In this context, if we remove the gray image from our red-filtered image, the regions that are red in the original will be close to white, while the other colored parts will be close to 0.

For example, the blue parts appear in gray tones when the red filter is applied, and if the gray tones are removed from this image, the blue color approaches 0 (black) as it will be of moderate value in the grayscale image.

red_filter_imag A = imsubtract (image (:,:, 1), rgb2gray (image));

The imsubtract command is used to extract images from Matlab. It is the subtraction of the values in the matrices of 2 separate images (red filtered image and gray image for us).

Example: The difference of two 3x3 matrices:

```
A =                     B =

     1     1     1         1     1     1
     2     2     2         1     1     1
     3     3     3         1     1     1
```

A-B= [1 1 1;2 2 2;3 3 3]-[1 1 1;1 1 1;1 1 1]= [0 0 0;1 1 1;2 2 2]

```
0     0     0
1     1     1
2     2     2
```

Similarly, when we remove our gray picture from the red filtered image, the red parts will be closer to white and the other parts will be closer to black. We assign the output to the variable "red_filter_image" to process it.

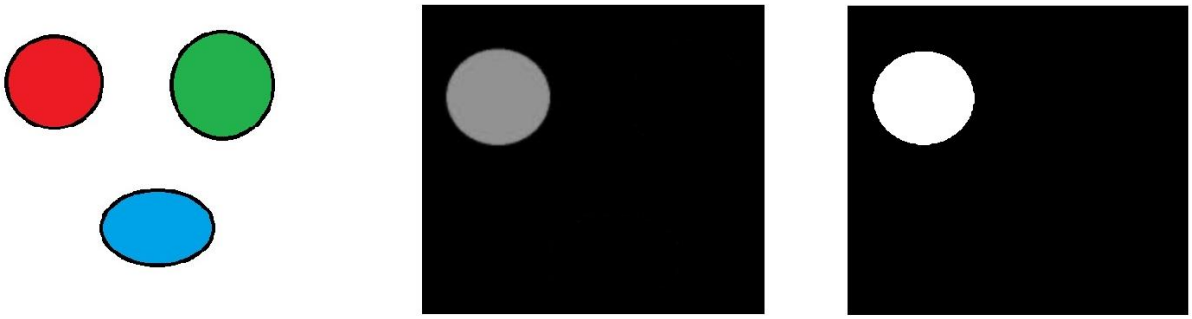
Our picture in “red_filter_image olabilir may contain noise and unclear areas. For example, if the image has small stains or fraying, we use image enhancement methods to correct it. The “medfilt2” command is one of these methods. Here, using f medfilt tamamla to complement the color filter ensures better results. We will talk about the median filter after the color filter is finished.

med_filter_image = medfilt2 (red_filter_image, [3 3]); By converting the median filter image to black and white, we can clearly distinguish between red and other colors. Because the red colors converge to white while others have applied our filter to converge to black. To change an image to black and white:

bw_imag A = im2bw (med_filter_imag A);

command.

In Matlab, we learned “RGB” color space and “grayscale” space. There is also a black-and-white space and a binary system. There are no intermediate colors in the black-white space “0 = black, 1 = white. Iz We cannot switch directly from the “RGB” color space to the“ black-white ”space, so we first took our colored image to the ays grayscale” space. When moving from the ays Grayscale space to the-black-white” space, values close to white are taken as values that are close to white black. The reason for our transition to the-black-white ”space is that it makes it easier for us to reduce the objects in the image to black and white while scanning the image, so that the red objects become white and the other colored objects become black. In the following image, we get an example of “red_filter_image” variable that we apply f medfilt ”and geçiş im2bw” command to switch to geçiş black-white ”space.



RGB renk uzayı **Kırmızı filtre** **Black-white uzayı**

4.2 MEDIAN FILTER

The median filter is a filter commonly used in image processing. Images contain unwanted signals called noise. Noise is the effect of external sources that pollute the image and reduce the image quality. Noise can occur from many different sources within an image. Thanks to filtering methods, images can be made free of noise and made desired.

As mentioned, the median filter is one of the most preferred algorithms and is not a linear process.

The median filters are nonlinear spatial filters. This is the process of assigning the middle value to the central pixel by sorting the image pixel values in the size that make up the mask. The media filter is a low pass filter.

The general working logic of Mean filters is; It is processed by trying to find an average value in a matrix window of NxN. In doing so, they use a floating window method that scrolls through the image. The median filter is the simplest of the mean filters. It takes the arithmetic mean of the values in the window and thus removes large jumps. After applying the filter; The pixels that are

separated from their positions are detected and cleaned. It softens local changes within the image. We will then provide a more detailed description of the median filtering.

The main reason for our application of the median filter is to reduce noise in the picture and to find more accurate results. We will show more detailed information and application examples for the median filter in the functions section.

5 LOOP FOR ANALYZE EVERY OBJECT SEPARATELY

In the previous topic, we explained how to filter, convert image to binary mode and apply a median filter. In this section, we will explain the main logic of doing each operation for each object from the bwconncomp () function and loop which we will use to analyze the objects separately.

To analyze the objects in the black and white image, it is more useful to handle each object separately to avoid confusion.

5.1 HOW CAN WE ANALYZE OBJECTS SEPARATELY?

Using the "bwconncomp (bw, conn)" function to handle the objects in the picture separately, we can learn the dimensions of the picture by adjusting the number of objects in the picture, how many pixels are counted when interacting with each other. In addition, by enumerating the objects in the picture, we can make it easier to process the object we want.

bwconncomp (bw, conn); Here is the bw function in parentheses. Connectivity is the number of interconnected pixels. With this function we can define all objects independently. The objects we define are numbered starting from 1. For example, if we assign bwconncomp (img, connectivity) to a variable such as 'cc', 'cc bulunur also has parameters named Connectivity, ImageSize, NumObjects, and PixelIdxList. Here we specify Connectivity within the bwconncomp (img, connectivity) function. For example, for bwconncomp (img, 4), connectivity is 4 and takes 4 side by side as a pixel object with defined values, if there are more composite pixels, it takes all of them as a single object. The ImageSize command holds the dimensions of the img image. NumObjects holds the number of objects in the image. PixelIdxList allows us to use a numbered object.

Take, for example, a picture that has been converted to black and white with a color filter.

Figure 5.1

cc = bwconncomp (bw, 4); With the command, the values found by the bwconncomp (bw, 4) function are stored in the variable ‘cc.. When you type cc from the command window and look at its output:

values. Here, Connectivity is the 4 value we write into the function. If 4 pixels side by side are 1, they are defined as objects, and all 1 values associated with those 4 pixels are included in the object. ImageSize holds the dimensions of the bw image. In this way, when we call any object alone, it allows us to analyze the object by ignoring the other objects and placing them in their original position. For example, consider the bwconncomp function with a connectivity value of 6 on another object.

NumObject counts the number of objects in the picture and we find the value 12. PixelIdList keeps the numbering of 12 objects one by one. For example, objects begin to be numbered by sliding from the top left to the right. Each object has a number. We use these numbers to call objects one by one. If we want to call any object by itself, we define a variable as empty in the original size of the image and take the empty state as false.

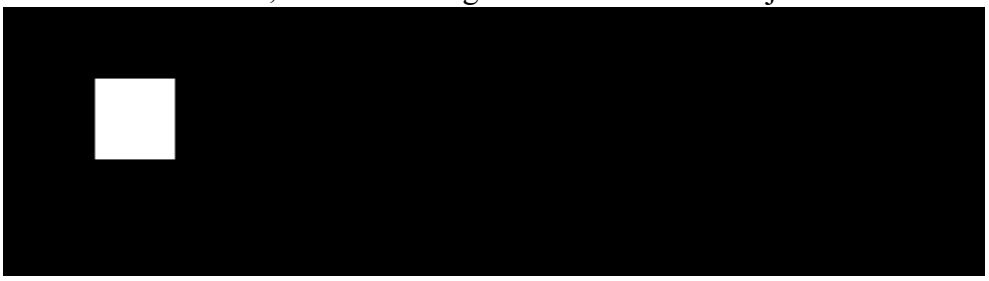
grain = false (size (bw));

Then we set this variable to true when we call the object with the desired number, and set it to true in the grain variable.

grain (cc.PixelIdxList {o}) = true;

Here, instead of the ‘o in parentheses, we can write the desired number and assign the desired object to the grain.

For ‘o’ = 1, we assign the first object to the grain function



imshow (grain); When we look into grain with imshow command, we see the first object.

Here we will explain the bwconncomp () function in chapter 9 in more detail.

After learning how to call objects one by one with this method, it is easier for us to perform separate operations for each object using a loop. For example, for an image with 12 objects, process over the value of ‘o ranging from 1 to 12 and use grain (cc.PixelIdxList {o}) = true; If we place it in the loop, we define the grain variable for each object in the loop and we can process and measure it.

6 FINDING PARAMETERS FOR EVERY OBJECTS

We have learned to use the loop and ‘bwconncomp () function to separately analyze the objects of the combined pixels in the image. In this section, we will create indexes for comparison with various operations using the loop we created to obtain information about the shape of each object. This information is the length of the longest and shortest line passing through the center, the centers of the objects, edge detector, corner points, radius, diameter, area, surrounding box, the ratio of the area of the object to the area surrounding the object and the circularity.

Here, we will take advantage of the regionprops command for some measurements, and we will do some mathematical measurements. The regionprops function and the working principles of its sub-features will be explained in more detail in chapter 9.

6.1 REGION PROPS

The Regionprops function is a function that determines the desired properties of each defined point in the black and white image. Usage:

stats = regionprops (bw 'boundingbox', 'centroid', 'majoraxislength', 'minoraxislength', 'Area', 'Extent');

Here we write the bprop function inside the regionprops function. Other values are the information we want. Details will be given in section 9 to avoid slowing down the narrative of the project.

6.2 CENTROID

They are the central points of objects. There are center points of each object in the loop by examining each object separately. Found points “bc = stats (o) .Centroid”; for object 1, instead of ‘o 1, we assign the coordinates of the first object's center point to the variable bc. In short, the finding of the center point is the midpoint of the length of the longest defined line (having a value of 1) of the object in mathematically uniform ways. It is the intersection point of the diagonal in objects such as triangles.

stats = regionprops (bw, 'Centroid');

For example, finding the centers of objects in an input image with different shapes:

image = imread ('C: \ Users \ turhal \ Desktop \ adddd.png');

imshow (image);

```
diff_im = imsubtract(image(:,:,1),rgb2gray(image));
diff_im = medfilt2(diff_im, [3 3]);
diff_im = im2bw(diff_im,0.18);
diff_im = bwareaopen(diff_im,300);
bw = bwlabel(diff_im, 8);
stats=regionprops(bw, 'Centroid');
```

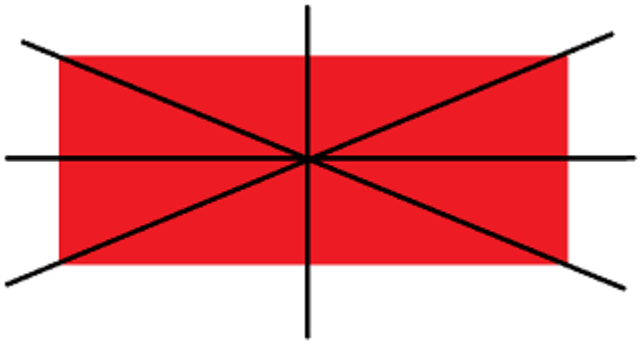
stats.Centroid % gives the center coordinates of each object. Stores information in a 4x1 matrix for 4 objects. Further details will be given in Chapter 9.

6.3 MAJOR-MINOR LENGTH

With the prop Regionprops () ”function, we can find the lengths of the longest and shortest line passing through the center of the objects. stats = regionprops (bw 'majoraxislength', 'minoraxislength');

Figure 6.3

For example, let's find the major and minor lengths of the rectangle in the figure. First we need to find the center point.



The center point and some lines passing through the center are shown in the figure above. Major and minor lengths are the lengths that intersect two linear edges passing through the center of the object. Here the major length equals the diagonal of the rectangle and the minor length equals the short edge. For a circle, these lengths are equal to the diameter.

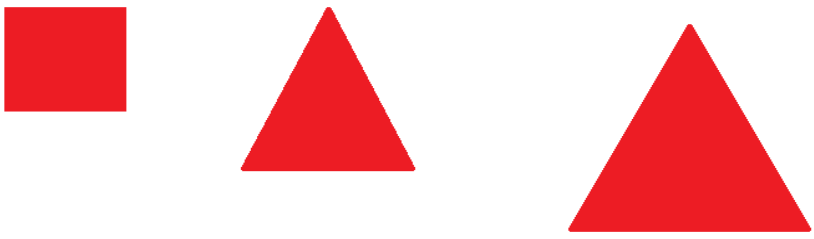
6.4 AREA

The presence of areas of objects. For area finding, it is sufficient to find the number of all defined pixels (having a value of 1) of the object. The scanning method can be used for this. The scanning method is briefly checking each pixel one by one, looking at the value of all pixels starting at the top left, increasing the field variable starting from 0 for each '1' value to find the area. More detailed information will be given in Chapter 9.

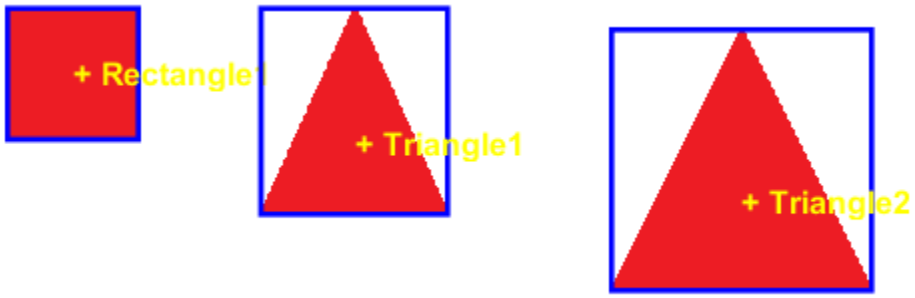
6.5 BOUNDING BOX

It is the narrowest rectangle that covers the end points of the object. For this purpose, the outer corner points of the object are considered as the end point and parallel lines are drawn to create the smallest rectangle covering the object. Thanks to the Bounding box, we have an idea about the area of movement of the objects besides the space they occupy.

Example:



Bounding box of objecst:



More information on the availability of the Bounding Box will be described in chapter 9.

6.6 EXTENT

The extent is the ratio of the area of the object to the inner area of the boundingbox rectangle surrounding the object. This ratio is 1 for a rectangle or square. For triangles and circles, values are less than 1.

As we have seen in the example above, the bounding box for a rectangle or square matches exactly the area of the object. For convex objects such as triangles, this ratio is less than 1.

6.7 RADIUS-DIAMETER

We will try to calculate the radius and the diameter of the objects. For this circular shape, the radius and diameter parameters help us. For example, circular objects are connected to the radius.

To calculate diameter we're not using regionprops () function but we are using major and minor axis lengths. d = mean ([stats (o) .MajorAxisLength stats (o). MinorAxisLength], 2); We are finding the diameter of the circle. From the diameter of half the diameter.

6.8 Circularity

Circularity is a unitless number that gives us information about the circularity of the object. The closer the Circularity value is to 1, the more circular the object is. The calculated radius value of the object is used to calculate the Circularity. Circularity value:

(2 * π * r) ^ 2 / (4 * π * area). We know that the area for circular shapes is πr ^ 2. When we replace this field with circular values, circularity goes to 1.

6.9 EDGE DETECTOR-CORNER POINTS

Edge detection is an image processing technique to find the boundaries of objects in images. It works by detecting discontinuities in brightness. Used for image segmentation and data extraction in areas such as edge detection, image processing, computer vision and machine vision. Using the edge detector, we can find the boundary points of the objects in the picture, thus reducing the

processing load. Once the edges are found, we obtain the coordinates of the corner points with the corner () function.

For example, if we use the following image as input:

diff_im = imsubtract (image (:,:, 1), rgb2gray (image));

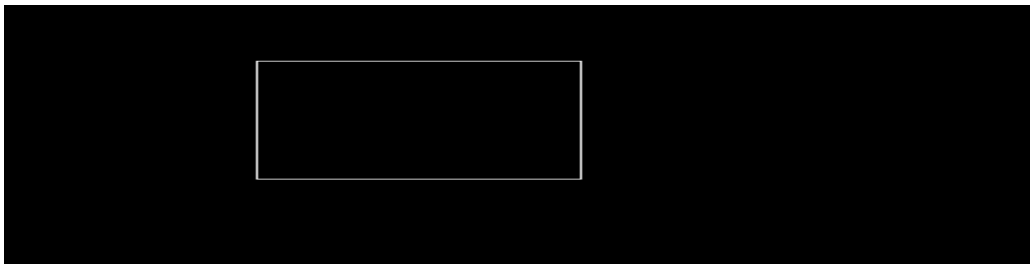
diff_im = medfilt2 (diff_im, [3 3]);

diff_im = im2bw (diff_im, 0.18);

diff_im = bwareaopen (diff_im, 300);

bw = bwlabel (diff_im, 8); Here we applied the process of converting the picture to black and white. Then, when we apply the Roberts filter to the black and white image:

BW1 = edge (bw 'Roberts');



Common edge detection algorithms include Sobel, Canny, Prewitt, Roberts and fuzzy logic methods. In this study, we will use Roberts edge detection method. We will give more detailed information about edge detectors and the corner () function in chapter 9.

7 SHAPE CLASSIFICATION

In this section we will explain how we find the shape of the objects. We find shapes such as circles, rectangles and triangles for red, green, blue colors. We will use some functions (major-minor axis, centroid, area etc.) to find these. First of all, we start the code with the for loop to distinguish objects from color to color. This cycle will rotate 3 times because the "RGB" space consists of 3 primary colors. Then we find out whether the object is a triangle, a rectangle or a circle with the condition functions we use.

7.1 FINDING RECTANGLES

To find a rectangle, we first need corner points. We found these corner points by using detector edge detector function and assigned their coordinates to variables like" ver1, ver2, ver3, ver4."

These variables are 1 * 2 vectors. The first column represents the x coordinate and the second column represents the y coordinate. We assign the edge coordinates to the condition function as follows; if (ver1 (1) == ver2 (1)) && (ver3 (1) == ver4 (1)). We can explain the condition here if the first column of the ver1 vector and the first column of the ver2 vector are equal and the first column of the ver3 vector and the first column of the ver4 vector are equal. This means that if the x coordinates of the first and second edges and then the third and fourth edges are equal. When this condition is met, we can call it a rectangle.

For an even more precise result, we can add the following to the condition function; "(0.99 <= A) && (A <= 1.05)." The "A" here is the variable we mentioned in 6.6. Summarizing again, the variable "A oran is the ratio of the area of the rectangle we draw around the object to the area of our object. When we think about the rectangle, "A = 1" should be. However, since there are minor disturbances, we evaluate the "A" variable between 0.99 and 1.05.

When all the conditions are met, we increase our variable "recshape once". The reason we chose recshape as a name is because MATLAB has a function called rectangle.

7.2 FINDING CIRCLES

No corner points are required to find a circle. We need "circularity" and "extend" functions. We mentioned the "Circularity" function in 6.8. In short, it is a comparison of the field's field with the circle formula, and the closer this result is to 1, the more circular the object is. "Circularity" and "extend" functions are as follows; if (0.98 <= circularity) && (circularity <= 1.05) && (0.75 <A) && (A <0.80). Since there may be distortions in the circle on the picture, we take the "circularity" function within a certain range. The ratio of the rectangle we draw outside the circle corresponds to approximately 0.78 and we value between 0.75 and 0.80. If these conditions are met, we enter into the condition.

We will use the imfindcircles function to find the circle. Let us briefly explain the command f imfindcircles; The values of Rmin and Rmax indicate the minimum and maximum diameter range of the circles we want to look in the picture. The imfindcircles function locates the centers of the objects in the image and starts scanning pixels by pixel. Because we write 'bright içine in the function, it works on white places. Pixel value 1 continues quite long, but only until it reaches 0. When it reaches 0, it calculates how many pixels it reaches 0 in other places, and indicates that if these values are equal to each other, it is a circle. We will explain the "Imfindcircles" function in more detail in 9.7.

Once we find our circle, we draw with viscircles. The Viscircles function allows you to mark the circles we find with imfindcircles. We can draw the perimeter of the circle in the desired color and the way we want (straight line, dashed line, etc.). Then we increase the "circshape" variable 1 time.

7.3 FINDING TRIANGLE

The functions we use to find triangles are as follows; "Edge detector", "circularity", "extend". We write them with the condition function as follows; if (ver3 (2) == ver2 (2)) && ((A> 0.5) && (A <0.99) || (A> 1.05)) && (circularity> 1.1) || (Circularity <0.99). First of all, we check whether the base edge coordinates are equal, if it is equal, we look at the "extend" function, this ratio should

be between 0.5 and 0.99 or greater than 1.05. Cular Circularity ”should be greater than 1.1. If the conditions are met, we increase our ish trishape ”variable once.

8. SOME IMPORTANT FUNCTIONS

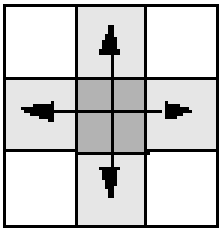
8.1 bwconncomp ()

We have briefly mentioned the “bwconncomp” function in 5.1. The main purpose of the function is to find out how many objects are on the image after applying the filter. The function is implemented in 2-D and 3-D images and the data types are as follows; single, double, int8, int16, int32, int64 and logical. We'll do my review on 2-D pictures. If we talk about working logic, it examines pixels with a value of 1 on 2D images. When it encounters a pixel with a value of 1, it scans around the pixel according to which value we entered in the connectivity parameter. If there is a collection of pixels that we enter, it accepts it as an object and enters this value into the NumObjects vector.

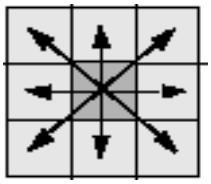
We can see how it scans according to the connectivity value entered in 2-D and 3-D images with the following definitions.

For two-dimensional images

4-way connection; After seeing a pixel with a value of 1, it looks up, down, and sides, and identifies it as an object if there are pixels with a value of 1.

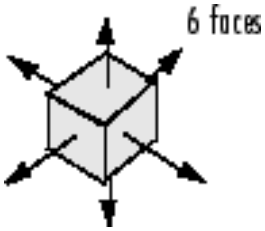


8-way connection; After seeing the pixel of value 1, it examines horizontally, vertically and diagonally.

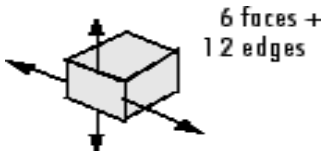


For three-dimensional images

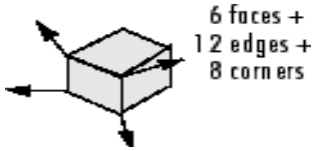
6-way connection; The unit examines the surfaces of the cube and identifies them as objects if there are other connections.



18-way connection; The unit scans across the surfaces and corners of the cube and identifies them as objects if there are other connections.



26 connection; The unit scans along the surfaces, edges and corners of the cube and identifies it as an object if there are other connections.



We examined how the function scans. Now let's examine the parameters of the function;

Connectivity; It is the parameter that we assign the number of connections we want. If we do not type anything, it automatically assigns 8 for two-dimensional images and 26 for three-dimensional images.

ImageSize; This is the parameter we assign the size of our BW format image.

NumObjects; Records how many linked objects are available after connection. We take the number of objects from this parameter.

PixelIdxList; Creates a vector with a value equal to the size of each object in it. We perform our operations by taking value from this parameter.

For example;

BW = imread ('example.png');

imshow (BW)

cc = bwconncomp (BW, 4)

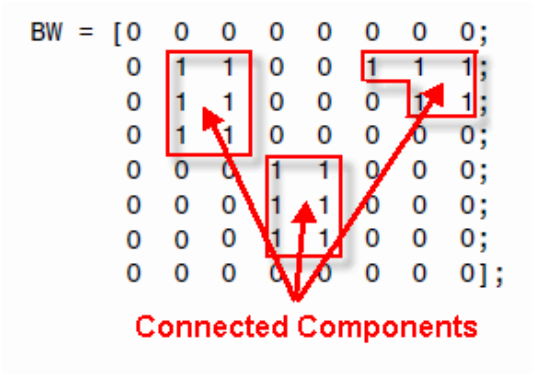
cc =

Connectivity: 4

ImageSize: [8 8]

NumObjects: 3

PixelIdxList: {1x3 cell}



8.2 regionprops()

In this part we will talk about regionprops() function, we used regionprops() at black white image to analyze regional properties of objects. As we mentioned images at matlab is defined as matrix. Regionprops() using this matrix and give us some informations.

STATS = regionprops(L,properties) measures a set of properties for each labeled region in the label matrix L. Positive integer elements of L correspond to different regions. For example, the set of elements of L equal to 1 corresponds to region 1; the set of elements of L equal to 2 corresponds to region 2; and so on. The return value STATS is a structure array of length max(L(:)). The fields of the structure array denote different measurements for each region, as specified by properties. We can get different properties. There are properties using with regionprops() :

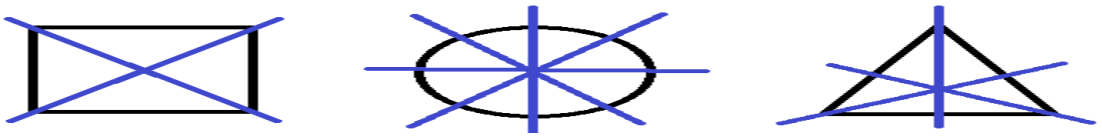
'Area'	'EulerNumber'	'Orientation'
'BoundingBox'	'Extent'	'Perimeter'
'Centroid'	'Extrema'	'PixelIdxList'
'ConvexArea'	'FilledArea'	'PixelList'
'ConvexHull'	'FilledImage'	'Solidity'
'ConvexImage'	'Image'	'SubarrayIdx'
'Eccentricity'	'MajorAxisLength'	
'EquivDiameter'	'MinorAxisLength'	

Centroid

At project we used centroid, major-minor axis length, area, bounding box and extent. We will explain this property.

'Centroid'-- 1-by-ndims(L) vector; the center of mass of the region. Note that the first element of Centroid is the horizontal coordinate (or *x*-coordinate) of the center of mass, and the second element is the vertical coordinate (or *y*-coordinate). All other elements of Centroid are in order of dimension.

We are working at rectangles circles and triangles, and analyzed every object one by one so we have clearly every objects pixel. When we use centroid, we can easily reach corner and edge of objects after this with length of lines (corner to corner, edge to edge) finding max lengths. For every objects it is finding diagonals and intersection point of this diagonals is the center point of objects.



Major-Minor Axis Lengths

When using centroid function used diagonals but we didn't get length of this diagonals. So, we used major-minor axis length properties. We needed this length to decide objects shape and finding circles radius, diameters.

'MajorAxisLength'-- Scalar; the length (in pixels) of the major axis of the ellipse that has the same normalized second central moments as the region. This property is supported only for 2-D input label matrices.

'MinorAxisLength' -- Scalar; the length (in pixels) of the minor axis of the ellipse that has the same normalized second central moments as the region. This property is supported only for 2-D input label matrices.

To find this lengths function finding lines normalized to center of objects. At our objects defined pixels are valued ‘1’ so to find length just summing values of pixels one by one than function defines major and minor length due to this summision.

Area

This property is easily founding by summing every pixels of objects. Our objects pixel values are 1 so summing every pixel value easily give area.

Bounding Box:

With boundingbox property we finding box around objects. This function finding major and minor points of objects. This point are outer edge and corner points. With edge detection and corner detection we find them. After finding this point we use rectangle function to draw lines which normalized x and y axis and we get box around objects.

Extent:

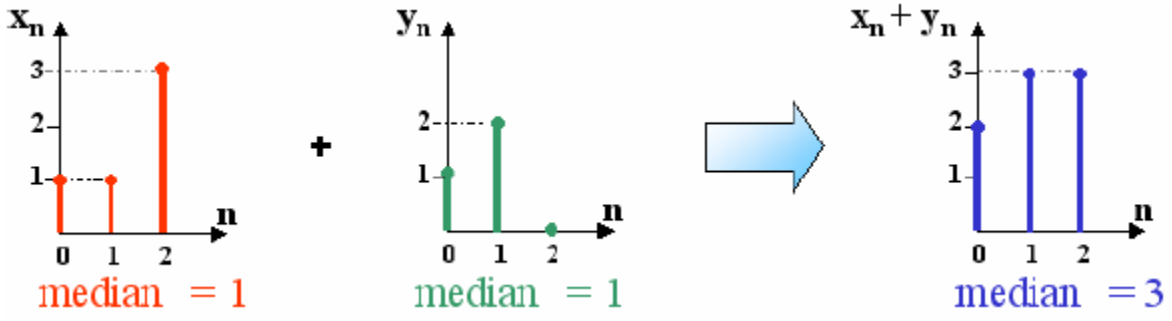
This is the value that area rate of object and bounding box. First bounding box filling, it means every pixel in box become ‘1’ and function sum this value to find area of bounding box. After this finding value that $\frac{boundingbox\ area}{object\ area}$. We use this value to shape classification.

8.3 medfilt2()

We used medfilt2() to clean noise on images. That was not necessary to computer drawings images but for getting images from cameras or other devices we need to use medfilt. This function works on two-dimension images.

- Median filtering is a non-linear operation:

Generally: median { xm + ym } ≠ median { xm } + median { ym }
example with signal sequences (length = 3):



- Odd window sizes are commonly used in median filtering:

3 × 3 ; 5 × 5 ; 7 × 7
example: 5-pixel cross-shaped window

- For even-sized windows (2 K values): the filter sorts the values then takes the average of the two middle values :

$$\text{Output value} = \frac{(\text{K}^{\text{th}} \text{ classified value} + (\text{K}+1)^{\text{th}} \text{ classified value})}{2}$$

Examle:

- 3×3 original image:

91	55	90
77	68	95
115	151	210

Median filtering using the full 3×3 neighborhood:

↳ we sort the original image values on the full 3×3 window:

55 , 68 , 77 , 90 , **91** , 95 , 115 , 151 , 210

⇓

median value = 91

Median filtering using 5-pixel cross-shaped size:

	55	
77	68	95
	151	

↳ We sort the original image values on the 3×3 cross window:

55 , 68 , **77** , 95 , 151

⇓

median value = 77

When we use medfilt2() on our image (matrix) we are doing convolution for example 3x3 median matrix going through all pixels with group of 3x3 matrices and clearing noise factors. We know that medfilt2() is lowpass filter.

Example:

This example adds salt and pepper noise to an image, then restores the image using medfilt2.

```
I = imread('eight.tif');
J = imnoise(I,'salt & pepper',0.02);
K = medfilt2(J);
imshow(J)
figure, imshow(K)
```

medfilt2 uses ordfilt2 to perform the filtering :

Ordfilt2()

Type of Filtering Operation	MATLAB code	Neighborhood	Sample Image Data, Indicating Selected Element
Median filter	`B = ordfilt2(A,5,ones(3,3))`	1 1 1 1 1 1 1 1 1	88 16 56 5 3 30 21 63 42

Domain is equivalent to the structuring element used for binary image operations. It is a matrix containing only 1's and 0's; the 1's define the neighborhood for the filtering operation.

For example, B = ordfilt2(A,5,ones(3,3)) implements a 3-by-3 median filter;

B= ordfilt2(A,1,ones(3,3)) implements a 3-by-3 minimum filter;

and B = ordfilt2(A,9,ones(3,3)) implements a 3-by-3 maximum filter.

B = ordfilt2(A,1,[0 1 0; 1 0 1; 0 1 0]) replaces each element in A by the minimum of its north, east, south, and west neighbors.

8.4 mean ()

The main purpose of the function is to take the average of arrae. In Section 6.7, we have stated that we use it to find a radius and we will explain in more detail here. It cannot average an array of size 1. The "mean" function can average vectors, matrices, and large arrays from the matrix. Let's show how to use the Mean function;

M = mean (A, ‘all’) averages all the elements of matrix A. (This command is only available for MATLAB R2018b and later)

M = mean (A, dim) Averages the rows or columns of matrix A. If we type 1 instead of “dim her, each column averages each row if we type 2. If we don't type anything, it automatically assigns a value of 1.

For example;

A = [0.11; 2 3 2; 13.2; 4 2 2]

A = 4x3

0 1 1

2 3 2

1 3 2

4 2 2

M = mean (A)

M = 1x3

1.7500 2.2500 1.7500

A = [0.11; 2 3 2; 3 0 1; 1 2 3]

A = 4 × 3

0 1 1

2 3 2

3 0 1

1 2 3

M = mean (A, 2)

M = 4 × 1

0.6667

2.3333

1.3333

2.0000

M = mean (A, ecstasy) The average of all elements of the matrix A. This is the same as the "all" command, except that it is used for versions of MATLAB before R2018b. The fact that A is a multidimensional array does not affect this situation and gives the average of all its elements.

For example;

A (:,: 1) = [2,4]; -2 1];

A (:, 2, 2) = [9 13; -57];

A (:, 3) = [4; 8 -3];

M1 = mean (A, [1,2])

M1 =

M1 (:,:, 1) =

1.2500

M1 (:,:, 2) =

6

M1 (:,:, 3) =

3.2500

M2 = mean (A, [1,2 3])

M2 = 3.5000

Mall = mean (A, 'all')

Mall = 3.5000

M = mean (__, outtype)

8.5 edge () function

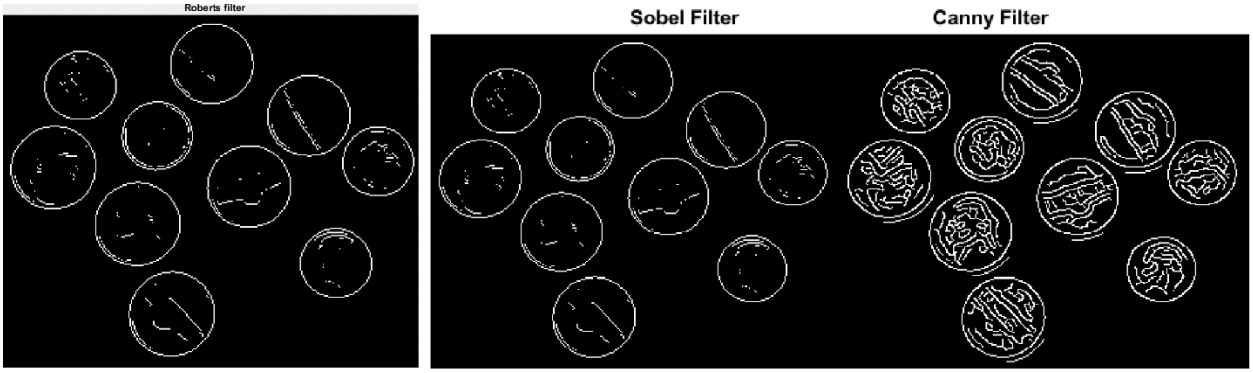
The function is used to find the edges of objects. We talked a little bit in Section 6.9. There are multiple methods to use the function, which we will talk about later. With these methods, only the edges of the object remain and we find the corner points with this function.

Im Let us explain in more detail how the “Edge” function works. First, let's talk about the methods used; “Sobel (default), Prewitt, Roberts, log, zerocross, Canny, approxcanny.” There are 7 methods. First of all, let's make a description of the methods and then show some of them by example.

- Sobel; According to the Sobel approach, the derivative finds the edges at the maximum gradient of the picture.
- Prewitt; According to the Prewitt approach, the derivative finds the edges at the maximum gradient of the picture.
- Roberts; According to Roberts approximation, he takes the derivative and finds the edges at the maximum gradient of the picture.
- log; Gaussian Laplace method on the image filter and search for zero transitions to find the edges.
- zerocross; Finds edges by searching for zero transitions with a filtering method that we specify.
- Canny; Looks at the local maximum points in the image gradient by filtering with the Gaussian derivative. This method uses two thresholds for strong and weak edges, so that it is more likely to find weak edges. This method is less likely to be deceived by noise than other methods.
- approxcanny; Canny method is a method of finding less sensitive edges as a result of faster application.

Let us give an example by applying some of these methods;

This is the picture we will use as an example. Now let's show the filtered version;



Now let's see how we use the function;

BW = edge (I) Automatically uses the Sobel method. It records the intersection of 0 and 1 on the pictures in binary mode. It then plots these intersections.

BW = edge (I, method) We can enter here whatever method we want to use.

BW = edge (I, method, threshold) returns all edges that are stronger than threshold.

BW = edge (I, method, threshold, direction) Specifies the edge directions of the object. The Sobel and Prewitt method can detect the edges vertically, horizontally or both. The Roberts method can detect horizontal edges of 45 or 135 degrees. This function is only available with the Sobel, Prewitt and Roberts method.

8.6 corner () function

 With the help of the edge detection function described in 8.5, we find the corner points. Once we find the edges of the object, we create a 4x2 matrix to find the corner points. Then we assign the smallest x and y points, then the smallest x and largest y points, then the largest x and y points, and finally the largest x and smallest y points in this matrix. Then we assign these values to the vectors we will create and now we have corner points. Let's examine how the Corner function is used;

corner C = (I); It identifies the corner points in the image and assigns these coordinates to the C matrix.

C = corner (I, method); We can write the method we want to find the corner points here. There are methods such as Harris (automatically uses this method if nothing is written) or um MinumumEigenvalue.

C = corner (I, N); Finds up to N corner points on the image.

C = corner (I, method, N); Finds up to N corner points using the method we want.

To give an example;

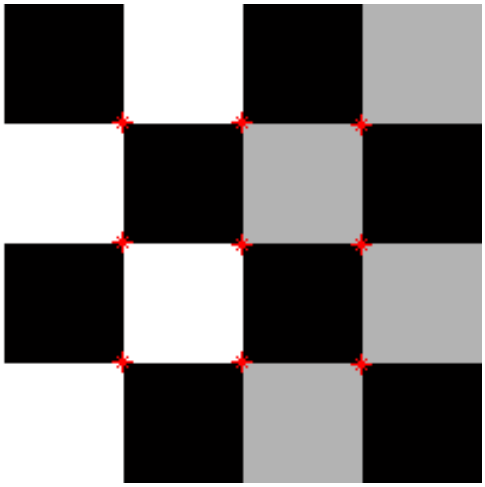
I = checkerboard (50,2,2);

C = corner (I);

imshow (I)

hold on

plot (C (:, 1), C (:, 2), R * ');



8.7 imfindcircles () function

Hough transform is the function that finds a circle. We enter certain values into the function and detect the circles in the picture. After finding the center of the object, the basic working system starts scanning the object within the desired radius range (for example, minimum 10 and maximum 20). It draws circles in the range of 10 to 20 pixels from the center and accepts it as a circle if all values are equal to the last place it reaches. Use of the function;

centers = imfindcircles (A, radius) A is the name of our picture and radius is the radius. The radii are assigned to a matrix, both in the x direction and in the y direction. Then they take the intersection as the center.

[centers, radii] = imfindcircles (A, radiusRange) We search for circles in the radius range of our choice. It assigns the intersection point of the radii of the circles it finds to the center variable.

[centers, radii, metric] = imfindcircles (A, radiusRange) Rotates the column vector for each circle to create a metric containing accumulator array peaks. The center and radius correspond to these metric rows.

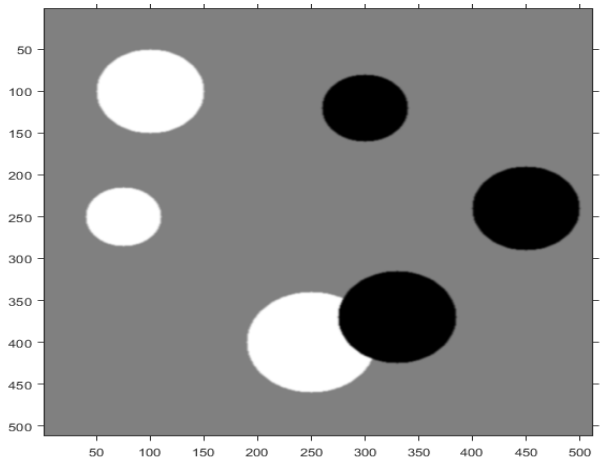
We use the viscircles () function to represent the circles we find. This function actually creates a new circle. Draws a new circle outside the circle using the information from the f "imfindcircles" function. We can adjust this line in any way and color. The use of this function is as follows;

viscircles (centers, radii); Draws a circle using the center and radius information we assign.

viscircles (ax, centers, radii) “ax ere using the center and radius information to the places we specify circles.

viscircles (___, Name, Value) We can draw circles around the circles we want to draw in any color (solid line, dashed line, dotted line, etc.).

When we write the code with MATLAB, we print it like this;



Original picture.

MATLAB code;

A = imread ('circlesBrightDark.png');

imshow (A)

R min = 30;

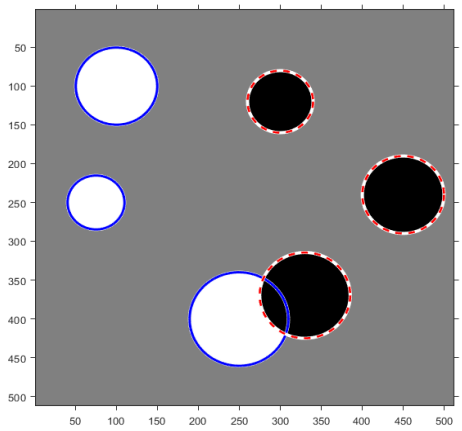
R max = 65;

[centersBright, radiiBright] = imfindcircles (A, [Rmin Rmax], 'ObjectPolarity', 'bright');

[centersDark, radiiDark] = imfindcircles (A, [Rmin Rmax], 'ObjectPolarity', 'dark');

viscircles (centersBright, radiiBright, 'Color', 'b');

viscircles (centersDark, radiiDark, 'LineStyle', '-');



8.8 Hough Transform

Our objects may not always appear properly. It may not have some parts, or another object may have come on it. Hough transform works by predicting possible geometric shapes of these edges. These steps can be summarized as follows;

• Edges are specified on the source image.

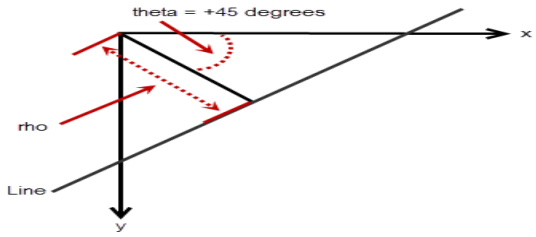
• The image is rendered binary (black and white) using a thresholding method.

• For each edge pixel, the values of the possible geometric shapes in which the point may be on the polar coordinate are increased one by one on the accumulator matrix used so that each edge pixel can vote on the possible shapes.

• The figures with the highest accumulator value are the most rated shapes and are most likely to be visible on the image.

• The shapes found can be optionally printed on the image.

To put it further, $\rho = x * \cos(\theta) + y * \sin$ (matematik) is the mathematical formula. The function starts with a vertical line and starts to rotate by changing the angle. Below is an example;



9. MANUELLY FUNCTION WORKS

Bu bölümde daha önce yapmış olduğumuz renk ile obje tanımlamayı kendimiz fonksiyon kullanmadan yapmaya çalıştık. Objeleri tanımladıktan sonra objelerin içini piksel piksel taradık. Sonra bu piksellerin toplamını oluşturduğumuz vektör değişkenlerin içine atadık. En büyük değerleri bularak bazı işlemler yaptık ve koşulları sağlayan objelerin hangi renk ve hangi şekil olduğunu yazdırdık.

```
image=imread('C:\Users\cevozby\Desktop\ders\project\resim\example.png');
for renk=1:3
diff_im = imsubtract(image(:,:,renk),rgb2gray(image));
diff_im = medfilt2(diff_im, [3 3]);
diff_im = im2bw(diff_im,0.18);
```

Bu kısımda resmi MATLAB ortamına aktarıyoruz ve renk filtresi uyguluyoruz. Daha önce açıkladığımız matlab kodundan farklı olarak burada for döngüsü vardır. For döngüsü burada renk filtresini değiştirmektedir "1=Red, 2=Green, 3=Blue". Böylece 3 tane ana renk üzerinde istediğimiz işlemleri yapabiliriz.

```
diff_im = bwareaopen(diff_im,300);
```

bwareaopen komutu binary resim üzerinde işlem yapar. Resim üzerinde istemediğimiz küçük parçaları yok eder. Örneğin bizim kullandığımız kodda 300'den daha az olan pixel topluluğunu otomatik olarak siliyor.

```
bw = bwlabel(diff_im, 8);
```

bwlabel komutu binary resim üzerinde "1" olan pixellerini tek bir etiket altına alıyor. Böylece resim üzerindeki objelerimizi tanımlamış oluruz

```
s=size(bw);
s2=size(bw');
```

Burada ise "s" ve "s2" matrislerini oluşturuyoruz ve bu matrislerin boyutunu resmin boyutuna göre ayarlıyoruz. "s2" matrisi için resmimizi boyutunun transpozu büyüklüğünde ayarlıyoruz. Bunu yapma sebebimiz ilerleyen zamanda kodu tarayacağımız zaman oluşabilecek boyut hatasını engellemektir.

```
cc = bwconncomp(bw,4);
```

"cc" adlı bir değişken oluşturuyoruz ve bu değişkenin için "bwconncomp" ile parametreler atıyoruz. Bu fonksiyona parantez içinde "bw,4" komutunu atıyoruz. Buradaki komutta binary resim olduğunu ve dört pixel yan yana gördüğünde bunu bir obje olarak almasını sağlıyoruz.

```
xlong=zeros(s(1),cc.NumObjects);
ylong=zeros(s2(1),cc.NumObjects);
```

"xlong, ylong" adlı matris oluşturuyoruz ve bu matrisin satır sayısını cc.NumObjects parametresi belirliyor. Sütün sayısını ise "s (1), s2 (1)" değerleri belirliyor. Örnek verecek olursak 5 tane objemiz olsun ve resmin satır sayısı 100 olsun. Xlong matrisi 5*100'lük bir sıfır matrisi oluyor. Bu matrisin amacı ileride x ve y eksenlerini tararken alacağımız pixel değerlerini içine atmaktır. Böylece kolayca hesap yapabiliriz ve istediğimiz zaman objenin hangi eksende ne kadar uzunluğu olduğunu bulabiliriz.

```
xmaxlong=0;
ymaxlong=0;
```

Bu değişkenlerin amacı ileride objenin en büyük değerlerini bulup onları objeleri tanımlamak için kullanmaktır.

```
cember=0;
kare=0;
dikdortgen=0;
ucgen=0;
tanimsiz_obje=0;
```
Burada değişkenlerimizi tanımlıyoruz ve her seferinde en başa gelip sıfırlayacak şekilde yazıyoruz
```
    for o=1:cc.NumObjects
        xlengths=[];
        ylengths=[];
```
Buradaki değişkenlerimizin amacı x ve y koordinatları için resim üzerinde bulduğumuz pixel uzunlukları vektör halinde tutmaktır.
```
        xtutucu=0;
        ytutucu=0;
```
Buradaki değişkenlerimizin amacı "xlengths" ve "ylengths" vektörlerinde işlem yapmak için geçici bir değer atıyoruz.
```
        grain = false(size(bw));
        grain(cc.PixelIdxList{o}) = true;
        A=double(grain);
        B=A';
        for i=1:s(1)-1
            for j=1:s(2)-1
                X=A(i,j);
                if X==1
                    xlong(i,o)=xlong(i,o)+1;
                else
                    xlong(i,o)=xlong(i,o);
                end
            end
```
Burada bulunan iç içe 2 tane for döngüsü resmi pixel pixel taramamıza olanak sağlıyor. "i" değişkeni sütunları ve "j" değişkeni satırları inceler. Yani "i" değişkeni y ekseninde değişirken "j" değişkeni x ekseninde değişir. "i" her yeni y noktasına geçtiğinde "j" bütün x değerlerini tarar ve her karşılaştı "1" pixeli "xlong" değişkenine atar.
```
            if xlong(i,o)>0
                xlengths=[xlengths xlong(i,o)];
            end
        end
```
Buradaki if komutunun amacı resimdeki bulduğumuz "xlong" sayısını "xlengths" vektörünün içine atar
```
        for k=1:length(xlengths)-1
            if xlengths(k)>xlengths(k+1)
                xtutucu=xlengths(k);
                xlengths(k)=xlengths(k+1);
                xlengths(k+1)=xtutucu;
            end
```

```
            xmaxlong=xtutucu;
        end
```

Burada ise “xlengths” içindeki en büyük sayıyı bulmaya çalışıyoruz ve bu sayıyı “xmaxlong” değişkenine atıyoruz. Böylece objenin uzunluğunu bulmuş oluyoruz

```
        for m=1:s2(1)-1
            for n=1:s2(2)-1
                Y=B(m,n);
                if Y==1
                    ylong(m,o)=ylong(m,o)+1;
                else
                    ylong(m,o)=ylong(m,o);
                end
            end
            if ylong(m,o)>0
                ylengths=[ylengths ylong(m,o)];
            end
        end
```

Burada ise az önceki x eksenini tarayan for döngüsünü y eksenini taramak için kullanıyoruz. Y eksenini taramak için resmin büyüklüğünün transposunu alıp s2 değişkenine atamıştık böylece tekrar aynı işlemi yaparak y eksenindeki uzunlukları bulup “ylengths” değişkeninin içine atadık.

```
        for l=1:length(ylengths)-1
            if ylengths(l)>ylengths(l+1)
                ytutucu=ylengths(l);
                ylengths(l)=ylengths(l+1);
                ylengths(l+1)=ytutucu;
            end
            ymaxlong=ytutucu;
        end
```

Burada ise objenin y ekseninde en uzun uzunluğu “ymaxlong” değişkenine atıyoruz. Buda bizim objenin yüksekliği oluyor.

```
        ilk_kenar=sqrt(ymaxlong^2+(xmaxlong/2)^2);
        ikinci_kenar=ilk_kenar;
        ucuncu_kenar=xmaxlong;

ucgen_cevre=(ilk_kenar+ikinci_kenar+ucuncu_kenar)/2;
        ucgen_cevre_ile_alan=sqrt(ucgen_cevre*(ucgen_cevre-
ilk_kenar)*(ucgen_cevre-ikinci_kenar)*(ucgen_cevre-
ucuncu_kenar));
        ucgen_alan=xmaxlong*ymaxlong/2;
```

Burada objenin üçgen olup olmadığını anlamak için önceden koşul belirliyoruz. Kenar uzunluklarını bulduğumuz yükseklik ve en uzunlukları ile hesaplıyoruz. Sonra kenar uzunlukları ile alan hesaplıyoruz. Bu bulduğumuz alan sonucunu daha sonrasında koşul fonksiyonun içine koyacağız.

```
        if(sum(xlengths)*0.99<= (pi*(xmaxlong/2)^2)*1.01+15)
&&   (sum(xlengths)*0.99>=   (pi*(xmaxlong/2)^2)*1.01-15   &&
xmaxlong==ymaxlong) && (ilk_kenar~=pi*xmaxlong/2)
            cember=cember+1;
        elseif  (sum(xlengths)<=  (xmaxlong*ymaxlong)+5)  &&
(sum(xlengths)>=          (xmaxlong*ymaxlong)-5)          &&
(xmaxlong==ymaxlong)
            kare=kare+1;
        elseif  (sum(xlengths)<=  (xmaxlong*ymaxlong)+5)  &&
(sum(xlengths)>=          (xmaxlong*ymaxlong)-5)          &&
(xmaxlong~=ymaxlong)
            dikdortgen=dikdortgen+1;
        elseif    (ucgen_cevre_ile_alan<=ucgen_alan+10)    &&
(ucgen_cevre_ile_alan>=ucgen_alan-10)
            ucgen=ucgen+1;
        else
            tanimsiz_obje=tanimsiz_obje+1;
        end
    end
```

Buradaki koşullarda objenin ne olduğunu bulmaya çalışıyoruz. İçerisindeki bütün pixellerin toplamı alan formülü(pi*r^2) ile uyuşuyorsa ve en ve uzunluk birbirine eşit ise bunu çember olarak atar. İçerisindeki pixellerin toplamı alan formülü(a*b) ile eşit ve en ve uzunluk birbirine eşit ise kare, en ve uzunluk birbirine eşit değilse dikdörtgen olarak atar. İçerisindeki pixellerin toplamı daha önce oluşturduğumuz üçgenin çevre alanına eşitse üçgen olarak atar. Paint kullanarak yaptığımız resimlerdeki objeler mükemmel olmadığı için içerisindeki pixellerin toplamı ile alan formülü eşit olamayabiliyor bu yüzden "+10,-10" gibi eklemeler veya çıkarmalar yaparak bu eşitliği belli bir aralığa almaya çalışıyoruz

```
    if renk==1
        kirmizi_cember=cember;
        kirmizi_kare=kare;
        kirmizi_dikdortgen=dikdortgen;
        kirmizi_ucgen=ucgen;
        kirmizi_tanimsiz_obje=tanimsiz_obje;
    elseif renk==2
        yesil_cember=cember;
        yesil_kare=kare;
        yesil_dikdortgen=dikdortgen;
        yesil_ucgen=ucgen;
        yesil_tanimsiz_obje=tanimsiz_obje;
    elseif renk==3
        mavi_cember=cember;
        mavi_kare=kare;
        mavi_dikdortgen=dikdortgen;
        mavi_ucgen=ucgen;
```

```
        mavi_tanimsiz_obje=tanimsiz_obje;
    end
end
```

Buradaki if ve elseif komutlarında objeleri renk renk ayırmaya çalışıyoruz. Biz resimdeki objeleri hangi renk ortamında inceliyorsak direkt olarak o koşula gidiyoruz ve bir önceki koşulda bulduğumuz "çember,kare,dikdörtgen,ucgen" değişkenlerini renk belirterek yeniden değişken oluşturuyoruz. Daha sonra kodun başında bu değişkenleri tekrardan sıfırlıyoruz.

```
fprintf('%d tane kırmızı çember \n%d tane kırmızı kare \n%d tane kırmızı dikdörtgen \n%d tane kırmızı üçgen \n%d tane kırmızı tanımsız obje \n',kirmizi_cember,kirmizi_kare,kirmizi_dikdortgen,kirmizi_ucgen,kirmizi_tanimsiz_obje);
fprintf('%d tane yeşil çember \n%d tane yeşil kare \n%d tane yeşil dikdörtgen \n%d tane yeşil üçgen \n%d tane yeşil tanımsız obje \n',yesil_cember,yesil_kare,yesil_dikdortgen,yesil_ucgen,yesil_tanimsiz_obje);
fprintf('%d tane mavi çember \n%d tane mavi kare \n%d tane mavi dikdörtgen \n%d tane mavi üçgen \n%d tane mavi tanımsız obje \n',mavi_cember,mavi_kare,mavi_dikdortgen,mavi_ucgen,mavi_tanimsiz_obje);
```

Kodun son kısmında ise bulduğumuz objeleri hangi renk olduğunu ve hangi nesne olduğunu çıktı olarak veriyoruz.

10 FINAL FORM OF THE PROJECT

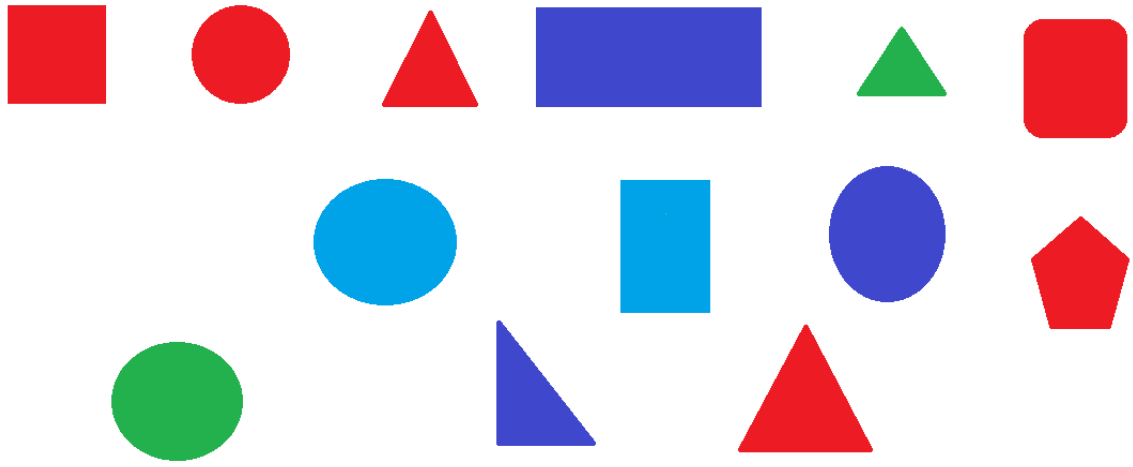
10.1 MATLAB CODE

```
image = imread('C:\Users\turhal\Desktop\kkkk.png'); %resmin matlaba aktarılması
diff_im = imsubtract(image(:,:,1),rgb2gray(image)); %renk filtresi uygulanması
diff_im = medfilt2(diff_im, [3 3]); %resimdeki gürültünün azaltılması
diff_im = im2bw(diff_im,0.18); %resimin siyah beyaza dönü?türülmesi
diff_im = bwareaopen(diff_im,300); %300 pixelden küçük kısımların yok sayılması
bw = bwlabel(diff_im, 8);
cc = bwconncomp(bw,4); %objelerin ayrıştırılması
stats = regionprops(bw, 'BoundingBox', 'Centroid','MajorAxisLength','MinorAxisLength','Area','Extent');
imshow('C:\Users\turhal\Desktop\kkkk.png'); %objelerin özelliklerinin stats değişkenine atılması
trishape=0; % başlangıç üçgen sayısı
recshape=0; %başlangıç kare sayısı
circshape=0; %başlangıç daire sayısı
for o=1:cc.NumObjects %her bir objeyi ayrı analiz etmek için döngü
    grain = false(size(bw)); %objelerin yerleştirileceği binary image boyutunun belirlenmesi
    grain(cc.PixelIdxList{o}) = true; %numaralandırılmış objenin çağırılması
    bb = stats(o).BoundingBox; %objenin bounding box özellikleri
    bc = stats(o).Centroid; %objenin merkezi
    d=mean([stats(o).MajorAxisLength stats(o).MinorAxisLength],2); %objenin çapı
    r=d/2; %objenin yarıçapı
    area=stats(o).Area %objenin alanı
    A=stats(o).Extent  %objenin alanının bounding boxa oranı
    circularity = ((2*pi*r) .^ 2) ./ (4 * pi * area); %objenin çembersellik değeri
    bw1=edge(grain,'Roberts'); %kenar dedektörü uygulanması
    data=corner(bw1); %kö?e noktalarının bulunması
    %köşe nokta koordinatlarının değişkenlere atanması
    ver1=zeros(1,2);
    ver1(1,1)=data(1,1);
    ver1(1,2)=data(1,2);
    ver2=zeros(1,2);
    ver2(1,1)=data(2,1);
    ver2(1,2)=data(2,2);
    ver3=zeros(1,2);
    ver3(1,1)=data(3,1);
    ver3(1,2)=data(3,2);
    ver4=zeros(1,2);
    ver4(1,1)=data(4,1);
    ver4(1,2)=data(4,2);
     if (0.99<=A)&&(A<=1.05) && (ver1(1)==ver2(1)) && (ver3(1)==ver4(1)) %kare olma koşulu,
karelerin sayılması ve işaretlenmesi
        rectangle('Position',bb,'EdgeColor','b','LineWidth',2);
        recshape=recshape+1;
         a=text(bc(1),bc(2), strcat('+ Rectangle   ',num2str(recshape)));
    set(a, 'FontName', 'Arial', 'FontWeight', 'bold', 'FontSize', 12, 'Color', 'yellow');
     else if (0.98<=circularity)&&(circularity<=1.05) && (0.75<A)&&(A<0.80) % çember olma koşulu,
çemberlerin işaretlenmesi ve sayılması
                Rmin=30;
    Rmax=600;
    [centersBright, radiiBright] = imfindcircles(diff_im,[Rmin Rmax],'ObjectPolarity','bright');
```

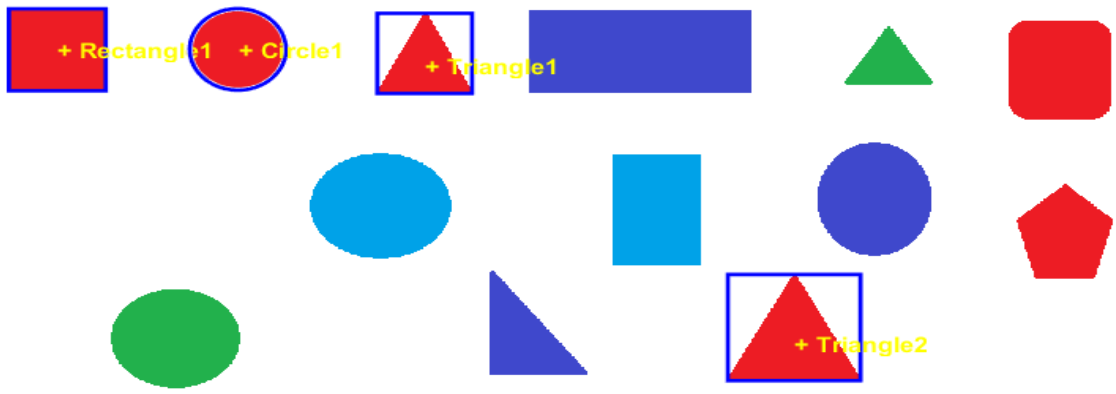
```
    [centersDark, radiiDark] = imfindcircles(diff_im,[Rmin Rmax],'ObjectPolarity','dark');
    viscircles(centersBright, radiiBright,'Color','b');
    circshape=circshape+1;
    a=text(bc(1),bc(2), strcat('+ Circle   ',num2str(circshape)));
    set(a, 'FontName', 'Arial', 'FontWeight', 'bold', 'FontSize', 12, 'Color', 'yellow');
        end
     end
    if (ver3(2)==ver2(2)) && ((A>0.5)&&(A<0.99)||(A>1.05)) && (circularity>1.1) || (circularity<0.99)
%üçgen olma koşulları, üçgenlerin işaretlenmesi ve sayılması
              rectangle('Position',bb,'EdgeColor','b','LineWidth',2);
              trishape=trishape+1
            a=text(bc(1),bc(2), strcat('+ Triangle   ',num2str(trishape)));
            set(a, 'FontName', 'Arial', 'FontWeight', 'bold', 'FontSize', 12, 'Color', 'yellow');
           end
end
```

10.2 INPUT-OUTPUT IMAGES

Input:



Output:



11 WORKING WITH INPUT VIDEO

In this part we will look how can we get input video and we will see how can we track objects. At this stage, we will only talk about the tracking system of red and circular objects, because when we include other objects, the processing speed is very low. If we want to follow other objects, it is enough to place our main code in the loop.

In the first step, we transfer the video we will examine as input to the matlab environment.

```
videoFileReader=vision.VideoFileReader('dizin\Red Ball.mp4');
```

In order to follow the circular objects in the video, we will examine each frame separately and perform the video conversion process again. After finding and marking the location of one object in one frame, we mark the location of the other frame. We do this for all frames and re-create it in the form of video. At this point, we assign the frames to a variable.

```
videoFrame      = step(videoFileReader);
```

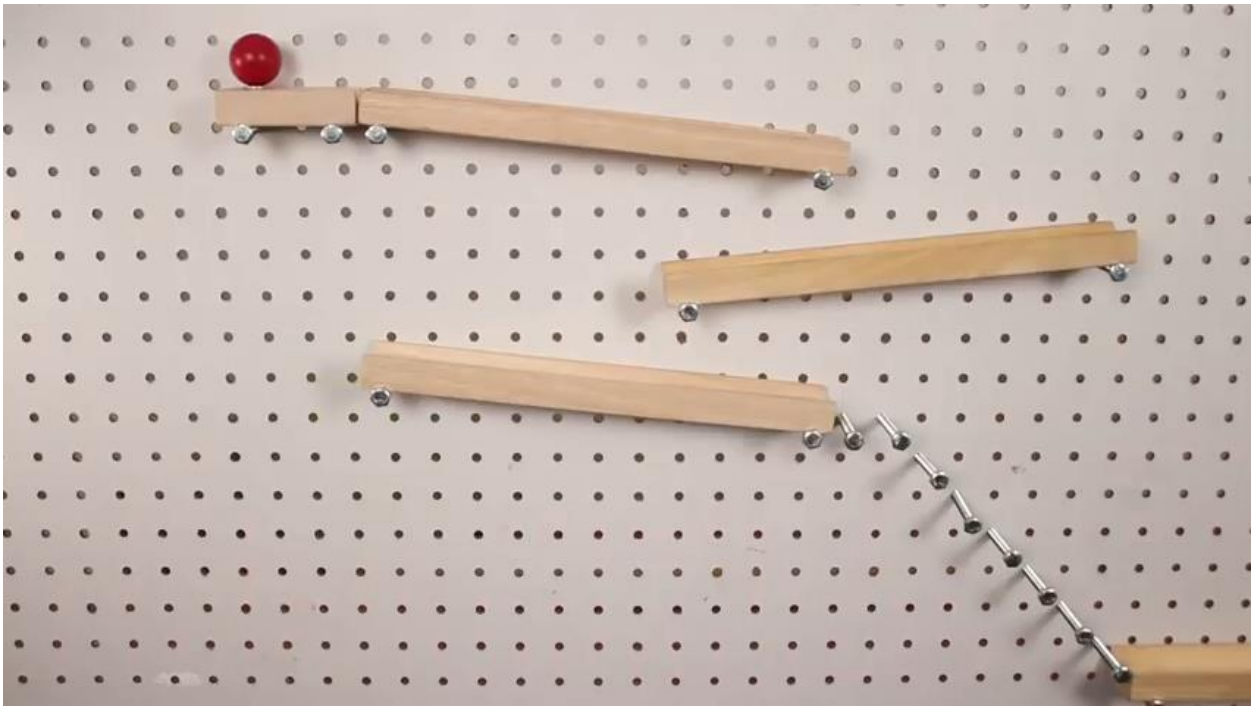
After reviewing the frames individually, we create a blank video with the properties of the original video to create the video again.

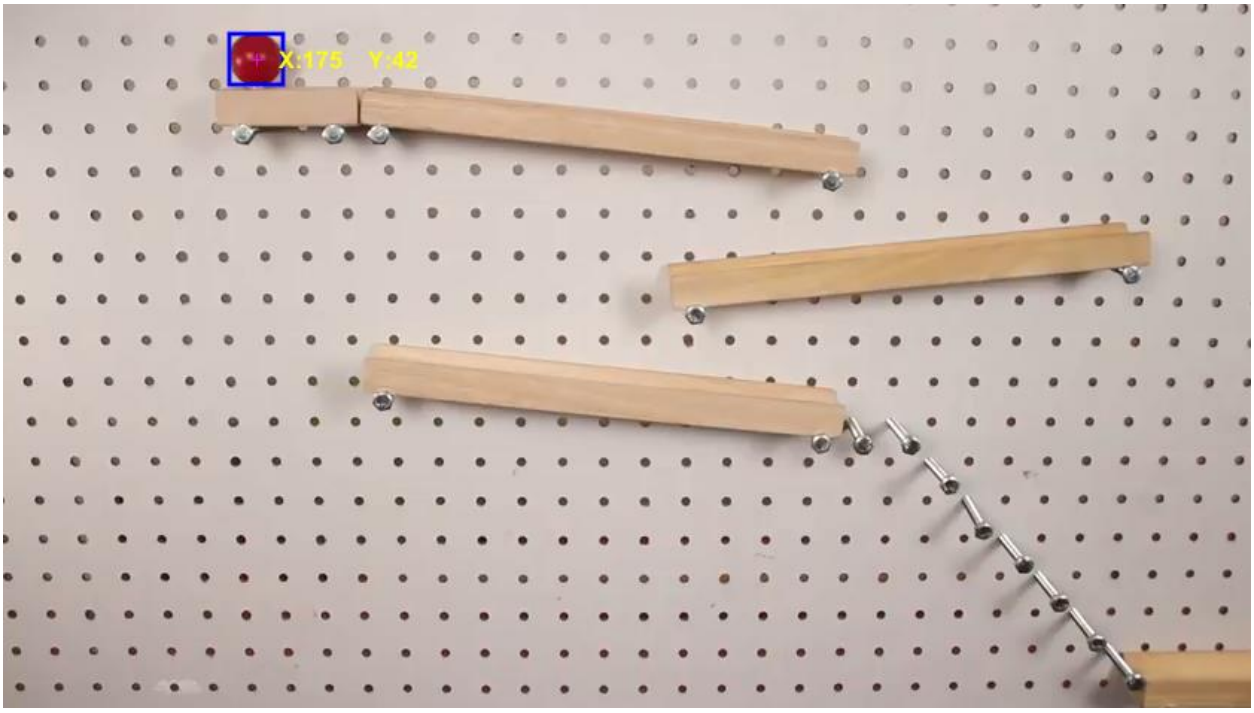
```
videoInfo    = info(videoFileReader);
videoPlayer  = vision.VideoPlayer('Position',[300 300 videoInfo.VideoSize+30]);
```

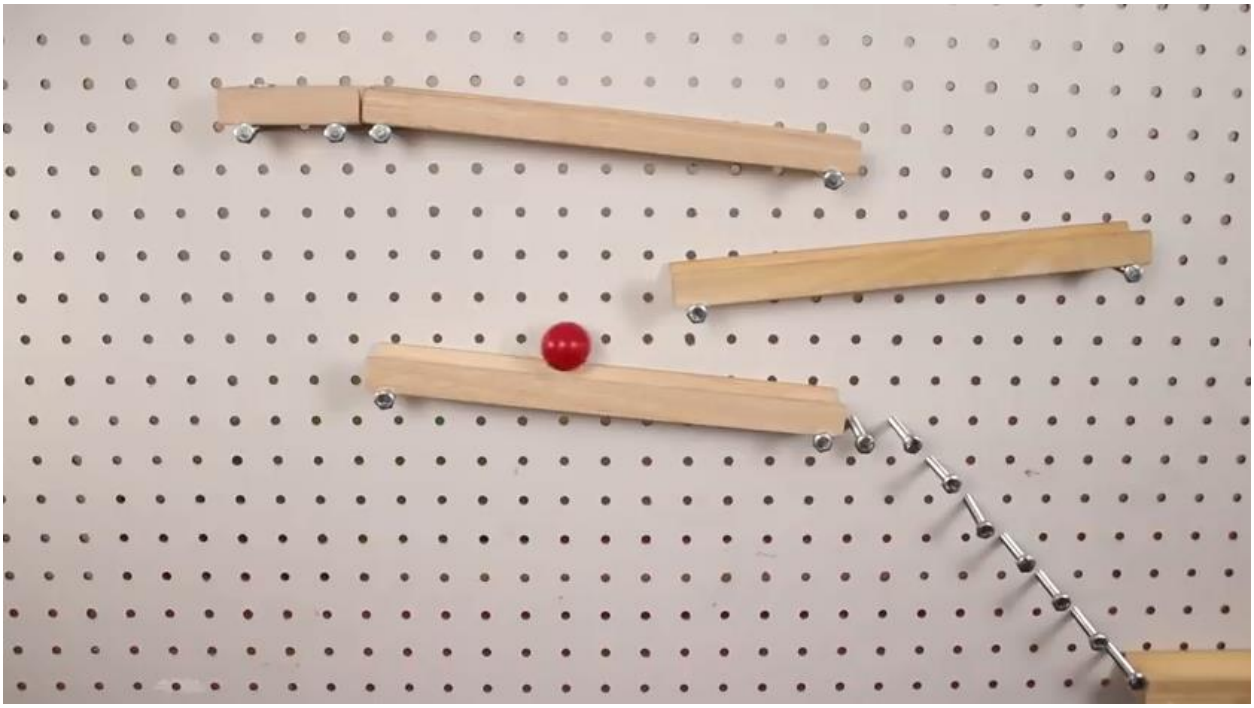
We use a while loop to process each frame separately.

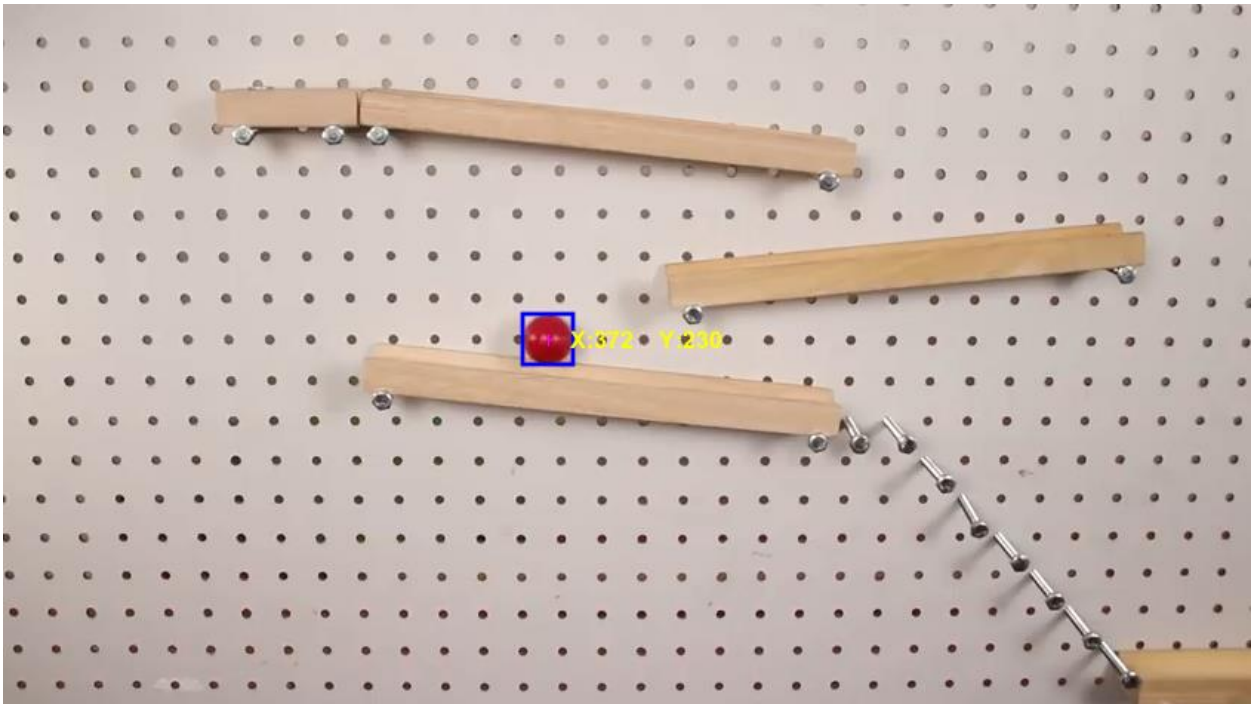
```
while ~isDone(videoFileReader)
    videoFrame = step(videoFileReader); %framelerin ayrı ayrı işlenmesi için
değişkene atanması.
    diff_im   =   imsubtract(videoFrame(:,:,1),   rgb2gray(videoFrame));   %renk
filtresi
diff_im = medfilt2(diff_im, [3 3]); %median filter
diff_im = im2bw(diff_im,0.18); %siyah beyaz filtre
diff_im = bwareaopen(diff_im,300); %300 pikseldenküçük objelerin yok sayılması
bw = bwlabel(diff_im, 8); %piksellerin bütünlenmesi
stats   =   regionprops(bw,   'BoundingBox',   'Centroid');   %objenin   konumsal
özelliklerinin alınması
imshow(videoFrame) %videonun ilgili framesinin gösterilmesi
hold on
 for object = 1:length(stats) %tespit edilen her obje için bölgesel özelliklerin
değişkenlere atanması
        bb = stats(object).BoundingBox;
        bc = stats(object).Centroid;
        rectangle('Position',bb,'EdgeColor','b','LineWidth',2)
        plot(bc(1),bc(2), '-m+')
        a=text(bc(1)+15,bc(2), strcat('X: ', num2str(round(bc(1))), '    Y: ',
num2str(round(bc(2)))));
        set(a, 'FontName', 'Arial', 'FontWeight', 'bold', 'FontSize', 12,
'Color', 'yellow');

 end
 step(videoPlayer,videoFrame); %oluşturulan videonun gösterilmesi
hold off
end
```

Examples of frames of video:




X:175 Y:42





12 CONCLUSION

We worked on basic object identification, tracking and counting methods in image processing for our Smart Traffic Systems using matlab. In order to identify the objects, we first examined the principles of the application of color filters. Then, we analyzed that the picture is in matrix form in MATLAB environment and the regional and positional properties of the objects in the picture with the operations we perform on the matrix. With the data we obtained, we ensured the classification of objects under various conditions. We have marked for tracking the objects that meet the condition. In order to process the video by using this method, we have ensured that the frames in the video are processed one by one in order to follow the movements of the objects.

The shape classification and object recognition we have made in the first stage can be developed in various fields even though Intelligent Traffic Simulation can be used in areas such as tumor detection in x-rays and missing element detection in the circuit board.

In Phase 2 of the project, we will focus on the recognition and tracking of cars.

REFERANCES

- https://www.mathworks.com/help/index.html
- https://muratdelen.com/hough-transform
- http://faculty.petra.ac.id/resmana/private/matlab-help/toolbox/images/medfilt2.html
- http://matlab.izmiran.ru/help/toolbox/images/regionprops.html
- https://www.mathworks.com/help/images/ref/regionprops.html
- http://www.unit.eu/cours/videocommunication/Non-linear_filtering.pdf
- http://roboturka.com/matlab/matlab-calisma-ortami-tanitimi/
- https://www.elektrikport.com/universite/matlab-ile-goruntu-isleme-2-elektrikport-akademi/8197#ad-image-0
- https://erencelik.com/matlab-nesne-takibi/
- https://engineertuncay.blogspot.com/2017/09/matlab-ile-krmz-mavi-ve-yesil-obje.html
- https://www.cs.tau.ac.il/~dcor/Graphics/cg-slides/MATLAB-tutorial.pdf