# Report, Project, TDT4195

Kim Rune Solstad, Bjørn Bråthen

November 17, 2013

## 1 Abstract

This is the report for the main project in the course TDT4195. The assignment was to graphically represent a image with different coloured circular objects using computer-graphics and image-processing techniques. The task was solved in two parts, one image-processing part using Matlab, and one computer-graphics part using OpenGL. The Matlab-script writes a .txt file with coordinates, radius and color of the objects found. The OpenGL program then reads that file and represents the content as different coloured models in a three dimensional coordinate system. The programs are further explained in the sections below.

## 2 Matlab

### 2.1 Pre-processing

Before an object in our image can be detected as a circle, the image has to be processed in different ways. Each step is described below.

#### 2.1.1 Normalize color-values

The example pictures has different lightening conditions, one of them e.g., has subtile shadows along the edges of the sweets, the others don't. Most of the sweets in the pictures have shadow on themselves. These shadows often becomes a problem when we try to separate the colored sweets from each other. The way to get around this problem is to normalize the colors as a step before the color separation process. For each color plane, R, G, B, we divide the color-values by the magnitude of the combined R,G,B vector:

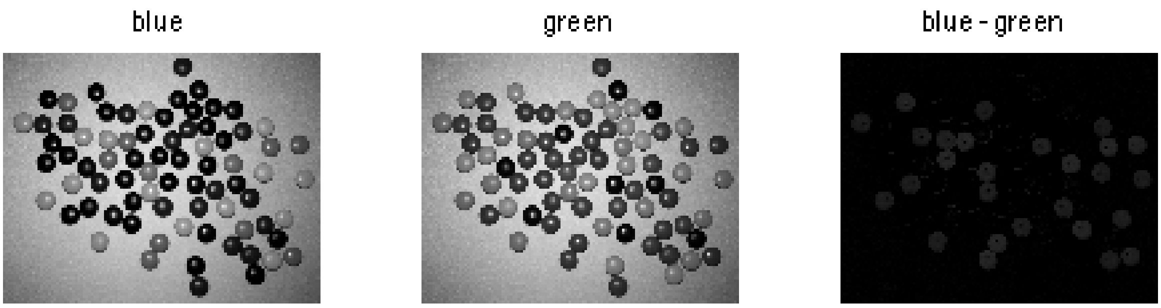
```
R = R./sqrt(R.^2 + G.^2 + B.^2);
G = G./sqrt(R.^2 + G.^2 + B.^2);
B = B./sqrt(R.^2 + G.^2 + B.^2);
```

This results into the following image:

### 2.1.2 Separating colors

SeparateColors.m function is used to filter out the different colors. It takes in a multi-color-image matrix, and returns six images. Each image contains only the non-stops of a certain color. The images are generated by subtracting the unwanted colors, and increasing the values of the wanted colors.

For example: The blue non-stops are separated by subtracting the green-image from the blue-image, thus removing green nonstops.




As the green and the blue images look quite alike, the resulting image is mostly black. To be able to further subtract colors from the image, the max values are increased. Red colors are then subtracted from the image.


Below is the code that generated the resulting image.

```
only_green = greenImg;
only_green = only_green - redImg;
[max_green, ~] = max(only_green(:));
only_green = only_green.*(1/max_green);
only_green = only_green - blueImg;
[max_green, ~] = max(only_green(:));
only_green = only_green.*(1/max_green);
```

The image created so far has the blue objects represented as the brightest circles. To remove the unwanted representation of other objects, thresholding is used.

After the thresholding process the image has blobs of pixels, but because of the highlight created by the flashlight, the thresholded sweets has holes inside them. The holes are therefore filled with an inbuilt matlab function, *imfill(image, type)*,

```
only_red = imfill(only_red,'holes');
```

To remove noise from the images, the images are median filtered. This is done by the function *medianFilter(img, filterSize)*.

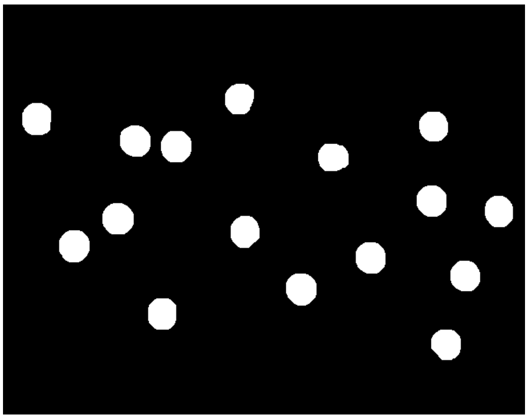
```
% Remove some noise with a median filter. SLOW!
function [ output_args ] = medianFilter( img, filterSize )
    [m, n] = size(img);
    output_args = zeros(m, n);
    l = floor(filterSize / 2);
```

```
    for x = 1 + l: n - l
        for y = 1 + l: m - l
            set = img(y - l:y + l, x - l: x + l);
            output_args(y, x) = median(set(:));
        end
    end
end
```


Even though the colors now are separated, the binary shape of the sweets is uneven and some of the sweets look like a circle with waves on it. Some of the sweets is therefore hard to detect as circles, as is. This uneveness is resolved by executing a sequence of open and closing filters, with a disk filter,

```
    I = imclose(I,strel('disk',1));
    I = imopen(I,strel('disk',10));
    I = imopen(I,strel('disk',10));
    J = imclose(I,strel('disk',1));
```

After all these steps, the final bit-mask will look like this:



## 2.2 Circle detection and localization

Now that the colored sweets are separated and represented as circular binary objects, the detection of the sweets' centroid and radius can be detected. We tried to implement our own circle detector, but that's before we heard that we - for this particular project - could use the inbuilt matlab function *imfindcircles(img, [radmin, radmax])*. Since we implemented some parts of this circular detection system, we will describe how and why we tried this approach. To understand how the canny edge detector and the hough transform worked, we read some papers [1, 2] that is also the basis of our implementation.

### 2.2.1 Canny edge-detector

Canny edge detector takes an image and finds the edges based on the gradients found in the image. It also finds the direction of the edges and their magnitude. All this information is passed on to the hough algorithm later on. The gradients is found by convolving the image with a sobel operator,

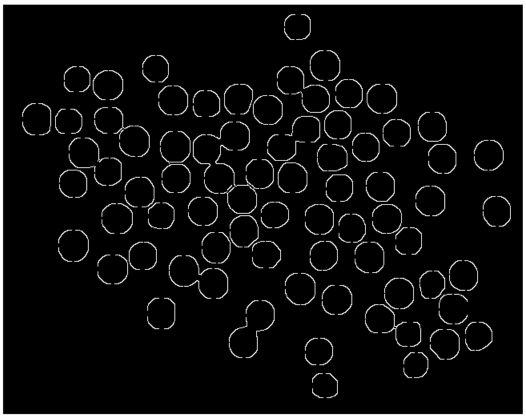
```
w = [-1 0 1; -2 0 2; -1 0 1];
h = [1 2 1; 0 0 0; -1 -2 -1];

x = conv2(I,w, 'same');
y = conv2(I,h, 'same');
```

The magnitude and angle (theta) is calculated for each edge,

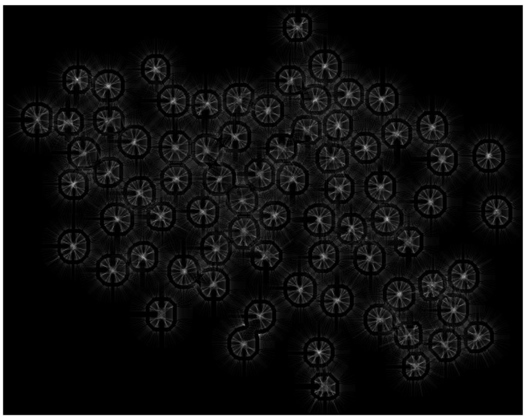
```
magnitude = sqrt(Dx.^2+Dy.^2);
% The direction of edges
theta = zeros(size(Dx));
theta = double(atand((Dy./Dx)));
```

Now, all those angles has to be detected by their direction in a 8-connected neighbour fashion, and the edges has to be stripped down, so the maximum edge value is preserved, while the other less-valued edge values is thrown away. Next step, the image is thresholded, so the hard edges and soft edges is separated. We then want to find pixels which has low values that's connected to a pixel with a high value, to extend the edges in our edge map. This process is called hysteresis. Now we have a complete edge map, as seen below. For more insight in this process, see *edgeCanny.m*.

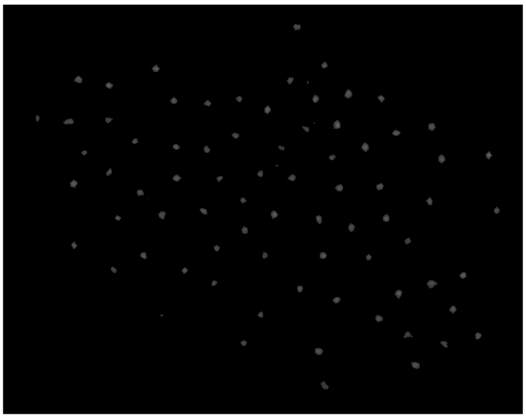


### 2.2.2 Hough circular transform

Now when the edges have been detected, we can try detecting the centroids using a hough transform. Basically we transform each circular edge into local polar coordinates; we read the angle of the edge and walk pixel by pixel in both directions until we get to the max radius defined when the function is called. We add the value of the magnitude of the edge divided by the length to that particular point. This creates a linear fall-off along the edge line. So if the circle is smaller than the max radius, the centeriod should be highlighted, because of all the lines hitting its' center. We end up with an something similar to the image below.



This function is work in progress since we chose to use the inbuilt matlab function instead. We basically got the idea from the tech-paper anyways. Futher on, we apply a medianfilter and a thresholding to these star-shaped objects. By doing this we neglate the least important potential centeriods, and get a basis of pixels we can segment and run non-maximal suppression on. If we do so, we would find an approximate centroid for all the circular objects in the scene. The last step would have been to find the radius of each circle as well.



## 2.3 Improvements

A more adjustable but computational heavier approach to separate colors in the image, is by filtering. Each matrix would then have to be filtered, to extract the r, g and b values. An appropriate range would have to be set, for this method to work. For example: The blue non-stops in a image would be separated by filtering the values 0 to 50 from the red matrix, 0 to 50 from the green matrix and 205 to 255 from the blue image. A new matrix is then generated. The overlapping values are represented as 1.0 in the new matrix. The remaining values are set to 0.0.

# 3 OpenGL

The program created for this part of the assignment, is made by modifying code given to us for the graphics part of the course. The color-buffers and the vertex-buffer is created in the initialization phase. For each frame, RenderScene is called. The txt-file is read once for each object everytime RenderScene is called. The text-file is read using ReadFile.cpp. ReadFile uses the windows library conio.h to read the file. The read function used to get the values, requires an integer representing a line in the .txt as input. Parameters requiered to represent the object is extracted from that line, and placed in a array. A pointer to the first parameter is then returned. By reading the txt-file each time the scene is rendered, the program is able to change the scene while running. This is done by modifying the txt-file. Each cube needs four parameters to be represented. One for color, one for radius, one for the x-coordinate and one for the y-coordinate. After setting the values, a model-matrix for the cube is made. The model-matrix consists of an identity-matrix multiplyed with a translation-matrix, multiplyed with a scale-matrix. The model-matrix for the cube represents the radius and position of the object. After multiplying the model-matrix with the mvp-matrix, a collor-buffer is bound before drawing the cube. The process is repeated until all objects are represented.

## 3.1 Improvements

Today, the cubes are given color by static premade color-buffers. To support all colors, one buffer for each different color given in the input-file should be made. This solution was considered, but because of memoryleaks in the program, the values from the file where overwritten. Lack of experience programming in the C-language, resulted in our program not being able to support dynamically made colorbuffers.

# References


[1] Unknown author, *Canny Edge Detector*. `http://www.cse.iitd.ernet.in/~pkalra/csl783/canny.pdf`, March 23, 2009.

[2] Jaroslav Borovicka, *Circle Detection Using Hough Transforms Documentation*. `https://files.nyu.edu/jb4457/public/files/research/bristol/hough-report.pdf`, March 04, 2003.

# 4 Code attachments

../matlab/main.m

```
close('all');

% the file which we process
fileToLoad = 'sweetsA03';

% read in image from file
originalImg = imread([fileToLoad '.png']);

% scale image to 0-1 values
originalImg = double(originalImg)/255.0;

% Get the separate color masks.
seperateColoredObjects = separateColors(originalImg);

red_sweets = seperateColoredObjects(:,:,1);
green_sweets = seperateColoredObjects(:,:,2);
blue_sweets = seperateColoredObjects(:,:,3);
yellow_sweets = seperateColoredObjects(:,:,4);
pink_sweets = seperateColoredObjects(:,:,5);
orange_sweets = seperateColoredObjects(:,:,6);

% merge all maskes to one complete mask
bitmask = red_sweets | green_sweets | blue_sweets | yellow_sweets |
     pink_sweets | orange_sweets;

% create copy of original
maskedImage = originalImg;

% mask image (all colors)
for i = 1:1:3
    maskedImage(:,:,i) = double(originalImg(:,:,i).*bitmask);
end

% find sweets in every color
[red_c,red_r] = imfindcircles(red_sweets,[3,23]);
[green_c,green_r] = imfindcircles(green_sweets,[2,22]);
[blue_c,blue_r] = imfindcircles(blue_sweets,[3,23]);
[yellow_c,yellow_r] = imfindcircles(yellow_sweets,[3,23]);
[pink_c,pink_r] = imfindcircles(pink_sweets,[3,23]);
[orange_c,orange_r] = imfindcircles(orange_sweets,[2,22]);

% self implemented hough
C = circlesHough(double(bitmask),3,23);

%display detected circles
f = figure('Name','Detected sweets','NumberTitle','off','visible','
    off'); imshow(originalImg);
viscircles(red_c,red_r,'EdgeColor','r');
viscircles(green_c,green_r,'EdgeColor','g');
viscircles(blue_c,blue_r,'EdgeColor','b');
viscircles(yellow_c,yellow_r,'EdgeColor','yellow');
viscircles(pink_c,pink_r,'EdgeColor',[1,0.4,0.6]);
viscircles(orange_c,orange_r,'EdgeColor',[1,0.56,0]);
detectedSweets = getframe(f);

% Write images to files.
imwrite(originalImg,['results/' fileToLoad '_original.png']);
imwrite(maskedImage,['results/' fileToLoad '_masked.png']);
```

```
imwrite(bitmask,['results/' fileToLoad '_total_mask.png']);
imwrite(detectedSweets.cdata,['results/' fileToLoad '
    _detected_sweets.png']);
imwrite(red_sweets,['results/' fileToLoad '_red_sweets.png']);
imwrite(green_sweets,['results/' fileToLoad '_green_sweets.png']);
imwrite(blue_sweets,['results/' fileToLoad '_blue_sweets.png']);
imwrite(yellow_sweets,['results/' fileToLoad '_yellow_sweets.png'])
    ;
imwrite(pink_sweets,['results/' fileToLoad '_pink_sweets.png']);
imwrite(orange_sweets,['results/' fileToLoad '_orange_sweets.png'])
    ;

% write coordinates for the circles to file
% colorstate, radius, x-pos, y-pos
%fileName = [fileToLoad '_test.txt'];
fileName = 'test.txt';

fileId = fopen(fileName,'w');
if fileId ~= -1
    for i = 1:size(red_c,1)
      fprintf(fileId,'%d%d%4.4d%4.4d\r\n',1,int16(red_r(i,1)),int16
          (red_c(i,1)),int16(red_c(i,2)));
    end;
    for i = 1:size(green_c,1)
      fprintf(fileId,'%d%d%4.4d%4.4d\r\n',2,int16(green_r(i,1)),
          int16(green_c(i,1)),int16(green_c(i,2)));
    end;
    for i = 1:size(blue_c,1)
      fprintf(fileId,'%d%d%4.4d%4.4d\r\n',3,int16(blue_r(i,1)),
          int16(blue_c(i,1)),int16(blue_c(i,2)));
    end;
    for i = 1:size(yellow_c,1)
      fprintf(fileId,'%d%d%4.4d%4.4d\r\n',4,int16(yellow_r(i,1)),
          int16(yellow_c(i,1)),int16(yellow_c(i,2)));
    end;
    for i = 1:size(pink_c,1)
      fprintf(fileId,'%d%d%4.4d%4.4d\r\n',5,int16(pink_r(i,1)),
          int16(pink_c(i,1)),int16(pink_c(i,2)));
    end;
    for i = 1:size(orange_c,1)
      fprintf(fileId,'%d%d%4.4d%4.4d\r\n',6,int16(orange_r(i,1)),
          int16(orange_c(i,1)),int16(orange_c(i,2)));
    end;
    fclose(fileId);
end
```

../matlab/normalizeColorValues.m

```
% This function normalizes the colors, so its easier to
% overcome those shadow-related problems.
function [R, G, B] = normalizeColorValues(R,G,B)
    R = R./sqrt(R.^2 + G.^2 + B.^2);
    G = G./sqrt(R.^2 + G.^2 + B.^2);
    B = B./sqrt(R.^2 + G.^2 + B.^2);
```

../matlab/separateColors.m

```
% Separates differen sweet-colors from each other. It separates
    into
% red, green, blue, yellow, pink and orange.

function [ output_args ] = separateColors( img )
```

```
[m, n, ~] = size(img);

% get individual color planes.
redImg      = img(:, :, 1);
greenImg    = img(:, :, 2);
blueImg     = img(:, :, 3);

% normalize the color values before we start.
[redImg, greenImg, blueImg] = normalizeColorValues(redImg,greenImg,
    blueImg);

% RED SWEETS
only_red = redImg - greenImg.*4;
only_red = only_red > 0.42;
only_red    = medianFilter(only_red, 4);

% GREEN SWEETS
only_green = greenImg;
only_green = only_green - redImg;
[max_green, ~] = max(only_green(:));
only_green = only_green.*(1/max_green);
only_green = only_green - blueImg;
[max_green, ~] = max(only_green(:));
only_green = only_green.*(1/max_green);

only_green = only_green > -0.4;
only_green  = medianFilter(only_green, 4);

% BLUE SWEETS
only_blue = blueImg;
only_blue = only_blue - greenImg;
[max_blue, ~] = max(only_blue(:));
only_blue = only_blue.*(1/max_blue);
only_blue = only_blue - redImg;
[max_blue, ~] = max(only_blue(:));
only_blue = only_blue.*(1/max_blue);

only_blue = only_blue > -0.6;
%only_blue   = medianFilter(only_blue, 4);

% YELLOW SWEETS
only_yellow = redImg - blueImg;
[max_yellow, ~] = max(only_yellow(:));
only_yellow = only_yellow.*(1/max_yellow);
only_yellow = only_yellow + greenImg;
[max_yellow, ~] = max(only_yellow(:));
only_yellow = only_yellow.*(1/max_yellow);

only_yellow = only_yellow > 0.9;
%only_yellow = medianFilter(only_yellow, 14);

% PINK SWEETS
only_pink = redImg - greenImg;
[max_pink, ~] = max(only_pink(:));
only_pink = only_pink.*(1/max_pink);
only_pink = only_pink + blueImg;
[max_pink, ~] = max(only_pink(:));
only_pink = only_pink.*(1/max_pink);

only_pink = only_pink > 0.8;
only_pink   = medianFilter(only_pink, 10);
```

```
% ORANGE SWEETS
only_orange = redImg - blueImg;
only_orange = only_orange - only_green;
only_orange = only_orange - only_blue;
only_orange = only_orange - only_yellow;
only_orange = only_orange - only_red;
only_orange = only_orange - only_pink;

only_orange = only_orange > 0.45;
only_orange = medianFilter(only_orange, 8);


% FILL THE HOLES!
only_red = imfill(only_red, 'holes');
only_green = imfill(only_green, 'holes');
only_blue = imfill(only_blue, 'holes');
only_yellow = imfill(only_yellow, 'holes');
only_pink = imfill(only_pink, 'holes');
only_orange = imfill(only_orange, 'holes');

% Some sweets wants to become another color,
% so remove those who does this.
only_blue = only_blue & ~only_pink;
only_pink = only_pink & ~only_red;
only_green = only_green & ~only_yellow;

% Even out the shape of those sweets!
only_red = openAndClose(only_red);
only_green = openAndClose(only_green);
only_blue = openAndClose(only_blue);
only_yellow = openAndClose(only_yellow);
only_pink = openAndClose(only_pink);
only_orange = openAndClose(only_orange);

% Create output
output_args = zeros(m, n, 6);

output_args(:,:, 1) = only_red;
output_args(:,:, 2) = only_green;
output_args(:,:, 3) = only_blue;
output_args(:,:, 4) = only_yellow;
output_args(:,:, 5) = only_pink;
output_args(:,:, 6) = only_orange;

end
```

../matlab/medianFilter.m

```
% Remove some noise with a median filter. SLOW!
function [ output_args ] = medianFilter( img, filterSize )
   [m, n] = size(img);
   output_args = zeros(m, n);
   l = floor(filterSize / 2);
   for x = 1 + l: n - l
       for y = 1 + l: m - l
           set = img(y - l:y + l, x - l: x + l);
           output_args(y, x) = median(set(:));
       end
   end
end
```

../matlab/openAndClose.m

```
function J = openAndClose(I)
    % Closing and opening filters, to remove some uneven edges
    I = imclose(I,strel('disk',1));
    I = imopen(I,strel('disk',10));
    I = imopen(I,strel('disk',10));
    J = imclose(I,strel('disk',1));
```

../matlab/edgeCanny.m

```
% Described here:
% http://www.cse.iitd.ernet.in/~pkalra/csl783/canny.pdf
function [J, theta, magnitude] = edgeCanny(I, highThresh, lowThresh
    )


    % Smooth image
    J = gaussian(I,7,1.4);

    % Find gradients
    %L = findGradients(J);
    [Dx, Dy] = findGradients(J);

    %figure('Name','Horizontal gradients','NumberTitle','off');
        imshow(Dx);
    %figure('Name','Vertical gradients','NumberTitle','off');
        imshow(Dy);

    % Combine both x and y direction (get the magnitude)
    % for faster computation use this
    % aprox_magnitude = abs(Dx)+abs(Dy);

    magnitude = sqrt(Dx.^2+Dy.^2);
    % The direction of edges
    theta = zeros(size(Dx));
    theta = double(atand((Dy./Dx)));

    figure('Name','Theta','NumberTitle','off'); imshow(theta);

    nomaxsup = zeros(size(J,1),size(J,2));

    % normalize angles

    for x = 1:size(nomaxsup,1)
        for y = 1:size(nomaxsup,2)

            % Putting each angle inside their respective quadrant

            if ((theta(x,y) > -22.5 && theta(x,y) < 22.5) || (theta
                (x,y) > 157.5) && (theta(x,y) < -157.5))
                if (y - 1 > 0 && y + 1 < size(nomaxsup,2))
                    % do a non-maximum suppression
                    if (magnitude(x, y) > magnitude(x, y+1) &&
                        magnitude(x, y) > magnitude(x, y-1))
                        nomaxsup(x,y) = magnitude(x,y);
                    else
                        nomaxsup(x,y) = 0;
                    end;
                end;
            end;
```

```
        if ((theta(x,y) < -112.5) && (theta(x,y) > -157.5) || (
            theta(x,y) > 22.5) && (theta(x,y) < 67.5))
            if (y - 1 > 0 && y + 1 < size(nomaxsup,2) && x - 1
                > 0 && x + 1 < size(nomaxsup,1))
                % do a non-maximum suppression
                if (magnitude(x, y) > magnitude(x-1, y+1) &&
                    magnitude(x, y) > magnitude(x+1, y-1))
                    nomaxsup(x,y) = magnitude(x,y);
                else
                    nomaxsup(x,y) = 0;
                end;
            end;
        end;

        if ((theta(x,y) > 67.5 && theta(x,y) < 112.5) || (theta
            (x,y) < -67.5 && theta(x,y) > -112.5))
            if (x - 1 > 0 && x + 1 < size(nomaxsup,1))
                % do a non-maximum suppression
                if (magnitude(x, y) > magnitude(x+1, y) &&
                    magnitude(x, y) > magnitude(x-1, y))
                    nomaxsup(x,y) = magnitude(x,y);
                else
                    nomaxsup(x,y) = 0;
                end;
            end;
        end;

        if ((theta(x,y) < -22.5 && theta(x,y) > -67.5) || (
            theta(x,y) > 112.5 && theta(x,y) <= 157.5))
            if (y - 1 > 0 && y + 1 < size(nomaxsup,2) && x - 1
                > 0 && x + 1 < size(nomaxsup,1))
                % do a non-maximum suppression
                if (magnitude(x, y) > magnitude(x-1, y-1) &&
                    magnitude(x, y) > magnitude(x+1, y+1))
                    nomaxsup(x,y) = magnitude(x,y);
                else
                    nomaxsup(x,y) = 0;
                end;
            end;
        end;
    end
end


% normalize values max gets to be 1, etc.
%max_intensity = max(nomaxsup(:));

%mul = 1 / max_intensity;

%nomaxsup = nomaxsup.*mul;


%figure(); imshow(nomaxsup);

L = zeros(size(nomaxsup,1),size(nomaxsup,2));
H = zeros(size(nomaxsup,1),size(nomaxsup,2));

% Hysteresis and thresholding

% Define which pixel is high threshold, which is low. Store in
    seperate
% matrixes
```

```
for x = 1:size(nomaxsup,1)
    for y = 1:size(nomaxsup,2)
        if(nomaxsup(x,y) > highThresh)
            H(x,y) = 1;
            L(x,y) = 0;
        else
            if(nomaxsup(x,y) >= lowThresh && nomaxsup(x,y) <=
                highThresh )
                H(x,y) = 0;
                L(x,y) = 1;
            else
                if (nomaxsup(x,y) < lowThresh)
                    L(x,y) = 0;
                    H(x,y) = 0;
                end
            end
        end
    end
end

%figure('Name','LowThreshold','NumberTitle','off');imshow(L);
%figure('Name','HighThreshold','NumberTitle','off');imshow(H);

% Go through the low threshold matrix and see whether the low
    threshold
% pixels are adjecent to a high threshold blob. Dooin it twice.
for x = 1:size(nomaxsup,1)
    for y = 1:size(nomaxsup,2)
        if (x-1 > 0 && y-1 > 0 && y+1 < size(nomaxsup,2) && x+1
            < size(nomaxsup,1) && L(x,y) == 1)
            if (nomaxsup(x,y) > highThresh || nomaxsup(x-1,y) >
                 highThresh || nomaxsup(x-1,y-1) > highThresh
                || nomaxsup(x,y-1) > highThresh || nomaxsup(x
                +1,y) > highThresh || nomaxsup(x+1,y+1) >
                highThresh || nomaxsup(x,y+1) > highThresh ||
                nomaxsup(x-1,y+1) > highThresh || nomaxsup(x+1,
                y-1) > highThresh)
                H(x,y) = 1;
            else
                H(x,y) = 0;
            end;
        end;
    end;
end;

%     for y = 1:size(nomaxsup,2)
%         for x = 1:size(nomaxsup,1)
%             if (x-1 > 0 && y-1 > 0 && y+1 < size(nomaxsup,2) && x
    +1 < size(nomaxsup,1) && L(x,y) == 1)
%                 if (nomaxsup(x,y) > highThresh || nomaxsup(x-1,y)
     > highThresh || nomaxsup(x-1,y-1) > highThresh || nomaxsup(x,y
    -1) > highThresh || nomaxsup(x+1,y) > highThresh || nomaxsup(x
    +1,y+1) > highThresh || nomaxsup(x,y+1) > highThresh ||
    nomaxsup(x-1,y+1) > highThresh || nomaxsup(x+1,y-1) >
    highThresh)
%                     H(x,y) = 1;
%                 else
%                     H(x,y) = 0;
%                 end;
%             end;
%         end;
%     end;
```

```
    J = H;
    theta = theta.*(pi/180);
    %theta = ((theta./pi)+0.5).*255;
    theta(~H) = 0;
```

../matlab/gaussian.m

```
function J = gaussian(I,size,sigma)

    % create the gaussian
    [x, y] = meshgrid(round(-size/2):round(size/2), round(-size/2):
        round(size/2));
    g = exp(- (x.*x/(2 * sigma * sigma))-(y.*y/(2 * sigma * sigma))
        );
    g = g./sum(g(:));

    % apply it by convolution
    J = conv2(I,g,'same');
```

../matlab/findGradients.m

```
function [x, y] = findGradients(I)
    w = [-1 0 1; -2 0 2; -1 0 1];
    h = [1 2 1; 0 0 0; -1 -2 -1];

    x = conv2(I,w,'same');
    y = conv2(I,h,'same');
```

../matlab/circlesHough.m

```
% https://files.nyu.edu/jb4457/public/files/research/bristol/hough-
    report.pdf
function C = circlesHough(I, min_rad, max_rad)
% find edges with canny edge detector
[edges, theta, magnitude] = edgeCanny(I, 0.2, 0.1);
figure('Name','edges','NumberTitle','off'); imshow(edges);

houghDomain = zeros(size(I));
dx = 0;
dy = 0;
px1 = 1;
px2 = 1;
py1 = 1;
py2 = 1;
figure('Name','Hough','NumberTitle','off');imshow(theta);

size_x = size(edges,1);
size_y = size(edges,2);

for i = 1:size(edges,1)
    for j = 1:size(edges,2)
        if(edges(i,j)>0)
            a = min_rad * cos(theta(i,j));
            b = min_rad * sin(theta(i,j));

            if(theta(i,j) < pi/4 && theta(i,j) > -pi/4)
                dx = ((a > 0) * 2) - 1;
                dy = dx * tan(theta(i,j));
            else
                dy = ((b > 0) * 2) - 1;
```

```
                dx = dy / tan(theta(i,j));
            end;

            while (sqrt(a^2 + b^2) < max_rad)
                px1 = int32(j + a);
                px2 = int32(j - a);
                py1 = int32(i - b);
                py2 = int32(i + b);

                if (px1 >= 1 && px1 < size_y && py1 >= 1 && py1 <
                    size_x)
                    houghDomain(py1,px1) = houghDomain(py1,px1) + (
                        magnitude(i,j)./sqrt(a^2+b^2));
                end;
                if (px2 >= 1 && px2 < size_y && py2 >= 1 && py2 <
                    size_x)
                    houghDomain(py2,px2) = houghDomain(py2,px2) + (
                        magnitude(i,j)./sqrt(a^2+b^2));
                end;
                a = a + dx;
                b = b + dy;

            end;

        end;
    end;
end;
maxVal = max(houghDomain(:));
houghDomain = houghDomain./maxVal;

figure('Name','Hough','NumberTitle','off');imshow(houghDomain);

houghDomain = medianFilter(houghDomain,6);
bitmask = houghDomain > 0.2;

houghDomain(~bitmask) = 0;

figure('Name','Hough_hotspot','NumberTitle','off');imshow(
    houghDomain);
C = 0;
```