עבודה 4 מבוא לעיבוד תמונה

מגישים:

נתן דוידוב 211685300

ניקולאי קרחמלוב 320717184

Edge Detection .1

Reading the Image 1.1

ראשית קראנו את התמונה ונרמלנו אותה

```
%% 1.1.1 Reading the Image
img = imread('cameraman.tif');

img_double = im2double(img);
minImg = min(img_double(:));
maxImg = max(img_double(:));
norm_img = ((img_double-minImg)/(maxImg-minImg));
```

Prewitt Edge Detector 1.2

1. כעת כתבנו פונקציה שעושה את Prewitt Edge Detector

```
function edge_after_thresh = dip_prewitt_edge(img,thresh)
    G_px = [-1 0 1; -1 0 1; -1 0 1]/6;
    G_py = [1 1 1; 0 0 0; -1 -1 -1]/6;
    edge_x = conv2(img, G_px, "same");
    edge_y = conv2(img, G_py, "same");
    edge = sqrt(edge_x.^2 + edge_y.^2);
    edge_b = edge>thresh;
    edge_after_thresh = edge.*edge_b;
end
```

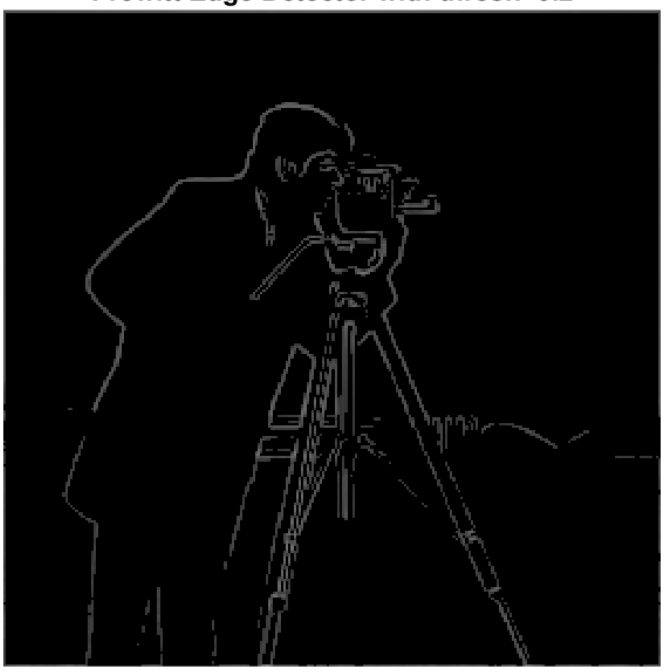
2. כעת נקרא לפונקציה מהסעיף הקודם עם שני thresholds שונים thresh = 0.1, 0.2

```
%% 1.2 Prewitt Edge Detector
edge_t1 = dip_prewitt_edge(norm_img, 0.1);
figure;
imshow(edge_t1);
title('Prewitt Edge Detector with thresh=0.1')
edge_t2 = dip_prewitt_edge(norm_img, 0.2);
figure;
imshow(edge_t2);
title('Prewitt Edge Detector with thresh=0.2')
```

ואלה התוצאות:

Prewitt Edge Detector with thresh=0.1

Prewitt Edge Detector with thresh=0.2



ניתן לראות כי עבור ה-thresh הגבוה יותר נקבל שהשפות דקות יותר מכיוון שאנו מסננים רק את השפות החזקות יותר ו"זורקים את הרעשים"

Canny Edge Detector 1.3

1. הפונקציה edge() של מאטלב מקבלת תמונה ומחזוריה תמונה שמכילה '1' במקומות שהפונקציה זיהתה שפות ואפסים בכל שאר התמונה. הפונקציה מקבלת בארגומנט גם את השם של האלגוריתם שבו תשתמש למצוא את השפות, האופציות הן: Sobel, Prewitt, Roberts, log, zerocross, Canny, approxcanny, כאשר הדיפולטי הוא Sobel.
הפונקציה גם מקבלת את ערך ה-threshold ומתעלמת מכל השפות שערכן נמוך יותר מ-threshold.
כאשר האלגורתים הוא Canny או approxcanny אז הפונקציה תקבל 2 ערכי threshold, היא תשתמש בגבוה מבינהם בשביל לשמור את ערכי השפה שגבוהים ממנו ותשתמש בנמוך על מנת למחוק את כל השפות עם ערך נמוך ממנו. עבור כל הערכים שנמצאים בין שני ה-thresholds האלגוריתם יבדוק האם השפות האלה מחוברות לשפות חזקות, אם כן ישמור אותן, אם לא אז ימחק אותן. אם נכניס רק ערך threshold אחד אז הוא ישמש כערך thresh הגבוה, והנמוך יהיה 0.4 כפול הערך שהוכנס
הפונקציה יכולה לקבל גם ארגומנט direction שהוא הכיוון של זיהוי השפות, שהערך הדיפולטי שלו הוא "both", אך ניתן לבחור גם "horizontal" או "vertical".
בנוסף ניתן לבחור גם את sigma שהוא הסטיית תקן של הפילטר הגאוסי, הערך הדיפולטי הוא $\sqrt{2}$.

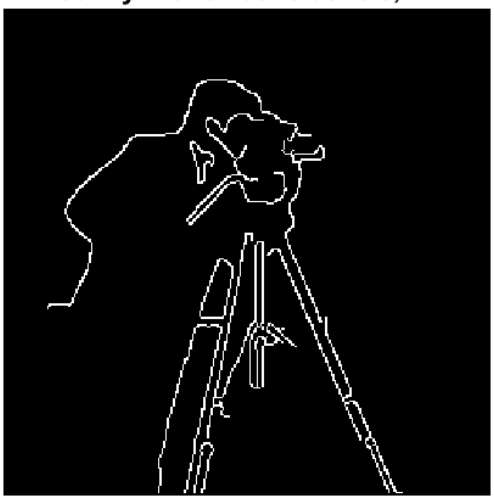
2. השתמשנו בפונקציית edge פעם אחת עם ה-threshold הדיפולטי שמחושב אוטומטית ע"י הפונקציה, ופעם אחת הגדנו את ה-threshold הגבוה להיות 0.8 והנמוך 0.2

```
%% 1.3 Canny Edge Detector
edge1 = edge(norm_img,"canny");
edge2 = edge(norm_img,"canny",[0.2,0.8]);
figure;
imshow(edge1);
title("Canny with default threshold");
figure;
imshow(edge2);
title("Canny with thresholds: 0.8,0.2");
```

ולהלן התוצאות:

Canny with default threshold

Canny with thresholds: 0.8,0.2



ניתן לראות שכאשר לא הכנסנו ערך ל-threshold קיבלנו את כל השפות בתמונה, וכאשר בחרנו את הערכים להיות 0.8,0.2 קיבלנו רק את השפות הבולטות ביותר ולכן זה תלוי בצורך שלנו, אם אנחנו מחפשים בתמונה רק את האדם אז הבחירה של 0.8,0.2 יותר טובה לנו, אבל אם אנחנו צריכים את כל השפות אז הדיפולטי עדיף.

3. נבדוק מהם ערכי ה-threshold הדיפולטים שהפונקציה מחזירה:

```
[edge3, threshold] = edge(norm_img,"canny");
disp(threshold)
```

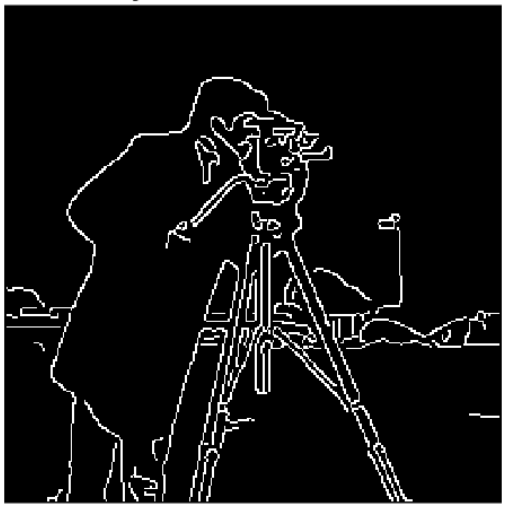
נקבל

```
    0.0312    0.0781
```

לכן נכניס לפונקציה ערכי threshold שקרובים לערכים האלה, אך נעלה קצת את הערכים כי לדעתנו כל השפות שמזוהות על האדמה לא הכרחיות.

```
edge4 = edge(norm_img,"canny",[0.1,0.2]);
imshow(edge4);
title("Canny with thresholds: 0.1,0.2")
```

Canny with thresholds: 0.1,0.2



כאן קיבלנו את התמונה עם כל השפות החשובות ולדעתנו אלה ערכים שיותר מתאימים מהערכים הדיפולטים.

Hough Transform .2

Hough Line Transform 2.1

בסעיף זה התבקשנו לעבוד עם קובץ התמונה floor.jpg. לשם כך תחילה המרנו אותה לgrayscale ולאחר מכן נירמלנו אותו לטווח הערכים [0,1] . השתמשנו בפונקציית edge() על מנת למצוא את כל השפות בתמונה. Edge() מקבלת את הפרמטרים הבאים: method, threshold, direction, filter, sigma.

Method- מגדיר באיזה שיטה המטודה עושה שימוש לשם חיפוש השפות בתמונה, כאשר האופציות הן Sobel (שהוא ברירת המחדל),Prewitt,Roberts,log,zerocross,Canny,approxcanny.

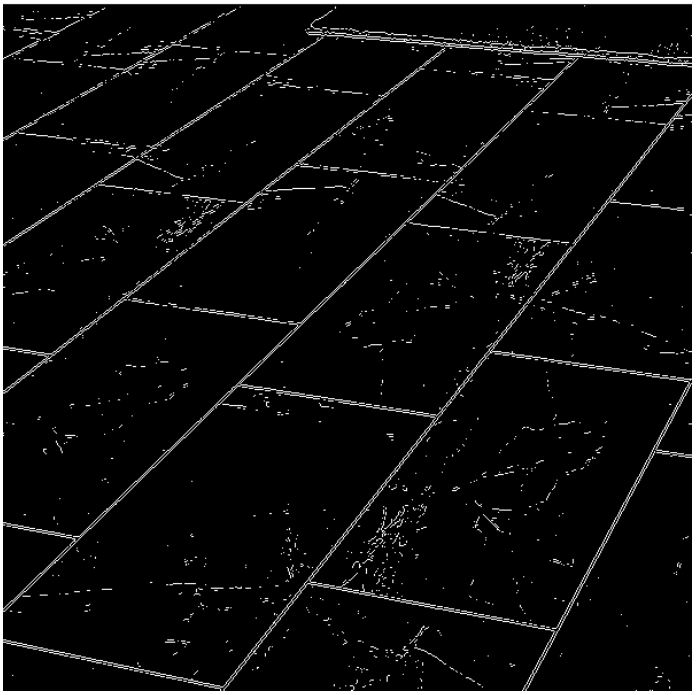
Threshold- ערך מספרי משטרתו לסנן שפות לא רלוונטיות. המטודה תזהה שפות רק אם תוך שימוש בשיטה ערך הפיקסל הנבדק יהיה גדול מהthreshold. כברירת מחדל ערך זה מחושב ע"י המטודה לכל תמונה.

Direction – מבטא את הציר לפיו המטודה מחפש תאת השפות בתמונה. כברירת מחדל החיפוש הוא בשני הצירים.

Filter- זהו פרמטר שרלוונטי רק כאשר משתמשים בzerocross method.

Sigma- זהו מפמרטר שרלוונטי רק כאשר משתמשיפ בCanny method.

להלן תמונת השפות כפי שחושבה ע"י מטלאב:



ניתן לראות כי לצד הקווים הברורים של הגבולות בין מרצפות הריצפה גם יש שפות שנובעות משינוי בצבע לאורך כל מרצפת.

על מנת להצליח לזהות אך ורק את הקווים הישרים שנוצרים ומהגבולות בין המרצפות רשמנו פונקצייה משלנו dip_hough_lines(BW,R0,Θ0) שמתטרתה להשתמש בhough line transform על מנת לזהות את הקווים הישרים. הזיהוי נעשה ע"י בניית מטריצת הצבעה שתחילה כולה אפסים ומימדיה נתונים ע"י טווחי הערכים של R ו –Θ , כאשר $R=[-\sqrt{M^2+N^2},\sqrt{M^2+N^2}]$ עבור תמונה בגודל MxN ואילו Θ מקבל את הערכים [-90,90] בקפיצות של R0 ו-Θ0 בהתאמה.

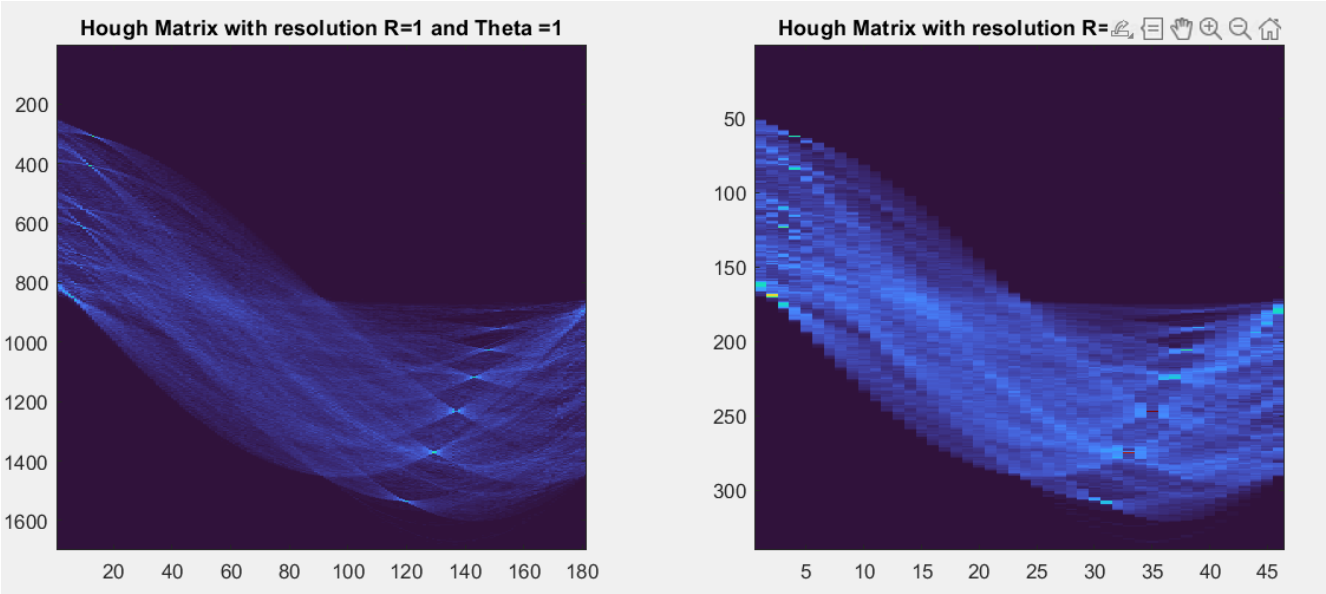
לאחר שבונים את המטריצה מחשבים לפי הפרמצטריזציה של קו ישר את r עבור כל פיקסל שפה בתמונת השפות וכל Θ, ומוסיפים הצבעה למקום ה(r, Θ) המתאים במטריצת ההצבעות. הקוד נראה כך:

```
%Natan Davidov 211685300, Nikolai Krokhmal 320717184

function edgeImg = dip_hough_lines(BW,R0,tta0)
    [N, M] = size(BW);
    % constructiong the R and Theta vectors which define the accumulation matrix size
    rMax = sqrt((N^2)+(M^2));
    R_vec = -round(rMax) : R0 : round(rMax);
    tta_vec = -90 : tta0 : 90;
    lenR = length(R_vec);
    lenTTA = length(tta_vec);
    %constructing accumulation matrix
    edgeImg = zeros(lenR,lenTTA);
    % voting for edges
    for y = 1:M
        for x = 1:N
            if BW(y,x) == 1
                for ttaIndx = 1:lenTTA
                    tta = tta_vec(ttaIndx);
                    r = round(x*cosd(tta)+y*sind(tta)); %calculate r for x,y of the edge pixel and all angles
                    [~, rIndx] = min(abs(R_vec-r)); % find the best r index in the matrix
                    edgeImg(rIndx,ttaIndx) =  edgeImg(rIndx,ttaIndx) + 1; %cast vote to bin
                end
            end
        end
    end


end
```

הרצנו את הפונקציה שכתבנו פעמיים על תמונת הריפצה, פעם עם פרמטרים (BW,1,1) ופעם עם הפרמטרים (BW,5,4). להלן התצאות של מטריצות ההצבעה המוצגות בעזרת imagesc(), colormap('turbo') לשם בהירות התצוגה:



ראשית נבחין כי הפרמטרים של ההרצה השנייה גרמו לתמונה ברזולוציה נמוכה יותר שכן גודל של כל פסיעה גדל ולכן מימדיי המטריצה קטנים יותר. בנוסף, מאחר והרזולוציה נמוכה יותר, קיבלנו מטריצת הצבעות עם בהירות גבוהה יותר, שכן מספר ההצבעות שתלוי במספר הפיקסלים השייכים לשפות בתמונה המקורית לא השתנה. כל מה שקרה הוא שכתוצאה מהורדת הרזולוציה ירדה כמות האפשרויות להצביע ולכן בכל פיקסל של המטריצה יש בממוצע ערך גבוהה יותר. אפקט זה תלוי ביחס בין המספר השורות המבטא את ערכי הרדיוס האפשריים למספר העמודות.

להורדת הרזולוציה יש יתרון בכך שהיא מורידה אז זמן הריצה והדרישה למפח אחסון (פחות זיויות Θ במערך ומטירצה קטנה יותר באופן כללי), יותר עמידות לרעש בתמונת השפות שכן נקודות שפה שמקורן ברעש והן בקירוב לישר פשוט יצביעו לאותו ישר. מצד שני הורדת הרזולוציה עלולה להביא גם לפספוס של שפות דקות או שפות כפולות ולירידה כללית בדיוק. ישרים קרובים במיוחד זה לזה יצביעו לאותו הפיקסל וכך אחד מהם יתפספס.

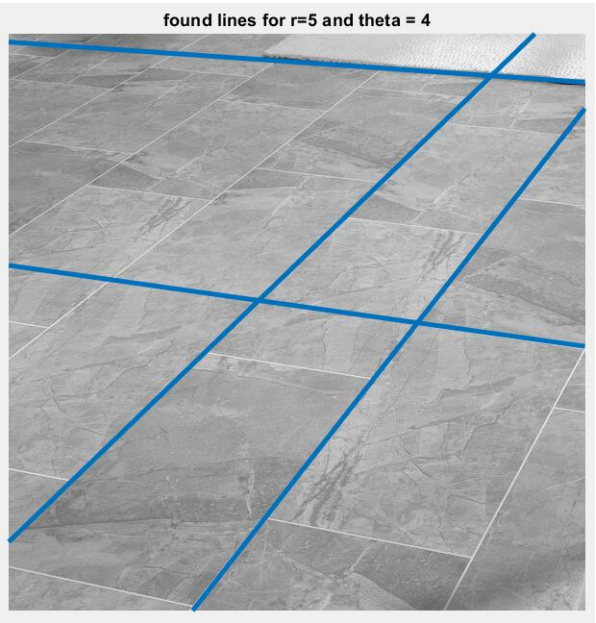
מנגד לרזולוציה גבוהה יש את היתרון של זיהוי פרטים קטנים במחיר של פחות עמידות לרעש. בנוסף, רזולוציה גדולה מידיי עלולה להביא למצב שכל נקודת שפה תצביע לפיקסל אחר ואז אלול להיווצר קושי במציאת נקודות החיתוך שיצביעו לנו על סבירות גבוהה לישר.

כעת נשתמש בתמונות השפות הנל כדי לסמן את ארבעת הישרים הכי דומיננטיים בתמונה. נשתמש בפונקציית houghpeaks(BW,4) כדי למצוא את ארבעת הפיקים במצטריצה ונשחזר מהם את הקווים. לשחזור הקווים בחזרנו ערכי x1=0 וx2=600 בתור שתי נק על התמונה המקורית, וחישבנו את ערכי y שלהם לפי הפרמטריזציה וערכי Θ ,r שקיבלנו ברשימת הפיקים. להלן הקוד:

```
% here we calculate and draw the lines
peaks1 = houghpeaks(houghLine1,4);
figure();
imshow(norm_img21);
title("found lines for r=1 and theta = 1")
hold("on");
%here we calculate the actual r and theta from the indices
[N,M] = size(BW);
rMax = sqrt((N^2)+(M^2));
R_vec = -round(rMax) : 1 : round(rMax);
tta_vec = -90:1:90;
%here we calculate 2 points and draw a line
for i = 1:length(peaks1)
    r = R_vec(peaks1(i,1));
    tta = tta_vec(peaks1(i,2));
    x1 = 0;
    x2 = 600;
    y1 = (r-x1.*cosd(tta))/sind(tta);
    y2 = (r-x2.*cosd(tta))/sind(tta);
    line([x1,x2],[y1,y2],'LineWidth',3);
end
```

```
%same here again
peaks2 = houghpeaks(houghLine2,4, "Threshold", 0.3*max(houghLine2(:)));
figure();
imshow(norm_img21);
title("found lines for r=5 and theta = 4")
hold("on");
R_vec2 = -round(rMax) : 5 : round(rMax);
tta_vec2 = -90:4:90;
for i = 1:length(peaks2)
    r = R_vec2(peaks2(i,1));
    tta = tta_vec2(peaks2(i,2));
    a1 = 0;
    a2 = 600;
    b1 = (r-a1.*cosd(tta))/sind(tta);
    b2 = (r-a2.*cosd(tta))/sind(tta);
    line([a1,a2],[b1,b2],'LineWidth',3);
end
```

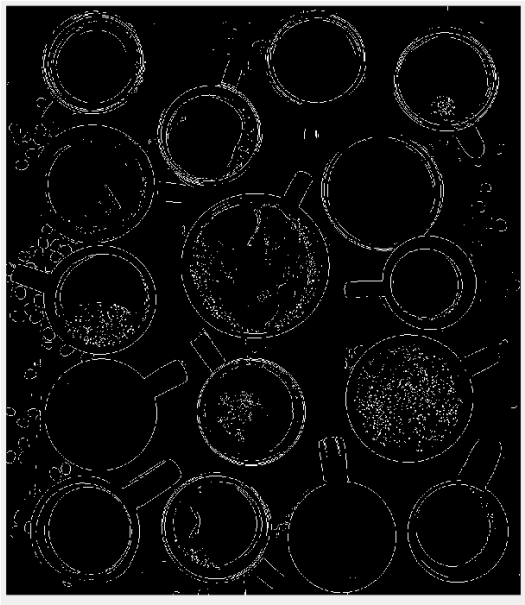
להלן התוצאות:



ניתן לראות כי עבור שני הסטים של R0 ו-Θ0 התקבלו תוצאות שונות. בהרצה הראשונה קיים ישר שלא קיים בהרצה השנייה ולהיפך. על פנוי עבור ערכי R0 ו-Θ0 גדולים יותר נצפה לדיוק נמוך יותר של זיהוי ישרים אך במקרה הזה קשה לומר שהדיוק ירד. מצד שני נשים לב שעל מנת לקבל 4 ישרים עבור הערכים (5,4) נאלצנו להוריד את הthreshold לערך של 0.3 מערך ההצבעה המקסימלי, אחרת ()Houghpeaks הצליח לזהות רק שלושה ישרים, כנראה כי הצבעות שהיו מיועדות לישר אחד נפלו בפיקסל של ישר אחר ואז הוא לא עבר את אחוז החסימה.

Hough Circle Transform 2.2

בסעיף זה התבקשנו לעבוד עם קובץ התמנה coffee.jpg על מנת לזהות מעגלים בתמונה. תחילה המרנו את התמונה לgrayscale ונירמלנו אותה ולאחר מכן התשמשנו שוב במטודת ()edge כדי לייצר תמונת שפות בדיוק כמו בסעיף הקודם. להלן תמונת השפות.



ניתן לראות שגם כאן בנוסף לעיגולים של כוסות הקפה ישנן עוד שפות רבות שמטלאב זיהה, בדיוק כמו בסעיף הקודם.

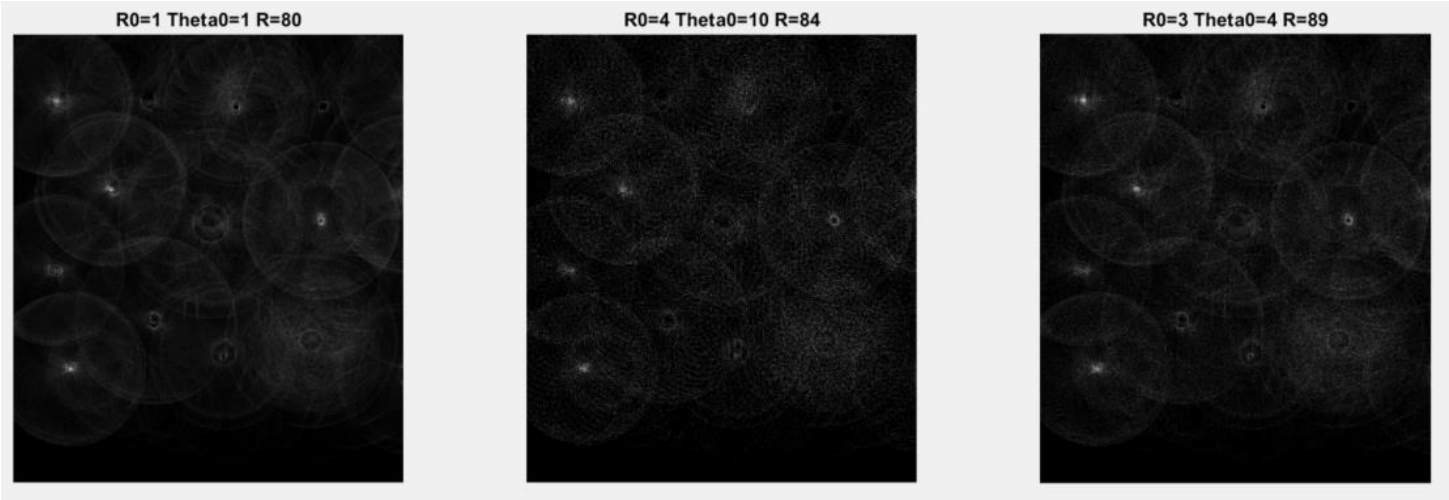
על מנת לזהות את המעגלים בתמונה כתבנו את הפונקציה dip_hough_circles(BW,R0, Θ0) שמטרתה להחזיר מטריצת הצבעה עבור מרכזי מעגלים ורדיוסים אפשריים. הפונקציה תחילה בומה מטריצת אפסים תלת מיימדית כאשר שני המימדים הראשונים זהים למיימדי התמונה והמימד השלישי הוא עבור כל רדיוס של מעגל אותו נחפש. במקרה שלנו עבור R0=1 גודל המטריצה היה 1024x894x21, שכן חיפשנו מעדלים ברדיוס בין 80 ל-100. לאחר בניית המטריצת האפסים תהליך ההצבעה כלל מעבר על כל הפיקסלי השפות בתמונת השפות וחישוב מעגלים ברדיוס המתאים סביב כל פיקסל כזה. לאחר מכן הכנסנו למטריצת ההצבעה כל חלק של מעגל שהיה בטווח התמונה. להלן הקוד:

```
%Natan Davidov 211685300, Nikolai Krokhmal 320717184

function houghCircles = dip_hough_circles(BW, R0, tta0)
    [M,N] = size(BW);
    Rmin = 80;
    Rmax = 100;
    rVec = Rmin:R0:Rmax;
    houghCircles = zeros(M,N,length(rVec));
    for y = 1:N
        for x=1:M
            if BW(x,y) == 1
                for rIndx = 1:length(rVec)
                    for tta = 0:tta0:359
                        r = rVec(rIndx);
                        a = round(y-r*cosd(tta));
                        b = round(x-r*sind(tta));
                        if(a>0&&b>0&&a<=size(BW,1)&&b<=size(BW,2))
                         houghCircles(a,b,rIndx) = houghCircles(a,b,rIndx) + 1;
                        end
                    end
                end
            end
        end
    end


end
```

זיהוי המעגלים נעשה על ידי זיהוי ערכי המקסימום עבור כל רדיוס בנפרד. ערכים אלו הם נק החיתוך של מעגלים בעלי רדיוס קבוע שמרכזם הוא נמצא על פני שפת מעגל בעל רדיוס זהה, כך שנק החיתוך היא מרכז המעגל. זמן הריצה של האלגוריתם תלוי במספר הרדיוסים הנבדקים והערך Θ0 שמגדיר את הצפיפות של כל מעגל. אם נוריד את הרזולוציה (נגדיל את Θ0 ואת R0) אז נוכל להוריד את זמן הריצה של אלגוריתם החיפוש שכן לכל פיקסל שפה של התמונה המקורית נוריד את מספר הרדיוסים ונחשב פחות נק להצבעה במטריצה ההצבעה (נחשב את נק המעגל המתקבל בצפיפות נמוכה יותר מאחר והקפיצת הזווית יגדלו). לשם המחשה הרצנו את האלגוריתם עם הערכים (1,1) ומדדנו את הזמן עד להשלמה. לקח 21.68 שניות. לאחר משחק מעט עם הערכים מצאנו שלערכים (3,4) זמן הריצה של החיפוש היה 1.96 שניות עבור אותו הזיהוי. סהכ פי 11 יותר מהיר. להן תמנות מתוך מטריצת ההאף התלת מיימדית המתקבלת עבור שתי ההרצות הנ"ל והרצה נוספת עם הערכים (4,10):



ניתן לשים לב כי התמונה השמאלית נראית פחות רועשת ומנוקדת לעומת התמונות האחרות מאחר והפרמטר Θ0 שלה שווה 1.

את ההשוואה בין התוצאות ביצאנו על ידיי זיהוי של אותן 5 כוסות קפה. את זיהוי הכוסות עשינו ע"י כתיבת הפונקציה dip_houghpeaks3d(HoughMatrix). הפונקציה מקבלת את מטריצת ההצבעת התלת מיימדית, משטחת אותה, מחפשת ערכי מקסימום ומחזירה את האינדקסים שלהם בתור רשימה. בתוך הפונקציה אנחנו מחפשים עד 20 ערכי מקסימום שונים, והגדרנו

threshold = 0.6*max(HoughMatrix) כדי להימנע מערכי מקסימום נמוכים ולאפשר השוואה בין זמן הריצה. הערך נבחר כך שנקבל 5 מעגלים שונים עבור הרצה עם הפרמטרים (1,1). להלן הקוד:

```
%Natan Davidov 211685300, Nikolai Krokhmal 320717184

function peaks = dip_houghpeaks3d(HoughMatrix)
    peakAmount = 20;
    thresholdCoefficient = 0.6;
    peaks = zeros(peakAmount,3);
    [maxVal ,~] = max(HoughMatrix(:));
    for i=1:peakAmount
        [val, idx] = max(HoughMatrix(:));
        if val > thresholdCoefficient * maxVal
            [idx1, idx2, idx3] = ind2sub(size(HoughMatrix), idx);
            peaks(i ,:) = [idx1, idx2, idx3];
            HoughMatrix(idx1,idx2 ,idx3) = 0;
        end
    end
end
```

לאחר קבלת רשימת הפיקים שמכילה את קורדינטות מרכזי המעגלים והרדיוסים המתאימים להן ציורנו את המעגלים על תמונת כוסות הקפה עבור כל אחד מההרצות:

כפי שאמרנו, ניתן לראות כי שמרנו על דיוק של זיהוי בין הרצות על ערכים (1,1) ו- (3,4), תוך הורדת זמן הריצה בערך פי 11. גם עבור הרצה עם פרמטרים (4,10) קיבלנו זיהוי של 5 כוסות, 4 מתוכן משותפות לאלו שזיהינו בהרצות אחרות ואחת שונה. באופן כללי, בדומה ל Hough Lines transform הורדת הרזולוציה מורידה את זמן החישוב, הדרישה בזיכרון, ומעלה את העמידות לרעש על חשבון הדיוק והיכולת לזהות מעגלים עדינים. גם כאן רזולוציה נמוכה יותר משמעותה פחות הצבעות פר רדיוס ופחות רדיוסים שונים שהאלגוריתם בודק. כתוצאה מכך יכולים להתפספס מעגלים שהרדיוס שלהם לא נבדק. מנגד, לעלאת הרזולוציה תאפשר זיהוי מעגלים עדינים יותר, לכל מרכז מעגל יהיו הרבה יותר הצבעות מה שיהפוך אותו למובהק יותר ודומיננטי יותר, במחיר של זמן חישוב ארוך יותר ועמידות מופחת לרעשים. במקרה שלנו נראה שקפיצות של 4 מדלגות על הרדיוס של הכוס קפה שמצליחות לזהות ההרצות האחרות, וכתוצאה מכך ההצבעות של פיסקלי הספות של הכוס הזו לא מתלכדות ליידי מרכז ברור. את מקומו עם כן תפסה כוס קפה אחרת. בנוסף נשים לב לכל כוס קפה כמעט יש מספר של זיהויים, מאחר והאלגוריתם שלנו מחפש עד 20 מעגלים, ומעגלים רבים בתמונה הם בעלי שפות עבות מעט (הבדל של פיקסלים בודדים). כתוצאה מרזולוציה גבוהה מידיי אנחנו מזהים מעגלים ברדיוסים כמעט זהים (נניח 88 פיקסלים, 89 פיקסלים , 90 פיקסלים וכו) ועם מרכז כמעט זהה – מה שמוביל לזיהוי של אותו מעגל מספר רב של פעמים ברמות דיוק שונות.