(12967) ראיה אנושית גישה חישובית | תרגיל 2

שם: רונאל חרדים | ת"ז:208917641

שאלה 6

(א)

נשים לב כי העץ זז מהר יותר אשר הפרחים.

(ב)

הרצתי את האלגוריתם Full_LK_alg על שתי תתי התמונות הבאות, עם מסכה בגודל התמונות:

נשים לב כי אין חפיפה בין שתי התמונות, לכן ככל שמספר האיטרציות עלה, ה $velocities$ בין התמונות עלה.

לדוגמא:

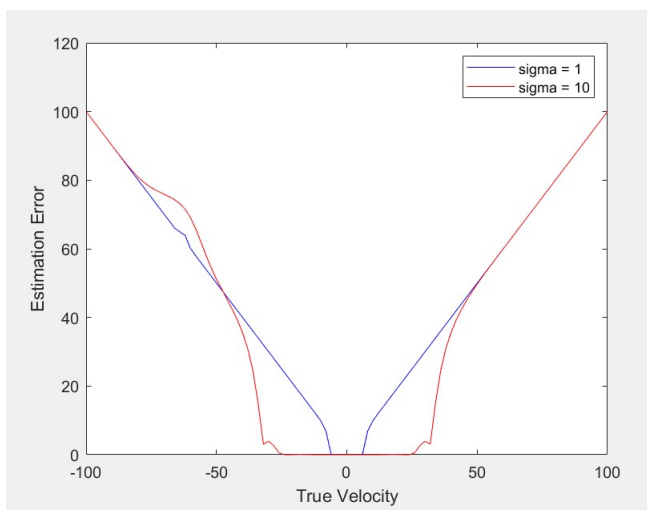
```
num of iterations:
    10


v =

   -4.2774    3.2427

num of iterations:
   100


v =

  -16.4259   12.1119

num of iterations:
   200


v =

  -33.1666   18.2562
```

שאלה 7

(א)

הרצתי את הפונקציה עם מהירות משתנה בטווח $[-100, 100]$ בקפוצות של 2 .

הגרף שהתקבל הוא הגרף הבא:



הסבר:

כאשר $\sigma = 10$**:** נשים לב בגרף כי ככל שה $velocity$ עולה (בכיוון החיובי או השלילי) - השגיאה עולה.

כשהמהירות יורדת - השגיאה יורדת.

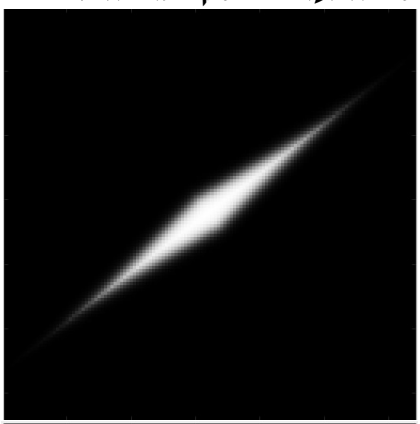
נשים לב כי השגיאה נמוכה לאורך זמן, כי יש חפיפה יותר גדולה בין התמונות כי $\sigma = 10$. לכן יותר קל למצוא את v שיתאים את התמונות.

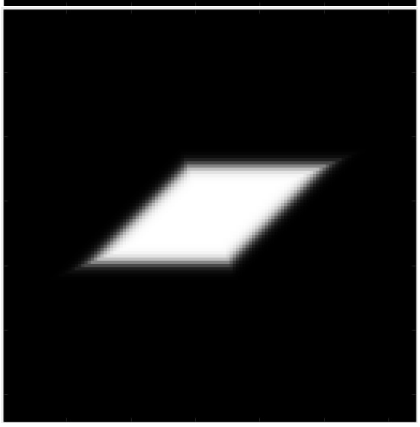
כאשר $\sigma = 1$**:** הגאוסיאן צר יותר, השגיאה יורדת אך באופן איטי יותר. כי יש פחות חפיפה בין הגאוסיאנים. ולכן כשאין חפיפה ה error עולה. כי אין שום v שיתאים את התמונות.

שאלה 8

(א)

שני המעויינים שקיבלתי הם:

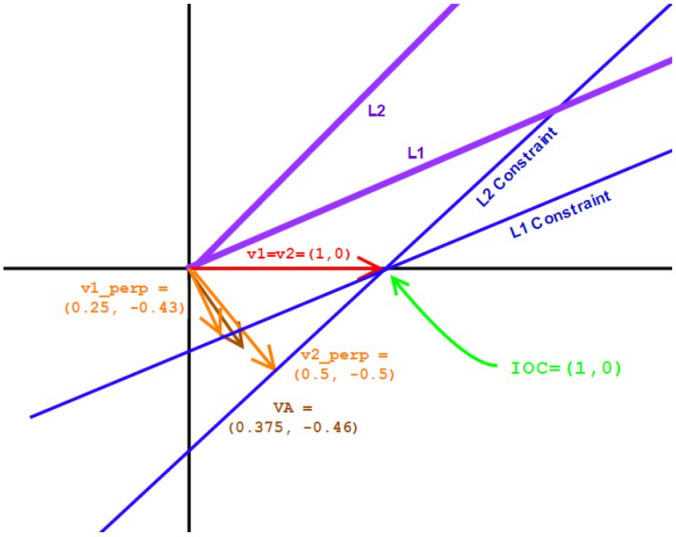




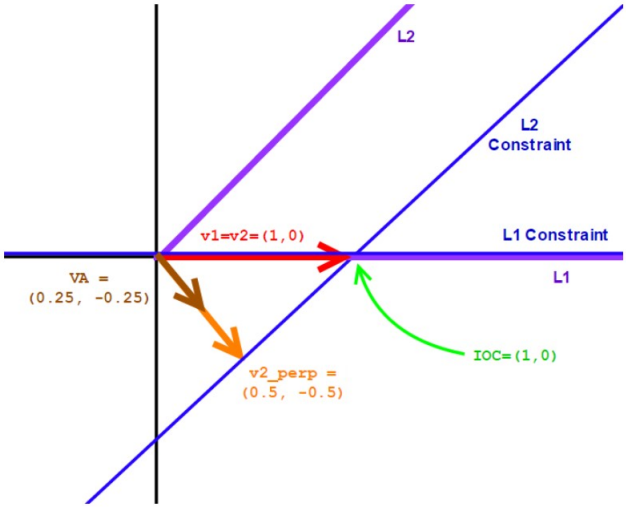
(ב)

נחשב את IOC **ו** VA **של שני המעוינים:**

המעויין הצר:



המעויין הרחב:

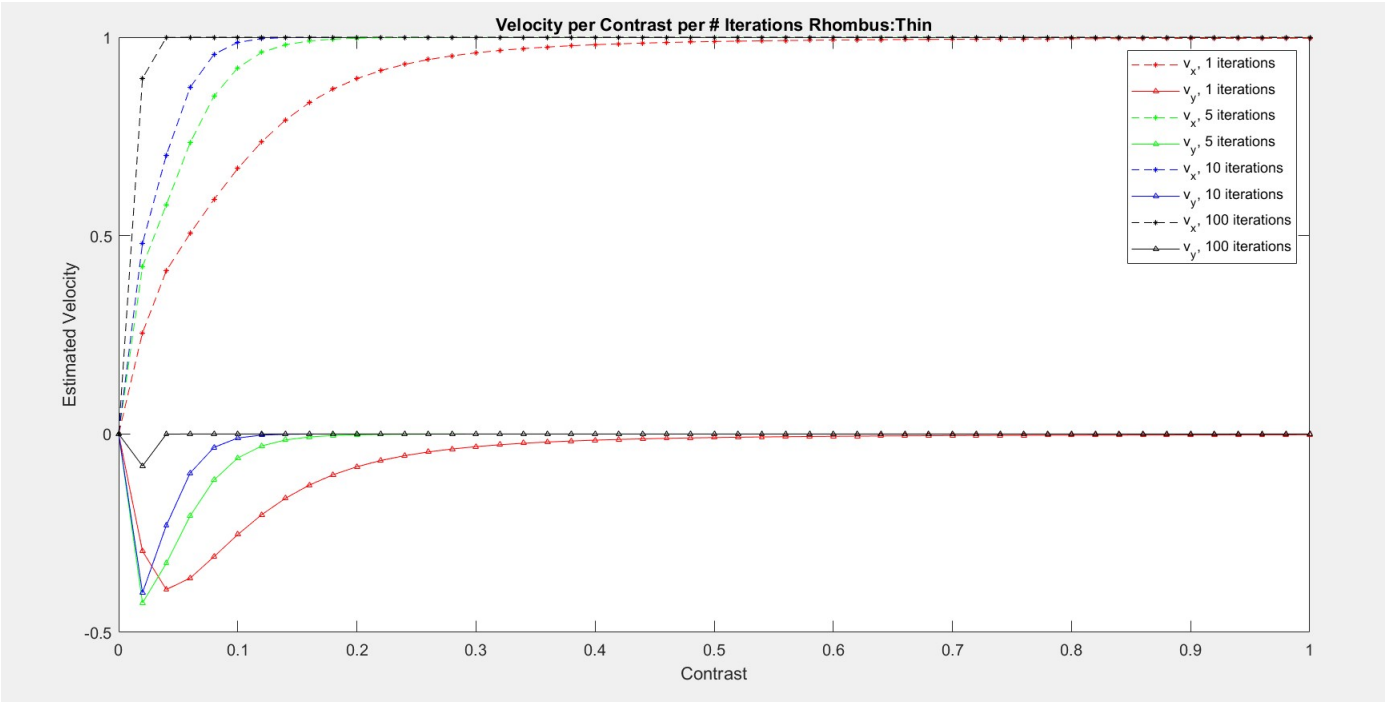


(ג)

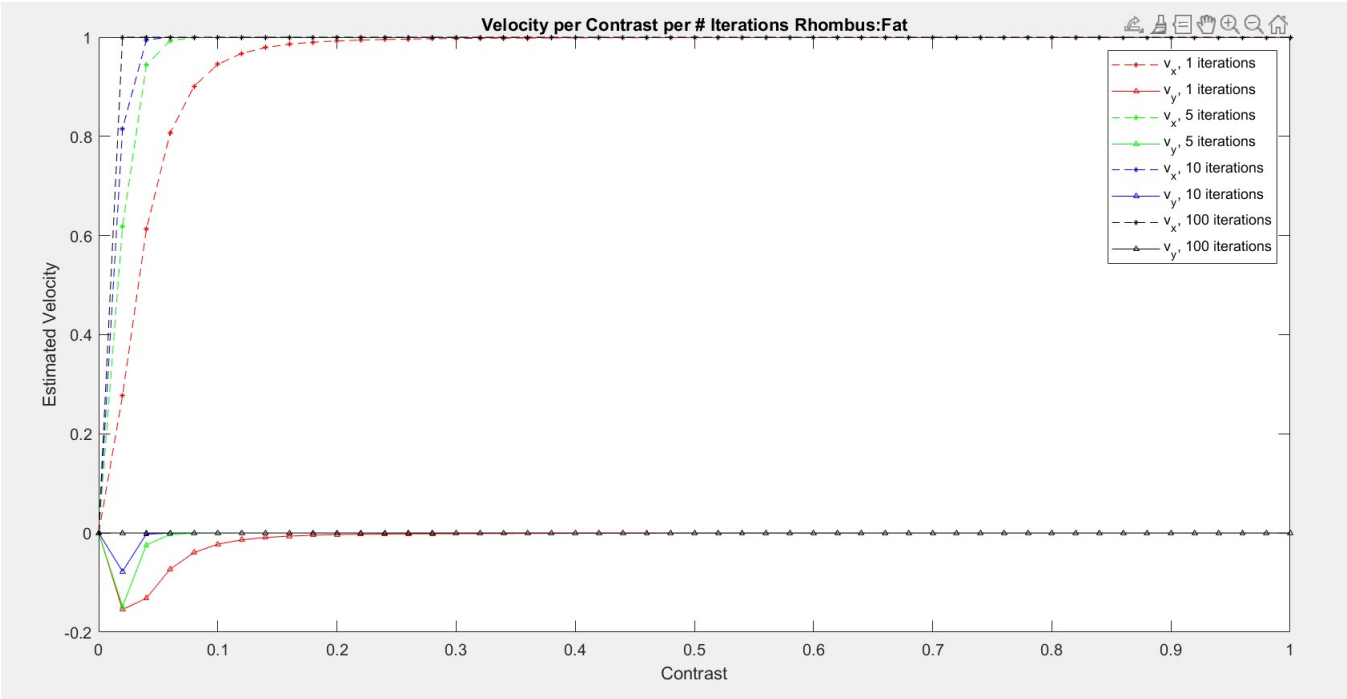
הרצתי עם מספר האיטרציות הבא: $[1, 5, 10, 100]$
עם זוויות בטווח $[0, 1]$ עם קפיצות של 0.2.

הגרפים שקיבלתי הם הגרפים הבאים:

עבור המלבנים הצרים:



עבור המלבנים הרחבים:



הסבר:

נשים לב, כי ככל שהניגודיות עולה, וככל שמספר האיטרציות עולה, הגרף מתכנס ל V מהר יותר.

כמו כן ככל שהמלבן רחב יותר - הגרף מתכנס ל V מהר יותר.

משום שכאשר המלבן רחב יש יותר חפיפה ויותר קל לחשב את V במעט איטרציות. כמו כן אם הניגודיות גבוה יותר, אזי יותר קל להבין איפה כל מלבן ומה שטח החפיפה ולכן ההתכנסות מהירה יותר.

```
function [Ix, Iy, It] = ImageDerivatives(im1, im2)
    % Computes the derivatives of two images across the X dimension, the Y dimension, and time.

    Ky = 0.25*[-1,-1 ; 1,1];
    Kx = 0.25*[1,-1 ; 1,-1];
    Kt = 0.25*[1,1 ; 1,1];

    % Compute the spatial derivatives using a Sobel filter
    Ix = conv2(im1, Kx, 'same') + conv2(im2, Kx, 'same');
    Iy = conv2(im1, Ky, 'same') + conv2(im2, Ky, 'same');
    It = conv2(im2, Kt, 'same') - conv2(im1, Kt, 'same');
end

function v = LK_alg(I1, I2, lamda, mask, v_initial, num_iterations)

    % Initialize the velocity
    v_prev = v_initial;
    new_mask = mask;

    % Iterate for num_iterations
    for i = 1:num_iterations
        % Warp I2 using the current velocity estimate
        [Iw2, warpMask] = warp(I2, v_prev);

        % Apply mask to the warpMask
        new_mask = new_mask .* warpMask;


        % Compute the image gradient
        [Ix, Iy, It] = ImageDerivatives(I1, Iw2);
        Ixt = Ix .* new_mask;
        Iyt = Iy .* new_mask;
        Itt = It .* new_mask;

        xx = sum(sum(Ixt .^ 2)) + lamda;
        yx = sum(sum(Ixt .* Iyt));
        yy = sum(sum(Iyt .^ 2)) + lamda;
        A = [xx, yx; yx, yy];


        xt = sum(sum(Ixt .* Itt));
        yt = sum(sum(Iyt .* Itt));
        B = -([xt; yt]);


        invA = inv(A);
        v_prev = v_prev + (invA * B);
    end
    v = v_prev;
end


function blured_im = blur_downsample(im)
    % Read the Gaussian kernel from file (or define your own)
    load('GaussKernel.mat', 'GaussKernel');
    % Convolve the image with the Gaussian kernel
    blured_im = conv2(im, GaussKernel, 'same');
    % Downsample the image by taking every second pixel in both dimensions
    blured_im = blured_im(1:2:end, 1:2:end);
end

function v = Full_LK_alg(im1, im2,lamda, mask, num_iterations)
    blured_im1 = blur_downsample(im1);
    blured_im2 = blur_downsample(im2);
    new_mask = mask(1:2:end, 1:2:end);
    v_initial = [0 0]';
    v_blurred = LK_alg(blured_im1, blured_im2 , lamda, new_mask, v_initial, 1);
    v = LK_alg(im1, im2 , lamda, mask, 2*v_blurred, num_iterations);
    v = v';
end

function q6
    % flower garden
    im1 = double(imread('flower-i1.tif'));
    im2 = double(imread('flower-i2.tif'));
```

```
    [M] = mymovie(im1, im2, 0);

    lamda = 0;

    %create subimages
    subimage1 = im1(1:end, 1:90);
    subimage2 = im1(1:end, 91:end);
    mask = ones(120,90);

    show(subimage1);
    show(subimage2);

    for num_iterations = [10, 100, 200]
        disp("num of iterations:")
        disp(num_iterations)
        v = Full_LK_alg(subimage1, subimage2,lamda, mask,num_iterations)
    end
end


function q7

    x_axis = -100:2:100;

    for sigma = [1, 10]     % For each sigma: 1, 10
        err = [];

        % Set "base" Gaussian
        g_base = GausSpot(128, sigma, [0 0]);

        for x = x_axis    % For each velocity

            % Set Gaussian after movement and calculate its velocity and error
            g_moved = GausSpot(128, sigma, [x 0]);
            v = Full_LK_alg(g_base, g_moved, 0, ones(128), 1);
            err = [err abs(v(1) - x)];
        end

        % Draw plots (sigma 1 in blue and sigma 10 in red)
        if sigma == 1
            plot(x_axis , err , 'b');
        else
            hold on
            plot(x_axis , err , 'r');
        end
    end

    % Label axis
    xlabel('True Velocity');
    ylabel('Estimation Error');
    legend('sigma = 1','sigma = 10');
end


function [] = q8()
    flags = {[1, 'Fat'] , [0, 'Thin']};

    for i = 1:length(flags)
        tuple = flags{i};
        fatFlag = tuple(1);
        type = tuple(2:end);
        lambda = 0.01;
        conts = 0:0.02:1;

        figure;

        iter_arr=   {[1 , "r"], [5,"g"] ,[10,"b"] ,[100,"k"]} ;
        for j = 1:length(iter_arr)
            tuple2 = iter_arr{j};
            num_iterations = str2num(tuple2(1));
            color = tuple2(2);
            res_x = [];
            res_y = [];

            for cont = conts
```

```
                [rhomb1, rhomb2] = rhombusMovie(fatFlag, cont);
                v = Full_LK_alg(rhomb1, rhomb2, lambda, ones(size(rhomb1)), num_iterations);
                res_x = [res_x v(1)];
                res_y = [res_y v(2)];
            end

            plot(conts, res_x, color, 'Marker', '*', 'MarkerSize', 3, 'LineStyle', '--');
            hold on;
            plot(conts, res_y, color, 'Marker', '^', 'MarkerSize', 3);

            % Label axis

            title(['Velocity per Contrast per # Iterations Rhombus:', type]);

            xlabel('Contrast');
            ylabel('Estimated Velocity');

            legend('v_x, 1 iterations', 'v_y, 1 iterations', ...
                    'v_x, 5 iterations', 'v_y, 5 iterations', ...
                    'v_x, 10 iterations', 'v_y, 10 iterations', ...
                    'v_x, 100 iterations', 'v_y, 100 iterations');
        end
    end
end
```