600.641 HOMEWORK 1

LIUJIANG YAN

Written Assignment 1

(a) The disk lies in a plane parallel to the image plane. Consider the contour of the disk, which is a circle of radius r, and can be presented as [x,d,z] and has following relation

$$(x - x_0)^2 + (z - z_0)^2 = r^2$$

Through the pinhole [x,d,z] will be mapped to [x',-f,z'], satisfying following relation

$$\frac{x}{x'} = \frac{d}{-f} = \frac{z}{z'}$$

So the expression of the mapped disk's contour can be derived by,

$$x = x'\frac{d}{-f};\ z = z'\frac{d}{-f};$$

$$(x-x_0)^2+(z-z_0)^2 = r^2 \rightarrow (x'\frac{d}{-f}-x_0)^2+(z'\frac{d}{-f}-z_0)^2 = r^2 \rightarrow (x'-x_0')^2+(z'-z_0')^2 = (\frac{d}{f}r)^2$$

From the expression, we know that after the mapping through pinhole the image of the disk is still circular but with a different radius as $\frac{d}{f}r$.

(b) From the formula we derived for the image of the disk, the relations between radius and area of the original disk and the image disk are,

$$\frac{r}{r'} = \frac{f}{d} \rightarrow \frac{area}{area'} = (\frac{f}{d})^2 \rightarrow area' \propto d^2$$

As the problem states, $area' = 1mm^2$ and $d = 1meter$, if the distance is doubled, we have,

$$area_{double} = aera'\frac{1}{2}^2,\ aerarea_{double} = 0.25mm^2$$

(c) When the disk is replaced by the sphere, the image will change from a circle to an ellipse, if the sphere is not placed in the lie on the optical axis.
For a generally case, a sphere with radius of r is placed in (x,d,z). Knowing that all the light from the sphere and pass the pinhole to the image plane are limited in the range consisted of the lines tangent to the sphere and pass the pinhole, which make up as a contour of the sphere. The imaging process could be sketched as Figure 1.
So the problem of mapping the sphere through a pinhole the image plane can be simplified as a disk with contour of B,D,G pointed in the sketch, which is ellipse. The image of such a ellipse shape through a pinhole will be a skew ellipse shape, known from the sketch. The reason is that plane where the ellipse shape lies is not parallel to the image plane.

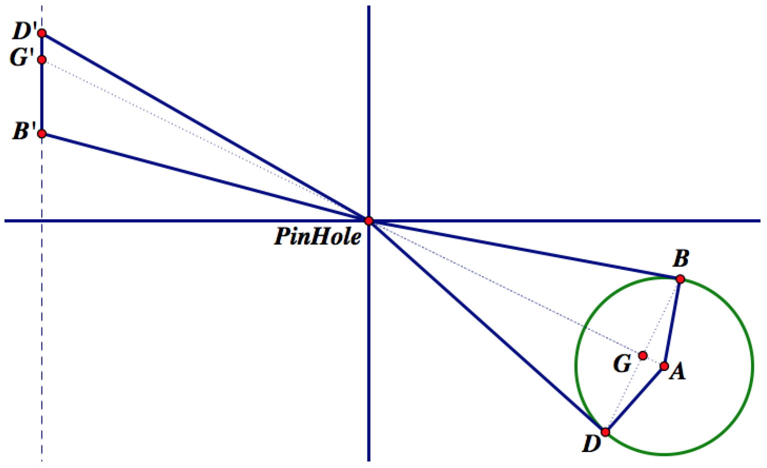

FIGURE 1. Written Assignment 1(3)

Written Assignment 2
A set of parallel lines corresponds to a vanishing point in the image plane through the pinhole. All the lines in the plane are perpendicular to the normal vector of the plane. Knowing that the plane is,

$$Ax + By + Cz + D = 0;$$

the normal vector can be derived as,

$$n = [A, B, C];$$

Let $[x_1, x_2, x_3]$ present the direction of the lines in the plane, then the inner product between this line and the normal vector is zero, due to the perpendicular relation.

$$[x_1, x_2, x_3] \cdot [A, B, C] = 0;$$

$$x_1 A + x_2 B + x_3 C = 0;$$

so that the vector of lines in the plane can be represented by two variables x_1 and x_2.

$$v = [x_1, x_2, -\frac{x_1 A + x_2 B}{C}]$$

We need two points to determine a line. Let pick [1,1] and [1,-1] for $[x_1, x_2]$. Knowing that this two line pass the pinhole, which is the origin of the frame. The lines' functions can be presented as,

$$\frac{x}{1} = \frac{y}{1} = \frac{z}{-\frac{A+B}{C}}$$

$$\frac{x}{1} = \frac{y}{-1} = \frac{z}{-\frac{A-B}{C}}$$

The image plane lies in the y=-f. Substitute y=-f to functions gets the responding two points,

$$[-f, -f, \frac{A+B}{C} f],\ [f, -f, -\frac{A-B}{C} f]$$

Eventually, from the derived two points we get the line consisted of all the vanishing points.

$$\frac{x+f}{-2f} = \frac{Cz - (A+B)f}{2Af},\ y = -f;$$

Programming Assignment 1

(a) Write a MATLAB function named p1 that converts a gray-level image to a binary one using a threshold value.

```
function binary_out = p1(gray_in, thresh_val)
    % create the output of exact size as input
    binary_out = zeros(size(gray_in));
    % find the index of pixels with value over the thresh_val
    index = (gray_in > thresh_val);
    % assign 1 to those pixels
    binary_out(index) = 1;
    % assign 0 to others
    binary_out(~index) = 0;
end
```

With threshold of 150, p1() can clear the images and leave only the objects.

```
gray_in_1 = imread('two_objects.pgm');
gray_in_2 = imread('many_objects_1.pgm');
gray_in_3 = imread('many_objects_2.pgm');
binary_out_1 = p1(gray_in_1, 150);
binary_out_2 = p1(gray_in_2, 150);
binary_out_3 = p1(gray_in_3, 150);
```

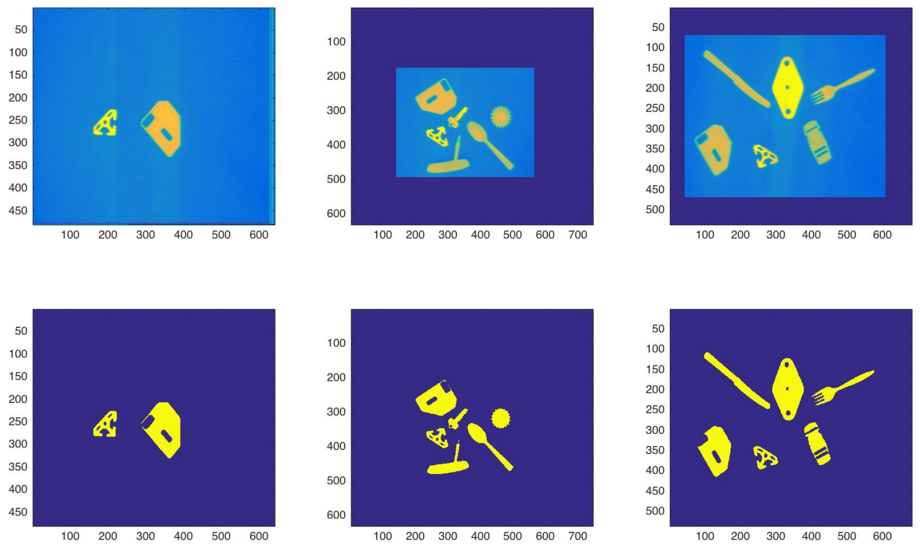

Figure 2. Programming Assignment 1(a)

(b) Implement the sequential labeling algorithm (and name the function p2) that segments a binary image into several connected regions.

Key point: use vector as a tree to serve as an equivalence table.
e.g., [0,1,2,3,4] means the parent node of 5th position is '4', and the parent node of 4th position is '3'. Zero value means this position is the root and other child node of it should be aligned its label.

```
function labels_out = p2(binary_in)

    % labelarray for tracking the equivalence
    % use vector as rooted tree
    % the value at ith position refers to its parent node.
    labelarray = zeros(1,1000);

    [row, column] = size(binary_in);
    newlabel = 1;
    % the first pixel
    if binary_in(1,1) == 1
        labels_out(1,1) = newlabel;
        newlabel = newlabel + 1;
    end
    % the first column of image
    for j = 2:column
        if binary_in(1,j) == 0
            labels_out(1,j) = 0;
        else
            if labels_out(1,j-1) ~= 0
                labels_out(1,j) = labels_out(1,j-1);
            else
                labels_out(1,j) = newlabel;
                newlabel = newlabel + 1;
            end
        end
    end
    % the first row of image
    for i = 2:row
        if binary_in(i,1) == 0
            labels_out(i,1) = 0;
        else
            if labels_out(i-1,1) ~= 0
                labels_out(i,1) = labels_out(i-1,1);
            else
                labels_out(i,1) = newlabel;
                newlabel = newlabel + 1;
            end
        end
    end
    % assign the labels to other pixels
    for i = 2:row
        for j = 2:column
            % if the value of pixel is 0, then not labeled
            if binary_in(i,j) == 0
                labels_out(i,j) = 0;
            else
                % A B
                % C D
                % A labeled; then label D = label A;
                if labels_out(i−1,j−1) ~= 0
                    labels_out(i,j) = labels_out(i−1,j−1);
                % A not labeled; B labeled and C not labeled;
                % then label D = label B;
                elseif labels_out(i−1,j) ~= 0 && labels_out(i,j−1) == 0
                    labels_out(i,j) = labels_out(i−1,j);
                % A not labeled; C labeled and B not labeled;
                % then label D = label C;
                elseif labels_out(i−1,j) == 0 && labels_out(i,j−1) ~= 0
                    labels_out(i,j) = labels_out(i,j−1);
                % A not labeled; B and C labeled equally;
                % then label D = label B = label C;
                elseif labels_out(i−1,j) == labels_out(i,j−1) ...
                        && labels_out(i−1,j) ~= 0 && labels_out(i,j−1) ~= 0
                    labels_out(i,j) = labels_out(i−1,j);
                % A not labeled; B and C labeled inequally;
                % then label D = label B, notes that label B = label C;
                elseif labels_out(i−1,j) ~= labels_out(i,j−1) ...
                        && labels_out(i−1,j) ~= 0 && labels_out(i,j−1) ~= 0
                    labels_out(i,j) = labels_out(i−1,j);

                    % find the root of tree which B belongs to
                    root_right = labels_out(i−1,j);
                    while labelarray(root_right) ~= 0
                        root_right = labelarray(root_right);
                    end
                    % find the root of tree which C belongs to
                    root_left = labels_out(i,j−1);
                    while labelarray(root_left) ~= 0
                        root_left = labelarray(root_left);
                    end
                    % if they are the same, do nothing;
                    % otherwise, assign the C tree root to the B tree root
                    if root_right ~= root_left
                        labelarray(root_right) = root_left;
                    end

                % A B C not labeled, assign a new label to D;
                else
                    labels_out(i,j) = newlabel;
                    newlabel = newlabel + 1;
                end
            end
        end
    end
    % deal with the equivalence tree
    labels = [];
    for i = 1:length(labelarray)
        j = i;
        while labelarray(j) ~= 0
            j = labelarray(j);
        end
        index = (labels_out==i);
        labels_out(index) = j;
        if ~ismember(j, labels)
            labels = [labels, j];
        end
    end
    % reassign the labels to the image
    reassign = 1;
    for i = 1:length(labels)
        index = (labels_out==labels(i));
        labels_out(index) = reassign;
        reassign = reassign + 1;
    end

end
```

From the figures below, p2() can clearly label different parts.

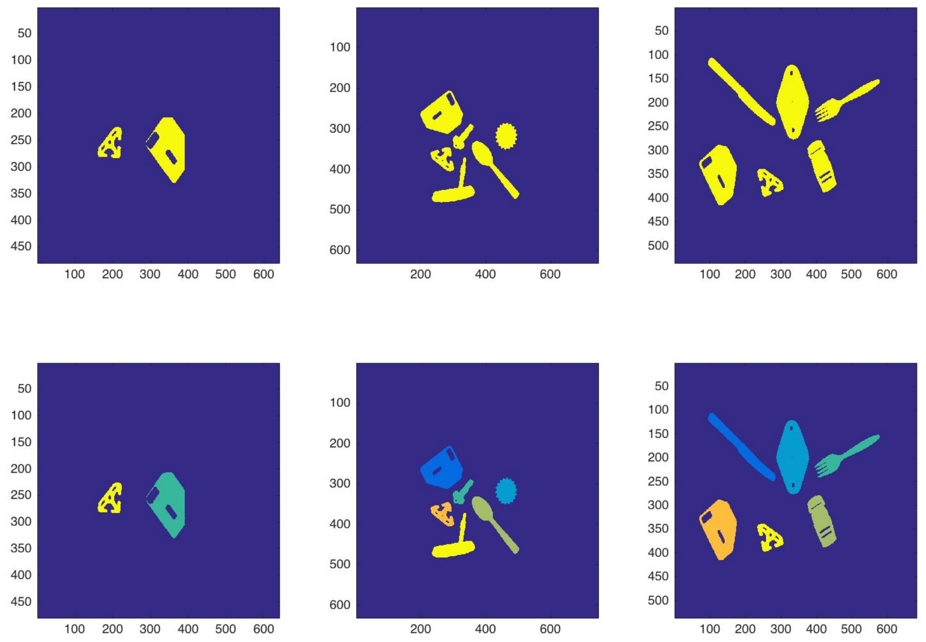

Figure 3. Programming Assignment 1(b)

(c) Write a MATLAB function named p3 that takes a labeled image and computes object attributes, and generates the objects database.
The formulas to compute the image's attributes are presented following.
Area:

$$a = \sum_{i=1}^{n}\sum_{j=1}^{m} b_{ij}$$

Position (center of area):

$$\overline{x} = \frac{1}{A}\sum_{i=1}^{n}\sum_{j=1}^{m} ib_{ij},\ \overline{y} = \frac{1}{A}\sum_{i=1}^{n}\sum_{j=1}^{m} jb_{ij}$$

Second Moments:

$$a' = \sum_{i=1}^{n}\sum_{j=1}^{m} i^2 b_{ij},\ b' = \sum_{i=1}^{n}\sum_{j=1}^{m} ijb_{ij},\ c' = \sum_{i=1}^{n}\sum_{j=1}^{m} j^2 b_{ij},$$

Orinetation and Roundness

$$E = a' sin(2\theta) - b' sin(\theta) cos(\theta) + c' cos(2\theta)$$

$$(orientation)\ \theta_{min} = \frac{atan2(b', a' - c')}{2}$$

$$\theta_{max} = \theta_{min} + \pi/2$$

$$(roundness)\ r = \frac{E_{min}}{E_{max}} = \frac{E(\theta_{min})}{E(\theta_{max})}$$

```
function [database_out, overlays_out] = p3(labels_in)
    overlays_out = zeros(size(labels_in));
    database_out = [];

    % find the num of objects in labels_in
    maxlabel = max(labels_in(:));
    labels = 1:maxlabel;

    % loop the labels and count the variable of each objects
    for label = 1:length(labels)
        % assign the label to the object
        database_out(label).object_label = label;
        index = (labels_in == label);
        % zeroth moment
        area = sum(index(:));
        database_out(label).area = area;

        % find the position of each pixel of this object
        xarray = [];
        yarray = [];
        [row, col] = size(labels_in);
        for i = 1:row
            for j = 1:col
                if labels_in(i,j) == label
```

```
                    xarray = [xarray;i];
                    yarray = [yarray;j];
                end
            end
        end

        % first moment
        xposition = sum(xarray)/length(xarray);
        yposition = sum(yarray)/length(xarray);
        database_out(label).x_position = xposition;
        database_out(label).y_position = yposition;
        % second moment
        xstararray = xarray - xposition;
        ystararray = yarray - yposition;
        a = sum(xstararray.*xstararray);
        b = sum(xstararray.*ystararray);
        c = sum(ystararray.*ystararray);
        theta = atan2(b,a-c)/2;
        database_out(label).orientation = theta;
        % roundness
        Emin = a*sin(theta)^2-b*sin(theta)*cos(theta)+c*cos(theta)^2;
        theta_ = theta + pi/2;
        Emax = a*sin(theta_)^2-b*sin(theta_)*cos(theta_)+c*cos(theta_)^2;
        roundness = Emin/Emax;
        database_out(label).min_moment = Emin;
        database_out(label).roundness = roundness;

        % plot the center and orientation of labeled zone
        for i=-3:3
            for j=-3:3
                overlays_out(round(xposition)+i, ...
                    round(yposition)+j) = label;
            end
        end
        for i=1:40
            overlays_out(round(xposition+i*cos(theta)), ...
                round(yposition+i*sin(theta))) = label;
        end
    end
end
```

(d)Write a function named p4 that recognizes objects from the database.
In order to recognize the similar objects in given image comparing to the database, here we use the roundness and area as the criteria.
We consider two objects are similar by,

$$|roundness_1 - roundness_2| < 0.1$$

$$|\frac{area_1 - area_2}{area_1}| < 0.15$$

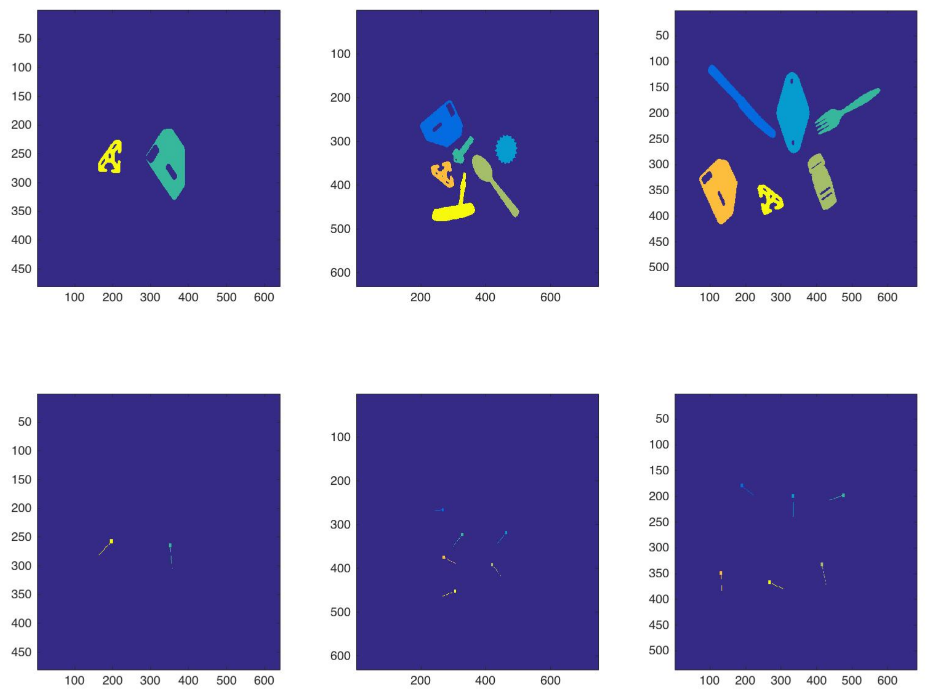

FIGURE 4. Programming Assignment 1(c)

```
function overlays_out = p4(labels_in, database_in)
    overlays_out = zeros(size(labels_in));
    % get the database of labels_in for comparing
    [database_compare, ~] = p3(labels_in);
    % record the label of compared database\'s similar objects
    labels = [0];

    for a = database_in
        for b = database_compare
            % use roundness and area as criteria
            if abs(a.roundness-b.roundness) < 0.1 && ...
                    abs((a.area - b.area)/a.area) < 0.15
                % if the object is not recorded
                if ~ismember(labels, b.object_label)
                    labels = [labels, b.object_label];
                    % draw the position and orientation
                    for i=-3:3
                        for j=-3:3
                            overlays_out(round(b.x_position)+i, ...
                                round(b.y_position)+j) = b.object_label;
                        end
                    end
                    for i=1:40
```

```
                              overlays_out(round(b.x_position+i*cos(b.orientation)),...
                                  round(b.y_position+i*sin(b.orientation))) = b.object_label;
                          end
                      end
                  end
              end
          end
      end
```

With such criteria, the results displayed below illustrates the object recognition is successful.

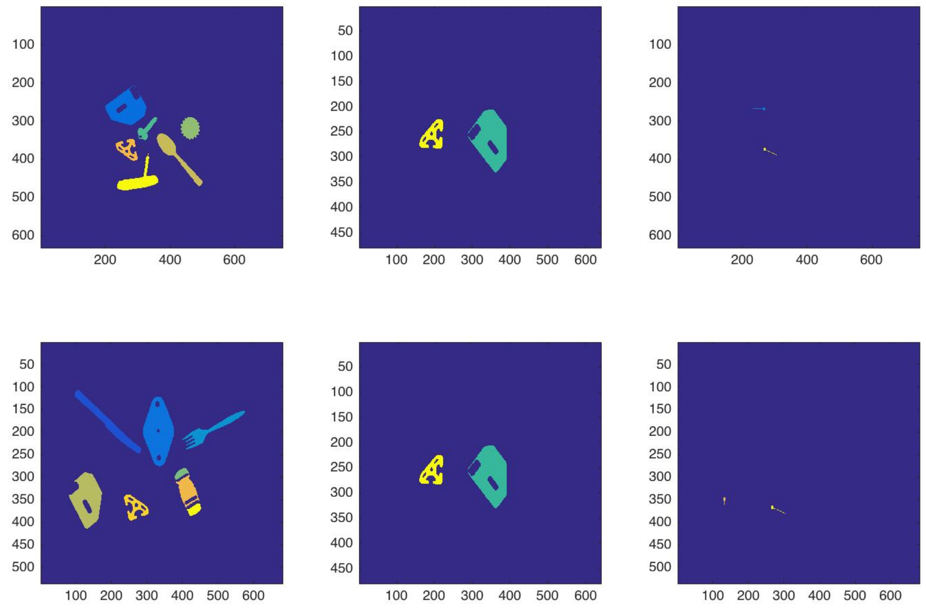

FIGURE 5. Programming Assignment 1(d)

Programming Assignment 2

(a) Find the locations of edge points in the image. Here we use the convolution masks proposed by Sobel.

$$\frac{\partial I}{\partial x}\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix};\ \frac{\partial I}{\partial y}\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix}$$

```
function edge_image_out = p5(image_in)
    % using sobel as gradient operator
    sobelmatrix_x = [-1 0 1; -2 0 2; -1 0 1];
    sobelmatrix_y = [1 2 1; 0 0 0; -1 -2 -1];
    % convolute the input image matrix in x,y direction respectively
    edge_image_out_x = filter2(sobelmatrix_x, image_in);
    edge_image_out_y = filter2(sobelmatrix_y, image_in);
    % compute the overall magnitude of each pixel
    edge_image_out = sqrt(edge_image_out_x.^2 + edge_image_out_y.^2);
end
```

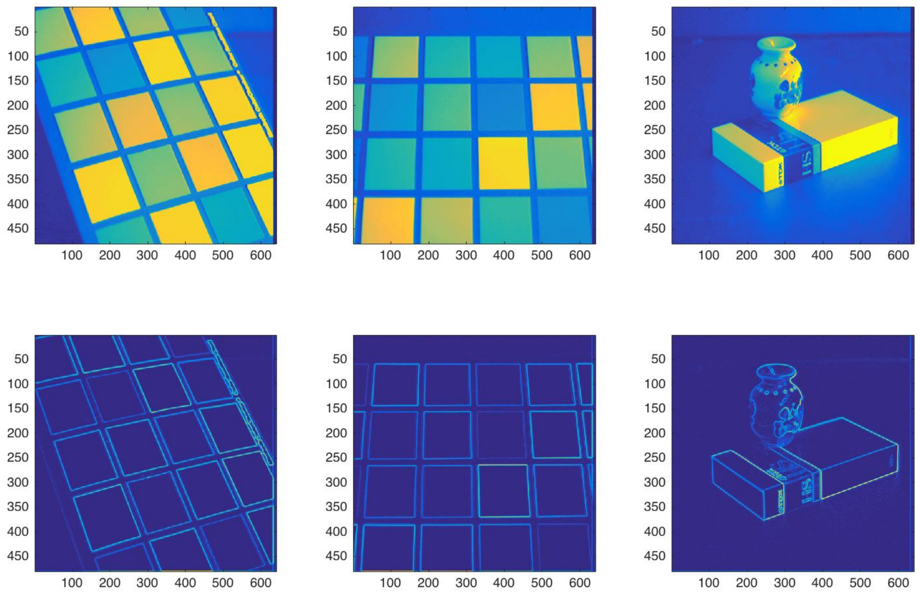

FIGURE 6. Programming Assignment 2(a)

(b) Threshold the edge image so that the image is left with only strong edges and implement the Hough Transform for line detection.
Instead of Cartesian coordinate, we use polar coordinate where two parameters can be bounded in limited range.
The line equation can be presented as,

$$xcos(\theta) - ysin(\theta) + \rho = 0;$$

where,

$$\theta \in [-\pi/2, \pi/2],\ \ \rho \in [-\rho_{max}, \rho_{max}]$$

and,

$$\rho_{max} = \sqrt{width^2 + length^2}$$

After thresholding the image, we discrete the θ and ρ into a matrix of parameters for voting. Knowing that every point in image space corresponds to a sine curve in parameter space, we firstly loop the image, for each edge point, we loop the θ, compute the corresponding ρ and do the voting. After the voting, we scale the hough matrix with value between 0 to 255.

```
function [edge_image_thresh_out, hough_image_out] = ...
    p6(edge_image_in, edge_thresh)

    % get the index of pixels with value over the threshold
    % assign the index with value 1 and others with 0
    index = (edge_image_in >= edge_thresh);
    edge_image_thresh_out = zeros(size(edge_image_in));
    edge_image_thresh_out(index) = 1;

    % the rho_max is the length of diagonal
    rho_max = sqrt(length(edge_image_thresh_out(1,:))^2+...
        length(edge_image_thresh_out(:,1))^2);
    % discrete the parameter
    theta_number = 360;
    rho_number = 1000;
    hough_image_out = zeros(theta_number, rho_number);

    % loop the pixels
    for x=1:length(edge_image_thresh_out(:,1))
        for y=1:length(edge_image_thresh_out(1,:))
            % if the pixel is in the edge
            if edge_image_thresh_out(x,y) == 1
                % loop the theta from -pi/2 to pi/2
                for i=1:theta_number
                    theta = pi/theta_number*i-pi/2;
                    % compute the corresponding rho
                    rho = y*cos(theta)-x*sin(theta);
                    j = ceil((rho+rho_max)/(2*rho_max)*rho_number);
                    hough_image_out(i,j) = hough_image_out(i,j)+1;
                end
            end
        end
    end

    % scale the hough matrix with values between 0 and 255
    hough_image_out = hough_image_out.*(255/max(hough_image_out(:)));
end
```

Threshold the edge images with different value shown below can keep the contour and reduce the noise.

```
[edge_image_thresh_out_1, hough_image_out_1] = p6(edge_image_out_1, 100);
[edge_image_thresh_out_2, hough_image_out_2] = p6(edge_image_out_2, 50);
[edge_image_thresh_out_3, hough_image_out_3] = p6(edge_image_out_3, 100);
```

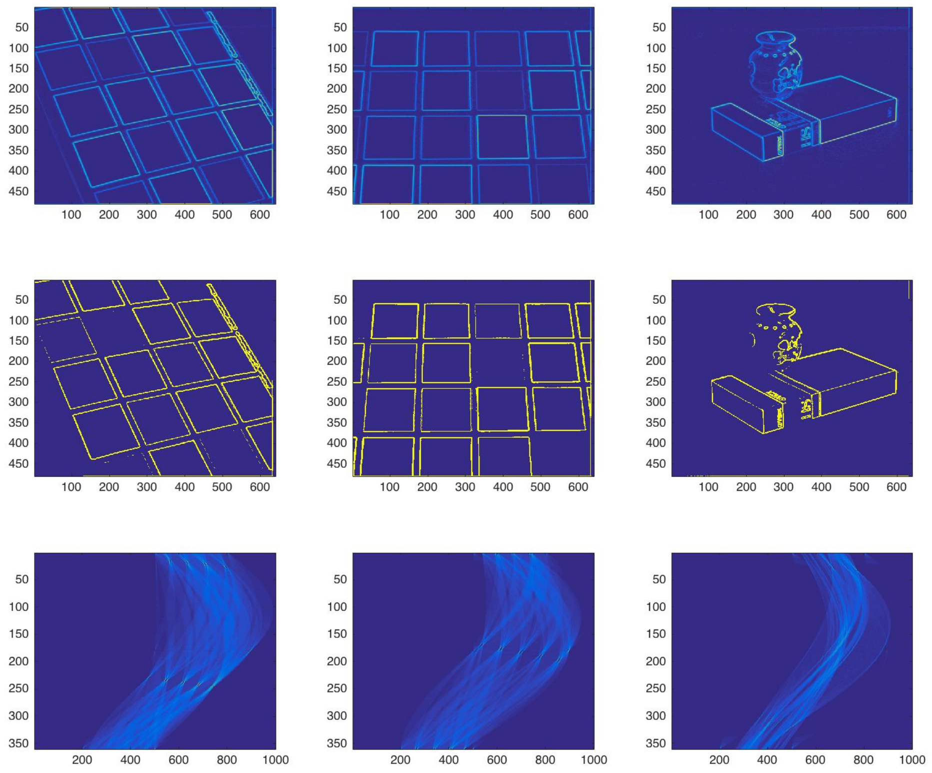

Figure 7. Programming Assignment 2(b)

(c) Loop the voting matrix from part (b), draw the lines based on parameters with votes over threshold.

```
function line_image_out = p7(image_in, hough_image_in, hough_thresh)
    line_image_out = image_in;

    % threshold the hough matrix by the hough threshhold input
    index = (hough_image_in >= hough_thresh);
    hough_image_in = zeros(size(hough_image_in));
    hough_image_in(index) = 1;
    [row, column] = size(hough_image_in);

    % define the rho max as the length of diagonal
    % define the line length for drawing
    rho_max = sqrt(length(image_in(1,:))^2 + length(image_in(:,1))^2);
    linelength = 10000;
    % display the image for line drawing
    imagesc(line_image_out);
    % loop the hough matrix
    for i=1:row
        for j=1:column
            % get the parameters with votes over threshold
            if hough_image_in(i,j)==1
                % compute the corresponding parameters
                theta = pi*i/row-pi/2;
                rho = 2*rho_max*j/column - rho_max;
                % draw the liens
                x0 = -rho/sin(theta);
                x1 = x0 + linelength*cos(theta);
                y1 = linelength*sin(theta);
                x2 = x0 - linelength*cos(theta);
                y2 = -linelength*sin(theta);
                hold on;
                line([y1,y2], [x1,x2], 'Color', 'r');
            end
        end
    end
end
```

Threshold the edge images with different value shown below can keep most of the edges.

```
line_image_out_1 = p7(image_in_1, hough_image_out_1, 125);
line_image_out_2 = p7(image_in_2, hough_image_out_2, 90);
line_image_out_3 = p7(image_in_3, hough_image_out_3, 100);
```

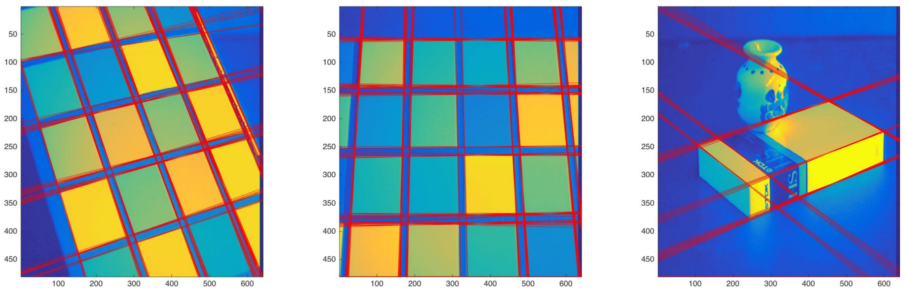

FIGURE 8. Programming Assignment 2(c)

(d) Implement an algorithm that prunes the detected lines so that they correspond to the line segments from the original image.
In order to prune the detected lines, here presents an algorithm which record and update the endpoints of each pair of parameter $[\theta, \rho]$ while voting.
The endpoints of a line have such characteristics: the coordinate value x and y are the maximum or minimum along the line. We use four matrix with same size as the hough (parameters) matrix to record the endpoints' position $[x_{max}, x_{min}, y_{max}, y_{min}]$.

```
function cropped_lines_image_out = p8(image_in, hough_image_in,...
    edge_thresholded_in, hough_thresh)

    cropped_lines_image_out = [];
    rho_max = sqrt(length(edge_thresholded_in(1,:))^2 + ...
        length(edge_thresholded_in(:,1))^2);
    theta_number = 360;
    rho_number = 1000;

    hough_image_out = zeros(theta_number, rho_number);

    % four matrix of same size as hough matrix to record endpoints postion
    endpoint_max_x = -inf*ones(theta_number, rho_number);
    endpoint_max_y = zeros(theta_number, rho_number);
    endpoint_min_x = inf*ones(theta_number, rho_number);
    endpoint_min_y = zeros(theta_number, rho_number);

    % loop the pixels
    for x=1:length(edge_thresholded_in(:,1))
        for y=1:length(edge_thresholded_in(1,:))
            % if the pixel is in the edge
            if edge_thresholded_in(x,y) == 1
                % loop the theta from -pi/2 to pi/2
                for i=1:theta_number
                    theta = pi/theta_number*i-pi/2;
                    % compute the corresponding rho
                    rho = y*cos(theta)-x*sin(theta);
                    j = ceil((rho+rho_max)/(2*rho_max)*rho_number);
                    hough_image_out(i,j) = hough_image_out(i,j)+1;
                    % record the maximum or minimum x value
                    % and corresponding endpoints
                    if x > endpoint_max_x(i,j)
                        endpoint_max_x(i,j) = x;
                        endpoint_max_y(i,j) = y;
                    elseif x < endpoint_min_x(i,j)
                        endpoint_min_x(i,j) = x;
                        endpoint_min_y(i,j) = y;
                    end
                end
            end
        end
    end
    % scale the hough matrix with values between 0 and 255
    hough_image_out = hough_image_out.*(255/max(hough_image_out(:)));

    % threshold the hough
    index = (hough_image_out >= hough_thresh);
    hough_image_out = zeros(size(hough_image_out));
    hough_image_out(index) = 1;
    [row, column] = size(hough_image_out);
    % display the original image for line drawing
    imagesc(image_in);
    for i=1:row
        for j=1:column
            if hough_image_out(i,j)==1
                hold on;
                x1 = endpoint_max_x(i,j);
                x2 = endpoint_min_x(i,j);
                y1 = endpoint_max_y(i,j);
                y2 = endpoint_min_y(i,j);
                line([y1,y2], [x1,x2], 'Color', 'r');
            end
        end
    end
end
```

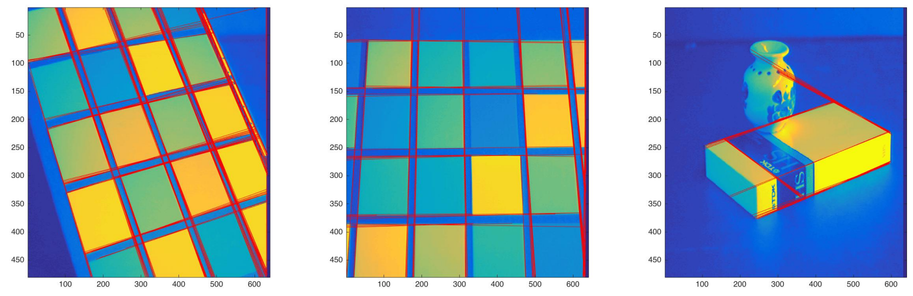

FIGURE 9. Programming Assignment 2(d)