Advanced Vision Assignment 1

Joe Yearsley
Edward Stevinson
University of Edinburgh
s1567581@sms.ed.ac.uk
s567688@sms.ed.ac.uk

March 16, 2016

1 Introduction

The assignment involved detecting and tracking people in an overhead video when they move and interact in the frame. For code mentioned in the report please see the appendix.

2 Person Detector

The first stage is to detect the people in each frame so this information can be used in the tracking. The function `normalise.m` takes the uploaded image, casts it as a double, and normalises its RGB values. This is done for the background image and for the image at each time step. It returns a normalised frame as well as a 2D matrix containing the sum of all pixel colour intensities. `extract_objects.m` performs the object detection. It first calculates the lightness values of each pixel, `s = (R + G + B) / 3`, and the chromaticity coordinates, $(r, g, b) = (\frac{R}{R+G+B}, \frac{G}{R+G+B}, \frac{B}{R+G+B})$. `lightness_test.m` and `chromaticity_test.m` then use the constants $\alpha = 0.75$ and $\beta = 1.35$ to determine whether each pixel belongs to a background or foreground object. The chromaticity returns only the red and green channels, since the blue is redundant information. We decided to use this method to remove the background instead of the background subtraction method, due to it performing better at removing shadows and light inconsistencies.

We limit the tests to the corner of the screen we are interested in, to ignore objects such as the laptop and stationary figure. This was achieved by limiting the loops in the test methods to only cover the corner of interest. This enables us quickly remove the objects, by setting the values we aren't interested in to 0. We join the tests by seeing that there is a one in either of these tests, representing that one of the tests believes that the pixel is a foreground image. We didn't believe it needed the extension to chromaticity modelling due to it having the same background, and similar lighting conditions throughout, however we could have used it to remove the laptop and caller if we didn't know much about the video.

`clean_image.m` performs operations to clean the detections. It first erodes the image passed to it a little to remove spurs, using the `imerode()` command and a structuring element of a diamond of size 3. Next we dilated the image using a structuring element of a square of size 12, as can be seen below:

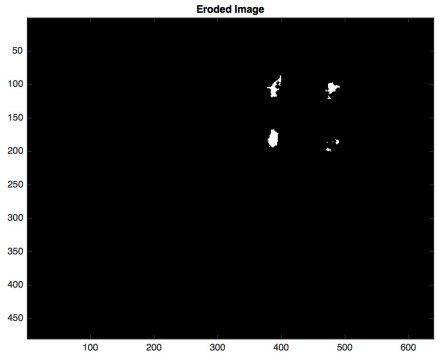

Figure 1: Erosion to remove spurs

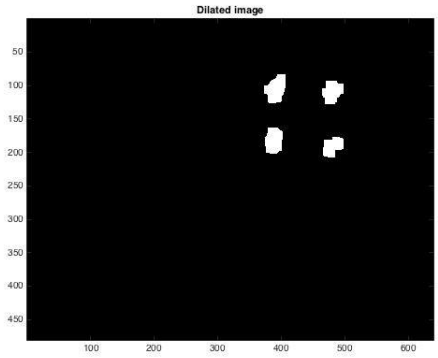

Figure 2: Dilation to fill gaps

Finally, we erode again to restore the objects to their original size. A structuring element of square of size 2 was used to ensure the objects stay connected.

As can be seen there is now some noise, however when we return the connected components by using the `bwconncomp` function (with a connectedness of 8 to ensure strong connectivity). To get key stats such as Area and Centroid positioning we use the region props command, before finally ensuring we remove the noise by using the labelmatrix function and removing regions with small areas. We use an area of 350, obtained by storing the Median (Robust Mean) for each frame, then taking the median again, resulting in a number around 400. To split

The first method we used to separate collisions where two people were being interpreted as one was a sobel filter. This split the larger detections into two (and sometimes more)

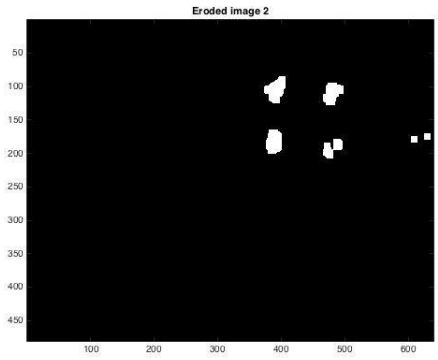

Figure 3: Erosion to restore correct size

detections but led to difficulties with detection size, so instead we treated every collision detection as an ellipse, calculated the major axis and used it to define two 'centroids' roughly in the position of each individual in the collision. An inprovement on this method would be to cut out along the minor axis to give two detections, which would return more accurate centroids.

We then return this cleaned image to the next function, `size_split`, which returns the statistics and orderings of the images. We first tried labelling the objects by comparing the normalised RGBs of the current frame regions to the first frame regions RGBs. This was done by using a bounding box based on the calculated radius of the detections (and multiplying by 0.9 to give us a slightly smaller area). This was an attempt to remove some of the excess background. We then saved all of these into a 4 by 4 array which we then get the unique minimum from by first getting the minimum of the rows, saving them into a 2d array (for each row), with the value and index (implemented by the `minmin.m` function). We then get the minimum of this array and save the column index into the row index of the returned array. Then to not allow everything to pick this value, we change it to a value larger than the max currently in the matrix. Before going onto the next row.

The detection was fairly successful. To calculate the accuracy of matches we counted all the centroids which were in a euclidean distance of 10 to the ground truth centroids. We then divided by 840 and multiplied by 100, resulting in a 73.6% match. To get the mean, we calculated the mean distance for each ground truth to each match near by, then we got the double median of the matrix, resulting in a median distance of 4.1424 pixels. There

were no false person detections as we we succesfully cleaned the detections and removed any detections with size below a threshold. However, problems arose with the labelling. The Battacharyya distance comparison between normalised histograms frequently did not succeed in returning the correct labels. Even when we created a separation value based on the euclidean distance between object detections and the difference between histograms, it frequently lead to mislabels. This lead to problems with the condensation tracker. Despite our attempts to create masks and vary the bounding box, we could not create observed data with a correct labelling percentage of greater than 60%.

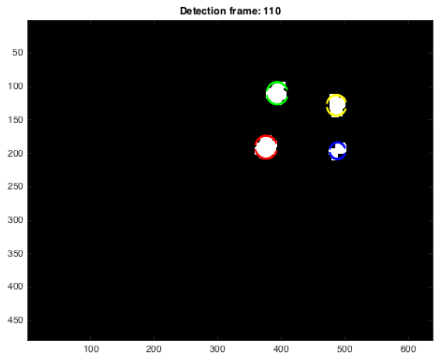

Figure 4: Object detection in frame 110

3 Tracking

A condensation tracker was used to track each person in the video. The code for this was in a function called `track.m` (see Appendix) which was called for each object to be tracked. A condensation filter is a particle filter that maintins an array of hypotheses of the states of the person, and uses the observations to select the best hypothesis in each time step. In each frame the hypotheses are sampled according to the weight ('probability') of the sample state vector. It uses a dynamic model with situation switching between states, which were chosen to be *still*, *moving*, and *colliding*, as shown in Figure 5. The condensation tracker hypothesises the state vector changes according to deterministic dynamics by randomly generating state transitions for each sample according to the estimated state transition probabilities. These were estimated to be $p(\text{collide}) = 0.3$, $p(\text{still}) = 0.1$, and $p(\text{move}) = 0.6$. Since an individual is not free to go from still to collide, conditional probabilities were calculated for $p(\text{move}|\text{collided})$ and $p(\text{move}|\text{was still})$. Each sample is a version of the state vector, and takes the form: $\mathbf{x} = (\text{x position}, \text{y position}, \text{x velocity}, \text{y velocity})^T$.

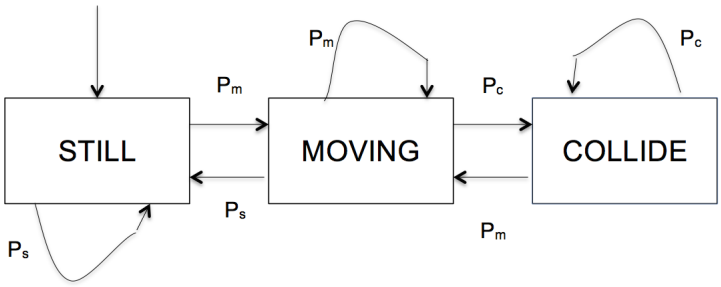

Figure 5: State Diagram

A simple motion model assuming constant velocity, $x_{t+1} = A * x_t$, is used to update the state vector using different matrices A modelling the different states. The *still* state has A_s and merely sets the velocities to zero. Changing to the *moving* state sets A_m and initiates x and y velocities randomly generated from a gaussian distributed random number (mean = 0, sigma = 1) to account for the differing speeds of the dancers. Finally, the *colliding* state has A_c and does a similar thing than but with a smaller velocity values.

$$A_s = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, A_m = A_c = \begin{pmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

The 'goodness' value of each sample state vector is then calculated by subtracting the hypotheses from the observed data and forming a value according to Equation X:

$$\frac{1}{|\mathbf{x} - \mathbf{z}|^2} \tag{1}$$

The best hypothesis (in relation to the observed data) is plotted as the output of the filter.

The condensation tracker would successfully pair people between consecutive frames until the distance between detections became small and the RGB histogram comparison failed to label the observed data correctly. At this point a tracker for one person is given observed data for another person, making the tracker lose its dancer. After a period of fluctuation (approx. 8 frames) the tracker tracks another of the people. Due to how our tracker was implemented there were no trajectories with only one detection.

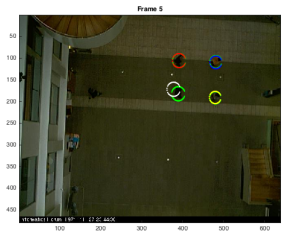

Figure 6: Frame 5

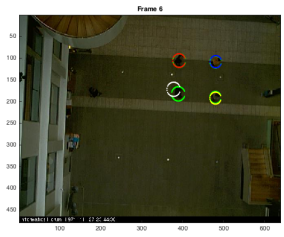

Figure 7: Frame 6

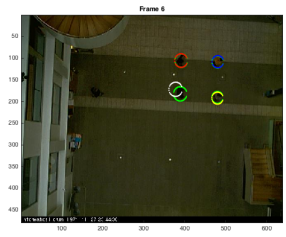

Figure 8: Frame 7

Figure 9 shows an image of all the ground-truth trajectories on top of the initial frame, each marked with a different colour. Figure 10 (see below) shows the initial trajectories of the tracker before failure. To ensure that it was the labels and not the tracker that was causing the failure, the tracker was run being fed the ground-truths as observation data and it successfully tracked each player with only one trajectory.

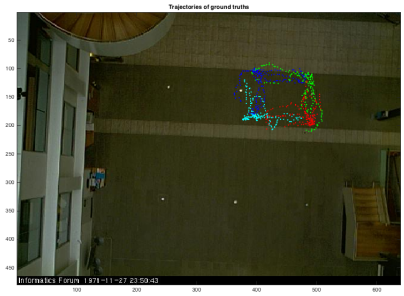

Figure 9: Trajectories of the ground truths

4 Appendix

4.1 Condensation Tracker

`track.m`

```
function [weights, trackstate, x, xc, oldsample] = track(person_identifier, ...
            number_of_frames, image_bg, colour, num_cond_samples, ...
            trackstate, x, xc, cc, cr, step, weights, oldsample)
```

Figure 10: Trajectories of the condensation filter

```
%% Transition probabilities
p_move = 0.3;
p_stop = 0.6;
p_collide = 0.1;
p_move_gs = 3/8; p_stop_gs = 5/8;
p_collide_gc = 0.25; p_move_gc = 0.75;
%% Motion model matrices
dt=1/9;
A1=[[1,0,0,0]',[0,1,0,0]',[0,0,0,0]',[0,0,0,0]'];  % stopped
A2=[[1,0,0,0]',[0,1,0,0]',[dt,0,1,0]',[0,dt,0,1]']; % moving
A3=[[1,0,0,0]',[0,1,0,0]',[dt,0,1,0]',[0,dt,0,1]']; % collides
[MR,MC,Dim] = size(image_bg);
%% Condensation Tracking starts here
if step ~= 1
    SAMPLE = 1000;
    ident = zeros(SAMPLE,1);
    idcount = 0;
    for newsample = 1 : num_cond_samples
        % number of samples to generate
        num = floor(SAMPLE * weights(newsample,step-1, person_identifier));
        if num > 0
            ident(idcount+1:idcount+num) = newsample * ones(1,num);
            idcount = idcount + num;
        end
    end
end
%% generate new state vectors
for newsample = 1 : num_cond_samples
```

```
% sample randomly from the ident()
if step == 1
    % beginning state vector is random
    xc(:, person_identifier) = [floor(MC*rand(1)),floor(MR*rand(1)),0,0]';
else
    % select which old sample
    while oldsample(person_identifier) == 0
        oldsample(person_identifier) = ident(ceil(1000*rand(1)));
    end
    % get its state vector
    xc(:, person_identifier) = x(oldsample(person_identifier),step-1,:,
     person_identifier);
end
%% hypothesize what state it will be in (deterministic dynamics)
% assume the person starts still (i.e. time = 1)
if step == 1
     xp = xc(:, person_identifier);   % no process at start
     A = A1;
     trackstate(newsample,step, person_identifier) = 1;
% now for when time is not 1...
else
    % create random probability for state selection
    r=rand(1);
    if trackstate(oldsample(person_identifier), step-1,
    person_identifier) == 1
        %% if the person is stopped then can either stop or move
        % stopped having been stopped
        if r < p_stop_gs
            xc(3, person_identifier) = 0; % i.e. no velocity in x or y
            xc(4, person_identifier) = 0;
            A = A1;
            trackstate(newsample,step, person_identifier)=1;
        % going from stopped to moving
        else
            xc(3, person_identifier) = random() * (rand()*20);
            xc(4, person_identifier) = random() * (rand()*20);
            A = A2;
            trackstate(newsample,step, person_identifier) = 2;
        end
    %% person collided in last state
    elseif trackstate(oldsample(person_identifier), step-1,
     person_identifier) == 3
         % collides again
         if r < p_collide_gc
             A = A3;
             xc(3, person_identifier) = (normrnd(0,1)*200); %random *
             xc(4, person_identifier) = (normrnd(0,1)*200); %random *
```

```
                % goes to moving
                else
                    A = A2;
                    xc(3, person_identifier) = random() * (rand()*20);
                    xc(4, person_identifier) = random() * (rand()*20);
                end
            %% person was moving in last frame...
            else
                % now stops...
                if r < p_stop
                    xc(3, person_identifier) = 0;
                    xc(4, person_identifier) = 0;
                    A = A3;
                % now moves...
                elseif r < (p_stop + p_move)
                    A = A2;
                    % I think if it just continues moving no values need setting?
                % now collides...
                else
                    A = A2; %Bu = Bu3;
                    xc(3, person_identifier) = (normrnd(0,1)*200); %random *
                    xc(4, person_identifier) = (normrnd(0,1)*200); %random *
                end
            end
        end
        %% Motion model: Update the state vector to get the next state vector
        xp = A * xc(:, person_identifier);


        %% Now have predicted states we compare to observed states
        % update & evaluate new hypotheses
        x(newsample,step,:, person_identifier) = xp;
        % evaluate probability of observed image given data
        dvec = [cc(step, person_identifier),cr(step, person_identifier)] -
        [x(newsample,step,1,         person_identifier),x(newsample,step,
        2, person_identifier)];
        weights(newsample,step, person_identifier) = 1 / (dvec * dvec');
    end

% rescale new hypothesis weights
totalw = sum(weights(:,step, person_identifier)');
weights(:,step, person_identifier)=weights(:,step, person_identifier)/totalw;
% select top hypothesis to draw
subset = weights(:,step, person_identifier);
top = find(subset == max(subset));
trackstate(top,step, person_identifier);
% display final top hypothesis
```

```
  figure(1)
  hold on
  for c = -0.99*radius: radius/10 : 0.99*radius
        r = sqrt(radius^2-c^2);
        plot(x(top,step,1, person_identifier)+c,x(top,step,2, person_identifier)+r,
        [colour(person_identifier), '.'])
        plot(x(top,step,1, person_identifier)+c,x(top,step,2, person_identifier)-r,
        [colour(person_identifier), '.'])
  end

end
```

4.2 Main Program

final_main.m

```
%% Main program
% Clear all figures
delete(findall(0,'Type','figure'))
 med_prev_image = zeros(1,480,640,3);
%% Detection constants
alpha = 0.75;
beta = 1.35;
colour = ['r','w','b','y'];
NUM_PEOPLE = 4;
% Define eval matrix 210 frames, 4 people, 2 co-ords
eval_matrix = zeros(210,4,2);

%% Condensation tracker initialisation
num_cond_samples = 10000;          % number of condensation samples
num_people = 4;
num_frames = 210;
number_of_frames = 210;
num_states = 4; % number of states: x_pos, y_pos, x_vel, y_vel
x = zeros(num_cond_samples,num_frames,num_states, num_people);   % state vectors
weights = zeros(num_cond_samples,num_frames, num_people);  % probability of state
trackstate = zeros(num_cond_samples,num_frames, num_people);   % 1,2,3,4 =
 % x_pos, y_pos, x_vel, y_vel
oldsample = zeros(num_people,1);
xc = zeros(4,num_people); % initialise selected state

%% Load and normalise background frame
% background frame
image_bg = imread('DATA1/bgframe.jpg','jpg');
```

```
% normalise background image
[sum_bg, sum_bg_norm] = normalise(image_bg);
% lightness values (lightness background = s_b)
s_b = sum_bg./3;
% Make size of for loop, for speed, for robust mean component size
medi = zeros([30,1]);
% Make size of for loop, for speed, to check number of labels
labeleds = zeros([30,1]);
lab = [];
%Boolean check for if the first loop
bool = 0;


%% Loop through each frame
for step = 1 : 210
    % Labels matrix
    lab = ones(NUM_PEOPLE);
    %Distance Matrix
    dist_matrix = ones(NUM_PEOPLE);
    lab1d = [];

    %% Load current image
    image = imread(['DATA1/frame',int2str(step+109), '.jpg'],'jpg');
    % background subtraction, image cleaning, and statistics of objects
    [cleaned_image, ordered, stats, labeled, med] = extract_objects(image, s_b,
     sum_bg, sum_bg_norm, alpha, beta);

    % Median, as we keep getting noise, so do one run to see robust median
    % of regions (we've seen more people than noise on average).
    % Adjust array to ensure we have correct median
    medi(step) = med;
    %Label array so we can ensure no noisy labels have been made
    labeleds(step) = max(max(labeled));

    % Median holder
    prev_image = zeros(210,480,640,3);

    %Check if first frame!
    if(bool == 0)
        bool = 1;
        % Save frame labels
        first_labels = labeled;
        % Save Regions of First Image
        first_stats = stats;
        prev_stats = stats;
        % First image
        first_image = image;
```

```
    prev_image(step,:,:,:) = image;
    updated_labels = [1; 2; 3; 4];
else
     cur_image = image;
     % These don't matter which order as should be same size....
     for reg = 1:size(stats)
         templab = [];
         for reg2 = 1:size(prev_stats)
             %% For Euclidean Distance, take stats and prev stats
             %% For RGB Distance
             x_s = prev_stats(reg2);
             x_radius = 0.9*sqrt(x_s.Area/pi);
             x_start = x_s.Centroid(1) - x_radius;
             y_start = x_s.Centroid(2) - x_radius;
             x_length = 2*x_radius;
             y_length = 2*x_radius;
             % Median of previous images
             med_prev_images(1,:,:,:) = median(prev_image(1:step-1,:,:,:),1);
             % Make our own bounding box
             cropped1 = imcrop(squeeze(med_prev_image),[x_start, y_start,
              x_length, y_length]);
             x_s = stats(reg);
             x_radius = 0.9*sqrt(x_s.Area/pi);
             x_start = x_s.Centroid(1) - x_radius;
             y_start = x_s.Centroid(2) - x_radius;
             x_length = 2*x_radius;
             y_length = 2*x_radius;
             % Make our own bounding box
             cropped2 = imcrop(cur_image,[x_start, y_start, x_length,
              y_length]);
             lab(reg,reg2) = (eD(stats(reg).Centroid,prev_stats(reg2).Centroid)) + ...
             2*hist(cropped1,cropped2);
         end
     end
     updated_labels = minmin(lab);
end
% Keep for next frame analysis
prev_stats = stats ; % needs reordering
prev_image(step,:,:,:) = image;

%% Display the cleaned image
%imagesc(cleaned_image); title(['cleaned image' num2str(i)]); colormap(gray);
imagesc(image); title(['cleaned image' num2str(i)]);

% Get CoM, radius and ... and then draw circles
NUM = size(ordered);
lab;
```

```
    for j = 1:4
        % get center of mass and radius of 2 largest
         cc(step, j) = stats(updated_labels(j)).Centroid(1);
         cr(step, j) = stats(updated_labels(j)).Centroid(2);
         radius(step,j) = sqrt(stats(updated_labels(j)).Area/pi);

        % draw circles around objects
         hold on;
         for c = -0.97*radius(step,j): radius(step,j)/20 : 0.97*radius(step,j)
             r = sqrt(radius(step, j)^2-c^2);
             plot(cc(step, j) +c,cr(step, j) +r,'g.')
             plot(cc(step,j)+c,cr(step,j)-r,'g.')
         end
         str = sprintf('Frame %d', step);
         title(str);
     end

    %% Tracking
    [weights, trackstate, x, xc, oldsample] = track(1, number_of_frames, image_bg,
     colour, num_cond_samples, trackstate, x, xc, cc, cr, step, weights, oldsample,
      radius);
    [weights, trackstate, x, xc, oldsample] = track(2, number_of_frames, image_bg,
     colour, num_cond_samples, trackstate, x, xc, cc, cr, step, weights, oldsample,
      radius);
    [weights, trackstate, x, xc, oldsample] = track(3, number_of_frames, image_bg,
     colour, num_cond_samples, trackstate, x, xc, cc, cr, step, weights, oldsample,
      radius);
    [weights, trackstate, x, xc, oldsample] = track(4, number_of_frames, image_bg,
     colour, num_cond_samples, trackstate, x, xc, cc, cr, step, weights, oldsample,
      radius);
    % Pause between frames
    pause(0.01)
end
```

4.3 Clean the detections

clean_image.m

```
%%
% Function which cleans an image and returns a labelled version
% and a labelled version
function [cleaned_image, labelled_image2, med] = clean_image(image, norm)

    %% Clean up the image
```

```
    % create structuring elements
    structuring_elem = strel('square',12);
    structuring_elem2 = strel('square',3);
    structuring_elem3 = strel('diamond',2);
    structuring_elem4 = strel('diamond',5);

    BW2 = image;
    %% Erode to remove spurs
    eroded_image = imerode(BW2,structuring_elem3);
    %imagesc(eroded_image); title('eroded1 image'); colormap(gray); figure;

    %% Dilate to fill internal holes
    dilated_image = imdilate(eroded_image,structuring_elem);
    %% Erode to restore size
    eroded_image2 = imerode(dilated_image,structuring_elem2);
    %% Collect into regions
    connected_regions2 = bwconncomp(eroded_image2,8);

    % get area of connected regions
    regions2 = regionprops(connected_regions2, 'Area', 'PixelIdxList', ...
            'BoundingBox','MajorAxisLength','MinorAxisLength');

    %Used robust mean to give good estimate to cut out noise
    % Return so we can get overall median to use as a estimator.
    med = median([regions2.Area]);
    % label each region
    labelled_image2 = labelmatrix(connected_regions2);
    % remove regions with less than 'x' pixels
    cleaned_image = ismember(labelled_image2, find([regions2.Area] >= 350));
    %Take 2
    connected_regions2 = bwconncomp(cleaned_image,8);
    % get area of connected regions
    regions2 = regionprops(connected_regions2, 'Area', 'PixelIdxList', ...
            'BoundingBox','MajorAxisLength','MinorAxisLength');

end
```

4.4 Splitting collisions

size_split.m

```
%%
% Get statistics of detections and split collisions
%%
```

```
function  [id, stats] = size_split(image)
    % connected parts in the image
    connected = bwconncomp(image,8);
    % label them
    labeled = bwlabel(image,4);
    % get statistics of the connected regions
    stats = regionprops(connected, 'Area', 'Centroid', 'BoundingBox',...
        'MajorAxisLength','MinorAxisLength', 'Orientation');
    sz = size(stats);
    statsBefore = stats;
    for i = 1:sz
     if stats(i).Area > 1000 && stats(i).MajorAxisLength > 50
        %Get the centroids x
        centx = stats(i).Centroid(1);
        %Get the centroids y
        centy = stats(i).Centroid(2);
        theta = stats(i).Orientation;
        majorAxis = (stats(i).MajorAxisLength/4.);
        stats(i).Centroid(1) = centx + (majorAxis*cos(theta));
        stats(i).Centroid(2) = centy + (majorAxis*sin(theta));
        stats(i).Area = pi*(majorAxis^2);
        sEnd = numel(stats)+1;
        stats(sEnd).Centroid(1) = centx - (majorAxis*cos(theta));
        stats(sEnd).Centroid(2) = centy - (majorAxis*sin(theta));
        stats(sEnd).Area = pi*(majorAxis^2);
     end
    end
    % stats of the connected objects
    [N,W] = size(stats);
    % bubble sort
    id = zeros(N,1);
    for i = 1 : N
      id(i) = i;
    end
```

4.5 Euclidean Distance Function

eD.m

```
%% Euclidean Distance
function euclidDistance = eD(x,y)
    euclidDistance = sqrt(sum( (x-y).^2));
end
```

4.6

minmin.m

```
%% Function that returns the minimum values of a matrix
function arr = minmin(matrix)
    arr = zeros(size(matrix,1),1);
    m_v = max(max(matrix))*3;
    for i = 1:size(matrix,1)
        [min_valforeach_col, row_loc] = min(matrix);
        [~, min_col] = min(min_valforeach_col);
        matrix(:,min_col) = m_v; % Sets column with min val in to m_v
        matrix(row_loc(min_col),:) = m_v; % sets row with min val in to m_v
        arr(row_loc(min_col)) = min_col;
    end;
end
```