Contents

1 Initialising functions

```
1
2  h = 900;
3
4  min_y = 120;
5  max_y = 460;
6  min_x = 75;
7  max_x = 310;
8
9  % Threshold values
10 min_thresh = 30;
11 max_thresh = 500;
12
13 % Get image from depth sensor
14 colorVid = videoinput('kinect',1);
15 depthVid = videoinput('kinect',2);
16 depth = getsnapshot(depthVid);
17 color = getsnapshot(colorVid);
18 raw_matrix = depth;
19 %%
20 %Run the sobel operator
21
22 depth = sobel_operator(depth);
23 shapes_after_sobel = depth;
24 %Run the threshold filter
25 depth = threshold(depth, min_thresh, max_thresh);
26 depth = print(depth, min_x, max_x, min_y, max_y);
27 depth_after_threshold = depth;
28
29 %%%%%%outline
30 depth = outline(depth);
31 final_img = only_outline_visible(depth);
32 edged_matrix = only_edge(depth);
33
34 new_depth = crop_depth_to_basket(edged_matrix,
       depth_after_threshold);
35 depth_tester = new_depth;
36
37 %OVERLAP
38 %%%%%%%%%%%%%%%%%%%%%%%%%%
39
40 %color: 1920x1080 met 84.1 x 53.8
41 %depth: 512x424  met 70.6 x 60
42
```

```
43 [reformed_depth,reformed_color, res_height_angle,
       res_width_angle] = reform(depth, color);
44 [pipemm_depth_H, pipemm_depth_W, pipemm_color_H,
       pipemm_color_W] = get_pipemm(res_height_angle,
       res_width_angle, h, reformed_depth,reformed_color);
45
46
47 [prop,nb_rows_color , nb_columns_color,nb_rows_depth,
       nb_columns_depth] = proportion(reformed_depth ,
       reformed_color);
48
49 tot_size = size_matching(prop);
50
51 total = overlap_depth_to_RGB(reformed_depth,
       reformed_color, pipemm_depth_H , pipemm_depth_W ,
       pipemm_color_H , pipemm_color_W,tot_size,nb_rows_color
        , nb_columns_color);
52
53 new_RGB = crop_RGB_to_basket(total);
54 image(new_RGB);
55
56 img = new_RGB;
57
58 THRESHOLD_VALUE = 2;
59
60 MIN_ROW_LINES_BETWEEN_GROUPS = 10; %25 %15
61 % Once the groups are found, the algorithm searches for
       groups too close
62 % near each other
63 % This is defined as the min distance between two groups
       (only searched
64 % vertical)
65 SAME_PIXELS_SEARCH_GRID_SIZE = 10;%25
66 % Grid size = this variable *2, it searches for pixels
       with the same value
67 % in this grid.
68 GROUP_SEARCH_GRID_SIZE = 15; %25
69 % Grid size = this variable * 2, it searches for pixels
       with a group number
70 % (not 0) in this grid.
71 SURROUDING_PERCENTAGE = 10;% %
72 MIN_NB_SURROUNDING_PIXELS = floor((
       SAME_PIXELS_SEARCH_GRID_SIZE * 2)^2 *
       SURROUDING_PERCENTAGE/100) ;%125 % 50
73 % The minimum number of pixels with the same value that
       are in the grid
```

```
74 % size defined by SAME_PIXELS_SEARCH_GRID
75 % The pixels that have a less number of surrounding
       pixels, are not defined
76 % as a group but as noise.
77
78 % CROPPING: Defining rectangle
79 %top_row = 290 ; top_col = 760; bottom_row = 690 ;
       bottom_col = 1440;
80 %top_row = 150; top_col = 750; bottom_row = 950;
       bottom_col = 1900;
81 %top_row = 200; top_col = 850; bottom_row = 750;
       bottom_col = 1850; % For pictues with x2_RGB_... in
       name
82 %top_row = 100, top_col = 100; bottom_row = 980;
       bottom_col = 1820; % For pictues with RGB in name.
83
84 disp("Step 1: loading the image...");
85 disp("Minimum distance between 2 objects (only straight
       vertical or straight horizontal = " + max([
       MIN_ROW_LINES_BETWEEN_GROUPS
       SAME_PIXELS_SEARCH_GRID_SIZE MIN_NB_SURROUNDING_PIXELS
       ;]) + " pixels");
86
87 disp("Step 2: converting the image to greyscale...");
88
89 A = greyscale(img); % Convert image to grayscale
90
91 %top_left_row, top_left_col, bottom_right_row,
       bottom_right_col
92 disp("Step 3: cropping the image...");
93
94 %A = simon_crop(A, top_row, top_col, bottom_row,
       bottom_col);
95 imshow(A, []);
96 %%
97 %A = simon_crop(A, 100,100,980,1820, 1); % USE FOR foto
       RGB X
98 %A = simon_crop(A, top_row,top_col,bottom_row, bottom_col
       ,1); % USE FOR foto XX RGB
99
100 disp("Step 4: blurring the image...");
101 A = gaussian_blur(mean_blur(A)); % Filters
102 % Method 3: First greyscale, then blur, then edge detect
       then threshold and then noise removal
103 disp("Step 5: edge detecting...");
104 first_edge_detect = edge_detect(A); % Laplacian edge
```

```
    detection
disp("Step 6: thresholding edge");
without_noise_removal = threshold_edge(remove_boundary(
    first_edge_detect, 15), THRESHOLD_VALUE); % Remove
    boundary around image & threshold the edges.
disp("Step 7: noise removing...");
%with_noise_removal = noise_deletion(
    without_noise_removal,5); % Noise removal
with_noise_removal = without_noise_removal;
disp("Step 8: grouping...");
[grouped, nb_of_groups] = group(~with_noise_removal,
    SAME_PIXELS_SEARCH_GRID_SIZE, GROUP_SEARCH_GRID_SIZE,
    MIN_NB_SURROUNDING_PIXELS); % Group pixels together

disp("Step 9: regrouping...");
[regrouped, nb_of_groups2] = regroup(grouped,
    nb_of_groups, MIN_ROW_LINES_BETWEEN_GROUPS); % Regroup
     (nessicary because group function works from top left
     to bottom right

%Find corner points of object (not really corner points
    on the boundary,
%but corner points for the boundary box)
disp("Step 10: calculating corner points...");
corner_points = find_corner_points(regrouped,
    nb_of_groups); % Make sure to use nb_of_groups and not
     groups 2 because some groups don't exist anymore!

disp("Step 11: removing objects within objects...");
%[updated_corner_points, nb_of_groups3] =
    remove_corner_points_within_corner_points(
    corner_points, nb_of_groups2); % To remove objects
    within objects
[updated_corner_points, nb_of_groups3] = remove_box_edge(
    corner_points, nb_of_groups2);
[updated_corner_points, nb_of_groups3] =
    remove_corner_points_within_corner_points(
    updated_corner_points, nb_of_groups3);
%updated_corner_points = corner_points;
%nb_of_groups3 = nb_of_groups2;

disp("Step 12: drawing boundary boxes...");
boundary_box = draw_boundary_box(A, updated_corner_points
    );
disp("Step 13: drawing red boundary boxes on full image
    ...");
```

```
131 red_boundary_box = draw_red_boundary_box(reformed_color,
        updated_corner_points, 1,1);
132 disp("Step 13: Done!!!");
133 toc
134 %%
135 % Original image
136 imshow(img, []);
137 imwrite(img, 'img_brent_2.png');
138 title("Original image");
139 %% After edge detection
140 imshow(first_edge_detect, []);
141 title("Edge detection");
142 %% Grouped image
143 imagesc(grouped(:,:,2));
144 title("Groups, #nb_objects = " + nb_of_groups);
145 %% Regrouped image
146 imagesc(regrouped(:,:,2));
147 title("Regrouped, Number of objects = " + nb_of_groups2);
148 %%
149 %imshow(recolor(regrouped(:,:,2), nb_of_groups2), []);
150 %% Result
151 imshow(boundary_box, []);
152 title("Boundary box + removed objects within objects,
        Number of objects ="+ nb_of_groups3);
153 %%
154 imshow(red_boundary_box, []);
155 title("# objects: "+ nb_of_groups3);
```

2 Functions for depth

2.1 Sobel operator

```
1  function shapes = sobel_operator(img)
2      % use the sobel-operator on the raw depth image
3      % this function returns a matrix of the same size as
           the original
4      % matrix with on every position the gradint
5
6      X = img;
7      Gx = [1 +2 +1; 0 0 0; -1 -2 -1]; Gy = Gx';
8      temp_x = conv2(X, Gx, 'same');
9      temp_y = conv2(X, Gy, 'same');
10     shapes = sqrt(temp_x.^2 + temp_y.^2);
11 end
```

2.2 Threshold for depth

2.2.1 threshold in values

```
function thresholded = threshold(img, min_thresh,
    max_thresh)
    % run the image through a threshold to get rid of
        impossible values
    % this function returns a binary matrix with a 1 on
        the edges

    matrix_size = size(img);

    MAX_ROW = matrix_size(1);

    MAX_COLUMN = matrix_size(2);

    for row = 1 : MAX_ROW
        for col = 1: MAX_COLUMN
            if (img(row, col) > min_thresh) && (img(row,
                col)< max_thresh)
                img(row, col) = 1;
            else
                img(row, col) = 0;
            end
        end
    end
    thresholded = img;
end

function printed = print(img, min_x, max_x, min_y, max_y)
    % this function uses a threshold to cut of part of
        the edges to get rid
    % of noise that appears in every image and replace
        them by '0'
    % it returns a binary image

    matrix_size = size(img);

    MAX_ROW = matrix_size(1);

    MAX_COLUMN = matrix_size(2);

    mat = zeros(MAX_ROW,MAX_COLUMN,1);

    for row = 1:MAX_ROW
```

```
37
38          for col = 1: MAX_COLUMN
39              if (row>min_x) && (row<max_x) && (col> min_y)
                    && (col<max_y)
40                  mat(row, col) = img(row, col);
41              end
42          end
43      end
44      printed = mat;
45  end
```

2.2.2 threshold in edges

```
1   function printed = print(img, min_x, max_x, min_y, max_y)
2       % this function uses a threshold to cut of part of
            the edges to get rid
3       % of noise that appears in every image and replace
            them by '0'
4       % it returns a binary image
5
6       matrix_size = size(img);
7
8       MAX_ROW = matrix_size(1);
9
10      MAX_COLUMN = matrix_size(2);
11
12      mat = zeros(MAX_ROW,MAX_COLUMN,1);
13
14      for row = 1:MAX_ROW
15
16          for col = 1: MAX_COLUMN
17              if (row>min_x) && (row<max_x) && (col> min_y)
                    && (col<max_y)
18                  mat(row, col) = img(row, col);
19              end
20          end
21      end
22      printed = mat;
23  end
```

2.3 Outline objects

2.3.1 Main outline

```
1     function outlined_matrix = outline(img)
2       % the main outline function, given a binary matrix,
            this function
```

```
    % outlines every shape defined by '1'
    % it returns a matrix with '-1' as value for the
        outlines


    matrix_size = size(img);

    MAX_ROW = matrix_size(1);

    MAX_COLUMN = matrix_size(2);

    x = 0;

    for row = 1: MAX_ROW
        col = 1;
        while col <= MAX_COLUMN
             position = img(row, col);
            if position == 0
                col = col + 1;
            elseif position == -1
                col = skip(img, row, col, MAX_COLUMN);
            elseif position == 1
                x = x + 1;
                img = outline_shape(img, row, col-1,
                    MAX_ROW, MAX_COLUMN);
                col = col - 1;
            end
        end
    end
    disp(x);
    outlined_matrix = img;
end
```

2.3.2 Skip column

```
function new_col = skip(img, row, col, MAX_COLUMN)
    % this function skips the part of the row that is
        defined to be inside
    % a shape
    % it returns the first column number outside a shape

    good_value = 0;
    while (good_value ~= 1) && (col < MAX_COLUMN)
        col = col+ 1;
        if img(row, col) == -1
            good_value = 1;
```

```
        end
    end
    new_col = col +1;
end
```

2.3.3 Outline the shape

```
function outlined_objects = outline_shape(img, row, col,
    MAX_ROW, MAX_COLUMN)
    %Given a binary matrix and a position that is
        connected to a '1', this
    %recursive function outlines the object and returns a
        matrix with the
    %value '-1' surrounding the object

    img(row, col) = -1;
    matrix = surrounded_matrix(img, row, col, MAX_ROW,
        MAX_COLUMN);
    for i = 1:3
        for j = 1:3
            if (matrix(i, j, 1) == 0) & (connected_to_one
                (img, matrix(i,j,2), matrix(i,j,3),
                MAX_ROW, MAX_COLUMN) == 1)
                img = outline_shape(img, matrix(i,j,2),
                    matrix(i,j,3), MAX_ROW, MAX_COLUMN);
            end
        end
    end
   outlined_objects = img;
end
```

2.3.4 Check if a one is connected

```
function is_connected_to_one = connected_to_one(img, row,
    col, MAX_ROW, MAX_COLUMN)
    % given a position that is equal to '0', this
        function checks in a
    % cross shape if a '1' is present

    position = [row, col];
    T = top(position, img, MAX_ROW, MAX_COLUMN);
    R = right(position, img, MAX_ROW, MAX_COLUMN);
    B = bottom(position, img, MAX_ROW, MAX_COLUMN);
    L = left(position, img, MAX_ROW, MAX_COLUMN);

    matrix = [0, T(1), 0; L(1), -1, R(1); 0, B(1), 0];
    is_connected  = 0;
```

```
    for i = 1:3
        for j = 1:3
            if matrix(i,j) == 1
                is_connected = 1;
            end
        end
    end
    is_connected_to_one = is_connected;


end
```

2.3.5 Create surrounding matrix

```
function created_matrix = surrounded_matrix(img, row, col
    , MAX_ROW, MAX_COLUMN)
    % given a position in a matrix, this matrix returns
        the value and
    % position of the 9 surrounding positions

    position = [row, col];
    TL = top_left(position, img, MAX_ROW, MAX_COLUMN);
    T = top(position, img, MAX_ROW, MAX_COLUMN);
    TR = top_right(position, img, MAX_ROW, MAX_COLUMN);
    R = right(position, img, MAX_ROW, MAX_COLUMN);
    BR = bottom_right(position, img, MAX_ROW, MAX_COLUMN)
        ;
    B = bottom(position, img, MAX_ROW, MAX_COLUMN);
    BL = bottom_left(position, img, MAX_ROW, MAX_COLUMN);
    L = left(position, img, MAX_ROW, MAX_COLUMN);

    matrix_1 = [TL(1), T(1), TR(1); L(1), -1, R(1); BL(1)
        , B(1), BR(1)];
    matrix_2 = [TL(2), T(2), TR(2); L(2), row, R(2); BL
        (2), B(2), BR(2)];
    matrix_3 = [TL(3), T(3), TR(3); L(3), col, R(3); BL
        (3), B(3), BR(3)];

    matrix_total = matrix_1;
    matrix_total(:,:,2) = matrix_2;
    matrix_total(:,:,3) = matrix_3;

    created_matrix = matrix_total;

end
```

2.3.6 all surrounding positions

```
function placing = top_left(position, img, MAX_ROW,
    MAX_COLUMB)
    % returns the position top left of the given position
    x = position(1) -1;
    y = position(2) -1;

    if (0 < x) && (x <= MAX_ROW) && (0 < y) && (y <=
        MAX_COLUMB)

        value = img(x, y);
        placing = [value, x, y];
    else

        value = -2;
        placing = [value, x, y];
    end
end
function placing = top(position, img, MAX_ROW, MAX_COLUMB
    )
    % returns the position above the given position
    x = position(1) -1;
    y = position(2) ;

    if (0 < x) && (x <= MAX_ROW) && (0 < y) && (y <=
        MAX_COLUMB)
        value = img(x, y);
        placing = [value, x, y];
    else

        value = -2;
        placing = [value, x, y];
    end
end
function placing = top_right(position, img, MAX_ROW,
    MAX_COLUMB)
    % returns the position top right of the given
        position
    x = position(1) - 1;
    y = position(2) + 1;

    if (0 < x) && (x <= MAX_ROW) && (0 < y) && (y <=
        MAX_COLUMB)

        value = img(x, y);
```

```
38          placing = [value, x, y];
39      else
40
41          value = -2;
42          placing = [value, x, y];
43      end
44  end
45  function placing = right(position, img, MAX_ROW,
        MAX_COLUMB)
46      % returns the position to the right of the given
            position
47      x = position(1) ;
48      y = position(2) +1;
49
50      if (0 < x) && (x <= MAX_ROW) && (0 < y) && (y <=
            MAX_COLUMB)
51
52          value = img(x, y);
53          placing = [value, x, y];
54      else
55
56          value = -2;
57          placing = [value, x, y];
58      end
59  end
60  function placing = bottom_right(position, img, MAX_ROW,
        MAX_COLUMB)
61      % returns the position bottom right of the given
            position
62      x = position(1) +1;
63      y = position(2) +1;
64
65      if (0 < x) && (x <= MAX_ROW) && (0 < y) && (y <=
            MAX_COLUMB)
66
67          value = img(x, y);
68          placing = [value, x, y];
69      else
70
71          value = -2;
72          placing = [value, x, y];
73      end
74  end
75  function placing = bottom(position, img, MAX_ROW,
        MAX_COLUMB)
76      % returns the position below the given position
```

```
77      x = position(1) +1;
78      y = position(2) ;
79
80      if (0 < x) && (x <= MAX_ROW) && (0 < y) && (y <=
            MAX_COLUMB)
81
82          value = img(x, y);
83          placing = [value, x, y];
84      else
85
86          value = -2;
87          placing = [value, x, y];
88      end
89  end
90  function placing = bottom_left(position, img, MAX_ROW,
        MAX_COLUMB)
91      % returns the position bottom left of the given
            position
92      x = position(1) +1;
93      y = position(2) -1;
94
95      if (0 < x) && (x <= MAX_ROW) && (0 < y) && (y <=
            MAX_COLUMB)
96
97          value = img(x, y);
98          placing = [value, x, y];
99      else
100
101         value = -2;
102         placing = [value, x, y];
103     end
104 end
105 function placing = left(position, img, MAX_ROW,
        MAX_COLUMB)
106     % returns the position to the left of the given
            position
107     x = position(1);
108     y = position(2) -1;
109
110     if (0 < x) && (x <= MAX_ROW) && (0 < y) && (y <=
            MAX_COLUMB)
111
112         value = img(x, y);
113         placing = [value, x, y];
114     else
115
```

```
116             value = -2;
117             placing = [value, x, y];
118         end
119     end
```

3 Functions for overlap

3.1 Get the needed values

3.1.1 Crop depth and RGB to the same aspect ratio

```
1   function [reformed_depth,reformed_color,
        resulting_height_angle,resulting_width_angle] = reform
        (depth, color) %met h= height camera
2       % this function modifies the incomming color and
            depth matrices to give
3       % them the same aspect ratio
4
5       %breedte van color naar 70.6 brengen
6       width_color_angle = 84.1;
7       height_color_angle = 53.8;
8
9       width_depth_angle = 70.6;
10      height_depth_angle = 60;
11
12      resulting_height_angle = height_color_angle;
13      resulting_width_angle = width_depth_angle;
14
15      [~, nb_columns_color,~]=size(color);
16      nb_pixels_color_per_degree_width = nb_columns_color /
            width_color_angle;
17
18
19      nb_width_pixels_removed_color = (width_color_angle-
            width_depth_angle) *
            nb_pixels_color_per_degree_width ;
20          %totaal aantal pixels dat in de breedte
                weggehaald moeten worden bij color
21
22      reformed_color = color(:,80 + round(
            nb_width_pixels_removed_color/2,0): round(
            nb_columns_color-(nb_width_pixels_removed_color/2)
            ,0),:);
23          %Dit is een 1080 x (aangepaste breedte) matrix
24
25
```

```
26      % hoogte van depth naar 53.8 brenge
27      [nb_rows_depth,~]=size(depth);
28
29      nb_pixels_depth_per_degree_height = nb_rows_depth /
            height_color_angle;
30
31      nb_height_pixels_removed_depth = (height_depth_angle-
            height_color_angle)*
            nb_pixels_depth_per_degree_height;
32      reformed_depth = depth(round(
            nb_height_pixels_removed_depth/2,0): round(
            nb_rows_depth -(nb_height_pixels_removed_depth/2)
            ,0),:);
33  end
```

3.1.2 Get the pixels per mm

```
1   function [pipemm_depth_H, pipemm_depth_W, pipemm_color_H,
        pipemm_color_W] = get_pipemm(res_height_angle,
        res_width_angle, h, reformed_depth,reformed_color)
2       % this function returns the pixels per millimeter for
            the given depth
3       % and color matrices
4
5       depth_size = size(reformed_depth);
6
7       MAX_ROW_DEPTH = depth_size(1);
8
9       MAX_COLUMN_DEPTH = depth_size(2);
10
11      color_size = size(reformed_color);
12
13      MAX_ROW_COLOR = color_size(1);
14
15      MAX_COLUMN_COLOR = color_size(2);
16
17      tot_width = 2*h*tan(((res_width_angle)/2)*(pi/180));
18
19      tot_height = 2*h*tan(((res_height_angle)/2)*(pi/180))
            ;
20
21      pipemm_depth_H = MAX_ROW_DEPTH/tot_height;
22
23      pipemm_depth_W = MAX_COLUMN_DEPTH/tot_width;
24
25      pipemm_color_H = MAX_ROW_COLOR/tot_height;
```

```
26
27      pipemm_color_W = MAX_COLUMN_COLOR/tot_width;
28
29  end
```

3.1.3 Get the proportion between depth and RGB pixels

```
1   function [prop,nb_rows_color , nb_columns_color,
        nb_rows_depth, nb_columns_depth] = proportion(
        reformed_depth , reformed_color)
2       % this function returns the size of the given color
            and depth matrices,
3       % and the proportion between the depth and color
            pixels
4
5       [nb_rows_color , nb_columns_color,~]=size(
            reformed_color);
6       [nb_rows_depth, nb_columns_depth]= size(
            reformed_depth);
7
8       nb_pixels_color=nb_rows_color * nb_columns_color;
9       nb_pixels_depth=nb_rows_depth * nb_columns_depth;
10
11      x= max(nb_pixels_color,nb_pixels_depth);
12      y= min(nb_pixels_color,nb_pixels_depth);
13
14      prop = x/y;
15
16  end
17
18  function the_size=size_matching(prop)
19      the_size= round(sqrt(prop));
20  end
```

3.1.4 Get the exact positions from depth to RGB

```
1   function [row_start, row_stop, col_start, col_stop]=
        depth_to_color(pipemm_depth_H , pipemm_depth_W ,
        pipemm_color_H , pipemm_color_W,row, col,the_size,
        nb_rows_color , nb_columns_color)
2
3       mm_width_from_left = col/pipemm_depth_W;
4       mm_height_from_top = row/pipemm_depth_H;
5
6       corr_pixel_col_color = round(mm_width_from_left *
            pipemm_color_W);
```

```
7       corr_pixel_row_color = round(mm_height_from_top *
            pipemm_color_H);
8
9       steps = floor(the_size/2);
10      %steps=5;
11
12      row_start=corr_pixel_row_color-steps;
13      row_stop=corr_pixel_row_color+steps;
14
15      col_start=corr_pixel_col_color-steps;
16      col_stop=corr_pixel_col_color+steps;
17
18      if row_start<1
19          row_start = 1;
20      end
21
22      if row_stop > nb_rows_color
23          row_stop=nb_rows_color;
24      end
25
26      if col_start<1
27          col_start = 1;
28      end
29
30      if col_stop > nb_columns_color
31          col_stop = nb_columns_color;
32      end
33
34
35
36
37  end
```

3.2 Overlap from depth to RGB

```
1   function overlapped_matrix = overlap_depth_to_RGB(
        reformed_depth, reformed_color, pipemm_depth_H,
        pipemm_depth_W, pipemm_color_H, pipemm_color_W,
        the_size,nb_rows_color, nb_columns_color)
2
3       depth_size = size(reformed_depth);
4
5       MAX_ROW_DEPTH = depth_size(1);
6
7       MAX_COLUMN_DEPTH = depth_size(2);
8
```

```
9       for row = 1:MAX_ROW_DEPTH
10          for col = 1:MAX_COLUMN_DEPTH
11              if(reformed_depth(row, col, 1) == -1)
12                  [row_start, row_stop, col_start, col_stop
                        ] = depth_to_color(pipemm_depth_H ,
                        pipemm_depth_W , pipemm_color_H ,
                        pipemm_color_W,row, col,the_size,
                        nb_rows_color , nb_columns_color);
13                  reformed_color(row_start:row_stop,
                        col_start:col_stop, 1) = 255;
14                  reformed_color(row_start:row_stop,
                        col_start:col_stop, 2) = 0;
15                  reformed_color(row_start:row_stop,
                        col_start:col_stop, 3) = 0;
16              end
17          end
18      end
19      overlapped_matrix = reformed_color;
20  end
```

3.3 Crop RGB to basket

```
1   function usefull_matrix = crop_RGB_to_basket(img)
2
3       z = 20;
4
5       matrix_size = size(img);
6
7       MAX_ROW = matrix_size(1);
8
9       MAX_COLUMN = matrix_size(2);
10
11      row = 1;
12      col = 1;
13      %thicken the edge
14      for i = (1+z): (MAX_ROW-z)
15          for j = (1+z) : (MAX_COLUMN-z)
16              if (img(i, j, 1) == 255) && (img(i, j, 2) ==
                    0) && (img(i, j, 3) == 0)
17                  img(i-z:i+z, j-z:j+z, 1) = 0;
18                  img(i-z:i+z, j-z:j+z, 2) = 0;
19                  img(i-z:i+z, j-z:j+z, 3) = 255;
20              end
21          end
22      end
23      %go from left to right
```

```
24      while (row ~= MAX_ROW)
25          if col == MAX_COLUMN
26              col = 1;
27              row = row + 1;
28
29          elseif (img(row, col, 1) == 0 ) && (img(row, col,
                2) == 0) && (img(row, col, 3) == 255)
30              col = 1;
31              row = row + 1;
32
33          else
34              img(row, col, 1) = 255;
35              img(row, col, 2) = 255;
36              img(row, col, 3) = 255;
37              col = col + 1;
38          end
39      end
40      %go from right to left
41      row = MAX_ROW;
42      col = MAX_COLUMN;
43      while (row ~= 1)
44          if col == 1
45              col = MAX_COLUMN;
46              row = row - 1;
47
48          elseif (img(row, col, 1) == 0) && (img(row, col,
               2) == 0) && (img(row, col, 3) == 255)
49              col = MAX_COLUMN;
50              row = row - 1;
51
52          else
53              img(row, col, 1) = 255;
54              img(row, col, 2) = 255;
55              img(row, col, 3) = 255;
56              col = col - 1;
57          end
58      end
59      %go from top to bottom
60      row = 1;
61      col = 1;
62      while (col ~= MAX_COLUMN)
63          if row == MAX_ROW
64              row = 1;
65              col = col + 1;
66
67          elseif (img(row, col, 1) == 0) && (img(row, col,
```

```
            2) == 0) && (img(row, col, 3) == 255)
68              row = 1;
69              col = col + 1;
70
71          else
72              img(row, col, 1) = 255;
73              img(row, col, 2) = 255;
74              img(row, col, 3) = 255;
75              row = row + 1;
76          end
77      end
78      %go from bottom to top
79      row = MAX_ROW;
80      col = MAX_COLUMN;
81      while (col ~= 1)
82          if row == 1
83              row = MAX_ROW;
84              col = col - 1;
85
86          elseif (img(row, col, 1) == 0) && (img(row, col,
                2) == 0) && (img(row, col, 3) == 255)
87              row = MAX_ROW;
88              col = col - 1;
89
90          else
91              img(row, col, 1) = 255;
92              img(row, col, 2) = 255;
93              img(row, col, 3) = 255;
94              row = row - 1;
95          end
96      end
97      %add in the white edge
98      for i = 1: MAX_ROW
99          for j = 1 : MAX_COLUMN
100             if (img(i, j, 1) == 0) && (img(i, j, 2) == 0)
                    && (img(i, j, 3) == 255)
101                 img(i,j, 1) = 255;
102                 img(i,j, 2) = 255;
103                 img(i,j, 3) = 255;
104             end
105         end
106     end
107
108
109
110
```

```
111     usefull_matrix = img;
112
113 end
```

4 Functions for colour

4.1 Greyscale

```
1 function grey = greyscale(img)
2
3     grey = img(:,:,1) * 0.2989 + img(:,:,2) * 0.5870 +
          img(:,:,3) * 0.1140;
4
5 end
```

4.2 Blurring the image

4.2.1 Mean blur

```
1 function mean_blurred = mean_blur(img)
2     mean = (1/9) * [ 1 1 1; 1 1 1; 1 1 1];
3     mean_blurred = conv2(img, mean);
4 end
```

4.2.2 Gaussian blur

```
1 function gaussian_blurred = gaussian_blur(img)
2     gaussian = (1/159) * [2 4 5 4 2; 4 9 12 9 4; 5 12 15
          12 5; 4 9 12 9 4; 2 4 5 4 2;];
3     gaussian_blurred = conv2(img, gaussian);
4 end
```

4.3 Laplacian edge detect

```
1 function edge = edge_detect(img)
2     klaplace=[0 -1 0; -1 4 -1;  0 -1 0];              %
          Laplacian filter kernel
3     edge=conv2(img,klaplace);                          %
          convolve test img with
4 end
```

4.4 Threshold for the edge

```
1 function thresholded_img = threshold_edge(img,
      THRESHOLD_VALUE)
2     THRESHOLD_VALUE = 2;
3     matrix_size = size(img);
4     MAX_ROW = matrix_size(1);
```

```
    MAX_COLUMN = matrix_size(2);
    THICKNESS = 1; % 3

    thresholded_img = zeros(MAX_ROW,MAX_COLUMN,1);
    for row=1:MAX_ROW
        for col=1:MAX_COLUMN
            if img(row, col) > THRESHOLD_VALUE
                value = 1;
                for i=1:THICKNESS
                    % Create thicker edges (edges of
                        THICKNESS pixels thick)
                    if (col - i) > 0
                        thresholded_img(row, col-i) = 1;
                    end

                    if (col + i) <= MAX_COLUMN
                        thresholded_img(row, col+i) = 1;
                    end

                    if (row - i) > 0
                        thresholded_img(row -i, col) = 1;
                    end

                    if (row + i) <+ MAX_ROW
                        thresholded_img(row +i, col) = 1;
                    end

                end
            else
                value = 0;
            end
            thresholded_img(row, col) = value;
        end
    end
end
```

4.5 Group the edges

```
function [result, nb_of_groups] = group(img,
    SAME_PIXEL_SEARCH_GRID_SIZE, GROUP_SEARCH_GRID_SIZE,
    MIN_NB_SURROUNDING_PIXELS)
    % Goal, group pixels.
    % First loop from left to right to find an object
    % Check if it's connected
    % Number connected pixels in the second dimension
    WHITE = 1;
```

```
7       BLACK = 0;
8       matrix_size = size(img);
9       MAX_ROW = matrix_size(1);
10      MAX_COLUMN = matrix_size(2);
11
12      groups = 0;
13
14      result = zeros(MAX_ROW,MAX_COLUMN,2); % Dimension 2
            is for the group number.
15      for row=1:MAX_ROW
16          for col=1:MAX_COLUMN
17            pixel_value = img(row, col);
18            result(row, col,1) = pixel_value; % Transfer
                  picture to result variable (in dim 1)
19            if pixel_value == BLACK
20                % This is an edge
21                connecting_pixels = same_pixels_in_range(
                      img, row, col,
                      SAME_PIXEL_SEARCH_GRID_SIZE);
22                %connecting_pixels = real_connecting_pixels
                      (img, row, col);
23
24
25                if connecting_pixels >
                      MIN_NB_SURROUNDING_PIXELS
26                    % This is defined as an object outline.
27                    group_number = find_group_in_range(
                          result, row, col,
                          GROUP_SEARCH_GRID_SIZE);
28
29                    if group_number == 0
30                        % assign new group
31                        groups = groups + 1;
32                        group_number = groups;
33                    end
34                    %disp("connecting pixels=" +
                          connecting_pixels + " group number="
                           + group_number + " pos=" + row + ",
                           " + col);
35                    result(row, col, 2) = group_number;
36                end
37            end
38            %imagesc(result(:,:,2));
39
40          end
41      end
```

```
    nb_of_groups = groups;
end
```

4.6 Regroup the edges

```
function [result, nb_groups] = regroup(grouped_img,
    nb_of_groups, MIN_ROW_LINES_BETWEEN_GROUPS)
    % Loop from (right)top to (left)bottom
    % Check if there are connecting groups.

    matrix_size = size(grouped_img);
    MAX_ROW = matrix_size(1);
    MAX_COLUMN = matrix_size(2);

    nb_groups = nb_of_groups;
    for col_i=1:MAX_COLUMN
        for row=1:MAX_ROW
            col = MAX_COLUMN - col_i+1;
            group_nb = grouped_img(row, col, 2);
            if group_nb ~= 0
                for row_i=1:MIN_ROW_LINES_BETWEEN_GROUPS
                    if is_valid_position(MAX_ROW,
                        MAX_COLUMN, row + row_i, col) == 1
                         && grouped_img(row + row_i, col,
                        2) ~= 0 && grouped_img(row+row_i,
                        col,2) ~= group_nb
                        % Found a different group in the
                            next 5 pixels
                        % below this one
                        % Replace next group with
                            previous group number
                        grouped_img = group_replace(
                            grouped_img, grouped_img(row+
                            row_i, col, 2), group_nb);
                        nb_groups = nb_groups - 1;
                        break;
                    end
                end
            end
        end
    end


    result = grouped_img;
end
```

4.7 Find the corner points

```
function result = find_corner_points(img, nb_groups)
    % Loop through grouped image
    % find MIN_ROW & MIN_COL and MAX_ROW & MAX_COL
     matrix_size = size(img);
    MAX_ROW = matrix_size(1);
    MAX_COLUMN = matrix_size(2);

    GROUP_MAX_ROW = zeros(1,nb_groups);
    GROUP_MAX_COL = zeros(1,nb_groups);
    GROUP_MIN_ROW = zeros(1,nb_groups);
    GROUP_MIN_COL = zeros(1,nb_groups);

     for row=1:MAX_ROW
         for col=1:MAX_COLUMN
             group_nb = img(row, col, 2);
             if group_nb ~= 0
                 % Group found (==0 means nothing is set)
                 if GROUP_MAX_ROW(1,group_nb) == 0 ||
                     GROUP_MAX_ROW(1,group_nb) < row
                      GROUP_MAX_ROW(1,group_nb) = row;
                 end

                 if GROUP_MAX_COL(1,group_nb) == 0 ||
                     GROUP_MAX_COL(1,group_nb) < col
                      GROUP_MAX_COL(1,group_nb) = col;
                 end

                 if GROUP_MIN_ROW(1,group_nb) == 0 ||
                     GROUP_MIN_ROW(1,group_nb) > row
                      GROUP_MIN_ROW(1,group_nb) = row;
                 end
                 if GROUP_MIN_COL(1,group_nb) == 0 ||
                     GROUP_MIN_COL(1,group_nb) > col
                      GROUP_MIN_COL(1,group_nb) = col;
                 end
             end

         end
         result = [GROUP_MIN_ROW; GROUP_MIN_COL;
             GROUP_MAX_ROW; GROUP_MAX_COL];
     end

end
```

4.8 Remove objects within objects

4.8.1 Remove box edge

```
function [result, new_nb_of_groups] = remove_box_edge(
    corner_points, nb_of_groups)
    mat_size = size(corner_points);
    groups = mat_size(2);
    surfaces = zeros(groups); % Every column is a group,
        the value is the distance

    for i=1:groups

        min_row = corner_points(1,i);
        min_col = corner_points(2,i) ;
        max_row = corner_points(3,i);
        max_col = corner_points(4,i);

        surfaces(i) = (max_row - min_row) * (max_col -
            min_col);
    end

    %Now find biggest surface
    [max_value, max_col] = max(surfaces);
    for i=1:4
        % Set the coordinates of the outer points to 0
        corner_points(i, max_col) = 0;
    end

    result = corner_points;
    new_nb_of_groups = nb_of_groups-1;
end
```

4.8.2 Remove corner points within corner points

```
function [updated_corner_points, nb_of_groups] =
    remove_corner_points_within_corner_points(
    corner_points, nb_groups)
    mat_size = size(corner_points);
    groups = mat_size(2); % This is the original
        number_of_groups
    nb_of_groups = nb_groups; % This is the
        number_of_groups after regroup
    updated_corner_points = corner_points;

    for first=1:groups
        % Loop through every group
```

```
9           % Now draw boundary box
10          min_row_first = corner_points(1,first);
11          min_col_first = corner_points(2,first);
12          max_row_first = corner_points(3,first);
13          max_col_first = corner_points(4,first);
14          for second = 1:groups
15              if first ~= second && max_row_first ~= 0 &&
                    corner_points(4, second) ~= 0 % If the max
                     values would be 0, this won't be a group
16                  % Same groups, cant lay within eachother
17                  min_row_second = corner_points(1,second);
18                  min_col_second = corner_points(2,second);
19                  max_row_second = corner_points(3,second);
20                  max_col_second = corner_points(4,second);
21
22                  % Check if second lays within first
23
24                  if min_row_second >= min_row_first &&
                        min_col_second >= min_col_first &&
                        max_row_second <= max_row_first &&
                        max_col_second <= max_col_first
25                      % Second object lays within first
                            object
26                      % Remouve this object
27                      updated_corner_points(:, second) =
                            zeros(4,1);
28
29                      nb_of_groups = nb_of_groups - 1;
30                  end
31              end
32          end
33      end
34  end
```

4.9 Draw the boundary box

```
1   function img = draw_red_boundary_box(img, corner_points,
        top_row, top_col)
2       mat_size = size(corner_points);
3       groups = mat_size(2);
4       THICKNESS = 5;
5
6       matrix_size = size(img);
7       MAX_ROW = matrix_size(1);
8       MAX_COLUMN = matrix_size(2);
9       for i=1:groups
```

```
10          % Loop through every group
11          % Now draw boundary box
12          min_row = corner_points(1,i) + top_row;
13          min_col = corner_points(2,i) + top_col;
14          max_row = corner_points(3,i) + top_row;
15          max_col = corner_points(4,i) + top_col;
16          % First draw horizontal lines
17          for col=min_col:max_col
18              for e=0:THICKNESS
19                  if is_valid_position(MAX_ROW, MAX_COLUMN,
                        min_row+e, col) == 1
20                      img(min_row+e, col, 1) = 255;
21                      img(min_row+e, col, 2) = 1;
22                      img(min_row+e, col, 3) = 1;
23                  end
24                  if is_valid_position(MAX_ROW, MAX_COLUMN,
                        max_row-e, col) == 1
25                      img(max_row-e, col,1) = 255;
26                      img(max_row-e, col,2) = 1;
27                      img(max_row-e, col,3) = 1;
28                  end
29              end
30          end
31
32          % Vertical lines
33          for row=min_row:max_row
34              for e=0:THICKNESS
35                  if is_valid_position(MAX_ROW, MAX_COLUMN,
                        row, min_col + e) == 1
36                      img(row, min_col+e, 1) = 255;
37                      img(row, min_col+e, 2) = 1;
38                      img(row, min_col+e, 3) = 1;
39                  end
40                  if is_valid_position(MAX_ROW, MAX_COLUMN,
                        row, max_col - e) == 1
41                      img(row, max_col-e, 1) = 255;
42                      img(row, max_col-e, 2) = 1;
43                      img(row, max_col-e, 3) = 1;
44                  end
45              end
46
47          end
48      end
49  end
```