

# Contents

<b>1</b>	<b>Initialising functions</b>	<b>2</b>
<b>2</b>	<b>Functions for depth</b>	<b>5</b>
2.1	Sobel operator . . . . .	5
2.2	Threshold for depth . . . . .	5
2.2.1	threshold in values . . . . .	5
2.2.2	threshold in edges . . . . .	6
2.3	Outline objects . . . . .	6
2.3.1	Main outline . . . . .	6
2.3.2	Skip column . . . . .	7
2.3.3	Outline the shape . . . . .	7
2.3.4	Check if a one is connected . . . . .	8
2.3.5	Create surrounding matrix . . . . .	8
2.3.6	all surrounding positions . . . . .	9
<b>3</b>	<b>Functions for overlap</b>	<b>11</b>
3.1	Get the needed values . . . . .	11
3.1.1	Crop depth and RGB to the same aspect ratio . . . . .	11
3.1.2	Get the pixels per mm . . . . .	12
3.1.3	Get the proportion between depth and RGB pixels . . . . .	12
3.1.4	Get the exact positions from depth to RGB . . . . .	13
3.2	Overlap from depth to RGB . . . . .	13
3.3	Crop RGB to basket . . . . .	14
<b>4</b>	<b>Functions for colour</b>	<b>16</b>
4.1	Greyscale . . . . .	16
4.2	Blurring the image . . . . .	16
4.2.1	Mean blur . . . . .	16
4.2.2	Gaussian blur . . . . .	16
4.3	Laplacian edge detect . . . . .	17
4.4	Threshold for the edge . . . . .	17
4.5	Group the edges . . . . .	18
4.6	Regroup the edges . . . . .	18
4.7	Find the corner points . . . . .	19
4.8	Remove objects within objects . . . . .	20
4.8.1	Remove box edge . . . . .	20
4.8.2	Remove corner points within corner points . . . . .	20
4.9	Draw the boundary box . . . . .	21
<b>5</b>	<b>Implementation: packaging code</b>	<b>22</b>
5.1	Gathering objects . . . . .	22
5.1.1	Get objects . . . . .	22
5.1.2	Object highlighter . . . . .	23
5.1.3	Insertion sort . . . . .	24
5.1.4	Single object . . . . .	25
5.2	fitting the objects . . . . .	25
5.2.1	Boundary boxed image rotator . . . . .	25
5.2.2	Packaged object . . . . .	26
5.2.3	Rotator . . . . .	27
5.2.4	Generic crop . . . . .	27
5.3	total packaging . . . . .	28
5.3.1	Smallest Package . . . . .	28
5.3.2	Black Edged . . . . .	30
5.3.3	Position tester . . . . .	31
5.3.4	Package appender . . . . .	31

# 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
105 disp("Step 6: thresholding edge");
106 without_noise_removal = threshold_edge(remove_boundary(first_edge_detect,
    15), THRESHOLD.VALUE); % Remove boundary around image & threshold the
    edges.
107 disp("Step 7: noise removing...");
108 %with_noise_removal = noise_deletion(without_noise_removal,5); % Noise
    removal
109 with_noise_removal = without_noise_removal;
110 disp("Step 8: grouping...");
111 [grouped, nb_of_groups] = group(~with_noise_removal,
    SAME_PIXELS.SEARCH_GRID_SIZE, GROUP_SEARCH_GRID_SIZE,
    MIN_NB.SURROUNDING_PIXELS); % Group pixels together
112
113 disp("Step 9: regrouping...");
114 [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
115
116 %Find corner points of object (not really corner points on the boundary,
117 %but corner points for the boundary box)
118 disp("Step 10: calculating corner points...");
119 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!
120
121 disp("Step 11: removing objects within objects...");
122 %[updated_corner_points, nb_of_groups3] =
    remove_corner_points_within_corner_points(corner_points, nb_of_groups2
    ); % To remove objects within objects
123 [updated_corner_points, nb_of_groups3] = remove_box_edge(corner_points,
    nb_of_groups2);
124 [updated_corner_points, nb_of_groups3] =
    remove_corner_points_within_corner_points(updated_corner_points,
    nb_of_groups3);
125 %updated_corner_points = corner_points;
126 %nb_of_groups3 = nb_of_groups2;
127
128 disp("Step 12: drawing boundary boxes...");
129 boundary_box = draw_boundary_box(A, updated_corner_points);
130 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 imshow(red_boundary_box, []);
134 title("# objects: "+ nb_of_groups3);

```

```

135
136 %%
137 % Starting the packaging pocess
138
139 % Gathering every object from the original image
140 objects = get_objects(updated_corner_points , gray_image , regrouped);
141 % Creating the total package
142 total_package = smallest_package(objects);
143
144 % Showing the end package
145 imshow(total_package , []);

```

## 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 gradient
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

```

1 function thresholded = threshold(img, min_thresh, max_thresh)
2     % run the image through a threshold to get rid of impossible values
3     % this function returns a binary matrix with a 1 on the edges
4
5     matrix_size = size(img);
6
7     MAXROW = matrix_size(1);
8
9     MAXCOLUMN = matrix_size(2);
10
11    for row = 1 : MAXROW
12        for col = 1: MAXCOLUMN
13            if (img(row, col) > min_thresh) && (img(row, col) < max_thresh)
14                img(row, col) = 1;
15            else
16                img(row, col) = 0;
17            end
18        end
19    end
20    thresholded = img;
21 end
22
23 function printed = print(img, min_x, max_x, min_y, max_y)
24     % this function uses a threshold to cut of part of the edges to get
25     % rid
26     % of noise that appears in every image and replace them by '0'

```

```

26     % it returns a binary image
27
28     matrix_size = size(img);
29
30     MAXROW = matrix_size(1);
31
32     MAXCOLUMN = matrix_size(2);
33
34     mat = zeros(MAXROW,MAXCOLUMN,1);
35
36     for row = 1:MAXROW
37
38         for col = 1: MAXCOLUMN
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     MAXROW = matrix_size(1);
9
10    MAXCOLUMN = matrix_size(2);
11
12    mat = zeros(MAXROW,MAXCOLUMN,1);
13
14    for row = 1:MAXROW
15
16        for col = 1: MAXCOLUMN
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
3     % outlines every shape defined by '1'
4     % it returns a matrix with '-1' as value for the outlines
5
6

```

```

7     matrix_size = size(img);
8
9     MAXROW = matrix_size(1);
10
11    MAXCOLUMN = matrix_size(2);
12
13    x = 0;
14
15    for row = 1: MAXROW
16        col = 1;
17        while col <= MAXCOLUMN
18            position = img(row, col);
19            if position == 0
20                col = col + 1;
21            elseif position == -1
22                col = skip(img, row, col, MAXCOLUMN);
23            elseif position == 1
24                x = x + 1;
25                img = outline_shape(img, row, col-1, MAXROW, MAXCOLUMN)
26                ;
27                col = col - 1;
28            end
29        end
30        disp(x);
31        outlined_matrix = img;
32    end

```

### 2.3.2 Skip column

```

1 function new_col = skip(img, row, col, MAXCOLUMN)
2     % this function skips the part of the row that is defined to be
3     % inside
4     % a shape
5     % it returns the first column number outside a shape
6
7     good_value = 0;
8     while (good_value ~= 1) && (col < MAXCOLUMN)
9         col = col+ 1;
10        if img(row, col) == -1
11            good_value = 1;
12        end
13    end
14    new_col = col +1;
15 end

```

### 2.3.3 Outline the shape

```

1 function outlined_objects = outline_shape(img, row, col, MAXROW,
2     MAXCOLUMN)
3     %Given a binary matrix and a position that is connected to a '1',
4     %this
5     %recursive function outlines the object and returns a matrix with the
6     %value '-1' surrounding the object
7
8     img(row, col) = -1;
9     matrix = surrounded_matrix(img, row, col, MAXROW, MAXCOLUMN);
10    for i = 1:3

```

```

9         for j = 1:3
10             if (matrix(i, j, 1) == 0) & (connected_to_one(img, matrix(i,j
,2), matrix(i,j,3), MAXROW, MAXCOLUMN) == 1)
11                 img = outline_shape(img, matrix(i,j,2), matrix(i,j,3),
MAXROW, MAXCOLUMN);
12             end
13         end
14     end
15     outlined_objects = img;
16 end

```

#### 2.3.4 Check if a one is connected

```

1 function is_connected_to_one = connected_to_one(img, row, col, MAXROW,
MAXCOLUMN)
2     % given a position that is equal to '0', this function checks in a
3     % cross shape if a '1' is present
4
5     position = [row, col];
6     T = top(position, img, MAXROW, MAXCOLUMN);
7     R = right(position, img, MAXROW, MAXCOLUMN);
8     B = bottom(position, img, MAXROW, MAXCOLUMN);
9     L = left(position, img, MAXROW, MAXCOLUMN);
10
11     matrix = [0, T(1), 0; L(1), -1, R(1); 0, B(1), 0];
12     is_connected = 0;
13     for i = 1:3
14         for j = 1:3
15             if matrix(i,j) == 1
16                 is_connected = 1;
17             end
18         end
19     end
20     is_connected_to_one = is_connected;
21
22
23 end

```

#### 2.3.5 Create surrounding matrix

```

1 function created_matrix = surrounded_matrix(img, row, col, MAXROW,
MAXCOLUMN)
2     % given a position in a matrix, this matrix returns the value and
3     % position of the 9 surrounding positions
4
5     position = [row, col];
6     TL = top_left(position, img, MAXROW, MAXCOLUMN);
7     T = top(position, img, MAXROW, MAXCOLUMN);
8     TR = top_right(position, img, MAXROW, MAXCOLUMN);
9     R = right(position, img, MAXROW, MAXCOLUMN);
10    BR = bottom_right(position, img, MAXROW, MAXCOLUMN);
11    B = bottom(position, img, MAXROW, MAXCOLUMN);
12    BL = bottom_left(position, img, MAXROW, MAXCOLUMN);
13    L = left(position, img, MAXROW, MAXCOLUMN);
14
15    matrix_1 = [TL(1), T(1), TR(1); L(1), -1, R(1); BL(1), B(1), BR(1)];
16    matrix_2 = [TL(2), T(2), TR(2); L(2), row, R(2); BL(2), B(2), BR(2)];
17    matrix_3 = [TL(3), T(3), TR(3); L(3), col, R(3); BL(3), B(3), BR(3)];

```



```

18
19     matrix_total = matrix_1;
20     matrix_total(:, :, 2) = matrix_2;
21     matrix_total(:, :, 3) = matrix_3;
22
23     created_matrix = matrix_total;
24
25 end

```

### 2.3.6 all surrounding positions

```

1 function placing = top_left(position, img, MAXROW, MAXCOLUMB)
2     % returns the position top left of the given position
3     x = position(1) - 1;
4     y = position(2) - 1;
5
6     if (0 < x) && (x <= MAXROW) && (0 < y) && (y <= MAXCOLUMB)
7
8         value = img(x, y);
9         placing = [value, x, y];
10    else
11
12        value = -2;
13        placing = [value, x, y];
14    end
15 end
16 function placing = top(position, img, MAXROW, MAXCOLUMB)
17     % returns the position above the given position
18     x = position(1) - 1;
19     y = position(2) ;
20
21     if (0 < x) && (x <= MAXROW) && (0 < y) && (y <= MAXCOLUMB)
22         value = img(x, y);
23         placing = [value, x, y];
24     else
25
26        value = -2;
27        placing = [value, x, y];
28    end
29 end
30 function placing = top_right(position, img, MAXROW, MAXCOLUMB)
31     % returns the position top right of the given position
32     x = position(1) - 1;
33     y = position(2) + 1;
34
35     if (0 < x) && (x <= MAXROW) && (0 < y) && (y <= MAXCOLUMB)
36
37        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, MAXROW, MAXCOLUMB)
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 <= MAXROW) && (0 < y) && (y <= MAXCOLUMB)
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, MAXROW, MAXCOLUMB)
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 <= MAXROW) && (0 < y) && (y <= MAXCOLUMB)
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, MAXROW, MAXCOLUMB)
76     % returns the position below the given position
77     x = position(1) +1;
78     y = position(2) ;
79
80     if (0 < x) && (x <= MAXROW) && (0 < y) && (y <= MAXCOLUMB)
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, MAXROW, MAXCOLUMB)
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 <= MAXROW) && (0 < y) && (y <= MAXCOLUMB)
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, MAXROW, MAXCOLUMB)
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 <= MAXROW) && (0 < y) && (y <= MAXCOLUMB)
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,
2     resulting_width_angle] = reform(depth, color) %met h= height camera
3     % this function modifies the incoming color and depth matrices to
4     % give
5     % them the same aspect ratio
6
7     %breedte van color naar 70.6 brengen
8     width_color_angle = 84.1;
9     height_color_angle = 53.8;
10
11     width_depth_angle = 70.6;
12     height_depth_angle = 60;
13
14     resulting_height_angle = height_color_angle;
15     resulting_width_angle = width_depth_angle;
16
17     [~, nb_columns_color, ~] = size(color);
18     nb_pixels_color_per_degree_width = nb_columns_color /
19         width_color_angle;
20
21     nb_width_pixels_removed_color = (width_color_angle - width_depth_angle)
22         * nb_pixels_color_per_degree_width ;
23     %totaal aantal pixels dat in de breedte weggehaald moeten worden
24     %bij color
25
26     reformed_color = color(:, 80 + round(nb_width_pixels_removed_color
27         / 2, 0): round(nb_columns_color - (nb_width_pixels_removed_color / 2), 0)
28         , :);
29     %Dit is een 1080 x (aangepaste breedte) matrix
30
31     % 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     MAXROWDEPTH = depth_size(1);
8
9     MAXCOLUMNDEPTH = depth_size(2);
10
11     color_size = size(reformed_color);
12
13     MAXROWCOLOR = color_size(1);
14
15     MAXCOLUMNCOLOR = 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 = MAXROWDEPTH/tot_height;
22
23     pipemm_depth_W = MAXCOLUMNDEPTH/tot_width;
24
25     pipemm_color_H = MAXROWCOLOR/tot_height;
26
27     pipemm_color_W = MAXCOLUMNCOLOR/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    MAXROWDEPTH = depth_size(1);
6
7    MAXCOLUMNDEPTH = depth_size(2);
8
9    for row = 1:MAXROWDEPTH
10       for col = 1:MAXCOLUMNDEPTH
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     MAXROW = matrix_size(1);
8
9     MAXCOLUMN = matrix_size(2);
10
11     row = 1;
12     col = 1;
13     %thicken the edge
14     for i = (1+z):(MAXROW-z)
15         for j = (1+z) : (MAXCOLUMN-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 ~= MAXROW)
25         if col == MAXCOLUMN
26             col = 1;
27             row = row + 1;

```

```

28
29     elseif (img(row, col, 1) == 0) && (img(row, col, 2) == 0) && (
30         img(row, col, 3) == 255)
31         col = 1;
32         row = row + 1;
33
34     else
35         img(row, col, 1) = 255;
36         img(row, col, 2) = 255;
37         img(row, col, 3) = 255;
38         col = col + 1;
39     end
40 %go from right to left
41 row = MAXROW;
42 col = MAXCOLUMN;
43 while (row ~= 1)
44     if col == 1
45         col = MAXCOLUMN;
46         row = row - 1;
47
48     elseif (img(row, col, 1) == 0) && (img(row, col, 2) == 0) && (img
49         (row, col, 3) == 255)
50         col = MAXCOLUMN;
51         row = row - 1;
52
53     else
54         img(row, col, 1) = 255;
55         img(row, col, 2) = 255;
56         img(row, col, 3) = 255;
57         col = col - 1;
58     end
59 %go from top to bottom
60 row = 1;
61 col = 1;
62 while (col ~= MAXCOLUMN)
63     if row == MAXROW
64         row = 1;
65         col = col + 1;
66
67     elseif (img(row, col, 1) == 0) && (img(row, col, 2) == 0) && (img
68         (row, col, 3) == 255)
69         row = 1;
70         col = col + 1;
71
72     else
73         img(row, col, 1) = 255;
74         img(row, col, 2) = 255;
75         img(row, col, 3) = 255;
76         row = row + 1;
77     end
78 %go from bottom to top
79 row = MAXROW;
80 col = MAXCOLUMN;

```

```

81     while (col ~= 1)
82         if row == 1
83             row = MAXROW;
84             col = col - 1;
85
86         elseif (img(row, col, 1) == 0) && (img(row, col, 2) == 0) && (img
87             (row, col, 3) == 255)
88             row = MAXROW;
89             col = col - 1;
90
91         else
92             img(row, col, 1) = 255;
93             img(row, col, 2) = 255;
94             img(row, col, 3) = 255;
95             row = row - 1;
96         end
97     %add in the white edge
98     for i = 1: MAXROW
99         for j = 1 : MAX_COLUMN
100             if (img(i, j, 1) == 0) && (img(i, j, 2) == 0) && (img(i, j,
101                 3) == 255)
102                 img(i, j, 1) = 255;
103                 img(i, j, 2) = 255;
104                 img(i, j, 3) = 255;
105             end
106         end
107     end
108
109
110     usefull_matrix = img;
111
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) *
4         0.1140;
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
3         4; 2 4 5 4 2;];
4     gaussian_blurred = conv2(img, gaussian);
5 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
3     kernel
4     edge=conv2(img, klaplace);                     % convolve test img
5     with
6 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     MAXROW = matrix_size(1);
5     MAXCOLUMN = matrix_size(2);
6     THICKNESS = 1; % 3
7
8     thresholded_img = zeros(MAXROW,MAXCOLUMN,1);
9     for row=1:MAXROW
10         for col=1:MAXCOLUMN
11             if img(row, col) > THRESHOLD_VALUE
12                 value = 1;
13                 for i=1:THICKNESS
14                     % Create thicker edges (edges of THICKNESS pixels
15                     thick)
16                     if (col - i) > 0
17                         thresholded_img(row, col-i) = 1;
18                     end
19                     if (col + i) <= MAXCOLUMN
20                         thresholded_img(row, col+i) = 1;
21                     end
22                     if (row - i) > 0
23                         thresholded_img(row -i, col) = 1;
24                     end
25                     if (row + i) <+ MAXROW
26                         thresholded_img(row +i, col) = 1;
27                     end
28                 end
29             end
30         else
31             value = 0;
32         end
33         thresholded_img(row, col) = value;
34     end
35 end
36 end
37 end
38 end

```

## 4.5 Group the edges

```
1 function [result , nb_of_groups] = group(img, SAME_PIXEL_SEARCH_GRID_SIZE,
    GROUP_SEARCH_GRID_SIZE, MIN_NB_SURROUNDING_PIXELS)
2     % Goal, group pixels.
3     % First loop from left to right to find an object
4     % Check if it's connected
5     % Number connected pixels in the second dimension
6     WHITE = 1;
7     BLACK = 0;
8     matrix_size = size(img);
9     MAXROW = matrix_size(1);
10    MAXCOLUMN = matrix_size(2);
11
12    groups = 0;
13
14    result = zeros(MAXROW,MAXCOLUMN,2); % Dimension 2 is for the group
    number.
15    for row=1:MAXROW
16        for col=1:MAXCOLUMN
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
42    nb_of_groups = groups;
43 end
```

## 4.6 Regroup the edges

```
1 function [result , nb_groups] = regroup(grouped_img , nb_of_groups ,
    MIN_ROW_LINES_BETWEEN_GROUPS)
```

```

2      % Loop from (right)top to (left)bottom
3      % Check if there are connecting groups.
4
5      matrix_size = size(grouped_img);
6      MAXROW = matrix_size(1);
7      MAXCOLUMN = matrix_size(2);
8
9      nb_groups = nb_of_groups;
10     for col_i=1:MAXCOLUMN
11         for row=1:MAXROW
12             col = MAXCOLUMN - col_i+1;
13             group_nb = grouped_img(row, col, 2);
14             if group_nb ~= 0
15                 for row_i=1:MIN_ROW_LINES_BETWEEN_GROUPS
16                     if is_valid_position(MAXROW, MAXCOLUMN, row + row_i
17                                     , col) == 1 && grouped_img(row + row_i, col, 2) ~=
18                         0 && grouped_img(row+row_i, col, 2) ~= group_nb
19                         % Found a different group in the next 5 pixels
20                         % below this one
21                         % Replace next group with previous group number
22                         grouped_img = group_replace(grouped_img,
23                                     grouped_img(row+row_i, col, 2), group_nb);
24                         nb_groups = nb_groups - 1;
25                         break;
26                     end
27                 end
28             end
29         end
30     end
31 end
32
33 result = grouped_img;
34 end

```

#### 4.7 Find the corner points

```

1 function result = find_corner_points(img, nb_groups)
2     % Loop through grouped image
3     % find MIN_ROW & MIN_COL and MAX_ROW & MAX_COL
4     matrix_size = size(img);
5     MAXROW = matrix_size(1);
6     MAXCOLUMN = matrix_size(2);
7
8     GROUP_MAXROW = zeros(1, nb_groups);
9     GROUP_MAXCOL = zeros(1, nb_groups);
10    GROUP_MINROW = zeros(1, nb_groups);
11    GROUP_MINCOL = zeros(1, nb_groups);
12
13    for row=1:MAXROW
14        for col=1:MAXCOLUMN
15            group_nb = img(row, col, 2);
16            if group_nb ~= 0
17                % Group found (==0 means nothing is set)
18                if GROUP_MAXROW(1, group_nb) == 0 || GROUP_MAXROW(1,
19                    group_nb) < row
20                    GROUP_MAXROW(1, group_nb) = row;
21                end
22            end
23        end
24    end
25
26    result = img;
27 end

```

```

21
22         if GROUP_MAX_COL(1,group_nb) == 0 || GROUP_MAX_COL(1,
23             group_nb) < col
24             GROUP_MAX_COL(1,group_nb) = col;
25         end
26         if GROUP_MIN_ROW(1,group_nb) == 0 || GROUP_MIN_ROW(1,
27             group_nb) > row
28             GROUP_MIN_ROW(1,group_nb) = row;
29         end
30         if GROUP_MIN_COL(1,group_nb) == 0 || GROUP_MIN_COL(1,
31             group_nb) > col
32             GROUP_MIN_COL(1,group_nb) = col;
33         end
34     end
35     result = [GROUP_MIN_ROW; GROUP_MIN_COL; GROUP_MAX_ROW;
36             GROUP_MAX_COL];
37 end
38 end

```

## 4.8 Remove objects within objects

### 4.8.1 Remove box edge

```

1 function [result , new_nb_of_groups] = remove_box_edge(corner_points ,
    nb_of_groups)
2     mat_size = size(corner_points);
3     groups = mat_size(2);
4     surfaces = zeros(groups); % Every column is a group, the value is the
    distance
5
6     for i=1:groups
7
8         min_row = corner_points(1,i);
9         min_col = corner_points(2,i) ;
10        max_row = corner_points(3,i);
11        max_col = corner_points(4,i);
12
13        surfaces(i) = (max_row - min_row) * (max_col - min_col);
14    end
15
16    %Now find biggest surface
17    [max_value , max_col] = max(surfaces);
18    for i=1:4
19        % Set the coordinates of the outer points to 0
20        corner_points(i , max_col) = 0;
21    end
22
23    result = corner_points;
24    new_nb_of_groups = nb_of_groups-1;
25 end

```

### 4.8.2 Remove corner points within corner points

```

1 function [updated_corner_points , nb_of_groups] =
    remove_corner_points_within_corner_points(corner_points , nb_groups)
2     mat_size = size(corner_points);
3     groups = mat_size(2); % This is the original number_of_groups
4     nb_of_groups = nb_groups; % This is the number_of_groups after
        regroup
5     updated_corner_points = corner_points;
6
7     for first=1:groups
8         % 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                    % Remove 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     MAXROW = matrix_size(1);
8     MAXCOLUMN = 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(MAXROW, MAXCOLUMN, min_row+e, col)
20                 = 1
21                 img(min_row+e, col, 1) = 255;
22                 img(min_row+e, col, 2) = 1;
23                 img(min_row+e, col, 3) = 1;
24             end
25             if is_valid_position(MAXROW, MAXCOLUMN, max_row-e, col)
26                 = 1
27                 img(max_row-e, col, 1) = 255;
28                 img(max_row-e, col, 2) = 1;
29                 img(max_row-e, col, 3) = 1;
30             end
31         end
32     end
33     % Vertical lines
34     for row=min_row:max_row
35         for e=0:THICKNESS
36             if is_valid_position(MAXROW, MAXCOLUMN, row, min_col +
37                 e) == 1
38                 img(row, min_col+e, 1) = 255;
39                 img(row, min_col+e, 2) = 1;
40                 img(row, min_col+e, 3) = 1;
41             end
42             if is_valid_position(MAXROW, MAXCOLUMN, row, max_col -
43                 e) == 1
44                 img(row, max_col-e, 1) = 255;
45                 img(row, max_col-e, 2) = 1;
46                 img(row, max_col-e, 3) = 1;
47             end
48         end
49     end

```

## 5 Implementation: packaging code

### 5.1 Gathering objects

#### 5.1.1 Get objects

```

1 function objects = get_objects(updated_corner_points, original_img,
2     regrouped)
3     % Returns a list of all objects in the given image counted by the
4     % counting system.
5     % Each object is placed in the smallest possible package.
6     % The list is sorted from largest to smallest object.
7
8     % Initializing variables
9     mat_size = size(updated_corner_points);
10    groups = mat_size(2);

```

```

10     unsorted_objects = {};
11     last_added_img = 1;
12
13     % Gathering every group.
14     for groupnb=1:groups
15
16         % Updated_cornern_points can contain zeros as corner points,
17         % these
18         % aren't valid.
19         if updated_corner_points(1,groupnb) ~= 0
20
21             % Getting the object cut out of the full image.
22             [objAlone, min_row_i, min_col_i, max_row_i, max_col_i,] =
23                 single_object(original_img, updated_corner_points, groupnb)
24             ;
25             obj_points = [min_row_i; min_col_i; max_row_i; max_col_i];
26
27             % Making the whole image white except the object itself.
28             objAlone = object_highlighter(objAlone, obj_points, regrouped
29                 , groupnb);
30             % Fitting the object in the smallest possible package
31             img = boundaryBoxedImgRotator(objAlone);
32
33             % Adding the object to the list of objects
34             unsorted_objects{last_added_img} = img;
35             last_added_img = last_added_img + 1;
36         end
37     end
38
39     % Sort the list of objects
40     objects = imgs_InsertionSort(unsorted_objects);
41
42 end

```

### 5.1.2 Object highlighter

```

1 function newImg = object_highlighter(img, obj_points, regrouped, groupnb)
2     % Makes everything besides the object with a given group number white
3     .
4
5     % Initializing variables
6     matrix_size = size(img);
7     MAXROW = matrix_size(1);
8     MAXCOL = matrix_size(2);
9     newImg = ones(MAXROW,MAXCOL);
10
11     % Iterate over every row
12     for row = 1:MAXROW
13         first_pixel = -1;
14         last_pixel = -1;
15
16         % First iteration over the row
17         % Mark for each row the first and the last pixel of the object,
18         % as
19         % seen by the regrouping algortihm
20         for col = 1:MAXCOL
21             if regrouped(row + obj_points(1)-1,col + obj_points(2)-1,2)

```

```

20         == groupnb
21         last_pixel = col-2;
22         if first_pixel == -1
23             first_pixel = col+2;
24         end
25     end
26
27     % Second iteration over the row
28     % If the pixel falls inbetween the two marked points, it belongs
29     % to
30     % the object and is given the same value as the original image.
31     % If
32     % the pixel is outside those points or there are no marked points
33     % at all it is coloured white (255).
34     for col = 1:MAXCOL
35
36         if first_pixel == -1
37             newImg(row,col) = 255;
38         end
39
40         if col >= first_pixel && col <= last_pixel
41             newImg(row,col) = img(row,col);
42         else
43             newImg(row,col) = 255;
44         end
45     end
end

```

### 5.1.3 Insertion sort

```

1 function sorted_imgs = imgs.InsertionSort(listed_imgs)
2     % Insurction sort based fucntion to sort a list of images from biggest
3     % to smallest surface area.
4
5     listSize = size(listed_imgs);
6
7     % Iterate over every image
8     for i=1:listSize(2)
9
10        % Calculating the surface size of images i
11        img_i = listed_imgs{i};
12        img_i_sizes = size(img_i);
13        img_i_surface_size = img_i_sizes(1)*img_i_sizes(2);
14
15        % Run through the all images before i
16        for j=1:i
17
18            % Calculating the surface size of image j
19            img_j = listed_imgs{j};
20            img_j_sizes = size(img_j);
21            img_j_surface_size = img_j_sizes(1)*img_j_sizes(2);
22
23            % Swap the two if the image i is bigger than image j
24            if img_i_surface_size > img_j_surface_size
25                temp = listed_imgs{j};

```



```

26         listed_imgs{j} = listed_imgs{i};
27         listed_imgs{i} = temp;
28         img_i_surface_size = img_j_surface_size;
29     end
30
31 end
32 end
33
34 sorted_imgs = listed_imgs;
35 end

```

#### 5.1.4 Single object

```

1 function [objAlone, min_row_group, min_col_group, max_row_group,
    max_col_group] = single_object(img, corner_points, groupnb)
2     % Returns a part of the image which fully contains the group
    associated with the given group number.
3
4     min_row_group = corner_points(1,groupnb);
5     min_col_group = corner_points(2,groupnb);
6     max_row_group = corner_points(3,groupnb);
7     max_col_group = corner_points(4,groupnb);
8     % Crop around the group
9     objAlone = simon_crop(img, min_row_group, min_col_group, max_row_group,
        max_col_group);
10 end

```

## 5.2 fitting the objects

### 5.2.1 Boundary boxed image rotator

```

1 function rotated_objec = boundaryBoxedImgRotator(boxedObjec)
2     % Bisection method based algorithm to find the best fitting
3     % package/boundary box
4
5     % Initialize variables
6     lower_rad = 0;
7     upper_rad = pi*3/8;
8     lower_dim = packaged_objec(boxedObjec, lower_rad, 1);
9     upper_dim = packaged_objec(boxedObjec, upper_rad, 1);
10
11     i = 0;
12     while i < 10
13         % Calculating pivot values
14         pivot_rad = (upper_rad-lower_rad)/2+lower_rad;
15         pivot_dim = packaged_objec(boxedObjec, pivot_rad, 1);
16
17         % Comparing the lower and upper points and changing their values
18         % accordingly.
19         if lower_dim <= upper_dim
20             upper_rad = pivot_rad;
21             upper_dim = pivot_dim;
22         else
23             lower_rad = pivot_rad;
24             lower_dim = pivot_dim;
25         end
26         i=i+1;
27     end

```

```

28     % Returning the the rotated image for the elevnth pivot.
29     rotated_objec = packaged_objec(boxedObjec, (abs((upper_rad -
        lower_rad)/2) + min(upper_rad, lower_rad)), 0);
30
31 end

```

### 5.2.2 Packaged object

```

1 function boxedDim = packaged_objec(boxedObjec, angle, flag)
2     % Function which will return either the dimensions of the new
        boundary
3     % box after rotating the image (if flag == 1) or returns the whole
4     % image after it has been cropped to the new boundary box of the
5     % rotated object (if flag == 0)
6
7     % Initializing constants
8     THRESHOLD.VALUE = 2;
9     MIN_ROW_LINES.BETWEEN.GROUPS = 10;
10    SAME_PIXELS.SEARCH_GRID.SIZE = 10;
11    GROUP.SEARCH_GRID.SIZE = 15;
12    SURROUDING.PERCENTAGE = 10;
13    MIN_NB.SURROUNDING.PIXELS = floor((SAME_PIXELS.SEARCH_GRID.SIZE * 2)
        ^2 * SURROUDING.PERCENTAGE/100);
14
15    % Rotate the image
16    rotatedObj = rotator(boxedObjec, angle);
17
18    % Finding the object in the rotated image.
19    rotatedObj = gaussian_blur(mean_blur(rotatedObj));
20    first_edge_detect = edge_detect(rotatedObj);
21    without_noise_removal = threshold_edge(remove_boundary(
        first_edge_detect, 15), THRESHOLD.VALUE);
22    [grouped, nb_of_groups] = group(~without_noise_removal,
        SAME_PIXELS.SEARCH_GRID.SIZE, GROUP.SEARCH_GRID.SIZE,
        MIN_NB.SURROUNDING.PIXELS);
23    [regrouped, nb_of_groups2] = regroup(grouped, nb_of_groups,
        MIN_ROW_LINES.BETWEEN.GROUPS);
24    corner_points = find_corner_points(regrouped, nb_of_groups);
25    [updated_corner_points, nb_of_groups3] =
        remove_corner_points_within_corner_points(corner_points,
        nb_of_groups2);
26
27    % Checking whether one object is found
28    if nb_of_groups3 == 1
29
30        cropped_rotated_boxedObjec = generic_crop(rotatedObj,
            updated_corner_points);
31        % imshow(cropped_rotated_boxedObjec, []);
32
33
34        if flag == 1
35            % Return the new boundary box dimensions.
36            matSize = size(cropped_rotated_boxedObjec);
37            boxedDim = matSize(1)*matSize(2);
38        elseif flag == 0
39            % return the whole image.
40            boxedDim = cropped_rotated_boxedObjec;

```

```

41         end
42
43     % If there are multiple objects , no rotated image is returned
44     else
45         disp("not one object");
46         boxedDim = 0;
47     end
48 end

```

### 5.2.3 Rotator

```

1 function rotated = rotator(img, rads)
2     % Calculating the size of the padding matrix
3     [ROWS, COLS] = size(img);
4     diagonal = sqrt(ROWS^2 + COLS^2);
5     rowPad = ceil(diagonal - ROWS) + 2;
6     colPad = ceil(diagonal - COLS) + 2;
7
8     % Creating the padddding matrix and filling it whith the original
9     % image
10    % in the middle and everything else white.
11    padding_mat = ones(ROWS+rowPad, COLS+colPad)*255;
12    padding_mat(ceil(rowPad/2):(ceil(rowPad/2)+ROWS-1), ceil(colPad/2):(
13        ceil(colPad/2)+COLS-1)) = img;
14
15    % Calcularting the mid coordinates of the matrices.
16    padding_size = size(padding_mat);
17    midx=ceil((padding_size(2)+1)/2);
18    midy=ceil((padding_size(1)+1)/2);
19
20    % Creating the rotated image
21    rotated=ones(padding_size)*255;
22    rotSize = size(rotated);
23
24    % For each position in the rotated image get the value out of the
25    % padding matrix which corresponds with the position if it were
26    % rotated
27    % by the given angle.
28    for i=1:rotSize(1)
29        for j=1:rotSize(2)
30
31            x = (i-midx)*cos(rads)+(j-midy)*sin(rads);
32            x=round(x)+midx;
33            y=-(i-midx)*sin(rads)+(j-midy)*cos(rads);
34            y=round(y)+midy;
35
36            if x >= 1 && y >= 1 && x <= padding_size(2) && y <=
37                padding_size(1)
38                rotated(i,j)=padding_mat(x,y);
39            end
40        end
41    end
42 end

```

### 5.2.4 Generic crop

```

1 function img_crop = generic_crop(img, fourp)

```

```

2      % A function which crops the given image so that the edges are
3      % defined by the four point given in fourp.
4      mat_size = size(fourp);
5      groups = mat_size(2);
6
7      for i=1:groups
8          if fourp(1,i)~=0
9              MIN_ROW = fourp(1,i);
10             MIN_COL = fourp(2,i);
11             MAX_ROW = fourp(3,i);
12             MAX_COL = fourp(4,i);
13         end
14     end
15
16     img_crop = zeros(MAX_ROW-MIN_ROW+1,MAX_COL-MIN_COL+1,1);
17     for row = MIN_ROW:MAX_ROW
18         for col = MIN_COL:MAX_COL
19
20             img_crop(row - MIN_ROW + 1,col - MIN_COL + 1,1) = img(row
21                                     ,col);
22         end
23     end
24 end

```

## 5.3 total packaging

### 5.3.1 Smallest Package

```

1 function replacedObjects = smallest_package(objects)
2     % This function creates a possible packaging for all the objects
3     % given.
4     % This package has to be as small as possible, but isn't the optimal
5     % packaging. This is a greedy algorithm which fills the package from
6     % the biggest to smallest objects.
7
8     % The function returns an image of the package with each individual
9     % object's package outlined in black.
10
11     % OBJECTS HAS TO BE SORTED FROM BIGGEST TO SMALLEST BEFOREHAND
12
13     % The optimal package for the biggest object is the object itself.
14     replacedObjects = black_edger(objects{1});
15
16     % Iterate over every object except the biggest and make a new package
17     % of the old package and the new object
18     listSize = size(objects);
19     for i=2:listSize(2)
20
21         % Initialize variables
22         mat_size = size(replacedObjects);
23         object_size = size(objects{i});
24         extra_size_smallest = inf;
25         smallest_col = 0;
26         smallest_row = 0;
27         flag = 0;
28

```

```

29 % Iterate over every pixel
30 for row=1:mat_size(1)+1
31     for col=1:mat_size(2)+1
32
33         % Checking whether there is an object at this location.
34         if row ~= mat_size(1)+1 && col ~= mat_size(2)+1
35             % If there is an object, continue with the next pixel.
36             if replacedObjects(row,col) ~= -1
37                 continue
38             end
39         end
40
41         % Measuring the size of the package if it were appended
42         % with
43         % object at on this position.
44         vert_diff = object_size(1) + row - 1;
45         hor_diff = object_size(2) + col - 1;
46
47         if vert_diff < mat_size(1)
48             vert_diff = mat_size(1);
49         end
50         if hor_diff < mat_size(2)
51             hor_diff = mat_size(2);
52         end
53
54         extra_size = vert_diff * hor_diff;
55
56         % Measuring the same size as above, but the object is
57         % rotated
58         % 90 .
59         vert_diff_rot = object_size(2) + row - 1;
60         hor_diff_rot = object_size(1) + col - 1;
61
62         if vert_diff_rot < mat_size(1)
63             vert_diff_rot = mat_size(1);
64         end
65         if hor_diff_rot < mat_size(2)
66             hor_diff_rot = mat_size(2);
67         end
68
69         extra_size_rot = vert_diff_rot * hor_diff_rot;
70
71         % If the this position results in a smaller package than
72         % the
73         % previous one use this one as the best position.
74         if extra_size < extra_size_smallest
75             % Appending the object at this position may not result
76             % in overlap of objects.
77             bool = position_tester(replacedObjects, objects{i}, row
78                                     , col);
79             if bool == 1
80                 extra_size_smallest = extra_size;
81                 smallest_row = row;
82                 smallest_col = col;
83                 flag = 0;

```

```

81         end
82     end
83
84     % Same as above but for the rotated object.
85     if extra_size_rot < extra_size_smallest
86         % Appending the rotated object at this position may not
            result
87         % in overlap of objects.
88         bool = position_tester(replacedObjects, transpose(
            objects{i}), row, col);
89         if bool == 1
90             extra_size_smallest = extra_size_rot;
91             smallest_row = row;
92             smallest_col = col;
93             flag = 1;
94         end
95     end
96 end
97
98 % Append the object on the best position to the old package.
99 % If the optimal measurments are reached bij rotating the object
100 % transpose it.
101 if flag == 1
102     replacedObjects = package_appender(replacedObjects,
103         black_edger(transpose(objects{i})), smallest_row,
104         smallest_col);
105 else
106     replacedObjects = package_appender(replacedObjects,
107         black_edger(objects{i}), smallest_row, smallest_col);
108 end
109 imshow(replacedObjects, []);
110
111 end
112 end

```

### 5.3.2 Black Edged

```

1 function blackEdged = black_edger(img)
2     % Function which the outer one pixel in both dimensions of an image
3     % black.
4     % Used to see the edge of each individual package in the combined one
5     .
6
7     img_size = size(img);
8
9     for row = 1:img_size(1)
10        for col = 1:img_size(2)
11            if row == 1 || row == img_size(1) || col == 1 || col ==
                img_size(2)
12                img(row, col) = 1;
13            end
14        end
15    end

```

```

16     blackEdged = img;
17
18 end

```

### 5.3.3 Position tester

```

1 function result = position_tester(package, object, smallest_row,
   smallest_col)
2     % The result of this function is true if and only if appending the
3     % package with the given object on the given row and col doesn't
   result
4     % in overlap.
5
6     % Remember that every non-object pixel in package has a value of -1.
7
8     % Initialize variables
9     result = 1;
10    mat_size = size(package);
11    object_size = size(object);
12
13    % Determine the boundaries of iteration. If the object fully overlaps
14    % with the current package use the object dimensions + the position
   as
15    % the upper-boundary else use the package dimensions as the boundary.
16    if object_size(1) + smallest_row - 1 > mat_size(1)
17        biggest_row = mat_size(1);
18    else
19        biggest_row = object_size(1) + smallest_row - 1;
20    end
21
22    if object_size(2) + smallest_col - 1 > mat_size(2)
23        biggest_col = mat_size(2);
24    else
25        biggest_col = object_size(2) + smallest_col - 1;
26    end
27
28    % Run through the part of the package with which the object would
29    % overlap, if there is another object the result will be changed to 0
30    for row=smallest_row:biggest_row
31        for col=smallest_col:biggest_col
32            if package(row,col) ~= -1
33                result = 0;
34            end
35        end
36    end
37
38 end

```

### 5.3.4 Package appender

```

1 function new_package = package_appender(package, object, smallest_row,
   smallest_col)
2     % Appends the package with the given object on the given position.
3     % If the object dimensions exceed the dimensions of the package, a
   new
4     % one will be created which will fit both. Empty spaces in the package
5     % are denoted by a value of -1.
6     mat_size = size(package);

```

```

7      object_size = size(object);
8
9
10     % The old package can fit the object.
11     if mat_size(1) >= object_size(1) + smallest_row && mat_size(2) >=
        object_size(2) + smallest_col
12         % The package is appended with the object at the given position.
13         for row = smallest_row:(object_size(1)+smallest_row-1)
14             for col = smallest_col:(object_size(2)+smallest_col-1)
15                 package(row,col) = object(row - smallest_row + 1, col -
                    smallest_col + 1);
16             end
17         end
18         new_package = package;
19
20
21     % The old package can't fit the object horizontally.
22     elseif mat_size(1) >= object_size(1) + smallest_row && mat_size(2) <
        object_size(2) + smallest_col
23         % Create a new package which can fit the object.
24         new_package = ones(mat_size(1), object_size(2) + smallest_col)
            *-1;
25
26         % Fill the new package with the old one.
27         for row = 1:mat_size(1)
28             for col = 1:mat_size(2)
29                 new_package(row,col) = package(row,col);
30             end
31         end
32
33         % The package is appended with the object at the given position.
34         for row = smallest_row:(object_size(1)+smallest_row-1)
35             for col = smallest_col:(object_size(2) + smallest_col-1)
36                 new_package(row,col) = object(row - smallest_row+1, col -
                    smallest_col+1);
37             end
38         end
39
40
41     % The old package can't fit the object vertically.
42     elseif mat_size(1) < object_size(1) + smallest_row && mat_size(2) >=
        object_size(2) + smallest_col
43         % Create a new package which can fit the object.
44         new_package = ones(object_size(1) + smallest_row, mat_size(2))
            *-1;
45
46         % Fill the new package with the old one.
47         for row = 1:mat_size(1)
48             for col = 1:mat_size(2)
49                 new_package(row,col) = package(row,col);
50             end
51         end
52
53         % The package is appended with the object at the given position.
54         for row = smallest_row:(object_size(1) + smallest_row-1)
55             for col = smallest_col:(object_size(2)+smallest_col-1)

```



```

56         new_package(row,col) = object(row - smallest_row+1, col -
57             smallest_col+1);
58     end
59
60
61     % The old package can't fit the object both vertically and
62     % horizontally.
63     else
64         % Create a new package which can fit the object.
65         new_package = ones(object_size(1) + smallest_row, object_size(2)
66             + smallest_col)*-1;
67
68         % Fill the new package with the old one.
69         for row = 1:mat_size(1)
70             for col = 1:mat_size(2)
71                 new_package(row,col) = package(row,col);
72             end
73         end
74
75         % The package is appended with the object at the given position.
76         for row = smallest_row:(object_size(1) + smallest_row-1)
77             for col = smallest_col:(object_size(2) + smallest_col-1)
78                 new_package(row,col) = object(row - smallest_row + 1, col
79                     - smallest_col + 1);
80             end
81         end
82     end

```