

```

1 function res = main(path, image_type, start_offset, time_step)
2     if nargin < 2
3         image_type = 'jpg';
4     end
5     if nargin < 3
6         start_offset = 0;
7     end
8     if nargin < 4
9         time_step = 0.33;
10    end
11
12    files = dir(sprintf('%s/*.%s', path, image_type));
13    filenames = {files.name};
14    [~, num_files] = size(filenames);
15
16    [m n ~] = size(imread(sprintf('%s/%s', path, filenames{1})));
17    track_mask = zeros(m, n);
18
19    rc = zeros(num_files - start_offset, 2);
20    gc = zeros(num_files - start_offset, 2);
21    bc = zeros(num_files - start_offset, 2);
22
23    reds = cell(num_files - start_offset, 1);
24    greens = cell(num_files - start_offset, 1);
25    blues = cell(num_files - start_offset, 1);
26
27    for i = 1 + start_offset : num_files
28        image = imread(sprintf('%s/%s', path, filenames{i}));
29        [directions, centroids] = analyse_image(image);
30
31        r = uint16(centroids(1,:));
32        if r(1) > 0 && r(2) > 0
33            track_mask = overlay_cross(track_mask, 1, r(1), r(2));
34            rc(i,:) = [r(1), r(2)];
35        end
36
37        g = uint16(centroids(2,:));
38        if g(1) > 0 && g(2) > 0
39            track_mask = overlay_cross(track_mask, 2, g(1), g(2));
40            gc(i,:) = [g(1), g(2)];
41        end
42
43        b = uint16(centroids(3,:));
44        if b(1) > 0 && b(2) > 0
45            track_mask = overlay_cross(track_mask, 3, b(1), b(2));
46            bc(i,:) = [b(1), b(2)];
47        end
48
49        reds{i} = image(:,:,1);
50        greens{i} = image(:,:,2);
51        blues{i} = image(:,:,3);
52        imshow(overlay_mask(image, directions));
53
54        pause(time_step);
55    end
56
57    rc = rc(any(rc,2),:);
58    gc = gc(any(gc,2),:);
59    bc = bc(any(bc,2),:);
60
61    red_median = median(cat(3, reds{:}), 3);
62    green_median = median(cat(3, greens{:}), 3);
63    blue_median = median(cat(3, blues{:}), 3);
64

```

```

65     bg = cat(3, red_median, green_median, blue_median);
66
67     bg = overlay_polygon(bg, rc, [255, 255, 255]);
68     bg = overlay_polygon(bg, gc, [255, 255, 255]);
69     bg = overlay_polygon(bg, bc, [255, 255, 255]);
70     bg = overlay_mask(bg, track_mask);
71     imshow(bg)
72 end
73
74
75 function [pretty_mask, varargout] = analyse_image(image)
76     [num_rows, num_cols, num_channels] = size(image);
77     blob_mask = mask_colors(image);
78     [convex_mask, ~] = mask_convex_regions(image, blob_mask);
79     [convex_mask] = demask_triangles(image, convex_mask, false);
80     [convex_mask, ~] = mask_convex_regions(image, convex_mask);
81     [~, centroids, ~, triangle_centroids] = ...
82         demask_triangles(image, convex_mask, true);
83     pretty_mask = overlay_rays(zeros(num_rows, num_cols, num_channels), ...
84         centroids, triangle_centroids, 99, ...
85         'Color', [1 0 0; 0 1 0; 0 0 1]);
86     varargout{1} = centroids;
87     varargout{2} = triangle_centroids;
88     varargout{3} = convex_mask;
89 end
90
91
92 % this function can used to find both - triangles and masks without
93 % triangles. To find triangles it must be run with
94 % get_triangle_centroids=true. Also application of the function int the
95 % following fashion gives better results.
96 %     [convex_mask, ~] = mask_convex_regions(image, blob_mask);
97 %     [convex_mask] = demask_triangles(image, convex_mask, false);
98 %     [convex_mask, ~] = mask_convex_regions(image, convex_mask);
99 function [mask, varargout] = demask_triangles(image, mask, ...
100         get_triangle_centroids)
101     [num_rows, num_cols, num_channels] = size(image);
102     centroids = zeros(num_channels, 2);
103     triangle_centroids = zeros(num_channels, 2);
104     trinagle_mask = mask;
105     for c = 1 : num_channels
106         channel = image(:,:,c);
107         channel_mask = mask(:,:,c);
108         channel_trinagle_mask = mask(:,:,c);
109         if ~any(channel_mask)
110             continue;
111         end
112         rgb_values = zeros(sum(channel_mask(:)), 1);
113         idx = 1;
114         for nr = 1 : num_rows
115             for nc = 1 : num_cols
116                 if channel_mask(nr, nc) == 0
117                     continue;
118                 end
119                 rgb_values(idx) = channel(nr, nc);
120                 idx = idx + 1;
121             end
122         end
123         mean_rgb = mean(rgb_values(:));
124         channel_mask(channel < mean_rgb) = 0;
125         mask(:,:,c) = channel_mask;
126         props = regionprops(channel_mask, 'Centroid');
127         centroid = props.Centroid;
128         centroids(c,:) = [centroid(2), centroid(1)];
129         if get_triangle_centroids

```

```

130         channel_trinagle_mask(channel > mean_rgb) = 0;
131         channel_trinagle_mask=filter_mask(channel_trinagle_mask);
132         trinagle_mask(:,:,c) = channel_trinagle_mask;
133         props = regionprops(channel_trinagle_mask, 'Centroid');
134         centroid = props.Centroid;
135         triangle_centroids(c,:) = [centroid(2), centroid(1)];
136     end
137 end
138 varargout{1} = centroids;
139 varargout{2} = trinagle_mask;
140 varargout{3} = triangle_centroids;
141 end
142
143
144 function [color_mask, varargout] = mask_convex_regions(image, mask)
145     [num_rows, num_cols, num_channels] = size(image);
146     color_mask = zeros(num_rows, num_cols, num_channels);
147     convex_centroids = zeros(num_channels, 2);
148     for c = 1 : num_channels
149         channel = mask(:,:,c);
150         if ~any(channel(:))
151             continue;
152         end
153         props = regionprops(channel, 'Centroid', 'ConvexImage', 'BoundingBox');
154         convex_props = regionprops(props.ConvexImage, 'Centroid');
155         convex_centroid = convex_props.Centroid;
156         convex_centroid = [convex_centroid(2) + props.BoundingBox(2), ...
157             convex_centroid(1) + props.BoundingBox(1)];
158         convex_centroids(c,:) = convex_centroid;
159         convex_image = props.ConvexImage;
160         [num_rows_convex, num_cols_convex] = size(convex_image);
161         for row = 1 : num_rows_convex
162             for col = 1 : num_cols_convex
163                 if convex_image(row, col) == 1
164                     newrow = round(row + props.BoundingBox(2));
165                     newcol = round(col + props.BoundingBox(1));
166                     color_mask(newrow, newcol, c) = 1;
167                 end
168             end
169         end
170     end
171     varargout{1} = convex_centroids;
172 end
173
174
175 function image_mask = mask_colors(image)
176     [num_rows, num_cols, ~] = size(image);
177     num_pixels = num_rows * num_cols;
178     image_mask = zeros(num_pixels, 3);
179     rgb = double(reshape(image, num_pixels, 3));
180     rgbN = double(reshape(normalise_rgb(image, 'approximate'), num_pixels, 3));
181     rN_sdev = std(rgbN(:,1));
182     gN_sdev = std(rgbN(:,2));
183     bN_sdev = std(rgbN(:,3));
184     rN_mean = mean(rgbN(:,1));
185     gN_mean = mean(rgbN(:,2));
186     bN_mean = mean(rgbN(:,3));
187     hsv = reshape(rgb2hsv(image), num_pixels, 3);
188     for c = 1 : num_pixels
189         rN = rgbN(c,1);
190         gN = rgbN(c,2);
191         bN = rgbN(c,3);
192         hue = hsv(c,1) * 360;
193         % current pixel is red
194         if (hue >= 330 || hue <= 30) && ...

```

```

195         (normal_prob(rN, rN_mean, rN_sdev) < 0.001)
196         image_mask(c,1) = 1;
197     % current pixel is green
198     elseif (hue >= 80 && hue < 180) && ...
199         (normal_prob(gN, gN_mean, gN_sdev) < 0.007)
200         image_mask(c,2) = 1;
201     % current pixel is blue
202     elseif (hue >= 150 && hue <= 270) && ...
203         (normal_prob(bN, bN_mean, bN_sdev) < 0.0000085)
204         image_mask(c,3) = 1;
205     end
206 end
207 image_mask = reshape(image_mask, num_rows, num_cols, 3);
208 image_mask = remove_noise(image_mask);
209 image_mask = remove_outliers(image_mask);
210 image_mask = enforce_similar_channel_areas(image_mask);
211 end
212
213 function x = normal_prob(val, mu, sigma)
214     x = 1.0 / (sigma * sqrt(2 * pi)) * exp(-(val - mu) ^ 2 / (2 * sigma ^ 2));
215 end
216
217
218 function image = remove_noise(image)
219     [~, ~, num_channels] = size(image);
220     for c = 1 : num_channels
221         channel = image(:,:,c);
222         channel = bwmorph(channel, 'majority', Inf);
223         channel = bwmorph(channel, 'bridge', Inf);
224         image(:,:,c) = channel;
225     end
226 end
227
228
229 % finds the connected components in each channel of |image| and removes those
230 % that are far away from the centroid of the pixels in that channel
231 % here, 'far away' means more distant than |distance_proportion_threshold| times
232 % the average distance of each connected component to the channel centroid
233 % this removes big areas of noise such as the big green blob inside of the black
234 % arrow of the red robot in data/1/00000006.jpg
235 function image = remove_outliers(image, distance_proportion_threshold)
236     if nargin < 2
237         distance_proportion_threshold = 0.5;
238     end
239     [~, ~, num_channels] = size(image);
240     for c = 1 : num_channels
241         channel = image(:,:,c);
242         if ~any(channel(:))
243             continue;
244         end
245         channel_properties = regionprops(channel, 'Centroid');
246         channel_centroid = channel_properties.Centroid;
247         regions = bwconncomp(channel);
248         regions_properties = regionprops(regions, 'Centroid', 'PixelIdxList');
249         regions_centroids = {regions_properties.Centroid};
250         distances = cellfun(@(x) norm(x - channel_centroid), regions_centroids);
251         mean_distance = mean(distances);
252         for d = 1 : length(distances)
253             if distances(d) > mean_distance * distance_proportion_threshold
254                 idx = regions_properties(d).PixelIdxList;
255                 channel(idx) = 0;
256             end
257         end
258     end
259     image(:,:,c) = channel;

```

```

260     end
261 end
262
263
264 % we know that the robots are all about the same size - we can thus remove any
265 % channels in the mask that have a much smaller area than the other channels
266 % this catches some problems like the shadow of the blue robot in
267 % data/1/00000095.jpg being detected as a red blob
268 function image = enforce_similar_channel_areas(image, area_proportion_threshold)
269     if nargin < 2
270         area_proportion_threshold = 0.5;
271     end
272     [~, ~, num_channels] = size(image);
273     areas = zeros(num_channels, 1);
274     for c = 1 : num_channels
275         channel = image(:,:,c);
276         if ~any(channel(:))
277             continue;
278         end
279         region_props = regionprops(channel, 'Area');
280         areas(c) = region_props.Area;
281     end
282     max_area = max(areas(areas > 0));
283     for c = 1 : num_channels
284         area = areas(c);
285         if area == 0
286             continue;
287         end
288         if (area < max_area * area_proportion_threshold) || ...
289             (area > max_area / area_proportion_threshold)
290             image(:,:,c) = image(:,:,c) * 0;
291         end
292     end
293 end
294
295
296 % this filters out all connected regions but the biggest one
297 function mask = filter_mask(mask)
298     [x, y, num_channels] = size(mask);
299     for c = 1 : num_channels
300         channel = zeros(x, y);
301         channel = reshape(channel, x * y, 1);
302         blob_info = bwconncomp(mask(:,:,c));
303         blob_list = blob_info.PixelIdxList;
304         [nrows, ncols] = cellfun(@size, blob_list);
305         largest_blob_pixels = blob_list{find(nrows == max(nrows))};
306         for i = 1 : max(nrows)
307             channel(largest_blob_pixels(i)) = 1;
308         end
309         channel = reshape(channel, x, y);
310         mask(:,:,c) = channel;
311     end
312 end
313
314
315 % normalises the values of the red, green, and blue channels of |image| in order
316 % to eliminate illumination differences in the image
317 % formula used: {r, g, b} = {r, g, b} / sqrt(r^2 + g^2 + b^2)
318 % if 'approximate' is passed as an additional parameter, instead use
319 % {r, g, b} = {r, g, b} / (r + g + b)
320 % this is approximately two times faster than the exact normalisation
321 function normalised_image = normalise_rgb(image, varargin)
322     approximate = ~isempty(find(strcmpi(varargin, 'approximate')));
323     red = double(image(:,:,1));
324     green = double(image(:,:,2));

```

```

325 blue = double(image(:,:,3));
326 if approximate
327     euclid_rgb = red(:,:,) + green(:,:,) + blue(:,:,);
328 else
329     euclid_rgb = sqrt(red(:,:,).^2 + green(:,:,).^2 + blue(:,:,).^2);
330 end
331 red_norm = round(red(:,:,) ./ euclid_rgb .* 255);
332 green_norm = round(green(:,:,) ./ euclid_rgb .* 255);
333 blue_norm = round(blue(:,:,) ./ euclid_rgb .* 255);
334 % some pixels are absolute black (r = g = b = 0) which causes division by
335 % zero errors during normalisation and NaN values in the normalised channels
336 % need to filter these values out
337 red_norm(isnan(red_norm)) = 0;
338 green_norm(isnan(green_norm)) = 0;
339 blue_norm(isnan(blue_norm)) = 0;
340 red_norm = uint8(red_norm);
341 green_norm = uint8(green_norm);
342 blue_norm = uint8(blue_norm);
343 normalised_image = cat(3, red_norm, green_norm, blue_norm);
344 end
345
346 % returns image with a cross centered on pixel (|x|, |y|) drawn in |channel|
347 % [0, 0] is the top left corner of the image
348 function image = overlay_cross(image, channel, x, y)
349     [h w ~] = size(image);
350
351     if channel == 1
352         other1 = 2;
353         other2 = 3;
354     end
355     if channel == 2
356         other1 = 1;
357         other2 = 3;
358     end
359     if channel == 3
360         other1 = 1;
361         other2 = 2;
362     end
363
364     if y + 1 < h
365         image(x, y + 1, channel) = 1;
366         image(x, y + 1, other1) = 0;
367         image(x, y + 1, other2) = 0;
368     end
369
370     if y - 1 > 0
371         image(x, y - 1, channel) = 1;
372         image(x, y - 1, other1) = 0;
373         image(x, y - 1, other2) = 0;
374     end
375
376     if x + 1 < w
377         image(x + 1, y, channel) = 1;
378         image(x + 1, y, other1) = 0;
379         image(x + 1, y, other2) = 0;
380     end
381
382     if x - 1 > 0
383         image(x - 1, y, channel) = 1;
384         image(x - 1, y, other1) = 0;
385         image(x - 1, y, other2) = 0;
386     end
387
388     image(x, y, channel) = 1;

```

```

390     image(x, y, other1) = 0;
391     image(x, y, other2) = 0;
392
393 end
394
395
396 % returns the result of putting |mask| onto |image|
397 % for each pixel that is set in some channel of |mask|, saturates the pixel in
398 % the equivalent channel of |image|
399 function image = overlay_mask(image, mask, varargin)
400     % parse options
401     argc = size(varargin, 2);
402     c = 1;
403     while c <= argc
404         arg = varargin{c};
405         if strcmpi(arg, 'GrayScale')
406             grayscale = 1;
407         elseif strcmpi(arg, 'Saturation')
408             if c + 1 > argc
409                 error('Saturation option should be followed by a double');
410             end
411             saturation = varargin{c + 1};
412             c = c + 1;
413         elseif strcmpi(arg, 'Lightness')
414             if c + 1 > argc
415                 error('Lightness option should be followed by a double');
416             end
417             lightness = varargin{c + 1};
418             c = c + 1;
419         end
420         c = c + 1;
421     end
422
423     % modify background image
424     if exist('grayscale', 'var')
425         gray_image = rgb2gray(image);
426         image = cat(3, gray_image, gray_image, gray_image);
427     end
428     if exist('saturation', 'var')
429         hsv_image = rgb2hsv(image);
430         hsv_image(:,:,2) = hsv_image(:,:,2) * saturation;
431         image = hsv2rgb(hsv_image);
432     end
433     if exist('lightness', 'var')
434         hsv_image = rgb2hsv(image);
435         hsv_image(:,:,3) = hsv_image(:,:,3) * lightness;
436         image = hsv2rgb(hsv_image);
437     end
438
439     % lay mask onto background image
440     num_channels = size(image, 3);
441     channels = 1 : num_channels;
442     for c = 1 : num_channels
443         channel = image(:,:,c);
444         mask_pixels = find(mask(:,:,c) == 1);
445         channel(mask_pixels) = 255;
446         image(:,:,c) = channel;
447         for d = setdiff(channels, c)
448             channel = image(:,:,d);
449             channel(mask_pixels) = 0;
450             image(:,:,d) = channel;
451         end
452     end
453 end
454

```

```

455
456 % returns image with lines drawn from the nth point in |points| to the n+1th
457 % [0, 0] is the top left corner of the image
458 function image = overlay_polygon(image, points, color)
459     if nargin < 3
460         color = [255 255 255];
461     end
462
463     points = round(points);
464
465     num_channels = size(image, 3);
466     for c = 1 : num_channels
467         channel = image(:,:,c);
468         channel = overlay_polygon_channel(channel, points, color(c));
469         image(:,:,c) = channel;
470     end
471 end
472
473
474 % Bresenham's line algorithm (simplified version)
475 % http://en.wikipedia.org/wiki/Bresenham's\_line\_algorithm#Simplification
476 function channel = overlay_polygon_channel(channel, points, color)
477     [xmax, ymax] = size(channel);
478     for c = 1 : length(points) - 1
479         start = points(c,:);
480         stop = points(c + 1,:);
481         x0 = start(1);
482         y0 = start(2);
483         x1 = stop(1);
484         y1 = stop(2);
485
486         dx = abs(x1 - x0);
487         dy = abs(y1 - y0);
488
489         if x0 < x1
490             sx = 1;
491         else
492             sx = -1;
493         end
494         if y0 < y1
495             sy = 1;
496         else
497             sy = -1;
498         end
499
500         err = dx - dy;
501
502         while 1
503             if x0 > 0 && x0 <= xmax && y0 > 0 && y0 <= ymax
504                 channel(x0, y0) = color;
505             end
506             if x0 == x1 && y0 == y1
507                 break;
508             end
509             e2 = 2 * err;
510             if e2 > -dy
511                 err = err - dy;
512                 x0 = x0 + sx;
513             end
514             if e2 < dx
515                 err = err + dx;
516                 y0 = y0 + sy;
517             end
518         end
519     end

```



```

520 end
521
522
523 function image = overlay_rays(image, from, to, length, varargin)
524     % parse options
525     argc = size(varargin, 2);
526     c = 1;
527     while c <= argc
528         arg = varargin{c};
529         if strcmpi(arg, 'Color')
530             if c + 1 > argc
531                 error('Color option should be followed by an integer tripplet');
532             end
533             colors = varargin{c + 1};
534             c = c + 1;
535         end
536         c = c + 1;
537     end
538     % option defaults
539     if ~exist('colors', 'var')
540         colors = [255 255 255];
541     end
542
543     num_rays = size(from, 1);
544     if size(colors, 1) == 1;
545         colors = repmat(colors, num_rays, 1);
546     end
547
548     for c = 1 : num_rays
549         if from(c,:) == to(c,:)
550             continue;
551         end
552         x0 = from(c,1);
553         y0 = from(c,2);
554         x1 = to(c,1);
555         y1 = to(c,2);
556         dx = x0 - x1;
557         dy = y0 - y1;
558         lambda = min(sqrt(length ^ 2 / (dx ^ 2 + dy ^ 2)), ...
559                     -sqrt(length ^ 2 / (dx ^ 2 + dy ^ 2)));
560
561         x = x0 + lambda * dx;
562         y = y0 + lambda * dy;
563
564         image = overlay_polygon(image, [x0 y0; x y], colors(c,:));
565     end
566 end

```