```
1 function res = main(path, image type, start offset, time step)
 2
       if nargin < 2</pre>
3
           image type = 'jpg';
 4
       end
 5
       if nargin < 3</pre>
 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 \ \sim] = 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
                track_mask = overlay_cross(track_mask, 1, r(1), r(2));
33
34
                rc(i,:) = [r(1), r(2)];
35
           end
36
37
           g = uint16(centroids(2,:));
           if g(1) > 0 \&\& g(2) > 0
38
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(:,:,\mathbf{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]);
        bg = overlay_polygon(bg, gc, [255, 255, 255]);
bg = overlay_polygon(bg, bc, [255, 255, 255]);
68
69
 70
        bg = overlay mask(bg, track mask);
        imshow(bg)
71
72 end
73
74
75 function [pretty mask, varargout] = analyse image(image)
        [num rows, num cols, num channels] = size(image);
76
77
        blob_mask = mask_colors(image);
78
        [convex_mask, ~] = mask_convex_regions(image, blob_mask);
        [convex mask]=demask triangles(image, convex mask, false);
79
        [convex mask, ~] = mask convex regions(image, convex mask);
80
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);
         [convex_mask]=demask_triangles(image, convex_mask, false);
97 %
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
            channel = image(:,:,c);
106
107
            channel mask = mask(:,:,c);
108
            channel trinagle mask = mask(:,:,c);
109
            if ~any(channel mask)
110
                continue;
111
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
123
            mean_rgb = mean(rgb_values(:));
            channel_mask(channel < mean_rgb) = 0;</pre>
124
            mask(:, :, c) = channel mask;
125
126
            props = regionprops(channel_mask, 'Centroid');
127
            centroid = props.Centroid;
            centroids(c,:) = [centroid(2), centroid(1)];
128
129
            if get triangle centroids
```

```
130
                channel trinagle mask(channel > mean rgb) = 0;
131
                channel trinagle mask=filter mask(channel trinagle mask);
                trinagle mask(:,:,c) = channel trinagle mask;
132
                props = regionprops(channel trinagle_mask, 'Centroid');
133
134
                centroid = props.Centroid;
                triangle_centroids(c,:) = [centroid(2), centroid(1)];
135
136
            end
137
        end
138
        vararqout{1} = centroids;
        varargout{2} = trinagle mask;
139
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);
        convex centroids = zeros(num_channels, 2);
147
148
        for c = 1 : num\_channels
149
            channel = mask(:,:,c);
150
            if ~any(channel(:))
151
                continue:
152
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), ...
                                convex_centroid(1) + props.BoundingBox(1)];
157
158
            convex centroids(c,:) = convex centroid;
159
            convex image = props.ConvexImage;
            [num_rows_convex, num_cols_convex] = size(convex image);
160
            for row = 1 : num rows convex
161
                for col = 1 : num cols convex
162
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
            end
169
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);
        rgb = double(reshape(image, num pixels, 3));
179
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);
            bN = rgbN(c,3);
191
            hue = hsv(c, 1) * 360;
192
193
            % current pixel is red
194
                    (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) && ...
                    (normal_prob(gN, gN_mean, gN_sdev) < 0.007)
199
200
                       image_mask(c,2) = 1;
201
            % current pixel is blue
                   (hue >= 150 && hue <= 270) && ...
202
            elseif
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
214 function x = normal prob(val, mu, sigma)
        x = 1.0 / (sigma * sqrt(2 * pi)) * exp(-(val - mu) ^ 2 / (2 * sigma ^ 2));
215
216 end
217
218
219 function image = remove noise(image)
        [~, ~, num channels] = size(image);
220
221
        for c = 1 : num\_channels
            channel = image(:,:,c);
222
            channel = bwmorph(channel, 'majority', Inf);
223
            channel = bwmorph(channel, 'bridge', Inf);
224
225
            image(:,:,c) = channel;
226
        end
227 end
228
229
230 % finds the connected components in each channel of |image| and removes those
231 % that are far away from the centroid of the pixels in that channel
232 % here, 'far away' means more distant than |distance_proprtion_threshold| times
233 % the average distance of each connected component to the channel centroid
234 % this removes big areas of noise such as the big green blob inside of the black
235 % arrow of the red robot in data/1/00000006.jpg
236 function image = remove outliers(image, distance proportion threshold)
237
        if nargin < 2</pre>
238
            distance proportion threshold = 0.5;
239
        end
        [~, ~, num_channels] = size(image);
240
241
        for c = 1 : num_channels
242
            channel = image(:,:,c);
243
            if ~any(channel(:))
244
                continue;
245
246
            channel_properties = regionprops(channel, 'Centroid');
            channel centroid = channel properties.Centroid;
247
248
            regions = bwconncomp(channel);
249
            regions_properties = regionprops(regions, 'Centroid', 'PixelIdxList');
250
            regions centroids = {regions properties.Centroid};
251
            distances = cellfun(@(x) norm(x - channel centroid), regions centroids);
252
            mean distance = mean(distances);
253
            for d = 1 : length(distances)
254
                if distances(d) > mean distance * distance proportion threshold
255
                    idx = regions_properties(d).PixelIdxList;
256
                    channel(idx) = 0;
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</pre>
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);
            if ~any(channel(:))
276
277
                continue;
278
279
            region props = regionprops(channel, 'Area');
280
            areas(c) = region props.Area;
281
        end
282
        \max \text{ area} = \max(\text{areas}(\text{areas} > 0)):
283
        for c = 1: num channels
284
            area = areas(c);
285
            if area == 0
286
                continue;
287
            end
288
                (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 conneceted regions but the biggest one
297 function mask = filter mask(mask)
        [x, y, num channels] = size(mask);
298
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);
            largest_blob_pixels = blob_list{find(nrows == max(nrows))};
305
306
            for i = 1 : max(nrows)
                channel(largest blob pixels(i)) = 1;
307
308
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)
        approximate = ~isempty(find(strcmpi(varargin, 'approximate')));
322
323
        red = double(image(:,:,1));
        green = double(image(:,:,2));
324
```

```
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);
        blue norm = round(blue(:,:) ./ euclid rqb .* 255);
333
        % some pixels are absolute black (r = g = b = 0) which causes division by
334
        % zero errors during normalisation and NaN values in the normalised channels
335
        % need to filter these values out
336
337
        red norm(isnan(red norm)) = 0:
        green norm(isnan(green norm)) = 0;
338
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);
        normalised image = cat(3, red norm, green norm, blue norm);
343
344 end
345
346
347 % returns image with a cross centered on pixel (|x|, |y|) drawn in |channel|
348 % [0, 0] is the top left corner of the image
349 function image = overlay cross(image, channel, x, y)
350
        [h w \sim] = size(image);
351
352
        if channel == 1
353
            other1 = 2;
354
            other2 = 3;
355
        end
        if channel == 2
356
357
            other1 = 1:
358
            other2 = 3;
359
        end
        if channel == 3
360
361
           other1 = 1;
362
           other2 = 2;
363
        end
364
365
        if y + 1 < h
366
            image(x, y + 1, channel) = 1;
367
            image(x, y + 1, other1) = 0;
368
            image(x, y + 1, other2) = 0;
369
        end
370
371
        if y - 1 > 0
372
            image(x, y - 1, channel) = 1;
373
            image(x, y - 1, other1) = 0;
374
            image(x, y - 1, other2) = 0;
375
        end
376
377
        if x + 1 < w
378
            image(x + 1, y, channel) = 1;
379
            image(x + 1, y, other1) = 0;
380
            image(x + 1, y, other2) = 0;
381
        end
382
383
        if x - 1 > 0
384
            image(x - 1, y, channel) = 1;
            image(x - 1, y, other1) = 0;
385
386
            image(x - 1, y, other2) = 0;
387
        end
388
389
        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')
                grayscale = 1;
406
            elseif strcmpi(arg, 'Saturation')
407
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
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
            channel = image(:,:,c);
443
444
            mask_pixels = find(mask(:,:,c) == 1);
445
            channel(mask pixels) = 255;
446
            image(:,:,c) = channel;
            for d = setdiff(channels, c)
447
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 % <a href="http://en.wikipedia.org/wiki/Bresenham">http://en.wikipedia.org/wiki/Bresenham</a>'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,:);
             stop = points(c + 1,:);
480
481
             x0 = start(1);
482
             y0 = start(2);
483
             x1 = stop(1);
484
             v1 = 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
506
                 if x0 == x1 & y0 == y1
507
                      break;
508
                 end
509
                 e2 = 2 * err;
510
                 if e2 > -dy
511
                      err = err - dy;
                      x0 = x0 + sx;
512
513
                 end
514
                 if e2 < dx
515
                      err = err + dx;
516
                      y0 = y0 + sy;
                 end
517
             end
518
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')
                if c + 1 > argc
530
                     error('Color option should be followed by an integer tripplet');
531
532
                 end
533
                 colors = varargin{c + 1};
534
                 c = c + 1;
535
            end
536
            c = c + 1;
537
        end
538
        % option defaults
        if ~exist('colors', 'var')
539
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);
            x1 = to(c, 1);
554
555
            y1 = to(c, 2);
556
            dx = x0 - x1;
557
            dy = y0 - y1;
            lambda = min(sqrt(length ^ 2 / (dx ^ 2 + dy ^ 2)), ...
558
                         -sqrt(length ^ 2 / (dx ^ 2 + dy ^ 2)));
559
560
561
            x = x0 + lambda * dx;
            y = y0 + lambda * dy;
562
563
564
            image = overlay_polygon(image, [x0 y0; x y], colors(c,:));
565
        end
566 end
```