

```

1  #include <opencv2/opencv.hpp>
2  #include <iostream>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <cmath>
6
7  #define LINES 'L'
8  #define RECTS 'R'
9
10 using namespace std;
11 using namespace cv;
12
13  /*****
14  *                               EXERCISE 4                               *
15  *                               -----                               *
16  *                               CONTOURS                               *
17  *****/
18
19  /// Function Prototypes
20 IplImage * simpleThresholding(IplImage *, IplImage *);
21 IplImage * getHistogram(IplImage *, char);
22 IplImage * getBinImagePlus(IplImage *, unsigned);
23 void searchContour(IplImage *, IplImage *);
24 void gradSearchContour(IplImage *, IplImage *);
25
26 int main(int argc, char **argv) {
27
28  /*****
29  *                               Gradient Method                               *
30  *****/
31  IplImage * img = cvLoadImage(argv[1], CV_LOAD_IMAGE_GRAYSCALE);
32
33  cvNamedWindow("Grey scale", 0);
34  cvShowImage("Grey scale", img);
35  cvSaveImage("Grey_Scale.png", img);
36
37  /// Create a binary image
38  IplImage * his = getHistogram(img, LINES);
39
40  cvNamedWindow("histogram", 0);
41  cvShowImage("histogram", his);
42  cvSaveImage("Histogram.png", his);
43
44  cvWaitKey(0);
45
46  IplImage * bin = simpleThresholding(img, his);
47
48  cvNamedWindow("Binary Image", 0);
49  cvShowImage("Binary Image", bin);
50  cvSaveImage("Binary_Image.png", bin);
51
52  /// Doesn't produce a very good result and requires some priori knowledge
53  /// about the scene (e.g. if one expects darker objects on a less dark
54  /// background, or viceversa).
55  IplImage * gradContour = cvCreateImage(cvSize(img->width - 2, img->height - 2),
IPL_DEPTH_8U, 1);
56  cvSet(gradContour, 0);
57  gradSearchContour(img, gradContour);
58
59  cvNamedWindow("Gradient Method", 0);
60  cvShowImage("Gradient Method", gradContour);
61  cvSaveImage("Gradient.png", gradContour);
62
63  IplImage * I = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 3);
64  cvCvtColor(img, I, COLOR_GRAY2RGB);
65  IplImage * m = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
66  cvSet(m, 0);
67
68  cvSetImageROI(m, cvRect(1, 1, img->width - 2, img->height - 2));
69  cvCopy(gradContour, m);
70  cvResetImageROI(m);
71  const CvMat M = cvarrToMat(m);
72  cvSet(I, cvScalar(255, 0, 0), &M);
73  cvNamedWindow("Show Gradient", 0);
74  cvShowImage("Show Gradient", I);
75  cvSaveImage("Show_Gradient.png", I);
76
77
78  /// every black pixel that has an adjacent white pixel is a border
79  IplImage * binGradContour = cvCreateImage(cvSize(img->width - 2, img->height - 2),
IPL_DEPTH_8U, 1);
80  cvSet(binGradContour, 0);
81  searchContour(bin, binGradContour);
82

```

```

83 cvNamedWindow("Binary-Gradient Contour",0);
84 cvShowImage("Binary-Gradient Contour", binGradContour);
85 cvSaveImage("Binary_Gradient_Contour.png", binGradContour);
86
87 cvCvtColor(img, I, COLOR_GRAY2RGB);
88 m = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
89 cvSet(m, 0);
90 cvSetImageROI(m, cvRect(1,1,img->width-2,img->height-2));
91 cvCopy(binGradContour, m);
92 cvResetImageROI(m);
93 const CvMat M_1 = cvarrToMat(m);
94 cvSet(I, cvScalar(0,0,255), &M_1);
95 cvNamedWindow("Show Binary Contour",0);
96 cvShowImage("Show Binary Contour", I);
97 cvSaveImage("Show_Binary_Contour.png", I);
98
99
100 /*****
101 * Gradient Method - the right way *
102 *****/
103
104 IplImage * Xim = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
105 IplImage * Yim = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
106 IplImage * gradientImage = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
107
108 float Prewitt_x [3][3] = {{ -1 , 0 , 1},
109                          { -1 , 0 , 1},
110                          { -1 , 0 , 1}};
111 float Prewitt_y [3][3] = {{ -1, -1, -1},
112                          { 0 , 0 , 0},
113                          { 1 , 1 , 1}};
114
115 CvMat PrewittXK, PrewittYK;
116 cvInitMatHeader(&PrewittXK, 3, 3, CV_32F, Prewitt_x);
117 cvInitMatHeader(&PrewittYK, 3, 3, CV_32F, Prewitt_y);
118
119 cvFilter2D(img, Xim, &PrewittXK, cvPoint(-1,-1));
120 cvFilter2D(img, Yim, &PrewittYK, cvPoint(-1,-1));
121
122 for(int i = 0; i < Xim->height; i++){
123     char * px = Xim->imageData + i*Xim->widthStep;
124     char * py = Yim->imageData + i*Yim->widthStep;
125     char * pmag = gradientImage->imageData + i*gradientImage->widthStep;
126     for(int j = 0; j < Xim->width; j++){
127         float x = (float)(uchar)(*px);
128         float y = (float)(uchar)(*py);
129         *pmag = (uchar)sqrt(x*x + y*y);
130         px++;
131         py++;
132         pmag++;
133     }
134 }
135
136 cvNamedWindow("Prewitt", 0);
137 cvShowImage("Prewitt", gradientImage);
138 cvSaveImage("Prewitt.png", gradientImage);
139
140 //////////////////////////////////////
141 float Sobel_x [3][3] = {{ -1 , 0 , 1},
142                        { -2 , 0 , 2},
143                        { -1 , 0 , 1}};
144 float Sobel_y [3][3] = {{ -1, -2, -1},
145                        { 0 , 0 , 0},
146                        { 1 , 2 , 1}};
147
148 CvMat SobelXK, SobelYK;
149 cvInitMatHeader(&SobelXK, 3, 3, CV_32F, Sobel_x);
150 cvInitMatHeader(&SobelYK, 3, 3, CV_32F, Sobel_y);
151
152 cvFilter2D(img, Xim, &SobelXK, cvPoint(-1,-1));
153 cvFilter2D(img, Yim, &SobelYK, cvPoint(-1,-1));
154
155 for(int i = 0; i < Xim->height; i++){
156     char * px = Xim->imageData + i*Xim->widthStep;
157     char * py = Yim->imageData + i*Yim->widthStep;
158     char * pmag = gradientImage->imageData + i*gradientImage->widthStep;
159     for(int j = 0; j < Xim->width; j++){
160         float x = (float)(uchar)(*px);
161         float y = (float)(uchar)(*py);
162         *pmag = (uchar)sqrt(x*x + y*y);
163         px++;
164         py++;
165         pmag++;
166     }

```

```

167     }
168
169     cvNamedWindow("Sobel", 0);
170     cvShowImage("Sobel", gradientImage);
171     cvSaveImage("Sobel.png", gradientImage);
172
173     ///////////////////////////////////////////////////
174     IplImage * CannyIm = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
175     cvLaplace(img, CannyIm, 5);
176     cvNamedWindow("Laplace", 0);
177     cvShowImage("Laplace", gradientImage);
178     cvSaveImage("Laplace.png", gradientImage);
179
180     ///////////////////////////////////////////////////
181     // Resulting image very dark
182     float Roberts_x [2][2] = {{ 0 , 1},
183                               {-1 , 0}};
184     float Roberts_y [2][2] = {{ 1 , 0},
185                               { 0 , -1}};
186
187     CvMat RobertsXK, RobertsYK;
188     cvInitMatHeader(&RobertsXK, 2, 2, CV_32F, Roberts_x);
189     cvInitMatHeader(&RobertsYK, 2, 2, CV_32F, Roberts_y);
190
191     cvFilter2D(img, Xim, &RobertsXK, cvPoint(-1,-1));
192     cvFilter2D(img, Yim, &RobertsYK, cvPoint(-1,-1));
193
194     for(int i = 0; i < Xim->height; i++){
195         char * px = Xim->imageData + i*Xim->widthStep;
196         char * py = Yim->imageData + i*Yim->widthStep;
197         char * pmag = gradientImage->imageData + i*gradientImage->widthStep;
198         for(int j = 0; j < Xim->width; j++){
199             float x = (float)(uchar)(*px);
200             float y = (float)(uchar)(*py);
201             *pmag = (uchar)sqrt(x*x + y*y);
202             px++;
203             py++;
204             pmag++;
205         }
206     }
207
208     cvNamedWindow("Roberts", 0);
209     cvShowImage("Roberts", gradientImage);
210     cvSaveImage("Roberts.png", gradientImage);
211
212     //*****
213     *                               Laplacian Method                               *
214     //*****
215
216     IplImage * LG_contour = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
217
218     /*double LG_mat [9][9] = {{0,0,1,2,2,2,1,0,0},
219                               {0,1,5,10,12,10,5,1,0},
220                               {1,5,15,19,16,19,15,5,1},
221                               {2,10,19,-19,-64,-19,19,10,2},
222                               {2,12,16,-64,-148,-64,16,12,2},
223                               {2,10,19,-19,-64,-19,19,10,2},
224                               {1,5,15,19,16,19,15,5,1},
225                               {0,1,5,10,12,10,5,1,0},
226                               {0,0,1,2,2,2,1,0,0}};*/
227
228     /*double LG_mat [7][7] = {{0, 1, 2, 4, 2, 1, 0},
229                               {1, 7, 24, 31, 24, 7, 1},
230                               {2, 24, 17, -51, 17, 24, 2},
231                               {4, 31, -51, -248, -51, 31, 4},
232                               {2, 24, 17, -51, 17, 24, 2},
233                               {1, 7, 24, 31, 24, 7, 1},
234                               {0, 1, 2, 4, 2, 1, 0}};*/
235
236     double LG_mat [5][5] = {{1,9,19,9,1},
237                               {9,58,12,58,9},
238                               {19,12,-432,12,19},
239                               {9,58,12,58,9},
240                               {1,9,19,9,1}};
241
242     ///////////////////////////////////////////////////
243     /*double LG_mat [3][3] = {{ 1, 4, 1},
244                               { 4, -20, 4},
245                               { 1, 4, 1}};*/
246
247     CvMat LG_kernel;
248     cvInitMatHeader(&LG_kernel, 5, 5, CV_64F, LG_mat);
249
250     cvFilter2D(img, LG_contour, &LG_kernel, cvPoint(-1,-1));
251
252     cvNamedWindow("Laplacian Contour",0);

```

```

251 cvShowImage("Laplacian Contour", LG_contour);
252 cvSaveImage("Laplacian_Contour.png", LG_contour);
253
254
255 /*****
256 *                               Contour Search                               *
257 *****/
258
259 // To implement
260
261 cvWaitKey(0);
262
263
264 /*****
265 *                               Clean-Up                               *
266 *****/
267 cvReleaseImage(&img);
268 cvReleaseImage(&his);
269 cvReleaseImage(&bin);
270 cvReleaseImage(&gradContour);
271 cvReleaseImage(&binGradContour);
272 cvReleaseImage(&LG_contour);
273
274 cvDestroyAllWindows();
275
276 return 0;
277 }
278
279
280
281 bool gradCheck (char *, char *, int);
282
283 void gradSearchContour(IplImage * img, IplImage * contour) {
284     for(int i = 0; i < img->height - 2; i++) {
285         char * p_img = img->imageData + (i + 1)*img->widthStep + 1;
286         char * p_contour = contour->imageData + i*contour->widthStep;
287         for(int j = 0; j < img->width - 2; j++) {
288             if ( gradCheck(p_img, p_img - 1 - img->widthStep, img->widthStep) )
289                 *p_contour = 255;
290             p_img++;
291             p_contour++;
292         }
293     }
294 }
295
296 bool gradCheck (char * c, char * p, int step) {
297     for(int i = 0; i < 3; i++) {
298         char * lp = p + i*step;
299         for(int j = 0; j < 3; j++) {
300             if (i == 1 && j == 1) {
301                 lp++;
302                 continue;
303             }
304             float diff = (float)((uchar)(*lp) - (uchar)(*c));
305             if(diff > 23)
306                 return true;
307             lp++;
308         }
309     }
310     return false;
311 }
312
313
314 bool check(char *, int);
315
316 void searchContour(IplImage * bin, IplImage * contour) {
317     for(int i = 0; i < bin->height - 2; i++) {
318         char * p_bin = bin->imageData + (i + 1)*bin->widthStep + 1;
319         char * p_contour = contour->imageData + i*contour->widthStep;
320         for(int j = 0; j < bin->width - 2; j++) {
321             if ( ((uchar)*p_bin == 0) && check(p_bin - 1 - bin->widthStep, bin->widthStep) )
322                 *p_contour = 255;
323             p_bin++;
324             p_contour++;
325         }
326     }
327 }
328
329 bool check (char * p, int step) {
330     for(int i = 0; i < 3; i++) {
331         char * lp = p + i*step;
332         for(int j = 0; j < 3; j++) {
333             if (i == 1 && j == 1) {
334                 lp++;

```

```

335         continue;
336     }
337     if((uchar)(*lp) == 255)
338         return true;
339     lp++;
340 }
341 }
342 return false;
343 }
344
345
346 /*****
347 * This function allows the user to select a threshold and draws a new
348 * histogram with a visual indication of the threshold. At the same time
349 * computes the necessary numbers for the centre of mass.
350 *
351 * The function needs:
352 * - a pointer to a GREY SCALE image to apply the threshold to;
353 * - a pointer to a preliminary histogram image (starting point to
354 * draw the new histogram with the threshold);
355 * - 3 pointers to int to compute the center of mass coordinates
356 *****/
357 IplImage * simpleThresholding(IplImage * G, IplImage * his) {
358     unsigned int t = 0; // Threshold
359     char st = 4; // To place correctly the threshold line in the image
360     IplImage * bin; // To contain the binary image
361
362     printf("\nInsert Threshold: ");
363     scanf("%u", &t);
364
365     IplImage * Chis = cvCreateImage(cvGetSize(his), IPL_DEPTH_8U, 3);
366     /// Represent a grey scale histogram in a 3-channel image.
367     /// It will be useful later to draw the threshold on the histogram.
368     cvCvtColor(his, Chis, COLOR_GRAY2RGB);
369     cvFlip(Chis, Chis, 0);
370
371     cvLine(Chis, cvPoint(t*st, 0), cvPoint(t*st, 600), cvScalar(0,0,255), 2, 4);
372     cvShowImage("histogram", Chis);
373     cvSaveImage("histogram.png", Chis);
374     cvReleaseImage(&Chis);
375
376     bin = getBinImagePlus(G, t);
377     cvNamedWindow("Binary Image", 0);
378     cvShowImage("Binary Image", bin);
379
380     return bin;
381 }
382
383
384 /*****
385 * From the pointer to a greyscale image, draws a histogram image and
386 * returns a pointer to it. Two possible styles can be selected:
387 * LINES (a line for each bin)
388 * RECTS (a rectangle for each bin)
389 *****/
390 IplImage * getHistogram(IplImage * gray, char style) {
391     int dep = 256;
392     int hist[dep]; // Histogram for gray scale image
393     int st = 4; // To separate lines in the histograms, for better look ;)
394
395     /// Initialize histograms
396     for (int i = 0; i < dep; i++)
397         hist[i] = 0;
398
399     /// Calculate histogram
400     for(int i = 0; i < gray->height; i++) {
401         char * ptr = gray->imageData + i*gray->widthStep;
402         for(int j = 0; j < gray->width; j++) {
403             if(*ptr < 0) {
404                 char c = *ptr;
405                 uchar x = (uchar) c;
406                 hist[x] += 1;
407             }
408             else
409                 hist[(unsigned)(*ptr)] += 1;
410             ptr++;
411         }
412     }
413
414     /// Create image for the histograms
415     IplImage * his = cvCreateImage(cvSize(st*dep,600), IPL_DEPTH_8U, 1);
416     cvSet(his, 0); // Initialize image (all black)
417     his->origin = IPL_ORIGIN_BL; // Set the origin in the bottom left corner
418

```

```

419     /// Draw a line/Rectangle for each bin
420     if (style == LINES)
421         for (int i = 0; i < dep; i++)
422             if (hist[i] != 0)
423                 cvLine(his, cvPoint(i*st, 0), cvPoint(i*st, hist[i]/10), 150, 1, 4);
424     else if (style == RECTS)
425         for (int i = 0; i < dep; i++)
426             if (hist[i] != 0)
427                 cvRectangle(his, cvPoint(i*st, hist[i]/10), cvPoint((i+1)*st, 0), 150, -1,
428                             4);
429     else {
430         printf("\nError. No style selected.\n");
431         cvReleaseImage(&his);
432         return NULL;
433     }
434     return his;
435 }
436
437
438     /*****
439     * Input: a pointer to a GREY SCALE image, a threshold, 3 pointers to
440     * integer for the necessary computation to determine the center of mass.
441     * Forms a binary image, counts the number of pixels set to 0, sums
442     * the values of the 2 coordinates for the pixels set to 0 (used later to
443     * calculate the center of mass coordinates).
444     *
445     * PS. It is possible to calculate the coordinates even in this function
446     * but it is done later to respect the structure of the exercise
447     *****/
448 IplImage * getBinImagePlus(IplImage * img, unsigned t) {
449     IplImage * I = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
450     for(int i = 0; i < img->height; i++) {
451         char * ptr = img->imageData + i*img->widthStep;
452         char * p = I->imageData + i*I->widthStep;
453         for(int j = 0; j < img->width; j++) {
454             if ((uchar)(*ptr) >= (uchar)t)
455                 *p = 255;
456             else {
457                 // In this case we are on the object
458                 *p = 0;
459             }
460             ptr++;
461             p++;
462         }
463     }
464     return I;
465 }
466

```