霍夫变换--现在可以公开的情报

hough-transform

1.霍夫变换检测圆

1.1 原理

圆的方程可以写成如下的形式

知道圆的圆心(x0,y0)和半径r，遍历360度可以得到圆上所有的点。

反过来想，如果知道圆上的所有点(xi,yi),和圆半径r,根据

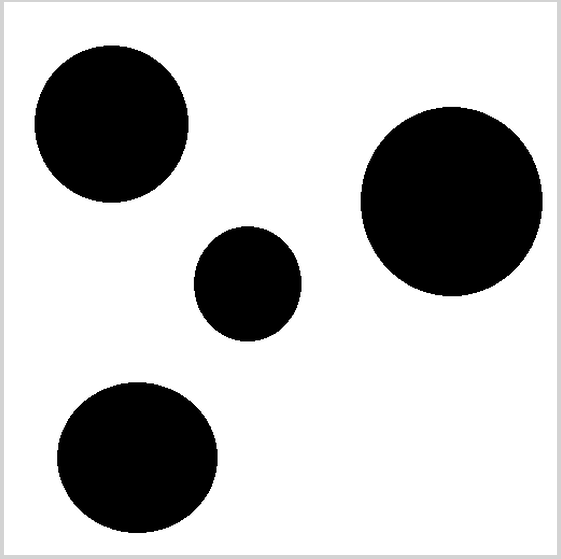
构造三维霍夫空间 进行投票累加，找到空间中的局部最大值则是最可能的圆心坐标(x0,y0)

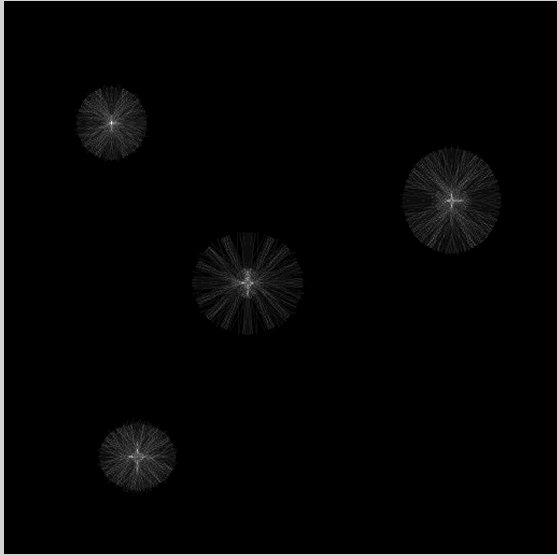
1.2 流程图

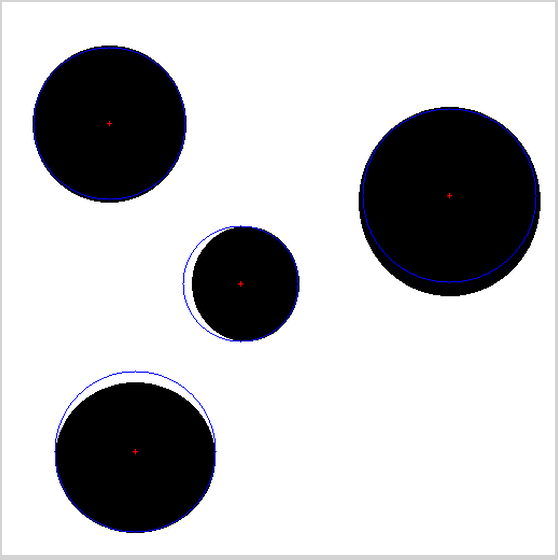
输入图像边缘检测（强度和方向）构造霍夫空间H(x\_0,y\_0,r) 。根据目标半径的最大最小值，将每个边缘点转换到圆心的强度累计。设定投票最低阈值，对高于阈值的圆进行筛选圆心距在设定范围内?霍夫票数更高更新并保存最佳圆yesnoyesno

1.3 运行效果

（参数设置：投票阈值5，最小半径40最大半径100，最小圆心距140）

*原图*   


*霍夫投票强度图*   


*绘制结果*   


1.4 核心代码

1. //////////////////////////////////////////////////
2. //houghCircle功能说明：霍夫圆检测
3. //Input
4. // edgeInformation \*edgeArray 边缘点信息
5. // IMG\_INT eNum 边缘点个数
6. // IMG\_RBUF angle\_rBuf 梯度方向图
7. // int voteScore 投票阈值
8. // int minRadius 最小半径
9. // int maxRadius 最大半径
10. // int center\_Dis 圆心距离
11. //Output
12. // vector<houghCircle3f> &bestCircles 输出所有最佳圆
13. //////////////////////////////////////////////////
14. int CVisHoughTransform::houghCircle(edgeInformation \*edgeArray, IMG\_INT eNum, /\*IMG\_WBUF mag\_wBuf,\*/IMG\_RBUF angle\_rBuf, int voteScore, int minRadius, int maxRadius, int center\_Dis, vector<houghCircle3f> &bestCircles)
15. {
16. VisBuf setVisbuf;
17. int HEIGHT = angle\_rBuf.size.height;
18. int WIDTH = angle\_rBuf.size.width;
19. int DEPTH = maxRadius;
20. int \*\*\*H;
21. // Allocate memory
22. H = new int\*\*[HEIGHT + 1];
23. for (int i = 0; i < HEIGHT; i++)
24. {
25. H[i] = new int\*[WIDTH + 1];
26. for (int j = 0; j < WIDTH; j++)
27. {
28. H[i][j] = new int[DEPTH + 1];
29. memset(H[i][j], 0, sizeof(int)\* (DEPTH + 1) );
30. }
31. }
32. //accumulate1
33. for (int k = 0; k < eNum; k++)
34. {
35. for (int r = minRadius; r <= maxRadius; r++)
36. {
37. int temp\_x = edgeArray[k].xyInteger.x;
38. int temp\_y = edgeArray[k].xyInteger.y;
39. int x0 = (int)round(edgeArray[k].xyDecimal.x - r \* cos(angle\_rBuf.ptr[temp\_y \* WIDTH + temp\_x] / 180 \* PI));
40. int y0 = (int)round(edgeArray[k].xyDecimal.y - r \* sin(angle\_rBuf.ptr[temp\_y \* WIDTH + temp\_x] / 180 \* PI));
41. inc\_if\_inside(H, x0, y0, HEIGHT, WIDTH, r);
42. }
43. }
44. //show hAcc;
45. IMG\_WBUF hAcc\_wBuf;
46. IMG\_WORD \*pHAccData = new IMG\_WORD[WIDTH \* HEIGHT];
47. memset(pHAccData, 0, sizeof(IMG\_WORD) \* WIDTH \* HEIGHT);
48. int pos = 0;
49. for (int y0 = 0; y0 < HEIGHT; y0++)
50. {
51. for (int x0 = 0; x0 < WIDTH; x0++)
52. {
53. for (int r = minRadius; r <= maxRadius; r++)
54. {
55. if (H[y0][x0][r] > pHAccData[pos + x0] )
56. {
57. pHAccData[pos + x0] = H[y0][x0][r];
58. }
59. }
60. }
61. pos += WIDTH;
62. }
63. setVisbuf.set\_IMG\_WBUF(hAcc\_wBuf, pHAccData, { (IMG\_UWORD)WIDTH,(IMG\_UWORD)HEIGHT }, WIDTH \* sizeof(IMG\_WORD));
64. //compute optimal circles
65. for (int y0 = 0; y0 < HEIGHT; y0++)
66. {
67. for (int x0 = 0; x0 < WIDTH; x0++)
68. {
69. for (int r = minRadius; r <= maxRadius; r++)
70. {
71. if (H[y0][x0][r] > voteScore)
72. {
73. houghCircle3f circle;
74. circle.centerX = (float)x0;
75. circle.centerY = (float)y0;
76. circle.radius = (float)r;
77. int i;
78. for (i = 0; i < (int)bestCircles.size(); i++) {
79. int xCoord = (int)bestCircles[i].centerX;
80. int yCoord = (int)bestCircles[i].centerY;
81. int radius = (int)bestCircles[i].radius;
82. if (abs(xCoord - x0) < center\_Dis && abs(yCoord - y0) < center\_Dis) //圆心距在一定范围（同圆）
83. {
84. if (H[y0][x0][r] > H[yCoord][xCoord][radius]) //票数高，替换
85. {
86. bestCircles.erase(bestCircles.begin() + i);
87. bestCircles.insert(bestCircles.begin(), circle);
88. }
89. break;
90. }
91. }
92. if (i == bestCircles.size()) {
93. bestCircles.insert(bestCircles.begin(), circle);
94. }
95. }
96. }
97. }
98. }
99. //free
100. for (int i = 0; i < HEIGHT; i++)
101. {
102. for (int j = 0; j < WIDTH; j++)
103. {
104. if (H[i][j] != NULL)
105. {
106. delete[] H[i][j];
107. }
108. }
109. }
110. for (int i = 0; i < HEIGHT; i++)
111. {
112. if (H[i] != NULL)
113. delete[] H[i];
114. }
115. if(H != NULL)
116. delete[] H;
117. delete[] pHAccData;
118. return 0;
119. }

1.5 待思考的问题

* 1. 不同图像在霍夫空间投出来的票数不一样，不应该通过设阈值来筛选最优圆心，而是改为寻找局部最大值。
* 2. 圆内有圆的情况怎么处理？
* 3. 优化速度的方法：随机霍夫变换？

2.广义霍夫变换

2.1 原理

广义霍夫变换之所以能处理任意形状的图形并不是找到了可以表示任意图形的方程（这是不可能的），而是使用表的形式描述一种图形，把图形边缘点坐标保存在一张表中，有了表之后就需要找到一种可以把图形点集投射到参数空间的一点的转换算法。

首先我们需要一个模板，这个模板包含边缘点的梯度强度和梯度方向的信息，我们还需要模板的一个参考点(xref,yref)。 计算每个边缘点(xi,yi)到参考点的距离并记录与梯度方向垂直的直线与x轴方向的夹角 。遍历模板的特征点，将特征向量，保存在一个叫R-Table的表里。

对于待搜索的图像，我们同样遍历所有的边缘点(xi',yi')，对应的角度为,然后在先前的R\_Table中寻找时的点对。

（1）假设目标没有缩放和旋转，则我们可以构造一个二维霍夫空间H(xref',yref'),新的参考点坐标为(xref',yref') = (xi' + , yi' + ),在相应的H(xref',yref')加一票。……   
最终得票最高的H(xref',yref')对应的参考点(xref',yref')即为所求。

（2）假设目标有旋转和缩放。根据公式推导：

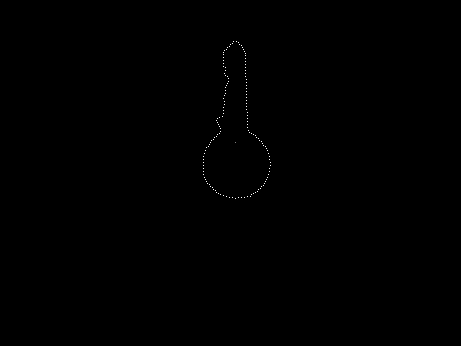
我们只需要对原先存在R-Table里的作相应的旋转缩放变换，霍夫空间相应的增加到四维H(xref',yref'，S , ),最终同样可以找到票数最高的参考点(xref',yref')。

（具体算法流程见[这里](http://www.itriacasa.it/generalized-hough-transform/tutorial.html)，解释得很详细）。

2.2 流程图

输入模板图像和待搜索图像创建模板（记录特征点信息）并构造模板的R-Table设置旋转角度和缩放大小的步长，根据旋转和缩放构造R-rotated-scaled-Table。对待搜索图像进行边缘提取，获取特征点后根据角度匹配找到R-rotated-scaled-Table中对应的点，相加计算出目标参考点位置。相应在霍夫空间H(xref',yref'，S , theta)投票获得最佳参考点(xref',yref')

2.3 运行结果

*模板图*   


*待搜索的图*   


*搜索结果*   


2.4 核心代码 （省略了模板构造R-Table的部分）

1. // fill accumulator matrix
2. void accumulate(cv::Mat& input\_img){
3. showimage = input\_img;
4. // transform image to grayscale:
5. Mat src\_gray;
6. src\_gray.create( Size(input\_img.cols, input\_img.rows), CV\_8UC1);
7. cvtColor(input\_img, src\_gray, CV\_BGR2GRAY);
8. // reduce noise with a kernel 3x3 and get cannyedge image:
9. Mat detected\_edges;
10. blur( src\_gray, detected\_edges, Size(3,3) );
11. Canny( detected\_edges, detected\_edges, thr1, thr2, 3 );
12. //imshow("detected\_edges", detected\_edges);
13. // get Scharr matrices from image to obtain contour gradients
14. Mat dx;
15. dx.create( Size(input\_img.cols, input\_img.rows), CV\_16SC1);
16. Sobel(src\_gray, dx, CV\_16S, 1, 0, CV\_SCHARR);
17. Mat dy;
18. dy.create( Size(input\_img.cols, input\_img.rows), CV\_16SC1);
19. Sobel(src\_gray, dy, CV\_16S, 0, 1, CV\_SCHARR);
20. // load all points from image all image contours on vector pts2
21. int nl= detected\_edges.rows;
22. int nc= detected\_edges.cols;
23. float deltaphi = pi/intervals;
24. float inv\_deltaphi = (float)intervals/pi;
25. float inv\_rangeXY = (float)1/rangeXY;
26. float pi\_half = pi\*0.5f;
27. std::vector<Rpoint2> pts2;
28. for (int j=0; j<nl; ++j) {
29. uchar\* data= (uchar\*)(detected\_edges.data + detected\_edges.step.p[0]\*j);
30. for (int i=0; i<nc; ++i) {
31. if ( data[i]==255 ) // consider only white points (contour)
32. {
33. short vx = dx.at<short>(j,i);
34. short vy = dy.at<short>(j,i);
35. Rpoint2 rpt;
36. rpt.x = i\*inv\_rangeXY;
37. rpt.y = j\*inv\_rangeXY;
38. float a = atan2((float)vy, (float)vx); // gradient angle in radians
39. float phi = ((a > 0) ? a-pi\_half : a+pi\_half); // contour angle with respect to x axis
40. int angleindex = (int)((phi+pi\*0.5f)\*inv\_deltaphi); // index associated with angle (0 index = -90 degrees)
41. if (angleindex == intervals) angleindex=intervals-1;// -90癮ngle and +90?has same effect
42. rpt.phiindex = angleindex;
43. pts2.push\_back( rpt );
44. }
45. }
46. }
47. // OpenCv 4-dimensional matrix definition and in general a useful way for defining multidimensional arrays and vectors in c++
48. // create accumulator matrix
49. int X = ceil((float)nc/rangeXY);
50. int Y = ceil((float)nl/rangeXY);
51. int S = ceil((float)(wmax-wmin)/rangeS+1.0f);
52. int R = ceil(phimax/deltaphi)-floor(phimin/deltaphi);
53. if (phimax==pi && phimin==-pi) R--;
54. int r0 = -floor(phimin/deltaphi);
55. int matSizep\_S[] = {X, Y, S, R};
56. accum.create(4, matSizep\_S, CV\_16S);
57. accum = Scalar::all(0);
58. // icrease accum cells with hits corresponding with slope in Rtable vector rotatated and scaled
59. float inv\_wtemplate\_rangeXY = (float)1/(wtemplate\*rangeXY);
60. // rotate RTable from minimum to maximum angle
61. for (int r=0; r<R; ++r) { // rotation
62. std::cout << "r : R" << " " << r << " : " << R << std::endl;
63. int reff = r-r0; //because phi is (-pi ~ pi)
64. std::vector<std::vector<Vec2f>> Rtablerotated(intervals);
65. // cos and sin are computed in the outer loop to reach computational efficiency
66. float cs = cos(reff\*deltaphi);
67. float sn = sin(reff\*deltaphi);
68. for (std::vector<std::vector<Vec2i>>::size\_type ii = 0; ii < Rtable.size(); ++ii){
69. for (std::vector<Vec2i>::size\_type jj= 0; jj < Rtable[ii].size(); ++jj){
70. int iimod = (ii+reff) % intervals; //?????????? //shoule correspond to theta
71. Rtablerotated[iimod].push\_back(Vec2f(cs\*Rtable[ii][jj][0] - sn\*Rtable[ii][jj][1], sn\*Rtable[ii][jj][0] + cs\*Rtable[ii][jj][1]));
72. }
73. }
74. // scale the rotated RTable from minimum to maximum scale
75. for (int s=0; s<S; ++s) { // scale
76. std::vector<std::vector<Vec2f>> Rtablescaled(intervals);
77. int w = wmin + s\*rangeS;
78. float wratio = (float)w\*inv\_wtemplate\_rangeXY;
79. for (std::vector<std::vector<Vec2f>>::size\_type ii = 0; ii < Rtablerotated.size(); ++ii){
80. for (std::vector<Vec2f>::size\_type jj= 0; jj < Rtablerotated[ii].size(); ++jj){
81. Rtablescaled[ii].push\_back(Vec2f(wratio\*Rtablerotated[ii][jj][0], wratio\*Rtablerotated[ii][jj][1]));
82. }
83. }
84. // iterate through each point of edges and hit corresponding cells from rotated and scaled Rtable
85. for (vector<Rpoint2>::size\_type t = 0; t < pts2.size(); ++t){ // XY plane
86. int angleindex = pts2[t].phiindex;
87. for (std::vector<Vec2f>::size\_type index = 0; index < Rtablescaled[angleindex].size(); ++index){
88. float deltax = Rtablescaled[angleindex][index][0];
89. float deltay = Rtablescaled[angleindex][index][1];
90. int xcell = (int)(pts2[t].x + deltax);
91. int ycell = (int)(pts2[t].y + deltay);
92. if ( (xcell<X)&&(ycell<Y)&&(xcell>-1)&&(ycell>-1) ){
93. //(\*( (short\*)(accum.data + xcell\*accum.step.p[0] + ycell\*accum.step.p[1] + s\*accum.step.p[2]+ r\*accum.step.p[3])))++;
94. (\*ptrat4D(accum, xcell, ycell, s, r))++;
95. }
96. }
97. }
98. }
99. }
100. }
101. // show the best candidate detected on image
102. void bestCandidate(){
103. double minval;
104. double maxval;
105. int id\_min[4] = { 0, 0, 0, 0};
106. int id\_max[4] = { 0, 0, 0, 0};
107. minMaxIdx(accum, &minval, &maxval, id\_min, id\_max);
108. int nl= showimage.rows;
109. int nc= showimage.cols;
110. Mat input\_img2 = showimage.clone();
111. Vec2i referenceP = Vec2i(id\_max[0]\*rangeXY/\*+(rangeXY+1)/2\*/, id\_max[1]\*rangeXY/\*+(rangeXY+1)/2\*/); //???why plus (rangeXY+1)/2
112. //Vec2i referenceP = Vec2i(id\_max[0] \* rangeXY, id\_max[1] \* rangeXY);
113. // rotate and scale points all at once. Then impress them on image
114. std::vector<std::vector<Vec2i>> Rtablerotatedscaled(intervals);
115. float deltaphi = pi/intervals;
116. int r0 = -floor(phimin/deltaphi);
117. int reff = id\_max[3]-r0;
118. float cs = cos(reff\*deltaphi);
119. float sn = sin(reff\*deltaphi);
120. int w = wmin + id\_max[2]\*rangeS;
121. float wratio = (float)w/(wtemplate);
122. for (std::vector<std::vector<Vec2i>>::size\_type ii = 0; ii < Rtable.size(); ++ii){
123. for (std::vector<Vec2i>::size\_type jj= 0; jj < Rtable[ii].size(); ++jj){
124. int iimod = (ii+reff) % intervals;
125. int dx = roundToInt(wratio\*(cs\*Rtable[ii][jj][0] - sn\*Rtable[ii][jj][1]));
126. int dy = roundToInt(wratio\*(sn\*Rtable[ii][jj][0] + cs\*Rtable[ii][jj][1]));
127. int x = referenceP[0] - dx;
128. int y = referenceP[1] - dy;
129. //Rtablerotatedscaled[ii].push\_back(Vec2i( dx, dy));
130. if ( (x<nc)&&(y<nl)&&(x>-1)&&(y>-1) ){
131. input\_img2.at<Vec3b>(y, x) = Vec3b(0, 255, 255);
132. }
133. }
134. }
135. // show result
136. bool alt = false;
137. for (;;)
138. {
139. char c = (char)waitKey(750);
140. if (c == 27)
141. break;
142. if (alt) {
143. imshow("input\_img", input\_img2);
144. }
145. else {
146. imshow("input\_img", showimage);
147. }
148. alt = !alt;
149. }
150. imwrite("res.bmp", input\_img2);
151. }

2.5 待思考的问题

* 1. 代码中最后算参考点Vec2i referenceP = Vec2i(id\_max[0]\*rangeXY/\*+(rangeXY+1)/2\*/, id\_max[1]\*rangeXY/\*+(rangeXY+1)/2\*/);里   
  (rangeXY+1)/2应该改成(rangeXY)/2？   
  因为对于(rangeXY \* rangeXY的一个“像素块”，(rangeXY)/2才是中心)。
* 2. 速度慢的惊人，要对特征点进行筛选。