论文思路

图像处理，首先对数据进行观察，发现图像具有以下几个问题：在拍摄过程中摄像机有晃动；拍摄的相机并不与坩埚口的平面垂直，导致坩埚口的形状并不是正圆而是椭圆；图像中并不完全是坩埚，还有许多其他无关内容；图像不够清晰有噪音；晶体的颜色、边界与背景区别不大等。这些问题都对目标晶体的提取以及追踪产生了一定的困难，需要逐个解决。

首先解决的问题是对于坩埚口的定位以及图像的裁剪。我们首先对图像的尺寸进行变换，将坩埚口变成正圆型，随后使用了“霍夫算法”对正圆型进行特征提取，特征提取完成后以圆形的外接正方形进行裁剪（图像的边缘与坩埚口相切）。这样一次性就解决了三个问题：消除镜头的晃动的影响、消除图像中无关内容对晶体图像识别提取的影响、便于通过坩埚口的直径测量晶体的物理尺度。

接下来进行晶体的识别，我们尝试了二值化的方法，但是效果不是很好，最终我们选择了floodFill算法进行识别。floodFill算法的基本原理是选取种子点，然后将种子点周围颜色相近的点纳入识别的范围，这样依次循环直到找不到颜色相近的点。在使用floodFill算法标记出来晶体的点后，使用凸包算法画出晶体的轮廓并且将晶体的轮廓视作一个凸多边形，存储多边形的形心作为晶体的质心，存储多边形的各个顶点作为多边形的轮廓信息，进行后续的面积、速度、加速度等信息的处理。

代码：

Hough.py（第一步）

1. **import** cv2
2. **import** numpy as np
3. **import** os
5. root\_dir = r".\imgs"
7. **def** process(filename):
8. open\_filename = root\_dir + "\\" + filename
9. write\_filename1 = ".\\hough\_light\_res\_3" + "\\" + filename
10. write\_filename2 = ".\\hough\_light\_circle\_3" + "\\" + filename
11. img = cv2.imread(open\_filename, 0)
12. gray = img[10:1090, 450:1680]
13. gray = cv2.resize(gray, (0, 0), fx=0.905, fy=1)
14. light = np.zeros\_like(gray)
15. light\_y, light\_x = light.shape
16. light[round(light\_y/2):light\_y, 0:round(light\_x / 2)] = 50
17. detected\_edges = cv2.GaussianBlur(gray, (15, 15), 0)
18. detected\_edges = cv2.add(detected\_edges, light)
19. detected\_edges = cv2.equalizeHist(detected\_edges)
20. circles1 = cv2.HoughCircles(detected\_edges, cv2.HOUGH\_GRADIENT, 1,
21. 500, param1=80, param2=20, minRadius=480, maxRadius=520)
22. circles = circles1[0, :, :]
23. circles = np.uint16(np.around(circles))
24. min\_x, min\_y = 9999, 9999
25. max\_x, max\_y = 0, 0
26. **for** i **in** circles[:]:
27. x\_p = 566; y\_p = 550; r\_p = 517
28. cv2.circle(detected\_edges, (x\_p, y\_p), r\_p, (255, 0, 0), 5)
29. cv2.circle(detected\_edges, (i[0], i[1]), 2, (255, 0, 255), 10)
30. **if**(min\_x > i[0] - i[2]):
31. min\_x = x\_p - r\_p
32. **if**(min\_y > i[1] - i[2]):
33. min\_y = y\_p - r\_p
34. **if**(max\_x < i[0] + i[2]):
35. max\_x = x\_p + r\_p
36. **if**(max\_y < i[1] + i[2]):
37. max\_y = y\_p + r\_p
38. **print**("圆心坐标", i[0], i[1])
39. gray = gray[min\_y:max\_y, min\_x:max\_x]
41. cv2.imwrite(write\_filename1, gray)
42. cv2.imwrite(write\_filename2, detected\_edges)
44. **if** \_\_name\_\_ == "\_\_main\_\_":
45. **for** file **in** os.listdir(root\_dir):
46. process(file)

floodFIll.py（第二步）

1. **import** cv2
2. **import** numpy as np
3. drawing = False
4. mode = True
5. ix, iy = -1, -1
6. color = (0, 0, 255)
7. lower = np.array([0, 253, 253])
8. upper = np.array([3, 255, 255])
9. hull = None
10. point = None

13. **def** draw\_circle(event, x, y, flags, param):
14. **global** ix, iy, drawing, mode, hull, point
16. **if** event == cv2.EVENT\_LBUTTONDOWN:
17. **if** mode:
18. cv2.floodFill(img, None, (x, y), color, (scale, scale, scale), (scale, scale, scale), 4)
19. **else** :
20. cv2.circle(img, (x, y), 4, (255,255,255), -1)
22. **elif** event == cv2.EVENT\_MOUSEMOVE **and** flags == cv2.EVENT\_FLAG\_LBUTTON:
23. **if** mode:
24. cv2.floodFill(img, None, (x, y), color, (scale, scale, scale), (scale, scale, scale), 4)
25. **else** :
26. cv2.circle(img, (x, y), 4, (255,255,255), -1)
28. **elif** event == cv2.EVENT\_RBUTTONDOWN:
29. bag()
31. **elif** event == cv2.EVENT\_LBUTTONUP:
32. **pass**

35. **def** showSetting(img):
36. **pass**
38. **def** write(fileno):
39. df = open(".\\data\_2\\" + fileno + ".txt", "w")
40. df.write("center:" + str(point) + "\n")
41. this\_img = orgin\_img.copy()
42. **for** i **in** range(len(hull)):
43. cv2.line(this\_img, tuple(hull[i][0]), tuple(hull[(i + 1) % len(hull)][0]), (0, 255, 255), 2)
44. cv2.circle(this\_img, point, 2, (0, 255, 255), -1)
45. df.write("edge:" + str(tuple(hull[i][0])) + "\n")
47. cv2.imwrite(".\\data\_2\\" + fileno + ".bmp", this\_img)
49. **def** bag():
50. **print**('yes')
51. **global** hull, point
52. hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)
53. mask = cv2.inRange(hsv, lower, upper)
54. contours, hierarchy = cv2.findContours(mask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_TC89\_KCOS)
55. cnt = contours[0]
56. **for** cnt **in** contours:
57. hull = cv2.convexHull(cnt)
58. length = len(hull)
59. **if** length > 5:
60. **for** i **in** range(length):
61. cv2.line(img, tuple(hull[i][0]), tuple(hull[(i + 1) % length][0]), (0, 255, 0), 1)
62. M = cv2.moments(cnt)
63. cX = int(M["m10"] / M["m00"])
64. cY = int(M["m01"] / M["m00"])
65. point = (cX, cY)
66. cv2.circle(img, (cX, cY), 2, (0, 255, 255), -1)


70. **if** \_\_name\_\_ == "\_\_main\_\_":
71. fileno = '0575'
72. scale = 1
74. img = cv2.imread('.\\all\_res\\' + fileno + '.bmp', 0)
75. img = cv2.equalizeHist(img)
76. img = cv2.cvtColor(img, cv2.COLOR\_GRAY2RGB)
77. orgin\_img = img.copy()
78. # img = cv2.blur(img, (5,5), 0)
79. cv2.namedWindow('image')
80. linetype=cv2.LINE\_AA
81. showSetting(img)
82. cv2.setMouseCallback('image', draw\_circle)
83. **while** (1):
84. cv2.imshow('image', img)
85. k = cv2.waitKey(1) & 0xFF
86. **if** k == ord('w'):
87. write(fileno)
88. **elif** k == ord('1'):
89. scale = 1
90. **elif** k == ord('2'):
91. scale = 2
92. **elif** k == ord('3'):
93. scale = 3
94. **elif** k == ord('4'):
95. scale = 0
96. **elif** k == ord('m'):
97. mode = False **if** mode **else** True
98. **elif** k == ord('m'):
99. bag()
100. **elif** k == 27:  # ESC
101. **break**
102. cv2.destroyAllWindows()

calculate.py（计算信息）

1. **import** re
2. **import** cv2
3. **import** numpy as np
4. **import** os
5. **import** matplotlib.pyplot as plt
6. **from** scipy.special **import** comb, perm

9. **def** calculate\_2():
10. r\_3s = []
11. delta = 0.01
12. **for** ploy **in** datas:
13. n = 3
14. c\_x, c\_y = ploy[0]
15. r\_3\_sum = 0
16. remain = 0
17. dots = []
18. **for** i **in** range(len(ploy[1])):
19. x1, y1 = ploy[1][i]
20. x2, y2 = ploy[1][(i + 1) % len(ploy[1])]
21. **if** x1 == x2:
22. y1, x1 = ploy[1][i]
23. y2, x2 = ploy[1][(i + 1) % len(ploy[1])]
24. k = (y2 - y1) / (x2 - x1)
25. b = y2 - k \* x2
26. remain\_x = remain / (k \*\* 2 + 1)
27. delta\_x = delta / (k \*\* 2 + 1)
28. **for** j **in** range(round((np.math.sqrt((x1 - x2) \*\* 2 + (y1 - y2) \*\* 2)-remain) / delta - 0.5)):
29. dot\_x = x1 + remain\_x + j \* delta\_x
30. dot\_y = k \* dot\_x + b
31. dots.append([dot\_x - c\_x, dot\_y - c\_y])
32. remain = np.math.sqrt((x1 - x2) \*\* 2 + (y1 - y2) \*\* 2) % 0.1

35. **for** i **in** range(len(dots) - 1):
36. r\_3 = (dots[i][0]\*\*2 + dots[i][1]\*\*2)\*\*((n-1)/2)
37. r\_3 \*= np.math.sqrt(((dots[i][0]-dots[i+1][0])\*\*2 + (dots[i][1]-dots[i+1][1])\*\*2))
38. r\_3\_sum += r\_3
40. **print**(r\_3\_sum)
42. r\_3s.append(r\_3\_sum)

45. **def** calculate\_4():
46. r\_3s = []
47. m = 10000
48. **for** ploy **in** datas:
49. n = 4
50. c\_x, c\_y = ploy[0]
51. r\_3\_sum = 0
52. **for** i **in** range(len(ploy[1])):
53. x1, y1 = ploy[1][i]
54. x2, y2 = ploy[1][(i + 1) % len(ploy[1])]
55. l = np.math.sqrt((x2-x1)\*\*2 + (y2-y1)\*\*2)/m
56. **for** j **in** range(m):
57. dot\_x = x1 + (x2 - x1) / m \* (j+1/2) - c\_x
58. dot\_y = y1 + (y2 - y1) / m \* (j+1/2) - c\_y
59. r\_3\_sum += (dot\_y\*\*2 + dot\_x\*\*2)\*\*((n-1)/2)\*l
60. **print**(str(r\_3\_sum) + '\t' + str((r\_3\_sum / np.math.pi / 2) \*\* (1/n)))
61. r\_3s.append(r\_3\_sum)

64. **def** calculate\_5():
65. j = 2
66. r\_5s = []
67. **for** ploy **in** datas:
68. r\_5\_sum = 0
69. c\_x, c\_y = ploy[0]
70. **for** i **in** range(len(ploy[1])):
71. x1, y1 = ploy[1][i]
72. x2, y2 = ploy[1][(i + 1) % len(ploy[1])]
73. x1 -= c\_x
74. x2 -= c\_x
75. y1 -= c\_y
76. y2 -= c\_y
77. **if** x1 == x2:
78. temp = x1
79. x1 = y1
80. y1 = temp
81. temp = x2
82. x2 = y2
83. y2 = temp
84. c = x1 - x2
85. b = y1 - y2
86. a = np.math.sqrt(b\*\*2 + c\*\*2) / abs(c)\*\*(j\*2+1)
87. j\_sum = 0
88. **for** p **in** range(j+1):
89. q\_sum = 0
90. **for** q **in** range(2\*p+1):
91. q\_value = comb(2\*p, q)
92. q\_value \*= b\*\*q \* c\*\*(2\*j - 2\*p)
93. q\_value \*= (c\*y1 - b\*x1)\*\*(2\*p - q)
94. q\_value \*= x2\*\*(2\*j - 2\*p + q + 1) - x1\*\*(2\*j - 2\*p + q + 1)
95. q\_value /= 2\*j - 2\*p + q + 1
96. q\_sum += q\_value
97. j\_sum += comb(j, p) \* q\_sum
98. r\_5\_sum += abs(a\*j\_sum)
99. r\_5s.append(r\_5\_sum)
100. **print**(str(r\_5\_sum) + '\t' + str((r\_5\_sum / np.math.pi / 2) \*\* (1/(2\*j+1))))

103. **def** calculate\_3():
104. r\_3s = []
105. **for** ploy **in** datas:
106. r\_3\_sum = 0
107. c\_x, c\_y = ploy[0]
108. **for** i **in** range(len(ploy[1])):
109. x1, y1 = ploy[1][i]
110. x2, y2 = ploy[1][(i + 1) % len(ploy[1])]
111. x1 -= c\_x
112. x2 -= c\_x
113. y1 -= c\_y
114. y2 -= c\_y
115. **if** x1 == x2:
116. temp = x1; x1 = y1; y1 = temp
117. temp = x2; x2 = y2; y2 = temp
118. a = x1 - x2
119. b = y1 - y2
120. r3 = (a\*\*2+b\*\*2)\*(x2\*\*3-x1\*\*3)/3
121. r3 += 2\*(b)\*(x1\*y2 - x2\*y1)\*(x2\*\*2 - x1\*\*2)/2
122. r3 += (-a)\*(x1\*(b) + y1\*(-a))\*\*2
123. r3 \*= np.math.sqrt(b\*\*2+a\*\*2)/a\*\*3
124. r3 = abs(r3)
125. r\_3\_sum += r3
126. r\_3s.append(r\_3\_sum)
127. **print**(str(r\_3\_sum) + '\t' + str((r\_3\_sum / np.math.pi / 2) \*\* (1/3)))
128. # print((r\_3\_sum / np.math.pi) \*\* (1/3))

131. **def** add\_circle():
132. p\_num = 6
133. delta\_theta = 2 \* np.math.pi / p\_num
134. pts = []
135. **for** i **in** range(p\_num):
136. theta = i \* delta\_theta
137. x = 1 \* np.math.cos(theta)
138. y = 1 \* np.math.sin(theta)
139. pts.append([x, y])
140. datas.append([[0, 0], pts])

143. **def** add\_poly():
144. **for** p\_num **in** range(3,11):
145. delta\_theta = 2 \* np.math.pi / p\_num
146. pts = []
147. **for** i **in** range(p\_num):
148. theta = i \* delta\_theta
149. x = 1 \* np.math.cos(theta)
150. y = 1 \* np.math.sin(theta)
151. pts.append([x, y])
152. datas.append([[0, 0], pts])

155. **def** calculate\_1():
156. perimeters = []
157. areas = []
158. **for** ploy **in** datas:
159. perimeter = 0
160. **for** i **in** range(len(ploy[1])):
161. perimeter += np.math.sqrt((ploy[1][i][0] - ploy[1][(i + 1) % len(ploy[1])][0]) \*\* 2 +
162. (ploy[1][i][1] - ploy[1][(i + 1) % len(ploy[1])][1]) \*\* 2)
164. area = PolyArea(np.array(ploy[1]).T[0], np.array(ploy[1]).T[1])
165. perimeters.append(perimeter)
166. areas.append(area)
167. # print(perimeter, area)
169. **print**(str(perimeter) + '\t' + str(area))
171. res.append(perimeters)
172. res.append(areas)

175. **def** calculate\_va():
176. vs = []
177. **for** i **in** range(len(datas)-1):
178. x1, y1 = datas[i][0]
179. x2, y2 = datas[i+1][0]
180. v = np.math.sqrt((x2-x1)\*\*2 + (y2-y1)\*\*2)
181. vs.append(v)
182. **for** i **in** range(len(vs)-1):
183. v1 = vs[i]
184. v2 = vs[i+1]
185. a = v2 - v1
186. **print**(str(vs[i]) + '\t' + str(a))
187. **print**(vs[-1])

190. **def** draw():
191. plt.plot(range(497, 608), res[0], linewidth=2.0)
192. plt.show()
193. plt.plot(range(497, 608), res[1], linewidth=2.0)
194. plt.show()
195. **pass**

198. **def** PolyArea(x, y):
199. **return** 0.5 \* np.abs(np.dot(x, np.roll(y, 1)) - np.dot(y, np.roll(x, 1)))

202. **def** read\_file():
203. **for** file **in** os.listdir(root\_dir):
204. **if** re.match('.\*\.txt', file, flags=0):
205. proportion = 8 / 1022 **if** int(file.split('.')[0]) <= 519 **else** 8 / 1034
206. open\_filename = root\_dir + "\\" + file
207. with open(open\_filename, 'r') as fio:
208. lines = fio.readlines()
209. center = lines[0].split(':')[1]
210. center = center.split(',')
211. center = list((int(center[0][1:])\*proportion, int(center[1][:-2])\*proportion))
212. # print(str(center[0]) + '\t' + str(center[1]))
213. points = []
214. **for** line **in** lines[1:]:
215. point = re.split('edge:\*', line)[1]
216. point = point.split(',')
217. point = list((int(point[0][1:])\*proportion, int(point[1][:-2])\*proportion))
218. points.append(point)
219. datas.append([center, points])

222. **def** calculate\_r():
223. rjs = []
224. **for** j **in** range(0, 50):
225. r\_5s = []
226. **for** ploy **in** datas:
227. r\_5\_sum = 0
228. c\_x, c\_y = ploy[0]
229. **for** i **in** range(len(ploy[1])):
230. x1, y1 = ploy[1][i]
231. x2, y2 = ploy[1][(i + 1) % len(ploy[1])]
232. x1 -= c\_x
233. x2 -= c\_x
234. y1 -= c\_y
235. y2 -= c\_y
236. **if** abs(x1 - x2) < 0.01:
237. temp = x1
238. x1 = y1
239. y1 = temp
240. temp = x2
241. x2 = y2
242. y2 = temp
243. c = x1 - x2
244. b = y1 - y2
245. a = np.math.sqrt(b \*\* 2 + c \*\* 2) / abs(c) \*\* (j \* 2 + 1)
246. j\_sum = 0
247. **for** p **in** range(j + 1):
248. q\_sum = 0
249. **for** q **in** range(2 \* p + 1):
250. q\_value = comb(2 \* p, q)
251. q\_value \*= b \*\* q \* c \*\* (2 \* j - 2 \* p)
252. q\_value \*= (c \* y1 - b \* x1) \*\* (2 \* p - q)
253. q\_value \*= x2 \*\* (2 \* j - 2 \* p + q + 1) - x1 \*\* (2 \* j - 2 \* p + q + 1)
254. q\_value /= 2 \* j - 2 \* p + q + 1
255. q\_sum += q\_value
256. j\_sum += comb(j, p) \* q\_sum
257. r\_5\_sum += abs(a \* j\_sum)
258. r\_5s.append((r\_5\_sum / np.math.pi / 2) \*\* (1 / (2 \* j + 1)))
259. rjs.append(r\_5s)
261. **for** i **in** range(len(rjs[0])):
262. string = ''
263. **for** j **in** range(len(rjs)):
264. string += str(rjs[j][i]) + '\t'
265. **print**(string)

268. **if** \_\_name\_\_ == '\_\_main\_\_':
270. root\_dir = r".\data\_2"
271. datas = []
272. # datas.append([[4/3,1],[[0,3],[0,0],[4,0]]])
273. # datas.append([[0,1/3],[[-1,0],[0,1],[1,0]]])
274. # datas.append([[2/3,1/3],[[0,0],[1,1],[1,0]]])
275. # datas.append([[1, 1],[[0,0],[2,0],[2,2],[0,2]]])
276. # datas.append([[0, 0], [[-1, -1], [1, -1], [1, 1], [-1, 1]]])
277. res = []
278. read\_file()
279. # add\_circle()
280. # add\_poly()
281. # calculate\_1()
283. # calculate\_2()
284. # calculate\_3()
285. # calculate\_4()
286. # calculate\_5()
287. # calculate\_r()
288. calculate\_va()

291. # draw()