主要工作：运用激光雷达模块跟踪物体

1.目标物检测

由于激光雷达返回的数据有噪声存在，先用均值滤波对数据进行预处理。运用最近邻聚类算法，将激光雷达探测到的点云聚类成不同的物体。根据目标物的长度和反射强度特性，将检测到的所有物体与目标物体相匹配，如果匹配成功则获取它相对于小车的角度和距离。如果场景中不存在目标物则继续匹配。



图1：激光雷达点云图

2.小车运动策略

在获取到目标物相对于小车的角度和距离之后。优先对角度进行判断，若目标物不在小车的正前方，则让小车旋转，直至目标物出现在正前方。随后进行距离判断，由于竞赛过程中要对目标物进行抓取，因此小车的停止点必须和目标物保持固定的距离。如果检测到的距离大于这个阈值，则让小车前行，否则小车停止。

3.程序代码

1. #include <ros/ros.h>
2. #include <sensor\_msgs/LaserScan.h>
3. #include <geometry\_msgs/Twist.h>
4. #include <wiringPi.h>
5. #include <math.h>
7. #define pi 3.141592653589
9. ros::Publisher cmdVelPub;
10. geometry\_msgs::Twist speed;
11. **int** stop\_time = 0;
13. **void** run\_forward();
14. **void** run\_stop();
15. **void** turn\_angle\_left(**float** angle);
16. **void** turn\_angle\_right(**float** angle);
17. **void** my\_callback(**const** sensor\_msgs::LaserScan::ConstPtr& msg);
19. **int** main(**int** argc, **char**\*\* argv)
20. {
21. wiringPiSetup();
22. ros::init(argc, argv, "find\_bench");
23. ros::NodeHandle nh;
24. cmdVelPub = nh.advertise<geometry\_msgs::Twist>("cmd\_vel", 1);
26. ros::Subscriber scan\_sub = nh.subscribe("scan", 1, my\_callback);
27. ros::spin();
28. **return** 0;
29. }
31. **void** run\_forward()
32. {
33. speed.linear.x = 0.05;
34. speed.angular.z = 0;
35. cmdVelPub.publish(speed);
36. }
38. **void** run\_stop()
39. {
40. speed.linear.x = 0;
41. speed.angular.z = 0;
42. cmdVelPub.publish(speed);
43. }
45. **void** turn\_angle\_left(**float** angle)
46. {
47. **float** time = angle / 18.0;
48. speed.linear.x = 0;
49. speed.angular.z = 0.3;
50. cmdVelPub.publish(speed);
51. ros::Duration(time).sleep();
52. speed.angular.z = 0;
53. cmdVelPub.publish(speed);
54. ros::Duration(2).sleep();
55. }
57. **void** turn\_angle\_right(**float** angle)
58. {
59. **float** time = angle / 18.0;
60. speed.linear.x = 0;
61. speed.angular.z = -0.3;
62. cmdVelPub.publish(speed);
63. ros::Duration(time).sleep();
64. speed.angular.z = 0;
65. cmdVelPub.publish(speed);
66. ros::Duration(2).sleep();
67. }
69. **void** my\_callback(**const** sensor\_msgs::LaserScan::ConstPtr& msg)
70. {
71. **int** i;
72. **float** ranges[360];
73. **for**(i = 0; i < 360; i++)
74. ranges[i] = msg -> ranges[i];
75. **for**(i = 0; i < 360; i++)//预处理
76. {
77. **int** previous, next;
78. **if**(ranges[i] < 0.1)
79. {
80. previous = i - 1;
81. **if**(previous < 0)
82. previous = 359;
83. next = i + 1;
84. **if**(next > 359)
85. next = 0;
86. ranges[i] = (ranges[previous] + ranges[next]) / 2;
87. }
88. }
90. **int**\* classify = **new** **int**[360];//存放聚类的类别
91. **int** num = 1;//存放类的总数
92. **int** temp\_num = 1;//存放每一个目标的长度
93. **int**\* class\_len = **new** **int**[360];//存放每一类数组的长度
94. **float**\* object\_len = **new** **float**[360];//存放每个物体的长度
95. **int** center\_point\_angle[360];//存储每个目标中间点的角度
96. **float** center\_point\_distance[360];//存储每个目标中间点的距离
97. **float** temp0;
98. **float** temp1 = ranges[0];
99. **float** temp\_len;
100. classify[0] = 1;
102. **for**(i = 1; i < 360; i++)//最近邻聚类并存储每一类的长度
103. {
104. temp0 = temp1;
105. temp1 = ranges[i];
106. temp\_len = abs(temp1 - temp0);
107. **if**(temp\_len > 0.1)
108. {
109. class\_len[num - 1] = temp\_num;
110. num++;
111. temp\_num = 0;
112. }
113. temp\_num = temp\_num + 1;
114. classify[i] = num;
115. }
116. class\_len[num - 1] = temp\_num;//存储最后一个目标的长度
117. //将首尾的类别进行合并
118. **if**(abs(ranges[359] - ranges[0]) <= 0.1)//首尾合并的情况
119. {
120. **int** j;
121. **for**(j = 360 - class\_len[num - 1]; j < 360; j++)
122. classify[j] = 1;
124. //计算第1个目标的长度
125. **float** temp\_angle = 0;
126. **float** start\_side = 0;
127. **float** end\_side = 0;
128. temp\_angle = (**float**)(class\_len[0] + class\_len[num - 1] - 1) / 180.0 \* pi;
129. start\_side = ranges[360 - class\_len[num - 1]];
130. end\_side = ranges[class\_len[0] - 1];
131. object\_len[0] = sqrt(pow(start\_side, 2) + pow(end\_side, 2) - 2 \* start\_side \* end\_side \* cos(temp\_angle));
132. center\_point\_angle[0] = (class\_len[0] - class\_len[num - 1]) / 2;//中心点
133. **if**(center\_point\_angle[0] < 0)
134. center\_point\_angle[0] = 359 + center\_point\_angle[0];
135. center\_point\_distance[0] = ranges[center\_point\_angle[0]];
136. //ROS\_INFO("The 0 angle is %f, start is %f, end is %f, len is %f", temp\_angle, start\_side, end\_side, object\_len[0]);
137. num = num - 1;
139. //依次计算后续目标的长度
140. **int** len\_all = 0;//存放前方所有目标的数组长度
141. **for**(j = 1; j < num; j++)//存放每个目标的长度和中心点
142. {
143. len\_all = len\_all + class\_len[j - 1];
144. temp\_angle = (**float**)(class\_len[j] - 1) / 180.0 \* pi;
145. start\_side = ranges[len\_all];
146. end\_side = ranges[len\_all + class\_len[j] - 1];
147. object\_len[j] = sqrt(pow(start\_side, 2) + pow(end\_side, 2) - 2 \* start\_side \* end\_side \* cos(temp\_angle));
148. center\_point\_angle[j] = (2 \* len\_all + class\_len[j]) / 2;
149. center\_point\_distance[j] = ranges[center\_point\_angle[j]];
150. }
152. class\_len[0] = class\_len[0] + class\_len[num];
153. class\_len[num] = 0;
154. }
155. **else**//首尾不用合并的情况
156. {
157. **float** temp\_angle = 0;
158. **float** start\_side = 0;
159. **float** end\_side = 0;
160. **int** len\_all = 0;
161. **int** j;
162. **for**(j = 0; j < num; j++)//存放每个目标的长度和中心点
163. {
164. temp\_angle = (**float**)(class\_len[j] - 1) / 180.0 \* pi;
165. start\_side = ranges[len\_all];
166. end\_side = ranges[len\_all + class\_len[j] - 1];
167. object\_len[j] = sqrt(pow(start\_side, 2) + pow(end\_side, 2) - 2 \* start\_side \* end\_side \* cos(temp\_angle));
168. center\_point\_angle[j] = (2 \* len\_all + class\_len[j]) / 2;
169. center\_point\_distance[j] = ranges[center\_point\_angle[j]];
170. len\_all = len\_all + class\_len[j];
171. }
172. }
174. //for(i = 0; i < num; i++)
175. //  ROS\_INFO("The %d length is %f", i, object\_len[i]);
176. **if**(object\_len[0] >= 0.2 && object\_len[0] <= 0.22)
177. {
178. stop\_time = 0;
179. **if**(center\_point\_distance[0] <= 0.2)//不用行走的情况
180. {
181. **if**(ros::ok())
182. run\_stop();
183. }
184. **else**//需要行走的情况
185. {
186. **if**(ros::ok())
187. run\_forward();
188. }
189. }
190. **else** **if**(stop\_time <= 2)
191. stop\_time = stop\_time + 1;
192. **else**
193. {
194. stop\_time = 0;
195. **for**(i = 0; i < num; i++)
196. {
197. **if**(object\_len[i] >= 0.2 && object\_len[i] <= 0.22 && msg -> intensities[center\_point\_angle[i]] > 1000 && msg -> intensities[center\_point\_angle[i]] < 7000)//找到目标
198. {
199. **if**(center\_point\_angle[i] <= 10 || center\_point\_angle[i] >= 350)//不用旋转的情况
200. {
201. **if**(center\_point\_distance[i] <= 0.2)//不用行走的情况
202. {
203. **if**(ros::ok())
204. run\_stop();
205. }
206. **else**//需要行走的情况
207. {
208. **if**(ros::ok())
209. run\_forward();
210. }
211. }
212. **else**//需要旋转的情况
213. {
214. **if**(center\_point\_angle[i] <= 180)//左转
215. {
216. **if**(ros::ok())
217. turn\_angle\_left((**float**)center\_point\_angle[i]);
218. }
219. **else**
220. {
221. **if**(ros::ok())
222. turn\_angle\_right((**float**)(360 - center\_point\_angle[i]));
223. }
224. }
225. //ROS\_INFO("the object is %d", i);
226. //ROS\_INFO("The length is %f, the intensity is %f", object\_len[center\_point\_angle[i]], msg -> intensities[center\_point\_angle[i]]);
227. **break**;
228. }
229. }
230. }
232. //ROS\_INFO("the center angle is %d, the center distance is %f", center\_point\_angle[0], center\_point\_distance[0]);
233. //ROS\_INFO("The %d length is %f", 0, object\_len[0]);
234. //ROS\_INFO("The forward distance is %f", msg -> ranges[0]);
235. //ROS\_INFO("The forward intensity is %f", msg -> intensities[0]);
236. //ROS\_INFO("The num is %d", num);
237. //ROS\_INFO("######################");

240. /\*
241. float forward\_distance = msg -> ranges[0];
242. int i = 0;
243. if(forward\_distance != 0 && forward\_distance < 0.3)
244. {
245. if(ros::ok())
246. run\_forward();
247. }
248. else
249. {
250. if(ros::ok())
251. run\_stop();
252. }
253. \*/
254. }

4.演示视频