

SHANGHAI JIAO TONG UNIVERSITY



多机器人系统与控制

编队选择避障

报告名称： 多机器人系统与控制实验四报告

小组编号： 第二组

组员姓名： 陈炜昊 王喆隆

易子淇 章雨悠

指导教师:  熊振华 董伟 吴建华

2021年12月

* **编队选择避障**
* **实验目的和要求**
* **基本任务**

控制不少于5台机器人的两种避障队形；假设每个机器人都知道有两种队形可以穿越队形，请编写程序实验预期的队形选择；分析实验成功或者失败的原因。

* **附加任务**

1. 增加控制的机器人数量并实现功能
2. 增加可供选择的避障队形数量并实现控制

* **算法设计思路**

整个编队行进避障任务被分解为5个阶段：

1. 选择并形成初始队形（十字，楔形，圆形）
2. 保持初始队形，靠近障碍物
3. 选择并形成避障队形（总共设计三种避障队形：一列，两列，三列）
4. 保持避障队形行进穿越障碍物
5. 选择并形成最终队形（十字，楔形，圆形）

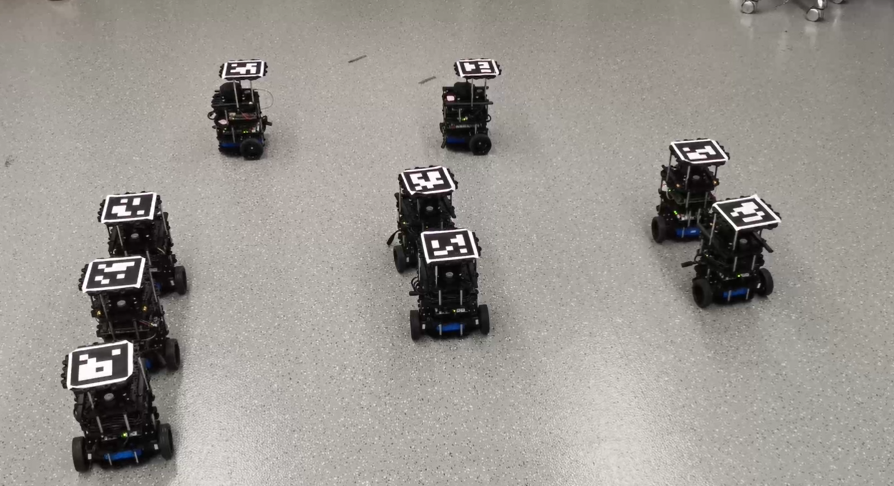
其中设计的队形对机器人的数量没有限制，理论上可以添加任意多台机器人进行编队。

队形选择、编队控制与实验3相同。

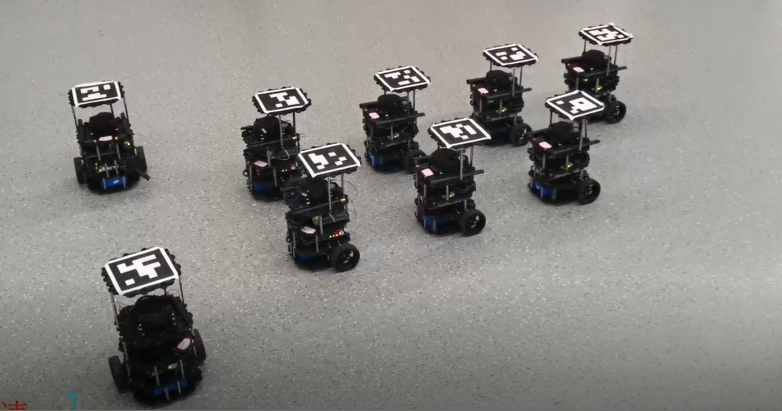
* **实验效果**

见视频。

**三列避障队形**



* **两列避障队形**



* **实验分析**

本次实验总体上完成效果较好，可以改进的细节有两点：

1. 在编队前进的时候进行速度一致性控制，减小行进过程中的队形误差。
2. 调节控制增益与判断收敛的阈值，可以缩短队形收敛的时间。

* **附录：实验代码**

1. \* Date: 2021-12-16
2. \* Description: Formation selection obstacle avoidance
3. \* 增加机器人数量只需修改编号，分布式队形选择
4. \*/
6. #include <swarm\_robot\_control.h>
7. #include <cmath>
8. #include <algorithm>
10. #define pi 3.1415926
12. /\*\*\*\*\*\*\*\*\*\*\*add by group\*\*\*\*\*\*\*\*\*/
13. /\* 集群初始化 \*/
14. std::vector<**int**> swarm\_robot\_id{1, 2, 3, 4, 5, 6};         // 集群机器人编号
15. std::vector<**int**> obstacle\_id{9, 10};                       // 障碍机器人编号
16. /\* 基本参数设定 \*/
17. **double** k\_w = 0.3;               // 角速度系数
18. **double** k\_v = 0.1;               // 线速度系数
19. **double** MAX\_W = 1;               // 角速度最大值
20. **double** MIN\_W = 0.05;            // 角速度最小值
21. **double** MAX\_V = 0.2;             // 线速度最大值
22. **double** MIN\_V = 0.01;            // 线速度最小值
23. **double** Delta = 2.5;             // 预设通讯范围
24. Eigen::MatrixXd lap(swarm\_robot\_id.size(), swarm\_robot\_id.size());  // 拉普拉斯矩阵
25. Eigen::MatrixXd L\_x(swarm\_robot\_id.size(), swarm\_robot\_id.size());  // 连通保持x矩阵
26. Eigen::MatrixXd L\_y(swarm\_robot\_id.size(), swarm\_robot\_id.size());  // 连通保持y矩阵
27. std::vector<std::vector<**double**> > current\_robot\_pose(swarm\_robot\_id.size());
28. Eigen::VectorXd cur\_x(swarm\_robot\_id.size());    // 机器人当前x方向位置
29. Eigen::VectorXd cur\_y(swarm\_robot\_id.size());    // 机器人当前y方向位置
30. Eigen::VectorXd cur\_theta(swarm\_robot\_id.size());// 机器人当前偏航角
31. Eigen::VectorXd del\_x(swarm\_robot\_id.size());
32. Eigen::VectorXd del\_y(swarm\_robot\_id.size());
33. Eigen::VectorXd del\_theta(swarm\_robot\_id.size());
34. /\* 队形变换参数设置 \*/
35. **int** formationA\_type = 3;            // 行进编队种类
36. **int** formationB\_type = 3;            // 避障编队种类
37. **double** k\_zeta = 0.2;                // 整体编队代价估计量一致性系数
38. **double** k\_xi = 0.05;                 // 局部编队代价实时量系数
39. **double** characteristic\_1 = 0.6;      // 行进编队特征长度（圆形）
40. **double** characteristic\_2 = 0.3;      // 行进编队特征长度（星型及楔形）
41. **double** characteristic\_3;            // 避障编队特征长度（单列及双列），自主根据障碍物宽度进行调整
42. **double** characteristic\_4;            // 避障编队特征长度（三列），自主根据障碍物宽度进行调整
43. **double** threshold = 0.16;            // 避障阈值
44. **double** Formation\_centor[2];         // 队形中心位置
45. Eigen::VectorXi target\_formation(swarm\_robot\_id.size());     // 各机器人选择编队序号
46. Eigen::VectorXd pic\_x(swarm\_robot\_id.size());                // pic\_x = cur\_x - Formation\_x
47. Eigen::VectorXd pic\_y(swarm\_robot\_id.size());                // pic\_y = cur\_y - Formation\_y
49. /\* PID-control yaw \*/
50. **void** yaw\_control(SwarmRobot& swarm, **int** index, **double** yaw\_error){
51. **double** w = swarm.checkVel(k\_w\*yaw\_error, MAX\_W, MIN\_W);
52. swarm.moveRobot(index, 0.0, w);
53. }
54. /\* PID 位置控制 \*/
55. **void** pos\_control(SwarmRobot& swarm\_robot, **int** index, **double** x\_error, **double** y\_error, **double** yaw){
56. **double** w = atan(y\_error/x\_error);
57. **double** v = sqrt(x\_error\*x\_error + y\_error\*y\_error);
58. **if** (x\_error < 0){
59. **if** (y\_error > 0){
60. w += pi;
61. }
62. **else** {
63. w -= pi;
64. }
65. }
66. w = w - yaw;
67. v \*= pow(cos(w), 3);
68. v \*= k\_v;
69. w \*= k\_w;
70. v = swarm\_robot.checkVel(v, MAX\_V, MIN\_V);
71. w = swarm\_robot.checkVel(w, MAX\_W, MIN\_W);
72. swarm\_robot.moveRobot(index, v, w);
73. }
74. /\* 获取各机器人位置 \*/
75. **void** get\_pos(SwarmRobot& swarm\_robot){
76. swarm\_robot.getRobotPose(current\_robot\_pose);
77. Formation\_centor[0] = Formation\_centor[1] = 0;
78. **for**(**int** i = 0; i < swarm\_robot\_id.size(); i++) {
79. cur\_x(i) = current\_robot\_pose[i][0];
80. Formation\_centor[0] += cur\_x(i);
81. cur\_y(i) = current\_robot\_pose[i][1];
82. Formation\_centor[1] += cur\_y(i);
83. cur\_theta(i) = current\_robot\_pose[i][2];
84. }
85. Formation\_centor[0] /= swarm\_robot\_id.size();
86. Formation\_centor[1] /= swarm\_robot\_id.size();
87. }
88. /\* 更新队形实时量 \*/
89. **void** update\_formation\_cost(**const** **int** formation\_num, Eigen::MatrixXd& Formation\_x, Eigen::MatrixXd& Formation\_y, Eigen::MatrixXd& xi){
90. **for** (**int** i = 0; i < formation\_num; ++i){
91. **for** (**int** j = 0; j < swarm\_robot\_id.size(); ++j){
92. **double** tmp = 0;
93. **for** (**int** k = 0; k < swarm\_robot\_id.size(); ++k){
94. **if** (k == j){
95. **continue**;
96. }
97. tmp += sqrt(pow(cur\_x(j) - cur\_x(k) - Formation\_x(j, i) + Formation\_x(k, i), 2) + pow(cur\_y(j) - cur\_y(k) - Formation\_y(j, i) + Formation\_y(k, i), 2));
98. }
99. xi(j, i) = tmp;
100. }
101. }
102. }
103. /\* 迭代队形估计量 \*/
104. **void** conv\_formation\_cost(**const** **int** formation\_num, Eigen::MatrixXd& Formation\_x, Eigen::MatrixXd& Formation\_y, Eigen::MatrixXd& zeta, Eigen::MatrixXd& xi){
105. update\_formation\_cost(formation\_num, Formation\_x, Formation\_y, xi);
106. **for** (**int** i = 0; i < formation\_num; ++i){
107. zeta.block(0, i, swarm\_robot\_id.size(), 1) += k\_xi\*(xi.block(0, i, swarm\_robot\_id.size(), 1) - zeta.block(0, i, swarm\_robot\_id.size(), 1)) - k\_zeta\*lap\*zeta.block(0, i, swarm\_robot\_id.size(), 1);
108. }
109. }
110. /\* 队形选择 \*/
111. **void** formation\_selection(**const** **int** formation\_num, Eigen::MatrixXd& zeta){
112. **for** (**int** i = 0; i < swarm\_robot\_id.size(); ++i){
113. **double** min\_cost = 1e+9;
114. **for** (**int** j = 0; j < formation\_num; ++j){
115. **if** (zeta(i, j) < min\_cost){
116. target\_formation(i) = j;
117. min\_cost = zeta(i, j);
118. }
119. }
120. }
121. std::cout << "当前各机器人队形选择为: ";
122. **for** (**int** i =0; i < swarm\_robot\_id.size(); ++i){
123. std::cout << target\_formation[0] << " ";
124. }
125. std::cout << std::endl;
126. }
127. /\* 队形控制量 \*/
128. **void** cal\_formation(**const** **int** formation\_num, Eigen::MatrixXd& Formation\_x, Eigen::MatrixXd& Formation\_y){
129. **for** (**int** i = 0; i < swarm\_robot\_id.size(); ++i){
130. pic\_x(i) = cur\_x(i) - Formation\_x(i, target\_formation(i));
131. pic\_y(i) = cur\_y(i) - Formation\_y(i, target\_formation(i));
132. }
133. **for** (**int** i = 0; i < swarm\_robot\_id.size(); ++i){
134. **for** (**int** j = 0; j < swarm\_robot\_id.size(); ++j){
135. **if** (j == i){
136. **continue**;
137. }
138. L\_x(i, j) = L\_x(j, i) = -2\*((Delta - std::fabs(Formation\_x(i, target\_formation(i)) - Formation\_x(j, target\_formation(j)))) - std::fabs(pic\_x(i) - pic\_x(j)))/\
139. pow((Delta - std::fabs(Formation\_x(i, target\_formation(i)) - Formation\_x(j, target\_formation(j))) - std::fabs(pic\_x(i) - pic\_x(j))), 2);
140. }
141. }
142. **for** (**int** i = 0; i < swarm\_robot\_id.size(); ++i){
143. **for** (**int** j = 0; j < swarm\_robot\_id.size(); ++j){
144. **if** (j == i){
145. **continue**;
146. }
147. L\_y(i, j) = L\_y(j, i) = -2\*((Delta - std::fabs(Formation\_y(i, target\_formation(i)) - Formation\_y(j, target\_formation(j)))) - std::fabs(pic\_y(i) - pic\_y(j)))/\
148. pow((Delta - std::fabs(Formation\_y(i, target\_formation(i)) - Formation\_y(j, target\_formation(j))) - std::fabs(pic\_y(i) - pic\_y(j))), 2);
149. }
150. }
151. **for** (**int** i = 0; i < swarm\_robot\_id.size(); ++i){
152. **double** tmp1 = 0;
153. **double** tmp2 = 0;
154. **for** (**int** j = 0; j < swarm\_robot\_id.size(); ++j){
155. **if** (j == i){
156. **continue**;
157. }
158. tmp1 -= L\_x(i, j);
159. tmp2 -= L\_y(i, j);
160. }
161. L\_x(i, i) = tmp1;
162. L\_y(i, i) = tmp2;
163. }
164. del\_x = -L\_x \* pic\_x;
165. del\_y = -L\_y \* pic\_y;
166. }
167. **int** main(**int** argc, **char**\*\* argv) {
168. /\* 节点初始化 \*/
169. ros::init(argc, argv, "swarm\_robot\_control\_formation");
170. ros::NodeHandle nh;
171. SwarmRobot swarm\_robot(&nh, swarm\_robot\_id);
172. SwarmRobot obstacle(&nh, obstacle\_id);
173. std::vector<std::vector<**double**> > obstacle\_pose(obstacle\_id.size());
174. **for** (**int** i = 0; i < swarm\_robot\_id.size(); ++i){
175. **for** (**int** j = 0; j < swarm\_robot\_id.size(); ++j){
176. lap(i, j) = -1;
177. }
178. }
179. **for** (**int** i = 0; i < swarm\_robot\_id.size(); ++i){
180. lap(i, i) = swarm\_robot\_id.size() - 1;
181. }
183. /\* 基本参数设定 \*/
184. **int** array\_flag[swarm\_robot\_id.size()];                   // 角色分配标记
185. Eigen::MatrixXd circle(swarm\_robot\_id.size(), 2);        // 圆形编队相对位置 [x y]
186. Eigen::MatrixXd star(swarm\_robot\_id.size(), 2);          // 星形编队相对位置
187. Eigen::MatrixXd wedge(swarm\_robot\_id.size(), 2);         // 楔形编队相对位置
188. Eigen::MatrixXd colume1(swarm\_robot\_id.size(), 2);       // 单路纵队相对位置 [x y]
189. Eigen::MatrixXd colume2(swarm\_robot\_id.size(), 2);       // 两路纵队相对位置
190. Eigen::MatrixXd colume3(swarm\_robot\_id.size(), 2);       // 三路纵队相对位置
191. **double** obstacle\_centor[2];                               // 障碍物中心位置
192. **int** role\_assignment = 1;                                 // 角色分配种类
193. **for** (**int** i = 0; i < swarm\_robot\_id.size(); ++i){
194. role\_assignment \*= (i + 1);
195. array\_flag[i] = i;
196. }
197. **int** formationA\_num = formationA\_type\*role\_assignment;
198. **int** formationB\_num = formationB\_type\*role\_assignment;
199. **double** conv\_th = 0.03\*swarm\_robot\_id.size();             // 角度收敛阈值
200. **double** conv\_pos = 0.03\*swarm\_robot\_id.size();            // 位置收敛阈值
201. **bool** angle\_conv = **false**;           // 角度收敛标记
202. **bool** pos\_conv = **false**;             // 位置收敛标记
204. /\* 获取障碍物信息 \*/
205. obstacle.getRobotPose(obstacle\_pose);
206. obstacle\_centor[0] = (obstacle\_pose[0][0] + obstacle\_pose[1][0])/2;
207. obstacle\_centor[1] = (obstacle\_pose[0][1] + obstacle\_pose[1][1])/2;
208. characteristic\_3 = std::fabs(obstacle\_pose[0][1] - obstacle\_pose[1][1])/2 - threshold - 0.1;
209. characteristic\_4 = std::fabs(obstacle\_pose[0][1] - obstacle\_pose[1][1]);
211. /\* 队形信息设定 \*/
212. **for** (**int** i = 0; i < swarm\_robot\_id.size(); ++i){
213. circle(i, 0) = characteristic\_1\*cos(i\*2\*pi/swarm\_robot\_id.size());
214. circle(i, 1) = characteristic\_1\*sin(i\*2\*pi/swarm\_robot\_id.size());
215. }
216. **for** (**int** i = 0; i < swarm\_robot\_id.size(); ++i){
217. star(i, 0) = characteristic\_2\*cos((i - 1)\*pi/2)\***int**((i + 3)/4);
218. star(i, 1) = characteristic\_2\*sin((i - 1)\*pi/2)\***int**((i + 3)/4);
219. }
220. **for** (**int** i = 0; i < swarm\_robot\_id.size(); ++i){
221. wedge(i, 0) = characteristic\_2\*cos(-2\*pi/3)\***int**((i + 1)/2);
222. wedge(i, 1) = characteristic\_2\*sin(-2\*pi/3)\*pow(-1, i)\***int**((i + 1)/2);
223. }
224. **for** (**int** i = 0; i < swarm\_robot\_id.size(); ++i){
225. colume1(i, 0) = characteristic\_2\*i;
226. colume1(i, 1) = 0;
227. }
228. **for** (**int** i = 0; i < swarm\_robot\_id.size(); ++i){
229. colume2(i, 0) = characteristic\_3\***int**(i/2);
230. colume2(i, 1) = characteristic\_3\*pow(-1, **int**(i/2));
231. }
232. **for** (**int** i = 0; i < swarm\_robot\_id.size(); ++i){
233. colume3(i, 0) = characteristic\_2\***int**(i/3);
234. colume3(i, 1) = characteristic\_4\*(i%3 - 1);
235. }
236. Eigen::MatrixXd FormationA\_x(swarm\_robot\_id.size(), formationA\_num);
237. Eigen::MatrixXd FormationA\_y(swarm\_robot\_id.size(), formationA\_num);
238. Eigen::MatrixXd FormationB\_x(swarm\_robot\_id.size(), formationB\_num);
239. Eigen::MatrixXd FormationB\_y(swarm\_robot\_id.size(), formationB\_num);
240. Eigen::MatrixXd zeta\_A(swarm\_robot\_id.size(), formationA\_num);  // 队形代价估计数值
241. Eigen::MatrixXd xi\_A(swarm\_robot\_id.size(), formationA\_num);    // 队形代价实时数值
242. Eigen::MatrixXd zeta\_B(swarm\_robot\_id.size(), formationB\_num);  // 队形代价估计数值
243. Eigen::MatrixXd xi\_B(swarm\_robot\_id.size(), formationB\_num);    // 队形代价实时数值
244. **for** (**int** i = 0; i < role\_assignment; ++i){
245. **for** (**int** j = 0; j < swarm\_robot\_id.size(); ++j){
246. FormationA\_x(j, i) = circle(array\_flag[j], 0) - circle(array\_flag[0], 0);
247. FormationA\_y(j, i) = circle(array\_flag[j], 1) - circle(array\_flag[0], 1);
248. FormationA\_x(j, i + role\_assignment) = star(array\_flag[j], 0) - star(array\_flag[0], 0);
249. FormationA\_y(j, i + role\_assignment) = star(array\_flag[j], 1) - star(array\_flag[0], 1);
250. FormationA\_x(j, i + 2\*role\_assignment) = wedge(array\_flag[j], 0) - wedge(array\_flag[0], 0);
251. FormationA\_y(j, i + 2\*role\_assignment) = wedge(array\_flag[j], 1) - wedge(array\_flag[0], 1);
252. FormationB\_x(j, i) = colume1(array\_flag[j], 0) - colume1(array\_flag[0], 0);
253. FormationB\_y(j, i) = colume1(array\_flag[j], 1) - colume1(array\_flag[0], 1);
254. FormationB\_x(j, i + role\_assignment) = colume2(array\_flag[j], 0) - colume2(array\_flag[0], 0);
255. FormationB\_y(j, i + role\_assignment) = colume2(array\_flag[j], 1) - colume2(array\_flag[0], 1);
256. FormationB\_x(j, i + 2\*role\_assignment) = colume3(array\_flag[j], 0) - colume3(array\_flag[0], 0);
257. FormationB\_y(j, i + 2\*role\_assignment) = colume3(array\_flag[j], 1) - colume3(array\_flag[0], 1);
258. }
259. std::next\_permutation(array\_flag, array\_flag + swarm\_robot\_id.size());
260. }
262. /\* 形成 A 队形 \*/
263. get\_pos(swarm\_robot);
264. update\_formation\_cost(formationA\_num, FormationA\_x, FormationA\_y, xi\_A);
265. zeta\_A = xi\_A;
266. **while**(! pos\_conv) {
267. get\_pos(swarm\_robot);
268. conv\_formation\_cost(formationA\_num, FormationA\_x, FormationA\_y, zeta\_A, xi\_A);
269. formation\_selection(formationA\_num, zeta\_A);
270. cal\_formation(formationA\_num, FormationA\_x, FormationA\_y);
271. pos\_conv = **true**;
272. **if**(std::fabs(del\_x.norm()) > conv\_pos || std::fabs(del\_y.norm()) > conv\_pos) {
273. pos\_conv = **false**;
274. }
275. **for**(**int** i = 0; i < swarm\_robot\_id.size(); i++) {
276. pos\_control(swarm\_robot, i, del\_x(i), del\_y(i), cur\_theta(i));
277. }
278. ros::Duration(0.05).sleep();
279. }
280. **while**(!angle\_conv){
281. get\_pos(swarm\_robot);
282. del\_theta = -lap \* cur\_theta;
283. angle\_conv = **true**;
284. **if**(std::fabs(del\_theta.norm()) > conv\_th) {
285. angle\_conv = **false**;
286. }
287. **for**(**int** i = 0; i < swarm\_robot\_id.size(); i++) {
288. yaw\_control(swarm\_robot, i, del\_theta(i));
289. }
290. }
291. ROS\_INFO\_STREAM("Formation A got!");
292. pos\_conv = **false**;
293. angle\_conv = **false**;
295. /\* 靠近障碍物 \*/
296. get\_pos(swarm\_robot);
297. update\_formation\_cost(formationA\_num, FormationA\_x, FormationA\_y, xi\_A);
298. zeta\_A = xi\_A;
299. **while**(! pos\_conv) {
300. get\_pos(swarm\_robot);
301. conv\_formation\_cost(formationA\_num, FormationA\_x, FormationA\_y, zeta\_A, xi\_A);
302. formation\_selection(formationA\_num, zeta\_A);
303. cal\_formation(formationA\_num, FormationA\_x, FormationA\_y);
304. **double** error\_x = obstacle\_centor[0] - Formation\_centor[0] - 2\*characteristic\_1;
305. **double** error\_y = obstacle\_centor[1] - Formation\_centor[1];
306. **for** (**int** i = 0; i < swarm\_robot\_id.size(); ++i){
307. del\_x(i) += error\_x;
308. del\_y(i) += error\_y;
309. }
310. pos\_conv = **true**;
311. **if**(std::fabs(del\_x.norm()) > conv\_pos || std::fabs(del\_y.norm()) > conv\_pos) {
312. pos\_conv = **false**;
313. }
314. **for**(**int** i = 0; i < swarm\_robot\_id.size(); i++) {
315. pos\_control(swarm\_robot, i, del\_x(i), del\_y(i), cur\_theta(i));
316. }
317. ros::Duration(0.05).sleep();
318. }
319. **while**(!angle\_conv){
320. get\_pos(swarm\_robot);
321. cal\_formation(formationA\_num, FormationA\_x, FormationA\_y);
322. angle\_conv = **true**;
323. **if**(std::fabs(cur\_theta.norm()) > conv\_th) {
324. angle\_conv = **false**;
325. }
326. **for**(**int** i = 0; i < swarm\_robot\_id.size(); i++) {
327. yaw\_control(swarm\_robot, i, -cur\_theta(i));
328. }
329. }
330. ROS\_INFO\_STREAM("Warning, obstacle!");
331. pos\_conv = **false**;
332. angle\_conv = **false**;
334. /\* 形成 B 队形 \*/
335. get\_pos(swarm\_robot);
336. update\_formation\_cost(formationB\_num, FormationB\_x, FormationB\_y, xi\_B);
337. zeta\_B = xi\_B;
338. **while**(! pos\_conv) {
339. get\_pos(swarm\_robot);
340. conv\_formation\_cost(formationB\_num, FormationB\_x, FormationB\_y, zeta\_B, xi\_B);
341. formation\_selection(formationB\_num, zeta\_B);
342. cal\_formation(formationB\_num, FormationB\_x, FormationB\_y);
343. **double** error\_x = obstacle\_centor[0] - Formation\_centor[0] - 2\*characteristic\_1;
344. **double** error\_y = obstacle\_centor[1] - Formation\_centor[1];
345. **for** (**int** i = 0; i < swarm\_robot\_id.size(); ++i){
346. del\_x(i) += error\_x;
347. del\_y(i) += error\_y;
348. }
349. pos\_conv = **true**;
350. **if**(std::fabs(del\_x.norm()) > conv\_pos || std::fabs(del\_y.norm()) > conv\_pos) {
351. pos\_conv = **false**;
352. }
353. **for**(**int** i = 0; i < swarm\_robot\_id.size(); i++) {
354. pos\_control(swarm\_robot, i, del\_x(i), del\_y(i), cur\_theta(i));
355. }
356. ros::Duration(0.05).sleep();
357. }
358. **while**(!angle\_conv){
359. get\_pos(swarm\_robot);
360. cal\_formation(formationB\_num, FormationB\_x, FormationB\_y);
361. angle\_conv = **true**;
362. **if**(std::fabs(cur\_theta.norm()) > conv\_th) {
363. angle\_conv = **false**;
364. }
365. **for**(**int** i = 0; i < swarm\_robot\_id.size(); i++) {
366. yaw\_control(swarm\_robot, i, -cur\_theta(i));
367. }
368. }
369. ROS\_INFO\_STREAM("Formation B got!");
370. pos\_conv = **false**;
371. angle\_conv = **false**;
373. /\* 穿过障碍 \*/
374. get\_pos(swarm\_robot);
375. update\_formation\_cost(formationB\_num, FormationB\_x, FormationB\_y, xi\_B);
376. zeta\_B = xi\_B;
377. **while**(! pos\_conv) {
378. get\_pos(swarm\_robot);
379. conv\_formation\_cost(formationB\_num, FormationB\_x, FormationB\_y, zeta\_B, xi\_B);
380. formation\_selection(formationB\_num, zeta\_B);
381. cal\_formation(formationB\_num, FormationB\_x, FormationB\_y);
382. **double** error\_x = obstacle\_centor[0] - Formation\_centor[0] + 2\*characteristic\_1;
383. **double** error\_y = obstacle\_centor[1] - Formation\_centor[1];
384. **for** (**int** i = 0; i < swarm\_robot\_id.size(); ++i){
385. del\_x(i) += error\_x;
386. del\_y(i) += error\_y;
387. }
388. pos\_conv = **true**;
389. **if**(std::fabs(del\_x.norm()) > conv\_pos || std::fabs(del\_y.norm()) > conv\_pos) {
390. pos\_conv = **false**;
391. }
392. **for**(**int** i = 0; i < swarm\_robot\_id.size(); i++) {
393. pos\_control(swarm\_robot, i, del\_x(i), del\_y(i), cur\_theta(i));
394. }
395. ros::Duration(0.05).sleep();
396. }
397. **while**(!angle\_conv){
398. get\_pos(swarm\_robot);
399. cal\_formation(formationB\_num, FormationB\_x, FormationB\_y);
400. angle\_conv = **true**;
401. **if**(std::fabs(cur\_theta.norm()) > conv\_th) {
402. angle\_conv = **false**;
403. }
404. **for**(**int** i = 0; i < swarm\_robot\_id.size(); i++) {
405. yaw\_control(swarm\_robot, i, -cur\_theta(i));
406. }
407. }
408. ROS\_INFO\_STREAM("Successfully penetrated!");
409. pos\_conv = **false**;
410. angle\_conv = **false**;
412. /\* 形成 A 队形 \*/
413. get\_pos(swarm\_robot);
414. update\_formation\_cost(formationA\_num, FormationA\_x, FormationA\_y, xi\_A);
415. zeta\_A = xi\_A;
416. **while**(! pos\_conv) {
417. get\_pos(swarm\_robot);
418. conv\_formation\_cost(formationA\_num, FormationA\_x, FormationA\_y, zeta\_A, xi\_A);
419. formation\_selection(formationA\_num, zeta\_A);
420. cal\_formation(formationA\_num, FormationA\_x, FormationA\_y);
421. **double** error\_x = obstacle\_centor[0] - Formation\_centor[0] + 2\*characteristic\_1;
422. **double** error\_y = obstacle\_centor[1] - Formation\_centor[1];
423. **for** (**int** i = 0; i < swarm\_robot\_id.size(); ++i){
424. del\_x(i) += error\_x;
425. del\_y(i) += error\_y;
426. }
427. pos\_conv = **true**;
428. **if**(std::fabs(del\_x.norm()) > conv\_pos || std::fabs(del\_y.norm()) > conv\_pos) {
429. pos\_conv = **false**;
430. }
431. **for**(**int** i = 0; i < swarm\_robot\_id.size(); i++) {
432. pos\_control(swarm\_robot, i, del\_x(i), del\_y(i), cur\_theta(i));
433. }
434. ros::Duration(0.05).sleep();
435. }
436. **while**(!angle\_conv){
437. get\_pos(swarm\_robot);
438. del\_theta = -lap \* cur\_theta;
439. angle\_conv = **true**;
440. **if**(std::fabs(del\_theta.norm()) > conv\_th) {
441. angle\_conv = **false**;
442. }
443. **for**(**int** i = 0; i < swarm\_robot\_id.size(); i++) {
444. yaw\_control(swarm\_robot, i, del\_theta(i));
445. }
446. }
447. ROS\_INFO\_STREAM("Re-form the A formation!");
449. /\* 程序结束，停止机器人 \*/
450. swarm\_robot.stopRobot();
451. ROS\_INFO\_STREAM("Succeed!");
452. **return** 0;
453. }