

Copyright Undertaking

This work is protected by copyright, with all rights reserved.

By reading and using the work, the reader understands and agrees to the following terms:

1. The reader will abide by the rules and legal ordinances governing copyright regarding the use of the work.
2. The reader will use the work for the purpose of research or private study only and not for distribution or further reproduction or any other purpose.
3. The reader agrees to indemnify and hold the University harmless from and against any loss, damage, cost, liability or expenses arising from copyright infringement or unauthorized usage.

IMPORTANT

If you have reasons to believe that any materials in this work are deemed not suitable to be distributed in this form, or a copyright owner having difficulty with the material being included in our database, please contact lbsys@polyu.edu.hk providing details. The Library will look into your claim and consider taking remedial action upon receipt of the written requests.

Appendix II

```
1. #include "ros/ros.h"
2. #include "boost/units/systems/si.hpp"
3. #include "boost/units/io.hpp"
4. #include "geometry_msgs/Twist.h"
5. #include "std_msgs/String.h"
6. #include "geometry_msgs/PoseStamped.h"
7. #include <geometry_msgs/Pose2D.h>
8. #include <cmath>
9. #include <vector>
10. #include <math.h>
11.
12.
13. // static float      ndt_pose
14. ros::Publisher platformPublisher;
15. ros::Publisher armPublisher;
16. ros::Publisher gripperPublisher;
17. int count;
18. int count_3=0;
19. int guidance=0;
20. bool complete;
21. const float avoid=0.0000000001;
22. const int Max=1000;
23. int block_count=0;
24. int current_count=0;
25. int count_2=0;
26. int count_4=0;
27. int count_5=0;
28. bool count_1=false;
29. int recount_1;
30. int recount_2;
31. bool addstart=false;
32. bool can_move=false;
33. int times_n=1;
34. double obj_x[3],obj_y[3];
35. int obj=0;
36. float block_x,block_y;
37. double abs_x,abs_y;
38.
```

```

39. double block_angle_1[10000]={0};
40. double present_angle_1[10000]={0};
41. double length_1[10]={0};
42. bool addtrue=false;
43. bool restart=false;
44. struct cluster
45. {
46. float clu_location_x;
47. float clu_location_y;
48. int length;
49. float block_pose_x;
50. float block_pose_y;
51.
52. }clu[20];
53.
54. //struct block
55. //{
56.     float block_x1;
57.     float block_y1 ;
58.
59. //}block_2[10000];
60.
61. //struct curent
62. //{
63. float yaw;
64. float current_ndt_pose_x;
65. float current_ndt_pose_y;
66.
67. //}current_1[10000];
68.
69. float the_y;
70. float the_x;
71. float the_y1;
72. float the_x1;
73. void Merge(std::vector<int> &Array, int front, int mid, int end)
74. {
75.
76.     // 利用 std::vector 的 constructor,

```

```

77.    // 把 array[front]~array[mid]放
    進 LeftSub[],place array[front]~array[mid] to LeftSub[]
78.    // 把 array[mid+1]~array[end]放
    進 RightSub[],place array[mid+1]~array[end] to RightSub[]
79.    std::vector<int> LeftSub(Array.begin()+front, Array.begin()+mid+1),
80.                                RightSub(Array.begin()+mid+1, Array.begin()+end+1);
81.
82.    LeftSub.insert(LeftSub.end(), Max);    // 在 LeftSub[]尾端加入值
    為 Max 的元素, add max number at the end of LeftSub[]
83.    RightSub.insert(RightSub.end(), Max);    // 在 RightSub[]尾端加入值
    為 Max 的元素, add max nubmer at the end of RightSub[]
84.
85.    int idxLeft = 0, idxRight = 0;
86.
87.    for (int i = front; i <= end; i++) {
88.
89.        if (LeftSub[idxLeft] <= RightSub[idxRight] ) {
90.            Array[i] = LeftSub[idxLeft];
91.            idxLeft++;
92.        }
93.        else{
94.            Array[i] = RightSub[idxRight];
95.            idxRight++;
96.        }
97.    }
98. }
99.
100. void MergeSort(std::vector<int> &array, int front, int end)
101. {
102.                                     // front 與 end 為矩陣範圍
103.    if (front < end) {                // 表示目前的矩陣範圍是有效的
        ,present the range of the matrix
104.        int mid = (front+end)/2;    // mid 即是將矩陣對半分的
        index, mid is the index that divides the matrix
105.        MergeSort(array, front, mid);    // 繼續 divide 矩陣的前半段
        subarray , sort the first half of matrix
106.        MergeSort(array, mid+1, end);    // 繼續 divide 矩陣的後半段
        subarray, sort the last half of the matrix

```

```

107.             Merge(array, front, mid, end);    // 將兩個 subarray 做比較, 並
            合併出排序後的矩陣, compare the subarray of the matrix
108.         }
109.     }
110.
111.     void start_compare()
112.     {
113.         if (addtrue==1)
114.         {
115.             std::vector<int> array(length_1, length_1+sizeof(length_1)/sizeof
                (int));
116.             MergeSort(array,0,9);
117.             std::cout<<"The minimum distance is"<<array[0]<<std::endl;
118.             if (count_5<10)
119.             {
120.                 if(clu[count_5].length==array[0])//determine the distance wh
                    ich has lowest value
121.                 {
122.
123.                     std::cout<<"The absolute location of block of x is"<<
                        clu[count_5].clu_location_x+clu[count_5].block_pose_x<<std::endl;
124.                     std::cout<<"The absolute location of block of y is"<<
                        clu[count_5].clu_location_y+clu[count_5].block_pose_y<<std::endl;
125.
126.                     abs_x=clu[count_5].clu_location_x;
127.                     abs_y=clu[count_5].clu_location_y;
128.                     block_x=clu[count_5].block_pose_x;
129.                     block_y=clu[count_5].block_pose_y;
130.                     can_move=true;
131.                 }
132.                 count_5++;
133.
134.             }
135.             if(count_5==10)
136.             {
137.                 count_2=0;
138.                 count_5=0;
139.                 count_4=0;

```

```

140.         addtrue=0;
141.     }
142. }
143. }
144.
145. void givevalue()
146. {
147.
148.     if(count_4<10&&(addstart==1))
149.     {
150.         clu[count_4].length=length_1[count_4];
151.         //std::cout<<"The distance is"<<clu[count_4].length<<std::endl;
152.         //std::cout<<length_1[count_4]<<std::endl;
153.         count_4++;
154.         addtrue=0;
155.     }
156.     if(count_4==10)
157.     {
158.         start_compare();
159.         addtrue=1;
160.     }
161. }
162.
163. void guidance_1(float yaw_1,float block_x11,float block_y11,float current_pose_x1, float current_pose_y1)//determine the block in the range or not
164. {
165.
166.     complete=false;
167.     if(restart==true||count_1==false)//if there is count1. the recount occurs
168.     {
169.         if(guidance<10000)
170.         {
171.
172.             count_1=true;
173.

```

```

174.         if(count_2<20)
175.         {
176.             count_1=true;
177.             if (M_PI*0.5<yaw_1<M_PI)
178.             {
179.                 //std::cout<<"The targeted block has distance
180.                 1"<<block_x11<<" "<<block_y11<<std::endl;
181.                 //std::cout<<"1"<<std::endl;
182.                 the_y1=abs(current_pose_y1);
183.                 the_x1=abs(current_pose_x1+avoid);
184.                 present_angle_1[guidance]=-
185.                 1*tanh(the_y1/the_x1)+M_PI;
186.                 if(block_x11<0&&block_y11>0)
187.                 {
188.                     //std::cout<<"11"<<std::endl;
189.                     the_y=abs(block_y11);
190.                     the_x=abs(block_x11+avoid);
191.                     block_angle_1[guidance]=-
192.                     1*tanh(the_y/the_x)+M_PI;
193.                     //std::cout<<"The block angle"<<guidance<
194.                     <" is"<<block_angle_1[guidance];
195.                     complete=true;
196.                 }
197.                 if(block_x11<0&&block_y11<0)
198.                 {
199.                     //std::cout<<"12"<<std::endl;
200.                     the_y=abs(block_y11);
201.                     the_x=abs(block_x11+avoid);
202.                     block_angle_1[guidance]=tanh(the_y/abs(th
203.                     e_x))+M_PI;
204.                     //std::cout<<"The block angle"<<guidance<
205.                     <" is"<<block_angle_1[guidance];
206.                     complete=true;
207.                 }
208.                 if(block_x11>0&&block_y11>0)
209.                 {
210.                     //std::cout<<"13"<<std::endl;

```

```

206.                the_y=abs(block_y11);
207.                the_x=abs(block_x11+avoid);
208.                block_angle_1[guidance]=tanh(the_y/the_x)
                ;
209.                //std::cout<<"The block angle"<<guidance<
                <" is"<<block_angle_1[guidance];
210.                complete=true;
211.            }
212.            if(block_x11>0&&block_y11<0)
213.            {
214.                //std::cout<<"15"<<std::endl;
215.
216.                complete=true;
217.            }
218.
219.            if((yaw-
                0.1<block_angle_1[guidance]<yaw+0.1)&&(block_x+block_y!=0))
220.            {
221.                //std::cout<<"14"<<std::endl;
222.                length_1[count_2]=(pow(block_x11,2)+pow(b
                lock_y11,2));
223.                //std::cout<<"The distance is 11 "<<lengt
                h_1[count_2]<<std::endl;
224.                clu[count_2].block_pose_x=block_x11;
225.                clu[count_2].block_pose_y=block_y11;
226.                clu[count_2].clu_location_x=current_pose_
                x1;
227.                clu[count_2].clu_location_y=current_pose_
                y1;
228.                count_2++;
229.
230.            }
231.
232.        }
233.
234.        if (-1*M_PI<yaw_1<-0.5*M_PI)
235.        {

```

[illegible]

```

266.             the_x=abs(block_x11+current_pose_x1+avoid
);
267.             block_angle_1[guidance]=tanh(the_y/the_x)
+M_PI;
268.             //std::cout<<"The block angle"<<guidance<
<" is"<<block_angle_1[guidance];
269.             complete=true;
270.
271.
272.         }
273.         if(M_PI+abs(M_PI+yaw_1)-
0.1<block_angle_1[guidance]<M_PI+abs(M_PI+yaw_1)+0.1&&(block_x+block_y!=0))
274.         {
275.
276.             //std::cout<<"24"<<std::endl;
277.             length_1[count_2]=(pow(block_x11,2)+pow(b
lock_y11,2));
278.             //std::cout<<"The distance is 2 "<<length
_1[count_2]<<std::endl;
279.             clu[count_2].block_pose_x=block_x11;
280.             clu[count_2].block_pose_y=block_y11;
281.             clu[count_2].clu_location_x=current_pose_
x1;
282.             clu[count_2].clu_location_y=current_pose_
y1;
283.             count_2++;
284.
285.         }
286.
287.     }
288.
289.     //if (-M_PI/2<yaw_1<0)
290.     if (0<yaw_1<0.5*M_PI)
291.     {
292.         //std::cout<<"The targeted block has distance
3"<<block_x11<<" "<<block_y11<<std::endl;

```

```

293.                //std::cout<<"The targeted block has distance
                3"<<block_x1<<block_y1;
294.                //std::cout<<"3"<<std::endl;
295.                the_y1=abs(current_pose_y1);
296.                the_x1=abs(current_pose_x1+avoid);
297.                present_angle_1[guidance]=tanh(the_y1/the_x1)
                ;
298.                if(block_x1<0&&block_y1>0)
299.                {
300.                    //std::cout<<"31"<<std::endl;
301.                    the_y=abs(block_y1);
302.                    the_x=abs(block_x1+avoid);
303.                    block_angle_1[guidance]=-
                    1*tanh(the_y/the_x)+M_PI;
304.                    //std::cout<<"The block angle"<<guidance<
                    <" is"<<block_angle_1[guidance];
305.                    complete=true;
306.                }
307.                if(block_x1>0&&block_y1<0)
308.                {
309.                    //std::cout<<"32"<<std::endl;
310.                    the_y=abs(block_y1);
311.                    the_x=abs(block_x1+avoid);
312.                    block_angle_1[guidance]=-
                    1*tanh(the_y/the_x);
313.                    //std::cout<<"The block angle"<<guidance<
                    <" is"<<block_angle_1[guidance];
314.                    complete=true;
315.
316.                }
317.                if(block_x1>0&&block_y1>0)
318.                {
319.                    //std::cout<<"33"<<std::endl;
320.                    the_y=abs(block_y1);
321.                    the_x=abs(block_x1+avoid);
322.                    block_angle_1[guidance]=tanh(the_y/the_x)
                    ;

```

[illegible]

```

351.                present_angle_1[guidance]= -
                    tanh(the_y1/the_x1)+2*M_PI;
352.                if(block_x11<0&&block_y11<0)
353.                {
354.                    //std::cout<<"41"<<std::endl;
355.                    the_y=abs(block_y11);
356.                    the_x=abs(block_x11+avoid);
357.                    block_angle_1[guidance]=tanh(the_y/the_x)
                    +M_PI;
358.                    //std::cout<<"The block angle"<<guidance<
                    <" is"<<block_angle_1[guidance];
359.                    complete=true;
360.
361.                }
362.                if(block_x11>0&&block_y11<0)
363.                {
364.                    //std::cout<<"42"<<std::endl;
365.                    the_y=abs(block_y11);
366.                    the_x=abs(block_x11+avoid);
367.                    block_angle_1[guidance]= -
                    1*tanh(the_y/the_x)+2*M_PI;
368.                    //std::cout<<"The block angle"<<guidance<
                    <" is"<<block_angle_1[guidance];
369.                    complete=true;
370.
371.                }
372.                if(block_x11>0&&block_y11>0)
373.                {
374.                    //std::cout<<"43"<<std::endl;
375.                    the_y=abs(block_y11);
376.                    the_x=abs(block_x11+avoid);
377.                    block_angle_1[guidance]=tanh(the_y/the_x)
                    +2*M_PI;
378.                    //std::cout<<"The block angle"<<guidance<
                    <" is"<<block_angle_1[guidance];
379.                    complete=true;
380.
381.                }

```

```

382.
383.             if(2*M_PI+yaw_1-
                0.1<block_angle_1[guidance]<2*M_PI+yaw_1+0.1&&(block_x+block_y!=0))
384.             {
385.                 //std::cout<<"44"<<std::endl;
386.                 length_1[count_2]=(pow(block_x11,2)+pow(b
                lock_y11,2));
387.                 //std::cout<<"The targeted block"<<count_
                2<<"has distance"<<block_x11+block_y11;
388.                 //std::cout<<"The distance is 4  "<<length_
                h_1[count_2]<<std::endl;
389.                 clu[count_2].block_pose_x=block_x11;
390.                 clu[count_2].block_pose_y=block_y11;
391.                 clu[count_2].clu_location_x=current_pose_
                x1;
392.                 clu[count_2].clu_location_y=current_pose_
                y1;
393.                 count_2++;
394.
395.             }
396.
397.         }
398.     }
399.
400.         //if(count_2==10)
401.         //{
402.
403.             //}
404.         }
405.
406.         //if(count_2==10)
407.         //{
408.             //addstart=1;
409.             //givevalue();
410.             //}
411.         //else
412.             //addstart=0;
413.     }

```

```
414.
415.         if(count_2==20)
416.         {
417.             addstart=1;
418.             givevalue();
419.         }
420.         else
421.             addstart=0;
422.
423.     }
424.
425.
426.
427.     void chatterCallback(const geometry_msgs::Point::ConstPtr& centroid)
428.     {
429.
430.         //ROS_INFO("I heard: Henry");
431.         if (block_count<10000)
432.         {
433.             block_x1 = (*centroid).x;
434.             block_y1=(*centroid).y;
435.             //std::cout<<sqrt(pow(block_x1,2)+pow(block_y1,2))<<std::endl;
436.             //block_2[block_count].block_x1 = (*centroid).x;
437.             //block_2[block_count].block_y1=(*centroid).y;
438.             //std::cout<<"block_x is"<<block_x1 <<std::endl;
439.             //std::cout<<"block_y is"<<block_y1 <<std::endl;
440.             //std::cout<<sqrt(pow(block_x1,2)+pow(block_y1,2))<<std::endl;
441.             block_count++;
442.         }
443.
444.     }
445.
446.     void pose2dCallback(const geometry_msgs::Pose2D::ConstPtr& pose2d)
447.     {
448.         //ROS_INFO("I heard pose2d");
449.
450.         if (current_count<10000)
```

```

451.     {
452.         yaw=(*pose2d).theta;
453.         current_ndt_pose_x=(*pose2d).x;
454.         current_ndt_pose_y=(*pose2d).y;
455.         //current_1[current_count].yaw=(*pose2d).theta;
456.         //current_1[current_count].current_ndt_pose_x=(*pose2d).x;
457.         //current_1[current_count].current_ndt_pose_y=(*pose2d).y;
458.         //std::cout<<"yaw is"<<yaw<<std::endl;
459.         //std::cout<<"current x is"<<current_ndt_pose_x<<std::endl;
460.         //std::cout<<"current y is "<<current_ndt_pose_y<<std::endl;
461.
462.         //std::cout<<-
            1*tanh((current_1[current_count].current_ndt_pose_y)/(current_1[current_count].current_ndt_pose_x+avoid))+M_PI;
463.         //std::cout<<sqrt(pow(block_2[block_count].block_y1,2)+pow(block_2[block_count].block_y1,2));
464.         current_count++;
465.     }
466.     //if(current_count==100&&block_count==100)
467.     //{
468.     if(guidance<10000)
469.     {
470.         guidance_1(yaw,block_x1,block_y1,current_ndt_pose_x,current_ndt_pose_y);
471.         restart=true;
472.         guidance++;
473.     }
474.     //}
475.
476.     //if (can_move==true)
477.     //{
478.         //movePlatform(current_ndt_pose_x,current_ndt_pose_y);
479.     //}
480.     //std::cout<<"yaw_p "<<yaw;
481.
482.     }
483.
484.

```

```
485.     int main(int argc, char **argv)
486.     { // double pose.position.x;
487.       // double pose.position.y;
488.
489.       obj_x[0]=10;obj_y[0]=5;
490.       ros::init(argc, argv, "sam2");
491.       ros::NodeHandle n;
492.
493.       platformPublisher = n.advertise<geometry_msgs::Twist>("cmd_vel",
494.         1);
495.       //ros::Subscriber sub = n.subscribe("ndt_pose", 10000, ndt_poseCa
496.         llback);
497.       ros::Subscriber sub1 = n.subscribe("chatter", 10000, chatterCallb
498.         ack);
499.       ros::Subscriber sub2 =n.subscribe("pose2d",10000, pose2dCallback)
500.         ;
501.
502.       //current_ndt_pose_x=input->ndt_pose.x;
503.       //current_ndt_pose_y=input->ndt_pose.y;
504.
505.       sleep(1);
506.       ROS_INFO("mmm");
507.
508.       sleep(1);
509.       // ros::shutdown();
510.
511.       ros::spin();
512.       return 0;
513.     }
```