评分：_____

机器人入门项目设计 II

Robocon2019 激光雷达定位设计方案

学　院　粤港机器人学院

年　级　黄埔三期

组　别　第二组

姓　名　林俊坤

老　师　黄之峰、李东

2019 年 7 月

目录

1 绪论

1.1 题目背景及意义

第十八届全国大学生机器人大赛 ROBOCON 以“快马加鞭”为主题，灵感来自古代传递信息的驿传制度。我国古代为接待往来官员和传递政府文书、军令建立了驿传制度以及为此而征发的徭役制度。驿传制度始于殷商。甲骨文有“骑士”、“车”、“驿传”等记载。春秋战国已有邮驿，甘肃嘉峪关魏晋墓的砖画上已有驿使快马传书的记载。秦统一后，广建驿道，邮驿发展迅速。汉承秦制，在通路上每 30 里置一驿站，供传递军事情报的官员途中食宿、换马之用。唐有车、马、船之分，设水驿、陆驿、水陆兼办三种，由兵部统辖。宋驿传由军队统管，设军邮局，置日夜兼行的急递铺，专递送军事文书，有 800 里急报之说。元代设有蒙古站赤（驿传之音译）及汉地的驿站。明代驿传有陆驿、水驿、驿站、递运所、急递铺、马快船等。清代有驿站、塘、台、所、铺等。古代驿站一般备有人夫、车马，并指办禀给、口粮，供传递文书及过境官员使用。清光绪时，随着新式邮政的建立而渐被废止。

如今，信息以光速高速传递，互联网所形成的逻辑上单一且巨大的全球化网络使我们能不受空间限制并以多种形式进行信息交换。本年度的比赛用机器人演绎古代的信息传递方式，使我们对信息与知识的共享更有信心。

1.2 比赛过程简介

比赛在图 1.2.1 所示的场地上进行，红、蓝两队各占一半。比赛最多持续 3 分钟。每支参赛队有一个名为“机器信使 1”的手动机器人和一个名为“机器信使 2”的自动机器人。手动机器人携带作为信物的令牌从“龙门驿”（手动机器人的启动区）出发。它沿树林、桥梁行进，跨过界线 1，到达“大漠驿”（自动机器人的启动区）。这时，机器信使 1 将令牌交给在大漠驿的机器信使 2。一旦机器信使 2 成功地接收到令牌，它就可以沿着大漠区行进。机器信使 2 必须像马一样有四条腿，不能用轮子移动。机器信使 2 通过沙丘和草地，向“高山驿”进发。机器信使 2 到达高山驿后，机器信使 1 可以进入投掷区投掷兽骨得分。如果机器信使 1 获得 50 或更多得分，机器信使 2 就可以登山。此后，如果它到达山顶区，举起令牌，该队获胜，这就是所谓的“登顶”。

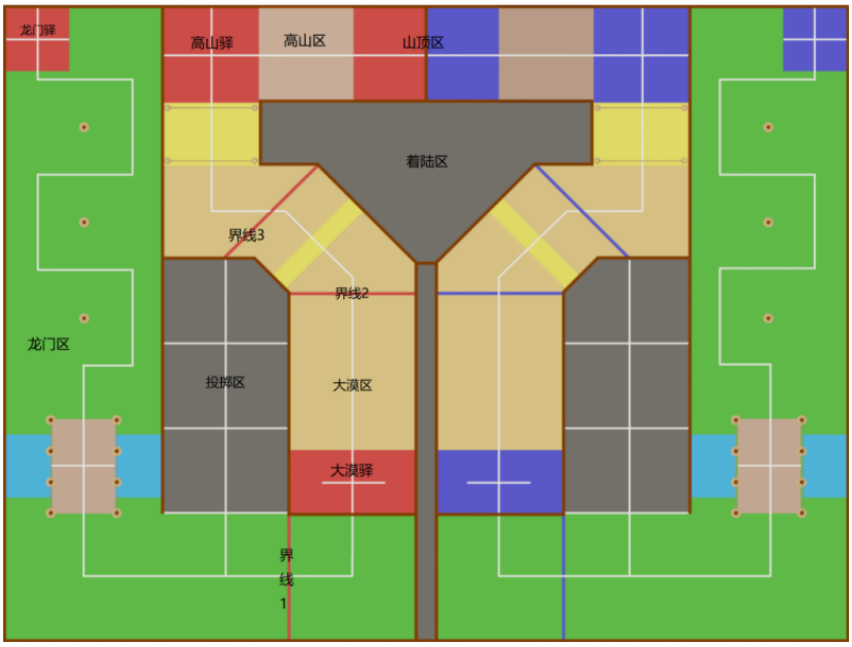


图 1.2.1 比赛地图

2 个人任务

2.1 个人任务概述

根据四足机械组的设计方案，提出一种可行的定位方案设计。四足机器人的机械结构图为下图 2.1，图中红色圆圈的物体为激光雷达。

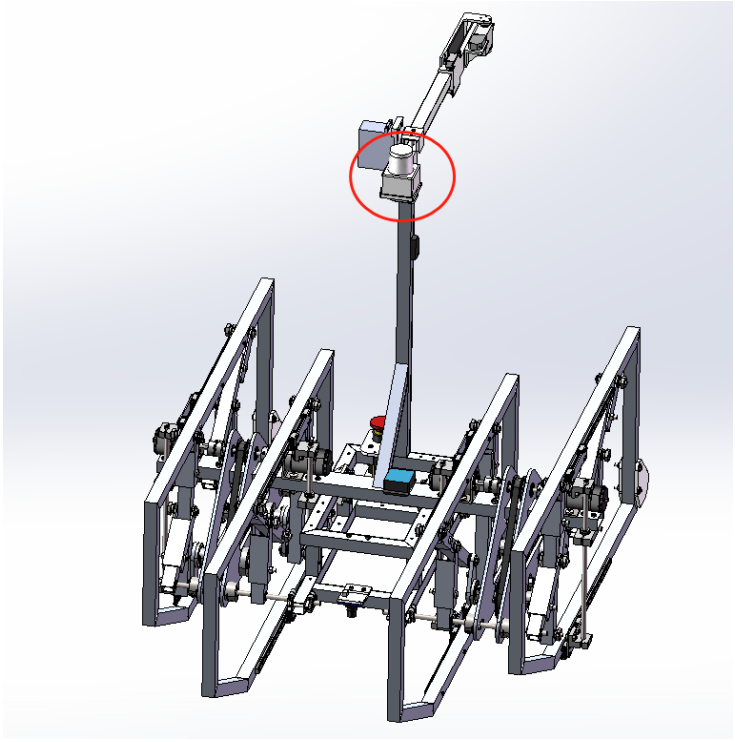


图 2.1 四足机器人整体设计

2.1 方案讨论

2.1.1 方案概述

利用现有的激光雷达给四足机器人提供 Robocon2019 比赛场地上的定位服务，广东工业大学粤港机器人学院现有的激光雷达型号为：UST-10LX 和 UST-30LX。它们的特点有：

1、扫描半径大。UST-10LX 的最大扫描半径(Detection range)是 30M，在反射率为 10%的情况下扫描距离降低为 4M。影响反射率的主要因素有两个：物体的颜色和表面的光滑程度。雷达的红外光反射受物体颜色和表面形状所影响。越光滑、颜色越明亮，红外光反射率越高。白石膏能够 100%反射红外光，而黑色泡沫橡胶只能反射 2.4%。

2、扫描频率高。UST-10LX 的扫描频率是 25ms。

3、角度分辨率低。UST-10LX 角度分辨率为 0.25°。角度分辨率(Angular resolution)为 0.25°是指两两光线之间的角度是 0.25°。雷达能否扫描到远处的物体，能够获得多少个点，就是根据这个参数来计算的。

4、扫描范围大。雷达扫描角度有 270°，即有 90°的盲区。

2.1.2 对比赛规则的分析：

根据比赛场地图和比赛规则可知，场地上的树林区有六个位置固定的柱形障碍物，为方便对定位方案的讨论，分别对它们命名为 A、B、C、D、E、F 柱（下图 2.2 中红色圆圈的物体，对应命名见圆圈旁的英文字母）。它们高 1200mm，形状大小如下图 2.3。

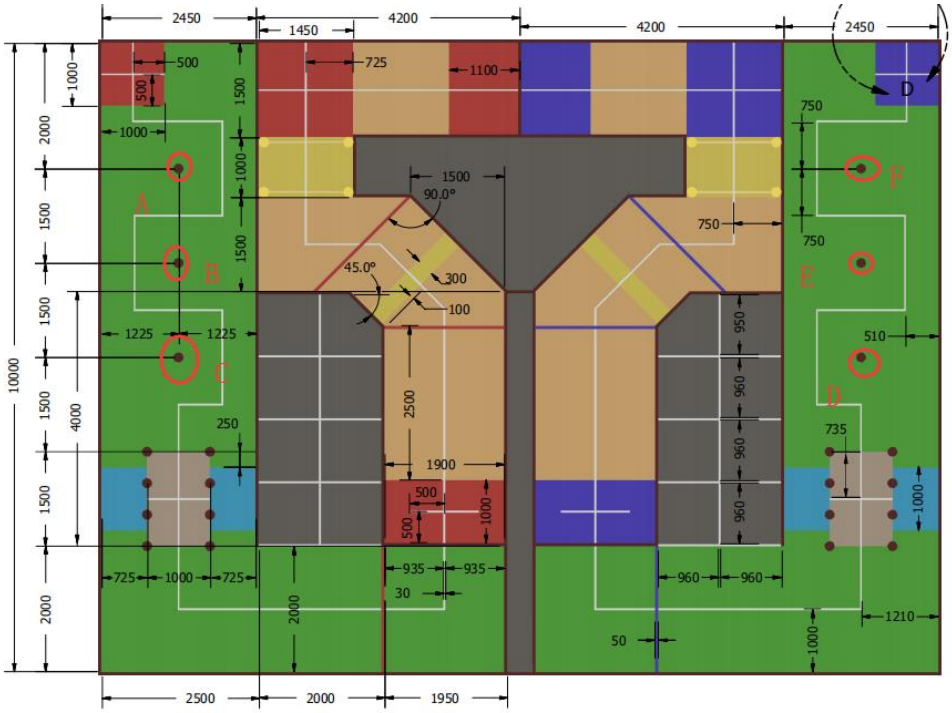


图 2.2 比赛场地图

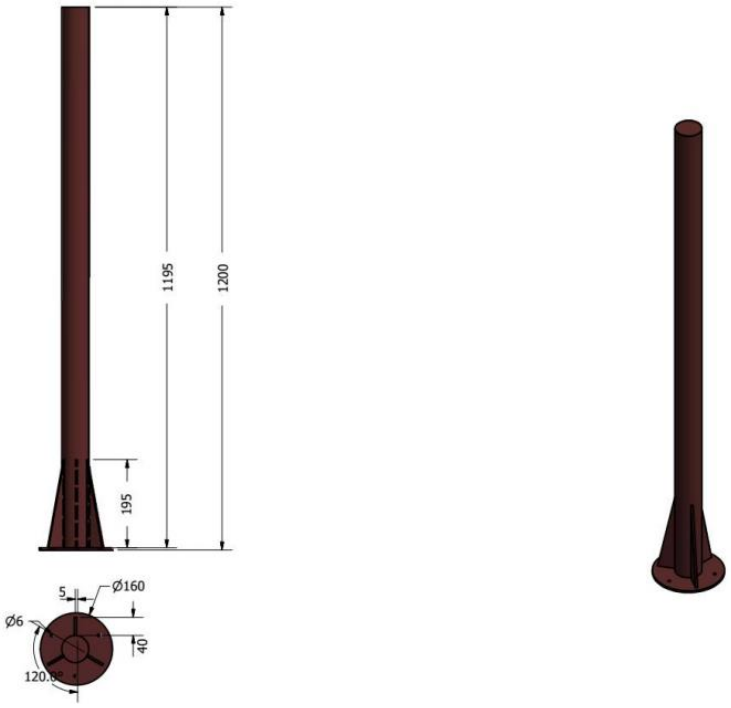


图 2.3 树林障碍物

比赛规则要求：“四足机器人的宽、长、高尺寸均不得小于 400mm。比赛中，机器人不得超过 800mm 宽、1000 mm 长、800mm 高。”由此可知，比赛过程中，机器人的高度始终低于 1200mm，所以激光雷达的高度始终低于柱子的最高点。

2.1.3 雷达定位方案一

利用雷达与场上的六个柱子距离信息，利用三角形的余弦定理和立体几何原理可以计算得出雷达在比赛场地上的位置。

1、可行性分析：

（1）通过计算证明雷达在场地上四足机器人的活动区域内，是否可以一直扫描得到场地上的柱子。

证明过程：

活动区域内离最远的柱子的距离为 8625，柱子的直径为 60mm，雷达的角分辨率为 0.25°，由三角形的余弦定理（公式 2.1）可得，当距离为 8625mm 时，可以扫描到的障碍物最小宽度为：

$$\sqrt{(8625^2 + 8625^2 - 2 * 8625 * 8625 * \cos(0.25°))} \approx 37\text{mm} < 60\text{mm}$$

结论：雷达可以在活动区域内一直扫描到柱子。

$$c = \sqrt{a^2 + b^2 - 2bc\cos c} \qquad \text{（公式 2.1）}$$

（2）雷达有 90° 盲区，是否可以保证四足机器人在比赛场上可以同时扫描得到己方和对方的柱子？

答：可以，只需把雷达按照如下图 2.4 中的方法放置即可保证同时扫描得到两侧的柱子且每侧至少有一个柱子（蓝色为雷达 90° 盲区）。

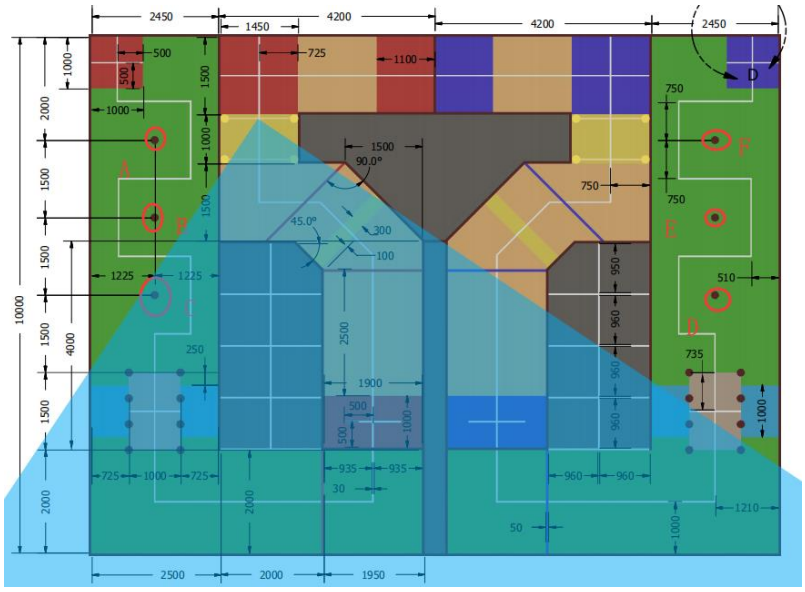


图 2.4 六点定位雷达放置方法

2、识别并分辨柱子的算法流程

（1）识别障碍物的算法流程：

识别障碍物的方法是通过计算两两根激光之间的长度差值小于设定的阈值即判断扫描到的障碍物为同一物体，否则为不同物体。如下图 2.5 中，红色矩

形为雷达，黑色矩形为障碍物，蓝色、黄色、绿色射线为雷达发出的激光。其中，蓝色激光 A 的与黄色激光 C 的长度差值大于阈值，即判断：激光 A 及之前的激光与 B 激光及之后的激光扫描到的障碍物为不同物体。黄色激光 B 与黄色激光 C 的长度差值小于设定阈值，即判断激光 B 和激光 C 扫描到的物体为同一物体，同理，B 到 D 的激光扫描到的物体为同一物体。黄色激光 D 与绿色激光 E 的长度差值的绝对值大于阈值，即判断激光 D 及激光 D 之前的激光与激光 E 及激光 E 之后的物体为不同物体。

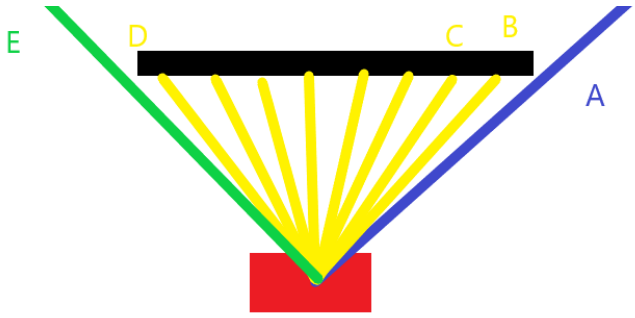


图 2.5 识别障碍物算法模型

识别障碍物的算法代码：

```
//此函数用来记录障碍物的数量，Distance[]为雷达的距离数组，block_mid 为固定大小的二维数组起始地址，用来存放筛选到的障碍物的信息
void block_record(u32 *Distance,float **block_mid)
{
    volatile int flag = 0;//用来判断柱子是否记录完全，1 为判断记录完全
    int block_min = 0, block_max = 0;//记录障碍物的起始 step 数
    int   i = 1,c = 0,dist_min;//dist_min 为到障碍物的最小距离
    double p = 0.25*3.14159265359/180.0;//从 step 数到角度的换算单位
    block_num=0;

    for (i = RangMinStep; i < RangMaxStep; i++)//400;i< 600; i++)//ALL_STEP; i++)//i < 30; i++)//ALL_STEP; i++)
    {
        if (Distance[i] < 9000)//筛选 9m 内的障碍物
        {
            block_min = i, block_max = i;
            flag = 0;
            dist_min=Distance[i];
            for (flag = 0; flag == 0 && i < RangMaxStep; i++)//测量柱子的宽度
            {
                if ( Abs((int)(Distance[i+1] - (int)Distance[i])) < (pillar_wid + 220) )//pillar_wid 为柱子的宽度，作为筛选障碍物的阈值
                {
                    if(dist_min>Distance[block_max])
                        dist_min=Distance[block_max];//取最短值作为距离
                    block_max = i+1;
```

```
                    }

                    else
                    {
                        if(dist_min>Distance[block_max])
                            dist_min=Distance[block_max];
                        flag = 1;//跳出测量障碍物宽度的循环
                        block_mid[block_step][block_num] = (block_max + block_min) / 2;
//                      block_mid[block_dist][block_num] = Distance[(block_max +
//block_min+1) / 2];//这里只去中间 step 的距离为障碍物的距离
                        block_mid[block_dist][block_num] = dist_min;
                        block_mid[block_widt][block_num] =
sqrt(pow((double)Distance[block_max],2) +

                            pow((double)Distance[block_min],2) -

                            2*( (double)Distance[block_max] ) *

                            (double)Distance[block_min]* (cos( (double)(block_max-
block_min+1) *p)));
                        i--;
//                      printf("min_dist:%d min_step:%d    max_dist:%d max_step:%d
\r\nwidt:%f\tdist:%f\r\n",
//
    Distance[block_min],block_min,Distance[block_max],block_max,block_mid[block_widt][bl
ock_num],block_mid[block_dist][block_num]);          //调试时取消注释

                        block_num++;
                    }
                }
            }
        }
//      printf("十米内有%d 个障碍物\r\n", block_num);//打印障碍物的数量调试时取消注释
}
```

（2）识别柱子的算法流程：

首先利用余弦定理，计算障碍物的宽度信息，如果障碍物的宽度在阈值范围里面，即判断为可能是柱子；然后在可能是柱子的障碍物里面判断是否满足比赛规则摆放要求，是则为可能为柱子，否则是其他无关障碍物。

识别柱子的算法代码：

```
////六点定位原筛选柱子的函数
void pillar_filter_6(float **block_mid, float **pillar)
{
    extern int block_num;
    volatile int i = 0;
```

```
    int j = 0;
    float temp_widt[50][5] = { 0 };//存放宽度合适的障碍物
    float temp_dist[50][5] = { 0 };//存放距离合适的障碍物

    int temp_pillar = 0;
    i = 0;
    pillar_num = 0;

    for (i = 0; i < 20; i++)
    {
        for (j = 0; j < 5; j++)
        {
            pillar[i][j] = 0;
        }
    }

    for (i = 0; i < block_num; i++)//筛选宽度合适的柱子,把合适的障碍物放到 temp_widt 数组里面
    {
        //          printf("block_mid[dist][%d]:%f block[widt][%d]:%f\r\n",i,block_mid[block_dist][i],i,block_mid[block_widt][i]);
        //if ((block_mid[block_widt][i] > (pillar_wid - 10)) && (block_mid[block_widt][i] < (pillar_wid + pillar_wid_deviation))) //计算宽度，如果障碍物宽度是在 60+-10mm 的话，即为柱子

        if ((block_mid[block_widt][i] > (58)) && (block_mid[block_widt][i] < (300)))
        {
            temp_widt[temp_pillar][step] = block_mid[block_step][i];
            temp_widt[temp_pillar][dist] = block_mid[block_dist][i];

//              printf("距离:%f\t\tsetp:%f\t 宽度:%f\r\n", temp_widt[temp_pillar][dist], temp_widt[temp_pillar][step], block_mid[block_widt][i]);
            temp_pillar++;
        }
    }
//////////筛选距离值符合要求的柱子\\\\\\\\\\\\
    for (i = 0; i < temp_pillar; i++)
    {
        for (j = i + 1; j < temp_pillar; j++)
            if (DistTwoPillar1(temp_widt, i, j) > AB_MIN-20 && DistTwoPillar1(temp_widt, i, j) < AB_MAX+20)
            {
                temp_dist[pillar_num][step] = temp_widt[i][step];
                temp_dist[pillar_num][dist] = temp_widt[i][dist];
```

```
                    temp_dist[++pillar_num][step] = temp_widt[j][step];
                    temp_dist[pillar_num][dist] = temp_widt[j][dist];
                    pillar_num++;
                    i = j;
                }
    }

    for (i = 0; i < temp_pillar; i++)
    {
        for (j = i + 1; j < temp_pillar; j++)
            if (DistTwoPillar1(temp_widt, i, j) > AC_MIN-20 && DistTwoPillar1(temp_widt,
i, j) < AC_MAX+20)
                {
                    temp_dist[pillar_num][step] = temp_widt[i][step];
                    temp_dist[pillar_num][dist] = temp_widt[i][dist];
                    temp_dist[++pillar_num][step] = temp_widt[j][step];
                    temp_dist[pillar_num][dist] = temp_widt[j][dist];
                    pillar_num++;
                    i = j;
                }
    }

    for (i = 0; i < temp_pillar; i++)
    {
        for (j = i + 1; j < temp_pillar; j++)
            if (DistTwoPillar1(temp_widt, i, j) > AD_MIN-20 && DistTwoPillar1(temp_widt,
i, j) < AD_MAX+20)
                {
                    temp_dist[pillar_num][step] = temp_widt[i][step];
                    temp_dist[pillar_num][dist] = temp_widt[i][dist];
                    temp_dist[++pillar_num][step] = temp_widt[j][step];
                    temp_dist[pillar_num][dist] = temp_widt[j][dist];
                    pillar_num++;
                    i = j;
                }
    }

    for (i = 0; i < temp_pillar; i++)
    {
        for (j = i + 1; j < temp_pillar; j++)
            if (DistTwoPillar1(temp_widt, i, j) > AE_MIN-20 && DistTwoPillar1(temp_widt,
i, j) < AE_MAX+20)
                {
                    temp_dist[pillar_num][step] = temp_widt[i][step];
```

```
                    temp_dist[pillar_num][dist] = temp_widt[i][dist];
                    temp_dist[++pillar_num][step] = temp_widt[j][step];
                    temp_dist[pillar_num][dist] = temp_widt[j][dist];
                    pillar_num++;
                    i = j;
                }
        }

        for (i = 0; i < temp_pillar; i++)
        {
            for (j = i + 1; j < temp_pillar; j++)
                if (DistTwoPillar1(temp_widt, i, j) > AF_MIN-20 && DistTwoPillar1(temp_widt,
i, j) < AF_MAX+20)
                {
                    temp_dist[pillar_num][step] = temp_widt[i][step];
                    temp_dist[pillar_num][dist] = temp_widt[i][dist];
                    temp_dist[++pillar_num][step] = temp_widt[j][step];
                    temp_dist[pillar_num][dist] = temp_widt[j][dist];
                    pillar_num++;
                    i = j;
                }
        }

        //排序并去掉相同的柱子

        for (i = pillar_num; i > 0; i--)//数组往后挪一位
        {
            temp_dist[i][step] = temp_dist[i - 1][step];
            temp_dist[i][dist] = temp_dist[i - 1][dist];
        }
        for (i = 1; i < pillar_num; i++)//直接插入排序
        {
            if (temp_dist[i + 1][step] < temp_dist[i][step] && temp_dist[i][step]!=0)
            {
                temp_dist[0][step] = temp_dist[i + 1][step];
                temp_dist[0][dist] = temp_dist[i + 1][dist];
                j = i + 1;
                do {
                    j--;
                    temp_dist[j + 1][step] = temp_dist[j][step];
                    temp_dist[j + 1][dist] = temp_dist[j][dist];
                } while (temp_dist[0][step] < temp_dist[j - 1][step]);
                temp_dist[j][step] = temp_dist[0][step];
                temp_dist[j][dist] = temp_dist[0][dist];
```

```
            }
        }

        for (i = 1; i < pillar_num + 1; i++)//数组往前挪一位
        {
            temp_dist[i - 1][step] = temp_dist[i][step];
            temp_dist[i - 1][dist] = temp_dist[i][dist];
        }
        temp_dist[pillar_num][step] = 0;
        temp_dist[pillar_num][dist] = 0;

        for (i = 1; i < pillar_num; i++)//去掉相同的柱子
        {
            j = i;
            if (temp_dist[i][step] == temp_dist[i - 1][step])
            {
                for (; j < pillar_num + 1; j++)//数组往前挪一位
                {
                    temp_dist[j][step] = temp_dist[j + 1][step];
                    temp_dist[j][dist] = temp_dist[j + 1][dist];

                }
                i--;
                pillar_num--;
            }
        }


        printf("right dist pillar:%d 个\r\n",pillar_num);
        printf("step:\r\n");
        for(i=0;i<pillar_num;i++)//筛选前
        {
            printf("%f,",temp_dist[i][step]);
        }
        printf("\r\n");
        printf("dist:\r\n");
        for(i=0;i<pillar_num;i++)//筛选前
        {
            printf("%f,",temp_dist[i][dist]);
        }
        printf("\r\n");

    //在 temp_dist 数组中寻找同一直线的柱子
        findLinePillar(temp_dist, pillar);
```

```
        printf("in one line pillar_num:%d 个\r\n",pillar_num);
        printf("in one line:\r\n");
        printf("step:\r\n");
        for(i=0;i<pillar_num;i++)//筛选后
        {
            printf("%f,",pillar[i][step]);
        }
        printf("\r\n");
        printf("dist:\r\n");
        for(i=0;i<pillar_num;i++)//筛选后
        {
            printf("%f,",pillar[i][dist]);
        }
        printf("\r\n");

//
//      for (i = 0; i < pillar_num; i++)//打印柱子状态
//          printf("筛选后的柱子:%f\t\tsetp:%f\r\n", pillar[i][dist], pillar[i][step]);

//      for (i = 0; i < pillar_num; i++)
//          printf("%f,", pillar[i][step]);
//      printf("\n");
//      for (i = 0; i < pillar_num; i++)
//          printf("%f,", pillar[i][step]);
//      printf("\r\n");

}
```

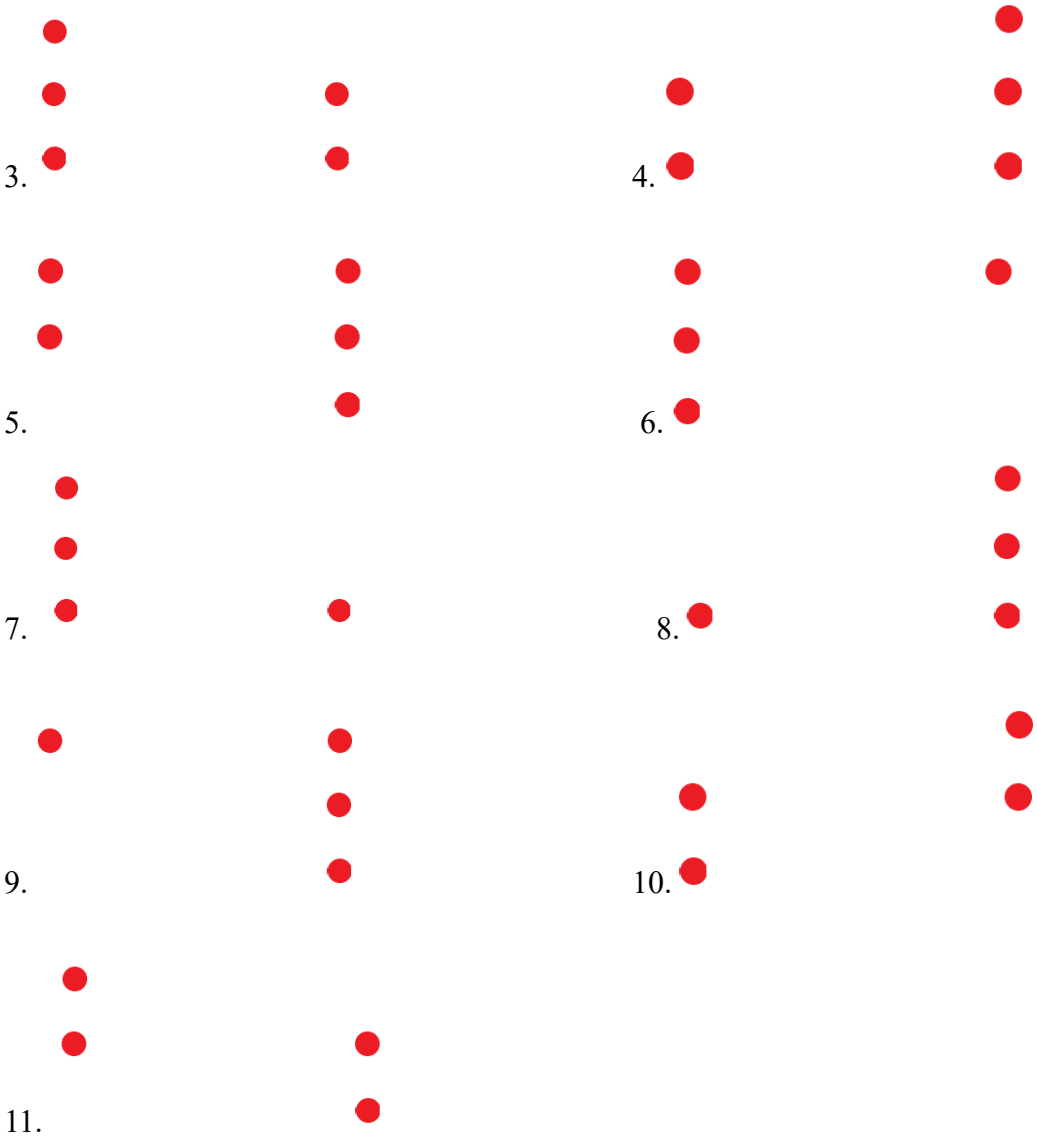
（3）给柱子命名的算法流程：

经过识别柱子的算法流程，如果没有判断为可能是柱子的障碍物，则返回上一次的位置信息。如果有判断为可能是柱子的障碍物，则计算障碍物之间的距离，把距离符合要求的柱子记录下来，如果至多只有 3 个符合要求的柱子，则根据上一次柱子的位置信息来对柱子进行命名；如果至少有 4 个符合要求的柱子则根据柱子间的距离来判断柱子分布情况是否唯一，如果唯一则根据距离信息来命名，如果不唯一，则根据上一次柱子的位置信息来命名。

以下情况为分布唯一且符合现实的情况：

1.

2.



给柱子命名的算法代码：

```
void NamePillarAndLocation(float **pillarNow) //给柱子命名
{
    float x1, y1;
    float x2, y2;
    float x3, y3;
    float vx1, vy1;
    float vx2, vy2;
    float vx3, vy3;

    float angle1, angle2;

    int group_finded;
    int left_state;
    int right_state;
```

```
DivideGroup(pillarNow);
right_state = RightState(pillarNow);//对左边柱子的状态分组
left_state = LeftState(pillarNow);//对右边柱子的状态进行分组

//finePillar_group_state = difinePillar_group(pillarNow);    ??left?right?state

InitPillarName();//对 pillara、pillarc、pillard、pillare、pillarf 置零
switch (pillar_num)//根据筛选到的柱子的数量进行命名
{
case 1://有一个柱子的时候无法命名
    printf("1 point cant work");
    break;
case 2:
    matchFromOld(pillarNow,pillarBefore);//与上一次记录完整的六个柱子的信息比较
    break;
case 3:
    if (left_state == 3) {//左边有三个柱子的时候
        pillara = 0;
        pillarb = 1;
        pillarc = 2;
    }
    if (right_state == 3) {//右边有三个柱子的时候
        pillard = 0;
        pillare = 1;
        pillarf = 2;
    }
    if (left_state == 1 && right_state == 0) {
        x1 = pol2rectX(pillarNow[0][1], pillarNow[0][0] * 0.25);
        y1 = pol2rectY(pillarNow[0][1], pillarNow[0][0] * 0.25);

        x2 = pol2rectX(pillarNow[1][1], pillarNow[1][0] * 0.25);
        y2 = pol2rectY(pillarNow[1][1], pillarNow[1][0] * 0.25);

        x3 = pol2rectX(pillarNow[2][1], pillarNow[2][0] * 0.25);
        y3 = pol2rectY(pillarNow[2][1], pillarNow[2][0] * 0.25);

        vx1 = x1 - x2;
        vy1 = y1 - y2;

        vx2 = x3 - x2;
        vy2 = y3 - y2;

        vx3 = x3 - x1;
```

```
        vy3 = y3 - y1;

        angle1 = t_angle(vx1, vy1, vx2, vy2);
        angle2 = t_angle(vx3, vy3, vx2, vy2);

        if (!(angle1 > 70 && angle1 < 110 || angle2>70 && angle2 < 110)) {
            if (t_angle(vx1, vy1, vx2, vy2) < 70) {
                pillard = 0;
                pillara = 1;
                pillarb = 2;
            }
            if (t_angle(vx1, vy1, vx2, vy2) > 110) {
                pillarf = 0;
                pillarb = 1;
                pillarc = 2;
            }
        }
        printf("cant find");
    }
    if (left_state == 1 && right_state == 0) {
        x1 = pol2rectX(pillarNow[0][1], pillarNow[0][0] * 0.25);
        y1 = pol2rectY(pillarNow[0][1], pillarNow[0][0] * 0.25);

        x2 = pol2rectX(pillarNow[1][1], pillarNow[1][0] * 0.25);
        y2 = pol2rectY(pillarNow[1][1], pillarNow[1][0] * 0.25);

        x3 = pol2rectX(pillarNow[2][1], pillarNow[2][0] * 0.25);
        y3 = pol2rectY(pillarNow[2][1], pillarNow[2][0] * 0.25);

        vx1 = x1 - x2;
        vy1 = y1 - y2;

        vx2 = x3 - x2;
        vy2 = y3 - y2;

        vx3 = x3 - x1;
        vy3 = y3 - y1;

        angle1 = t_angle(vx1, vy1, vx2, vy2);
        angle2 = t_angle(vx3, vy3, vx1, vy1);

        if (!(angle1 > 70 && angle1 < 110 || angle2>70 && angle2 < 110)) {
            if (angle1 < 70) {
                pillare = 0;
```

```
                    pillarf = 1;
                    pillarc = 2;
                }
                if (angle1 > 110) {
                    pillard = 0;
                    pillare = 1;
                    pillara = 2;
                }
            }
            else {
                printf("cant find");
            }
        }
        if (left_state == 2 && right_state == 0) {
            x1 = pol2rectX(pillarNow[0][1], pillarNow[0][0] * 0.25);
            y1 = pol2rectY(pillarNow[0][1], pillarNow[0][0] * 0.25);

            x2 = pol2rectX(pillarNow[1][1], pillarNow[1][0] * 0.25);
            y2 = pol2rectY(pillarNow[1][1], pillarNow[1][0] * 0.25);

            x3 = pol2rectX(pillarNow[2][1], pillarNow[2][0] * 0.25);
            y3 = pol2rectY(pillarNow[2][1], pillarNow[2][0] * 0.25);

            vx1 = x1 - x2;
            vy1 = y1 - y2;

            vx2 = x3 - x2;
            vy2 = y3 - y2;

            vx3 = x3 - x1;
            vy3 = y3 - y1;

            angle1 = t_angle(vx1, vy1, vx2, vy2);
            angle2 = t_angle(vx3, vy3, vx2, vy2);

            if (angle1 > 70 && angle1 < 110) {
                pillarf = 0;
                pillara = 1;
                pillarc = 2;
            }

            if (angle2 > 70 && angle2 < 110) {
                pillard = 0;
                pillara = 1;
```

```
            pillarc = 2;
        }
        if (angle1 < 80) {
            pillare = 0;
            pillara = 1;
            pillarc = 2;
        }

    }
    if (left_state == 0 && right_state == 2) {
        x1 = pol2rectX(pillarNow[0][1], pillarNow[0][0] * 0.25);
        y1 = pol2rectY(pillarNow[0][1], pillarNow[0][0] * 0.25);

        x2 = pol2rectX(pillarNow[1][1], pillarNow[1][0] * 0.25);
        y2 = pol2rectY(pillarNow[1][1], pillarNow[1][0] * 0.25);

        x3 = pol2rectX(pillarNow[2][1], pillarNow[2][0] * 0.25);
        y3 = pol2rectY(pillarNow[2][1], pillarNow[2][0] * 0.25);

        vx1 = x1 - x2;
        vy1 = y1 - y2;

        vx2 = x3 - x2;
        vy2 = y3 - y2;

        vx3 = x3 - x1;
        vy3 = y3 - y1;

        angle1 = t_angle(vx1, vy1, vx2, vy2);
        angle2 = t_angle(vx3, vy3, vx1, vy1);
        if (angle1 > 70 && angle1 < 110) {
            pillard = 0;
            pillarf = 1;
            pillara = 2;
        }

        if (angle2 > 70 && angle2 < 110) {
            pillard = 0;
            pillarf = 1;
            pillarc = 2;
        }
        if (angle1 < 80) {
            pillard = 0;
            pillarf = 1;
```

```
                    pillarb = 2;
                }

            }

            break;
            case 4: ……//计算过程一样
            case 5: ……//计算过程一样
            case 6: ……//计算过程一样
        default:
            break;
}
```

3、定位算法具体计算推导过程：

下图 2.6 为雷达在场地上的模拟情况,其中红色线为俯仰角（pitch）不为零、翻滚角（roll）不为零的情况，蓝色为既有俯仰角又有翻滚角的情况。BC 为 B 柱 C 和 C 柱之间的距离，值为 1500。B 为在俯仰角不为零、翻滚角为零时，雷达光线落在 B 柱的点；C 为俯仰角不为零、翻滚角为零时，雷达光线落在 C 柱上的点；BC 为俯仰角不为零、翻滚角为零时，计算得出的 BC 的长度。O 为雷达物理中心，B 为在俯仰角和翻滚角均不为零时雷达光线落在 B 柱上的点，C 为在俯仰角和翻滚角均不为零时雷达光线落在 C 柱上的点，H 为△OBC 内，O 在 BC 上的垂点。

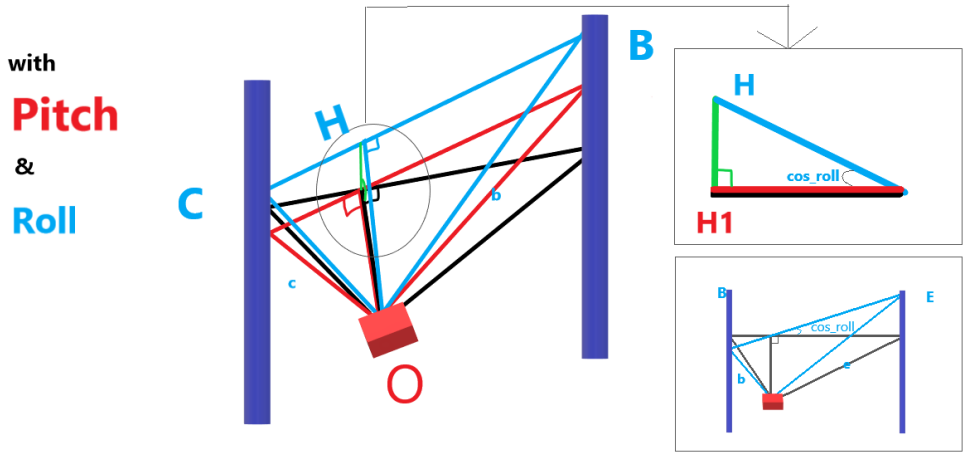


图 2.6 雷达在比赛场地上的 3D 模拟图

（蓝字表示扫描平面上的元素）

（**重要前提**：1、有 roll 也有 pitch 时相对于只有 pitch 的情况，雷达在 BC 上的高 h 与相对于 BC/BC 的高 h/h 存在夹角，夹角值即为 roll（如右上图所示）　　2、有 roll 也有 pitch 时，BC=BC）

已知：b，c，b 与 c 的夹角 cosbc

$BC=\sqrt{b^2+c^2-2bc\cos(bc)}$

$BE(E 为与 B 相对的对方场柱子)=\sqrt{b^2+e^2-2be\cos(be)}$

$\cos C = (c^2 + BC^2 - b^2) / (2c*BC)$

cos_pitch=BC(柱子间固定距离 1500mm)/BC
cos_roll=BE(两边场柱子间固定距离 10850mm)/BE

h(雷达到 BC 的高为 h，垂点为 H)=$\frac{bc\sin(bc)}{BC}$

X=h=h=h*cos_roll
CH1=CH1=c*cosC
Y=real_CH=CH1*cos_pitch
结论：

X=$\frac{b*c*\sin(bc)}{\sqrt{b^2+c^2-2bc\cos(bc)}}$* BE/$(\sqrt{b^2+e^2-2b*e*\cos(be)})$

Y= c* $(c^2+ b^2+c^2-2bc\cos(bc) -b^2)$ *BC/ $(2c*(b^2+c^2-2bc\cos(bc))$

定位算法代码（其一，根据不同的情况会选择不同的函数）：

```
void robot_location_3pitchroll_down(float **pillar, int b, int c, int d, int c_y)
 //e.g. b=pillara,c=pillarb,d=pillare
      // int c_y 即为c 所在y 坐标,单位为mm,如果 c=c,b_y=5000;if c=b,b_y=3500;if
c=a,b_y=2000;
{
      /*
      b
      c              Radar                    d
      */
      float Anglebc = Abs((pillar[b][step] - pillar[c][step]))*0.25;
      float Anglecd = Abs((pillar[c][step] - pillar[d][step]))*0.25;
      float Angleb540 = (pillar[b][step] - 540)*0.25;
      float CD, BC, cos_pitch, CH, real_CH, cos_roll, yaw, b1;          //BE,BC==柱子间测得
的距离，H=垂点,real_CH=距离C 的距离,b1=去掉roll 之后的b
      float cosC, cosB;                                                 //真实夹角
      BC = sqrt(pow(pillar[b][dist], 2) + pow(pillar[c][dist], 2) - 2 *
(pillar[b][dist])*(pillar[c][dist])*cos(Anglebc*PI / 180));
      CD = sqrt(pow(pillar[c][dist], 2) + pow(pillar[d][dist], 2) - 2 *
(pillar[c][dist])*(pillar[d][dist])*cos(Anglecd*PI / 180));
      //计算柱子间测得的距离
      cosC = ((float)pow(BC, 2) + (float)pow(pillar[c][dist], 2) - (float)pow(pillar[b][dist], 2)) /
(float)(2 * pillar[c][dist] * BC);
      cosB = ((float)pow(BC, 2) + (float)pow(pillar[b][dist], 2) - (float)pow(pillar[c][dist], 2)) /
(float)(2 * pillar[b][dist] * BC);
      cos_pitch = (1500 / BC);
      cos_roll = (10850 / CD);
      CH = pillar[c][dist] * cosC;

      real_CH = CH * cos_pitch;

      mr2_x[0] = (pillar[b][dist] * pillar[c][dist] * sin(Anglebc*PI / 180) / BC)*cos_roll + 1225;
```

```
    mr2_coordinate_x = mr2_x[0];



    mr2_y[0] = c_y - real_CH;

    mr2_coordinate_y = mr2_y[0];

    yaw = (acos(cosB)) * 180 / PI - Angleb540;   //角度制yaw(偏航)
    printf("雷达的 x 坐标:%f\n 雷达的 y 坐标:%f\n\n  偏航 yaw:%f\n", mr2_coordinate_x,
mr2_coordinate_y, yaw);
}
```

4、六点定位算法流程概述：

（1）根据雷达返回的距离信息筛选 9m 内的障碍物并计算其宽度。

（2）筛选出宽度在阈值内的障碍物。

（3）计算宽度符合要求的障碍物之间的距离，如果障碍物之间的距离和分布符合以下要求之一即判断为柱子：

1.两侧的障碍物之间的距离符合要求，单侧各个柱子之间的距离符合要求，两侧柱子所在的直线平行

2.其中一侧的柱子有三个且在同一直线上

（4）根据柱子的分布情况判断情况是否唯一，若不唯一，则根据上一次记录的柱子信息来命名柱子并计算返回位置信息；若分布情况唯一，则对识别得到的柱子命名、记录本次柱子分布情况、计算并返回位置信息。

如果没有识别得到的柱子，则返回上一次的位置信息。

5、六点定位算法优缺点：

优点：可以在不借助其他硬件的情况下得出翻滚角（roll）、俯仰角（pitch）、偏航角（yaw），得出较为精确的位置信息，经过测试，本算法的误差为 1cm

缺点：算法复杂，调试麻烦，对环境要求较高，因为要结合对方场地来进行判断，而比赛时对方场地上不知道会发生什么，对方的机器人可能会被误判为柱子，所以有可能会出错。

下图 2.7 为六点定位算法流程图：

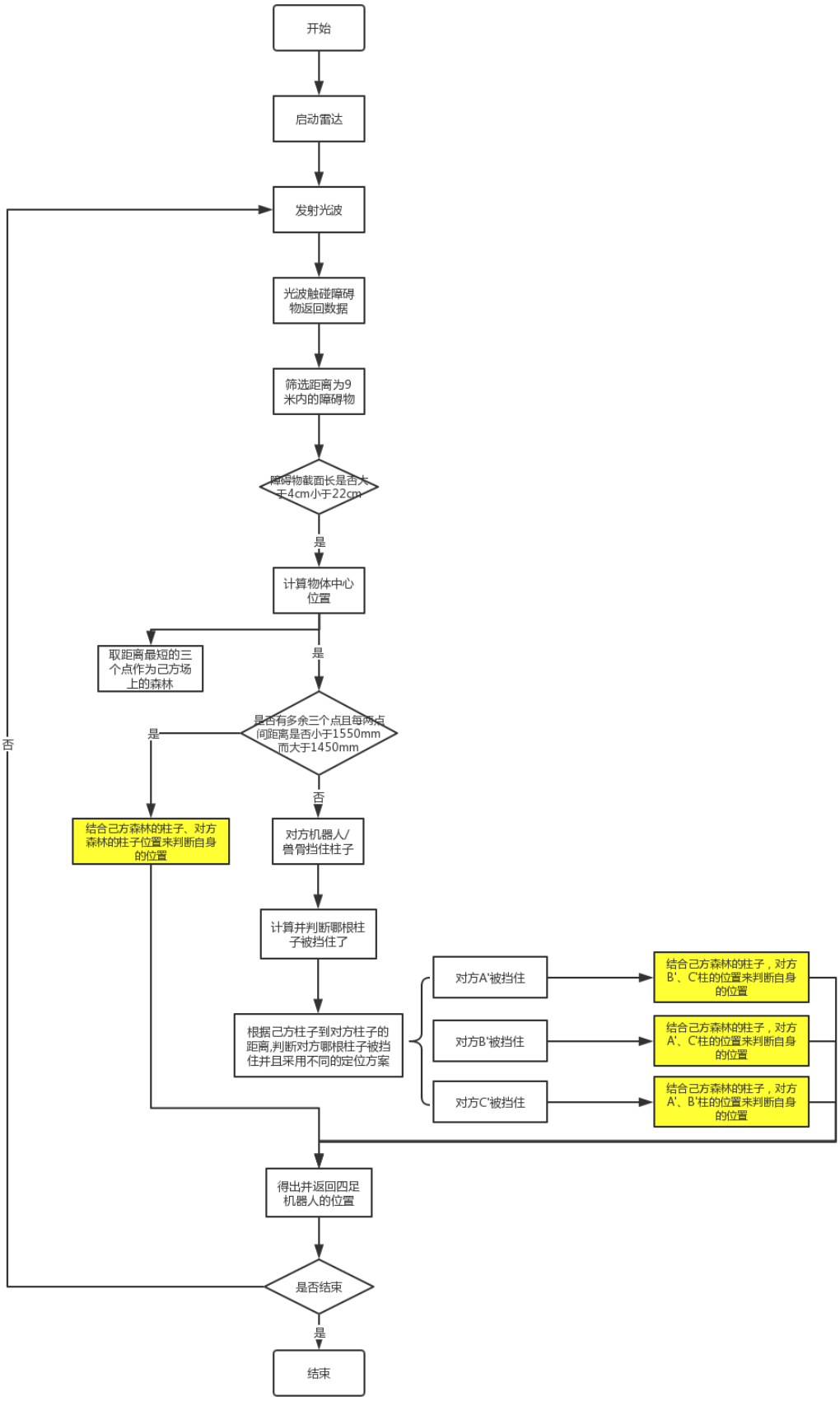


图 2.7 六点定位算法流程图

2.1.4 雷达定位方案二

根据己方场地的柱子，结合陀螺仪的翻滚角（roll）、俯仰角（pitch）信息，利用余弦定理来进行定位。

1、可行性分析：

因为己方场地上的柱子到四足机器人的距离要比对方场地的柱子到四足机器人的距离要近得多（见上图 2.2 比赛场地图），而地图上最远的柱子也能被扫描得到，所以己方场地上的柱子也一定可以扫描得到，证明方法及过程见 2.1.3 雷达定位方案一中的可行性分析。

2、识别并分辨柱子的算法流程：

（1）识别障碍物的算法流程同 2.1.3 雷达定位方案一中的识别障碍物算法流程

（2）识别柱子的算法流程同 2.1.3 雷达定位方案一中的识别柱子算法流程

（3）给柱子命名的算法流程：

判断各个柱子之间的距离是否符合要求，是则按先后顺序命名。否则根据筛选后的障碍物与上一次识别到的柱子进行比较，与上一次识别到的柱子相近的障碍物为上一次对应的柱子。

3、定位算法推导过程同 2.1.3 雷达定位方案一中的定位算法推导过程。

4、三点定位算法流程概述：

（1）根据雷达返回的距离信息筛选 7m 内的障碍物并计算其宽度

（2）筛选宽度在阈值内的障碍物

（3）计算并判断是否存在三个障碍物位于同一直线并且相邻两个障碍物之间的距离符合要求，是则按顺序命名并保存本次柱子信息，否则将本次筛选后符合宽度要求的柱子与上一次扫描到的柱子比较，并将障碍物命名为上一次相近的柱子。如果命名失败（没有与上一次扫描到的柱子相近的障碍物）则返回上一次的位置信息。

（4）如果识别到三根柱子，则缩小雷达的扫描范围，以免产生误判。

5、三点定位算法优缺点：

优点：算法简单，定位稳定，对四足的姿态要求相对宽松，重启也能进行定位，不会被对方场上的机器人干扰

缺点：要依靠其他硬件辅助

下图 2.8 为三点定位算法流程图

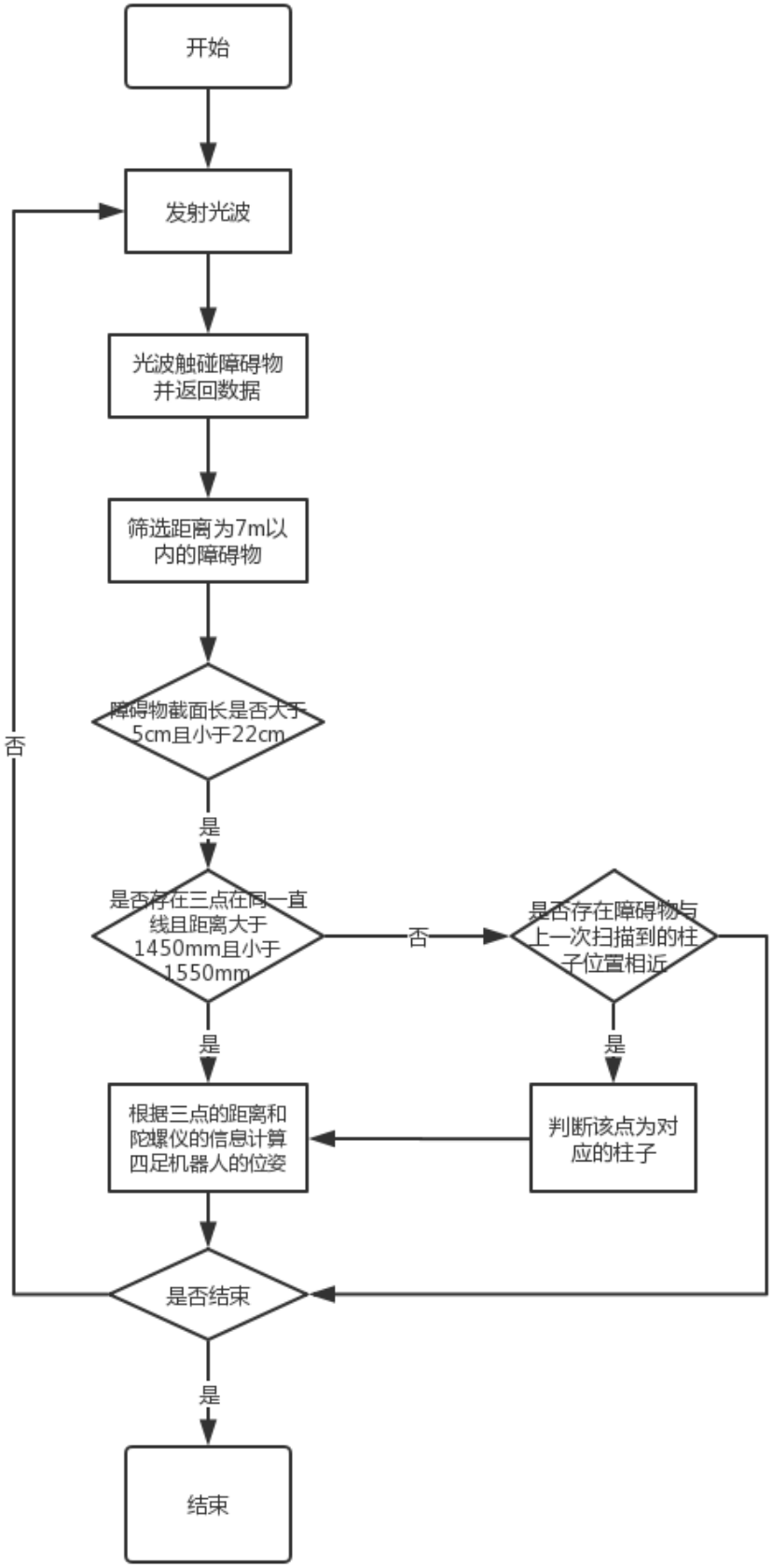


图 2.8 三点定位算法流程图

3 后勤工作

在本学期内，本人除了编写雷达定位算法外，还负责了部分后勤工作，主要包括有：

（1）寻找木工搭建场地、编写搭建场地合同。

（2）在假期中提前回来对木工的工作进行监工。

（3）帮助机械组搭建车体（包括自动机器人 MR1 和四足机器人 MR2）。

（4）比赛前场地的日常维护，主要是防雨防水.

4 本学期学习总结

1、打代码方面

（1）在明确自己的需求之后，最好先找一个可以成功运行的 demo 或者成熟的模版，在此基础之上改；尽量使用使用群体比较大的开发软件，这样能尽量减少遇到的问题或者能容易找到对应的解决方法。

（2）因为 Robocon 是一个争分夺秒的比赛，所以要尽量在细节优化每一步，代码中多余的步骤应该尽量减少，在重复次数比较多的程序中尽量减少运算量。当然，除了 Robocon 比赛，对于自己写的程序，也坚持能免则免原则。

（3）即便是简单的需求，也可以去搜一下其他人是怎么做到的，看看人家的算法与自己的算法之间有哪些差别，学会融会贯通。

2、在团队合作方面

（1）在让其他人帮忙写小程序之前，除了需求，还应当说明使用背景和注意事项。

（2）有时候应该分担一些工作给其他人，而不是全部自己一个人完成。

（3）做项目之前，应当先问有经验的人，工作分配是否合理，目标是否合理。