期末课程设计

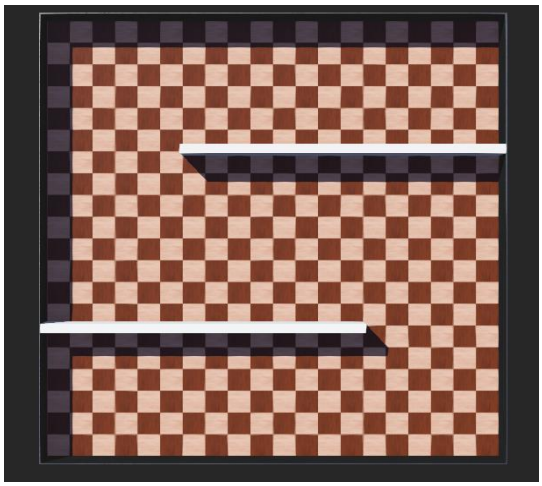
姓名	学号	分工
郝裕玮	18329015	局部路径规划+仿真调试+论文排版
马淙升	19335153	雷达感知+建立云图
杨荣杰	19335242	GPS 控制导航

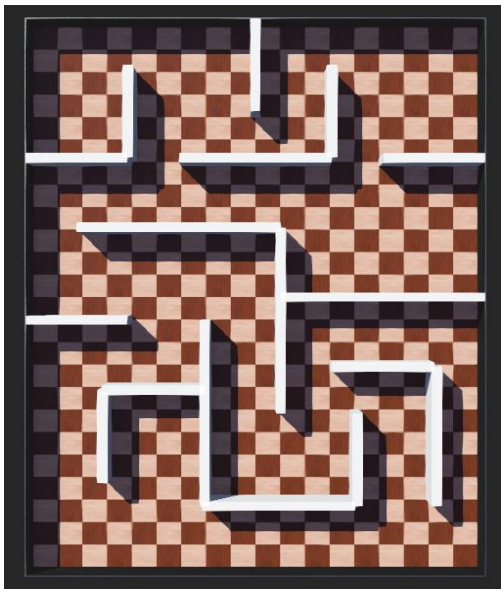
目录

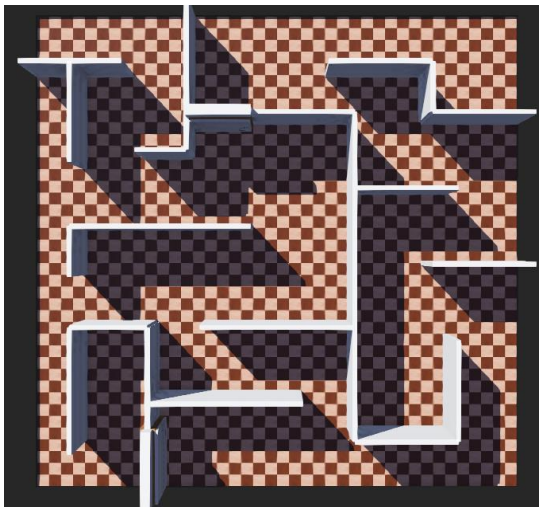
1 实验目标

实验要求：完成感知、规划、控制算法。在未知环境中，控制小车从起点运动到终点。

实验场景：共有 3 个 world，如下所示（其中 world3 含有可自动开关的门）：







2 实验内容与分析

2.1 环境配置

编程语言：C++

第三方库：OpenCV

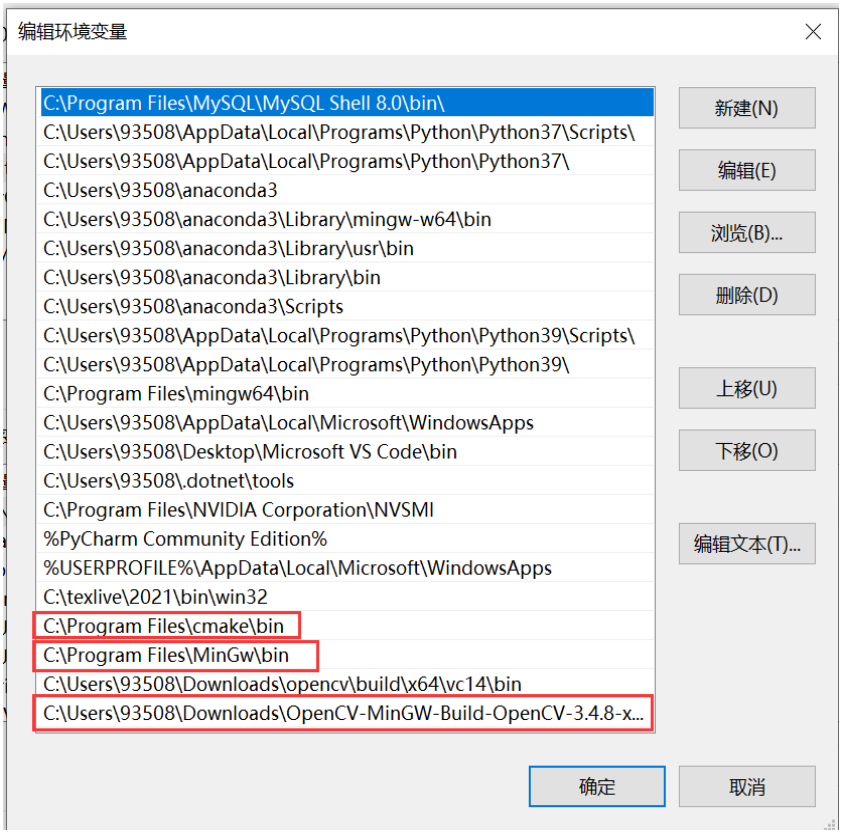
Webots 控制器代码调用 OpenCV 库的配置过程如下：

（1）下载 cmake

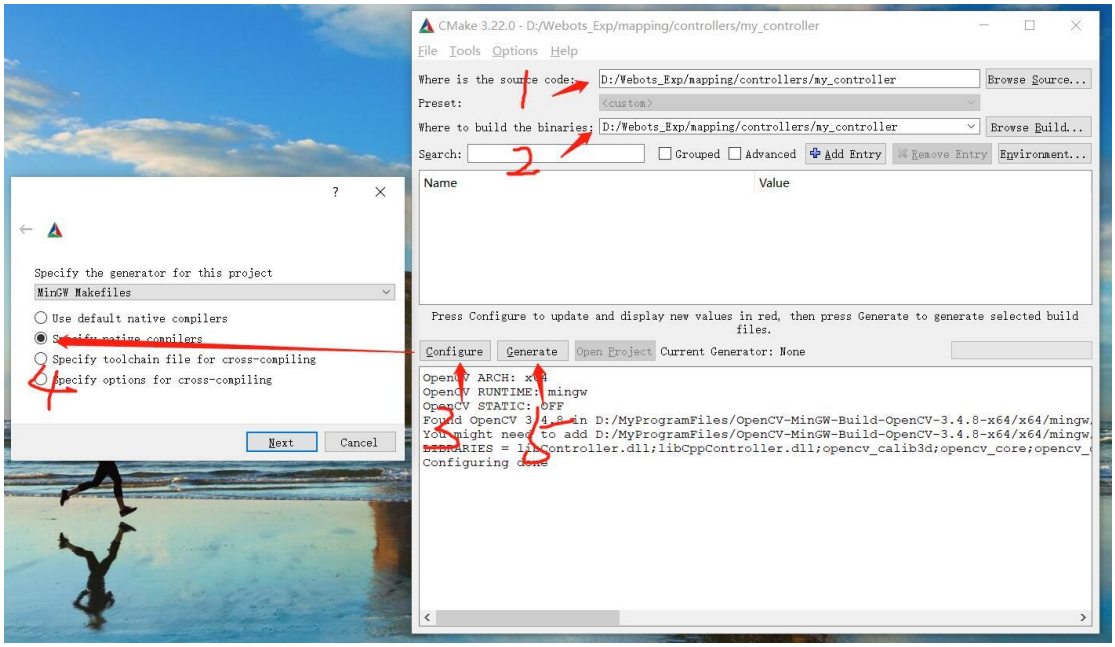
（2）下载 OpenCV 已编译好的安装包（https://github.com/huihut/OpenCV-MinGW-Build/tree/OpenCV-3.4.8-x64）

（3）下载 MinGW，版本为：x86_64-8.1.0-release-posix-seh-rt_v6-rev0（注意必须为 posix 版本，否则后续对 OpenCV 进行编译时会报错）

（4）将（1）（2）（3）步骤中的安装包路径添加到环境变量。



（5）在控制器代码所在的文件夹下新建 CMakeLists，编写相关内容（详见 CMakeLists 内部注释）。再打开 cmake-gui，按顺序进行下图操作：



其中 1,2 对应的是控制器代码文件所在目录，点击 4 之后需要添加 gcc 和 g++的目录（查找 mingw 目录下的 bin 文件夹即可找到 gcc 和 g++）。在点击 Generate 之后会发现控制器代码所在文件夹下已生成 makefile 文件，在当前目录下进行命令行编译：mingw32-make 即可。编译完成后打开代码对应的 world 即可仿真运行。

2.2 Robot 节点配置

我们需要在以前实验的基础上对 Robot 节点添加 1 个 lidar 和 2 个 GPS（添加 2 个 GPS 的原因会在后续说明）

2.3 控制器代码思路分析

2.3.1 流程总结

控制器代码的逻辑结构如下：

（1）雷达扫描，建立点云图

（2）读取最新的点云图，进行路径规划

（3）将规划路径的结果数组从像素点数据转为 GPS 数据

（4）GPS 控制导航，行驶规划路径的一部分，之后结束控制，跳回（1）重新建立新的点云图，开始新一轮的循环。（1）—（4）一直循环，直到抵达终点。

2.3.2 感知

（1）雷达部分：

这部分有两份代码，一份是自己编写的，另一份是助教提供的，接下来我们会对这两份代码分别进行说明。

在编写代码前，我们首先需要在 Webots 上添加雷达部件，我们给该部件命名以便控制器调用，并将雷达扫描方式改为全局扫描，修改雷达扫描范围（我们设置为 3）。

对 Webots 上的部件组装完成后，我们需要在控制器中调用雷达部件，首先是雷达句柄的获得和相关功能的使能。代码如下：

```
Lidar* ld = robot -> getLidar("a");
ld -> enable(1);
ld -> enablePointCloud();
```

使能后我们就可以调用雷达的相关接口获得雷达的探测信息，即每次激光探测到与障碍物的距离。我们获得的障碍物时相对于小车当前的位置，需要将它转化为实际的 GPS 值，这部分转化公式如下：

$$GPS_x = GPSCar_x + dis * \cos(p + r)$$
$$GPS_y = GPSCar_y + dis * \sin(p + r)$$

其中，p 为雷达初始转角，r 为雷达在全局探测过程中的转角。

为了获得当前的 p 值，我们在小车的车头和车尾分别安装了一个 GPS，记为 A、B，则初始偏角为（见下页）：

$$p = \begin{cases} \frac{\pi}{2} \quad , GPSA_x = GPSB_x \\ \arctan\left(\frac{GPSA_y - GPSB_y}{GPSA_x - GPSB_x}\right) \end{cases}$$

经过这些转化过程，我们便可以获得小车周围的障碍物情况，但是，如此设计过程在实践中遇到了一些问题——雷达探测的偏差。我们发现，调用接口

```
getRangeImage();
```

返回的距离并不完全与实际情况相同：

1、对于一些距离比较近的障碍物，雷达没办法探测到实际的距离，返回的结果为 0.

2、有些情况下，虽然有障碍物，但雷达返回的结果为 INF.

距离为 0 和 INF 的数据占据了探测结果的相当大的一部分，所以我们在进行建图的过程中遇到了比较大的困难。

实验初期我们提出了以下的解决思路：

1、丢弃所有为 0、INF 的数据，保证结果的准确性。

2、利用 0 和 INF 的数据，若在规划时遇到了“撞墙”的情况，再进行相应的处理。

由于我们返回的数据中 0 和 INF 占据了相当大的一部分（一半左右），使用方法一虽然实现比较简单，但是效率较低，所以我们选择了第二种方案。

同时，我们观察到再雷达射线初始位置的直线附近，探测结果比较准确，而在垂直于初始雷达射线的方向上有较多的无用数据，我们根据这个特点，增加了一个雷达，这个雷达初始射线的方向于第一个雷达初始射线方向垂直，利用这两个雷达同时进行更新，取得了比较良好的效果。

之后助教给了另一份代码，思路更加清晰，我们在实验中也结合了这份代码的思路，接下来我们对这份代码的思路做简单介绍：

助教的实现思路与我们类似，但调用的接口有所不同，助教调用的函数接口为：

```
    getRollPitchYaw();
```

根据获得结果我们仍然需要将相对位置转化为绝对位置并生成点云图用于后续的局部路径规划，这部分转化过程如下：

```
mapPointAngle = carAngle + M_PI - double(i) / double(lidarRes) * 2.0 *
M_PI;
mapPointX = lidarImage[i] * cos(mapPointAngle);
mapPointY = lidarImage[i] * sin(mapPointAngle);
mapPointX += carX;
mapPointY += carY;
```

这部分实际上与我们一开始的更新过程相同。

此外，助教的代码中增加了离群检测，如下：

```
double outlierCheck = 0.0;
for (int k = i - outlierCnt; k < i + outlierCnt; ++k)
{
    outlierCheck += abs(lidarImage[i] - lidarImage[k]);
}
outlierCheck /= (2 * outlierCnt);
if (outlierCheck > 0.1 * lidarImage[i])
    continue;
```

对于 GPS 点和像素点的转化，我们为了将这个过程抽象便于使用，我们编写了两个函数：

```
//像素点位置转为图中对应的 GPS 位置,便于后续 GPS 控制导航
void PixelToGps(int Pixelx,int Pixely,double& outx,double& outy) {
      outx = 1.0*Pixelx/world2pixel - worldWidth / 2.0;
      outy = 1.0*(mapHeight-Pixely)/world2pixel - worldWidth / 2.0;
}


//GPS 位置转为像素点位置,便于路径规划
void GpsToPixel(double Gpsx,double Gpsy,int& outx,int& outy) {
      outx = int((Gpsx + worldWidth / 2.0) * world2pixel);
      outy = mapHeight-int((Gpsy + worldWidth / 2.0) * world2pixel);
      if(outx < 0) outx = 0;
      if(outy < 0) outy = 0;
      if(outx > 499) outx = 499;
      if(outx > 499) outx = 499;
}
```

我们使用助教的离群点检测和使用 getRollPitchYaw 接口，与我们之前的代码结合，最终构成了我们提交的版本：大体思路使用了我们的初始思路，但在生成点云图的过程中使用了助教提供的离群点检测和使用 getRollPitchYaw 接口。

2.3.3 规划

该部分主要是根据获取的点云图进行 RRT 路径规划，并将规划路径结果序列（像素点坐标数组转换为 GPS 坐标数组）传给 GPS 用于控制导航。

具体代码如下（具体分析均已包含在代码注释中）：

（1）参数说明：

```
// 地图参数
//地图以像素为单位时的长和宽
const int mapHeight = 500;
const int mapWidth = 500;
//地图以距离(m)为单位时的长和宽
const double worldHeight = 5;
const double worldWidth = 5;
//像素—距离转换公式
const double world2pixel = mapHeight / worldHeight;

vector<tuple<int, int,int> > node;//存储路径结果序列(终点到起点)
vector<tuple<int, int,int> > node1;//存储路径结果序列(起点到终点)
int row, col;//存储图片的行数和列数
int endr, endc;//存储终点的像素点位置
```

（2）调用函数部分：

```
//像素点转 GPS
void PixelToGps(int Pixelx,int Pixely,double& outx,double& outy) {
    outx = 1.0*Pixelx/world2pixel - worldWidth / 2.0;
    outy = 1.0*(mapHeight-Pixely)/world2pixel - worldWidth / 2.0;
}

//GPS 转像素点
void GpsToPixel(double Gpsx,double Gpsy,int& outx,int& outy) {
    outx = int((Gpsx+ worldWidth / 2.0) * world2pixel);
    outy = mapHeight-int((Gpsy+ worldWidth / 2.0) * world2pixel);
    if(outx<0){
        outx=0;
    }
    if(outy<0){
        outy=0;
```

```
    }
    if(outx>=500){
        outx=499;
    }
    if(outy>=500){
        outy=499;
    }
}

//判断当前点是不是墙
bool is_wall(int r, int c) {
    if (r < 0 || c < 0 || r >= row || c >= col) {
        return true;
    }
    return image.at<Vec3b>(c, r)[0] < 50 && \
        image.at<Vec3b>(c, r)[1] < 50 && \
        image.at<Vec3b>(c, r)[2] < 50;
}

//判断两点间的连线是否存在墙
bool check(tuple<int, int,int>node1, tuple<int, int,int>node2) {
    int step = 100;
    double a = get<0>(node1), b = get<1>(node1);
    double deta_a = 1.0 * (get<0>(node2) - get<0>(node1)) / step;
    double deta_b = 1.0 * (get<1>(node2) - get<1>(node1)) / step;
    for (int i = 0; i < step; ++i) {
        a += deta_a;
        b += deta_b;
        if (is_wall(a, b)) return false;
    }
    return true;
}

//判断两点是否合法以及两点间的连线是否存在墙
bool ok(tuple<int, int,int>a, tuple<int, int,int>b) {
    if (get<0>(a) < 0 || get<0>(b) < 0 || get<0>(a) >= row ||
get<0>(b) >= row \
        || get<1>(a) < 0 || get<1>(b) < 0 || get<1>(a) >= col ||
get<1>(b) >= col \
        || is_wall(get<0>(a), get<1>(a)) || is_wall(get<0>(b),
get<1>(b)))
        return false;
    return check(a, b);
}
```

```
//RRT 规划
vector<tuple<int, int,int>> build_graph(double start1,double end1) {
    //对于 tuple<int,int,int>元组的 3 个 int 型变量依次代表:像素横坐标,像素纵
坐标,与该像素点连线的父节点序号
    //将当前所在位置的 GPS 转为像素点,并加入到 node 向量中
    int start,end;
    GpsToPixel(start1,end1,start,end);
    node.push_back({ start, end,-1 });

    //进行 RRT 树的生长
    //near:当前离终点最近的节点的序号
    //steps:撒点数量
    //length:每次生长的步长
    //rate:向随机点和终点生长的概率分布
    int near = 0;
    int steps = 100000, length = 40;
    double rate = 0.5;
    srand((unsigned)time(NULL));
    while (steps--) {
        int m = rand() % 100;
        double p = 1.0 * m / 100;
        bool flag = true;
        //向终点方向生长
        if (p < rate) {
            //通过三角形的比例法则得出生长后的点的坐标(nr,nc)
            int r = get<0>(node[near]), c = get<1>(node[near]);
            double p = sqrt((r - endr) * (r - endr) + (c - endc) * (c -
endc));
            int nr = r + length / p * (endr - r);
            int nc = c + length / p * (endc - c);
            //若两点间连线不存在墙
            if (ok({ nr,nc,near }, node[near])) {
                for (int i = 0; i < node.size(); ++i) {
                    //若当前已经存在的点与新生长的点的距离均大于 10 则将该点加
入到 RRT 树中
                    //反之则认为新生长的点与某一点重合,跳过此次循环
                    int m = (get<0>(node[i]) - nr) * (get<0>(node[i]) -
nr) + (get<1>(node[i]) - nc) * (get<1>(node[i]) - nc);
                    if (m < 100) {
                        flag = false;
                        break;
                    }
                }
```

```
                //若当前已经存在的点与新生长的点的距离均大于 10 则将该点加入到
RRT 树中
                if (flag == true) {
                    node.push_back({ nr,nc,near });
                    near = node.size() - 1;
                }
            }
            flag = true;
        }
        //向随机点方向生长
        else {
            //生成新随机点(r1,c1)
            int r1 = rand() % row, c1 = rand() % col;
            int index = -1;
            int k = 10000000;
            //找出 RRT 树中与新随机点距离最近的节点
            for (int i = 0; i < node.size(); ++i) {
                int m = (get<0>(node[i]) - r1) * (get<0>(node[i]) - r1)
+ (get<1>(node[i]) - c1) * (get<1>(node[i]) - c1);
                if (m < k) {
                    k = m;
                    index = i;
                }
            }
            //若距离太近,小于 10,则认为新随机点与某一点重合,跳过此次循环
            if (k < 100) {
                continue;
            }

            //通过三角形的比例法则得出生长后的点的坐标(nr,nc)
            int r = get<0>(node[index]), c = get<1>(node[index]);
            double p = sqrt((r - r1) * (r - r1) + (c - c1) * (c - c1));
            int nr = r + length / p * (r1 - r);
            int nc = c + length / p * (c1 - c);
            //若两点间连线不存在墙
            if (ok({ nr,nc,index }, node[index])) {
                //若当前 RRT 树中的 node[near](即离终点最近的节点)与终点的距
离 > 新生长点与终点的距离
                //则 near+1
                //最后再将该点加入到 RRT 树中(不管距离关系如何,只要连线合法就
可以将其加入 RRT 树)
                int a = (endr - get<0>(node[near])) * (endr -
get<0>(node[near])) + (endc - get<1>(node[near])) * (endc -
get<1>(node[near]));
```

```
                int b = (endr - nr) * (endr - nr) + (endc - nc) * (endc
- nc);
                if (a > b) near = node.size();
                node.push_back({ nr,nc,index });
            }
        }
        //若当前 RRT 树中的最新节点离终点的距离 < 根号 450(20 多),则终止撒点,开
始连线
        if (node.size()>1&&(endr - get<0>(node[node.size()-1])) * (endr
- get<0>(node[node.size() - 1])) + (endc - get<1>(node[node.size() -
1])) * (endc - get<1>(node[node.size() - 1])) < 450) {
            node.push_back({ endr,endc,near });
            break;
        }
    }

    int a=node.size()-1;
    vector<tuple<int, int,int> > Node;//存储路径结果序列(终点到起点)
    vector<tuple<int, int,int> > Node1;//存储路径结果序列(起点到终点)
    //利用回溯法画出终点到起点的路线(因为每个节点的父节点是唯一的)
    while (get<2>(node[a]) != -1) {
        int b = get<2>(node[a]);
        line(image, Point(get<0>(node[a]), get<1>(node[a])),
Point(get<0>(node[b]), get<1>(node[b])), Scalar(0, 0, 0), 5);
        Node.push_back(node[a]);
        Node.push_back(node[b]);
        a = b;
    }
    //将 Node 数组倒序赋值给 Node1 数组
    int len=Node.size();
    for(int i=0;i<=len-1;i++){
        Node1.push_back(Node[len-1-i]);
    }
    return Node1;
}
```

RRT 算法的原理如下：

①随机采样：每次选择生长方向时，有一定的概率会向着目标点延伸，也有一定的概率会随机在地图内选择一个方向延伸一段距离。

②生长点选择与碰撞检测：假设我们采样了图像中的某个随机点，那么我们可从现有的 RRT 树中筛选出离该采样点最近的一个点，并向采样点生长一段距离（长为 step_size）。并对检验生长过程中是否发生碰撞（两点间连线有无障碍点），最后还要保证该采样点与 RRT 树中现有的所有点的距离均大于距离阈值 area（若两点距离小于 area 则默认为两个点重合，是同一个点）。若以上 2 个条件均满足，则将该采样点加入 RRT 树中。

③同理，当某个采样点与终点的距离＜area 时，则认为已经到达终点，将其加入 RRT 树之后可终止循环。

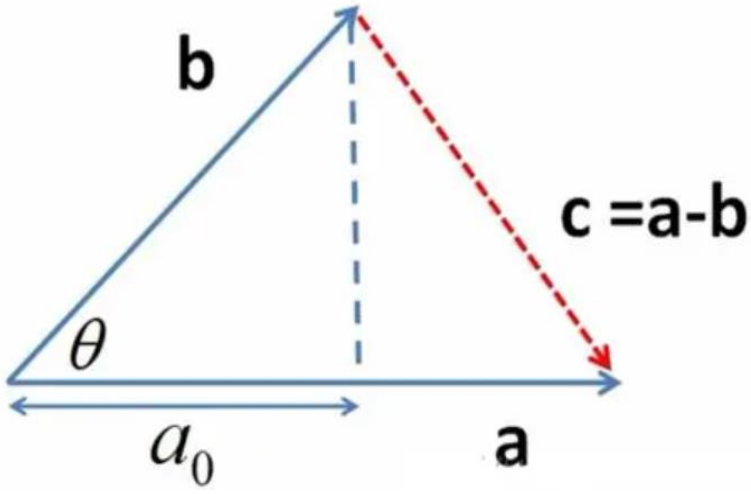
2.3.4 控制

GPS 控制小车移动的基本思路如下：

GPS 获取坐标信息—>转换成相对位置信息—>转换成向量信息—>转换成方向信息—>转换成具体的控制命令

需要用到的数学理论知识如下：

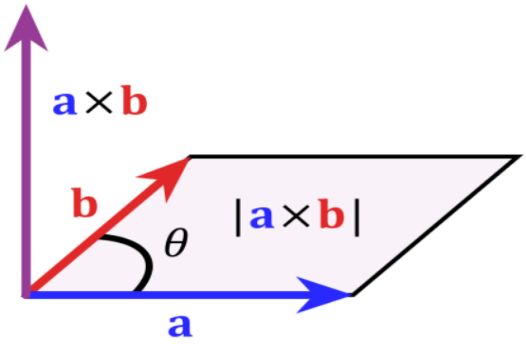
（1）向量夹角的意义：



由图可得公式$\cos\theta = \frac{ab}{|a||b|}$

即两个向量的夹角的余弦值可以由上面的公式求得。这是我们后面工作的重要数学依据。

（2）向量叉乘的几何意义：



- 二维向量叉乘：$(x_1, y_1) \times (x_2, y_1) = x_1y_2 - x_2y_1$
- 值的绝对值是两向量同起点，构成平行四边形的面积
- 值为正，(x_2, y_2)在(x_1, y_1)逆时针方向
- 值为负，(x_2, y_2)在(x_1, y_1)顺时针方向
- 值为 0，(x_2, y_2)和(x_1, y_1)共线

所以控制的具体思路如下：

当我们规划好路径上需要走的点的时候，我们需要利用 GPS 控制小车。

我们需要两个方向：一个是小车前进的方向 a，一个是小车向下一个目标的方向 b。

（1）若在同一直线上或偏差不大时我们就直行。

（2）如果 b 在 a 偏左方向，我们就让小车往左，否则让其往右。

那么我们应该如何得到这两个方向呢？我们需要借助 GPS 获取的

信息，转化成我们需要的位置信息。

具体方法为：我们在小车车身和车头各安装一个 GPS，获取两者的位置信息。我们以车身 GPS 到车头 GPS 的方向作为车前进的方向。坐标相减得出向量，向量的方向作为小车前进的方向。同样的方法我们也可以获取小车与下一个目标点的角度和方向。

我们计算这两个向量的夹角 θ：

（1）若 θ 接近于 0 时，也就是 cos θ 接近于 1，我们认为在同一直线上，继续前进即可（若 cos θ 接近于-1，在其正后方，并不应该继续向前）。

（2）我们计算它们的叉乘：

若叉乘结果 cross > 0，则目标点在小车的逆时针方向，我们让小车向左转。反之则在顺时针方向，我们让小车向右转。

具体代码如下（具体分析均已包含在代码注释中）：

```
        //GPS 控制导航
        //设置小车运动方向的各个轮子的速度
        double alpha=0.05;
        double speedForward[4] = {v, v, v, v};
        double speedBackward[4] = {-v, -v, -v, -v};
        double speedRightward[4] = {alpha* v, alpha * v, -alpha*v, -
alpha*v}; //重要
        double speedLeftward[4] = {-alpha*v, -alpha*v, alpha * v, alpha*
v};
        int target=1;
        int count=1;
        //每次只走路线规划中的前 3 个点
        //走完后跳出循环,进行下一次雷达扫描,重新建立点云图进行新一次的路径规划
        while(robot->step(timeStep) !=-1 && target<=3){
            double cur_x=0,cur_y=0,head_x=0,head_y=0;
            double gps_x,gps_y,head_gps_x,head_gps_y;
            gps_x=gps[0]->getValues()[0];
            gps_y=gps[0]->getValues()[1];
```

```
            head_gps_x=gps[1]->getValues()[0];
            head_gps_y=gps[1]->getValues()[1];
            //若有非法值则进行下一次循环,重新获取 GPS 值
            if(isnan(gps_x)||isnan(gps_y)||isnan(head_gps_x)||isnan(head
_gps_y)){
                continue;
            }

            cur_x=gps_x;
            cur_y=gps_y;
            head_x=head_gps_x;
            head_y=head_gps_y;

            int maxstep=500;//时间阈值
            double k=0.05;//阈值

            //判断是否到达目标点，到达则更新为下一个目标点，否则记录经过了一个
时钟周期
            if (dis(cur_x, cur_y, nodex[target], nodey[target]) < k)
            {
                target++;
                count=0;
                continue;
            }
            else{
                count++;
            }

            //若 500 个时钟周期过后发现仍为到达下一个目标点
            //则跳出循环,进入到下一次的雷达扫描当中
            //即开始重新规划路线
            if(count>maxstep){
                break;
            }
            int keyValue = 0;

            // 通过计算点乘计算余弦值
            double dot = (head_x - cur_x) * (nodex[target] - cur_x) +
(head_y - cur_y) * (nodey[target] - cur_y);
            double norm = dis(0, 0, head_x - cur_x, head_y - cur_y) *
dis(0, 0, nodex[target] - cur_x, nodey[target] - cur_y);
            double res = dot / norm;

            // 计算叉乘
```

```
            double cross = (head_x - cur_x) * (nodey[target] - cur_y) -
(head_y - cur_y) * (nodex[target] - cur_x);

            if (res >= 0.995)// 直走
            {
                keyValue = 'W';
            }

            else if (cross > 0)// 左转
            {
                keyValue = 'A';
            }
            else if (cross < 0)// 右转
            {
                keyValue = 'D';
            }

            //设置小车的行动
            if (keyValue == 'W')
            {
                for (int i = 0; i < 4; ++i)
                {
                    speed[i] = speedForward[i];
                }
            }
            else if (keyValue == 'S')
            {
                for (int i = 0; i < 4; ++i)
                {
                    speed[i] = speedBackward[i];
                }
            }
            else if (keyValue == 'A')
            {
                for (int i = 0; i < 4; ++i)
                {
                    speed[i] = speedLeftward[i];
                }
            }
            else if (keyValue == 'D')
            {
                for (int i = 0; i < 4; ++i)
                {
                    speed[i] = speedRightward[i];
```

```
            }
        }
        else
        {
            for (int i = 0; i < 4; ++i)
            {
                speed[i] = 0;
            }
        }
        for (int i = 0; i < 4; ++i)
        {
            motors[i]->setVelocity(speed[i]);
        }

        //将终点转换为 GPS 坐标
        //计算当前点和终点的距离
        //若小于 0.25 则说明已经抵达终点,跳出循环
        double endx,endy;
        PixelToGps(endr,endc,endx,endy) ;
        if(dis(cur_x,cur_y,endx,endy)<=0.25){
            flag=true;
            break;
        }
    }
```

补充说明：

（1）更新为下一个节点的判断：若我们走到了目标点的附近，我们就把目标点更改为下一个目标点，这样子更合理，不必苛求一定要走到目标点，可以保证正确性的同时加快运行时间。

```
        if (dis(cur_x, cur_y, nodex[target], nodey[target]) < k)
        {
            target++;
            count=0;
            continue;
        }
        else{
            count++;
        }
```

我们设置阈值 k=0.05,意思是小车与目标点距离在 0.05 以内时我们认为到达了目标点，更新为下一个目标点。

（2）无法到达目标点的判断：

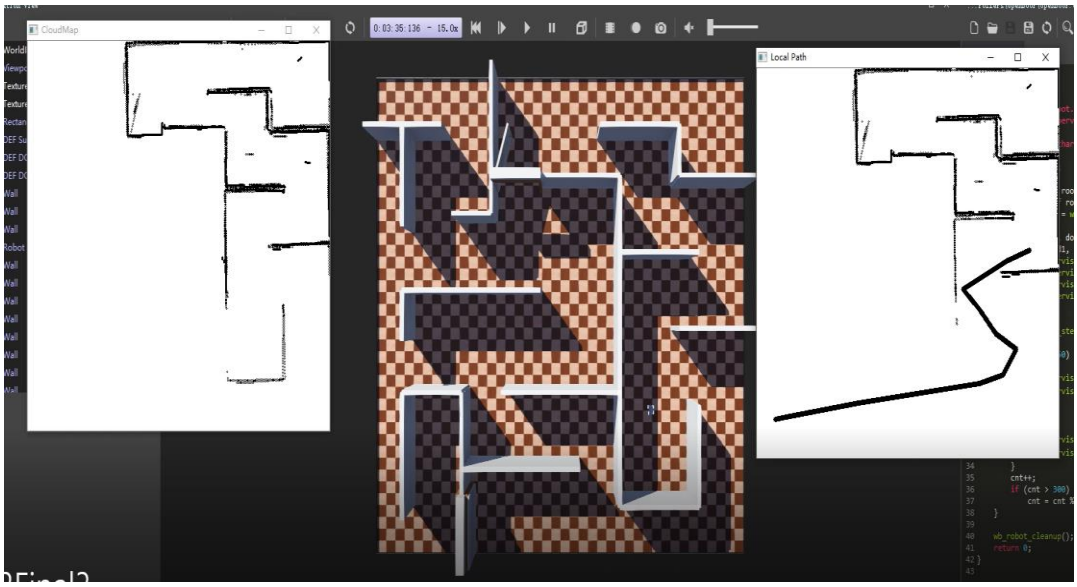
当我们一直无法走到下一个目标点的时候，极有可能是小车在行驶过程中被墙卡住（即路径规划图中显示这里没有墙，从而规划了一条穿墙的路线，导致小车一直在向墙另一边的目标点行驶）。所以我们设定：若小车在 maxstep 个时钟周期后还是无法到达下一个目标点，则我们选择停止控制，跳出循环，重新进行雷达扫描，建立新的云图并规划路径。

```
            //若 500 个时钟周期过后发现仍为到达下一个目标点
            //则跳出循环,进入到下一次的雷达扫描当中
            //即开始重新规划路线
            if(count>maxstep){
                break;
            }
```

3 实验结果

world1-3 的视频均已放在压缩包中，这里仅展示部分运行截图（见下页）。

补充说明：（1）由于图片的生成先后问题，所以视频中小车的行驶路线与展示的局部路线规划图是有时间先后差异的（即小车每次行驶完一部分路线之后，局部路线图才会更新显示为刚刚小车走过的路线）



（2）world1 和 world2 的 worldHeight 和 worldWidth 参数一致，均为（5,5）。

```
const double worldHeight = 5;
const double worldWidth = 5;
```

但是在运行 world3 时需要将上面两个数据均修改为 10。并重新使用命令行进行 mingw32-make 编译后才可以对 world3 进行仿真。

4 实验总结与心得

（1）郝裕玮：坦白来说，这次大作业对我的挑战是极大的。但在最终完成之后，我确确实实地收获到了很多新的知识。并且看着我们小组实现的成果，我得到了极大的成就感，这是我这学期其他的课程无法与之相比的。

在点云建图上，由于节点和函数接口使用不熟练，导致进展较慢。最后靠着助教的点云建图部分的代码，我们的实验才得以完成。

在 RRT 路径规划方面，由于我之前的代码都是 Python 写的，这次为了和队友统一，我用队友马淙升同学之前实现的 C++代码重新进

行了修改和优化，效率大幅度提升，撒 100000 个随机点的路径规划图基本可以做到在 1-2 秒内得出结果，这也为后面的小车实时路径规划奠定了基础。

最后我想在这里特别感谢 3 位助教和成慧老师这一学期给我们的帮助与指点，让我在这门课中受益良多。

（2）马淙升：本次实验主要是使用雷达传感器对未知世界的小车做路径规划的问题，通过本次实验我们大致了解了这一过程和一些实现的思路，在雷达感知上，我们在设计时遇到了与预期不完全相符的状况，可能出现探测误差，这个问题也困扰了我们一段时间，最后我们通过在代码中实现了一些容错机制，如撞墙停止，重新规划，等等。这也给我们去解决一些实际生活中可能出现的问题停供了一些思考的经验。

在全局规划设计上，我们也曾有一些不同的设计思路，如应用在我们本次实验中的使用全图的规划但是每次小车行驶时只走探索过的区域。我们还有过一些其他的设想，如使用启发搜索的思想，每次在我们探索的区域中选择一个较优的点，规划从当前点到该点的路线并行驶，当然这样需要对局部搜索过程做记录防止死循环的发生。本次实验丰富的思考空间也让我们在进行本次实验时有更大的挑战，也有了更大的乐趣。

（3）杨荣杰：期末大作业增进了我的团队合作精神，巩固了我所学的知识。

在开始做之前，我们进行了第一次第一次小组讨论，确定了大致的分工和进度；然后各自学习相应部分的知识后，我们开了第二次小组会议，讨论了真正实现时我们可能会遇到的困难以及可能的解决方案；最后我们还召开了几次小组会议，主要是为最后作业的效果和存在的一些 bug 进行讨论与处理。通过和组员的一次次讨论，我更加认识到了一个团队保持交流的重要性，这一点对我启发很大。

在作业中我主要负责控制的部分，我们使用的是 GPS 控制导航小车运动。在实践过程中我应用了课堂上学到了基础的数学理论知识，比如向量夹角，叉乘的几何意义等等，结合起来控制小车的运动。在实践过程中我还考虑到并解决了实际应用中会出现的问题，比如小车到达目标点的判断，避免小车控制过程中程序崩溃的容错机制等，我学会了要从实际出发，基于理论而不止于理论地看待问题和解决问题。