深蓝学院
shenlanxueyuan.com

第三章作业思路提示

主讲人 陈铭楠

纲要

- 第一部分：MATLAB实现RRT
- 第二部分：C++实现RRT*
- 第三部分：C++实现Informed RRT*

MATLAB实现RRT

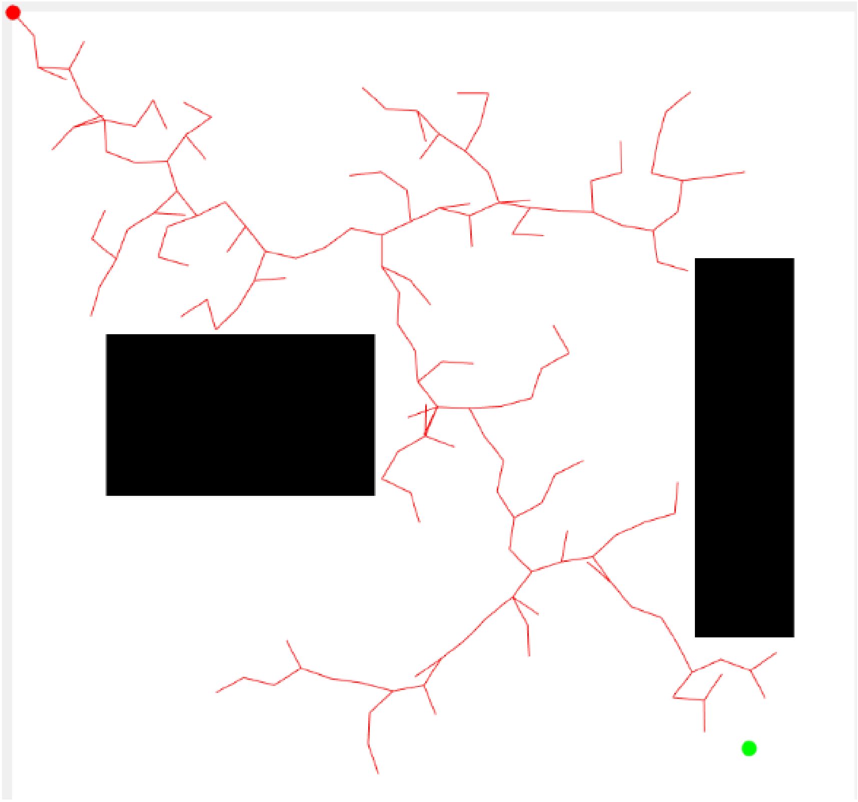
深蓝学院 shenlanxueyuan.com

● 这一部分具体流程可参考代码注解

```
for iter = 1:3000
    x_rand=[0 + xL * rand(1,1), 0 + yL * rand(1,1)];
    %Step 1: 在地图中随机采样一个点x_rand
    %提示：用（x_rand(1),x_rand(2)）表示环境中采样点的坐标

    %Step 2: 遍历树，从树中找到最近邻近点x_near
    %提示：x_near已经在树T里
    %put the distance into and array
    distArray = arrayfun(@(x) getdis(x_rand(1),x_rand(2),x.x,x.y),T.v);
    %The ~ represents an output that is discarded. It is essentially equivalent to
    %reference: https://de.mathworks.com/matlabcentral/answers/73735-what-does-it-mean-by-writing-idx-in-code
    [~,idx]=min(distArray);
    x_near=[T.v(idx).x,T.v(idx).y];

    %x_new=[];
    %Step 3: 扩展得到x_new节点
    %提示：注意使用扩展步长Delta
    %The direction of extension theta
    theta = atan2(x_rand(2) - x_near(2),x_rand(1) - x_near(1));
    x_new = [x_near(1)+Delta*cos(theta),x_near(2)+Delta*sin(theta)];

    %检查节点是否是collision-free
    if ~collisionChecking(x_near,x_new,Imp)
        continue;
    end
    count=count+1;

    if count>2000
        bFind = false;
    end

    %Step 4: 将x_new插入树T
    %提示：新节点x_new的父节点是x_near
    T.v(count).x = x_new(1);
    T.v(count).y = x_new(2);
    T.v(1).xPrev = x_near(1);     % 父节点改为x_near
    T.v(1).yPrev = x_near(2);
    T.v(1).dist=Delta;          % 从父节点到该节点的距离，这里可取欧氏距离
    T.v(1).indPrev = idx;     % 编号

    %Step 5:检查是否到达目标点附近
    %提示：注意使用目标点阈值Thr，若当前节点和终点的欧式距离小于Thr，则跳出当前for循环
    if getdis(x_new(1),x_new(2),x_G,y_G)<Thr
        plot([x_near(1),x_new(1)],[x_near(2),x_new(2)],'r');
        hold on;
        break;
    end
    %Step 6:将x_near和x_new之间的路径画出来
    %提示 1：使用plot绘制，因为要多次在同一张图上绘制线段，所以每次使用plot后需要接上hold on命令
    %提示 2：在判断终点条件弹出for循环前，记得把x_near和x_new之间的路径画出来
    plot([x_near(1),x_new(1)],[x_near(2),x_new(2)],'r');
    hold on;

    pause(0.05); %暂停一会，使得RRT扩展过程容易观察
end
```

MATLAB实现RRT

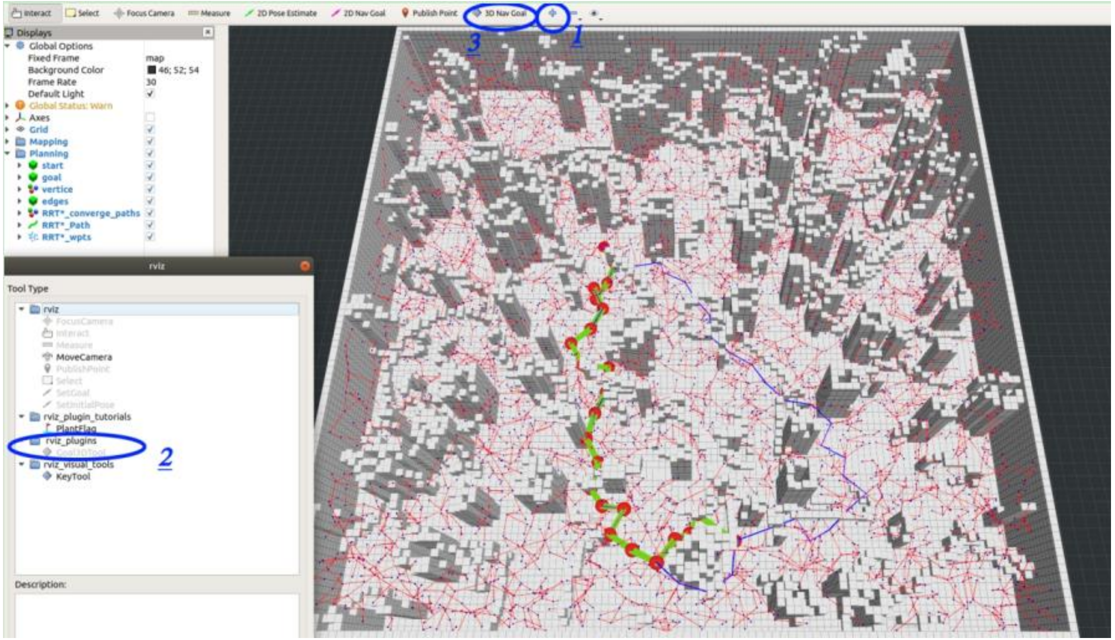
- 效果如下图



C++实现RRT*

深蓝学院
shenlanxueyuan.com

- 参考《hw3作业说明》创建工作空间与编译（这一部分比较简单，这里不做赘述）。按照文档要求操作，将能成功实现RRT效果。



能执行到这一步，说明环境配置没问题，接下了进行RRT*代码编写。

C++实现RRT*

- 首先我们梳理下RRT*的代码流程，其伪代码如下图：

Algorithm 2: RRT*Algorithm

Input: $\mathcal{M}$,x_{init},x_{goal}
Result: A path Γ from x_{init} to x_{goal}
$\mathcal{T}$.init();
for $i = 1$ *to* n **do**
  $x_{rand} \leftarrow Sample(\mathcal{M})$;
  $x_{near} \leftarrow Near(x_{rand}, \mathcal{T})$;
  $x_{new} \leftarrow Steer(x_{rand}, x_{near}, StepSize)$;
  if $CollisionFree(x_{new})$ **then**
    $X_{near} \leftarrow NearC(\mathcal{T}, x_{new})$;
    $x_{min} \leftarrow ChooseParent(X_{near}, x_{near}, x_{new})$;
    $\mathcal{T}.addNodEdge(x_{min}, x_{new})$;
    $\mathcal{T}.rewire()$;

作业需要补全rrt_star.h/rrt_star()函数中两处空白部分，分别为ChooseParent部分和rewire部分。

C++实现RRT*

- ChooseParent部分：注解提示：

// ! Hints:

// ! 1. Use map_ptr_->isSegmentValid(p1, p2) to check line edge validity;

// ! 2. Default parent is [nearest_node];

// ! 3. Store your chosen parent-node-pointer, the according cost-from-parent and cost-from-start

// ! in [min_node], [cost_from_p], and [min_dist_from_start], respectively;

// ! 4. [Optional] You can sort the potential parents first in increasing order by cost-from-start value;

// ! 5. [Optional] You can store the collison-checking results for later usage in the Rewire procedure

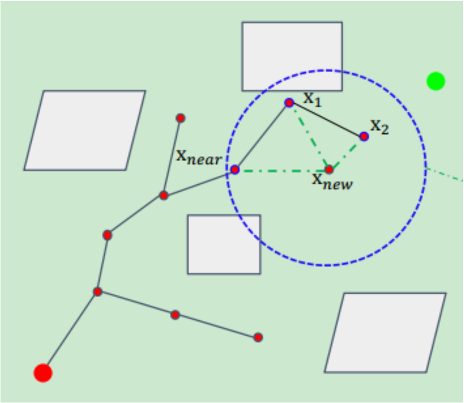
C++实现RRT*

● ChooseParent部分代码讲解：

```
double next_node_cost_start;
for (auto &curr_node : TreeNode * & : neighbour_nodes)
{

    if(map_ptr_->isSegmentValid( p0: x_new, p1: curr_node->x) && (curr_node->cost_from_start +
        calDist( p1: curr_node->x, p2: x_new))<min_dist_from_start){
        min_node = curr_node;
        cost_from_p = calDist( p1: curr_node->x, p2: x_new);
        next_node_cost_start = curr_node->cost_from_start + cost_from_p;
        min_dist_from_start = next_node_cost_start;
    }

}
```

首先 for (auto &curr_node : neighbour_nodes) 表示的是遍历x_new周围的所有的neighbour_node。该循环的目的在于：在所有curr_node中找到最小的dist_from_start，然后记录下min_dist_from_start， cost_from_p和min_node

以右图为例，neighbour_nodes有x_near, x1, x2



接下来for循环每执行一次，将检查两个条件：

1. x_new与当前计算的neighbour_node的连线(图中某条绿色连线)是否穿过障碍物
2. 判断该neighbour_node的cost是否会比初始的最小cost（以x_near为父节点，x_new为子节点的cost）小。

C++实现RRT*

- rewire部分代码讲解:

```
/* 3.rewire */
    // TODO Rewire according to potential cost-from-start values
    // ! Hints:
    // !  1. Use map_ptr_->isSegmentValid(p1, p2) to check line edge validity;
    // !  2. Use changeNodeParent(node, parent, cost_from_parent) to change a node's parent;
    // !  3. the variable [new_node] is the pointer of X_new;
    // !  4. [Optional] You can test whether the node is promising before checking edge collison.
    // ! Implement your own code between the dash lines [--------------] in the following loop
```

C++实现RRT*

深蓝学院
shenlanxueyuan.com

●rewire部分代码讲解：

```
for (auto &curr_node :TreeNode *& : neighbour_nodes)
{
  double best_cost_before_rewire = goal_node_->cost_from_start;
  // ! ----------------start write---------------------
  double dist_start_xnew_currentnode = new_node->cost_from_start + calDist( p1: curr_node->x,  p2: x_new);
  if(dist_start_xnew_currentnode < curr_node->cost_from_start && map_ptr_->isSegmentValid( p0: curr_node->x,  p1: x_new)){
      curr_node->parent = new_node;
      curr_node->cost_from_start = dist_start_xnew_currentnode;
  }
  // ! -----------------end write--------------------
```

主要过程如下：

1.计算curr_node与new_node的距离

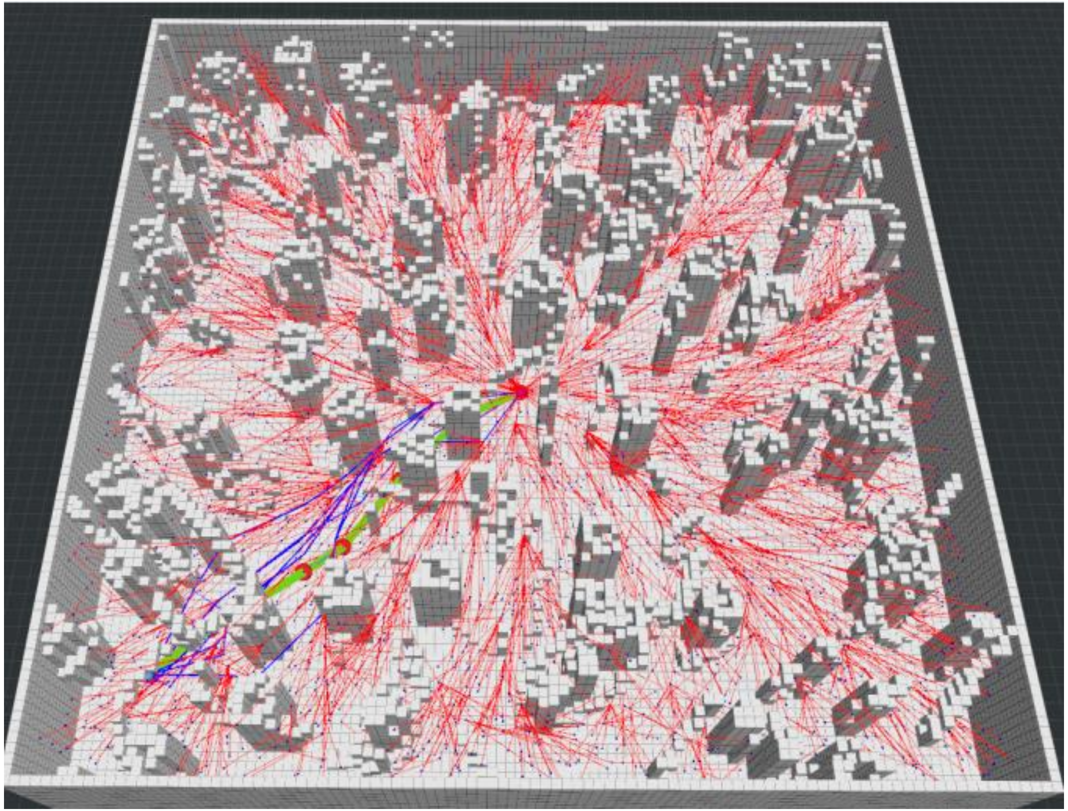
2.计算curr_node通过new_node连接与起点的距离dist_start_xnew_currentnode

3.对比curr_node按照原路径到起点的距离curr_node->cost_from_start和dist_start_xnew_currentnode，若dist_start_xnew_currentnode更小，则更换curr_node的父节点为new_node

C++实现RRT*

深蓝学院
shenlanxueyuan.com

- 效果如下：



C++实现Informed RRT*

- 伪代码如右图:

可以发现，Informed RRT*的实现思路和RRT*思路基本一样，不同的是采样函数的不同，Informed RRT*利用椭圆的特殊性质，在椭圆/球区域内采样能更快地获得最优路径。

Algorithm 1: Informed RRT*($\mathbf{x}_{start}, \mathbf{x}_{goal}$)

```
V ← {x_start};
E ← ∅;
X_soln ← ∅;
T = (V, E);
for iteration = 1 ... N do
    c_best ← min_{x_soln ∈ X_soln} {Cost(x_soln)};
    x_rand ← Sample(x_start, x_goal, c_best);
    x_nearest ← Nearest(T, x_rand);
    x_new ← Steer(x_nearest, x_rand);
    if CollisionFree(x_nearest, x_new) then
        V ← ∪ {x_new};
        X_near ← Near(T, x_new, r_RRT*);
        x_min ← x_nearest;
        c_min ← Cost(x_min) + c · Line(x_nearest, x_new);
        for ∀x_near ∈ X_near do
            c_new ← Cost(x_near) + c · Line(x_near, x_new);
            if c_new < c_min then
                if CollisionFree(x_near, x_new) then
                    x_min ← x_near;
                    c_min ← c_new;
        E ← E ∪ {(x_min, x_new)};
        for ∀x_near ∈ X_near do
            c_near ← Cost(x_near);
            c_new ← Cost(x_new) + c · Line(x_new, x_near);
            if c_new < c_near then
                if CollisionFree(x_new, x_near) then
                    x_parent ← Parent(x_near);
                    E ← E \ {(x_parent, x_near)};
                    E ← E ∪ {(x_new, x_near)};
        if InGoalRegion(x_new) then
            X_soln ← X_soln ∪ {x_new};
return T;
```

C++实现Informed RRT*

● 伪代码

在椭球区域采样过程如伪代码所示：

Algorithm 2: Sample $(\mathbf{x}_{\text{start}}, \mathbf{x}_{\text{goal}}, c_{\text{max}})$

if $c_{\max} < \infty$ **then**
- $c_{\min} \leftarrow \|\mathbf{x}_{\text{goal}} - \mathbf{x}_{\text{start}}\|_2$;
- $\mathbf{x}_{\text{centre}} \leftarrow (\mathbf{x}_{\text{start}} + \mathbf{x}_{\text{goal}})/2$;
- $\mathbf{C} \leftarrow \texttt{RotationToWorldFrame}(\mathbf{x}_{\text{start}}, \mathbf{x}_{\text{goal}})$;
- $r_1 \leftarrow c_{\max}/2$;
- $\{r_i\}_{i=2,\dots,n} \leftarrow \left(\sqrt{c_{\max}^2 - c_{\min}^2}\right)/2$;
- $\mathbf{L} \leftarrow \operatorname{diag}\{r_1, r_2, \dots, r_n\}$;
- $\mathbf{x}_{\text{ball}} \leftarrow \texttt{SampleUnitNBall}$;
- $\mathbf{x}_{\text{rand}} \leftarrow (\mathbf{C}\mathbf{L}\mathbf{x}_{\text{ball}} + \mathbf{x}_{\text{centre}}) \cap X$;

else
- $\mathbf{x}_{\text{rand}} \sim \mathcal{U}(X)$;

return $\mathbf{x}_{\text{rand}}$;

构造椭球的过程如下：

1. unit ball(单位球)：首先在单位圆内均匀采样（单位圆内均匀采样是有方法的，可以去搜一下）。
2. Scale：对单位球进行放缩，也就是长轴和短轴拉伸一下，呈椭球形状。
3. Rotation:做一个旋转。为求得旋转矩阵C，我们可以求出旋转后的椭球相对于原来椭球的欧拉角，再通过欧拉角转旋转矩阵的函数求得旋转矩阵C。通过求起点与终点的连线的向量求出yaw和pitch角（椭球长轴方向），由于椭球的长轴为对称轴，因此roll可取任意值。

4. Translation:做一个平移。平移量为起点终点连线的中点。

C++实现Informed RRT*

- 代码（个人作业代码仅供参考，欢迎交流~）

复制rrt_star.h文件并重命名为informed_rrt_start.h
旋转矩阵函数如下图：

```
//void Sample_infrrt_star()
static Eigen::Matrix3d euler2RotationMatrix(const double roll, const double pitch, const double yaw)
{
    Eigen::AngleAxisd rollAngle(angle: roll, axis: Eigen::Vector3d::UnitZ());
    Eigen::AngleAxisd yawAngle(angle: yaw, axis: Eigen::Vector3d::UnitY());
    Eigen::AngleAxisd pitchAngle(angle: pitch, axis: Eigen::Vector3d::UnitX());

    Eigen::Quaterniond q = rollAngle * yawAngle * pitchAngle;
    Eigen::Matrix3d R = q.matrix();
    cout << "Euler2RotationMatrix result is:" <<endl;
    cout << "R = " << endl << R << endl<<endl;
    return R;
}
```

C++实现Informed RRT*

- 代码（个人作业代码仅供参考，欢迎交流~）

由于我重写单位球采样的函数，因此我将此句注解：#include “sampler.h”,实现代码添加如下：

```
bool rrt_star(const Eigen::Vector3d &s, const Eigen::Vector3d &g)
{
  ros::Time rrt_start_time = ros::Time::now();
  bool goal_found = false;
  const double pai = 3.141596;

  std::random_device rd;
  std::mt19937_64 gen_(sd: rd());
  std::uniform_real_distribution<double> uniform_rand_;
  uniform_rand_ = std::uniform_real_distribution<double>(a: 0, b: 2.0);

  /* kd tree init */
  kdtree *kd_tree = kd_create(k: 3);
  //Add start and goal nodes to kd tree
  kd_insert3(tree: kd_tree, x: start_node_->x[0], y: start_node_->x[1], z: start_node_->x[2], data: start_node_);
  //informed RRT*:
  //Sample(x_start,x_goal,Cmax)
  //ellipse: (x_start,0) and (x_goal,0),
  //STEP1: c_min
  double c_min = calDist(p1: start_node_->x, p2: goal_node_->x);
  goal_node_->cost_from_start = 1.4 * c_min;
  double c_best = goal_node_->cost_from_start;
  //double c_best = 1000;
  cout << "goal_cost:"<<goal_node_->cost_from_start<<endl;
  //STEP2: the centre point between start_node and goal_node
  Eigen::Vector3d x_centre = (start_node_->x + goal_node_->x)/2.0;//
  //STEP4:rotation matric
```

C++实现Informed RRT*

- 代码（个人作业代码仅供参考，欢迎交流~）

```
double e_yaw = atan2(y: goal_node_->x[1]-start_node_->x[1], x: goal_node_->x[0]-start_node_->x[0]);// y/x
if(goal_node_->x[0]-start_node_->x[0]<0 && goal_node_->x[1]-start_node_->x[1]>0){
    e_yaw += pai;
}
if(goal_node_->x[0]-start_node_->x[0]<0 && goal_node_->x[1]-start_node_->x[1]<0){
    e_yaw -= pai;
}
double e_pitch = atan2(y: goal_node_->x[1]-start_node_->x[1], x: std::sqrt( x: std::pow( x: (goal_node_->x[0]-start_node_->x[0]), y: 2)
                + std::pow( x: (goal_node_->x[1]-start_node_->x[1]), y: 2)));
double e_roll = 0.01;
Eigen::Matrix3d Rotation_matrix = euler2RotationMatrix( roll: e_roll, pitch: e_pitch,  yaw: e_yaw);
```

C++实现Informed RRT*

- 代码

```
/* main loop */
int idx = 0;
for (idx = 0; (ros::Time::now() - rrt_start_time).toSec() < search_time_ && valid_tree_node_nums_ < max_tree_node_nums_; ++idx)
{
  /* biased random sampling */
  //STEP3
  double r1 = c_best / 2.0;//  Cmax/2
  //cout << "c_best"<<c_best<<endl;
  //cout << "idx:"<<idx<<endl;
  double r2 = std::sqrt(x: std::pow(x: goal_node_->cost_from_start, y: 2)-std::pow(x: c_min, y: 2))/2;
  double r3 = r2;

  //STEP 5: L_ellipse <- diag
  Eigen::Matrix<double,3,3> L_ellipse;
  L_ellipse << r1,0,0,
            0,r2,0,
            0,0,r3;

  //STEP 6: x_ball <- SampleUnitNBall    x_ball=x_rand
  Eigen::Vector3d x_rand;
  /.../
  //sampler_.samplingOnce(x_rand);
  x_rand[0] =  uniform_rand_(&: gen_);
  x_rand[1] =  uniform_rand_(&: gen_);
  x_rand[2] =  uniform_rand_(&: gen_);
  x_rand[0] = x_rand[0] -1.0;
  x_rand[1] = x_rand[1] -1.0;
  x_rand[2] = x_rand[2] -1.0;
```

C++实现Informed RRT*

深蓝学院
shenlanxueyuan.com

- 代码

```
x_rand[2] = x_rand[2] -1.0;
//cout <<"x_rand"<<x_rand<<endl;
if(std::sqrt(std::pow(x_rand[0],2) + std::pow(x_rand[1],2) + std::pow(x_rand[2],2)) > 1.0 ){
    continue;
}
  //cout <<"x_rand"<<x_rand<<endl;
//STEP 7:x_rand <- (Rotation_metrix * L * x_ball)   x_ball=x_rand  3*1 = 3*3 3*3 3*1
//x_rand = Rotation_matrix * L_ellipse * x_rand + x_centre;
x_rand[0] = x_rand[0]*r1 ;
x_rand[1] = x_rand[1]*r2 ;
x_rand[2] = x_rand[2]*r3 ;
x_rand = Rotation_matrix  * x_rand + x_centre;
//cout <<"x_rand"<<x_rand<<endl;
// samplingOnce(x_rand);
if (!map_ptr_->isStateValid(x_rand))
{
  continue;
}

struct kdres *p_nearest = kd_nearest3(kd_tree, x_rand[0], x_rand[1], x_rand[2]);
if (p_nearest == nullptr)
{
  ROS_ERROR("nearest query error");
  continue;
}
RRTNode3DPtr nearest_node = (RRTNode3DPtr)kd_res_item_data(p_nearest);
kd_res_free(p_nearest);
```

C++实现Informed RRT*

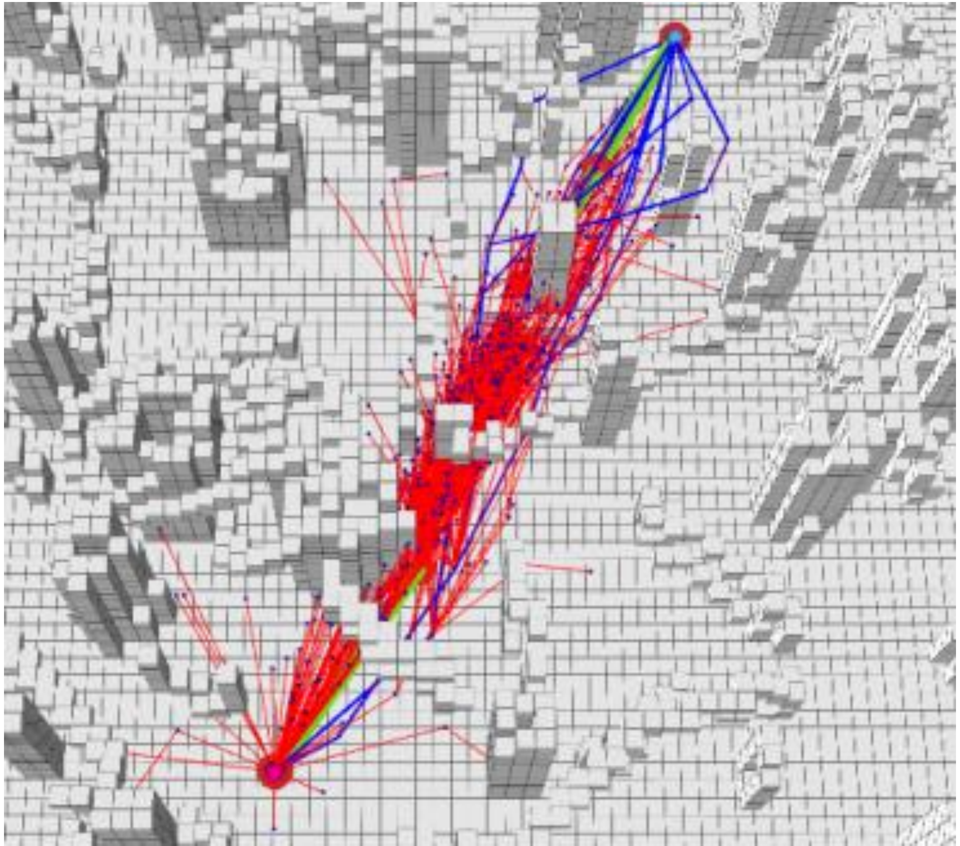
深蓝学院 shenlanxueyuan.com

- 代码（个人作业代码仅供参考，欢迎交流~）

```cpp
 Eigen::Vector3d x_new = steer(nearest_node_p: nearest_node->x,  rand_node_p: x_rand,  len: steer_length_);
// Eigen::Vector3d x_new_ball = L_ellipse.inverse() * Rotation_matrix.inverse() * (x_rand - x_centre);
Eigen::Vector3d x_new_ball = L_ellipse.inverse()  * (x_rand - x_centre);
x_new_ball =  Rotation_matrix.inverse() * x_new_ball;
 //cout <<"x_new_Ball:"<<x_new_ball<<endl;
 if(std::sqrt( x: std::pow( x: x_new_ball[0], y: 2) + std::pow( x: x_new_ball[1], y: 2) + std::pow( x: x_new_ball[2], y: 2)) > 1.0){
     continue;
 }
 if (!map_ptr_->isSegmentValid(nearest_node->x, x_new))
 {
   continue;
 }
```

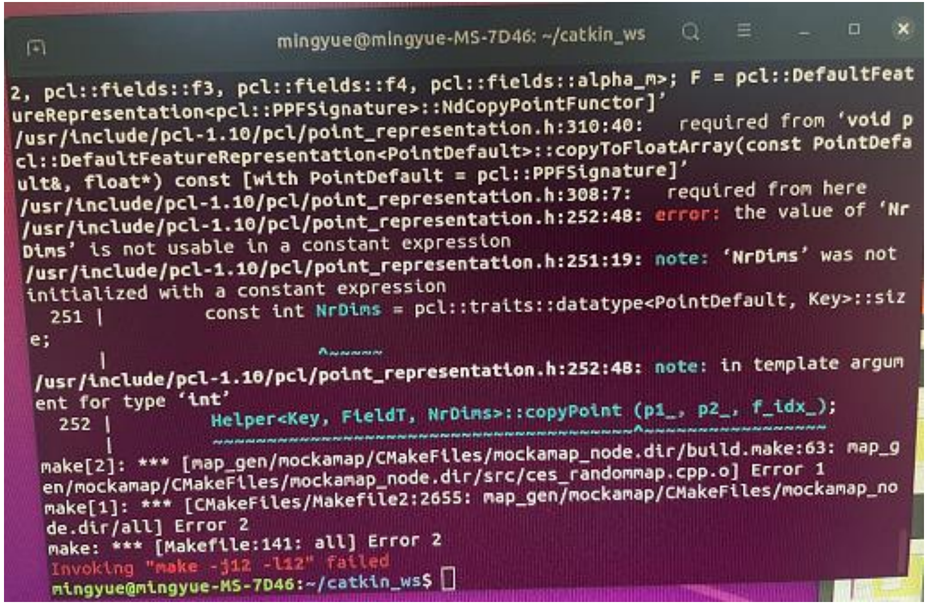
C++实现Informed RRT*

- 效果如图



Others

- 第三章作业群里反馈的问题



第一章与第二章作业能编译成功，第三章报如图错误（系统版本：ubuntu 20.04）

解决方法：

1.在cmakelist中将find_package(pcl 1.7 required)的1.7删除；

2.在cmakelist中加入set(CMAKE_CXX_STANDARD 14);

3.在cmakelist中将所有的能看到c++11全改成c++14

在线问答

Q&A

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening