为了向ROS添加一个新的全局路径规划器，新的路径规划器必须遵守nav\_core包中定义的nav\_core::BaseGlobalPlanner C ++接口。

<https://www.ncnynl.com/archives/201708/1887.html>

# 编写global\_planner.h

* 1. 库

#include <ros/ros.h>

#include <costmap\_2d/costmap\_2d\_ros.h>

#include <costmap\_2d/costmap\_2d.h>

#include <nav\_core/base\_global\_planner.h>

#include <geometry\_msgs/PoseStamped.h>

#include <angles/angles.h>

#include <base\_local\_planner/world\_model.h>

#include <base\_local\_planner/costmap\_model.h>

(1)黄色的前两个库,是路径规划所需要的.其中costmap\_2d::Costmap2D类 是输入的地图.当定义为插件时，路径规划器类将自动访问此地图。没有必要订阅costmap2d来获取ROS的代价地图。

(2)<nav\_core/base\_global\_planner.h> 用于导入接口nav\_core::BaseGlobalPlanner,这是插件必须要继承的父类。

* 1. 定义类的命名空间

namespace global\_planner {

class GlobalPlanner : public nav\_core::BaseGlobalPlanner {

* 1. 在nav\_core::BaseGlobalPlanner中定义的所有方法都必须在新类GlobalPlanner中总重写

public:

GlobalPlanner();

GlobalPlanner(std::string name, costmap\_2d::Costmap2DROS\* costmap\_ros);

/\*\* overriden classes from interface nav\_core::BaseGlobalPlanner \*\*/

void initialize(std::string name, costmap\_2d::Costmap2DROS\* costmap\_ros);

bool makePlan(const geometry\_msgs::PoseStamped& start,

const geometry\_msgs::PoseStamped& goal,

std::vector<geometry\_msgs::PoseStamped>& plan

);

构造函数GlobalPlanner(std::string name, costmap\_2d::Costmap2DROS\* costmap\_ros)用于初始化代价地图，即将用于规划的代价地图（costmap\_ros）以及规划器的名称（name）。

默认构造函数GlobalPlanner()即使用默认值作为初始化。

方法initialize(std::string name, costmap\_2d::Costmap2DROS\* costmap\_ros)是BaseGlobalPlanner的初始化函数，用于初始化costmap，参数为用于规划的代价地图（costmap\_ros）和规划器的名称(name)。

* 1. 初始化initialize

void CarrotPlanner::initialize(std::string name, costmap\_2d::Costmap2DROS\* costmap\_ros){

if(!initialized\_){

costmap\_ros\_ = costmap\_ros; //initialize the costmap\_ros\_ attribute to the parameter.

costmap\_ = costmap\_ros\_->getCostmap(); //get the costmap\_ from costmap\_ros\_ //这是costmap\_ros\_直接订阅话题 /scan?

// initialize other planner parameters

ros::NodeHandle private\_nh("~/" + name);

private\_nh.param("step\_size", step\_size\_, costmap\_->getResolution());

private\_nh.param("min\_dist\_from\_robot", min\_dist\_from\_robot\_, 0.10);

world\_model\_ = new base\_local\_planner::CostmapModel(\*costmap\_);

initialized\_ = true;

}

else

ROS\_WARN("This planner has already been initialized... doing nothing");

}

* 1. 函数makePlan

接着，bool makePlan（start，goal，plan）的方法必须要重写。最终规划将存储在方法的参数std::vector<geometry\_msgs::PoseStamped>& plan 中。该规划将通过插件自动发布为话题。

(1)其中需要调用Expander\* planner\_;

(2)makePlan()方法实现：在开始和目标参数用来获取初始位置和目标位置。在该说明性实现中，以起始位置（plan.push\_back(start))）作为开始的规划变量。然后，在for循环中，将在规划中静态插入20个新的虚拟位置，然后将目标位置作为最后一个位置插入规划。然后，此规划路径将发送到move\_base全局规划模块，该模块将通过ROS话题nav\_msgs/Path进行发布，然后将由本地规划模块接收。

(3)现在，您的全局规划类已经完成，您已准备好进行第二步，即为全局规划类创建插件，将其集成到[move\_base](http://wiki.ros.org/move_base" \t "/tmp/wps-zhjd/x/_blank)包的全局规划模块nav\_core::BaseGlobalPlanner中。

* 1. makePlan详解
     1. 类:

PotentialCalculator\* p\_calc\_; //不详  
Expander\* planner\_; // planner\_->calculatePotentials, 通过astar或者迪彻斯特算法,计算Potential  
Traceback\* path\_maker\_; //path\_maker\_->getPath ,从Potential得到path

* + 1. publishPotential(potential\_array\_);
    2. getPlanFromPotential函数

Traceback\* path\_maker\_;

path\_maker\_->getPath(potential\_array\_, start\_x, start\_y, goal\_x, goal\_y, path)

* + 1. void mapToWorld(double mx, double my, double& wx, double& wy)

将地图坐标转换为世界坐标

{  
 wx = costmap\_->getOriginX() + (mx+convert\_offset\_) \* costmap\_->getResolution();  
 wy = costmap\_->getOriginY() + (my+convert\_offset\_) \* costmap\_->getResolution();  
}

* + 1. void mapToWorld(double mx, double my, double& wx, double& wy) 同理
  1. 注册:PLUGINLIB\_EXPORT\_CLASS

这个要写在cpp文件中:

PLUGINLIB\_EXPORT\_CLASS(global\_planner::GlobalPlanner, nav\_core::BaseGlobalPlanner)

为此，必须包含库 #include <pluginlib/class\_list\_macros.h>

* 1. 编译:

要编译上面创建的全局规划库，必须将其添加到CMakeLists.txt中。添加代码：

add\_library(global\_planner\_lib src/path\_planner/global\_planner/global\_planner.cpp)

然后在终端中运行catkin\_make，在catkin工作空间目录中生成二进制文件。这将在lib目录中创建库文件 ~/catkin\_ws/devel/lib/libglobal\_planner\_lib。观察到“lib”附加到CMakeLists.txt中声明的库名global\_planner\_lib

* 1. :package.xml
     1. export部分:注册插件到ROS包系统
     2. depend:告诉编译器对于nav\_core包的依赖性.为了使上述导出export命令正常工作，提供程序包必须直接依赖于包含插件接口的程序包，这在全局规划程序的情况下是nav\_core。

<export>

<nav\_core plugin="${prefix}/global\_planner\_plugin.xml" />

</export>

<build\_depend>nav\_core</build\_depend>

<run\_depend>nav\_core</run\_depend>

* 1. 插件描述文件:global\_planner\_plugin.xml .

<library path="lib/libglobal\_planner\_lib">

<class name="global\_planner/GlobalPlanner" type="global\_planner::GlobalPlanner" base\_class\_type="nav\_core::BaseGlobalPlanner">

<description>This is a global planner plugin by iroboapp project.</description>

</class>

</library>

* 1. 在movebase中指定 全局规划器:

<node pkg="move\_base" type="move\_base" respawn="false" name="move\_base" output="screen">

<param name="base\_global\_planner" value="global\_planner/GlobalPlanner"/>

红色部分是global\_planner\_plugin.xml中规定的全局规划类的名称

* 1. 在movebase的main函数中:
     1. 载入基类nav\_core::BaseGlobalPlanner

bgp\_loader\_("nav\_core", "nav\_core::BaseGlobalPlanner"),

* + 1. 读取,自己写的base\_global\_planner的值,也就是”global\_planner/GlobalPlanner”

private\_nh.param("base\_global\_planner", global\_planner, std::string("navfn/NavfnROS"));

* + 1. 创建实例:

planner\_ = bgp\_loader\_.createInstance(global\_planner);

* + 1. 初始化:

planner\_->initialize(bgp\_loader\_.getName(config.base\_global\_planner), planner\_costmap\_ros\_);

* + 1. 路径规划:

planner\_->makePlan(start, p, global\_plan)

# 发布potential数量

* 1. 在planner-core.cpp的void GlobalPlanner::publishPotential(float\* potential)中添加

std\_msgs::Int16 num\_potential;  
num\_potential.data = 0;  
for (unsigned int i = 0; i < grid.data.size(); i++) {  
 if (potential\_array\_[i] >= POT\_HIGH) {  
 grid.data[i] = -1;  
 } else{  
 grid.data[i] = potential\_array\_[i] \* publish\_scale\_ / max;  
 num\_potential.data ++;  
 }  
}  
potential\_pub\_.publish(grid);  
num\_potential\_pub\_.publish(num\_potential);

* 1. 在planner\_core.h中增加

ros::Publisher num\_potential\_pub\_;

* 1. 在initialize函数中增加

num\_potential\_pub\_ = private\_nh.advertise<std\_msgs::Int16>("num\_potential", 1);

* 1. 在头文件中增加

#include "std\_msgs/Int16.h"

* 1. 修改.....,使得全局路径规划只发布一次

6.接收 :rostopic echo /move\_base/GlobalPlanner/num\_potential