# c++ APF环境配置和源码运行

2024年4月16日 9:55

# matpliot环境配置（为了能在c++程序里面使用python的matpliot画图显示功能）

1.github源码下载（下面那个测试指南里面有github源码链接）

把matplotcpp.h的文件复制到自己的项目里面

不需要下载python环境和numpy和matpliot库了，已经安装好了（之前）

但是要改一下Cmakelist

因为Cmakelist是针对python2.7的

我show了一下matpliot的库在python3，要找python3

2：Cmakelist的内容

```
# 声明要求的 cmake 最低版本
cmake_minimum_required(VERSION 2.8)
set(CMAKE_CXX_STANDARD 11)

project(TEST)
# 查找Python3库
find_package(Python3 COMPONENTS Interpreter Development)

# 设置Python头文件路径（通常不需要手动设置，find_package会帮你做这件事）
# 如果find_package没有正确找到，你可以手动设置
 set(PYTHON_INCLUDE_DIRS "/usr/include/python3.8")

# 添加头文件到工程
include_directories(${PYTHON_INCLUDE_DIRS})

# 添加一个可执行程序
add_executable(test test.cpp)

# 链接Python库到工程
target_link_libraries(test PRIVATE ${Python3_LIBRARIES})
```

3：然后测试是否成功

https://blog.csdn.net/weixin_44444810/article/details/125301609?ops_request_misc=&request_id=&biz_id=102&utm_term=matplotlib.h&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-0-125301609.142^v100^pc_search_result_base2&spm=1018.2226.3001.4187

# 4：C++使用Eigen/Densn库

dpkg -L libeigen3-dev 定位库的位置

```
/usr
/usr/include
/usr/include/eigen3
/usr/include/eigen3/Eigen
/usr/include/eigen3/Eigen/Cholesky
/usr/include/eigen3/Eigen/CholmodSupport
/usr/include/eigen3/Eigen/Core
/usr/include/eigen3/Eigen/Dense
/usr/include/eigen3/Eigen/Eigen
/usr/include/eigen3/Eigen/Eigenvalues
/usr/include/eigen3/Eigen/Geometry
/usr/include/eigen3/Eigen/Householder
/usr/include/eigen3/Eigen/IterativeLinearSolvers
```

路径重命名：

sudo ln -s /usr/include/eigen3/Eigen /usr/include/Eigen
这样就可以在C++程序里面，直接#include <Eigen/Dense>（一个向量库)

5：APF（人工势能法源码）
https://github.com/CHH3213/chhRobotics_CPP/blob/master/PathPlanning/Artifical_Potential_Field/APF.cpp
建议直接复制粘贴上面的源码，下面的code只提供注释（有问题，但是没找到）
APF.cpp

```
#include "APF.h"

APF::APF(double etaAtt, double etaRepOb, double etaRepEdge, double dmax, double n) : Eta_att(etaAtt),
                                                                                     Eta_rep_ob(etaRepOb),
                                                                                     Eta_rep_edge(etaRepEdge), d_max(dmax),
                                                                                     n(n) {}

void APF::setTargetPos(const Vector2d &targetPos) {
    target_pos = targetPos;
}

void APF::setObstaclePos(const vector<Vector2d> &obstaclePos) {
    obstacle_pos = obstaclePos;
}

void APF::setEtaAtt(double etaAtt) {
    Eta_att = etaAtt;
}

void APF::setEtaRepOb(double etaRepOb) {
    Eta_rep_ob = etaRepOb;
}

void APF::setEtaRepEdge(double etaRepEdge) {
    Eta_rep_edge = etaRepEdge;
}



void APF::setN(double n) {
    APF::n = n;
}

void APF::setD(double d) {
    APF::d = d;
}

void APF::setW(double w) {
    APF::w = w;
}

void APF::setDMax(double dMax) {
    d_max = dMax;
}

void APF::setLenStep(double lenStep) {
    len_step = lenStep;
}
//计算引力斥力 rotbot_state(x,y,v)
Vector2d APF::computeForce(VectorXd robot_state){
 //引力计算
 Vector2d delta_att=target_pos-robot_state.head(2);//目标与当前机器人坐标的欧式距离//向量由robot指向目标
 Vector2d F_att=Eta_att*delta_att;//引力势场
 double dist_att=delta_att.norm();//求两点距离值的平方
 Vector2d unite_att_vec=delta_att/dist_att;//力的单位向量，表征引力的方向
 //合力
 Vector2d F=F_att;
 //障碍物斥力势场
```

```
//在斥力势场函数增加目标调节因子（车辆到目标距离），以使车辆到达目标后斥力为0
for(Vector2d obs:obstacle_pos){//遍历障碍物
  Vector2d delta=robot_state.head(2)-obs;//障碍物与当前机器人坐标的欧式距离//向量由障碍物指向robot
  double dist=delta.norm();
  Vector2d unite_rep_vec=delta/dist;//斥力的单位向量
  Vector2d F_rep_ob;
  if(dist>=d_max){
        F_rep_ob=Vector2d(0,0);//超过障碍物的斥力作用场范围，为0
  }
  else{
     //障碍物的斥力1，方向由障碍物指向车辆
     //斥力1
     double F_rep1_norm=Eta_rep_ob*(1/dist-1/d_max)*pow(dist_att,n)/pow(dist,2);
     Vector2d F_rep_ob1=F_rep1_norm*unite_rep_vec;//系数承上障碍物指向车俩的单位向量
     //斥力2:增加一个由障碍物产生，指向目标点的斥力，避免斥力和引力在同点抵消，未到目标点而合力为0
     double F_rep2_norm=n/2*Eta_rep_ob*pow(1/dist-1/d_max,2)*pow(dist_att,n-1);
     Vector2d F_rep_ob2=F_rep2_norm*unite_att_vec;
     //合斥力
     F_rep_ob=F_rep_ob1+F_rep_ob2;
      }
      F+=F_rep_ob;//合力
  }
  //道路斥力
  Vector2d F_rep_edge;
  double v=robot_state(2);//车辆速度
    if(robot_state(1)>-d+w/2&&robot_state(1)<=-d/2){
    F_rep_edge = Vector2d (0,Eta_rep_edge * v * exp(-d / 2 - robot_state(1)));
  }else if(robot_state(1)>-d/2&&robot_state(1)<=-w/2){
    F_rep_edge = Vector2d (0,1/3*Eta_rep_edge*pow(robot_state(1),2));
  }else if(robot_state(1)>w/2&&robot_state(1)<=d/2){
    F_rep_edge = Vector2d (0,-1/3*Eta_rep_edge*pow(robot_state(1),2));
  }else if(robot_state(1)>d/2&&robot_state(1)<=d-w/2){
    F_rep_edge = Vector2d (0,Eta_rep_edge * v * exp( robot_state(1)-d / 2 ));
  }
  F+=F_rep_edge;//合力加上道路斥力

  Vector2d unit_F=F/F.norm();
  return unit_F;//返回受到的合力的单位向量
}


/**
 * 人工势场法控制器
 * @param robot_state
 * @return 车辆下一步位置
 */
VectorXd APF::runAPF(VectorXd robot_state){
 Vector2d unit_F=computeForce(robot_state);//算出单位合力向量
 Vector2d next_pos=robot_state.head(2)+len_step*unit_F;//步长*单位向量，确定一步增量，再加上原来的位置，
计算新的位置
 //更新状态
 robot_state<<next_pos(0),next_pos(1),next_pos(2);
 return robot_state;
}


APF.h
#ifndef CHHROBOTICS_CPP_APF_H
#define CHHROBOTICS_CPP_APF_H
#include<Eigen/Dense>
#include<iostream>
#include<vector>
```

```
#include<cmath>
#include<algorithm>
using namespace std;
using namespace Eigen;
class APF{
private:
  double Eta_att,Eta_rep_ob,Eta_rep_edge,d_max,n;
  //引力的增益系数，斥力的增力系数，道路边界的增益系数，障碍影响的最大距离，n系数
  Vector2d target_pos;
  vector<Vector2d>obstacle_pos;//障碍物位置
  double d,w;//道路标准步长，汽车宽度
  double len_step;//步长
public:
 APF(double etaAtt,double etaRepOb,double etaRepEdge,double dmax,double n);
    void setTargetPos(const Vector2d &targetPos);

    void setObstaclePos(const vector<Vector2d> &obstaclePos);

    void setEtaAtt(double etaAtt);

    void setEtaRepOb(double etaRepOb);

    void setEtaRepEdge(double etaRepEdge);

    void setDMax(double dMax);

    void setN(double n);

    void setD(double d);

    void setW(double w);

    void setLenStep(double lenStep);

    Vector2d computeForce(VectorXd robot_state);

    VectorXd runAPF(VectorXd robot_state);//VectorXd,可以处理任意唯度的双精度浮点数的向量
};

#endif //CHHROBOTICS_CPP_APF_H


Main.cpp
#include "matplotlibcpp.h"
#include"APF.h"
namespace plt=matplotlibcpp;

int main(){
 //初始化参数
 double d =3.5;//道路宽度
 double w=1.8;//汽车宽度
 double L=4.7;//车长
 double Eta_att=2;//引力的增益系数
 double Eta_rep_ob=1;//斥力的增益系数
 double Eta_rep_edge=10;//道路边界斥力的增益系数
 double d_max=5;//障碍影响的最大距离
 double len_step=0.5;//步长
 double n=1;//n系数
 int Num_iter=300;//最大循环迭代次数;

 Vector2d target(99,0);//目标和障碍物
  vector<Vector2d>obstacle_pos={Vector2d (15, 7 / 4),Vector2d (30, - 3 / 2),Vector2d (45, 3 / 2),Vector2d (60, - 3 /
4),Vector2d (80, 3/2)};
```

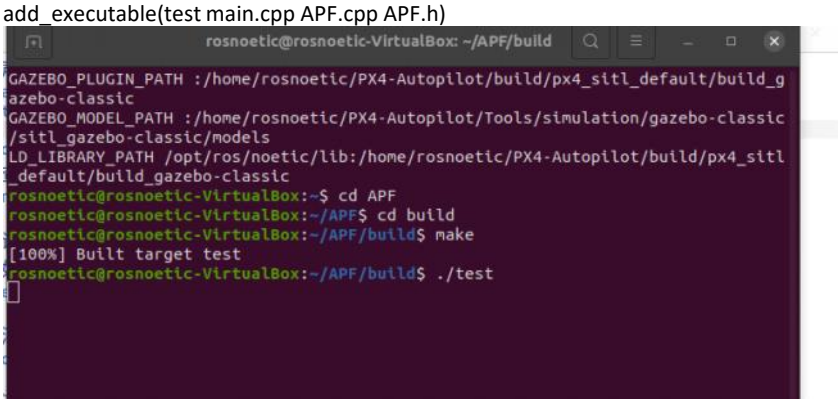
```
//初始化APF
 APF apf(Eta_att,Eta_rep_ob,Eta_rep_edge,d_max,n);
    apf.setD(d);
   apf.setW(w);
   apf.setLenStep(len_step);
   apf.setTargetPos(target);
   apf.setObstaclePos(obstacle_pos);
//初始化robot和robot的初始化位置
VectorXd robot_state(3),init_pos(2);
robot_state<<0,0,2;
init_pos<<0,0;
vector<double>x_,y_;//存储车辆运行轨迹便于i画图
//画图
double len_line=100;
vector<double>greyZone_x{-5,-5,len_line,len_line};
vector<double>greyZone_y{-d-0.5,d+0.5,d+0.5,-d-0.5};

 for(int i=0;i<Num_iter;i++){
  plt::clf();//用于清除当前图形。这意味着图形中当前显示的所有坐标轴、绘图和其他对象都将被移除
  robot_state=apf.runAPF(robot_state);//计算新的状态i
  x_.push_back(robot_state(0));
  y_.push_back(robot_state(1));

 //划分界限
       map<string, string> keywords;
      keywords["color"] = "grey";
  plt::fill(greyZone_x,greyZone_y,keywords);//用灰色填充（x,y）之间的 区域
      plt::plot({-5,len_line},{0,0},"w--");//前面是两个点对应的x坐标，后面是两个点对应的y坐标，画一个在y=0（x轴
上的）道路中间的线
      plt::plot({-5,len_line},{d,d},"w--");//道路两边
      plt::plot({-5,len_line},{-d,-d},"w--");

      for(Vector2d obs:obstacle_pos){
         plt::plot(vector<double>{obs(0)},vector<double>{obs(1)},"ro");//障碍物位置
      }
      plt::plot(vector<double>{target(0)},vector<double>{target(1)}, "gv");//目标位置
      plt::plot(vector<double>{init_pos(0)},vector<double>{init_pos(1)}, "bs");//起点位置

      //画轨迹
      plt::plot(x_, y_,"r");

      plt::grid(true);
      plt::ylim(-10,10);
      plt::pause(0.01);
      if ((robot_state.head(2) - target).norm() < 1) { break; }

 }

 //保存图片
    //// save figure
   const char* filename = "./apf_demo.png";
   cout << "Saving result to " << filename << std::endl;
   plt::save(filename);
   plt::show();
   return 0;

}
```

6：执行源码

可以直接把源码放进第3步里面的测试程序的文件结构里，直接拿源码替换掉test.cpp

然后修改一下Cmakelists

# 添加一个可执行程序
add_executable(test main.cpp APF.cpp APF.h)




文件结构：按照第三步里面的链接创建就好了