视觉巡线小车

18329015 郝裕玮

一、实验目标

为小车添加相机，GPS，使得小车沿着规定的轨道（黑线）按照一定的速度进行移动，并计算小车的绕行一圈的时间，平均速度和巡线误差。

二、实验内容与步骤

（1）在Robot的children中添加camera节点，并在camera的children中添加transform节点，再在transform的children中添加Shape节点，同时需要设置camera的句柄为camera，最后设置其位置，大小等参数即可；

（2）在Robot的children中添加GPS节点，并在camera的children中添加Solid节点，再在Solid的children中添加Shape节点，同时需要设置GPS的句柄为gps，最后设置其位置，大小等参数即可；

（3）在RectangleArena->floorAppearance->baseColorMap->url选择Circle图，替换之前的轨迹图。

控制器代码如下所示（**代码注释中已包含所有的思路分析和算法实现**）：

#include <webots/Robot.hpp>

#include <webots/Motor.hpp>

#include <webots/Keyboard.hpp>

#include <webots/Camera.hpp>

#include <webots/GPS.hpp>

#include <iostream>

#include <algorithm>

#include <iostream>

#include <limits>

#include <string>

#include<string.h>

using namespace std;

using namespace webots;

int main() {

    Motor \*motors[4];//电机和键盘都要用webots给的类型

    webots::Keyboard keyboard;

    char wheels\_names[4][8]={"motor1","motor2","motor3","motor4"};//对应RotationMotor里的句柄

    Robot \*robot=new Robot();//使用webots的机器人主体

    Camera \*camera = robot->getCamera("camera");//获取相机，句柄名为camera

    camera->enable(1);//设置相机每1ms更新1次

    keyboard.enable(1);//运行键盘输入设置频率是1ms读取一次

    GPS\* gps = robot->getGPS("gps");//获取GPS，句柄名为gps

    gps->enable(1);//设置GPS每1ms更新1次

    double speed[4];//此数组会在后面赋值给电机以速度

    double velocity=10;//初速度

    int i;

    double cur\_speed=0;//瞬时速度

    double avg\_speed=0;//平均速度

    double speed\_sum=0;

    //初始化

    for(i=0;i<=3;i++){

        motors[i]=robot->getMotor(wheels\_names[i]);//按照你在仿真器里面设置的名字获取句柄

        motors[i]->setPosition(std::numeric\_limits<double>::infinity());

        motors[i]->setVelocity(0.0);//设置电机一开始处于停止状态

        speed[i]=3;//给予小车一个初速度

    }

    double speed\_forward[4]={velocity,velocity,velocity,velocity};//前进方向

    double speed\_leftCircle[4]={velocity,-velocity,-velocity,velocity};//左自旋(即左转弯)

    double speed\_rightCircle[4]={-velocity,velocity,velocity,-velocity};//右自旋(即右转弯)

    int timeStep=(int)robot->getBasicTimeStep();//获取你在webots设置一帧的时间

    int cnt=0;//统计帧数

    double error\_sum=0;//记录瞬时的半径累加和

    double result=0;//记录误差

    double x0=0,y0=1.22;//手动大致测量圆心坐标

    double cur\_x,cur\_y;//记录GPS的实时坐标

    double pi;//当前的瞬时半径

    double cur\_error;//当前的瞬时误差

    double r=1.185;//手动测量的真实半径

    while(robot->step(timeStep)!=-1){//仿真运行一帧

        cnt++;//统计帧数

        const unsigned char \*a=camera->getImage();//读取相机抓取的最后一张图像。图像被编码为三个字节的序列，分别代表像素的红、绿、蓝

        int length=camera->getWidth();//图像长度

        int width=camera->getHeight();//图像宽度

        cout<<"图像尺寸为："<<length<<'\*'<<width<<endl;//输出图像尺寸

        int b1,b2,b3,b4;

        b1=length\*3\*width/2;//图像中间一行的最左边的像素点

        b2=length\*3\*width/2+(width/2+3)\*3;//图像中间一行的中间靠左的某像素点

        b3=length\*3\*width/2+(width/2+5)\*3;//图像中间一行的中间靠右的某像素点

        b4=length\*3\*width/2+(length-1)\*3;//图像中间一行的最右边的某像素点

        //其中b1，b2代表图像中间的黑色轨迹的两侧(大致估计)

        cur\_x=gps->getValues()[0];//获取实时x坐标

        cur\_y=gps->getValues()[1];//获取实时y坐标

        pi=sqrt((cur\_x-x0)\*(cur\_x-x0)+(cur\_y-y0)\*(cur\_y-y0));//计算当前的瞬时半径

        cur\_error=abs(pi-r);//实时误差=|pi-r|

        error\_sum+=cur\_error;//误差累加

        result=error\_sum/cnt;//计算当前的平均误差

        cur\_speed=gps->getSpeed();

        speed\_sum+=cur\_speed;

        avg\_speed=speed\_sum/cnt;

        cout<<"当前经历了"<<cnt<<"帧"<<endl;

        cout<<"当前实时半径为"<<pi<<endl;

        cout<<"当前瞬时速度为"<<cur\_speed<<","<<"当前平均速度为"<<avg\_speed<<endl;

        cout<<"当前实时误差为"<<cur\_error<<","<<"当前平均误差为"<<result<<endl;

        //以rgb的r为标准，当颜色为黑时，r的值必定小于80(10-30左右)

        if(a[b1]<80&&a[b2]>80&&a[b3]>80&&a[b4]>80){//若只有最左边像素点为黑色，则小车需要右转弯使得轨道黑线向中间靠拢

            for(i=0;i<=3;i++){

                speed[i]=speed\_rightCircle[i];//速度方向为右转

            }

        }

        else if(a[b1]>80&&a[b2]<80&&a[b3]<80&&a[b4]>80){//若中间两个像素判断点为黑色，则小车可选择继续直行

            for (i=0;i<=3;i++){

                speed[i]=speed\_forward[i];

            }

        }

        else if(a[b1]>80&&a[b2]>80&&a[b3]>80&&a[b4]<80){//若只有最右边像素点为黑色，则小车需要左转弯使得轨道黑线向中间靠拢

            for (i=0;i<=3; i++){

                speed[i]=speed\_leftCircle[i];

            }

        }

        else if(a[b1]>80&&a[b2]>80&&a[b3]>80&&a[b4]>80){//若四个判断像素点全为白色，则小车可选择继续直行

            for(i=0;i<=3;i++){

                speed[i]=speed\_forward[i];

            }

        }

        //将速度赋值给电机

        for(i=0;i<=3;i++){

            motors[i]->setVelocity(speed[i]);

        }

    }

    return 0;

}

三、实验结果与分析

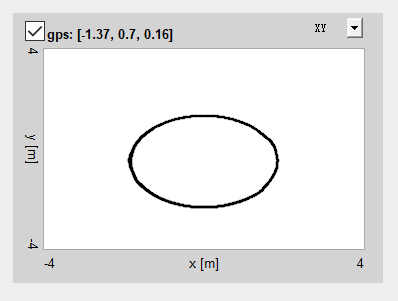
Robot的children节点结构如下图所示：



最终运行结果如下表所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 第1次 | 第2次 | 第3次 | 平均值 |
| 运动时间 | 19.456 | 19.552 | 19.424 | 19.477 |
| 运行帧数 | 607 | 610 | 606 | 607.677 |
| 平均速度 | 0.670808 | 0.670202 | 0.670917 | 0.670642 |
| 巡线误差 | 1.3173 | 1.31133 | 1.31936 | 1.31597 |

运行轨迹如下图所示：



四、实验中的问题和解决方法

（1）一开始没有在网上找到相关的参考资料，导致安装相机后无从下手，不知道该如何将相机拍摄到的图像转换为判断小车前进方向的依据。后来通过查询Webots controller API（C++）明白了

Camera库中的可用函数，最终选择采用getImage函数获取图像每个像素点的RGB值，并根据粗略的特定位置的像素点是否为黑色来判断小车的前进方向；

（2）本想再添加2个Robot并添加GPS，使得可在圆上任意一点和圆心处再放置2个GPS，但是项目报错，说只能添加1个机器人实例，只好作罢。后来发现群里助教说可大致估计圆的半径并作为已知量使用，所以我自行移动了Robot的GPS将其定位在圆心，记录其坐标，并只改变x值使得GPS再停留在圆的边缘上，计算x的差值即可得到圆的半径的估计值，约为1.185m。

（3）一开始我发现GPS的速度只有0.62m/s，觉得不符合常理（毕竟电机速度为10rad/s）。以为是我调用函数有问题，Debug了很久，最后发现该结果实际上是正确的，证明过程如下：

理论最优平均速度== 0.382m/s

实际估算平均速度= = 0.806m/s

又因为实际轨迹不可能是真正的圆，只是近似于圆，所以最终实际速度必定在[0.382,0.806]这个区间内，所以结果无误。