

Player Stage 技术文档



上海交通大学软件学院
嵌入式实验室群体智能组(S9)

之清障小组

修订历史记录

日期	版本	说明	作者
2009-12-15	<1.0>	初始版本	刘道勇 (tomany1985@163.com)
2009-12-20	<1.1>	加入如何安装 Ubuntu9.04 桌面版操作系统	周鑫 (zhouxin@sjtu.edu.cn)
2010-3-29	<1.2>	增加如何在 Stage 上实现算法部分	王奕恒 (yihengw@gmail.com)

目录

一、开发环境.....	2
二、安装	2
1. 用 Wubi 从硬盘安装 Ubuntu 操作系统.....	2
1.1 什么是 Wubi.....	2
1.2 用 Wubi 来安装 Ubuntu 9.04 的步骤	2
2. 安装 Player Stage	5
2.1 安装准备工作.....	5
2.1.1 依赖包安装	5
2.1.2 环境变量设置	5
2.2 player 安装	6
2.3 stage 安装	6
三、测试 player stage	7
3.1 stage 测试.....	7
3.2 player 测试.....	9
四、配置自己的应用场景.....	9
4.1.world 文件配置.....	9
4.1.1 头文件声明	9
4.1.2 地图配置机器人配置以及传感器配置.....	10
4.1.3 机器人配置以及传感器配置	10
4.2. .cfg 文件配置.....	12
4.2.1 simulation 驱动.....	12
4.2.2 机器人驱动.....	12
五、如何编写自己的控制算法来控制机器人运动.....	13
六、一个完整的例子.....	13
6.1 demo1.world 文件(场景地图、机器人、传感器等配置文件)	13
6.2 demo1.cfg 文件(机器人、仿真器、传感器等驱动文件).....	16
6.3 demo1.cc 文件(控制算法算法文件).....	17
6.4 编译及运行.....	21
七、如何在 Stage 上进行算法仿真	13
7.1 为什么要在 stage 上进行算法仿真.....	13
7.2 Stage 的安装	16
7.3 Stage 上如何运行自己的算法	17

7.4 Stage 上怎样写程序.....	21
八、参考资料和实用网站.....	25

一、开发环境

硬件：hp Pentinu Dual-Core CPU E5200@2.5GH/1G/160G

操作系统：ubuntu 9.04 桌面版

工具：Player/Stage

二、安装

1. 用 Wubi 从硬盘安装 Ubuntu 9.04 桌面操作系统

1.1 什么是 Wubi

Wubi 是 Windows based Ubuntu Installer 缩写，是一个专门针对 Windows 用户的 Ubuntu 安装工具。

Wubi 有哪些特点

Wubi 让你如同 Windows 操作系统里的其他软件一样安装卸载 Ubuntu，你需要做的只是点击几下鼠标而已。在整个安装过程中用户不需要关心分区的设置，不需要修改启动文件。大大地降低了安装 Ubuntu Linux 的难度。

1.2 用 Wubi 来安装 Ubuntu 9.04 的步骤

第一步：（不会分区的菜鸟们这一步就可以跳过）这样做只是为了方便管理文件，不另外分区也可以完成安装的！首先 我们可以用分区工具分出一个 10G（大小根据你自己的实际情况而定）的盘出来，可以用的是[Norton PartitionMagic 8.0](#)，分好后的盘如图。最好使用 fat32 格式分区安装，成功率要高很多。



第二步：到官网下载 Wubi 以及 Ubuntu 9.04 的镜像文件，建议下载 32 位的 Ubuntu 9.04 镜像文件（32 位的系统可用软件更多）。下载完后将 wubi 和 Ubuntu 9.04 的镜像文件放在同一个分区的根目录下，这样我们就要开始安装了。相信菜菜们很激动了吧！

第三步：找到我们刚才下的 wubi，双击运行 Wubi.exe。



这个就是 Wubi



在此选择你要的将 Ubuntu 安装在磁盘的哪一个分区上，给 Ubuntu 分配空间的大小，设置语言环境以及设定你的用户名和登陆密码，这些你可以根据自己的实际情况来设置，然后点安装。



wubi 程序自动开始安装，wubi 会自动找到你下的[镜像文件](#)，务必确定你的计算机连接在网络上，因为还需要在网上下载少量安装所需的文件，所以要保持网络连接。但是下载的很少，大多数文件在光盘映像中。中间的过程都是自动，笔者就不多说了！

第四步：稍后，wubi 会提示你需要重启计算机



重启计算机后在启动菜单中选择 ubuntu 。进入 Ubuntu 后才真正开始安装，都是自动的而大家只需耐心的等待。大约二十分钟后就安装好了，机器会自动重启。

第五步.再次开机在启动菜单中选择 ubuntu 进入，在登陆界面出现后输入你预设的用户名和密码后，如果显卡驱动无问题就可以了，到这里我相信可爱的 ubuntu 界面已经出现在你的眼前了吧！祝贺一下！

2. 安装 Player Stage

• 2.1: 安装准备工作

Player Stage 的安装需要很多依赖包以及需要进行环境变量的设置，一开始就应该先设置好，以免在安装后遇到不必要的麻烦。

• 2.1.1: 依赖包安装

Player 依赖包安装: `sudo apt-get install build-essential libgtk2.0-dev libltdl3-dev cmake libgnomecanvas2-dev libgdk-pixbuf-dev swig python-dev`

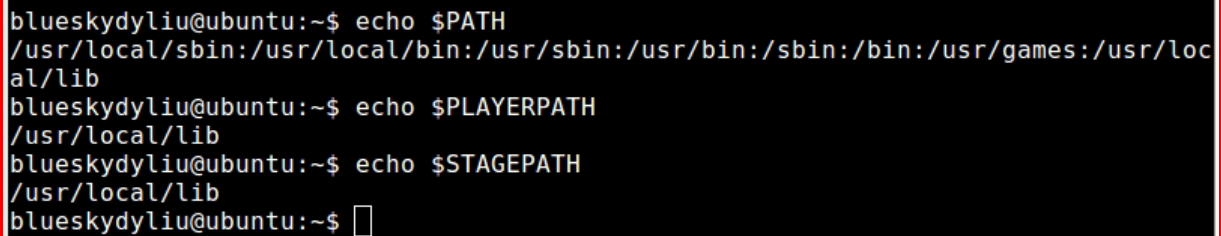
Stage 依赖包安装: `sudo apt-get install libfltk1.1-dev mesa-common-dev libglu1-mesa-dev`

• 2.1.2: 环境变量设置

打开终端输入下面命令:

```
export PATH="/usr/local/lib"
export LD_LIBRARY_PATH=/usr/local/lib
export STAGEPATH=/usr/local/lib
export PLAYERPATH=/usr/local/lib
```

或者使用 sudo 用户打开 .bashrc 文件，直接进行修改。修改完成以后关闭所有终端，然后再打开，输入 echo 命令进行测试环境变量是否设置成功，如下图所示:



```
blueskydyliu@ubuntu:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/lib
blueskydyliu@ubuntu:~$ echo $PLAYERPATH
/usr/local/lib
blueskydyliu@ubuntu:~$ echo $STAGEPATH
/usr/local/lib
blueskydyliu@ubuntu:~$
```

• 2.2 player 安装

版本: player-3.0.0

安装包: player-3.0.0-rc3.tar.gz

下载地址: <ftp://sword:king@202.120.40.124:2121/project/+WSN/Software/player-3.0.0.tar.gz>

也可以到<http://playerstage.sourceforge.net/> 去下载最新的版本进行安装。

安装过程:

注意: 在 Ubuntu 系统中建立自己的文件夹, 将安装包放入其中, 不建议将安装包放入系统文件夹中进行安装。此过程同样适合于 Stage 的安装。

1. 解压 player 包:

```
tar zxvf player-3.0.0-rc3.tar.gz
```

2. 安装以及编译:

```
cd player-3.0.0
```

```
mkdir build
```

```
cd build
```

```
cmake ../
```

```
make
```

```
sudo make install
```

至此 player 安装成功。可能会出现一些依赖包找不到的情况, 主要运行 `sudo apt-get install` 命令将依赖包安装好再重复 2 过程即可。

• 2.3 stage 安装

版本: Stage 3.2.2

安装包: Stage-3.2.1-source.tar.gz

下载地址: <ftp://sword:king@202.120.40.124:2121/project/+WSN/Software/Stage-3.2.2-Source.tar.gz>

也可以到<http://playerstage.sourceforge.net/> 去下载最新的版本进行安装。

安装过程:

1. 解压 stage 包:

```
tar zxvf Stage-3.2.2-source.tar.gz
```

2. 安装以及编译:

```
cd Stage-3.2.2
```

```
mkdir build
```

```
cd build
```

```
cmake ../
```

```
make
```

```
sudo make install
```


至此，stage 安装完毕。

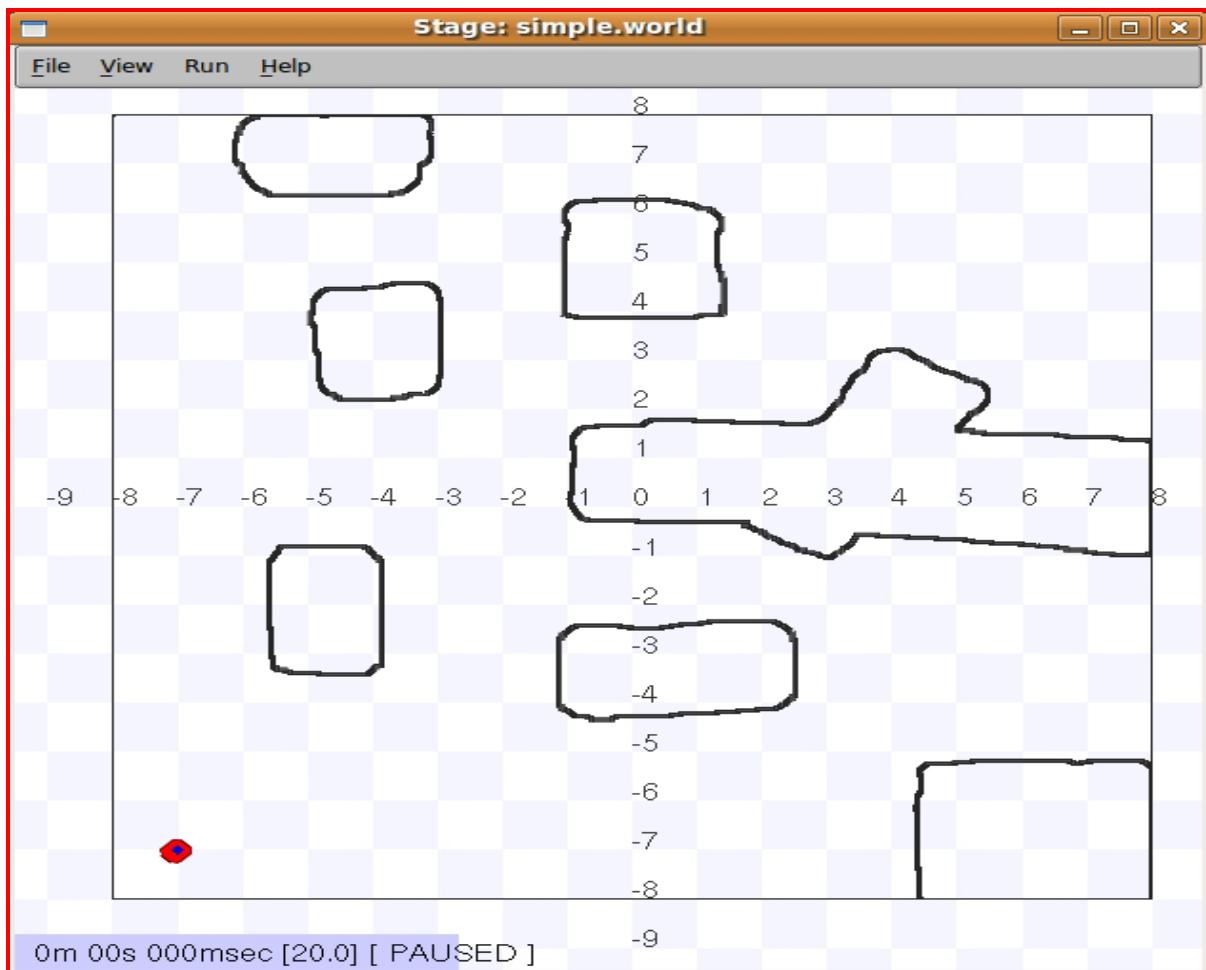
三、测试 player stage

Player Stage 安装完毕，需要对其进行测试是否能够成功运行。具体如下：

- 3.1、stage 测试：

进入到 Stage 3.2.2 文件目录中的 worlds 文件下输入：

stage simple.world 当出现如下界面时表示 stage 安装成功。



可能会出现如下或者类似如下错误：

Stage: error while loading shared libraries: libplayerdrivers.so.3.0: cannot open shared object file:

No such file or directory"

这里问题主要是 stage 对库的加载时默认从 `usr/lib` 下面加载的，而我们安装的库是放在 `usr/local/lib` 下面。

解决方法：

1. 将 `libplayerdrivers.so.3.0` 文件从 `/usr/local/lib` 目录实用 `link` 命令链接或者实用 `cp` 命令复制到 `/usr/lib` 目录下。
2. 将 `/usr/local/lib` 目录下的文件全部拷贝到 `/usr/lib` 下面

```
"sudo cp -fr /usr/local/lib/ /usr/lib"
```

推荐使用第二种方法，因为第一种方法会在测试 player 的时候还会出现新的共享库找不到的问题。原理和上面一样，此时可以再使用第一种解决方法进行逐个将所需文件从 `/usr/local/lib` 目录下链接或者拷贝到 `/usr/lib` 目录下。第二种方法在进行 player 测试的时候不会出现任何问题，当然它的弊端是拷贝了很多不必要的文件，占用了磁盘空间。

• 3.2 player 测试：

同样在 `worlds` 文件目录里面输入：

`player simple.cfg` 出现如下界面表示 player 可以正常运行。



注：这里可能会出现 stage 中类似的出错情况，解决方法参考 stage 中的错误解决方法。

四、配置自己的应用场景

该场景配置是基于清障组的实际配置来进行说明的。但同样适合于其它应用场景。

• 4.1、.world 文件配置

4.1.1、头文件声明：

```
include "pioneer.inc" //该头文件中定义了 robot 的基类模型，可以直接继承使用
```

```
include "map.inc"
```

//定义了地图相关的模型，直接使用即可

```
include "sick.inc"//定义了激光传感器模型。
```

4.1.2、地图配置：

```
# load an environment bitmap
```

```
floorplan
```

```
(
```

```
    name "cave"
```

```
    size [20.000 20.000 0.800]
```

```
//地图大小的长度宽度以及高度
```

```
    pose [0 0 0 0]
```

```
//位置
```

```
    bitmap "bitmaps/map.png"
```

```
//地图文件导入的位置
```

```
    color "cyan"
```

```
//地图显示颜色
```

```
    gui_grid 0
```

```
//有无划线格子 0 有, 1 无。
```

```
)
```

自己自己定义的地图可以使用画图软件画成.png 格式，然后放入到 worlds 文件下面，导入即可。

4.1.3、 机器人配置以及传感器配置：

```
pioneer2dx
```

```
//该机器人模型是从 pioneer.inc 文件中继承下来的类
```

```
(
```

```
    name "r0"
```

```
//机器人名字
```

```
    pose [ 6 6 0 90 ]
```

```
//机器人在地图中的位置
```

```
    sicklaser( )
```

```
//添加激光传感器
```

```

gripper( pose [0.230 0 -0.200 0] color "yellow" )//添加抓斗计用于夹起障碍物

blobfinder(
//添加 blobfinder 传感器，主要是用于探测清理的“垃圾”，区分障碍物与“垃圾”
    colors_count 2//可探测的颜色种类 2 种：red blue

    colors [ "red" "blue" ]

    image[160 120] #resolution
//blobfinder 探测到“垃圾”时显示“屏幕”大小。
    range 5.00
//blobfinder 所能探测到的范围
    alwayson 1
//一直打开
)

localization "gps"//使用 gps 定位
localization_origin [ 0 0 0 0 ] //以该位置为中心。
)

```

注：两个重要的传感器说明：

1. laser：激光传感器用于避障使用，防止机器人在运行过程中撞到障碍物，范围可以在 sick.inc 中设置。

2 blobfinder 传感器：用于探测所要清理的“垃圾”，blobfinder 中定义的颜色用于区分需要清理的垃圾和障碍物之间的区别。换言之在 blobfinder 中定义到的颜色是“垃圾”的颜色，垃圾的颜色必须与该颜色一致，没有定义的颜色就可以用做障碍物的颜色，比如说墙壁等。

• 4.2、.cfg 文件配置

4.2.1、simulation 驱动

```

driver

(

    name "stage"
    //表明我们使用仿真器，而非实体机器人。仿真器名为“stage”
    provides [ "simulation:0" ]
    //仿真器配置
    plugin "stageplugin"
    //导入所需要的库
    worldfile "demo1.world" //导入相应的.world 文件到 simulation 中

)

```

4.2.2、机器人驱动

```

driver

(

    name "stage"
    //机器人使用 stage 仿真器
    provides [ "6665:position2d:0" "6665:laser:0" "6665:sonar:0"
    "6665:blobfinder:0" "6665:speech:0" "6665:graphics2d:0" "6665:graphics3d:0"]
    //机器人相应的驱动配置。
    model "r0" //使用.world 文件中定义的 r0 模型。

)

```

说明：机器人 provides 中驱动声明的格式为：host:robot:interface:index, 对于在本地机器上进行开发的程序我们只需要定义端口号：接口：索引 即可。在多机器上就需要机器人的 host 地址了。

五、如何编写自己的控制算法来控制机器人运动

player stage 中控制机器人运动的算法有两种方式，一种是在 stage 层直接编写，另一种是基于 player 层利用代理的方式来编写。这里介绍第二种方法。

在 player stage 上进行开发的过程为：编写 world 文件->编写.cfg 文件->编写.cc 文件。

控制算法都是写在.cc 文件当中的，.cc 文件通过代理来于 player 交互，通过 player 来读取机器人的传感器信息，发送的控制命令也是通过 player 来“告诉”机器人的。可以把我们的机器人看作是计算机硬件，player 看作是操作系统，而.cc 文件看作是应用程序。应用程序通过操作系统与计算机硬件交互。下面的例子可以很好的介绍如何写控制算法。

六、一个完整的例子

该例子利用代理的方式为单机器人写了初步的避障算法和随机探测算法，可以帮助初学者尽快的熟悉开发流程。算法采用注释的方式进行介绍。

6.1:demo1.world 文件

代码：

```
include "pioneer.inc"
```

```
include "irobot.inc"
```

```
include "map.inc"
```

```
include "sick.inc"
```

```
window
```

```
(
```

```
    size [ 600.000 600.000 ] # in pixels
```

```
    scale 30.000 # pixels per meter, so the window is 20 meters high and 20
meters width
```

```
    center [ -0.019 -0.282 ]
```

```
    rotate [ 0 0 ]
```

```
show_data 1 # 1=on 0=off
```

```

)

# load an environment bitmap

floorplan

(

  name "cave"

  #size [300.000 300.000 0.800]

  size [20.000 20.000 0.800]

  pose [0 0 0 0]

  #bitmap "bitmaps/frieburg.png"

  #bitmap "bitmaps/java.png"

  #bitmap "bitmaps/example1.png"

  bitmap "bitmaps/map.png"

  color "cyan"

  gui_grid 0

  gui_outline 1
)

pioneer2dx

(
  # can refer to the robot by this name
  name "r0"
  pose [ 6 6 0 90 ]

```



```

sicklaser(

# ctrl "lasernoise" # uncomment this line to run a laser noise generator

)

#add gripper by liu

gripper( pose [0.230 0 -0.200 0] color "yellow" )

#add blod finder by liu

blobfinder(

    colors_count 2

    colors [ "red" "blue" ]

    image[160 120] #resolution

    range 5.00

    alwayson 1

)

#ctrl "wander"
# report error-free position in world coordinates
localization "gps"

localization_origin [ 0 0 0 0 ]

)

```

- 6.2: demo1.cfg 文件

源码

```
#load the Stage plugin simulation driver

# load the Stage plugin simulation driver

driver

(

  name "stage"

  provides [ "simulation:0" ]

  plugin "stageplugin"

  # load the named file into the simulator

  worldfile "demo1.world"

)

# Create a Stage driver and attach position2d and laser interfaces

# to the model "r0"

# "6665:sonar:0"

# to the model "r0"

driver

(

  name "stage"

  provides [ "6665:position2d:0" "6665:laser:0" "6665:sonar:0"
"6665:blobfinder:0" "6665:speech:0" "6665:graphics2d:0" "6665:graphics3d:0"]
```

```
model "r0"
```

```
)
```

• 6.3: demo1.cc 文件（控制算法）

源码:

```
#include<stdio.h>
```

```
#include<unistd.h>
```

```
#include<time.h>
```

```
#include <libplayerc++/playerc++.h>
```

```
using namespace std;
```

```
using namespace PlayerCc;
```

```
//wander function
```

```
void Wander(double *forwardSpeed, double *turnSpeed)
```

```
{
```

```
    int maxSpeed=1; //3 meter/second
```

```
    int maxTurn= 60;
```

```
    double fspeed,tspeed;
```

```
    fspeed = rand()%11;
```

```
    //(fspeed/10) is between 0 and 1;
```

```
    fspeed=(fspeed/10)*maxSpeed;//the speed is random
```

```
    tspeed=rand()%(2*maxTurn);
```

```

    tspeed=tspeed-maxTurn;

    //tspeed is between -maxTurn and maxTurn

    *forwardSpeed=fspeed;

    *turnSpeed=tspeed;

}

//avoid obstacle by laser :new edition

void AvoidObstacles(double *forwardSpeed, double
*turnSpeed, PlayerCc::LaserProxy &sp)

{

    double avoidDistance = 1.0;

    double add = 0.6;

    double avoidTurnSpeed = 60;

    if (sp[180] < avoidDistance )

    {

        *forwardSpeed = -0.2;

        *turnSpeed = avoidTurnSpeed;

    }

    else if (sp[90] < avoidDistance + add)

    {

```

```

        //turn left;

        *forwardSpeed = 0;

        *turnSpeed = (avoidTurnSpeed -20);

    }

    else if (sp[270] < avoidDistance + add )

    {

        //turn right

        *forwardSpeed = 0;

        *turnSpeed = (-1)*avoidTurnSpeed + 20;

    }

}

int main(int argc, char *argv[])

{
    PlayerClient    robot("localhost",6665); //定义客户端
    Graphics2dProxy gp2dProxy( &robot, 0 ); //定义 2 维代理
    Position2dProxy p2dProxy( &robot, 0 ); //定义定位代理
    BlobfinderProxy blobProxy( &robot, 0 ); //定义 blobfinder 代理
    LaserProxy      laserProxy( &robot, 0 ); //定义激光代理
    SimulationProxy simProxy( &robot, 0 ); //定义仿真代理
    double forwardSpeed=1;

```

```

    double turnSpeed=20;
    p2dProxy.RequestGeom();
//告诉代理获取设备信息
    laserProxy.RequestGeom();
//
//note:blobfinder does not have geometry

p2dProxy.SetMotorEnable(1); //使能机器人

// not generate the same random-number
srand ( time(NULL) );

robot.Read();

while(1)

{
    robot.Read();
    //move by random

    Wander(&forwardSpeed, &turnSpeed);

    AvoidObstacles(&forwardSpeed, &turnSpeed, laserProxy);

    p2dProxy.SetSpeed(forwardSpeed, dtor(turnSpeed)); //设置速度
    cout << "the speed is: " << forwardSpeed << "\t" << "the turnSpeed is:
" \

    << turnSpeed << endl; //打印速度信息
    sleep(1);

}

return 0;

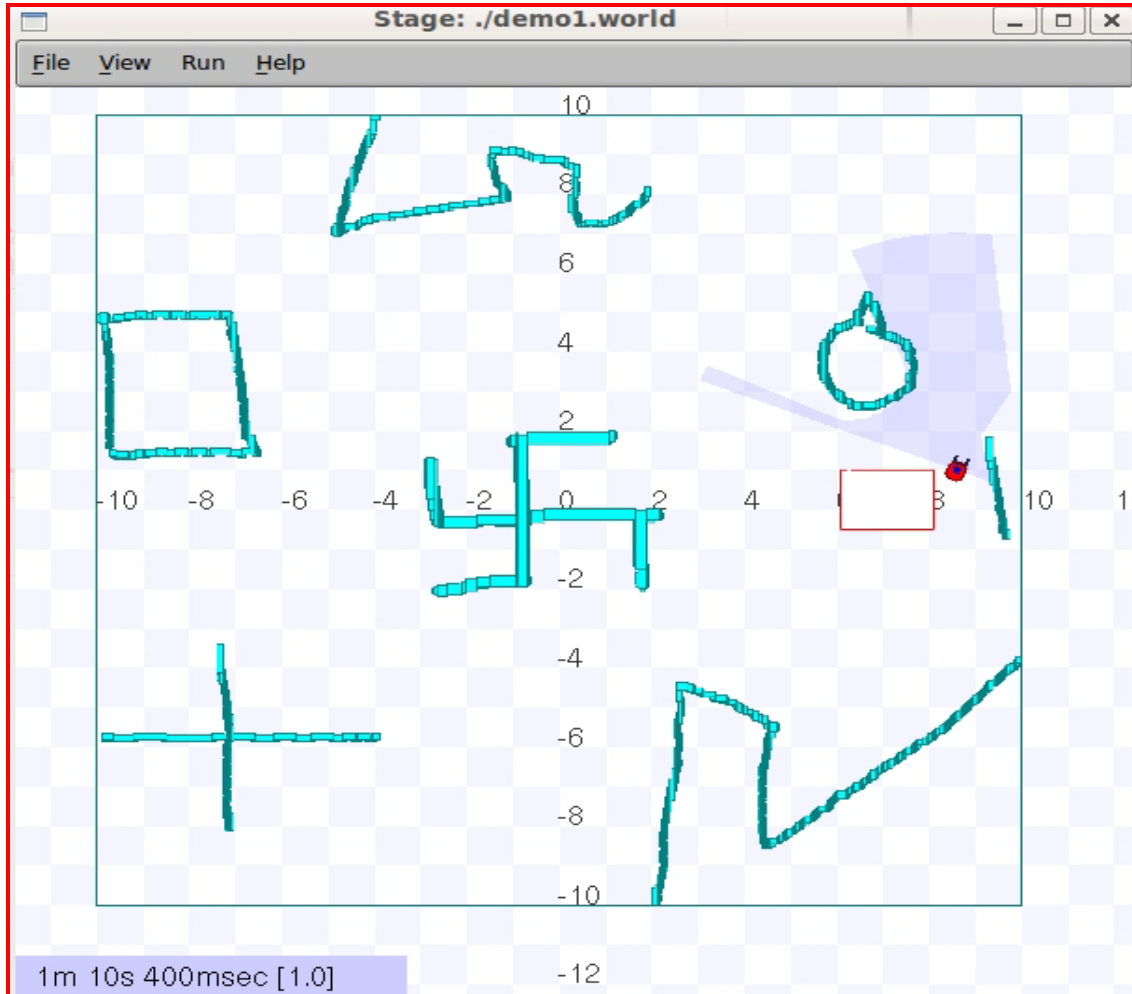
}

```

• 6.4 编译及运行：

打开一个终端进入到 worlds 目录：输入 `g++ -o demo $(pkg-config --libs --cflags playerc++) demo1.cc`

再打开一个新的终端进入到 worlds 文件下面，一个终端中运行 `player demo1.cfg`，会出现一个静态画面，切换到第一个终端，输入 `./demo` 回车，就会看到机器人可以成功跑起来，并能够满足基本的避障功能，终端不断打印出机器人的速度等信息，如下：



七、如何在 Stage 上进行算法仿真

7.1 为什么要在 stage 上进行算法仿真

Player/Stage 的架构是为了在不同机器人平台和仿真器上向用户提供统一的接口，以方便程序的移植。由于 Player 对机器人系统要求较高（需要运行在 Linux 系统上，以及支持 TCP 协议），并不是所有的机器人上都可以使用这套架构。如果不用 Player 的话，或者仅仅是进行仿真，这套架构就显得冗余，会增加算法实现的复杂度。Stage 上已经通过动态库的形式为用户提供编程接口，并且支持功能模块的扩展。如果没有什么特殊需求的话，完全可以在 stage 上实现你的算法。

7.2 Stage 的安装

可参照前面的介绍。如果用包含 wifi 模块 stage 的话，需要下载这个版本：
<ftp://sword:king@202.120.40.124:2121/project/+WSN/Software/Simulator/PlayerStage/wifistg.tar.gz>。



为什么安装的时候会提示找不到 libboost-thread1.35-dev libboost-signals1.35-dev 两个包？

这是由于操作系统版本问题，例如 Ubuntu9.04 中包含这两个包而 Ubuntu9.10 中这两个包已经升级为其他版本。解决的方法也很简单，换成更新版本的包就可以了。



为什么编译的时候会报错：error: new declaration ‘char* basename(const char*)’？

也是版本问题造成的。解决方法为在 /replace/replace.h 文件中添加两个宏定义，`#define HAVE_DIRNAME 1` 和 `#define HAVE_BASENAME 1`。

7.3 Stage 上如何运行自己的算法

这里以一个程序为例子，介绍如何在 Stage 上实现你的算法。该例子 `wander_wifi.cc` 文件在上一小节提供的 `stage` 包中可以找到的，位于 `/examples/ctrl/` 路径下。这一小节主要介绍程序的编译和运行。至于程序怎么写，在下一小节程序分析中介绍。

首先是把这个程序文件编译成为一个动态库文件供 `Stage` 调用，我们用 `gcc` 来做，在终端下输入这两条命令：

```
gcc -fPIC -c wander_wifi.cc
gcc -shared -o test.so wander_wifi.o
```

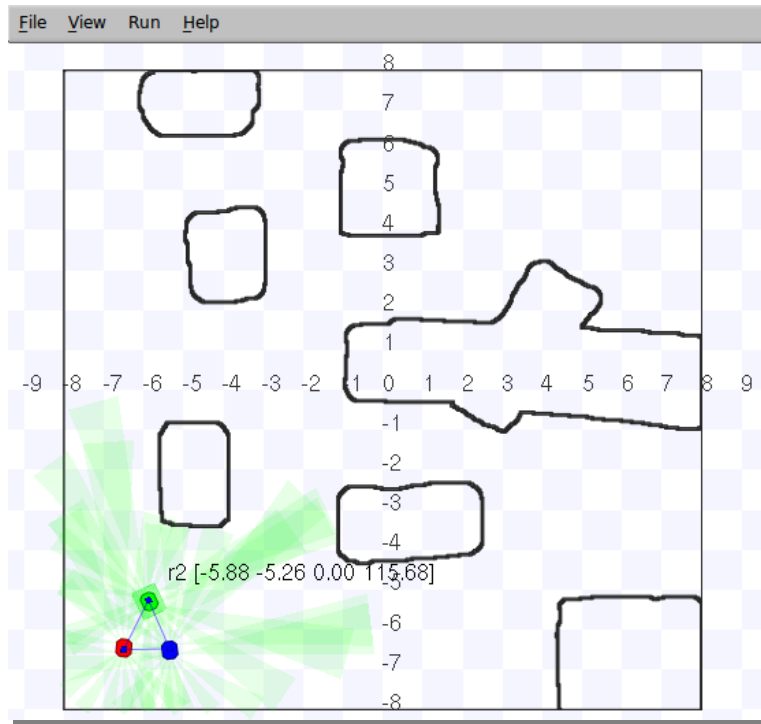


为什么编译的时候提示找不到 stage.hh 文件？

可以先看看 `wander_wifi.cc` 程序，尝试下自己解决这个问题。实际上 `stage.hh` 文件就在程序包中。把它拷到 `wander_wifi.cc` 存在的目录中就可以了。还有一个 `config.h` 的头文件也要拷过来，并且修改 `stage.hh` 中的 `#include <config.h>` 为 `#include "config.h"`。请大家思考一下为什么编译 `stage` 的时候这里没有报错。

然后把生成的动态库文件拷到 `/worlds/` 路径下，我们用其中的 `wifi.world` 文件做测试。修改 `wifi.world` 机器人控制定义字段，原来是 `"ctrl "wander_wifi""`，改为 `"/test.so"`。在终端中输入以下命令，如果 `stage` 跑起来了，那么恭喜你成功了。

```
stage wifi.world
```

程序运行示例

7.4 Stage 上怎样写程序

这里先把例子程序贴出来，通过对这个例子的分析，来看看如何在 **stage** 上实现自己的算法。为了看的清晰，我已经把一些细枝末节都用省略号代替，只留下程序主体结构。

```
#include "stage.hh"
using namespace Stg;

...

typedef struct
{
    uint32_t id;
    ModelPosition* pos;
    ModelLaser* laser;
    ModelWifi* wifi;
    int avoidcount, randcount;
} robot_t;

...

int LaserUpdate( Model* mod, robot_t* robot );
int PositionUpdate( Model* mod, robot_t* robot );
int WifiUpdate( Model* mod, robot_t* robot );

...
```

```

// Stage calls this when the model starts up
extern "C" int Init( Model* mod, CtrlArgs* args )
{
    // local arguments
    ...

    robot_t* robot = new robot_t;

    ...
    //注册类和 Update 函数
    robot->pos = (ModelPosition*)mod;
    robot->laser = (ModelLaser*)mod->GetChild( "laser:0" );
    robot->laser->AddUpdateCallback( (stg_model_callback_t)LaserUpdate,
robot );
    robot->wifi = (ModelWifi*)mod->GetChild("wifi:0");
    robot->wifi->AddUpdateCallback( (stg_model_callback_t)WifiUpdate,
robot);
    ...
    return 0; //ok
}

// inspect the laser data and decide what to do
int LaserUpdate( Model* mod, robot_t* robot )
{
    ...
    return 0;
}

int PositionUpdate( Model* mod, robot_t* robot )
{
    Pose pose = robot->pos->GetPose();

    printf( "Pose: [%.2f %.2f %.2f %.2f]\n",
        pose.x, pose.y, pose.z, pose.a );

    return 0; // run again
}

int WifiUpdate( Model* mod, robot_t* robot )
{
    ...

    return 0;
}

...

```

看了这个程序后，对动态库稍懂一些的人应该已经有些眉目了。程序最关键的是 Init 函数，这是个入口函数，其作用相当于一般程序中的 main 函数。Stage 运行时，会先调用 Init 函数，完成其他函数的注册和变量的初始化。

那这个例子在 Init 中做了些什么呢？首先我们看到 ModelPosition、ModelLaser、ModelWifi 这些类，每个类对应着仿真中机器人对应的元件。在每个时钟节拍，stage 会把机器人元件探测到的信息放到对应的类中，并把类的命令转化为仿真器中元件的动作。当然，这些前提是把 Stage 中机器人的元件和对应的类联系起来，也就是注册。还有一类 Update 函数，在每个时钟周期会被 Stage 调用，算法可以放到这里来实现，这些函数也要注册。

红色注释下面的代码就是完成注册功能。Init 函数传进来的 Model 类指针指向的是 Stage 中元件的地址，通过调用该类中的 GetChild 函数和 AddUpdateCallback 函数进行类和 Update 函数的注册。当然要在算法程序中使用某个元件，在场景定义的 world 文件中也要定义好。

这一节主要是介绍 stage 上实现算法的方法，对于具体元件如 wifi、laser 的使用，请参考相关的例子程序，在 /examples/ctrl 下都可以找得到。只要对 C++ 有基本的了解，一般都看的懂如何使用。

八、参考资料和实用网站

- [1] playerstage_instructions_2.0.pfd （强烈推荐，目前找到最全的一个手册）
- [2] player.pdf
- [3] <http://playerstage.sourceforge.net> player/stage 官方网站，这里可以下载最新版本软件以及补丁
- [4] <http://old.nabble.com/Player-Stage-Gazebo-f4302.html> 非常好的论坛，里面有 player stage 的开发人员和一些大牛，遇到问题可以在这里和他们交流。