Quick Start Guide

WHITE BOX™ ROBOTICS

PC-Bot 914

Disclaimer

Working with electronics and installing the plastics will require care and patience. **PROPER GROUNDING PROCEDURES** before handling the electronics. Touching the robot chassis (which is common grounded throughout) and at the same time touching something that is grounded in your home/office like metal pipes or the kitchen tap.

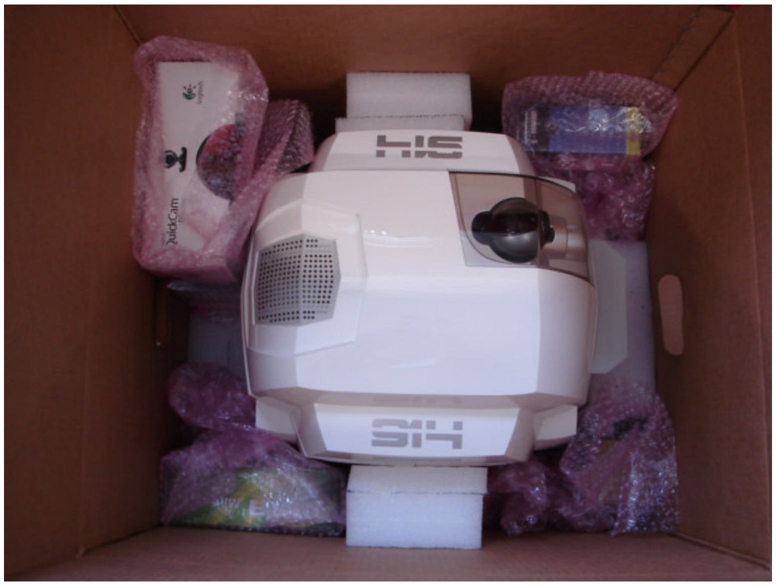
It is also expected that, working in a small area inside the unit that you perform careful and safe handling of the hardware so as not to cause short circuits. We ask that you make sure you give yourself enough working area and time for installation so as not to damage either the plastics or the electronics.

Liability

In no event will White Box Robotics, Inc. or Frontline Robotics, Inc. be liable for any damage, including loss of data or profits, cost of cover, or other incidental, consequential or indirect damages arising from the installation maintenance, use, performance, failure or interruption of White Box Robotics, Inc. or Frontline Robotics Inc. products, whatever the cause and on any theory of liability. This limitation applies even if White Box Robotics, Inc. or Frontline Robotics Inc. has been advised of the possibility of such damage.

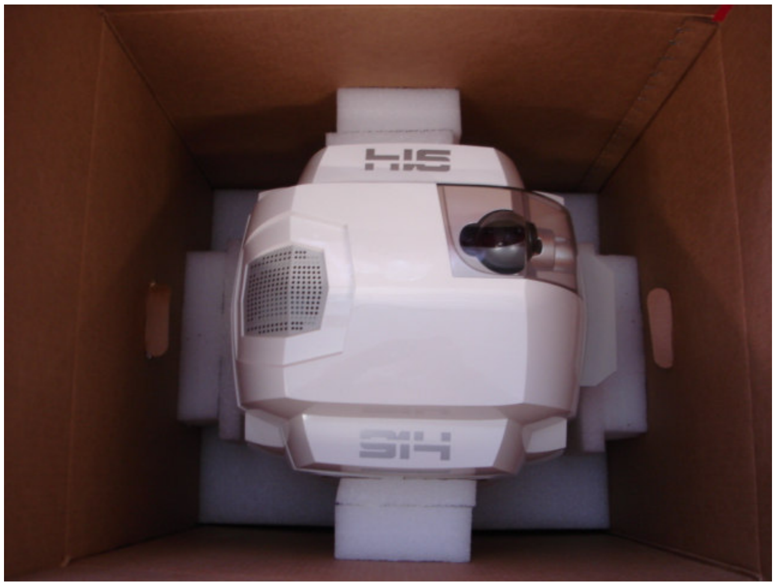
Unpacking the Robot

The following image shows how the PC-BOT should have arrived in the shipping box.



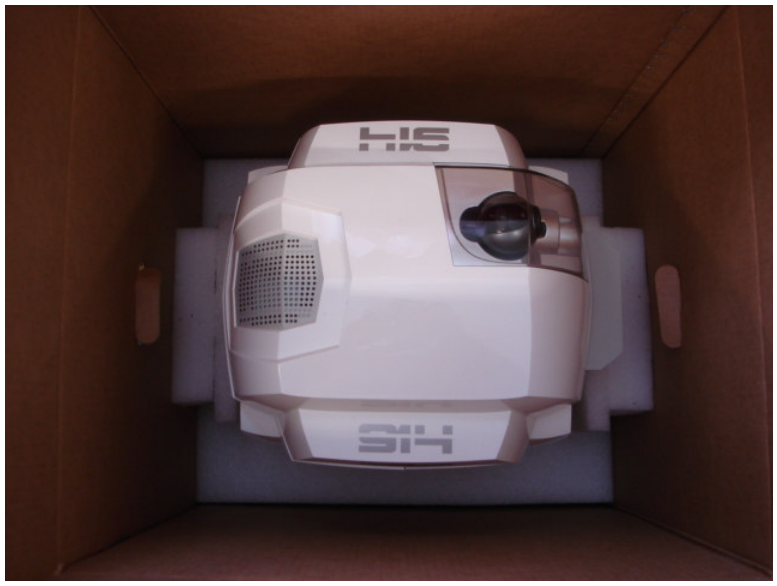
STEP 1 – Un-packing the robot

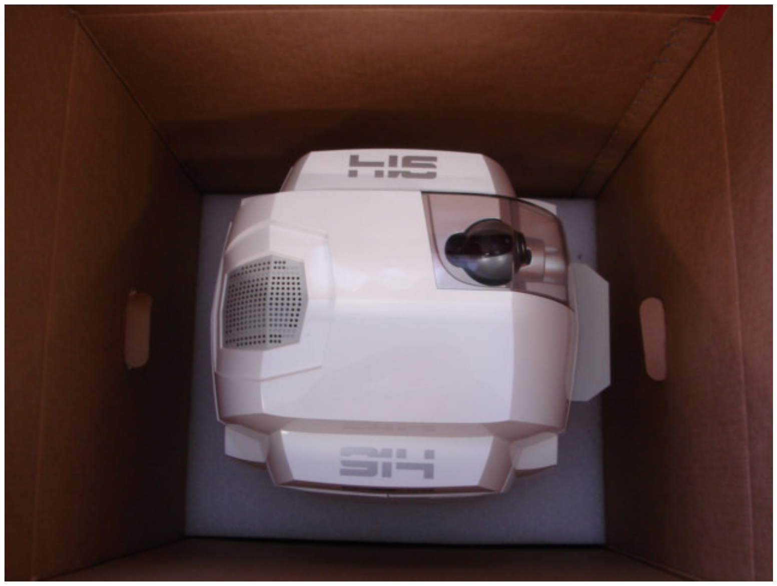
Remove the peripheral equipment wrapped in bubble wrap.



STEP 2 – Un-packing the robot cont.

Carefully remove the side and front/back supporting foam.

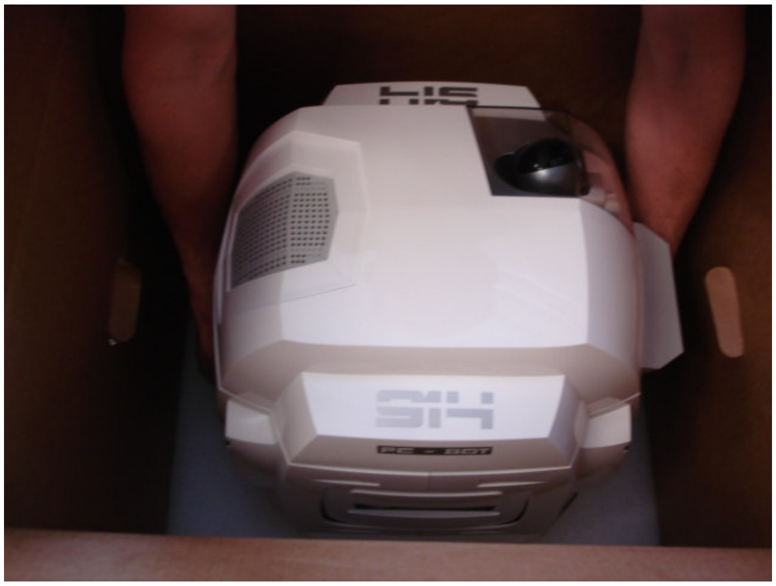




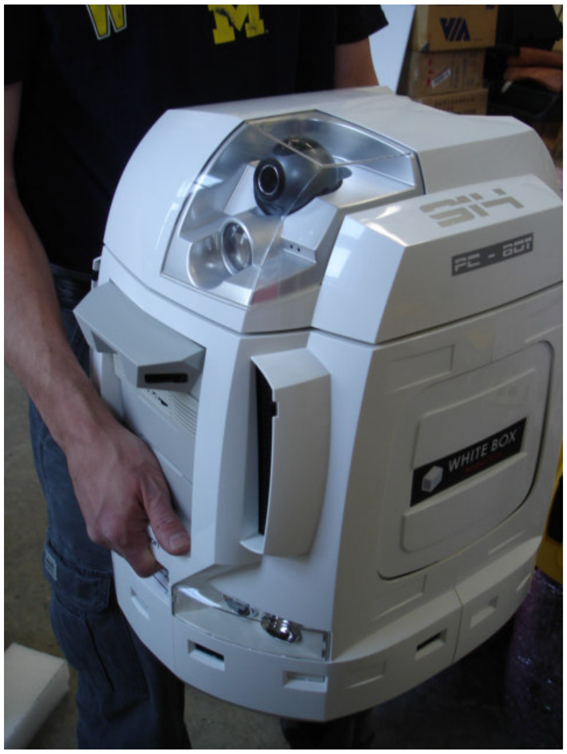
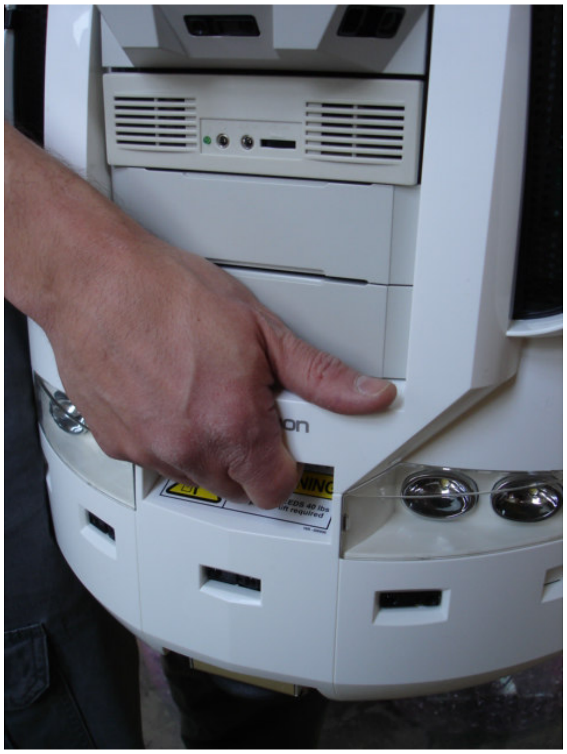
STEP 3 - Un-packing the robot cont.

Reach into the box and grab the handles on the PC-BOT. They are somewhat hidden from above. Lift the robot straight up and out of the box.

WARNING! This robot is heavy. People with back problems should not attempt to lift this robot. Always use proper lifting technique when lifting heavy items. Keep you knees bent and your back straight and lift gently.



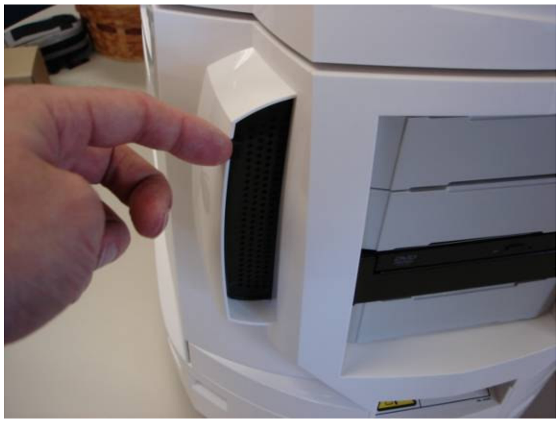
The following image shows where the handles are located, you'll have to feel for them when the robot is in the box. There are two handles front and back.

Setting-up the robot

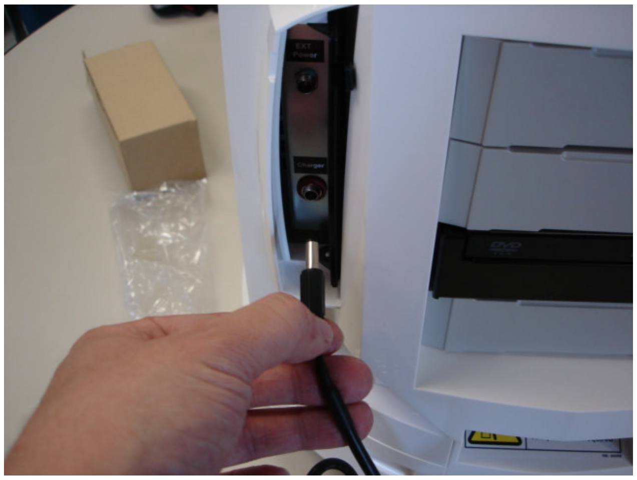
STEP 1 – Charging the robot batteries

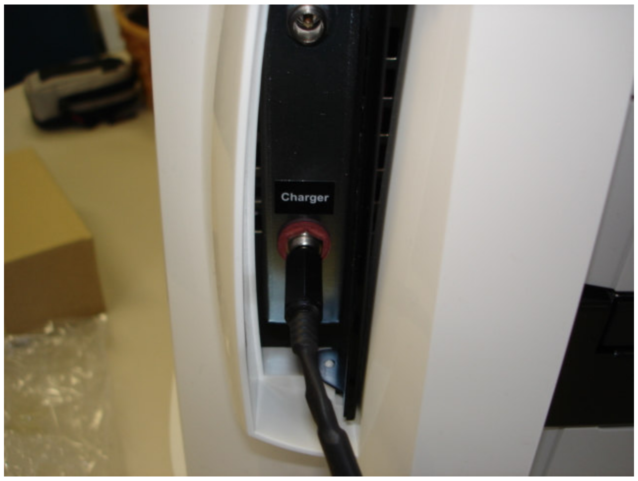
Now that the robot is out of the box, you should plug the charger in to charge the batteries. The charger connector is found behind the left rear vent door on the back of the robot.



Flip the door open as shown in the above picture.

Unpack the charger and plug the charger jack into the lower connector on the robot labeled “Charger”.





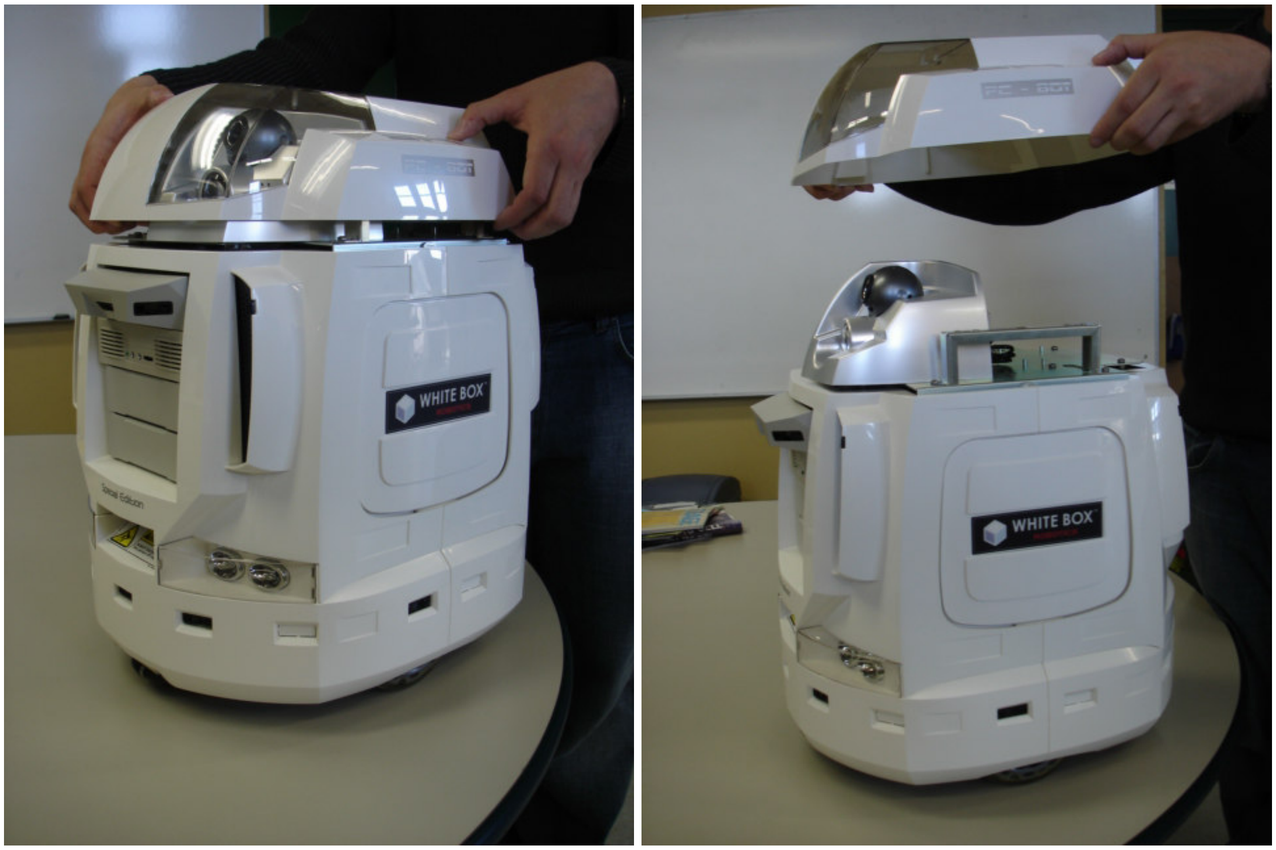
The charger LED will first be GREEN after the AC cord is plugged into the wall. It will then change to ORANGE when it is charging the batteries on the PC-BOT. It will then switch back to GREEN when the batteries are fully charged (it typically takes 2 -2.5 hours to charge the batteries from the auto-shutoff point).

Charging and shutdown

If the battery level goes below **11.2V** the robot will auto-shutdown. Either while you are running the robot, and especially when you are finished with it for the day, PLEASE REMEMBER TO RE-CHARGE THE BATTERIES.

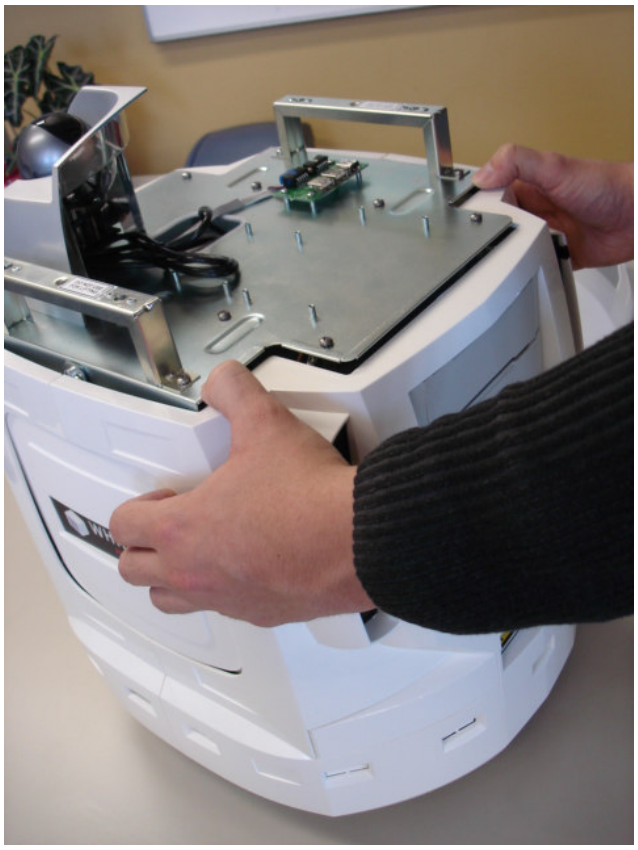
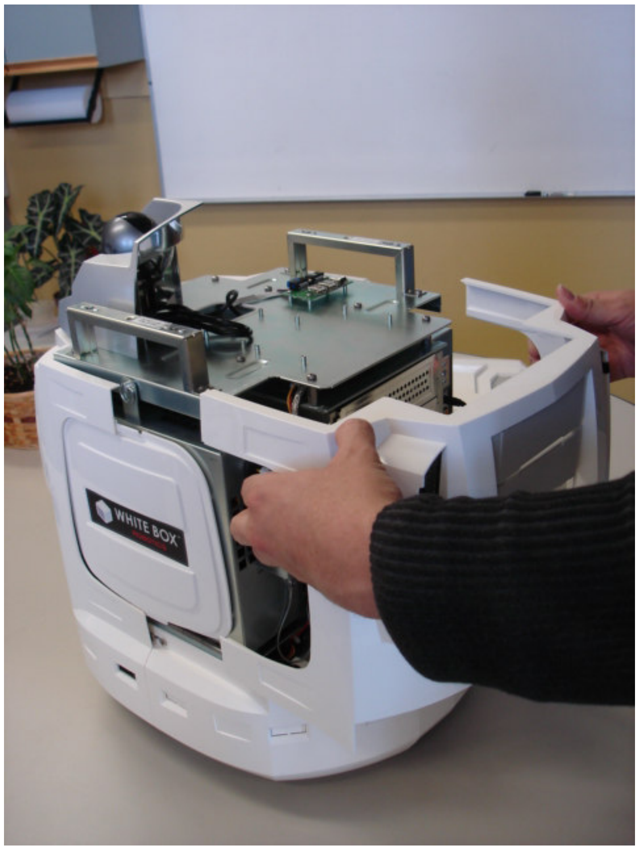
STEP 2 – Removing the head panel

The head and front/back torso body panels are attached with ball studs and sockets. This allows for easy removal of the panels without tools. Gently remove the head by grabbing under the separation (between the head and torso plastics) and pulling straight up.

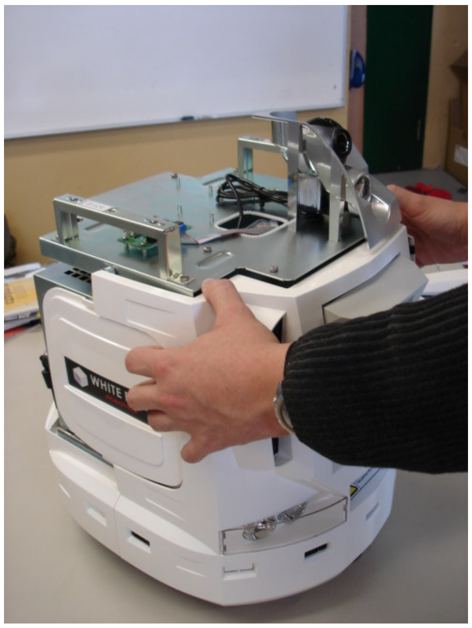
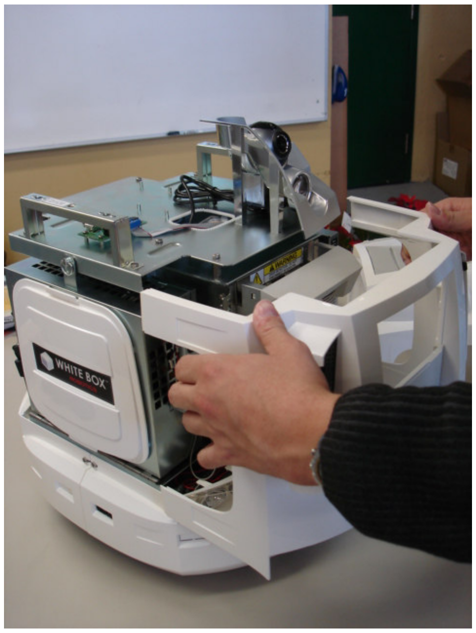


STEP 3 – Removing the body panels

The torso body panels are removed in the same way as shown, except pulled backwards or forwards depending on the torso body panel. You can use your thumbs to leaver while pulling on the body panels with your finger tips.

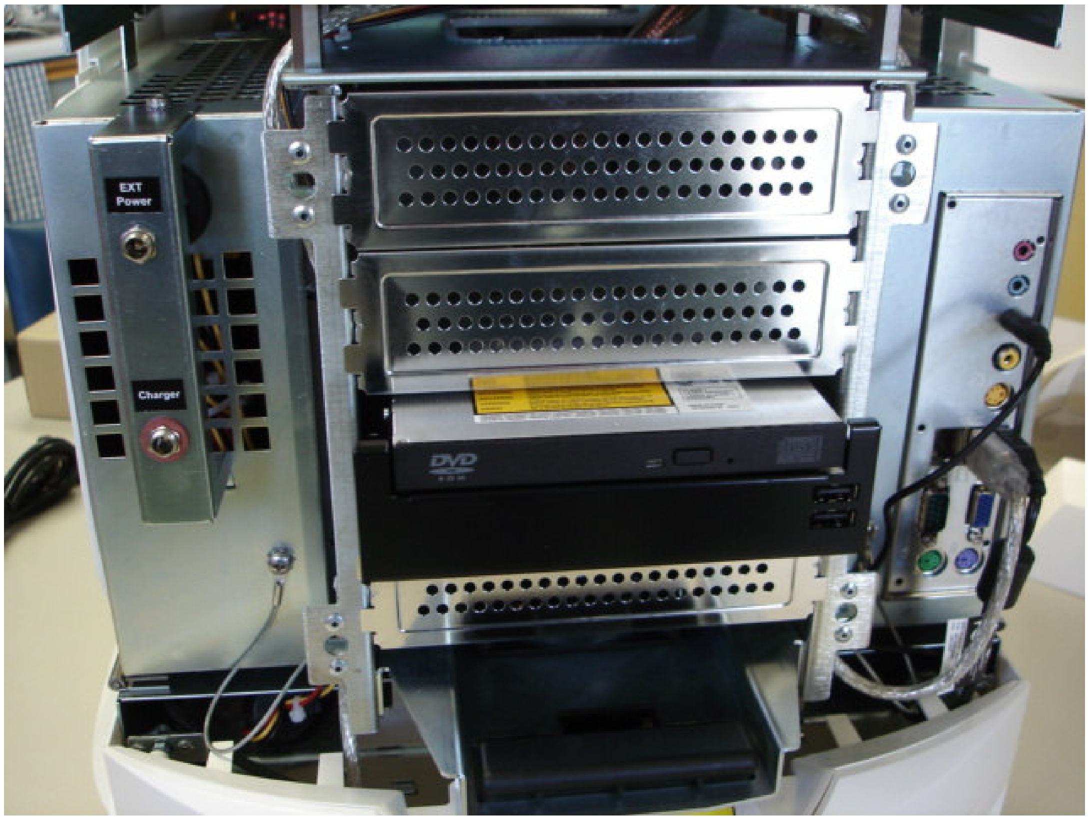



Both body panels are removed in the same way.

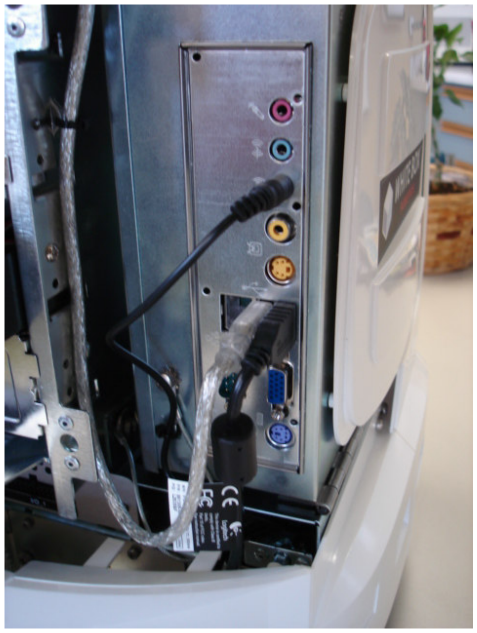



STEP 4 – Plug in the monitor, keyboard and mouse

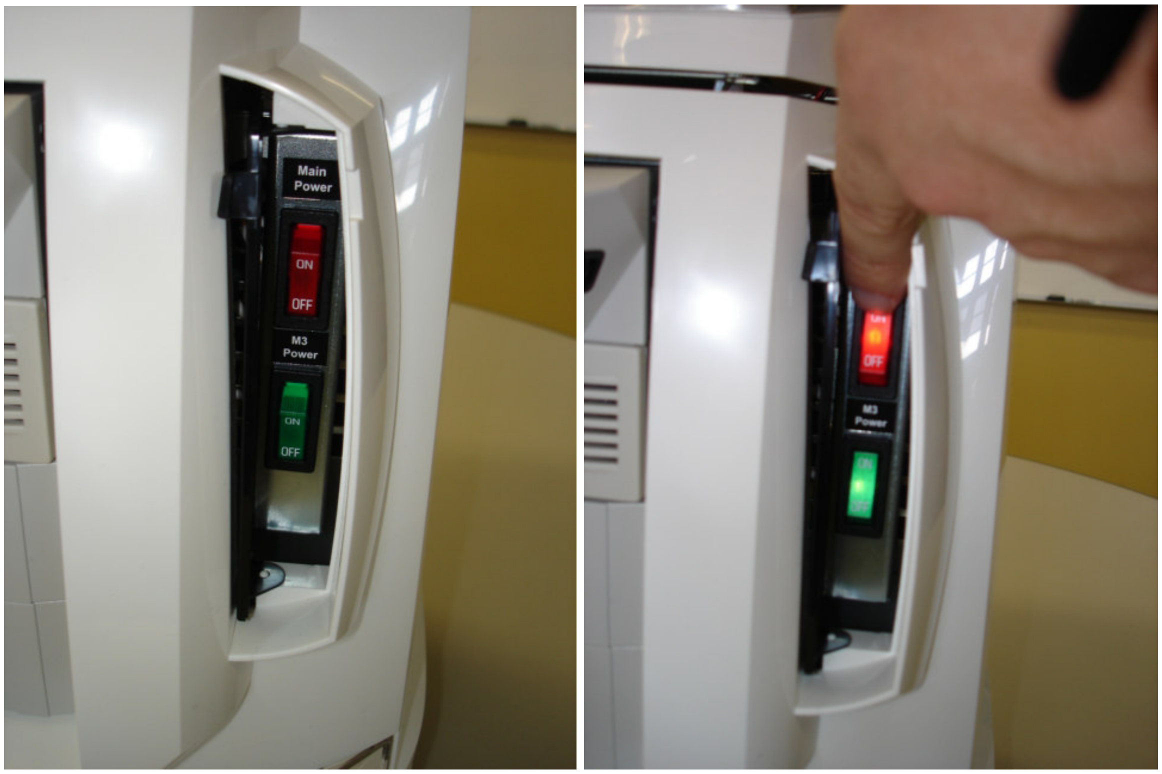
When viewing the back of the robot, you will see the External Power Supply jack (power supply not included) and Charger jack on the left side, and on the right side the back panel of the Mini-ITX.



Plug the monitor, keyboard and mouse into the Mini-ITX back panel in order to setup your robot, like the network etc.



STEP 5 – Start your system



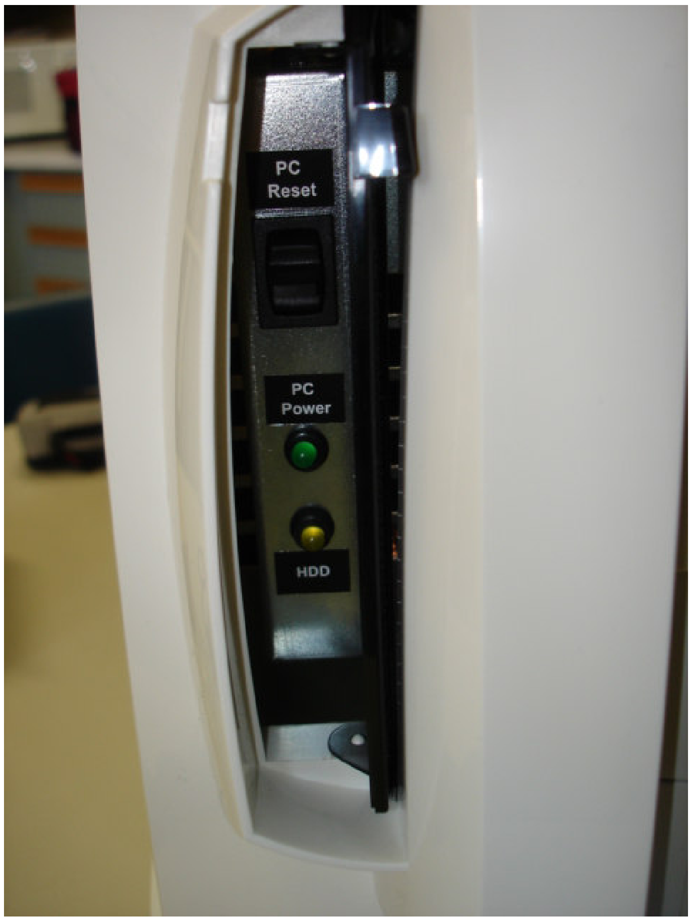
Switch on the Main Power switch (RED) **AND** M3 Power switch (GREEN) on the front of the robot. **WAIT 5 seconds** and the M2-ATX Power Supply will cycle the power to the PC automatically.

The M3 Power switch is used to turn on the M3 Controller. If this switch is OFF, the M3 will not be able to drive the motors or read the sensors.

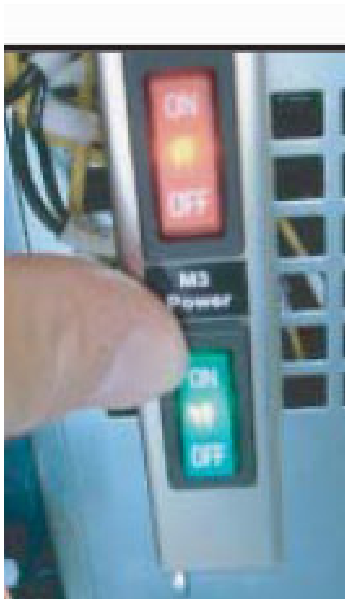
When the battery voltage drops to **11.2V** (i.e. when the batteries need recharging, the M2-ATX will cycle the power to the PC again to auto-shutdown before the power is cut (45 seconds later) and the batteries are protected from total discharge.

This is the same as the user pushing the PC ON switch on the CPU and shutting down the computer.

On the other side of the front of the robot, you can still use the PC ON and PC Reset switch just like on a regular computer.

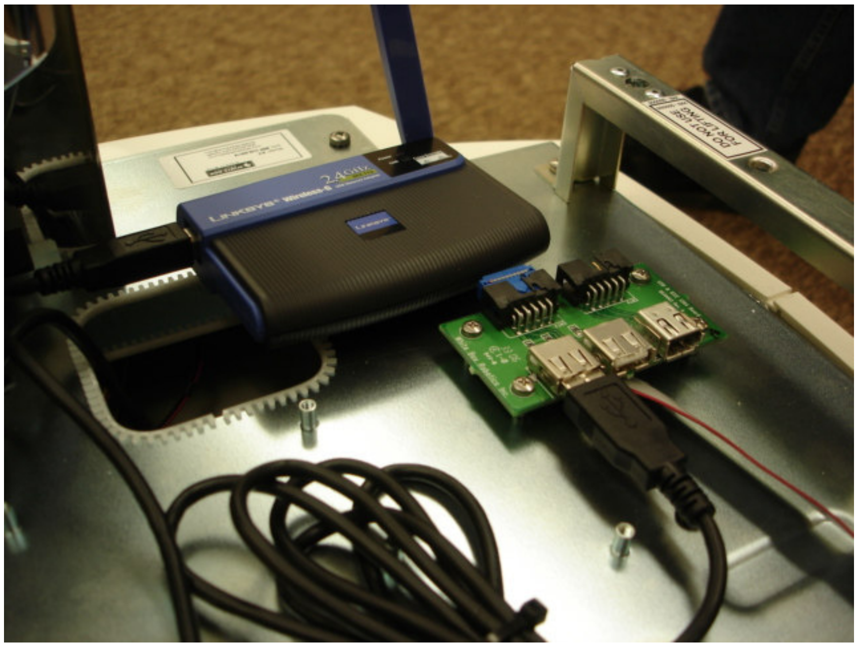


NOTE: If the robot is sitting on a desk, it might be advisable to turn the M3 Switch OFF so that the robot cannot drive off the table.



STEP 6 – Connecting Peripheral Devices

You should connect the Wireless device to the USB board mounted on the head if this is not already done.



In this location the head can still be placed back on the robot and not interfere with the Wireless USB Network Adapter or its antennae.

Reference Documents and Links

White Box Robotics Webpage: http://www.whiteboxrobotics.com/

The Official Enthusiast Site for the 914 PC-BOT: http://www.914pcbots.com/community/

Support Documents – (Quick Start Guides, Basic Unit Computer Installation, Plastics Assembly, etc.)
http://www.whiteboxrobotics.com/2006/PCBOTs/support.html

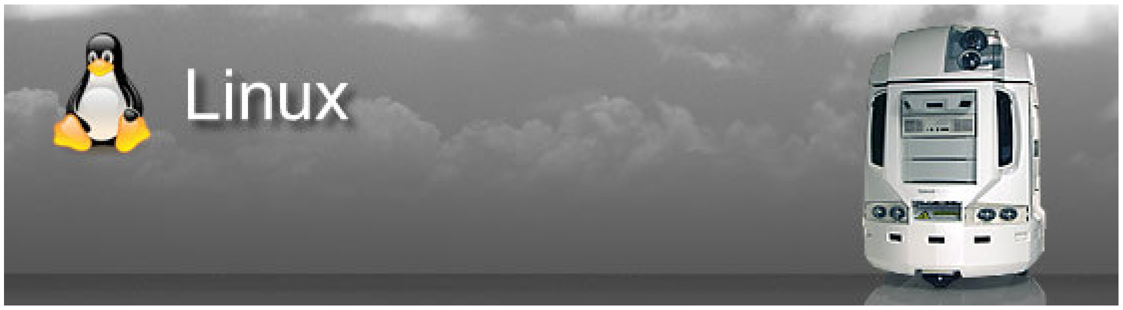
Technical Specifications

Wiring and Power: http://www.whiteboxrobotics.com/2006/PCBOTs/pdf/PC-Bot_Tech_Spec-WiringPowerv1.1.pdf

I/O Board Block Diagram: http://www.whiteboxrobotics.com/2006/PCBOTs/pdf/PC-Bot_Tech_Spec-IOBoardBlockDiagramv1.1.pdf

Infra-red Sensors: http://www.whiteboxrobotics.com/2006/PCBOTs/pdf/PC-Bot_Tech_Spec-Infra-redSensorsv1.1.pdf

PC-BOT Software Section



914 PC-BOT, Linux Configuration, Systems information

OS installed: Linux - Ubuntu 6.06.1 LTS
Username: wbr
Password: wbr
(SU) **Root password**: wbr

Disclaimer:

This handbook is intended to introduce the basics of using Player/Stage as driver software for the White Box Robotics 914 PC-BOT and is not meant to replace the actual Player and Stage manuals. It is expected that you already have deep and existing knowledge of Linux as well as knowledge of the general constraints of working inside the Linux environment and some knowledge of working with Player/Stage and its components.

Initial Linux Application Configurations and uses

Remote Desktop – To allow for 'over the network' connections to the desktop of the robot.

Remote desktop control is provided through x11vnc. For detailed information see: http://www.karlrunge.com/x11vnc/

x11vnc is run each time Gnome gdm is started (each time the PC-BOT reboots).

To accomplish this, the following line was added to **/etc/X11/gdm/Init/Defaults**

```
x11vnc -rfbauth /home/wbr/.vncpasswd -forever -bg
```

The program runs in the background. To kill the program, execute the following command:

```
$ sudo kill -9 `pidof x11vnc`
```

password is **wbr**

You can verify the program was terminated by executing:

```
$ ps aux|grep x11vnc
```

You should not see any instance of x11vnc running.

To restart the service after it has been terminated, simply execute the command below:

```
$ x11vnc -display :0 -rfbauth /home/wbr/.vncpasswd -auth /home/wbr/.Xauthority -forever -bg
```

To connect to the PC-BOT desktop remotely, use vncviewer (in Linux) or any Windows-based VNC client. In Linux, execute the following command to connect:

$ **vncviewer IP-of-PC-BOT:0** # you need to know the IP address of the PC-BOT on your local network the password has been set to '**whitebox**'. See the x11vnc link above for details on setting the password. To exit vncviewer, hit **Ctrl-C** in the terminal window or click X in the vncviewer window.

If using Windows, a good VNC Viewer to use is Real VNC: http://www.realvnc.com/

If using Linux (Windows is also supported), then you can use: http://www.tightvnc.com/

All VNC applications are supported and often a VNC viewer is already installed in your Linux OS.

Player Server – Running in the Background at Start-up

The Player server is started in the background at startup. To accomplish this, the following lines were added to **/etc/rc.local**

```
if [ -x /usr/local/bin/player ]; then
  echo "*********** Trying to run player **********"
  /usr/local/bin/player /home/wbr/wbr914.cfg &
fi
```

To verify it is running execute:

```
$ ps aux|grep player
```

You should see the instance of player running. Since it is running in the background, the only way to terminate the service is by executing:

```
$ sudo kill -9 `pidof player`
```

password is **wbr**

To verify the service has been terminated, type the following command:

```
$ ps aux|grep player
```

There are two ways to restart the server: 1. Re-running the rc.local script which will run the server in the background; 2. Opening a new terminal window and manually starting player.

For option 1: execute: $ **sudo /etc/rc.local** # press enter after you see: Listening on ports: 6665 and player will be running in the background.

For option 2: open a new terminal window and execute: $ **player /home/wbr/wbr914.cfg**
To close this instance of player, press **Ctrl-C** in the terminal window.

Player/Stage Software Introduction

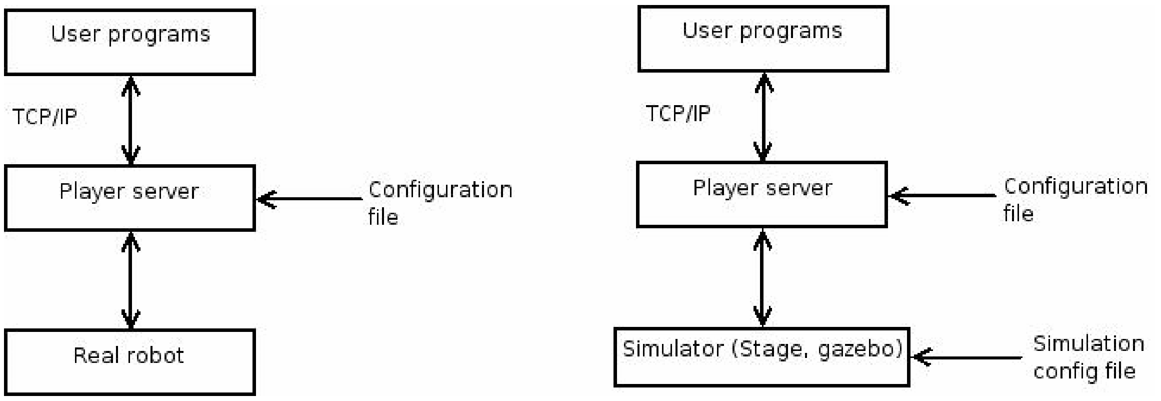
Player

Player is a network server for robot control. Installed on your robot, Player provides an elegant and simple interface to the robot's sensors and actuators over IP networks. The client program, potentially loaded on a remote client-computer talks to Player via a TCP socket, reading data from sensors, writing commands to actuators, and configuring devices on the fly. It is officially described as a server running on a robot providing access to hardware and sensors to remote programs.

How Player works

The Player server software provides an abstract interface to multiple robotic devices, including mobile robot bases, sensors, etc as well as your 914 PC-BOT. Player communicates with these specific devices using individual device drivers, but provides to its client-side a standard generic device interface. For example, Player may use SICK LMS-200 and Pioneer drivers, but simply provides to clients "laser" and "position" ("position" is a movable mobile robot base), "sonar", etc. interfaces. This allows clients to be portable to other robots.

The readers should note: All three programs, Player, Stage and Gazebo, have the same interface so programs used in either of the simulation programs can be used unchanged on the robots. They are designed to work together and provide distinct features.

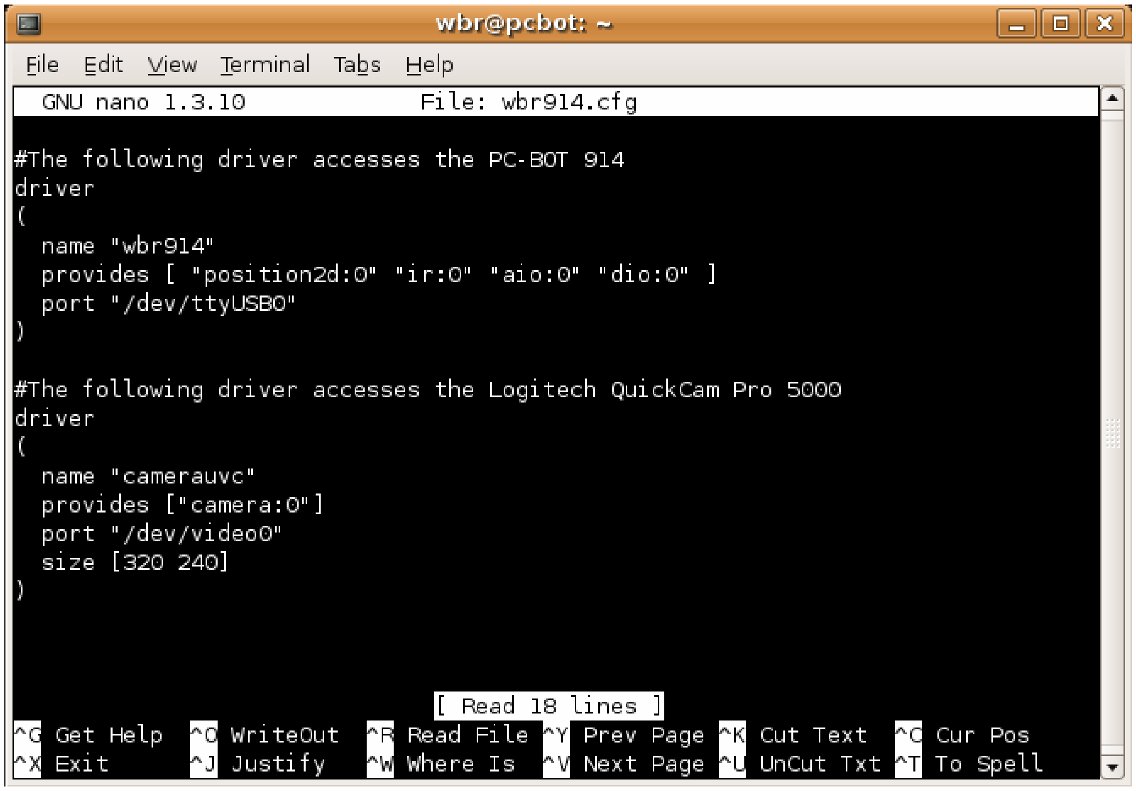


The most common way to use Player is to run the player server on your robot, then access your robot's devices with a client program running on a desktop or laptop computer.

Robot Operation

To access a robot, you need to write (or edit) a Player configuration file, (usually with the extension **.cfg)** that points at the driver(s) software required to control your robot. This links Player, through the device driver and configures the hardware to the Player software. **The job of this configuration file is to map the physical devices on the robot to the virtual Player devices (so that is shows and can be controlled inside the simulated environment).**

We provide the following for the default configuration 914 PC-BOT at this location:
/home/wbr/wbr914.cfg



```
  GNU nano 1.3.10             File: wbr914.cfg

#The following driver accesses the PC-BOT 914
driver
(
  name "wbr914"
  provides [ "position2d:0" "ir:0" "aio:0" "dio:0" ]
  port "/dev/ttyUSB0"
)

#The following driver accesses the Logitech QuickCam Pro 5000
driver
(
  name "camerauvc"
  provides ["camera:0"]
  port "/dev/video0"
  size [320 240]
)



                               [ Read 18 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Txt ^T To Spell
```

The wbr914 driver provides the following device interfaces:

position2d
This interface returns position data, and accepts velocity commands.

ir
This interface returns the IR range data.

aio
This interface returns the analog input data from the optional 2nd I/O board.

dio
This interface returns the digital input information and allows control of the digital outputs on all installed White Box Robotics I/O boards. The first I/O board supplies 8 digital inputs and outputs and the **optional** second I/O board supplies an additional 8 digital inputs and outputs.

Camera
Camera imagery.

Adding more sensors to the physical robot

As you add more sensors to the physical robot, you will need to create additional driver files to communicate with the hardware and return the sensor readings.

A number of drivers have already been written for the most popular sensors and are included in Player by default. (Here is a listing of the available drivers: http://playerstage.sourceforge.net/doc/Player-2.0.0/player/group__drivers.html) All related configuration options can be found there. To use those devices simply add them to the **.cfg** file to allow the robot to access the sensors.

Examples:

Hokuyo URG scanning laser range-finder

Text to add to your robot .cfg file

```
driver
(
  name "urglaser"
  provides ["laser:0"]
  port "/dev/ttyACM0"
)
```

Sick LMS-200

Text to add to your robot .cfg file

```
driver
(
  name "sicklms200"
  provides ["laser:0"]
  port "/dev/ttyS0"
  resolution 100   # Angular resolution 1 degree (181 readings @ 10Hz)
  range_res 10     # Range resolution 1 cm (maximum range 81.92m)
)
```

Logitech QuickCam Orbit

Text to add to your robot .cfg file

```
driver
(
  name "spheredriver"
  provides ["camera:0"
            "ptz:0"]
  port "/dev/video0"
  size [320 240]
  mode "RGB24"
  automatic_wb "fl"
  framerate 30
  shutter_speed   64000
  automatic_gain  10000
  dump_settings 0
)
```

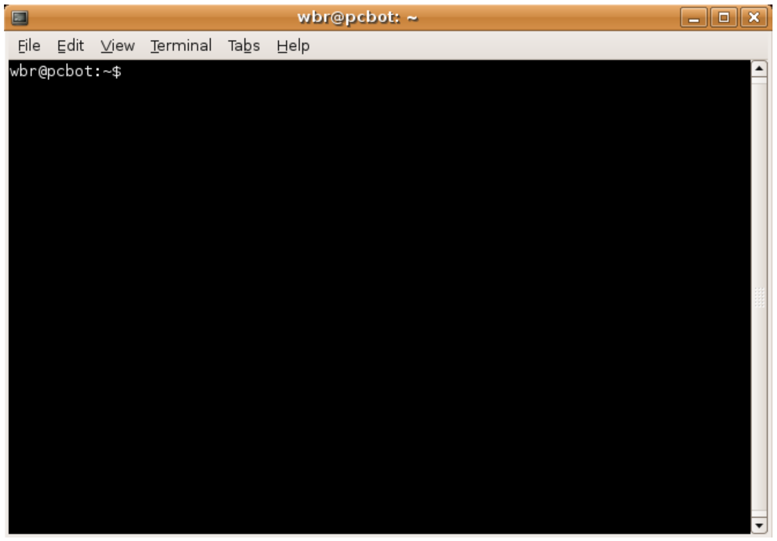
From the above examples, adding one of the above sensors to your robot the .cfg file would look like the following:

```
driver
(
  name "wbr914"
  provides [ "position2d:0" "ir:0" "aio:0" "dio:0" ]
  port "/dev/ttyUSB0"
)

driver
(
  name “camerauvc”
  provides [“camera:0”]
  port “/dev/video0”
  size [320 240]
)

driver
(
  name "urglaser"
  provides ["laser:0"]
  port "/dev/ttyACM0"
)
```

Controlling the PC-BOT – Using ‘**playerv**’

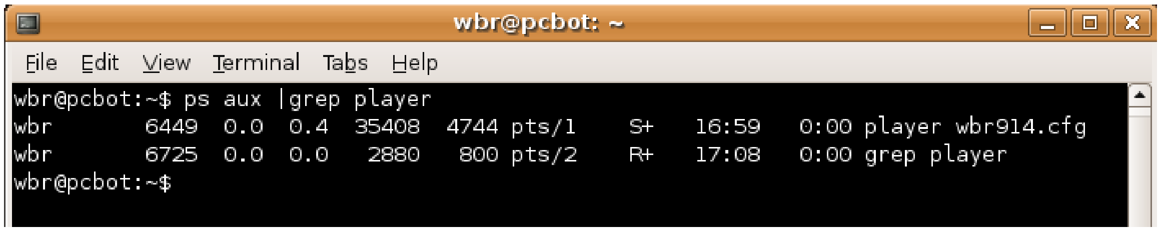
Step 1: Make sure the physical robot is in a safe place before attempting to drive the motors. Or, raise the robot (using block under the base) so that the wheels are not touching the surface if it is sitting on a table.

Step 2: Open a terminal window.



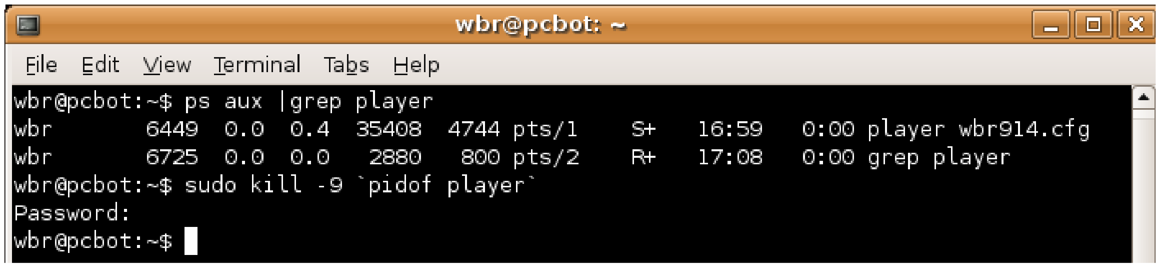
```
wbr@pcbot:~$
```

Step 3a: Player automatically runs upon boot-up, to allow the user to easily connect. This can be seen by typing ‘ps aux lgrep player’.



```
wbr@pcbot:~$ ps aux |grep player
wbr       6449  0.0  0.4  35408  4744 pts/1    S+   16:59   0:00 player wbr914.cfg
wbr       6725  0.0  0.0   2880   800 pts/2    R+   17:08   0:00 grep player
wbr@pcbot:~$
```

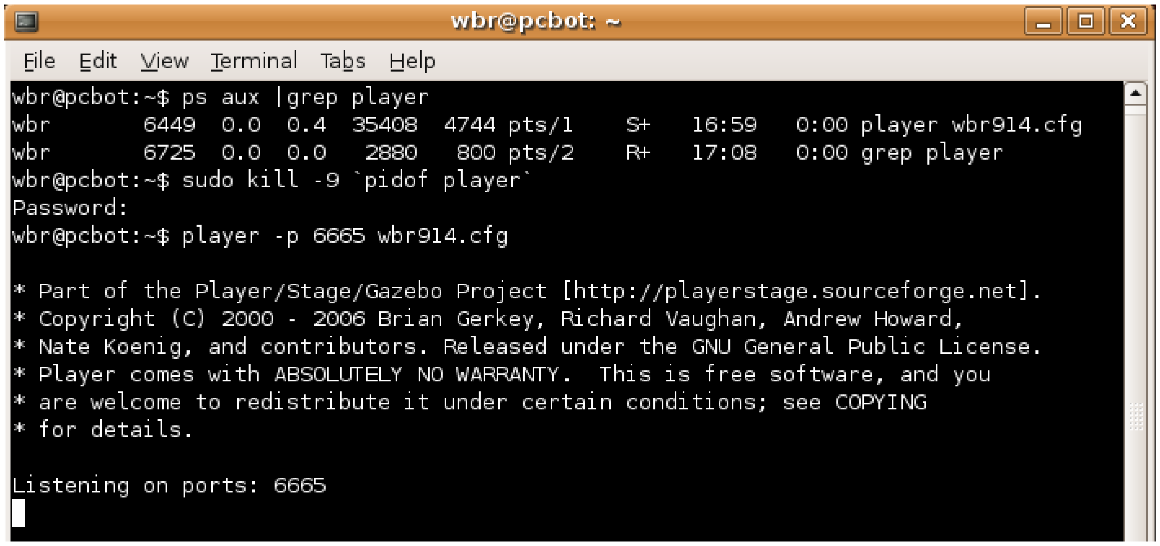
Typing this line can stop Player: ‘sudo kill -9 `pidof player`
Password is ‘wbr’



```
wbr@pcbot:~$ ps aux |grep player
wbr       6449  0.0  0.4  35408  4744 pts/1    S+   16:59   0:00 player wbr914.cfg
wbr       6725  0.0  0.0   2880   800 pts/2    R+   17:08   0:00 grep player
wbr@pcbot:~$ sudo kill -9 `pidof player`
Password:
wbr@pcbot:~$
```

Note: Player automatically uses the ‘/home/wbr/wbr914.cfg’ file. If you want to run a different player session, for example using the Stage simulation, you will need to stop this process before running the ‘player’ command with the new ‘.cfg’ file.

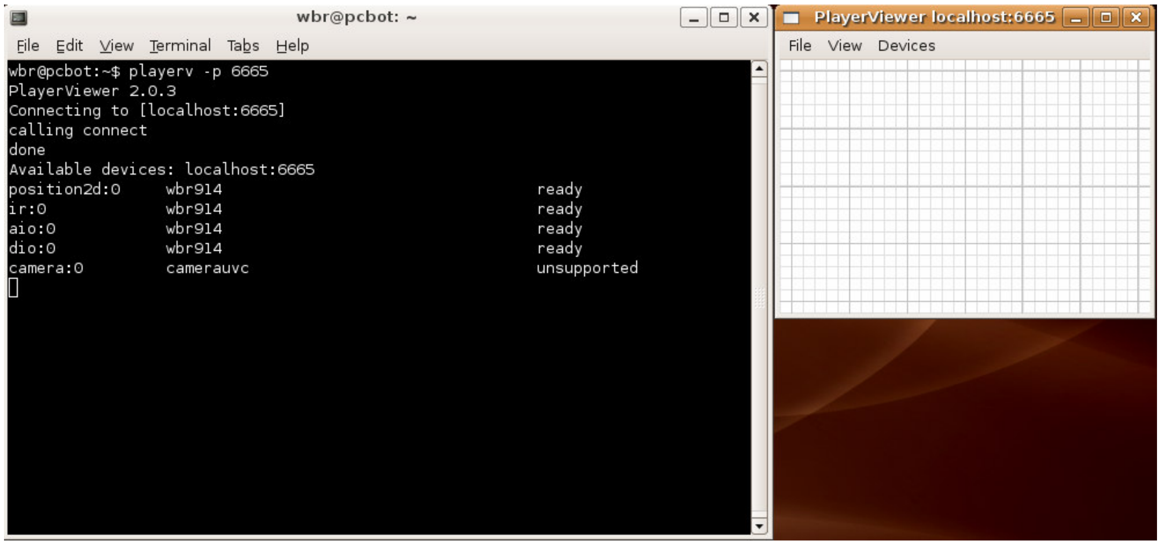
Step 3b: On the command line, type: ‘player –p 6665 wbr914.cfg’



```
wbr@pcbot:~$ ps aux |grep player
wbr       6449  0.0  0.4  35408  4744 pts/1    S+   16:59   0:00 player wbr914.cfg
wbr       6725  0.0  0.0   2880   800 pts/2    R+   17:08   0:00 grep player
wbr@pcbot:~$ sudo kill -9 `pidof player`
Password:
wbr@pcbot:~$ player -p 6665 wbr914.cfg

* Part of the Player/Stage/Gazebo Project [http://playerstage.sourceforge.net].
* Copyright (C) 2000 - 2006 Brian Gerkey, Richard Vaughan, Andrew Howard,
* Nate Koenig, and contributors. Released under the GNU General Public License.
* Player comes with ABSOLUTELY NO WARRANTY.  This is free software, and you
* are welcome to redistribute it under certain conditions; see COPYING
* for details.

Listening on ports: 6665
```

This initiates the player server running on the physical PC-BOT to listen on port 6665 (the default port).

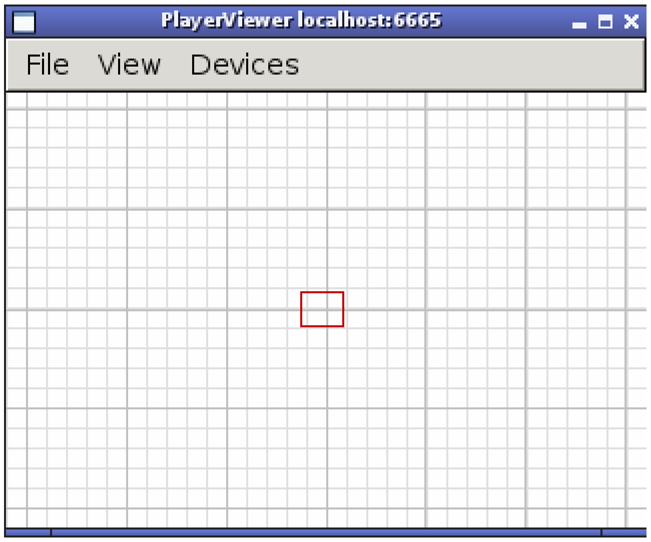
Step 4: Now open another terminal window and type 'playerv –p 6665'



```
wbr@pcbot:~$ playerv -p 6665
PlayerViewer 2.0.3
Connecting to [localhost:6665]
calling connect
done
Available devices: localhost:6665
position2d:0     wbr914                              ready
ir:0             wbr914                              ready
aio:0            wbr914                              ready
dio:0            wbr914                              ready
camera:0         camerauvc                           unsupported
```

A player client-side window opens and initially should display nothing at all. This window however will remain open and will pass the results to you from the open and now connected robot. Having found the robot a further window opens on the right (see above).

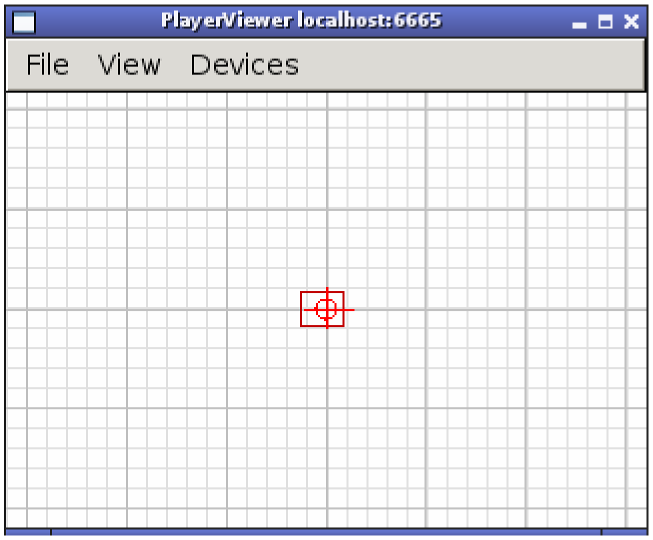
Step 5: Go to the 'Devices' menu and scroll down to 'position2d:0' and choose 'Subscribe'



A position box should now be displayed on the player viewer. Your physical robot is now connected to the virtual client-side display and it is shown to you as the centre of a grid.

Step 6: Open the 'Devices' menu again and scroll down to the 'position 2d:0' and choose "Enable"

Step 7: Open the 'Devices' menu a third time and scroll down to 'position 2d:0' and choose "Command"



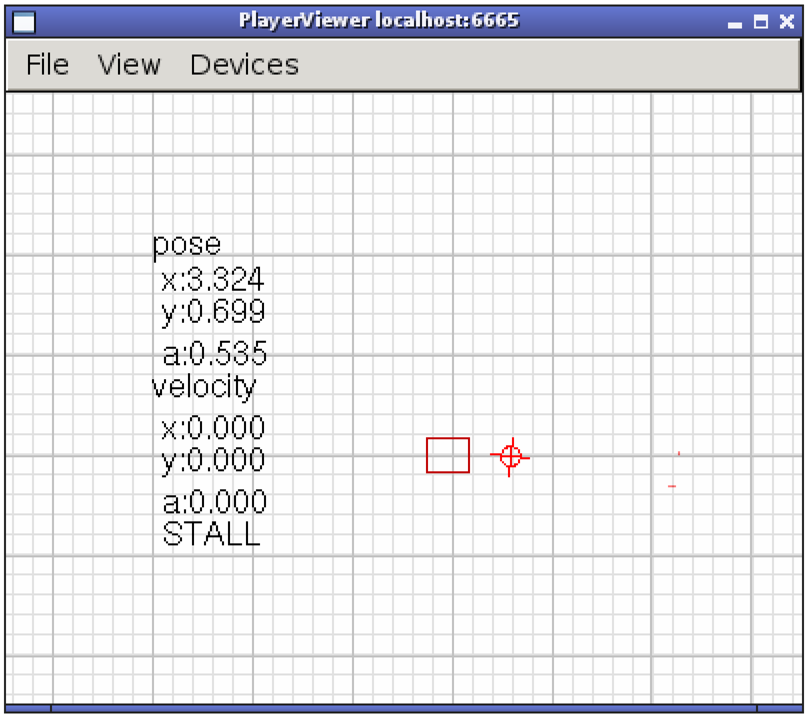
A "bulls-eye" appears in the middle of the grid. Using the left mouse button you can click and drag this bullseye to the '**right'** to move the robot '**forward'**. If you drag it to the 'left' the robot will move 'backwards'. Similarly dragging it to the 'right' and 'up' will make it turn left and 'forward' and 'down', it will turn to the right.

Note: When you release the mouse button, the robot stops and the bulls-eye returns to the box.

Step 8: Position Mode: Open the 'Devices' menu and scroll down to 'Position2d:0' and choose 'Position mode'. Again with the left mouse button click and drag the bulls-eye. The physical robot will spin and drive in a forward direction to achieve the goal position until you drop the bulls-eye. It will stop when it gets close to the goal.

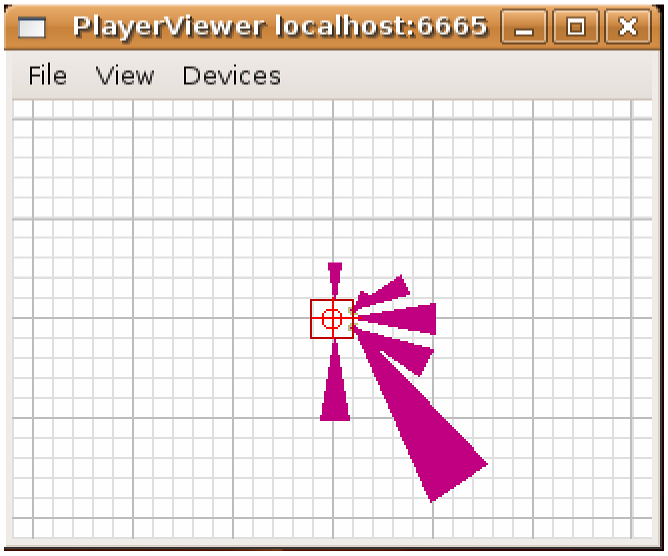
Note: As the sensors are not initiated, the physical robot will attempt to drive through any objects in its path. There is no obstacle avoidance in this kind of straightforward positioning.

Step 9: To see the current robot pose and velocity, open 'Devices' from the menu and scroll down to 'Position2d:0' and choose 'Show values'.



```
pose
 x:3.324
 y:0.699
 a:0.535
velocity
 x:0.000
 y:0.000
 a:0.000
 STALL
```

Step 10: To display the robot's sensors, again open the 'Devices' menu and scroll down to 'ir:0' and choose 'Subscribe'.



Step 11: To zoom in and out in the PlayerViewer windows, click and hold the right mouse button and drag towards or away from the circle that appears. To move the environment, click and hold the left mouse button on the grid surface and drag it.

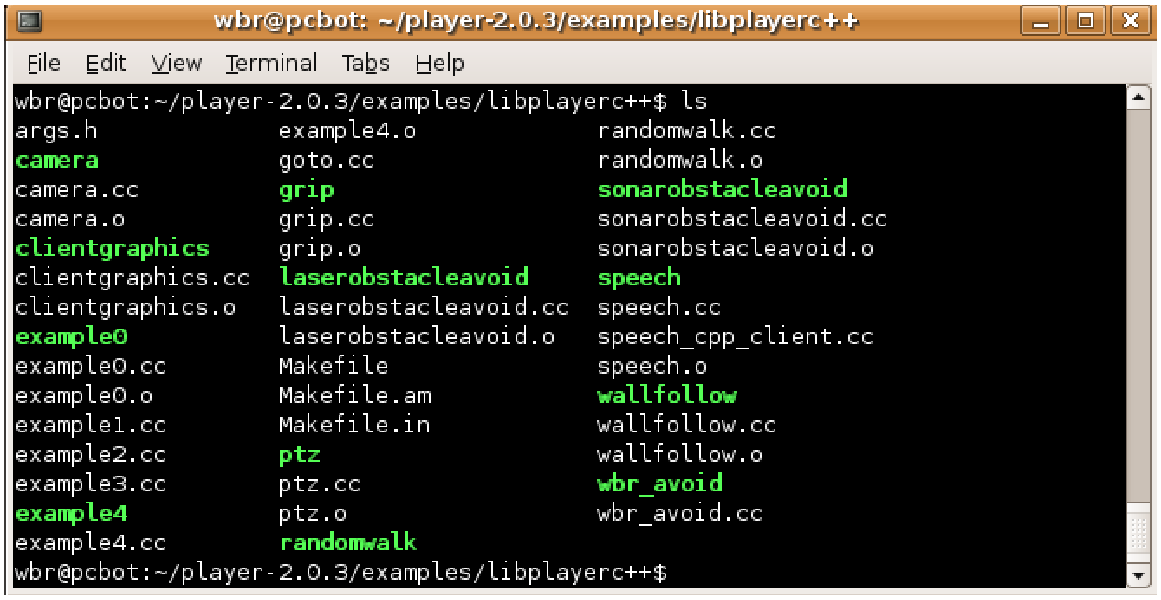
Step 12: Close the ‘playerv’ application by choosing File -> Exit or clicking the ‘x’ in the top right corner.

Controlling the PC-BOT
Using a client program (eg: wbr914 sample avoid program)

The PC-BOT can be run autonomously and controlled using C++ code that has been written and compiled using the standard Player commands.

The following is a PC-BOT IR avoid program (wbr_avoid.cc) that can be used as a basis for your own control programs.

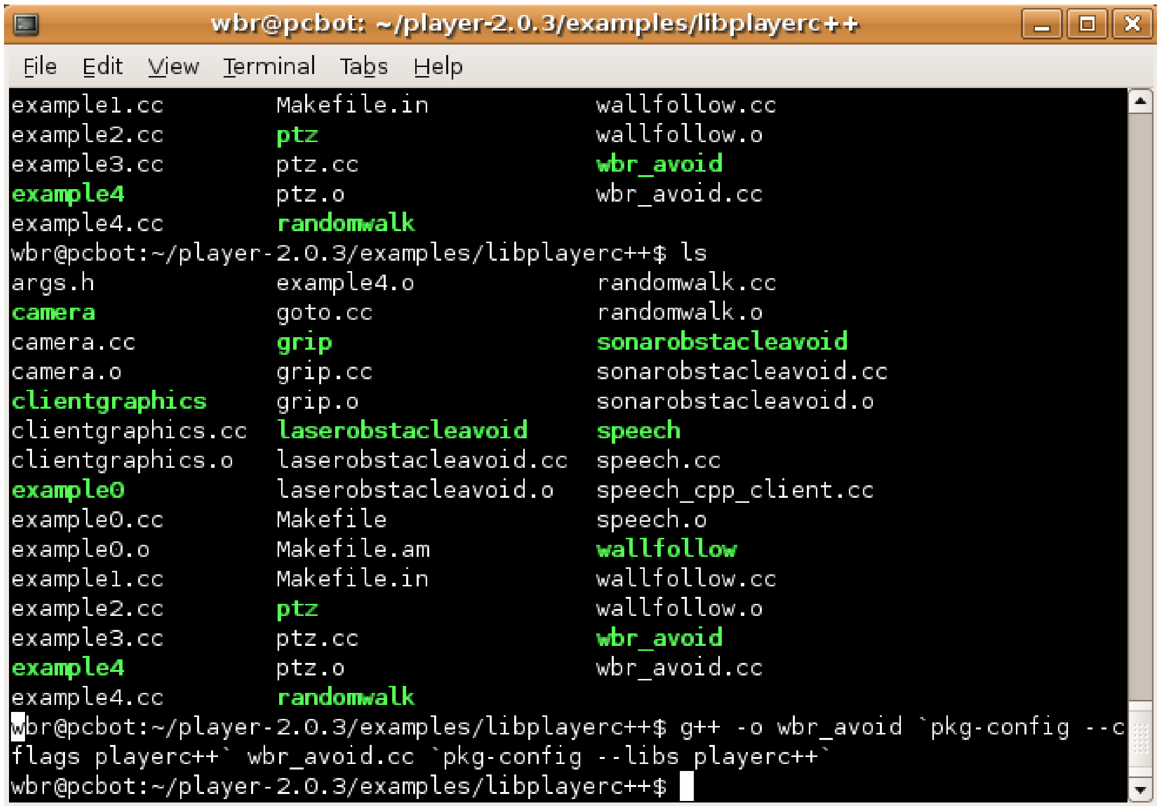
It is located under this directory: **~/player-2.0.3/examples/libplayerc++**


wbr@pcbot: ~/player-2.0.3/examples/libplayerc++
File Edit View Terminal Tabs Help

```
wbr@pcbot:~/player-2.0.3/examples/libplayerc++$ ls
args.h              example4.o              randomwalk.cc
camera              goto.cc                 randomwalk.o
camera.cc           grip                    sonarobstacleavoid
camera.o            grip.cc                 sonarobstacleavoid.cc
clientgraphics      grip.o                  sonarobstacleavoid.o
clientgraphics.cc   laserobstacleavoid      speech
clientgraphics.o    laserobstacleavoid.cc   speech.cc
example0            laserobstacleavoid.o    speech_cpp_client.cc
example0.cc         Makefile                speech.o
example0.o          Makefile.am             wallfollow
example1.cc         Makefile.in             wallfollow.cc
example2.cc         ptz                     wallfollow.o
example3.cc         ptz.cc                  wbr_avoid
example4            ptz.o                   wbr_avoid.cc
example4.cc         randomwalk
wbr@pcbot:~/player-2.0.3/examples/libplayerc++$
```

The 'wbr_avoid.cc' can be edited and re-compiled easily by typing the following command:

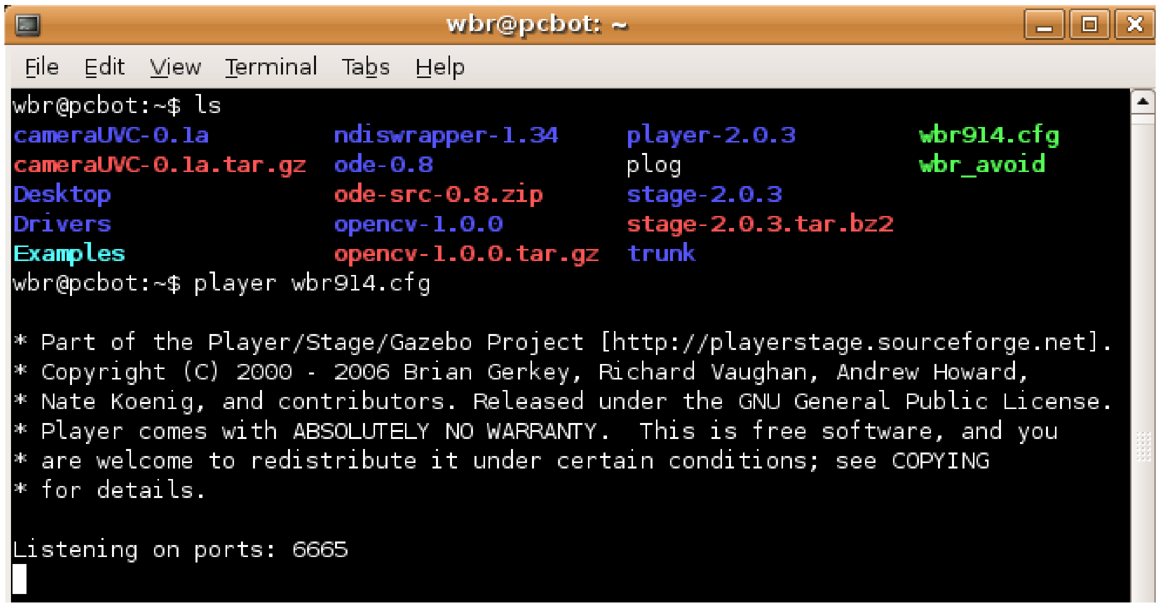
'g++ -o wbr_avoid `pkg-config –cflags playerc++` wbr_avoid.cc `pkg-config –libs playerc++`'


wbr@pcbot: ~/player-2.0.3/examples/libplayerc++
File Edit View Terminal Tabs Help

```
example1.cc          Makefile.in          wallfollow.cc
example2.cc          ptz                  wallfollow.o
example3.cc          ptz.cc               wbr_avoid
example4             ptz.o                wbr_avoid.cc
example4.cc          randomwalk
wbr@pcbot:~/player-2.0.3/examples/libplayerc++$ ls
args.h               example4.o           randomwalk.cc
camera               goto.cc              randomwalk.o
camera.cc            grip                 sonarobstacleavoid
camera.o             grip.cc              sonarobstacleavoid.cc
clientgraphics       grip.o               sonarobstacleavoid.o
clientgraphics.cc    laserobstacleavoid   speech
clientgraphics.o     laserobstacleavoid.cc speech.cc
example0             laserobstacleavoid.o speech_cpp_client.cc
example0.cc          Makefile             speech.o
example0.o           Makefile.am          wallfollow
example1.cc          Makefile.in          wallfollow.cc
example2.cc          ptz                  wallfollow.o
example3.cc          ptz.cc               wbr_avoid
example4             ptz.o                wbr_avoid.cc
example4.cc          randomwalk
wbr@pcbot:~/player-2.0.3/examples/libplayerc++$ g++ -o wbr_avoid `pkg-config --c
flags playerc++` wbr_avoid.cc `pkg-config --libs playerc++`
wbr@pcbot:~/player-2.0.3/examples/libplayerc++$
```

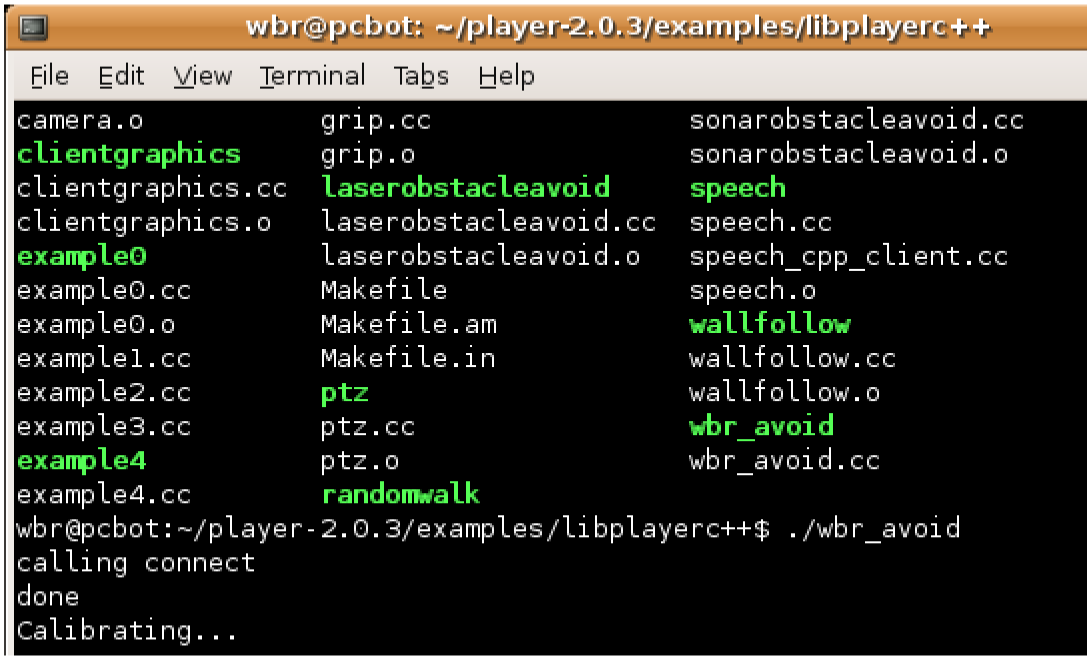
Step 13: Open another terminal window and go to the '~/player-2.0.3/examples/libplayerc++/' directory

Step 14: Verify that the Player server is still running the ~/wbr914.cfg file. If it is not running, then initiate it using that file.



```
wbr@pcbot:~$ ls
cameraUVC-0.1a          ndiswrapper-1.34     player-2.0.3         wbr914.cfg
cameraUVC-0.1a.tar.gz   ode-0.8              plog                 wbr_avoid
Desktop                 ode-src-0.8.zip      stage-2.0.3
Drivers                 opencv-1.0.0         stage-2.0.3.tar.bz2
Examples                opencv-1.0.0.tar.gz  trunk
wbr@pcbot:~$ player wbr914.cfg

* Part of the Player/Stage/Gazebo Project [http://playerstage.sourceforge.net].
* Copyright (C) 2000 - 2006 Brian Gerkey, Richard Vaughan, Andrew Howard,
* Nate Koenig, and contributors. Released under the GNU General Public License.
* Player comes with ABSOLUTELY NO WARRANTY.  This is free software, and you
* are welcome to redistribute it under certain conditions; see COPYING
* for details.

Listening on ports: 6665
```

Step 15: On the command line of the terminal window that is in the C++ examples directory, type './wbr_avoid' and press enter. This will initiate the IR avoid program that will cause the robot to drive forward and avoid both obstacles and drop-offs. **Note: The 'wbr_avoid' program can only be run directly on the robot and NOT remotely as the port number and IP address are hardcoded.**


wbr@pcbot: ~/player-2.0.3/examples/libplayerc++
File Edit View Terminal Tabs Help

```
camera.o              grip.cc                sonarobstacleavoid.cc
clientgraphics        grip.o                 sonarobstacleavoid.o
clientgraphics.cc     laserobstacleavoid     speech
clientgraphics.o      laserobstacleavoid.cc  speech.cc
example0              laserobstacleavoid.o   speech_cpp_client.cc
example0.cc           Makefile               speech.o
example0.o            Makefile.am            wallfollow
example1.cc           Makefile.in            wallfollow.cc
example2.cc           ptz                    wallfollow.o
example3.cc           ptz.cc                 wbr_avoid
example4              ptz.o                  wbr_avoid.cc
example4.cc           randomwalk
wbr@pcbot:~/player-2.0.3/examples/libplayerc++$ ./wbr_avoid
calling connect
done
Calibrating...
```

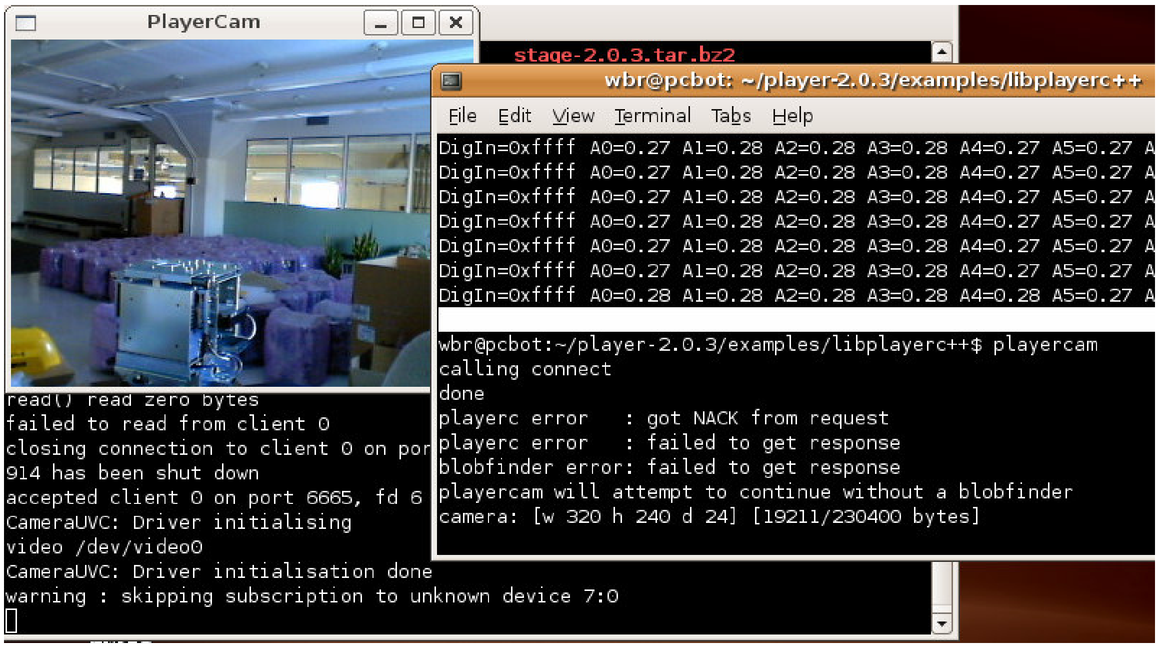
Step 16: Although they do not come as a default configuration on the 914 PC-BOT, sonar sensors and laser scanners may be added to the robot. Within the 'libplayerc++' directory there are also sample laser obstacle avoid and sonar obstacle avoid programs, which are run in the same way as the wbr914 (IR obstacle avoid) program.

Step 17: To run the control programs from a remote computer, the robot must be connected on the same network and the IP address must be known. Type the following command with the IP address in place of the <IPaddress>:
"./laserobstacleavoid –h <IPaddress> -p 6665"

Viewing the video on the web camera using ‘playercam’

Step 18: Make sure the Player server is running using the ‘~/wbr914.cfg’ file.

Step 19: On the command line of the terminal program type ‘playercam’.


PlayerCam
stage-2.0.3.tar.bz2
wbr@pcbot: ~/player-2.0.3/examples/libplayerc++
File Edit View Terminal Tabs Help

```
DigIn=0xffff A0=0.27 A1=0.28 A2=0.28 A3=0.28 A4=0.27 A5=0.27 A
DigIn=0xffff A0=0.27 A1=0.28 A2=0.28 A3=0.28 A4=0.27 A5=0.27 A
DigIn=0xffff A0=0.27 A1=0.28 A2=0.28 A3=0.28 A4=0.27 A5=0.27 A
DigIn=0xffff A0=0.27 A1=0.28 A2=0.28 A3=0.28 A4=0.27 A5=0.27 A
DigIn=0xffff A0=0.27 A1=0.28 A2=0.28 A3=0.28 A4=0.27 A5=0.27 A
DigIn=0xffff A0=0.27 A1=0.28 A2=0.28 A3=0.28 A4=0.27 A5=0.27 A
DigIn=0xffff A0=0.28 A1=0.28 A2=0.28 A3=0.28 A4=0.28 A5=0.27 A

wbr@pcbot:~/player-2.0.3/examples/libplayerc++$ playercam
calling connect
done
playerc error   : got NACK from request
playerc error   : failed to get response
blobfinder error: failed to get response
playercam will attempt to continue without a blobfinder
camera: [w 320 h 240 d 24] [19211/230400 bytes]
```

```
read() read zero bytes
failed to read from client 0
closing connection to client 0 on por
914 has been shut down
accepted client 0 on port 6665, fd 6
CameraUVC: Driver initialising
video /dev/video0
CameraUVC: Driver initialisation done
warning : skipping subscription to unknown device 7:0
```

A video window opens and displays the camera images. You can see in the above picture how the playercam command connects to Player. You will receive some errors in the playercam call as the blobfinder is not being used.

STAGE - Simulated Robot Environment

Stage simulates a population of mobile robots, with operational sensors and objects in a two-dimensional bitmapped environment.

There are a number of files required to create a simulated environment:

.cfg files – (see previous Player chapter) the player configuration file defines what Stage will simulate and what drivers will be called. It also refers to the '.world' file that will be used for the models and maps.

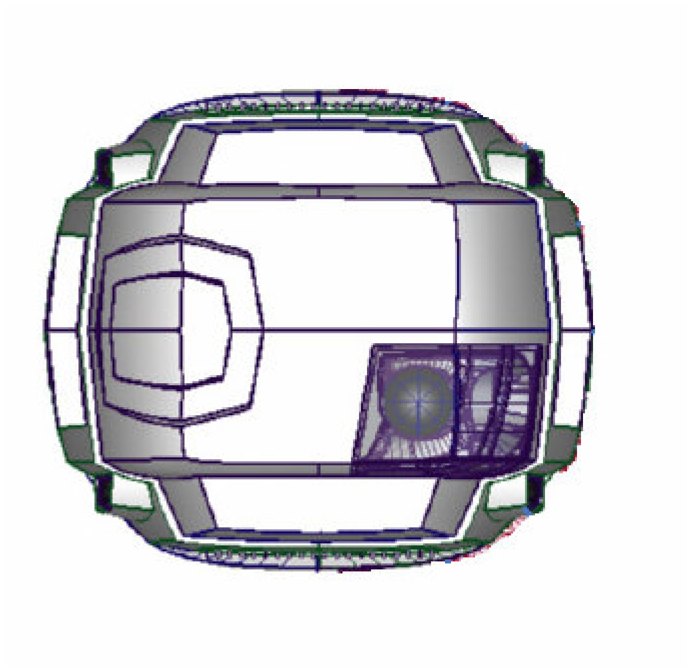
.world files – lists the map that will be used, the robot/sensor models (devices) that will be used as well as initial locations of each model.

.inc files – Device definitions for the robots, sensors, maps, etc. This is where you define how the objects look.

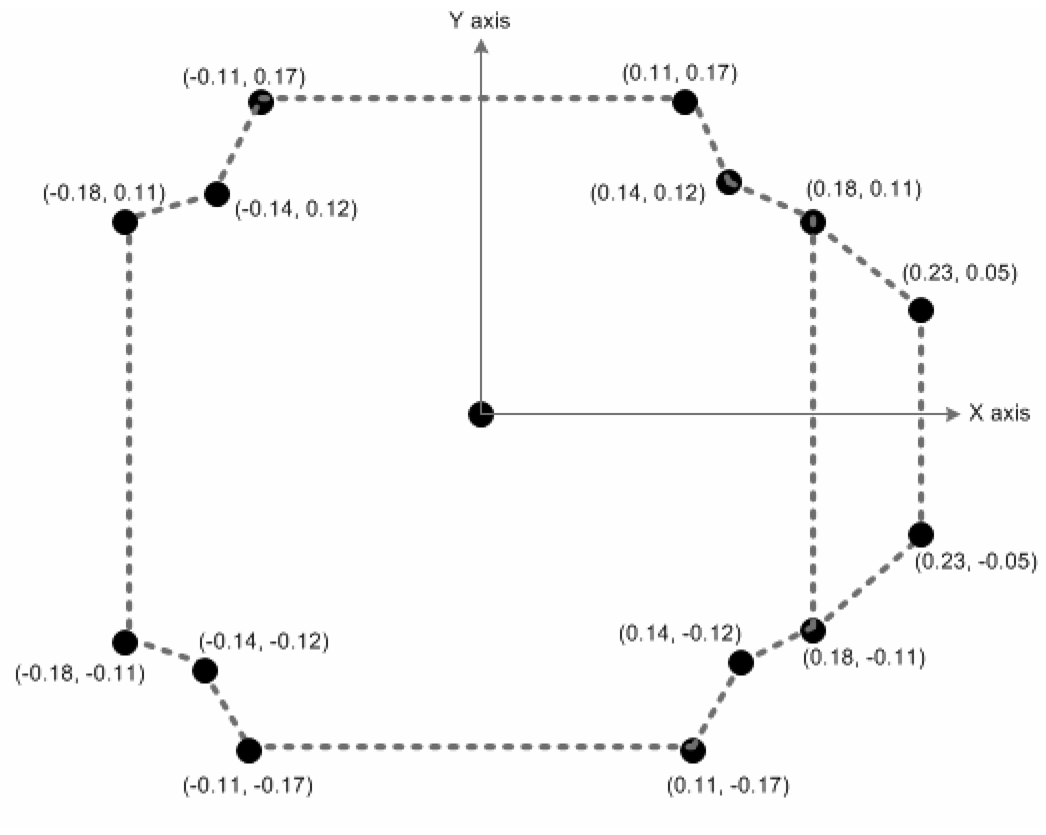
Examples of these files can be found in this directory: **~/stage-2.0.3/worlds**

A number of sample files have been created for the 914 PC-BOT and can be found in the above directory. Reading through these and seeing what the result is in Stage is the optimum way to understand how to change/create them to fit with your own project work.

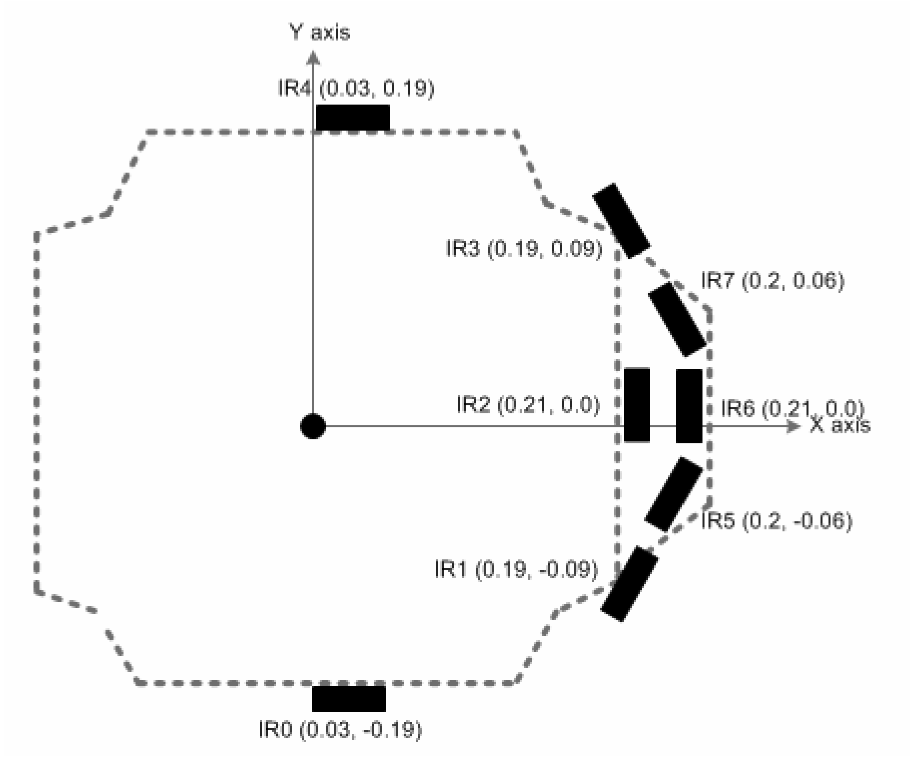
Begin with the ‘**wbr914.inc’** file. This file models the robot and places the sensors on it with the proper ranges and pose. The model is based on the following images.



<u>Coordinates for the model polygons</u>



Coordinates for the IR sensors



'wbr914.inc' file code

```
# Desc: Device definitions for Whitebox Robotics PC-BOT 914 robot.
# Author: Neil Schubert
# Date: 10 Mar 2007

# The PC-BOT IR array
define wbr914_ir ranger
(

  # sensor count used in this ranger
  scount 8

  # define the pose of each (sensor) transducer [xpos ypos heading]
  spose[4] [ 0.03 0.19 90 ]
  spose[3] [ 0.19 0.09 25 ]
  spose[2] [ 0.21 0.00 0 ]
  spose[1] [ 0.19 -0.09 -25 ]
  spose[0] [ 0.03 -0.19 -90 ]

  # these 3 transducers are the downward pointing for stairs and drops
  # because they point down, they have little effect in the Stage 2D
  # environment

  spose[7] [ 0.20 0.06 56 ]
  spose[6] [ 0.21 0.00 0 ]
  spose[5] [ 0.20 -0.06 -56 ]


  # define the field of view of each transducer [range_min range_max view_angle]
  sview [0.1 0.8 10]

  # the view of the downward facing ones are set at the ground hit
  # range even though they do extend farther.

  sview[5] [0.2 0.68 10]
  sview[6] [0.2 0.46 10]
  sview[7] [0.2 0.68 10]

  # define the size of each transducer [xsize ysize] in meters
  ssize [0.01 0.03]
)

# a PC-BOT 914 in standard configuration

define wbr914 position
(

  # actual size of robot in meters for scaling

  size [0.45 0.35]

  # the PC-BOT's center of rotation is offset from its center of area

  origin [0 0.0 0]

  # draw a nose on the robot so we can see which way it points

  gui_nose 1

  # estimated mass in KG

  mass 25.0

  # this polygon approximates the shape of a PC-BOT 914
  # Use two polygons to draw the robot

  polygons 2

  # details of the first polygon
  # polygon[index].points (total number of polygon points)

  polygon[0].points 12

  # polygon[index].point[index] [ xpos ypos ]

  polygon[0].point[0] [  0.11  0.17 ]
  polygon[0].point[1] [  0.13  0.12 ]
  polygon[0].point[2] [  0.18  0.11 ]
  polygon[0].point[3] [  0.18 -0.11 ]
  polygon[0].point[4] [  0.13 -0.12 ]
  polygon[0].point[5] [  0.11 -0.17 ]
  polygon[0].point[6] [ -0.11 -0.17 ]
  polygon[0].point[7] [ -0.14 -0.12 ]
  polygon[0].point[8] [ -0.18 -0.11 ]
  polygon[0].point[9] [ -0.18  0.11 ]
  polygon[0].point[10] [ -0.14  0.12 ]
  polygon[0].point[11] [ -0.11 0.17 ]
  # details of the second polygon

  polygon[1].points 4
  polygon[1].point[0] [ 0.18  0.08 ]
  polygon[1].point[1] [ 0.23  0.05 ]
  polygon[1].point[2] [ 0.23 -0.05 ]
  polygon[1].point[3] [ 0.18 -0.08 ]


  # differential steering model

  drive "diff"

  # laser_return refers to making the robot detectable by laser sensors
  # if the robot's sensor was mounted on the head, it would not detect     other
  # similar robots because the beam would go over the robot. This adds more
  # reality to the simulation (and worse detection) if you uncomment the line

  # laser_return 0

  # use the IR array defined above

  wbr914_ir()
)
```

Once you have the '.inc files' you include them in the world model that sets the 'stage' for the simulator that will be called later in the '.cfg file'. This file ties together the models (robots/sensors/objects) and the map for the world (.png file that can be created by the user, examples are found under the 'bitmaps' directory) surrounding your robot. Essentially replicates the physical world into the Stage simulated world.

<u>'wbr914sim.world' file code</u>

```
# Desc: PC-BOT Stage demo with lots of models

# the size of a pixel in Stage's underlying raytrace model in meters
resolution     0.02

interval_sim 100  # milliseconds per update step
interval_real 100 # real-time milliseconds per update step

# defines Pioneer-like robots
include "pioneer.inc"

# defines PC-BOT robot
include "wbr914.inc"

# defines 'map' object used for floorplans
include "map.inc"

# defines the laser models `sick_laser' configured like a Sick LMS-200
# and defines Hokuyo URG Laser
include "laser.inc"

#defines the size of the world
size [40 20 ]

gui_disable 0
gui_interval 100
gui_menu_interval 20

window(
  size [ 755.000 684.000 ]
#  size [ 500 500 ]
  center [-7.707 2.553]
  scale 0.009
)

map(
  bitmap "bitmaps/hospital_section.png"
  map_resolution 0.02
  size [40 18]
  name "hospital"
)


# a block for gripping
define puck model(
  size [ 0.08 0.08 ]
  gripper_return 1
  gui_movemask 3
  gui_nose 0
  fiducial_return 10
```

```
)


puck( pose [-9.114 0.467 -105.501 ] color "red" )
puck( pose [-9.045 0.624 -37.717 ] color "purple" )
puck( pose [-8.959 0.752 -461.643 ] color "orange" )


# extend the pioneer2dx definition from pioneer.inc
#
define loaded_pioneer pioneer2dx
(
  sick_laser
  (
    pose [0.03 0.000 0.000 ]
    fiducialfinder( range_max 8 range_max_id 5 )

    ptz(
      blobfinder(
        channel_count 6
        channels [ "red" "blue" "green" "cyan" "yellow" "magenta" ]
                )
       )
  )

  fiducial_return 1
  gripper_return 0

  localization "gps"
  localization_origin [ 0 0 0 ]
)

loaded_pioneer
(
  name "pioneer1"
  pose [-10.071 0.186 -722.333]

  gripper( pose [0.200 0.000 0.000] color "gray" )
  speech()
)

loaded_pioneer
(
  color "red"
  name "pioneer2"
  pose [-5.645 3.034 -162.098]
)

define loaded_wbr914 wbr914
(
  hokuyo_URG_laser
  (
   pose [0.0 0.0 0.0 ]
  )

  fiducial_return 1
  gripper_return 0

  localization "gps"
  localization_origin [ 0 0 0 ]
)

loaded_wbr914
(
  color "white"
  name "wbr914_1"
  pose [-6.009 2.481 -194.220]
)

loaded_wbr914
(
  color "yellow"
  name "wbr914_2"
  pose [-6.492 2.156 -199.781]
)
```

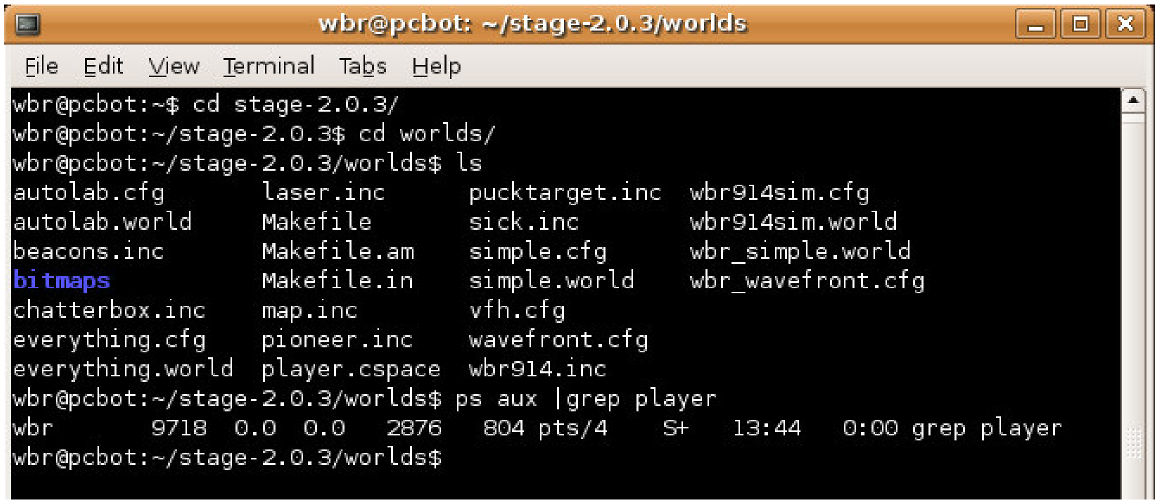
Finally the .world file is linked into the simulator STAGE and the various sensor readings, maps, etc. are created and can be accessed with many types of client programs (playerv, playerjoy, playernav, avoidance programs, etc.)

‘wbr914sim.cfg’ file code

```
# Desc: PC-Bot Player sample configuration file for controlling Stage
# devices
# Date: 18 Mar 2007

driver
(
  name "stage"
  provides ["simulation:0"]
  plugin "libstageplugin"

  # world file has links to .inc files and defines for all the models and
  # maps

  worldfile "wbr914sim.world"
)

driver( name "stage" provides ["map:0" ] model "hospital" )

# 1st robot
driver(
 name "stage"
 provides [ "position2d:0" "sonar:0" "laser:0" "ptz:0" "blobfinder:0" "fiducial:0"
"gripper:0" "graphics2d:0" "speech:0"]
 model "pioneer1"
)

driver( name "vfh" requires [ "position2d:0" "laser:0" ] provides [ "position2d:1"
] )



# 2nd robot
driver(
 name "stage"
 provides ["6666:position2d:0" "6666:sonar:0" "6666:laser:0" "6666:blobfinder:0"
"6666:fiducial:0"]
 model "pioneer2"
)

# 3rd robot
driver(
 name "stage"
 provides ["6667:position2d:0" "6667:sonar:0" "6667:laser:0"]
 model "wbr914_1"
)

# 4th robot
driver(
 name "stage"
 provides ["6668:position2d:0" "6668:sonar:0" "6668:laser:0"]
 model "wbr914_2"
)
```

All these files are combined to create a virtual world inside the STAGE simulator.

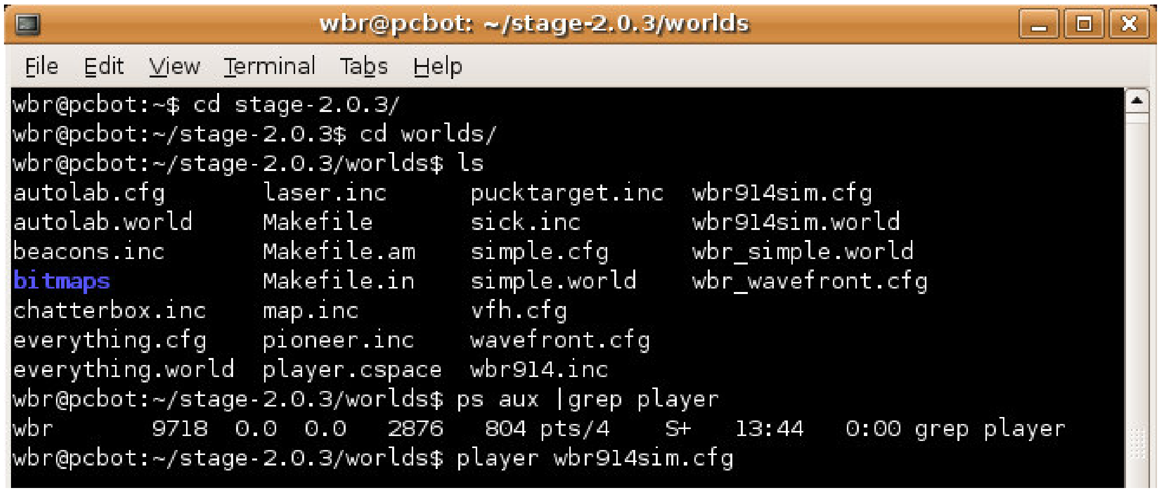
Step 20: Open a terminal windows and go to the '~/stage-2.0.3/worlds' directory. Make sure the player server is not currently running by typing the command 'ps aux lgrep player'.


wbr@pcbot: ~/stage-2.0.3/worlds
File Edit View Terminal Tabs Help

```
wbr@pcbot:~$ cd stage-2.0.3/
wbr@pcbot:~/stage-2.0.3$ cd worlds/
wbr@pcbot:~/stage-2.0.3/worlds$ ls
autolab.cfg      laser.inc      pucktarget.inc  wbr914sim.cfg
autolab.world    Makefile       sick.inc        wbr914sim.world
beacons.inc      Makefile.am    simple.cfg      wbr_simple.world
bitmaps          Makefile.in    simple.world    wbr_wavefront.cfg
chatterbox.inc   map.inc        vfh.cfg
everything.cfg   pioneer.inc    wavefront.cfg
everything.world player.cspace  wbr914.inc
wbr@pcbot:~/stage-2.0.3/worlds$ ps aux |grep player
wbr       9718  0.0  0.0   2876   804 pts/4    S+   13:44   0:00 grep player
wbr@pcbot:~/stage-2.0.3/worlds$
```

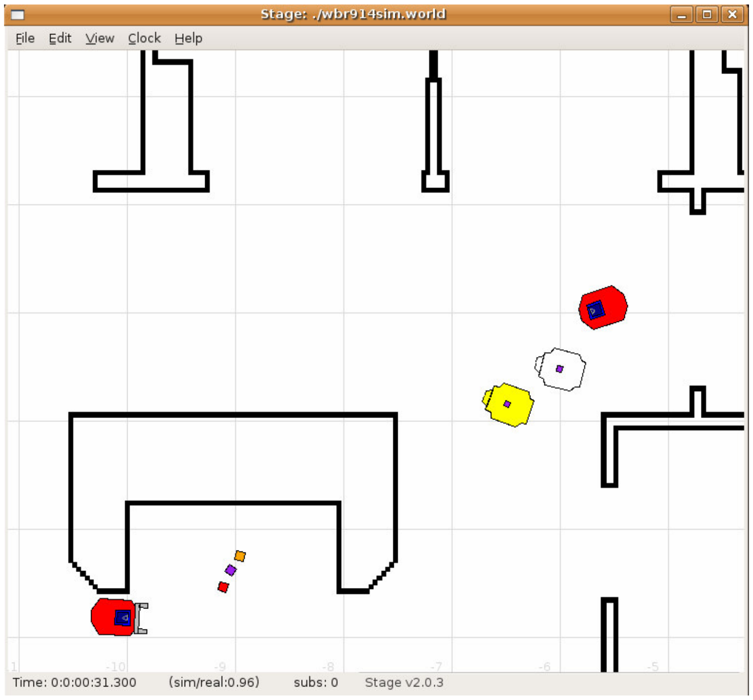
The only thing listed is the grep command, so player server is NOT running.

Step 21: Now type the command **'player wbr914sim.cfg'**.

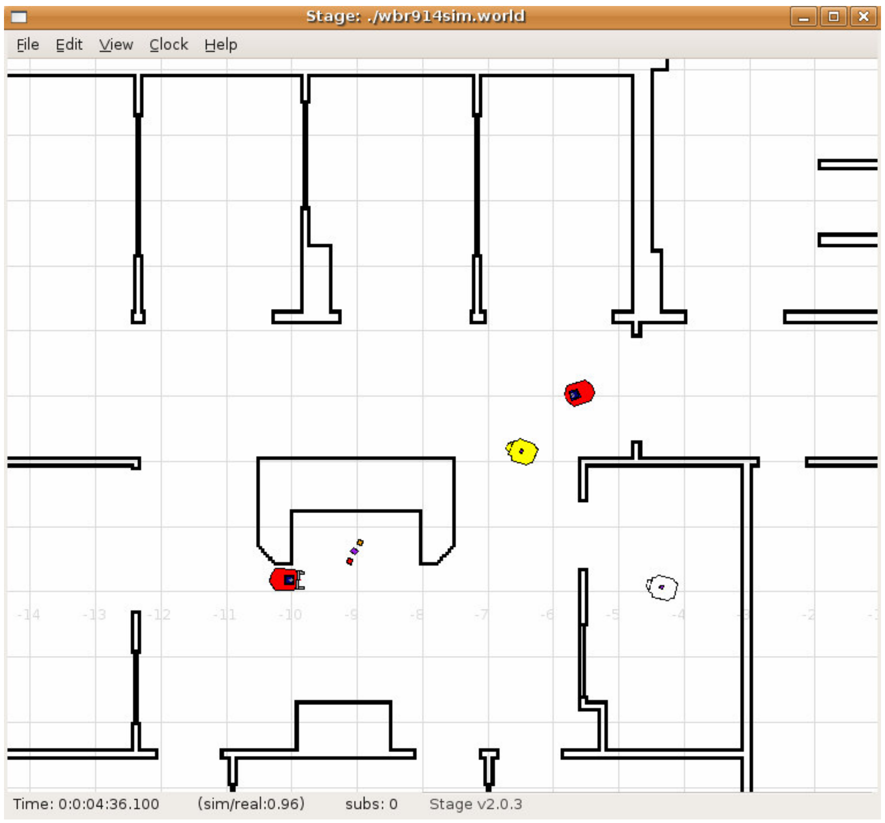

wbr@pcbot: ~/stage-2.0.3/worlds
File Edit View Terminal Tabs Help

```
wbr@pcbot:~$ cd stage-2.0.3/
wbr@pcbot:~/stage-2.0.3$ cd worlds/
wbr@pcbot:~/stage-2.0.3/worlds$ ls
autolab.cfg      laser.inc      pucktarget.inc  wbr914sim.cfg
autolab.world    Makefile       sick.inc        wbr914sim.world
beacons.inc      Makefile.am    simple.cfg      wbr_simple.world
bitmaps          Makefile.in    simple.world    wbr_wavefront.cfg
chatterbox.inc   map.inc        vfh.cfg
everything.cfg   pioneer.inc    wavefront.cfg
everything.world player.cspace  wbr914.inc
wbr@pcbot:~/stage-2.0.3/worlds$ ps aux |grep player
wbr       9718  0.0  0.0   2876   804 pts/4    S+   13:44   0:00 grep player
wbr@pcbot:~/stage-2.0.3/worlds$ player wbr914sim.cfg
```

The following Stage window, showing the virtual world, opens:


Stage: ./wbr914sim.world
File Edit View Clock Help
Time: 0:0:00:31.300 (sim/real:0.96) subs: 0 Stage v2.0.3

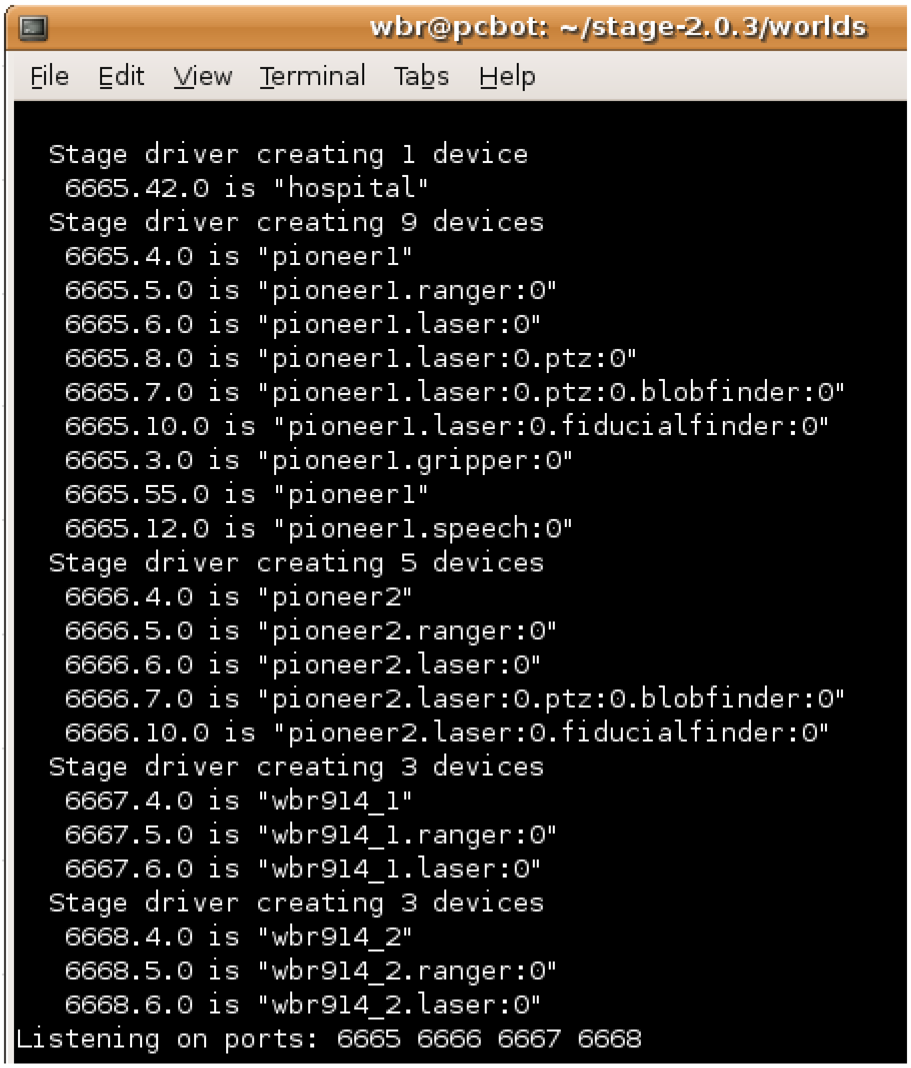
The virtual robots can be moved in this window by ‘left’ clicking on them and dragging them. You can zoom in and out by right clicking on the map and dragging towards or away from the center of the circle that is created.


Stage: ./wbr914sim.world
File Edit View Clock Help
Time: 0:0:04:36.100 (sim/real:0.96) subs: 0 Stage v2.0.3

This Stage world was created with 4 robots each under a different port number.

6665 – Red Pioneer with a gripper, ptz, blobfinder and Sick Laser
6666 – Red Pioneer with a Sick Laser, ptz and blobfinder
6667 – White 914 PC-BOT robot with Hokuyo Laser
6668 – Yellow 914 PC-BOT robot with Hokuyo Laser

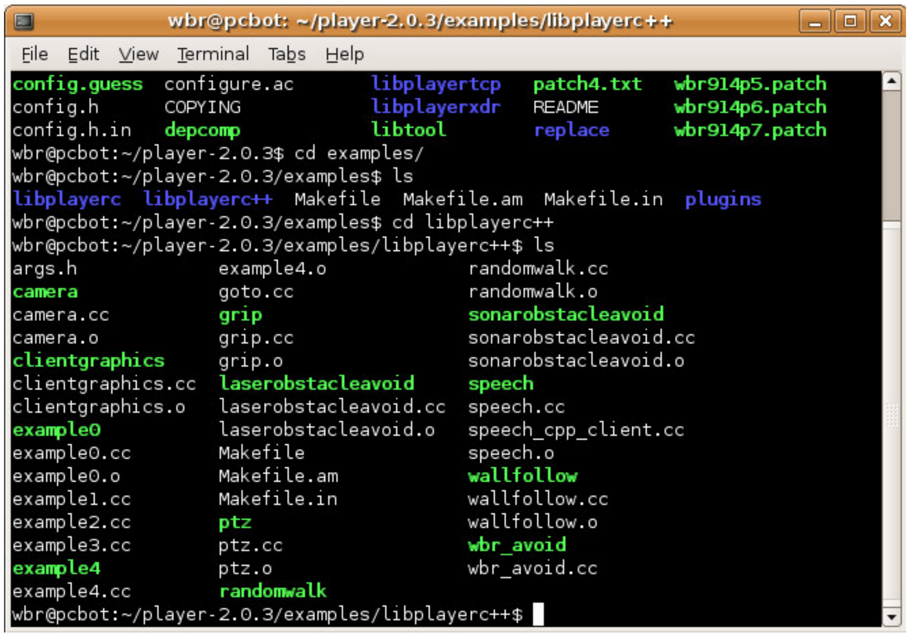
You can see this in the 'player server' window:


wbr@pcbot: ~/stage-2.0.3/worlds
File Edit View Terminal Tabs Help

```
  Stage driver creating 1 device
   6665.42.0 is "hospital"
  Stage driver creating 9 devices
   6665.4.0 is "pioneer1"
   6665.5.0 is "pioneer1.ranger:0"
   6665.6.0 is "pioneer1.laser:0"
   6665.8.0 is "pioneer1.laser:0.ptz:0"
   6665.7.0 is "pioneer1.laser:0.ptz:0.blobfinder:0"
   6665.10.0 is "pioneer1.laser:0.fiducialfinder:0"
   6665.3.0 is "pioneer1.gripper:0"
   6665.55.0 is "pioneer1"
   6665.12.0 is "pioneer1.speech:0"
  Stage driver creating 5 devices
   6666.4.0 is "pioneer2"
   6666.5.0 is "pioneer2.ranger:0"
   6666.6.0 is "pioneer2.laser:0"
   6666.7.0 is "pioneer2.laser:0.ptz:0.blobfinder:0"
   6666.10.0 is "pioneer2.laser:0.fiducialfinder:0"
  Stage driver creating 3 devices
   6667.4.0 is "wbr914_1"
   6667.5.0 is "wbr914_1.ranger:0"
   6667.6.0 is "wbr914_1.laser:0"
  Stage driver creating 3 devices
   6668.4.0 is "wbr914_2"
   6668.5.0 is "wbr914_2.ranger:0"
   6668.6.0 is "wbr914_2.laser:0"
Listening on ports: 6665 6666 6667 6668
```

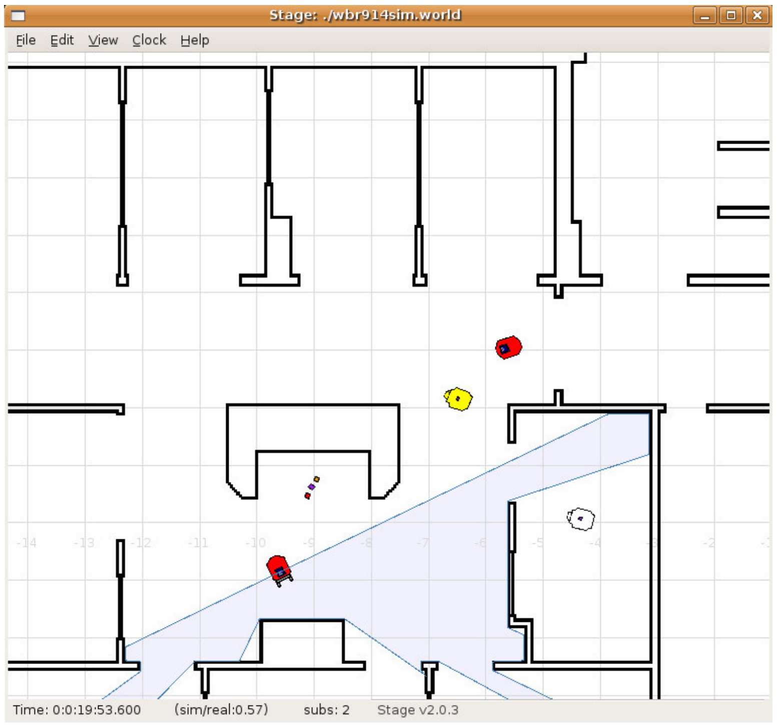
Each one of those robots can be individually controlled using any of the client programs (playerv, playerjoy, laserobstacle avoidance, etc.).

Step 22: Open a terminal window and go to the '~/player-2.0.3/examples/libplayerc++' Where the example obstacle avoidance programs are.


wbr@pcbot: ~/player-2.0.3/examples/libplayerc++
File Edit View Terminal Tabs Help

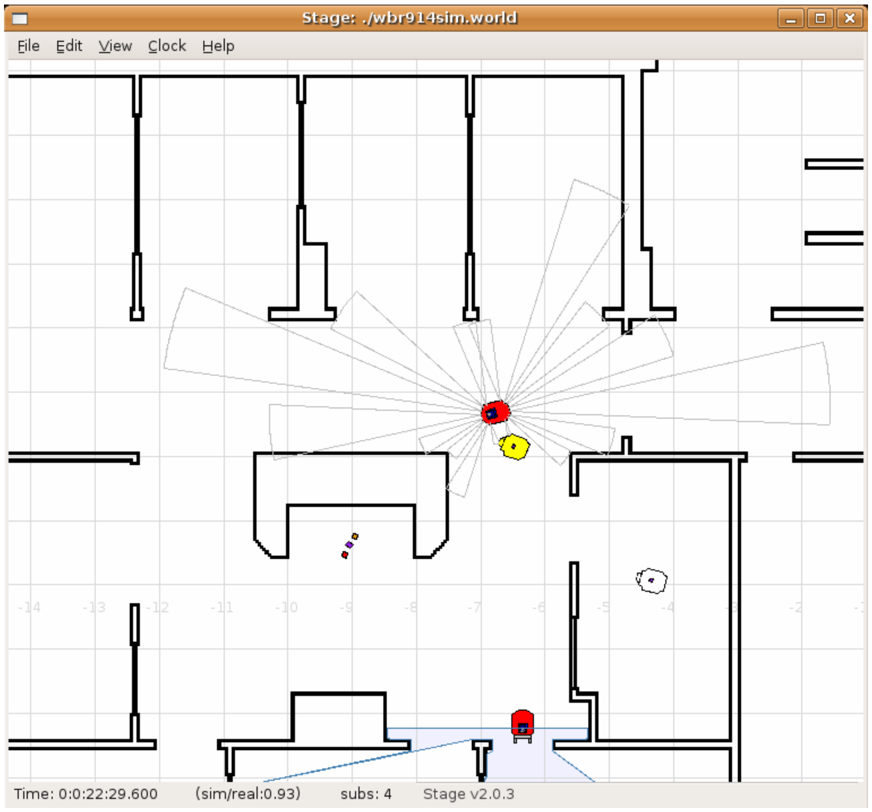
```
config.guess  configure.ac     libplayertcp   patch4.txt    wbr914p5.patch
config.h      COPYING          libplayerxdr   README        wbr914p6.patch
config.h.in   depcomp          libtool        replace       wbr914p7.patch
wbr@pcbot:~/player-2.0.3$ cd examples/
wbr@pcbot:~/player-2.0.3/examples$ ls
libplayerc  libplayerc++  Makefile  Makefile.am  Makefile.in  plugins
wbr@pcbot:~/player-2.0.3/examples$ cd libplayerc++
wbr@pcbot:~/player-2.0.3/examples/libplayerc++$ ls
args.h             example4.o             randomwalk.cc
camera             goto.cc                randomwalk.o
camera.cc          grip                   sonarobstacleavoid
camera.o           grip.cc                sonarobstacleavoid.cc
clientgraphics     grip.o                 sonarobstacleavoid.o
clientgraphics.cc  laserobstacleavoid     speech
clientgraphics.o   laserobstacleavoid.cc  speech.cc
example0           laserobstacleavoid.o   speech_cpp_client.cc
example0.cc        Makefile               speech.o
example0.o         Makefile.am            wallfollow
example1.cc        Makefile.in            wallfollow.cc
example2.cc        ptz                    wallfollow.o
example3.cc        ptz.cc                 wbr_avoid
example4           ptz.o                  wbr_avoid.cc
example4.cc        randomwalk
wbr@pcbot:~/player-2.0.3/examples/libplayerc++$
```

Step 23: Type './laserobstacleavoid –p 6665' this will cause the laser obstacle avoidance program to run on the first pioneer robot.

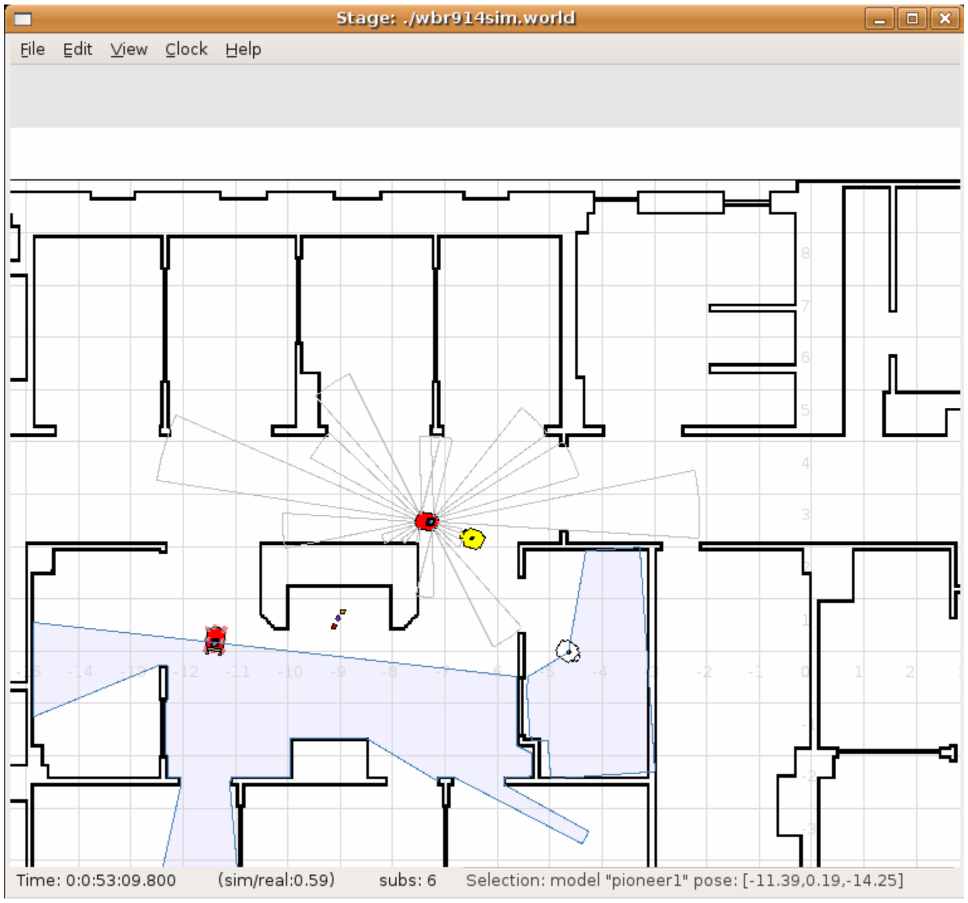

Stage: ./wbr914sim.world
File Edit View Clock Help
Time: 0:0:19:53.600 (sim/real:0.57) subs: 2 Stage v2.0.3

The robot begins to move and displays the scan of the SICK Laser scanner.

Step 24: Open another terminal window and go to the same directory as before. Type './sonarobstacleavoid –p 6666'. This will cause the sonar obstacle avoidance program to control the second pioneer robot.

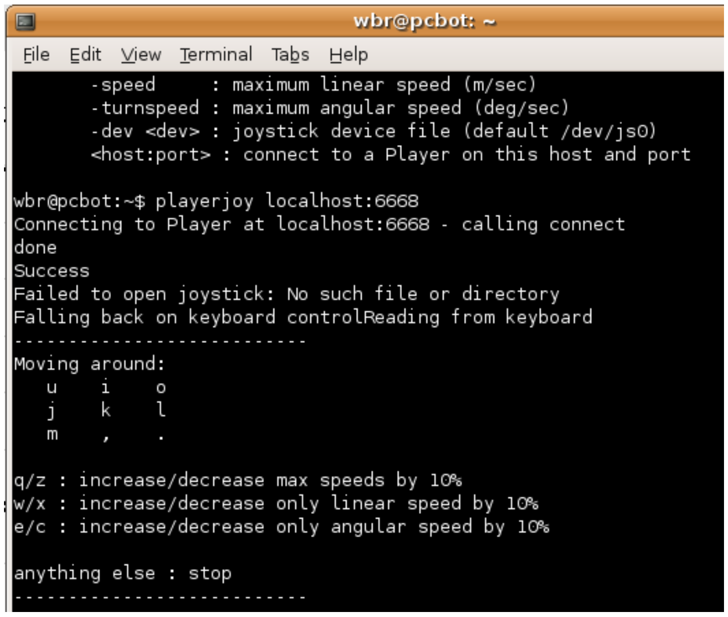

Stage: ./wbr914sim.world
File Edit View Clock Help
Time: 0:0:22:29.600 (sim/real:0.93) subs: 4 Stage v2.0.3

Step 25: Open another terminal window and go to the same directory as before. Type './laserobstacleavoid –p 6667'. This will cause the laser avoidance program to run on the first 914 PC-BOT using the Hokuyo Laser instead of the Sick laser. **(Note: wbr_avoid program works only on the real robot and not on the Stage simulator. This is due to the IR proxy call, which is not currently supported in Stage)**

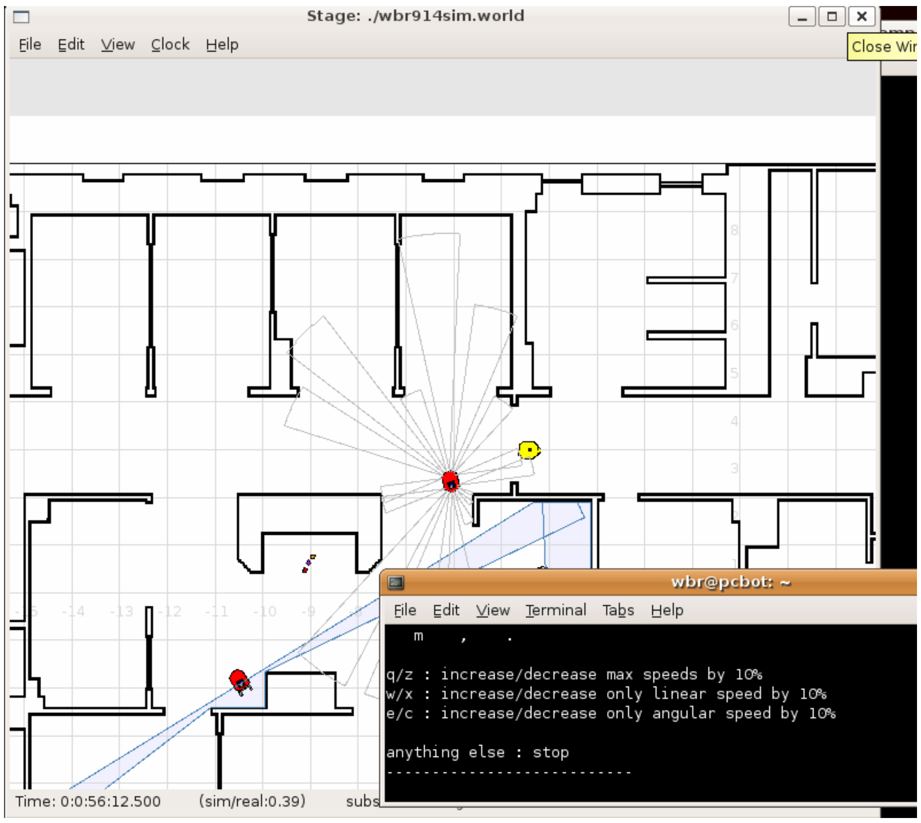


Time: 0:0:53:09.800 (sim/real:0.59) subs: 6 Selection: model "pioneer1" pose: [-11.39,0.19,-14.25]

Step 26: For the last robot, open a terminal window and type the following command 'playerjoy localhost:6668'. This will insert the use of a joystick control on the robot. The controller defaults to using the keyboard to control the robot in the absence of an identified joystick. The terminal window outputs the usage keys.



```
       -speed     : maximum linear speed (m/sec)
       -turnspeed : maximum angular speed (deg/sec)
       -dev <dev> : joystick device file (default /dev/js0)
       <host:port> : connect to a Player on this host and port

wbr@pcbot:~$ playerjoy localhost:6668
Connecting to Player at localhost:6668 - calling connect
done
Success
Failed to open joystick: No such file or directory
Falling back on keyboard controlReading from keyboard
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%

anything else : stop
---------------------------
```



```
   m    ,    .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%

anything else : stop
---------------------------
```

Close all the client programs and the player server. Use **CTRL-C** for the programs.

‘Playernav’ utility

There are some more advanced functions/drivers in player. A number of drivers are linked together and can be used in the playernav utility.

Description:
‘playernav’ is a GUI client that provides control over localize and planner devices. This utility allows you to set your robots' localization hypotheses by dragging and dropping them in the map. You can set global goals the same way, and see the planned paths and the robots' progress toward those goals.

Some example files have been created using the 914 PC-BOT in Stage which has a laser scanner (which is required for navigation and path planning).

‘wbr_wavefront.cfg’ code file

```
# Desc: PC-BOT Wavefront/Playernav Stage demo using the 'simple.world'

driver
(
  name "stage"
  plugin "libstageplugin"
  provides ["6665:simulation:0"]
  worldfile "wbr_simple.world"
)

driver
(
  name "stage"
  provides ["6665:map:0"]
  model "cave"
)

driver
(
  name "stage"
  provides ["6665:position2d:0" "6665:laser:0"]
  model "wbr914_1"
)

driver
(
  name "vfh"
  provides ["6665:position2d:1"]
  requires ["6665:position2d:0" "6665:laser:0"]
  distance_epsilon 0.3
  angle_epsilon 5
)

driver
(
  name "amcl"
  provides ["6665:localize:0" "6665:position2d:2"]
  requires ["odometry::6665:position2d:0" "6665:laser:0" "laser::6665:map:0"]
)
```

```
driver
(
  name "wavefront"
  provides ["6665:planner:0"]
  requires ["output::6665:position2d:1" "input::6665:position2d:2" "6665:map:0"]
  safety_dist 0.15
  distance_epsilon 0.5
  angle_epsilon 10
  alwayson 0
)
```

You can see from this file above that three (3) drivers are used in order to create a robot that can navigate waypoints and reach an end goal.

VFH
http://playerstage.sourceforge.net/doc/Player-2.0.0/player/group__driver__vfh.html
The vfh driver implements the Vector Field Histogram Plus local navigation method by Ulrich and Borenstein. VFH+ provides real-time obstacle avoidance and path following capabilities for mobile robots. Layered on top of a laser-equipped robot, vfh works great as a local navigation system

AMCL
http://playerstage.sourceforge.net/doc/Player-2.0.0/player/group__driver__amcl.html
The amcl driver implements the Adaptive Monte-Carlo Localization algorithm described by Dieter Fox. At the conceptual level, the amcl driver maintains a probability distribution over the set of all possible robot poses, and updates this distribution using data from odometry, sonar and/or laser range-finders. The driver also requires a pre-defined map of the environment against which to compare observed sensor values.

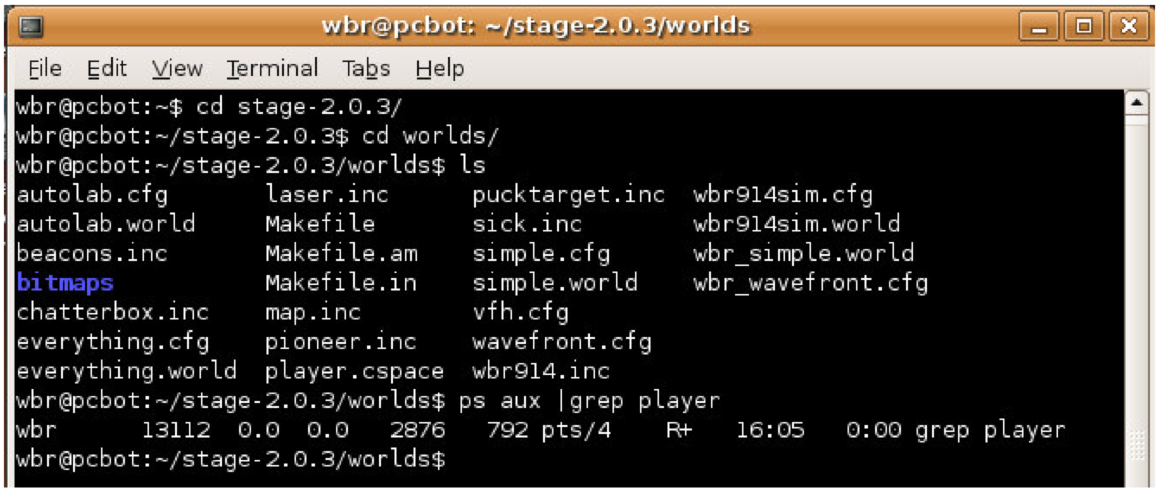
Wavefront
http://playerstage.sourceforge.net/doc/Player-2.0.0/player/group__driver__wavefront.html
The wavefront driver implements a global path planner for a planar mobile robot.

Along with the wbr_wavefront.cfg file you need the linked world file which specifies which map and which robot will be used.

‘wbr_simple.world’ file code

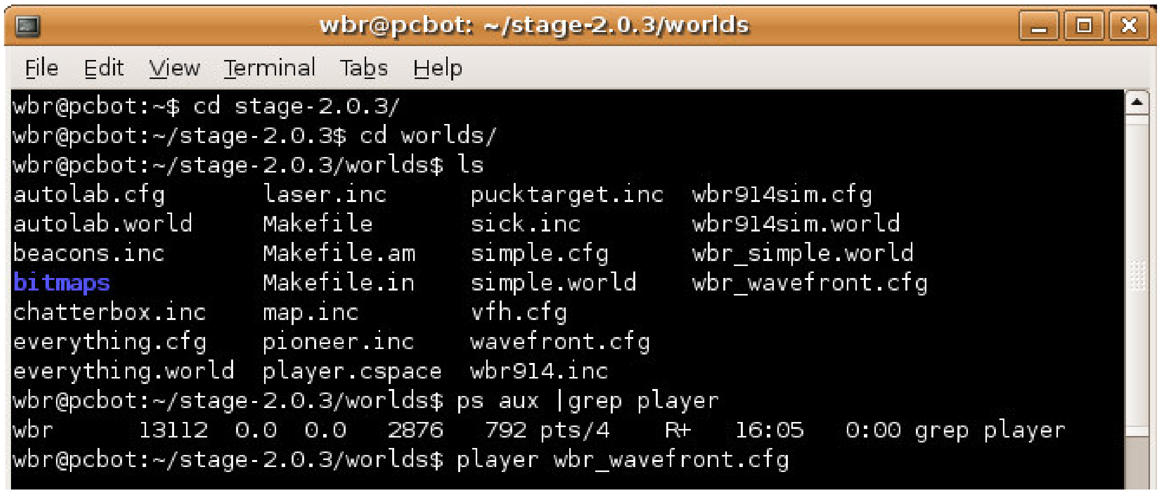
```
# Desc: 1 PC-BOT robot with laser

include "pioneer.inc"

#defines Whitebox Robotics PC-BOT 914 robot
include "wbr914.inc"

#defines laser scanner sensors
include "laser.inc"

# defines 'map' object used for floorplans
include "map.inc"

# size of the world in meters
size [16 16]

# set the resolution of the underlying raytrace model in meters
resolution 0.02

# update the screen every 10ms (we need fast update for the stest demo)
gui_interval 20

# configure the GUI window
window
(
  size [ 591.000 638.000 ]
  center [-0.010 -0.040]
  scale 0.028
)

# load an environment bitmap
map
(
  bitmap "bitmaps/cave.png"
  size [16 16]
  name "cave"
)

#defines the robot model
wbr914
(
  name "wbr914_1"
  color "white"
  pose [-5.5 -6.5 45]
  hokuyo_URG_laser( pose [ 0 0 0 ] )
)
```

Step 27: Open a terminal window and go to the '~/stage-2.0.3/worlds/' directory. Again verify that no other player servers are running with the 'ps aux |grep player' command.


wbr@pcbot: ~/stage-2.0.3/worlds
File Edit View Terminal Tabs Help

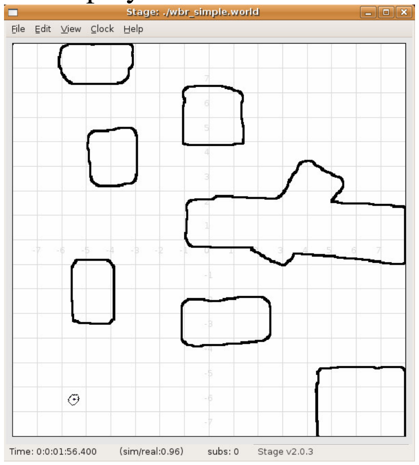
```
wbr@pcbot:~$ cd stage-2.0.3/
wbr@pcbot:~/stage-2.0.3$ cd worlds/
wbr@pcbot:~/stage-2.0.3/worlds$ ls
autolab.cfg       laser.inc      pucktarget.inc  wbr914sim.cfg
autolab.world     Makefile       sick.inc        wbr914sim.world
beacons.inc       Makefile.am    simple.cfg      wbr_simple.world
bitmaps           Makefile.in    simple.world    wbr_wavefront.cfg
chatterbox.inc    map.inc        vfh.cfg
everything.cfg    pioneer.inc    wavefront.cfg
everything.world  player.cspace  wbr914.inc
wbr@pcbot:~/stage-2.0.3/worlds$ ps aux |grep player
wbr      13112  0.0  0.0   2876   792 pts/4    R+   16:05   0:00 grep player
wbr@pcbot:~/stage-2.0.3/worlds$
```

Step 28: Type the following command 'player wbr_wavefront.cfg'

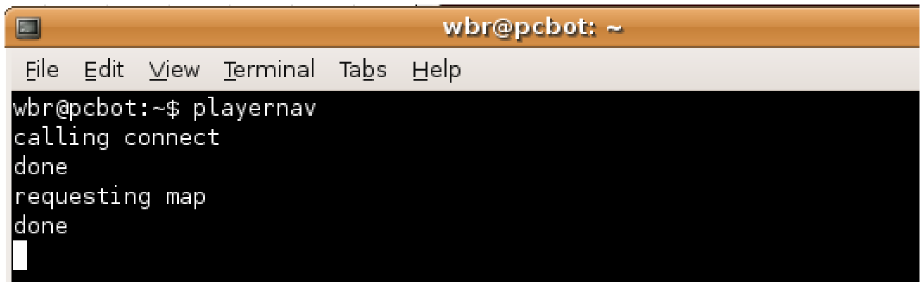

wbr@pcbot: ~/stage-2.0.3/worlds
File Edit View Terminal Tabs Help

```
wbr@pcbot:~$ cd stage-2.0.3/
wbr@pcbot:~/stage-2.0.3$ cd worlds/
wbr@pcbot:~/stage-2.0.3/worlds$ ls
autolab.cfg       laser.inc      pucktarget.inc  wbr914sim.cfg
autolab.world     Makefile       sick.inc        wbr914sim.world
beacons.inc       Makefile.am    simple.cfg      wbr_simple.world
bitmaps           Makefile.in    simple.world    wbr_wavefront.cfg
chatterbox.inc    map.inc        vfh.cfg
everything.cfg    pioneer.inc    wavefront.cfg
everything.world  player.cspace  wbr914.inc
wbr@pcbot:~/stage-2.0.3/worlds$ ps aux |grep player
wbr      13112  0.0  0.0   2876   792 pts/4    R+   16:05   0:00 grep player
wbr@pcbot:~/stage-2.0.3/worlds$ player wbr_wavefront.cfg
```

The following window opens up once the player server loads.

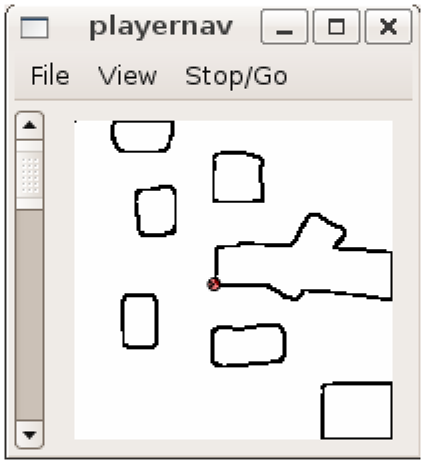

Stage: ./wbr_simple.world
File Edit View Clock Help
Time: 0:0:01:56.400 (sim/real:0.96) subs: 0 Stage v2.0.3

Step 29: Open another terminal window. Type the following command in 'playernav'.

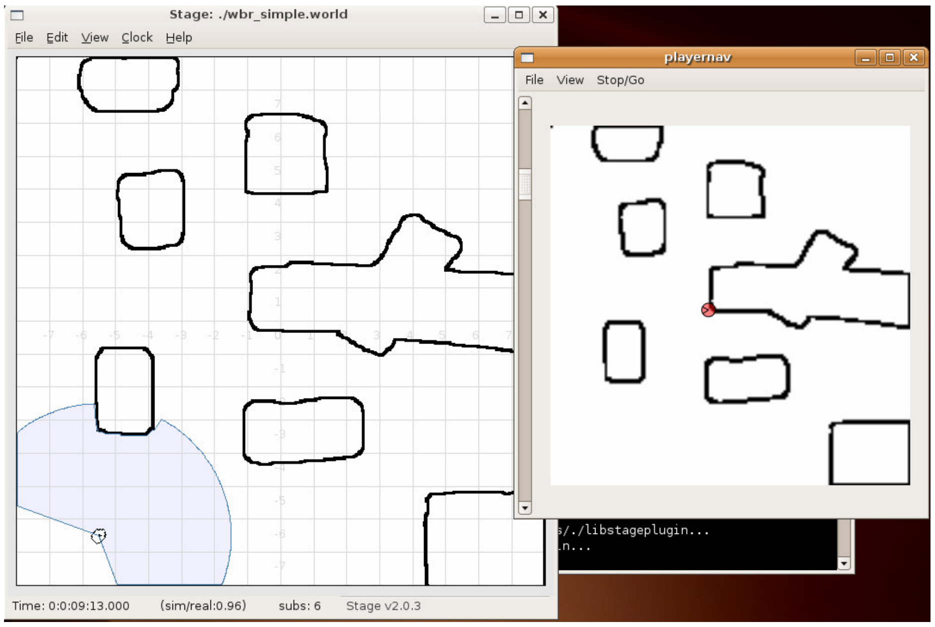


```
wbr@pcbot:~$ playernav
calling connect
done
requesting map
done
```

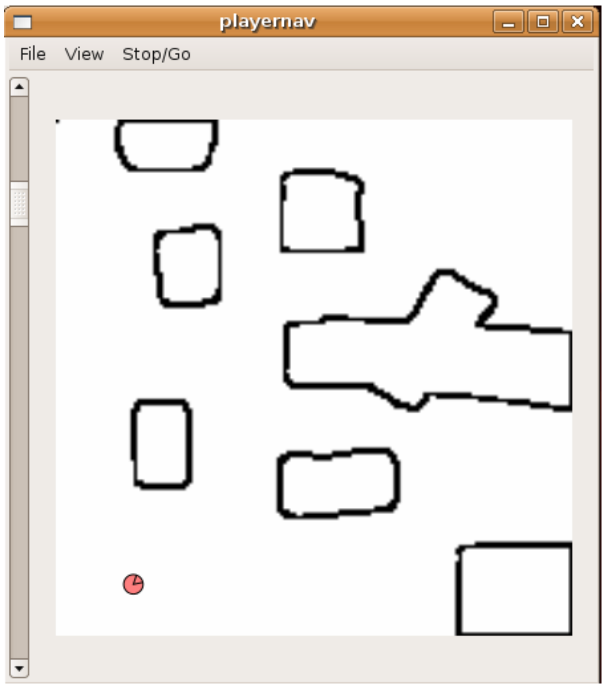
Another viewer window opens and shows the same map as in the Stage window.



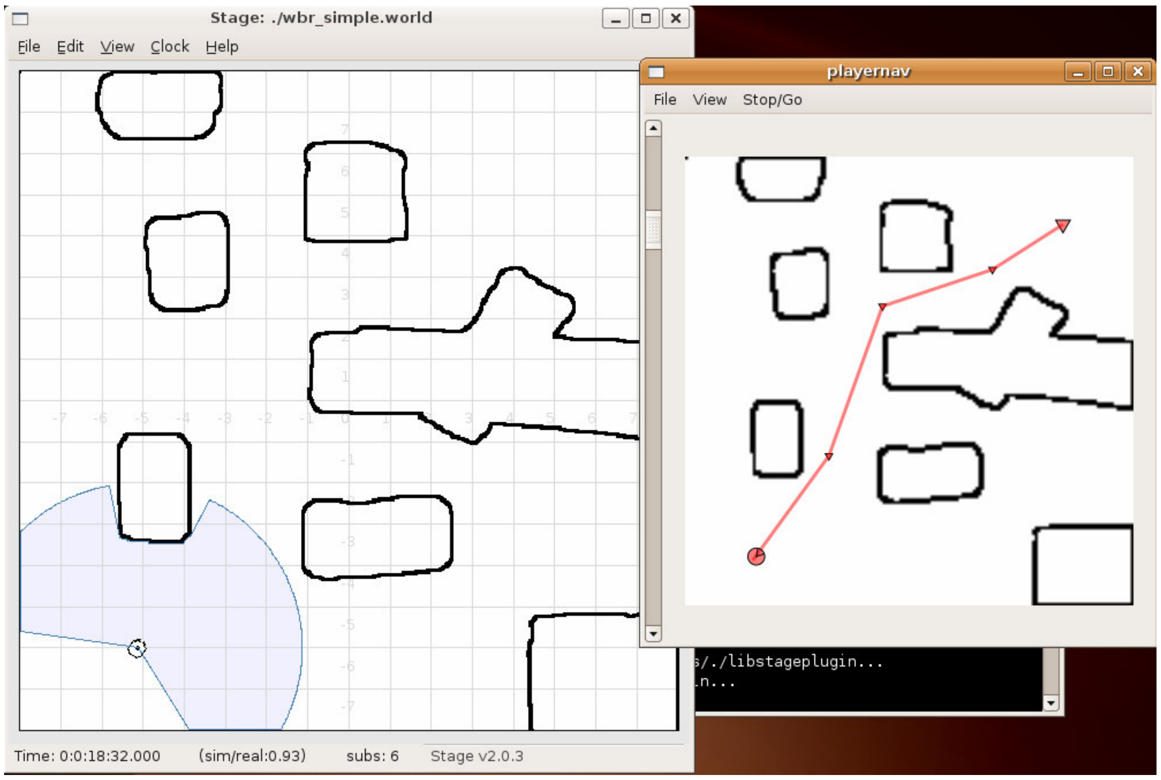
This window should be expanded so that you can drag the scroll bar down to zoom in. Put the two windows side by side.


Stage: ./wbr_simple.world
File Edit View Clock Help
playernav
File View Stop/Go
Time: 0:0:09:13.000 (sim/real:0.96) subs: 6 Stage v2.0.3

Step 30: On the playernav windows, click with the ‘left’ mouse button, drag and move the robot model (identified with the red circle) to the same location as shown in the Stage window. You need to do this or the robot will have to attempt to localize itself by driving and scanning the walls to learn where it is in the world. Click with the left mouse button again on top of the robot when the line appears.

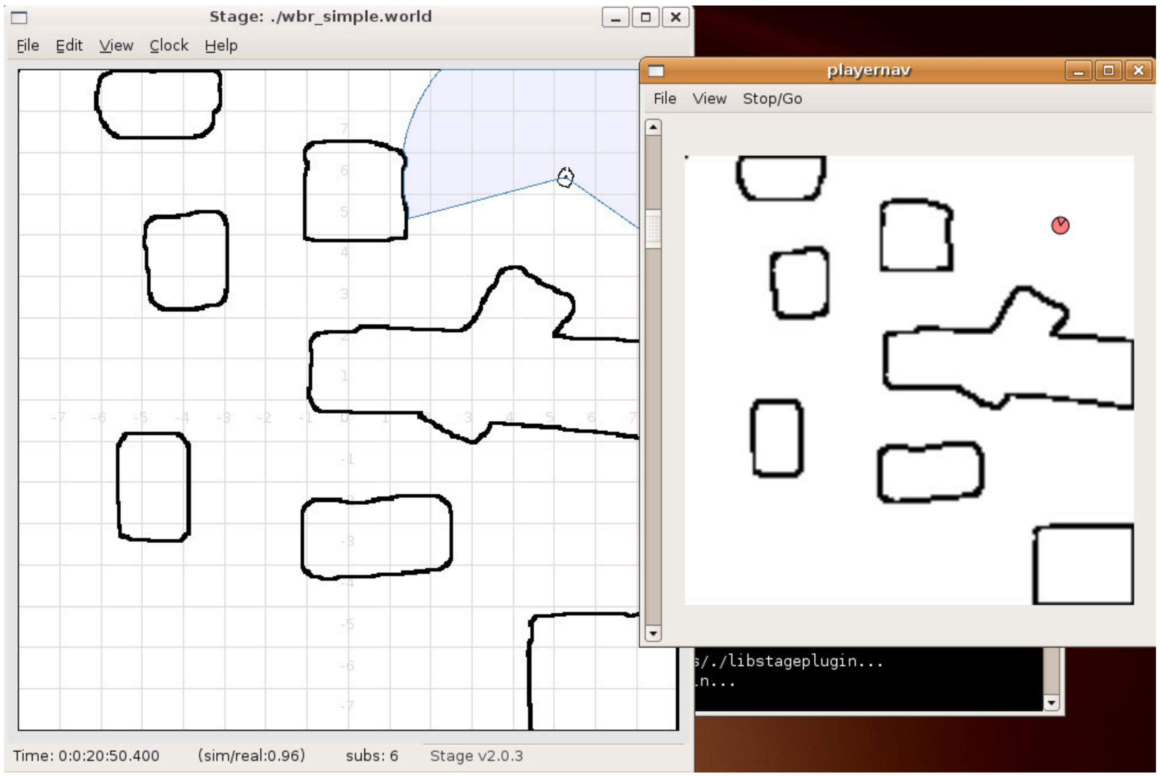


Step 31: In the playernav window, right click and drag the triangle that appears to some other location in the playernav window. When you let go of the right mouse button the triangle will be placed designating a general ultimate location. You then need to left click and have the waypoints plotted to show the robot path. The robot begins to drive to the end point optimizing the path through sensed objects.



```
s/./libstageplugin...
.n...
```

Step 32: As the robot drives, the waypoints will disappear. Notice also that the robot in the Stage window drives also to the end point.



```
s/./libstageplugin...
.n...
```

Time: 0:0:20:50.400 (sim/real:0.96) subs: 6 Stage v2.0.3

Close the playernav utility and player server when finished.

<u>Reference Documents and Web links for Player/Stage/Gazebo</u>

Note: All webpages and documentation is offered in English only.

Player/Stage/Gazebo Project Main Webpage:
http://playerstage.sourceforge.net/

Player WIKI page:
http://playerstage.sourceforge.net/wiki/Main_Page

Player/Stage/Gazebo Documentation: http://playerstage.sourceforge.net/index.php?src=doc

Basic frequently asked questions (good place if you have a problem)
http://playerstage.sourceforge.net/wiki/Basic_FAQ