

# Robotic Simulation Guide

Aaron Staranowicz, Gian Luca Mariottini  
ASTRA Robotics Lab, Department of Computer Science and Engineering  
University of Texas at Arlington

## Table of Contents

1	Introduction .....	4
1.1	Test bed.....	4
1.1.1	Dual boot with Ubuntu .....	5
2	COIL.....	6
2.1	Introduction .....	6
2.2	Installation of Create libraries.....	6
2.3	Example of connection to the robot .....	7
2.4	Example of interaction with the robot .....	9
2.4.1	Commanding in velocity.....	11
2.4.2	Reading the robot position .....	12
2.4.3	Commanding in position .....	13
3	Player/Stage/Gazebo .....	14
3.1	Introduction .....	14
3.2	Installation Player/Stage .....	15
3.2.1	Installation for Player .....	16
3.2.2	Installation for Stage .....	17
3.2.3	Environment Path for Player/Stage .....	19
3.3	Testing Player/Stage Install.....	20
3.4	Example of interaction to the robot .....	23
3.4.1	Commanding in velocity.....	24
3.4.2	Reading the robot position .....	28
3.4.3	Commanding in position .....	30
3.5	Installation Gazebo .....	32
3.5.1	Example of interaction with the simulation.....	35
4	Robot Operating System- ROS .....	42
4.1	Installation of ROS.....	42
4.1.1	Setup your sources.list.....	42
4.1.2	Set up your keys .....	43
4.1.3	Installation .....	43
4.1.4	Environment Setup .....	43
4.2	Example of interaction to the robot .....	44

4.2.1	Installation of Pyserial.....	44
4.2.2	Interaction with the robot .....	44
4.2.3	Commanding in position .....	45
4.2.4	Commanding in velocity.....	46
4.2.5	Reading the robot position .....	47

Appendix A

Appendix B

# 1 Introduction

The current panorama of robot control and simulation software systems witnessed a rapid growth in the past years.

However, there is still a lack for an effort to compare the characteristics and the performance of all these software systems. This is an important need, in particular for the novice user (amateur, student, researcher, etc.) willing to select the best software for their application.

In this work we present an in-depth analysis of the most recent and widely-used software control and simulation systems. The five software applications are: COIL, Player/Stage, Robot Operating System (ROS), Microsoft Robotic Development Studio, and USARSim.

After describing each of these tools, we will detail all the procedures necessary for their installation and describe how to face minor installation troubles. In addition, we will describe C/C++ code-examples that allow the user to start interacting with the robot. While most of these software applications can be easily specialized to be used for different robotic systems, we focus on the Create from iRobot. The Create is an inexpensive but effective robotic platform widely employed for research and education purposes.

All the examples contained in the Robotic Simulation guide have been realized under Ubuntu 10.04 Operating System (OS). Each system has been installed to the default directory `"/usr/local"`. All of the code needed to install and to run the sample programs are installed on the `"/home/$user/Desktop"` directory.

As you read this guide and the website, most of these software applications were developed for Linux and Mac OS based-systems with limited or no functionality for Windows based-systems (with the exception of Microsoft Robotic Development Studio).

## 1.1 Test bed

All of the software applications that are being installed are installed and tested on two different laptops setups which are:

1st laptop- Windows Vista 32-bit with Virtual Box running Ubuntu 10.04 32-bit

2nd laptop- dual boot of Windows 7 64-bit and Ubuntu 10.04 64-bit

The 1st laptop is used to install, test and run the iRobot Create, then the 2nd laptop is used to validate that the installation order is correct and no errors were missed. Note: This guide does not test on any Windows or Mac OS systems. Since most of the simulators, with the exception of Microsoft Robotic Developer Studio, were developed in a Linux environment and are supported in a Linux environment, time was not taken to test on a Windows or Mac OS system.

There are common errors between both installs but also there are errors associated with only to the 32-bit and only to the 64-bit operating systems.

Before we can begin describing the installation of each robotic software application, we need to ensure that Ubuntu 10.04 is up-to-date. First make sure you have the latest build-essential package.

Open a terminal window and type the following:

```
$ sudo apt-get install build-essential
```

Enter “y” when prompted if an update or install occurs. Once the installation is complete, restart your computer.

### 1.1.1 Dual boot with Ubuntu

The purpose of setting up a dual boot system with Linux operating system is that most of the software applications that are going to be described are for use on Linux. For this guide, we used Ubuntu 10.04 for installing, testing, and executing code. Information on installing and booting from either a CD or USB stick can be found on the Ubuntu website: <http://www.ubuntu.com/desktop/get-ubuntu/download>. Choosing to use a USB-stick can be helpful in the case that your computer does not have a CD drive (as for a notebook or the MacBook Air). To use a USB stick to install Ubuntu 10.04, Ubuntu recommends that you use a 3rd party software. We used PenDriveLinux and followed the instructions contained in: <http://www.pendrivelinux.com/universal-usb-installer-easy-as-1-2-3/>.

The basic idea is that you download and save the .ISO file to a directory you can access, then using PenDriveLinux software, you can format a 2GB or higher USB drive to become a bootable source. You can encounter an error using the PenDriveLinux software is when the software starts to compile the .ISO file on to USB stick. To correct this error, click No Persistence on the same screen when you first select the .ISO to use.

The next error encountered is during the installation of Ubuntu on to the target computer. The error is Ubuntu cannot find a file, directory, or continue with the setup to install, the only way to solve this error is to download a new .ISO file and start over from the PenDriveLinux software. The only explanation for this error that was found is the .ISO file was corrupted while downloading.

When the installation gets to the Partitioning step, the ideal situation is that you give half of the space to Windows and the other half to Ubuntu, but Ubuntu only needs 25% or 30% of the total space.

## 2 COIL

### 2.1 Introduction

COIL stands for Create Open Interface Library and is an open-source library that is supported by the GNU General Public License. The website is located at <http://code.google.com/p/libcreateoi/>. COIL is a C wrapper library for the iRobot Create's Open Interface. There are several advantages and disadvantages to this library.

The advantage is that it makes it much easier to start controlling the robot as well as reading data from it by using C functions instead of having to send the op-codes to the robot. COIL's size makes it easier to run on a net-book or one of the Create gumstixs.

A drawback is that COIL does not work on Windows OS. Linux and Mac OS systems are the only supported systems at this time. COIL requires an additional dependence which is libserial which is a collection of C++ classes that allow access to serial ports. Libserial is located at <http://libserial.sourceforge.net/index.html>.

### 2.2 Installation of Create libraries

The source code is on the Source tab on the website. You will need svn to checkout this library. If you do not have svn type the next command in the terminal window:

```
$ sudo apt-get install subversion
```

In the terminal, move to a place where you want the library to be extracted to, for example the Desktop.

```
$ cd ~/Desktop
```

```
$ svn checkout http://libcreateoi.googlecode.com/svn/trunk/libcreateoi-read-only
```

Before we are able to use the library and code, you will need to have a serial wrapper library that is used with COIL. The library is located at <http://libserial.sourceforge.net/index.html>. To download the library, click on the download tab. Once you download the .tar file. Extract it to the desktop.

```
$ tar -xf libserial-<version>.tar.gz (<version> is the version of the libserial that you downloaded)
```

```
$ cd libserial-<version>
```

```
$ ./configure (This allows you to see if you have the proper dependencies)
```

Before you can make and install libserial you will need to fix some errors. To fix these errors, you will need to add some "include" headings to the following C++ files.

Table 1 - "Include" headings that need to be added

Folder	File	Need to be added
/src	SerialStreamBuf.cc	#include <string.h>
/src	SerialPort.cpp	#include <stdlib.h> #include <math.h> #include <string.h>
/src	PosixSignalDispatcher.cpp	#include <stdlib.h> #include <string.h>
/examples	read_port.cpp	#include <stdio.h> #include <stdlib.h>
/examples	write_port.cpp	#include <stdio.h> #include <stdlib.h>

Once you entered all the #include in all files. In the terminal, make sure you are still in the libserial folder then type:

```
$ make
$ sudo make install (sudo is used to give access to root, this is a default install)
```

The Ncurses library needs to be installed to use the sample programs that came with the libcreateoi-read-only folder. The purpose of Ncurses library is to allow programmers to write text user interfaces. The sample programs use the Ncurses library to create a separate window to enter commands in to the program instead of using the terminal window. To install the Ncurses library type in the terminal:

```
$ sudo apt-get install libncurses5-dev
```

Once the libserial library and Ncurses library is installed and no further errors occur, then you can continue with installing COIL.

```
$ cd ~/Desktop/libcreateoi-read-only
$ make
$ sudo make install
```

Once the install is complete with no errors, COIL is ready to be used.

## 2.3 Example of connection to the robot

Plug in the USB-to-Serial cable to the computer and to the robot. Turn on the iRobot create and check to see which port is in use for the USB-to-Serial cable. In the terminal type:

```
$ dmesg |grep -i usb
```

Dmesg is used to print the message buffer of the kernel and grep searches for the keywords. In this case we want to list all possible matches to usb, ignoring case “-i”.

You should see the following message in the terminal if the USB-to-Serial cable is properly installed.

```
[ 288.715172] usbcore: registered new interface driver usbserial
[ 288.719578] USB Serial support registered for generic
[ 288.723304] usbcore: registered new interface driver usbserial_generic
[ 288.723390] usbserial: USB Serial Driver core
[ 288.743264] USB Serial support registered for pl2303
[ 288.833367] usb 2-1: pl2303 converter now attached to ttyUSB0
[ 288.833599] usbcore: registered new interface driver pl2303
[ 288.833654] pl2303: Prolific PL2303 USB to serial adaptor driver
```

Figure 1 - Results of dmesg

If you did not see these lines, type:

```
$ lsusb
```

This will show USB devices currently connected. You should get the following output.

```
aaron@aaron-laptop: ~
File Edit View Terminal Help
aaron@aaron-laptop:~$ lsusb
Bus 002 Device 002: ID 0557:2008 ATEN International Co., Ltd UC-232A Serial Port [pl2303]
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
aaron@aaron-laptop:~$
```

Figure 2- Results of lsusb

If you do not see the Serial Port [pl2303] line, unplug the USB-to-Serial cable, run lsusb, check to see which USB devices are running, then plug the USB-to-Serial cable back in to the computer and run lsusb again. You should see a line added to the previous ones in Fig.2 which should be the USB-to-Serial cable.

```
$ sudo modprobe usbserial vendor=0x0557 product=0x2008
(this will install the drivers for the USB-to-Serial cable)
```

Note: The vendor and product numbers will come from the ID 1d6b:0002. When the added line shows up use that vendor and product numbers.



Check to see if the USB-to-Serial cable is working properly.

```
$ dmesg |grep -i usb
$ lsusb
```

## 2.4 Example of interaction with the robot

The sample drive program is a good example to show if the USB-to-Serial cable, the libraries, and dependencies that were installed are working correctly. Plug in the USB-to-Serial cable to both the computer and the iRobot Create and turn on the iRobot Create.

```
$ cd ~/Desktop/libcreateoi-read-only/samples
$ make
```

You might run in to some errors for the other programs but as long as the executable for the drive program is created then you don't need to worry about the other errors. Those errors could be related to Open CV not being installed. The tracker program uses some OpenCV functions to pull images and a stream from the webcam.

You will need to get the USB device location of the USB-to-Serial cable.

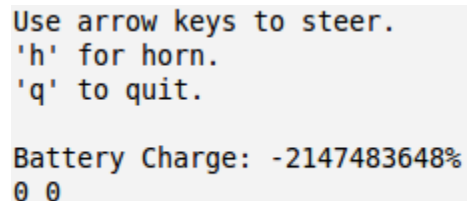
```
$ dmesg |grep -i usb
```

Look for this specific line: `usb 3-2: pl2303 converter now attached to ttyUSB0`  
This line tells you which location the USB-to-Serial cable is connected to.

To run the program, type:

```
$ ./drive <usb device location>    ( ex. /dev/ttyUSB0)
```

A new window should open up. The first thing you should look for if the power button on the robot changes from a green light to a red light. Next, check if the battery line actually has a real value.



```
Use arrow keys to steer.
'h' for horn.
'q' to quit.
Battery Charge: -2147483648%
0 0
```

Figure 3-Example of Not Connected to the Create

```

Use arrow keys to steer.
'h' for horn.
'q' to quit.

Battery Charge: 99%
0 0

```

Figure 4- Example of Connected to Create

If the Battery Charge: % is correct and a red light is shown on the power button, you should be able to control the iRobot Create with the up, down, left, and right arrows.

Up arrow – move the robot forward at a constant speed

Down arrow – move the robot backwards at a constant speed

Left arrow – turns the robot to the left

Right arrow – turns the robot to the right

The iRobot Create might not connect properly and to ensure the robot connects properly, try the following solutions:

1. To restart the computer
2. The iRobot Create needs to be charged
3. Delete the executable with `$ make clean` then recompile with `$ make`

Another example of COIL:

exampleCOIL.c

```

/** exampleCOIL.c
 * A simple program to move the robot forward with a constant velocity and
 * prints the current velocity
 */
#include <createoi.h> //this is the heading file that COIL uses
#include <stdlib.h>
#include <stdio.h>

int main(int argv, char* argc[])
{
    int not_done = 1;
    int v = 0;
    int velocity = 100; //velocity of the robot
    int radius = 0; // radius the robot should move in

    startOI_MT ("/dev/ttyUSB0");

    while(not_done)

```

```

        {drive (velocity, radius);
         v = getVelocity ();
         printf("Velocity %d \n",v);
        }
    stopOI_MT();//stops the program
}

```

## Makefile

```

# This is a very simple Makefile for compiling COIL sample
# applications.
CC = gcc -g
INCLUDE = /usr/local/include
LIBS = -lpthread -lm -lncurses -lcreateoi
CVFLAGS = $(shell pkg-config --libs opencv) $(shell pkg-config --cflags
opencv)

default: all

all: exampleCOIL

exampleCOIL: exampleCOIL.c
    $(CC) exampleCOIL.c $(LIBS) $(CVFLAGS) -o exampleCOIL
clean:
    rm -f *.o
    rm -f exampleCOIL

```

Type `make` in the terminal to compile and create the executable. Once the robot is connected and turned on. Run the executable. The robot should move forward with a constant velocity and the velocity will be printed to the terminal window.

### 2.4.1 Commanding in velocity

With COIL, most of the functionality to control the iRobot create has been coded in the `createoi.h` file.

To move the robot, the `drive ()` function is used.

```
int drive (short vel, short rad)
```

**NOTE:** For any function to work there needs to be a space between the function name and the beginning parenthesis.

The first parameter is how fast the robot will move both wheels. The speed ranges from -500 to 500 mm/s. A positive number moves the robot forward and a negative number moves the robot backwards.

The second parameter is the radius the robot should drive in. Its values range from -2000 to 2000 mm. A positive number turns the robot left and a negative number turns the robot right.

The `drive ()` function returns a 0 if it was successful or -1 otherwise.

Ex: To move the robot 100mm/s in a straight line (aka 0 radius):

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

int main()
{
    startOI_MT ("/dev/ttyUSB0");
    while(1)
    {drive (10, 0);
    }
    stopOI_MT();
}
```

## 2.4.2 Reading the robot position

The position of the robot can be read by using the `getDistance ()` function.

```
int getDistance ()
```

The `getDistance ()` function returns the distance the Create moved since the last time the function was called or since the distance sensor was polled.

Ex: To read the robot's position according to the encoders:

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

int main()
{    int x = 0;
    startOI_MT ("/dev/ttyUSB0");

    while(1)
    {drive (10, 0);
      x = getDistance ();
      printf("Distance traveled: %d ", x);
    }
    stopOI_MT();
}
```

### 2.4.3 Commanding in position

The robot can also move to a specific distance. The `driveDistance ()` function is used.

```
int driveDistance (short vel, short rad, int dist, int interrupt)
```

The first and second parameter is the same as the `drive ()` function.

The third parameter is the desired distance in mm.

The fourth parameter is the interrupt. The interrupt is set to 0 if you want to ignore collisions and set to 1 if you want to terminate on collision.

The function returns distance travelled or `INT_MIN` on error

Ex: Move the robot at 300mm/s in a straight line for 200mm NOT ignoring collisions.

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

int main()
{
    startOI_MT ("/dev/ttyUSB0");
    while(1)
    {driveDistance (300, 0, 200, 1);
    }
    stopOI_MT();
}
```

## 3 Player/Stage/Gazebo

### 3.1 Introduction

Player/Stage is open-source software that allows direct access to the iRobot Create but also allows the user to simulate an iRobot Create. Player provides a network interface to a variety of robot and sensor hardware. Player's client/server model allows robot control programs to be written in any programming language and to run on any computer with a network connection to the robot. One of the main features of Player is in its support for multiple concurrent client connections to devices, creating new possibilities for distributed and collaborative sensing and control.

Stage simulates a population of mobile robots moving in and sensing a two-dimensional bitmapped environment. Various sensor models are provided, including sonar, scanning laser rangefinder, pan-tilt-zoom camera with color blob detection and odometry. Stage devices present a standard Player interface so few or no changes are required to move between simulation and hardware. Many controllers designed in Stage have been demonstrated to work on real robots.

Gazebo is a multi-robot simulator for outdoor environments. Like Stage, it is capable of simulating a population of robots, sensors and objects, but does so in a three-dimensional world. It generates both realistic sensor feedback and physically plausible interactions between objects (it includes an accurate simulation of rigid-body physics). Gazebo presents a standard Player interface in addition to its own native interface. Controllers written for the Stage simulator can generally be used with Gazebo without modification (and vice-versa).

More details can be found on: <http://playerstage.sourceforge.net/>.

The advantage of Player/Stage is the ability to move from simulation to the robot by changing a few parameters. This transition from a simulation to the robot allows for a wider range of tests that can be performed before moving to a real robot. The network connections between different computers allows for control and supervision of the robots in real time at a remote terminal instead of having to record and play-back the data.

The disadvantage is the learning curve on this software application. Due to the unavailability of a complete and step-by-step guide, it does take time to learn how to use Player/Stage so this is not beginner friendly. However, after this initial struggle, Player/Stage reveals a powerful tool to have. Another downfall of Player/Stage is a large number of dependencies required to have Player/Stage operate to its fullest.

## 3.2 Installation Player/Stage

For installation instruction on Player/Stage, this website helps with the installing process:

[http://www.control.aau.dk/~tb/wiki/index.php/Installing\\_Player\\_and\\_Stage\\_in\\_Ubuntu](http://www.control.aau.dk/~tb/wiki/index.php/Installing_Player_and_Stage_in_Ubuntu)

The order of installation is important. First the system libraries must be up-to-date. Then the dependency libraries for Player/Stage must be downloaded and installed before Player/Stage. After all the libraries are up-to-date and are installed correctly, Player must be installed and working before Stage can be installed.

Run the following commands in the terminal to update the system to the latest version. Then restart the computer:

```
$ sudo apt-get update  
$ sudo apt-get upgrade
```

The easiest way to install the dependency libraries is to use the “.markings” files. The link for the “.marking” files is at: [http://playerstage.sourceforge.net/wiki/Install\\_ubuntu\\_packages](http://playerstage.sourceforge.net/wiki/Install_ubuntu_packages).

All of the “.markings” files will need to be installed for Player, Stage, and Gazebo to work. It is best to install all the dependencies so that the installation of Player/Stage goes smoothly but also to ensure that the installation of Gazebo goes smoothly.

From the link, copy each section on the website to a new text file and name the text file the title of the “.markings” section such as “ubuntu-building.markings”. Once you have the 4 packages in text files, open Synaptic Package Manager (in menu System/Administration) from the Linux menu. Click on File-> Read Markings and open the “.markings” file. You should see at the bottom of the Synaptic Package Manager an updated status that shows a number of packages that will be installed and upgraded.

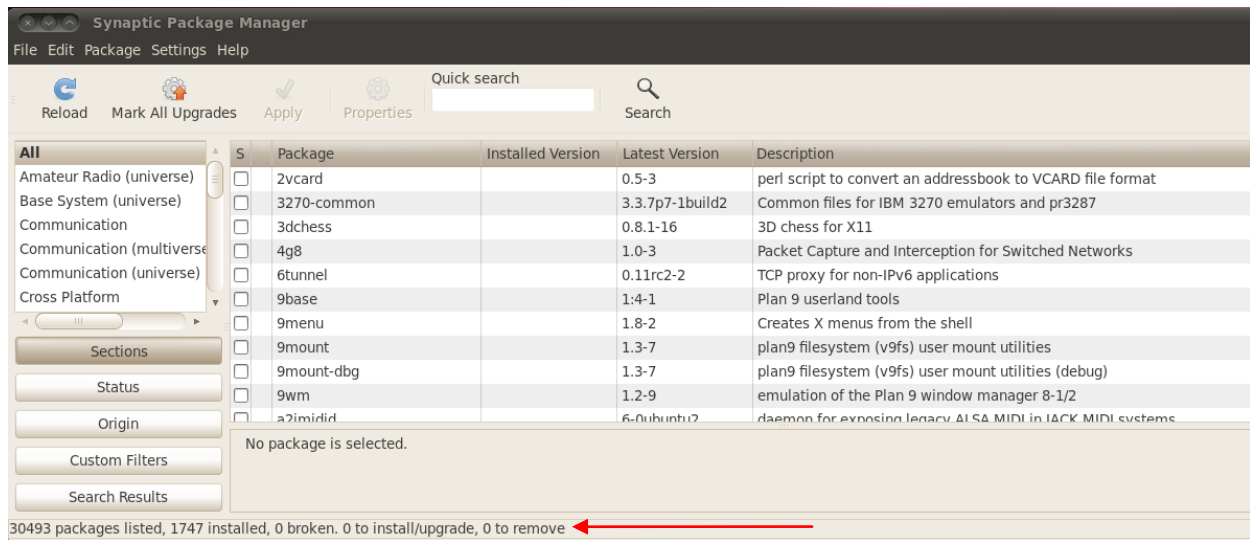


Figure 5- Synaptic Package Manager in Ubuntu 10.04

Example of the status at the bottom of the Synaptic Package Manager in Linux: 30156 packages listed, 1411 installed, 0 broken, **115 to install/upgrade**, 0 to remove, 325 MB will be used. Once all “.markings” files are opened which should total about 115 packages, click on Apply to begin installing them. Once all of the packages are installed, restart the computer.

We can start the installation of Player and Stage. Download the “.tar” files from the links:

<http://sourceforge.net/projects/playerstage/files/Player/> (for Player)

<http://sourceforge.net/projects/playerstage/files/Stage/> (for Stage)

The version of Player we are using for this installation guide is 3.0.2 and for Stage is 3.2.2. This installation guide should still be of use for the latest version of Player and Stage.

Move the tar files from your download directory to the desktop and unpack the “.tar” files. Either unpack by using the Archive Manger or by typing in the terminal:

```
$ cd ~/Desktop
$ tar xzvf player-<version>.tar.gz (the version is the version of player you downloaded.)
$ tar xzvf Stage-<version>-Source.tar.gz (the version is the version of stage you downloaded.)
```

Player has to be installed before Stage (Stage will look for Player during its installation). If Player is not found by Stage, linking errors will occur and they can only be fixed with uninstalling Stage.

### 3.2.1 Installation for Player



This guide used the default configuration settings for Player. In Player 3.0.0 and later version, use cmake with the path to configure the install. To begin the installation process type in the terminal:

```
$ cd player-<version>
$ mkdir build
$ cd build
$ cmake ../
```

When the configuration is completed, without errors, you can begin installing Player with:

```
$ make
$ sudo make install
```

While Player can be installed in any directory, we recommend the novice user to install using the defaults configurations.

Type the following command to see if player did install correctly:

```
$ player
```

Once the installation is complete with no errors<sup>1</sup>, Stage can be installed.

If you get an error that a library cannot be found, try adding these lines to the .bashrc file that is hidden in the home directory:

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

If the error still persists, try typing `$export PLAYERPATH=/usr/local/lib`, then running `$sudo ldconfig`

Type the following command to see if player is working correctly:

```
$ player
```

### 3.2.2 Installation for Stage

To begin the installation process for Stage type in the terminal:

---

<sup>1</sup> Installation errors could happen during the installation process. The only solution is to:

```
$ sudo make uninstall (in the Player build directory)
```

Restart the process over again by downloading a new “.tar” file.

```
$ cd Stage-<version>
$ mkdir build
$ cd build
$ cmake ../
```

Once the configuration is complete, without errors, then complete the installation with:

```
$ make
$ sudo make install
```

As with Player, the default configuration is recommended but for advance users a different install directory could be configured. If you do use a different install directory, it is recommended that both Player and Stage are installed in the same directory. It makes it easier for Stage to find Player.

Type the following command to see if stage did install correctly with no errors<sup>2</sup>:

```
$ stage
```

There is an error where a library cannot be found. Type:

```
$ export STAGEPATH=/usr/local/lib
$ sudo ldconfig
```

If both installations completed with no errors, we can verify that both Player and Stage installed correctly.

First ensure that player is in the PATH, by typing:

```
$ which player
```

This should result in the following output to the terminal window: <INSTALL\_DIR>/bin/player

Second ensure that stage is in the PATH, by typing:

```
$ which stage
```

This should result in the following output to the terminal window: <INSTALL\_DIR>/bin/stage

---

<sup>2</sup> The error encountered is the libraries could not link Player and Stage together. The only solution is:

```
$ sudo xargs rm < install_manifest.txt (in the Stage build directory)
```

Restart the process over again by downloading a new “.tar” file.

```
irobot2@irobot2-laptop:~$ which player
/usr/local/bin/player
irobot2@irobot2-laptop:~$ which stage
/usr/local/bin/stage
irobot2@irobot2-laptop:~$ █
```

Figure 6- Terminal Results of Player and Stage

### 3.2.3 Environment Path for Player/Stage

The path needs to be defined for Player and Stage for Player and Stage to work correctly. If the above yields the correct results then the path is defined, but if they do not then add these lines to your .bashrc file (a hidden file) located in the root of your home directory:

```
export PKG_CONFIG_PATH="<INSTALL_DIR>/lib/pkgconfig":$PKG_CONFIG_PATH
export LD_LIBRARY_PATH="<INSTALL_DIR>/lib"
export STAGEPATH="<INSTALL_DIR>/lib"
export PLAYERPATH="<INSTALL_DIR>/lib"
```

Run the .bashrc file:

```
$ source .bashrc
```

A restart is needed to ensure that the exports were written to the .bashrc file.

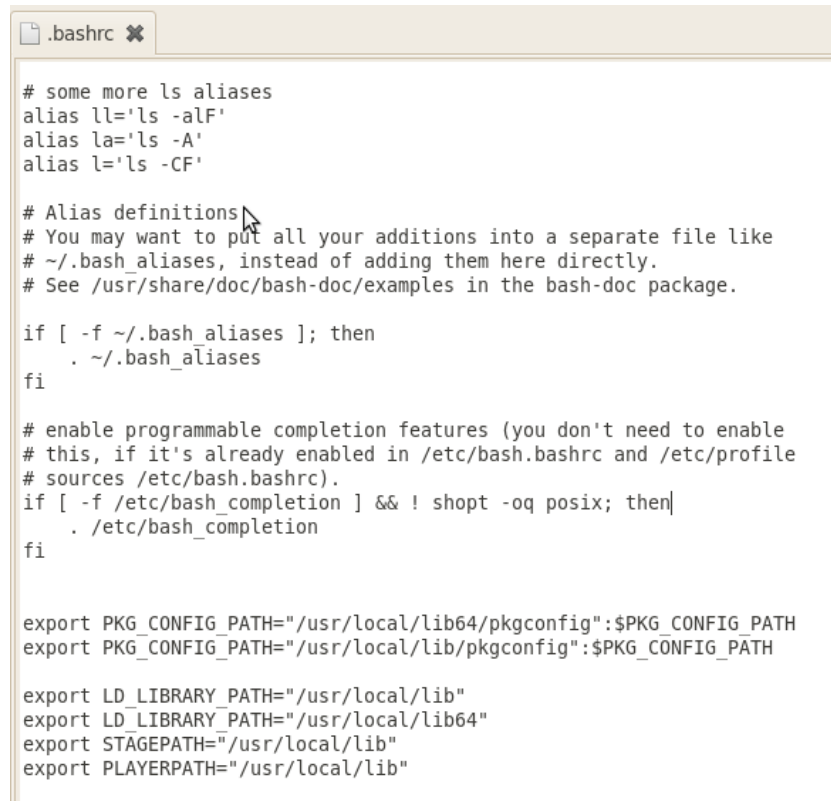
If you have a 64-bit system check to see if the player libraries installed in the `"/usr/local/lib64"` directory, if Player did install in the 64-bit lib. Then type this line to the .bashrc file:

```
export PKG_CONFIG_PATH="<INSTALL_DIR>/lib64/pkgconfig":$PKG_CONFIG_PATH
export LD_LIBRARY_PATH="<INSTALL_DIR>/lib64"
```

Run the .bashrc file:

```
$ source .bashrc
```

A restart is needed to ensure the exports were written to the .bashrc file.



```
.bashrc
# some more ls aliases
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'

# Alias definitions
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
    . /etc/bash_completion
fi

export PKG_CONFIG_PATH="/usr/local/lib64/pkgconfig:$PKG_CONFIG_PATH"
export PKG_CONFIG_PATH="/usr/local/lib/pkgconfig:$PKG_CONFIG_PATH"

export LD_LIBRARY_PATH="/usr/local/lib"
export LD_LIBRARY_PATH="/usr/local/lib64"
export STAGEPATH="/usr/local/lib"
export PLAYERPATH="/usr/local/lib"
```

Figure 7- .bashrc file with the exports needed for Player and Stage

### 3.3 Testing Player/Stage Install

The following is the testing procedure for a correct installation of Player and Stage.

```
$ cd ~/Desktop/Stage-<version>-Source/worlds
```

```
$ stage simple.world
```

A new window should open. The simulation might start paused, to unpause press “p”. A path-related error might appear: in order to fix it, go to the home folder, click on View->View Hidden Folder and find the .bashrc file. At the bottom of the file you should see the paths that were typed earlier, if not then check Section 3.2.3.

```
irobot2@irobot2-laptop:~$ cd Desktop/
irobot2@irobot2-laptop:~/Desktop$ cd Stage-3.2.2-Source/
irobot2@irobot2-laptop:~/Desktop/Stage-3.2.2-Source$ cd worlds/
irobot2@irobot2-laptop:~/Desktop/Stage-3.2.2-Source/worlds$ stage simple.world
Stage 3.2.2
[Loading simple.world][Include pioneer.inc][Include map.inc][Include sick.inc]
]
```

Figure 8- Terminal Result

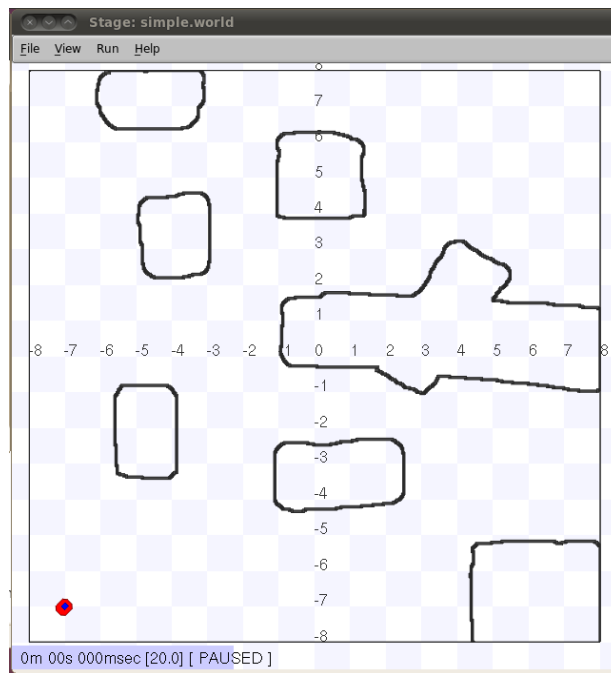


Figure 9- Stage

If a window does open, then move on to testing Player. In the same directory, in the terminal type:

```
$player simple.cfg
```

```

irobot2@irobot2-laptop:~/Desktop$ cd Stage-3.2.2-Source/
irobot2@irobot2-laptop:~/Desktop/Stage-3.2.2-Source$ cd worlds/
irobot2@irobot2-laptop:~/Desktop/Stage-3.2.2-Source/worlds$ stage simple.world
Stage 3.2.2
[Loading simple.world][Include pioneer.inc][Include map.inc][Include sick.inc]
User exited via menu
irobot2@irobot2-laptop:~/Desktop/Stage-3.2.2-Source/worlds$ player simple.cfg
Registering driver
Player v.3.0.2

* Part of the Player/Stage/Gazebo Project [http://playerstage.sourceforge.net].
* Copyright (C) 2000 - 2009 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.

invoking player driver init()...
Stage driver plugin init

** Stage plugin v3.2.2 **
* Part of the Player Project [http://playerstage.sourceforge.net]
* Copyright 2000-2009 Richard Vaughan, Brian Gerkey and contributors.
* Released under the GNU General Public License v2.

success
Stage plugin: 6665.simulation.0 is a Stage world
[Loading ./simple.world][Include pioneer.inc][Include map.inc][Include sick.inc]
]

Stage plugin: 6665.position2d.0 is "r0"
Stage plugin: 6665.laser.0 is "r0.laser:0"
Stage plugin: 6665.speech.0 is "r0"
Stage plugin: 6665.graphics2d.0 is "r0"
Stage plugin: 6665.graphics3d.0 is "r0"
listening on 6665
Listening on ports: 6665

```

Figure 10 - Terminal Result using Player

The same window should open as the `$stage` command. If a new window does not open up then the path might not be set properly check Section 3.2.3. If you run in to the error that `libplayerdrivers.so` can not be open, try typing the following in the terminal window:

```
$ sudo ldconfig
```

If `$ sudo ldconfig` does not work, the best thing to do is uninstall player and stage:

#### Uninstall Player:

```
$ cd player-<version>/build
$ make uninstall
```

#### Uninstall Stage:

```
$ cd Stage -<version>-Source/build
```

```
$xargs rm < install_manifest.txt
```

Once Player and Stage have been uninstalled, download new “.tar” files and start from the beginning; this revealed to be the best solution to the problem of drivers and libraries not linked properly. The bug seems to carry over from version to version and most claim that it is a bad download.

### 3.4 Example of interaction to the robot

This is a simple example of controlling and connecting the iRobot. The following code is a configuration file used by Player to detect what environment to setup. A good resource on configuration files used by Player is located here: <http://playerstage.sourceforge.net/doc/Player-1.6.5/player-html/configfile.php>. When connecting to real robots like the Create the “alwayson 1” must be included in the driver. This allows the driver to start as Player is started rather than waiting for the client to connect. If “alwayson 1” was not included the Create would not connect as the Create does not send packets without being told to.

```
driver
(
    name "create"
    provides ["position2d:0"]
    port "/dev/ttyUSB0" //(Note: How to find this is described earlier in
the guide)
    alwayson 1
)
```

Copy and paste into a text file and save it on the desktop as “create.cfg”. Connect the USB-to-Serial cable to the computer and turn on the iRobot Create. Open two terminal windows and change to the desktop directory on both.

In the first terminal window, type:

```
$player create.cfg
```

This command starts Player and any drivers that were defined in the configuration file.

To control the robot, in the second terminal window, type:

```
$playerv
```

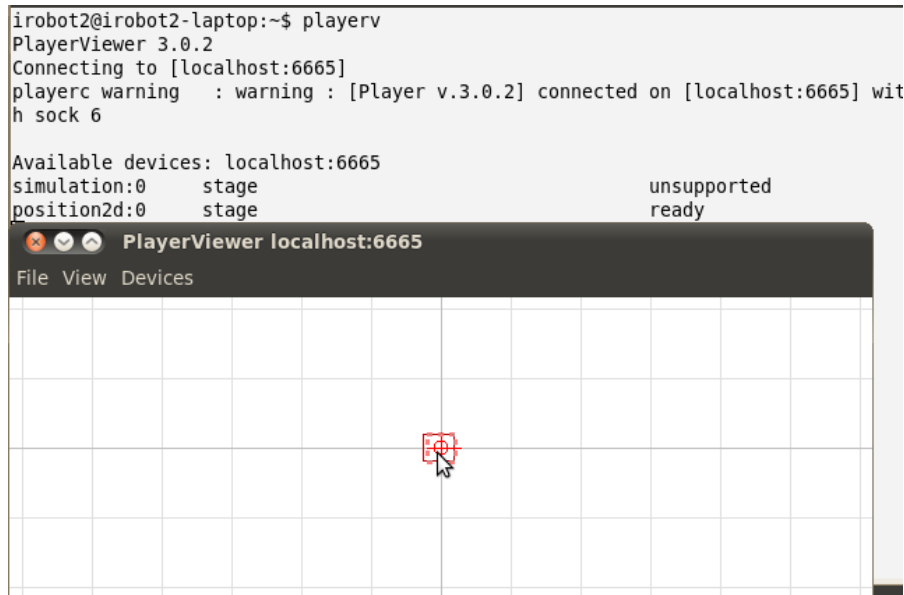


Figure 11- Playerv

This command is a software joystick. Playerv gives a top-down view of the robot on a grid. When the drivers have been subscribed to, the paragraphs below will tell you how to subscribe, Playerv allows a user to send velocity commands to the robot. Playerv also receives data from the robot through Player and shows the data in the window.

In the terminal, messages should be displayed if the position2d:0 is ready or not. If it is not ready then you have an error. Close Player and Playerv, then redo the commands over again.

This opens a new window that looks like a graph. On the menu bar click Devices -> position2d:0 -> Subscribe and then Command.

The robot should move according to the dragging of the circle along the graph in player.

### 3.4.1 Commanding in velocity

For this example, we will be working both with the simulation (Stage) and the real robot to show that the code is transferable and works both on Stage and the robot. All code will stay with the observed standard of Player/Stage.

A “.tar” file will be created and available for download to allow users to follow along with the following examples. The files are: args.h, Makefile, map.inc, pioneer.inc, simple.cfg, simple.world, commandVelocity.cc, square.png, and the create.cfg. The args.h file is a header file need to use some of the Player commands. The Makefile is used to compile the sample code. The map.inc defines some of the map functions used by Player and Stage to create the map of the environment. The pioneer.inc defines the model of the pioneer and also some of the functions used by Player and Stage to create the



pioneer. Simple.cfg is the configuration file used by Player and defines the drivers needed to setup the simulation in Stage. Simple.world is the model of the world that the pioneer will move around in. CommandVelocity.cc is a C++ file that defines the control code for the pioneer. Square.png is a drawing of the environment that Player will draw in the “world”. Create.cfg is the configuration file used by Player to setup the drivers needed to control the Create. The two files that will be described are modified examples that have been reduced down to commanding the robot with a certain velocity. The two important files are the Makefile and the modified file commandingVelocity.cc.

## Makefile

```
# Example makefile for Player client programs
#
# NOTE: Be sure you have LD_LIBRARY_PATH defined, to include /usr/local/lib
#       (typically, this goes in your .cshrc file (or appropriate variant
#       for the shell you are using) as "setenv LD_LIBRARY_PATH
/usr/local/lib")

PLAYERDIR = /pkgs/player-stage-2.0.3

CXX = g++
INCLUDES = -I$(PLAYERDIR)/include/player-2.0/ -I$(PLAYERDIR)/include/boost-
1_33_1/
CFLAGS = -Wall -g $(INCLUDES)

commandVelocity:
    $(CXX) $(CFLAGS) -o commandVelocity `pkg-config --cflags playerc++`
commandVelocity.cc `pkg-config --libs playerc++`

clean:
    rm -f a.out core commandVelocity *.o
    rm -f a.out core readPosition *.o
    rm -f a.out core commandPosition *.o

readPosition:
    $(CXX) $(CFLAGS) -o readPosition `pkg-config --cflags playerc++`
readPosition.cc `pkg-config --libs playerc++`

commandPosition:
    $(CXX) $(CFLAGS) -o commandPosition `pkg-config --cflags playerc++`
commandPosition.cc `pkg-config --libs playerc++`
```

This Makefile has to remain the same as it links the libraries to your control code. If you need to tailor the Makefile, change only the name that relates to your file name. Replace only “commandVelocity” with your file name.

commandVelocity.cc

```
/* commandVelocity.cc
```

```

    * a simple demo to move the robot forward at a certain velocity
    */
#include <libplayerc++/playerc++.h> //this library allows the use of Player
defined functions
#include <iostream>
#include "args.h"

int main(int argc, char **argv)
{parse_args(argc,argv);
  // we throw exceptions on creation if we fail
  try
  {using namespace PlayerCc;

    //connects to the robot/simulated robot
    PlayerClient robot(gHostname, gPort); //gHostname and gPort are defined in
the .cfg file. This is where 'port' in the driver section tells what port for
Player to send commands to
    //connects to the position sensor of the robot
    //enables writing commands to the robot
    Position2dProxy pp(&robot, gIndex); //gIndex is the index of the robot,
this is where if you have multiple robots, its own Index. From the .cfg file
provides ["position2d:0"], 0 is the index of the robot. Increment this number
to include more robots.

    std::cout << robot << " robot" << std::endl;
    pp.SetMotorEnable (true);

    double newspeed = 1.0; //robot velocity
    double newturnrate = 0.10; // turn velocity

    for(;;)
    {// this blocks until new data comes; 10Hz by default
      robot.Read();
      // write commands to robot
      pp.SetSpeed(newspeed, newturnrate);
    }
  }
  catch (PlayerCc::PlayerError e)
  { std::cerr << e << std::endl; //if the robot and pp fail to initialize
then the error is printed
    return -1;
  }
}

```

Once the files are unzipped on the desktop we can start to use the examples. Open two terminals, one will be to start Player/Stage and the other will to run the control code.

**Using Stage (the simulation):**

In one terminal:

```
$ make commandVelocity  
$ player simple.cfg //the configuration file for the simulation on  
Stage
```

Stage should be started and the robot sitting still.

In the other terminal:

```
$ ./commandVelocity
```

The robot should be moving at a constant velocity of 1 meter per second until it hits the wall.

**Using the real robot:**

Make sure that the robot is connected to the computer and is turned on. Also ensure that the robot is on the floor and you are able to grab it as it could go faster than expected.

In one terminal:

```
$ player create.cfg
```

The message in the terminal should print that it is connected to the robot.

In the other terminal:

```
$ ./commandVelocity
```

The robot should move forward with a constant velocity.

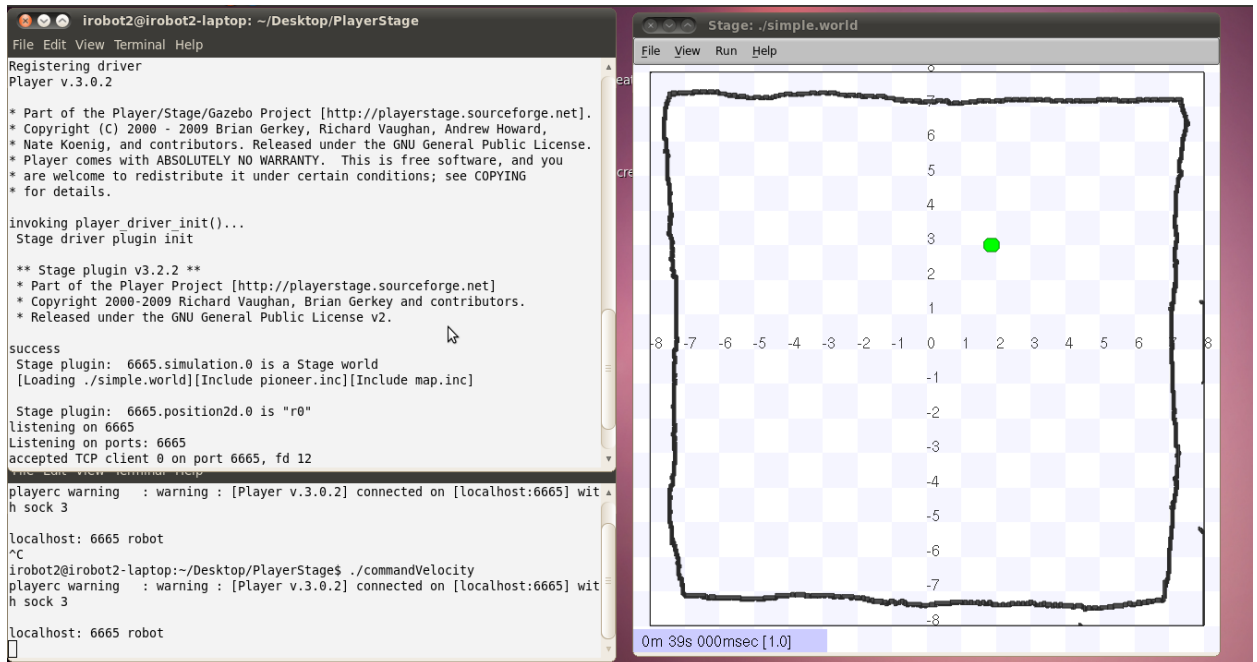


Figure 12 - Command Velocity in Stage

### 3.4.2 Reading the robot position

The `readposition.cc` code reads the  $x$  and  $y$  position of the moving robot. The wheels start at  $(x,y)=(0,0)$ . More information such as the yaw angle,  $x$  velocity,  $y$  velocity, and yaw angular velocity can be retrieved with functions defined by Player.

`readPosition.cc`

```
/* readPosition.cc
 * a simple demo to read the x and y position of the robot while moving at a
 * constant velocity
 */
#include <libplayerc++/playerc++.h>
#include <iostream>
#include "args.h"

int main(int argc, char **argv)
{ parse_args(argc,argv);
  // we throw exceptions on creation if we fail
  try
  { using namespace PlayerCc;
    //connects to the robot/simulated robot
    PlayerClient robot(gHostname, gPort);
    //connects to the position sensor of the robot
```

```

//enables writing commands to the robot
Position2dProxy pp(&robot, gIndex);

std::cout << robot << " robot" << std::endl;
pp.SetMotorEnable (true);

double newspeed = 1.0; //robot velocity
double newturnrate = 0.0; // turn velocity

//infinite loop to show effects on the simulator/robot
for(;;)
{ /* this blocks until new data comes; 10Hz by default */
  robot.Read();
  // write commands to robot
  pp.SetSpeed(newspeed, newturnrate);

  //gets the x position of the robot
  double x = pp.GetXPos();
  //gets the y position of the robot
  double y = pp.GetYPos();

  // Prints the x and y position to the terminal
  std::cout << x << " x pos" << std::endl;
  std::cout << y << " y pos" << std::endl;
}
}
catch (PlayerCc::PlayerError e)
{ std::cerr << e << std::endl;
  return -1;
}
}

```

### Using Stage (the simulation):

In one terminal:

```

$ make readPosition
$ player simple.cfg

```

Stage should be started and the robot sitting still.

In the other terminal:

```

$ ./readPosition

```

The robot should be moving at a constant velocity printing the x and y position until it hits the wall.

Using the real robot:

If you want to run this control code by connecting to the real robot, please run `$player create.cfg` and follow the same instructions as in Sect. 3.4.1.

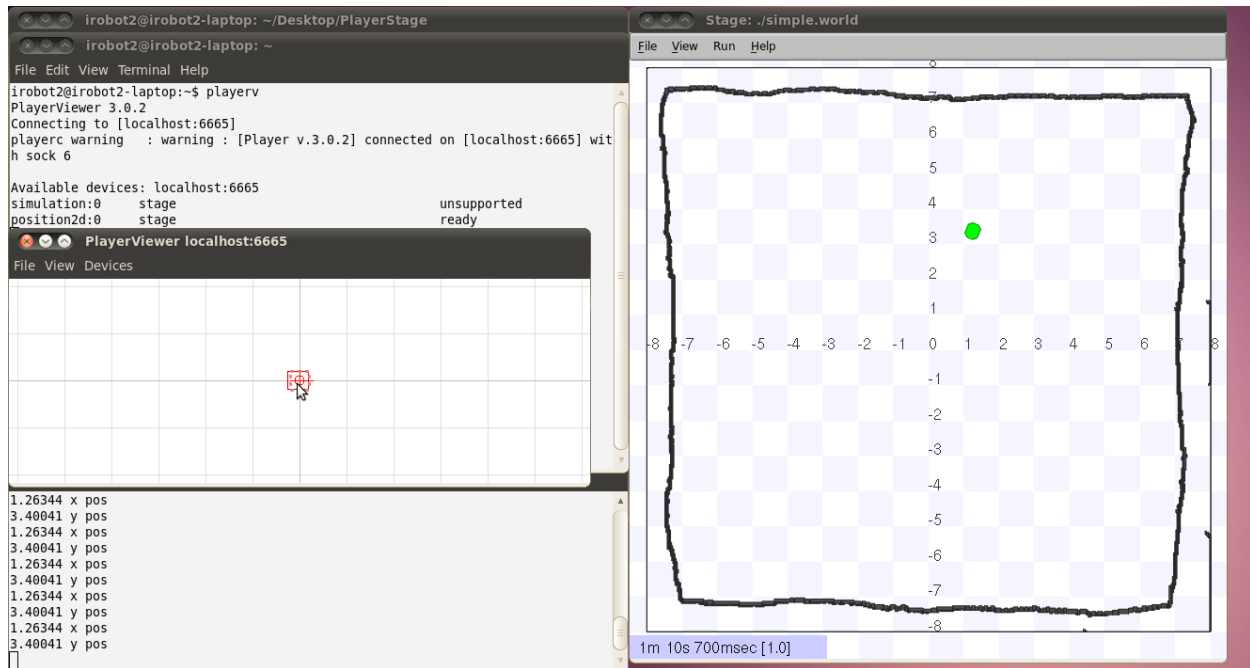


Figure 13- Read Position in Stage with Playercv to control the robot

### 3.4.3 Commanding in position

The `commandPosition.cc` moves the robot to the desired position with the desired velocity.

`commandPosition.cc`

```
/* commandPosition.cc
 * a simple demo to move the robot to a desired position
 */

#include <libplayerc++/playerc++.h>
#include <iostream>
#include "args.h"

int main(int argc, char **argv)
{ parse_args(argc, argv);
  // we throw exceptions on creation if we fail
  try
  { using namespace PlayerCc;
    //connects to the robot/simulated robot
    PlayerClient robot(gHostname, gPort);
    //connects to the position sensor of the robot
```

```

//enables writing commands to the robot
Position2dProxy pp(&robot, gIndex);

//the desired position in x, y and angle of the robot
player_pose2d_t pos;
pos.px = 1.0; //x position in meters
pos.py = 0.0; //y position in meters
pos.pa = 0.0; // angle of the robot in radians

// the desired velocity in x, y and the angle of the robot
player_pose2d_t vel;
vel.px = 5.0; //x desired position in meters
vel.py = 0.0; //y desired position in meters
vel.pa = 0.0; //angle desired in radians

std::cout << robot << " robot" << std::endl;
pp.SetMotorEnable (true);

//infinite loop to show effects on the simulator/robot
for(;;)
{
    // this blocks until new data comes; 10Hz by default
    robot.Read();
    // writes the the robot the desired position and velocity
    pp.GoTo(pos, vel);
}
}
catch (PlayerCc::PlayerError e)
{
    std::cerr << e << std::endl;
    return -1;
}
}

```

### Using Stage (the simulation):

In one terminal:

```

$make commandPosition
$player simple.cfg

```

Stage should be started and the robot sitting still.

In the other terminal:

```

$ ./commandPosition

```

The robot starts at (0,3) x and y respectively with the robot facing 0 radians and moves to (1,0) with the robot facing 0 radians with a velocity of 5 meters per second in the x direction.

Using the real robot:

This code will not work on the robot. Since the robot does not know its starting position then it cannot move to the desired position.

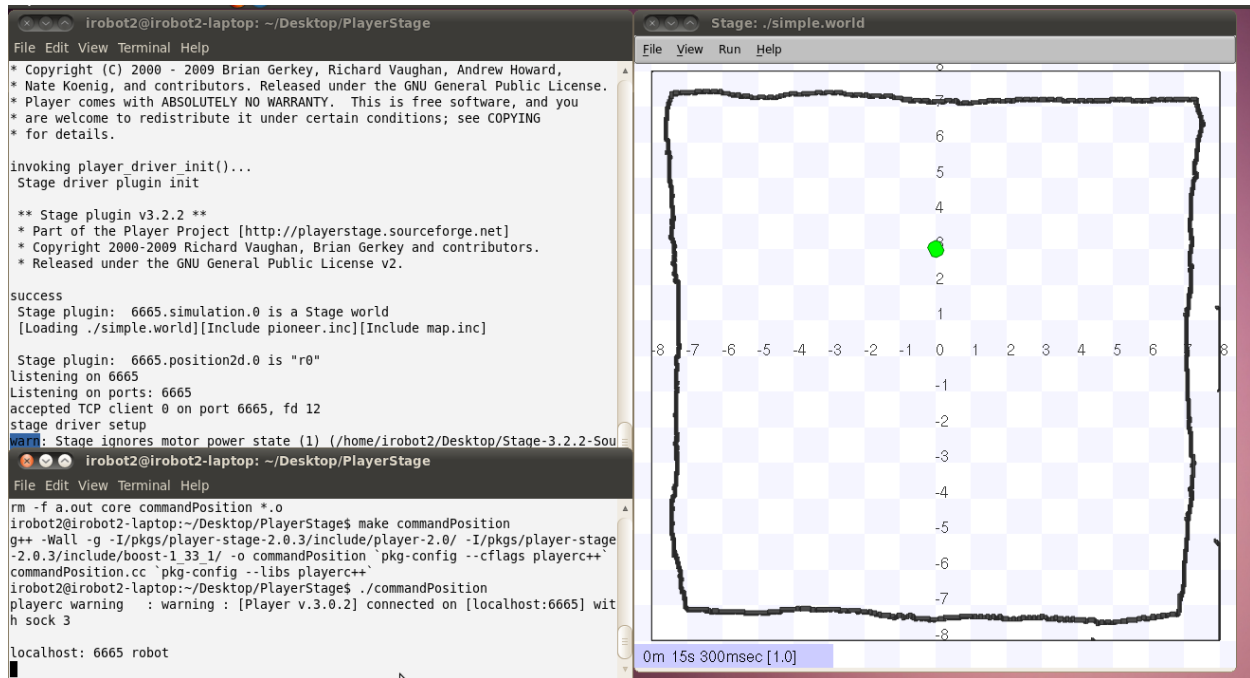


Figure 14- Command Position in Stage

```
pioneer2dx
(
  name "r0"
  color "green"
  pose [ 0 3 | 0 0 ]

  # report error-free position in world coordinates
  localization "gps"
  localization_origin [ 0 0 0 0 ]
)
```

Figure 15- Position of the robot starting at (0,3)

### 3.5 Installation Gazebo

The installation of Gazebo is a bit harder and requires more 3rd-party software than required by Stage. The software is listed below:

Geospatial Data Abstraction Library (GDAL) <http://www.remotesensing.org/gdal>



OpenDynamicsEngine (ODE) <http://opende.sourceforge.net>

Ogre3D <http://www.ogre3d.org>

Bullet <http://bulletphysics.org/wordpress/>

**GDAL** is library for geospatial data formats and is open source.

**ODE** is an open source, high performance library for simulating rigid body dynamics. It is fully featured, stable, mature and platform independent with an easy to use C/C++ API. It has advanced joint types and integrated collision detection with friction. ODE is useful for simulating vehicles, objects in virtual reality environments and virtual creatures. It is currently used in many computer games, 3D authoring tools and simulation tools.

**Ogre 3D** (Object-Oriented Graphics Rendering Engine) is a scene-oriented, flexible 3D engine written in C++ designed to make it easier and more intuitive for developers to produce applications utilizing hardware-accelerated 3D graphics.

**Bullet** is a Collision Detection and Rigid Body Dynamics Library.

ODE and Ogre3D are mandatory. Gazebo will not compile without ODE and Ogre 3D. Most of the models used come from using ODE and Ogre allows the models to be rendered in a 3D environment.

GDAL and Bullet are optional. GDAL is recommended in case you need to simulate terrain environments. Bullet is recommended when you need to detect collisions. For this guide, we stuck with the required libraries as Gazebo should work without optional libraries.

Ogre3D has one mandatory dependency and some optional dependencies which are described in the Ogre3D Readme file. Most of these dependencies would already be installed when the “.markings” files were downloaded and installed. But Freetype will have to be installed from the source from this website: <http://www.freetype.org>.

In order for Gazebo to work properly, we suggest to use the following versions of the dependencies: Ode-0.11.1, Ogre\_src\_v1-6-4, and Gazebo-0.9.0. Below is the order of installations that need to be performed so Gazebo works properly.

In order:

#### **Make freetype**

```
$ ./configure
$ make
$ sudo make install
```

#### **Make ODE 0.11.1**

```
$ ./configure --with-pic -enable-shared -disable-asserts
$ make
```

```
$ sudo make install
```

### Make Ogre v1.7.1

```
$ mkdir build
$ cd build
$ cmake ..
$ make
$ sudo make install
```

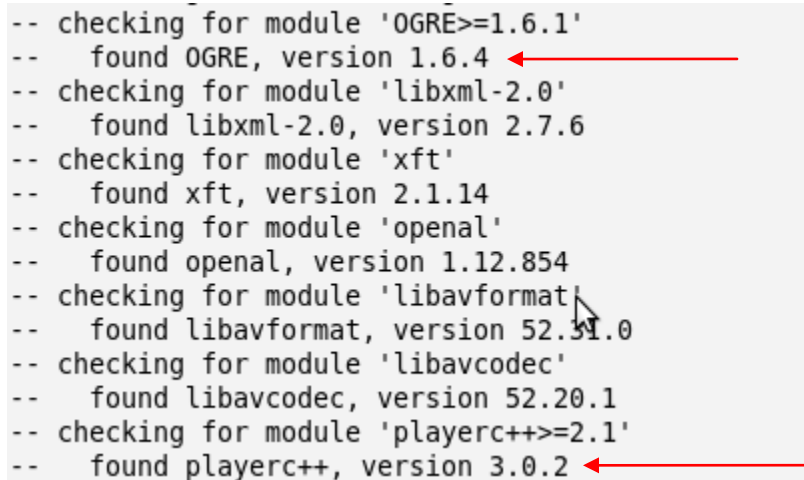
### Make Ogre v1.6.4<sup>3</sup>

```
$ ./bootstrap
$ ./configure
$ make
$ sudo make install
```

### Make Gazebo 0.9.0

```
$ mkdir build (inside the gazebo-trunk directory)
$ cd build
$ cmake ..
$ make
$ sudo make install
```

Make sure when `$ cmake ..` is finished that you check to see that Player and Ogre are found by looking at the list that is printed in the terminal.



```
-- checking for module 'OGRE>=1.6.1'
--   found OGRE, version 1.6.4
-- checking for module 'libxml-2.0'
--   found libxml-2.0, version 2.7.6
-- checking for module 'xft'
--   found xft, version 2.1.14
-- checking for module 'openal'
--   found openal, version 1.12.854
-- checking for module 'libavformat'
--   found libavformat, version 52.31.0
-- checking for module 'libavcodec'
--   found libavcodec, version 52.20.1
-- checking for module 'playerc++>=2.1'
--   found playerc++, version 3.0.2
```

Figure 16- Part of Gazebo Configuration Results

---

<sup>3</sup> Only if needed if Ogre v1.7.1 does not work. See Appendix A, Item 3

I tried to use Gazebo 0.10.0 before reverting to Gazebo 0.9.0 since the robot would not move in Gazebo 0.10.0. Appendix A is a list of fixes for common errors with Gazebo 0.10.0.

When Gazebo finishes with the installation, check to see if gazebo is install properly by typing:

```
$ gazebo
```

You should see a list of commands and options that could be used with gazebo.

### 3.5.1 Example of interaction with the simulation

The examples I use are the examples that came with the gazebo source code. I ran in to some errors which are explained in Appendix B.

To run the simulator, three terminal windows need to be open. First terminal window will run the command:

```
$ gazebo <world_file>
```

The second terminal window will run the player command:

```
$ player <.cfg file>
```

The third terminal window will run either playerv or the control code:

```
$ playerv
```

Or

```
$ ./<executable>
```

A simple example will be to move a pioneer2dx robot from the examples.

In the first terminal window:

```
$ cd gazebo-0.9.0/worlds
```

```
$ gazebo pioneer2dx.world
```

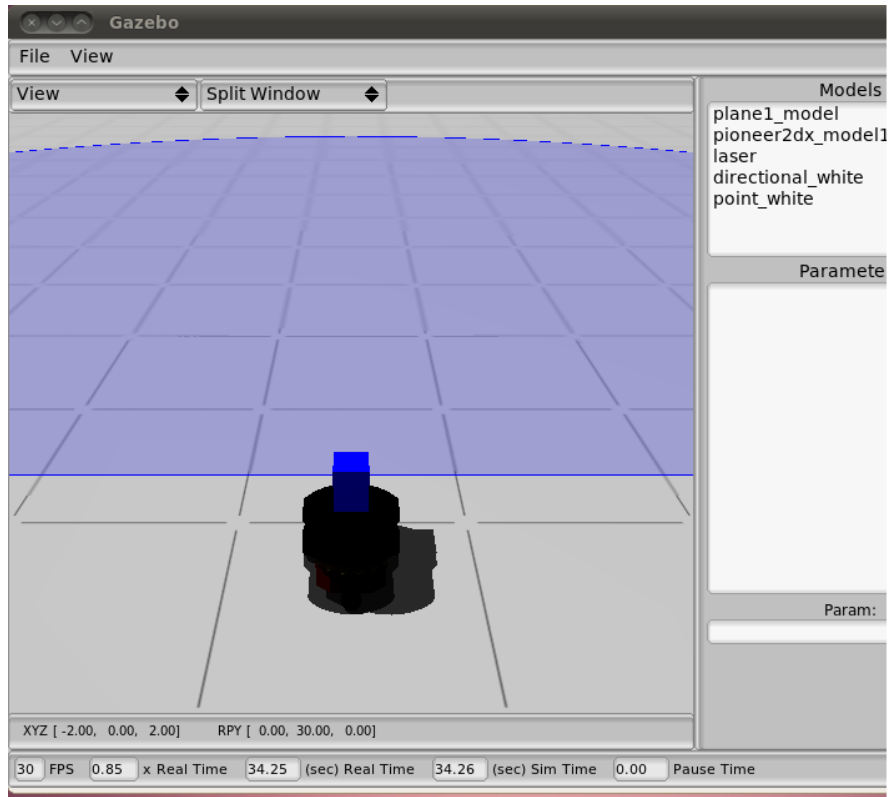


Figure 17- Gazebo with 1 Pioneer

In the second terminal window:

```
$ cd gazebo-0.9.0/player_cfgs
```

```
$ player gazebo.cfg (before running this command make sure you update the .cfg file
with the model_name in the gz_id field)
```

```
driver
(
  name "gazebo"
  provides ["position2d:0"]
  gz_id "pioneer2dxmodel1::position_iface_0"
)
```

Once player starts, wait until the message in the terminal says "Listening on port 6665" before \$  
playerv is used.

The third terminal window:

```
$ playerv
```

A small window should open up.

Click on Devices ->position2d:0(gazebo)->Subscribe

Devices->position2d:0(gazebo)->Command

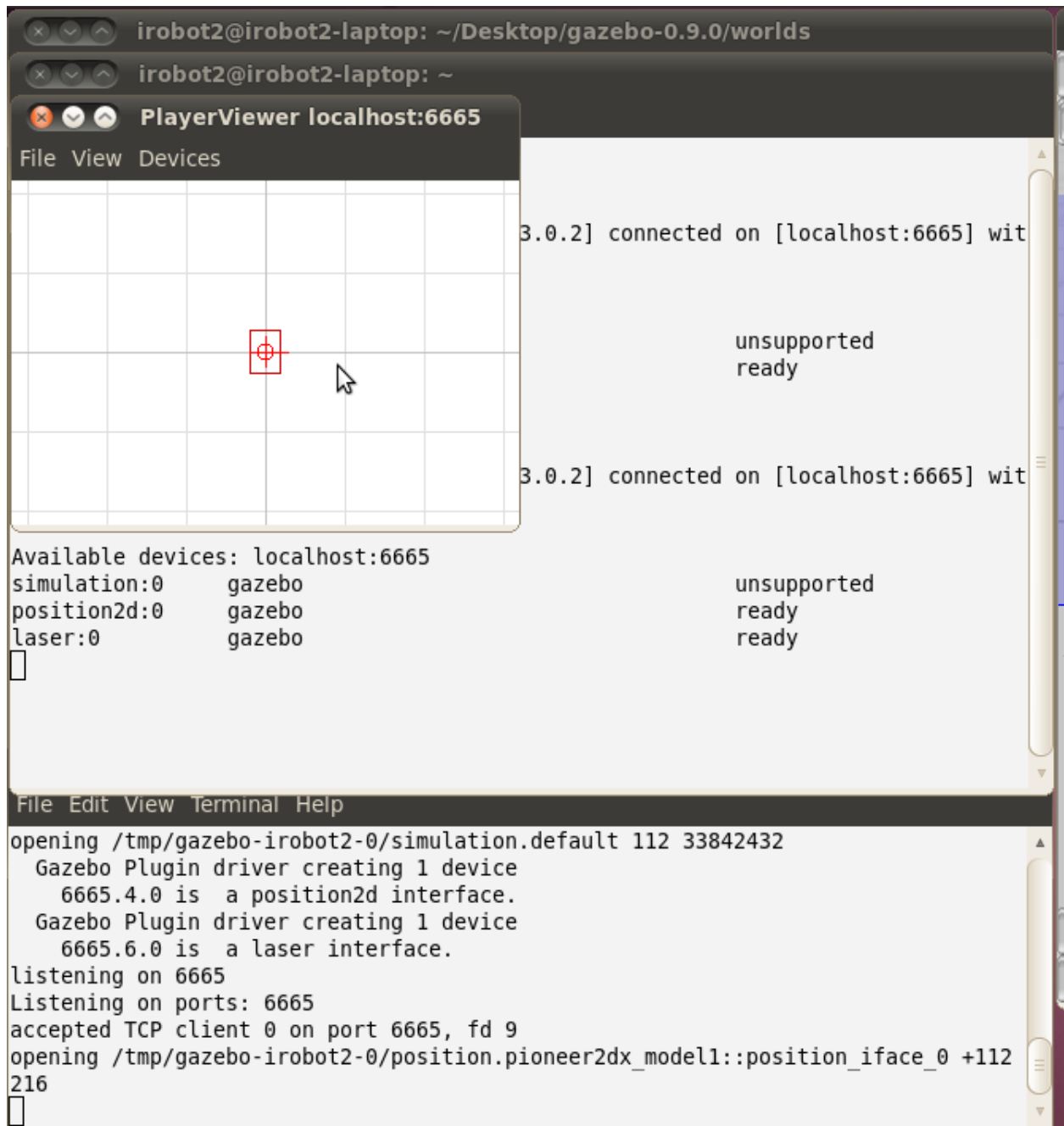


Figure 18- Player using gazebo.cfg and Playerv

The pioneer should move by moving the red cross-hairs in playerv. The robot has a lot of inertia built in due to the physics engine. To change the inertia of the robot, add the following lines to the pioneer2dx.model in the "chassis\_geom" section:

```
<geom:box name="chassis_geom">
  <xyz>0 0 0.0</xyz>
  <rpz>0.445 0.277 0.17</rpz>
  <mass>2.0</mass>
  <!-- add this part to keep the inertia low on the robot -->
  <kp>1000000000</kp>
  <kd>1.0</kd> <!-- Damping constraint -->
  <bounce>0.0</bounce> <!-- How bouncy the surface is -->
    <bounceVel>10</bounceVel> <!-- Max velocity before bounce is applied

  <!-- add this part to keep the inertia low on the robot -->

  <mul>1</mul>
```

### 3.5.1.1 Commanding in velocity

The control should be the same as the Player/Stage example. The difference between the examples for Player/Stage and for Gazebo is the makefile. Scons does work for 32bit systems. If you are using a 64 bit system then use the following makefile.

#### Makefile

```
# Example makefile for Player client programs in 64bit systems
#
# NOTE: Be sure you have LD_LIBRARY_PATH defined, to include
#       /usr/local/lib
#       (typically, this goes in your .cshrc file (or appropriate
#       variant
#       for the shell you are using) as "setenv LD_LIBRARY_PATH
#       /usr/local/lib")

PLAYERDIR = /pkgs/player-stage-2.0.3

CXX = g++
INCLUDES = -I$(PLAYERDIR)/include/player-3.0/ -
I$(PLAYERDIR)/include/boost-1_33_1/
CFLAGS = -Wall -g3 $(INCLUDES)

commandVelocity:
    $(CXX) $(CFLAGS) -o commandVelocity `pkg-config --cflags
playerc++` commandVelocity.cc `pkg-config --libs playerc++`
```

clean:

```
rm -f a.out core commandVelocity *.o
rm -f a.out core readPosition *.o
rm -f a.out core commandPosition *.o
```

readPosition:

```
$(CXX) $(CFLAGS) -o readPosition `pkg-config --cflags playerc++`
readPosition.cc `pkg-config --libs playerc++`
```

commandPosition:

```
$(CXX) $(CFLAGS) -o commandPosition `pkg-config --cflags
playerc++` commandPosition.cc `pkg-config --libs playerc++`
```

Copy this makefile in to: `$ gazebo-0.9.0/examples/player/position2d`

CommandVelocity.cc is the same file used in Stage. The same code should work for both Stage and Gazebo.

Once the files are unzipped on the desktop we can start to use the examples. Open 3 terminal windows. One will be for Gazebo, another for Player, and the last one will be for the control code.

Using Gazebo:

In one terminal:

```
$ cd gazebo/worlds
$ gazebo pioneero2dx.world
```

Gazebo should open a new window with 2 pioneers sitting back to back.

In the second terminal:

```
$ cd gazebo/player_cfgs
$ player gazebo.cfg
```

This will start up Player to pass information between Gazebo and the control code.

In the third terminal:

```
$ cd <move to where the commandVelocity.cc and Makefile are located>
$ make commandVelocity
$ ./commandVelocity
```

The robot should move forward with a constant velocity.

### 3.5.1.2 *Reading the robot position*

The readposition.cc code reads the x and y position of the robot while in motion. The wheels start at 0 x position and 0 y position. More information can be read such as the yaw angle, x velocity, y velocity, and yaw angular velocity.

readPosition.cc

```
/* readPosition.cc
 *
 * a simple demo to read the x and y position of the robot while
moving at a constant velocity
 */

#include <libplayerc++/playerc++.h>
#include <iostream>
#include "args.h"

int
main(int argc, char **argv)
{
    parse_args(argc,argv);

    // we throw exceptions on creation if we fail
    try
    {
        using namespace PlayerCc;

        //connects to the robot/simulated robot
        PlayerClient robot(gHostname, gPort);
        //connects to the position sensor of the robot
        //enables writing commands to the robot
        Position2dProxy pp(&robot, gIndex);

        std::cout << robot << " robot" << std::endl;

        pp.SetMotorEnable (true);

        double newspeed = 1.0; //robot velocity
        double newturnrate = 0.0; // turn velocity
```



```

//infinite loop to show effects on the simulator/robot
for(;;)
{
    /* this blocks until new data comes; 10Hz by default */
    robot.Read();

    // write commands to robot
    pp.SetSpeed(newspeed, newturnrate);

    //gets the x position of the robot
    double x = pp.GetXPos();
    //gets the y position of the robot
    double y = pp.GetYPos();

    // Prints the x and y position to the terminal
    std::cout << x << " x pos" << std::endl;
    std::cout << y << " y pos" << std::endl;

}

}
catch (PlayerCc::PlayerError e)
{
    std::cerr << e << std::endl;
    return -1;
}
}

```

**Using Gazebo:**

**In one terminal:**

```

$ cd gazebo/worlds
$ gazebo pioneero2dx.world

```

Gazebo should open a new window with 2 pioneers sitting back to back.

**In the second terminal:**

```

$ cd gazebo/player_cfgs
$ player gazebo.cfg

```

This will start up Player to pass information between Gazebo and the control code.

In the third terminal:

```
$ cd <move to where the readPosition.cc and Makefile are located>
$ make readPosition
$ ./readPosition
```

The robot should move forward with a constant velocity.

The robot should move forward with a constant velocity reading the x and y position. The robot starts at 0 x position and 0 y position.

### *3.5.1.3 Commanding in position*

There is not an example to move the robot to a desired position. The problem is that the robot does start near (0,0) but it is constantly increasing as the robot stands still.

## **4 Robot Operating System- ROS**

Robot Operating System (ROS) is an open-source operating system for robots. <http://www.ros.org/> ROS provides hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management as described on the ROS website.

On the Downloads page, there are several supported OS versions. For the purpose of this guide we are using Ubuntu.

### **4.1 Installation of ROS**

This is copied from <http://www.ros.org/wiki/cturtle/Installation/Ubuntu>. The installation process is more defined for ROS than for COIL or Player/Stage. ROS has almost no errors in the installation process.

#### **4.1.1 Setup your sources.list**

Setup your sources.list file to accept Debian packages from the ROS server.

**Ubuntu 10.04 (Lucid)**

- o `$ sudo sh -c 'echo "deb  
http://code.ros.org/packages/ros/ubuntu lucid main" >  
/etc/apt/sources.list.d/ros-latest.list'`

#### 4.1.2 Set up your keys

- `$ wget http://code.ros.org/packages/ros.key -O - | sudo apt-key add -`

#### 4.1.3 Installation

Make sure you have re-indexed the ROS.org server:

- `$ sudo apt-get update`

There are several different versions of ROS

Choose your preferred install:

- **ROS only:**
  - o `$ sudo apt-get install ros-cturtle-ros`
- **Base:** ROS plus robot-generic stacks (e.g. navigation, visualization)
  - o `$ sudo apt-get install ros-cturtle-base`
- **PR2:** ROS plus PR2-specific stacks, including PR2 simulator.
  - o `$ sudo apt-get install ros-cturtle-pr2`
  - o **Note: You will get a prompt about hddtemp:** you can safely answer no to the prompt if you are not installing on an actual PR2. To avoid getting the prompt, you can set the debconf selection ahead of time:
    - `$ echo "hddtemp hddtemp/daemon boolean false" | sudo debconf-set-selections`
- **PR2 All:** ROS plus PR2 and bleeding edge research/experimental stacks.
  - o `$ sudo apt-get install ros-cturtle-pr2all`
  - o **Note: You will get a prompt about hddtemp:** you can safely answer no to the prompt if you are not installing on an actual PR2. To avoid getting the prompt, you can set the debconf selection ahead of time:
    - `$ echo "hddtemp hddtemp/daemon boolean false" | sudo debconf-set-selections`

#### 4.1.4 Environment Setup

It's convenient if the ROS environment variables are automatically added to your bash session every time a new shell is launched:

```
$ echo "source /opt/ros/cturtle/setup.sh" >> ~/.bashrc
$. ~/.bashrc
```

If you just want to change the environment of your current shell, you can type:

```
$ source /opt/ros/cturtle/setup.sh
```

Once ROS is installed and works correctly. You can test by typing in the terminal:

```
$ rospack list
```

Rospack list should list out all the packages that are currently installed on your computer.

## 4.2 Example of interaction to the robot

A small test package, we used Brown University's ROS package for the iRobot Create. The link to the website for ROS package is located here: [http://code.google.com/p/brown-ros-pkg/wiki/irobot\\_create\\_2\\_1](http://code.google.com/p/brown-ros-pkg/wiki/irobot_create_2_1).

The list of downloads are located here: <http://code.google.com/p/brown-ros-pkg/downloads/list>. Download the irobot\_create\_2\_1.tar.bz2. Unpack on to your desktop. Before we can use this package, we need to install the drivers for py-serial.

### 4.2.1 Installation of Pyserial

PySerial can be found at <http://pyserial.sourceforge.net/>. PySerial is a module that encapsulates the access for the serial port. To download the ".tar" file go here: <http://pypi.python.org/pypi/pyserial> Unpack to the desktop. The instructions are <http://pyserial.sourceforge.net/pyserial.html#installation>.

In the terminal, type:

```
$ tar -xzvf pyserial-<version>.tar.gz
$ cd pyserial-<version>
$ python setup.py install      (for version 2.x)
or
$ python3 setup.py install (for version 3.x)
```

If you don't know what version to use, type python in the terminal and it should tell you the version.

### 4.2.2 Interaction with the robot

Once the python setup is finished, we can move on to controlling the iRobot Create with the irobot\_create\_2\_1 package. The information is on this website: [http://code.google.com/p/brown-ros-pkg/wiki/irobot\\_create\\_2\\_1](http://code.google.com/p/brown-ros-pkg/wiki/irobot_create_2_1)

```
$ cd ~/Desktop/irobot_create_2_1
$ rosmake irobot_create_2_1
$ export ROS_PACKAGE_PATH=<INSTALL_DIR>:$ROS_PACKAGE_PATH
```

Example, if you have the package on the desktop the path would be:

```
$ export ROS_PACKAGE_PATH=~/Desktop:$ROS_PACKAGE_PATH
```

In the first terminal:

```
$ roscore
```

In the second terminal:

```
$ rosrun irobot_create_2_1 driver.py
```

Note: Make sure the robot is connected and is turned on before running this command.

In another terminal, different from the other two:

```
$ rosservice list
```

This should show you a list of services currently running, such as:

```
/brake
/tank
/leds
/turn
```

These are described here: [http://code.google.com/p/brown-ros-pkg/wiki/irobot\\_create\\_2\\_1](http://code.google.com/p/brown-ros-pkg/wiki/irobot_create_2_1)

To control the robot,

```
$ rosservice call /brake 1 (to stop the robot)
$ rosservice call /turn 1 50 (this turns the robot to the right)
$ rosservice call /tank 1 50 50 (this makes the robot drive in a
straight line)
```

### 4.2.3 Commanding in position

The example used is with another package created just for the iRobot Create. This package does not use service commands. The control code is similar to the one used in the player/stage examples. The package is called `irobot_create`. The package can be found here: <http://cu-ros-pkg.googlecode.com/svn/trunk/icreate/>. This package has a lot of extras but for this guide, I have made another package using this code as a base and took out all the extras. This package does not contain a function to command the robot to go to a certain position and stop.

#### 4.2.4 Commanding in velocity

Before the irobot\_create package can be run, the package needs to be found by ROS. Since I do use the desktop more often than where the packages are created, I have to set the path every time by using this command in the terminal:

```
$ export ROS_PACKAGE_PATH=~/.catkin_ws/src:$ROS_PACKAGE_PATH
```

Sample Code:

Drives the robot in a straight line for 4 seconds, then stops.

drive\_straight.cpp

```
#include "iRobot_Create/irobot_create.h"

int main(int, char **)
{
    IRobotCreate robot; //creates an IRobotCreate object
    robot.Start(); //connects to the iRobot

    double x =0,y=0,th=0;
    double x_vel, w_vel;
    time_t sec;

    x_vel = 0.2;
    w_vel = 0.0;

    //updates the state of the robot, this needs to be called everytime
    you want something from the robot
    robot.updateStates();
    robot.getOdometricPos(x,y,th);//gets the current x, y, angle
    position
    cout <<"x= " << x << " y= " << y << " th = " << th << endl;// simple
    print function to the terminal window
    robot.setVelocity(x_vel,w_vel);//sets the robot's velocity with the
    parameters linear velocity and angular velocity

    sec = time(NULL);

    while (time(NULL) - sec < 4.0)
    {

    }
    robot.updateStates();
}
```

```

robot.getOdometricPos(x,y,th);
cout << "here2" <<endl;
cout <<"x= " << x << " y= "<< y << " th =" << th << endl;
robot.setVelocity(0.0,0.0);

return 0;
}

```

In the terminal window:

```

$ cd iRobot_Create (my simple version of irobot_create)
$ rosmake
$ ./drive_straight

```

The robot should go forward at a 0.2m/s for 4 seconds then stop.

#### 4.2.5 Reading the robot position

Reading the robot's position is done with 2 function calls. From the above code of drive\_straight.cpp:

```

robot.updateStates();
robot.getOdometricPos(x,y,th);

```

getOdometricsPos(x,y,th) returns the x, y, and angular position of the robot. The updateStates() function is needed each time you want to get the current position.

## Appendix A

Errors and Solutions encountered while installing Gazebo:

- 1) Make Gazebo 0.10.0
  - Error: recompile with -fPIC when running make of gazebo
  - You need to compile ode with -fPIC :  
./configure --with-pic -enable-shared
- 2) Gazebo 0.10.0 error at libavutil/common.h
  - Fix is to add the following lines right after #define AVUTIL\_COMMON\_H:

```
#ifdef __cplusplus
#define __STDC_CONSTANT_MACROS
#ifdef _STDINT_H
#undef _STDINT_H
#endif
#include <stdint.h>
#endif
```
- 3) Gazebo 0.10.0 cannot control the robot:
  - Do not use Gazebo 0.10.0 and Ogre 1.7.\*
    - Reason: Gazebo 0.10.0 has some problems with model names
    - Reason: Ogre 1.7.\* locks memory space and does not allow other programs to use that memory space.
    - Use Gazebo 0.9.0 and Ogre 1.6.4
- 4) Gazebo 0.10.0 low frame rate
  - Usually to deal with graphic cards
- 5) Gazebo (any version) error cannot find libode.so.1 or any lib
  - \$ sudo ldconfig
- 6) Gazebo (any version) screen flickers black when executing an example
  - System -> Preferences -> Apperances, Last tab (visual effects) select none.
- 7) Gazebo (any version) cannot find ogre path/ error during make
  - Make sure that the ogre files are in the usr/local/lib folder
  - Recompile gazebo, make install
- 8) Gazebo 0.9.0
  - Libplayerxdr/playerxdr.h not found during \$make
  - Comment out #include <libplayerxdr/playerxdr.h> in the following files:
    - LaserInterface.cc
    - FiducialInterface.cc
    - IRInterface.cc



## Appendix B

Common errors when running Gazebo:

- 1) Package not found in pkg-config search path (64bit systems only)
  - Work around: don't use scon, use a make file
- 2) ODE Internal Error \_dDnormalize4() in ode/odemath.h failed
  - Ode has to be compiled with -disable-asserts
  - Reason "bNormalizationResult" error is a result of the order of the bodies in some hinge2 joints in the packaged version of the simplecar model being reversed
  - This does not work for all examples.
- 3) Trying to subscribe to position and other sensors on the robot
  - I have not figured out the other sensors but I have for the position of the robot and to move the robot.
  - Example:

```
driver
(
  name "gazebo"
  provides ["position2d:0"]
  gz_id "pioneer2dxmodle1::position_iface_0"
)
```

In the gz\_id field you need to specify model\_name::interface\_name