

Summer School on Rescue Robots

8th International UJI Robotics School

Practical Session with USARSim and Player

**Antonio Morales
Javier Felip
Juan Carlos García**

**Robotic Intelligence Lab
Universitat Jaume I**

Introduction to USARSim

1. What is USARSim

USARSim was designed as a high fidelity simulation of urban search and rescue (USAR) robots and environments intended as a research tool for the study of human-robot interaction (HRI) and multirobot coordination. USARSim is designed as a simulation companion to the National Institute of Standards' (NIST) Reference Test Facility for Autonomous Mobile Robots for Urban Search and Rescue (Jacoff, et al. 2001). The NIST USAR Test Facility is a standardized disaster environment. The USAR task focuses on robot behaviours, and physical interaction with standardized but disorderly rubble filled environments. USARSim supports HRI by accurately rendering user interface elements (particularly camera video), accurately representing robot automation and behaviour, and accurately representing the remote environment that links the operator's awareness with the robot's behaviours.

High fidelity at low cost is made possible by building the simulation on top of a game engine. By offloading the most difficult aspects of simulation to a high volume commercial platform which provides superior visual rendering and physical modelling, our full effort can be devoted to the robotics-specific tasks of modelling platforms, control systems, sensors, interface tools and environments.

As a simulation user, you are expected to supply the user interfaces, automation, and coordination logic you wish to test. For debugging and development "Unreal spectators" can be used to provide egocentric (attached to the robot) or exocentric (third person) views of the simulation. A test control interface is provided for controlling robots manually.

Exercise 1

Start Unreal Tournament 2004 by running:
`$ wine ./PATH/ut2004-bin`

In the menu, choose the "Unirse a partida" option and then click on LAN tab to show the servers. Connect to the UJI server and explore the scenario. This is the spectator mode used to debug control programs and see what the robot does.

2. Controlling the robots: player

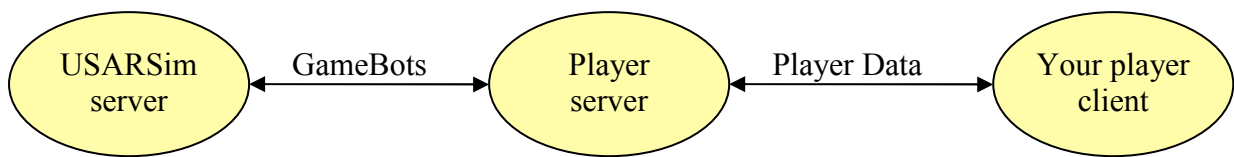
To control the robots in USARSim, you can program all the communications with the server using the GameBots protocol and implement the GUI or, on the other hand you can use middleware like Player, Pyro or MOAST that implement all the communications with USARSim allowing you to focus on programming the behaviour of the robots.

Player is a network server for robot control. Running on your robot, Player provides a clean and simple interface to the robot's sensors and actuators over the IP network. Your client program talks to Player over a TCP socket, reading

data from sensors, writing commands to actuators, and configuring devices on the fly.

Player supports a variety of robot hardware. The original Player platform is the ActivMedia Pioneer 2 family, but several other robots and many common sensors are supported. Player's modular architecture makes it easy to add support for new hardware, and an active user/developer community contributes new drivers. Player runs on Linux (PC and embedded), Solaris and *BSD.

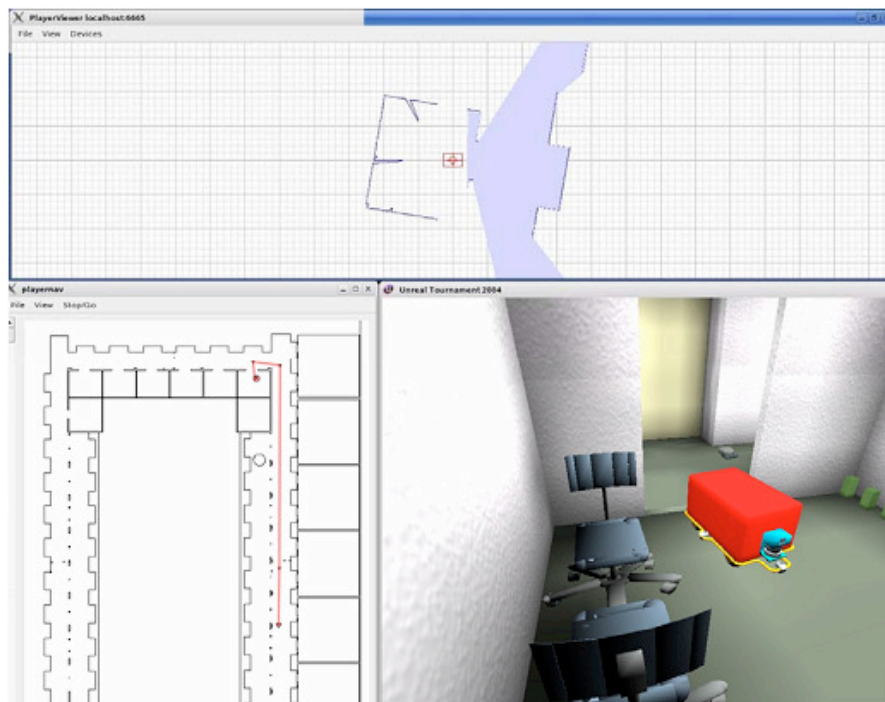
Using the player-usarsim driver, the player server can connect to the USARSim server and spawn a robot. When we connect to the player server, the driver converts all player commands to the GameBots protocol making the interaction between player and USARSim transparent to the programmer.



Any player program written for a real or simulated robot can be run using this player-usarsim driver, but there are some limitations. At the moment, the driver only implements few features of player:

Position2D Proxy (Only implements velocity control, position control could not be used)

- Laser Proxy
- Sonar Proxy



[Fig. 1] USARSim robot controlled through Player server

3. Programming a player client

To get data from sensors and send commands to motors, player uses a set of objects called proxies. These proxies connect automatically with the corresponding device on player server and allow the programmer to send commands by calling its methods.

Exercise 2: Run a player sample program over USARSim server

Open a terminal and type the following command to start player server
~\$ sh run_player.sh

When player server is connected to USARSim server, we can run a player client to control the robot spawned by the player server.

~\$./laserobstacleavoid

To see the behaviour of the robot, connect to the USARSim server with the Unreal Tournament client in spectator mode.

3.1. Moving the robot

The starting point to program with player is the following template and the documentation of player client that can be found at:

<http://playerstage.sourceforge.net/doc/Player-2.1.0/player/index.html>

```
1 #include <libplayerc++/playerc++.h>
2 #include <iostream.h>
3
4 using namespace PlayerCc;
5
6 //Configure port and hostname of player server
7 int port = 6665;
8 char* host = "localhost";
9
10int main( int argc, char* argv[] ) {
11    // Connect to Player Server
12    PlayerClient* pc = new PlayerClient( host, port );
13
14    // Proxy creation ZONE
15    //Position2dProxy* pp = new Position2dProxy( pc );
16    //SonarProxy* sp = new SonarProxy( pc );
17    //LaserProxy* lp = new LaserProxy( pc );
18
19    //Main control loop
20    while(1) {
21        // used to read in sensor data
22        pc->Read();
23
24        // YOUR CODE HERE //
25        // YOUR CODE HERE //
26        // YOUR CODE HERE //
27    }
28    return 0;
29}
```

To move the robot, you have to use the object `Position2dProxy`. Using the methods of this proxy you can move the robot in the simulator. For example, to move the robot forwards at 0.5m/s you should use the method: **pp→SetSpeed(0.5,0)** where the first parameter is the linear speed in m/s and the second parameter is the angular speed in rad/s.

Exercise 3

Make a program to move the robot with the following keyboard controls.

- W -> Move forward
- A -> Turn left
- S -> Stop
- D -> Turn right
- X -> Move backwards

3.2. Reading range sensors

To read the laser values you have to use the `[]` operator with the laser proxy. There is one value for each degree with a range from 0 to 180 degrees and starting from right to left.

For example, to show the distance to the left, front and right of the robot we can use the following code:

```
cout<< " Left: " << lp[179]
cout<< " Front: " << lp[90]
cout<< " Right: " << lp[0]
```

Exercise 4

Add obstacle avoidance when moving forward to exercise 3.

Exercise 5

Add a wall following mode to exercise 4.

4. References

Unreal Tournament 2004 official webpage

<http://www.unrealtournament2003.com/>

“UnrealEd3.0” Unreal scenario editor tutorial

http://architectonic.planetunreal.gamespy.com/first_level.html

Unreal Tournament about importing models from 3DStudio Max

<http://utforums.epicgames.com/showthread.php?p=25368400>

USARSim official website

<http://usarsim.sourceforge.net/>

USARSim project official website

<http://sourceforge.net/projects/usarsim>

MOAST Project official webpage

<http://sourceforge.net/projects/moast/>

Player Project official website

<http://playerstage.sourceforge.net/>

Player client online manual

<http://playerstage.sourceforge.net/doc/Player-2.1.0/player/index.html>

PyroBotics official webpage

<http://pyrorobotics.org/>

Robo Cup Rescue official website

<http://www.robocuprescue.org/>