המסלול האקדמי
המכללה למינהל

Introduction to Robotics
Lesson 3 – Player/Stage

Lecturer: Dr. Yehuda Elmaliah, Mr. Roi Yehoshua
Elmaliahy@colman.ac.il
roiyeho@gmail.com

Agenda

המסלול האקדמי
המכללה למינהל

- Player Server Architecture
- Stage Simulator
- Writing code for Player/Stage
- Working with sensors
- Obstacle avoidance

Player

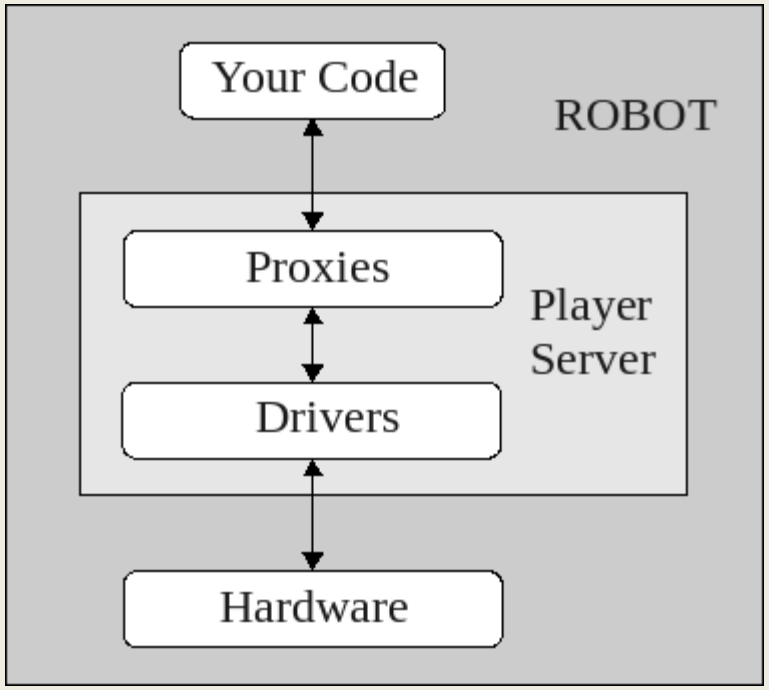
המסלול האקדמי
המכללה למינהל

- Player is a network server for robot control.
- One of the most popular open-source robot control interface in the world.
- 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 Architecture

המסלול האקדמי
המכללה למינהל

- The server/client control structure of Player when used on a robot.



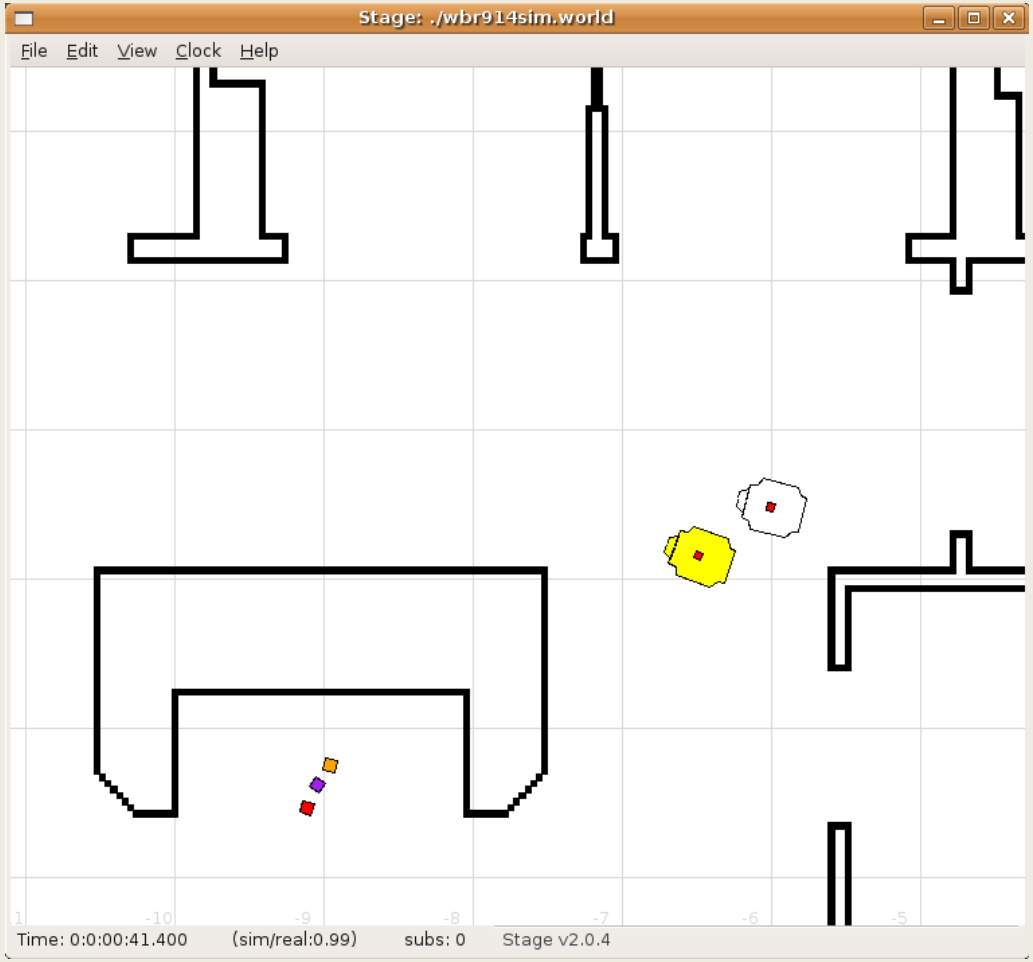
Stage

המסלול האקדמי
המכללה למינהל

- 2D Multi-Robot Simulator
- Stage simulates a population of mobile robots, sensors and objects in a 2D environment.
- Stage listens to what Player is telling it to do and turns these instructions into a simulation of your robot.
- It also simulates sensor data and sends this to Player which in turn makes the sensor data available to your code.

Stage

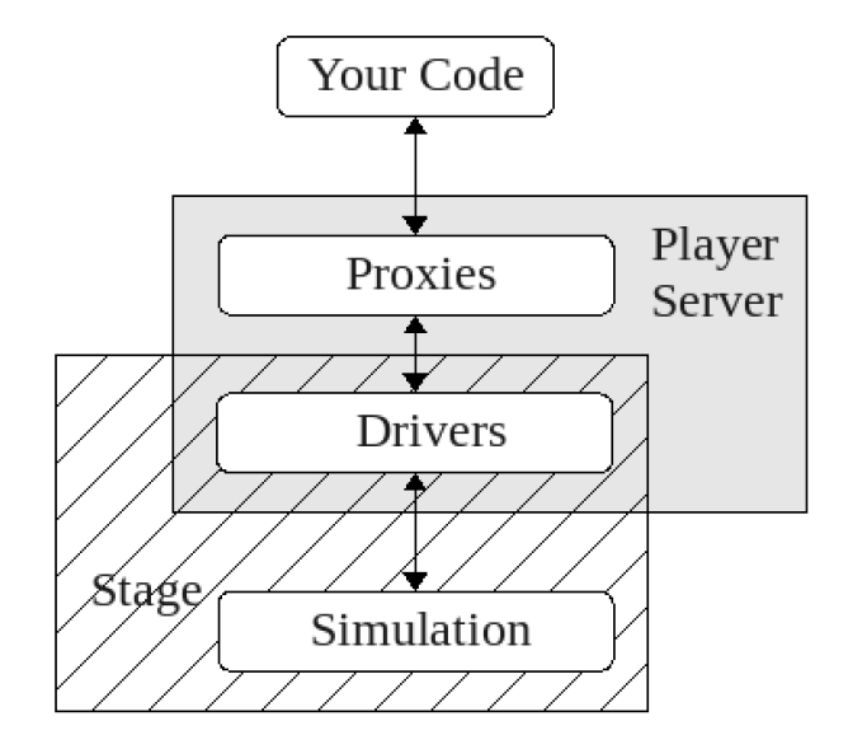
המסלול האקדמי
המכללה למינהל



Player/Stage Architecture

המסלול האקדמי
המכללה למינהל

- The server/client control structure of Player/Stage when used as a simulator.



Player Source Code

המסלול האקדמי
המכללה למינהל

http://playerstage.sourceforge.net/

Important File Types

המסלול האקדמי
המכללה למינהל

- There are 3 kinds of file that you need to understand to get going with Player/Stage:
  - a .world file
  - a .cfg (configuration) file
  - a .inc (include) file

World File

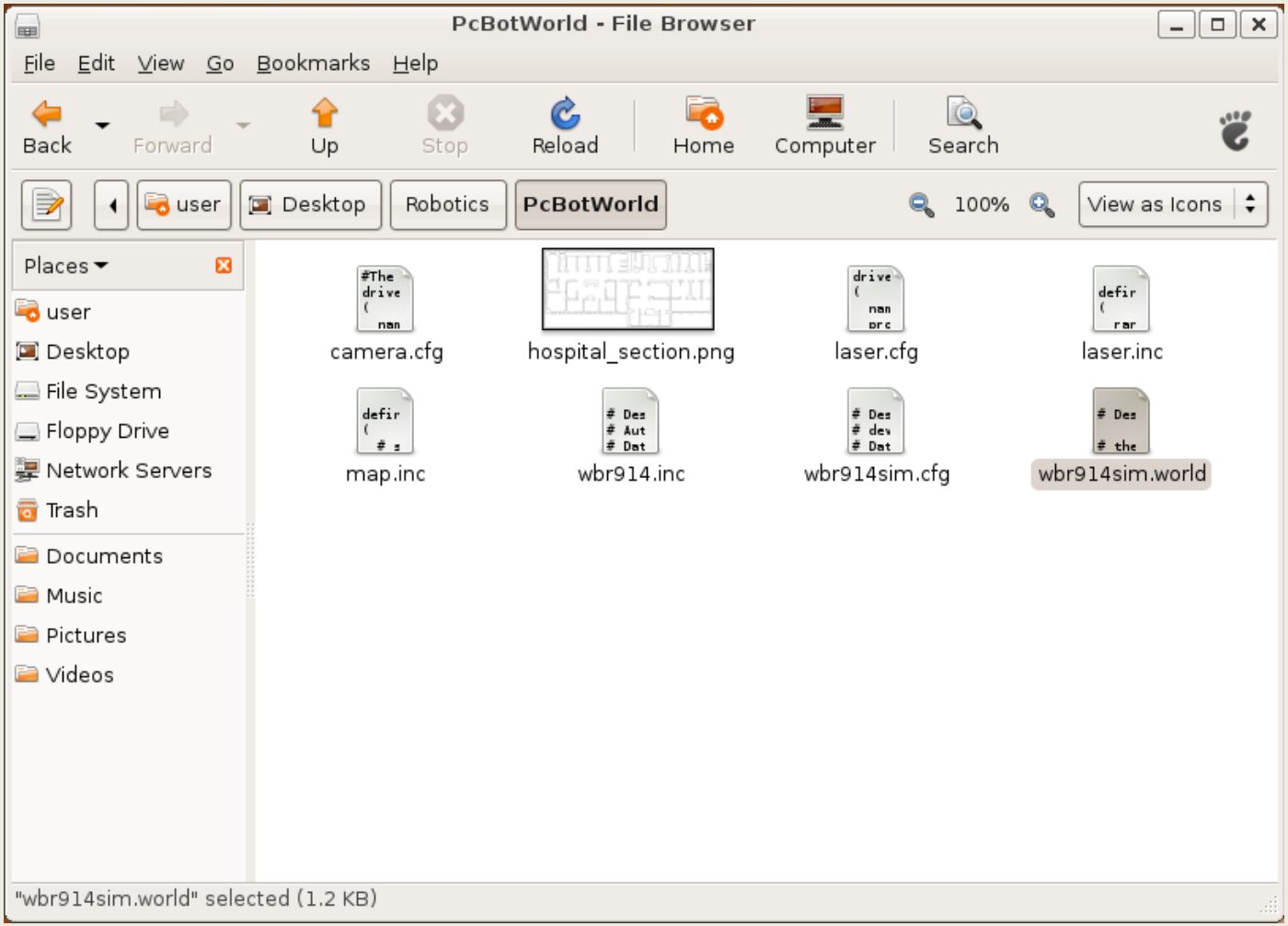
המסלול האקדמי
המכללה למינהל

- The .world file tells Player/Stage what things are available to put in the world.
- In this file you describe your robot, any items which populate the world and the world's layout.
- You can find PC-bot world file in your VM at the directory:

  ~/Desktop/Robotics/PcBotWorld/wbr914sim.world

World File

המסלול האקדמי
המכללה למינהל



PC-Bot World File Part (1)

המסלול האקדמי
המכללה למינהל

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

# the size of a pixel in Stage's underlying raytrace model in meters
resolution     0.2

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

# 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 ]
```

PC-Bot World File Part (2)

המסלול האקדמי
המכללה למינהל

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

map(
  bitmap "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
)
```

PC-Bot World File Part (3)

המסלול האקדמי
המכללה למינהל

```
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" )

wbr914
(
  color "white"
  name "wbr914_1"
  pose [-6.009 2.481 -194.220]
)

wbr914
(
  color "yellow"
  name "wbr914_2"
  pose [-6.492 2.156 -199.781]
)
```

Include File

המסלול האקדמי
המכללה למינהל

- The .inc file follows the same syntax and format of a .world file but it can be included.
- Usually the model of your robot will be defined in a .inc file
- That way if you ever want to change your robot description then you only need to do it in one place

PC-Bot Model File

המסלול האקדמי
המכללה למינהל

- Your PC-Bot model include file is located at:

  ~/Desktop/Robotics/PcBotWorld/wbr914sim.world

PC-Bot Model File Part (1)

המסלול האקדמי
המכללה למינהל

```
# Desc: Device definitions for Whitebox Robotics PC-BOT 914 robot.

# 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 ]
```

PC-Bot Model File Part (2)

המסלול האקדמי
המכללה למינהל

```
 # 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]
)
```

PC-Bot Model File Part (3)

המסלול האקדמי
המכללה למינהל

```
# 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
```

PC-Bot Model File Part (4)

המסלול האקדמי
המכללה למינהל

```
# 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[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
  # laser_return 0

  # use the IR array defined above
  wbr914_ir()
  hokuyo_URG_laser( pose [ 0.0 0.0 0.0 ])
)
```

Config File

המסלול האקדמי
המכללה למינהל

- The .cfg file is what Player reads to get all the information about your robot
- It tells Player which drivers it needs to use in order to interact with the robot
  - If you're using a real robot these drivers are built into Player
  - If you’re using a simulation, the driver is always Stage (this is how Player uses Stage in the same way it uses a robot: it thinks that it is a hardware driver and communicates with it as such).

Config File

המסלול האקדמי
המכללה למינהל

- The .cfg file tells Player which devices are attached to it and how to interpret any data from the drivers so that it can be presented to your code.
- Items described in the .world file should be also described in the .cfg file if you want your code to be able to interact with that item
- If you don't need your code to interact with the item then this isn't necessary.

PC-Bot Config File

המסלול האקדמי
המכללה למינהל

```
# Desc: PC-Bot Player sample configuration file for controlling Stage devices
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" )

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

driver(
 name "stage"
 provides ["6666:position2d:0" "6666:sonar:0" "6666:laser:0"]
 model "wbr914_2"
)
```

Interfaces, Derivers and Devices

המסלול האקדמי
המכללה למינהל

- Drivers are pieces of code that talk directly to hardware.
  - There are specific drivers to each piece of hardware, e.g. a laser driver will be different from a camera driver, or different drivers for different brands of laser.
- Interfaces are a standard way for a driver to send and receive information from Player.
  - They specify the syntax and semantics of how drivers and Player interact.
- A device is a driver that is bound to an interface so that Player can talk to it directly.
  - For example, the sicklms200 driver can be bound to the laser interface to create a device

Devices Addresses

המסלול האקדמי
המכללה למינהל

- Each device has a unique address, for example **localhost:6665:laser:0**
- The fields in the address are:

  host:robot:interface:index
  - The host and robot fields (default localhost and 6665) indicate where the device is located.
  - The interface field indicates which interface the device supports, and thus how it can be used.
  - The index field allows you to pick among the devices that support the given interface and are located on the given host:robot (e.g., if the same robot has more than one laser)

Run Simulation

המסלול האקדמי
המכללה למינהל

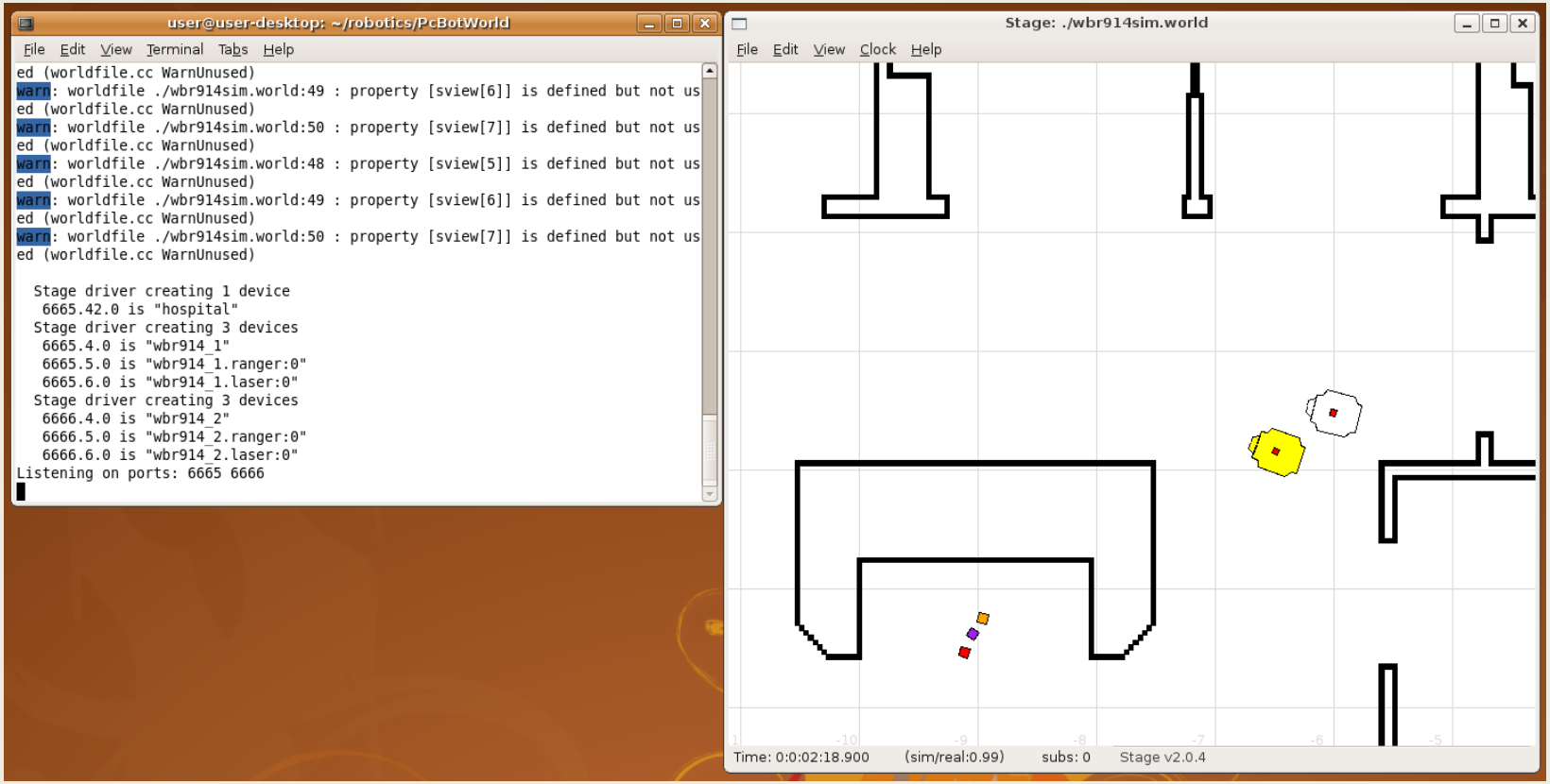
- To run Player with the PC-Bot simulation, type the following commands in the terminal:

```
$ cd robotics/PcBotWorld
$ player wbr914sim.cfg
```

- This starts up the Player server and connects all the necessary hardware devices to the server

Run Simulation

המסלול האקדמי
המכללה למינהל



Writing Code for Player

המסלול האקדמי
המכללה למינהל

- Player is compatible with C, C++ or Python code
- We will only really be using C++ because it is the simplest to understand.
- **libplayerc++** is a C++ client library for the player server
- The C++ library is built on a "service proxy" model in which the client maintains local objects that are proxies for remote services.

libplayerc++ Header File

המסלול האקדמי
המכללה למינהל

- The first thing to do within your code is to include the Player header file:

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

Main Classes

המסלול האקדמי
המכללה למינהל

- The core of libplayerc++ is based around the following classes
  - PlayerCc::PlayerClient
  - PlayerCc::ClientProxy
  - PlayerCc::PlayerError

PlayerClient

המסלול האקדמי
המכללה למינהל

- The PlayerClient is used for communicating with the player server
- One PlayerClient object is used to control each connection to a Player server
- This object contains methods for obtaining access to devices

PlayerClient

המסלול האקדמי
המכללה למינהל

- Creating a new PlayerClient object:

```
PlayerClient pc("localhost", 6665);
```

- First parameter is the host name
  - If your code is running on the same computer (or robot) as the Player server then the hostname is localhost, otherwise it will be the IP address of the computer or robot.
- Second parameter is the port number
  - This is an optional parameter. It is only useful if your simulation has more than one robot in and you need your code to connect to both robots.

PlayerClient

המסלול האקדמי
המכללה למינהל

- If you are using only one robot and your code runs on the same machine as the Player server, you can use the default c'tor:

```
PlayerClient pc();
```

Proxy Classes

המסלול האקדמי
המכללה למינהל

- Access to a device is provided by a device-specific proxy class
- Once we have established a PlayerClient we can connect our code to device proxies so that we can exchange information with them.
- The proxies that you can connect to are dependent on what you have put in your .config file.

Proxy Classes

המסלול האקדמי
המכללה למינהל

- There are about 40 different proxies which you can choose to use
- They all inherit from the base class ClientProxy
- Proxies take the name of the interface which the drivers use to talk to Player, e.g.:
  - SonaryProxy
  - LaserProxy
  - Position2dProxy
- Each proxy will have different commands depending on what it controls

Position2dProxy

המסלול האקדמי
המכללה למינהל

- The Position2dProxy is the number one most useful proxy there is.
- It controls the robot's motors and keeps track of the robot's odometry (where the robot thinks it is based on how far its wheels have moved).

SetSpeed

המסלול האקדמי
המכללה למינהל

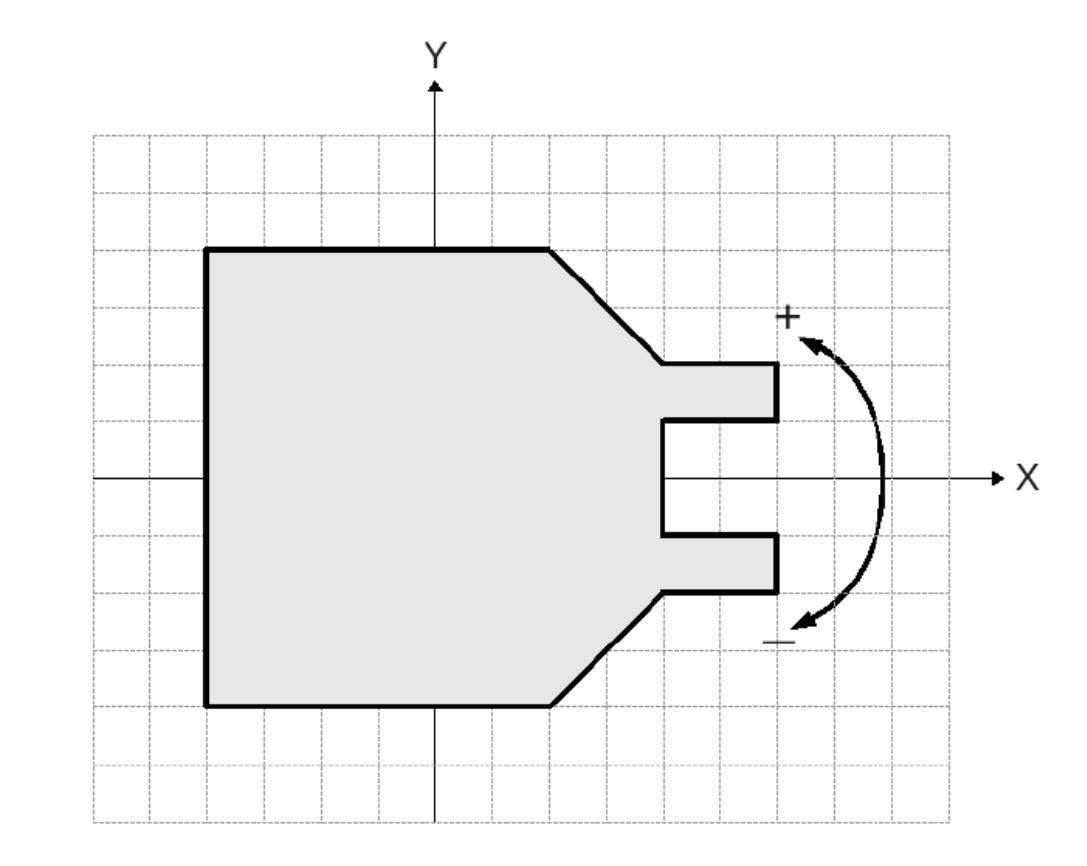
- The SetSpeed command is used to tell the robot's motors how fast to turn.
- For differential robots, use the following:

```
SetSpeed(double XSpeed, double YawSpeed)
```

  - The x speed is the rate at which the robot moves forward in meters per second
  - The yaw speed controls how fast the robot is turning and is given in radians per second
    - You can use a built-in global function dtor() to convert degrees into radians

Position2dProxy

המסלול האקדמי
המכללה למינהל



GetSpeed

המסלול האקדמי
המכללה למינהל

- The GetSpeed commands return the current speed relative to the robot
  - **GetXSpeed**: forward speed (metres/sec)
  - **GetYawSpeed**: turning speed (radians/sec).

Get_Pos

המסלול האקדמי
המכללה למינהל

- These functions allow you to monitor where the robot thinks it is based on the robot's odometry.
- **GetXPos()**: gives current x coordinate relative to its x starting position.
- **GetYPos()**: gives current y coordinate relative to its y starting position.
- **GetYaw():** gives current yaw relative to its starting yaw.

SetMotorEnable()

המסלול האקדמי
המכללה למינהל

- This function takes a boolean input, telling Player whether to enable the motors or not.
- If you are using Player/Stage, then the motors will always be enabled and this command doesn't need to be run.
- However, if your code is moved onto a real robot you need to explicitly enable the motors in your code.

Read()

המסלול האקדמי
המכללה למינהל

- To make the proxies update with new sensor data we need to tell the player server to update.
- All we have to do is run the command pc.Read() every time the data needs updating (where pc is the name you gave the PlayerClient object).
- Until this command is run, the proxies and any sensor information from them will be empty.

Moving the Robot Forward

המסלול האקדמי
המכללה למינהל

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

using namespace PlayerCc;

int main()
{
    PlayerClient pc;
    Position2dProxy pp(&pc);

    pp.SetMotorEnable(true);

    // Move the robot forward for 2 seconds
    for (int i = 1; i < 20; i++) {
        // Read from the proxies
        pc.Read();

        // Command the motors
        pp.SetSpeed(0.2, 0);
    }
    return 0;
}
```

SonarProxy

המסלול האקדמי
המכללה למינהל

- The sonar proxy can be used to receive the distance from the sonar to an obstacle in meters.
- To do this you use the command:

```
sp[sonar_number]
```

  - where sp is the SonarProxy object and sonar_number is the number of the ranger
  - The sonar numbers come from the order in which you described the ranger devices in the worldfile
- If no obstacle is detected then the function will return whatever the ranger's maximum range is.

Obstacle Avoidance

המסלול האקדמי
המכללה למינהל

- Now we will write a function that checks the sonars for any obstacles and amends the motor speeds accordingly.

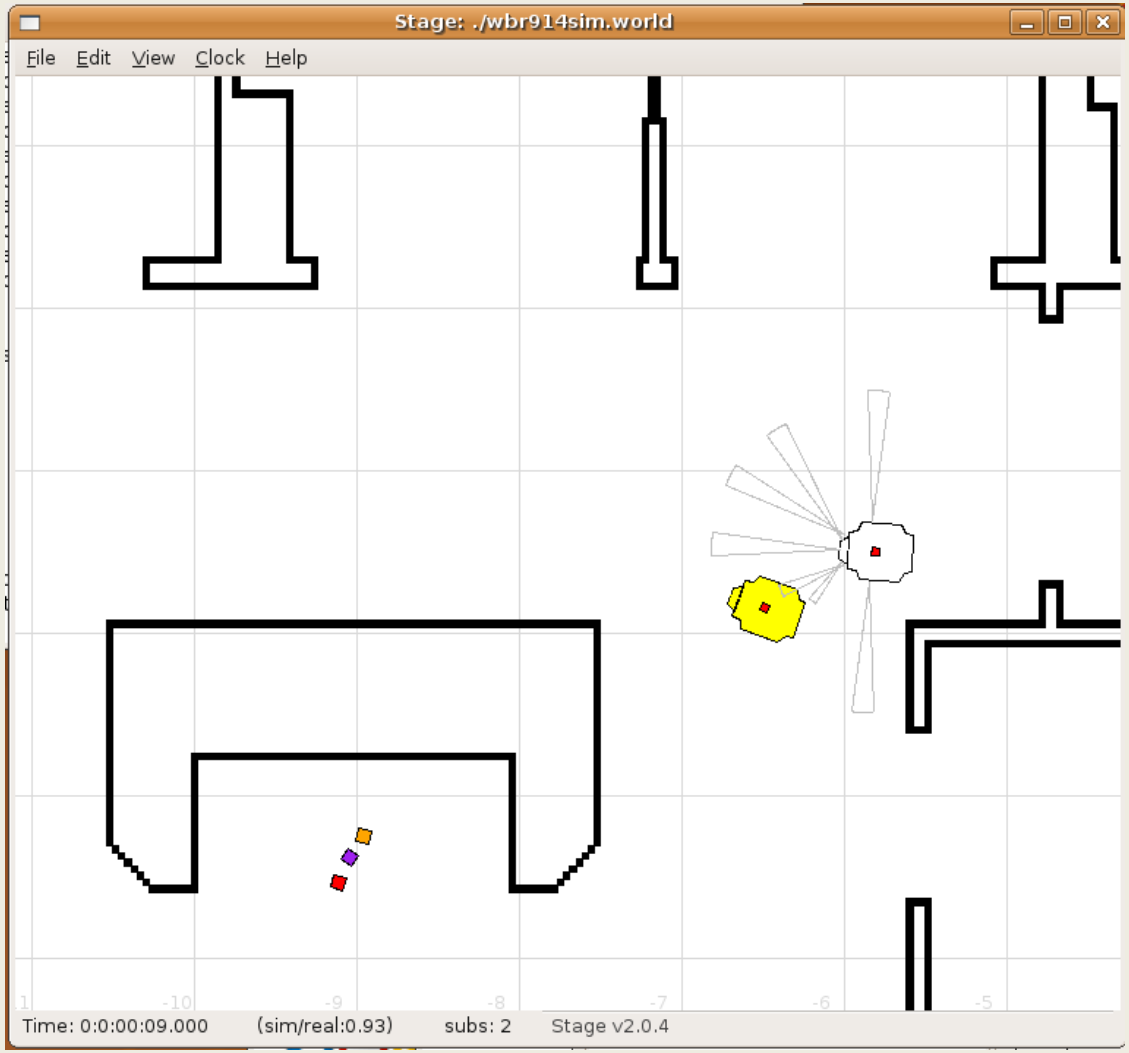
Obstacle Avoidance

המסלול האקדמי
המכללה למינהל

```
int main()
{
    PlayerClient pc;
    SonarProxy sp(&pc);
    Position2dProxy pp(&pc);

    while (true) {
        pc.Read();
        cout << sp << endl;

        if (sp[0] < 0.4)
            pp.SetSpeed(0, 0.5);
        else if (sp[0] < 0.7)
            pp.SetSpeed(0.3, 0.3);
        else
            pp.SetSpeed(0.5, 0);
    }

    return 0;
}
```

Obstacle Avoidance

המסלול האקדמי
המכללה למינהל



Exercise

המסלול האקדמי
המכללה למינהל

- Improve the obstacle avoidance code to reduce the cases where the robot gets stuck in objects