Robotics
Guidelines for student projects

Marco Della Vedova
<marco.dellavedova@unipv.it>

Monday 6th October, 2014

13:51

http://robot.unipv.it/toolleeo

Overview

develop a virtual mobile robot in the CPSS simulator

Overview

develop a virtual mobile robot in the CPSS simulator

(and get up to 5 extra-points in the final grade)

The simulator

CPSS Simulator

CPSS = Cyber-Physical System Simulator

- it is an open source (GPL v2) simulator for mobile robots
  `http://sourceforge.net/projects/cpss/`
- written in C++ with Qt
- developed at the Universidade de Aveiro (PT)
- used for the CiberRato and CyberMouse competitions
  `http://microrato.ua.pt`

CPSS introduction

The simulation system creates a **virtual arena**, populated by **obstacles**, where a starting grid, a **target area**, signaled by a **beacon**, and a home area are integrated.

Participants must provide the software agents which control the movement of the virtual robots, in order to accomplish the competition goal.

The simulation system creates a **virtual arena**, populated by **obstacles**, where a starting grid, a **target area**, signaled by a **beacon**, and a home area are integrated.

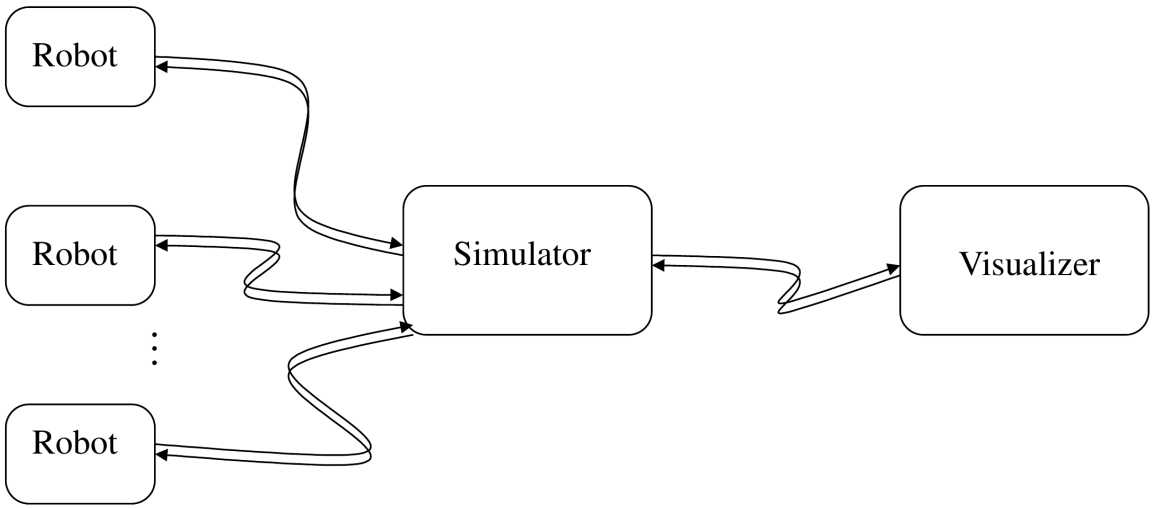
Participants must provide the software agents which control the movement of the virtual robots, in order to accomplish the competition goal.

Robots are supposed to be **mice** seeking cheese!

CPSS's mascot

CPSS software architecture

CPSS is characterized by a modular architecture, in which the simulator, the visualizer and robots are separated processes that communicate via socket.



Simulator process

The simulator is responsible for:

- Implementing the virtual bodies of the robots.
- Estimating sensor measurements and sending them to the corresponding agent.
- Moving robots within the arena, according to orders received from corresponding agent and taking into account environment restrictions. For instance, a robot can not move over an obstacle.
- Routing the messages sent by robots, taking into account communication constraints.
- Updating robot score, taking into account the fulfilled goals and applied penalties.
- Sending scores and robots positions to the visualizer.
- Making available a control panel to start/restart and stop the competition.

Visualizer process and robot processes

The visualizer is responsible for:

- Graphically showing robots in competition arena, including their states and scores.
- Making available a control panel to start/restart and stop the competition.

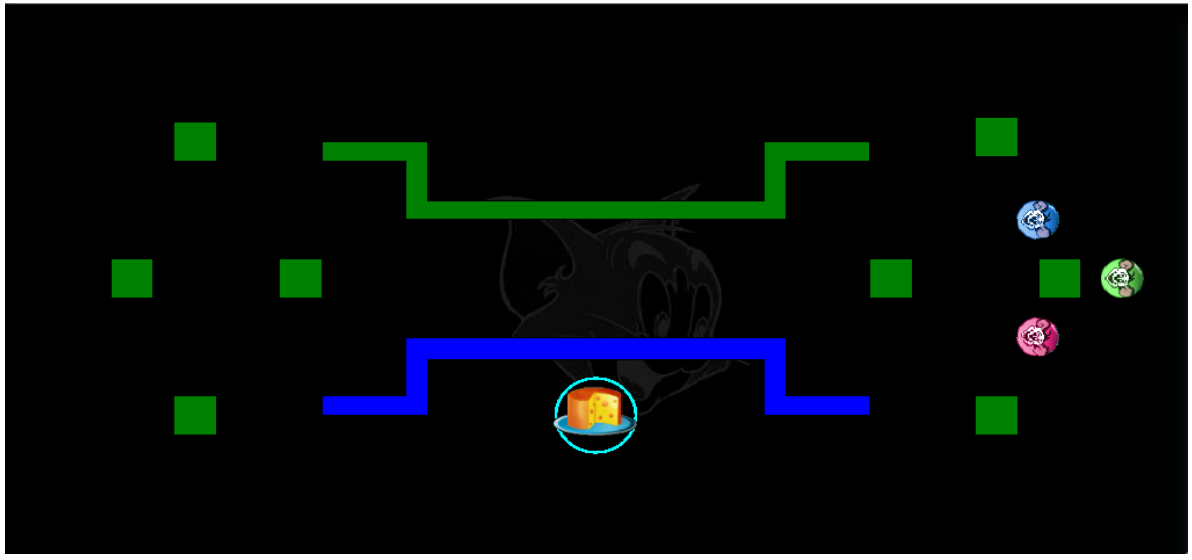
Robot processes implements the control logic of the mobile robots. One process for each robot.

Simulation notes

- The simulation system is discrete and time-driven.
- In each time step the simulator sends sensor measurements to agents, receives actuating orders, applies them, and updates scores.
- The cycle time is 50 milliseconds.
- All time intervals are measured as multiples of the cycle time. We denote u_t our unit of time, representing the cycle time.
- All elements into play, namely arena, obstacles, and robots, are virtual, thus there is no need for a real length unit. Hence, we use u_m as the unit of length.

Arena

The game scenario is composed of a rectangular arena, outer delimited, with obstacles, a target area, a home area and a starting grid inside. Obstacles are elements placed within the arena to hamper the robot movements; they can, for instance, represent walls, watercourses, or trenches. The target area is a circle with a beacon in its center. The starting grid defines the robots initial positions. The home area is a circle, without a beacon in its center, centered in grid position number 1.



Obstacles

- All obstacles have planar surfaces, at least 0.3 um wide.
- All corners have angles ranging from 90 to 270 degrees.
- Some obstacles can be higher than the beacon, defining shadow zones.
- Any passage between obstacles is at least 1.5 um wide.

Home area

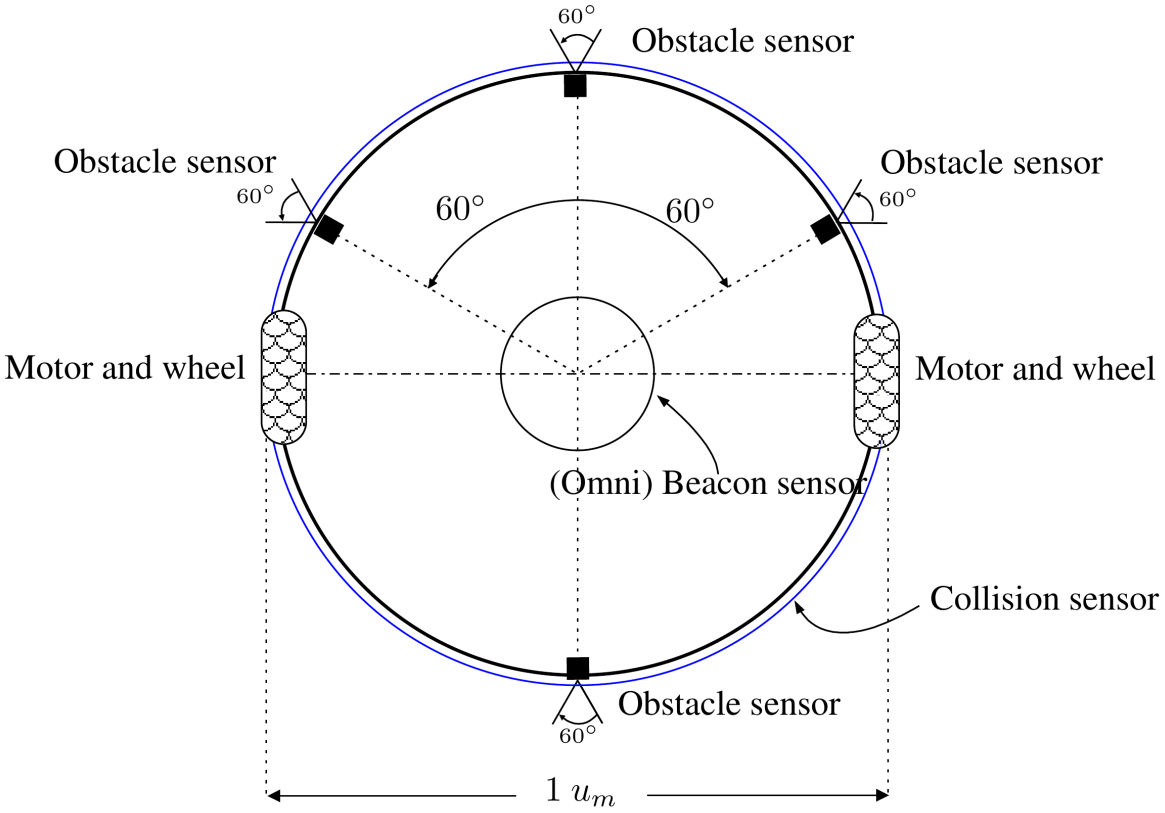
- The home area radius is at least 2.0 um wide.
- The home area is centered in starting position number 1.

Target area

- The target area radius is at least 2.0 um wide.
- There is a beacon in the center of the target area.
- The beacon does not act as an obstacle to robot movement.
- The beacon can be not visible.

Robot

Bodies of the virtual robots have a circular shape, $1_u m$ wide, and are equipped with sensors, actuators and command buttons.



Robot - sensors

Sensor elements in each robot include: 4 obstacle sensors, 1 beacon sensor, 1 compass, 1 bumper (collision sensor), 1 ground sensor, and a GPS.

Sensor models try to represent real devices. Thus, on one side, their **measures are noisy**. On the other side, the reading of sensors is affected by n time units **latency**, where n depends on the particular sensor. This means that the respective values are about n simulation cycles old, that is, when an agent receives a value it represents a measure performed n cycles ago.

Obstacle sensors measure distances between the robot and its surrounding obstacles, including other robots. They can be put in any place in the robot periphery. Each sensor has a 60 degrees aperture angle. The measure is inversely proportional to the lowest distance to the detected obstacles, and ranges between 0.0 e 100.0, with a resolution of 0.1.

The beacon sensor is positioned in the center of the robot, and has an omnidireccional covering. It measures the angular position of the beacon with respect to the robot's frontal axis. The measure ranges from -180 to +180 degrees, with a resolution of 1 degree. A beacon is not detected by the beacon sensor, if: (i) there is a high obstacle between it and the sensor; (ii) the distance from it to the sensor is higher than 5 u_m.

The compass is positioned in the center of the robot and measures its angular position with respect to the virtual North. We assume the X axis is facing the virtual North. Its measures range from -180 to +180 degrees, with a 1 degree resolution.

The GPS is a device that returns the position of the robot in the arena, with resolution 0.1. It is located at the robot center.

The bumper corresponds to a ring put around the robot. It acts as a boolean variable enabled whenever there is a collision.

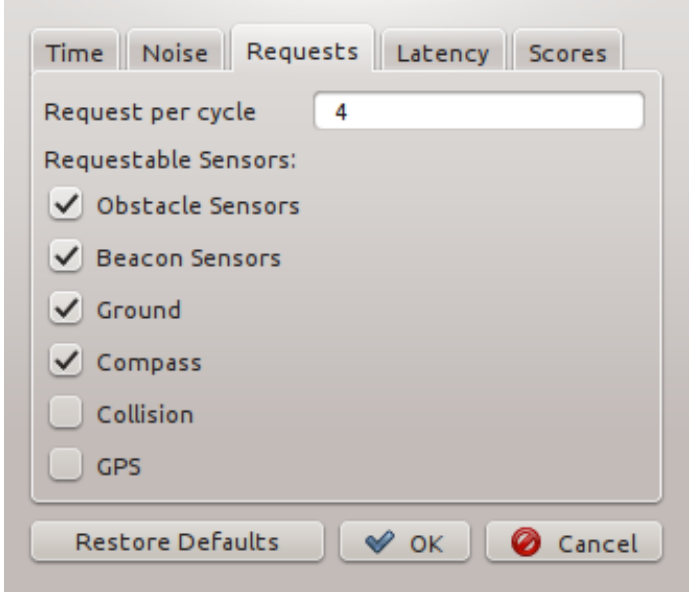
The ground sensor is a device that detects if the robot is completely over the target area or the home area.

Robot - sensors request

The simulation is configurable such that each type of sensor can be:

- always available, or
- available on request.

Moreover, a limit can be set for the number of sensor requests per simulation cycle (e.g., a maximum of 4 measurements each cycle).



Robot - actuators

The virtual robot has 2 motors and 3 signaling leds (lights). The motors try to represent, although roughly, real motors. Thus, they have inertia and noise.

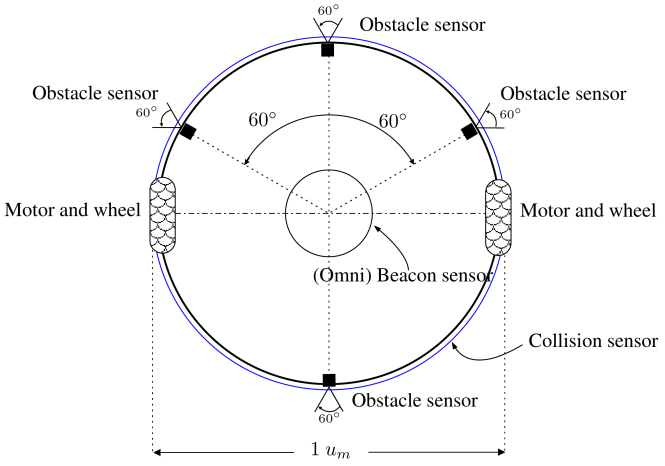
The 3 leds are named

- visiting led,
- returning led, and
- end led.

They are used to signal the attainment of goals. The way they must be used depends on the competition challenge.

Robot - actuators

The 2 motors drive two wheels. Robot movement depends on the power applied to the two motors. Both translational and rotational movements are possible. If the same power values are applied to both motors the robot moves along its frontal axis. If the power values are symmetric the robot rotates. The power accepted by motors ranges between -0.15 e +0.15, with resolution 0.001. However this is not the power applied to wheels because of inertia and noise. A power order applied to a motor keeps in effect until a new order is given. For instance, if an agent applies a given power to a motor at a given time step, that power will be continuously applied in the following time steps until a new power order is sent by the agent.



Robot - motor inertia and noise

Movement depends on power applied to wheels. This power differs from power order sent by agents because of motor inertia and noise. The relation between both is given by

$$\begin{aligned} lOutPow(t) &= (lOutPow(t-1) + lInPow(t))/2 \\ rOutPow(t) &= (rOutPow(t-1) + rInPow(t))/2 \\ lNoisyOutPow(t) &= lOutPow(t) \cdot lNoise(t) \\ rNoisyOutPow(t) &= rOutPow(t) \times rNoise(t) \end{aligned}$$

where,

- $lInPow(t)$ and $rInPow(t)$ are the power orders received by the simulator at instant t;
- $lOutPow(t-1)$ and $rOutPow(t-1)$ are the power values produced by motors at instant $t-1$, that is, in the previous simulation step;
- $lOutPow(t)$ and $rOutPow(t)$ are the power values produced by motors at instant t, that is, in the current simulation step;
- $lNoise(t)$ and $rNoise(t)$ are randomly calculated motor noise;
- $lNoisyOutPow(t)$ and $rNoisyOutPow(t)$ are the power values to be applied to wheels at instant t.

Robot - motor inertia and noise

Movement approach implemented by simulator decomposes it into two components, one linear along frontal axis of the robot and one rotational around its center. The simulator applies first the linear component, then the rotational one. These components are given by the following equations:

$$lin(t) = (lNoisyOutPow(t) + rNoisyOutPow(t))/2$$
$$rot(t) = (rNoisyOutPow(t) - lNoisyOutPow(t))/diam$$

where

- $lin(t)$, given in u_m , is the linear component of the movement, at instant t;
- $rot(t)$, given in radians, is the rotational component of the movement, at instant t;
- $diam$ is the robot diameter;

Robot motion

Simulation evolves in **discrete time**. Robot positions are modified, simultaneously to all robots, at the beginning of the simulation cycle. Nothing happens meanwhile.

Rules for robot motion:

1. New positions for all robots are computed, based on previous equations and assuming there are no obstacles.
2. Robots that, as a consequence of their movement, collide with obstacles or other robots are signaled.
3. Robots that do not collide assume the new positions.
4. For the colliding robots the rotational component of the movement is applied, the collision sensor is activated, and the collision penalty is applied.

Robot - communication

Robots are equipped with a broadcast communication system.

A robot can broadcast a message to all other robots. But there are constraints:

- Per cycle, a robot can send (broadcast) up to **100 bytes**.
- A message is only "heard" by a robot if the receiver robot is not further than **8** u_m from the transmitter robot. Obstacles do not interfere with communication.
- There is a **latency of 1** u_t for every communicated message.
- A fully linked configuration is not guaranteed at start. This means that, when the robots are in the starting grid positions, is not guaranteed that a message from a robot can be routed to all the others.
- All communications between robots must be performed **through the simulator** by sending appropriate commands. Direct communication between robot agents is not allowed.

How to develop a cyber-mouse

Any programming language that can use sockets.

Available APIs in C, C++, Java.

Simulator-robot communication protocol

Communication between simulator and agents is based on UDP sockets, being the messages formatted into XML structures. There are 4 message tags to consider:

1. request for registry
2. grant/refusal reply
3. sensor data
4. actuation order

Registration phase

The agent registers itself on the simulator sending a request for registry message to port 6000 of the IP address of the computer running the simulator.

```
<Robot Name="name" Id="pos">
    <IRSensor Id="sid" Angle="sangle"/>
    ⋮
</Robot>
```

The simulator response can be:

```
<Reply Status="Ok">
    <Parameters SimTime="time" KeyTime="time" CycleTime="time"
            NBeacons="nbeacons"
            BeaconNoise="noise" CompassNoise="noise"
            ObstacleNoise="noise" MotorsNoise="noise"
</Reply>
```

or

```
<Reply Status="Refused"></Reply>
```

Sensors data

After registration, the simulator, at every cycle, sends to the robot a message with sensor data. The message sent is a subset of the one shown bellow, the subset being depending on the sensor requests received.

```
<Measures Time="time">
    <Sensors Compass="angle" Collision="yesno" Ground="targetid">
        <BeaconSensor Id="0" Value="beaconmeasure"/>
        <IRSensor Id="0" Value="irmeasure"/>
        <IRSensor Id="1" Value="irmeasure"/>
        <IRSensor Id="2" Value="irmeasure"/>
        <IRSensor Id="3" Value="irmeasure"/>
        <GPS X="coord" Y="coord"/>
        <Message From="robotid"><![CDATA[msg]]></Message>
    </Sensors>
    <Leds EndLed="onoff" VisitingLed="onoff" ReturningLed="onoff"/>
    <Buttons Start="onoff" Stop="onoff"/>
</Measures>
```

Actuating order

At each cycle, agents can send to the simulator 1 or more actuating orders. If more than one is received, only the last one will be considered.

```
<Actions LeftMotor="pow" RightMotor="pow"
    EndLed="act"
    <SensorRequests IRSensor0="Yes" IRSensor1="Yes" ···
            Beacon0="Yes"
            Ground="Yes" Compass="Yes"/>
    <Say><![CDATA[msg]]></Say> </Actions>
```

CiberRato task

In CiberRato 2013 a team of 5 robots has the following mission:

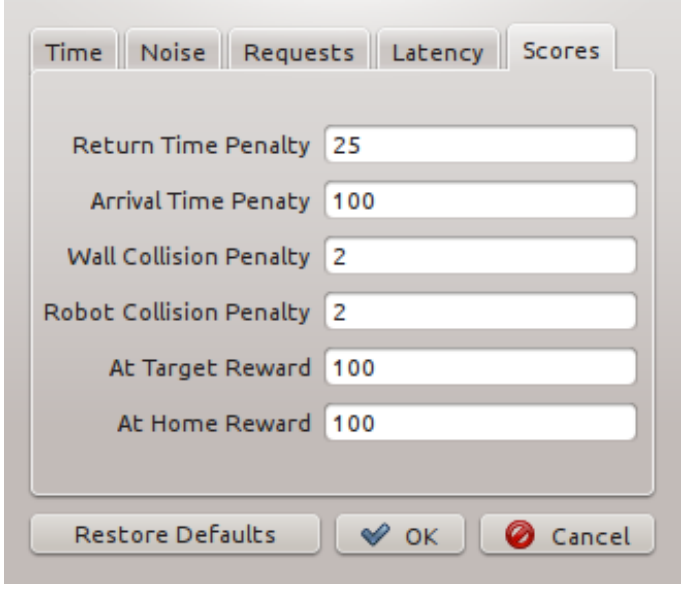
1. locate the target area (the position of which is unknown at the beginning)
2. position all robots on the target area
3. all robots return in the home area

Scores

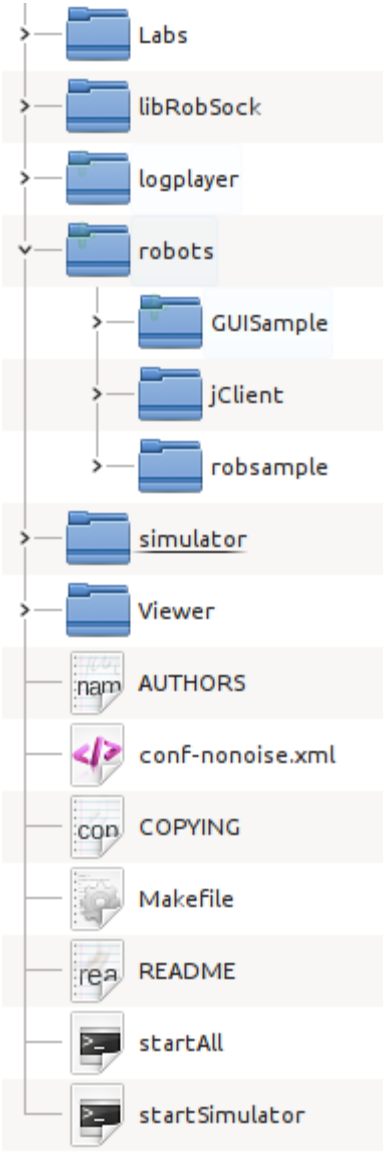
The simulator determines and assigns the score to each robot. The computed score takes into account the accomplishment of goals and the incurring penalties.

At the beginning 200 penalty points are assigned to each robot.

The less the robot scored, the better it has performed.



CPSS Installation



- Download the CPSS simulator form the web page of the course
- Make sure to have the Qt4 libraries installed
- Compile the CPSS files with `make`
- You find:
  - `Labs` contains many XML files for the arena
  - `simulator` contains the files for the simulator process
  - `Viewer` contains the files for the visualizer process
  - `robots` contains robot examples: `GUISample` in C++, `jClient` in Java, and `robsample` in C. Your robots go here!
  - `libRobSock` contains the C/C++ API
  - `startSimulator` launches the simulator with the correct configuration

Your tasks

Your task

Your (**first, mandatory**) task is to develop a single robot

1. locate the target area (the position of which is unknown at the beginning)
2. position the robot on the target area
3. return to the home area

Notes

- you can **work in team**. A team can be composed of up to 3 students.

Notes

- you can **work in team**. A team can be composed of up to 3 students.
- you must present your work anytime before the exam registration and you can get up to **3 extra-points**.

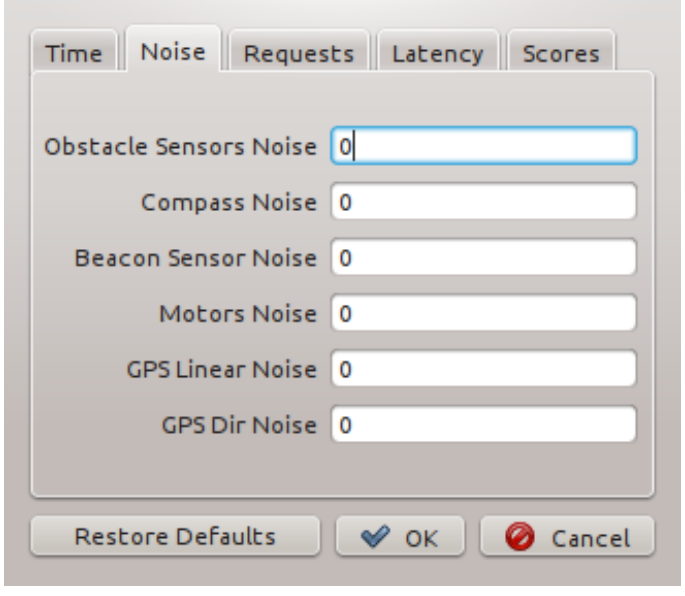
Notes

- you can **work in team**. A team can be composed of up to 3 students.
- you must present your work anytime before the exam registration and you can get up to **3 extra-points**.
- we evaluate the ability of the robot to perform the task, implementation choices and code

CPSS configuration

The simulator should be configured such that:

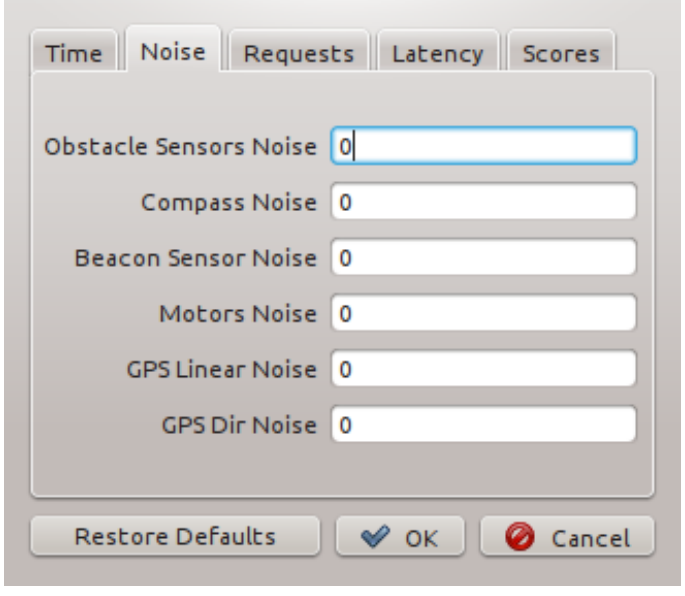
- sensors are not affected by noise and have 0 latency
- all sensors are available anytime (no request)
- motors are not affected by noise



CPSS configuration

The simulator should be configured such that:

- sensors are not affected by noise and have 0 latency
- all sensors are available anytime (no request)
- motors are not affected by noise



Yes, we are just too good...!

Student competition

Your (second, optional) task is to compete against other students.

Student competition

Your (second, optional) task is to compete against other students.

The competition will be held in March 2015.

Student competition

Your (second, optional) task is to compete against other students.

The competition will be held in March 2015.

We use the same rules of CiberRato 2013 (no a-priori notion of the map, the same **noise settings** and the twofold find-the-target/return-home task), except that teams are composed by **just one robot** and sensors are always available (no requests needed).

Student competition

Your (second, optional) task is to compete against other students.

The competition will be held in March 2015.

We use the same rules of CiberRato 2013 (no a-priori notion of the map, the same **noise settings** and the twofold find-the-target/return-home task), except that teams are composed by **just one robot** and sensors are always available (no requests needed).

Winners get 2 extra-points!

Tips for developing a good robot

Sense-Think-Plan-Act cycle

A good way to structure your software is to use the Sense-Think-Plan-Act cycle.

1. Sense: the robot senses the external world
2. Think: it updates its notion of the world
3. Plan: it decides what to do (strategy goes here)
4. Act: it acts accordingly

Finite State Machines

Use Finite State Machines to model your strategy

Bresenham's algorithm

A useful algorithm to construct a grid map from the obstacle vertexes is the Bresenham's algorithm `http://en.wikipedia.org/wiki/Bresenham's_line_algorithm`