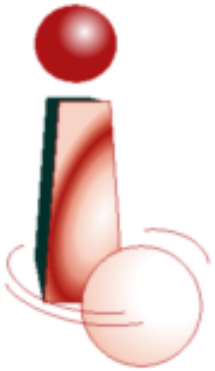
Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro

CiberRato 2013

Rules and Technical Specifications



(April, 2013)

Contents

1 Changes to last version

Rules to the *CiberRato* 2013 competition are exactly the same as for the previous one, the *CiberRato* 2012 competition.

2 Introduction

CiberRato is a robotic competition, which takes place in a simulation environment running in a network of computers. 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. It also creates the virtual bodies of the robots. Participants must provide the software agents which control the movement of the virtual robots, in order to acomplished the competition goal.

All virtual robots have the same kind of body. It is composed of a circular base, equipped with sensors, actuators, and command buttons. The simulator estimates sensor measures which are sent to the agents. Reversely, it receives and applies actuating orders coming from the agents.

Agents, playing as a team of five, are given the following challenge: starting from their position in the *starting grid* they must find a specific spot in a maze (*target area*), have a team meeting inside the *target area* and then make all the robots go to the *home area*. Each robot has to be controlled by an independent program, i.e. the team is actually a collection of, possibly identical, programs. Robots are allowed to communicate with each other within given restrictions. Score depends on fulfilment of challenge goals and on suffered penalties.

This document describes the rules and technical specifications applicable to the *CiberRato* 2013 edition.

3 Simulation environment

The virtual system that supports *CiberRato* contest is based on a distributed architecture, where 3 different type of applications enter into play: the simulator, the visualizer, and the agents (see figure 1).

The simulator is responsible for:

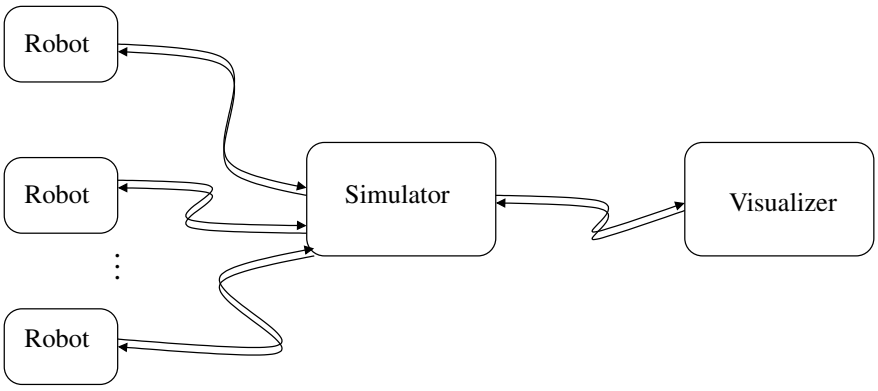


Figure 1: Overview of simulation system.

- 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.

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.

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. For the *CiberRato* 2013 edition the cycle time is 50 milliseconds.

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. All time intervals are measured as multiples of the cycle time. We denote u_t our unit of time, representing the cycle time.

4 Robot Body

Bodies of the virtual robots have a circular shape, 1 u_m wide, and are equipped with sensors, actuators and command buttons (see figure 2).

4.1 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. Some sensors are always available, namely the bumper and GPS. The others — ground, obstacle, beacon and compass sensors — are only available on request, with a limit of 4 per cycle.

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.

A description for each kind of sensor follows. A summary is given in table 1.

- **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. Figure 2 shows their default positions.

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. Noise is added to the ideal measure following a normal (gaussian) distribution with mean 0 (zero) and standard deviation 0.1. Obstacle sensors have a latency of 0 time units.

- 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. Noise is added to the ideal measure following a normal (gaussian) distribution with mean 0 (zero) and standard deviation 2.0. The beacon sensor has a latency of 4 time units.

  A beacon is not detected by the beacon sensor, if

  - there is a high obstacle between it and the sensor (we say the robot is in a shadow area);
  - the distance from it to the sensor is higher than 5 units of lenght (robot diameters).

- 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. Noise is added to the ideal measure following a normal (gaussian) distribution with mean 0 (zero) and standard deviation 2.0. The compass has a latency of 4 time units.

- 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. When the simulation starts, the arena is randomly positioned in the world, being the origin coordinates (the left, bottom corner) assigned a pair of values in the range 0–1000. Noise is added to the ideal measures following a normal (gaussian) distribution with mean 0 (zero) and standard deviation 0.5. It has a latency of 0 time units.

- The **bumper** corresponds to a ring put around the robot. It acts as a boolean variable enabled whenever there is a collision, with a latency of 0 time units.

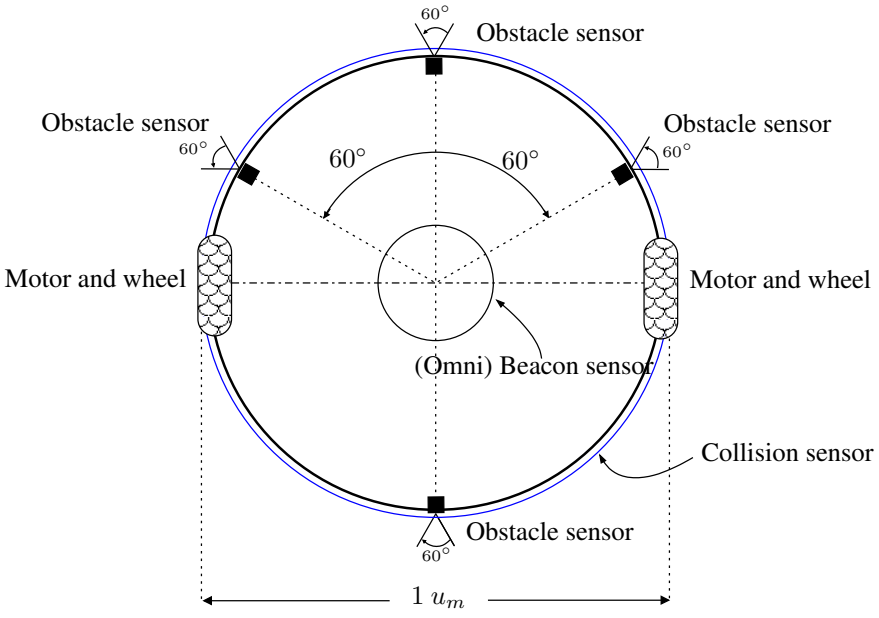


Figure 2: Body of the virtual robot.

- The **ground sensor** is a device that detects if the robot is completely over the *target area* or the *home area*. It has a latency of 0 time units.

4.2 Communication

Robots are equipped with a broadcast communication system. A robot can broadcast a message to all of its teammates. But there are constraints. Per cycle, a robot can send (broadcast) up to 100 bytes, and receive up to 400 (one full message from each of the teammates). A message is only 'heard' by a robot if the receiver robot isn't further than 8 distance units from the transmitter robot. Obstacles do not interfere with communication. There is a latency of 1 cycle 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.

4.3 Actuators

The virtual robot has 2 motors and 3 signalling leds (lights). The motors try to represent, although roughly, real motors. Thus, they have inertia and noise. A description for each kind of actuator follows. A summary is given in table 2.

- The 2 **motors** drive two wheels, placed as shown in figure 2. 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. See section 8 for a description of the input/output power relationship, that is, the relationship between power requested by agents and power applied to wheels. The noise is multiplicative, following a normal (gaussian) distribution with mean and standard deviation equal to 1 and 1.5%, respectively.

  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.

Sensor	Range	Resolution	Noise type	Deviation	Latency	On request
Obstacle s.	$[\,0.0, 100.0\,]$	0.1	additive	0.1	0	yes
Beacon s.	$[\,-180, +180\,]$	1	additive	2.0	4	yes
Compass	$[\,-180, +180\,]$	1	additive	2.0	4	yes
GPS (position)		0.1	additive	0.5	0	no
Bumper	Yes/No	..... N/A			0	no
Ground s.	Yes/No	..... N/A			0	yes

Table 1: Sensors characterization. On request sensors are limited to a maximum of 4 per cycle.

Actuator	Range	Resolution	Noise type	Standard deviation
Motor	$[-0.15, +0.15]$	0.001	multiplicative	1.5%
end led	On/Off	..... N/A		

Table 2: Actuators characterization.

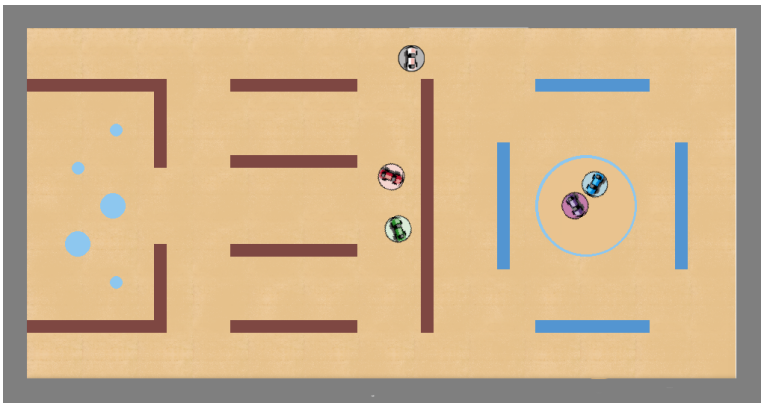


Figure 3: A game scenario.

- The 3 **leds** are named *visiting led*, *returning led* and *end led* and are used to signal the attainment of goals. The way they must be used depends on the competition challenge. See section 6 for details.

4.4 Buttons

Each virtual robot is equipped with 2 buttons, named *Start* and *Stop*. They are used by the simulator to start and interrupt competition. The *Start* button is pressed to start a competition or to restart a previously interrupted one. The *Stop* button is pressed when a competition is interrupted. Agents must read the status of these buttons and must act accordingly.

5 Arena

The game scenario (see figure 3 for an example) 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. The following rules are observed:

Arena

1. The arena maximum dimensions are 14 u_m high and 28 u_m wide.

Obstacles

2. All obstacles have planar surfaces, at least $0.3\ u_m$ wide.
3. All corners have angles ranging from 90 to 270 degrees.
4. Some obstacles can be higher than the beacon, defining shadow zones.
5. Any passage between obstacles is at least $1.5\ u_m$ wide.

Home area

6. The *home area* radius is at least $2.0\ u_m$ wide.
7. The *home area* is centered in starting position number 1.

Target area

8. The *target area* radius is at least $2.0\ u_m$ wide.
9. There is a beacon in the center of the *target area*.
10. The beacon does not act as an obstacle to robot movement.
11. The beacon can be not visible; see the beacon sensor description for details.

6 Competition

6.1 Computational structure

Competition takes place in a network of computers. One, operating in Linux, is used to run the simulator and the visualizer. The others are reserved to the participants (the agents of a team must run in at least two computers). They can operate in Windows, Linux or other OS with IP stack, Ethernet connection and the appropriated libraries.

6.2 Challenge

A team of 5 robots, playing simultaneously, is given a twofold challenge. First, it must locate and position all their robots on the *target area*, placed somewhere in a maze. After that, all the robots must position themselves within the *home area*.

When a robot is completely within the *target area* it can turn on its *visiting led*. The robot is disqualified if the led is turned on in a different situation. The second part of the challenge, going to the *home area*, can only be started when the *returning led* is on. This led is automatically turned on by the simulator when all the team's robots, with their *visiting leds* turned on, join within the *target area*. Note that a robot is not required to be stopped within the *target area* while it is waiting for its team mates. It can move around. When a robot reaches the *home area* it must turn on its *end led* and stop.

Since communication is possible, the team should explore the maze, sending their robots in different directions, in order to minimize the time needed to accomplish the goals.

There are two time limits to accomplish a trial, referred to as *key time* and *total time*. The latter represents the total time available to a team to complete the trial. The former is used to benefict the fastest robots (see section 6.4). The *total time* can range from 1800 to 3600 u_t, depending on the scenario difficulty.

6.3 Competition structure

Competition unfolds into 3 legs. In the first and second legs all teams participate. The three better qualified teams after the second leg go to the final one.

At each leg, every team participates in one single trial. At each trial, a team of 5 robots competes alone.

Game scenario can differ from leg to leg, but it is the same during all trials throughout a leg. The scenarios are unknown to the teams.

6.4 Scoring

At the end of each trial a score is assigned to the participant team. It is given by the sum of the scores of all team's robots (agents).

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

1. At the beginning 200 points are assigned to each robot.
2. Whenever a robot collides with an obstacle, 2 points are added to its score.
3. Whenever a robot collides with another robot, 2 points are added to its score. The simulator is responsible for deciding which robot caused the collision. Possibly, it can be both.
4. When the *target area* is correctly reached — turning on the *visiting led* — an amount of points is subtracted to the robot score. The amount is 100 if the *visiting led* is turned on at (or before) *key time*. After *key time*, it ranges linearly from 100 (*visiting led* turned on at *key time*) to 50 (*visiting led* turned on at *total time*).
5. Turning on the *visiting led* led outside the *target area* forces the end of robot competition, keeping the score held at that time.
6. When the *home area* is correctly reached[1] — turning on the *end led* and stopping the robot — an amount of 100 points is subtracted to the robot score.
7. Let T_R be the number of u_t (simulation cycles), rounded to the closest integer, a robot takes to cover the distance from the *target area* to the *home area*, driving at maximum speed. For each 25 u_t in excess of $T_R + 25$ taken in the returning trip, the robot score is incremented by 1 point.
8. No robot can score more than 250 points. A robot is assigned that score if it is excluded by judge decision or finishes with a higher value.

Besides score, each team is also assigned a number and a time, both used as criteria to play off.

1. The number represents the number of robots that correctly reached the *home area* (well-succeeded robots) plus the number of robots that correctly reached the *target area*.

[1] Recall that before a robot can go to the *home area* the *returning led* must be turned on and for that all the team robots must join within the *target area* with their *visiting leds* turned on.

2. The time is related to the slowest robot reaching the *target area*. If any of the robots of the team correctly reached the *target area*, the assigned time is equal to the one spent by the slowest robot; otherwise, it is equal to the *total time* for that trial.

6.5 Ranking

Ranking is defined by the ascending order of assigned scores. In case of equal scores, play off is determined primarily by the descending order of the number of well-succeeded robots and secondarily by the ascending order of times assigned to the teams. Assigned scores, numbers and times differ from leg to leg. They are defined as follows:

1. At the end of the first leg, each team is assigned the score, the number of robots and the time obtained in its trial. (All agents pass to second leg.)
2. At the end of the second leg, each team is assigned a score, a number of robots and a time, which are equal to the sum of the scores, numbers and times obtained in the first and second legs, respectively. (Only the 3 better qualified pass to the third (final) leg.)
3. For the final ranking only the score, the number of robots and the time obtained at final leg count.

6.6 Panel of judges

The panel of judges is the maximum authority in terms of rules interpretation and application. Their mission is to verify rules observation by robots and to aid the referee in his/her decisions.

You can not appeal against panel decisions.

The panel is designated by the *CiberRato Organization*.

6.7 Referee

The referee controls the competition and ensures contest rules observance. The referee can interrupt the competition to consult the panel of judges. In all omitted issues he/she must, compulsorily, consult the panel of judges.

The referee is designated by the *CiberRato Organization*.

6.8 Abnormal circumstances

As a consequence of an abnormal situation the referee can interrupt the competition at any time in order to consult the panel of judges. When this happens, all robots are notified through the *Stop* button and are immobilized in the simulator scope. Time is also frozen.

The panel can decide to resume, finish, or repeat the current trial.

Resuming or repeating competition

1. The process of resuming a previously interrupted competition is controlled by the referee, being the robots notified through the *Start* button. Spacial and angular positions of the robots at restart time are exactly the same they had at interrupt time.
2. Repetition is done substituting excluded robots with robots from the *CiberRato Organization*.

7 Simulation parameters

Configuring the simulator for a leg is done passing it the following elements:

- Cycle time and total competition time.
- Noise levels for sensors and motors.
- Maze and *starting grid* descriptions.

Configuration files are written based on XML descriptions. There are 3 main XML tags, `Parameters`, `Lab` and `Grid`. Since XML tags are self-explanatory we just give an example for each case.

```
<Lab Name="Default LAB" Height="14" Width="28">
    <Beacon X="24" Y="7,0" Height="4,0"/>
    <Target X="24" Y="7,0" Radius="2,0"/>
    <Target X="7" Y="7,0" Radius="2,0"/>
    <Wall Height="5,0">
        <Corner X="10,0" Y="4,0"/>
        <Corner X="11,0" Y="4,0"/>
        <Corner X="11,0" Y="10,0"/>
        <Corner X="10,0" Y="10,0"/>
    </Wall>
</Lab>

<Grid>
    <Position X="3,0" Y="9,0" Dir="0,0"/>
    <Position X="4,0" Y="8,0" Dir="0,0"/>
    <Position X="5,0" Y="7,0" Dir="0,0"/>
    <Position X="4,0" Y="6,0" Dir="0,0"/>
    <Position X="3,0" Y="5,0" Dir="0,0"/>
</Grid>

<Parameters SimTime="1800" KeyTime="1350" CycleTime="50"
        CompassNoise="2.0" BeaconNoise="2.0"
        ObstacleNoise="0.1"
        GPS="On"
        Lab="lab.xml" Grid="grid.xml"/>
```

Any attribute can be absent, in which case a default value is assumed.

8 Simulation models

The simulator is a complex system that runs in discrete time. Some type of sensors and actuators equipping a robot have complex real behaviour. Their simulation counterparts have, often, models that are simplified approximations. Since these models can impact agent development they are presented next.

Discrete time

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

Robot movement

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

$$\texttt{lOutPow}_t = (\texttt{lOutPow}_{t-1} + \texttt{lInPow}_t) \ / \ 2$$
$$\texttt{rOutPow}_t = (\texttt{rOutPow}_{t-1} + \texttt{rInPow}_t) \ / \ 2$$
$$\texttt{lNoisyOutPow}_t = \texttt{lOutPow}_t \star \texttt{lNoise}_t$$
$$\texttt{rNoisyOutPow}_t = \texttt{rOutPow}_t \star \texttt{rNoise}_t$$

where,

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

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.

$$\texttt{lin}_t = (\texttt{lNoisyOutPow}_t + \texttt{rNoisyOutPow}_t) \ / \ 2$$
$$\texttt{rot}_t = (\texttt{rNoisyOutPpow}_t - \texttt{lNoisyOutPow}_t) \ / \ \texttt{diam}$$

where

- $\texttt{lin}_t$, given in u_m, is the linear component of the movement, at instant t;
- $\texttt{rot}_t$, given in radians, is the rotational component of the movement, at instant t;
- `diam` is the robot diameter;

As stated before, movement is processed simultaneously for all robots. The following steps are used.

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.

9 Communication Protocols

Communication between simulator and agents is based on UDP *sockets*, being the messages formatted into XML structures. There are 5 message tags to consider: *request for registry*, *grant response*, *refusal response*, *sensor data*, and *actuation order*. You only need to read this section if you plan to use a programming language different from C, C++, or Java. Otherwise, you can use the libraries of functions available in the tool package (RobSock.h and ciberIF.java).

Request for registry

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. The message looks like

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

where:

- `name` is the robot name, the one appearing in the scoreboard;
- `pos` is the robot position in the *starting grid*;
- `sid` is the id of an obstacle sensor, ranging from 0 to 3;
- `sangle` is the angular position of the sensor in robot periphery, ranging from `-180.0` to `+180.0`.

Tags `IRSensor` are optional. You must use them if you want to change the default position of the obstacle sensors.

Refusal response

If the simulator refuses the request for registry it sends to the agent the message

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

Grant response

If the simulator accepts the request for registry it sends to the agent the message

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

where

- `time` is an integer value, representing a time, in u_t;
- `nbeacons` is the number of beacons/*target area*, which in this competition is always 1[2];
- `noise` is a real value, representing a noise level.

The agent must memorize the port where this response came from and send all new messages to there.

Actuating orders

At each cycle, agents can send to the simulator 1 or more actuating orders. However, the number of orders per device is limited to one. If more than one is received, only the last one will be considered. Each *actuating order* message is a subset of

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

where

- `pow` is a real value, representing a power;
- `act` is the word "On" or "Off", representing an order to turn-on or turn-off a *led*;
- `msg` is the message to be broadcasted;
- the number of sensor requests per cycle is limited to 4 — if more are requested, only 4, arbitrarily chosen, are considered.

Motor orders are persistent, in the sense that the order is kept until a new one is received by the simulator. Sensor requests are not persistent. If an agent wants to read the same sensors in two or more consecutive cycles, it needs to send the sensor requests on each cycle.

[2] In this competition there is only one beacon present in a maze. However, the tools allow for more than one.

Sensor 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>
```

where

- `time` is an integer value representing a time;
- `angle` is a real value representing an angle, in radians;
- `yesno` is the word "Yes" or "No";
- `targetid` is 0 if the robot is completely inside the *target area*, 1 if it is completely inside the *home area* and -1 otherwise;
- `beaconmeasure` is a real value representing a beacon sensor measure, if beacon is visible, or the word "NotVisible" othersiwe;
- `irmeasure` is a real number, representing an obstacle sensor measure;
- `coord` is a real number, representing a GPS spatial position;
- `msg` is the message broadcasted by robot number `robotid`;
- `onoff` is the word "On" or "Off", representing a *led* or button state.