



Departamento de Electrónica, Telecomunicações e Informática  
Universidade de Aveiro

# *Ciber-Mouse*

## Rules and Technical Specifications



(July, 2007)

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Simulation System</b>	<b>3</b>
<b>3</b>	<b>Robot Body</b>	<b>4</b>
3.1	Sensors . . . . .	4
3.2	Actuators . . . . .	6
3.3	Buttons . . . . .	7
<b>4</b>	<b>Arena</b>	<b>7</b>
<b>5</b>	<b>Competition</b>	<b>8</b>
5.1	Computational structure . . . . .	8
5.2	Challenge . . . . .	9
5.3	Competition structure . . . . .	9
5.4	Scoring . . . . .	9
5.5	Ranking . . . . .	10
5.6	Anomalous circumstances . . . . .	11
5.7	Panel of Judges . . . . .	11
5.8	Arbiter . . . . .	11
<b>6</b>	<b>Simulation parameters</b>	<b>12</b>
<b>7</b>	<b>Simulation models</b>	<b>13</b>
<b>8</b>	<b>Communication Protocols</b>	<b>14</b>

## 1 Introduction

*Ciber-Mouse* is a competition among virtual robots, which takes place in a simulation environment running in a network of computers. The simulation system creates a virtual arena with a starting grid, a target area, signaled by a beacon, and populated of obstacles. 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 accomplished some goals.

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

Agents are given the following challenge: starting from their position in starting grid they must visit the target area and then return the their starting point. Score depends on fulfilment of challenge goals and on penalties suffered.

This document describes the rules and technical specifications applicable to the RTSS07 special edition.

## 2 Simulation System

The virtual system that supports *Ciber-Mouse* contest is based on a distributed architecture, where 5 applications enter into play: the simulator, the visualizer, and three agents (see figure 1).

The simulator is responsible for:

- Implementing the virtual bodies of the robots.

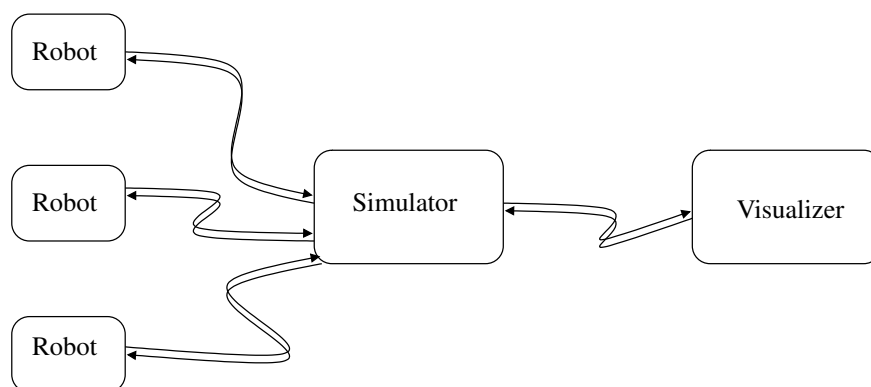


Figure 1: Overview of simulation system.

---

- Estimating sensor measurements and sending them to the corresponding agent.
- Moving robots within arena, according to orders received from corresponding agent and taking into account environment restrictions. For instance, a robot can not cross an obstacle.
- Updating robot score, taking into account the goals fulfilled and penalties applied.
- Sending scores and robots positions to visualizer.

The visualizer is responsible for:

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

The simulation system is discrete and time-driven. In each time step the simulator sends sensor measurements to agents, receives actuations orders, apply them, and update scores. For the RTSS07 special edition the cycle time is 25 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.

### 3 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).

#### 3.1 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<sup>1</sup>. The others — ground, obstacle, beacon and compass sensors — are only available on request, with a limit of two per cycle.

Sensor models try to represent real devices. Thus, in one side, their measures are noisy. In another 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 done  $n$  cycles ago.

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

---

<sup>1</sup>The GPS is not available during competition

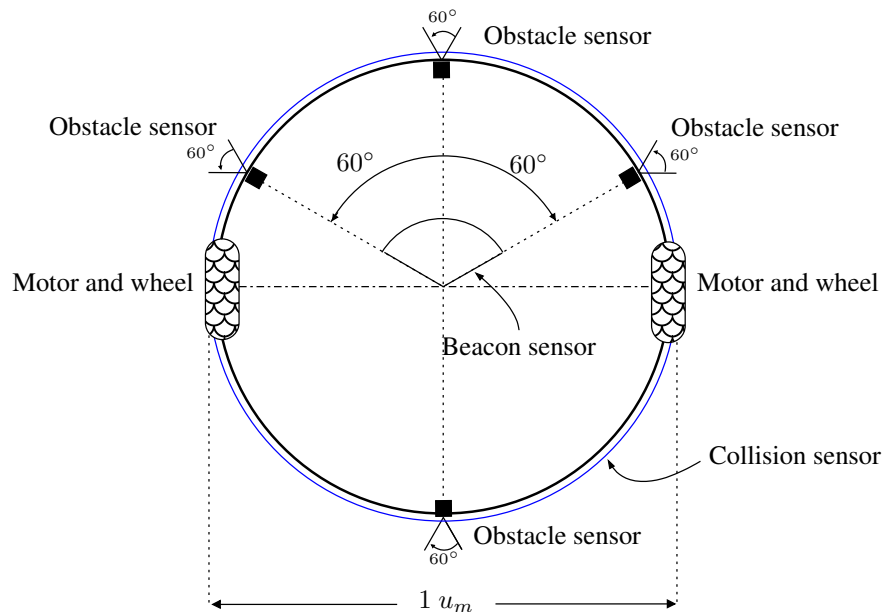


Figure 2: Body of the virtual robot.

- The **obstacle sensors** measure distance between 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 apertura 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, turned forward, and with an aperture angle of 120 degrees, 60 to each side. It measures the angular position of the robot with respect to its frontal axis.

A beacon can be not visible, because:

- the robot is not facing it, that is, the beacon is out of the sensor aperture angle;
- there is at least a high obstacle between it and the robot (we say the robot is in a shadow area).

If the beacon is visible the sensor measure ranges between  $-60$  e  $+60$  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 9 time units.

- The **compass** is positioned at the robot center and measures its angular position with respect to

Table 1: Sensors characterization.

Sensor	Range	Resolution	Noise type	deviation	Latency	On request
Obstacle s.	[ 0.0, 100.0 ]	0.1	aditive	0.1	0	yes
Beacon s.	[ -60, +60 ]	1	aditive	2.0	9	yes
Compass	[ -180, +180 ]	1	aditive	2.0	9	yes
Bumper	Yes/No	.....N/A.....			0	no
Ground s.	Yes/No	.....N/A.....			0	yes

the *virtual North*. We assume the X axis is facing the virtual north.

Its measures range between  $-180$  e  $+180$  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 9 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.
- The **ground sensor** is a device the detects if the robot is completely over the target area. Whenever this happens it produces an output equal to the target area id, with a latency of 0 time units.
- The **GPS** is a device that returns the spacial position of the robot. It is located in the robot center. It is ideal, that is, not noisy, and has a latency of 0 time units. It can only be used during development. During competition it is disabled.

### 3.2 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. It follows a description of each kind of actuator. A summary is given in table 2.

- The 2 **motors** drive two wheels, put as shown in figure 2. Robot movement depends on 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 7 for a description of the input/output power relationship, that is, the relationship between power request 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.

Table 2: Actuators characterization.

Actuator	Range	Resolution	Noise type	Standard deviation
Motor	$[-0.15, +0.15]$	0.001	multiplicative	1.5%
<i>led End</i>	On/Off	.....	N/A.....	.....
<i>led Return</i>	On/Off	.....	N/A.....	.....
<i>led Beacon</i>	On/Off	.....	N/A.....	.....

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.

- The three leds, named *Beacon*, *Return* and *End*, are used to signal goals. The robot must turn-on led *Beacon* to signal it is over the target area. It must turn-on led *Return* to signal it is starting the returning phase. Finally, it must turn-on led *End* to signal it has finished its run.

### 3.3 Buttons

The 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 by the simulator 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.

## 4 Arena

The game scenario (see figure 3 for an example) is composed of a rectangular arena, outer delimited, with obstacles and target area inside. The target area is a circle with a beacon in its center. Obstacles are elements put to make difficult the robot movement. They can, for instance, represent walls, watercourses, or trenches. A starting grid defines the robots initial positions. The following rules are observed:

### Arena

1. The arena is  $14 u_m$  high and  $28 u_m$  wide.

### Obstacles

2. All obstacles have planar surfaces, with at least  $0,4 u_m$  wide.

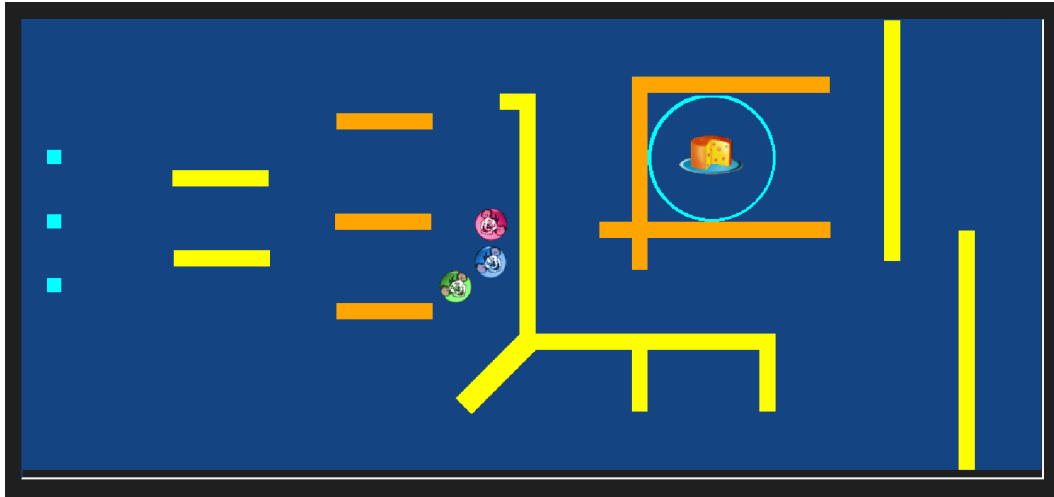


Figure 3: A game scenario.

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.

### Target area

6. The target area is at least  $1,0 u_m$  wide.
7. The beacon in the center of the target area does not act as an obstacle to robot movement.

## 5 Competition

### 5.1 Computational structure

Competition takes place in a network of 4 computers. One, operating in Linux, is used to run the simulator and visualizer. The others are reserved to participants, one agent per computer. They can operate in Windows, Linux or other OS with IP stack, Ethernet connection and an appropriate library.

Eventually, a participant can use his/her own computer to run the agent. In that case, he/she must ask for a connection to the network.

## 5.2 Challenge

A robot must accomplish two goals in order to conclude its competition. Firstly, it must visit the target area. Inside that area it must turn-on the corresponding *Beacon* led, which must be turned-off after leaving it. After that, and only after that, the robot must return, as close as possible, to its position in the starting grid. The robot must signal the beginning of its return by turning-on its *Return* led inside the target area.<sup>2</sup> The robot must turn-on its *End* led to signal its competition is over.

There is a time limit to accomplish the two goals, ranging between 1800 and 3600  $u_t$ . The time limit can vary from scenario to scenario.

## 5.3 Competition structure

Competition is structured in 3 stages. In the first and second stages all teams participate. Every team runs three times at each stage, being chosen the run with the best score. The three better qualified teams after the second stage go to the final.

Three teams run at a time. Thus, at the beginning of each stage agents are distributed into groups of 3. Distribution is defined by lot.

The final stage unfolds into 3 runs. Robot positions in starting grid are drawn to the first run. Then they rotate.

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

## 5.4 Scoring

At the end of each competition the simulator determines and assigns a score to each robot. The computed score takes into account the accomplishment of goals and the incurring penalties. The lowest the score, the best. The following rules are observed:

1. At the beginning 300 points are assigned to each robot.
2. Whenever a robot collides with an obstacle, 5 points are added to its score.
3. Whenever a robot collides with another robot, 5 points are added to its score. The simulator is responsible for deciding which robot caused the collision. Possibly, it can be both.
4. When visiting the target area (corresponding *Beacon* led turned-on) 100 points are subtracted to robot score. Note that entering the target area without turning on the corresponding *Beacon* led does not represent a visit.

---

<sup>2</sup>Note that, inside the target area, two leds must be turned-on, one to conclude the first goal, the other to start the second one. They don't need to be turned-on at the same time. A robot can decide to explore the maze before start the return.

5. For each time cycle a robot keeps the *Beacon* led turned-on outside a target area, 5 points are added to its score.
6. Let  $D_G$  be the shortest distance from the starting grid position of a robot to the center of the target area. Let  $T_G$  be the number of  $u_t$  (simulation cycles), rounded down to the closest integer, a robot takes to cover the previous  $D_G$  distance, driving at maximum speed. For each 100  $u_t$  in excess of  $T_G + 100$  taking in the going trip, the robot score is incremented 1 point.
7. When the robot correctly signals the beginning of its returning trip (turning on the *Return* led inside the target area) 100 points are subtracted to its score.
8. Turning on *Return* led outside the target area forces the end of robot competition, keeping the score held at that time.
9. Let  $D_R$  be the shortest distance from the point a robot signals the beginning of its returning trip to its position in the starting grid. Whenever the robot moves near its starting grid position  $D_R/100$ , its score is decremented 1 point. Whenever it moves away  $D_R/100$  its score is incremented 1 point.
10. Let  $T_R$  be the number of  $u_t$  (simulation cycles), rounded to the closest integer, a robot takes to cover the previous  $D_R$  distance, driving at maximum speed. For each 25  $u_t$  in excess of  $T_R + 25$  taking in the returning trip, the robot score is incremented 1 point.
11. A robot ends its competition when it turns on its led *End*. If by the end of the time limit a robot has not turned on its led *End*, the simulator forces the end and applies a 15 points penalty.
12. No robot can score more than 300 points. A robot is assigned that score if it is excluded by judge decision or finishes with a higher value.

Besides score, each robot is also assigned the time it takes to conclude its run. It is used as a second criterion to play off.

## 5.5 Ranking

Ranking is defined by the ascending order of assigned scores and, in case of equal scores, by the ascending order of run times. Assigned scores and times differ from stage to stage. They are defined as follows:

1. At the end of the first stage, each agent is assigned the score and the time obtained in its best run. (All agents pass to second stage.)
2. At the end of the second stage, each robot is assigned a score and the time, which are equal to the sum of the scores and times obtained in its best runs in the first and second stages, respectively. (Only the 3 better qualified pass to third (final) stage.)
3. The final stage unfolds into 3 runs. At the end of the this stage, each agent is assigned a score and the time, which are equal to the sum of the scores and times obtained in the two best runs.

## 5.6 Anomalous circumstances

As a consequence of an anomaly the arbiter (see section 5.8) can interrupt the competition at any time in order to consult on the panel of judges. Doing so all robots are notified through the *Stop* button and are immobilized in the simulator scope. Time is also frozen.

The panel can decide to exclude a robot from the competition. It can also decide to resume, finish, or repeat the current run.

Exclusion can be justified with a robot behaviour that is harmful beyond normal interference between robots. For instance, a robot can be excluded because:

- it deliberately and/or repeatedly collides with other robots;
- it repeatedly sends messages, causing system overload.

### Resuming or repeating competition

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

## 5.7 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 arbiter in his/her decisions. The most severe penalties, like robot exclusion, can only be applied by the panel.

You can not appeal against panel decisions.

The panel is designated by the *Ciber-Mouse Organization*.

## 5.8 Arbiter

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

The arbiter is designated by the *Ciber-Mouse Organization*.

## 6 Simulation parameters

Configuring the simulator for a stage 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 Id="1" X="24" Y="7,0" Height="4,0"/>
  <Target Id="1" X="24" Y="7,0" Radius="1,5"/>
  <Beacon Id="2" X="14" Y="7,0" Height="4,0"/>
  <Target Id="2" X="14" Y="7,0" Radius="1,5"/>
  <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="4,0" Y="9,0" Dir="0,0"/>
  <Position X="5,0" Y="7,0" Dir="0,0"/>
  <Position X="4,0" Y="5,0" Dir="0,0"/>
</Grid>
```

```
<Parameters SimTime="1800" CycleTime="25"
  CompassNoise="2.0" BeaconNoise="2.0" ObstacleNoise="0.1"
  MotorsNoise="1.5"
  GPS="Off"
  Lab="lab.xml" Grid="grid.xml"/>
```

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

## 7 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

$$\begin{aligned} lOutPow_t &= (lOutPow_{t-1} + lInPow_t) / 2 \\ rOutPow_t &= (rOutPow_{t-1} + rInPow_t) / 2 \\ lNoisyOutPow_t &= lOutPow_t * lNoise_t \\ rNoisyOutPow_t &= rOutPow_t * rNoise_t \end{aligned}$$

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  $t$ .

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.

$$\begin{aligned} lin_t &= (lNoisyOutPow_t + rNoisyOutPow_t) / 2 \\ rot_t &= (rNoisyOutPow_t - lNoisyOutPow_t) / diam \end{aligned} \quad \text{where}$$

- $lin_t$  and  $rot_t$  are the linear and rotational components 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 collides 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.

## 8 Communication Protocols

Communication between simulator and agents is based on UDP *sockets*, being the messages formatted into XML structures. There are 5 messages 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, we 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 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 scoreboard;
- `pos` is robot position in starting grid;
- `sid` is the id of an obstacle sensor, ranging between 0 and 3; and
- `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="stime" CycleTime="ctime"
    CompassNoise="cnoise"
    NBeacons="nbeacons" BeaconNoise="bnoise"
    ObstacleNoise="onoise" MotorsNoise="mnoise"
  </Reply>
```

where

- `stime` and `ctime` are the total competition and cycle times;
- `nbeacons` is the number of beacons/target areas;
- `cnoise`, `bnoise` and `onoise` are the compass, beacon sensor and obstacle sensor noise levels;
- `mnoise` is the motors noise level.

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

## Sensor data

After registration, the simulator sends every cycle a message with data from robot sensors. The message sent is a subset of the one shown below, the subset being depending on the sensor requests from previous cycle.

```
<Measures Time="curtime">
  <Sensors Compass="compassvalue"
    Collision="collisionvalue" Ground="groundvalue">
    <IRSensor Id="0" Value="irs0value"/>
    <IRSensor Id="1" Value="irs1value"/>
    <IRSensor Id="2" Value="irs2value"/>
    <IRSensor Id="3" Value="irs3value"/>
```

```

    <GPS X="xvalue" Y="yvalue" Dir="dirvalue"/>
  </Sensors>
  <Leds EndLed="endledvalue" ReturningLed="retledvalue"/>
  <Buttons Start="startvalue" Stop="stopvalue"/>
</Measures>

```

where

- `curtime` is an integer value representing the current simulation (competition) time;
- `compassvalue` is a real value representing the compass measure;
- `beaconvalue` is a real value representing the beacon sensor measure if beacon is visible, or the word "NotVisible" otherwise;
- `collisionvalue` is the word "Yes" if the robot had collided (and it was robot's fault) or the word "No" otherwise;
- `groundvalue` is the id of a target area (in the range 0 to N-1, where N is the number of target areas) if the robot is completely inside one, or -1 otherwise;
- `irs0value`, `irs1value`, `irs2value` e `irs3value` are real numbers representing obstacle sensors measures;
- `xvalue`, `yvalue` e `dirvalue` are real values representing the GPS measures;
- `endledvalue` and `retledvalue` are the words "On" or "Off", stating if leds *End* and *Return* are turned-on or turned-off, respectively;
- `startvalue` and `stopvalue` are the words "On" or "Off", stating if competition is running or stopped (because of an interruption).

### 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="lpow" RightMotor="rpow"
  VisitingLed="vact" EndLed="eact" ReturningLed="ract">
  <SensorRequests IRSensor0="Yes" IRSensor1="Yes"
    IRSensor2="Yes" IRSensor3="Yes"
    Beacon0="Yes" Ground="Yes" Compass="Yes"/>
</Actions>

```

where

- $l_{power}$   $r_{power}$  are real values representing the power to be applied to the left and right motor, respectively;
- $v_{act}$ ,  $e_{act}$   $e_{ract}$  are the words "On" or "Off" representing an order to turn-on or turn-off leds *Beacon*, *End* and *Return*, respectively.

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.