



Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro

CiberMouse@RTSS08 2008
Rules and Technical Specifications



(August, 2008)

Contents

1	Introduction	3
2	Simulation System	3
3	Robot Body	4
3.1	Sensors	4
3.2	Communication	6
3.3	Actuators	6
3.4	Buttons	7
4	Arena	7
5	Competition	9
5.1	Computational structure	9
5.2	Challenge	9
5.3	Competition structure	9
5.4	Scoring	9
5.5	Ranking	10
5.6	Abnormal circumstances	10
5.7	Panel of judges	11
5.8	Referee	11
6	Simulation parameters	11
7	Simulation models	12
8	Communication Protocols	13

1 Introduction

CiberMouse@RTSS08 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, populated by obstacles, where a starting grid and a target area, signaled by a beacon, 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 accomplish 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, receives and apply actuating orders coming from the agents.

Agents, playing as a team, are given the following challenge: starting from their position in the starting grid they must find a specific spot in a maze (target area) and then position every robot of the team near that spot. 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 *CiberMouse@RTSS08* 2008 edition.

2 Simulation System

The virtual system that supports *CiberMouse@RTSS08* 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).

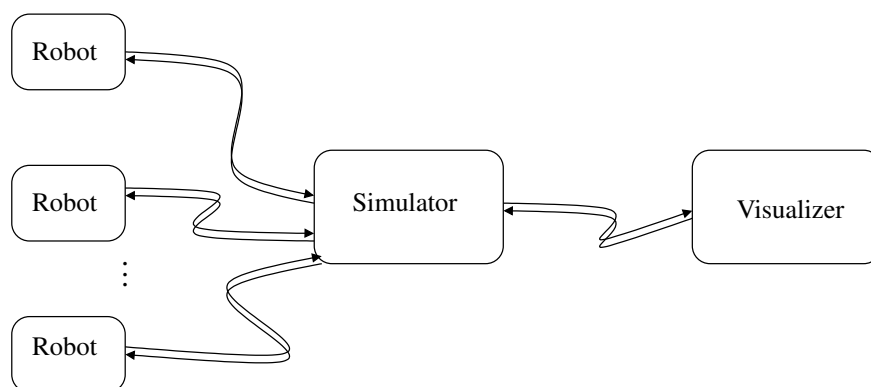


Figure 1: Overview of simulation system.

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.
- Route 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.

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 *Ciber-Mouse@RTSS08* 2008 edition the cycle time is 25 milliseconds.

All members 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. The others — ground, obstacle, beacon and compass sensors — are only available on request, with a limit of 2 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 done n cycles ago.

A description for each kind of sensor follows. A summary is given in table 3.1.

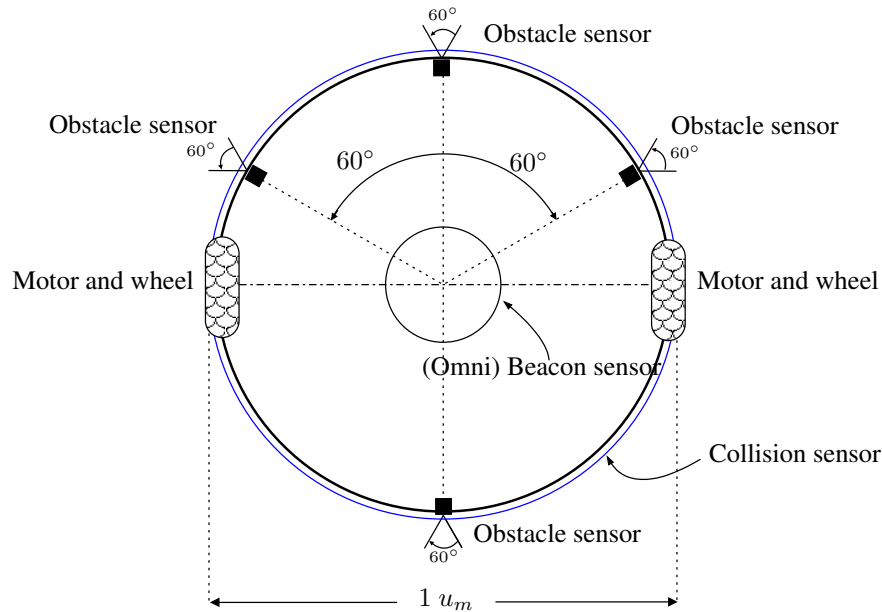


Figure 2: Body of the virtual robot.

- 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 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 sensors** are positioned in the center of the robot, and has an omnidirectional covering. They measure the angular position of a beacon with respect to the robot's frontal axis.

A beacon can be undetected, if there is at least a high obstacle between it and the robot (we say the robot is in a shadow area). The beacon sign has a limit reach. So, it is undetected if a robot is far away from it. If a beacon is detected the corresponding sensor 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.

- The **compass** is positioned at the robot center 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

Sensor	Range	Resolution	Noise type	Deviation	Latency	On request
Obstacle s.	[0.0, 100.0]	0.1	aditive	0.1	0	yes
Beacon s.	[-180, +180]	1	aditive	2.0	4	yes
Compass	[-180, +180]	1	aditive	2.0	4	yes
GPS (position)		1	aditive	0.5	0	no
GPS (orientation)	[-180, +180]	1	aditive	5.0	0	no
Bumper	Yes/NoN/A.....			0	no
Ground s.	Yes/NoN/A.....			0	yes

Table 1: Sensors characterization.

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 and orientation of the robot in the arena, and is located at the robot center. Noise is added to the ideal measures following a normal (gaussian) distribution with mean 0 (zero) and standard deviation 0.5, for the position, and 5.0, for the orientation. 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.
- The **ground sensor** is a device the detects if the robot is completely over a target area. Whenever this happens it produces an output equal to the target area id, with a latency of 0 time units.

3.2 Communication

Robots are equipped with a broadcast communication system. A robot can broadcast a message up 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 it 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.

All communications between robots must be performed through the simulator by sending appropriate commands. Direct communication between robot agents is not allowed.

3.3 Actuators

The virtual robot has 2 motors and 1 signalling led (light). The motors try to represent, although roughly, real motors. Thus, they have inertia and noise. A description for each kind of actuator

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

Table 2: Actuators characterization.

follows. A summary is given in table 3.3.

- 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 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.

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 led, named and *End led*, is used to signal the attainment of goal. The robot must turn-on *End led* to signal it has finished its trial.

3.4 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 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 areas inside. A target area is a circle with a beacon in its center. Obstacles

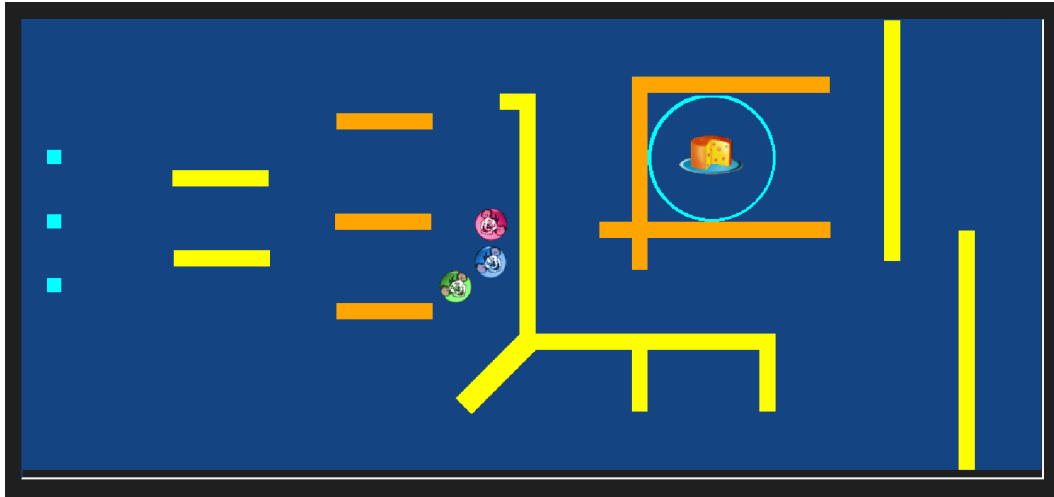


Figure 3: A game scenario.

are elements placed within the arena to make difficult the robot movements. 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.
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 radius is at least $2.0 u_m$ 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 computers. One, operating in Linux, is used to run the simulator and the visualizer. The others are reserved to the participants (each participant should use at least two computers to run their agents). They can operate in Windows, Linux or other OS with IP stack, Ethernet connection and the appropriated libraries.

5.2 Challenge

A team of 5 robots, playing simultaneously, must locate and position all their members on a target area placed somewhere in a maze. 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 goal.

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 benefit the fastest agents. The *total time* can range from 1800 to 3600 u_t , depending on the scenario difficulty.

5.3 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 compete 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.

5.4 Scoring

At the end of each trial 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 applied:

1. At the beginning 100 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 the target area is correctly reached — turning on the *End led* and stopping the robot — an amount of points is subtracted to the robot score. The amount is 100 if the *key time* was not exceeded, and ranges from 100 to 0 otherwise.
5. No robot can score more than 200 points. A robot is assigned that score if it is excluded by judge decision or finishes with a higher value.

The score of a team is given by the sum of the scores of all its members.

Besides score, each team is also assigned a time, used as a second criterion to play off. If all the robots of the team correctly reach the target area, the time is equal to the one spent by the slowest robot; otherwise, it is equal to the time limit for that trial.

5.5 Ranking

Ranking is defined by the ascending order of assigned scores and, in case of equal scores, by the ascending order of visiting times. Assigned scores 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 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 and the time, which are equal to the sum of the scores and times obtained in the first and second legs, respectively. (Only the 3 better qualified pass to third (final) leg.)
3. For the final ranking only the score and time obtained at final leg count.

5.6 Abnormal circumstances

As a consequence of an abnormal situation the referee (see section 5.8) can interrupt the competition at any time in order to consult on 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 *CiberMouse@RTSS08 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 referee in his/her decisions.

You can not appeal against panel decisions.

The panel is designated by the *CiberMouse@RTSS08 Organization*.

5.8 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 *CiberMouse@RTSS08 Organization*.

6 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="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="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" CycleTime="25"
  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.

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.

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

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;

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.

8 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" 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 areas, which in this competition is always 1¹;
- `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 2 — if more are requested, only 2, 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.

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.

¹In this competition there is only one beacon present in a maze. However, the tools allow for more than one.

```

<Measures Time="time">
  <Sensors Compass="angle" Collision="yesno" Ground="id">
    <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" Dir="angle"/>
    <Message From="id"><![CDATA[msg]]></Message>
  </Sensors>
  <Leds EndLed="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";
- `id` is 0 if the robot is completely inside the target area and -1 otherwise;
- `beaconmeasure` is a real value representing a beacon sensor measure, if beacon is visible, or the word "NotVisible" otherwise;
- `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 `id`;
- `onoff` is the word "On" or "Off", representing a *led* or button state.