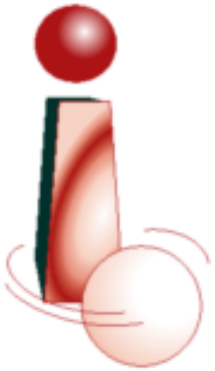
Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro

CiberRato 2018

Rules and Technical Specifications



(March, 2018)

Contents

1 Introduction

This document describes the rules and technical specifications applicable to the *CiberRato* 2018 edition, called **Maze explorer**.

CiberRato is a robotic competition, which takes place in a simulation environment running in a network of computers. The simulation system creates a virtual maze, which the competing robots have to solve. The maze is built on a grid of squared cells, being one of them defined as the *starting cell* and another as the *target cell*.

The simulation system also creates the virtual bodies of the robots. All virtual robots have the same kind of body. It is composed of a circular base, equipped with sensors, actuators, and command buttons.

Participants must provide the software, referred to as the *agent*, that controls the movement of a virtual robot, in order to acomplished the competition goal. The simulator estimates sensor measures which are then sent to the agent. Reversely, it receives and applies actuating orders coming from the agent. Thus, the agent acts as the brain of the robot.

The challenge in the 2018 edition of *CiberRato*, is as follows. At start, the robot is placed in the center of an unknown starting cell. It must explore the maze in order to locate the target cell, while constructing a representation of the maze explored so far. Then, it must return back to the starting cell, through the possible shortest path. Score depends on fulfilment of challenge goals and on suffered penalties.

2 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:

- Implementing the virtual bodies of the robots.
- Estimating sensor measurements and sending them to the corresponding agent.
- Moving robots within the maze, according to orders received from corresponding agent and taking into account environment restrictions. For instance, a robot can not move through a wall.
- 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.

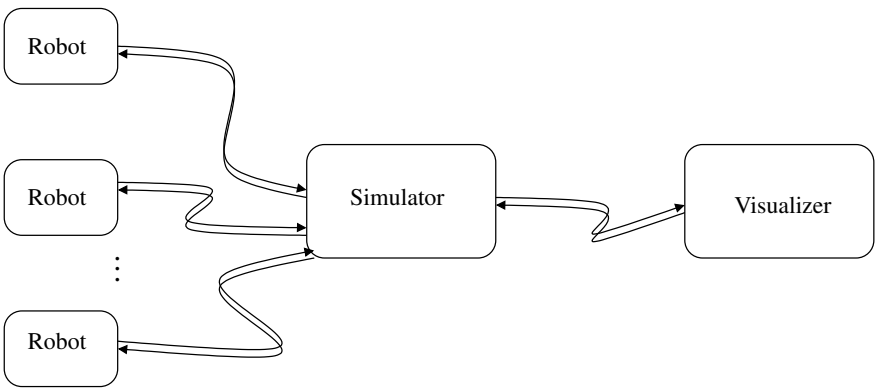


Figure 1: Overview of simulation system.

The visualizer is responsible for:

- Graphically showing robots in competition maze, 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* 2018 edition the cycle time is 50 milliseconds.

All elements into play, namely maze and robots, are virtual, thus there is no need for a real length unit. Hence, we use u_m as the unit of length, which corresponds to the robot diameter. 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 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 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 provided in table 1.

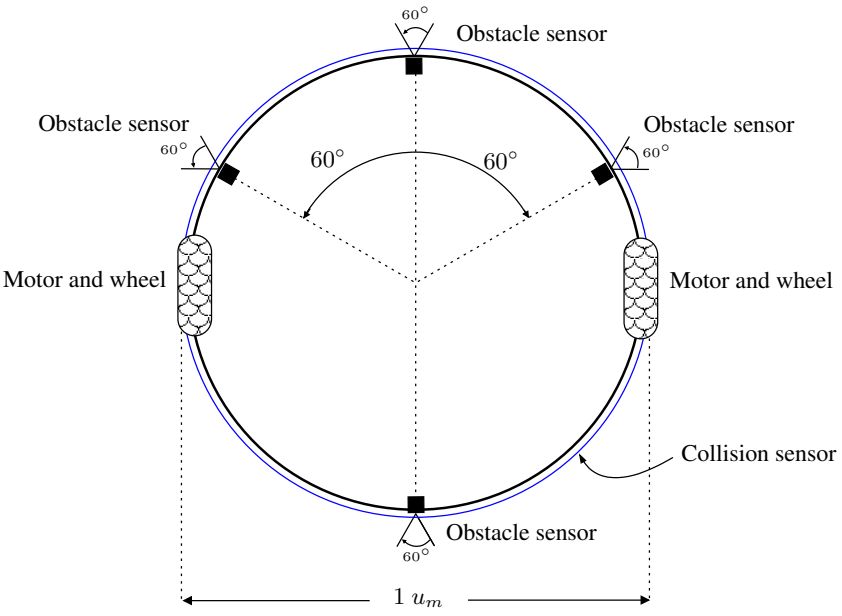


Figure 2: Body of the virtual robot.

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

Sensor	Range	Resolution	Noise type	Deviation	Latency	On request
Obstacle sensor	$[\,0.0, 100.0\,]$	0.1	additive	0.1	0	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 sensor	Yes/No	..... N/A			0	yes

- **Obstacle sensors** measure distances between the robot and its surrounding obstacles, including other robots. They have predefined positions, but can be repositioned on robot initialization, however only at 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 **compass** is positioned in the center of the robot and measures its angular position with respect to the *virtual North*. We assume the X (horizontal) 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 world, with resolution 0.1. It is located at the robot center. When the simulation starts, the maze 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 can be 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 GPS is **not** available during competition, but can be used during development to test your localization algorithm.

- The **bumper** corresponds to a ring placed 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 that detects if the robot is completely over the *target area*. It has a latency of 0 time units.

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. A description for each kind of actuator follows. A summary is provided 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

Table 2: Actuators characterization.

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

to both motors the robot moves along its frontal axis. If the power values are symmetric the robot rotates around its center.

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 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.

- The 2 **leds** are named *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 5 for details.

3.3 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.

4 Competing scenario

The competition scenario (see figure 3 for an example) is composed of a grid of squared cells, outer delimited, with a *starting cell*, and a *target cell* inside. Wall segments can be placed between adjacent cells, to hamper the robot movements, creating the maze. The *target cell* is the cell that should be reached by the robot and has a ground material identifiable by the ground sensor. The *starting cell* is the departing cell of the robot and is indistinguishable from the others, except the target cell. The following rules are observed:

1. The side of each cell measures $2\,u_m$.
2. The maze maximum dimensions are 7 cells high and 14 cells wide.

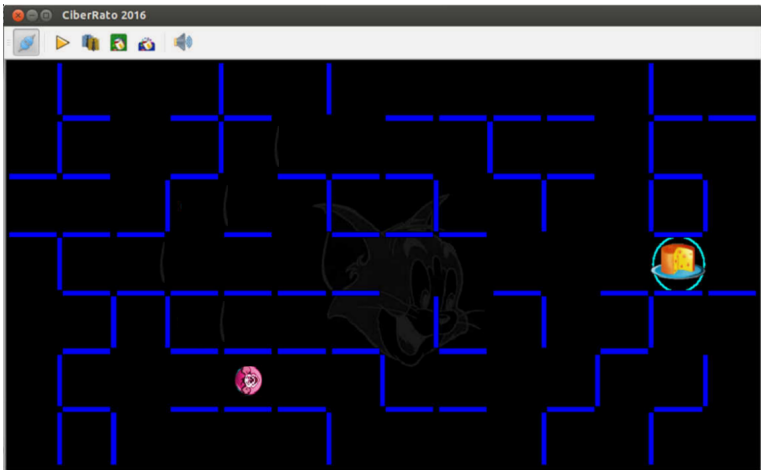


Figure 3: A competition scenario.

3. All wall segments are $0.1\,u_m$ wide.
4. The robot pose at start is always 0 or 180 degrees.
5. There is always a possible path from the starting to the target cell.

5 Competition

5.1 Computational structure

Competition takes place in a network of two computers. One, typically operating in Linux, is used to run the simulator and the visualizer. The other is used by the participant to run the agent. It can operate in Windows, Linux or other OS with IP stack, Ethernet connection and the appropriate libraries.

5.2 Challenge

The main objective is the development of a robotic agent to command a mobile robot, making it move to an unknown target cell in a semi-structured environment and then return back to the starting cell through the shortest possible path.

A robot can visit the target cell several times before it decides to return back to its starting cell. However, when it decides to do so, it must turn on the *returning led* inside the target cell. Then, when it reaches the *starting cell*, it must turn on the *end led* and stop.

In order to better fulfill the challenge objectives, participants must be aware that:

- Information from the obstacle sensors has to be used in order to avoid and follow the walls.
- The target position can be detected using the ground sensor, as the value measured by this sensor changes when the robot is completely inside this region.
- The starting cell can not be detected by any special sensor.
- There are no encoder sensors in the wheels, but the robot pose may be estimated using the motor model and the velocity commands sent to the simulation. The grid structure of the maze may also be used to improve self-localization.
- To deal with the noise in sensors, some kind of filtering should be used.
- Some way to represent the environment or the possible navigation paths should be used, in order to compute the shortest path to the starting cell.

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, the robot 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 in advance to the teams.

5.4 Scoring

At the end of each trial a score is assigned to the participant team. The computed score takes into account the accomplishment of goals and the incurred penalties. The following rules are applied:

To be defined

5.5 Ranking

To be defined

5.6 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*.

5.7 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*.

5.8 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. 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.

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 description and starting cell coordinates.

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">
    <Target X="25" Y="7" Radius="1"/>
    <Row Pos="12" Pattern="  |        |     |                 |     "/>
    <Row Pos="11" Pattern="  +--+  +--+--+  +  +--+--+--+--+  +--+--"/>
    <Row Pos="10" Pattern="  |        |           |        |     "/>
    <Row Pos="9"  Pattern="--+--+  +--+  +--+--+--+  +--+--+  +--+  "/>
    <Row Pos="8"  Pattern="        |        |     |     |     |  |  "/>
    <Row Pos="7"  Pattern="--+--+--+  +--+  +--+--+--+  +  +  +--+  "/>
    <Row Pos="6"  Pattern="  |                                       "/>
    <Row Pos="5"  Pattern="  +--+--+--+--+--+--+  +  +--+  +--+--+--"/>
    <Row Pos="4"  Pattern="     |  |              |     |     |     "/>
    <Row Pos="3"  Pattern="  +--+  +--+--+--+--+  +--+  +  +--+  +  "/>
    <Row Pos="2"  Pattern="  |                 |           |     |  "/>
    <Row Pos="1"  Pattern="  +--+  +--+--+--+  +--+--+  +--+  +--+  "/>
    <Row Pos="0"  Pattern="  |  |           |           |     |     "/>
</Lab>
```

```
<Grid>
   <Position X="3" Y="5" Dir="0"/>
<Grid>

<Parameters SimTime="1800" KeyTime="1350" CycleTime="50"
      CompassNoise="2.0" ObstacleNoise="0.1"
      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

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

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 the 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;

- `rot`$_t$, given in radians, is the rotational component of the movement, at instant t;
- `diam` is the robot diameter;

The following steps are followed:

1. A new robot position is computed, based on previous equations and assuming there are no obstacles.
2. If, the new position implies a collision with an obstacle (wall), the linear component of the movement is ignored, being only the rotational one applied,
3. Moreover, 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` for C/C++, `ciberIF.java` for Java, and `croblink.py` for python).

8.1 Registry

Each agent must register itself on the simulator by 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">
   <IRSensor Id="sid" Angle="sangle"/>
</Robot>
```

where `name` is the robot name, the one appearing in the scoreboard, `sid` is the id of an obstacle sensor, ranging from 0 to 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.

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

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

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

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

where `time` is an integer value, representing a time, in u_t, and `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.

8.2 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 received will be considered. Each *actuating order* message is a subset of

```
<Actions LeftMotor="pow" RightMotor="pow"
   VisitingLed="act" ReturningLed="act" EndLed="act"
   <SensorRequests IRSensor0="Yes" IRSensor1="Yes" ···
         Ground="Yes" Compass="Yes"/>
</Actions>
```

where `pow` is a real value, representing a power, and `act` is the word "On" or "Off", representing an order to turn-on or turn-off a *led*. 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.

8.3 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="groundid">
      <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"/>
   </Sensors>
   <Leds EndLed="onoff" VisitingLed="onoff" ReturningLed="onoff"/>
   <Buttons Start="onoff" Stop="onoff"/>
</Measures>
```

where `time` is an integer value representing the current time, `angle` is a real value representing an angle, in radians, `yesno` is the word "Yes" or "No", `groundid` is 0 if the robot is completely inside the *target cell* and -1 otherwise, `irmeasure` is a real number representing an obstacle sensor measure, `coord` is a real number representing a GPS spatial coordinate and `onoff` is the word "On" or "Off" representing a *led* or button state.