

CiberRato Competition and Tools

Robótica Móvel e Inteligente
IEETA/IRIS Lab – DETI – Universidade de Aveiro



- Micro-Rato in a simulated environment
 - Simulator implements maze and bodies of virtual robots
 - Virtual sensors and motors are **identical for all robots**
- Participants develop the robot brain (control software)
 - Brains take decision **autonomously** (software agents)
- Focus on software components of robotics

<http://sourceforge.net/projects/cpss/>
<http://microrato.ua.pt>



Ciber-Rato: the environment

- Maze with high and low walls, a starting grid, one or more beacon-areas and a home-area



- **Initial State:** Robots are placed at starting grid
- **First Objective:** Get into BEACON area; if there is more than one Beacon all areas must be visited
- **Second Objective:** Return to the closest point to departure in the least possible time
- **Score** = Distance to starting point
+ Excess of returning time
+ Penalties
 - Collisions
 - Lack of fulfillment of first objective
 - Not signaling finish

- Rules changed to increase links with sensor networks and cooperative multi-agent systems research
 - More agents: 5
 - Agents can communicate, but communication distance is limited
 - GPS may be used; has noise
 - Target: All agents must reach the beacon area in minimal time
 - Target 2: All agents must return to a home area (since 2011)
- **Robots must cooperate to achieve challenge!!!**
- Firstly used in CiberMouse@RTSS2008, Barcelona, Spain
- CyberRescue@RTSS2009 in Washington DC, USA
- CiberRato 2009-2015, Aveiro

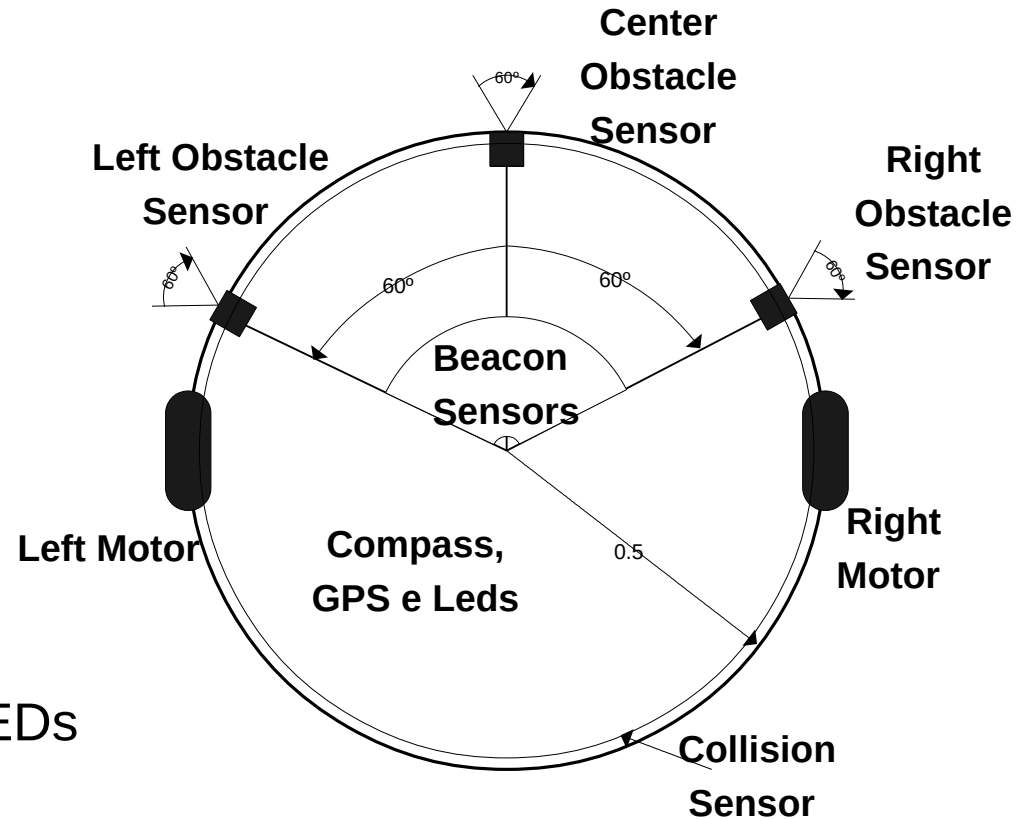
- The Virtual Robot is equipped with:

- **Sensors**

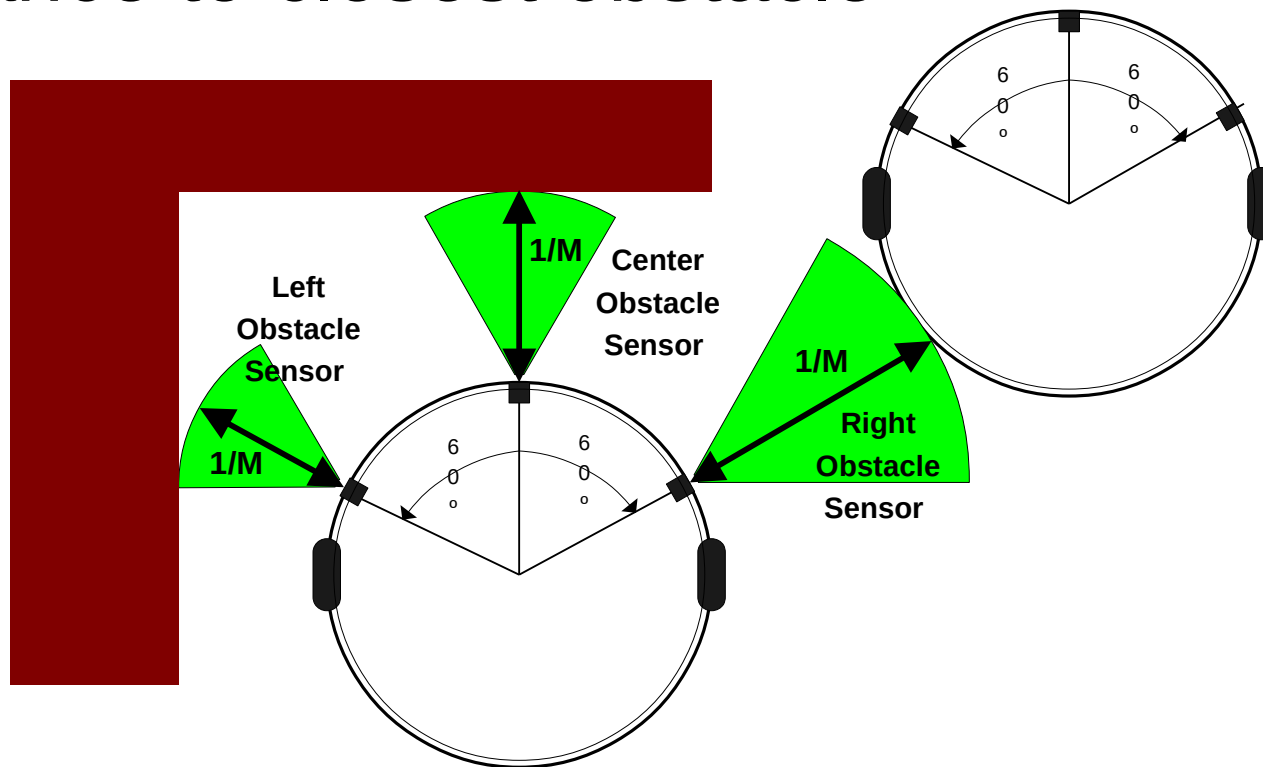
- Obstacles
- Beacon
- Ground
- Collisions
- Compass
- GPS (debug)

- **Actuators**

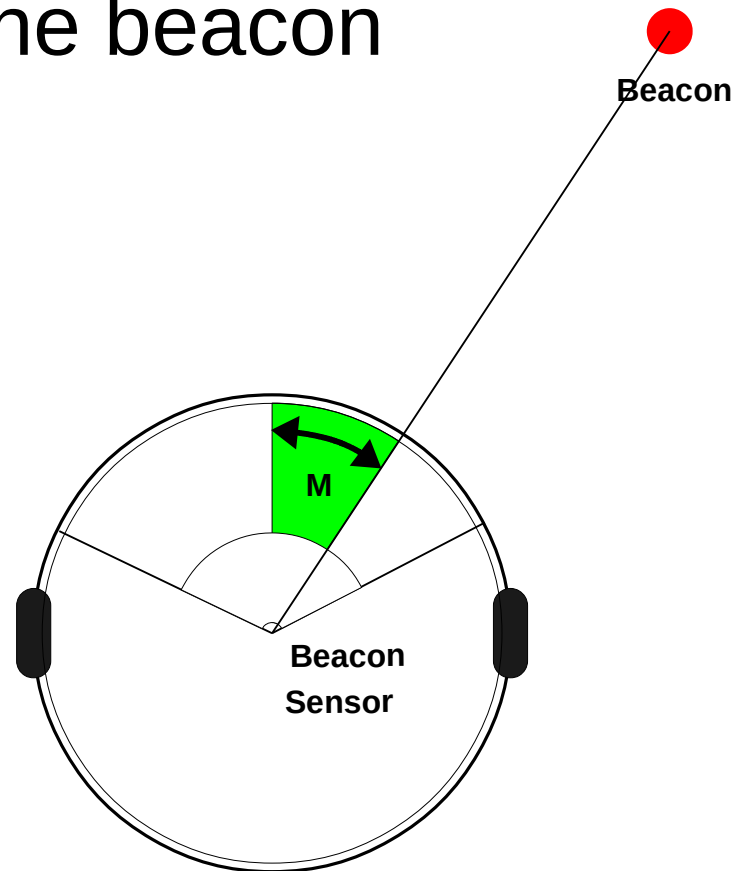
- 2 Motors
- Some signaling LEDs



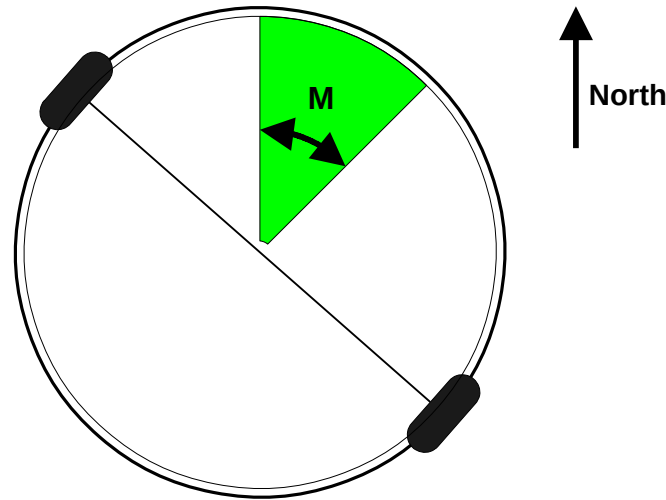
- Measure is inversely proportional to distance to closest obstacle



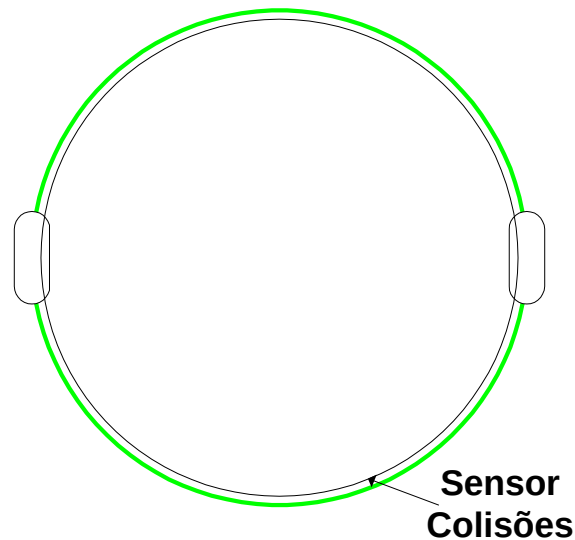
- Measure is the angle from the front of the robot to the beacon



- Measure is the angle from the Virtual North to the front of the robot



- Binary Sensor active when the robot collides

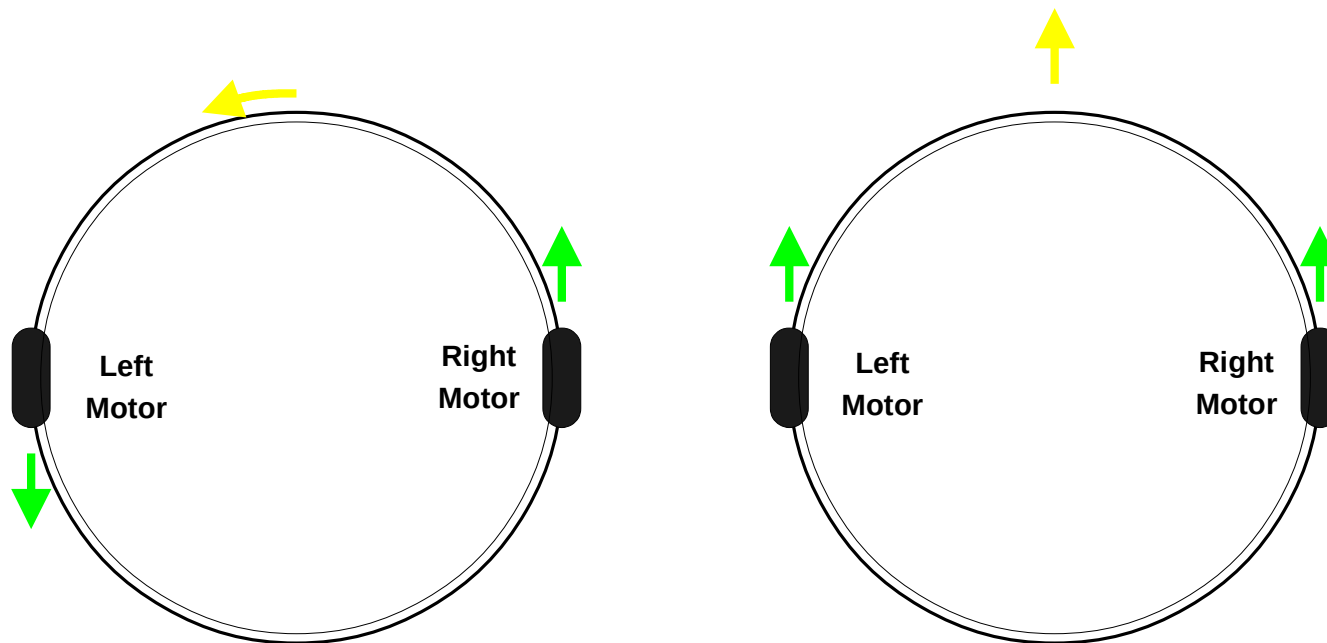


Sensor requests

- Starting at CiberMouse@RTSS2006 the number of sensors that is received in each cycle can be limited
- Agents must request the sensor measures they want to receive in the next cycle
- Also the latencies of sensors can be higher than 0

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

- Agents control robot movement defining motor speeds of two independent motors

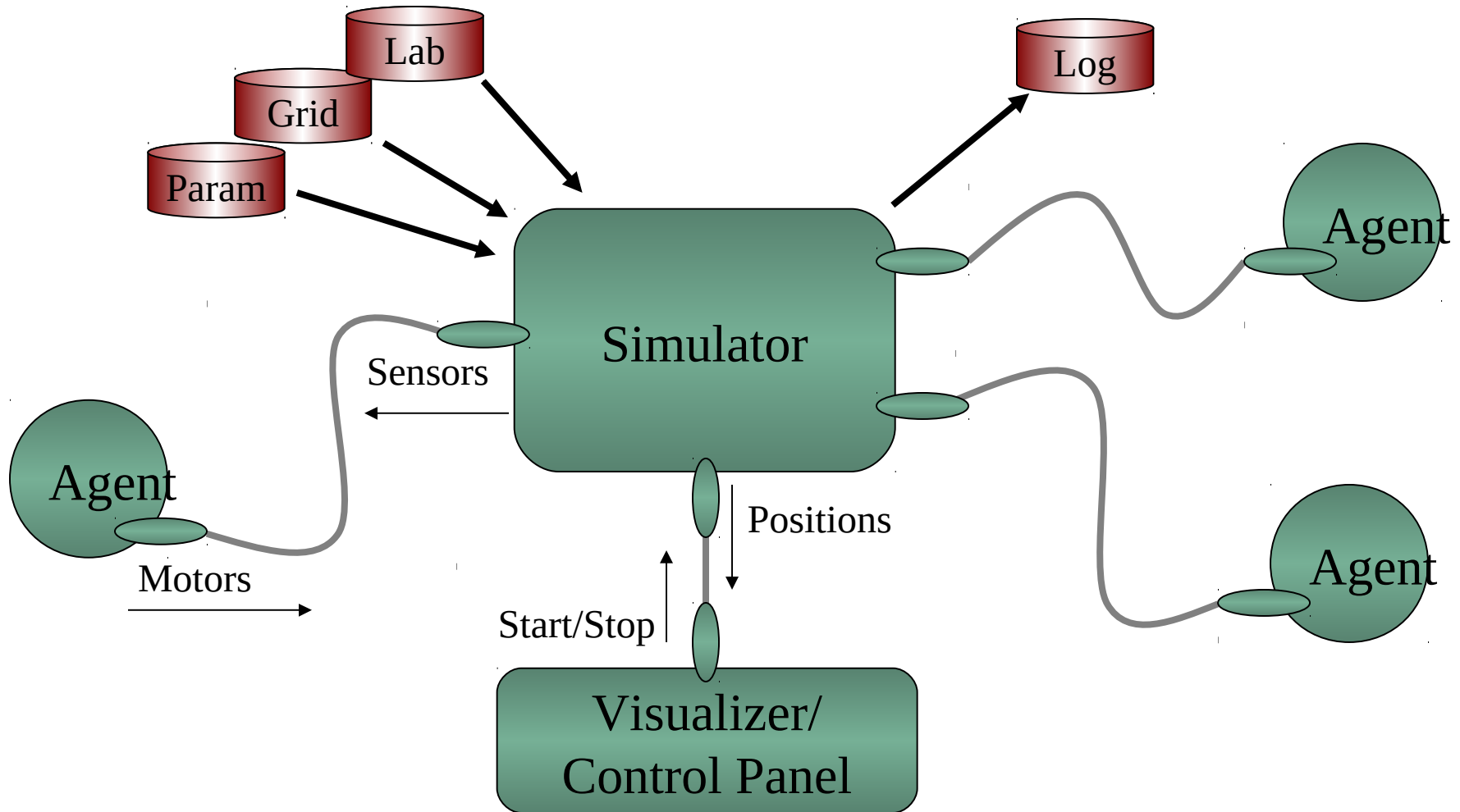


- Motors: to drive the robot
 - Simulator adds noise to actuation commands
 - Motor model uses inertia
- Leds: to signal specific events

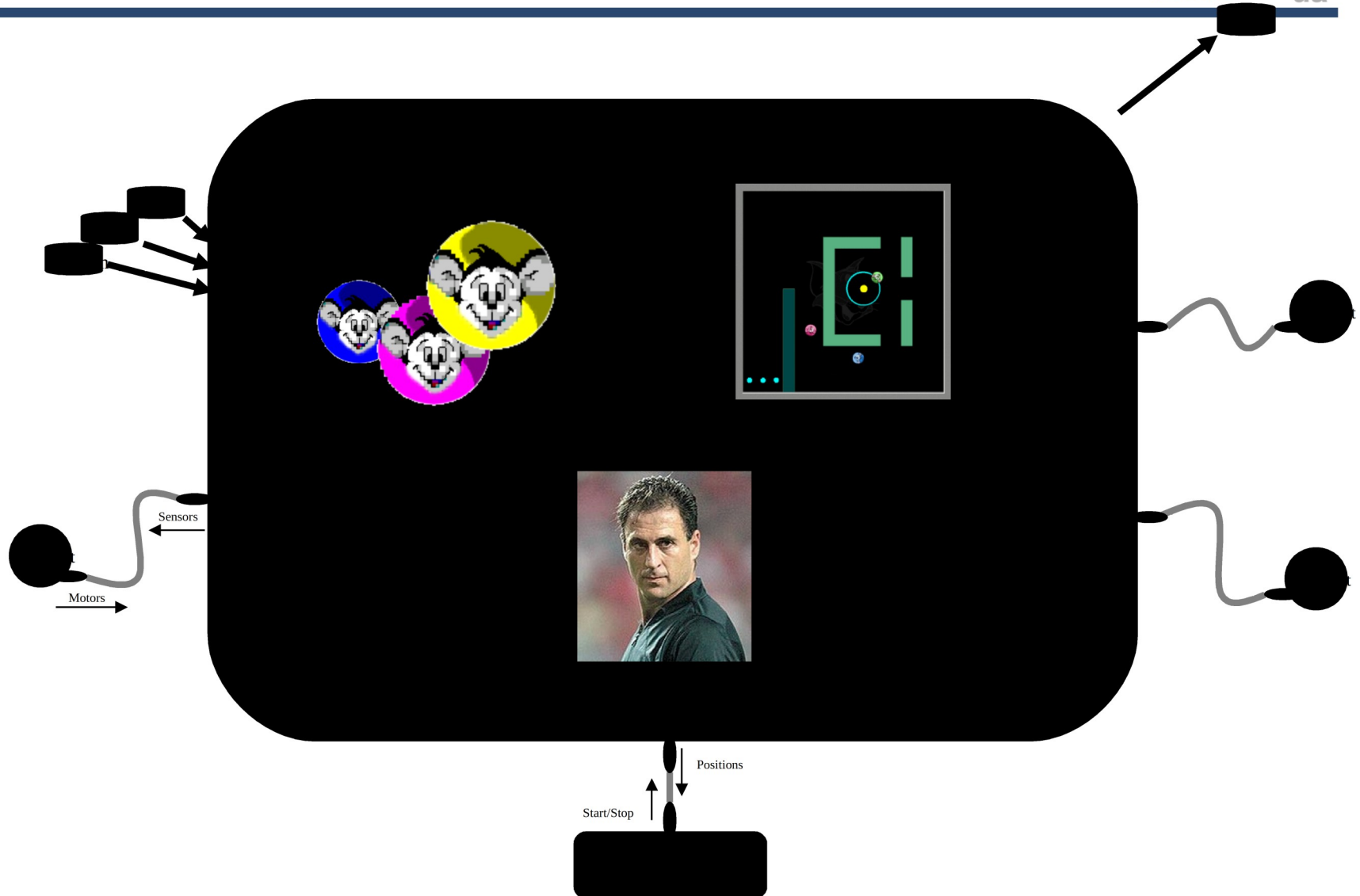
Actuator	Range	Resolution	Noise type	Standard deviation
Motor	$[-0.15, +0.15]$	0.001	multiplicative	1.5%
<i>End led</i>	On/Off	N/A.....	
<i>Return led</i>	On/Off	N/A.....	
<i>Beacon led</i>	On/Off	N/A.....	

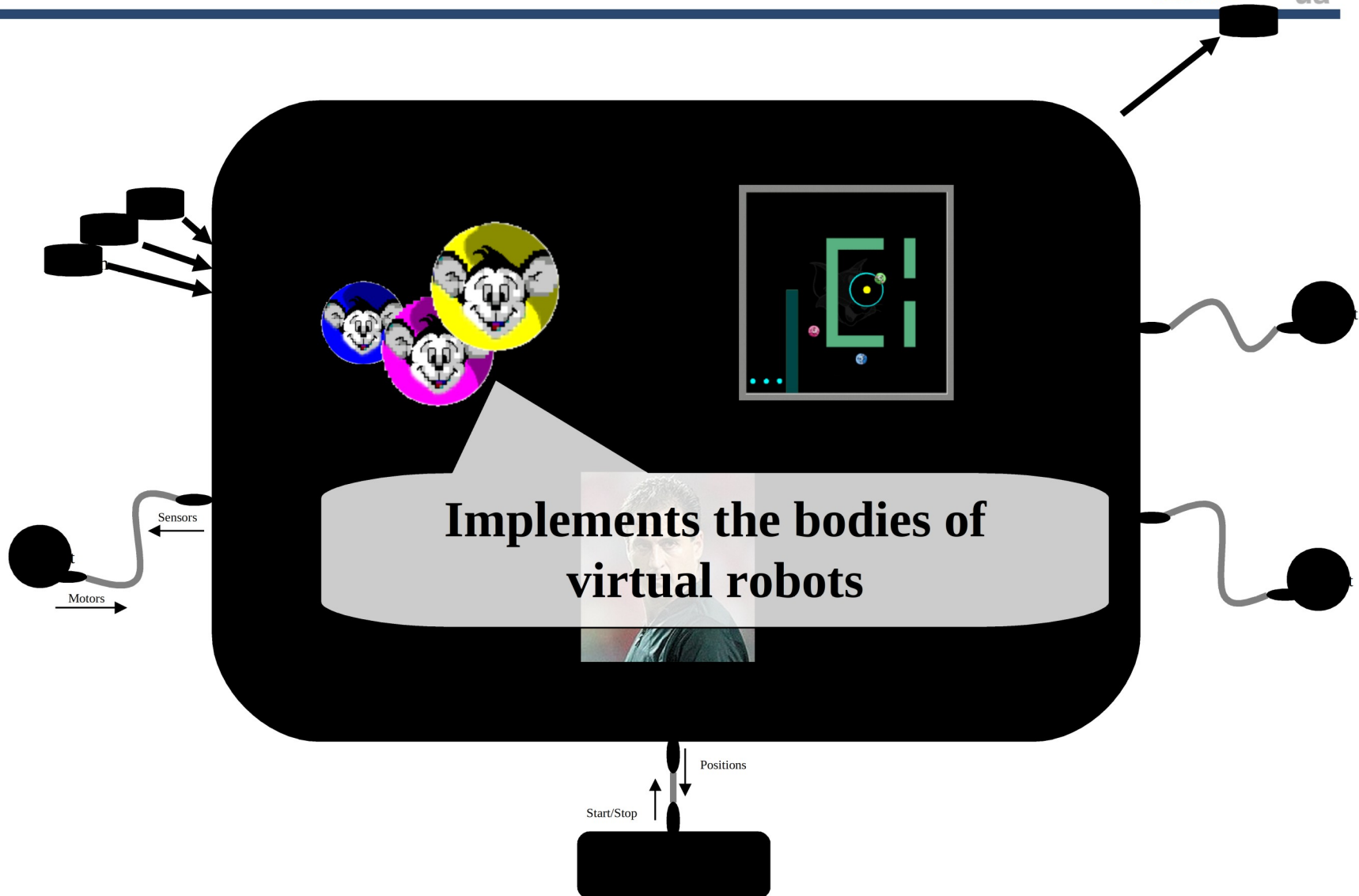
- Client-server distributed system
 - Server: Simulator
 - Clients: Agents and Visualization Tools
- Communication using UDP Sockets
 - XML Messages
- Configuration and Log Files in XML
 - Maze, starting grid, simulation parameters

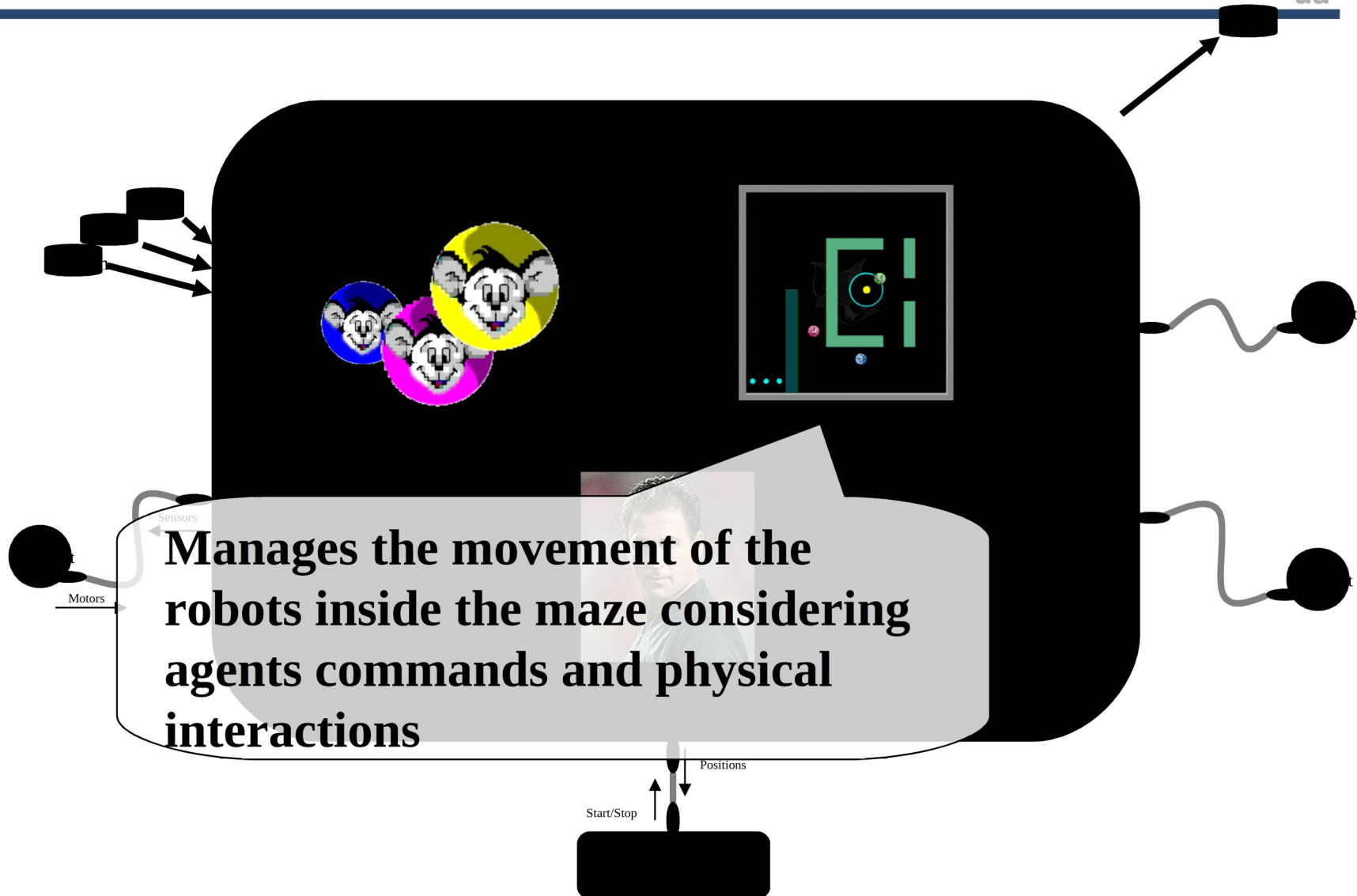
General Architecture

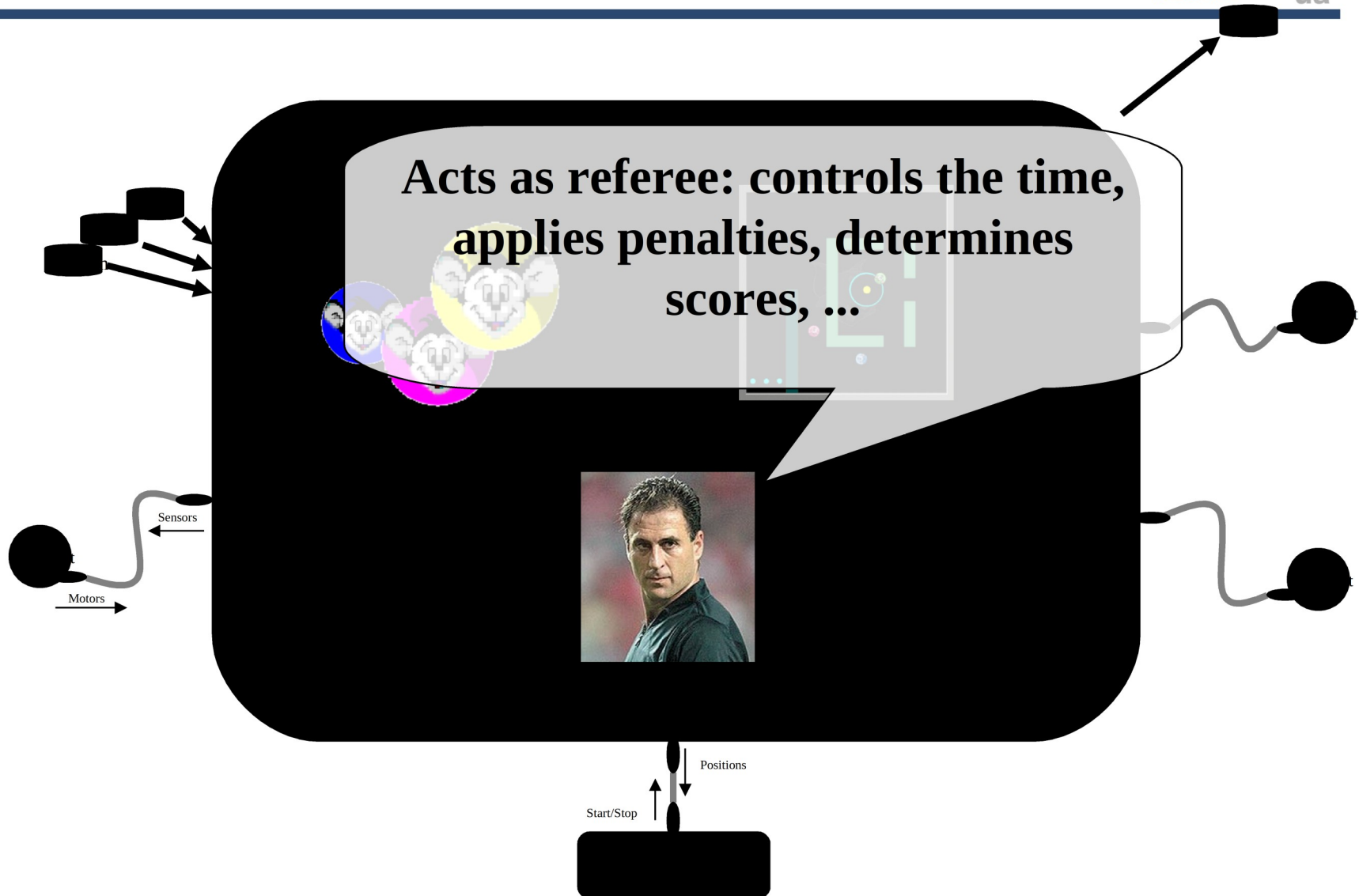


Simulator

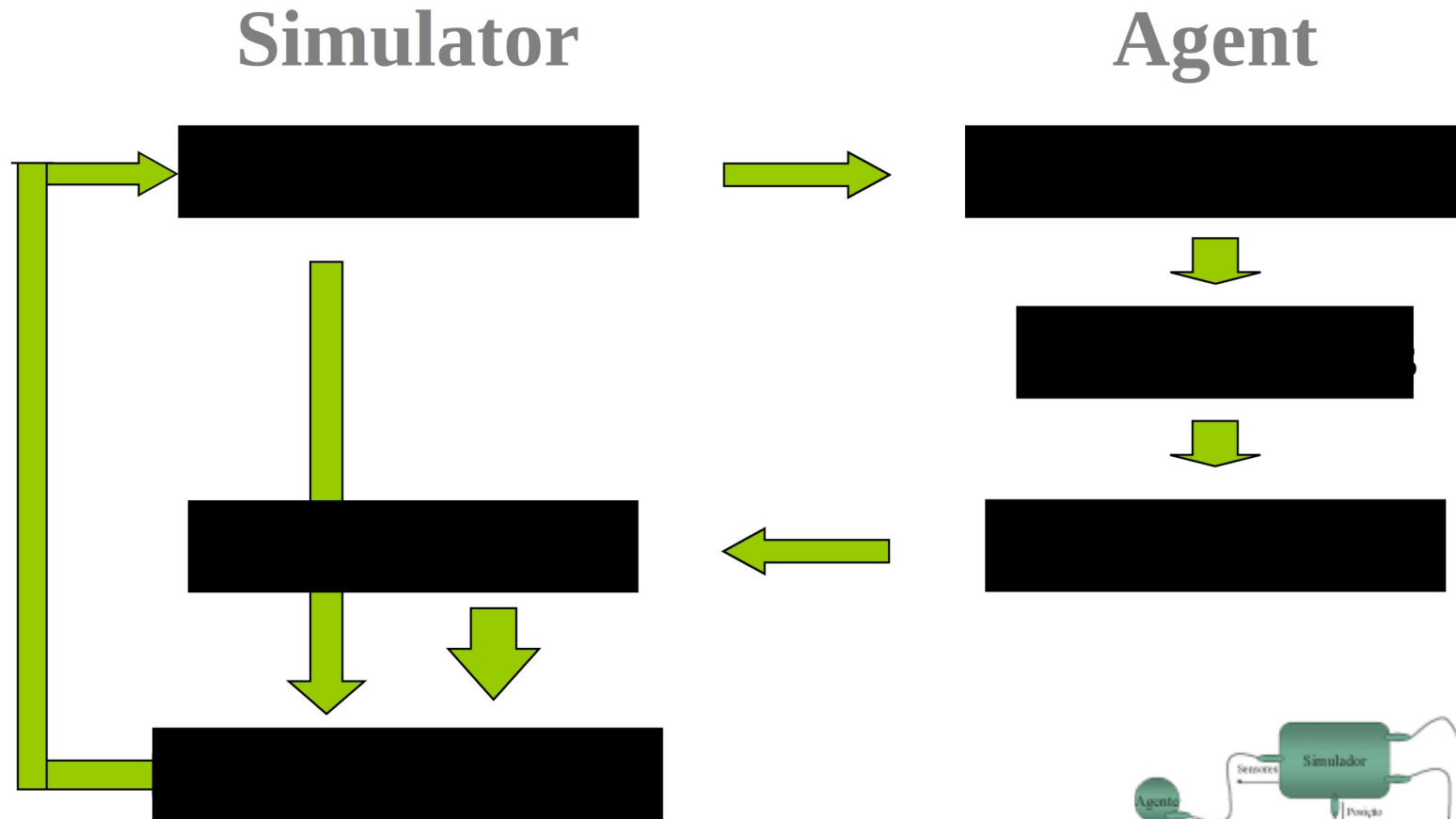






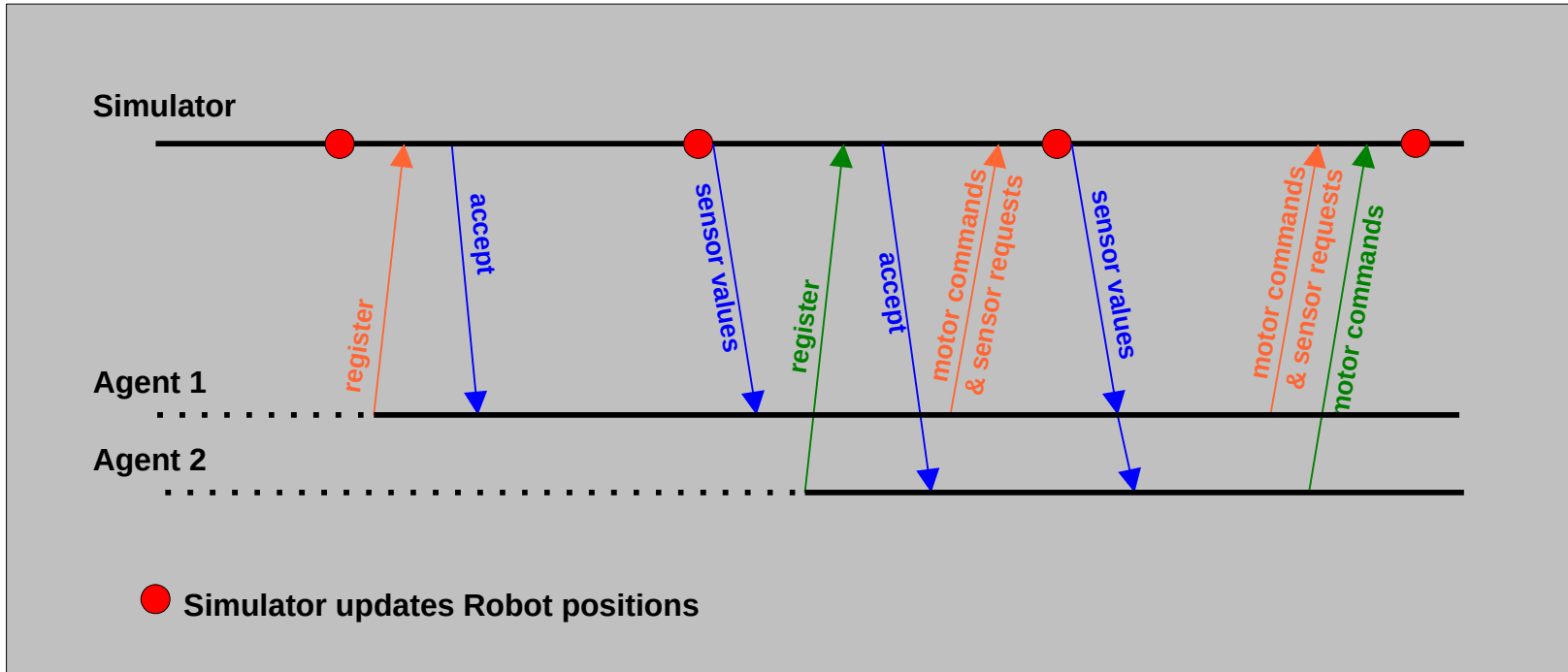


Simulator-Agent interaction

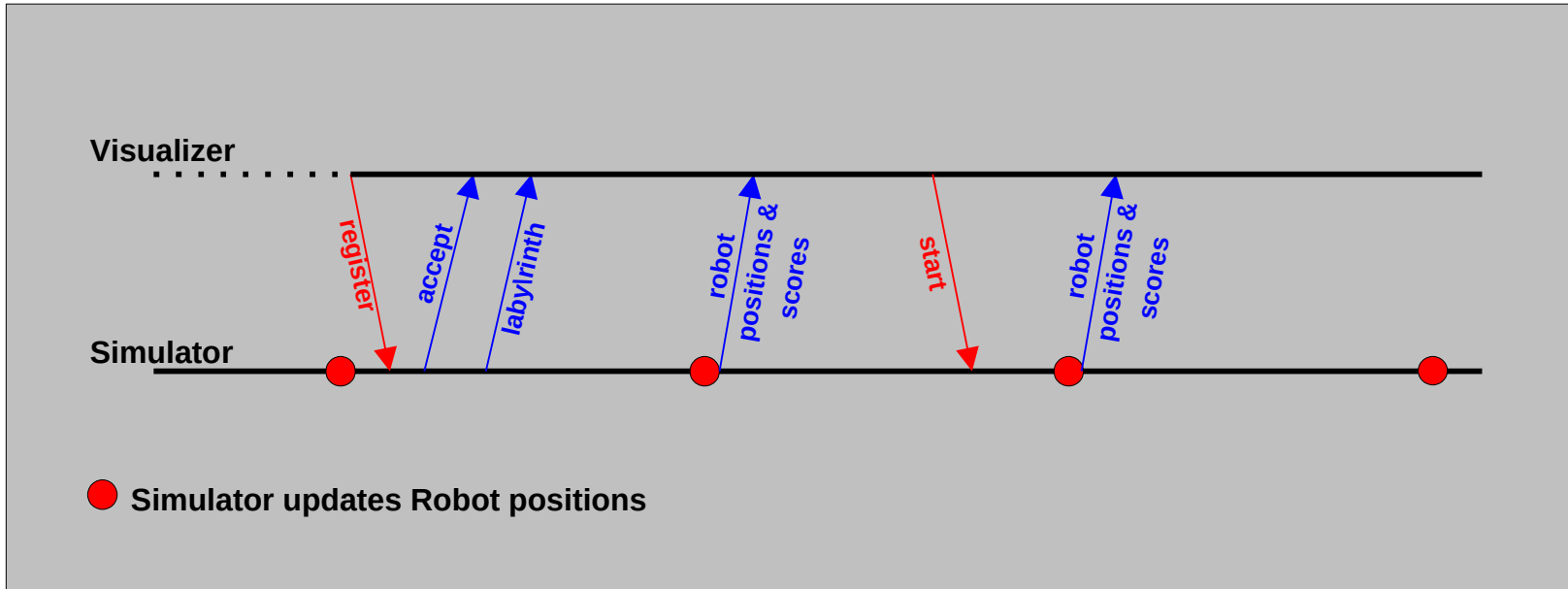


Time is discrete

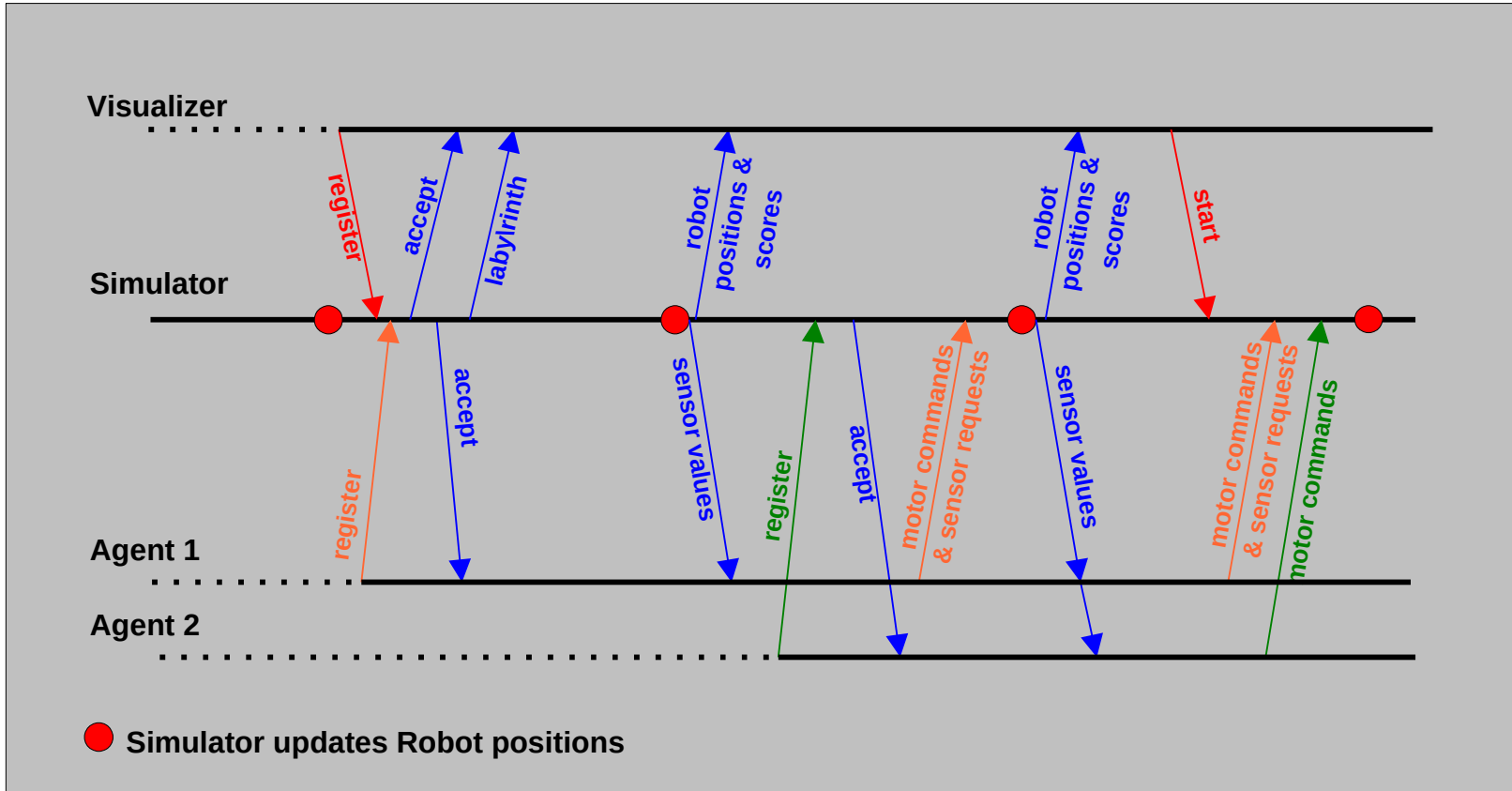
Simulator-Agent interaction



Simulator-Visualizer interaction



Simulator operation



- First order approximation determines linear and rotational displacement of robot

$$lin = \frac{out_{right} + out_{left}}{2} \quad rot = \frac{out_{right} - out_{left}}{diam}$$

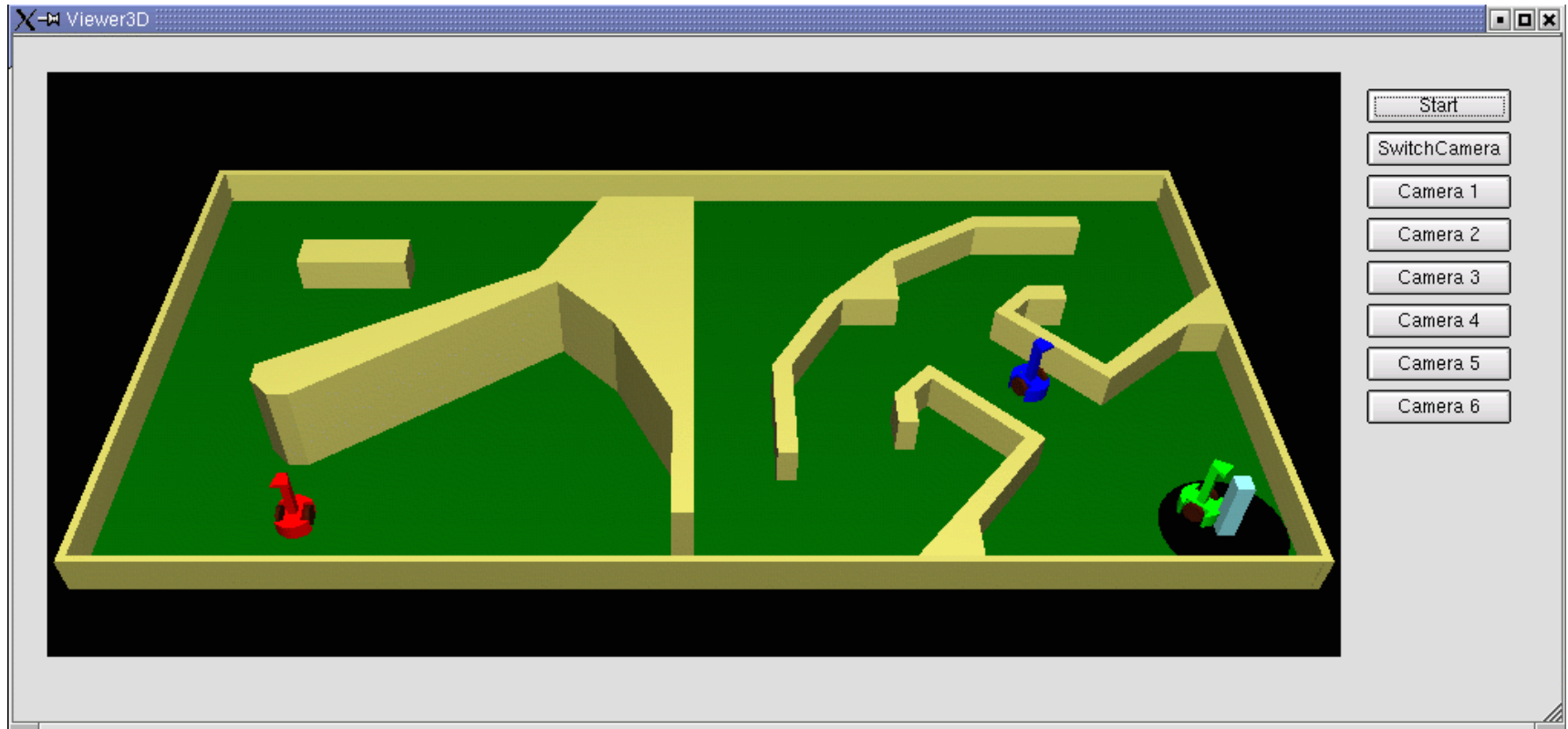
- Inertia is simulated through an IIR filter

$$out_t = (in_t * 0.5 + out_{t-1} * 0.5) * noise$$

Visualization



Visualization



- Programmed in any language/OS
 - Communication libraries in C, Prolog, Java, and Visual Basic are available
 - Participants: Windows/Linux, C/C++/Delphi/VB
- C reactive agent is part of the published tools
- Some development support tools are also available
- Binaries from previous years are available for testing purposes

- Avoiding obstacles

```
while(1)
{
    ReadSensors();
    RequestSensors();
    left  = GetObstacleSensor(LEFT);
    right = GetObstacleSensor(RIGHT);
    center= GetObstacleSensor(CENTER);
    if(center>4.5 || right>4.5 || left>4.5 )
        DriveMotors(0.06, -0.06); /* Rotate */
    else
        DriveMotors(0.1, 0.1);    /* GO */
}
```

- Agent Architecture
 - How should the agent be structured?
 - Which are the modules of the agent?
- Localization
 - Where is the robot?
- Mapping
 - Build a map of the maze from past experience
 - Find the beacon position
- Path planning
 - Find the shortest path to the next beacon, or to the starting position
- Sensor Fusion and communication
 - Best estimate from several sensors and communicated info

- Navigation
 - Avoid obstacles
 - Detect cycles
- Plan execution and monitoring
 - Which is the strategy?
 - Explore or return?
 - Is the planned path possible?
- Agent development/debugging/tuning
 - Tools to foster the development