Collabot

A Collaborative Robotic Agent for the CiberRato Competition

Pedro Pontes and Tiago Varela

Faculdade de Engenharia da Universidade do Porto

Abstract. This paper discusses the implementation of the navigation and target localization strategies for a collaborative robotic agent using the Ciber-Rato Simulation Tools. Nevertheless, communication and mapping strategies were also implemented in the discussed agent, however those implementations are further explained in the paper [**?**]. To this end, we particularly focus on developing an interesting solution for a maze solver agent and explore the collaboration between robots with similar architectures.

1 Introduction

Ciber-Rato is a modality of the Micro-Rato contest from University of Aveiro. This modality is a competition between small autonomous robots trying to solve mazes. [1] In this project, we faced the challenge of developing a collaborative robotic agent architecture using the Simulation Tools created for the Ciber-Rato competition. The agents should be able to solve simple mazes by finding a beacon and returning to his original position, while avoiding obstacles, collisions and dealing with time constraints. Furthermore, in the collaborative competition, the agents are playing in a team of 5 robots, all the robots, must meet in the target area, and after that return to their original position. The maze is only considered solved when all the mice return to their original position.

At the starting point, the agents have no previous information about the world state, namely the target position, maze's topology and even his or other mice positions. Therefore, a simple reactive robot architecture was not suitable for this problem and so proper communication, mapping, navigation and target localization strategies were be developed in order to maximize the agent efficiency.

In this paper, first we present the simulation system architecture. Then the agent architecture and design are analyzed, mainly focusing in the navigation and target localization strategies, since communication and mapping strategies are further discussed in the paper [**?**]. After that, the results of the developed strategies are discussed. Finally, in the last section, are presented the main conclusions and some possible future developments.

2 Simulation System Architecture

To develop this project the Ciber-Rato Simutalion Tools, 2012 edition, were chosen as a simulation platform. This platform allows the developers to focus only on the development of an efficient agent algorithm, eliminating the problems and challenges associated with real robots construction by providing a simulation environment that models all the hardware components of the robots and a allows the developed algorithms to be tested[1].

The simulation environment is composed by a Simulator, a Simulation Viewer and Virtual Robots. The first is responsible for modelling all the hardware components of the robots, the maze and ensure that all the execution rules are applied. The Simulation Viewer displays the maze, the robots movements and the remaining execution time. The virtual Robots are detailed in the section bellow.

All the specifications presents in this paper are based on the Ciber-Rato 2011 Rules and Technical Specifications [2], and only a brief specification is present in here, for a more detailed specification please consult the document mentioned above.

2.1 Virtual Robots

The virtual robots have circular bodies and are equipped with sensors, actuators and command buttons. Only the robots' sensors and actuators used by the agent that we developed are mentioned in this section.

Sensors
For the developed agent the following sensors were used, from the ones available in the simulation environment:

Obstacle Sensors 4 proximity sensors, 3 oriented to the front of the robot (left, middle and right sides), and one in the rear. Each sensor has a 60 aperture angle.

Beacon Sensor Measures the angular position of the beacon with respect to the robots frontal axis. The measure ranges from 180 to +180 degrees, with a resolution of 1 degree.

Bumper Active when the robot collides.

Ground Sensor Active when the robot is completely in target area.

Compass Positioned in the center of the robot and measures its angular position with respect to the virtual North (X axis).

GPS Returns the position of the robot in the arena, with resolution 0.1. It is located in the center of the robot.

Actuators
The actuators components of the robots used in this project are 2 motors and 1 signalling LED.

Motors Motors have inertia and noise in order to more closely represent real motors, and the translation or rotation movements can be achieved by applying different power values to each motor.

LED The LED is used to signal that the robot has already found the beacon.

Buttons
Two buttons, named Start and Stop, are provided in each robot and are used by the simulator to start and interrupt the competition.

2.2 Arena

The arena is randomly positioned in the world, which means that the starting coordinates of the robot may differ for every attempt to solve the maze, and has a maximum size of 14x28 um. The arena is populated with obstacles, a target area, and a starting grid. For the same maze different starting grids can also be used. The obstacles within the arena can be higher then the beacon, making it invisible for the beacon sensor.

2.3 Communication

Communication between robots can be made by sending appropriate commands, through the simulator. The other agents will be then responsible for reading the messages in the simulator. However the following constraints are be applied:

- Per cycle, a robot can send (broadcast) up to 100 bytes;
- Per cycle, a robot can read up to 400 bytes;
- Per cycle, a robot can read up to 400 bytes;
- Robots can only read messages sent from a maximum of 8 units from its current position;
- Obstacles do not interfere with communication;
- Latency of 1 cicle for sent messages.

3 Agent Architecture

In this project we faced the challenge of creating a completely autonomous and purely reactive robot. In order to achieve that goal our agent must be able to fully operate in an unstructured environment by avoiding obstacles and finding the beacon in a simple to moderately complex map. This purely reactive architecture means that the robot acts merely based on its sensors, i.e. it does not keep information on how the current world state is.

To accomplish this we developed a layered architecture that can be viewed on figure 1.

With this system architecture the top layers are activated when certain conditions are detected by the sensors, overriding the actions of the bottom layers,

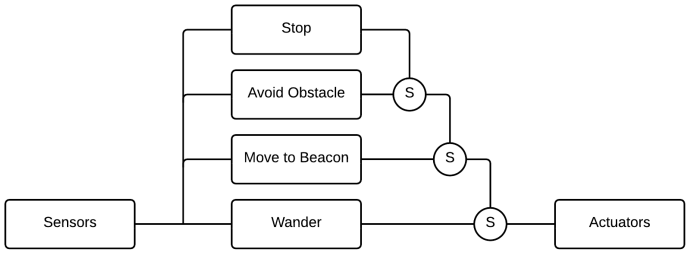

Fig. 1. Layered architecture.

e.g. if an obstacle is detected the robot will start its obstacle avoidance routine, instead of moving towards the beacon or wandering around.

This architecture was inspired by the subsumption layered model that Brooks proposed for mobile robot control system.[3]

Each of these layers will now be described in further detail:

Stop This layer is used to detect if the robot has reached the beacon location and to stop its motion, also signalling to the simulator that the robot has finished the course.

Avoid Obstacle In this layer we detect if an object is close to the robot by comparing the sensor measurement with a threshold that we defined. If the sensors exceed this threshold we proceed to the avoiding obstacle routine. In this routine we check the sensors' values and proceed accordingly:

If the threshold is exceeded on the frontal sensor we avoid to the right with an 85% probability and to the left with 15% probability. In the beginning we had a 50% probability for each side but we encountered a problem where the robot steered to right and then the to the left on following iteration eventually colliding with the wall.

If the threshold is exceeded on one of the lateral sensors the robot avoids the wall by turning to the opposite direction.

The lateral sensors threshold is lower than the frontal one to give them priority when avoiding an obstacle.

Move to Beacon This layer detects if the beacon is visible and if it is the robot rotates itself to align with the beacon direction and then moves forward. This rotation value is calculated accordingly to beacon sensor information, e.g. if the beacon direction is opposite to the robot's facing direction then the rotation value is higher than if the beacon direction was only to the left or to the right of the robot.

An important component of this layer is that it is only active on every 50 iterations and lasts 10 iterations, i.e. it's active on iterations 50 through 60, 100 through 110, 150 through 160 and so forth. This is done to avoid cycles between the layers "Move to Beacon" and "Avoid Obstacle", with this in

place the robot does not try to move to the beacon right after avoiding an obstacle, allowing it to somewhat contour the obstacle.

Wander This layer provides the default action which is for the robot to move forward, veering to the left or to the right every 100 iterations, creating an undulatory locomotion similar to that of a snake.

3.1 Limitations and Benefits

This reactive architecture includes some limitations, such as:

- if the robot is following the beacon and there is an concave obstacle in the way, most of the times, the robot will be stuck around the same place with conflicting orders to follow the beacon and to avoid the obstacle in front. The rule to only follow the beacon on every 50 iterations was developed to try to circumvent this cases and provided promising results;
- by not being able to store the world state the robot wanders around the map without knowing if it has already been there, and this can also create cycles, specially if the robot is surrounded by walls and there is only one way out.

The benefits of this type of architecture are that the robot is able to quickly react to dynamic changes in the environment, such as the introduction of a new robot in the system, and also able to react to some environment variable that the robot's programmer has not foreseen.

4 Conclusion

With this assignment we got some good insights on the Ciber-Rato Simulation Tools and we were able to get properly familiarized with the development environment. Some time was lost on finding better documentation for the provided Java interface and this was only solved by adding some debug messages to the provided client.

Also with this simple agent we realized the limitations of a purely reactive architecture and tried different methods to circumvent them.

References

[1] Lau, N., Pereira, A., Melo, A.: Ciber-rato: Um ambiente de simulaçao de robots móveis e autónomos. Revista do DETUA **1**(6) (2002) 5–8

[2] Departamento de Electrónica, T.o.e.I., de Aveiro, U.: CiberRato 2011 Rules and Technical Specifications. (2011)

[3] Brooks, R.: A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation **2**(1) (1986) 14–23