Brain: A Deliberative CiberMouse Agent

Luis Rei*
Faculty of Engineering
University of Porto
Rua Dr. Roberto Frias, s/n, 4200-465, Porto,
Portugal
luis.rei@gmail.com

Luis Paulo Reis**
Faculty of Electrical Engineering
University of Porto
and
LIACC - Artificial Intelligence and Computer
Science Lab of the University of Porto
Rua Dr. Roberto Frias 4200-465, Porto, Portuga
lpreis@fe.up.pt

***Abstract*— The challenge of the CiberMouse competition is to design an agent that controls a virtual robot which is part of a team of virtual robots. The team must locate a target area within a labyrinth and position itself within it. This paper describes the the overall design and specific components of the Brain agent focusing on mapping, path planning and cooperation between several brain agents, including changes made after the CiberMouse challenge at the Real-Time System Symposium in 2008. The results show that cooperation between homogeneous agents can be valuable. Notable solutions used is the A* search algorithm, probabilistic mapping and message hoping. The results showed that cooperation between homogeneous agents can be valuable, enabling to achieve very good results when solving several distinct CiberMouse mazes.**

I. Introduction

CiberMouse is a set of simulated robotics competitions, held by the University of Aveiro, using a particular simulation system composed by a 2D virtual environment, a maze with obstacles and targets signaled by beacons, and one or more robots initially placed in a predetermined starting position. The competitors provide the software that controls the robots in order to for them to accomplish a set of predefined goals which usually include reaching target areas [1]. The CiberMouse@RTSS08 [2] was a CiberMouse competition held at the IEEE Real-Time Systems Symposium in 2008 [3]. The goal was for a team of 5 homogeneous robots to find a target area inside an arena with obstacles in unknown positions and position themselves within it as quickly and with as few collisions as possible. The robots must be controlled by independent programs which may be identical. Brain is such a program. The robots can communicate, thus it is possible for them to explore different regions of the arena and communicate their findings to each other. Brain implements this strategy: each robot attempts to find the target area in a different location and once one of them finds it, it communicates its location to the others which must then get to it. In addition, Brain implements basic cooperative map building. Section 2 will provide a summary of the CiberMouse@RTSS08 challenge. Section 3 looks at the overall architecture of Brain. Section 4 will look in depth at mapping. Section 5 looks at planning a path between two points in the generated map and how the agents follow that path given the various uncertainties associated with the process. Section 6 deals with communication and cooperation in a team of homogeneous agents. Section 7 looks at the tools developed to help debug the software. Section 8 presents and discusses results obtained. Finally, section 9 suggests possible improvements to Brain.

II. Problem Formulation

The CiberMouse@RTSS08 can be divided into 4 major components: the simulation system, the virtual robot, the arena and the communications between the robots. A good solution to the problem means overcoming a set of challenges in autonomous robot control [7] such as world mapping, path planning, error control and exploration.

A. The Simulation System

The simulator is responsible for implementing the virtual bodies of the robots, estimating their sensor measurements, communicating with the appropriate software agent controlling the robot, enforcing the restrictions on the movement and communications of the robots and keeping the score. Because everything is simulated, no real units are used. Length is measured in *um* and time intervals are measured as multiples of the cycle time, denoted *ut*. 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. Each cycle lasts 25 milliseconds. Thus an agent has at most 25 milliseconds to read the sensor information, deliberate and send actuator commands to the simulator. Thus the software must make a compromise between the quality of the deliberation and how quickly that deliberation is made.

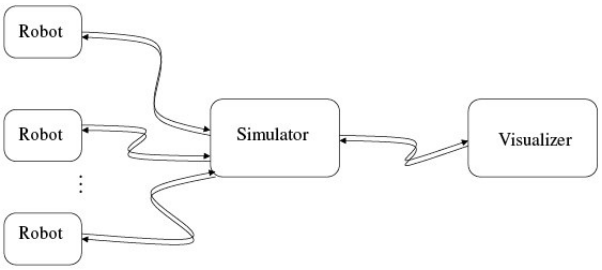

Fig. 1. The simulation system. Adapted from [1]

Aside from the simulator itself, there is a visualizer that graphically shows robots in competition arena, including their states and scores and allows to control the simulation through start and stop buttons. Both the software agents that control the robots and the visualizer connect to the simulator via the network and communicate using XML messages.

B. The Virtual Robot Body

The main aspects of the robot bodies are their circular shape, 1 *um* wide and their sensors and actuators. Each robot has 4 obstacle sensors, 1 beacon sensor which gives the direction and of the beacon within a limited distance, 1 compass, 1 collision sensor (bumper), 1 ground sensor and a GPS. Some sensors are always available, namely the GPS and bumper while the others are only available on request, with a limit of 2 per cycle. All sensor measures are noisy and some are affected by latency, namely the beacon and compass sensors. Brain does not use the compass. The robot bodies also have 2 motors, affected by inertia and noise. Figure 2 shows a schematic of the virtual robot.

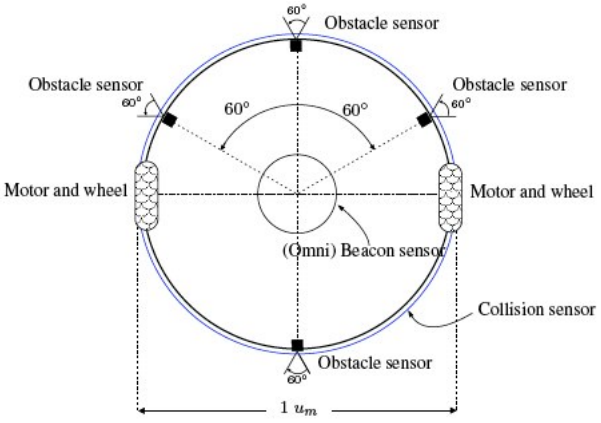

Fig. 2. Virtual Robot Schematic. Adapted from [1]

The relation between the power sent by the agents and the actual power applied to the wheels is given by the following equations:

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

where *lInPowt* and *rInPowt* 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; $OutPow_t$ 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 is implemented with both linear and rotational components based on the power applied on the wheels according to the following equations:

$$lin_t = (lNoisyOutPow_t + rNoisyOutPow_t) / 2$$
$$rot_t = (rNoisyOutPow_t - lNoisyOutPow_t) / diam$$

where lin_t, given in *um*, 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* and *diam* is the robot diameter.

C. The Arena

The arena is 14 *um* high and 28 *um* wide. The obstacles placed inside are at least 0,4 *um* wide and some are higher than the beacon, making it impossible for the beacon sensor to detect the beacon even if it is within range. Any passage between obstacles is at least 1.5 *um* wide. The target is at least 2.0 *um* wide.

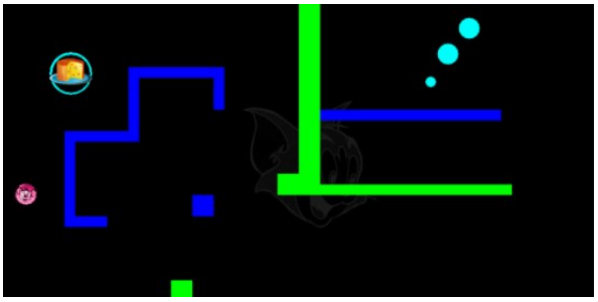

Fig. 3. The CiberMouse@RTSS06 Final Arena

D. Communications

The software agents do not communicate directly: they send the message to the simulator which than broadcasts it to the other agents. Each robot can broadcast one message per cycle and receive all the messages that were broadcasted by its teammates which means the robot can receive up to 4 simultaneous messages. A message can be at most 100 bytes long. The range of communication is limited to 8 distance units from the transmitter. There is a latency of 1 cycle for every communicated message.

E. Scoring

Scoring is a fundamental part of the problem formulation. While the actual objective is for the team of robots to reach a target area, the fact that points are deducted from collisions means that it can be better, in terms of score, for a team of agents to just stand still than to reach the target area with a very large number of collisions. The final score is calculated by the simulator taking into account the accomplishment of goals and any penalties incurred. The lower the score the better, the worst score for an individual robot is 200 points, thus 1000 is the worst possible team score. Each robot starts with 100 points. Points are added for each collision thus it is desirable to avoid collisions while reaching the target area results in the subtraction of points - 100 if reached within a predetermined key time, less if otherwise. Thus it's important

to reach the target quickly. The time to reach the target area can be used as a second criterion to play off.

III. Architecture

Brain is a deliberative agent based on an adapted subsumption architecture [4]. It maintains an internal state which includes information about it's own state and the perceived state of the world. Brain uses the internal state to choose a behavior to execute. Behaviors control the power of the robot's motors. A diagram of this architecture is displayed in figure 4.

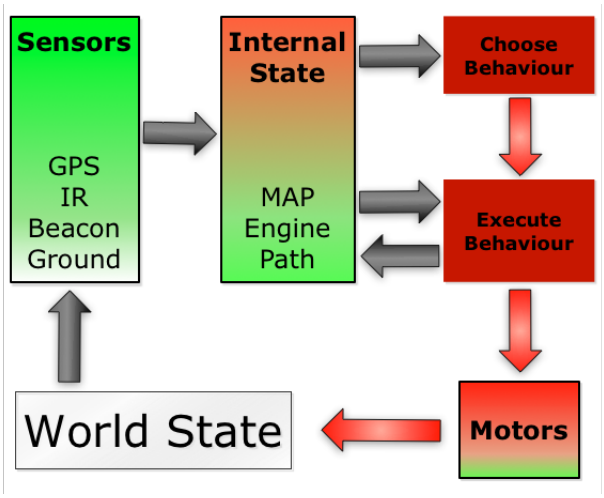

Fig. 4. Brain Architecture Diagram

The agent can be described in pseudocode in the following way:

```
state = updateState(state, sensorData)
Behavior = chooseBehavior(state)
motors = Behavior.execute(state)
```

There are various behaviors, some are simple and basic such as *Rotate* which rotates the robot in place and *MoveForward* which moves the robot forward in a straight line; some are more complex such as *RandomWalk* which moves the robot to a random point nearby and *WallFollow* which makes the robot move along the direction of a nearby wall and some are composite such as *FollowBeacon* which follows the beacon using simpler behaviors, *GoToPoint* which uses multiple *MoveForward* and *Rotate* behaviors (a diagram is shown in figure 5) and *FollowPath* which uses multiple *GoToPoint* behaviors. Another important behavior is *ExploreBehavior* which uses the probabilities in the internal map to determine where to move robot next in order to explore previously unexplored areas of the map.

Behaviors have priorities. A high priority behavior will be executed even if a lower priority Behavior is already executing. The highest priority behavior is AvoidObstacles which attempts to avoid obstacles in the agents path.

The main design choices in Brain, made after the Ciber-Mouse@RTSS08, are to keep it simple and fast, using the least complex approach and making as few computations

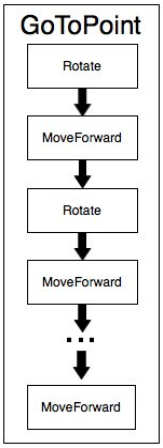

Fig. 5. A diagram of the *GoToPoint* composite behavior. It consists of a sequence of *Rotate* and *MoveForward* behaviors.

as possible. The purpose is to ensure that the deliberation always takes less than the cycle time and that the software is crash free. A crashed software agent can not guide the virtual robot to the beacon or explore the map, worse, it will probably collide. It may even block a path that the other agents need to follow in order to reach the target.

IV. Mapping

Mapping is essentially done using the GPS and obstacle sensors. Brain uses an Extended Kalman Filter (EKF) [5] in the same way other agents such as Jitters [10] to reduce the noise from the GPS. An EKF is a recursive algorithm that estimates the state of a noisy non-linear dynamic system.

A. Probabilistic Mapping

Brain uses probabilistic mapping [5] - an area is not marked as either open or obstacle but rather is assigned a probability of being an obstacle. The map is divided into square cells 0.1 *um* wide (280x140 rectangles). Each cell has associated with it its position, the probability of being an obstacle and the number of times it has been observed either by the obstacle sensors or because the GPS indicated that the robot moved over it. To reduce the error, the agent ignores sensor values smaller than a certain value. The obstacle sensor model used is similar to that of YAM [8]: if a sensor reports an object with a value V bigger than the threshold value, the map is updated with that data: all the cells between the sensor and 1/(V+0.1), where 0.1 is the standard deviation of the error, have their probability of being obstacles decreased (area A); all the cells between that 1/V also have their probability decreased but by a smaller amount (area B); finally all the cells between 1/V and (1/V)+0.4, where 0.4 is the minimum wall width, have their probability increased (area C). The different areas of probabilistic mapping with the obstacle sensors are shown in figure 6. All cells that are located within the area occupied by the robot, as indicated by the GPS have their probability decreased by a significantly greater amount than if they were in area A.

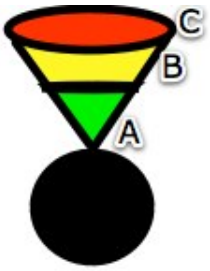

Fig. 6. Probabilistic Mapping With The Obstacle Sensors

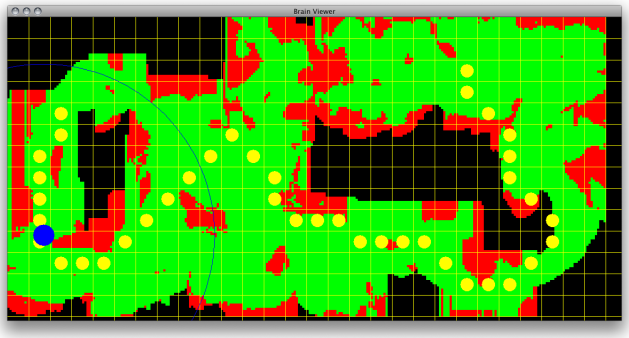

Fig. 7. A plan from the target area back to the starting position in the partially explored CiberMouse@RTSS06 Final Arena. (Brain-Viewer)

Additionally, each cell also has associated with it the probability of being in the target area. This is done with the use of the beacon sensor. If the robot can see the beacon (through the use of the beacon sensor) it knows the direction, of the beacon. The size of the target area that has the beacon at its centre is also known, and through the use of the ground sensor, the robot knows if it is within the target area. The model is somewhat similar to that of the obstacle mapping: all cells in the direction of the beacon, with a given aperture and within the maximum range of the beacon, have their probabilities of being the target area increased while all others have their probabilities decreased. Also the areas occupied by the robot, have their probability decreased by a significantly greater amount if the ground sensor does not indicate the robot has reached the target area.

V. Path Planning and Path Execution

Path or trajectory planning determines the best, usually the shortest, trajectory from the agents current location to a target location. It is a common part of deliberative CiberMouse agents such as [6] and [8]. Brain creates a sequence of waypoints which it will follow from its current location to another location using the A* search algorithm [9] to find the best known path between the 2 points. First, the internal map resolution is reduced, from square cells of side 0.1 *um* to cells of side 1 *um*. The probability of each new cell in the map being occupied is the average of the sum of the probability of the old cells contained within it. This was done in order to speed the execution of the A* algorithm and can be easily changed if, for some reason, it becomes necessary or prudent to do so - the A* algorithm can be applied directly on the internal high resolution map. Secondly, the A* algorithm is applied and the center point of each intermediate cell is used as a waypoint. The heuristic used by the A* is the euclidean distance between the cells and the destination point. FIgure 7 shows an example of a plan made by a brain agent.

Executing the path means moving from waypoint to waypoint in order, from the present location to the target location. There are several problems which may arise due to the different mapping, sensor and actuator errors:

- The robot might miss the exact waypoint, forcing it to turn back. Considering motor error, inertia and GPS error it would possibly have to circle around the point for a while until it managed to actually pass over it.
- The robot may pass over the waypoint without noticing it, later forcing it to turn back.
- The waypoint may be placed inside an obstacle, making it unreachable.

To deal with these problems, Brain takes 3 measures:

- The robot does not need to pass over the waypoint, it only needs to pass next to it with a tolerance of 0.5 *um*.
- The AvoidObstacle Behavior has a higher priority than FollowPath so even if the waypoint is inside an obstacle, the robot will not hit the obstacle. Once the obstacle has been avoided, path following is resumed.
- If the robot finds itself passing over a waypoint which is not the next waypoint but one after, it automatically resumes the path from that waypoint.

The combination of these measures has been shown to work well experimentally.

VI. Communication and Cooperation

The team of virtual robots is homogenous. They are different only in their internal identification numbers that are equal to the number of the position that each robot starts in. These identification numbers can be used to create complex behaviors where each robot knows which task to perform based on his own identification number such as dividing the map into regions to be explored by different robots. This can also be used to identify the origin of a message that is broadcasted.

If the robot hasn't found the target yet, it sends a message every cycle with the current position of the robot, including its direction, and the values of the obstacle sensors. Data from the beacon sensor could also be added. This data is used by the recipients to build their map and is used in the same way as their own sensor data would be used as shown in figure 8.

If the target has been found, the message that is broadcasted contains a set of points that form a path from the initial position of the robot to the target area. Upon receiving such message, other robots calculate the path to the nearest point in the received message and follow the points until they reach the target, while repeating the message they received with the solution.

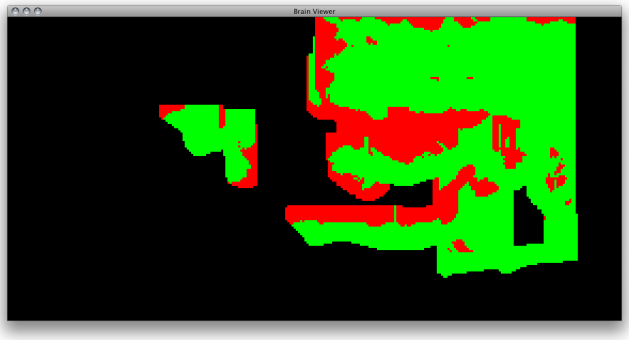

Fig. 8. A mapped area is shown that is not connected to the other parts of the map. This area was not mapped by this agent, rather it was mapped by another agent that then broadcasted it. (Brain-Viewer)

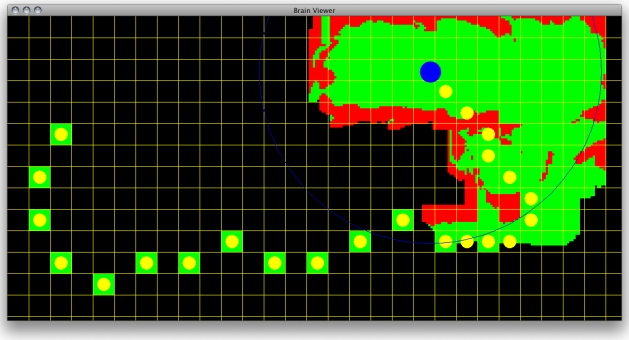

Fig. 9. A path from the start of the map to the target area that was set from a message. The path shown goes through areas that the agent clearly hasn't explored. (Brain-Viewer)

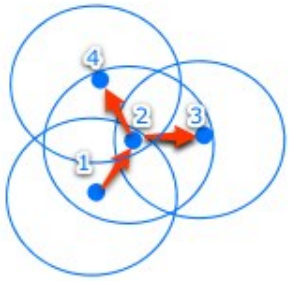

Fig. 10. Robot 2 relaying a message sent by 1 to 3 and 4. The numbered points represent robots, the circles represent their respective communication ranges

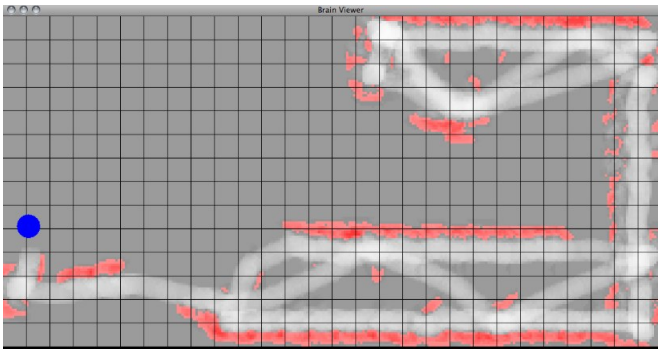

Fig. 11. Brain Viewer displaying the internal map

A. *Message Hopping*

Considering a team of three robots, 1, 2 and 3, there will be situations in which robot 1 is within communication range of robot 2 and robot 2 is in range of robot 3 but robot 1 and 3 are not in range of each other. In that case, robot 2 can relay messages from 1 to 3 and vice-versa. A schematic of this message relaying is shows in figure 10. Brain achieves this by appending the messages that will be relayed to the end of each message it broadcasts if there is enough space left to append the message.

VII. Debugging

In order to debug Brain, two important modules were created: the Internal State Viewer and the Simulation Viewer.

A. *The Internal State Viewer*

The Internal State Viewer, also known as Brain Viewer, is a graphical visualization system that graphically display the state of a Brain agent. Namely it shows the internal map that the agent built, the position of the agent in that map, its communication range and if the agent is following a path, the points in that path.

B. *The Simulation Viewer*

The simulation viewer is a non-graphical software component that connects to the simulator as the CiberMouse's own visualizer would. This allows it to know the exact location of each robot. This module thus can not be used in the actual competition but is important for debugging the EKF. More generally, this module is required for the implementation of any machine learning techniques that require the correct results for training. Furthermore, this module, in conjunction with external scripts, can be used to automate experiments.

VIII. Results and Discussion

A single Brain agent can choose any behavior and execute it within the time of a cycle, even when there are multiple agents running on the same computer. Presently, the EKF is not working properly, as such, the experiments in this section were performed with the GPS error disabled in the simulator. While the advantage of communications in map building can be shown easily, the behaviors which use the agents internal map to explore the arena are not working sufficiently well to provide reliable and useful results. Thus, a single Brain agent reaching the target area is currently a matter of luck. Therefore, the only way to show the advantage of cooperation for reaching the target is to compare the number of times the entire team reaches the target area against the times only part of the team reaches it.

Experiments were performed using ciberTools version 1.5.2.2 with a maximum run time of 5000 cycles. The hardware used was an Apple Macbook 2.1 with a 2.16GHz dual-core intel processor and 3GB of RAM. For every

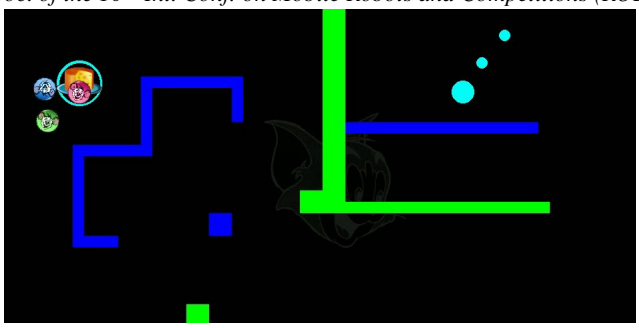

Fig. 12. Three agents reach the target area.

map and for every given number of agents, 10 runs where performed with and without communications.

Map	Communications	# Agents	Finish	Partial Finish	Not Found
CiberRTSS06_Final *	ENABLED	2	4	1	5
		3	6	3	1
		5	**10**	**0**	**0**
	DISABLED	2	1	5	4
		3	0	7	3
		5	**0**	**10**	**0**
CiberRTSS06_Leg1 *	ENABLED	2	3	1	6
		3	10	0	0
		5	**10**	**0**	**0**
	DISABLED	2	1	4	5
		3	2	6	2
		5	**0**	**10**	**0**
CiberRTSS08_Leg1	ENABLED	2	8	1	1
		3	10	0	0
		5	**10**	**0**	**0**
	DISABLED	2	3	5	2
		3	1	8	1
		5	**0**	**10**	**0**

When communications are enabled, if a single agent finds the target it tells the others the path to it. When 5 agents are simultaneously exploring the map, in these tests, at least one agent was able to reach the target area. When communications are enabled, that agent tells the others the path to the target which they are able to reach. When communications are disabled and each agent is on its own, the entire team never managed to reach the target area.

From an architectural standpoint, the choice to build complex and composite behaviors on top of simple behaviors instead of a more simple approach of long if-then-else clauses [11] proved to facilitate both debugging and adding new behaviors ultimately making the core brain software a good starting point for a CyberMouse agent. The Internal state viewer provides also a starting point for viewing the internal state of an agent and the simulation viewer is a necessary piece of software to implement certain machine learning techniques, both can be helpful to create a good agent.

IX. Future Improvements

Currently, the main problems with Brain are that the behaviors which use the internal map to explore the arena are not working. Once they are, coordinated exploration strategies can be explored. In order to better support it, communications should be improved. The use of more efficient data encoding and possibly compression should increase considerably the amount of data that can be transmitted in a single message. A smarter message hopping algorithm which determines wether a message needs to be relayed based on known positions of other agents might also maximize the amount of useful data in each message. The EKF needs to be fixed and alternatives, such as Double exponential smoothing-based prediction (DESP) which might be faster [12] will be considered. A slightly better A* heuristic should provide with significant improvements in path planning. Brain clearly needs better sensor-motor control as experiments showed that it was unable to reliably navigate through maps with very narrow paths, often resulting in collisions with the map walls. A possible solution is to implement machine learning techniques to solve that specific problem, improving obstacle detection and avoidance. Other areas could also be improved by using machine learning, such as optimizing exploration and path following. NetSqueak successfully implemented a path following behavior where movement control of the CiberMouse robot was done by a neural network obtained by evolution with genetic algorithms [13]. Machine learning could also be applied to the development of a cooperative exploration strategy for the team of agents. The creation of external scripts integrated with the Simulation Viewer module will allow for more experiments to be conducted and optimization and machine learning techniques, such as neural networks to be more easily used.

References

[1] Lau, N., Pereira, A., Manual do ciberrato. Technical report, Departamento de Electrnica, Telecomunicaes e Informtica da Universidade de Aveiro, 2008

[2] University of Aveiro, CiberMouse@RTSS08 Rules and Technical Specifications, 2008

[3] The 29th IEEE Real-Time Systems Symposium (RTSS), http://cse.unl.edu/rtss2008/archive/rtss2008/index.php, last consulted January 2010

[4] R. Brooks, "A Robust Layered Control System for a Mobile Robot", IEEE Journal of Robotics and Automation, 2 (1), pp. 14-23, 1986.

[5] Probabilistic Robotics by Sebastian Thrun, Wolfram Burgard and Dieter Fox, 2005, MIT Press, ISBN 0-262-20162-3

[6] Reis, L.P.: Ciber-feup - um agente para utilizar o simulador ciber-rato no ensino da inteligłncia artificial e robtica inteligente, Revista do DETUA, 3(7):655 658, 2002

[7] Lau, N., Pereira, A., Melo, A., Neves, A., Figueiredo, J., Ciber-rato: Um ambiente de simulaco de robots mveis e autnomos, Revista do DETUA, 3(7):647650 (2002)

[8] Pedro Ribeiro. Yam (yet another mouse) - um robot virtual com planeamento de caminho a longo prazo. Revista do DETUA, 3(7):672674, 2002.

[9] Russell, Stuart J. and Norvig, Peter, Artificial Intelligence: A Modern Approach, 2003, Pearson Education, ISBN 0137903952

[10] Cunha, J., Oliveira, L., Ribeiro, L., Sequeira, R., Jitters at CiberMouse RTSS 2008, Proceedings of CiberMouse@RTSS2008, 2008

[11] Alvaro Monteiro, Fabio Aguiar and Sara Carvalho, SpeedyGonzalez: A Path Planning and Plan Execution Agent, Proceedings of CiberMouse@RTSS2008, 2008

[12] Joseph J. LaViola Jr., An Experiment Comparing Double Exponential Smoothing and Kalman Filter-Based Predictive Tracking Algorithms, vr, pp.283, IEEE Virtual Reality Conference 2003 (VR 2003), 2003

[13] Hugo Peixoto, Joo Portela, Rui Teixeira, Filipe Castro and Lus Paulo Reis, Netsqueak on Wheels: Neural Networks Applied to Movement Optimization, in L.S.Lopes, N.Lau, P.Mariano and L.Rocha editors, New Trends in Artificial Intelligence, Proceedings of the 14th Portuguese Conference on Artificial Intelligence, EPIA 2009, pp. 279-287, Universidade de Aveiro, Aveiro, Portugal, October 12-15, 2009, ISBN 978-972-96895-4-3