Intelligent Robotic Mapping and Exploration with Converging Target Localization

João Certo[1,2], João Lobato Oliveira[1] and Luís Paulo Reis[1,2]

[1] FEUP – Faculty of Engineering of the University of Porto, Portugal;
[2] LIACC – Artificial Intelligence and Computer Science Lab., University of Porto, Portugal;
joao.certo@fe.up.pt, jmldso@gmail.com, lpreis@fe.up.pt

Abstract. This paper presents different methodologies for a maze solving agent in unknown environments. The developed methodologies include an odometry based localization system, a converging target localization based on the centre of mass of recursive triangulations, and a regressive obstacle distance function, along with an obstacle setting and sweeping mechanism for mapping. A quad-tree solution was used for map representation, and an innovative exploring A* for path planning was developed. A quasi-reactive, wall-following agent serves as a comparison basis and a simulation environment called ciber-rato was used to test the implemented architecture and validate the developed methods within gradually increasing difficult scenarios. Besides the reactive agent being typically less time consumptive, the exploring architecture granted greater consistency and robustness. The results also evinced the time-improvement of using the exploring A* and the efficiency of this target localization method as gradually converging to the real target position.

1 Introduction

Cyber-Mouse (Ciber-Rato) is a modality included in the "Micro-Rato" competition, directed to teams interested in the algorithmic issues and software control of mobile autonomous robots [1]. This modality is supported by a software environment, which simulates both robots and a labyrinth [2]. The Cyber-Mouse has a body diameter of 1 mouse unit (*Um*) and seven available sensors although only two, user selectable, can be used at any given time. The final purpose is to reach the cheese, identified by a ground sensor and detectable through a direction providing beacon sensor visible trough low walls. Mouse's performance is evaluated through success on reaching the cheese, the time it took and the number of collisions.

The cyber-mouse competition has been used, amongst other applications as a testbed for long-term planning [3], as a scenario for the detection and avoidance of dangerously shaped obstacles [4], or even as a tool for the teaching of Artificial Intelligence and Robotics [5]. In this paper we evaluate the problems of mapping, localization and path planning by building a deliberative agent that can find its way from a starting position to the target without prior knowledge of the maze. Ultimately this architecture's performance is compared to a reactive approach.

The paper structure is as follows. The next section discusses the problems of mapping and self-localization, navigation and path planning together with some

related state of the art algorithms leading to the chosen approaches. Section 3 presents the developed methodologies. Section 4 contains a description of a comparative reactive agent, the testing environments and the respective results. Finally, section 5 concludes this paper and points to future work.

2 Robotic Mapping and Planning Overview

Mapping is the process of building an estimate of the metric map of the environment [6]. The mapping problem is generally regarded of most importance in the pursuit of building truly autonomous mobile robots, but still mapping unstructured, dynamic, or large-scale environments remains largely an open research problem.

Planning is the process of deciding which route to take based on and expressed in terms of the current internal representation of the terrain. Typically this process calculates the cost of each motion decision towards the target, based on a given heuristics, and chooses the "cheapest" one.

2.1 Mapping and Localization Problem

To acquire a map, robots must possess sensors that enable it to perceive the outside world. Sensors commonly brought to carry out this task include cameras; range finders (using sonar, laser or infrared technology), radars, tactile sensors, compasses, and GPS. However, all these sensors are subject to errors, often referred to as measurement noise, and to strict range limitations.

So, considering these issues several different challenges can arise to robotic mapping: statistically dependent sensors measurement noise, high dimensionality of the entities that are being mapped, data association problem (determining if sensor measurements taken at different points in time correspond to the same physical object), environments changing over time and robot exploration.

The motion commands issued during environment exploration also carry important information for building maps, since they convey information about the locations at which different sensor measurements were taken. Robot motion is also subject to errors and the controls alone are therefore insufficient to determine a robot's pose (location and orientation) relative to its environment. If the robot's pose was known all along, building a map would be quite simple. Conversely, if we already had a map of the environment, there are computationally elegant and efficient algorithms for determining the robot's pose at any point in time. In combination, however, the problem is much harder.

Considering the map representation problem, which has a significant impact on robot control [7], we can account for three main methods: Free space maps (road mapping), as spatial graphs, including Voronoi diagrams, and generalised Voronoi diagrams; object maps; and composite maps (cell decomposition) as point grids, area grids and quad trees.

Virtually all state-of-the-art algorithms for robotic mapping in the literature are probabilistic. They all employ probabilistic models of the robot and its environment relying on probabilistic inference for turning sensor measurements into maps [6].

2.2 Navigation and Path Planning

In artificial intelligence, planning originally meant a search for a sequence of logical operators or actions that transform an initial world state into a desired goal state [8]. Robot motion planning focuses primarily on the translations and rotations required to navigate, considering dynamic aspects, such as uncertainties, differential constraints, modelling errors, and optimality. Trajectory planning usually refers to the problem of taking the solution from a robot motion planning algorithm and determining how to move along the solution in a way that respects the mechanical limitations of the robot.

The classic path planning problem is then finding a collision-free path from a start configuration to a goal configuration, in a reasonable amount of time, given the robot's body constitution and the map representation, as retrieved in the mapping process.

In an unknown environment the mapping and motion planning must be processed in parallel through exploration and dynamic navigation decisions. This structure requires plans updating. A natural way of updating plans is to first select a path based on the present knowledge, then move along that path for a short time while collecting new information, and re-planning the path based on new findings.

Considering the application many algorithms have been proposed for path planning: A and A Star (A*), Dijkstra, Best-First, Wavefront Expansion, Depth-First Search, Breadth-First Search.

3 Exploring Agent Methods and Architecture

The architecture of our exploring agent is presented in four independent modules, concerning the self-localization, target (goal) localization, mapping and navigation, and path planning problem. These modules were integrated to solve various mazes facing an unknown environment without any previous knowledge.

3.1 Self-Localization

The self-localization is based on the robots' odometry which is defined by a dynamic inertial movement model [9]. Due to Gaussian noise the simulator model induces a linear motion maximum error given by Eq. 1.

$$\delta \leq \frac{Max(MotorPow) * NoiseDeviation + MotorResolution/2}{Max(MotorPow)} \quad (\%) \qquad (1)$$

As such, for each position estimate there is a maximum δ deviation for the Cartesian coordinates and 2*δ for the rotation angle. The simulator defines *Max(MotorPow)=0.15*, *NoiseDeviation=1.5%* and *MotorResolution=0.001*; which infers δ≈1.83% and a rotation error of 3.66%, acceptable for this application. In order to correct cumulative odometry rotation errors, the compass is read every 50 cycles, always accounting for the compass sensor latency of 4 cycles.

3.2 Target Localization– Centre of Mass from Recursive Triangulations

In order to define an initial plan of attack two methodologies were considered for marking the initial target position. If the beacon isn't visible a random function defines its disposition as being within any of the four map corners, changing every user-defined number of cycles (usually 500) to one corner different from the former. As soon as the beacon is visible an initial rough estimation of the target is calculated by intercepting the line defined by the mouse and the beacon points with the map bounds. From the resultant 4 points the initial target position is considered as being the convergent point (in the beacon direction – see Eq. 2 closer to the mouse.

A more accurate target position is recursively estimated by computing the centre of mass resultant from successive triangulations; as illustrated in Fig. 1. To achieve this, within every 25 cycles, if the beacon is visible, a line Y_A is traced along the target direction. In order to retrieve the real beacon direction, α, this process accounts for the beacon sensor's latency of 4 cycles. Each line is then intercepted with each of the former ones resulting in a conjunction of intercept points P_i (X_i, Y_i). In order to restrict these points to the ones converging to the target Eq. 2 was applied:

$$\begin{cases} P_i = P_i \; if\big((-90° < \theta < 90°)\; AND\; X_i > X_A\big) \\ P_i = P_i \; if\big((-180° < \theta < -90°\; OR\; 90° < \theta < 180°)\; AND\; X_i < X_A\big) \\ else\; Remove\; Pi \end{cases} \tag{2}$$

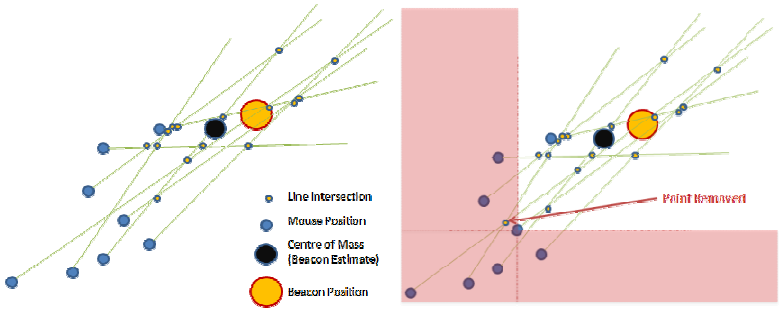

Fig. 1. Target localization from recursive triangulations (left to right): method overview **(a)**; removing intercept points displaced from the mouse-target square, at a mouse position **(b)**.

All the points outside the map bounds are also filtered. After having 5 valid intercept points, while the mouse moves towards the estimated target, every point displaced from the mouse-beacon square (see Fig. 1b), along with the ones distancing more than 3*Um* from the current target estimate, are removed from the list.

$$\begin{cases} X_C = \left(\sum_{i=0}^{N_{points}} X_i\right)/N_{points} \\ Y_C = \left(\sum_{i=0}^{N_{points}} Y_i\right)/N_{points} \end{cases} \tag{3}$$

Finally the target position, given by Eq. 3, is calculated as the centre of mass of the intercept points in the given time. Whenever a new line, Y_A, is traced the target position is re-calculated, granting a robust target estimation, which is enhanced as the mouse moves towards the target.

3.3 Mapping

Our mapping algorithm is based on a deterministic model representing the distance to an obstacle given the obstacle sensor value. The map is represented using a multi-resolution quad-tree decomposition since this representation grants good performance for this application, with low processing cost.

In the presence of an obstacle we used a quad-tree gridding to subdivide each of the obstacle cells. Our quad-tree strategy uses an adaptive division depth to a deepest cell size (granularity) of 0.1*Um*. Fig. 2 illustrates different granularities for different known maps.

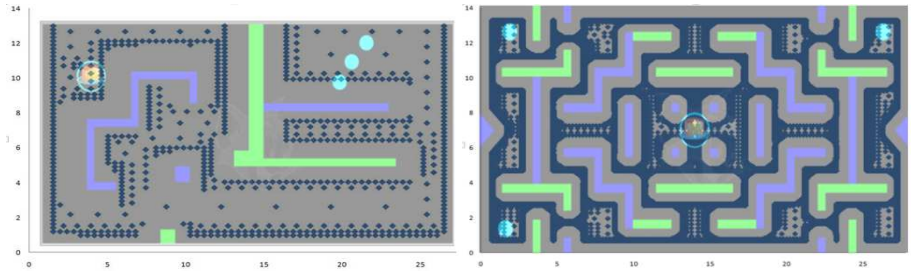

Fig. 2. Real map overlapped with quad-tree representation (left to right): RTSS06Final with 0.7*Um* depth **(a)**; 2005Final with 0.1*Um* depth **(b)**.

3.3.1 Obstacle Detection

The navigation and consequent mapping is based on the obstacles disposition along the map. To calculate the robot's distance to an obstacle relative to its sensor values, a series of successive experimental measurements were taken as seen in Fig. 3.

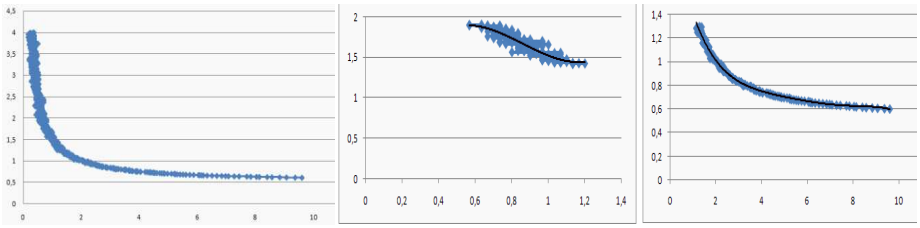

Fig. 3. Obstacle distance distribution – obstacle sensor values in horizontal axis (units); obstacle distance in vertical axis (mouse units) (from left to right): the full distribution **(a)**; distribution for sensor values ranging from 0.9 to 1.0 **(b)**; distribution for sensor values ranging from 1.1 to 4.5 **(c)**.

Through linear regression it was possible to obtain the following equation (Eq. 4) where d is the distance and x the sensor values.

$$\begin{cases} d = 4.1981x^3 - 10.84x^2 + 8.1978x - 0.0403, & if\ 0.9 < x < 1.0 \\ d = -0.0001x^5 + 0.0046x^4 - 0.0561x^3 + 0.3435x^2 - 1.095x + 2.2027, & if\ 1.1 < x < 4.6 \\ d = 0.6, & if\ x > 4.6 \end{cases} \quad (4)$$

These functions estimate the obstacle distance (from the mouse's body centre) with a maximum error, δ, of $0.213Um$ for sensor values in the range of 0.9-1.0, and $0.189Um$ for values in the 1.1-4.5 range.

3.3.2 Obstacle Setting and Sweeping

The sensor available in the cyber-mouse simulation system has an aperture angle of 60º, as depicted in Fig. 4a.

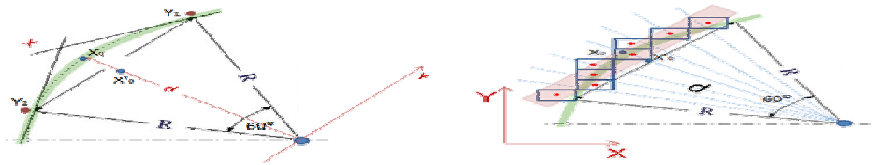

Fig. 4. Obstacle sensor coverage **(a)**; quad-tree cells identified as obstacles **(b)**.

As the furthest detectable obstacle has a distance of approximate $1.6Um$ (derived from Eq. 4) for the maximum x equal to $4.5Um$. This corresponds to a possible obstacle between Y_1 and Y_2 (Fig. 4) of $1.6Um$. As the detected obstacle can be in any part of that space, a set of cells between Y_1 and Y_2 are marked as an obstacle. So, in order to assure that all cells are marked (depicted in Fig. 4b) a set of points is generated. The points are placed along the arc defined by the distance to obstacle minus a user defined safety distance (at least $0.5Um$ for the mouse body, but usually $0.65Um$). As the number of points depend both on this distance and the deepest cell maximum width (w), the number of marked points is given by Eq. 5:

$$N(d,w) = \begin{cases} \pi/3 \Big/ \sin^{-1}\left(\frac{w/2}{d}\right), & \frac{w/2}{d} < 1 \\ 1, & \frac{w/2}{d} \geq 1 \end{cases} \quad (5)$$

The previous method can lead to cells wrongly marked as obstacles. In order to solve this problem a sweeping mechanism was developed to clean blocked (obstacle) cells making them passable again. This mechanism is very similar to the obstacle setting mechanism with the differences of marking passable cells (instead of obstacles), fixating d (user definable) and restricting the aperture of sweeping to half (30º). These restrictions were made so that the robot wouldn't be cyclically marking the same cells as blocked or passable.

An exemplar resultant internal map representation of the robot is shown in Fig. 5.

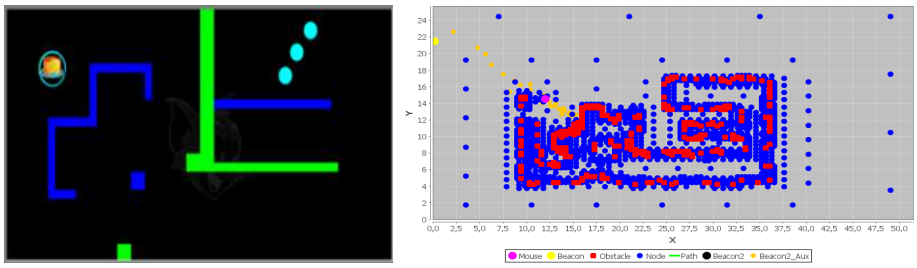

Fig. 5. Robot internal map after solving a maze.

3.4. Path Planning - Exploring A*

To find a path between a previously computed objective and the agent, the A-Star (A*) algorithm is used. This decision was made by balancing implementation cost with a guarantee of a solution. This algorithm is implemented over the quad-tree map representation, defining the shortest path towards the target by marking waypoints in the correspondent map cell centres.

Eq. 6 represents the cost between the source point and the target point, which passes through node n.

$$f(n) = g(n) + h(n) \tag{6}$$

Here *g(n)* is the real cost from the source to node *n*, and *h(n)* is the estimated cost between node *n* and the target. *f(n)* is the total cost of the path that passes through node *n*. The customary heuristic function would be the Euclidean distance between the source and the target position. However, when solely using the Euclidian distance as the cost for reaching a target, the robot would spend a great amount of time mapping around the same obstacle. In order to solve this problem, a different heuristic was developed where the waypoint cost is the Euclidian distance affected by the quad-tree cell depth and a weight (*f*), as defined in Eq. 7.

$$Cost\ (A,B) = Dist(A,B) * Depth(B)^{f}, f\ \geq 1 \tag{7}$$

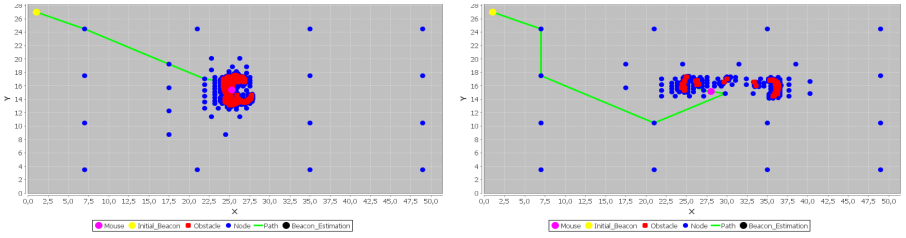

Fig. 6. Comparison of exploring algorithms: classic A*(**a**); Exploring A* (**b**).

This method assures that the mouse explores the map, towards the beacon, preferably following big cells. Consequently it increases the exploring step, as can be

seen in Fig. 6b, and the mouse's velocity. The guarantee of solution given by the classic A* is maintained as the exploring A* would revert to passing trough small cells after exploring the larger cells.

3.5 Navigation and Control

The robot is controlled by following each waypoint centre given by the A* algorithm towards the target. The robot rotates to each waypoint centre and accelerates in that direction. The waypoint centre is considered reached if the robot's coordinate values are within a certain error margin of that centre.

The agent navigation speeds are dynamically adjusted and are dependent on several factors. Simple control optimizations include a speed increase in rotation if there are big differences between the current angle and the waypoint direction or increasing speed if a waypoint is far away. More advanced implemented speed optimizations take into account subsequent waypoints and their relative direction in order to further increase the mouse' performance.

3.6 Visualization System

In order to visualize the current internal robot's map representation, in real-time, a visualization system was developed. This interface, shown in Fig. 7, was designed upon the Java open-source JFreeChart API [10]. Its zooming capabilities, as well as its Cartesian disposition, extend this system to debugging purposes. As observable, 5 different series, with different colours, were designed; each representing an individual estimation: pink for the mouse position, yellow for the initial (attack) beacon position, red for the obstacle's cell centres, blue for the passable cell centres, green for the planned path, and black for the current beacon estimation.

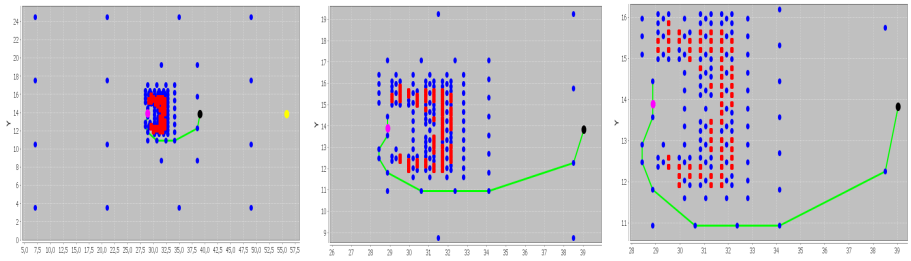

Fig. 7. Visualization system for different, incremental, zoom levels.

4. Experiments and Results

In order to test and validate the proposed approach, this section is divided in three: a presentation of a reactive agent architecture serving as a comparative base, a description of different evaluation scenarios and a presentation of the results.

4.1 Reactive Agent Architecture

Although this quasi-reactive (wall-following) architecture isn't the focus of this paper, it is well suited to serve as a comparative base for results evaluation. The designed reactive agent consists on a behaviour-based state-machine.

When the simulation starts, the robot begins with the state *Find Beacon*, rotating the robot around itself until it finds the beacon, or walking randomly until a wall is found. When the beacon is found, the robot changes its state to *Follow Beacon* and goes forward until it reaches the ground beacon area or finds a wall. If the beacon is reached, the state changes to *Beacon Area Reached* and the simulation ends. On the other hand, if a wall is found instead, the robot changes state to *Change Direction*, rotating itself to the side which has no detectable walls. Once the robot stops detecting a wall directly in front, it changes to the state *Follow Wall*. On this state the robot simply goes forward until it stops detecting the side wall. When it stops detecting the side wall, or detects another wall in front it changes back to the *Find Beacon* state.

Besides the state, two additional non-reactive elements were included: the time since the mouse was near a wall and a memory of the relative side of the wall being followed. The notion of time allowed the mouse to wander randomly to a wall when no beacon is found (high-walls) and to maintain a direction for a short time after leaving the wall. Remembering the wall being followed allowed the avoidance of a problem where the robot would cyclically alternate between close opposing walls.

4.2 Evaluation Scenarios

In order to evaluate each experiment the following, gradually increasing difficult scenarios, were chosen.

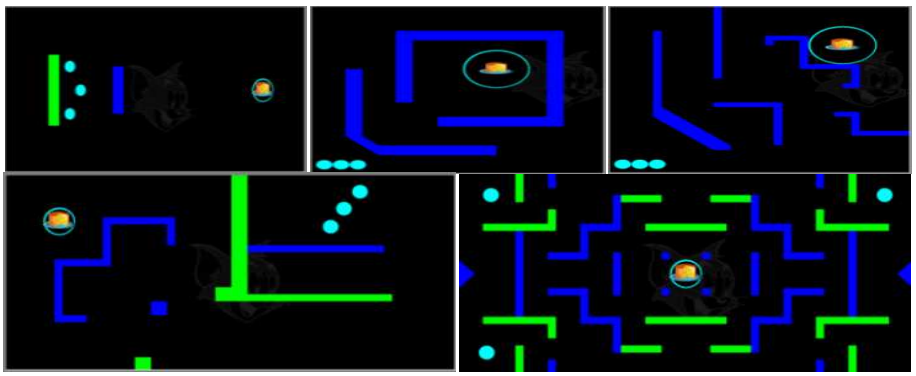

Fig. 8. Evaluation maps (from top-left to bottom-right): Basic **(a)**; MicRato98 **(b)**; 2001Final **(c)**; RTSS06Final **(d)**; 2005Final **(e)**.

As can be seen in Fig. 8: the *Basic* map has a small wall between the mouse and the beacon (a); *MicRato98* is an easy map with only low walls (b); *2001Final*, a medium difficulty map with only low walls (c); *RTSS06Final*, a hard map with low and high walls (d); *2005Final*, a very hard map with low and high walls (e).

The evaluation was done by observing if the mouse reached the cheese or not and the time it took to do it. For the exploring agent, since collisions impose errors in the self-localization procedure which would make the robot fail the waypoints (given by A*) towards the target, the number of collisions weren't considered. Additionally, when relevant, an observational description of the mouse' behaviour during the experiment may be included to further evaluate and compare the approaches.

4.3 Results

Each map was tested with the two described maze solving experiments. For results comparison, in both experiments the tests were made with two self-localization systems: through odometry measurement (see 3.1) and with GPS. Different deepest cell's maximum sizes (considered in the quad-tree decomposition) were used: a fixed resolution of 0.1*Um* that guarantees map solving (for the maximum 1.5*Um* obstacle distance) and one variable, granting the best performance for each map. Since the simulator adds some noise in the sensors and actuators, three different runs for each map and agent were performed. As such, conclusions can be made from averaging the results and thus overcoming the stochastic nature of the simulator.

4.3.1. Maze solving in an Unknown Map - Reactive Agent Evaluation

In this experiment we tested our (quasi-)reactive agent, *Smart-Follower*, for paradigm comparison. The results are shown in Table 1.

Table 1. Experimental results for the *Smart-Follower* agent.

Follower	Experiments									Observations
	1		2		3		Average			
	Time	Collisions	Time	Collisions	Time	Collisions	Successful Exp.	Time	Collisions	
Map1 - Basic	1260	0	228	0	1244	0	3	911	0	NA
Map2 - MicRato98	950	0	450	1	886	0	3	762	0	NA
Map3 - 2001Final	638	0	790	0	NA	NA	2	714	0	Enclosure Conflict
Map4 - RTSS06Final	NA	NA	1366	0	NA	NA	1	1366	0	Wall-Beacon Conflict
Map5 - 2005Final	NA	NA	NA	NA	NA	NA	0	NA	NA	Wall-Beacon Conflict

The *enclosure* conflict happens when the mouse is surrounded by obstacles very close to each other. Although the side sensors detect an obstacle there was enough room for the mouse to pass. The *wall-beacon* happens when the mouse is near a U-shaped wall. At each wall end the mouse turns towards the beacon coincident with the obstacle centre. As such, the mouse follows the same wall in the opposite direction.

4.3.2 Exploring Agent Evaluation

The following Table 2 then presents the exploring agent performance in resolving the proposed evaluating scenarios, for the self-localization and GPS methods. In order to visually depict the agent's deliberations, namely its mapping, planning and target estimation abilities, while solving some of the tested mazes (and as proof of concept), Fig. 9 presents some screen shots of a successful experiment.

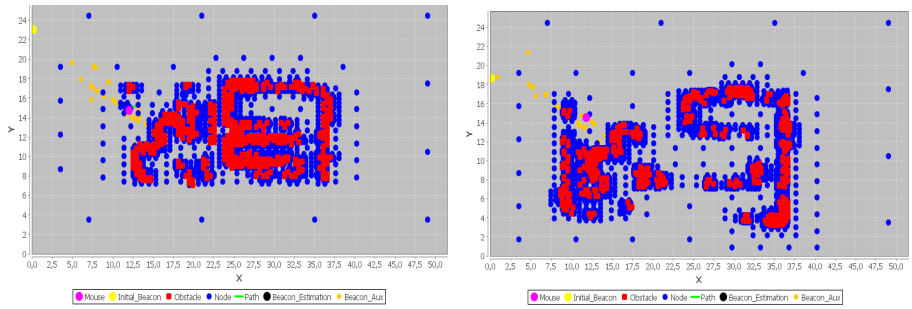

Fig. 9. Exploring agent runs on RTSS06Final map (from left to right): GPS **(a)**; Odometry **(b)**.

Table 2. Exploring Agent Map Results.

Unknown Map		Experiments					
		1	2	3	Average	Success.	Observations
		Time	Time	Time	Time	Exp.	
Map1 - Basic	GPS	260	658	562	493	3	NA
	Odometry	260	364	682	435	3	NA
Map2 - MicRato98	GPS	2544	NA	2634	2589	2	Corner Collision
	Odometry	NA	2328	NA	2328	1	Small Obstacles' Aperture
Map3 - 2001Final	GPS	3432	2598	NA	3015	2	Corner Collision
	Odometry	NA	NA	4266	4266	1	Corner Collision
Map4 - RTSS06Final	GPS	10234	7322	NA	8778	2	Corner Collision
	Odometry	5344	5546	6954	5948	3	NA
Map5 - 2005Final	GPS	NA	NA	NA	NA	0	Small Obstacles' Aperture
	Odometry	NA	NA	NA	NA	0	Small Obstacles' Aperture

In this experiment the corner collision is self-explanatory as representing the collisions at the obstacles' corners level. We found some problems at solving the *Final2005* and the *MicRato98* maps due to the distance between walls, which is kept at the lowest margin (1.5*Um*) in many situations, blocking the mouse, due to its mapping methodology limitations.

5. Conclusions and Future Work

As observable in the quasi-reactive experiment (4.3.1), approaches featuring some deliberations can quite effectively resolve most of the simpler maps (first 3 in Table 1) and situations with simple algorithms.

During the simulations, the localization method gradually converges to the real target position as the new measurements add points clustering around the beacon, resulting in a better centre of mass that gives a better beacon estimative.

The initial approach of using classic A* was too time consuming to be practical. As an example, comparing to the worst time from Table 2 in Map 4 (10234 cycles), the agent using classic A* was still very far from the target, due to navigation and exploration very close to obstacles. Nevertheless we believe that classic A* would eventually allow the agent to reach the beacon. As such, using the exploring A* with

an *f* factor of 1.2 (trial-error adjust) lead to a great time improvement. Yet, in worst case scenarios, with large clusters of small obstacles, the exploring A* can take longer but still guarantees a solution.

In comparison to the reactive agent, the exploring mouse evinced better map solving capabilities. In terms of consistency, results with the exploring agent were more favourable as it was capable of repeatedly solving the same maps (*Basic*, *2006Final* and *2001Final* – Table 2), contradicting the reactive mouse's behaviour (*Basic* and *MicRato98* in Table 1). This was possible due to the exploring agent's abilities to bypass U-shaped obstacles and recognizing its position, thus leaving already explored areas.

As a limitation, for the maps that the reactive agent could solve, the time taken for the exploring agent to conclude each map (except for the Basic map) was greater than with the reactive implementation. This was to be expected as map navigation (cell marking) isn't as effective as sensor navigation (wall following).

Within the reactive agent when comparing odometry to GPS navigation (Table 2) we conclude that the GPS's had a superior rate of success but odometry was more time effective. Besides the less need for adjustments to reach the cell centre, the reason for time effectiveness of odometry is the small error in obstacle detection that eliminates some hysteresis caused by setting and sweeping obstacles.

In the future the use of both lateral proximity sensors along with a probabilistic model for obstacles' detection should greatly improve the mapping efficiency. Parameters like the exploring weight, sensor apertures for marking and cleaning obstacles, distance for cleaning obstacles can be set to optimal by using reinforcement learning mechanisms. The maximum cell depth can also be dynamically adjusted using a greedy algorithm, like hill-climbing, in order to improve performance.

References

[1] Almeida, L., Fonseca, P., Azevedo, J.L.: The Micro-Rato Contest: a popular approach to improve self-study in electronics and computer science. SMC'2000, IEEE Int. Conference on Systems, Man and Cybernetics, Vol. 1, Nashville, USA (2000) 701 – 705.

[2] Lau, N., Pereira, A., Melo, A., Neves, A., Figueiredo, J.: Ciber-Rato: Um Ambiente de Simulação de Robots Móveis e Autónomos. Revista do DETUA 3 (2002) 647 - 650.

[3] Ribeiro, P.: YAM (Yet Another Mouse) - Um Robot Virtual com Planeamento de Caminho a Longo Prazo. Revista do DETUA 3 (2002) 672-674

[4] Luís, P., Martins, B., Almeida, P., Silva, V.: Detecção de Configurações de Obstáculos Perigosas: Aplicação no Robô EnCuRRalado. Revista do DETUA 3 (2002) 659-661.

[5] Reis, L.P.: Ciber-FEUP - Um Agente para Utilizar o Simulador Ciber-Rato no Ensino da Inteligência Artificial e Robótica Inteligente. Revista do DETUA 3 (2002) 655-658.

[6] Thrun, S. – Robotic Mapping: A Survey CMU-CS-02-111 (2002).

[7] Murphy, R.R.: An Introduction to AI Robotics. Bradford Book, MIT Press : Cambridge, Massachussets, London England (2000)

[8] LaValle, S.: Planning Algorithms. Cambridge University Press, 2006.

[9] Lau, N., CiberRato 2008 Rules and Technical Specifications, online at: http://microrato.ua.pt/main/Docs/RegrasMicroRato2008_EN.pdf accessed 15 April 2009.

[10] Gilbert, D.: The JFreeChart Class Library Reference Documentation. Simba Management Limited (2002) 158.