Local Array Mapping in an Autonomous Robot[*]

José Gonçalves, Jorge Ribeiro, Pedro Kulzer, Keith L. Doty, Francisco Vaz

***Resumo* - Neste projecto implementa-se um agente robótico autónomo que realiza uma série de comportamentos reactivos simples e que se adapta a condições ambientais locais. Visto que comportamentos oscilatórios em condições de cantos parecem ser característica inerente a comportamentos puramente reactivos, o agente robótico necessitará de memória para melhorar a eficiência global ao evitar obstáculos. Neste estudo, um robot lembra-se dos lugares de obstáculos previamente encontrados, através dum mapa local gerado na sua memória de curta duração. O mapa fornece informação suficiente para que o robot seja capaz de evitar ser encurralado em cantos e de oscilar indefinidamente.**

***Abstract* - This project implements an autonomous mobile robot platform that performs a series of simple reactive behaviors and adapts to local environmental conditions. Since oscillatory motion under cornered conditions appears to be an inherent characteristic of purely reactive obstacle avoidance mechanisms in autonomous robots, a robot agent will require memory to enhance overall obstacle avoidance efficiency. In this study, a robot remembers previously encountered nearby obstacle sites by means of a self-generated local map in its short term memory. The map provides sufficient information for the robot to avoid becoming trapped in corners and oscillating endlessly.**

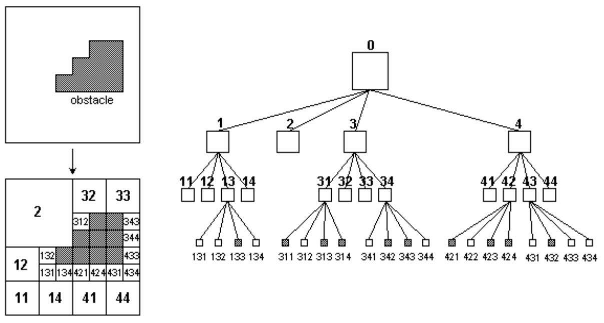

Fig. 1 - Zelinsky's *quad-tree* that holds information about obstacles. This is a global map construction where space is decomposed into cells of different sizes. (adapted from [10])

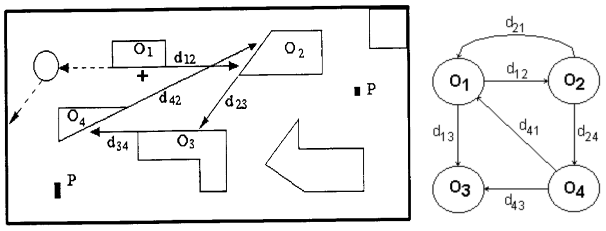

Fig. 2 - Doty's interconnected landmark map structure. Here, a global map holds distance information between landmarks. Space is thus efficiently coded into a relativistic data structure. (adapted from [3])

I. INTRODUCTION

Mapping in real robotic autonomous agents has been an important focus of experimental research [5][6] and other theoretical studies. These models are difficult to implement, test and improve. Investigations attempt to implement intuitive ideas about how simple mapping strategies should be worked out in a real robot that must build global maps [10] (see Fig. 1) [2][3][4] (see Fig. 2).

On the contrary, this work presents a simple and interesting way of implementing a local-only mapping strategy on a real robotic platform with low processing and sensing capabilities. Here, local mapping means that the robot is able to remember only detected objects that exist in its immediate neighborhood, namely obstacles it avoided within a short radial space-range. Although without any global mapping capabilities, the robot showed to behave rather well in local mapping problems.

[*] Trabalho realizado no âmbito da Disciplina de Projecto.

II. PLATFORM DESCRIPTION

The realized robotic platform evolved from previously developed platforms designed under minimalist constraints [1][7].

The robot has five IR (infrared) sensors, three in front for obstacle detection, one at the top of its mast for high obstacle detection, and the last one turned to the floor for hole detection (see Fig. 3). Hole detection is interpreted as a deadly obstacle detection.

The front three IR sensors have adaptive thresholds that allow the robot to adjust itself to local conditions. Specifically, the robot maintains object detection sensitivity to avoid obstacles, but, at the same time, attempts to pass narrow passages by lowering the sensitivity. Otherwise, the robot will suffer from "claustrophobia" in tight places.

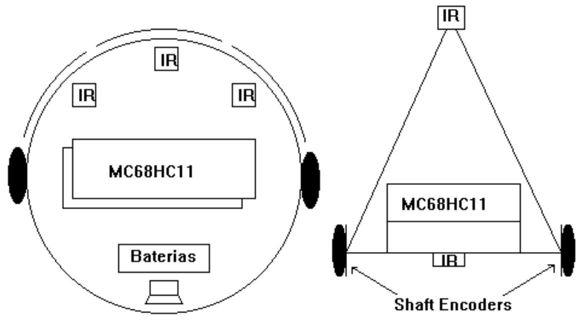

Fig. 3 - Robot top view. Robot side view.

If the robot becomes to aggressive and attempts to drive through a passage narrower than its body, it will eventually collide with an object. Upon collision, a corresponding bumper switch, one for each IR front sensor, forces "re-sensitization" of the corresponding IR proximity sensor. See Fig. 4 for an example of a tight passage.

To realize the local mapping, we use two shaft encoders, one per wheel, which measure distances traveled. From these distances, turn angles and displacement vectors may be computed.

Locomotion is realized by two pulse-width-modulation controlled motors and a caster wheel to give three-point stability.

Processing power is delivered by a single Motorola 68HC11 microcontroller with 32 KBytes of RAM (only about 8 KBytes actually used) and programmed in interpreted C language "IC" [9].

III. BEHAVIORS AND IMPROVEMENTS

A. Adaptive threshold

In essence, the robot's threshold adaptation to obstacle detection works in a manner similar to the retina's sensitivity change with varying light conditions. A simple differential equation (1) assumes unity time-steps (the value of which depends upon the execution speed of the

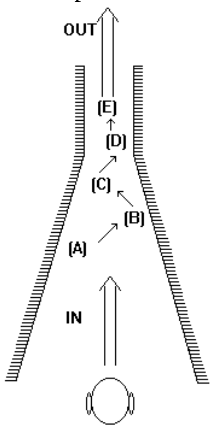

Fig. 4 - The robot negotiating a narrow passage.

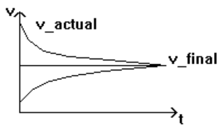

Fig. 5 - Speed progression starting at the current speed.

code) and equates the threshold to 2% of the actual sensor reading plus 98% of the current threshold value (see Eq.2). Fig. 4 shows an example of the robot adjusting its collision avoidance sensitivity while negotiating a narrow passage.

$$\frac{\Delta Threshold}{\Delta T} = 0.02 \cdot (Sensor - Threshold) \tag{1}$$

$$Threshold^{[n]} = 0.02 \cdot Sensor + 0.98 \cdot Threshold^{[n-1]} \tag{2}$$

B. Motors drive

The robot controller transfers power to the motors in a gradual manner to produce smooth motion transitions (Fig. 5). A differential equation approximation, similar to the IR sensor adaptive threshold function, produces exponential speed progressions. Eq. 3 implements the approximation, with the gain value of 0.05 determined by experiments. Again, this gain factor depends upon the execution speed of the code.

$$\frac{\Delta Speed}{\Delta T} = 0.05 \cdot \left(speed^{[desired]} - speed^{[current]}\right) \tag{3}$$

Note that the increment *ΔSpeed* may be positive or negative, increasing or decreasing the actual speed, respectively.

C. Shaft-encoders

One shaft encoder on each wheel allows the robot to compute distances, turn angles and corresponding displacements. From these measurements the robot can construct a local map of obstacles surrounding it.

An IR emitter-receiver pair, separated by a small gap and mounted on each wheel, detects transparent and black sectors on a plastic disk rotating in the gap. The shaft encoder disc (Fig. 6) interrupts the beam as it rotates, resulting in a rectangular wave shape that calls interrupt service routines for periodic distance computation. Fig. 6 shows the transparent disk with the sectors, used for the

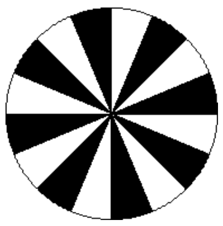

Fig. 6 - Shaft encoder disk.

purpose. Eight black and eight transparent sectors were sufficient to generate a coarse local mapping, as will be explained later.

Impulses created by the interrupted IR beam generate interrupts which call code that increases (forward motion of the wheel) or decreases (backward motion of the wheel) two internal counters. These counters hold the accumulated algebraic sum of encoder sectors that have passed by the beam. An internal motor direction control flag in the software distinguishes the direction of motion and determines whether a shaft encoder counter should be incremented or decremented.

IV. MAPPING ALGORITHM

The mapping method, based on limited dead-reckoning capabilities, generates a local $(2n+1) \times (2n+1)$ array which images obstacles in the immediate vicinity of the robot's current position. Each array cell corresponds to a square with *d* millimeters on a side and indicates the presence or not of an obstacles in that square. Visualizing the entire robot environment as tessellated with $d \times d$ squares, the local array can be visualized as a window of the entire space, centered about the robot's current position. As the robot moves, the center of the window moves along with the robot and the window acquires new squares and looses other squares which are no longer in view.

After a brief exploration, the robot generates the first complete window or local map of obstacles in its immediate environment (Fig. 7). Subsequently, whenever the robot rotates or shifts forward, the local map (window) updates to maintain the correct position of the obstacles relative to the robot at the center of the new array. When an obstacle "falls" off the array, i.e., gets clipped by the window, then it is completely "forgotten". The initial, and succeeding window images of the local environment viewed by the robot derive from the rotational and translational movement information computed from the wheel shaft-encoders.

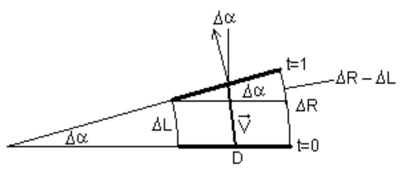

Fig. 9 - Incremental displacement computed from wheel distance counts.

With this internal mapping behavior, the robot is able to accomplish the following goals:

- Improve overall efficiency when negotiating local obstacles and corners, by predicting their location inside the map window,
- Avoid endless and cyclic avoidance behaviors near dead-ends, concave corners and other symmetric obstacle arrangements, and
- Find an exit from an obstacle cluttered or almost enclosed local environment.

A. Details of the Mapping Algorithm

The mapping algorithm is based on on-line displacement vector constructions as shown in Fig. 8. Robot displacement increments are summed together to form the *local displacement vector*. The resulting vector displacement dictates the new window location.

To extract the vector value of each incremental displacement (see Fig. 9), the autonomous robot must compute its magnitude and phase as follows. To obtain the magnitude, the robot takes the average of the distances traveled by each wheel. To obtain the phase, or net rotation $\Delta\alpha$, the robot takes the difference between those distances and divides the result by D, the robot's inter-wheel distance. Note that these displacements and angles are only proportional ones, and do not directly represent traveled meters or turned degrees (see Eq. 4 and Eq. 5). Constant *D*, the robot's inter-wheel distance is expressed in shaft-encoder counts, to keep compatible distance units. For small phase differences, we will have

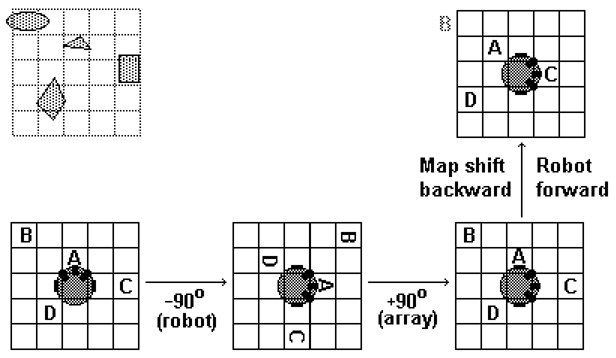

Fig. 7 - Illustration of the basic mapping process with a 5x5 map array.

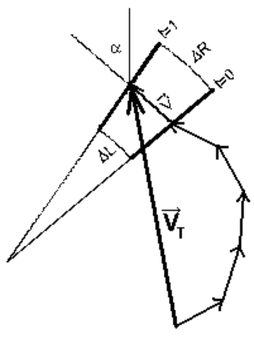

Fig. 8 - To compute path integration vector V_T, it is necessary to integrate all infinitesimal displacement vectors V.

$$\Delta\alpha \approx \tan\Delta\alpha \approx \frac{\Delta R - \Delta L}{D} = \frac{Count1 - Count2}{D} \quad (4)$$

$$\begin{cases} \|\vec{V}\| \approx \dfrac{\Delta R + \Delta L}{2} \\ \hat{V} = \Delta\alpha = \dfrac{\Delta R - \Delta L}{D} \end{cases} \quad (5)$$

The total turning angle α can be exactly computed anytime as shown in Eq. 6 (no approximation).

$$\alpha_{compass} = \int_t d\alpha = \int_t \frac{dR - dL}{D} = \frac{1}{D}\int_t (dR - dL) = \frac{R - L}{D} \quad (6)$$

The total vector V_T that points from a start position to the current one, is given by Eq. 7.

$$V_T = \int_t dV = \left(\int_t dx; \int_t dy\right) = \left(\int_t \|dV\|\cos\alpha; \int_t \|dV\|\sin\alpha\right) \quad (7)$$

The robot will compute as many samples of V as possible and sum all of them to compute the total displacement in real-time using the approximation shown in 8.

$$V_T \approx \left(\sum_t \|\Delta V\|\cos\alpha; \sum_t \|\Delta V\|\sin\alpha\right) \quad (8)$$

To simplify the algorithm and increase execution speed, the turn angle is quantized into 45^o intervals with round-off accumulation. For example, when the robot turns 50^o, it rotates the map window relative to the center by the minimum quantization step of -45^o and subtracts those 45^o from the actual turn angle. The remaining 5^o accumulate towards the next quantization trigger of $\pm45^o$.

In reality, instead of maintaining the total displacement vector V_T, two more directly useful displacement values are maintained: total vertical (forward) and horizontal (lateral) displacements *X* and *Y* relative to the robot, whose incremental contributions ΔX and ΔY are computed by Eq. 9.

$$\begin{cases} \Delta Y = V \cdot \sin\alpha \\ \Delta X = V \cdot \cos\alpha \end{cases} \quad (9)$$

These incremental forward and lateral steps are summed as in Eq. 8, yielding the total displacement.

When the robot redraws its local map it must compensate for the current turn angle and translation by means of the forward and lateral displacement components. Since rotations change by increments of $\pm45^o$, the new rotated *X* and *Y* values of the map squares can easily be computed from the current values, as demonstrated in the next few equations.

We have now Eq. 10 and Eq. 11 which show the basic starting points. We want to compute V^{new} from V^{old}, after a rotation by $\pm45^o$. Eq. 12 shows the steps needed to derive Eq. 13 , a truly simple formula for updating X' and *Y*. Since we know that

$$V^{old} = \sqrt{x^2 + y^2}\angle arctg(\frac{y}{x}) \quad (10)$$

and that

$$V^{new} = \sqrt{x^2 + y^2}\angle arctg(\frac{y}{x} \pm 45^o) \quad (11)$$

we can now compute

$$\begin{aligned} x' &= \sqrt{x^2 + y^2}\cos(arctg(\frac{y}{x}) \pm 45) \\ &= \sqrt{x^2 + y^2}\left\{\cos\left[arctg(\frac{y}{x})\right]\cos(45) \mp \sin\left[arctg(\frac{y}{x})\right]\sin(45)\right\} \\ &= \sqrt{x^2 + y^2} \cdot \frac{\sqrt{2}}{2}\left(\frac{x}{\sqrt{x^2 + y^2}} \mp \frac{y}{\sqrt{x^2 + y^2}}\right) \end{aligned} \quad (12)$$

which leads to the final results,

$$\begin{aligned} x' &= \frac{\sqrt{2}}{2}(x \mp y) = 1.4 \cdot (x \mp y) \\ y' &= \frac{\sqrt{2}}{2}(x \pm y) = 1.4 \cdot (x \pm y) \end{aligned} \quad (13)$$

V. EXPERIMENTAL RESULTS

In preliminary experiments, the robot approached a wall on its left and on its right. Inspection of the local array map reveals the appropriate obstacle marks. Fig. 10 and Fig. 11 illustrate how the robot marks the wall obstacles in the local map array by an "X" after detection with the IR sensors. Note that the black square in the middle always denotes the robot's position in its local map. Also, observe that the robot's map of its local environment "rotates" and "translates" in the opposite direction of the corresponding robot rotation and translation in order to preserve the spatial relationships between the robot and the obstacle.

Fig. 12 shows the robot arriving at position "A", where it marks an obstacle on the front left side. When it turns away from it (to the right), it again sees an obstacle that is marked in front of the robot. Note that the previous obstacle marker has rotated to the left as a compensation for the robot's rotation to the right. Here, the robot

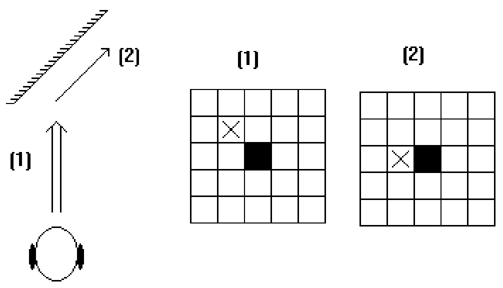

Fig. 10 - Behavior and map construction with an obstacle on the left.

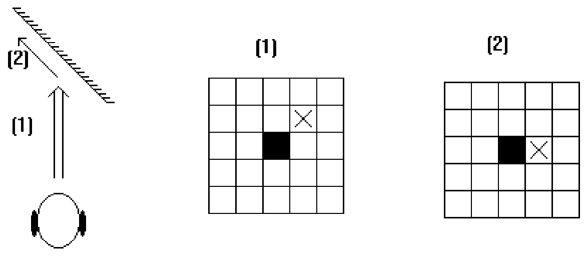

Fig. 11 - Behavior and map construction with an obstacle on the right.

decides to continue to turn right since it knows there is an obstacle on its left by reading its own map, even though its sensor arrangement does not permit the robot to detect the left obstacle while traveling parallel to the wall.

Thanks to this mapping behavior, the robot performs much better and more deterministically in complex cul-de-sacs or box canyons, such as the one shown in Fig. 13. The robot efficiently exits the cul-de-sac without the annoying random or oscillatory behavior typical of a non-directed search for an exit [8]. Turning the mapping function on and off dramatically demonstrated the enhanced performance gained through autonomously generated local mapping. With the mapping function active, the robot finds the exit with little to no hesitation, as if pre-programmed to go directly to the exit. When mapping is inactive, the robot assumes ordinary, reactive, obstacle avoidance behavior and exhibits the problems inherent to such behavior. Namely, the robot "bounces" randomly from wall to wall until it accidentally finds the exit or it becomes dynamically trapped in a corner, oscillating from side to side.

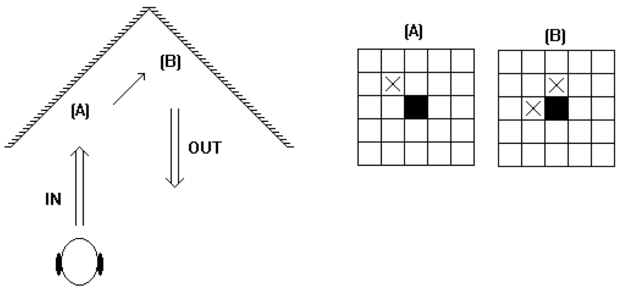

Fig. 12 - Evasive behavior in a corner enabled by local map utilization.

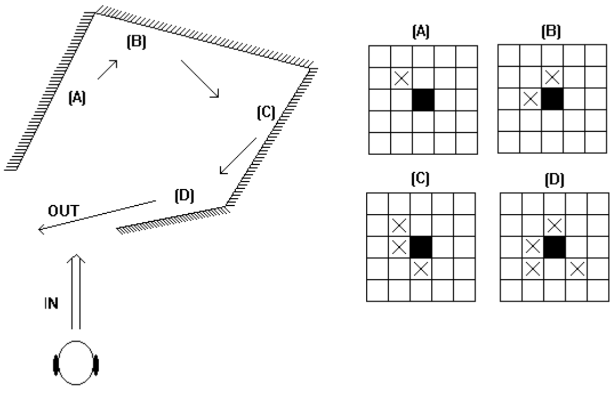

Fig. 13 - Behavior in a more complex dead-end corner.

VI. CONCLUSIONS AND FUTURE WORK

In this work, we designed, developed and implemented a real-time, autonomously generated local mapping scheme designed to enhance reactive obstacle avoidance behavior of a real robot. Experiments demonstrated the efficiency of our approach. The robot performs obstacle avoidance more efficiently when utilizing, self-generated local mappings.

Future work would include the use such mapping structures as part of a much larger robotic architecture [8]. Also, instead of using a grid-like array, a polar map with multi-distance sectioned map structure might be more "natural" or biological. For example, in Fig. 14 space is partitioned into sectors with the size of sector segments increasing with radial distance. The three gray segments depicted in the figure indicate where the robot perceives obstacles.

Fig. 15, Fig. 16, and Fig. 17 show some views of the real robot implemented in this work. Note the mast in the middle, where the "high obstacles" sensor resides.

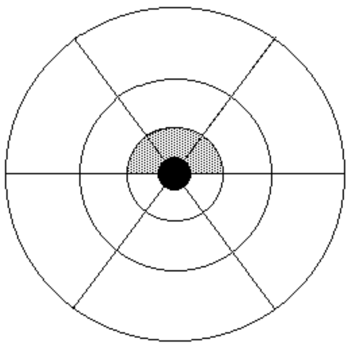

Fig. 14 - Polar coordinate descriptions might be a more natural way of generating a self-relative mapping structure.

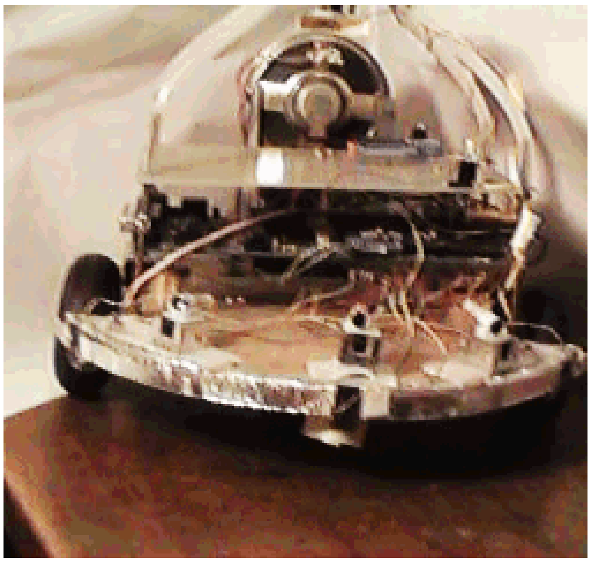
Fig. 15 - Robot front view.

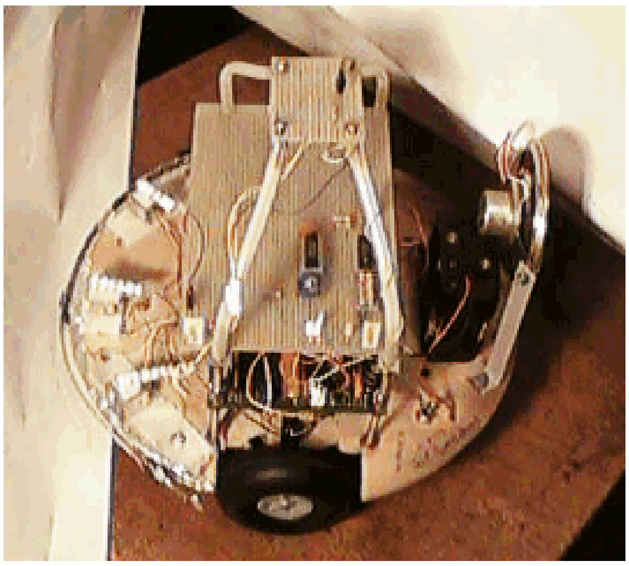
Fig. 16 - Robot top view.

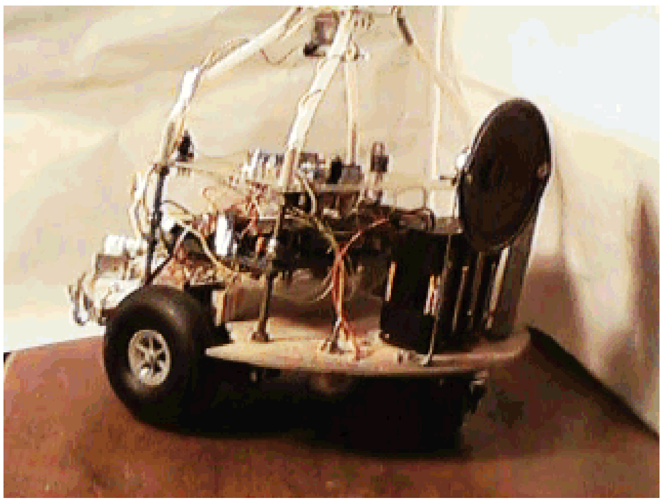
Fig. 17 - Robot lateral view.

REFERENCES

[1] A. Branco, P. Kulzer, “WHISPER - A mobile robot platform for surveillance purposes”, Internal publication of the Departamento de Electrónica e Telecomunicações da Universidade de Aveiro, 1995.

[2] S. Caselli, K. Doty, R. Harrison, F. Zanichelli, “Mobile robot navigation in enclosed large-scale space”, Machine Intelligence Laboratory, University of Florida, 1994

[3] K. Doty, S. Caselli, R. Harrison, F. Zanichelli, F., “Landmark map construction and navigation in enclosed environments”, Machine Intelligence Laboratory, University of Florida.

[4] K. Doty, S. Seed, “Autonomous agent map construction in unknown enclosed environments”, *MLC-COLT ‘94 Robot Learning Workshop*, 1994.

[5] M. Mataric, “Integration of representation into goal directed behavior”, *IEEE Transactions on Robotics and Automation*, 8, 304-312, 1992.

[6] P. Kulzer, “NAVBOT” - Autonomous robotic agent with neural network learning of autonomous mapping and navigation strategies, Master’s Thesis, University of Aveiro, Portugal.

[7] P. Kulzer, Drafts from lectures about *Mobile Robotics* given by Prof. Keith Doty, at Universidade de Aveiro, Portugal, 1995.

[8] P. Kulzer, K. Doty, “Refined Neural Architecture for Opinion-Guided Reaction learning in Autonomous Agents”, in progress at the University of Aveiro, Portugal.

[9] R. Sargent, A. Wright, *IC - Interactive C* for the *68HC11*, 1994.

[10] A. Zelinsky, “A mobile robot exploration algorithm”, *IEEE Transactions on Robotics and Automation*, 8, 707-717, 1992.