International Journal of Sciences and Techniques of Automatic control & computer engineering
IJ-STA, Volume 1, N° 2, December 2007, pp. 155–166.

Modelling, Real time simulation and Fuzzy control of a differential wheels mobile robot

Wassim Filali, Imen Ayari, Abderrazak Chatti

Institut National des Sciences Appliquées et de Technologie (INSAT),
Centre Urbain Nord, BP. 676, 1080 Tunis Cedex, Tunisie

wassim.filali@laposte.net, ayari.imen@gmail.com, abderrazak_chatti@yahoo.fr,

Abstract. *In this paper, we present all the steps to fulfil a real time simulation experiment. It starts by taking a real differential wheels robot as reference, modelling its dynamics, electrical motors and its infrared sensors. Then a fuzzy controller is elaborated to perform an exploration task.*

Keywords. *Modelling, Mobile Robot, Real time simulation, Fuzzy control.*

1. Introduction

The first step of developing new controllers must go through simulation to avoid first unpredictable results and to be able to adjust parameters to get the desired behavior. Even if simulation is just an approximation, it's important to make it as close to the real process as possible, including not only dynamical models but also models of actuators and sensors. We would like to check how much real time simulation helps in developing a Fuzzy controller.

2. Real robot presentation

A real robot has been conceived around a hardware architecture that facilitates the communication with a remote computer in different ways. It was based on different modules as shown in Figure 1. Every module has processing capabilities that allows updating the type of the low level current controller for the motor driver (2), test different dead reckoning approximations for the wheel encoders that determines the position (3) or compute different averaging methods for the infrared sensors (4). The robot can be totally free with an embedded master controller while transmitting wireless information to a computer, or the computer can be its master taking inputs and giving orders.

This paper was recommended for publication in revised form by the editor Staff.
Edition: CPU of Tunis, Tunisia, ISSN: 1737-7749

A mechanical structure handling this architecture was based on two DC motors controlling through gears two differential wheels. Wheels are provided with encoders that allows after processing determining an approximate relative position of the robot. Eight Infrared sensors are placed on a circle every 45°, they sense for obstacles by measuring the reflected light.

3. Robot environment modelling

As summarized by the figure 2, this step concern modelling the whole robot with its environment to provide equations applicable on a simulation virtual level. The simulation will take into consideration the robot dynamics, the electrical motors, the infrared sensors with the obstacles they sense. All this will allow simulating the controller that will take the inputs of the simulated world and generate the orders.

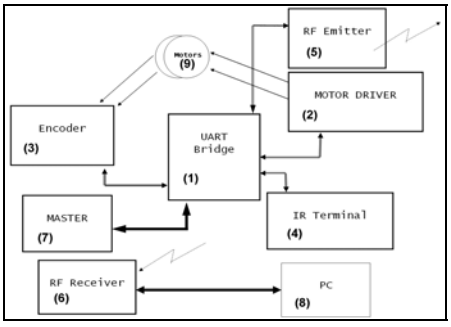

Fig. 1. Robot Architecture

Fig. 2. Modelling the robot

3.1 Robot dynamical model

The figure 3 shows the robot parameters where C is the centre of the wheels and G is the gravity centre, d separate these two centers as the in the general case d is always different than zero.

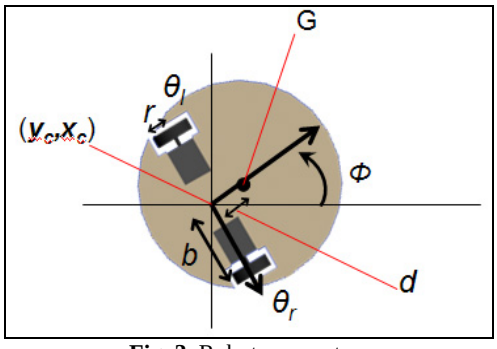

Fig. 3. Robot parameters

The movement vector q containing variables of the robot has the Cartesian coordinates of the center, the orientation and angle of each of left and right wheels:

$$q = (x_c, y_c, \phi, \theta_r, \theta_l)$$

Given the kinematics equations of this model:

$$\dot{y}_c \cos\phi - \dot{x}_c \sin\phi - d\dot{\phi} = 0$$

$$\dot{x}_c \cos\phi + \dot{y}_c \sin\phi - b\dot{\phi} = r\dot{\theta}_l$$

$$\dot{x}_c \cos\phi + \dot{y}_c \sin\phi + b\dot{\phi} = r\dot{\theta}_r$$

We can obtain the Pfaffian kinematics constraint:

$$A(q)\dot{q} = 0$$

With

$$A(q) = \begin{bmatrix} -\sin\phi & \cos\phi & -d & 0 & 0 \\ -\cos\phi & -\sin\phi & -b & r & 0 \\ -\cos\phi & -\sin\phi & b & 0 & r \end{bmatrix}$$

Deriving the Kinetic energy equation and considering the system with the Pfaffian constraint, we obtain the matrix differential system:

$$M(q)\ddot{q} + V(q,\dot{q}) = E(q)\tau - A^T(q)\lambda$$

$$A(q)S(q) = 0$$

$$S^T(MS\dot{\upsilon}(t) + M\dot{S}\upsilon(t) + V) = \tau$$

The resulting system is based on non-stationary matrices that will be re-computed every simulation step:

$$\dot{x} = \begin{bmatrix} S\upsilon \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ I \end{bmatrix} u$$

The conclusion of this part is that we have an approximate model of the robot dynamics. Matrices of this differential equation have parameters measured on the real robot and some others are variables computed on the run with the simulation like position and speed.

The input of this equation is the torque applied on every wheel. Resolving theses equations provide the speed and position of the robot.

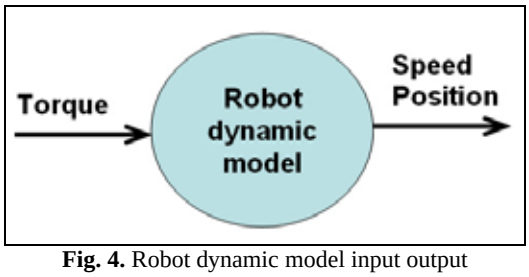

Fig. 4. Robot dynamic model input output

3.2. Robot electrical control

The real robot has a current regulator efficient enough to be considered as part of the motors model as it answers much quicker than the robot dynamics.

So the algorithm should just provide the current reference that should be applied. A motor electric model gives the torque that would be generated under such conditions of speed rotation and motor properties.

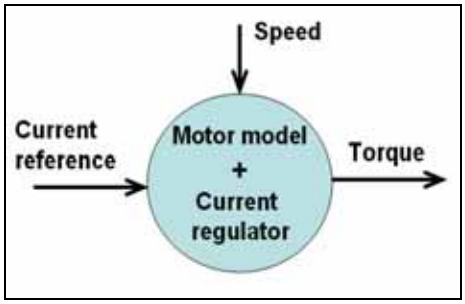

Fig. 5. Electric motors model input output

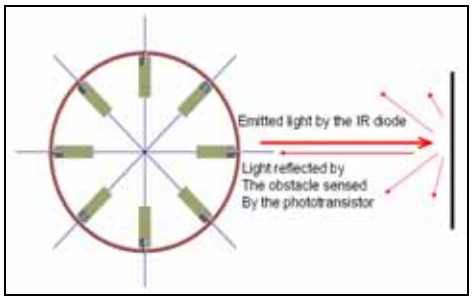

Fig. 6. IR sensors (IR diode+Phototransistor)

3.3. Robot sensors model

The robot sensors are eight couples of infrared light emitting diodes, plus a phototransistor

Far obstacles need high light emitting power to be detected, from another side; phototransistors can be saturated with much reflected light. To allow phototransistors to be in their well working margin, the IR diodes light emitting power L can be adjusted so it can be raised to a value allowing the detection of far obstacles, or lowered for detecting the reflection level of close obstacles.

The light reflection and diffusion depending on the angle of observation is a property of the material of the obstacle and its color. For testing purposes, we started by limiting all obstacles materials with white mat papers. This same paper kind has been used for taking a big amount of measures that allow reconstituting the sensor model; hence our sensor model will be based on this kind of material.

Ambient light is a big problem that wouldn't allow clean measures of the obstacles reflected light. This problem has been resolved by a program running on the microcontroller managing the IR sensors. The program loops on switching off completely the IR diodes light, taking measures of the ambient light level in every sensor, then switching on the IR lights and subtracting the ambient light from the measure made, all this within 40 milliseconds, so it seems like if the sensors are instantly giving both ambient light and reflected light values.

Reflected Light = Incoming Light – Ambient Light

Angle

The first parameter taken into consideration for the sensor model recognition is the angle with the obstacle. We remind that the light emitted is not always perpendicular to the obstacle, but the light source and the sensor are stick together and they see the same variable angle with the obstacle. Fixing all other parameters, many measures were made. A tricky way to have precise values for every exact angle without even the need for an angle sensor (such as potentiometer or rotational encoder), is to make the obstacle rotate at a constant speed, all measures are taken with a constant sample period, we also know that the curve would see a maximum when it's perpendicular to the sensor, so we just have to detect two maximums and spread values among the radians values. Figure 7 shows an already reconstituted mathematical function of the sensor angle response.

A good model for us is the equation that gives the best ratio between precise modelling and simple equation that would fit for real time computation. A first attempt was to have one polynomial equation for the whole [–Pi/2,Pi/2] interval. Ninth, tenth, eleventh degrees were reached without a visible good result. Then half [0,Pi/2] was considered using the symmetry extension. Result was better but still the closer we were to Pi/2, the bigger was the error. As shown in Figure 7, the shape target to be approximated seemed to have a strange little dome that was on top of one big dome. This gave the idea to divide this in two equations, the first fits only the big dome with the less possible error, the second fits just the small one, then an interpolation was used to stick both equations together whether we were more in the centre (small dome) or far from it (big dome). This gave satisfying results just with two polynomials of the fifth degree.

The small dome is like the beam of the reflected light that is stronger than the diffused light. A problematic point due to experimentations limitations are facts that we didn't got zero value for a light that is parallel to the obstacle, this can be explained by fact that tests were made not very far from the obstacle so not only parallel rays reach it, but also rays making a certain angle with the IR light source. This particular case will not bother with our mobile robot simulation as given the shape of the robot, obstacles can be parallel to the robot sensors only if they're already far enough to be out of reach of sensors, and if we impose that obstacles should be made of convex shapes of long enough segments, perpendicular obstacles would be detected before the robot is close to a parallel obstacle.

Distance

The Infrared light source (as any light source) gives a fixed quantity of light energy on a fixed solid angle. A solid angle s in steradians equals the area a, that can intersect with than angle divided by the square of the distance d of that area:

$$s = a/d2 \quad \textbf{(1)}$$

If we consider obstacles big enough to enclose most of light energy, we can consider that the same fixed area a, is having the light reflection. As light energy is

proportional to s, we conclude that light sensed depends on the reversed square of the distance of the obstacle.

The practical experiment easily fitted that equation, and we had just to identify the scaling parameter C based on (C/X2) function.

On figure 8 we see the shape of the 1/X2 function. We have there three different curves made out of different emitted light powers. If the emitted light power is big, the sensor saturates on close distances so the experimental blue (dark) curve stops and can be completed by the green theoretical one (bright).

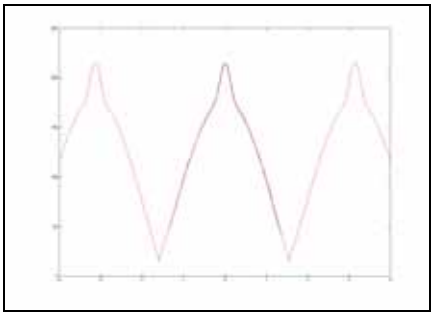

Fig. 7. Angle response of the sensor

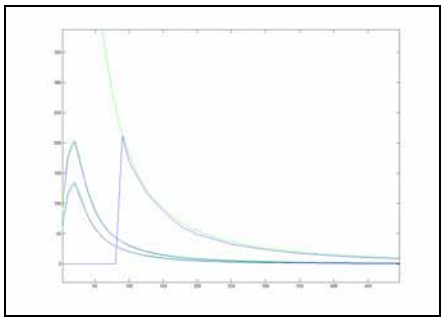

Fig. 8. Light sensed – distance

The (C/X2) function cannot be convenient approximation for all covered distances. As when the obstacle gets very close to the sensor, there start the effect of the non neglected distance between the light source and the light sensor. On far distance we could assume that the cone of the emitted light was the same as the cone on which the sensor could detect. On close distance, these two cones start to loose intersection till the source and the sensor get to see completely independent parts of the obstacle, then, no light is sensed. Figure 9 shows how the closer the distance, the smaller is the common area for source and sensor.

This cone intersection explains why the reflected light have a maximum value then it get to decrease, so it respects no more the (C/X2) equation. Visually, the shape on close distance looked like a parabolic shape and it could easily be approximated with a parabolic equation of DX2. On far distance, the first function is used, on close distance the second, in between, an average of functions is computed, so the whole together well fits all conditions of distance.

Light intensity

As enounced on the beginning of this part, the IR diodes light emitting power L can be adjusted to have enough light power for reaching far distances and low power that do not saturate the sensor on close distances. This light power is adjusted by pulse modulation that results in giving any fraction of the complete light power. The pulse width in percentage is analogue to the same percentage of lighting power.

The figure 10 shows experimental results and the approximated modelling function. We noted that for fixed parameters as distance and angle, the lighting power

is proportional to the sensed light, except that higher than a certain value, the sensor gets totally saturated.

Figure 11 and 12 shows the sensor response (Height dimension) depending on the distance and the angle. We see in it a combination all the functions we previously met in this part.

It clarifies how we have to make the compromises for varying the light intensity as the deep blue (dark down) part is unusable due to sensor's noise on low level of detection, and the dark red (dark top) is also an unusable part due to sensors saturation.

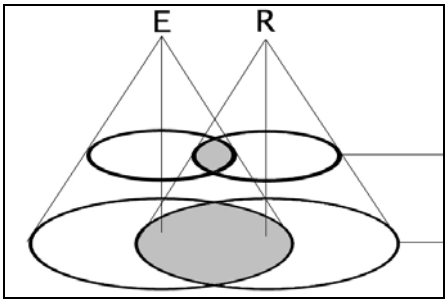

Fig. 9. Intersection zone of different cones

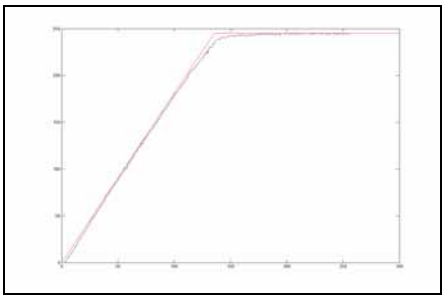

Fig. 10. Lighting power

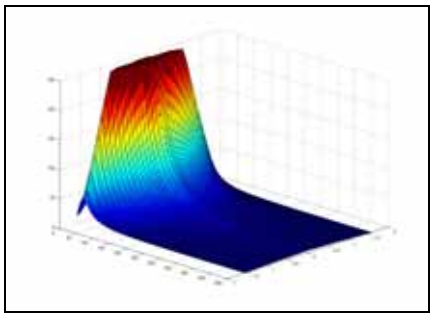

Fig. 11. 3D sensor model L = 27%

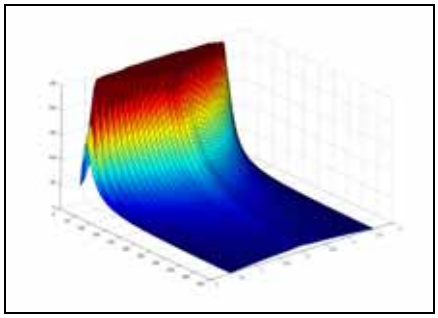

Fig. 12. 3D sensor model L = 78 %

On the environment of the robot, it's easy to enter coordinates of obstacles. Having the coordinates of the robot we can geometrically compute where will be (if existent) the light reflection point on the nearer obstacle. Once this reflection point determined, the angle and distance are computed in geometrical way too. These are the parameters needed for the sensor model. The light intensity is a parameter that should be fixed by the robot master controller algorithm.

Figure 13 shows how to isolate the sensor model input and output. As the sensor model is modeled by a static not dynamic model, we can use it just as a function call every time we need it, hence every simulation step when the robot has moved to a new discrete location.

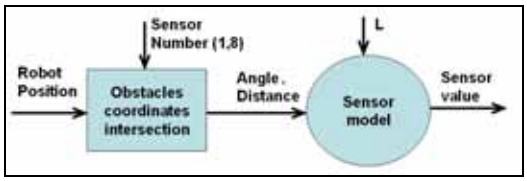

Fig. 13. Sensors model input output

4. Real time simulation

There are two ways to fulfill a real time simulation, whether to master a real time operating system using some hardware counters that gives exact response timing for executing the callbacks handling the simulation process, this way we can guarantee that the simulation steps are going hand in hand with the real scrolling time.

The second way that is most spread as it's easier to use and to implement. It's the independent process that just loops, depends on the operating system, or even which simulation step period can vary with the complexity of the cases to compute. On such systems, everything is not lost, as if execution cannot be controlled, we always can at least have an idea about the time spent since the last step. It's a variable stepping period, but we can correct this by tuning our simulation algorithms to make them not to drift and fit again with the real time scroll.

Let's note **Dt** the preceding period, we can resolve equations assuming that **Dt** is small enough, but taking it varying as it is: $X = X + X' \mathbf{Dt}$.

Another better variant that allows to enhance this process and to use any desired precision is to take $\mathbf{Dt} = n\,\mathbf{dt}$ and repeat n times: $X = X + X' \mathbf{dt}$.

Figure 14 shows the interface of the application of the real time simulation including the models of the robot and its sensors. The application has easy to use facilities to draws walls for making a complete environment.

The application can show three robots. The **real robot**'s coordinates that are updated from the robot that is connected through the serial port, the **simulated robot** position, that is the robot executing the preprogrammed algorithm, and a third robot that is a **user robot**, this robot do not have controlling algorithm but takes orders from a user through a joystick connection, the user can easily record data of the robot trajectory, sensors and control orders, these can be useful to treat and extract parameters for an algorithm using a reversed learning method for example.

One of advantages of simulation against real world experiments that offers this application, other than fact of not breaking the robot, it's the adjustable speed of execution, no need to wait that the robot goes with slow speed in a big surface when we're waiting to know when will he cover 90% of the area, the speed of results become depending only on the processing power used.

This environment can well be saved and reloaded to remake the same experiments with the same conditions. So we can profit of another of advantages of simulations, it's the near exact repeatability of experiments we can lead. It's useful to be able to repeat exact conditions so we make fewer things variable in the whole experiment and focus on just the aspect that we study. So we can compare different variants in exactly the same conditions. On real time, our repeatability is nearly exact but not exact,

because the variable simulation period introduces results that are not exactly similar but even an error with the lowest digit of float numbers can be cumulated over simulation steps. And as the robot has to make decisions on a very critical situation, even a very small difference can make one way with higher priority than the other, which makes the robot take a complete different way. As we see on Figures 15, the robot had the chance to find the exit much quicker in case B than in case A, though all initial conditions are exactly the same.

Fig. 14. Real time simulation interface

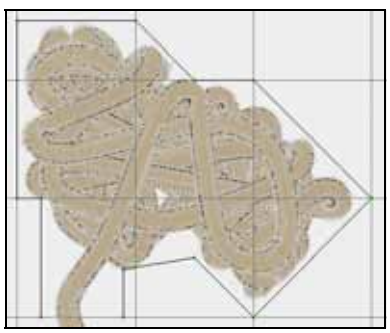

Fig. 15. A - way one

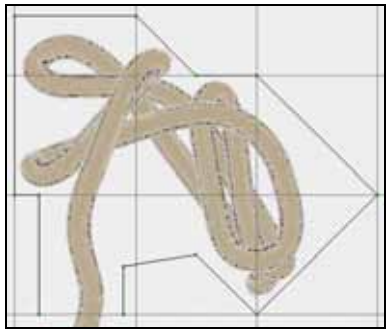

Fig. 15. B - way two

It's easy to check if this difference is really due to the variable sampling period. We run the simulation with having a fixed sampling period, thus our simulation became no more real-time, as we could see the robot slow down or accelerate a little bit when the process had more or less computer processing resources. As good result we've seen the repeatability go up to the maximum till a point that both final pictures generated by the application in two different tests, were exactly the same.

Such repeatability can never happen in real life due to the uncertainties of movements in the micro world. So it's clear that with these limited conditions, we would prefer the real time behavior of the simulation because it's the closest way to check whether our simulated model well fits its target that is the real robot. This application provides all the tools to be able to apply the same user control for both the simulated and real robot and check if they have the same behavior, and also to run the same controlling algorithm on the computer and in the microcontroller of the real robot and see if with the same environments, robots have the same behavior.

5. Fuzzy control

Our objective from this first control algorithm for the robot is to make him perform an exploration task, as everyone that wakes up in an unknown environment, he would first try look what's around. This is easy to do for a human to have a quick look around with two eyes allowing depth sensing of every obstacle. But for the limited field of view sensors of our robot, he needs to be moving to get close to the obstacles to sense them. This exploration task can be achieved just by avoiding the obstacles and go around looking for a free zone, till the next obstacle is met. This task can be divided into small rules that basis are:

- Go ahead
- **If** an obstacle is on the front **then** go back
- **If** an obstacle is on the left **then** go more on right
- **If** an obstacle is on the right **then** go more on left
- **If** an obstacle is on the back **then** go fast ahead

Figures 16 and 17 shows inputs and outputs functions of our fuzzy system. For using a simple system word like {front, left, back, right} with our eight sensors, we can have another fuzzyfication inference process to know for example if an obstacle is met on sensor S2, how much it's on front and how much it's on Left.

Input function is a modular or circular fuzzy function, as it always loops and continue in its area, that brings one limitation for example we cannot extend fuzzy functions outside the function zone to make one side with heavier weight than the other.

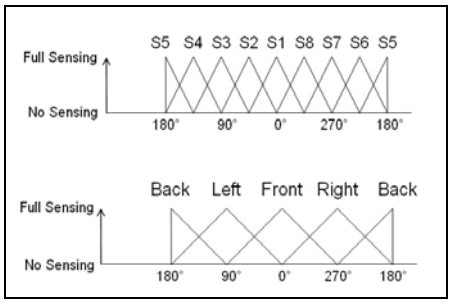

Fig. 16. Fuzzy function of inputs

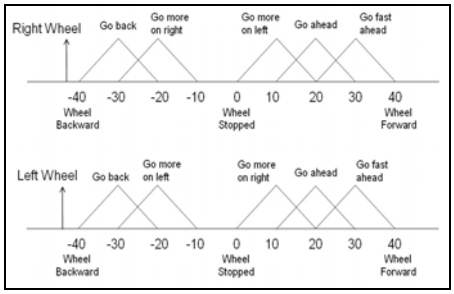

Fig. 17. Fuzzy function of outputs

We note here that the differential wheel robot needs to turn the right wheel forward more than the left wheel to be able to turn more on left. Right wheel full forward and left wheel full backward makes the smallest and quickest turn to the left.

To fulfill the needed behavior, we had to introduce an unconditional rule {Go ahead}, that just means that this rule is always activated whatever is the input of the sensor, and that is most useful when not any sensor sees an obstacles, that means if the robot is in a clear area, it should not stop, but continue going ahead till next obstacle.

With this inference engine, the robot do not have enough reactivity to turn when obstacles are met in front of it, as at that time, obstacle can not to be seen from left and right sensors. This front obstacle can make the robot just to stop in front of it.

To add turning reactivity when obstacles are in the front, we must act on sensors that are on the front left and front right (S2, S8) and make them activate quick turning rules.

To refine the previous behavior, our next inference engine will take into consideration every sensor independently:

- Go ahead
- **If** an obstacle is on S1 (front) **then** go back
- **If** an obstacle is on S3 (left) **then** go more on right
- **If** an obstacle is on S7 (right) **then** go more on left
- **If** an obstacle is on S5 (back) **then** go fast ahead
- **If** an obstacle is on S2 (front left) **then** quick turn right
- **If** an obstacle is on S4 (left back) **then** go fast ahead **and** Go more on right
- **If** an obstacle is on S6 (right back) **then** go fast ahead **and** Go more on left
- **If** an obstacle is on S8 (front right) **then** quick turn left

Figure 18 shows the new introduced rules of quick turn functions

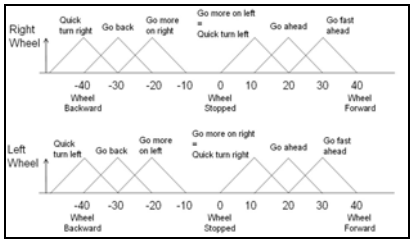

Fig. 18. Output function with quick turning

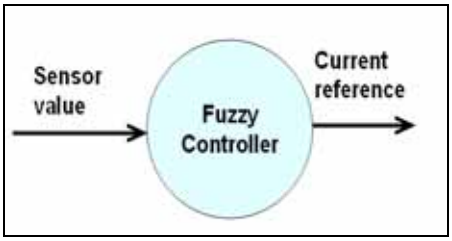

Fig. 19. Fuzzy Controller

The rules introduced for S4 and S6 were intended to keep their effect unchanged compared to the previous rules, as both rules activated at the same time will have averaged result, while sensor S4 (for example) on previous rules was already activating the rules of left and back by the same level.

Figures 15 A and B already show results of this fuzzy engine, we see that the robot fulfilled its task of exploring the majority of the space of the room where he was trapped. In 15 A he occupied most of the room space.

Figure 19 shows the last peace that makes our real time simulation works autonomously.

6. Conclusion

In these experiments, we made many simulated parts work together to compare the whole to the real mobile robot, and we succeeded in reflecting the controller effect on the robot behavior. The simulation allowed then to develop a fuzzy controller of the desired behavior.

Still there are a lot to do to take advantage of this method. By eyesight, we see the real robot having the same behavior, but better would be to compare the trajectories and evaluate the similarity in the behavior mathematically. More complex behaviors now have to be intended where another controller should monitor the fuzzy logic one, to detect when the robot gets back to the same place so the robot can have a mapping of the area in its memory. We didn't take advantage of the light adjust depending on obstacles distance.

A real time simulation that fits the real model has also the advantage of allowing the control of robots on conditions when signal takes a lot of time to go and come back between the control center and the robot, during that time, controlling the simulated model can be considered with sending orders to both the simulated and the real robot, then when signal of real position gets back, some corrections are just brought to the simulated model.

7. References

1. architecture for a car-like vehicle, In Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems, Vancouver (1998).
2. J. P. Laumond: La robotique mobile, Collection of Articles and Researshs, Paris, Hermès Sciences Publications (2001).
3. Y. Kanayama,Y. Kimura, F. Myazaki and T. Noguchi: A Stable Tracking Control for an Autonomous Mobile Robot, IEEE International Workshop on Intelligent Robots and Systems, Osaka (1991) 1236-1241
4. Y. Kanayama, Y. Kimura, F. Myazaki, and T. Noguchi: A stable tracking control method for a non-holonomic mobile robot, In Proc. of the IEEE-RSJ Int. Workshop on Intelligent Robots and Systems, Vol. 2 , Osaka, (1991) 1236-241
5. P. Reignier : Pilotage réactif d'un robot mobile - étude du lien entre la perception et l'action, Thèse de doctorat, Inst. Nat. Polytechnique de Grenoble, (1994)
6. I. Rivals, L. Personnaz, G. Dreyfus and D. Canas: Real-time control of an autonomous vehicle -A neural network approach to the path following problem, In Neuro-Nimes, Nimes (1994) 219-229
7. T. Lozano-Pérez: Spatial planning- A configuration space approach, IEEE Transactions on Robotics and Automation, (1983)
8. O. Khatib: Real-time obstacle avoidance for manipulators and mobile robots, International Journal on Robotics Research, Vol. 5, no. 1, Spring (1986) 90–98
9. L. E. Kavraki and J.-C. Latombe: Randomized preprocessing of configuration space for fast path planning, in Proceedings of the International Conference on Robotics and Automation (ICRA). San Diego, CA: IEEE Press, (1994) 2138–2139.
10. J. Laumond, F. Lamiraux, and S. Sekhavat: La robotique mobile. Systèmes automatisés. Hermes Sciences, (2002)
11. F. Lamiraux, D. Bonnafous, and C. V. Geem: Control Problems in Robotics Springer, Path Optimization for Nonholonomic Systems: Application to Reactive Obstacle Avoidance and Path Planning (2004)
12. F.Lamiraux, D.Bonnafous, and O. Lefebvre: Reactive path deformation for non-holonomic mobile robots, IEEE Transactions on Robotics and Automation, (2004)
13. O. Lefebvre, F. Lamiraux, and C. Pradalier: Obstacles avoidance for car-like robots: integration and experimentation on two robots, in International Conference on Robotics and Automation. New Orleans: IEEE, April (2004)