Autonomous Mobile Robots

Final Report

Vacuum Cleaning Robot

Nuno Gomes
Joaquim Ferreira

Instructors:
Professor Keith L. Doty
Professor Scott Jantz

21 July 1995

Dept. of Electronics and Telecommunications
University of Aveiro

Index

ABSTRACT 3

1 - INTRODUCTION 3

1 - INTEGRATED SYSTEM 4

2 - MOBILE PLATFORM 5

3 - SENSORS 6

4 - BEHAVIOURS 7

5 - CONCLUSIONS 9

APPENDIX A - ADD-ON ELECTRONICS 10

APPENDIX B - COMMENTED PROGRAM CODE 11

APPENDIX C - PHOTOGRAPHS 16

Abstract

This report describes the work done on the development of a vacuum cleaner robot prototype, both on hardware and software point of view. The basic assumptions made about the robot are that it has to transport the vacuum tool and a dust bag on a mobile platform driven by sensors and controlled by a set of behaviours that enable it to randomly navigate through a room or a house with the minimum human assistance. The features of the robot includes obstacle avoidance, collision detection, self detection of anomalies (fan motor overload, either due to bag full of dust or fan stuck, and low power on the batteries) with the consequent heading through a maintenance station.
The mechanical platform on which the hardware lies is a circle of wood with a diameter of 25 cm, with a hole in the middle where the vacuum cleaner fan motor is attached.

1 - Introduction

The research and development of an autonomous mobile robot prototype able to vacuum cleaning a room or even an entire house is not a trivial challenge. In order to tackle such a task, so that it could be completed in six weeks (the duration of the course), some simplifications and assumptions were made to the designers initial idea of an “ideal” autonomous vacuum cleaner. In this way, some functional requirements that would improve the robot performance were not taking into account due either to their inherent complexity or to their mechanical implications. Probably the decision that most affects the robot complexity is the ability of mapping the environment so that it would exhibit a much better efficiency (area coverage) when compared with the minimalist approach as the one followed (random navigation).
With the aim of keeping our robot as simple as possible, while able to perform the initial goals, i.e. an autonomous vacuum cleaner robot able to randomly navigate through a room or a house with the minimum human assistance, the following specifications were found:

- Obstacle avoidance.
- Floor detection.
- Collision detection.
- Battery monitoring.
- Autonomous battery charging.
- Fan motor current monitoring.
- Autonomous dust bag dump.

These specifications correspond to some of the expected behaviors that will be programmed into the robot. Other behaviors that will increase the overall performance of the robot, such as self calibration of the sensors and navigation with some memory (not completely random) were also considered.

1 - Integrated System

We will not spend much report space describing the background hardware that constitutes our robot, since most of it was presented to us as a "final" product by Scott Jantz, namely all the hardware interconnections, IR sensors hacking, etc. We only present the add-on electronics schematics designed by us in Appendix A.

We can resume the functionality of our robot as a programmer will see them:

- Eight analog inputs (Address 0x4000) for the IR sensors.
- Eight digital inputs (Address 0x5000) for the bumpers & control switches.

2 - Mobile Platform

In the design process of our vacuum cleaning robot we assumed that it has to transport the vacuum tool and a dust bag on a mobile platform driven by sensors and controlled by a set of behaviours. The vacuum tool includes the fan motor and a strip overture a few millimeters from the ground (underneath the platform) and almost as long as the diameter of the platform, so the efficiency of the fan could be maximized. The dust bag is attached to the fan and is situated on the top of the platform (figure 2).

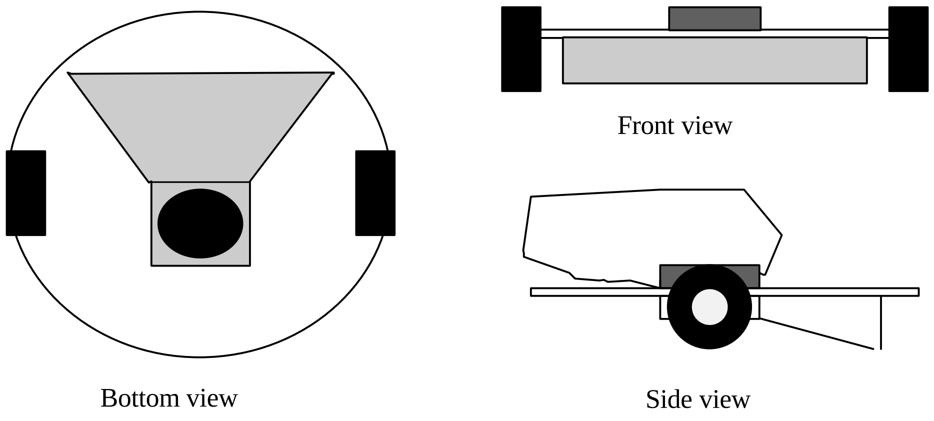


Figure 1 - Mechanical platform.

One of the most difficult problem we had to face on the design process was the relatively high weight the platform had to support. This is due to large batteries that we use (8*NiCd 1,2V and 4Ah), so we could have enough energy to drive both the robot itself (μP board, sensors, motors) and the fan motor (12V - 0,14 A) for at least a couple of hours. As a consequence of this extra weight the plastic glue that was used to attach the wheels to the motors keep on cracking. We choose not to substitute this glue by a stronger one because with this glue if we made a mistake we could always repair it easily.

3 - Sensors

The vacuum cleaner robot has a suite of sensors that enable it to perform the tasks described on the Introduction.

Table 1 - Vacuum cleaner sensor suite

Sensor Type	Function	Location	Number
Infrared (IR)	Obstacle avoidance	Front	3
Infrared (IR)	Floor detection	Near the wheels (bottom)	2
Infrared (IR)	Beacon detection	Top front	3
Contact Switches	Collision detection	Periphery	4
Switch	Fan current high (overload)	μP Board	1
Switch	Battery power low (hungry)	μP Board	1

The IR sensors used for obstacle avoidance and floor detection are shown in figure 2.

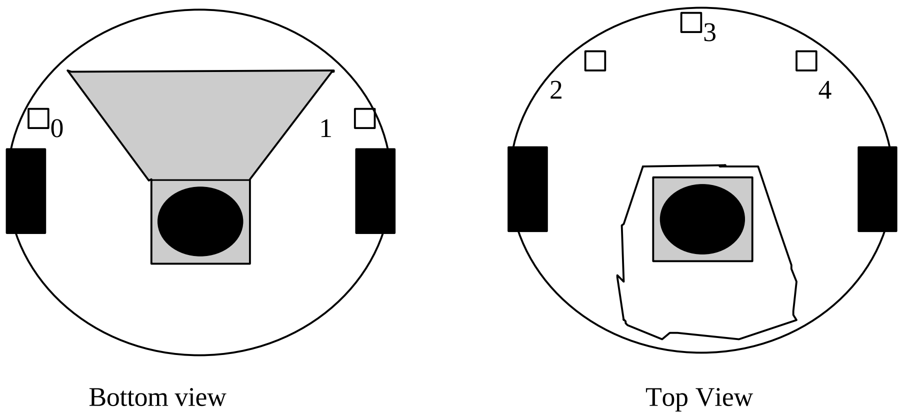


Figure 2 - Bottom & Top view of the robot and IR sensor distribution.

In the previous figure, IR sensor 3 is facing front, IR sensor 2 and 4 are also facing front but they are rotated by 45 degrees. IR sensors 0 and 1 are aimed at the floor near the wheels, to prevent the robot to fall in stairs (or even a single wheel of it), independently of the angle on which the robot forward approaches the stairs.

Another three IR sensors are used for beacon heading behaviour. These three IR sensors are placed on the top of a 30 cm plastic mast (white tube). One of these IR sensors was designed to distinguish between polarized IR and not polarized IR, unfortunately the polarized glasses lens that we bought were unable to polarize the IR radiation. If the polarization had work as we expected then the robot could discriminate two beacons. Since this approach was unsuccessful, we chose to include in our specifications only one beacon that identifies both the charging and the maintenance stations of our robot. We call the neighbourhood of the beacon the docking area.

We choose not to implement the electronics associated with the fan motor current monitoring and the one related with the battery power sensing, for two main reasons. First, we would have to multiplex the μP analog inputs, since we were already using all the eight available (for the eight IR). Second we think that is more important to develop the respective behaviours than to spend time doing the electronics and since we did not have enough time to do it we decided not to. Instead of using real analog measurements for these two parameters we decide to input them to our control program as digital inputs, so two extra switches were added to the μP board.

4 - Behaviours

The behaviours exhibit by our robot are the following:

- **Obstacle avoidance.& Floor detection** - This reactive behaviour is driven by five IR sensors distributed by the front (three) and the bottom (two) of the robot.

- **Collision detection** - The collision detection behaviour processes the sensory data provided by the four contact switches bumpers placed along the periphery of the robot, three in the front and one in the back. This capability is needed because the IR sensors are quite directive and, thus, they cannot detect sharp obstacles as chair legs, for example.

The previous behaviours are integrated in the same software function called *avoid_obstacles()* since the latter extends the concept of the former.

- **Overload** - The robot only processes this behaviour when the correspondent switch is activated. In this way we fake that the fan motor current is high. When this situation occurs two reasons can be behind it, either the dust entrance is blocked or the dust bag is full. The robot reaction to this digital input will be stopping the fan motor for a while, trying to remove a possible blocking material. Then, the fan motor will is turn on again and if the its current keeps the same value as before (digital input still activated) the robot knows that the dust bag is full or the fan motor is being blocked. In either case the fan motor is stopped and the robot looks for the docking area (maintenance station in this case). If the fan motor current decreases to its normal value (digital input deactivated), it means that an object was blocking the fan motor and it dropped when the fan motor was stopped. In this situation, the robot proceeds its normal activity.

- **Hungry -** On this behaviour, as on the previous one, the robot only processes this behaviour when the correspondent switch is activated. The robot is hungry when the battery charge drops bellow a certain level. In this case (switch activated), the fan motor is stopped and the robot looks for a charging station. If meanwhile the robot looses the beacon heading because it bumped into some object or avoided some object, then it goes to a random behaviour and tries to look for the beacon again. Hopefully it will succeed and it will find the heading again.

- **IR sensor self calibration**. A continuous self calibration mechanism for the IR sensors was devised. Basically we read the five analog inputs from the *sensor_module()* when the robot is not seeing any obstacle and he is seeing the floor. If the sum of these measurements, taken on these conditions, is not in the vicinity of the sum of the current thresholds, then those measurements will become the new threshold values. In this way, the robot should be able to adapt itself to the light intensity changes of the environment.

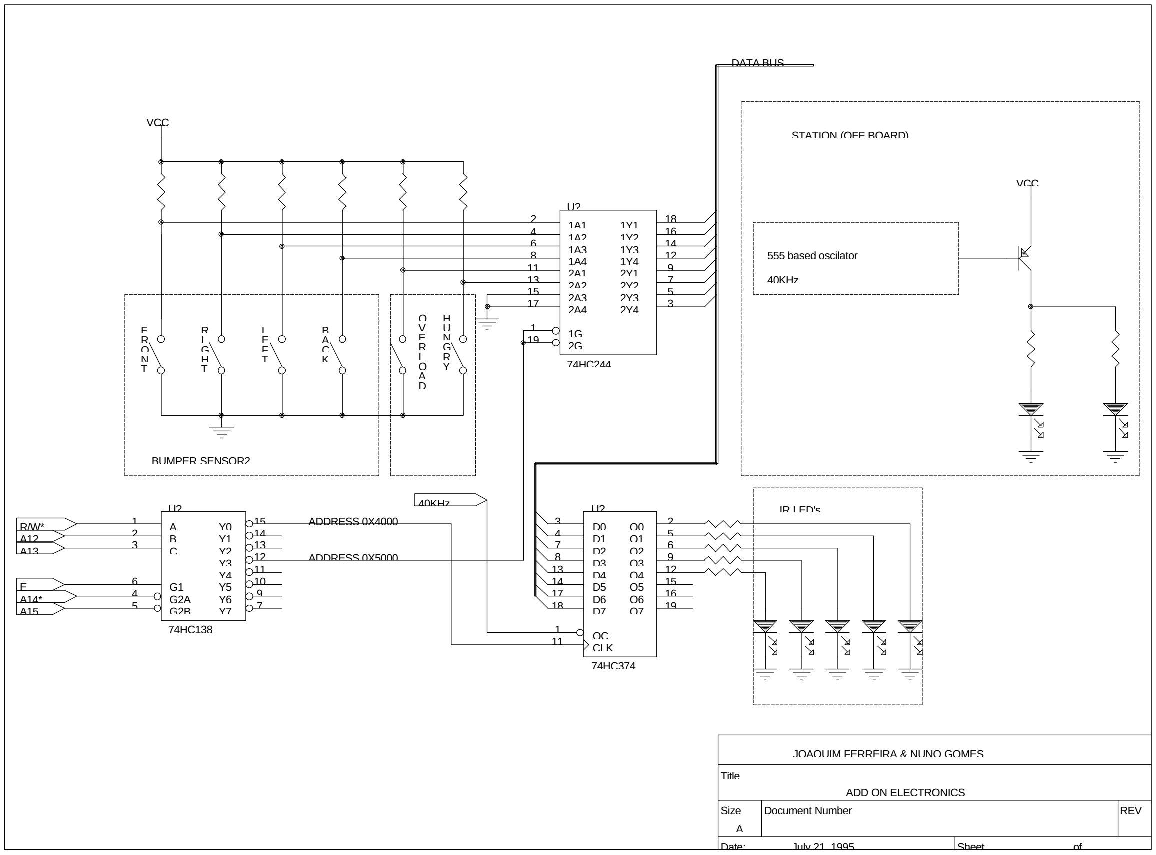
A detailed explanation of the last behaviours can be found on Appendix B, Commented Program Code.

5 - Conclusions

Throughout the development of the robot several problems arisen due, mainly, to time constrains. Nevertheless, we were able to set up a basic platform, including a sensorial suite, that can be used in the future as a tool to develop more sophisticated algorithms. The basic obstacle avoidance and collision detection behaviours has been accomplished and it provides reasonable performance. However, if we have to start over again, we would use non-linear dynamics based algorithms for obstacle avoidance behaviour.

There was not enough time to fully test the performance and limitations of our robot, so we do not have a clear idea about some fundamental parameters as the area coverage and its efficiency. These parameters are fundamental in our design because they should be the emergent functionality of a vacuum cleaning robot.

Appendix A - Add-on Electronics



Appendix B - Commented Program Code

```
/*      program file   :        robot.c (pre release version 1.0)              */
/*      programers     :        Joaquim Ferreira and Numo Gomes  */
/*      date           :        1995, July 21st

/*      Global declarations and initialisations                                */
int     pid1,pid2,pid3,pid4,
        Left, Right,Center, FloorRight, FloorLeft,Floor0,Floor1,
        FloorMissing=0,
        ObjectRight,ObjectLeft,
        ObjectFront,NoObject=1,
        SUM_M=500,SUM_T=500,TS=100,T0=95,T1=105,T2=95,T3=115,T4=105,
        s_right,s_left,s_front,
        digital_in,
        bumper_front,
        bumper_right,
        bumper_left,
        bumper_back,
        hungry,overload,
        docking_area,
        motor_right=1,
        motor_left=0;

float   wtime=0.2,spin_time=0.25;

/*      Self calibration module                 */
void get_thresholds()
{
if ( !(FloorMissing) && (NoObject) )        /* I'm running cool */
  if ( (SUM_M > SUM_T+15) || (SUM_M < SUM_T-15) ) /* Ops! Suddenly there is too much
*/
  {                                                    /* or not enougth light                    */
      poke(0x4000,0xff);
      sleep(wtime);
      T0=analog(0);     /* These will become the new thresholds for */
      T1=analog(1);     /* obstacles and floor detection                      */
      T2=analog(2);
      T3=analog(3);
      T4=analog(4);
      poke(0x4000,0x00);
      SUM_T=T0+T1+T2+T3+T4;     /* This will become the new perception of environmental
*/
                                /* brightness */
  }
}

void sensor_module()
{
 while(1)
   {
    /* Avoidance readings */
    poke(0x4000,0xff);          /* Turn on the LED's */
    sleep(wtime);               /* Wait for the transients to die */
    Floor0=analog(0); /* Take the readings */
    Right=analog(1);
    Center=analog(2);
    Left=analog(3);
    Floor1=analog(4);

    /* Beacon searching readings */
    s_front=analog(5);
    s_right=analog(6);
    s_left =analog(7);

    poke(0x4000,0x00);          /* Turn off the LED's */

    /* Bump readings */
    digital_in=peek(0x5000); /* Take the readings (negative logic) */
    digital_in = ~digital_in;  /* Invert the readings and go to positive logic  */
    bumper_front = digital_in & 0x01;         /* Mask the individual bump switches */
    bumper_right = digital_in & 0x02;
    bumper_left  = digital_in & 0x04;
    bumper_back  = digital_in & 0x08;

    /* Hungry and overload readings */
    overload = digital_in & 0x10;
    hungry   = digital_in & 0x20;
    docking_area=(s_front >= 130) && hungry;      /* If the beacon is blinding me and     */
                                                  /* I'm hungry then I reach the docking area */
    SUM_M=Floor0+Floor1+Right+Left+Center;    /* How much I see from every where?
          */
    ObjectRight=(Right > T1);          /* Is there an obstacle by my...*/
    ObjectLeft=(Left > T3);
    ObjectFront=(Center > T2);
    FloorRight=(Floor0 > T0);          /* Is there floor by my... */
    FloorLeft=(Floor1 > T4);

    FloorMissing=( !(FloorRight) || !(FloorLeft) );
    NoObject=( !(ObjectRight) && !(ObjectLeft) && !(ObjectFront) );
   }
}


void avoid_obstacles()
{
 while(1)
   {
    if (bumper_front)
    {
      motor(motor_right,-75.0);
      motor(motor_left,-75.0);
      sleep(spin_time);
      motor(motor_left,75.0);
      sleep(spin_time);
    }
    else if (bumper_left)
    {
      motor(motor_left,-40.0);
      motor(motor_right,-40.0);
      sleep(spin_time);
      motor(motor_right,75.0);
      sleep(spin_time);
    }
    else if (bumper_right)
    {
      motor(motor_left,-40.0);
      motor(motor_right,-40.0);
      sleep(spin_time);
      motor(motor_left,75.0);
      sleep(spin_time);
    }
    else if (bumper_back)
    {
      motor(motor_right,75.0);
      motor(motor_left,75.0);
      sleep(spin_time);
      motor(motor_left,-75.0);
      sleep(spin_time);
    }
    else if (!(FloorRight) && (FloorLeft) )
    {
    motor(motor_right,-75.0);
    motor(motor_left,0.0);
    }
    else if ( (FloorRight) && !(FloorLeft) )
    {
         motor(motor_left,-75.0);
         motor(motor_right,0.0);
    }
    else if ( !(FloorRight) && !(FloorLeft) )
    {
         motor(motor_right,-75.0);
         motor(motor_left,-25.0);
    }
   else if( (NoObject) && !(FloorMissing) )
      {
        motor(motor_left,75.0);
        motor(motor_right,75.0);
       }

    else if ( (ObjectFront) && !(FloorMissing) )
    {
         motor(motor_right,75.0);
         motor(motor_left,-75.0);
         sleep(spin_time);
    }
    else if ((ObjectRight) && !(ObjectLeft || ObjectFront) && !(FloorMissing))
        {
         motor(motor_left,75.0);
         motor(motor_right,0.0);
        }

    else if ((ObjectLeft) && (!(ObjectRight || ObjectFront)) && !(FloorMissing))
        {
         motor(motor_left,0.0);
         motor(motor_right,75.0);
        }
}
}

void search_station(void)
{
while(1)
```

```
{
if (!hungry) defer(); /* If I’m not hungry then I will do something else */
if ((s_front > TS) && (!docking_area))      /* I found the beacon but I’m not in the docking
*/
                                            /* area. Then... */
{
  if ((s_left > s_right) && (s_left >= s_front))
  {
    motor(motor_left,-20.0);
    motor(motor_right,40.0);
    sleep(spin_time);
  }
  else if ((s_left < s_right) && (s_right >= s_front))
  {
    motor(motor_left,40.0);
    motor(motor_right,-20.0);
    sleep(spin_time);
  }
  else if ((s_front > s_right) && (s_front > s_left))
  {
    motor(motor_left,40.0);
    motor(motor_right,40.0);
  }
}
}
}

/* This is the fan control module! */
void fan_control(void)
{
while(1)
{
  if (overload || hungry) poke(0x1000,0); /* Too simple to comment! */
  else poke(0x1000,0xff);
}
}

void main()
{
pid1=start_process(sensor_module());
pid4=start_process(fan_control());
pid2=start_process(avoid_obstacles());
pid3=start_process(search_station(),200);

while(1)
{
if (docking_area)      /* I was looking for the maintenance station */
                       /* because I was hungry or overloaded an I */
                       /* have just found it. Then...                */
{
     motor(motor_right,0.0); /* Stop the motors... */
     motor(motor_left,0.0);
     kill_process(pid4);                /* ...and have some peaceful rest!! */
     kill_process(pid1);
     kill_process(pid2);
     kill_process(pid3);
}
}
}
```

Appendix C - Photographs

-

Photograph 1

Photograph 2

Photograph 3

Photograph 4

Photograph 5

Photograph 6