EEL 5666 Summer 1999

Intelligent Machine Design Laboratory

# ττ

# *The Sweeper Robot*

## Final Report

Laurent Houssay

Graduate Research Assistant

Mobile Robotics for Hazardous Laboratory

Nuclear & Radiological Engineering Department

University of Florida

# Table of Contents

# Abstract

This robot is designed to clean the dust on the white lines of the roads, it also follows the kerb to clean the same side of the road. This robot is autonomous and intelligent that means that it analyses its environment and reacts by itself, it also uses its own source of energy.

# Executive Summary

The robots are generally designed to execute works that are repetitive or dangerous. With this application the work is both repetitive and dangerous. The purpose of this robot is to clean the side of the road that is not accessible because of the traffic. This sweeper robot has two way to execute this job, the first one is to follow the white line on the road. The other way consists in following the kerb. The following paper describes the design and the methods used to solve this technical problem. This robot has now built and works fine. Unfortunately its application that was initially supposed to be outside, is now limited to a more regulated environment in a room.

# Introduction

I have observed that dust, small stones, etc… generally hide the continuous white lines of the road. In order to increase the visibility of the lines and consequently the security, there is a need for a sweeping. Sometimes, the traffic is so intense that it is very dangerous to clean safely the left side of the road. Furthermore, in many states the roads

are so long that a white line can be continuous for more than 10 miles. Such a robot equipped with solar cells would clean these lines for a little money and increase the security.

# Integrated System

This robot can be divide into several parts:

- The brain: 68HC11 EVBU and ME11
- The actuators: Motors , wheels and sweep
- The data acquisition system: the four multiplexers and the latch
- The sensors

These elements interact together and with the environment of the robot: (Cf. Figure 1)

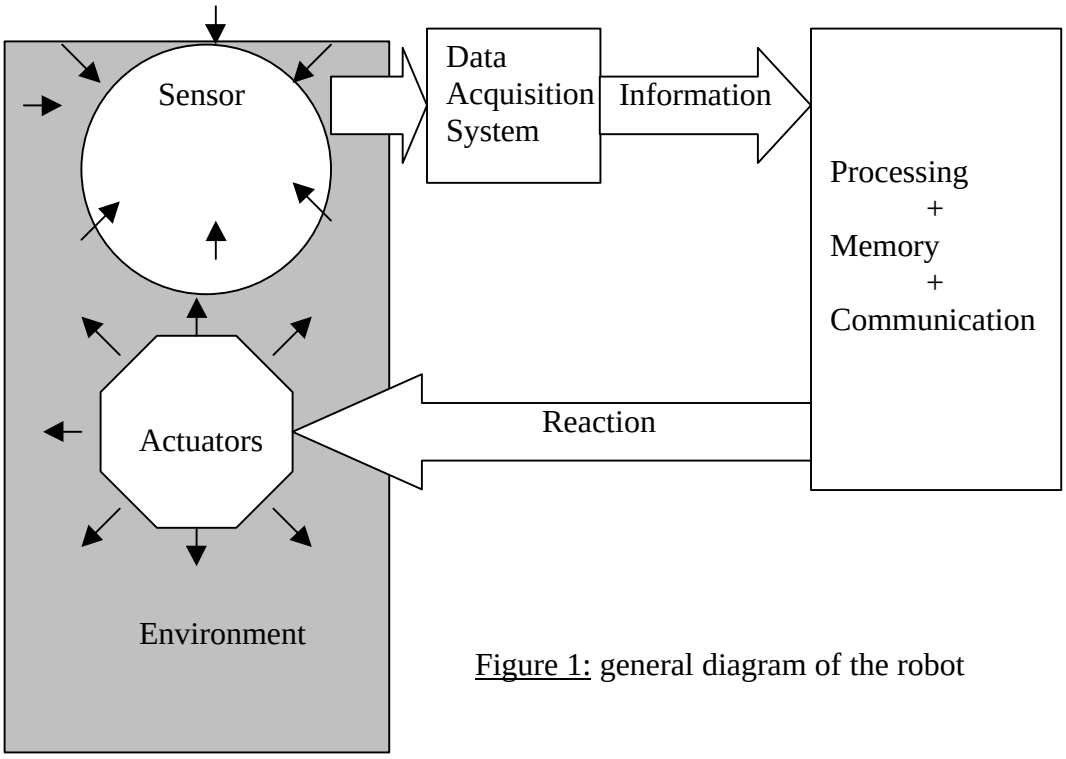


Figure 1: general diagram of the robot

The complexity of the road cleaning work has specific needs. The road is an “hostile” environment where unexpected things may happen or be found. Therefore the robots needs a lot of inputs to analyze as best as possible its environment and to give the most appropriate reaction. With this robot thirty inputs both digital and analog inform the robot about the modifications of its environment.

Each element of the robot is well explained in the following chapter, but it is possible to make some interesting remarks.

- Since the DC motor that spins the sweep is driving a lot of courant, two separated pack of batteries supply the robot. One eight pack is powering the ME11 and the EVBU only. Another pack supplies the DC motor of the sweep and also all the data-acquisition system. This way the board is not reset when the motor is turned on. Furthermore, if a bad design leads to a short in the data acquisition system, this separated circuit does not damage the 68HC11.
- The robot has about thirty inputs from the sensors, but it also has several outputs. The most obvious are the motors that control the wheels and the sweep. There are also many LEDs that indicate the state of a sensor like for the bump switches and the mercury sensors.

# Mobile Platform

The platform of the robot is realized in wood. Many pieces were cut using the T-tech machine, but several pieces were designed by hand. The most obvious thing on this robot is the “cow pusher “. More than just fun, it has a specific purpose. When cleaning the road, small stones in contact with the front bump should not stop the robot. Therefore, a selection must be done. As shown in figure 2 and 3, the “cow pusher” method enables the robot to start a collision avoidance procedure only with big obstacles and to push the small objects.

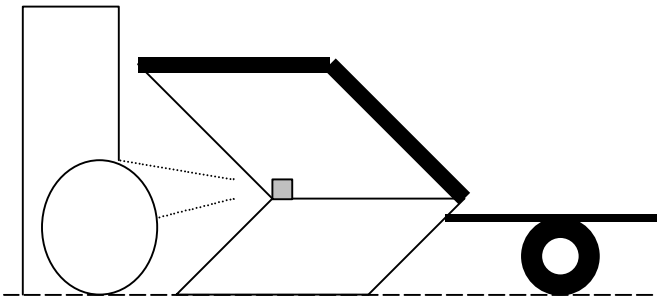

Figure 2: Detection of big obstacles

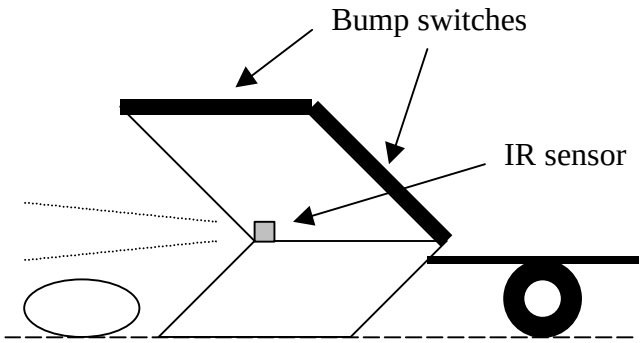


Figure 3: Ignorance of small objects

The rotating sweeper and its motor are placed under the platform, behind the “cow pusher”. The dust is swept forward and is kept in a small tank between the reflector and

the sweep. The free wheel is also place at this position. The two DC motors and their wheels are attached to the platform behind the motor of the sweep. There are several thickness of wood between the motor and the main platform in order to have the same height as the sweep. The two eight packs of batteries are located between the two motors under the platform. Finally the CDS cells are installed and the backs of the platform very close to the ground with an air gap of only 2-mm. See figure 4.

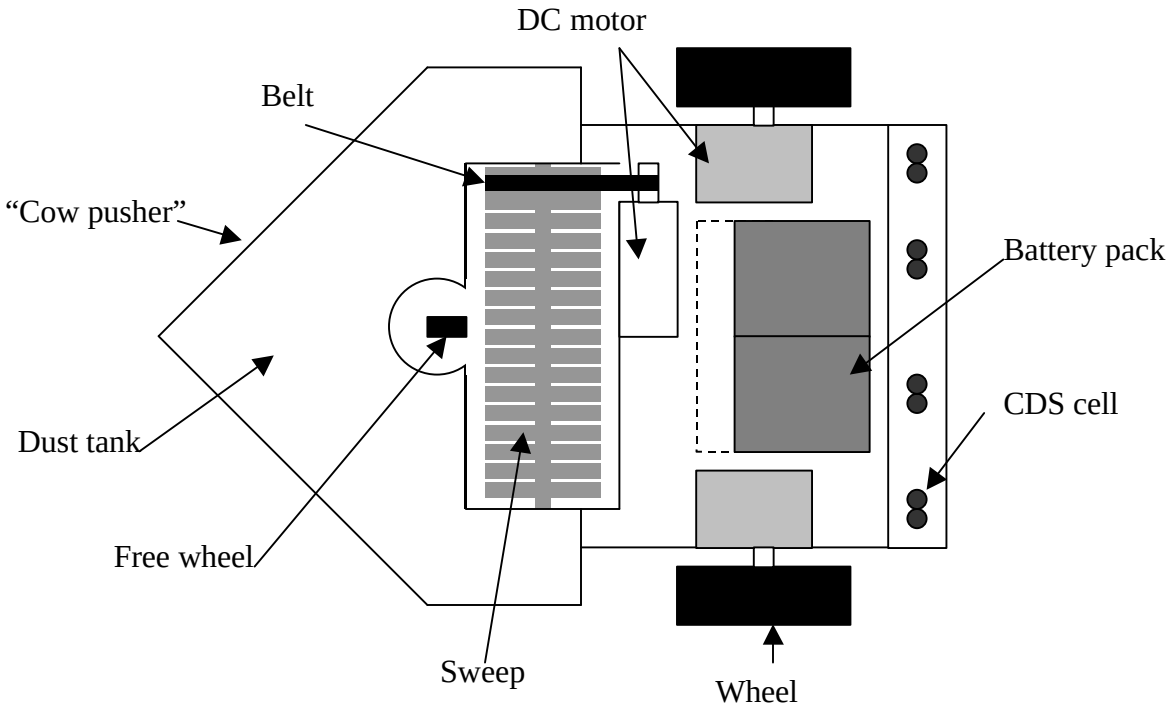


Figure 4: Platform, bottom view

# Actuation

## Scope

A total of ten bump switches are installed on the robot (Figure 5). Each bump The robot moves with two wheels that are attached to two DC motors. The motors are fully hacked servos from Mekatronix. A last free wheel is placed in front of the robot. A powerful DC motor directly connected to an eight-pack battery moves the rotating sweep.

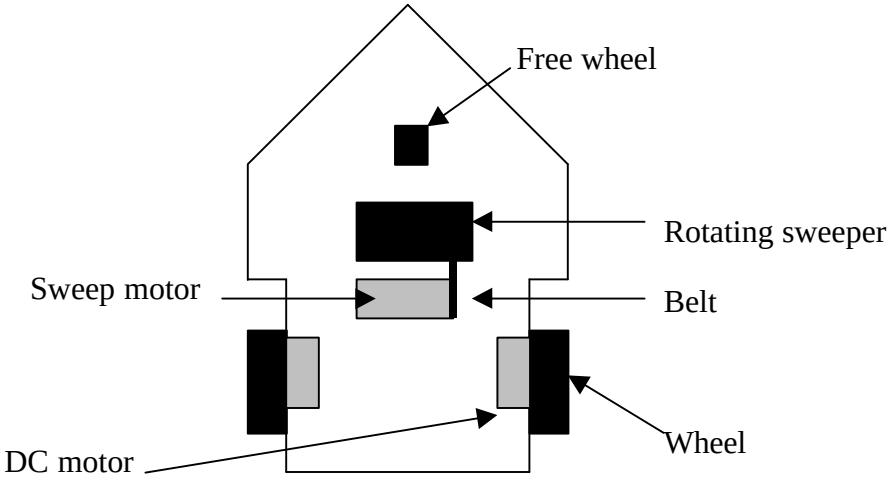


Figure 5: Position of the actuators

## DC motors & wheels

The two wheels that move the robot are attached to two small DC motors. Those motors are Mekatronix fully hacked servos connected to the ME11. The 68HC11 sends a Pulse Width Modulated (PWM) signal to the motor driver. This signal

controls the motor driver to modify the speed from 0 to + maximum speed or – maximum speed. The command to turn the motor on is: motor_me(A,B);

A is 0 for the left motor and 1 for the right motor. B is the speed: an integer between –100 and 100. Unfortunately these motors have many non-linearity. For instance in order to move the robot straight, the speed of the left motor must be 75 whereas the speed of the right motor is 100. Secondly, the velocity is not proportional to the value B. For example, if B=25 the velocity of the robot is half the velocity at B=100. In addition, those values are different if the robot is going backward. This is really unfortunate because I wrote a program that was reversible. Whenever the line or the wall is lost during a line or wall-following operation, the robot was suppose to explore its environment to find another wall or line or to go back to its initial position if it finds an obstacle. The robot would go back to its initial position by executing the procedure backward. I tested it, it worked fine, except that even if the procedure was executed backward, the robot did not go back to its exact initial position. This was because even if the program is the same, the actual movement of the robot is different backward and forward.

A contribution to this problem is certainly the free wheel. It is not completely free and tends to keep its initial position. Therefore if the robot turns, the free wheel acts as a brake. I installed elastic to this free wheel, then this wheel always goes back to its initial central position and loses a portion of its non-reproducibility.

For a long time (!) I had a problem with the DC motors. When I turned them on, they did not start and reset the board. If I helped them to start however, they

worked fine without resetting the board. For a long time I thought it was a battery problem, when Scott helped me to find that I just plug the motor driver backward (!). The lesson is that a motor can work when the motor driver is plugged backward but gives the same clue as a low battery problem.

## Sweeper

The major task of the robot is to clean the road. I used a hairbrush as a sweep. This brush spins because of a belt linked to a pretty big DC motor I found in my lab (Cf. Figure 6). I do not have any information about this motor. I only know that it works fine with an eight pack of batteries. The DC motor that spins this sweep is connected to the batteries. One 5-volt relay from Radioshack is used as a switch to turn on this motor at the same time the IR LEDs are turned on. In addition to this relay, a manual switch enables the use of the motor to save the energy.

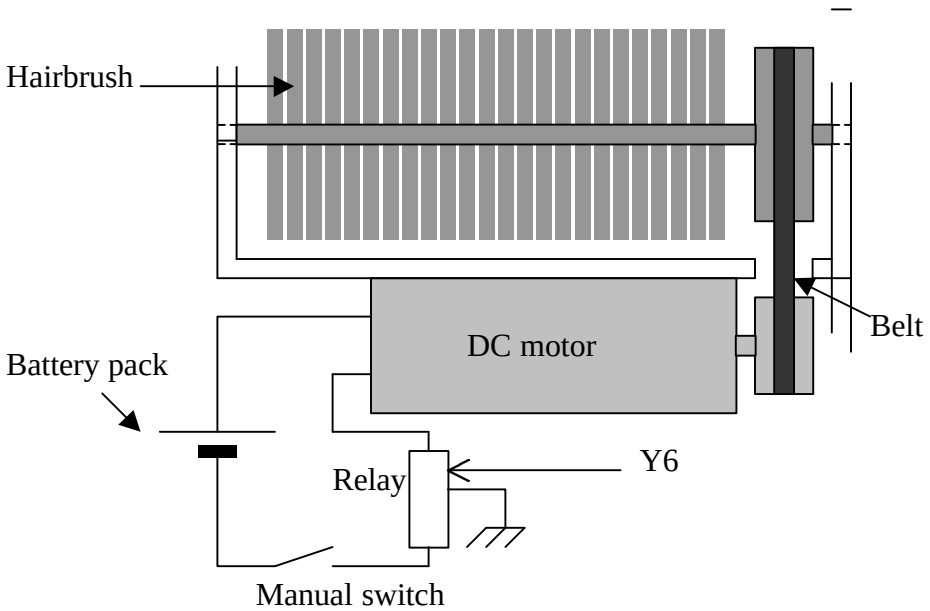


Figure 6: Rotating sweeper module

# Sensors

## Scope

My robot uses four types of sensors: bump switches, CDS cells, IR sensors and a stability sensor. The total number of sensor analog inputs is thirty. To designate which sensor I want to calibrate, I also add some switches that are connected to the analog port of the 68HC11. Since the 68HC11 has only eight analog inputs, I installed four eight-input multiplexers. This gives a maximum number of forty inputs. I am using only thirty-five of those forty inputs.

## Infrared Sensors

Five sharp infrared sensors are installed on the platform (Cf. Figure 7). One is placed at the front of the robot to detect obstacles. The four other IR sensors are located on each side of the robot. On each side two IR sensors control the wall following behavior by measuring the distance to a wall.

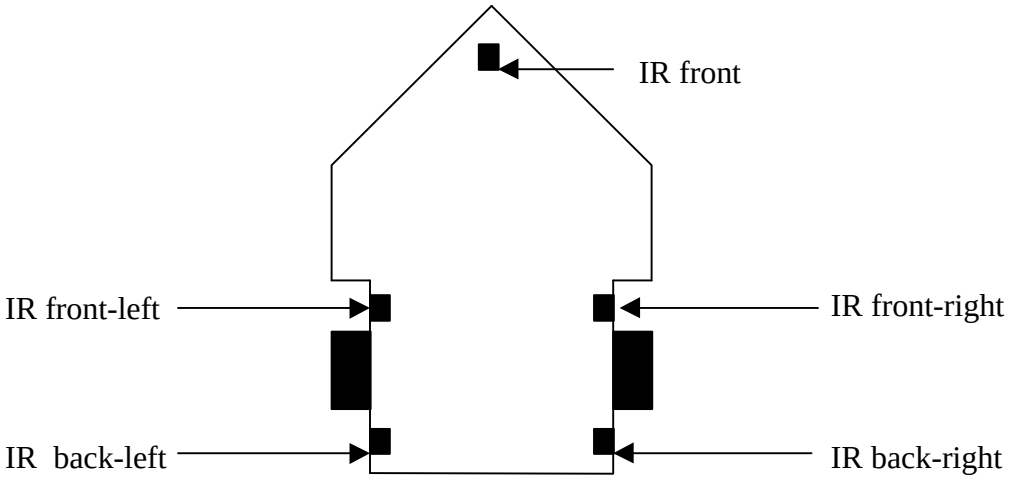


Figure 7: Position of the IR sensors

Each IR sensor is composed of two elements(Cf. Figure 8). The first element is a LED emitting light in the IR This LED associated to a 300 ohms resistor is powered by a +5 volts source modulated at 40 kHz. When the light hits an obstacle, a fraction of the emitted light arrives to the IR receiver. This receiver is hacked in order to convert the normally digital output to an analog output (Cf. Appendix 1). When no 40 kHz modulated IR light is detected, the output is about 1.7 volts. The maximum reading is 2.5 volts when the obstacle is exactly in front of the sensor. The LED and the receiver are assembled together on the same piece of board. A short piece of black tube is attached around the LED to focus the IR beam.

The output of the IR receiver is not linear but logarithmic. The useful range of utilization is from 0 to 40 cm, which is exactly the range needed for a good wall-following operation.

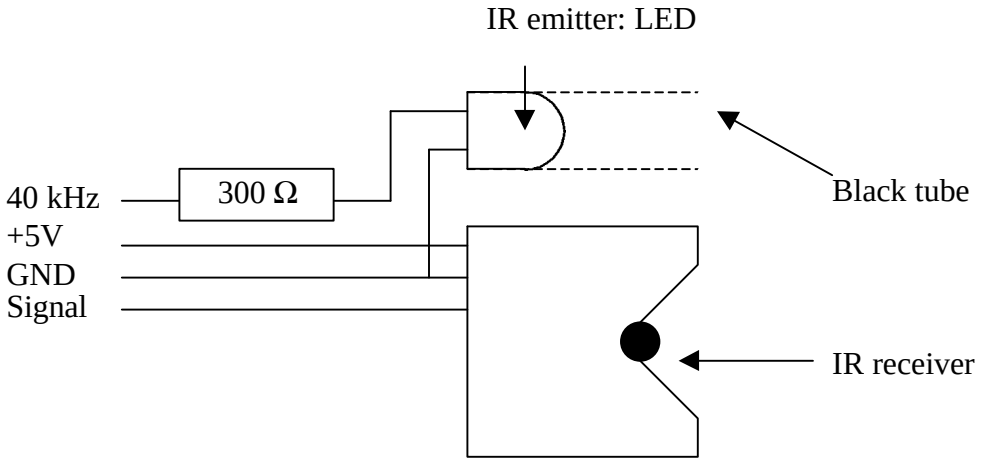


Figure 8: Composition of the IR sensor

# Bump Switches

A total of ten bump switches are installed on the robot (Figure 9). Each bump sensor is composed of one bump, which is a long piece of wood, and two contact switches. I use very tough contact switches that I found in my lab. When the bump is pressed, it produces a contact in the switch and connects the +5 volt to the corresponding analog input. When the bump is not pressed, the switch is pushing the bump to the outside and no contact occurs. The bump must be pressed more than 3 millimeters to produce a contact (Cf. Figure10).

Each bump module is connected to a LED in order to provide an efficient debugging method. Whenever a bump is pressed, the LED lights on and when the corresponding analog port is read the reading is equal to 255 or 254 enabling the appropriate reaction.

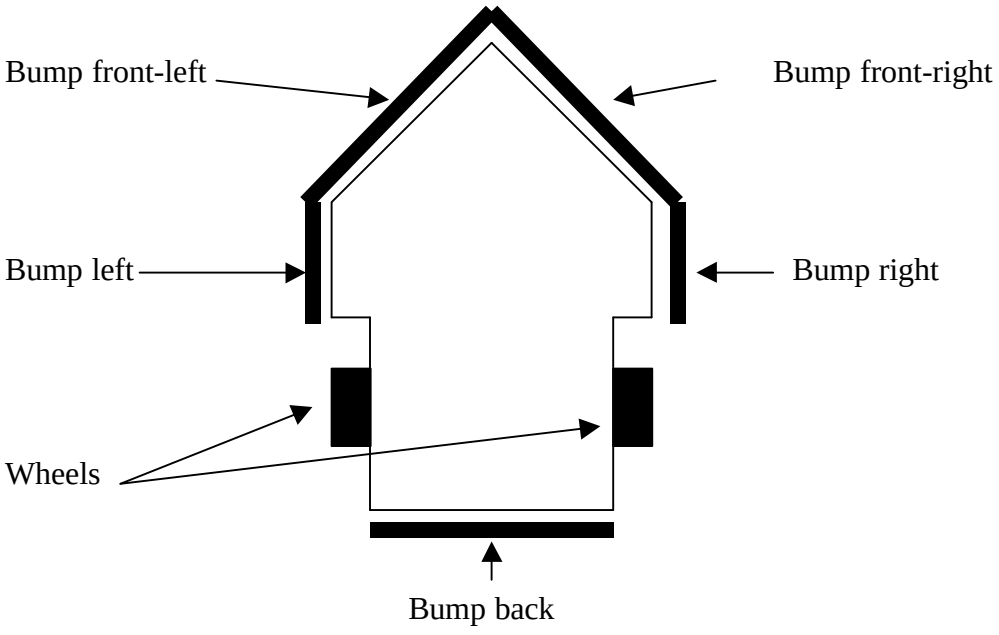


Figure 9: Bump Switches Location.

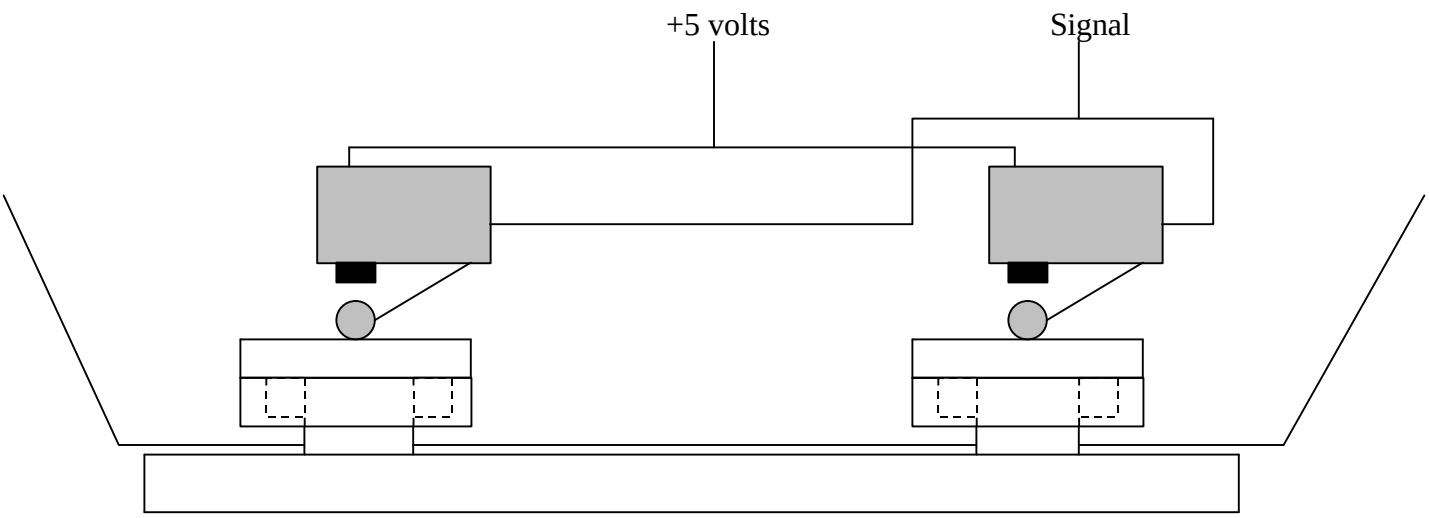


Figure 10: Bump switch

# CDS Cells

An array of eight CDS cells is used to follow a line on the floor. My first idea was to use two arrays of cells: one when the environment is dark (under a tree) and the other when the environment is brighter, depending on the value of the resistor I would add. . The additional CDS placed on the top of the robot would then control the choice of the best array. This method is inadequate because the CDS have to be as close to the ground as possible in order to avoid the noise from the outside. Therefore, a CDS reading based on the detection of the fraction of daylight reflected by a line is inefficient. I changed then the approach, and add four LEDs to light the line.The CDS are located at the back of the robot, with a very small gap (2 mm) between the cell and the floor.(Cf. Figure 12).

Now each module of CDS is like the following: see figure 11:

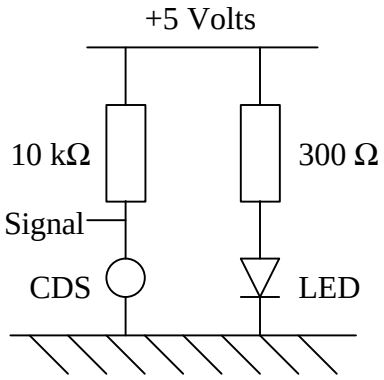


Figure X: CDS cell circuit

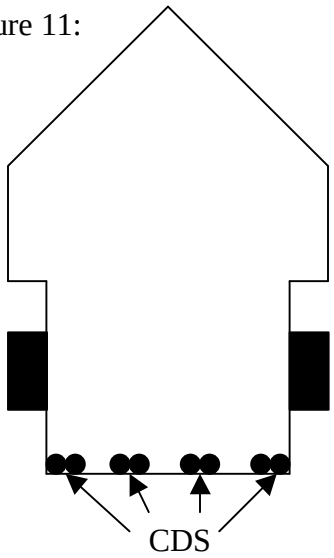


Figure 12: CDS cell location

# Mercury Stability Sensor

The purpose of this sensor is to provide a tool to the robot in order to stop and to avoid a fall. The robot is designed to clean the white lines of roads. Many damages would occur if the robot falls down into a ditch. Such behavior must be avoided by detecting changes in the horizontal plan of the robot. The slopes of roads, especially highways are limited by federal regulations and should not create any interference with the final goals, which are the detection and the avoidance of ditches. Many mercury detectors are available for the same purpose. This one however has the advantage of giving the direction of highest slope. This way, it is possible to modify the movement of the robot to put it in the right direction.

The theory of operation of this sensor is extremely simple. In a small and squared mercury tank, a central screw provides +5V through the mercury to eight screws that are distributed on a circle. (Cf. Figure 13).

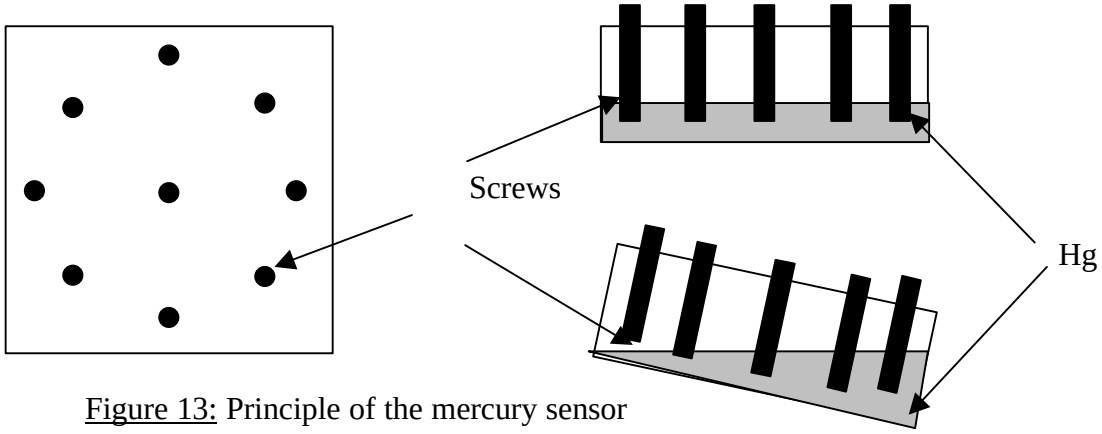


Figure 13: Principle of the mercury sensor

Whenever the ground is not horizontal, the contact is lost with the screw opposed to the direction of highest slope. The eight screws are connected to eight analog ports of the 68HC11. With a threshold value, it is easy to determine whether there is a contact or not. Each screw is connected to a specific input in order to keep the information of the direction of fall. Then the microprocessor runs a fall avoidance procedure to correct the trajectory.

The waste management lab of the Department of Nuclear Engineering provided the mercury. The tank is a wood box, 6*6*2cm. The box is coated inside and outside with latex in order to seal the mercury hermetically. The screws are basic screws, 3cm long. The quantity of mercury provides an amount at least 4mm deep at the bottom of the box.

The critical angle where the contact is lost is not absolutely isotropic, it varies from 4 degrees to 7 degrees depending on the direction. The measurement is never exactly reproducible. Those values seem very small, but represent a slope from 7% to 12%, which is the order of magnitude of the maximum slope of the highways. Each electrode is connect to an analog input and to an LED associated to a protection resistor. This way, it is easy to check if the sensor is working properly. (Cf. Figure 14)

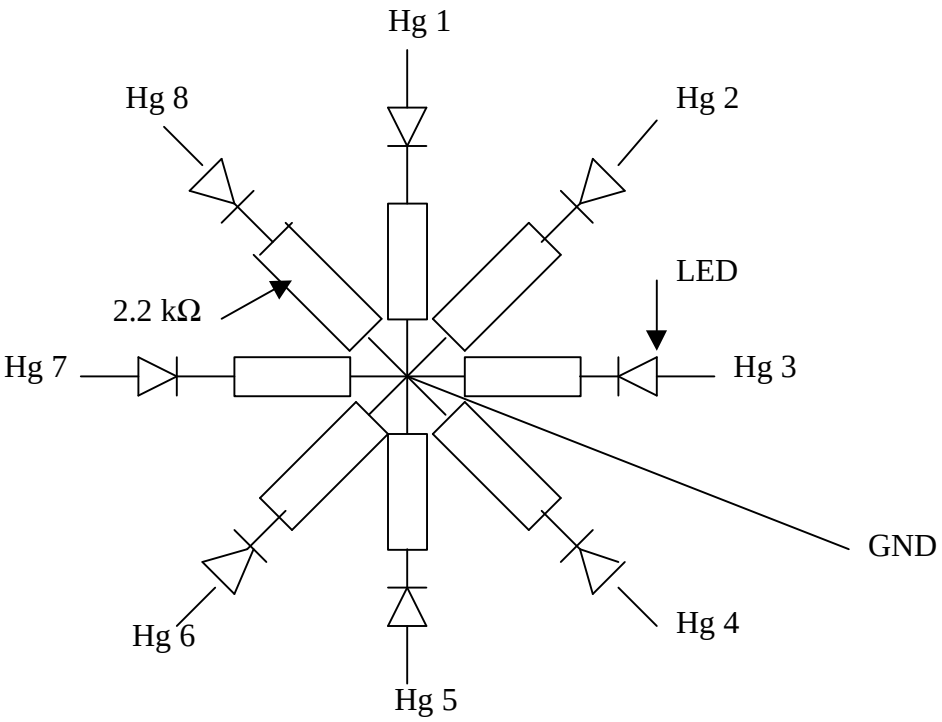


Figure 14: Mercury sensor debugging circuit.

As unbelievable as it seems, there is no electrical contact between the mercury and the screws. The electrical resistance between screws is bigger than several Megohms. There is no failure in the design, meaning that there is plenty of

mercury to keep the contact and the screws are also long enough. Therefore, there has to be other reasons. The first would be some kind of chemical incompatibility, which is hard to believe for metals. The best explanation is certainly the shape of the screws. The irregular surface of the screw does not provide enough space to beat the strong surface tension of the mercury, the two surfaces are only tangent. (Cf. Figure 15). Consequently a bolt is added at the end of each screw and provides an excellent contact with the mercury.

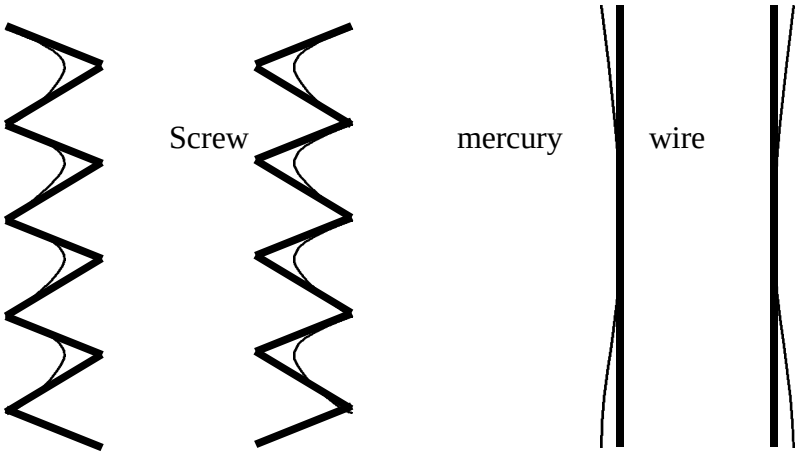


Figure 15: problem of electric contact between the mercury and a screw

Another problems come from the strong surface tension of the mercury. Inside the box, the mercury does not necessarily occupy all the available space. On a horizontal plan, it does, but when there is a slope it is different. The following figure shows that the critical angle arrives sooner than expected. (Cf Figure 16)

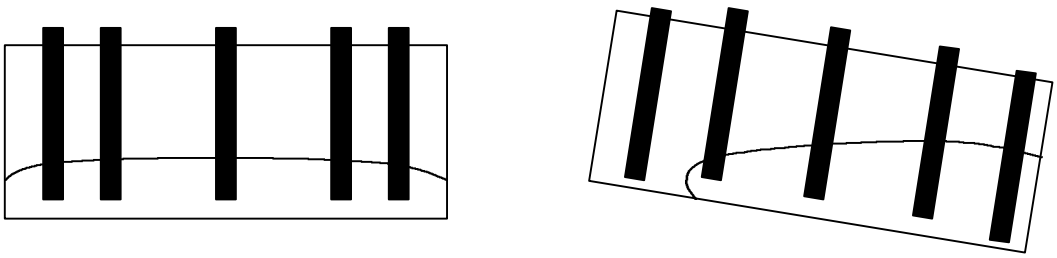

Figure 16: Problem of Mercury surface tension

Finally, a last problem completes the unreliability of this sensor. If the sensor is not powered, an ohmmeter shows that the contact always occurs whenever it is check. Unfortunately, I have discovered that as soon as a voltage is introduced between the electrodes, an impressive amount of black powder is produce. This powder is lighter than the mercury and floats on the mercury. A significant amount of powder is then deposited on the bolt and the contact is lost. So far, I have been able to get as much volume of powder as the initial volume of mercury I had. This chemical reaction is not an oxydo-reduction because the electrodes are the same. In addition, the volume of mercury does not decrease and is the same as at the beginning. The best explanation is certainly that the mercury is not pure, and that several products are dissolved inside. The voltage produces a chemical reaction and a solidification of these products. If this explanation is correct, the production of powder should decrease with the time. Unfortunately the amount of impurity is so big that I have been able to get a least three tea spoons of this product and the volume of mercury is still the same. If a comparison is made with the maximum concentration of sugar in water, this is not surprising.

Therefore, even if this mercury sensor would work a short time, the overall reliability is not an acceptable for this robot. It is surprising how many problems has a simple sensor like that.

## Multiplexers

Since the number of sensor outputs is more than thirty and the number of analog inputs of the 68HC11 is eight, the use of multiplexers is needed. I followed the "single chip analog multiplexer" handouts available on the IMDL web-site (Cf. Appendix 2).

One MC74HC574A (Octal 3-S-Inverter flip-flop) is taking care of the address latching from the port C. Four MC74HC4051 HSL 8-CH Analog Multiplexer/Demultiplexer take care of the signal from the sensor. One of the eight inputs of the multiplexer is selected by the 3-bit code in the pins 9,10,11. In order to output at the pin 3 the signal coming from the sensor, the multiplexer must be enabled by having a 0 at its pin 6. Since four multiplexers are used, and 3 bit are reserved to designate an input, the 8 outputs of the  flip are enough to take care of the multiplexing. I do not remember why I had some logic to the 74HC574 to select the right input of the right multiplexer, I realize now that it is useless. Finally I end up by using only 3 bit instead of four to designate a multiplexer. This coding uses one MC74HC04: a hexadecimal inverter and two MC74HC08 that are quadruple 2-input NAND gates. Five LEDs visualize the output of the latch. This way, it is easier to check which sensor is actually selected to be read

by the analog port. To read the analog value of a sensor, the latch must received the address corresponding to the desired sensor. The pin 11 of the latch is connected to the Y4 output on the ME11. Whenever the following command is typed in IC,

```
poke (0x6000, 0xXX);
analog(Z);
```

the analog port gives the value of the sensor at the address XX, if the corresponding multiplexer is connected to the analog port Z.

The actual program “ckeckup” that read every sensor is available in Appendix 3.

# Behaviors

## Scope

This robot was designed to have four behaviors: self-calibration, line following, wall following and random path. The last task has not been realized because the timing was too short and because that was the easiest behavior to improve.

Each behavior is select with switches. At the front of the robot is placed the “control panel of the robot” (Cf. Figure 17)

The first switch powered up the three parts of the circuit. The second switch enables the use of the sweep. The Dc motor controlling the sweep is turned on by the relay. The third switch controls the choice of the IR or the CDS cells during the calibration procedure. It is associated with the switch number 6. The sixth switch is a four-position switch. Depending on the position of the second switch it

controls the calibration of each CDS and IR. (See chapter self-calibration). The first press button is the reset button, it does the same thing as the reset button on the EVBU board. The second press button is used during the self-calibration procedure to start the calibration of a sensor. The fourth switch is the mode control switch. It has three position: power save on (only the memory is powered), download mode and run mode (mode A and mode B of the 68HC11). The fifth switch is the behavior select switch. From the left to the right it gives the choice between four different behaviors: Self-calibration, Line following Wall following and Random path.

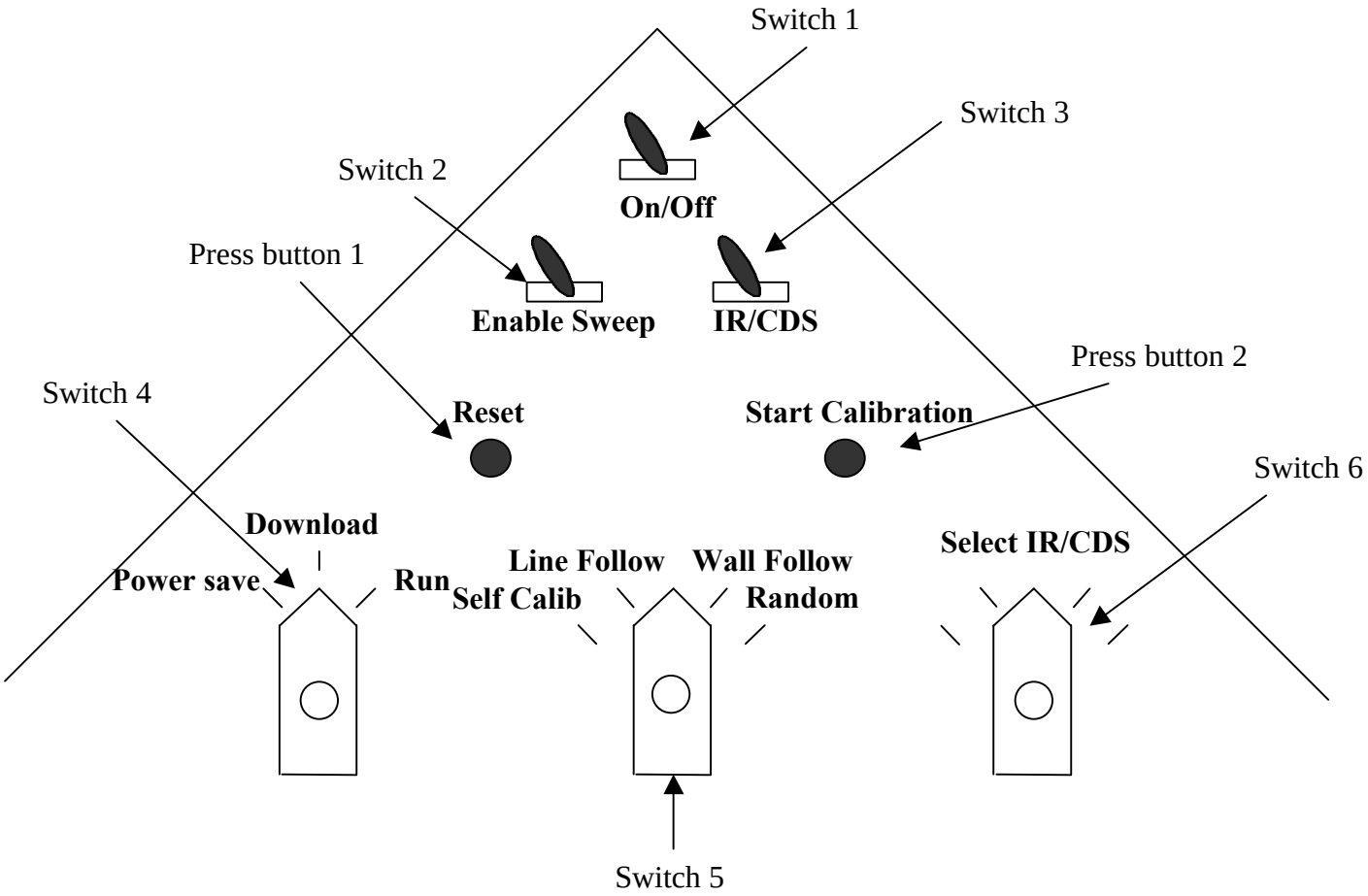


Figure 17: Control Panel

# Self-calibration

The calibration is very critical for efficient line and wall following. Each IR and CDS cell must be calibrated independently. This behavior should be run for each new environment of the robot. The following table 1 indicates which sensor is calibrated depending on the position of the switches 3 and 6.

| | Switch 6: Extreme left | Switch 6: Left | Switch 6: Right | Switch 6: Extreme right |
|---|---|---|---|---|
| Switch 3: IR Left position | Front IR: threshold<br>Right IRs: maximum | Front IR: maximum<br>Left IRs: maximum | Front IR: minimum<br>Left IRs: threshold<br>Right IRs: minimum | Left IRs: minimum<br>Right IRs: threshold |
| Switch 3: CDS Right position | White floor or white line | White floor or white line | Black floor or black line | Black floor or black line |

Table 1: Self-calibration, choice of sensor

To run the self-calibration, the program called “calib” (Cf Appendix 4 ) must be download. The switch 4 should be in the run position, the switch 5 in the “self-calibration” position. As soon as the reset button is pressed, the program starts.
The yellow light should flash. If the start calibration button is pressed, the yellow light stops flashing. It means that there are two seconds available to setup everything correctly (keep a wall straight for instance) and after this time, the calibration occurs. When the calibration of one sensor is done, the yellow light is flashing again. For a full calibration, this operation has to occur for the eight

positions of switches. If a calibration does not meet the requirement, just press the start calibration button and starts again. All the results are available whenever the behavior switch is going to the “ line following” position.

Initially the program “calib” was supposed to be a part of the main program. Unfortunately, I met some difficulties because the robot refused to execute any bigger program. Finally I wrote two programs: “calib” and another one which include both line and wall following.

## Line following

To follow a white line on the road the robot has to execute a line following function. The position of the CDS cell array may be surprising at the back of the robot. The reason is simple: no room was available anywhere else!

The program of wall following is included into the program called “demoday” available in appendix 5.

The theory of operation is simple (Cf. figure 18). The robot has four groups of two CDS cells. The values coming from each CDS cell from a couple are added to average the reading. The robot moves in order to keep the two central couple over the line and the two extreme couple out of the line. Each time this situation does not occur, the robot goes backward shortly, then turns to the most appropriate side and goes straight again.

In the listing of the “demoday” program, do not be afraid by the comments in the “printf” It is only some debugging sentence in French, useless for the program, but I did not have time to delete it. Since I had problems to follow a white line on

the road, I modified the program so that now the robot follows a black line on a white floor. Furthermore, I disabled the mercury sensor because of its unreliability.

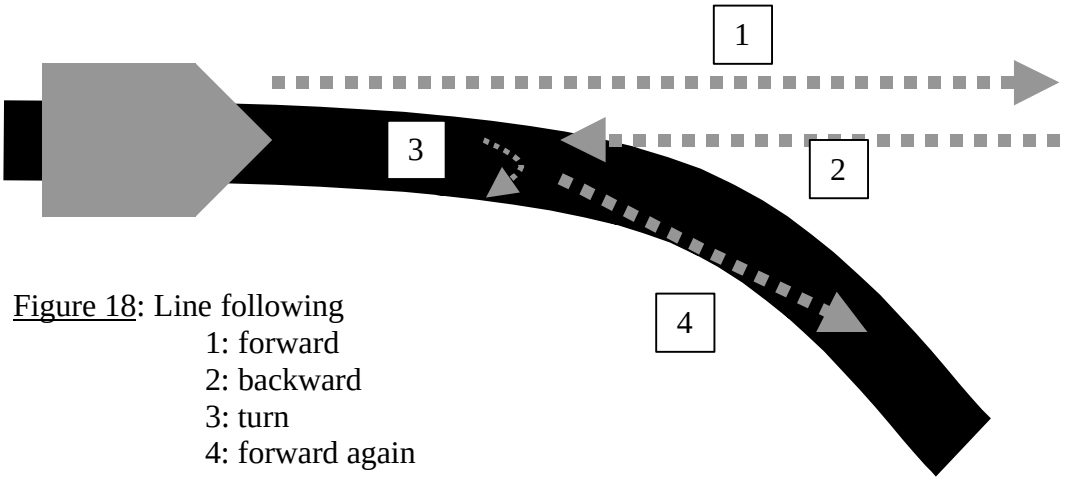


Figure 18: Line following
1: forward
2: backward
3: turn
4: forward again

To run the line following behavior, the first step is to download the “demoday” program and then turn the behavior switch (number 5) to its second position. Place the robot over a line and press reset: the line following starts.

# Wall following

In order to follow a kerb, the robot must be able to perform a wall –following task. A couple of IR sensor on each side of the robot indicates to the robot the distance to the next wall. This way it is possible to control the robot to keep this distance a constant as possible. Each IR has three level of security. Under a first threshold, the robot considers that the wall is too far. Between this first threshold and a second threshold, the robot does not correct its way. If the value from the IR is greater than the second threshold, the wall is considered as too close. The robot

tries then to correct its path to be a constant distance to the wall. If for any reason, the wall is “lost”, the robot keeps on turning to the direction of this wall. When the Robot feels an obstacle, from its bump switches, its front IR or the mercury sensor, it goes backward to avoid the object, then turn and tries to find the wall again. (Cf. Figure 19)

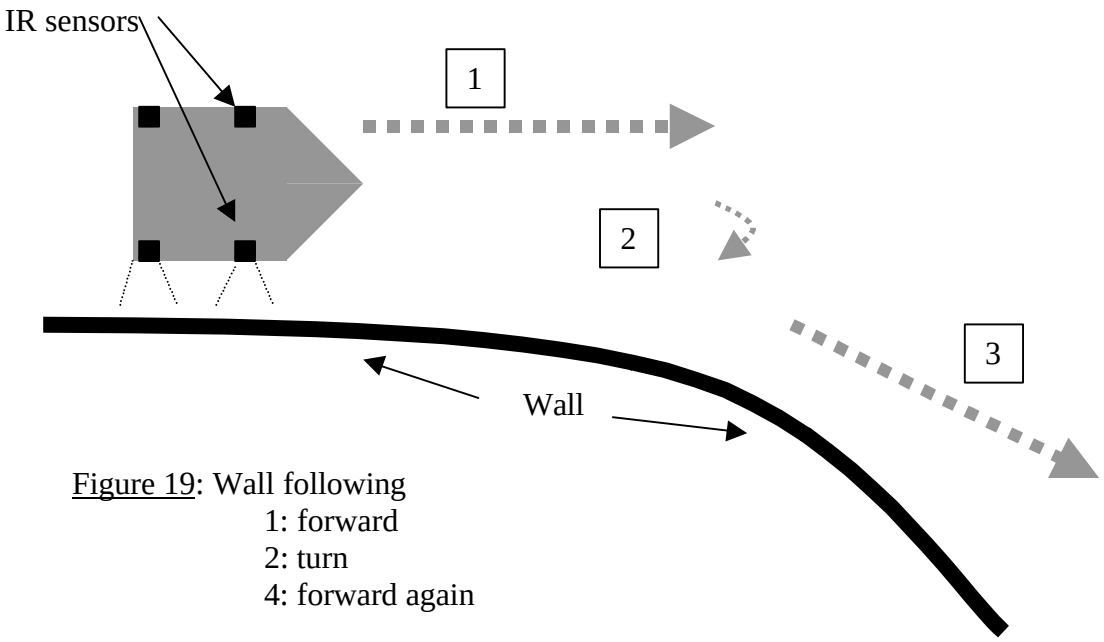


Figure 19: Wall following
1: forward
2: turn
4: forward again

To run the line following behavior, the first step is to download the “demoday” program and then turn the behavior switch (number 5) to its third position. Place the robot close to a line and press reset: the line following starts. Because the same program is used for the wall and the line following, it is not necessary to download “demoday” a second time after the line following.

## Random path

The program of this behavior has not been written, because I did not have enough time.

# Experimental Layout and Results

The actual testing of the robot provides few results. First of all, it is not possible the initial tasks which was to clean the roads. Here are many reasons for that:

-The autonomy with the battery is much too short (few minutes with the sweep turned on) to produce a significant work.

-The wall following is impossible because the kerb is too dark to be detected by the IR sensors.

-The line following is not efficient because the quality of the white lines of the roads is really poor. The limit between the white line and the road is not very neat when the line is old. Furthermore, there are a lot of black spots on the white line that would lead the robot to a misunderstanding.

-Finally, the overall reliability of the mercury sensor recommends not to use it. It would work for a short time but unfortunately after all the modifications introduced to this sensor and several purification, there is not very much mercury left. The stability checking is still possible but the level of mercury is now so low that even the movements of the robot can make the sensor lose the electric contact.

Despite those fact, the wall following and the line following are working pretty well indoor. The robot may lose the line if the turn radius is smaller than the turn radius of the robot. This problem does not consider the wall following because the robot turns until it finds the wall again. The DC motor used to spin the sweep is certainly too big for this application and drives a lot of current that heat the wires and even the board.

# Conclusion

Even if the initial goal of working outside is not reached, this robot is a success because it shows that a more elaborate robot with sensors adapted to the outdoor environment would work. It performs the main tasks efficiently. This robot also demonstrates that a high number of inputs or sensors is not a major obstacle.

# Appendix 1

Sharp IR Sensor Hack for Analog Distance Measurement
Mil Lab University of Florida

Location:

http://www.mil.ufl.edu/imdl/handouts/sharphack.pdf

# Appendix 2

Single chip  analog multiplexer
Mil Lab University of Florida

Location:

http://www.mil.ufl.edu/imdl/circuits/mux4351.pdf

# Appendix 3

Program “ checkup”

```
/*******************************************************/
Program checkup
Check every sensor of the robot
/*******************************************************/

#include <tjpbase.h>
#include <stdio.h>

#define CALIBRATION*(unsigned char *)(0x4000)
#define PROBLEM*(unsigned char *)(0x5000)
#define MUX *(unsigned char *)(0x6000)
#define INFRARED *(unsigned char *)(0x7000)

int Mode;
int Select;

int Selecthreshold1=65;
int Selecthreshold2=143;
int Selecthreshold3=176;
int Switchthreshold=128;
int Pressthreshold=245;

int BUMPbackleft;
int BUMPfrontleft;
int BUMPfrontright;
int BUMPbackright;
int BUMPback;

int IRbackleft;
int IRfrontleft;
int IRfrontright;
int IRbackright;
int IRfront;

int Modeselect;
int Sensorselect;
int Startcalib;
int IRorCDS;

int CDSambient;
int CDSextleftdark;
int CDSleftdark;
int CDSrightdark;
int CDSextrightdark;
int CDSextleftbright;
int CDSleftbright;
int CDSrightbright;
int CDSextrightbright;

int HGfront;
int HGfrontright;
int HGright;
int HGbackright;
int HGback;
int HGbackleft;
int HGleft;
int HGfrontleft;
```

```
int clignotant=1;

void waitloop(int);

/*****************************************/
void waitloop(nmsec)
{
      float counter;
      float factor;
      factor=1.8;
      counter=nmsec*factor;

      while(counter>0)
      {
            counter=counter-1;
      }
}
/*************************************************/

void main(void)
{
      init_analog();
      init_serial();
      init_motorme();
      init_clocktjp();

      INFRARED=(0xff);

      while(1)
      {
            IRfrontright=analog(1);
            IRfront=analog(2);

            MUX=(0x00);
            Modeselect=analog(3);
            MUX=(0x40);
            IRorCDS=analog(3);
            MUX=(0x80);
            Startcalib=analog(3);
            MUX=(0xC0);
            Sensorselect=analog(3);


            MUX=(0x04);
            HGfront=analog(4);
            MUX=(0x24);
            HGback=analog(4);
            MUX=(0x44);
            HGright=analog(4);
            MUX=(0x64);
            HGleft=analog(4);
            MUX=(0x84);
            HGfrontright=analog(4);
            MUX=(0xA4);
            HGbackleft=analog(4);
            MUX=(0xC4);
```

```
            HGbackright=analog(4);
            MUX=(0xE4);
            HGfrontleft=analog(4);

            MUX=(0x0C);
            BUMPfrontright=analog(5);
            MUX=(0x2C);
            BUMPbackright=analog(5);
            MUX=(0x4C);
            BUMPbackleft=analog(5);
            MUX=(0x6C);
            IRbackright=analog(5);
            MUX=(0x8C);
            BUMPfrontleft=analog(5);
            MUX=(0xAC);
            IRbackleft=analog(5);
            MUX=(0xCC);
            BUMPback=analog(5);
            MUX=(0xEC);
            IRfrontleft=analog(5);

            MUX=(0x14);
            CDSextrightbright=analog(6);
            MUX=(0x34);
            CDSleftbright=analog(6);
            MUX=(0x54);
            CDSextrightdark=analog(6);
            MUX=(0x74);
            CDSextleftbright=analog(6);
            MUX=(0x94);
            CDSrightbright=analog(6);
            MUX=(0xB4);
            CDSleftdark=analog(6);
            MUX=(0xD0);
            CDSrightdark=analog(6);
            MUX=(0xF4);
            CDSextleftdark=analog(6);

            CDSambient=analog(7);

            printf("Startcalibration{%d}  IRorCDS
      {%d}\n",Startcalib,IRorCDS);
            printf("Modeselect      {%d}  Sensorselect      {%d}
      Ambient-light     {%d}\n",Modeselect,Sensorselect,CDSambient);
            printf("HG-front  {%d}  HG-frontright     {%d}  HG-right
      {%d}  HG-backright
      {%d}\n",HGfront,HGfrontright,HGright,HGbackright);
            printf("HG-back         {%d}  HG-backleft {%d}  HG-left
      {%d}  HG-frontleft
      {%d}\n",HGback,HGbackleft,HGleft,HGfrontleft);
            printf("CDSextleftdk    {%d}  CDSleftdk   {%d}
      CDSextrightdk     {%d}  CDSrightdk
      {%d}\n",CDSextleftdark,CDSleftdark,CDSextrightdark,CDSrightdark);
            printf("CDSextleftbt    {%d}  CDSleftbt   {%d}
      CDSextrightbt     {%d}  CDSrightbt
      {%d)\n",CDSextleftbright,CDSleftbright,CDSextrightbright,CDSright
bright);
```

```
            printf("BUMPbackleft    {%d}  BUMPfrontleft     {%d}
      BUMPfrontright    {%d}  BUMPbackright     {%d}  BUMPback
      {%d}\n",BUMPbackleft,BUMPfrontleft,BUMPfrontright,BUMPbackright,B
UMPback);
            printf("IRbackleft      {%d)  IRfrontleft {%d}
      IRfrontright      {%d}  IRbackright {%d)  IRfront
      {%d)\n",IRbackleft,IRfrontleft,IRfrontright,IRbackright,IRfront);

            if(clignotant>0)
            {
                  CALIBRATION=(0x00);
                  PROBLEM=(0xff);
            }
            if(clignotant<0)
            {
                  CALIBRATION=(0xff);
                  PROBLEM=(0x00);
            }
            clignotant*=-1;

            if(Startcalib>Pressthreshold)
            {
                  printf("Start now!!!\n");
            }
            if(Modeselect>Selecthreshold1 &&
Modeselect<=Selecthreshold2)
            {
                  Mode=0;
                  if(IRorCDS<Switchthreshold &&
Sensorselect>Selecthreshold1 && Sensorselect<=Selecthreshold2)
                  {
                        Select=1;
                        printf("Calibration of left IRs\n\n\n");
                  }
                  if(IRorCDS<Switchthreshold &&
Sensorselect>Selecthreshold2 && Sensorselect<=Selecthreshold3)
                  {
                        Select=2;
                        printf("Calibration of front IR\n\n\n");
                  }
                  if(IRorCDS<Switchthreshold &&
Sensorselect>Selecthreshold3)
                  {
                        Select=3;
                        printf("Calibration of right IRs\n\n\n");
                  }
                  if(IRorCDS<Switchthreshold &&
Sensorselect<=Selecthreshold1)
                  {
                        Select=4;
                        printf("Calibration of back bump \n\n\n");
                  }
                  if(IRorCDS>=Switchthreshold &&
Sensorselect>Selecthreshold1 && Sensorselect<=Selecthreshold2)
                  {
                        Select=5;
```

```
                        printf("Calibration of CDS cell on a white line
in the dark\n\n\n");
                  }
                  if(IRorCDS>=Switchthreshold &&
Sensorselect>Selecthreshold2 && Sensorselect<=Selecthreshold3)
                  {
                        Select=6;
                        printf("Calibration of CDS cell on a white line
in the sun\n\n\n");
                  }
                  if(IRorCDS>=Switchthreshold &&
Sensorselect>Selecthreshold3)
                  {
                        Select=7;
                        printf("Calibration of CDS cell on a black road
in the dark\n\n\n");
                  }
                  if(IRorCDS>=Switchthreshold &&
Sensorselect<=Selecthreshold1)
                  {
                        Select=8;

                        printf("Calibration of CDS cell on a black road
in the sun\n\n\n");
                  }
            }
            if(Modeselect>Selecthreshold2 &&
Modeselect<=Selecthreshold3)
            {
                  Mode=1;
                  printf("Line following\n\n\n");
            }
            if(Modeselect>Selecthreshold3)
            {
                  Mode=2;
                  printf("Wall following\n\n\n");
            }
            if(Modeselect<Selecthreshold1)
            {
                  Mode=3;
                  printf("Random behavior\n\n\n");
            }
            waitloop(2000);
      }
}
```

# Appendix 4

Program “calib”

```
/************************************************/
Program calib
Self calibrate the robot
/************************************************/
#include <tjpbase.h>
#include <stdio.h>

#define CALIBRATION *(unsigned char *)(0x4000)
#define PROBLEM *(unsigned char *)(0x5000)
#define MUX *(unsigned char *)(0x6000)
#define INFRARED *(unsigned char *)(0x7000)

int Modeselect=0;
int Sensorselect=0;
int Startcalib=0;
int IRorCDS=0;

int Selecthreshold1=65;
int Selecthreshold2=143;
int Selecthreshold3=176;
int Switchthreshold=128;
int Pressthreshold=128;

int clignotant=1;
int dumm;

int BUMPbackleft;
int BUMPfrontleft;
int BUMPfrontright;
int BUMPbackright;
int BUMPback;
int BUMPbackthreshold=169;

int IRbackleft;
int IRfrontleft;
int IRfrontright;
int IRbackright;
int IRfront;
int IRbackleftthreshold;
int IRfrontleftthreshold;
int IRfrontrightthreshold;
int IRbackrightthreshold;
int IRfrontthreshold;
int IRbackleftlow;
int IRfrontleftlow;
int IRfrontrightlow;
int IRbackrightlow;
int IRfrontlow;
int IRbacklefthigh;
int IRfrontlefthigh;
int IRfrontrighthigh;
int IRbackrighthigh;
int IRfronthigh;

int CDSambientlow;
int CDSextleftdarklow;
int CDSleftdarklow;
```

```
int CDSrightdarklow;
int CDSextrightdarklow;
int CDSextleftbrightlow;
int CDSleftbrightlow;
int CDSrightbrightlow;
int CDSextrightbrightlow;
int CDSambienthigh;
int CDSextleftdarkhigh;
int CDSleftdarkhigh;
int CDSrightdarkhigh;
int CDSextrightdarkhigh;
int CDSextleftbrighthigh;
int CDSleftbrighthigh;
int CDSrightbrighthigh;
int CDSextrightbrighthigh;
int CDSambient;
int CDSextleftdark;
int CDSleftdark;
int CDSrightdark;
int CDSextrightdark;
int CDSextleftbright;
int CDSleftbright;
int CDSrightbright;
int CDSextrightbright;
int CDSambientthreshold;
int CDSextleftdarkthreshold;
int CDSleftdarkthreshold;
int CDSrightdarkthreshold;
int CDSextrightdarkthreshold;
int CDSextleftbrightthreshold;
int CDSleftbrightthreshold;
int CDSrightbrightthreshold;
int CDSextrightbrightthreshold;

int HGfront;
int HGfrontright;
int HGright;
int HGbackright;
int HGback;
int HGbackleft;
int HGleft;
int HGfrontleft;

int getBUMPbackleft(void);
int getBUMPfrontleft(void);
int getBUMPfrontright(void);
int getBUMPbackright(void);
int getBUMPback(void);

int getIRbackleft(void);
int getIRfrontleft(void);
int getIRfrontright(void);
int getIRbackright(void);
int getIRfront(void);

int getModeselect(void);
int getSensorselect(void);
```

```
int getStartcalib(void);
int getIRorCDS(void);

int getCDSambient(void);
int getCDSextleftdark(void);
int getCDSleftdark(void);
int getCDSrightdark(void);
int getCDSextrightdark(void);
int getCDSextleftbright(void);
int getCDSleftbright(void);
int getCDSrightbright(void);
int getCDSextrightbright(void);

int getHGfront(void);
int getHGfrontright(void);
int getHGright(void);
int getHGbackright(void);
int getHGback(void);
int getHGbackleft(void);
int getHGleft(void);
int getHGfrontleft(void);

void waitloop(int);
void waitcalibstart(void);
void waitcalib(void);
int meantool(int (*data)(void));
void calibration(void);
void line_following(void);
void wall_following(void);
void random_behavior(void);
void IRcalibration1();
void IRcalibration2();
void IRcalibration3();
void IRcalibration4();
void CDScalibration1();
void CDScalibration2();
void CDScalibration3();
void CDScalibration4();

/*****************************************/




/*****************************************/
void main(void)
{
      init_analog();
      init_serial();
      init_motorme();
      init_clocktjp();

      PROBLEM=0x00;

      while(1)
      {
            init_analog();
```

```
            Modeselect=getModeselect();
            printf("Modeselect1 {%d}\n\n",Modeselect);
            while(Modeselect>Selecthreshold1 &&
Modeselect<=Selecthreshold2)
            {
                  printf("Calibration\n\n\n");
                  calibration();
                  Modeselect=getModeselect();
            }
            while(Modeselect>Selecthreshold2 &&
Modeselect<=Selecthreshold3)
            {
                  printf("Line following\n\n\n");
                  line_following();
                  Modeselect=getModeselect();
            }
            while(Modeselect>Selecthreshold3)
            {
                  printf("Wall following\n\n\n");
                  wall_following();
                  Modeselect=getModeselect();
            }
            while(Modeselect<=Selecthreshold1 && Modeselect>=0)
            {
                  printf("Random behavior\n\n\n");
                  random_behavior();
                  Modeselect=getModeselect();
            }
            while(Modeselect<0)
            {
                  Modeselect=getModeselect();
                  printf("Modeselect problem {%d}\n\n",Modeselect);

            }
      }
}
/*****************************************/
void waitloop(nmsec)
{
      float counter;
      float factor;
      factor=1.8;
      counter=nmsec * factor;

      while(counter>0)
      {
            counter=counter-1;
      }
}
/*****************************************/
void calibration()
{
      IRorCDS=getIRorCDS();
      Sensorselect=getSensorselect();

/*IRcalibration1*/
```

```
      while(IRorCDS<Switchthreshold && Sensorselect>Selecthreshold1 &&
Sensorselect<=Selecthreshold2 && Modeselect>Selecthreshold1 &&
Modeselect<=Selecthreshold2)
      {
            printf("Calibration of IRs: front->threshold right-
>max\n");
            Startcalib=0;
            while(Startcalib<= Pressthreshold &&
IRorCDS<Switchthreshold && Sensorselect>Selecthreshold1 &&
Sensorselect<=Selecthreshold2 && Modeselect>Selecthreshold1 &&
Modeselect<=Selecthreshold2)
            {
                  Startcalib=getStartcalib();
                  if( Startcalib > Pressthreshold)
                  {
                        INFRARED=(0xff);
                        waitcalib();
                        IRcalibration1();
                        CALIBRATION=(0x00);
                        INFRARED=(0x00);
                        Startcalib=0;
                  }
                  else
                  {
                        waitcalibstart();
                        IRorCDS=getIRorCDS();
                        Sensorselect=getSensorselect();
                        Modeselect=getModeselect();
                  }
            }
      }

/*IRcalibration2*/
      while(IRorCDS<Switchthreshold && Sensorselect>Selecthreshold2 &&
Sensorselect<=Selecthreshold3 && Modeselect>Selecthreshold1 &&
Modeselect<=Selecthreshold2)
      {
            printf("Calibration of IRs: front->max left->max\n");
            Startcalib=0;
            while(Startcalib<=Pressthreshold && IRorCDS<Switchthreshold
&& Sensorselect>Selecthreshold2 && Sensorselect<=Selecthreshold3 &&
Modeselect>Selecthreshold1 && Modeselect<=Selecthreshold2)
            {
                  Startcalib=getStartcalib();
                  if(Startcalib>Pressthreshold)
                  {
                        INFRARED=(0xff);
                        waitcalib();
                        IRcalibration2();
                        CALIBRATION=(0x00);
                        INFRARED=(0x00);
                        Startcalib=0;
                  }
                  else
                  {
                        waitcalibstart();
                        IRorCDS=getIRorCDS();
```

```
                        Sensorselect=getSensorselect();
                        Modeselect=getModeselect();
                  }
            }
      }

/*IRcalibration3*/
      while(IRorCDS<Switchthreshold && Sensorselect>Selecthreshold3 &&
Modeselect>Selecthreshold1 && Modeselect<=Selecthreshold2)
      {
            printf("Calibration of IRs: front->min left->threshold
right->min\n");
            Startcalib=0;
            while(Startcalib<=Pressthreshold && IRorCDS<Switchthreshold
&& Sensorselect>Selecthreshold3 && Modeselect>Selecthreshold1 &&
Modeselect<=Selecthreshold2)
            {
                  Startcalib=getStartcalib();
                  if(Startcalib>Pressthreshold)
                  {
                        INFRARED=(0xff);
                        waitcalib();

      IRcalibration3(&IRfrontlow,&IRfrontrightlow,&IRbackrightlow);
                        CALIBRATION=(0x00);
                        INFRARED=(0x00);
                        Startcalib=0;
                  }
                  else
                  {
                        waitcalibstart();
                        IRorCDS=getIRorCDS();
                        Sensorselect=getSensorselect();
                        Modeselect=getModeselect();
                  }
            }
      }

/*IRcalibration4*/
      while(IRorCDS<Switchthreshold && Sensorselect<=Selecthreshold1 &&
Modeselect>Selecthreshold1 && Modeselect<=Selecthreshold2)
      {
            printf("Calibration of IRs: left->min right->threshold\n");
            Startcalib=0;
            while(Startcalib<=Pressthreshold && IRorCDS<Switchthreshold
&& Sensorselect<=Selecthreshold1 && Modeselect>Selecthreshold1 &&
Modeselect<=Selecthreshold2)
            {
                  Startcalib=getStartcalib();
                  if(Startcalib>Pressthreshold)
                  {
                        INFRARED=(0xff);
                        waitcalib();
                        IRcalibration4();
                        Startcalib=0;
                        CALIBRATION=(0x00);
                        INFRARED=(0x00);
```

```
                }
                else
                {
                        waitcalibstart();
                        IRorCDS=getIRorCDS();
                        Sensorselect=getSensorselect();
                        Modeselect=getModeselect();
                }
            }
      }
/*CDScalibration1*/
      while(IRorCDS>=Switchthreshold && Sensorselect>Selecthreshold1 &&
Sensorselect<=Selecthreshold2 && Modeselect>Selecthreshold1 &&
Modeselect<=Selecthreshold2)
      {
            printf("Calibration of CDS cell on a white line /
floor\n");
            Startcalib=0;
            while(Startcalib<=Pressthreshold &&
IRorCDS>=Switchthreshold && Sensorselect>Selecthreshold1 &&
Sensorselect<=Selecthreshold2 && Modeselect>Selecthreshold1 &&
Modeselect<=Selecthreshold2)
            {
                  Startcalib=getStartcalib();
                  if(Startcalib>Pressthreshold)
                  {
                        waitcalib();
                        CDScalibration1();
                        Startcalib=0;
                        CALIBRATION=(0x00);
                  }
                  else
                  {
                        waitcalibstart();
                        IRorCDS=getIRorCDS();
                        Sensorselect=getSensorselect();
                        Modeselect=getModeselect();
                  }
            }
      }

/*CDScalibration2*/
      while(IRorCDS>=Switchthreshold && Sensorselect>Selecthreshold2 &&
Sensorselect<=Selecthreshold3 && Modeselect>Selecthreshold1 &&
Modeselect<=Selecthreshold2)
      {
            printf("Calibration of CDS cell on a white line /floor\n");
            Startcalib=0;
            while(Startcalib<=Pressthreshold &&
IRorCDS>=Switchthreshold && Sensorselect>Selecthreshold2 &&
Sensorselect<=Selecthreshold3 && Modeselect>Selecthreshold1 &&
Modeselect<=Selecthreshold2)
            {
                  Startcalib=getStartcalib();
                  if(Startcalib>Pressthreshold)
                  {
                        waitcalib();
```

```
                        CDScalibration2();
                        Startcalib=0;
                        CALIBRATION=(0x00);
                  }
                  else
                  {
                        waitcalibstart();
                        IRorCDS=getIRorCDS();
                        Sensorselect=getSensorselect();
                        Modeselect=getModeselect();
                  }
            }
      }

/*CDScalibration3*/
      while(IRorCDS>=Switchthreshold && Sensorselect>Selecthreshold3 &&
Modeselect>Selecthreshold1 && Modeselect<=Selecthreshold2)
      {
            printf("Calibration of CDS cell on a black line/floor\n");

            Startcalib=0;
            while(Startcalib<=Pressthreshold &&
IRorCDS>=Switchthreshold && Sensorselect>Selecthreshold3 &&
Modeselect>Selecthreshold1 && Modeselect<=Selecthreshold2)
            {
                  Startcalib=getStartcalib();
                  if(Startcalib>Pressthreshold)
                  {
                        waitcalib();
                        CDScalibration3();
                        Startcalib=0;
                        CALIBRATION=(0x00);
                  }
                  else
                  {
                        waitcalibstart();
                        IRorCDS=getIRorCDS();
                        Sensorselect=getSensorselect();
                        Modeselect=getModeselect();
                  }
            }
      }

/*CDScalibration4*/
      while(IRorCDS>=Switchthreshold && Sensorselect<=Selecthreshold1
&& Modeselect>Selecthreshold1 && Modeselect<=Selecthreshold2)
      {
            printf("Calibration of CDS cell on a black line/floor\n");
            Startcalib=0;
            while(Startcalib<=Pressthreshold &&
IRorCDS>=Switchthreshold && Sensorselect<=Selecthreshold1 &&
Modeselect>Selecthreshold1 && Modeselect<=Selecthreshold2)
            {
                  Startcalib=getStartcalib();
                  if(Startcalib>Pressthreshold)
                  {
                        waitcalib();
```

```
                        CDScalibration4();
                        Startcalib=0;
                        CALIBRATION=(0x00);
                  }
                  else
                  {
                        waitcalibstart();
                        IRorCDS=getIRorCDS();
                        Sensorselect=getSensorselect();
                        Modeselect=getModeselect();
                  }
            }
      }
/*Processing*/
      CDSambientthreshold=(CDSambientlow+CDSambienthigh)/ 2;
      CDSextleftdarkthreshold=(CDSextleftdarkhigh+CDSextleftdarklow)/
2;
      CDSleftdarkthreshold=(CDSleftdarkhigh+CDSleftdarklow)/ 2;
      CDSrightdarkthreshold=(CDSrightdarkhigh+CDSrightdarklow)/ 2;
      CDSextrightdarkthreshold=(CDSextrightdarkhigh+CDSextrightdarklow)
/ 2;
      CDSextleftbrightthreshold=(CDSextleftbrighthigh+CDSextleftbrightl
ow)/ 2;
      CDSleftbrightthreshold=(CDSleftbrighthigh+CDSleftbrightlow)/ 2;
      CDSrightbrightthreshold=(CDSrightbrighthigh+CDSrightbrightlow)/
2;
      CDSextrightbrightthreshold=(CDSextrightbrighthigh+CDSextrightbrig
htlow)/ 2;
}
/*******************************************/
int getModeselect()
{
      MUX=(0x00);
      return analog(3);
}
/*******************************************/
int getSensorselect()
{
      MUX=(0xC0);
      return analog(3);
}
/*******************************************/
int getIRorCDS()
{
      MUX=(0x40);
      return analog(3);
}
/*******************************************/
int getStartcalib()
{
      MUX=(0x80);
      return analog(3);
}
/*******************************************/
void waitcalibstart()
{
      if(clignotant>0)
```

```
	{
		CALIBRATION=(0x00);
	}
	if(clignotant<0)
	{
		CALIBRATION=(0xff);
	}
	clignotant*=-1;
	waitloop(100);
}
/*****************************************/
void waitcalib()
{
	CALIBRATION=(0xff);
	waitloop(1000);
	CALIBRATION=(0x00);
	waitloop(1000);
	CALIBRATION=(0xff);
}
/*****************************************/
int meantool(int (*data)(void))
{
	int mean = 0 ;
	int i = 0 ;
	for(i = 0 ; i < 10 ; i ++)
	{
		mean = mean + (*data)();
		waitloop(100);
	}
	mean=mean / 10;
	return mean;
}
/*****************************************/
void IRcalibration1()
{
	IRfrontthreshold=meantool(getIRfront);
	IRfrontrighthigh=meantool(getIRfrontright);
	IRbackrighthigh=meantool(getIRbackright);
}
/*****************************************/
void IRcalibration2()
{
	IRfronthigh=meantool(getIRfront);
	IRfrontlefthigh=meantool(getIRfrontleft);
	IRbacklefthigh=meantool(getIRbackleft);
}
/*****************************************/
void IRcalibration3()
{
	IRfrontlow=meantool(getIRfront);
	IRfrontleftthreshold=meantool(getIRfrontleft);
	IRbackleftthreshold=meantool(getIRbackleft);
	IRfrontrightlow=meantool(getIRfrontright);
	IRbackrightlow=meantool(getIRbackright);
}
/*****************************************/
void IRcalibration4()
```

```
{
      IRfrontleftlow=meantool(getIRfrontleft);
      IRbackleftlow=meantool(getIRbackleft);
      IRfrontrightthreshold=meantool(getIRfrontright);
      IRbackrightthreshold=meantool(getIRbackright);
}
/*****************************************/
int getIRbackright()
{
      MUX=(0x6C);
      return analog(5);
}
/*****************************************/
int getIRfrontright()
{
      return analog(1);
}
/*****************************************/
int getIRfront()
{
      return analog(2);
}
/*****************************************/
int getIRfrontleft()
{
      MUX=(0xEC);
      return analog(5);
}
/*****************************************/
int getIRbackleft()
{
      MUX=(0xAC);
      return analog(5);
}
/*****************************************/
void CDScalibration1()
{
      CDSextleftdarklow=meantool(getCDSextleftdark);
      CDSleftdarklow=meantool(getCDSleftdark);
      CDSrightdarklow=meantool(getCDSrightdark);
      CDSextrightdarklow=meantool(getCDSextrightdark);
}
/*****************************************/
void CDScalibration2()
{
      CDSextleftbrightlow=meantool(getCDSextleftbright);
      CDSleftbrightlow=meantool(getCDSleftbright);
      CDSrightbrightlow=meantool(getCDSrightbright);
      CDSextrightbrightlow=meantool(getCDSextrightbright);
      CDSambientlow=meantool(getCDSambient);
}
/*****************************************/
void CDScalibration3()
{
      CDSextleftdarkhigh=meantool(getCDSextleftdark);
      CDSleftdarkhigh=meantool(getCDSleftdark);
      CDSrightdarkhigh=meantool(getCDSrightdark);
```

```
      CDSextrightdarkhigh=meantool(getCDSextrightdark);
      CDSambienthigh=meantool(getCDSambient);
}
/*****************************************/
void CDScalibration4()
{
      CDSextleftbrighthigh=meantool(getCDSextleftbright);
      CDSleftbrighthigh=meantool(getCDSleftbright);
      CDSrightbrighthigh=meantool(getCDSrightbright);
      CDSextrightbrighthigh=meantool(getCDSextrightbright);
}
/*****************************************/

int getCDSextrightbright()
{
      MUX=(0x14);
      return analog(6);
}
/*****************************************/
int getCDSleftbright()
{
      MUX=(0x34);
      return analog(6);
}
/*****************************************/
int getCDSextrightdark()
{
      MUX=(0x54);
      return analog(6);
}
/*****************************************/
int getCDSextleftbright()
{
      MUX=(0x74);
      return analog(6);
}
/*****************************************/
int getCDSrightbright()
{
      MUX=(0x94);
      return analog(6);
}
/*****************************************/
int getCDSleftdark()
{
      MUX=(0xB4);
      return analog(6);
}
/*****************************************/
int getCDSrightdark()
{
      MUX=(0xD0);
      return analog(6);
}
/*****************************************/
int getCDSextleftdark()
{
```

```
      MUX=(0xF4);
      return analog(6);
}
/*****************************************/
int getCDSambient()
{
      return analog(7);
}
/*****************************************/
void line_following()
{
      printf("IRbackleftlow   {%d}  IRbacklefttthreshold     {%d}
      IRbacklefthigh
      {%d}\n",IRbackleftlow,IRbacklefttthreshold,IRbacklefthigh);
      printf("IRfrontleftlow  {%d}  IRfrontleftthreshold    {%d}
      IRfrontlefthigh
      {%d}\n",IRfrontleftlow,IRfrontleftthreshold,IRfrontlefthigh);

      printf("IRfrontlow      {%d}  IRfrontthreshold  {%d}  IRfronthigh
      {%d}\n",IRfrontlow,IRfrontthreshold,IRfronthigh);
      printf("IRfrontrightlow {%d}  IRfrontrightthreshold   {%d}
      IRfrontrighthigh
      {%d}\n",IRfrontrightlow,IRfrontrightthreshold,IRfrontrighthigh);

      printf("IRbackrightlow  {%d}  IRbackrightthreshold    {%d}
      IRbackrighthigh
      {%d}\n",IRbackrightlow,IRbackrightthreshold,IRbackrighthigh);

      printf("CDSambientlow         {%d}  CDSambientthreshold
      {%d}  CDSambienthigh
      {%d}\n",CDSambientlow,CDSambientthreshold,CDSambienthigh);
      printf("CDSextleftdarklow     {%d}  CDSextleftdarkthreshold
      {%d}  CDSextleftdarkhigh
      {%d}\n",CDSextleftdarklow,CDSextleftdarkthreshold,CDSextleftdarkh
igh);
      printf("CDSleftdarklow        {%d}  CDSleftdarkthreshold
      {%d}  CDSleftdarkhigh
      {%d}\n",CDSleftdarklow,CDSleftdarkthreshold,CDSleftdarkhigh);
      printf("CDSrightdarklow       {%d}  CDSrightdarkthreshold
      {%d}  CDSrightdarkhigh
      {%d}\n",CDSrightdarklow,CDSrightdarkthreshold,CDSrightdarkhigh);
      printf("CDSextrightdarklow    {%d}  CDSextrightdarkthreshold
      {%d}  CDSextrightdarkhigh
      {%d}\n",CDSextrightdarklow,CDSextrightdarkthreshold,CDSextrightda
rkhigh);
      printf("CDSextleftbrightlow   {%d}  CDSextleftbrightthreshold
      {%d}  CDSextleftbrighthigh
      {%d}\n",CDSextleftbrightlow,CDSextleftbrightthreshold,CDSextleftb
righthigh);
      printf("CDSleftbrightlow      {%d}  CDSleftbrightthreshold
      {%d}  CDSleftbrighthigh
      {%d}\n",CDSleftbrightlow,CDSleftbrightthreshold,CDSleftbrighthigh
);
      printf("CDSrightbrightlow     {%d}  CDSrightbrightthreshold
      {%d}  CDSrightbrighthigh
      {%d}\n",CDSrightbrightlow,CDSrightbrightthreshold,CDSrightbrighth
igh);
```

```
      printf("CDSextrightbrightlow  {%d}  CDSextrightbrightthreshold
      {%d}  CDSextrightbrighthigh
      {%d}\n\n\n",CDSextrightbrightlow,CDSextrightbrightthreshold,CDSex
trightbrighthigh);
      waitloop(1000);
}
/*****************************************/
void wall_following(void)
{
      waitloop(100);
}
/*****************************************/
void random_behavior(void)
{
      waitloop(100);
      printf("Vient il randomer le petit?\n\n");
}
/*****************************************/
int getHGfront()
{
      MUX=(0x04);
      return analog(4);
}
/*****************************************/
int getHGback()
{
      MUX=(0x24);
      return analog(4);
}
/*****************************************/
int getHGright()
{
      MUX=(0x44);
      return analog(4);
}
/*****************************************/
int getHGleft()
{
      MUX=(0x64);
      return analog(4);
}
/*****************************************/
int getHGfrontright()
{
      MUX=(0x84);
      return analog(4);
}
/*****************************************/
int getHGbackleft()
{
      MUX=(0xA0);
      return analog(4);
}
/*****************************************/
int getHGbackright()
{
      MUX=(0xC4);
```

```
      return analog(4);
}
/*******************************************/
int getHGfrontleft()
{
      MUX=(0xE4);
      return analog(4);
}
/*******************************************/
int getBUMPbackleft()
{
      MUX=(0x4C);
      return analog(5);
}
/*******************************************/
int getBUMPfrontright()
{
      MUX=(0x0C);
      return analog(5);
}
/*******************************************/
int getBUMPbackright()
{
      MUX=(0x2C);
      return analog(5);
}
/*******************************************/

int getBUMPfrontleft()
{
      MUX=(0x8C);
      return analog(5);
}
/*******************************************/

int getBUMPback()
{
      MUX=(0xCC);
      return analog(5);
}
/*******************************************/
```

# Appendix 5

Program “demoday

```
/*****************************************************************/
Demoday program
      No HG sensor
      black line

/*****************************************************************/
#include <tjpbase.h>
#include <stdio.h>

#define CALIBRATION *(unsigned char *)(0x4000)
#define PROBLEM *(unsigned char *)(0x5000)
#define MUX *(unsigned char *)(0x6000)
#define INFRARED *(unsigned char *)(0x7000)

int Modeselect=0;
int Sensorselect=0;
int Startcalib=0;
int IRorCDS=0;
int time;
int side;
int increment;

int Selecthreshold1=65;
int Selecthreshold2=143;
int Selecthreshold3=176;
int Switchthreshold=128;
int Pressthreshold=128;

int problem_status=0x00;
int calib_status=0xff;
int clignotant=1;
int dumm;
int previouside=1;
int previous_line=1;
int nothing;
int wall;
int previouscorrection;

int BUMPbackleft;
int BUMPfrontleft;
int BUMPfrontright;
int BUMPbackright;
int BUMPback;
int BUMPbackthreshold=169;

int IRbackleft;
int IRfrontleft;
int IRfrontright;
int IRbackright;
int IRfront;
int IRbackleftthreshold=116;
int IRfrontleftthreshold=120;
int IRfrontrightthreshold=122;
int IRbackrightthreshold=116;
int IRfrontthreshold=105;

int CDSextleftdark;
```

```
int CDSleftdark;
int CDSrightdark;
int CDSextrightdark;
int CDSextleftbright;
int CDSleftbright;
int CDSrightbright;
int CDSextrightbright;
int CDSextleftthreshold=219;
int CDSleftthreshold=355;
int CDSrightthreshold=320;
int CDSextrightthreshold=308;

int HGfront;
int HGfrontright;
int HGright;
int HGbackright;
int HGback;
int HGbackleft;
int HGleft;
int HGfrontleft;

int getBUMPbackleft(void);
int getBUMPfrontleft(void);
int getBUMPfrontright(void);
int getBUMPbackright(void);
int getBUMPback(void);

int getIRbackleft(void);
int getIRfrontleft(void);
int getIRfrontright(void);
int getIRbackright(void);
int getIRfront(void);

int getSensorselect(void);
int getStartcalib(void);
int getIRorCDS(void);

int getCDSambient(void);
int getCDSextleftdark(void);
int getCDSleftdark(void);
int getCDSrightdark(void);
int getCDSextrightdark(void);
int getCDSextleftbright(void);
int getCDSleftbright(void);
int getCDSrightbright(void);
int getCDSextrightbright(void);

int getHGfront(void);
int getHGfrontright(void);
int getHGright(void);
int getHGbackright(void);
int getHGback(void);
int getHGbackleft(void);
int getHGleft(void);
int getHGfrontleft(void);

void waitloop(int);
```

```
void line_following(void);
void wall_following(void);

int checkbump(void);
int checkHG(void);
int checkIR(void);
int security(void);
int line_process(void);
void my_motor(int motor, int newspeed);
int checkCDS(void);
int line_follow();
void stop();
int getwall(void);
void getline();
void wallbypass();
/*****************************************/
void main(void)
{
      INFRARED=0xff;
      waitloop(100);
      init_analog();
      init_serial();
      init_motorme();
      init_clocktjp();

      PROBLEM=0x00;

      while(1)
      {
            init_analog();
            MUX=(0x00);
            Modeselect=analog(3);
            while(Modeselect>Selecthreshold1 &&
Modeselect<=Selecthreshold2)
            {
                  printf("Calibration not available\n\n\n");
                  MUX=(0x00);
                  Modeselect=analog(3);
            }
            while(Modeselect>Selecthreshold2 &&
Modeselect<=Selecthreshold3)
            {
                  printf("Line following\n\n\n");
                  line_following();
                  MUX=(0x00);
                  Modeselect=analog(3);
            }
            while(Modeselect>Selecthreshold3)
            {
                  printf("Wall following\n\n\n");
                  wall_following();
                  MUX=(0x00);
                  Modeselect=analog(3);
            }
            while(Modeselect<=Selecthreshold1 && Modeselect>=0)
            {
                  printf("Random behavior not available \n\n\n");
```

```
                  MUX=(0x00);
                  Modeselect=analog(3);
            }
            while(Modeselect<0)
            {
                  MUX=(0x00);
                  Modeselect=analog(3);
                  printf("Modeselect problem {%d}\n\n",Modeselect);
            }
      }
}
/*****************************************/
void waitloop(int nmsec)
{
      float counter;
      float factor;
      factor=1.8;
      counter=nmsec * factor;

      while(counter>0)
      {
            counter=counter-1;
      }
}
/*****************************************/
void line_following(void)
{
      int direction=0xff;
      int correction;


      while(direction ==0xff)
      {
            increment=1;
            direction=line_follow();
      }
/*    if(direction != 0xff)
      {
            printf("ouups!  ya un cailloux!\n");
            if(direction==0xfe)
            {
                  side=previouside;
                  bypass();
            }
            if(direction==0xfd || direction==0xfb)
            {
                  side=1;
                  bypass();
            }
            if(direction==0x7f || direction==0xbf)
            {
                  side=-1;
                  bypass();
            }
            if(direction==0xf7 || direction==0xef || direction==0xdf)
            {
```

```
                my_motor(1 , 0);
                my_motor(0 , 0);
                stop();
            }
      }
*/
}

/************************************/
int line_follow()
{
      int correction;
      int maxspeed;
      int index=10;
      int CDS;
      int direction;
      int avoid;

      maxspeed=40 * increment;
      correction=line_process();
      printf ("correction %d \n",correction);

      if(correction==0)
      {
            my_motor(1 , maxspeed);
            my_motor(0 , maxspeed);
      }
      if(correction==10)
      {
            CDS=0x0;
            while((CDS & 0x3)==0x0)
            {
                  CDS=checkCDS();
                  my_motor(1 , -1 * maxspeed);
                  my_motor(0 , -1 * maxspeed);
            }
      }
      if(correction!=0)
      {
            my_motor(1 , 0);
            my_motor(0 , 0);
            waitloop(300);

            while(CDS!=0x6 && index > 0)
            {
                  index =index - 1 ;
                  my_motor(1 , -1 * maxspeed);
                  my_motor(0 , -1 * maxspeed);
                  CDS=checkCDS();
                  printf("en arriere toute! phase 1   CDS ={%x}   index
{%d}\n",CDS,index);
            }
            index=30;
            while((index > 0) && (CDS==0x6))
            {
                  index = index - 1 ;
                  my_motor(1 , -1 * maxspeed);
```

```
                  my_motor(0 , -1 * maxspeed);
                  CDS=checkCDS();
                  printf("en arriere toute! phase 2   CDS ={%x}   index
{%d}\n",CDS,index);
            }
            if (correction > 0 && correction != 10)
            {
                  printf("ca tourne!\n");
                  my_motor(1 , -100);
                  my_motor(0 , 133);
                  waitloop(250);
                  if (correction ==2)
                        waitloop(200);
            }
            if (correction < 0)
            {
                  printf("ca tourne!\n");
                  my_motor(1 , 100);
                  my_motor(0 , -100);
                  waitloop(150);
                  if (correction ==-2)
                        waitloop(200);
            }
            CDS=checkCDS();
            index=25;
            while(CDS!=0x6 && index>0)
            {
                  index =index - 1;
                  my_motor(1 , maxspeed);
                  my_motor(0 , maxspeed);
                  waitloop(10);
                  CDS=checkCDS();
                  printf("en avant toute!             CDS ={%x}   index
{%d}\n",CDS,index);
            }
            if (index==0)
            {
                  if (CDS==0xc || CDS==0x3)
                  {
                        index=15;
                        while(index > 0)
                        {
                              index =index - 1;
                              my_motor(1 , -1 * maxspeed);
                              my_motor(0 , -1 * maxspeed);
                              waitloop(10);
                              printf("recule et attend la petite
correction  index {%d}\n",CDS,index);
                        }
                        CDS=checkCDS();

                        if (CDS==0xc)
                        {
                              printf("petite correction\n");
                              my_motor(1 , 100);
                              my_motor(0 , -100);
                              waitloop(100);
```

```
                        }
                        if(CDS==0x3)
                        {
                              printf("petite correction\n");
                              my_motor(1 , -100);
                              my_motor(0 , 133);
                              waitloop(200);
                        }
                        index=25;
                        while(CDS!=0x6 && index>0)
                        {
                              index =index - 1;
                              my_motor(1 , maxspeed);
                              my_motor(0 , maxspeed);
                              waitloop(10);
                              CDS=checkCDS();
                              printf("en avant toute!deuxieme essai
      CDS ={%x}   index {%d}\n",CDS,index);
                        }
                  }
            }
      }
      direction=security();
      return direction;
}
/*****************************************/
void wall_following(void)
{
      int direction=0xff;
      int correction;
      wall=0;
      while(wall==0)
      {
            wall=getwall();
      }
      while(direction ==0xff)
      {
            increment=1;
            direction=wall_follow();
            while(direction != 0xff)
            {
                  printf("ouups!  y a un obstacle : direction:
{%x}\n",direction);
                  {
                        if((direction & 0x01)==0x00)
                        {
                              printf ("Collision IR front
{%x}\n",direction);
                              wallbypass();
                        }
                        if(((direction & 0x02)==0x00) && wall==1)
                        {
                              printf ("Collision right bump
{%x}\n",direction);
                              wallbypass();
                        }
                        if(((direction & 0x80)==0x00) && wall==-1)
```

```
                        {
                              printf ("Collision left bump
{%x}\n",direction);
                              wallbypass();
                        }
                        direction=wall_follow();
                  }
            }
      }
}
/*************************************/
void wallbypass()
{
      int index=40;
      int IR;
      PROBLEM=0xff;
      while(index>0)
      {
            printf("en arriere toute!\n");
            my_motor(1 , -100);
            my_motor(0 , -100);
            index = index -1 ;
      }
      if (wall > 0)
      {
            index=30;
            while( index> 0)
            {
                  printf("ca tourne a gauche!\n");
                  my_motor(1 , 100);
                  my_motor(0 , -100);
                  index = index -1 ;
            }
      }
      if (wall < 0 )
      {
            index=40;
            while( index> 0)
            {
                  printf("ca tourne a droite!\n");
                  my_motor(1 , -100);
                  my_motor(0 , 133);
                  index = index -1 ;
            }
      }
      IR=checkIR();
      IR=IR & 0x0f;
      index=20;
      while(index>0)
      {
            my_motor(1 , 100);
            my_motor(0 , 100);
            index = index -1 ;
      }
      index=50;
      while(IR==0 && index>0)
      {
```

```
            printf("en avant toute!\n");
            my_motor(1 , 100);
            my_motor(0 , 100);
            index = index -1 ;
            IR=checkIR();
            IR=IR & 0x0f;
      }
      PROBLEM=0x00;
}
/************************************/
int wall_follow()
{
      int direction;
      int correction;
      int maxspeed;
      int index;
      int IR;

      correction=wall_process();
      printf("correction: {%d} \n",correction);
      maxspeed=40 * increment;


      if (correction > 0 && correction != 10)
      {
            printf("ca tourne a droite!\n");
            my_motor(1 , 0);
            my_motor(0 , 133);
            waitloop(100);
            if (correction ==2)
                  waitloop(75);
            if (wall>0)
                  index=15;
            else
                  index=10;
            while(index>0)
            {
                  my_motor(1 , maxspeed);
                  my_motor(0 , maxspeed);
                  index= index -1;
                  printf("j'avance!\n");
            }
      }
      if (correction < 0)
      {
            printf("ca tourne a gauche!\n");
            my_motor(1 , 100);
            my_motor(0 , 0);
            waitloop(100);
            if (correction ==-2)
                  waitloop(75);
            if (wall<0)
                  index=15;
            else
                  index=10;
            while(index>0)
            {
```

```
                  my_motor(1 , maxspeed);
                  my_motor(0 , maxspeed);
                  index= index -1;
                  printf("j'avance!\n");
            }
      }
      if(correction==10)
      {
            printf("y'a plus d'mur!!\n");
/*          index=5;
            if(wall==1)
            {
                  while(correction==10 && index>0)
                  {
                        my_motor(1 , maxspeed * 0.5);
                        my_motor(0 , maxspeed);
                        waitloop(10);
                        correction=wall_process();
                        index= index -1;
                  }
            }
            if(wall==-1)
            {
                  while(correction==10 && index>0)
                  {
                        my_motor(1 , maxspeed);
                        my_motor(0 , maxspeed *0.5);
                        waitloop(10);
                        correction=wall_process();
                        index= index -1;
                  }
            }
*/    }

      if(correction==0)
      {
            my_motor(1 , maxspeed);
            my_motor(0 , maxspeed);
            printf("ca roule!\n");
      }
      previouscorrection=correction;
      direction=security();
      return direction;
}
/***********************************************/
int security(void)
{
      int CDS;
      int BUMP;
      int HG;
      int IR;
      int direction=0xff;

      BUMP=checkbump();
      HG=checkHG();
      IR=getIRfront();
```

```
/* front */
      if((HG & 0x01)==0x01 ||(IR>IRfrontthreshold))
      direction= direction & 0xfe;

/* front right */
      if((HG & 0x02)==0x02 || (BUMP & 0x02)==0x02)
            direction=direction & 0xfd;

/* right */
      if((HG & 0x04)==0x04 || (BUMP & 0x01)==0x01)
            direction=direction & 0xfb;

/* backright */
      if((HG & 0x08)==0x08)
            direction=direction & 0xf7;

/* back */
      if((HG & 0x10)==0x10 || (BUMP & 0x04)==0x04)
            direction=direction & 0xef;

/* backleft */
      if((HG & 0x20)==0x20)
            direction=direction & 0xdf;

/* left */
      if((HG & 0x40)==0x40 || (BUMP & 0x10)==0x10)
            direction=direction & 0xbf;

/* frontleft */
      if((HG & 0x80)==0x80 || (BUMP & 0x08)==0x08)
            direction=direction & 0x7f;
      return direction;
}
/*****************************************/
int checkbump(void)
{
      int value=0x1f;
      int bump;

      bump=getBUMPbackright();
      if(bump<Pressthreshold)
            value=(value & 0xfe);

      bump=getBUMPfrontright();
      if(bump<Pressthreshold)
            value=(value & 0xfd);

      bump=getBUMPback();
      if(bump<Pressthreshold)
            value=(value & 0xfb);

      bump=getBUMPfrontleft();
      if(bump<Pressthreshold)
            value=(value & 0xf7);

      bump=getBUMPbackleft();
      if(bump<Pressthreshold)
```

```
            value=(value & 0xef);
      return value;
}
/*****************************************/
int checkIR(void)
{
      int value=0xff;
      int IR;

      IR=getIRbackright();
      if(IR<(IRbackrightthreshold - 3))
            value=(value & 0xfe);
      if(IR<(IRbackrightthreshold + 3))
            value=(value & 0xef);

      IR=getIRfrontright();
      if(IR<(IRfrontrightthreshold - 2))
            value=(value & 0xfd);
      if(IR<(IRfrontrightthreshold + 3))
            value=(value & 0xdf);

      IR=getIRfrontleft();
      if(IR<(IRfrontleftthreshold - 3))
            value=(value & 0xfb);
      if(IR<(IRfrontleftthreshold + 2))
            value=(value & 0xbf);

      IR=getIRbackleft();
      if(IR<(IRbackleftthreshold - 3))
            value=(value & 0xf7);
      if(IR<(IRbackleftthreshold + 3))
            value=(value & 0x7f);
      return value;
}
/*****************************************/
int checkHG(void)
{
      int value=0xff;
      int HG;

      HG=getHGfront();
      if(HG<Pressthreshold)
            value=(value & 0xfe);

      HG=getHGfrontright();
      if(HG<Pressthreshold)
            value=(value & 0xfd);

      HG=getHGright();
      if(HG<Pressthreshold)
            value=(value & 0xfb);

      HG=getHGbackright();
      if(HG<Pressthreshold)
            value=(value & 0xf7);

      HG=getHGback();
```

```
	if(HG<Pressthreshold)
		value=(value & 0xef);

	HG=getHGbackleft();
	if(HG<Pressthreshold)
		value=(value & 0xdf);

	HG=getHGleft();
	if(HG<Pressthreshold)
		value=(value & 0xbf);

	HG=getHGfrontleft();
	if(HG<Pressthreshold)
		value=(value & 0x7f);

	return value;
}
/*****************************************/
int checkCDS(void)
{
	int value=0xf;
	int CDS;

	CDS=getCDSextrightbright()+ getCDSextrightdark();
	if(CDS<CDSextrightthreshold)
	{	value=(value & 0xe);}

	CDS=getCDSrightdark()+ getCDSrightbright();
	if(CDS<CDSrightthreshold)
	{	value=(value & 0xd);}

	CDS=getCDSleftdark()+ getCDSleftbright();
	if(CDS<CDSleftthreshold)
	{	value=(value & 0xb);}

	CDS=getCDSextleftdark()+ getCDSextleftbright();
	if(CDS<CDSextleftthreshold)
	{	value=(value & 0x7);}

	return value;
}
/*********************************************************************
*****************/
int line_process(void)
{
	int CDS;
	int previous;

	CDS=checkCDS();
	printf("CDs= %x \n", CDS);
	if(!(CDS==0x2 || CDS==0x4 || CDS==0x5 || CDS==0x9 || CDS==0xa ||
CDS==0xb || CDS==0xd))
	{
		if(CDS==0x0)
		{
			return 10;
		}
```

```
            if(CDS==0x1)
            {
                  return 2;
            }
            if(CDS==0x3 || CDS==0x7)
            {
                  return 1;
            }
            if(CDS==0x8)
            {
                  return -2;
            }
            if(CDS==0xc || CDS==0xe)
            {
                  return -1;
            }
            if(CDS==0xf)
            {
                  return 0;
            }
      }
      return 0;
}
/*****************************************/
int getwall(void)
{
      int IR=0xff;
      IR=checkIR();
      IR=IR & 0x0f ;
      wall=0;
      if(IR==0x1 || IR==0x2 || IR==0x4 || IR==0x8 || IR==0x1 || IR==0x3
|| IR==0xc )
      {
            if((IR==0x01) || (IR==0x02) || (IR==0x03))
            {
                  wall= 1;
                  return wall;
            }
            if((IR==0x04) || (IR==0x08) || (IR==0x0c))
            {
                  wall= - 1;
                  return wall;
            }
            printf("mais y'en a un de mur!      IR {%d}     (IR &
0x03)=%x    (IR & 0x30)=%x\n", IR,(IR & 0x03),(IR & 0x30));
      }
      if(!(IR==0x1 || IR==0x2 || IR==0x4 || IR==0x8 || IR==0x1 ||
IR==0x3 || IR==0xc ))
      {
            PROBLEM=0xff;
            printf("ya pas de mur!\n");
            return 0;
      }
}
/*****************************************/
int wall_process(int increment)
{
```

```
      int IR=0xff;
      IR=checkIR();
/*    printf("wall (%d) IR {%x}\n",wall,IR);*/
      if( wall==1)
      {
            if((IR & 0x01)==0x00)
            {
                  if((IR & 0x02)==0x00)
                  {
/*                      printf("right wall back low front low\n\n");*/
                        return 2;
                  }
                  if(((IR & 0x02)==0x02)  &&    ((IR & 0x20)==0x00))
                  {
/*                      printf("right wall back low front
medium\n\n");*/
                        if(increment<0)
                              return 1;
                        if (increment>0)
                              return 0;
                  }
                  if((IR & 0x20)==0x20)
                  {
/*                      printf("right wall back low front high\n\n");*/
                        if(increment<0)
                              return 2;
                        if (increment>0)
                              return -2;
                  }
            }
            if(((IR & 0x01)==0x01)  &&    ((IR & 0x10)==0x00))
            {
                  if((IR & 0x02)==0x00)
                  {
                        printf("right wall back medium front low\n\n");
                        if(increment<0)
                              return 0;
                        if (increment>0)
                              return 2;
                  }
                  if(((IR & 0x02)==0x02)  &&    ((IR & 0x20)==0x00))
                  {
/*                      printf("right wall back medium front
medium\n\n");*/
                        return 0;
                  }
                  if((IR & 0x20)==0x20)
                  {
/*                      printf("right wall back medium front
high\n\n");*/
                        if(increment<0)
                              return 0;
                        if (increment>0)
                              return -1;
                  }
            }
            if((IR & 0x10)==0x10)
```

```
            {
                  if((IR & 0x02)==0x00)
                  {
                        printf("right wall back high front low\n\n");
                        if(increment<0)
                              return -2;
                        if (increment>0)
                              return 2;
                  }
                  if(((IR & 0x02)==0x02)  &&    ((IR & 0x20)==0x00))
                  {
/*                      printf("right wall back high front
medium\n\n");*/
                        if(increment<0)
                              return -1;
                        if (increment>0)
                              return 0;
                  }
                  if((IR & 0x20)==0x20)
                  {
/*                      printf("right wall back high front
high\n\n");*/
                        return -2;
                  }
            }
      }

      if( wall==-1)
      {
            if((IR & 0x04)==0x00)
            {
                  if((IR & 0x08)==0x00)
                  {
/*                      printf("left wall front low back low\n\n");*/
                        return -2;
                  }
                  if(((IR & 0x08)==0x08)  &&    ((IR & 0x80)==0x00))
                  {
                        printf("left wall front low back medium\n\n");
                        if(increment<0)
                              return 0;
                        if (increment>0)
                              return -2;
                  }
                  if((IR & 0x80)==0x80)   /**/
                  {
                        printf("left wall front low back high\n\n");
                        if(increment<0)
                              return 2;
                        if (increment>0)
                              return -2;
                  }
            }
            if(((IR & 0x04)==0x04)  &&    ((IR & 0x40)==0x00))
            {
                  if((IR & 0x08)==0x00)
                  {
```

```
/*                      printf("left wall front medium back
low\n\n");*/
                        if(increment<0)
                              return -1;
                        if (increment>0)
                              return 0;
                  }
                  if(((IR & 0x08)==0x08)  &&    ((IR & 0x80)==0x00))
                  {
/*                      printf("left wall front medium back
medium\n\n");*/
                        return 0;
                  }
                  if((IR & 0x80)==0x80)
                  {
/*                      printf("left wall front medium back
high\n\n");*/
                        if(increment<0)
                              return 1;
                        if (increment>0)
                              return 0;
                  }
            }
            if((IR & 0x40)==0x40)
            {
                  if((IR & 0x08)==0x00)
                  {
/*                      printf("left wall front high back low\n\n");*/
                        if(increment<0)
                              return -2;
                        if (increment>0)
                              return 2;
                  }
                  if(((IR & 0x08)==0x08)  &&    ((IR & 0x80)==0x00))
                  {
/*                      printf("left wall front high back
medium\n\n");*/
                        if(increment<0)
                              return 0;
                        if (increment>0)
                              return 1;
                  }
                  if((IR & 0x80)==0x80)
                  {
/*                      printf("left wall front high back high\n\n");*/
                        return 2;
                  }
            }
      }
/*    printf(" ca deconne  IR %x\n\n", IR);*/
}

/*****************************************/
void my_motor(int motor, int newspeed)
{
      if (newspeed >=0)
      {
```

```
            if (motor==0)
            {
                  newspeed= 0.75 * newspeed;
                  motorme(0 , newspeed);
            }
            if(motor==1)
            {
                  motorme(1, newspeed);
            }
      }
      if (newspeed <0)
      {
            if (motor==0)
            {
                  newspeed= 0.5 * newspeed;
                  motorme(0 , newspeed);
            }
            if(motor==1)
            {
                  motorme(1, newspeed);
            }
      }
}
/*****************************************/
int getHGfront(void)
{
      MUX=(0x04);
      return analog(4);
}
/*****************************************/
int getHGback(void)
{
      MUX=(0x24);
      return analog(4);
}
/*****************************************/
int getHGright(void)
{
      MUX=(0x44);
      return analog(4);
}
/*****************************************/
int getHGleft(void)
{
      MUX=(0x64);
      return analog(4);
}
/*****************************************/
int getHGfrontright(void)
{
      MUX=(0x84);
      return analog(4);
}
/*****************************************/
int getHGbackleft(void)
{
      MUX=(0xA0);
```

```
      return analog(4);
}
/*****************************************/
int getHGbackright(void)
{
      MUX=(0xC4);
      return analog(4);
}
/*****************************************/
int getHGfrontleft(void)
{
      MUX=(0xE4);
      return analog(4);
}
/*****************************************/
int getBUMPbackleft(void)
{
      MUX=(0x4C);
      return analog(5);
}
/*****************************************/
int getBUMPfrontright(void)
{
      MUX=(0x0C);
      return analog(5);
}
/*****************************************/
int getBUMPbackright(void)
{
      MUX=(0x2C);
      return analog(5);
}
/*****************************************/

int getBUMPfrontleft(void)
{
      MUX=(0x8C);
      return analog(5);
}
/*****************************************/

int getBUMPback(void)
{
      MUX=(0xCC);
      return analog(5);
}
/*****************************************/
int getCDSextrightbright(void)
{
      MUX=(0x14);
      return analog(6);
}
/*****************************************/
int getCDSleftbright(void)
{
      MUX=(0x34);
      return analog(6);
```

```
}
/*******************************************/
int getCDSextrightdark(void)
{
      MUX=(0x54);
      return analog(6);
}
/*******************************************/
int getCDSextleftbright(void)
{
      MUX=(0x74);
      return analog(6);
}
/*******************************************/
int getIRbackright(void)
{
      MUX=(0x6C);
      return analog(5);
}
/*******************************************/
int getIRfrontright(void)
{
      return analog(1);
}
/*******************************************/
int getIRfront(void)
{
      return analog(2);
}
/*******************************************/
int getIRfrontleft(void)
{
      MUX=(0xEC);
      return analog(5);
}
/*******************************************/
int getIRbackleft(void)
{
      MUX=(0xAC);
      return analog(5);
}
int getCDSrightbright(void)
{
      MUX=(0x94);
      return analog(6);
}
/*******************************************/
int getCDSleftdark(void)
{
      MUX=(0xB4);
      return analog(6);
}
/*******************************************/
int getCDSrightdark(void)
{
      MUX=(0xD0);
      return analog(6);
```

```
}
/*******************************************/
int getCDSextleftdark(void)
{
      MUX=(0xF4);
      return analog(6);
}
/*******************************************/
int getCDSambient(void)
{
      return analog(7);
}
/*******************************************/
void stop()
{
      while (1)
      {
            waitloop(1000);
      }
}
```