Multi Sensor Modular Platform for Robots

Final Thesis

David Viñas Arcas

June 2011

Master’s Thesis in Electronics

Master’s Program in Electronics/Telecommunications
Examiner: José Chilo
Supervisor: Niclas Björsell
Abstract

This project is about to create a multi sensor modular platform for robots. That means a platform created with several sensors, actuators and chips that had to be useful for future robot practice or projects.

As this project is a platform for future thesis, is required to use standard programming language, creating the control of the sensors so that it could be easily used by other students in the future, although they did not know the inner working of programs in particular.

Although this platform can be used for multiple applications, this thesis give special attention on a future project that the mainly purpose is create a robot capable of follow scents, or what is the same, different concentrations of gas. Thus, although the programs that we had to create could be used for various purposes, are specially done thinking of the future use for the creation of this robot tracker of smells.

So the main task of this project is to investigate and see how each of the sensors and actuators of our robot works and, in the other hand, learn to program a chip set not only to make it able to receive information from these sensors and send information to the actuators, but create some laws to control our multi sensor modular platform for robots to perform tasks from the acquired information.

Basically our robot is able to follow the direction that we want, with a close control loop in the direction using the information acquired from a three axis accelerometer, which allows too knowing the acceleration and consequently velocity and position of our robot. Also the robot is equipped with ultra sound sensors allowing detecting and avoiding obstacles. The smell sensor is implemented for the next versions of this robot.
Dedication

I want to dedicate this project to all my Erasmus friends who have transformed this final thesis into one of the best experiences of my live.

Thank you all for this time together. I'm sure I'll never forget.
Content

DEDICATION ____________________________________________3
CONTENT ______________________________________________4
LIST OF ABBREVIATIONS __________________________________7

1. INTRODUCTION ______________________________________ 8
1.1. Motivation of this project ...........................................10
1.2. Purpose ........................................................................11
1.3. Utility of this robot ........................................................12
   1.3.1 The smell sensors ....................................................12
1.4. State of the art ............................................................13

2. THEORY ______________________________________________ 15
2.1. Open and close loop systems ..........................................15
2.2. Responses of the systems depend of their order ...............16
   2.2.1 Response time step entry of first order systems ..........16
   2.2.2 Response time step entry of second order systems .......18
2.3. Identifying our system ..................................................19
2.4. Nyquist-Shannon Theorem ............................................21

3. HARDWARE ______________________________________________ 22
3.1. Microcontrollers .........................................................22
   3.1.1 The selected microcontroller for the robot .................22
   3.1.2 Main microcontroller functions used ........................24
   Output Compare ............................................................24
   PWM generation ............................................................24
   Double Comparation Mode ..............................................25
   Timers .....................................................................25
3.2. Electric engine ..............................................................26
3.3. Sensors ....................................................................28
   3.3.1 Ultra sound sensor (SRF05) ....................................29
   3.3.2 Accelerometer sensor ADXL327 ..............................31
   Capacitive technology accelerometers ..............................31
   Features of accelerometer ..............................................33
3.3.3 Smell Sensor – TGS 2442................................................................. 35
3.4. Programming board................................................................. 36

4. SOFTWARE .................................................................................. 37
4.1. Matlab 7.0 ............................................................................... 37
4.2. MPLab ..................................................................................... 38
      Menu 40
      Creating a project .................................................................. 41
      Writing the program ......................................................... 41
      Configuration of dsPIC: Easier with Visual Initializer .......... 42
      Simulating our program ..................................................... 42
4.3. C MPLAB C30 compiler .......................................................... 43

5. DEVELOPMENT ......................................................................... 44
5.1. Choosing the method of movement ........................................ 44
5.2. Main functions developed ....................................................... 46
      5.2.1 PWM Generator ........................................................... 46
      5.2.2 A/D converter ............................................................... 46
      5.2.3 Ultrasound sensor ......................................................... 47
5.3. Subprograms .......................................................................... 48
      5.3.1 Closed loop proportional control of speed .................... 48
      5.3.2 Closed loop proportional control of direction ............... 49
      5.3.3 Avoiding obstacles without speed and direction control 49

6. RESULTS ..................................................................................... 50
6.1. Final program algorithm ......................................................... 50
6.2. Electronic part of the robot ...................................................... 52
6.3. Building the model ............................................................... 53
6.4. Simulations ............................................................................. 54
6.5. Tests ...................................................................................... 54

7. CONCLUSIONS .......................................................................... 56

ACKNOWLEDGMENTS ..................................................................... 59

8. BIBLIOGRAPHY ......................................................................... 60

9. APPENDIX .................................................................................. 62
9.2. Creating two PWM............................................................... 68
9.3. Program for reading an analog signal: ................................................................. 70
9.4. Code of reading Ultrasonic sensor: ................................................................. 73
9.5. Program to have constant speed and direction ............................................. 75
9.6. Program to avoid obstacles but without the speed and direction control .... 82
9.7. Final program: ................................................................................................. 87
List of abbreviations

MCU  Microcontroller
DSP  Digital signal processing
A/D  Analog to Digital
DSC  Digital Signal Controller
PDIP  Plastic Dual In-line Package
PWM  Pulse Width Modulation: Periodic square signal
DTT  Digital Terrestrial Television
1. Introduction

Nowadays, our society is immersed in a spiral of unprecedented technologic growing. Since the first microprocessor commercialized in 1971 by Intel (Fig 1.1), the possibilities of creating new electronic devices have increased exponentially, thanks to the big consumption and the continuous research and development that have made them cheaper, smaller and more effective.

Today we are surrounded by technology in the day to day, reaching levels that would not expect 50 years ago. Our computers, mobile phones, television set-top DTT, scientific calculators and a multitude of other devices fill our lives and make them more easy and comfortable doing the arduous task and also they offer services that 50 years ago were only seen in science fiction like internet or video call.

Thus, although the word "Robot" may sound strange or futuristic to many people, the reality is that we are surrounded by them without noticing it, and every day the number of them increases. When the people thinks in a robot, always gives a humanoid shape that is far away from today's robots. A robot can be any programmable machine or electronic device, able to manipulate objects and perform operations previously with a own purpose.

Most robots currently in use may be found in industry, performing tasks that for humans can be dangerous, difficult or impossible. A clear example is the automobile industry uses robots to assemble parts as well as weld, paint, varnish and test their vehicles. But not only automobile industry uses robots, other industries see their future in robots like robotic pharmacies or healthcare operations robots that allow not only a much larger precision but work in a impossible environments for the human hand, allowing operations much less ostentatious, without almost scars and much safer.
But not only are present in the industry, but also may be found in the domestic environment to improve our quality of life. We can see robots vacuum cleaners, kitchen robots that cook, robots that cut the grass or even cyber-toys. The robots are coming to be near us, and they will. If it seems normal now that everyone has a computer at home and always carry a mobile phone that lets you call to any part of the world, is normal to expected that within 50 years our interaction with robots will be 24 hours a day, enjoying our robots to do household chores or give us alternative ways to have fun. But not only this, as we can read in the article *Friendly machines: interaction-oriented robots today and tomorrow* [1]. The robots will be able to interact with humans, learn lessons, recognize people, and have their own behavior.

Robots are generally composed of a data capture system and treating of these, as well as some peripherals that connect to the outside and help them both to receive information from the world around them, as to send signals to the world, thus we can find some processors or microprocessors which are elements that interpret instructions and process data from the programs, or say it in a simpler way, are the "brains" of the robot. These chips are found in the shape of chip or integrated circuit with a series of pins through which they can be programmed, send information from the outside world into the microprocessor or vice versa. These chips work digitally, meaning that instead of dealing with analog signals that can have infinite values, process information using a binary system with low and high level; it means that works with ones and zeros.
On the other hand, peripherals can distinguish between those who receive information from the microprocessor, called actuators, and those who give this information, called sensors. Sensors are important, because for each specific information that we want to capture of the outside world, we need a different sensor. It is for this reason that much of the development of robots is to create and produce new sensors that give us new information with which can treat, to apply robotics in new fields. We can see in the article *Industrial robot guidance using computer vision. Robotization of a radiator manufacturing process* [2] how the implementation of a new sensor, for example, the computer vision, can improve so much the efficiency of production and can allow the use of robotics in the assembly lines that were previously impossible to robotize.

Specifically we will talk about a sensor called accelerometers that are being used recently but they use in personal devices is increasing. These sensors are able to know the acceleration that they have, even dynamic or static one. They have been very famous since Apple used in their devices like Iphone and Ipad to make better the interface with the users. It allows the device know the orientation of the screen and the movements that the user do with these devices.

We also will explain how it works and which is the utility of the smell sensor, which is able to detect the different concentrations of gas in the air, so it can follow scents.

**1.1. Motivation of this project**

The motivation for doing this project was to deepen in the big world of robotics and automatic systems. Many times in the engineering university studies, the students learn a lot of theatrical knowledge to design the control algorithms of the robots, but they don’t know so much about how to put in practice all this theoretical knowledge. Although I had some practical classes in my studies, they were only a few and so much guides, so actually, they weren’t useful to understand and learn the whole process between the designing a control algorithm and make work a robot with this one.

As my studies specialty is automatic systems, I wanted some project that can help me to put in practice my theoretical knowledge and mix this, with the electric and electronic part of robots that is actually, the real part. So, a part from put in practice some of my lessons and knowledge that I got in my university studies, I wanted to
learn new electronic knowledge to, at the end of the project, not only be able to design the control algorithm, or study the stability of the system, but put this control algorithm into the reality by making a robot.

Also, I found very interesting to start knowing with new sensors and chips are used nowadays to be current and also know the brands that are important in the field of electronics and automatic systems, as after the final thesis, I will enter to the job world and it is important to know the main enterprises of this sector.

### 1.2. Purpose

The purpose of this project was to create a multi sensor modular platform for robots. That means a platform created with some sensors, actuators and chips that had to be useful for future robot practice or projects. The university gave us several sensors that we had to put together in one robot, in the way to make it work properly and controlled.

A part of doing that work, we had to program a chip, also given by the university, so these sensors would be able to interact with each other through the chip, in a controlled and efficient way. As this project is a platform for future thesis, it is required to use standard programming language, creating the control of the sensors so that it could be easily used by other students in the future, although they did not know the inner working of programs in particular.

Although this platform can be used for multiple applications, this thesis focuses on a future project that the mainly purpose is create a robot capable of tracking scents, or what is the same, different concentrations of gas. Thus, although the programs that we had to create could be used for various purposes, are specially done thinking of the future use for the creation of this robot tracker of smells. We also wanted to see how an accelerometer works, and how it can be used in a robot.

So the main task of this project is to investigate and see how each of the sensors and actuators of our robot works and, in the other hand, learn to program a chip set not only to make it able to receive information from these sensors and send information to the actuators, but create some laws to control our multi sensor modular platform for robots to perform tasks from the acquired information.
More specifically, our robot had to be equipped with a ultra-sound sensor, to allow the robot to detect the world around him, an accelerometer which was the newest and most complete sensor that we had, as we will explain, it is so used nowadays and detects the three axis acceleration, a smell sensor which is only needed to be implemented in the robot for the future projects and of course an electric engine to give to our robot mobility.

With all this elements, we started to search the best utility for all of them, and the best way to make it work together to achieve our goals.

1.3. Utility of this robot

As we have said, this robot is the first step for futures projects and thesis in the electronic and automatic field. This robot has to be a platform for the students to develop new and better projects, or just experiment with the electronic components. So the first and the most important function of this robot is to help in futures projects, developing news type of sensors and that will be able to add to our robot. As we develop the robot's motor bases, it can be used to follow various signals from the external world, such as light [3], sound or smell.

1.3.1 The smell sensors

This robot can be used for many applications, but especially it is prepared for is creating a smelling seeker. This robot can be used like a dog if we add the smell sensor. This smell sensor works detecting concentration of different types of gas, like a dog nose, so it can show the way to follow to our Robot. This could be very useful to detect gas leakages in domestic houses, especially if the people who live in is old, and have the sense of nose a little bit affected. This robot can help to detect where the highest concentration of this gas is, and can show some type of alarm to the people that lives in the house, if the concentration is to high for humans. Incorporate an olfactory sensor to a mobile robot, can be very useful to make different robots works together as we can see in the article [4] a group of robots can communicate between each other by leaving traces of a substance that evaporates, to perform the task of cleaning the floor of an un-mapped building, or any task that requires the
traversal of an unknown network, in the other hand, this “Smell Seeker” could be useful to detect gas leaks in cracks of landfill contaminants, which allows immediate action.

Another application of the gas sensors is the detection of excess CO2 in the environment, like in the city or near industries. Several robots like that can be work together in a network of simple sensors that be able to warn of abnormal concentrations of different gasses or just control these levels.

1.4. State of the art

After researching the best of our knowledge, the state of the art of robots that are able to follow scents, has not found any company or industry that manufactures robots like this, and there are not any presence in the nowadays market neither. But there are several universities researching in projects similar to that we proposed here. We can see how other universities are trying to do these projects such as the Faculty of Physics of the Autonomous University of Puebla (Spain), where they have develop a robot that is capable of detecting natural gas leaks. But the main development problem of this robot is the electronic nose. Quoting Professor of the Autonomous University of Puebla, and project coordinator Severino Muñoz Aguirre:
"There are delays in the development of an electronic nose, because it is complex in its structure. For the simulation of an electronic eye, for example, only requires three types of sensors, for taste is more complicated and the smell even more, because only the nose is composed of 900 different receptors and is required an equal number of sensors, so it remains one of the greatest delay in investigation needs."

[5]

This problem is also being investigated in other universities, as at the University of Malaga (Spain), where they got an electronic nose robot called "Rhodon" (Fig 0.4), which has software that creates maps of odor recognition, in which he states with a color code points it has located the highest concentration of gases.

But this is not the only problem with the development of these robots, as such as is it shown in the article Learning to Locate an Odour Source with a Mobile Robot [6], the odor detection so that a robot can follow a trail has several problems as air turbulence that delete the scent trail, or that samples collected by the sensors produce much noise in the signal. That is why there are another universities developing programs to solve this problem like, Monash Universit of Australia, such we can see in the article To Smell as naively Robot Has Smelt Before It [7] is the implementation of algorithms "sense-map-plan-act" style control strategy to model the airflow in the environment

using naive physics, then use the model to reason about odor dispersal, move to key positions gathering information, and go that way.
2. Theory

To understand properly the programs that we made to control the robot, it is important to know a little bit the main theory used.

2.1. Open and close loop systems

In the theory of control, we can find two big groups of systems: the open loop systems and the close loop systems.

Open loop systems are systems in which the output does not affect the control action. In an open loop system output is not measured or fed back to compare with the input. Therefore reference to each entry corresponds to a fixed operating condition, as a result, the system's accuracy depends on calibration. In the presence of disturbances, a system of open loop control does not perform the desired task because it can't notice that the real output signal is not the desired one. In practice, the open loop control is used only if we know the relationship between input and output and if no external or internal disturbances. Clearly, these systems are not interesting to study.

Control systems with feedback or closed loop systems, on the other hand, are more interesting to study. In a closed loop control, is used driver error signal as a action signal, which is the difference between the input and output feedback (which may be the same output signal or a signal function output and its derivatives and / or comprehensive) to reduce error and make the output of the system to a suitable value. The term closed loop control involves using a feedback control action to reduce the system error.
2.2. Responses of the systems depend of their order

To study the systems experimentally must know what are the responses of these systems depending on the input we give to them. The most typical studied response is that one that we have when we give and step signal. The step signal is a typical signal that widely used in the study of dynamic systems, which basically is a function such that:

\[
    u(t) = \begin{cases} 
        0 & \text{for } t < 0 \\
        1 & \text{for } t \geq 0 
    \end{cases}
\]  

To study the different systems we classify the order of the system, which is the maximum exponent of the denominator of the transfer function, which is the relation between the input and output. Thus, the simplest cases and also more simulated are normally first order and second order systems.

But work in the temporal domain is so hard, and normally people change to another domain that makes easier to understand all the main features of the systems, this is the Laplace domain. To make this domain change, we have to do the Laplace transform which is normally denoted by \( \mathcal{L}\{f(t)\} \) and it is a linear operator of a function \( f(t) \) with a real argument \( t \ (t \geq 0) \) that transforms it to a function \( F(s) \) with a complex argument \( s \). This transformation is essentially bijective for the majority of uses. The Laplace transform has one property very useful in the systems control, that many relationships and operations over the originals \( f(t) \) correspond to simpler relationships and operations over the images \( F(s) \).

2.2.1 Response time step entry of first order systems

We consider type systems like:

\[
    G(s) = \frac{1}{\tau s + 1}
\]

If the numerator is not the unit, it will be for linearity, multiplied for that constant but nothing else will change, the constant is called time constant, has dimensions of time and is measured in seconds.
The entry is a step signal:

\[ x(t) = u(t) \rightarrow X(s) = \frac{1}{s} \]  \hspace{1cm} (3)

And in consequence:

\[ Y'(s) = X(s) \cdot G(s) = \frac{1}{s} \cdot \frac{1}{\tau s + 1} = \frac{A}{s} + \frac{B}{\tau s + 1} \rightarrow Y(s) = \frac{1}{s} + \frac{1}{s + \frac{1}{\tau}} \]  \hspace{1cm} (4)

Where Laplace anti-transform is:

\[ y(t) = \left(1 - e^{-\frac{t}{\tau}}\right) u(t) \]  \hspace{1cm} (5)

And this looks like:

Fig. 2.2 Step response first order systems.
As can be seen in the Fig 2.2, after a time equal to , the system has reached 63% of its final value, after 3 to 95% and after 4 to 98%.

2.2.2 Response time step entry of second order systems

We can identify the systems of second order as:

\[ G(s) = \frac{k \cdot w_n^2}{s^2 + 2\xi w_n s + w_n^2} \]  

(7)

where \( k \) is the canonic gain, \( w_n \) is called the system's natural frequency and is measured in rad / s, \( \xi \) don't have dimensions and is called damping factor, since as we will see below, depending on this parameter, the response will be more or less oscillating.

The entry is a step signal:

\[ x(t) = u(t) \rightarrow X(s) = \frac{1}{s} \]  

(8)

And in consequence:

\[ Y(s) = \frac{k \cdot w_n^2}{s \cdot (s^2 + 2\xi w_n s + w_n^2)} \]  

(9)

To see the temporal response of this type of systems, we have to see first that the roots of this transfer function are \( s=0 \) and \( s = w_n \left( -\xi \pm \sqrt{\xi^2 - 1} \right) \) so the response will depend for the different values of \( \xi \).

Fig. 2.3 Step response second order systems.
As shown in the Fig 2.3 when $\xi = 0$ (blue curve) the oscillations continue indefinitely. For higher values of $\xi$ yields a faster decay of oscillations, but with a slower rise of the response (green curve has a value $\xi = 0.1$, while for the red $\xi = 0.5$). In the case where that $\xi = 1$, the system becomes critically damped to the point where the oscillations disappear (see pink curve).

2.3. Identifying our system

So first of all, we had to see what kind of system we could approximate to simulate it. The equivalent electric scheme that we use to model it is the following:

![Scheme of D.C. motor & Equivalent electric scheme of D.C. motor.](image)

Where the meaning of the variables are:
Watching this scheme we can find the following equations:

\[ v(t) = e(t) + i_a(t) \cdot R_a + L_a \cdot \frac{\partial i_a(t)}{\partial t} \]  \hspace{1cm} (10)

\[ e(t) = K_m \cdot \omega_M(t) \]  \hspace{1cm} (11)

\[ \tau_N(t) = K_t \cdot i_a(t) \]  \hspace{1cm} (12)

\[ J \cdot \frac{\partial \omega_M(t)}{\partial t} = \tau_M - \tau_L - b \cdot \omega_M(t) \]  \hspace{1cm} (13)

So, we can make the Laplace transform and get the same equations but in Laplace domain, but first we will make the simplification of \( \tau = 0 \) it will simplify a lot the equations and won’t change the type of our system. Then we will search it but we will see that it don’t cause important changes to our model. With this simplification we get the following equations:

\[ V(S) = E(S) + I_a(S) \cdot R_a + L_a \cdot I_a(S) \cdot S \]  \hspace{1cm} (14)

\[ E(S) = K_m \cdot W_M(S) \]  \hspace{1cm} (15)

\[ \tau_M(S) = K_t \cdot I_a(S) \]  \hspace{1cm} (16)

\[ J \cdot S \cdot W_M(S) = \tau_M - b \cdot W_M(S) \]  \hspace{1cm} (17)

Rearranging and simplifying these equations we can get the transfer function of our engine:

\[ \frac{W_M(S)}{V(S)} = \frac{\frac{K_t}{L_a \cdot J}}{S^2 + S \left( \frac{R_a \cdot J + L_a \cdot b}{L_a \cdot b} \right) + \frac{K_m \cdot K_t + R_a \cdot b}{L_a \cdot J}} \]  \hspace{1cm} (18)
As we can see, it is a second order system, so we can know the different responses of different stimulus if we know the value of the different constants of our model. As we will see [8] this constants can be found with Matlab software.

### 2.4. Nyquist-Shannon Theorem

The sampling Nyquist-Shannon theorem, also known as sampling theorem of Whittaker-Nyquist-Kotelnikov-Shannon, Nyquist or Nyquist theorem is a fundamental theorem of information theory, of particular interest in telecommunications.

The theorem shows that the exact reconstruction of a continuous periodic signal in base band from their samples, it is mathematically possible if the signal is limited in bandwidth and sampling rate is more than twice its bandwidth.

In other words, the complete information of the original analog signal that meets the above criteria is described by the total number of samples that resulted from the sampling process. Nothing, therefore, the evolution of the signal between samples is not perfectly defined by the total number of samples.

If the highest frequency contained in an analog signal is and the signal is sampled at a rate , then you can fully recover the information of the analog signal from their samples.

This theorem is useful when we have to make digital control algorithms, because digital systems have to deal with samples instead of analog signals. But in our project, as we will not sample periodic signals of high frequency, and the sample rate of our microprocessor is very high, we will not have to worry about that.
3. Hardware

3.1. Microcontrollers

The classic microcontrollers, called MCU, are chips that have a complete digital processor with auxiliary peripherals that make easier to develop the applications that they do. The MCU are similar to DSP but they differences make that they have been implanted in very different fields. Bravely, a DSP is a microcontroller equipped with the logic and physic resources to be able to handle the specific applications of digital processing of signals.

The complex arithmetic instructions of MCU are executed in several cycles, but the DSP ones only need one cycle. In DSP there are always fast and accurate A/D converters. The most of the programs for DSP are mathematic programs, so they are prepared for being programmed by high level languages, like C++ languages. The speed and efficiency of the DSP are higher than the normal MCU. The enterprise Microchip Technology Inc.® is in the first place in the world ranking of 8 bits microcontrollers since 2003. Their models are publicly known as PIC®. After the successful launch of the 16 bits microprocessors, people need news devices to support the function of digital process of signals to respond to the new tends of the market that were increase the connectivity with internet, improving related with sound and video, the control of the engines, etc. In that situation Microchip built a hybrid between MCU and DSP, that management were similar to microcontrollers but that includes the main features of DSP. That is how Digital Signal Controller was born (DSC). As we can see in the article [9] these new controllers have the main features of 16 bits PIC and the DSP of low range. Since then, they are being used in many devices and applications.

3.1.1 The selected microcontroller for the robot

The dsPIC30F4013 microcontroller has been selected for doing our Robot. The dsPIC30F4013 belongs to the family of general purpose, and this is a good reason to be selected because we can explore the possibilities of dsPIC and it will be very useful for future applications. On the other side, is a DSC with PDIP encapsulated (Plastic Dual In-line Package) that allows easier electronic circuits than other type of encapsulated. The only bad point of this controller is that it doesn't have the specific
hardware for the treatment of PWM (Pulse Width Modulation), very useful in engines control, but as we will see, it is not a problem because we will do it with another function, the Output Compare. The main features of this dsPIC are:

**Memory**
- Program memory of 48 K of capacity. Up to 16K instructions.
- 2048 bytes of SRAM memory
- 1024 bytes of EEPROM memory
- 16 work registers, with 16 bits each one

**Peripherals**
- 5 timers of 16 bits
- 4 capture modules of 16 bits
- 4 compare outputs of 16 bits
- 2 UART modules
- 1 SPI module

![Fig. 3.1 Diagram of connections of dsPIC30F4013](image-url)
3.1.2 Main microcontroller functions used

Output Compare

Output compare module is used in many programs to change the values of the digital outputs, for create a single pulse or for create a PWM. In our programs is used to create the PWM that will control the engines of the robot, and to create the single pulse that the ultra sound sensor need to work.

The dsPIC30F4013 has 4 output compare modules, which can be selected by the register OCxCON (where x can go from 1 to 4 ). As we have explained, this module can be used in many modes by setting the according value in the OCM register (Output Compare Mode). Here we only explain the two modes used in our programs, the PWM generation and the double pulse “comparation.”

Both of this functions need to select first the timers that we will use by setting the register OCTSEL (Output Compare Timer Selection) at 0 for timer 2 and OCTSEL = 1 for timer 3. Then we have to set the OCxR register (Output Compare Read), OCxRS (Output Compare Read Second) register and PRx register (Period Register).

PWM generation

In PWM generation our timer start increasing its value in the TMRx register (where x is the number of the timer), when TMRx has the same value than OCxR register, the digital signal of OCx pas from high, to low. Then the TMRx continues increasing its value until the register TMRx=PRx, or what is the same, until the value of the timer and the period for this OC are the same.

![PWM Diagram](image)
When this happen:

- TMRx value is reseted
- The digital signal of OCx passes from low to high
- The register OCxR takes the value of OCxRS register
- The process starts again, generating a PWM which Duty Cycle is:

\[
Duty\ Cycle = \frac{OCxR}{PRx + 1} \cdot 100
\]  

(19)

**Double Comparation Mode**

This is used to create a single pulse, the TMRx register start increasing its value and when the register TMRx has the same value of OCxR register, then the digital value of OCx pass to low to high, TMRx continues increasing its value, and when TMRx register has the same value of OCxRS register, the digital value of OCx (Output Compare and x says which number) pass from high to low. We can generate another pulse resetting the timer, otherwise the signal level will not change.

**Timers**

The controller has 3 timers that increments their value with the oscillation of a clock. Actually, this clock also called crystal oscillator, is an electronic oscillator circuit that uses the mechanical resonance of a vibrating crystal of piezoelectric material to create an electrical signal with a very precise frequency. This clock can be internal or external. To increase in one the value of the timer, is needed 4 pulses of the clock. Timers also have a frequency dividers called preescalers that allow the timer to increase in value more slowly. These prescaler can be set with the relations of 1:1, 1:8, 1:64 or 1:256 by changing the value of TxCON register. So if we want to know the PWM period, we can know it from the following expression:

\[
PWM\ Period = ((PRx) + 1) \times 4 \times TOSC \times \text{Prescale}
\]  

(20)
**Analog to Digital Converter**

This module will be used to read the signals of the accelerometer that will give us the acceleration of the robot. The dsPic 30F4013 A/D converters have 12 bits of resolution, that means that the difference of voltage between Ref+ pin and Ref- pin can be split in 4095 values. The ADCON1, ADCON2 and ADCON3 registers control the operation of the A/D module. The ADCHS registers selects the input channels to be converted. The ADPCFG register configures the port pins as analog inputs or as digital I/O. The ADCSSL register selects inputs for scanning. When the conversion is finish, automatically the DONE register is setting high.

![Fig. 3.3 Sampling of analog signal](image)

The module contains a 16-word dual port read only buffer, called ADCBUF0...ADCBUFF, to buffer the A/D results. The RAM is 12 bits wide but the data obtained is represented in one of four different 16-bit data formats. The contents of the sixteen A/D Conversion Result Buffer registers, ADCBUF0 through ADCBUFF, cannot be written by user software.

The speed of the A/D conversion is about 100 Ksps and is also possible during the sleep mode. This functions is useful when the sensor that has to change the from the sleep mode to normal mode give us a analog signal, otherwise it will be impossible.

### 3.2. Electric engine

The robot has two electric DC engines that can move one wheel each one. Those make possible the movement in every possible direction in a 2D space. An electric DC engine is a machine that transforms the electric power in mechanic power,
normally in a rotator movement. The speed of the rotations is given by the voltage difference between the terminals. If the difference grows, the speed will do it too.

The operating principle is that when the coil is traversed by current, it generates a magnetic field thus creating a north pole and a south pole. This magnetic field interacts with another magnetic field that can be generated by a permanent magnet or a other coil. In the case of motors with brushes, brushes are the part of the engine that allows the magnetic field rotates when the rotor of the engine turns. This make that the two poles of the same sign never fully align and the movement is perpetual.

According to the Lorentz law, when a driver by passing an electric current immersed in a magnetic field, the “conductor” suffers a force perpendicular to the plane formed by the magnetic field and the current, following the right-hand rule, with module:

\[ F = I \times B \times l \]

(21)

\( F \): Force in Newtons  
\( I \): intensity that runs through the conductor in amps  
\( l \): Length of conductor in meters  
\( B \): Density of magnetic field or flux density tesla

As we can see in the Lorentz law, and knowing the Ohm law, if we want to make slower the speed of the motor, we have to make lower the voltage given to the motor, but we make lower the intensity, making lower the torque of the engine. That’s why instead of a continuous signal, we will use a pulse width modulation or PWM (Fig 3.4.) This is a periodic signal that can be sinusoidal or square, it which we can modify that this periodic signal has a high value or a low value. Controlling the speed of the electric engine in this way, we can modify the speed without modifying the torque of the engine.

The speed will have a direct relation with the duty cycle that can be calculated by:

\[ D = \frac{t}{T} \]

(22)
\(D\) is the duty cycle

- is the time the function is positive (pulse width)

\(T\) is the period of the function

The D.C. geared motor with brushes that we will use for our robot is 3.9 Watt engine, powered by 12V and with a gearbox ratios options for 0.36 to 430 rpm. The nominal speed changes with the torque in a specific way (fig 3.5)

**3.3. Sensors**

In this chapter we discuss the technical specifications of the different sensors that use our Multi Sensor Modular Platform. Understand the mechanical and electronic
operation will be essential to make a proper use of the information that they you give us.

3.3.1 Ultra sound sensor (SRF05)

Ultrasound is a normal sound, except they have a higher frequency than the maximum audible to the human ear. It starts at about 16 Hz and has a limit above approximately 20 Khz. The basic operation of ultrasound as distance meters is very easy to understand (3.6). The sensor has a transmitter that emits an ultrasonic pulse that bounces on a particular object and the reflection on that pulse is detected by the ultrasound receiver. The SRF05 sensor as the most low-cost ultrasonic sensors is based on the emission of a pulse which field of action is tapered.

Measuring the time delay between the emission of sound and perception of the echo can set the distance that is the impediment that produced the refelction of the sound wave by the formula:

\[
\text{d} \text{ is the distance} \\
\text{V} \text{ is the speed of the sound, normally around 340 m/s} \\
\text{t} \text{ the time that it takes to go and came back}
\]

\[
(23)
\]

In this case, SRF05 can work in two different ways, transmitting and receiving the signal in the same channel, or in different. The way that we choose is in different channels because is easier to make it work. As it is shown in the Fig 3.7 the method of use is very simple:
1. Externally applied, by the user, a single pulse of 10µs of minimum duration.

2. The sequence begins; the module transmits a pulse train of 8 cycles at 40KHz.

3. At that time the signal “echo” passes to high level.

4. When the receiver receives the transmitted signal consequence of have rebounded from an object, this output goes back to low level.

5. The user should measure the length the “echo” pulse, which means, the time that the signal “echo” is in high level.

Echo pulse duration varies between 100µS output and 25ms, depending on the distance between capsules and the object module. The speed of sound is 29.15 mS / cm, as does a scan of leg return, is set in 58.30µS/cm. Thus the minimum range that can be measured is 1.7 cm (100µS/58) and the maximum of 431 cm (25mS/58).

Although as we can see in the article [10] the operation seems simple, but there are factor inherent in both the ultrasound and the real world, influencing in a factor on the measurements. It is important to know which factors can modify the measure to try to correct or avoid them.
Among the various factors that alter the readings carried out with ultrasonic sensors include:

- The shape of the ultrasound actuation field is conic, and the first echo that we get, is the one that is measured. So, if the obstacle is not in the middle of the actuation field, the object can be farther away than the lecture says.

- The amount of energy or sound reflected depends strongly of the material and shape of the obstacle. Depending of each material, it can absorb more sound or less, changing the lecture of our sensor.

- Low-cost ultrasonic sensors uses the same transducer as a transmitter and receiver, so after the emission of the ultrasound, the sensor wait a predeterminated time to vibrations in the sensor disappear and be prepared to receive the echo produced by the obstacle. This implies that there is a minimum distance d (proportional to relaxation time of the transducer) from which the sensor measured accuracy. In general, all objects that are below this distance, d, will be interpreted by the system as they are a distance equal to the minimum distance.

- The ambient can modify the speed of sound and the direction of the waves of sound.

### 3.3.2 Accelerometer sensor ADXL327

**Capacitive technology accelerometers**

This low-cost accelerometer is based in the capacitive technology. Such devices are responsible for changing the relative position of the plates of a micro condenser when it is subjected to accelerations.

In other words, the capacitive work in the way varying the capacitance between two or more conductors between which there is always a dielectric material.

Sensors based on this measuring technology provided the acceleration when they are integrated into a silicon chip. The integration in silicon chips reduces different
types of problems such as (humidity, temperature, parasitic capacitances, total number of terminals, etc.)

Principle of operation:

These sensors are formed internally by a fixed set of capabilities (anchored to circuit) and on the other hand there is a core set of plates are attached to elastic rings allowing the movement of the plates. The sensor is taken as an example consist in 46 core plaques and their size is 0.5 mm in cross section. Darkest areas are the anchored capacities into the integrated circuit (such as $C_{ao}$ and $C_{bo}$), while the mobile plaques are found in the center of two fixed capacities. As shown in figure 3.8

![Plate system for capacitive sensor](image)

**Fig. 3.8** Plate system for capacitive sensor

The idea is that when there is not acceleration in the sensor, the center plate between the capabilities $C_{bo}$ and $C_{ao}$ is right in the middle of these. Therefore the value of the capability will be the same for both ($C_o$).

When we apply acceleration in the direction of the central plate (located between the two capabilities) this will move causing a capacity increase and one of the capacitors, and a decrease in the other.

Mathematically speaking, the value of the capacities can be calculated as:

$$C_b = C_o \cdot \frac{d}{d - \varepsilon}$$

$$C_a = C_o \cdot \frac{d}{d + \varepsilon}$$

(22)
• $d$: Distance between the center plate and skills.
• $\xi$: the displacement
• $S$: is the surface of the plate.

ADXL327 acceleration sensor is a low gravity sensor that is designed to detect the dynamic and static acceleration of 3-axis XYZ. provides three output voltages for each in X, Y and Z, which are proportional to the value of the acceleration experienced by the device on each direction.

**Features of accelerometer**

The power supply of this sensor is from 1.8V until 3.6V. The lectures of the acceleration will be directly proportional to the input voltage, being the half of it when there is not acceleration. Working that way the sensor has the same range to positive acceleration than for negative one.

Also, the intensity consumption is more or less proportional to the power supply, as we can see in the Fig 3.10

![Fig. 3.10 Typical current consumption vs. Supply Voltage](image)

Another important point when choosing a device is the sensitivity. The sensitivity of accelerometers let you know how much of the acceleration perceive the sensor when this is applied to the device. In other words, we will have with voltage will be get in function of the acceleration supported by the accelerometer.
The units used typically by the manufacturers of sensors are typically provided to us like mV/g. In ADXL327 this sensibility is normally 420mV/g.

The following figure shows a function of the notch (mark in the sensor that indicates the device placement) as it establishes the axes of three-dimensional coordinates to perform properly the study and positioning of the accelerometer.

![Fig. 3.11 Block diagram](image)

![Fig. 3.12 Output Response vs. Orientation to Gravity](image)

![Fig. 3.13 Axes of acceleration sensitivity](image)
3.3.3 Smell Sensor – TGS 2442.

In our case, the smell sensor is a sensor able to detect the carbon monoxide in the air, and have a small sensibility to the H2 gas, but it can be switched by another one if we want to detect another gas. TGS2442 utilizes a multilayer sensor structure. A glass layer for thermal insulation is printed between a ruthenium oxide (RuO2) heater and alumina substrate.

![Smell sensor](image)

**Fig. 3.14 Smell sensor**

A pair of Au electrodes for the heater is formed on a thermal insulator. The gas sensing layer, which is formed of tin dioxide (SnO2), is printed on an electrical insulation layer which covers the heater. A pair of Au electrodes for measuring sensor resistance is formed on the electrical insulator. Activated charcoal is filled between the internal cover and the outer cover for the purpose of reducing the influence of noise gases.

TGS 2442 displays good selectivity to carbon monoxide, making it ideal for CO monitors. In the presence of CO, the sensor's conductivity increases depending on the gas concentration in the air. A simple pulsed electrical circuit operating on a one second circuit voltage cycle can convert the change in conductivity to an output signal which corresponds to gas concentration.

![Sensitivity](image)

**Fig. 3.15 Sensitivity**
3.4. Programming board

To burn the programs in the dsPic 30f4013, we did the programming board according to the indications of PICkit Users Guide [11]. By making this circuit called In-Circuit Serial Programming (ICSP) we can burn our dsPic connecting the PICkit usb to the computer and the other part connected to the programming board. To do that is required to have five signals as we can see in the Fig 3.16:

VPP – Programming Voltage; when applied, the device goes into Programming mode.

ICSPCLK or PGC – Programming Clock; a unidirectional synchronous serial clock line from the programmer to the target.

ICSPDAT or PGD – Programming Data; a bidirectional synchronous serial data line.

VDD – Power Supply positive voltage.

VSS – Power Supply ground reference.

---

Fig. 3.16 Typical ICSP application circuit
4. Software

To program our robot we used two different free software that you can find in internet easily and is very useful no only for programming, but for test our programs too. The most important software is called MPLab and it is a free product of Microchip Brand.

4.1. Matlab 7.0

In the beginning of the project we thought that the best option to make the control would be with the software Matlab or Labview running in a laptop. We thought that because I knew how these programs works, I had been working with them in my university studies and they have a lot of options that makes easiest to make controllers like PID or others. They have a friendly interface with the user that allows everybody with a few knowledge in programming to make a program to control sensors or engine. As we will see, this won’t be the option that we finally chose, but we did some work that it’s interesting and could be useful for next versions of this project.

To design a controller with Matlab, the first thing that we had to find was what type of system we had. The main component that we wanted to control was a D.C. geared motor with brushes. As we had found the type of system that we had, the only thing remaining to know everything about our system is the different constants that appear in the last equation. As we can see in the article [8] one time again, Matlab can help us with this problem with Matlab IDENT. This Toolbox software lets you construct mathematical models of dynamic systems for measured input-output data. Taking the information of which system we have, and the input-output data to different signals of different frequencies the program can approximate all the parameters of the model.
At this point of the project, the only thing that we needed was a sensor to get the output data. But we made a change in our plans, we got the help of the electronic department of the university, and after consider the good points and the bad points, we decided to change the software and the hardware that we wanted to use to make the program. We decided that it would be better to make the robot with a chip, instead of a laptop. With this choice, the robot could be smaller, and therefore, the engines and batteries more small and chip. This thing made easier to build the model and more economic. A part for that, the robot would be more professional and closer to the robots that are used in the industry and research. This change in the hardware was a change in the software part, we changed from Matlab to MPLab.

### 4.2. MPLab

This software tool, developed for Microchip, it is available free in the website of Microchip, and it is the main working tool for this project. The MPLAB IDE is software...
with a program editor, project manager, program debugger and several tools for the development of PIC and dsPIC applications.

The main features of MPLab are:

- Developed for be used in Windows Operative System
- Integrated code editor and highlighter code
- Possibility of assemble, compile or link the code
- Possibility of run the program step by step with the debugger or on real time.
- Make measures in real time with the debugger.
- Watch the variable value and memory positions with inspection windows
- Programming the controller firmware through the programming tools

When we are working with the develop environments of MPLAP IDE it’s so recommended to make easier the programmer job to work with “projects” and these
projects include some files. Ones of them are used to initialize properly the controller, define the different memory areas, reset, main program, etc. Other ones have the symbolic register names, memory directions, etc. And finally the “projects” have the main code programs. This division in several files, and the ones already included in the MPLAB IDE environment simplify the applications development.

Menu

MPLAB IDE has a classic menu in the top part of the screen, from which we can access to all the functions.

The menus File, Edit, Window and Help don’t have any special feature. But the menu View has specific features:

From the View menu we can configure the tools bars, we also can see the fundamental parts of the project, like the program memory, EEPROM memory, a usage memory graph, a window called “watch” allow us to see the value of different pins and variables, etc.

It is very useful in the simulations, because we can see the internal operation and detect mistakes that otherwise will be impossible to detect.

Another special menu is the “Project” menu, which is fundamental for doing programs and applications. We can start projects, compile, build and also add files to make work the program.
We can not see the “Debugger” menu properly if the MPLAB SIM tool is not selected. When this is selected, we have the possibility to run the programs step by step, put breakpoints in the code, reset the application, etc.

The programmer menu allow to burn the chip if we have a programmer board.

**Creating a project**

First of all, we have to choose which program language we will use, and in this case, it will be C. The dsPIC family has been optimized for the execution of programs developed in C. A good program in assembler language always will be more optimal in code size, and execution time, but the C language has the advantage of be easier to learn and to use, increasing the performance of programming task.

So first of all, we have to configure our project following these steps:

- Project -> Project Wizard…
- Next
- Select module dsPIC30F4013
- Select Microchip C30 Toolsuite
- Select the name and browse where we want to save the project
- Add C:\Program Files\Microchip\MPLAB 30\support\gld\p30f4013.gld
- Click Finish, and it is done.

**Writing the program**

Now, we can start writing our program doing *File*->*New* and adding this file to our project in the Project Window by clicking the *Source Files* -> *Add file*. When we have finished of writing the program, we have to check if everything it is right. For doing this we make:

- Project-> Build all.
Doing this, we will see if and the end appear BUILD SUCCEEDED that's means everything is it okay, or BUILD FAILURE, that means that we have some mistake in our program, a little bit above this line we can read the lines in our program that made the error. If we click on those lines we will go directly to the part of the code that has problems.

**Configuration of dsPIC: Easier with Visual Initializer**

Visual Initializer or VDI is a tool of MPLAB IDE to make easier the configuration of dsPIC. With Visual Initializer the only thing that we have to do is drag what we want to configure into our dsPIC and then, configure it easily with different windows. Visual Initializer check if there are some illogical things in the configuration of dsPIC and shows error messages showing where the inconsistency is. For having access to this tool we have to go to:

Tools -> Visual Initializer

There we can select the encapsulation of our chip, and add what we want dragging the icons of different features inside the chip. When we are finish, we select:

Visual Initializer -> Code Generation Options -> Generate code only for features/resources on processor package

It will generate a code with the entire configuration that we need in a file called “init_dsPIC30F4013.s” that we can add to our project. Then the only thing that we have to do is call this code at the beginning of our program with the instruction:

“call_VisualInitialization”

**Simulating our program**

Many times, we need to simulate our program to see if everything works as we want or to detect errors in the way that it works. For doing that we have toolbox MPLAB SIM, and then:

Debugger-> Select Tool -> MPLAB SIM

Then we are able to simulate our program with “run” options or simulate it line by line with “step into” or “step over”. When we are simulating our program is very useful to
use properly the window “view” to see what is happening. The most important windows for doing a good simulation are:

Locals: We can see the value of local variables

Watch: We can select variables that are interesting for us

Call Stack: We can see which command orders are being called

Simulator logic analyzer: We can make a graph to see which values have the different variables of our program.

Logic Trace: We can see the value of different inputs and outputs.

The simulator has also the possibility of set inputs and outputs with the debugger menu option Stimulus -> New workbook. Here we can put the values to different pins of our PIC to see how it is the program response.

The simulator allows to count the time between instructions using break points, this is very useful when we want to know the period of our program, the time that spend one A/D conversion or things like that.

4.3. C MPLAB C30 compiler

Microchip has a C compiler that can works with MPLAB IDE. This tool it’s not free for everyone, but there is a test version of 60 days, and if it’s for student projects, we can get the student version for free.
5. Development

5.1. Choosing the method of movement

We contemplate two different ways to make the robot move: Traction and direction motion vs. Differential wheel motion. The first was the easiest of both and is to have one engine that gives only the traction to the robot in the back wheels of it, and another engine that gives only the direction to the front wheels of it. It is the same system that uses the real cars with rear wheel drive. The good points of this system is that is easy to control because you can control the velocity and the acceleration of the robot only controlling one engine, and controlling the other engine, you can control the velocity. The bad points of this system is that the mechanical part of the robot is more complicate because you have to put a differential part in the traction part to take the curves, and a lot of articulations in the direction part to make it work properly. That makes the robot more complicate to build, and more sensitive to break or fail.

![Diagram of Traction and Direction Motion](image)

**Fig. 5.1** Scheme of the traction and direction way to move.

The other way to move is the differential wheel motion, this consist in two wheels in the back part of the robot controlled by two different engines. That allows to move
each wheel independent and therefore, we can make run each one in different velocities. So, if the two wheels rotate at the same speed, the robot will go straight ahead, but if one rotate faster than the other, the robot will turn. With this system we can control the traction and the direction of the robot only with the back wheels so in front, we only need to put one or two “crazy wheels” that have an articulation to rotate in every direction that we want.

The good points of this system are that the mechanical part is very easy to build, and the robot is simple and strong. That makes easier to build a model, like we wanted. The bad point is that is more difficult to make the control law, because in every moment we have to check if the two wheels are rotating as we want, since a small difference in the speed of the wheels will change the velocity and the direction of the robot.

As we can see in the scheme of above, the velocity of the robot will be given for the expression:

\[
V_{\text{Robot}} = \frac{V_i + V_d}{2}
\]  

And the angular velocity will be given for the expression:

\[
W_{\text{Robot}} = \frac{V_i - V_d}{R}
\]  

After analyze both systems we considered that the best system was the second one because for us the important part wasn’t the mechanical part, but the control and
electronic part. We thought that the fact that the mechanical part was easiest would help to build the model, and the fact that the control part was more difficult would make the project more interesting and useful to learn new things. To understand correctly the whole programs and control algorithm that we have done to do the final program, we have to understand how the main parts of this work. We can split our programs in three main functions, two subprograms and the final program.

5.2. Main functions developed

5.2.1 PWM Generator

This part of the program generates two different PWM that are used to move the engines, the period of each are the same, 20ms, and the only thing that we can change is the duty cycle of each one. This program uses one timer, TMR3, and two outputs compare pins, OC1 and OC2. To set the period of the signal we did the following operation:

\[
\text{Value of } \text{TIMER2} = \frac{20 \cdot 10^{-3}}{\frac{1}{8 \cdot 10^6} \cdot 4 \cdot 64} = 625(\text{Decimal}) = 271(\text{Hexadecimal})
\]

5.2.2 A/D converter

To read the signal of the smell sensor or the accelerometer, we need to read the analog signal and converted into a digital signal. This part of the program set some pins like the analog inputs, makes a reading of the signal, and then starts the A/D
conversion. When the conversion starts, the register DONE change its level from low to high, and when the conversion is finish, change it values from high to low, that is the way we know that we can read the digital conversion. The conversion is written in some buffer, there are thirteen buffers or channels to choose. Those buffers have the resolution of twelve bits, which means 4095 values to split the measure. As in our programs the reference voltage is the supply voltage, if we want to convert the measure to volts:

\[
\frac{5V}{4095\text{ values}} = 0.001221 \text{ V/value}
\]  

(26)

And in our case, as we want to convert the analog input of the accelerometer to the acceleration in g(m/s^2), we should do the following operation:

\[
\frac{420mV/value}{1.221mV/g} = 344 \text{ values/g}
\]  

(27)

5.2.3 Ultrasound sensor

This program sends a pulse of 100 µs to the ultrasound sensor and set one of the pins to read a digital input. When this digital input changes from low level to high level timer3 starts, then when this signal changes from high level to low level, the timer3 stop. Then the time that the signal was at high level is calculated:

\[
\text{Time in TIMER} = \frac{1}{8 \cdot 10^6} \cdot 4 \cdot 64 \cdot 1 = 3.2 \cdot 10^{-5} \text{ seg/clock}
\]  

(28)

And knowing that the ultrasound sensor sensibility, we can calculate the distance:

\[
\frac{3.2 \cdot 10^{-5} \text{ seg/clock}}{513 \cdot 10^{-6} \text{ seg/cm}} = 0.624 \text{ cm/clock}
\]  

(29)
5.3. Subprograms

5.3.1 Closed loop proportional control of speed

The basic function of this subprogram is to set the speed of our robot properly. To make that we used a closed loop proportional control, which means that error correction is proportional to this one.

As the sensor that we use is an accelerometer, first of all we have to read the acceleration, then find the new speed with the equation:

\[ V = V_0 + a \cdot t \]  

(30)

Where \( t \) is the time between lectures, and every time we find a new speed, this goes to register for the next loop. With this speed, we can compare with the reference speed and find the error. If the error is very big, the increasing of the speed, and therefore, the acceleration, will be big. Otherwise if the difference between the real speed and the reference speed is small, the increase of the acceleration will be small too. This process will continue until the difference of the speed is smaller enough to enter into the death zone where we set acceleration = 0. This death zone is very useful because otherwise, the correction of the speed will never end. This happens because of the noise in our accelerometer sensor and because we aren’t able to set any time in the duty cycle. This death zone goes from a little more of Vref. to a little less than it.

If we had a very good reading of the acceleration (that is not our case), we could find also an approximation of the position by the equation:
\[ X = X_0 + V_0 \cdot t + \frac{1}{2} \cdot a \cdot t^2 \]  \hfill (31)

But in this case, the noise of our sensor and the fact that the minimum acceleration detected is +/- 2g, don't allow to put this position control in practice.

To view this subprogram can be found in the annexes, properly annotated, to make it easier for your understanding.

5.3.2 Closed loop proportional control of direction

It is more or less, the same as before. The basic function of this subprogram is to set the direction of our robot properly. To make that we used a closed loop proportional control, just as well as before. In this case, we always will set always the reference velocity to zero. Because when we want to go straight ahead, the acceleration and therefore the velocity, perpendicular to the movement of the robot must be zero. Like before, if the sensor was better, we could know the position all the time, and instead of correct the velocity, correct the position which could be more interesting.

To view this subprogram can be found in the annexes, properly annotated, to make it easier for your understanding.

5.3.3 Avoiding obstacles without speed and direction control

This program reads if there is any obstacle in the front, at 1m or less, if there is any obstacle, the two PWM are set at the same speed and goes to read again. If there is any obstacle in the front, the robot reads the sides ultrasonic sensors, then turn to the side that have a farther away obstacle, and continue with the reading.
6. Results

6.1. Final program algorithm

The next program is to make the robot go straight ahead. To understand better the final program, we can see how it works in the following block diagram:

Fig. 6.1 Final program algorithm.
To view the final program can be found in the annexes, properly annotated, to make it easier for your understanding.

Therefore, the program to use for the next application of this multi sensor modular platform for robot as a smell seeker, it can be understood by the following diagram:

Since the purpose of this project was not to make a robot able follow scents, but a multi-sensor modular platform for robots for future robotics applications, and designed especially for one of these future applications that can be smell seeker, the first programs able to read the signal of the smell sensors was not done.
6.2. Electronic part of the robot

After all the simulations of our program, we made the model of our robot. This uses only a few pins of the microcontroller to allow adding new sensors, inputs and outputs for future projects or applications with this platform.

Fig. 6.3 Electronic Scheme of the Robot.
6.3. Building the model

This robot is made with a wooden platform where the subjection of the wheels has been bolted. Above, the sensors are strapped with a structure made of Lego. The robot is prepared to work with a small battery of 12V.

Fig. 6.4 Photos of the Multi Sensor Modular Platforms for Robots
6.4. Simulations

All the programs that we made, were compiled without problems and were simulated also to see if these programs worked well before try to make them work in the dsPIC. The windows that we normally had for these kind of simulations are like Fig 6.5. Often, we changed the timers and the value of different variables to appreciate the running of the program step by step without have to wait too much.

![Simulation window of one of test programs to read ultrasound sensor](image)

**Fig. 6.5** Simulation window of one of test programs to read ultrasound sensor

6.5. Tests

To makes the different tests, we used different equipment of the electronic laboratory. As we can see in the Fig 6.6. the main equipment was:
Power voltage source

With this power source, you can set the voltage that you want and also the intensity. This power source have three channels, so we used to simulate analog inputs like sensor signals, and also as a power supply for the dsPic, sensors and the engines.

Tester

Used to check the resistance, voltage and intensity. So useful to see which are the responses of the sensors and check the logic pins levels of dsPic.

Oscilloscope

Very useful in our case to check not only the analog signals of the accelerometer, but also to see the PWM in the screen and check that the period was correct.

Fig. 6.6 Testing one of test programs to read ultrasound sensor.
7. Conclusions

My conclusions after this project is that it has achieved the objectives that we set from the start, as well as not only programs developed are working without problems, but also the multi sensor modular platform for robots created is functional and useful for future projects. I’m sure that it will serve for future practices in the field of automatic or electronic systems, and also give a good response in case that somebody want to make the robot capable of tracking scents. Since this project has developed the basic programs for move the robot, the next generations of students will only have to implement the reading of the sensors they want to make the measurement such as light, sound, smell, white lines, etc and deduce the direction to follow, everything else is done. This will greatly facilitate the research in this kind of robots and it will allow further research in this area.

About the technical aspect, my conclusions are that the hardware has responded to the demands of this project and can meet future demands of bachelor or master projects. Maybe it would be nice to create a filter for the accelerometer sensor in the way to clean a little bit the signal and have a better measurement. Another future work that can be very useful is to make a PID to delete the small error in the speed of the engines.

But if this multi sensor modular platform for robots is used in high level applications like doctorate projects or high level research, my recommendation for future is to complement accelerometer readings with angular rate sensor. In this way it will have a more accurate reading of the speed and will be able to set it better. Also change the dsPic30f4013 for some dsPIC33F family as these have more special pins and routines specialized in the treatment and generation of PWM. If you want to have an absolute reference of position of the robot, it would be advisable to incorporate some sort of GPS sensor, although the implementation of this sensor would increase substantially the cost of the model.

On a personal level, this project has given me the opportunity to enter into the world of robotics and supply my theoretical knowledge with practical knowledge in the electronic and automatic systems field. It gave me also knowledge about how to program dsPic, and how to use MPLab software which is one of the most used programming software. So it allowed me to delve into the different families as well as learning how microprocessors works and their features, also it allowed me to gain
knowledge of avant-garde sensors like are accelerometers. All this knowledge will be very useful on the future working life

Finally, and for ending, this project has allowed me to participate in a larger project, which will give rise to other students to continue the work I've begun to result in better and more amazing robots. To my great satisfaction has been a great project to join.
Acknowledgments

First and foremost I want to thankful to the University of Gävle for allowing me to make my final thesis into the ERASMUS program, especially my tutor Mr. Niclas Björsell who has always been willing to answer any questions. Without his guidance this thesis would not have been possible.

I want to thank also Mr. Niclas Rothpfeffer for his help and technical support in the electronic field.

And finally to my family for they support in the distance, they gave me encouragement even when they were far away from me.
8. Bibliography


[4] Israel A. Wagner, *Smell as a Computational Resource- A Lesson We Can Learn from the Ant*, Department of Computer Science, Technion and IBM Haifa Research Lab, Matam, Israel.


http://www.foroconsultivo.org.mx/innovacion.gaceta/component/content/article/84-innovadores/113-nariz-electronica-para-detectar-fugas-de-gas


vanguardista, Spanish electronic magazine: ISSN 0482-6396, N° 575, 2002, pages 70-74


9. Appendix

9.1. Code of for initialize : init_dspIC30F4013.s :

; This program has to be attached into Source Files of every program we want to test.

; Initialization Code for dsPIC30F4013, Family: controller control, Package: 40-Pin PDIP 40pins

.include "p30F4013.inc"

; Filter Controls used to Generate Code:

; POR Match Filter OFF

; Provisioned Feature Filter ON

; Masks are Ignored and uses UnMasked Register Writes

.GLOBAL _VisualInitialization

; Feature=fuses - fuses (DCR) configuration

; B15:14=FSCKM1:0 B10:8=FOS2:0 B4:0=FPR4:0

    config __FOSC, 0xC100

; B15=FWDTEN B5:4=FWPSA1:0 B3:0=FWPSB3:0

    config __FWDT, 0x803F

; B15=MCLREN B7=BOREN B5:4=BORV1:0 B3:0=FPWRT3:0

    config __FBORPOR, 0x87B3

.text

_VisualInitialization:

; Feature=Interrupts - Disable Interrupts during configuration
; clear int flags:
; B15=CN B14=BCL B13=I2C B12=NVM B11=AD B10=U1TX B9=U1RX B8=SPI1
; B7=T3 B6=T2 B5=OC2 B4=IC2 B3=T1 B2=OC1 B1=IC1 B0=INT0

    CLR IFS0
; B15:12=IC6:3 B11=C1 B10=SPI2 B9=U2TX B8=U2RX
; B7=INT2 B6=T5 B5=T4 B4=OC4 B3=OC3 B2=IC8 B1=IC7 B0=INT1

    CLR IFS1
; B12=FLTB B11=FLTA B10=LVD B9=DCI B8=QEI
; B7=PWM B6=C2 B5=INT4 B4=INT3 B3:0=OC8:5

    CLR IFS2
    CLR IEC0
    CLR IEC1
    CLR IEC2

; Feature=Reset - Reset configuration
; B15=TRAPR B14=IOPWR B13=BGST B12=LVDEN B11:8=LVDL3:0
; B7=EXTR B6=SWR B5=SWDTEN B4=WDTO B3=SLEEP B2=IDLE B1=BOR B0=POR

    MOV #0x0503, W0
    MOV W0, RCON

; Feature=NVM - NVM configuration - not implemented

; Feature=Oscillator - Oscillator configuration

; method to override OSCCON write protect
CLR.B W0
MOV.B #0x78, W1
MOV.B #0x9A, W2
MOV.W #OSCCON, W3
MOV.B W1, [W3+1]
MOV.B W2, [W3+1]
MOV.B W0, [W3+1]

CLR.B W0
MOV.B #0x46, W1
MOV.B #0x57, W2
MOV.B W1, [W3+0]
MOV.B W2, [W3+0]
MOV.B W0, [W3+0]

; Feature=A2D - A2D configuration
; force all A2D ports to digital IO at first
MOV #0xFFFF, W0
MOV W0, ADPCFG

; Feature=A2D - A2D configuration
; Turn off A2D before setting registers
CLR ADCON1
; B15:0=CSSL15:0
    MOV #0x0001, W0
    MOV W0, ADCSSL
; B15:14=CH123NB1:0 B13=CH123SB B12=CH0NB B11:8=CH0SB3:0
; B7:6=CH123NA1:0 B5=CH123SA B4=CH0NA B3:0=CH0SA3:0
    CLR ADCHS
; B15:0=PCFG15:0
    CLR ADPCFG
; B12:8=SAMC4:0 B7=ADRC B5:0=ADCS5:0
    MOV #0x001D, W0
    MOV W0, ADCON3
; B15:13=VCFG2:0 B12=OFFCAL B10=CSCNA B9:8=CHPS1:0
; B7=BUFS B5:2=SMPI B1=BUFM B0=ALTS
    CLR ADCON2
; B15=ADON B13=ADSIDL B12=ADSTBY B9:8=FORM
; B7:5=SSRC B3=SIMSAM B2=ASAM B1=SAMP B0=CONV
    MOV #0x80E4, W0
    MOV W0, ADCON1
; Feature=required - Interrupt flags cleared and interrupt configuration
; interrupt priorities IP
; B14:12=T1 B10:8=OC1 B6:4=IC1 B2:0=INTO
MOV #0x4444, W0
MOV W0, IPC0
; B14:12=T3 B10:8=T2 B6:4=OC2 B2:0=IC2
MOV #0x4444, W0
MOV W0, IPC1
; B14:12=AD B10:8=U1TX B6:4=U1RX B2:0=SPI1
MOV #0x4444, W0
MOV W0, IPC2
; B14:12=CN B10:8= BCL B6:4=I2C B2:0=NVM
MOV #0x4444, W0
MOV W0, IPC3
; B14:12=OC3 B10:8=IC8 B6:4=IC7 B2:0=INT1
MOV #0x4444, W0
MOV W0, IPC4
; B14:12=INT2 B10:8=T5 B6:4=T4 B2:0=OC4
MOV #0x4444, W0
MOV W0, IPC5
; B14:12=C1 B10:8=SPI2 B6:4=U2TX B2:0=U2RX
MOV #0x4444, W0
MOV W0, IPC6
; B14:12=IC6 B10:8=IC5 B6:4=IC4 B2:0=IC3
; B14:12=OC8 B10:8=OC7 B6:4=OC6 B2:0=OC5
; B14:12=PWM B10:8=C2 B6:4=INT4 B2:0=INT3
; B14:12=FLTA B10:8=LVD B6:4=DCI B2:0=QEI
    MOV #0x4444, W0
    MOV W0, IPC10
; external interrupt enables
; B15=NSTDIS B10=OVATE B9=OVBT E B8=COVTE
; B4=MATHERR B3=ADDRERR B2=STKERR B1=OSCFAIL
    CLR INTCON1
; B15=ALTVT B4:0=INTnEP4:0
    CLR INTCON2
; Feature=CPU - CPU register configuration
    CLR SR
    CLR SR
    CLR W0
    CLR W1
    CLR W2
; Feature=Interrupts - enable interrupts
; feature interrupt enables IE
; B15=CN B14=BCL B13=I2C B12=NVM B11=AD B10=U1TX B9=U1RX B8=SPI1
; B7=T3 B6=T2 B5=OC2 B4=IC2 B3=T1 B2=OC1 B1=IC1 B0=INT0
    MOV #0x0800, W0
MOV W0, IEC0

; B15:12=IC6:3 B11=C1 B10=SPI2 B9=U2TX B8=U2RX
; B7=INT2 B6=T5 B5=T4 B4=OC4 B3=OC3 B2=IC8 B1=IC7 B0=INT1

CLR IEC1
; B12=FLTB B11=FLTA B10=LVD B9=DCI B8=QEI
; B7=PWM B6=C2 B5=INT4 B4=INT3 B3:0=OC8:5

CLR IEC2
return

9.2. Creating two PWM

// This program do PWM in OC1 and OC2 which // period is 20ms.
// Depend of the input of Port F is 0-3, the duty cycle // will change.
#include "p30f4013.h"

void InicializaOC1(void)
{
    // Internal clock of 8 Mhz
    OC1RS = 0; // Initializing registers OC1
    OC1R = 0;
    OC1CONbits.OCM = 0x0006; // Mode simple PWM without fail pin
    T2CONbits.TCKPS = 0x0002; // Prescaler 1:64
    T2CONbits.TON = 1; // Turn On
    OC2RS = 0; // Initializing registers OC2
OC1R = 0;
OC2CONbits.OCM = 0x0006;

} //Main Program

int main (void)
{
    InicializaOC1();
    TRISF=0x0003;       // The first two bits of Port F are inputs
    ADPCFG=0;            // Port F digital signals
    PR2=0x00271;        // Period of PWM
    while(1)
    {
        asm("clrwdt");    //We wait in an infinit loop
        if (PORTF == 0){
            OC1RS = 0x00250;
            OC2RS = 0x00170;
        }
        else if (PORTF == 1){
            OC1RS = 0x00250;
            OC2RS = 0x00190;
        }
        else if (PORTF == 2){
            OC1RS = 0x0250;
        }
9.3. Program for reading an analog signal:

#include "p30f4013.h"
#include "libpic30.h"

int main (void)
{
    long int n;
    // Set up which pins are which
    configure_pins();
    while(1)
    {
        // Read the analog channel. As the reference voltage // is the supply voltage, the
        // result is an
        // integer between 0 and 4095 inclusive.
        n = read_analog_channel(0); // In this case we read AN0
PORTDbits.RD0 = 1; // Just to see that the program is working

if (n<500) {PORTDbits.RD1 = 1;}
else if (n>500 & n<1500) {
    PORTDbits.RD2 = 1;
    PORTDbits.RD3 = 0;
    PORTDbits.RD8 = 0;
    PORTDbits.RD9 = 0;
}
else if (n>1500 & n<2500) {PORTDbits.RD2 = 1;
    PORTDbits.RD3 = 1;
    PORTDbits.RD8 = 0;
    PORTDbits.RD9 = 0;
}
else if (n>2500 & n<3500) {PORTDbits.RD2 = 1;
    PORTDbits.RD3 = 1;
    PORTDbits.RD8 = 1;
    PORTDbits.RD9 = 0;
}
else if (n>3500 & n<4500) {PORTDbits.RD2 = 1;
    PORTDbits.RD3 = 1;
    PORTDbits.RD8 = 1;
    PORTDbits.RD9 = 1;
}
else {
}

return 0;
void configure_pins()
{
    // Configure analog inputs
    TRISD = 0x0000;  // Port D all outputs
    TRISB = 0x01FF;  // Port B all inputs
    ADPCFG = 0xFF00; // Lowest 8 PORTB pins are analog inputs

    ADCON1 = 0;       // Manually clear SAMP to end sampling, start conversion
    ADCON2 = 0;       // Voltage reference from AVDD and AVSS
    ADCON3 = 0x0005;  // Manual Sample
    ADCON1bits.ADON = 1;  // Turn ADC ON

}

// This function reads a single sample from the specified analog input.
unsigned int read_analog_channel(int channel)
{
    ADCHS = channel;  // Select the requested channel
    ADCON1bits.SAMP = 1; // start sampling
    __delay32(30);    // 30 MIPS
    ADCON1bits.SAMP = 0;  // start Converting
    while (!ADCON1bits.DONE);
    return ADCBUF0;
9.4. Code of reading Ultrasonic sensor:

//This program reads the signal of a ultrasonic sensor and depending of the measure, some led turn on.

#include "p30f4013.h"

void InicializaOC1(void)
{
    TRISD = 0x0000;           //Port D all outputs
    TRISF = 0x0001;           //Port F All outputs but RF0 input
    ADPCFG=0;                 // Port F digital signals
    T3CONbits.TCKPS = 0x0002;  //Prescaler 1:64
    T3CONbits.TON=1;          //Turn on timer 3
}

//Main Program

int main (void)
{
    InicializaOC1();
    PORTDbits.RD0=1;
    int t;
}
t=0;
while(1){ //Infinite Loop
    TMR3=0; //Input pulse of 100 us
    PORTFbits.RF1=1; //Input pulse of 100 us(low-high)
    while (TMR3<4){;} //Input pulse of 100 us (high)
    PORTFbits.RF1=0; //Input pulse of 100 us (high-low)
    while (PORTFbits.RF0 == 0){;} //Wait to detect high level
    TMR3=0; //Start measuring the length of the pulse
    while (PORTFbits.RF0 == 1){;} //Finish the measuring of the length
    t=TMR3;
    if (t>0x0000 & t<0x0032){
        PORTDbits.RD1 =1;
        PORTDbits.RD2 =0;
        PORTDbits.RD3 =0;
        PORTDbits.RD8 =0;
    }else if (t>0x0032 & t<0x0064){
        PORTDbits.RD1 =0;
        PORTDbits.RD2 =1;
        PORTDbits.RD3 =0;
        PORTDbits.RD8 =0;
    }else if (t>0x0064 & t<0x096){
        PORTDbits.RD1 =0;
PORTDbits.RD2 =0;
PORTDbits.RD3 =1;
PORTDbits.RD8 =0;}
else if (t>0x096 & t<0xFFFF){
PORTDbits.RD1 =0;
PORTDbits.RD2 =0;
PORTDbits.RD3 =0;
PORTDbits.RD8 =1;}

if (TMR<0x061A){
while (TMR3<0x061A){;}//Wait until 50 ms has past from the start of the //lecture, in order to stabilize the module
}
else {;}
}

9.5. Program to have constant speed and direction

#include "p30f4013.h"
#include "libpic30.h"

void InicializaOC1(void)
{  
    //Internal clock of 8 Mhz
    OC1RS = 0;  //Initializing registers OC1
    OC1R = 0;
    OC2RS = 0;  //Initializing registers OC2
    OC1R = 0;
    OC1CONbits.OCM = 0x0006;  //Mode simple PWM without fail pin
    T2CONbits.TCKPS = 0x0002;  //Prescaler 1:64
    T2CONbits.TON=1;  //Turn On Timer 2
    T1CONbits.TCKPS = 0x0002;  //Prescaler 1:64
    T1CONbits.TON=1;  //Turn On Timer 1
    OC2CONbits.OCM = 0x0006;
    TMR1=0;
    TMR2=0;
}

int main (void)  //Initializing variables
{
    long int n0;
    float v00;
    float v10;
    float t0;
float t1;
float vref0;
float a0;
long int n1;
float v01;
float v11;
float vref1;
float a1;
int PWM1;
int PWM2;
int increment0;
int increment1;
float Kp0;
float Kp1;
float diff0;
float diff1;
PWM1=0;
PWM2=0;
diff0=0;
diff1=0;
a0=0;
n0=0;
n1=0;
v00=0;
v10=0;
t1=0;
t0=0;
vref0=10;  //Speed straight ahead in cm/clock timer 1
a1=0;
v01=0;
v11=0;
vref1=0;  //directional speed in cm/clock timer 1
InicializaOC1();

TRISF=0x0001;  //Los primeros dos bits de port F son entradas
ADPCFG=0;     // Port F señales digitales
PR2=0x00271;   // Period of PWM 20 ms
Kp0=5;
Kp1=2;
configure_pins();  // Set up which pins are which

while(1)
PORTDbits.RD0 = 1;  //Just to know that the program works

//Read the analog channel. The result is an Integer between 0 and 4095 inclusive.

n0 = read_analog_channel(0);  //Reading the accelerometer in AN0
n1 = read_analog_channel(1);  //Reading the accelerometer in AN0
n0 = n0-1228;  //Subtract the reference voltage (1.5 V / 0.001221V/value)
n1 = n1-1228;  //Subtract the reference voltage (1.5 V / 0.001221V/value)

a0 = (n0*344.0)/980.0;  //Acceleration in cm/s^2
a1 = (n1*344.0)/980.0;  //Acceleration in cm/s^2

TMR1/1355.0 ;  //Take the time that our robot had this acceleration, and pass it to seconds

TMR1=0;  //Reset Timer

v10= v00 + a0*t1;  //Find the new speed straight ahead
v11= v01 + a1*t1;  //Find the new speed straight ahead

v00=v10;  //This new speed will be the next initial speed in the next loop (v0)
v01=v11;  //This new speed will be the next initial speed in the next loop (v1)

diff0=vref0-v10;  //Speed error
diff1=vref1-v11;  //Direction error

if (diff1<1 & diff1>-1){  //If the direction is right
(Dead zone)
if (diff0<1 & diff0>-1) {;}

//If the speed is right (Dead zone)

else {  //Proporcional Speed Control

    if (v10<vref0){
        increment0 = (vref0-v10)*Kp0;
        PWM1= PWM1+increment0;
        PWM2= PWM2+increment0;
        OC1RS = PWM1;
        OC2RS = PWM2;
    }
    else {
        increment0 = (v10-vref0)*Kp0;
        PWM1= PWM1-increment0;
        PWM2= PWM2-increment0;
        OC1RS = PWM1;
        OC2RS = PWM2;
    }
}

else{  //Proporcional Direction Control

    if (v11<vref1){
        increment1 = (vref1-v11)*Kp1;
        PWM1= PWM1+increment1;
    }
}
PWM2 = PWM1 - increment1;
OC1RS = PWM1;
OC2RS = PWM2;
}
else {
increment1 = (v11 - vref1) * Kp1;
PWM1 = PWM1 - increment1;
PWM2 = PWM1 + increment1;
OC1RS = PWM1;
OC2RS = PWM2;
}
return 0;
}

void configure_pins()
{

  // Configure analog inputs
  TRISD = 0x0000;  // Port D all outputs
  TRISB = 0x01FF;  // Port B all inputs
  ADPCFG = 0xFF00;  // Lowest 8 PORTB pins are analog inputs
ADCON1 = 0;  // Manually clear SAMP to end sampling, start /
ADCON2 = 0;  // Voltage reference from AVDD and AVSS
ADCON3 = 0x0005;  // Manual Sample,
ADCON1bits.ADON = 1;  // Turn ADC ON

}  // This function reads a single sample from the //specified
// analog input.

unsigned int read_analog_channel(int channel)
{
    ADCHS = channel;  // Select the requested channel
    ADCON1bits.SAMP = 1;  // start sampling

    __delay32(30);  // 30 MIPS
    ADCON1bits.SAMP = 0;  // start Converting

    while (!ADCON1bits.DONE);
    return ADCBUF0;
}

9.6. Program to avoid obstacles but without the speed and direction control

#include "p30f4013.h"

void InicializaOC1(void)


```
{
  TRISD = 0x0000;  // Port D all outputs
  TRISF = 0x0011;  // Port F all outputs but RF0, RF1, RF2 input
  ADPCFG=0;       // Port F señales digitales

  T3CONbits.TCKPS = 0x0002;  // Prescaler 1:64
  T3CONbits.TON=1;           // Turn on timer 3
}
void InicializaOC2(void)
{
  OC1RS = 0;                  // Setting OC1 registers
  OC1R = 0;
  OC1CONbits.OCM = 0x0006;    // Mode simple PWM without fail pin
  T2CONbits.TCKPS = 0x0002;   // Prescaler 1:64
  T2CONbits.TON=1;           // Turn on timer 2
  OC2RS = 0;
  OC1R = 0;
  OC2CONbits.OCM = 0x0006;    // Setting OC1 registers
}
int FrontLecture (void)
{
  int t;
```
int Obstacle;
t=0;
TMR3=0;
PORTFbits.RF1=1;   //Turning on trigger Pulse
while (TMR3<4){;}   //Waiting 100us
PORTFbits.RF1=0;    //Turning off trigger pulse
while (PORTFbits.RF0 == 0){;} //Reading front
TMR3=0;            //Reading front
while (PORTFbits.RF0 == 1){;} // Reading front

t=TMR3;   // Value front
if (t>0x0000 & t<0x0160){ Obstacle = 1;}   //There is an obstacle in one
  meter or less
else {Obstacle = 0;}    //There is not any obstacle in one meter or
  more.

if (TMR3<0x061A){        //Waiting for doing the next lecture
  while (TMR3<0x061A){;}   //Espera 50 ms
}
else {}
return (Obstacle);

int SidesLecture (void)
{
    int tR;
int tL;
int Sides;
tR=0;
tL=0;
TMR3=0;
PORTFbits.RF1=1; //Turning on trigger Pulse
while (TMR3<4){;} //Waiting 100us
PORTFbits.RF1=0;  //Turning off trigger pulse
while (PORTFbits.RF4 == 0){;} //Reading right
TMR3=0;           //Reading right
while (PORTFbits.RF4 == 1){;} //Reading right
tR=TMR3;          // Value right
TMR3=0;
PORTFbits.RF1=1; //Turning on trigger Pulse
while (TMR3<4){;} //Waiting 100us
PORTFbits.RF1=0; //Turning off trigger pulse
while (PORTFbits.RF5 == 0){;} //Reading left
TMR3=0;          //Reading left
while (PORTFbits.RF5 == 1){;} //Reading left
tL=TMR3;         // Value left
if (tR>tL){Sides=0;}
else {Sides=1;}

if (TMR3<0x061A) {
    //Waiting for doing the next lecture
    while (TMR3<0x061A);  
    //Espera 50 ms
}

else {
    return(Sides);
}

//Programa principal

int main (void)
{
    int obstacle;
    int side;
    InicializaOC1();
    InicializaOC2();
    obstacle = 0;
    side = 0;
    PR2=0x00271;
    while(1){
        PORTDbits.RD8 = 1;
        obstacle= FrontLecture();
        if (obstacle == 1){
            side= SidesLecture();
if (side=0){
    OC1RS=0x00100;
    OC2RS=0x00000;
}
else {
    OC1RS=0x00000;
    OC2RS=0x00100;
}
else {
    PORTDbits.RD0= 1;
    OC1RS= 0x00150;
    OC2RS= 0x00150;
}

9.7. Final program:

#include "libpic30.h"
#include "p30f4013.h"

void InicializaOC1(void)
{
    TRISD = 0x0000;  //Port D all outputs
TRISF = 0x0011; // Port F all outputs but RF0, RF1, RF2 input
ADPCFG = 0; // Port F digital signals

T3CONbits.TCKPS = 0x0002; // Prescaler 1:64
T3CONbits.TON = 1; // Turn on timer 3

void InicializaOC2(void)
{
    // Internal clock of 8 Mhz
    OC1RS = 0; // Initializing registers OC1
    OC1R = 0;
    OC2RS = 0; // Initializing registers OC2
    OC1R = 0;
    OC1CONbits.OCM = 0x0006; // Mode simple PWM without fail pin
    T2CONbits.TCKPS = 0x0002; // Prescaler 1:64
    T2CONbits.TON = 1; // Turn On Timer 2
    T1CONbits.TCKPS = 0x0002; // Prescaler 1:64
    T1CONbits.TON = 1; // Turn On Timer 1
    OC2CONbits.OCM = 0x0006;
    TMR1 = 0;
    TMR2 = 0;
int FrontLecture (void)
{
    int t;
    int Obstacle;
    t=0;
    TMR3=0;
    PORTFbits.RF1=1; //Turning on trigger Pulse
    while (TMR3<4){;} //Waiting 100us
    PORTFbits.RF1=0; //Turning off trigger pulse
    while (PORTFbits.RF0 == 0){;} //Reading front
    TMR3=0; //Reading front
    while (PORTFbits.RF0 == 1){;} // Reading front
    t=TMR3; // Value front
    if (t>0x0000 & t<0x0160){ Obstacle = 1;} //There is an obstacle in one meter or less
    else {Obstacle = 0;} //There is not any obstacle in one meter or more.
    if (TMR3<0x061A){ //Waiting for doing the next lecture
        while (TMR3<0x061A){;} //Wait 50 ms
    }
    else {;}
}
return (Obstacle);

}

int SidesLecture (void)
{
    int tR;
    int tL;
    int Sides;
    tR=0;
    tL=0;
    TMR3=0;
    PORTFbits.RF1=1; //Turning on trigger Pulse
    while (TMR3<4){;}   //Waiting 100us
    PORTFbits.RF1=0; //Turning off trigger pulse
    while (PORTFbits.RF4 == 0){;}   //Reading right
    TMR3=0; //Reading right
    while (PORTFbits.RF4 == 1){;}   //Reading right
    tR=TMR3; // Value right
    TMR3=0;
    PORTFbits.RF1=1; //Turning on trigger Pulse
    while (TMR3<4){;}   //Waiting 100us
    PORTFbits.RF1=0; //Turning off trigger pulse
    while (PORTFbits.RF5 == 0){;}   //Reading left
TMR3=0;       //Reading left
while (PORTFbits.RF5 == 1){;}   //Reading left

TL=TMR3;       // Value left
if (tR>tl){Sides=0;}
else {Sides=1;}

if (TMR3<0x061A){       //Waiting for doing the next lecture
            while (TMR3<0x061A){}  //Espera 50 ms
        }
else {};

return(Sides);

}

void configure_pins()
{

}       // Configure analog inputs

ADCON1 = 0;       // Manually clear SAMP to end sampling, start / conversion
ADCON2 = 0;       // Voltage reference from AVDD and AVSS
ADCON3 = 0x0005;       // Manual Sample,
ADCON1bits.ADON = 1;       // Turn ADC ON
}
This function reads a single sample from the specified analog input.

```c
unsigned int read_analog_channel(int channel)
{
    ADCHS = channel; // Select the requested channel
    ADCON1bits.SAMP = 1; // start sampling
    __delay32(30); // 30 MIPS
    ADCON1bits.SAMP = 0; // start Converting
    while (!ADCON1bits.DONE);
    return ADCBUF0;
}
```

Main program

```c
int main (void)
{
    int obstacle;
    int side;
    long int n0;
    float v00;
    float v10;
```
float t0;
float t1;
float vref0;
float a0;
long int n1;
float v01;
float v11;
float vref1;
float a1;
int PWM1;
int PWM2;
int increment0;
int increment1;
float Kp0;
float Kp1;
float diff0;
float diff1;
PWM1=0;
PWM2=0;
diff0=0;
diff1=0;
a0=0;
n0=0;
n1=0;
v00=0;
v10=0;
t1=0;
t0=0;
vref0=10;  // Speed straight ahead in cm/s
v01=0;
v11=0;
vref1=0;  // Directional speed in cm/s
Kp0=5;
Kp1=2;
configure_pins();
InicializaOC1();
InicializaOC2();
obstacle = 0;
side = 0;
PR2=0x00271;
PORTDbits.RD8 = 1;  // Just to know that the program works

while(1){
PORTDbits.RD8 = 1;

obstacle = FrontLecture();

if (obstacle == 1) {  // There is an obstacle in 1 m or less
    side = SidesLecture();  // Look which side is the best to turn
    if (side == 0) {  // Turn right
        OC1RS = 0x00100;
        OC2RS = 0x00000;
    } else {  // Turn left
        OC1RS = 0x00000;
        OC2RS = 0x00100;
    }
} else {  // Go straight ahead with speed and direction control
    n0 = read_analog_channel(0);  // Reading the accelerometer in AN0
    n1 = read_analog_channel(1);  // Reading the accelerometer in AN0
    n0 = n0 - 1228;  // Subtract the reference voltage (1.5 V / 0.001221V/value)
    n1 = n1 - 1228;  // Subtract the reference voltage (1.5 V / 0.001221V/value)
a0 = (n0*344.0)/980.0; //Acceleration in cm/s^2
a1 = (n1*344.0)/980.0; //Acceleration in cm/s^2

\[ t_1 = \frac{TMR1}{1355.0} \]  //Take the time that our robot had this acceleration, //and pass it to seconds

TMR1=0;  //Reset Timer

v10= v00 + a0*t1;  //Find the new speed straight ahead
v11= v01 + a1*t1;  //Find the new speed straight ahead

v00=v10;  //This new speed will be the next initial speed in the //next loop (v0)

v01=v11;  //This new speed will be the next initial speed in the //next loop (v1)

diff0=vref0-v10;  //Speed error
diff1=vref1-v11;  //Direction error

if (diff1<1 & diff1>-1){  //If the direction is right (Dead zone)
    if (diff0<1 & diff0>-1) {}  //If the speed is right (Dead zone)
else {
    //Proporcional Control
    if (v10<vref0){
        increment0 = (vref0-v10)*Kp0;
        PWM1= PWM1+increment0;
        PWM2= PWM2+increment0;
        OC1RS = PWM1;
        OC2RS = PWM2;
    }
}
else {
    increment0 = (v10-vref0)*Kp0;
    PWM1 = PWM1-increment0;
    PWM2 = PWM2-increment0;
    OC1RS = PWM1;
    OC2RS = PWM2;
}
}
else{
    if (v11<vref1){
        increment1 = (vref1-v11)*Kp1;
        PWM1 = PWM1+increment1;
        PWM2 = PWM1-increment1;
        OC1RS = PWM1;
        OC2RS = PWM2;
    }
    else {
        increment1 = (v11-vref1)*Kp1;
        PWM1 = PWM1-increment1;
        PWM2 = PWM1+increment1;
        OC1RS = PWM1;
    }
}
OC2RS = PWM2;