MOWAY'S
USER MANUAL

Index

1. Prologue

The dawning of a new era; the era of the minirobots. Increasingly more mobile robotics applications enter our daily life. We can currently find robots which help us with simple tasks like cleaning household floors, mowing the lawn or keeping the swimming pool clean. As technology keeps improving, these small devices which blend mechanics, electronics and software are performing more and more complex tasks*. They are slowly introducing themselves into our lives in a useful manner and reducing the burden of unpleasant jobs.

It's not too far-fetched to think that the revolution which took place in the IT or telecommunications fields will be repeated with robotics in the next decade. Enough technology is currently available to manufacture these devices and society is also ready to receive them in the market. Yet, a specific catalyst is needed to start this revolution. People also need to be ready and prepared to identify in what fields microrobotics may have an opportunity and which new applications may be interesting to implement.

Up till now processors weren't able to move. But today things have changed. Software is one of the fundamental elements in the world of mobile robotics. The main difference between developing a program for these robots and running it with a personal computer is interaction with the environment. The environment isn't changing randomly in PC applications, so decision making and programming are simplified. On the other hand, when running commands for a minirobot application usually the result is unknown, therefore algorithms have to consider situations with a wider range of possibilities, some of them unexpected.

The mOway robots are tools specifically designed for teaching and research. Their purpose is to bring the world of autonomous robots closer to the teaching centers.

mOway's main purpose is to be a useful tool for those who are being introduced for the first time to the world of the minirobots as well as for those who are already experienced and wish to perform complex collaborative robotic applications.

mOway aims to stimulate enthusiasm for this new and exciting branch of engineering in a prompt and enjoyable way through the practical exercises included in this manual.

- An easy and entertaining way to learn.
- This book's purpose: to be mOway's Manual and not a comprehensive book on minirobotics.

This manual has been implemented to assist learning how to use mOway. It provides some basic notions on using mOway and its functions in a quick and clear manner.

This manual is divided in two parts. The first part includes a description of the elements which form part of the robot and their functioning. The second part of the manual includes a series of practical exercises that can be executed with mOway.

2. What is mOway?

mOway is an autonomous programmable small robot designed mainly to perform practical minirobotics applications.

It provides a perfect hardware platform for those wishing to take their first steps within the world of mobile robots as well as for those who have already worked with minirobots and want to develop more complex applications.

The mOway robot is equipped with a series of sensors which aid it to move in a real environment. It also includes a drive unit which allows it to move over smooth terrain commanded by a I2C communications bus. All these peripherals are connected to a microcontroller responsible for governing the robot.

This small robot incorporates I2C/SPI expansion bus options. As an example, a wireless communications module, a video camera or a prototype card can be connected to it as well as any other device considered interesting to perform a certain task.

mOway's external design is very compact, intended to move with grace and style avoiding standstills due to obstacles or corners. This small mobile device has been fittingly called a "pocket robot".

mOway is a perfect tool for those who want to both learn and teach minirobotics. The user will be pleasantly surprised by the speed in achieving results even if this is the first time he/she comes into contact with mobile robots.

3. Robot mOway

This chapter describes each of the parts that constitute the mOway. It is important to highlight that it is not necessary to know the total functioning of the robot to be able to program it, at least not at the level of detail explained here.

The following elements are to be found inside mOway:

- Processor
- Drive system
- Sensors and indicators group
- Power supply system
- An expansion connector

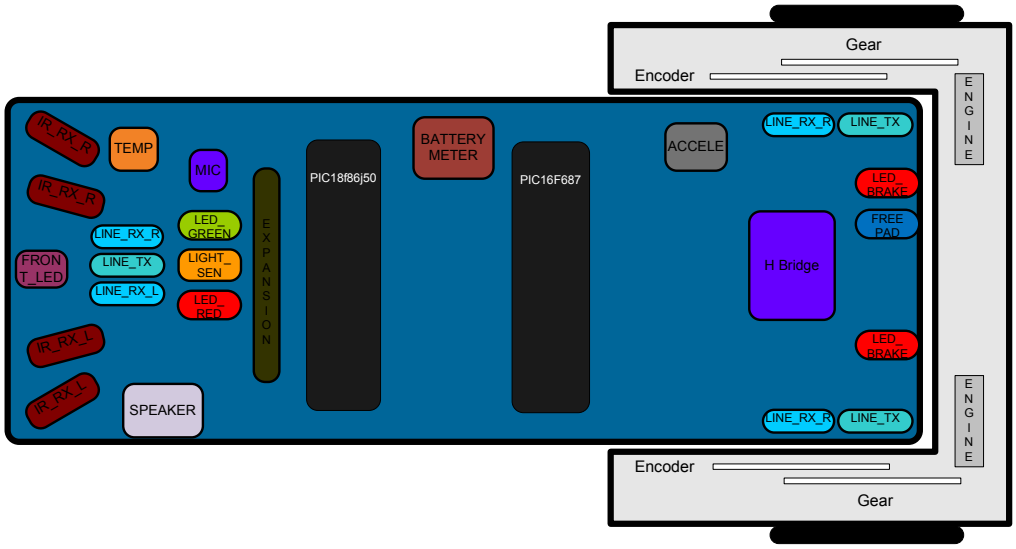


Image 1. Diagram of mOway's parts

3.1. Processor

mOways are governed by a 4 Mhz PIC18F87J50 microcontroller manufactured by Microchip Technologies. All the peripherals distributed throughout the whole robot are connected to its input/output ports. Some of them need a digital input or output, others need an analog input or output and others, instead, are controlled by one of the I2C/SPI communication buses. The table below describes how the microcontroller pins are distributed.

Table 1. PIC-sensors connections

Pin PIC	I/O	Sensor
PORTA		
RA0	I	Light
RA1	I	Central left infrared receiver
RA2	I	Right line sensor receiver
RA3	I	Side left infrared receiver
RA5	I	Left line sensor receiver
PORTB		
RB1	I	First interruption of the accelerometer
RB2	I	Second interruption of the accelerometer
RB3	O	Speaker
RB5	O	Top red LED
RB6	O	Top green LED
PORTC		
RC7	O	Front LED
PORTD		
RD1	O	Line sensors transmitter
RD4	I	SDO signal for the SPI communication (accelerometer)
RD5	O	SDI sinal for the SPI communication(accelerometer)
RD6	O	Clock sinal for the SPI communication(accelerometer)
RD7	I	Chip Select for the SPI communication(accelerometer)
PORTE		
RE5	O	Brake LED
PORTF		
RF5	I	Side right infrared receiver
RF6	I	Central right infrared receiver
PORTH		
RH5	I	Tempreature sensor
RH6	I	Battery measurer
RH7	I	Microphone
PORTJ		
RJ6	O	Infrared transmitter
RJ7	I/O	Free pad

3.2. Drive system

To be able to move the mOway uses a double servo-motor group. It includes both an electronic part and a mechanical one. The electronic part is mainly in charge of controlling the motor's speed and the mechanical part allow the mOway to move unhindered over different terrains with adequate power.

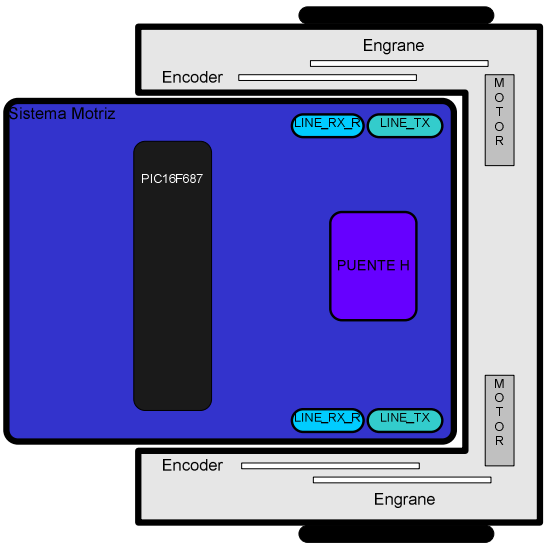


Image 2. Drive system: electronic and mechanical

The servo-motor group includes different features:

1. **Speed control**: controls the speed of each motor.
2. **Time control**: controls the time for each command with a 100 ms precision.
3. **Traveled distance control**: Controls the distance traveled by each command with a precision of 1 mm aprox.
4. **General speedometer**: counts distances traveled since the initial command.
5. **Angle control**: controls the angle when the mOway rotates.

The microcontroller sends the I2C command to the drive system that controls the motors and therefore releasing the main microcontroller so it can carry out other tasks.

Speed control is carried out by means of proportional control with negative feedback from the encoders' signal. The illustration displays the controlling system. The microcontroller feeds the motors through an H bridge controlled by pulse width modulation (PWM) signals. Wheel rotation is monitored by an encoding sticker and an infrared sensor. When the sticker shows its black segment, the logical output shall be 1 and when it shows the white sector the output shall be 0. The microcontroller analyzes these signals (it can determine the exact wheel speed by measuring the pulse width) and acts on the motors. This way, the mOway will be able to keep the speed constant on any surface.

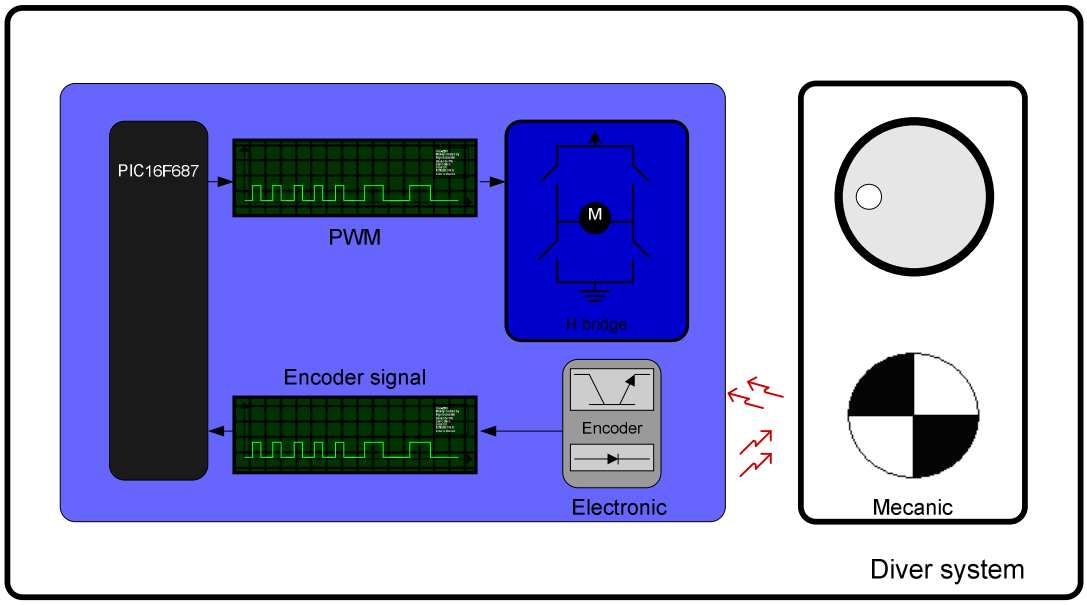


Image 3. Motor control

To send a movement command to the robot, via the main microcontroller, all we need to do is send the movement command parameters. To this end some libraries were designed in assembly and C language to simplify communications through some functions which are responsible for I2C communications. The format for these frames is explained in the motors and drive system library section.

The table below describes connections between the main PCB and the servo-motor unit.

Table 2. Processor - motor connections

Pin PIC	I/O	Sensor
PORTE		
RE0	I2C	I2C clock
RE1	I2C	I2C data
RE7	I	END_COMAND line

3.3. Sensor and indicators group

This group consists of different luminous sensors and indicators, connected to the mOway microprocessor, through which the robot interacts with the external world:

- Two line tracking sensors.
- Four obstacle detection sensors.
- A light sensor.
- An expansion connector.
- Four LED diodes.
- Temperature sensor.

- Speaker.
- Microphone.
- Accelerometer.
- Battery level.

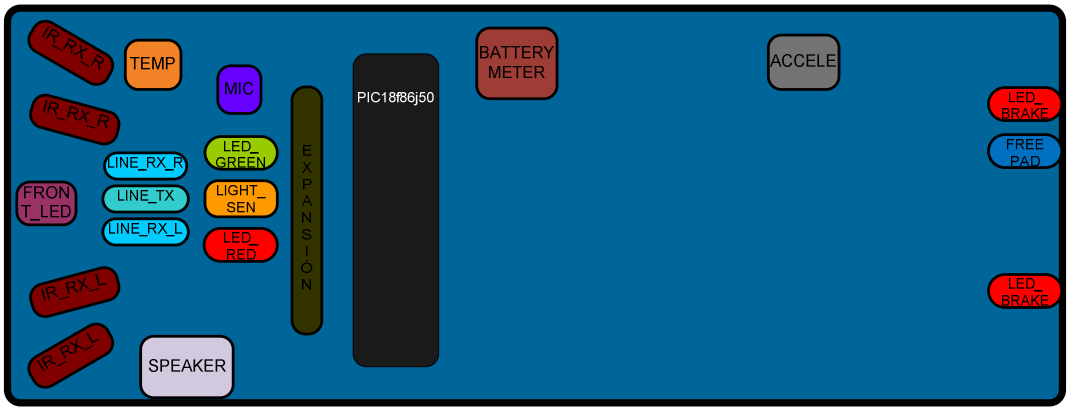


Image 4. Sensors and indicators group

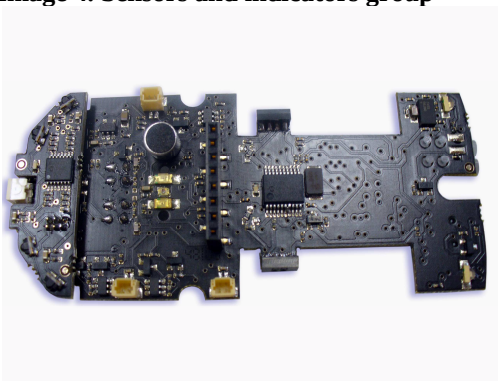


Image 5. Top-view of mOway´s PCB

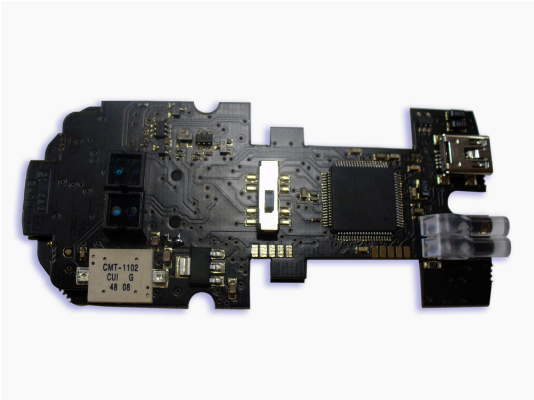


Image 6. Down-view of mOway´s PCB

3.3.1. Line sensors

The line tracking sensors are two reflection optocouplers mounted on the top front part of the robot. They use infrared light reflection to detect the color of the terrain at the point where the robot is.

These two sensors are connected to two microcontroller analog ports so strong terrain contrasts, like white lines on black backgrounds, can be detected. They are also capable of distinguishing different tones.

The Vishay CNY70 sensor has a compact construction where the emitting light source and the detector are arranged in the same direction to be able to detect by using the reflective IR beam the light reflected in the terrain.

In the images below the three possible cases can be observed:

1. **Clear surface**: A white surface reflects all the infrared light and therefore we obtain a low voltage reading at the transistor's output when in regular mode.

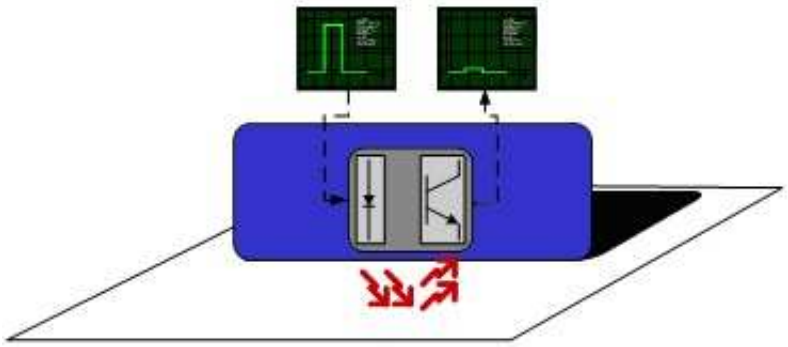


Image 7. Line tracking sensor on a clear surface.

- **Colored surface**: A colored surface reflects part of the emitted light obtaining an intermediate voltage at the microcontroller's analog channel input. This way colors are easily identified[1].

[1] Due to CNY70 tolerance two different sensor can differ.

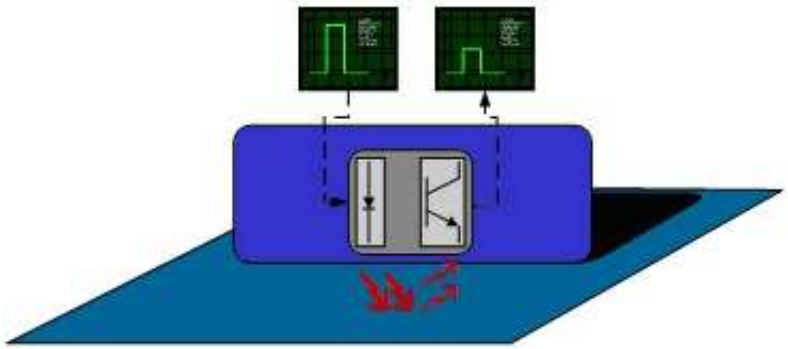


Image 8. Line tracking sensor on a colored surface.

1. **Dark surface**: A dark surface reflects very little light obtaining a high voltage reading at the sensor's output.

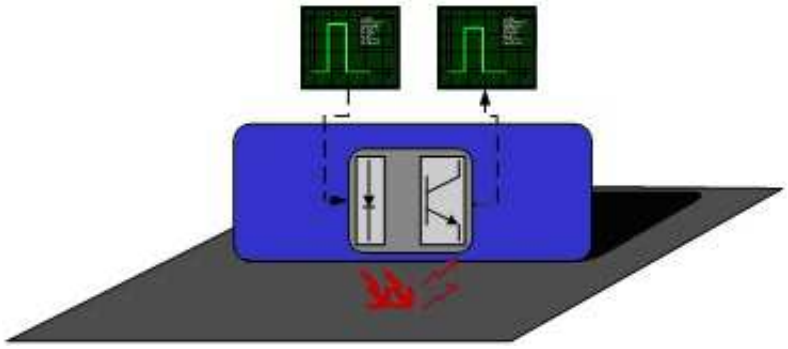


Image 9. Line tracking sensor on a dark surface.

Table 3. Line tracking sensors - PIC connections

Pin PIC	I/O	Sensor
PORTA		
RA2	I	Right line tracking sensor receiver
RA5	I	Left line tracking receiver
PORTD		
RD1	O	Left and right line tracking sensors transmitter

Image 10. Location of line sensors

3.3.2. Obstacle detection sensors

Similar to line tracking sensors, obstacle detection sensors also use infrared light to detect objects located in front of the mOway. The sensor includes two infrared light-emitting source (Kingbright KPA3010-F3C) and four receivers placed on both sides of mOway.

The output of the Sharp PT100F0MP receivers are connected to the microcontroller's analog inputs so it can detect the presence of any object (digital mode) and also measure how far away it is (analog mode)[2].

The sensor functions similarly to the line tracking sensor. The light emitter generates a 70us pulse which allows the receiver to capture any obstacle using a filtering and amplifying stage. Once the signal is processed electronically, the PIC can measure it by means of the ADC or as a digital input. The digital distance range is close to 3cm and a bright environment is recommended to enhance infrared light reflection.

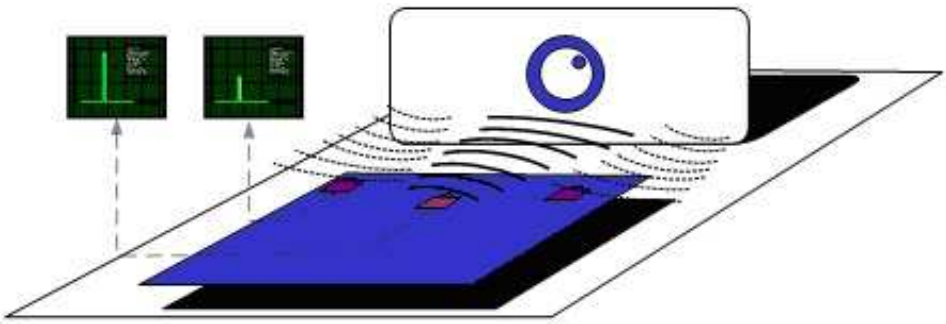


Image 11. Obstacle detection sensor

Table 4. Shock-proof sensor - PIC connections

Pin PIC	I/O	Sensor
PORTA		
RA1	I	Central right infrared receiver
RA3	I	Side left infrared receiver
PORTF		
RF5	I	Side right infrared receiver
RF6	I	Central left infrared receiver
PORTJ		
RJ7	O	Infrared transmitter

Image 12. Location of Obstacle Sensor

[2] Due to tolerance two different sensors can differ from each other.

3.3.3. Light sensor

This sensor allows mOway to recognize the light intensity that enters through a small half moon-shaped opening on the top part of the chassis. Since it is facing forward it enables it to detect where the light source is located and to act accordingly.

The output of the AVAGO APDS-9002 sensor is connected to the analog port of the microcontroller so that with a simple reading of the ADC we can register the light intensity level and any change in intensity levels based on the last reading[3].

Table 5. PIC - light sensor connection

Pin PIC	I/O	Sensor
PORTA		
RA0	I	Luz

Image 13. Location of Light Sensor

3.3.4. Expansion connector

This connector allows the mOway to connect with any commercial modules or electronic circuits the user may choose.

As shown in the above table, it is possible to connect commercial SPI devices. On the other hand, the RF BZI-RF2GH4 module available in the market is totally compatible with mOway and with specific libraries. This module enables the mOway to communicate with other robots and with a PC via the RFUsb. With this module it is possible to create complex collaboration applications without having to worry about complicated wireless communications.

Table 6. Expansion connector connections

Pin Expa	I/O	PIC
Pin1	O	Vcc 3.3v
Pin2	O	GND
Pin3	I/O /PMD3/AN12/P3C /C2INC	RH4
Pin4	I/O/PMA5/AN7/C2INB	RF2
Pin5	I/O /SCK1/SCL1	RC3
Pin6	I/O /SDO1/C2OUT	RC5
Pin7	I/O /SDI1/SDA1	RC4
Pin8	I/O/INT	RB0

[3] Top two-color LED has to be switched off to have a valid measure.

Image 14. RF modules into expansion connector.

3.3.5. Temperature sensor

mOway has installed as a temperature measurer an NTC thermistor from Murata, a semiconductor whose electrical variable resistance decreases as temperature increases. The sensor is located in the front part of the robot, very close to obstacle sensor.

The thermistor is connected to the analog port of the microcontroller so that with a simple reading of the ADC it is possible to get the temperature value in that moment and notice any change in it since the last reading[4].

Table 7. PIC-Temperature sensor connection

Pin PIC	I/O	Sensor
PORTH		
RH5	I	Temperature sensor

3.3.6. Speaker

The CMT-1102 speaker from CUI INC directly connected to the microcontroller, is capable to play tones from 250 Hz to 65 KHz.

Table 8. PIC-Speaker connection

Pin PIC	I/O	Sensor
PORTB		
RB3	O	Speaker

3.3.7. Microphone

The CMC-5042PF-AC microphone from CUI INC enables the robot to detect sounds from 100 Hz to 20 KHz.

[4] Temperature measured by the sensor can be 5ºC higher than external temperature.

The output is directly connect to an analog input of the microcontroller so that it is capable to detect not only if there is sound or not (digital mode) but also the intensity of the sound with a simple reading of the ADC (analog mode).

Table 9. PIC-Microphone connection

Pin PIC	I/O	Sensor
PORTH		
RH7	I	Microphone

3.3.8. Accelerometer

An accelerometer is a device that measures acceleration and the gravity induced forces: the movement and rotation. There are many types of accelerometers, most of them based on piezoelectric crystals, but their size is too big. Because of that, it was tried to design a small device in the field of microelectronics, which might improve the applicability. Then, the MEMS (Microelectromechanical Systems) accelerometers were created.

An easy way to create an accelerometer is measuring changes in a capacitor. Capacitors can work as sensors or as actuators. In the case of mOway, it is a capacitive accelerometer, which consists of two capacitors displaced in differential mode whose electrical capacity changes as the acceleration varies.

By measuring X, Y, Z axes of the MMA7455L accelerometer from FREESCALE Semiconductor, it is possible to know if mOway is correctly positioned, inverted or tilted.

Table 10. PIC-Accelerometer connection

Pin Acce	I/O	PIC
Pin7	I	RD7
Pin8	I	RB1
Pin9	I	RB2
Pin12	I	RD4
Pin13	O	RD5
Pin14	O	RD6

3.3.9. Battery level

The robot has a LiPo cell battery rechargeable. For proper operation of the microcontroller, the battery is connected to one of its analog inputs through a splitter. Thus, with a reading of the ADC battery level can be measured.

Table 11. PIC-Battery level connection

Pin PIC	I/O	Sensor
PORTH		

RH6	I	Battery level

3.3.10. Front LED

The front LED is a white LED placed on the front side of mOway. The output of the OSRAM LW A6SG LED is connected to a digital output of the microcontroller.

Table 12. PIC - front LED connections

Pin PIC	I/O	Sensor
PORTC		
RC7	O	Front LED

3.3.11. Top two-color LED

This double indicator and the light sensor share the same opening on the top part of the robot. They are connected to two microcontroller digital outputs[5].

Table 13. PIC-Top LED connection

Pin PIC	I/O	Sensor
PORTA		
RA4	O	Top red LED
PORTB		
RB6	O	Top green LED

Image 15. Robot with Front LED and red LED switched on

3.3.12. Brake LED

The brake LED is double indicator placed on the back side of mOway. The output is connected to one digital outputs of the microcontroller.

[5] Please note that since they share the same opening as the light sensor it is fundamental to switch them off when wanting to perform a light intensity reading.

Table 14. PIC- Brake LED connection

Pin PIC	I/O	Sensor
PORTE		
RE5	O	Brake LED

Image 16. Brake LED location. Switch on green LED.

3.3.13. Free Pad

mOway has implemented a free Pad to allow expert users to connect their electronics. It is accessible opening the robot and it´s located near brake LED[6].

Table 15. PIC-free Pad connection

Pin PIC	I/O	Sensor
PORTJ		
RJ7	I/O	Free Pad

3.4. Power Supply System

mOway's battery is located inside and accessible only by disassembling the product. It is a small rechargeable LiPo cell.

The battery can be charged via a computer's USB port through the mOway's MINI-USB-B port. There is no need to wait for the battery to be completely discharged, as it can be plugged in any time since these batteries do not have memory effect (also known as lazy battery effect). These batteries are a perfect power source for mOway due to their small size, lightness and flexibility.

Battery duration depends to a great extent on the active sensors and the amount of time the motors are used. Charging lasts about 2h.

Power supply system controls two LED located in the back part of the robot[7]. Green LED indicates that mOway is switched on and red LED indicates that the battery is charging. When the battery is full red LED will switch off[8].

[6] Advanced users only
[7] These LEDs can´t be controlled by the user.

Image 17. Charging (red) and switched on (green)

3.5. RF module and RFUsb[9]

RF module allows communicate with other mOways or with PC using RFUsb.[10].

Image 18. RF module

RF module is connected in expansion connector and it is very easy to use with mOwayGUI. The best way to start working with the module is using an example project provided in mOwayPack.

[8] This LED can swap between on and off when the battery is fully charge because there is energy consumption when mOway is plugged.

[9] Available in some packs

[10] Available in some packs

Image 19. RFUsb

The **BZI-RF2GH4** radio-frequency communications module is based on the nRF24L01 transceptor manufactured by "Nordic Semiconductors". This integrated circuit has been fitted with all the logic required to establish wireless bidirectional communications with acknowledgement of receipt. Communications with the microcontroller is made via an SPI bus.

The main characteristics of the BZI-RF2GH4 module are as follows:

- Low consumption.
- Working frequency: 2.4GHz,
- Transmitting power between-18 and 0 dBm,
- Transmission speed between 1 and 2 Mbps,
- 128 in transmission channels selectable by the SPI bus.

In addition to the CI nRF24L01, the **BZI-RF2GH4** is also fitted with all the associated electronics for its correct operation plus a microstrip antenna on the same board with the impedance adaptation network. In this way the user can forget completely about the hardware required to implement the radio part of his application.

As interface, the device has four pins available for the SPI bus, two more pins for controlling the module and another two for the supply.

In order to facilitate the handling of the module, a number of libraries have been developed to simplify and shorten the development time of wireless applications with these modules.

3.5.1. Technical specifications

Table 16. Maximum Ratings

Parameter	Min	Max	Unit
Vdd	-0.3	3.6	V
Vss		0	V
Data input voltage	-0.3	5.25	V
Data output voltage	Vss-Vdd	Vss-Vdd	V

Power dissipation		60	mW
Operating temperature	-40	+85	ºC
Storage temperature	-40	+125	ºC

Table 17. Specifications BZI-RF2GH4

Parameter	Value	Unit
Minimum supply voltage	1.9	V
Maximum supply voltage	3.6	V
Maximum power output	0	dBm
Maximum transmission speed	2000	Kbps
Current in transmission mode @ 0dbm power output	11.3	mA
Current in reception mode@ 2000kbps	12.3	mA
Current in *Power Down* mode	900	nA
Maximum frequency of the SPI bus	8	Mhz
Temperature range	-40 a +85	ºC

Table 18. Pinout BZI-RF2GH4

Pins	Nº	Description
Vcc	1	Supply voltage of the module
Vss	2	GND
CE	3	Chip Enable
CSN	4	Chip Select of the SPI
SCK	5	SPI bus clock
SDI	6	Data input to the RF module of the SPI bus (MOSI)
SDO	7	Data output from the RF module of the SPI bus(MOSI)
IRQ	8	Output interruption

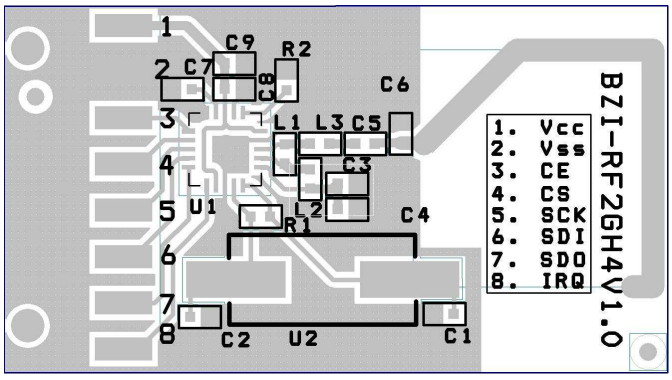


Image 20. Module layout

4. First Steps

4.1. mOway Pack installation

In mOwayPack (available in the webpage or in the installation CD) you will find the software, mOway´s libraries, test programs and documentations.

Following setup steps you will have all the resources:

- Beginner's and User manual.
  - Beginner's manual includes all you need to start working with mOway.
  - User manual contains detailed description of the robot.
- mOwayGUI software.
  - This software controls all aspects of the robot: program download, battery charge control, radio control, RFUsb[11] management and C or assembler programs download.
- Reference projects in assembler and C.
  - Example projects to start working with mOway easily.
- RFUsb Driver
  - Driver for RFUsb[12] that allows the communication between robots and PC.

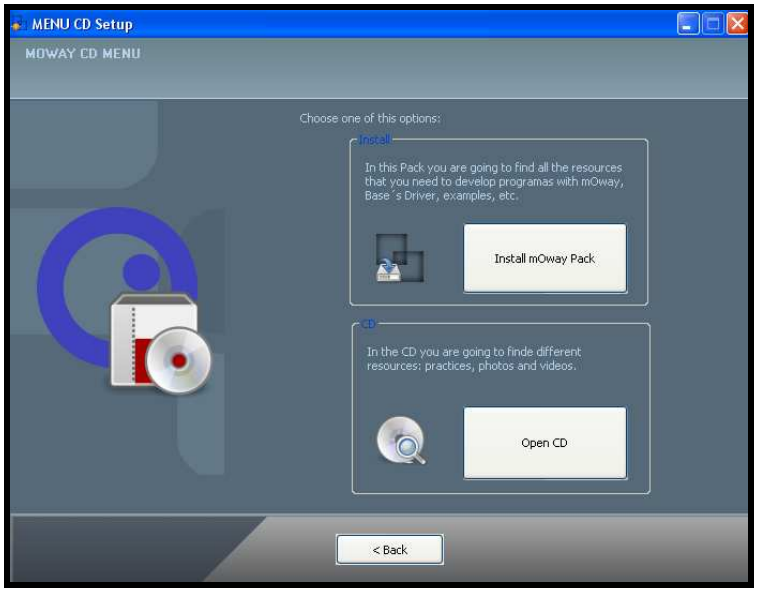


Image 21. CD

[11] Module not available in all kits
[12] Module not available in all kits

4.2. Download a program to mOway

Download process is always executed in mOwayGUI. This application can download to the robot mOwayGUI, assembler (compiled with Mplab o gputils) and C (C18 compiler) projects.

Steps to download a program to mOway:

- Connect mOway to the PC through USB. The robot doesn´t need any driver.
- Open mOwayGUI application.
- Open or create a project in mOwayGUI, or import a .HEX file from assembler or C project.
- Click download bottom. If a .HEX file has been imported the download progress will start automatically.
- Disconnect the robot and check the project.

mOwayPack provide 8 compiled projects: 3 to check sensors, 3 to check the drive system and 2 programs that are explained in the next section.

Project to check sensors:

- ASM_SEN_01: Assembler software to check sensors. Obstacle sensors are checked and if an object is detected LEDs and speakers are switched on. This project uses absolute **lib_sen_moway.inc** library.
- ASM_SEN_02 : Assembler software to check sensors. Obstacle sensors are checked and if an object is detected LEDs and speakers are switched on. This project uses relocatable **lib_re_sen_moway.inc** library.
- C_SEN_01 : C18 software to check sensors. Obstacle sensors are checked and if an object is detected LEDs and speakers are switched on.

Project to check drive system:

- ASM_MOT_01: Assembler software to check drive system. Different movements of mOway are executed. This project uses absolute **lib_mot_moway.inc** library.
- ASM_MOT_02: Assembler software to check drive system. Different movements of mOway are executed. This project uses relocatable **lib_re_mot_moway.inc** library.
- C_MOT_01: C18 software to check drive system. Different movements of mOway are executed.

Projects to check RF module:

- ASM_RF_01: Assembler software to check RF module. Sends data to channel 0x40 and address 0x02 every 2 sec. This project uses absolute **lib_rf2gh4.inc** library.

- ASM_RF_02: Assembler software to check RF module. Sends data to channel 0x40 and address 0x02 every 2 sec. This project uses relocatable **lib_re_rf2gh4.inc** library.
- ASM_RF_03: Assembler software to check RF module. Makes robot work as a repeater. Reception without interruption. This project uses absolute **lib_rf2gh4.inc** library.
- ASM_RF_04: Assembler software to check RF module. Makes robot work as a repeater. Reception without interruption. This project uses relocatable **lib_re_rf2gh4.inc** library.
- C_RF_01: C18 software to check RF module. Sends data to channel 0x40 and address 0x02 every 2 sec.
- C_RF_02: C18 software to check RF module. Makes robot work as a repeater. Reception without interruption.

First projects:

- mOway_first_project_ASM: Assembler project explained in this manual. mOway avoids obstacles rotating 180º.
- mOway_first_project_C18: C18 project explained in this manual. mOway avoids obstacles rotating 180º.

4.3. RFUsb instalation

- This is a device that allows to communicate the PC and mOway.

A driver that it´s included in mOwayPack is required:

- The first time the RFUsb is connected, the PC will detect it as a new device and an "Assistant for new hardware found" message will be displayed. Select the *No,not this time* option.
- In the following window select the recommended option: Install software automatically.

Image 22. Driver installation Wizard

- Now the installation process will begin.

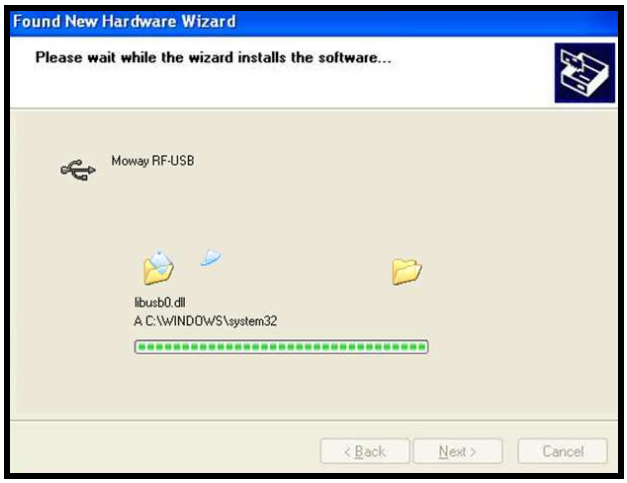


Image 23. Windows xp driver installation

- Assistant will then indicate that the hardware is installed.

Image 24. Driver installed in Windows xp

- Check if Moway's software has detected the RFUsb

4.4. RF modules

RF modules are very useful tool to introduce RF concept.

These are the steps to start working with them:

- Connect RF modules into the expansion connector. Check that the module is fully connected.
- Connect the robot to the PC through the USB cable.

- Open mOwayGUI.
- Open *mOway_RF_send* project included in the pack.
- Click the Program bottom.
- Disconnect and switch the robot on.
- Configure RFUsb module using RF window of mOwayGUI with cannel 0x00 and address 0x01.
- Check receiving data in mOwayGUI.

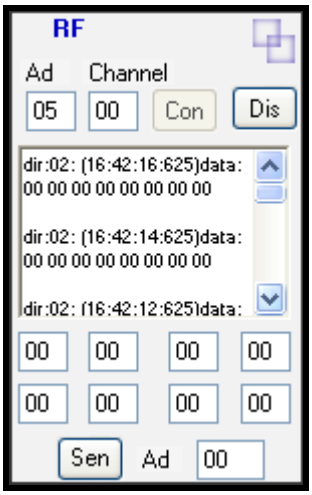


Image 25. RF window

5. Programming mOway in assembler

Microchip's MPLAB IDE is the most widely used PIC microcontroller programming environment (as Microchip also manufactures these microcontrollers). It basically uses assembly language, but other languages can also be added. Thanks to it source code can be compiled and hexadecimal files (.HEX) generated. This compiler can be downloaded, free of charge, from Microchip's Website.

mOwayPack offers sensor, motor and RF module managing libraries written for MPLAB.

Summary:

- Very interesting to learn assembly language programming (low level language).
- Ideal for large code size programs. Indispensable for critical response timeframes.

5.1. Creating a project

Use the MPLAB IDE Project Wizard to create the first project quickly.

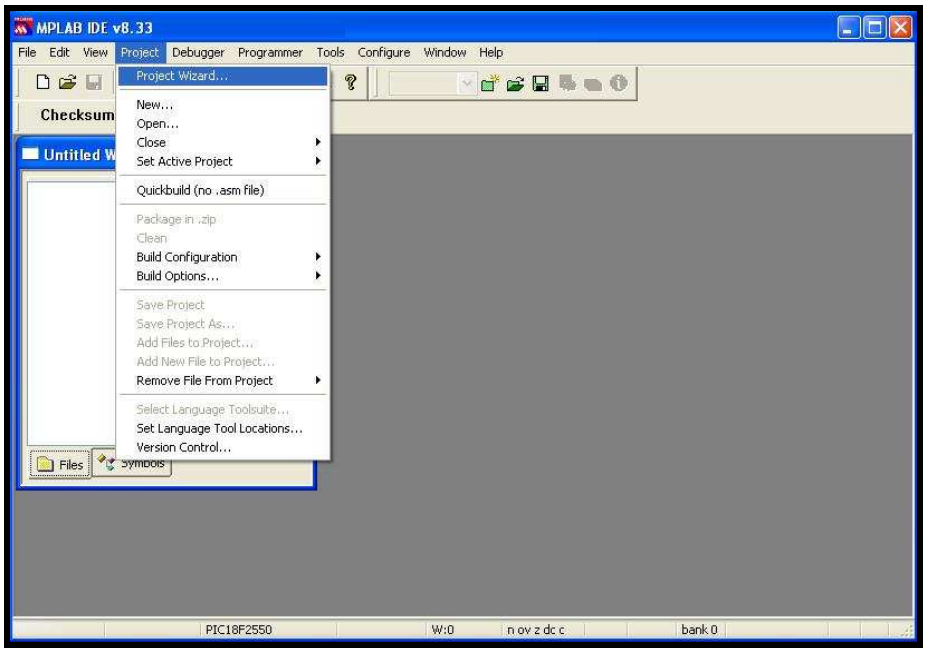


Image 26. Project Wizard

1. First select the PIC installed in mOway: PIC18F86J50

Image 27. PIC selection

2. Then select the assembly tool: MPASM.

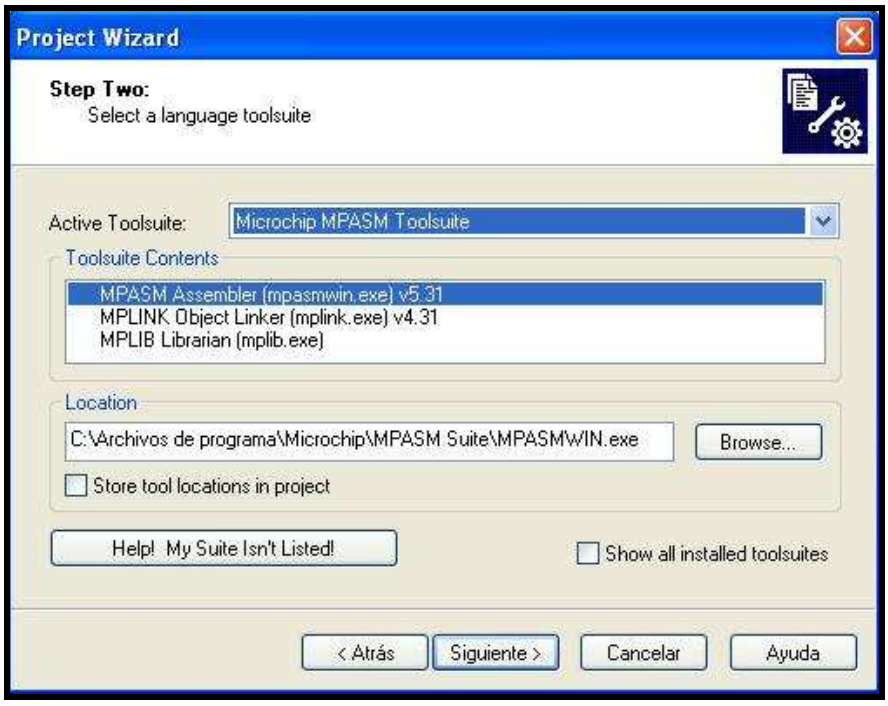


Image 28. Tool selection

3. In step three enter the project's name and location.

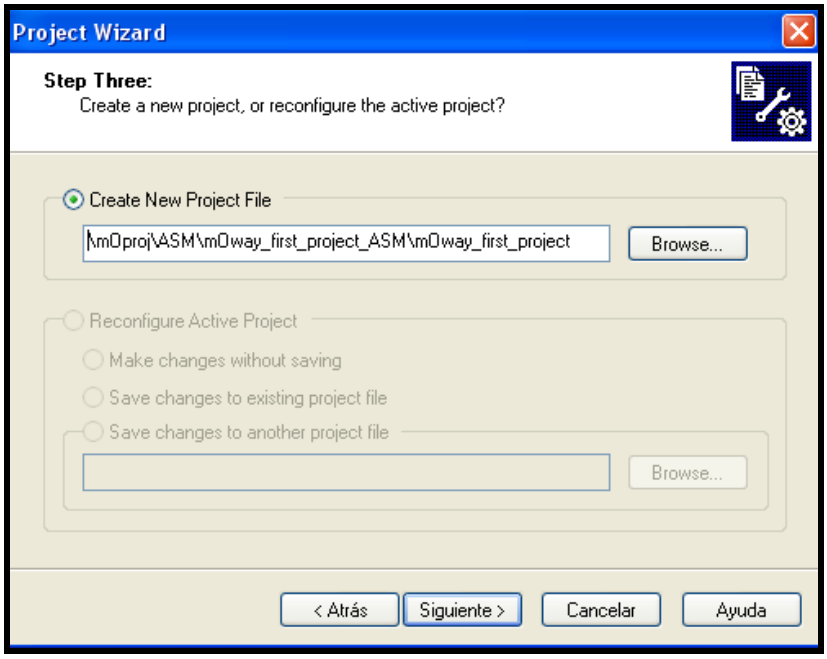


Image 29. Select name and folder

4. In the next step the mOway libraries which control different features of the robot are added to the project.

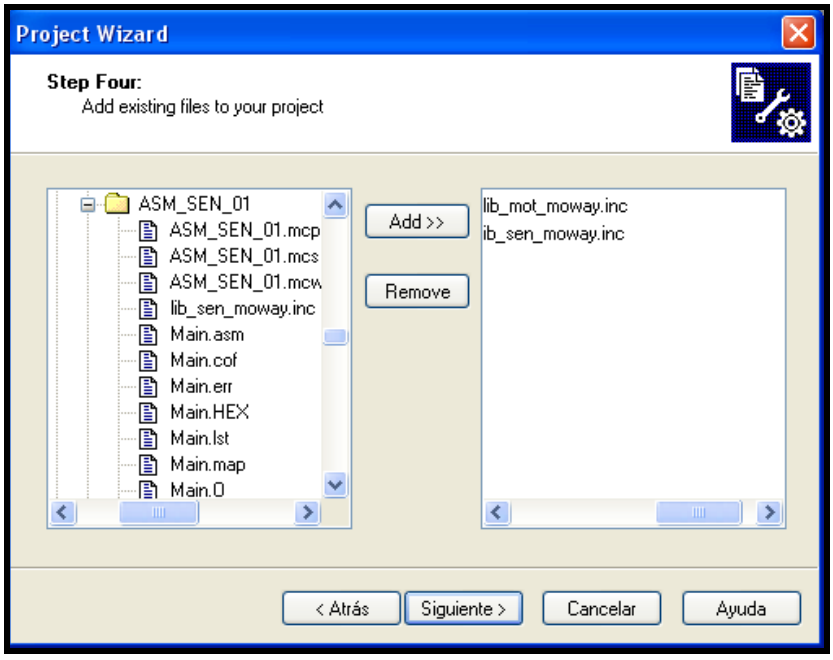


Imagen 30. Select mOway libraries

5. With the steps above the project will now be created, the next step is to create a .ASM file for the source code.

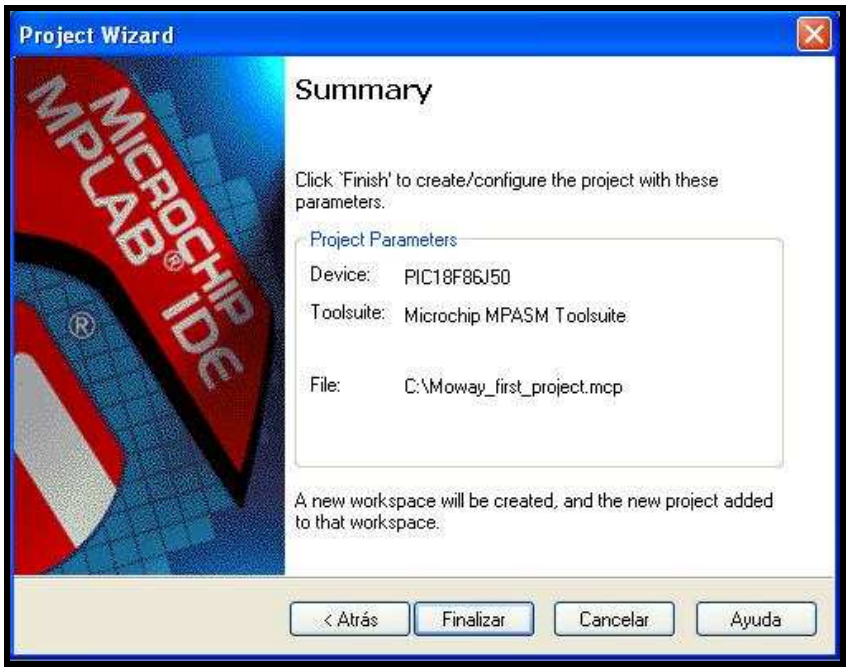


Image 31. Wizard ends

6. The next step is to open the project and create a new file (*New File*) saving it in the same folder of the project as *Main.asm*. This will be our source file.

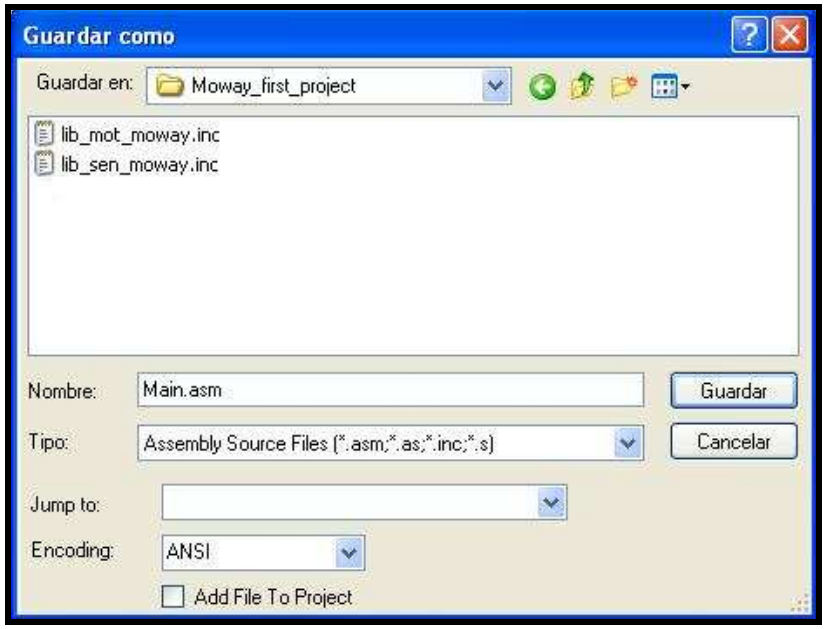


Image 32. .ASM creation

7. Finally, the source file is added to the project accessing *Project/Add Files to Project…*

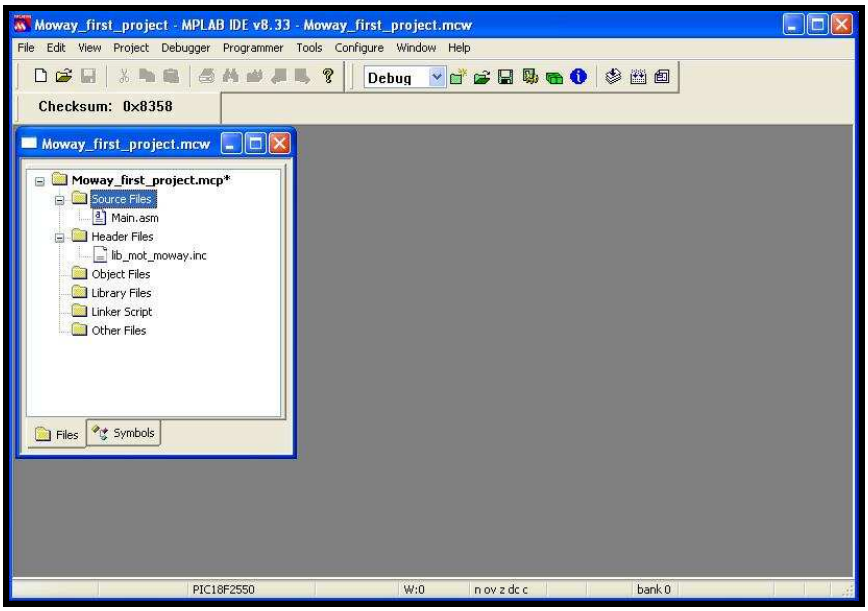


Image 33. Project with .ASM

5.2. First program in assembler

To generate the first program a project has to be created first (previous chapter). This first basic program will enable the mOway to avoid obstacles. (Absolute code)

1. First the list p=18F86J50 PIC installed in mOway has to be added to the Main.ASM file.
2. It is also necessary to include into the project folder the library for this microcontroller which can be found at the MPLAB installation directory or in mOway's pack testing programs. Once this library is copied to the folder enter: #include "P18F86J50.INC" in the Main.ASM file.
3. The next step is to add the starting (0x102A) and resetting (0x1000) vectors, and to include the mOway libraries.
4. INIT and MAIN labels are added to create a loop.
5. Next, the SEN_CONFIG function is called to configure the microcontroller's inputs and outputs.
6. Add winking to one of the LEDs.
7. Test the program on mOway programming it in mOwayGUI and verify that the green LED blinks.

```
    list p=18F86j50
;*  PIC register definition
#include "P18F86j50.INC"
;*  Reset vector
    org     0x1000
    goto    INIT
;*  Program memory
    org     0x102A
;***************************[MOWAY LIBRARY]*******************************
#include "lib_sen_moway.inc"
;*************************************************************************

INIT
    ;Microcontroller configuration to use the sensors
    call        SEN_CONFIG
    ;Green LED blink
    call        LED_TOP_GREEN_ON_OFF
    ;*****************************[MAIN PROGRAM]**************************
MAIN









goto    MAIN
    ;*************************************************************************

    END
```

Image 34. First program: configuration and LED

8. To detect obstacles call up the SEN_OBS_DIG function with OBS_CENTER_L parameter, in infinite loop, which will inform of the presence of an obstacle through the SEN_OBS variable.
9. If it detects an obstacle the front LEDs light up.
10. Test the program on mOway and verify that the LEDs switch on when an object is placed close to the front part of the mOway.

```
    list p=18F86j50
;*  PIC register definition
#include "P18F86j50.INC"
;*  Reset vector
    org     0x1000
    goto    INIT
;*  Program memory
    org     0x102A
;***************************[MOWAY LIBRARY]*******************************
#include "lib_sen_moway.inc"
;*************************************************************************

INIT
    ;Microcontroller configuration to use the sensors
    call        SEN_CONFIG
    ;Green LED blink
    call        LED_TOP_GREEN_ON_OFF
    ;*****************************[MAIN PROGRAM]**************************
MAIN

    ;Check central left obstacle sensor
    movlw       OBS_CENTER_L
    movwf       SEN_CHECK_OBS
    call        SEN_OBS_DIG

    ;If a obstacle is detected robot avoids it
    btfsc       SEN_OBS,0
    call        LED_FRONT_ON
    btfss       SEN_OBS,0
    call        LED_FRONT_OFF
    goto        MAIN



goto    MAIN
    ;*************************************************************************
```

Image 35. First program: detecting obstacles

11. We then add movement to the robot: unrestricted straight command until it encounters an obstacle.
12. lib_mot_asm.inc is added to the project.
13. MOT_CONFIG is called to be able to use Diver system.
14. Go straight on the first time.

```
    list p=18F86j50
;*  PIC register definition
#include "P18F86j50.INC"
;*  Reset vector
    org     0x1000
    goto    INIT
;*  Program memory
    org     0x102A
;***************************[MOWAY LIBRARY]************************************
#include "lib_sen_moway.inc"
#include "lib_mot_moway.inc"
;******************************************************************************

INIT
    ;Microcontroller configuration to use the sensors
    call    SEN_CONFIG
    ;Microcontroller configuration to use the engines
    call    MOT_CONFIG
    ;Green LED blink
    call    LED_TOP_GREEN_ON_OFF
    ;****************************[MAIN PROGRAM]*********************************
    ;Straight on 50% speed
    movlw       .50
    movwf       MOT_VEL
    movlw       .0
    movwf       MOT_T_DIST_ANG
    bsf         MOT_CON,FWDBACK
    bsf         MOT_CON,COMTYPE
    call        MOT_STR
MAIN

    ;Check central left obstacle sensor
    movlw       OBS_CENTER_L
    movwf       SEN_CHECK_OBS
    call        SEN_OBS_DIG
```

Image 36. Configuration and first movement

15. When it encounters an obstacle a command is sent to rotate 180º and the top red LED lights up (the front LEDs will not operate). The robot will wait until this command has ended and will then continue moving straight forward.

```
MAIN

    ;Check central left obstacle sensor
    movlw       OBS_CENTER_L
    movwf       SEN_CHECK_OBS
    call        SEN_OBS_DIG

    ;If a obstacle is detected robot avoids it
    btfsc       SEN_OBS,0
    goto        OBS_DETECT
    btfss       SEN_OBS,0
    call        LED_FRONT_OFF
    goto        MAIN

OBS_DETECT
    call        LED_FRONT_ON
    ;Rotate 180° at 20% speed
    movlw       .20
    movwf       MOT_VEL
    movlw       .50
    movwf       MOT_T_DIST_ANG
    movlw       0x01
    movwf       MOT_CENWHEEL
    bsf         MOT_CON,FWDBACK
    bcf         MOT_CON,COMTYPE
    bcf         MOT_CON,RL
    call        MOT_ROT
    ;Wait until comand finishes
    call        MOT_CHECK_END
    ;Straight on 50% speed
    movlw       .50
    movwf       MOT_VEL
    movlw       .0
    movwf       MOT_T_DIST_ANG
    bsf         MOT_CON,FWDBACK
    bsf         MOT_CON,COMTYPE
    call        MOT_STR

goto    MAIN
    ;*****************************************************************************
```

Image 37. . First program: detecting obstacles moving

This project is included in the mOway pack.

5.3. Libraries

5.3.1. mOway´s sensors library in assembly language

There are two libraries in assembly language which can be included in any mOway project and allow the user to control the sensors with ease. Both are identical except that one of them can relocate the code and the variables (using the MPLAB IDE projects).

It is essential to understand that every time a function library is called up it uses an additional call stack level. This means that at least two call stack levels must be free before calling one of these functions to avoid errors.

5.3.1.1. Description

The library includes a series of functions in charge of reading the data provided by the robot's sensors. They configure the input and output ports, the microcontroller's ADC and the luminous indicators.

5.3.1.2. Variables

SEN_STATUS

This read-only variable checks the validity of the data returned by the sensors.

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Unused	Unused	Unused	Unused	Unused	Unused	DWRONG	SENOK
-	-	-	-	-	-		

Bit 7-2: **Unused**

Bit 1: **DWRONG:** shows if input data is correct.
1 = Incorrect data.
0 = Correct data.

Bit 0: **SENOK:** shows if the sensor has been read correctly.
1 = Correct reading. Valid output data.
0 = Incorrect reading. Invalid output data.

SEN_ACCE_TAP

Read-only variable that shows if SEN_CHECK_ACCE_TAP function detects one or two taps.

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Unused	Unused	Unused	Unused	Unused	Unused	TAP_TAP	TAP
-	-	-	-	-	-		

Bit 7-2: **Unused**
Bit 1: **TAP_TAP:** shows if double tap is detected
1 = Double Tap detected
0 = Double Tap not detected

Bit 0: **TAP:** shows if tap is detected
1 = Tap detected
0 = Tap not detected

SEN_CHECK_OBS

This write-only variable shows which sensor must be read by obstacle functions.

Table 19. Allowed values for SEN_CHECK_OBS

Define	Value
OBS_CENTER_L	0
OBS_SIDE_L	1
OBS_CENTER_R	2
OBS_SIDE_R	3

SEN_CHECK_ACCE

This write-only variable shows which axis must be read by SEN_ACCE_XYZ_READ function.

Table 20. Allowed values for SEN_CHECK_ACCE

Define	Value
ACCE_CHECK_X	0
ACCE_CHECK_Y	1
ACCE_CHECK_Z	2

SEN_CHECK_LINE

This write-only variable shows which sensor must be read by line functions.

Table 21. Allowed values for SEN_CHECK_LINE

Define	Value
LINE_L	0
LINE_R	1

SEN_SPEAKER_ON_OFF

This write-only variable shows if speaker have to turn on, turn off or play an amount of time.

Table 22. Allowed values for SEN_SPEAKER_ON_OFF

Define	Value
SPEAKER_OFF	0
SPEAKER_ON	1
SPEAKER_TIME	2

SEN_LIGHT_P

This variable records the percentage of light reaching the light sensor. It is updated every time the SEN_LIGHT function is called.

SEN_BATTERY_P

This variable records the percentage of the battery level. It is updated every time the SEN_BATTERY function is called.

SEN_TEMPERATURE_C

This variable records the value of the temperature in ºC. It is updated every time the SEN_TEMPERATURE_C function is called.

SEN_MIC

This variable records the value of the microphone. The data will be digital or analog depending on the updating function: SEN_MIC_DIG and SEN_MIC_ANALOG.

SEN_SPEAKER_FREQ

This variable records the value of the frecuency, between 250 Hz and 65 KHz, to create the tone.

SEN_SPEAKER_TIME

These variable records the time the speaker must send the tone.

SEN_OBS

This variable stores the value of the obstacle sensor (SEN_CHECK_OBS). This value is updated when SEN_OBS_DIG or SEN_OBS_ANALOG functions are called.

SEN_ACCE

This variable stores the value of the acceleration The axis is selected by SEN_CHECK_ACCE. This value is updated when SEN_ACCE_XYZ_ functions is called.

SEN_ACCE_TAP

This variable records if mOway has been taped. It is updated every time the SEN_ACCE_CHECK_TAP function is called up.

SEN_LINE

This variable stores the value of the line sensor (SEN_CHECK_LINE). This value is updated when SEN_LINE_DIG or SEN_LINE_ANALOG functions are called.

5.3.1.3. Functions

A series of functions to control mOway's sensors and LED diodes are included in the lib_sen_moway and lib_re_sen_moway libraries.

Below is a brief description of each one of these function.

Table 23. ASM function summary

Name	Input variable	Output variable	Description
SEN_CONFIG	-	-	Configured to use the sensors.
SEN_LIGHT	-	SEN_LIGHT_P	Reads light sensor values.
SEN_BATTERY	-	SEN_BATTERY_P	Returns the battery level.
SEN_TEMPERATURE	-	SEN_TEMPERATURE_C	Detects the temperature in ºC.
SEN_MIC_ANALOG	-	SEN_MIC	Detects sound intensity.
SEN_MIC_DIG	-	SEN_MIC	Detects if there is sound or not.
SEN_SPEAKER	SEN_SPEAKER_FREQ SEN_SPEAKER_TIME SEN_SPEAKER_ON_OFF	-	Emits tones in a frequency between 250 Hz and 65 KHz.
SEN_OBS_DIG	SEN_CHECK_OBS	SEN_OBS SEN_STATUS	Detects obstacles
SEN_OBS_ANALOG	SEN_CHECK_OBS	SEN_OBS SEN_STATUS	Detects the distance to obstacles
SEN_ACCE_XYZ_READ	SEN_CHECK_ACCE	SEN_ACCE SEN_STATUS	Calculates the X,Y,Z axes acceleration of mOway.
SEN_ACCE_CHECK_TAP	-	SEN_ACCE_TAP SEN_STATUS	Detects if mOway has been taped.
SEN_LINE_DIG	SEN_CHECK_LINE	SEN_LINE SEN_STATUS	Detects dark zones (black lines)
SEN_LINE_ANALOG	SEN_CHECK_LINE	SEN_LINE SEN_STATUS	Detects surface colors
LED_BRAKE_ON	-	-	Brake LED on
LED_FRONT_ON	-	-	Front LED on
LED_TOP_RED_ON	-	-	Top red LED on
LED_TOP_GREEN_ON	-	-	Top green LED on
LED_BRAKE _OFF	-	-	Brake LED off
LED_FRONT_OFF	-	-	Front LED off
LED_TOP_RED_OFF	-	-	Top red LED off
LED_TOP_GREEN_OFF	-	-	Top green LED off
LED_BRAKE _ON_OFF	-	-	Brake LED blink
LED_FRONT_ON_OFF	-	-	Front LED blink
LED_TOP_RED_ON_OFF	-	-	Top red LED blink
LED_TOP_GREEN_ON_OFF	-	-	Top green LED blink

SEN_CONFIG

This function configures the inputs and outputs required to manage the sensors and initialize the variables.

Table 24. PIC-sensor connections

Pin PIC	I/O	Sensor
PORTA		
RA0	I	Light
RA1	I	Central left infrared receiver
RA2	I	Right line sensor receiver
RA3	I	Side left infrared receiver
RA5	I	Left line sensor receiver
PORTB		
RB1	I	First interruption of the accelerometer
RB2	I	Second interruption of the accelerometer
RB3	O	Speaker
RB5	O	Top red LED
RB6	O	Top green LED
PORTC		
RC7	O	Front LED
PORTD		
RD1	O	Line sensors transmitter
RD4	I	SDO signal for the SPI communication (accelerometer)
RD5	O	SDI sinal for the SPI communication(accelerometer)
RD6	O	Clock sinal for the SPI communication(accelerometer)
RD7	I	Chip Select for the SPI communication(accelerometer)
PORTE		
RE5	O	Brake LED
PORTF		
RF5	I	Side right infrared receiver
RF6	I	Central right infrared receiver
PORTH		
RH5	I	Tempreature sensor
RH6	I	Battery measurer
RH7	I	Microphone
PORTJ		
RJ6	O	Infrared transmitter
RJ7	I/O	Free pad

SEN_LIGHT

Output variables	
SEN_LIGHT_P	Percentage of ambient light.

The SEN_LIGHT function captures the analog value generated by the inciding light on the photo-transistor. To achieve this follow these steps:

- Activate the ADC.
- Wait for the data acquisition process to end (100us).

- Read the analog value.
- Calculate the inciding light percentage based on the analog voltage measurement.
- This information is then copied to the SEN_LIGHT_P variable.

SEN_BATTERY

Output variables	
SEN_BATTERY_P	Percentage of battery level.

The SEN_BATTERY function captures the analog value of the battery[13]. To achieve this, function follows these steps:

- Activate the ADC.
- Wait for the data acquisition process to end (100us).
- Read the analog value.
- Calculate the battery level percentage based on the analog voltage measurement.
- This information is then copied to the SEN_BATTERY_P variable.

SEN_TEMPERATURE

Output variables	
SEN_TEMPERATURE_C	Temperature in ºC.

The SEN_TEMPERATURE function captures the analog value that depends on the temperature captured by the thermistor[14]. To achieve this, function follows these steps:

- Activate the ADC.
- Wait for the data acquisition process to end (100us).
- Read the analog value.
- Calculate temperature based on the analog voltage measurement.
- This information is then copied to the SEN_TEMPERATURE_C variable.

SEN_MIC_ANALOG

Output variables	
SEN_MIC	Sound Intensity.

[13] The output value can differ from mOwayGUI.
[14] Sensor measures mOway´s temperature which can be different from ambient temperature.

The SEN_MIC_ANALOG function captures the analog value that depends on the sound intensity from the microphone. To achieve this, function follows these steps:

- Activate the ADC.
- Wait for the data acquisition process to end (100us).
- Read the analog value.
- This information is then copied to the SEN_MIC variable.

SEN_MIC_DIG

Output variables	
SEN_MIC	Indicates if there is sound or not.

The SEN_MIC_DIG function indicates if there is sound or not. To achieve this function follows these steps:

- Check if there is sound in the microphone.
- This information is then copied to the SEN_MIC variable.

SEN_SPEAKER

Input variables	
SEN_SPEAKER_FREQ	Sound frecuencia (see table).
SEN_SPEAKER_TIME	Time.
SEN_SPEAKER_ON_OFF	On, off or time.

The SEN_SPEAKER function emits tones in a frequency between 250 Hz and 65 KHz. SEN_SPEAKER_ON_OFF is going to say if we want to switch on, switch off or activate the speaker an amount of time (100ms intervals). To achieve this, function follows these steps:

- PWM on with frequency SEN_SPEAKER_FREQ and 50% of duty.
- If SEN_SPEAKER_ON_OFF is SPEAKER_TIME(2) function waits until command finishes.

Table 25. Allowed values for SEN_SPEAKER_ON_OFF

Define	Value
SPEAKER_OFF	0
SPEAKER_ON	1
SPEAKER_TIME	2

Table 26. SEN_SPEAKER_FREQ vs PWM frequency

SEN_SPEAKER_FREQ	PWM frequency Hz
0	0,0000000
10	5681,8181818
20	2976,1904762
30	2016,1290323
40	1524,3902439
50	1225,4901961
60	1024,5901639
70	880,2816901
80	771,6049383
90	686,8131868
100	618,8118812
110	563,0630631
120	516,5289256
130	477,0992366
140	443,2624113
150	413,9072848
160	388,1987578
170	365,4970760
180	345,3038674
190	327,2251309
200	310,9452736
210	296,2085308
220	282,8054299
230	270,5627706
240	259,3360996
250	249,0039841
255	244,1406250

SEN_OBS_DIG

Input variable	
SEN_CHECK_OBS	Which sensor must be read
Output variable	
SEN_OBS	Indicates if there is obstacle or not.
Output	
SEN_STATUS: SENOK DWRONG	

This function indicates if the obstacle is situated on the right front side or on the left front side. To achieve this function follows these steps:

- Ensure that there is no noise source interference before sending the infrared light pulse.
- Emit the infrared light pulse to detect obstacles. This light-beam will be reflected back if there is any existing obstacle and this signal will be perceived by the infrared receiver.
- Check for any eventual signals from the four IR receivers.
- Copy the digital receiver's value to the output variables.
- Deactivate the infrared diode.
- Check for interfering signals.
- If there is no signal interferences and the process develops normally the SENOK flag is activated.

Table 27. Allowed values for SEN_CHECK_OBS

Define	Value
OBS_CENTER_L	0
OBS_SIDE_L	1
OBS_CENTER_R	2
OBS_SIDE_R	3

SEN_OBS_ANALOG

Input variable	
SEN_CHECK_OBS	Which sensor must be read
Output variable	
SEN_OBS	Indicates if there is obstacle or not.
Output	
SEN_STATUS: SENOK DWRONG	

This function indicates if the obstacle is on the right front side or on the left front side and its distance from the robot. To achieve this follow the steps indicated below:

- Ensure that there is no noise source interferences before you send the infrared light pulse.
- Emit the infrared light pulse to detect obstacles.
- Activate the ADC.
- Check for any possible signals from the four IR receivers.
- Copy the analog receiver's value to the output variables. The higher the value the shorter the distance will be.
- Deactivate the infrared diode.

- Check for interfering signals. If there is no signal interferences and the process develops normally the SENOK flag is activated.

Table 28. SEN_CHECK_OBS allowed values

Define	Value
OBS_CENTER_L	0
OBS_SIDE_L	1
OBS_CENTER_R	2
OBS_SIDE_R	3

SEN_ACCE_XYZ_READ

Input variable	
SEN_CHECK_ACCE	Which axis must be read
Output variable	
SEN_ACCE	Acceleration value
Output	
SEN_STATUS: SENOK DWRONG	

SEN_ACCE_XYZ_READ returns the acceleration of the robot in the 3 axes. Resolution is ±0.0156G/bit. Value 0 is -2G and 255 is 2G.

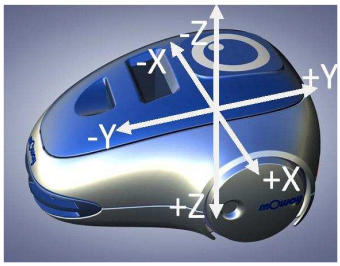


Image 38. Accelerometer axes

- Communication between microcontroller and accelerometer is SPI.
- Command is sent to change the mode of the accelerometer to “measure”.
- Function waits until the value is calculated.
- Value is read.
- Change the mode to “tap detection”.

Table 29. SEN_CHECK_ACCE allowed values.

Define	Value
ACCE_CHECK_X	0
ACCE_CHECK_Y	1
ACCE_CHECK_Z	2

SEN_ACCE_CHECK_TAP

Output variable	
SEN_ACCE_TAP	Detects taps
Output	
SEN_STATUS: SENOK DWRONG	

Accelerometer detects taps.

- Communication between microcontroller and accelerometer is SPI
- Checks if "tap interrupt" has been detected
- SEN_ACCE_TAP value is changed.

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Unused	Unused	Unused	Unused	Unused	Unused	TAP_TAP	TAP
-	-	-	-	-	-		

Bit 7-2: **Unused**

Bit 1: **TAP_TAP:** shows if double tap is detected
1 = Double Tap detected
0 = Double Tap not detected

Bit 0: **TAP:** shows if tap is detected
1 = Tap detected
0 = Tap not detected

SEN_LINE_DIG

Input variable	
SEN_CHECK_LINE	Which sensor must be read
Output variable	
SEN_LINE	Digital value of the sensor
Output	
SEN_STATUS: SENOK DWRONG	

The SEN_LINE_DIG function indicates whether the sensors are or are not on a dark surface. To achieve this follow the steps indicated below:

- Emit the infrared light pulse to detect the line. This light-beam will be reflected back if the line is detected and this signal will be perceived by the infrared receiver.
- Wait for the data acquisition process to end (900 us).
- Read the sensor.

- Copy the value to the SEN_LINE variable. If the surface is dark (no light is reflected) the variable will return a‘1’ value.

Table 30. SEN_CHECK_LINE allowed value

Define	Value
LINE_L	0
LINE_R	1

SEN_LINE_ANALOG

Input variable	
SEN_CHECK_LINE	Which sensor must be read
Output variable	
SEN_LINE	Analog value of the sensor
Output	
SEN_STATUS: SENOK DWRONG	

The SEN_LINE_ANALOG function indicates the light reflected in the optocouplers[15]. To do this follow the steps indicated below:

- Emit the infrared light pulse to detect the line. This light-beam will be reflected back if the line is detected and this signal will be perceived by the infrared receiver.
- Wait for the data acquisition process to end (900us).
- Read the sensor.
- Copy this value to the SEN_LINE variable. The higher the values the darker will the surfaces be.

Table 31. Allowed values for SEN_CHECK_LINE

Define	Value
LINE_L	0
LINE_R	1

LED_BRAKE_ON

This function switches on the brake LED.

LED_FRONT_ON

This function switches on the front LED.

LED_TOP_RED_ON

[15] Due to tolerance two different sensors can differ from each other.

This function switches on the red LED.

LED_TOP_GREEN_ON

This function switches on the green LED.

LED_BRAKE _OFF

This function switches off the brake LED.

LED_FRONT_OFF

This function switches off the front LED.

LED_TOP_RED_OFF

This function switches off the red LED.

LED_TOP_GREEN_OFF

This function switches off the green LED.

LED_BRAKE_ON_OFF

Blink brake LED.

LED_FRONT_ON_OFF

Blink front LED.

LED_TOP_RED_ON_OFF

Blink red LED.

LED_TOP_GREEN_ON_OFF

Blink green LED.

5.3.2. mOway´s motor library in assembly language

There are two libraries in assembly language which can be included in any mOway project and which allow the user to easily control the drive system. Both are

identical except that one of them can relocate the code and the variables (by means of MPLAB IDE projects).

It is essential to understand that every time a function library is called up it uses three additional call stack levels. This means that at least four call stack levels must be free before calling one of these functions to avoid return errors.

5.3.2.1. Description

The library includes a series of functions in charge of sending I2C commands to the Drive System, which will be responsible for controlling the motors and therefore releasing the main microcontroller so it can carry out other tasks.

Communications with the motor module are conducted via the I2C protocol. Any microcontroller with this kind of communications can control the motors; use the libraries in assembly. The format for the Driving System I2C frame can be observed in the following illustrations. Each of these frames lasts approximately 350 us.

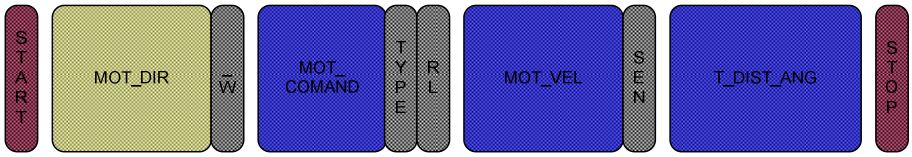


Image 39. Command format: MOT_STR, MOT_CHA_VEL

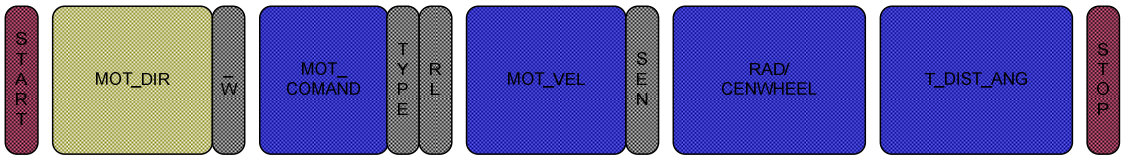


Image 40. Command format: MOT_CUR, MOT_ROT

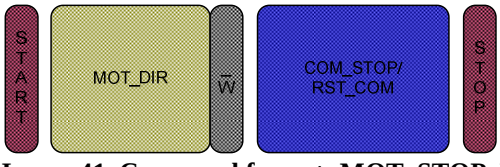


Image 41. Command format: MOT_STOP, MOT_RST

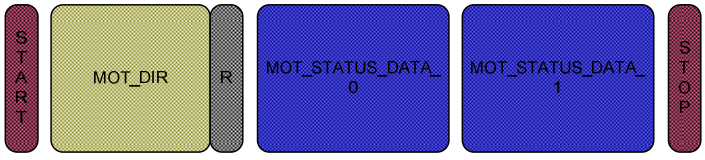


Image 42. Command format: MOT_FDBCK

5.3.2.2. Variables

MOT_STATUS

A register that shows the command's status.

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Unused	Unused	Unused	Unused	Unused	Unused	DWRONG	COMOK
-	-	-	-	-	-		

Bit 7-2: **Unused**

Bit 1: **DWRONG:** Appears if data is incorrect.
1 = Incorrect data.
0 = Correct Data.

Bit 0: **COMOK:** Appears if the command has been sent correctly by I2C.
1 = Correct dispatch.
0 = Incorrect dispatch.

MOT_CON

Control register. This register defines command parameters.

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Unused	Unused	Unused	Unused	Unused	COMTYPE	RL	FWDBACK
-	-	-	-	-			

Bit 7-3: **Unused**

Bit 2: **COMTYPE:** Type of command.
1 = Time.
0 = Distance or angle (in MOT_ROT).

Bit 1: **RL:** Right or Left
1 = Right.
0 = Left.

Bit 0: **FWDBACK:** Forward or backwards.
1 = Forward.
0 = Backwards.

MOT_VEL

Desired command speed.

MOT_T_DIST_ANG

According to the COMTYPE and command values, the variable will be the time, the distance or the angle.

MOT_CENWHEEL

Rotate on the robot's center or on one of the wheels.

MOT_RAD

Radius for the MOT_CUR command.

MOT_RST_COM

Type of reset desired.

MOT_STATUS_COM

Type of motor data to be read.

MOT_STATUS_DATA_0-1

These two variables store the value required by the MOT_FDBCK function.

5.3.2.3. Functions

A series of functions designed to control mOway's drive system are included in the lib_mot_moway and lib_re_mot_moway libraries.

Table 32. Summary of assembly languague functions for lib_mot_moway

Name	Input	Return	Description
MOT_CONFIG	-	-	Configuration to communicate with the motors
MOT_STR	MOT_VEL MOT_T_DIST ***MOT_CON*** FWDBACK COMTYPE	***MOT_STATUS*** COMOK DWRONG	A command to move in a straight line
MOT_CHA_VEL	MOT_VEL MOT_T_DIT ***MOT_CON*** FWDBACK COMTYPE RL	***MOT_STATUS*** COMOK DWRONG	A command to change the speed of a motor
MOT_ROT	MOT_VEL MOT_CENWHEEL MOT_T_ANG ***MOT_CON*** FWDBACK COMTYPE RL	***MOT_STATUS*** COMOK DWRONG	A command to rotate the robot

MOT_CUR	MOT_VEL MOT_RAD MOT_T_DIST ***MOT_CON*** FWDBACK COMTYPE RL	***MOT_STATUS*** COMOK DWRONG	A command to execute a curve
MOT_STOP	-	***MOT_STATUS*** COMOK DWRONG	A command to stop the robot
MOT_RST	MOT_RST_COM	***MOT_STATUS*** COMOK DWRONG	A command to reset the temporary variables for time and distance
MOT_FDBCK	STATUS_COM	MOT_STATUS_ DATA_0 MOT_STATUS_ DATA_1 ***MOT_STATUS*** COMOK DWRONG	A command to determine the motor's status

MOT_CONFIG

This function configures the inputs and outputs so the microcontroller can communicate with the Drive System.

Table 33. Pic-drive system connections

Pin PIC	I/O	Sensor
PORTE		
RE7	I	Indicates when the motor ends the command.
RE0	O	SCL of the I2C protocol
RE1	O	SDA of the I2C protocol

Port RE7 indicates the end of a command. This port is labeled as MOT_END in the library.

Example:

```
;Straight forward at 100% speed for 10 seconds (100ms x 100)
movlw       .100
movwf       MOT_VEL
movlw       .100
movwf       MOT_T_DIST_ANG
bsf         MOT_CON,FWDBACK
bsf         MOT_CON,COMTYPE
call        MOT_STR

;Nothing is done until the command has ended
CHECK_COMMANDO_END
btfss       MOT_END
goto        CHECK_COMMANDO_END
```

MOT_CHECK_END function also can be used.

MOT_STR

Input			
MOT_VEL	Desired speed	0	100
MOT_CON, FWDBACK	Movement direction	1-FWD	0-BACK
MOT_CON, COMTYPE	Type of command	1-TIME	0-DIST
MOT_T_DIST	Time value Distance value	0 0	255 255
Output variables			
FLAGS MOT_STATUS: COMOK and DWRONG			

Command to move in a straight line. You will have to specify speed, direction, type of command and the time or the distance to travel. The time has a resolution of 100 ms and the distance of 1mm and with a value of 0 returned by MOT_T_DIST the command shall be maintained until another order is specified.

Example:

```
;Straight ahead at 100% speed during 10 seconds (100ms x 100)
movlw       .100
movwf       MOT_VEL
movlw       .100
movwf       MOT_T_DIST_ANG
bsf         MOT_WITH,FWDBACK
bsf         MOT_WITH,COMTYPE
call        MOT_STR

;Straight backwards at 15% speed 100mm (1mm x 100)
movlw       .15
movwf       MOT_VEL
movlw       .100
movwf       MOT_T_DIST_ANG
bcf         MOT_CON,FWDBACK
bcf         MOT_CON,COMTYPE
call        MOT_STR
```

MOT_CHA_VEL

Input			
MOT_VEL	Desired speed	0	100
MOT_CON, FWDBACK	Movement direction	1-FWD	0-BACK
MOT_CON, RL	Left or right	1-RIGHT	0-LEFT
MOT_CON, COMTYPE	Type of command	1-TIME	0-DIST
MOT_T_DIST	Time value Distance value	0 0	255 255

Output variables
FLAGS MOT_STATUS: COMOK and DWRONG

A command to change the speed of any of the two motors. You will have to specify speed, direction, motor, type of command and the time or distance to cover. The time has a resolution of 100 ms and the distance 1 mm, and with a value of 0 returned by MOT_T_DIST the command shall be maintained until another order is specified.

Example:

```
;Change speed (80% forward) of the right motor for 10 seconds
;(100ms x 100)
movlw       .80
movwf       MOT_VEL
movlw       .100
movwf       MOT_T_DIST_ANG
bsf         MOT_CON,FWDBACK
bsf         MOT_CON,COMTYPE
bsf         MOT_CON,RL
call        MOT_CHA_VEL

;Change speed (20% backwards) of the left motor and cover a distance of 100 mm
;(1mm x 100)
movlw       .20
movwf       MOT_VEL
movlw       .100
movwf       MOT_T_DIST_ANG
bcf         MOT_CON,FWDBACK
bcf         MOT_CON,COMTYPE
bcf         MOT_CON,RL
call        MOT_CHA_VEL
```

MOT_ROT

Input			
MOT_VEL	Desired speed	0	100
MOT_CON, FWDBACK	Movement direction	1-FWD	0-BACK
MOT_CENWHEEL	On the center or on the wheel	0x01-CE	0x00-WH
MOT_CON, RL	Right or left	1-RIGHT	0-LEFT
MOT_CON, COMTYPE	Type of command	1-TIME	0-ANG
MOT_T_ANG	Time value Angle value	0 0	255 100
Output variables			
FLAGS MOT_STATUS: COMOK and DWRONG			

Command to make the mOway rotate. It is necessary to specify speed, direction, type of rotation, the motor, type of command and the time or the rotation angle. The time has a resolution of 100ms and with a value of 0 returned by MOT_T_ANG the command shall be maintained until another order is specified.

Regarding the angle, the following equations show how to calculate the value of MOT_T_ANG taking into account the desired rotation angle. If the rotation is produced on one of the wheels more resolution is obtained. On the other hand, mechanical inertia has to be considered, therefore it is advisable to reduce the speed to achieve greater precision.

Equation 1. MOT_T_ANG when rotating on its center

$$MOT\_T\_ANG = round\left(\frac{Angle^{o} \times 3.33}{12^{o}}\right)$$

Example:

```
;Rotate in relation to the center, to the right, at 80% speed for 10 seconds
;(100ms x 100)
movlw       .80
movwf       MOT_VEL
movlw       .100
movwf       MOT_T_DIST_ANG
movlw       0x01
movwf       MOT_CENWHEEL
bsf         MOT_CON,FWDBACK
bsf         MOT_CON,COMTYPE
bsf         MOT_CON,RL
call        MOT_ROT

;Rotate on the left wheel forward at 20% speed 180º
movlw       .20
movwf       MOT_VEL
movlw       .50
movwf       MOT_T_DIST_ANG
movlw       0x00
movwf       MOT_CENWHEEL
bsf         MOT_CON,FWDBACK
bcf         MOT_CON,COMTYPE
bcf         MOT_CON,RL
call        MOT_ROT
```

MOT_CUR

Input			
MOT_VEL	Desired speed	0	100
MOT_CON, FWDBACK	Movement direction	1-FWD	0-BACK
MOT_RAD	Radius	0	100
MOT_CON, RL	Right or left	1-RIGHT	0-LEFT
MOT_CON, COMTYPE	Type of command	1-TIME	0-DIST
MOT_T_DIST	Time value Distance value	0 0	255 255
Output variables			
FLAGS MOT_STATUS: COMOK and DWRONG			

Command to describe a curve. It is necessary to specify speed, direction, radius, course, type of command and the time or the distance to cover. The radius is the speed which shall be subtracted or added to the robot's global speed. This means that if the specified speed is 50 and the radius 10, one of the motors shall work at 60% speed and the other one 40%. Therefore the radius has to adhere to the following restrictions:

Equation 2. Condition 1 MOT_RAD

$$0 \leq MOT\_VEL - MOT\_RAD \leq 100$$

Equation 3. Condition 2 MOT_RAD

$$0 \leq MOT\_VEL + MOT\_RAD \leq 100$$

The time has a resolution of 100ms and the distance 1.7mm, and with a value of 0 returned by MOT_T_ANG the command shall be maintained until another order is specified.

The speedometer counts the distance traveled by the motor located on the external side of the curve.

Example:

```
;Curve forward to the right at 50% with a radius of 10 during 10 seconds
;(100ms x 100)
;VEL_I=60
;VEL_D=40
movlw       .50
movwf       MOT_VEL
movlw       .100
movwf       MOT_T_DIST_ANG
movlw       .10
movwf       MOT_RAD
bsf         MOT_CON,FWDBACK
bsf         MOT_CON,COMTYPE
bsf         MOT_CON,RL
call        MOT_CUR
```

```
;Curve forward to the left at 80% with a radius 15 during 100mm
;(1mm x 100)
;VEL_I=95
;VEL_D=65
movlw       .80
movwf       MOT_VEL
movlw       .100
movwf       MOT_T_DIST_ANG
movlw       .15
movwf       MOT_RAD
bcf         MOT_CON,FWDBACK
bcf         MOT_CON,COMTYPE
bcf         MOT_CON,RL
call        MOT_CUR
```

MOT_CHECK_END

Function that waits until the movement command finishes.

Example:

```
;Wait the end of the command
call    MOT_CHECK_END
```

MOT_STOP

Output variables
FLAGS MOT_STATUS: COMOK

A command to stop the robot.

Example:

```
;Stop the mOway
call    MOT_STOP
```

MOT_RST

Input		
MOT_RST_COM	The parameter that needs to be reset	RST_T RST_DIST RST_KM
Output variables		
FLAGS MOT_STATUS: COMOK		

Resets the motor's internal time, distance and speedometer temporary variables.

Example:

```
;Reset elapsed time
movlw       RST_T
movwf       MOT_RST_COM
call        MOT_RST

;Reset distance traveled
movlw       RST_D
movwf       MOT_RST_COM
call        MOT_RST
```

MOT_FDBCK

Input		
STATUS_COM	The parameter we want to look up	STATUS_T STATUS_A STATUS_V_R STATUS_V_L STATUS_D_R STATUS_D_L STATUS_KM
Output variables		
MOT_STATUS_DATA_0	First response byte (time, angle, speed, distance and first speedometer byte)	
MOT_STATUS_DATA_1	Second response byte (second speedometer byte)	
FLAGS MOT_STATUS: COMOK and DWRONG		

Command to recall different drive system parameters. We can look up elapsed time, angle (only through the MOT_ROT command), speed of each motor, distance traveled by each motor and the speedometer.

This function updates two variables where the required information will be saved. All the petitions except STATUS_KM return one byte (MOT_STATUS_DATA_0) maintaining MOT_STATUS_DATA_1 at a 0xFF value. These two variables are updated every time a new command is sent (e.g. recall the time elapsed since the last command). Whenever using STATUS_KM the two bytes must be considered. This command is very useful to calculate the length of a line while the robot is following it.

Table 34. Parameter resolution

Parameter	Resolution
STATUS_T	100ms/bit
STATUS_A	3.6º/bit

STATUS_V_R	1%/bit
STATUS_V_L	1%/bit
STATUS_D_R	1mm/bit
STATUS_D_L	1mm/bit
STATUS_KM	1mm/bit

Example:

```
;Recall time elapsed since the last command
movlw       STATUS_T
movwf       MOT_STATUS_COM
call        MOT_FDBCK
;E.g. Output:
;MOT_STATUS_DATA_0=0x7F   => 12.7 seconds elapsed since the
;last command
;MOT_STATUS_DATA_1=0xFF;  => Invalid data
```

```
;Recall distance traveled by the right motor since the last command
```

byte 1	byte 0
0x01	0x08
0000 0001	0000 0100
264	
Distance: 264*1mm	
264mm	

5.3.3. BZI-RF2GH4 library in assembly language

5.3.3.1. Description

With this library it is possible to communicate easily between mOway and the BZI-RF2GH4 module.

In turn it is important to take into account that in order to call any library function, three free stack levels are necessary and the “watchdog” must be deactivated.

In view of the fact that all the functions use the SPI protocol, it is necessary to enable the microcontroller hardware for this purpose. To do this, just add a few lines of code in the initial configuration of programme.

5.3.3.2. Variables

RF_STATUS

This read-only variable reports on the communications situation via the radio module.

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Unused	CONFIGOK	OFFOK	ONOK	RCVNW	RCVOK	ACK	SNDOK
-							

Bit 7: **Unused**

Bit 6: **CONFIGOK:** Shows whether the module has been configured correctly.
1 = The module has been configured correctly.
0 = Module has been de-configured. Communications with the module impossible due to the absence of or incorrect electrical connection.

Bit 5: **OFFOK:** Shows whether the module has been switched off correctly.
1 = The module has been switched off correctly.
0 = The module has not been switched off correctly. Communications with the module impossible due to the absence of, or incorrect electrical connection.

Bit 4: **ONOK:** Shows whether the module has been switched on correctly.
1 = The module has been switched on correctly.
0 = The module is not active. Communications with the module impossible due to the absence of, or incorrect electrical connection.

Bit 3: **RCVNW:** Shows whether there is still data to be read.
1 = There are data frames to be read in the radio module stack.
0 = After the last reading, the module data stack was empty. There are no pending messages.

Bit 2: **RCVOK:** Reports that data has been received correctly and is accessible for processing.
1 = Correct reception.
0 = No data has been received or the information received is corrupt.

Bit 1: **ACK:** Shows whether the ACK (confirmation) has been received from the receiver following a transmission.
1 = The receiver has confirmed that the data has been received correctly.
0 = Confirmation from the receiver has not been received. This may be due to the fact that the signal has not been received or that the stack is full and cannot store more messages.

Bit 0: **SNDOK:** This shows whether data was sent the last time.

1 = Radio module has sent the data. This bit does not indicate that someone has heard it.
0 = It has not been possible to send the data. This may be due to a failure in the communication with the radio module.

RF_DATA_OUT_0, RF_DATA_OUT _1,… RF_DATA_OUT _7

This group of variables consists of 8 bytes. In each transmission the contents of the 8 bytes is sent.

RF_DATA_IN_0, RF_DATA_IN _1,… RF_DATA_IN _7

This group of variables consists of 8 bytes. In each reception these 8 bytes are updated.

RF_DIR_OUT

This variable is of one byte only. This indicates the direction of the device that wants to send the data.

RF_DIR_IN

This variable is of one byte only. It indicates the address of the data received.

RF_DIR

This variable is of one byte only. It indicates the address with which the module is configured.

5.3.3.3. Functions

The library consists of nine functions that will make the task of developing a communications application with **BZI-RF2GH4** modules considerably easier. A brief description of each one of these functions is given below.

Table 35. Assembler RF functions.

Functions for the BZI-RF2GH4 module	
RF_CONFIG	Configures the inputs and outputs of the microcontroller as well as the radio module parameters.
RF_CONFIG_SPI	Configures the inputs and outputs of the microcontroller as well as the parameters required to use the SPI bus.
RF_ON	Activates the radio frequency module in receive mode.
RF_OFF	Deactivates the radio frequency module and leaves it in low

	consumption mode.
RF_SEND	Sends a data frame (8 Bytes) to the address indicated.
RF_RECEIVE	Checks whether a reception has occurred and if so, collects the frame.
RF_RECEIVE_INT	Carries out the same function as RF_RECEIVE but in this case checks whether there has been an interruption.
RF_INT_EN	This routine enables the external interruption for the radio module in the microcontroller.

RF_CONFIG_SPI

The speed of the SPI must not exceed 8 Mhz and therefore the use of this function is limited to PIC microcontrollers with a frequency of less than 32Mhz. The different parameters of the SPI module and the pins of the PIC are configured in the function.

Table 36. SPI configuration

PIN RF	PIN PIC
SCK	RC3
SDI	RC5
SDO	RC4

RF_CONFIG

Input variables	
RF_DIR	Device address. Must be a value of between 0x01 and 0xFE.
RF_CHN	Channel to be used in the communication. Must be a value of between 0x00 and 0x7F (128 channels).
Output variables	
FLAGS: CONFIGOK	

This function configures the transceptor, enabling its own watch address and the 'broadcast' address. In trun, it configures other parameters such as the PIC pins, the channel, the transmission speed, the emitting power, the address length, the CRC code length, etc.

Table 37. RF pin configuration

PIN RF	PIN PIC
IRQ	RB0
CSN	RF2
CE	RH4

The channel must be common to all the modules that are going to take part in the communication. Users can choose any channel from among the 128 available. Nevertheless, if there is more than one communication in the environment between modules in different channels, a spacing of 2 must be left between the channels to be used in order to avoid interferences, thus leaving 32 channels usable. Another question

to be taken into account is the existence of other technologies that use the ISM 2.4GHz band (Wi-Fi, Bluetooth, etc.) and that might also cause interference in one of the channels.

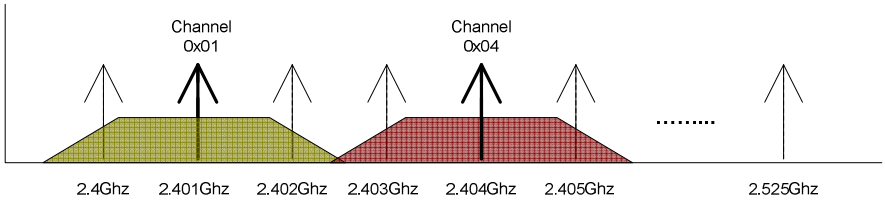


Image 43. RF channels

The address assigned to each device must be one-way within each channel.

If the SPI communication of the PIC is configured incorrectly, the connection has been made incorrectly or in the absence of the module; the CONFIGOK RF_STATUS bit of will remain at 0.

RF_ON

Output variables
FLAGS: ONOK

This routine activates the radio module in watch mode in order to be able to receive data and/or send data.

It is important to take into consideration that, following the call to this routine, the module requires 2.5 ms to be ready.

If the SPI communication of the PIC is configured incorrectly, the connection has been made incorrectly or in the absence of the module; the ONOK RF_STATUS bit will remain at 0.

Example:

```
;--[Configuration without interruption and activation routine]-----
  ;Configure SPI modules of the PIC
  call          RF_CONFIG_SPI

  ; Configure RF module (own channel and address)
  movlw         0x01          ; Own address
  movwf         RF_DIR

  movlw         0x40          ;Channel
  movwf         RF_CHN

  call          RF_CONFIG
  btfss         RF_STATUS,CONFIGOK
```

```
    nop                 ;Module not configured


    ; Activate RF module
    call        RF_ON
    btfss       RF_STATUS,ONOK
    nop                 ;Module not initialised
;-------------------------------------------------------------------------
```

RF_OFF

Output variables
FLAGS: OFFOK

This routine deactivates the radio module leaving this in low consumption mode. It does not clear the established configuration.

If the SPI communication of the PIC is configured incorrectly, the connection has been made incorrectly or in the absence of the module; the OFFOK RF_STATUS bit will remain at 0.

RF_SEND

Input variables	
RF_DIR_OUT	Direction to which it is required to send the data (1 byte).
RF_DATA_OUT_0 – RF_DATA_OUT_7	Variables to be transmitted (8 bytes).
Output variables	
FLAGS: SNDOK and ACK	

This function sends 8 bytes of data to the indicated address and reports the correct reception to the recipient. Following this, the device will return to watch mode.

If a message is sent to the address 0x00, this will be received by all the modules on the same channel. It must be taken into account that the module accepts the first ACK it receives, therefore we cannot be certain that the data has arrived at all the devices.

Example:

```
;---------------------[Data sending routine]-----------------------
   ; Preparation of the receiver address
   ; and of the data.

  movlw         0x02                ;Receiver address
```

```
movwf       RF_DIR_OUT
                                ;Data to be sent
clrf        RF_DATA_OUT_0
clrf        RF_DATA_OUT_1
clrf        RF_DATA_OUT_2
clrf        RF_DATA_OUT_3
clrf        RF_DATA_OUT_4
clrf        RF_DATA_OUT_5
clrf        RF_DATA_OUT_6
clrf        RF_DATA_OUT_7

call        RF_SEND             ;Send frame
btfss       RF_STATUS,SNDOK
nop                             ;Not sent
btfss       RF_STATUS,ACK
nop                             ;No ACK
```

```
;-------------------------------------------------------------------------
```

RF_RECEIVE

Output variables	
RF_DIR_IN	Address of the person who has sent the frame
RF_DATA_IN_0 – RF_DATA_IN_7	Frame received from the address indicated.
RCVOK and RCVNW	

This routine is responsible for checking whether a reception has taken place and if so, it returns the data received. Likewise, it reports whether there is any data that has not been read in the reception FIFO of the module.

When a frame is received, the RCVNW bit of the RF_STATUS variable must be checked and if this is active, the RF_RECEIVE function must be called up once again after processing the data. The transceptor has a 3-level stack, and therefore if the receive function is not called before the stack is filled, the device will be unable to receive more data.

As interruptions are not used, the probability of losing packages, with high traffic levels, is moderate. It is advisable to use this only in environments in which there are just a few devices and/or little data traffic. This problem can also resolved by causing the images to resend the same frame until the communication is correct, but in environments with a great deal of traffic, collisions increase exponentially, causing considerable increases in sending times.

Example:

```
;-----[ Reception routine without interruption]----------
```

```
RECEIVE_DATA
  call          RF_RECEIVE
  btfsc         RF_STATUS,RCVOK
  nop                                   ;Procces data
  btfsc         RF_STATUS,RCVNW
  goto          RECEIVE_DATA
;-------------------------------------------------------------------------
```

RF_RECEIVE_INT

Output variables	
RF_DIR_IN	Address of the person who has sent the frame
RF_DATA_IN_0 – RF_DATA_IN_7	Frame received from the address indicated.
RCVOK, RCVNW	

This is the optimum reception routine. This routine is virtually the same as RF_RECEIVE, the difference being that this one operates by interruption. For this reason, it must be placed within the interaction code and the interruptions must be configured beforehand (RF_INTER_EN). It is responsible for checking that an external interruption has occurred (RB0) and if so, it clears the interruption flag. The probability of losing packages is minimal. Even so, it is advisable for transmitters to resend if the send flag is not activated.

Example:

```
;--------- -[Data reception routine with interruption]---------
READ_MORE_DATA
  call          RF_RECEIVE_INT
  btfsc         RF_STATUS,RCVOK
  nop                                   ; Process data
  btfsc         RF_STATUS,RCVNW
  goto          READ_MORE_DATA
  goto          INTERRUPTION_OUT
;-------------------------------------------------------------------------
```

RF_INT_EN

This routine is responsible for enabling the external interruption of the microcontroller (RB0) that uses the RF module in data reception. For this reason, the RB0 pin is configured as input. Although the module can be managed without interruptions, the minimum response time is not guaranteed.

Example:

```
;--[Configuration with interruption and activation routine]-----
  ; Enable interruptions
  call          RF_INT_EN

  ;Configure SPI modules of the PIC
  call          RF_CONFIG_SPI

  ; Configure RF module (own channel and address)
  movlw         0x01          ; Own address
  movwf         RF_DIR

  movlw         0x40          ;Channel
  movwf         RF_CHN

  call          RF_CONFIG
  btfss         RF_STATUS,CONFIGOK
  nop                 ;Module not configured

  ; Activate RF module
  call          RF_ON
  btfss         RF_STATUS,ONOK
  nop                 ;Module not initialized
;-------------------------------------------------------------------------
```

5.3.3.4. Flow diagram for sending and receiving data

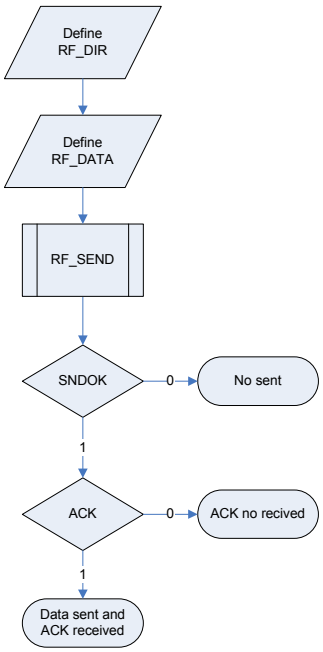


Diagram 1. Sent data in assembler

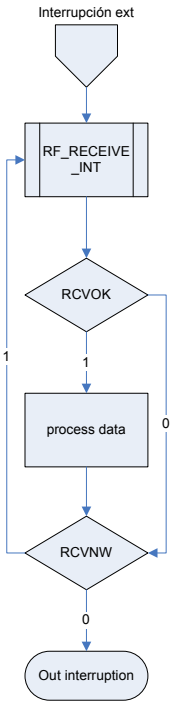


Diagram 2. Receive interruption in assembler

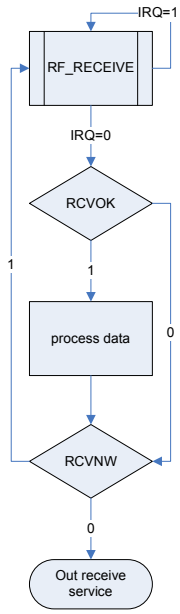


Diagram 3. Reception in assembler

6. Programming Moway with C18 Compiler

C18 is a compiler that can be acquired in the market and which supports the PIC18F86J50 microcontroller. In the Moway Website the libraries required to manage sensors, motors and RF modules, written for the compiler.

Its greatest advantage is that it compiles in C language. Managing numerical variables (char, int, etc.) and flow controlling structures (if, for, etc.) is very simple and it includes many pre-compiled functions which greatly assist programming (I2C, SPI). However, the size of the generated programs is larger than with assembly language.

To summarize:

- Very interesting if you wish to start working with Moway quickly.
- Very interesting to carry out easy or average difficulty tasks.
- Inadequate for programs with large coding.
- Inadequate for critical response timeframes

6.1. Creating a project

Use the MPLAB IDE Project Wizard to create the first project quickly. C18 compiler has to be installed. These example is made by MPlab v8.3.

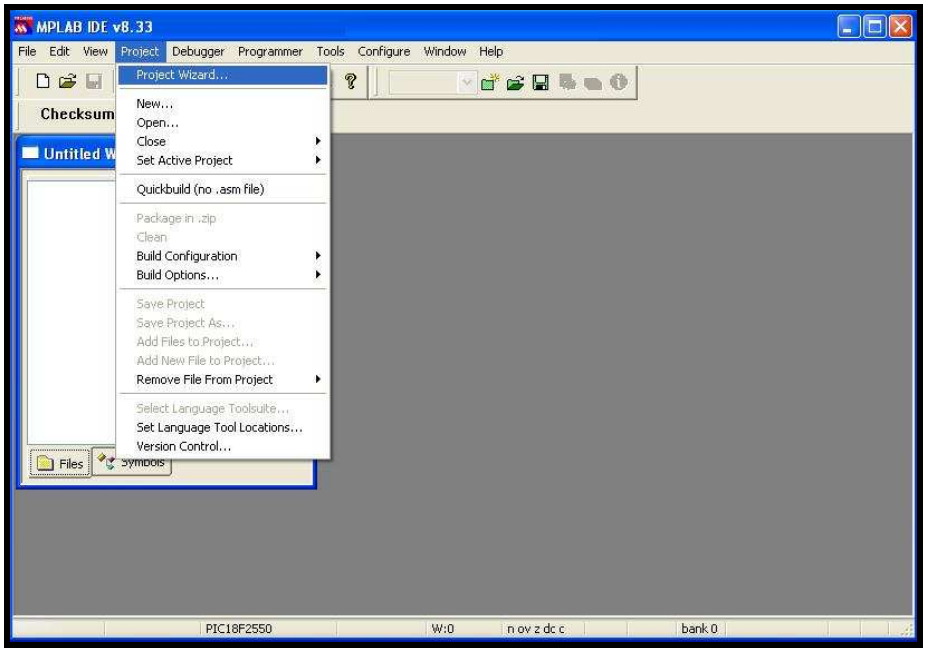


Image 44. Project Wizard

1. First select the PIC installed in mOway: PIC18F86J50.

Image 45. PIC selection

2. Choose C18 C compiler.

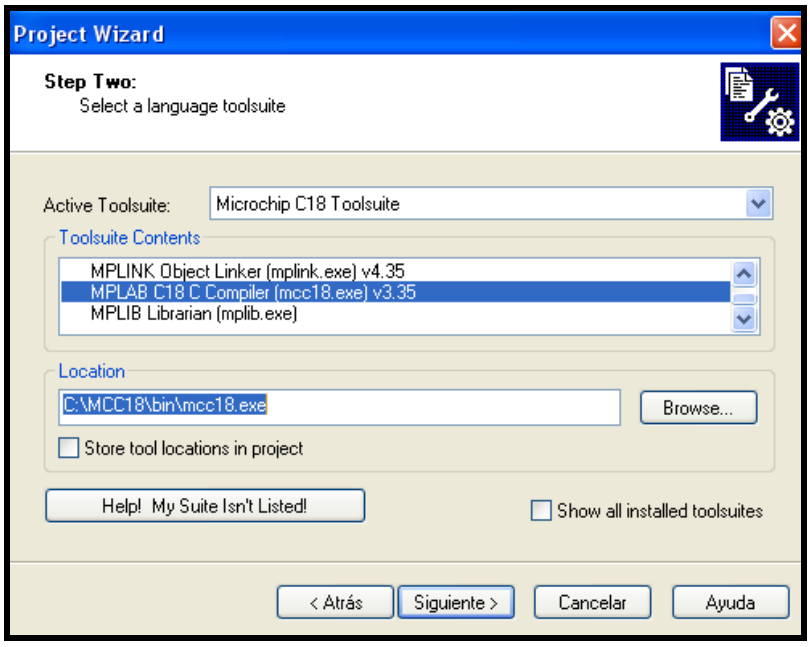


Image 46. Tool selection

3. In the next step user has to specify location.

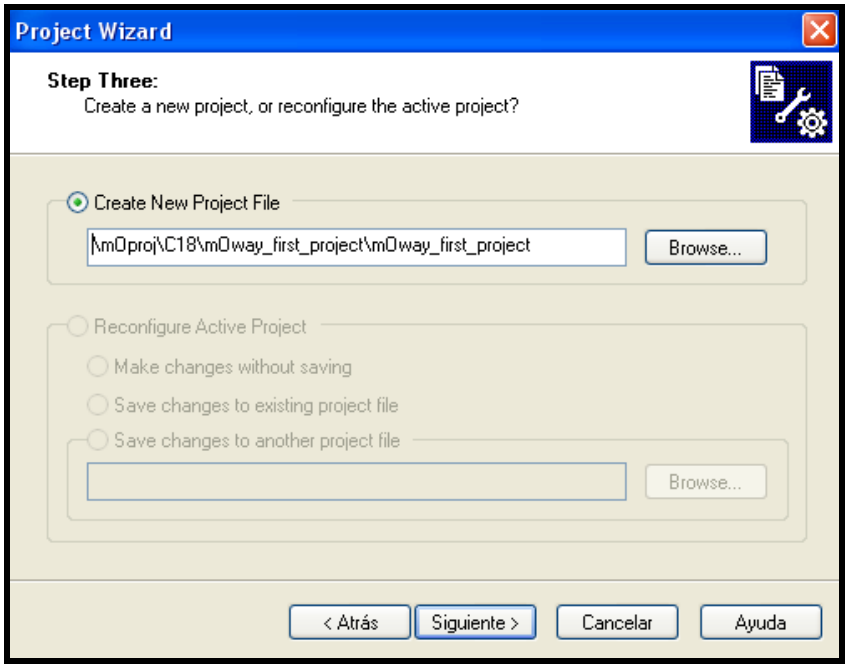


Image 47. Location

4. Add mOway libraries. It is highly recommended to copy those libraries to the folder.

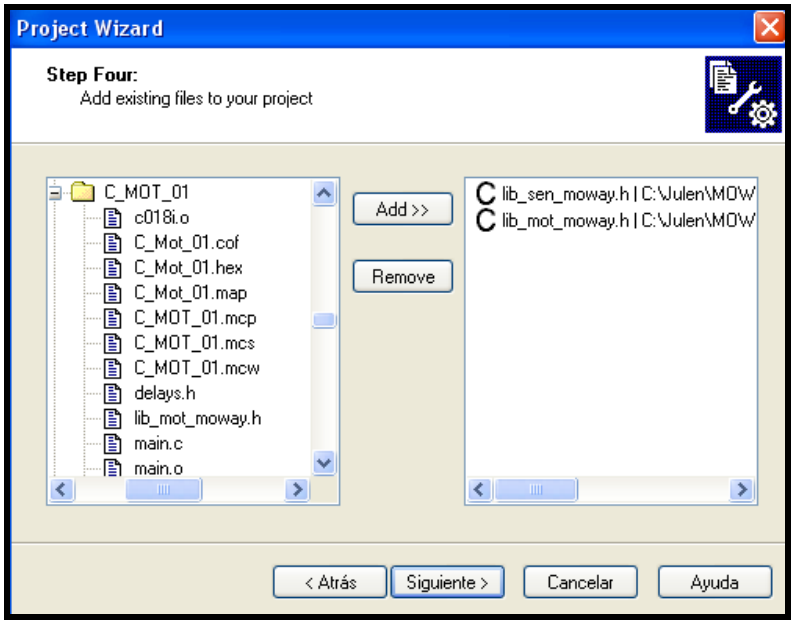


Image 48. Add libraries

8. With the steps above the project will now be created, the next step is to create a .C file for the source code.

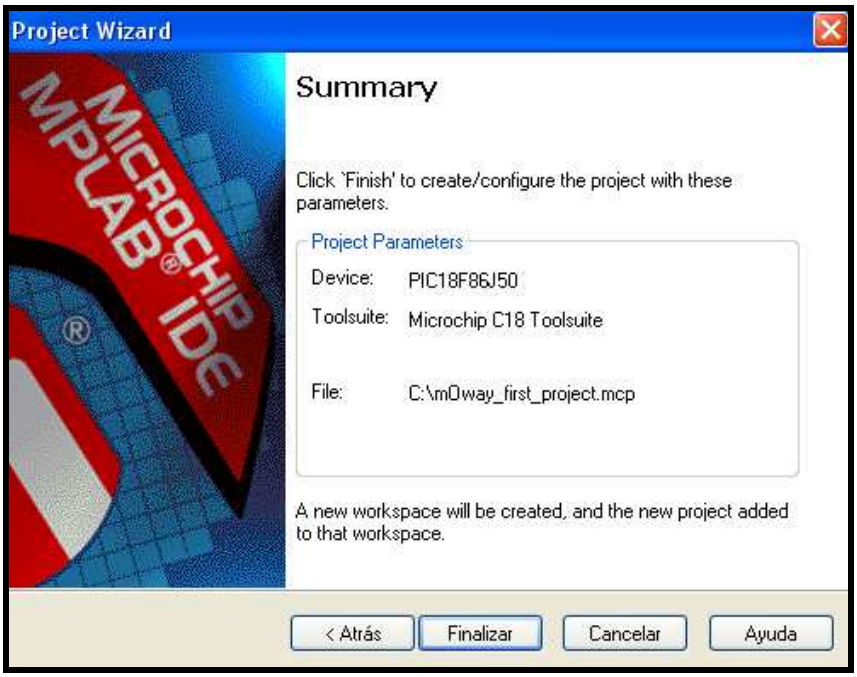


Image 49. Wizard ends

9. The next step is to open the project and create a new file (*New File*) saving it in the same folder of the project as *Main.c*. This will be our source file.

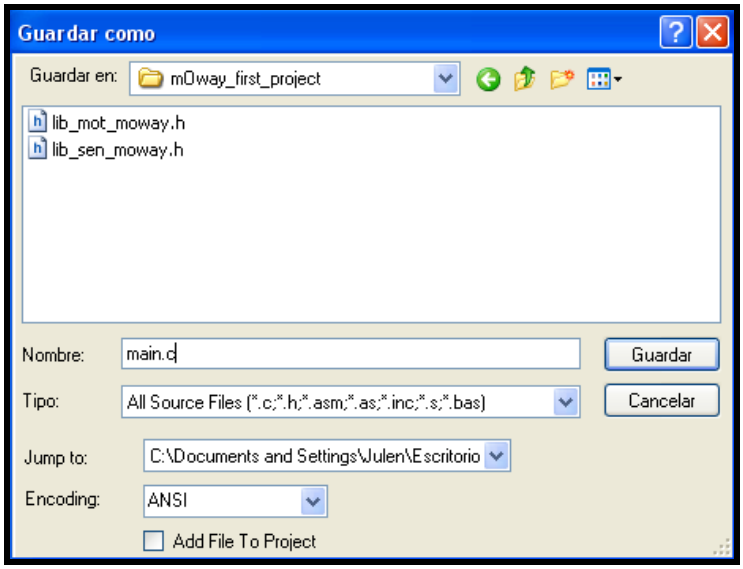


Image 50. .C creation

10. Finally, the source file is added to the project accessing *Project/Add Files to Project…* After that user has to add Linker Script to the project. This can find y mOwayPack o in another example project.

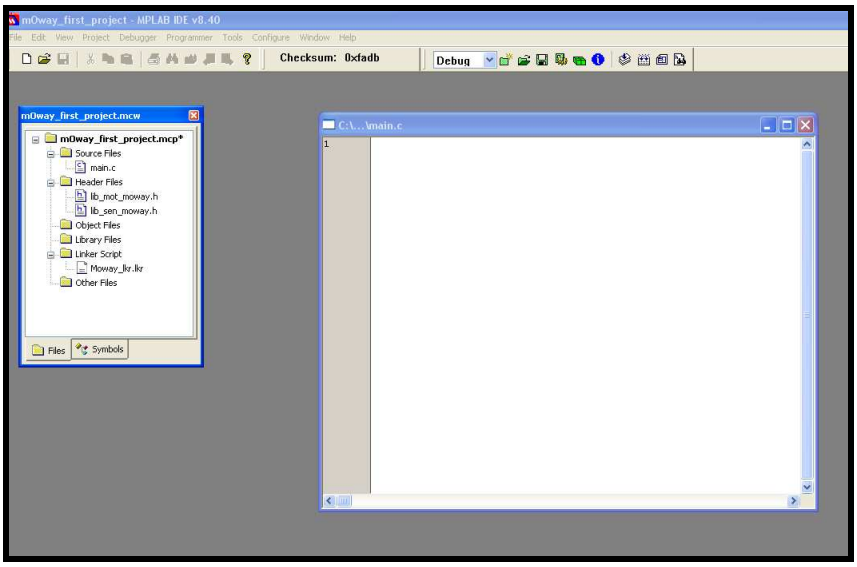


Image 51. Project with .C

6.2. First program in C18

To generate the first program a project has to be created first (previous chapter). This first basic program will enable the mOway to avoid obstacles.

1. Add code to redefinition of the reset and interrupt codes. This code is required. Also lib_sen_moway.h is added.

```
#include "p18f86j50.h"          //Moway microcontroller
#include "lib_sen_moway.h"      //Sensors library


//*************************************************************************************************
// JUMPING THE BOOTLOADER
//*************************************************************************************************
#define REMAPPED_RESET_VECTOR_ADDRESS           0x1000  //Reset adress for the correct bootloade
#define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS  0x1008  //High priority interruption adress for
#define REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS   0x1018  //Low priority interruption adress for t


void YourHighPriorityISRCode(); //Function that executes the needed code in case of the high pri
void YourLowPriorityISRCode();  //Function that executes the needed code in case of the low prio


//*********************RESET, HIGH AND LOW PRIORITY INTERRUPTION REMAP************************
extern void _startup (void);
#pragma code REMAPPED_RESET_VECTOR = REMAPPED_RESET_VECTOR_ADDRESS
void _reset (void)
{
    _asm goto _startup _endasm
}
#pragma code

#pragma code REMAPPED_HIGH_INTERRUPT_VECTOR = REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS
void Remapped_High_ISR (void) { _asm goto YourHighPriorityISRCode _endasm }

#pragma code REMAPPED_LOW_INTERRUPT_VECTOR = REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS
void Remapped_Low_ISR (void) { _asm goto YourLowPriorityISRCode _endasm }

#pragma code

#pragma interrupt YourHighPriorityISRCode
void YourHighPriorityISRCode() {
    //high priority interrupt code
}

#pragma interruptlow YourLowPriorityISRCode
void YourLowPriorityISRCode() {
    //low priority interrupt code
}


//***********************************[MAIN]************************************************
void main()
```

Image 52. First program: vector redefinition.

2. Next, the SEN_CONFIG function is called to configure the microcontroller's inputs and outputs.
3. Add winking to one of the LEDs.
4. Test the program on mOway programming it in mOwayGUI and verify that the green LED blinks.

```
//*************************************[MAIN]*****************************************
void main()
{

    //**********SENSOR CONFIGURATION***********//
    SEN_CONFIG();

    //Green LED blink
    LED_TOP_GREEN_ON_OFF();

    while(1){









    }
}
```

Image 53. First program: configuration and LED

5. To detect obstacles call up the SEN_OBS_DIG() with OBS_CENTER_L as input value.
6. If it detects an obstacle the front LEDs light up.
7. Test the program on mOway and verify that the LEDs switch on when an object is placed close to the front part of the mOway.

```
//*************************************[MAIN]*****************************************
void main()
{

    //**********SENSOR CONFIGURATION************//
    SEN_CONFIG();

    //Green LED blink
    LED_TOP_GREEN_ON_OFF();

    while(1){

    //Check Center Left Obstacle sensor
    if (SEN_OBS_DIG(OBS_CENTER_L)){
        LED_FRONT_ON();
    }
    else{

        LED_FRONT_OFF();
    }


    }
}
```

Image 54. First program: detecting obstacles

8. We then add movement to the robot: unrestricted straight command until it encounters an obstacle.
9. lib_mot_asm.inc is added to the project.
10. MOT_CONFIG is called to be able to use Diver system.
11. Go straight on the first time.
12. When it encounters an obstacle a command is sent to rotate 180º and the top red LED lights up (the front LEDs will not operate). The robot will wait until this command has ended and will then continue moving straight forward.

```
//************************************[MAIN]*************************************
void main()
{

    //**********SENSOR CONFIGURATION***********//
    SEN_CONFIG();
    //**********ENGINE CONFIGURATION***********//
    MOT_CONFIG();
    //Green LED blink
    LED_TOP_GREEN_ON_OFF();
    //Straight on
    MOT_STR(50, FWD, TIME, 0);
    while(1){

    //Check Center Left Obstacle sensor
    if (SEN_OBS_DIG(OBS_CENTER_L)){
        LED_FRONT_ON();
        //Rotation
        MOT_ROT(20, FWD, CENTER, LEFT, ANGLE, 50) ;
        while(!MOT_END){};
        //Straight on
        MOT_STR(50, FWD, TIME, 0);
    }
    else{

        LED_FRONT_OFF();
    }


    }
}
```

Image 55. First program: detecting obstacles moving

This project is included in the mOway pack.

6.3. Libraries

6.3.1. mOway´s sensors library in C18

6.3.1.1. Description

The library includes a series of functions in charge of reading the data provided by the robot's sensors. They configure the input and output ports, the microcontroller's ADC and the luminous indicators.

6.3.1.2. Functions

A series of functions to control mOway's sensors and LED diodes are included in the lib_sen_moway library.

Table 38. C function summary

Name	Input constants	Description
void SEN_CONFIGURAR(*void*)	-	Configured to use the sensors.
unsigned char SEN_LIGHT*(void)*	-	Reads light sensor values.
unsigned char SEN_BATTERY*(void)*	-	Returns the battery level.
unsigned char SEN_TEMPERATURE *(void)*	-	Detects the temperature in ºC.
unsigned char SEN_MIC_ANALOG *(void)*	-	Detects sound intensity.
unsigned char SEN_MIC_DIG *(void)*	-	Detects if there is sound or not.
unsigned char SEN_SPEAKER*(unsigned char, unsigned char, unsigned char)*	SPEAKER_OFF SPEAKER_ON SPEAKER_TIME	Emits tones in a frequency between 250 Hz and 65 KHz.
unsigned char SEN_ACCE_XYZ_READ(unsigned char)	ACCE_CHECK_X ACCE_CHECK_Y ACCE_CHECK_Z	Calculates the X,Y,Z axes acceleration of mOway.
unsigned char SEN_ACCE_CHECK_TAP*(void)*	-	Detects if mOway has been taped.
unsigned char SEN_OBS_DIG*(unsigned char)*	OBS_SIDE_L OBS_CENTER_L OBS_CENTER_R OBS_SIDE_R	Detects obstacles
unsigned char SEN_OBS_ANALOG*(unsigned char)*	OBS_SIDE_L OBS_CENTER_L OBS_CENTER_R OBS_SIDE_R	Detects the distance to obstacles
unsigned char SEN_LINE_DIG*(unsigned char)*	LINE_R LINE_L	Detects dark zones (black lines)
unsigned char SEN_LINE_ANALOG *(unsigned char)*	LINE_R LINE_L	Detects surface colors
void LED_FRONT_ON(*void*)	-	Front LED on
void LED_BRAKE_ON(*void*)	-	Brake LED on
void LED_TOP_RED_ON(*void*)	-	Top red LED on
void LED_TOP_GREEN_ON(*void*)	-	Top green LED on
void LED_FRONT _OFF(*void*)	-	Front LED off
void LED_BRAKE_OFF(*void*)	-	Brake LED off
void LED_TOP_RED_OFF(*void*)	-	Top red LED off
void LED_TOP_GREEN_OFF(*void*)	-	Top green LED off
void LED_FRONT _ON_OFF(*void*)	-	Front LED blink
void LED_BRAKE_ON_OFF(*void*)	-	Brake LED blink
void LED_TOP_RED_ON_OFF(*void*)	-	Top red LED blink
void LED_TOP_GREEN_ON_OFF(*void*)	-	Top green LED blink

void SEN_CONFIG(_void_)

This function configures the inputs and outputs required to manage the sensors and initialize the variables.

Table 39. PIC-sensor connections

Pin PIC	I/O	Sensor
PORTA		
RA0	I	Light
RA1	I	Central left infrared receiver
RA2	I	Right line sensor receiver
RA3	I	Side left infrared receiver
RA5	I	Left line sensor receiver
PORTB		
RB1	I	First interruption of the accelerometer
RB2	I	Second interruption of the accelerometer
RB3	O	Speaker
RB5	O	Top red LED
RB6	O	Top green LED
PORTC		
RC7	O	Front LED
PORTD		
RD1	O	Line sensors transmitter
RD4	I	SDO signal for the SPI communication (accelerometer)
RD5	O	SDI sinal for the SPI communication(accelerometer)
RD6	O	Clock sinal for the SPI communication(accelerometer)
RD7	I	Chip Select for the SPI communication(accelerometer)
PORTE		
RE5	O	Brake LED
PORTF		
RF5	I	Side right infrared receiver
RF6	I	Central right infrared receiver
PORTH		
RH5	I	Tempreature sensor
RH6	I	Battery measurer
RH7	I	Microphone
PORTJ		
RJ6	O	Infrared transmitter
RJ7	I/O	Free pad

unsigned char SEN_LIGHT(*void*)

Output
Percentage of ambient light.

The SEN_LIGHT function captures the analog value generated by the inciding light on the photo-transistor. To achieve this follow these steps:

- Activate the ADC.
- Wait for the data acquisition process to end (100us).
- Read the analog value.
- Calculate the inciding light percentage based on the analog voltage measurement.
- Returns the percentage of ambient light.

unsigned char SEN_BATTERY(*void*)

Output
Percentage of battery level.

The SEN_BATTERY function captures the analog value of the battery [16]. To achieve this, function follows these steps:

- Activate the ADC.
- Wait for the data acquisition process to end (100us).
- Read the analog value.
- Calculate the battery level percentage based on the analog voltage measurement.
- Returns battery level.

unsigned char SEN_TEMPERATURE(*void*)

Output
Temperature in ºC.

The SEN_TEMPERATURE function captures the analog value that depends on the temperature captured by the thermistor[17]. To achieve this, function follows these steps:

- Activate the ADC.
- Wait for the data acquisition process to end (100us).
- Read the analog value.
- Calculate the temperature based on the analog voltage measurement.
- Returns temperature in %.

unsigned char SEN_MIC_ANALOG(*void*)

Output
Sound intensity.

The SEN_MIC_ANALOG function captures the analog value that depends on the sound intensity from the microphone. To achieve this, function follows these steps:

- Activate the ADC.
- Wait for the data acquisition process to end (100us).
- Read the analog value.
- Returns amplified microphone value.

[16] The output value can differ from mOwayGUI
[17] Sensor measures mOway´s temperature which can be different from ambient temperature.

***unsigned char* SEN_MIC_DIG(*void*)**

Output
Indicates if a sound has been detected

The SEN_MIC_DIG function indicates if there is sound or not. To achieve this function follows these steps:

- Returns the digital value of microphone input.

***void* SEN_SPEAKER(*unsigned char* SEN_SPEAKER_FREQ, *unsigned char* SEN_SPEAKER_TIME, *unsigned char* SEN_SPEAKER_ON_OFF)**

Input variables	
SEN_SPEAKER_FREQ	Sound frequency (see table).
SEN_SPEAKER_TIME	Time.
SEN_SPEAKER_ON_OFF	On, off or time.

The SEN_SPEAKER function emits tones in a frequency between 250 Hz and 65 KHz. SEN_SPEAKER_ON_OFF is going to say if we want to switch on, switch off or activate the speaker an amount of time (100ms intervals). To achieve this, function follows these steps:

- PWM on with frequency SEN_SPEAKER_FREQ and 50% of duty.
- If SEN_SPEAKER_ON_OFF is SPEAKER_TIME(2) function waits until command finishes.

Table 40. Allowed values for SEN_SPEAKER_ON_OFF

Define	Valor
SPEAKER_OFF	0
SPEAKER_ON	1
SPEAKER_TIME	2

Table 41. SEN_SPEAKER_FREQ vs PWM frequency

SEN_SPEAKER_FREQ	PWM frequency Hz
0	0,0000000
10	5681,8181818
20	2976,1904762
30	2016,1290323
40	1524,3902439
50	1225,4901961
60	1024,5901639
70	880,2816901

80	771,6049383
90	686,8131868
100	618,8118812
110	563,0630631
120	516,5289256
130	477,0992366
140	443,2624113
150	413,9072848
160	388,1987578
170	365,4970760
180	345,3038674
190	327,2251309
200	310,9452736
210	296,2085308
220	282,8054299
230	270,5627706
240	259,3360996
250,	249,0039841
255	244,1406250

unsigned char SEN_OBS_DIG(*unsigned char* SEN_CHECK_OBS*)*

Input variables	
SEN_CHECK_OBS	Which sensor must be read
Output	
Indicates if there is obstacle or not.	

This function indicates if the obstacle is situated on the right front side or on the left front side. To achieve this function follows these steps:

- Ensure that there is no noise source interference before sending the infrared light pulse.
- Emit the infrared light pulse to detect obstacles. This light-beam will be reflected back if there is any existing obstacle and this signal will be perceived by the infrared receiver.
- Check for any eventual signals from the four IR receivers.
- Copy the digital receiver's value to the output variables.
- Deactivate the infrared diode.
- Check for interfering signals.
- If there is no signal interferences and the process develops normally returns value.

Table 42. Allowed values for SEN_CHECK_OBS

Define	Value
OBS_CENTER_L	0
OBS_SIDE_L	1
OBS_CENTER_R	2
OBS_SIDE_R	3

unsigned char SEN_OBS_ANALOG(*unsigned char* SEN_CHECK_OBS)

Input variable	
SEN_CHECK_OBS	Which sensor must be read
Output	
Indicates if there is obstacle or not.	

This function indicates if the obstacle is on the right front side or on the left front side and its distance from the robot. To achieve this follow the steps indicated below:

- Ensure that there is no noise source interferences before you send the infrared light pulse.
- Emit the infrared light pulse to detect obstacles.
- Activate the ADC.
- Check for any possible signals from the four IR receivers.
- Copy the analog receiver's value to the output variables. The higher the value the shorter the distance will be.
- Deactivate the infrared diode.
- Check for interfering signals. If there is no signal interferences and the process develops normally value is returned.

Table 43. SEN_CHECK_OBS allowed values

Define	Value
OBS_CENTER_L	0
OBS_SIDE_L	1
OBS_CENTER_R	2
OBS_SIDE_R	3

unsigned char SEN_ACCE_XYZ_READ(*unsigned char* SEN_CHECK_ACCE)

Input variable	
SEN_CHECK_ACCE	Which axis must be read
Output	
Acceleration value	

SEN_ACCE_XYZ_READ returns the acceleration of the robot in the 3 axes. Resolution is ±0.0156G/bit. Value 0 is -2G and 255 is 2G.

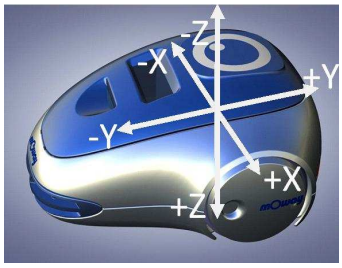


Image 56. mOway axes

- Communication between microcontroller and accelerometer is SPI.
- Command is sent to change the mode of the accelerometer to “measure”.
- Function waits until the value is calculated.
- Value is read.
- Change the mode to “tap detection”.

Table 44. SEN_CHECK_ACCE allowed values.

Define	Value
ACCE_CHECK_X	0
ACCE_CHECK_Y	1
ACCE_CHECK_Z	2

unsigned char SEN_ACCE_CHECK_TAP(*void*)

Output
1: Tap 2: Tap tap

Accelerometer detects taps.

- Communication between microcontroller and accelerometer is SPI
- Checks if “tap interrupt” has been detected
- SEN_ACCE_TAP value is changed.

unsigned char SEN_LINE_DIG(*unsigned char* SEN_CHECK_LINE)

Input variable	
SEN_CHECK_LINE	Which sensor must be read
Output	
Digital value of the sensor	

The SEN_LINE_DIG function indicates whether the sensors are or are not on a dark surface. To achieve this function follows the steps indicated below:

- Emit the infrared light pulse to detect the line. This light-beam will be reflected back if the line is detected and this signal will be perceived by the infrared receiver.
- Wait for the data acquisition process to end (900 us).
- Read the sensor.
- Copy the value to the SEN_LINE variable. If the surface is dark (no light is reflected) the variable will return a‘1’ value.

Table 45. SEN_CHECK_LINE allowed values

Define	Value
LINE_L	0
LINE_R	1

unsigned char SEN_LINE_ANALOG(*unsigned char* SEN_CHECK_LINE*)*

Input variables	
SEN_CHECK_LINE	Which sensor must be read
Output	
Analog value of the sensor	

The SEN_LINE_ANALOG function indicates the light reflected in the optocouplers [18]. To do this function follows the steps indicated below:

- Emit the infrared light pulse to detect the line. This light-beam will be reflected back if the line is detected and this signal will be perceived by the infrared receiver.
- Wait for the data acquisition process to end (900us).
- Read the sensor.
- Copy this value to the SEN_LINE variable. The higher the values the darker will the surfaces be.

Table 46. SEN_CHECK_LINE allowed values

Define	Value
LINE_L	0
LINE_R	1

void LED_BRAKE_ON(*void*)

Function to switches on the brake LED.

void LED_FRONT_ON(*void*)

Function to switches on the front LED.

[18] Due to tolerance two different sensors can differ from each other.

***void* LED_TOP_RED_ON(*void*)**

Function to switches on red LED.

***void* LED_TOP_GREEN_ON(*void*)**

Function to switches on green LED.

***void* LED_BRAKE_OFF(*void*)**

Function to switches off the brake LED.

***void* LED_FRONT_OFF(*void*)**

Function to switches off the front LED.

***void* LED_TOP_RED_OFF(*void*)**

Function to switches off the red LED.

***void* LED_TOP_GREEN_OFF(*void*)**

Function to switches off the green LED.

***void* LED_BRAKE_ON_OFF(*void*)**

Blink the brake LED.

***void* LED_FRONT_ON_OFF(*void*)**

Blink the front LED.

***void* LED_TOP_RED_ON_OFF(*void*)**

Blink the red LED.

***void* LED_TOP_GREEN_ON_OFF(*void*)**

Blink the green LED.

6.3.2. mOway´s motor library C18

6.3.2.1. Description

The library includes a series of functions in charge of sending I2C commands to the Drive System, which will be responsible for controlling the motors and therefore releasing the main microcontroller so it can carry out other tasks.

Communications with the motor module are conducted via the I2C protocol. Any microcontroller with this kind of communications can control the motors; use the libraries in assembly. The format for the Driving System I2C frame can be observed in the following illustrations. Each of these frames lasts approximately 350 us.

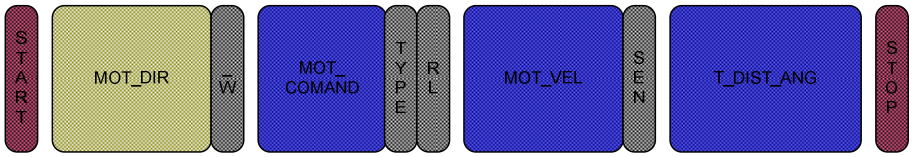


Image 57. Command format: MOT_STR, MOT_CHA_VEL

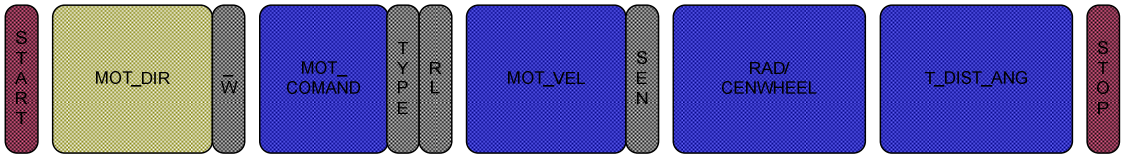


Image 58. Command format: MOT_CUR, MOT_ROT

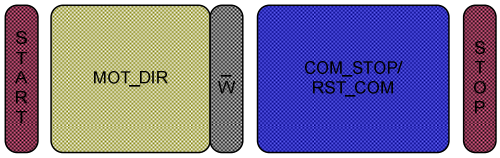


Image 59. Command format: MOT_STOP, MOT_RST

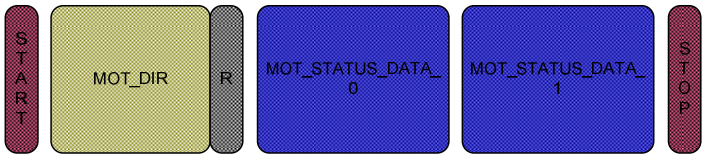


Image 60. Command format: MOT_FDBCK

6.3.2.2. Functions

A series of functions designed to control mOway's drive system are included in the lib_mot_moway library.

Table 47. Summary of functions in C for lib_mot_moway

Name	Input	Return	Description
***void* MOT_CONFIG(*void*)**	-	-	Configuration to communicate with the motors
***unsigned char* MOT_STR(** *unsigned char, unsigned char, unsigned char, unsigned char)*	MOT_VEL FWDBACK COMTYPE MOT_T_DIST	0: Correct dispatch 1: Incorrect dispatch 2: Incorrect data	A command to move in a straight line
***unsigned char* MOT_CHA_VEL(** *unsigned char, unsigned char, unsigned char, unsigned char, unsigned char)*	MOT_VEL FWDBACK RL COMTYPE MOT_T_DIST	0: Correct dispatch 1: Incorrect dispatch 2: Incorrect data	A command to change the speed of a motor
***unsigned char* MOT_ROT(** *unsigned char, unsigned char, unsigned char, unsigned char, unsigned char, unsigned char)*	MOT_VEL FWDBACK MOT_CENWHEEL RL COMTYPE MOT_T_ANG	0: Correct dispatch 1: Incorrect dispatch 2: Incorrect data	A command to rotate the robot
***unsigned char* MOT_CUR(** *unsigned char, unsigned char, unsigned char, unsigned char, unsigned char, unsigned char)*	MOT_VEL FWDBACK MOT_RAD RL COMTYPE MOT_T_DIST	0: Correct dispatch 1: Incorrect dispatch 2: Incorrect data	A command to execute a curve
***unsigned char* MOT_STOP(*void*)**		0: Correct dispatch 1: Incorrect dispatch	A command to stop the robot
***unsigned char* MOT_RST(** *unsigned char*)	RST_COM	0: Correct dispatch 1: Incorrect dispatch	A command to reset the temporary variables for time and distance
unsigned char* **MOT_FDBCK(*void*)**		0: Correct dispatch 1: Incorrect dispatch	A command to determine the motor's status

void MOT_CONFIG(*void*)

This function configures the inputs and outputs so the microcontroller can communicate with the Drive System.

Table 48. Pic-drive system connections

Pin PIC	I/O	Sensor
PORTE		
RE7	I	Indicates when the motor ends the command.
RE0	O	SCL of the I2C protocol
RE1	O	SDA of the I2C protocol

Port RE7 indicates the end of a command. This port is labeled as MOT_END in the library.

Example:

```
//Straight forward at 100% speed for 10 seconds (100ms x 100)
```

```
MOT_STR(100, FWD, TIME, 100);
//No action is taken until the command ends
while(!MOT_END){}
```

***unsigned char* MOT_STR(*unsigned char* MOT_VEL,*unsigned char* FWDBACK,*unsigned char* COMTYPE, *unsigned char* MOT_T_DIST)**

Input			
MOT_VEL	Desired speed	0	100
FWDBACK	Movement direction	FWD	BACK
COMTYPE	Type of command	TIME	DISTANCE
MOT_T_DIST	Time value Distance value	0 0	255 255
Function Return			
0: Correct dispatch	The command has been sent correctly		
1: Incorrect dispatch	The command has not been sent. Connection problem		
2: Incorrect data	The data is incorrect		

Command to move in a straight line. It is necessary to specify speed, direction, type of command and the time or the distance to cover. The time has a resolution of 100ms and the distance of 1 mm, and with a value of 0 returned by MOT_T_DIST the command will be maintained until another order is given.

Example:

```
//Straight forward at 100% speed during 10 seconds (100 ms x 100)
MOT_STR(100, FWD, TIME, 100);

//Straight backwards at 15% speed 100mm (1mm x 100)
MOT_STR(15, BACK, DISTANCE, 100);
```

***unsigned char* MOT_CHA_VEL(*unsigned char* MOT_VEL,*unsigned char* FWDBACK,*unsigned char* RL,*unsigned char* COMTYPE,*unsigned char* MOT_T_DIST)**

Input			
MOT_VEL	Desired speed	0	100
FWDBACK	Movement direction	FWD	BACK
RL	Right or left	RIGHT	LEFT
COMTYPE	Type of command	TIME	DISTANCE
MOT_T_DIST	Time value Distance value	0 0	255 255
Function Return			
0: Correct dispatch	The command has been sent correctly		
1: Incorrect dispatch	The command has not been sent. Connection problem		
2: Incorrect data	The data is incorrect		

A command to change the speed of any of the two motors. It is necessary to specify speed, direction, motor, type of command and the time or distance to be traveled. The time has a resolution of 100 ms and the distance 1mm, and with a value of 0 returned by MOT_T_DIST the command shall be maintained until another order is specified.

Example:

```
//Change speed (80% forward) of the right motor during 10 seconds
//(100 ms x 100)
MOT_CHA_VEL(80, FWD, RIGHT, TIME, 100) ;

//Change speed (20% backwards) of the left motor and travels a distance of 100
//mm (1 mm x 100)
MOT_CHA_VEL(20, BACK, LEFT, DISTANCE, 100) ;
```

***unsigned char* MOT_ROT(*unsigned char* MOT_VEL,*unsigned char* FWDBACK,*unsigned char* MOT_CENWHEEL,*unsigned char* RL,*unsigned char* COMTYPE,*unsigned char* MOT_T_ANG)**

Input			
MOT_VEL	Desired speed	0	100
FWDBACK	Movement direction	FWD	BACK
MOT_CENWHEEL	On center or wheel	CENTER	WHEEL
RL	Right or left	RIGHT	LEFT
COMTYPE	Type of command	TIME	DISTANCE
MOT_T_ANG	Time value Angle value	0 0	255 100
Function Return			
0: Correct dispatch	The command has been sent correctly		
1: Incorrect dispatch	The command has not been sent. Connection problem		
2: Incorrect data	The data is incorrect		

Command to make the Moway rotate. It is necessary to specify speed, direction, type of rotation, motor, type of command and the time or the angle to rotate. The time has a resolution of 100 ms, and with a value of 0 returned by MOT_T_ANG the command shall be maintained until another order is specified.

For the angle, the equations below illustrate how to calculate the value of MOT_T_ANG taking into account the desired rotation angle. If the rotation is produced on one of the wheels more resolution is obtained. On the other hand, mechanical inertia must also be considered; therefore to achieve greater precision it is advisable to reduce the speed.

Equation 4. MOT_T_ANG when rotating on its center

$$MOT\_T\_ANG = round\left(\frac{Angle^{o} \times 3.33}{12^{o}}\right)$$

Example:

```
//Rotation in relation to the center to the right at 80% speed for 10 seconds
//(100ms x 100)
MOT_ROT(80, FWD, CENTER, RIGHT, TIME, 100) ;

//Rotation in relation to the left wheel forward at 20% speed 180º
MOT_ROT(20, BACK, WHEEL, LEFT, ANGLE, 50) ;
```

***unsigned char* MOT_CUR(*unsigned char* MOT_VEL,*unsigned char* FWDBACK,*unsigned char* MOT_RAD,*unsigned char* RL,*unsigned char* COMTYPE, *unsigned char* MOT_T_DIST)**

Input			
MOT_VEL	Desired speed	0	100
FWDBACK	Movement direction	FWD	BACK
MOT_RAD	Radius	0	100
RL	Right or left	RIGHT	LEFT
COMTYPE	Type of command	TIME	DISTANCE
MOT_T_DIST	Time value Distance value	0 0	255 255
Function Return			
0: Correct dispatch	The command has been sent correctly		
1: Incorrect dispatch	The command has not been sent. Connection problem		
2: Incorrect data	The data is incorrect		

Command to describe a curve. It is necessary to specify speed, direction, radius, course, type of command and the time or the distance to be traveled. The radius is the speed that will be subtracted or added to the robot's global speed. This means that if the specified speed is 50 and the radius 10, one of the motors shall work at 60% speed and the other one 40%. Therefore the radius has to adhere to the following restrictions:

Equation 5. Condition 1 MOT_RAD

$$0 \le MOT\_VEL - MOT\_RAD \le 100$$

Equation 6. Condition 2 MOT_RAD

$$0 \le MOT\_VEL + MOT\_RAD \le 100$$

The time has a resolution of 100 ms and the distance 1 mm, and with a value of 0 returned by MOT_T_ANG the command shall be maintained until another order is specified. The motor measures the distance traveled by the motor located on the external side of the curve.

Example:

```
//Curve to the right at 50% with a radius of 10 for 10 seconds
//(100ms x 100)
//VEL_I=60
//VEL_D=40
MOT_CUR(50, FWD, 10, RIGHT, TIME, 100) ;

//Curve to the left at 80% with a radius 15 for 100mm
//(1mm x 100)
//VEL_I=95
//VEL_D=65
MOT_CUR(80, BACK, 15, LEFT, DISTANCE, 100) ;
```

unsigned char MOT_STOP(*void*)

Function Return	
0: Correct dispatch	The command has been sent correctly
1: Incorrect dispatch	The command has not been sent. Connection problem

Command to stop the robot.

Example:

```
// Moway stop
MOT_STOP() ;
```

unsigned char MOT_RST(*unsigned char* RST_COM)

Input		
RST_COM	The parameter that to be reset	RST_T RST_DIST RST_KM
Function Return		
0: Correct dispatch	The command has been sent correctly	
1: Incorrect dispatch	The command has not been sent. Connection problem	

Resets the motor's internal time, distance and speedometer temporary variables.

Example:

```
//Reset elapsed time
MOT_RST(RST_T);

//Reset distance traveled
MOT_RST(RST_D);
```

*unsigned char** MOT_FDBCK(*unsigned char* STATUS_COM)

Input		
STATUS_COM	The parameter to be recalled	STATUS_T STATUS_A STATUS_V_R STATUS_V_L STATUS_D_R STATUS_D_L STATUS_KM
Output		
Pointer to two char.		

A command to recall different drive system parameters: elapsed time, the angle (only through the MOT_ROT command), the speed of each motor, distance traveled by each motor and the speedometer.

This function returns a pointer to 2 chars. All the petitions except STATUS_KM return one byte MOT_FDBCK(STATUS_x)[0]) maintaining MOT_FDBCK(STATUS_x)[1] at a 0xFF value. These two variables are updated every time a new command is sent (e.g. recall the time elapsed since the last command). Whenever using STATUS_KM the two bytes must be considered. This command is very useful to calculate the length of a line while the robot follows it.

Table 49. Parameter resolution

Parameters	Resolution
STATUS_T	100ms/bit
STATUS_A	3.6º/bit
STATUS_V_R	1%/bit
STATUS_V_L	1%/bit
STATUS_D_R	1mm/bit
STATUS_D_L	1mm/bit
STATUS_KM	1mm/bit

Example:

```
// Recall elapsed time since the last command
char command_time;

command_time =MOT_FDBCK(STATUS_T)[0];
//E.g. Output:
  //MOT_FDBCK(STATUS_T)[0]=0x7F    => 12.7 seconds elapsed since the last
  //command
// MOT_FDBCK(STATUS_T)[1]=0xFF;  => Invalid data
```

```
//Request of distance traveled by the right motor from the last command
char mOway_km[2];
mOway_km[0]= MOT_FDBCK(STATUS_KM);
mOway_km[1]= MOT_FDBCK(STATUS_KM);

//e.g. Output:
// mOway_km[0]=0x08
// mOway_km[1]=0x01;
```

byte 1	byte 0
0x01	0x08
0000 0001	0000 0100
264	
Distance: 264*1mm	
264mm	

6.3.3. BZI-RF2GH4 library in C18

6.3.3.1. Description

With this library it is easy for mOway to communicate with the BZI-RF2GH4.

6.3.3.2. Functions

To manage the sending of parameters and the return of values, external values are used. These must be modified beforehand or verified after each call. What these are and how they act will be explained in each function.

Table 50. Summary of functions in C18.

Name	Input	Return	Description
void RF_CONFIG_SPI(void)	-	-	Configures the inputs and outputs of the microcontroller as well as the parameters necessary to use the SPI bus.
void RF_INT_EN(void)	-	-	This routine enables the external interruption for the radio module in the microcontroller.
unsigned char RF_CONFIG(unsigned char , unsigned char)	CHANNEL ADDRESS	1: Correct configuration 0: Not configured	Configures the inputs and outputs of the microcontroller as well as the radio module parameters.
unsigned char RF_ON(void)	-	0: Correct activation 1: Incorrect activation	Activates the radio frequency module in watch mode.
unsigned char RF_OFF(void)	-	0: Correct deactivation 1: Incorrect deactivation	Deactivates the radio frequency module and leaves it in low consumption mode.

unsigned char RF_SEND(unsigned char, unsigned char)	RF_DIR_OUT RF_DATA_OUT[]	0:Sent correctly 1:No ACK 2: Not sent	Sends a data frame (8 Bytes) to the address indicated.
unsigned char RF_RECEIVE(unsigned char*, unsigned char*)	RF_DIR_IN RF_DATA_IN[]	0: Single reception 1: Multiple reception 2: No reception.	Checks whether a reception has occurred and if so, collects the frame.

void RF_CONFIG_SPI(void)

The speed of the SPI must not exceed 8 Mhz. In the function, the different parameters of the SPI module and the PIC pins are configured.

Table 51. SPI configuration PIC ports

PIN RF	PIN PIC
SCK	RC3
SDI	RC5
SDO	RC4

Example:

```
//Configure SPI modules of the PIC
  RF_CONFIG_SPI();
```

unsigned char RF_CONFIG(unsigned char CHANNEL unsigned char ADDRESS)

Input variables	
RF_DIR	Device address. Must be a value of between 0x01 and 0xFE.
RF_CHN	Channel to be used in the communication. Must be a value of between 0x00 and 0x7F (128 channels).
Function Return	
1: Correct configuration	The module has been configured correctly.
0: Incorrect configuration	The module is not configured. Communications with the module impossible due to the absence of or incorrect electrical connection.

This function configures the transceptor, enabling its own watch address and the 'broadcast' address. In turn, it configures other parameters such as PIC pins, transmission speed, emission power, address length, the length of the CRC code, etc.

Table 52. RF module PIC ports configuration

RF PIN	PIC PIN
IRQ	RB0
CSN	RF2
CE	RH4

The channel must be common to all the modules that are going to take part in the communication. Users can choose any channel from among the 128 available.

Nevertheless, if there is more than one communication in the environment between modules in different channels, a spacing of 2 must be left between the channels to be used in order to avoid interferences, thus leaving 32 channels usable. Another question to be taken into account is the existence of other technologies that use the ISM 2.4GHz band (Wifi, Bluetooth,etc.) and that might also cause interference in one of the channels.

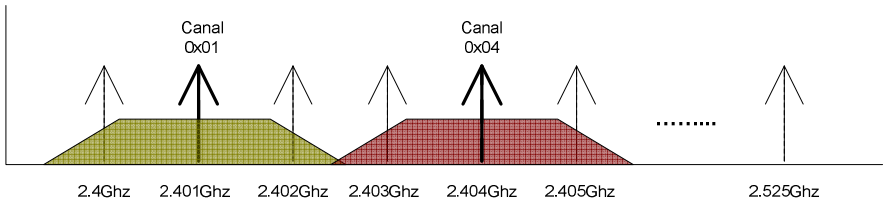


Image 61. RF channels

Before calling up this function, the SPI module must be configured.

Example:

```
//Configure RF module (channel and address)
 if(RF_CONFIG(0x40,0x01)==0){
 //Module not configured
 }
```

unsigned char RF_ON(void)

Function Return	
1: Correct activation	The module has been turned on correctly.
0: Incorrect activation	The module is not active. Communications with the module impossible or, on the other hand, absence of or incorrect electrical connection.

This routine activates the radio module in watch mode in order to be able to receive data and/or send data.

It is important to take into consideration that following the call to this routine, the module requires 2.5 ms to be ready.

Example:

```
//-----------[Configuration and activation routine without interruption]---------
//Configure SPI modules of the PIC
RF_CONFIG_SPI();

//Configure RF module (channel and address)
if(RF_CONFIG(0x40,0x01)==0){
//Module not configured
}
```

```
  //Activate the RF module
  if(RF_ON()==0){
  //Module not initialised
   }
//-------------------------------------------------------------------------------
```

unsigned char RF_OFF(void)

Function Return	
1: Correct deactivation	The module has been deactivated correctly.
0: Incorrect deactivation	The module has not been deactivated correctly. Communications with the module impossible due to the absence of or incorrect electrical connection.

This routine deactivates the radio module leaving this in low consumption mode. It does not clear the established configuration.

unsigned char RF_SEND(*unsigned char RF_DIR_OUT, unsigned char RF_DATA_OUT[]*)

Input variables	
RF_DATA_OUT	This is an 8 bytes variable. (RF_DATA_OUT[0 - 7]).
RF_DIR_OUT	Ouput address
Function Return	
0: Sent correctly (ACK OK)	The data has been sent and the ACK has been received from the receiver.
1: Incorrect reception of ACK (NO ACK)	The information has been sent but the ACK has not been received (incorrectly configured receiver, different channel in receiver, incorrect address).
2: Not sent	The information has not been sent.

This function sends 8 bytes of data to the indicated address and reports the correct reception to the recipient. Following this, the device will return to the watch mode.

If a message is sent to the address 0x00, this will be received by all the modules on the same channel. It must be taken into account that the module accepts the first ACK it receives; therefore we cannot be certain that the data has arrived at all the devices.

Example:

```
static char data_out[8];
static char dir_out;
//------------------------[Data sending routine]--------------------------------
```

```
 ret=RF_SEND(dir_out,data_out);
if(ret==0){
        //Data sent and ACK received
}
else if(ret==1){
        //Data sent and ACK not received
}
else{
        //Data not sent
}
//-------------------------------------------------------------------------------
```

unsigned char RF_RECEIVE(*unsigned char* RF_DIR_IN, *unsigned char* RF_DATA_IN)*

Output variables	
RF_DATA_IN*	This is an 8 bytes variable. It presents the information received (RF_DATA_IN[0 - 7]).
RF_DIR_IN*	This is a byte variable. It indicates the transmitter address.
Function Return	
0	Single reception. There is no more data in the reception stack.
1	Multiple receptions. There is more data in the reception stack. This occurs when the transmitter sends more than one frame before the receiver collects this.
2	No data have been received.

This routine is responsible for checking whether a reception has taken place and if so, it returns the data received. Likewise, it reports whether there is data that has not been read in the reception FIFO of the module.

When a frame is received the function output must be checked. If the function returns a 1, the RF_RECEIVE() function must be called up again, but before doing so, it is necessary to process the data or this will be lost. The transceptor has a 3-level stack, and therefore if the RF_RECEIVE() function is not called up before the stack is filled, the device will not be able to receive more data.

Example:

```
  char data_in[8];
  char data_in_dir;
//-------------[Reception routine with interruption]--------------------------
#pragma interrupt YourHighPriorityISRCode
void YourHighPriorityISRCode()       {
  RF_RECEIVE(&data_in_dir,&data_in[0]);
} //-------------------------------------------------------------------------------
```

```
//--------------[Reception routine without interruption]---------------------------
while(1){

     while(RF_RECEIVE(&data_in_dir,&data_in[0])!=2){
  // Replace with code required for processing data
      }
}
//-------------------------------------------------------------------------------
```

void RF_INT_EN(*void)*

This routine is responsible for enabling the external interruption of the microcontroller that uses the RF module in data reception. For this reason, the RB0 pin is configured as input. Although the module can be managed without interruptions, the minimum response time is not guaranteed.

Example:

```
//-----------[Configuration and activation routine with interruption]---------
  //Enable interruptions
  RF_INT_EN();

  //Configure SPI modules of the PIC
  RF_CONFIG_SPI();

  //Configure RF module (channel and address)
  if(RF_CONFIG(0x40,0x01)==0){
  //Module not configured
  }

  //Activate the RF module
  if(RF_ON()==0){
  //Module not initialised
  }
//-------------------------------------------------------------------------------
```

6.3.3.3. Flow diagram for sending and receiving data

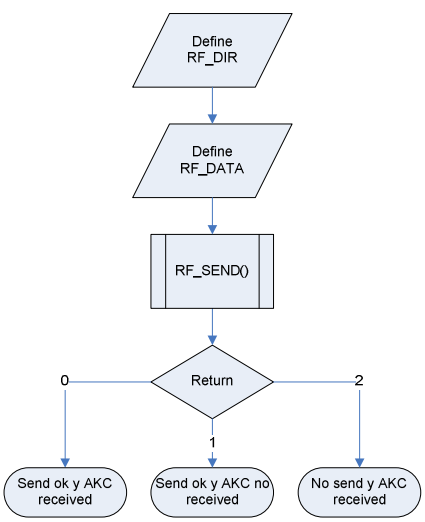


Diagram 4. Data send in C

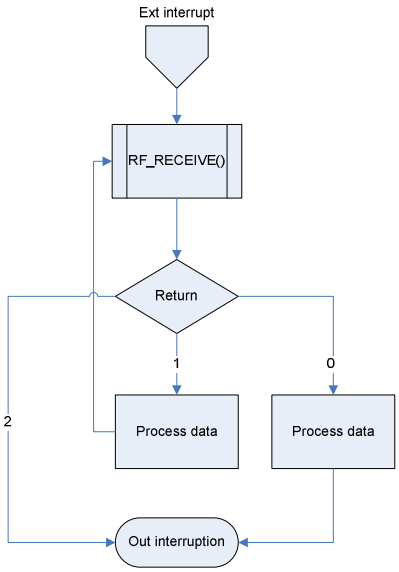


Diagram 5. Reception interruption in C

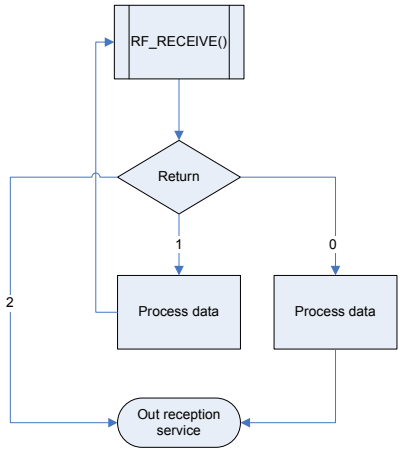


Diagram 6. Reception in C

7. mOwayGUI programming

7.1. Creating a Project

The only thing you have to do is to click on the new project icon.

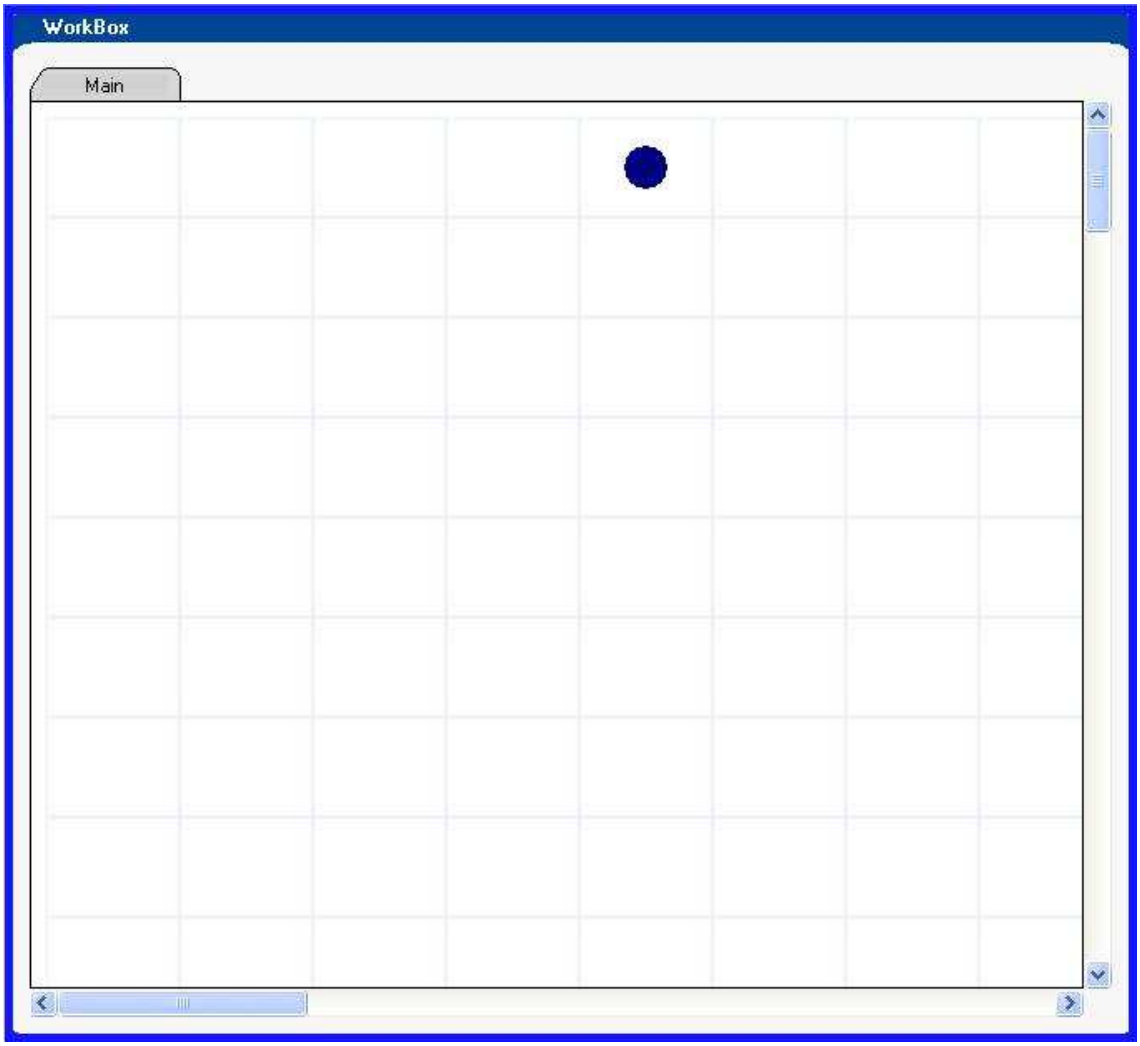


Image 62. mOwayGUI screen

7.2. First programme in mOwayGUI

In order to develop your first programme, first you must create a project (previous chapter). This first basic programme will make mOway avoid obstacles.

1. Next step is to add a 1-second delay: delay_ms(1000). Just add a module and double click in order to configure it. The *Pause* option is selected and configured as may be seen in the following image.

2. The command to make one of the LEDs blink is added. This is another module with the following configuration:

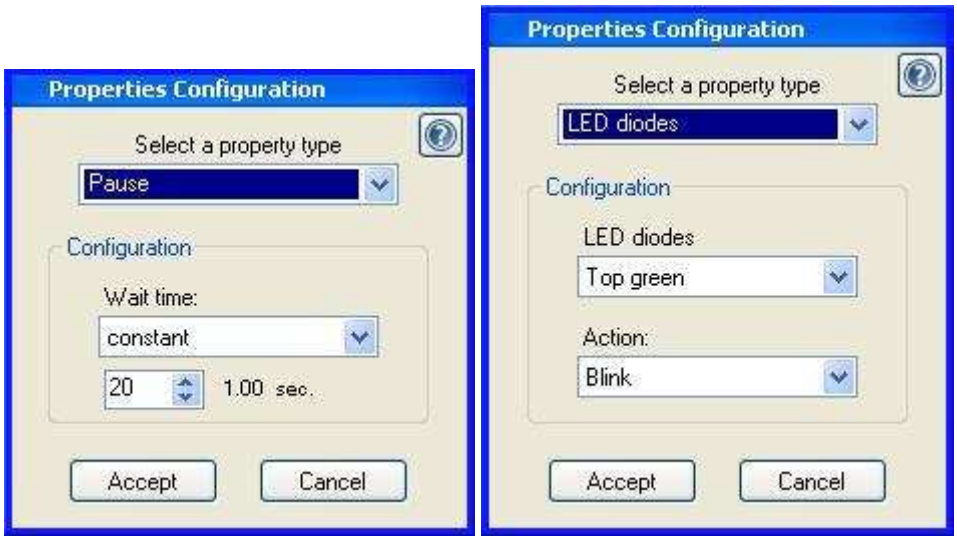


Image 63. Pause and LEDs configuration

3. The end of the program is added so that the application can be compiled.

4. The program is compiled and recorded into the robot by means of the record button in the menu.

5. Test the program and check that after waiting 1 second the green LED lights up.

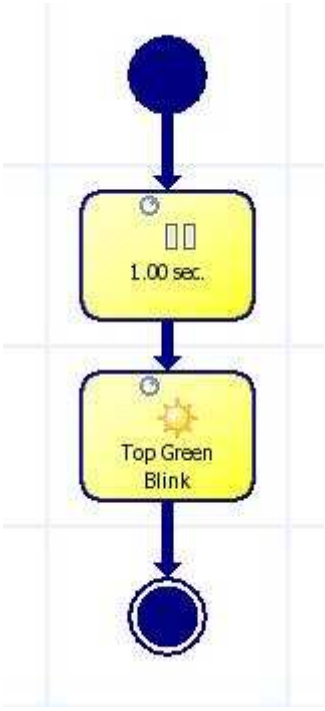


Image 64. First mOwayGUI program: Pause and blinking

6. In order to detect obstacles, the *Condition* modules are configured to check both sensors individually.

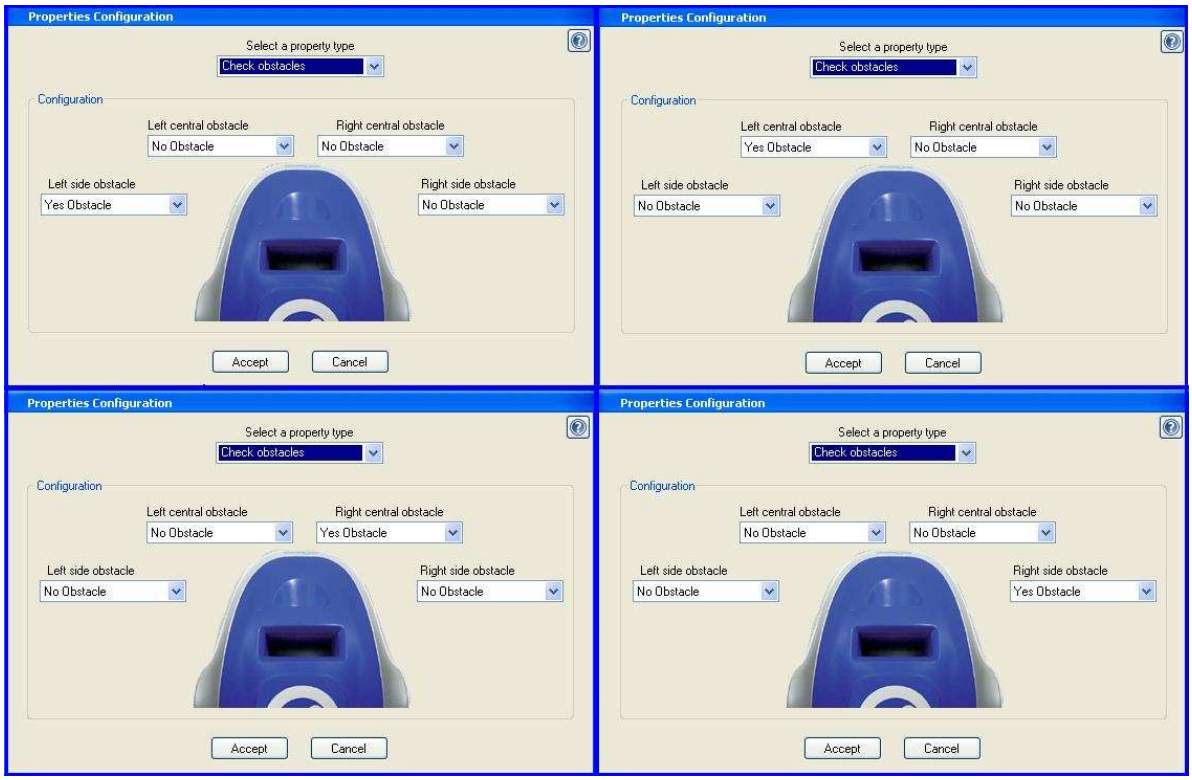


Image 65. Check obstacles configuration

7. Condition modules have a true output and false output. If the condition is true (obstacle detected) the corresponding LED lights up, otherwise it remains off.

8. Test the program and check that the front LEDs light up when an obstacle is detected.

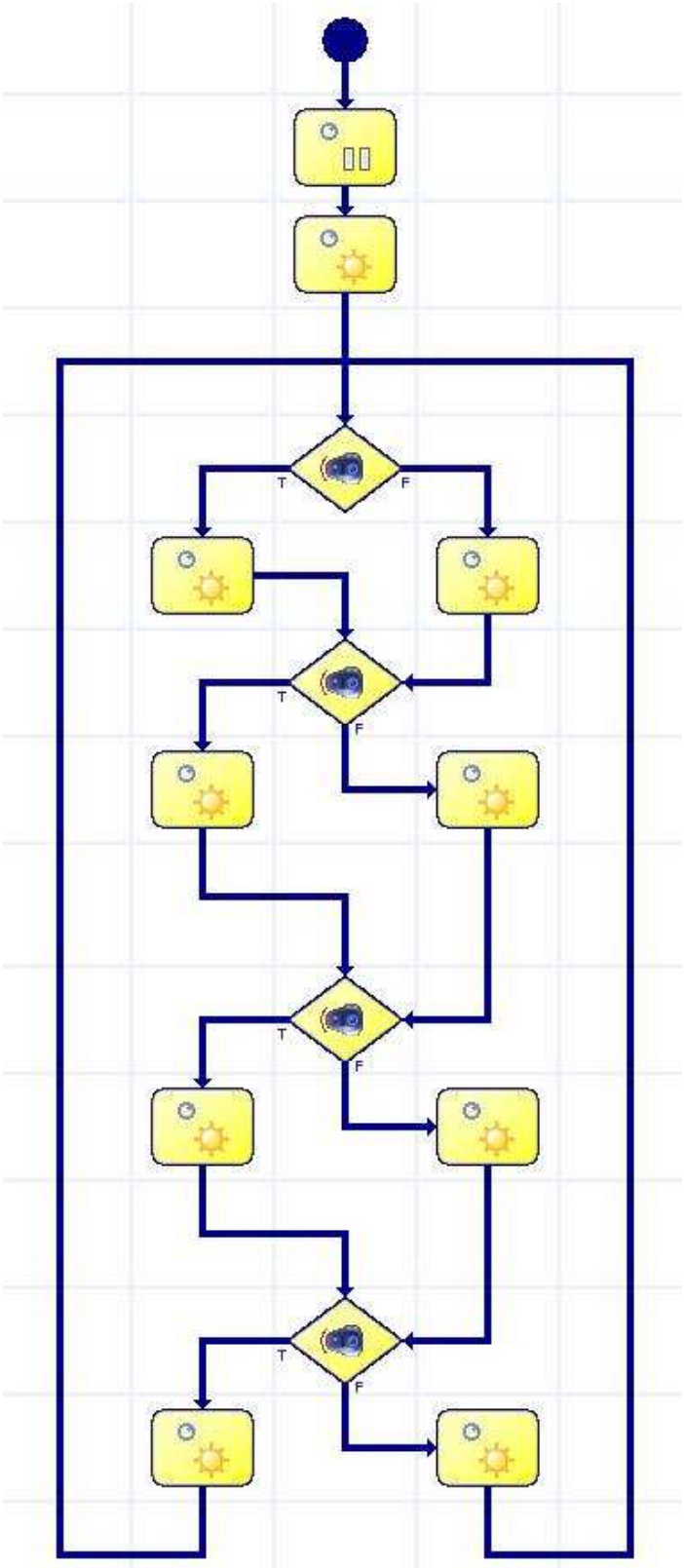


Image 66. First mOwayGUI program: obstacle detection

9. We add movement to the robot: straight on indefinitely until an obstacle is found.

10. When an obstacle is found, a command is sent to the robot to rotate 180º. The robot will continue to move in a straight line when the rotation is completed.

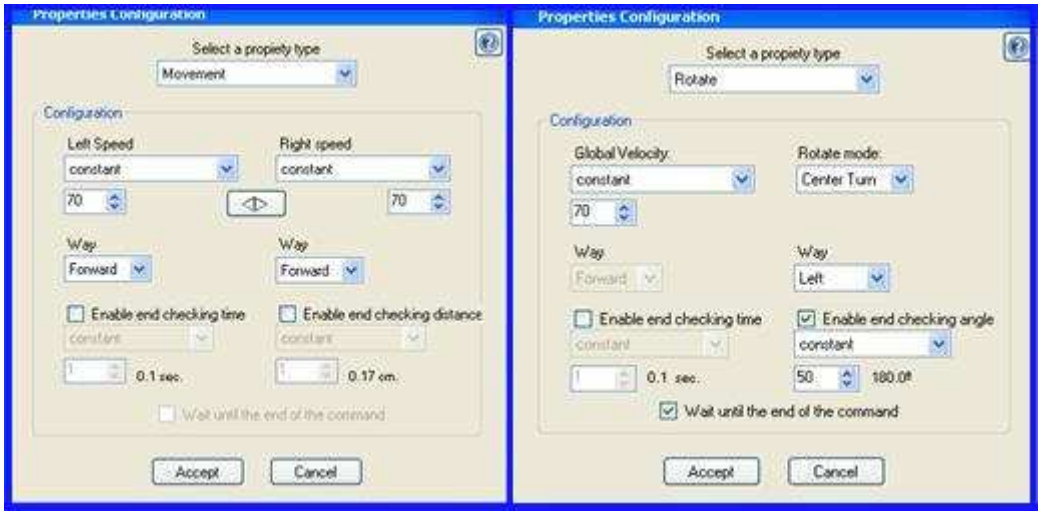


Image 67. Movement and Rotation configuration

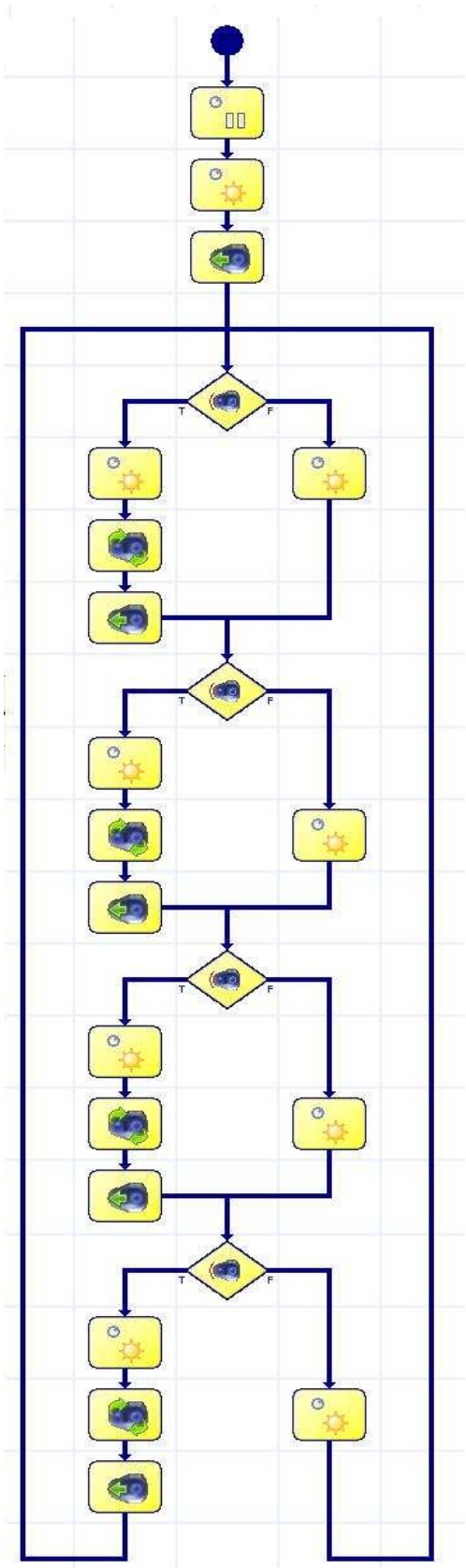


Image 68. End of first program in mOwayGUI

7.3. mOwayGUI

7.3.1. Modules

Modules are actions in which the output is unconditional: turn on a LED, send a movement command, etc. With modules, the following actions can be carried out:

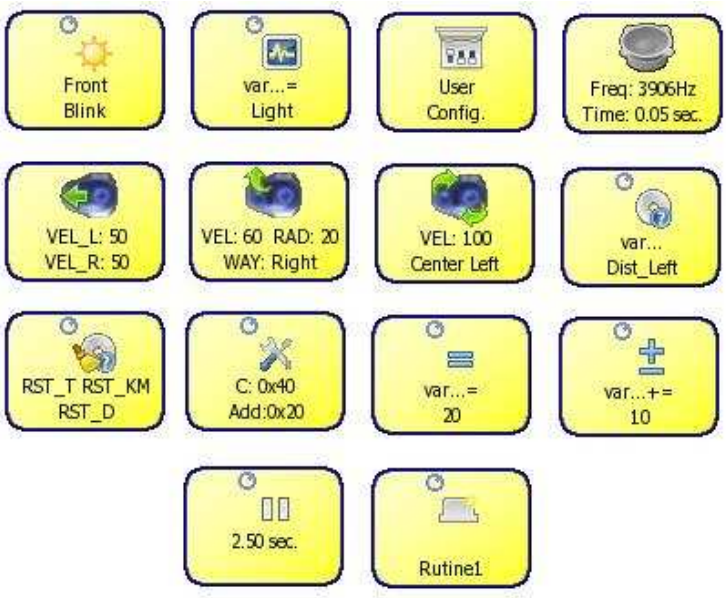


Image 69. First program: detecting obstacles moving

SENSORS

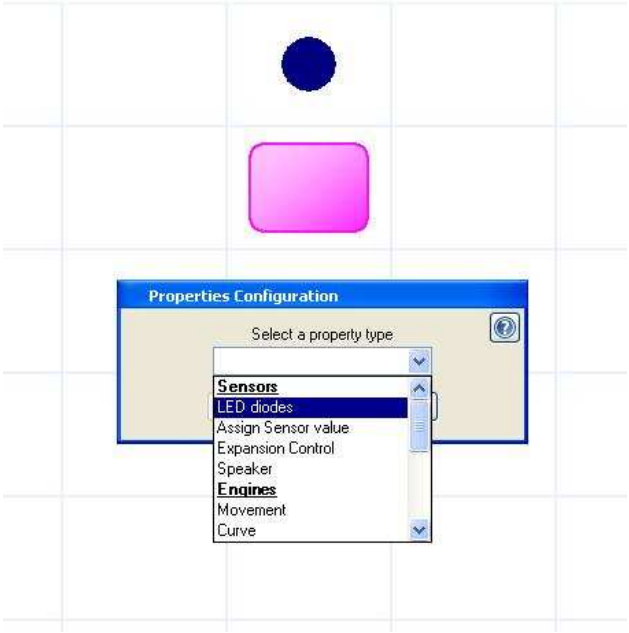


Image 70. Choose the type of module (Sensors)

- **LED Diodes**

This module enables you to operate on mOway's LED diodes. You can turn them on, turn them off or make them blink.

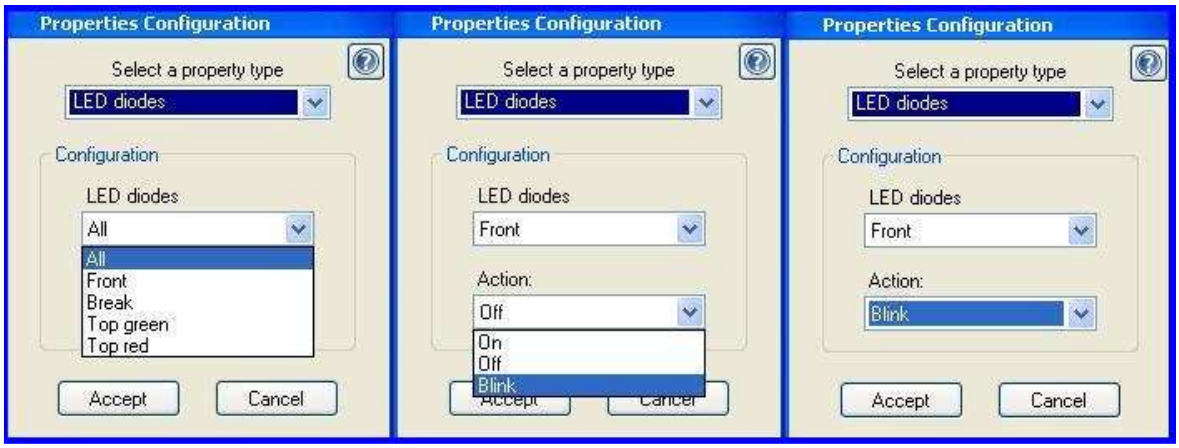


Image 71. Choose the LED diode and its action

- **Assign sensor value**

This function is to assign a variable to one sensor. This variable can be used to configure different aspects of the robot.

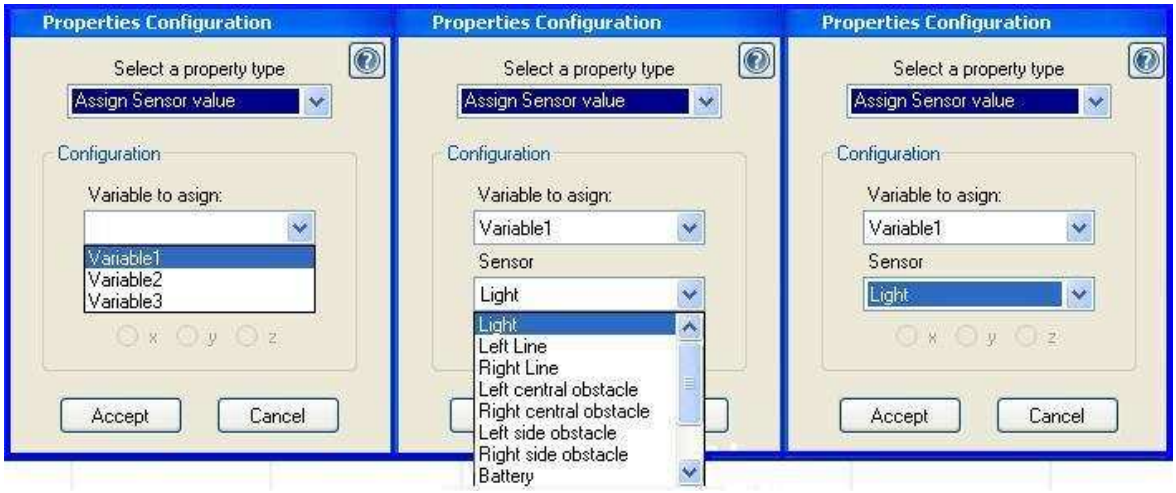


Image 72. Choose the variable to assign and the sensor

- **Expansion control**

Control of the expansion connector of the mOway robot. It can be used with the mOway expansion module.

Image 73. Choose pinout configuration of the expansion connector

WARNING!

Only advanced users can use the pinout configuration. Any incorrect connection of electronic elements to the expansion connector may damage the robot irreversibly.

- **Speaker**

This function enables mOway to emit tones from 250 Hz to 65 KHz in pair sequences of 100 ms. This is possible because of the speaker installed on the robot.

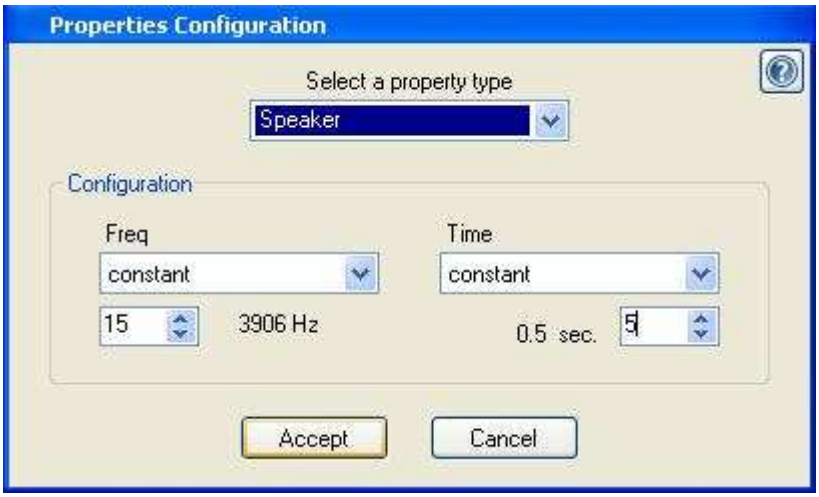


Image 74. Choose the frequency and the time of the tone

ENGINES

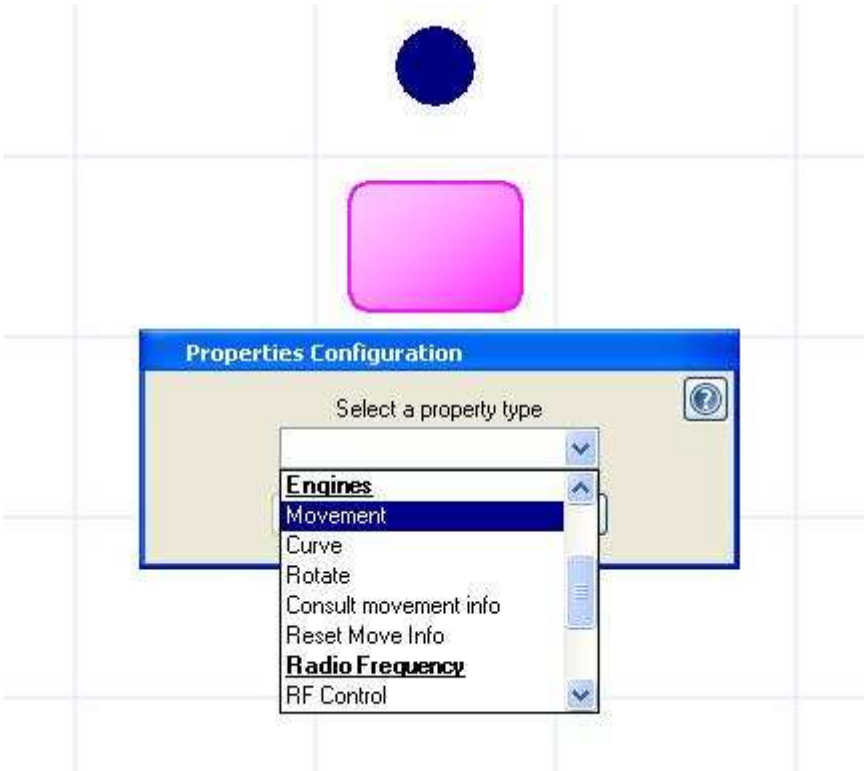


Image 75. Choose the type of module (Engines)

- **Movement**

mOway has two motors in each wheel. These give it a great flexibility in its movements. The movement commands allow the speeds of each motor to be controlled individually and to restrict movements in certain time or distance.

Another important option when configuring motors is the "**Wait until end of command**". This option allows you to block the movement command (the program stops running) until the movement module finishes (according to time or distance). This option is only accessible if we select the options "Enable end according to time" or "Enable end according to distance".

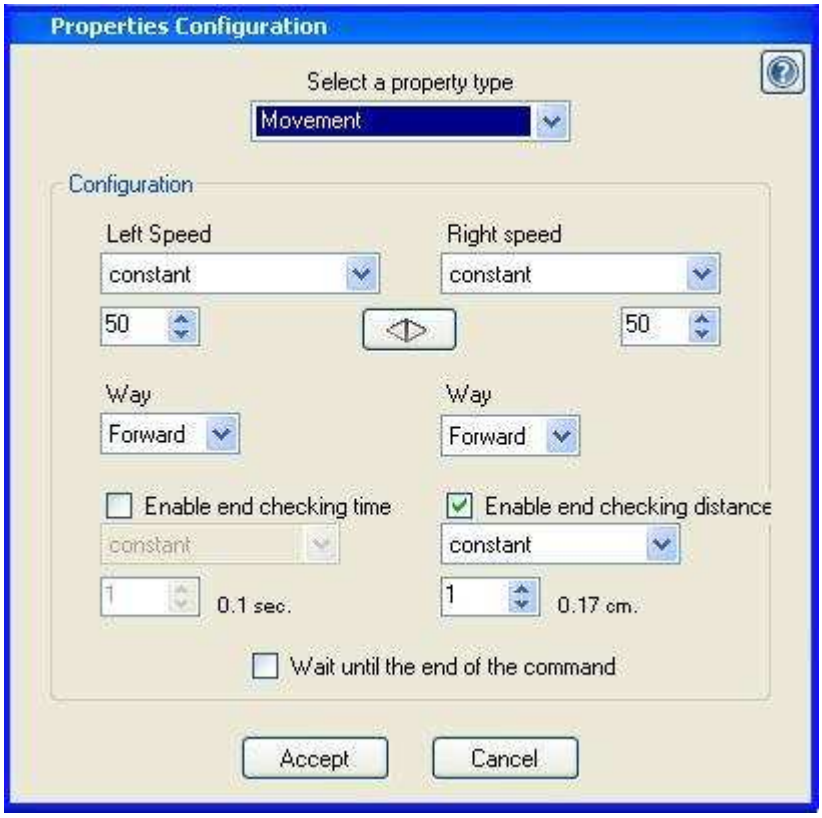


Image 76. Choose the speed, the way and the time or the distance

- **Curve**

The curve module is a specific case of the movement module. In this command, drive system will calculate the speed of the motors in order to be able to trace a curve, indicating the speed and turning radius.

We can also use here the time restrictions and distance and the "Wait until end of command" option.

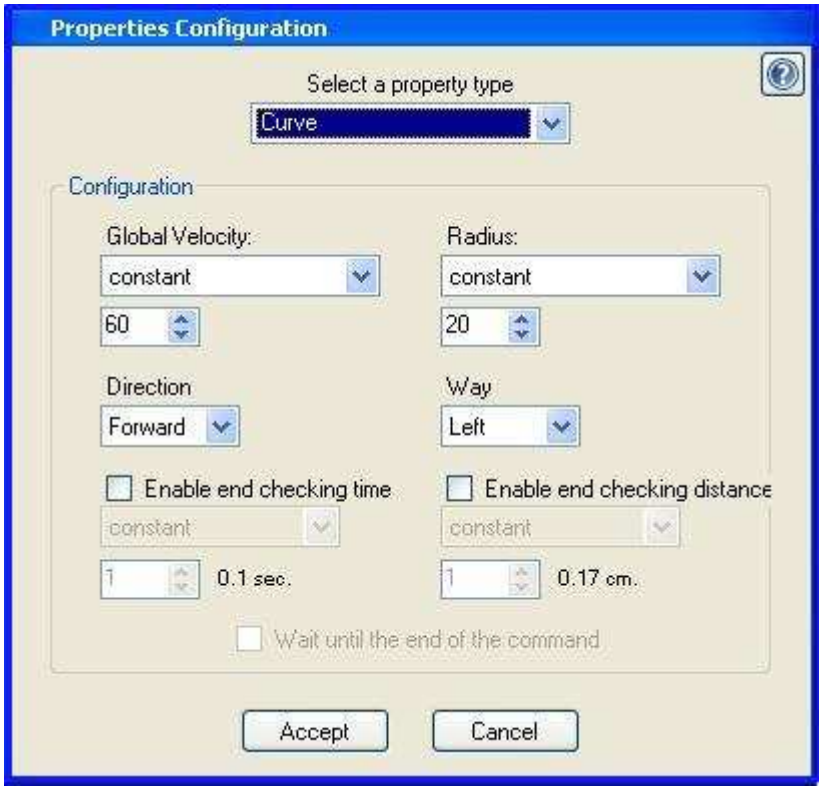


Image 77. Choose the speed, the curvature radio, the direction, the way and the time or the distance

- **Rotation**

The rotation module is another specific case of the movement module. With this command, mOway will rotate either on its centre or on one of its two wheels. We can configure the turning direction and the rotation speed.

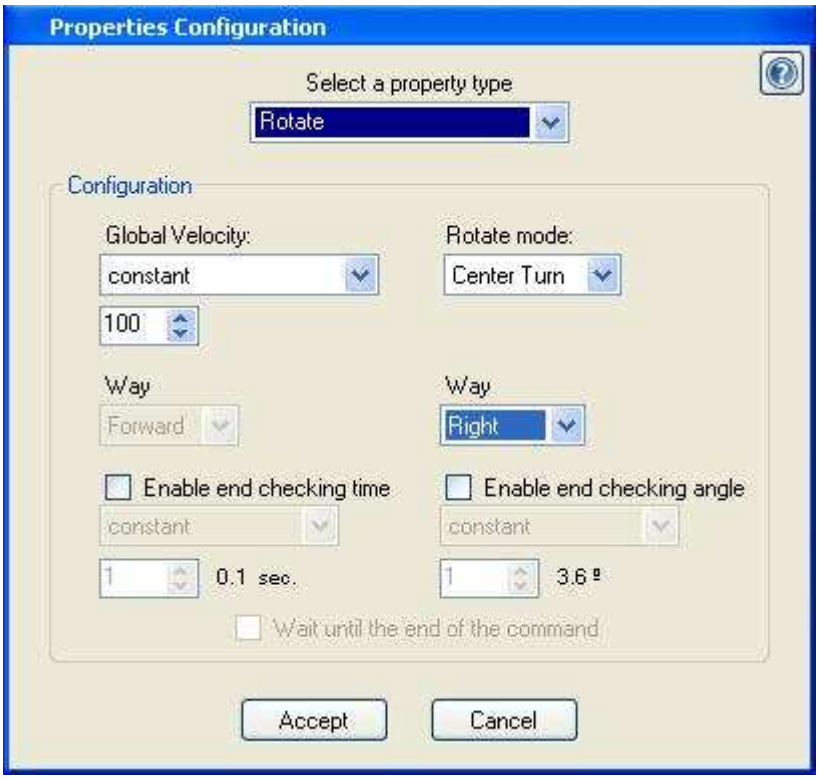


Image 78. Choose the speed, the rotate mode, the way and the time or the distance

We can also use here the time restrictions and turning angle as well as the "Wait until end of command" option.

- **Stop**

The Stop module is another specific case of the movement module. With this command, mOway will stop moving.

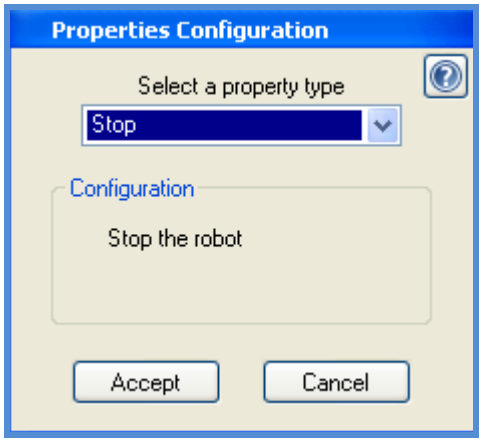


Image 79. Stop mOway

- **Consult movement info**

mOway keeps a record of the data of the movements it makes. This module enables you to consult this record: Current speed, Distance covered, Angle of turn, etc.

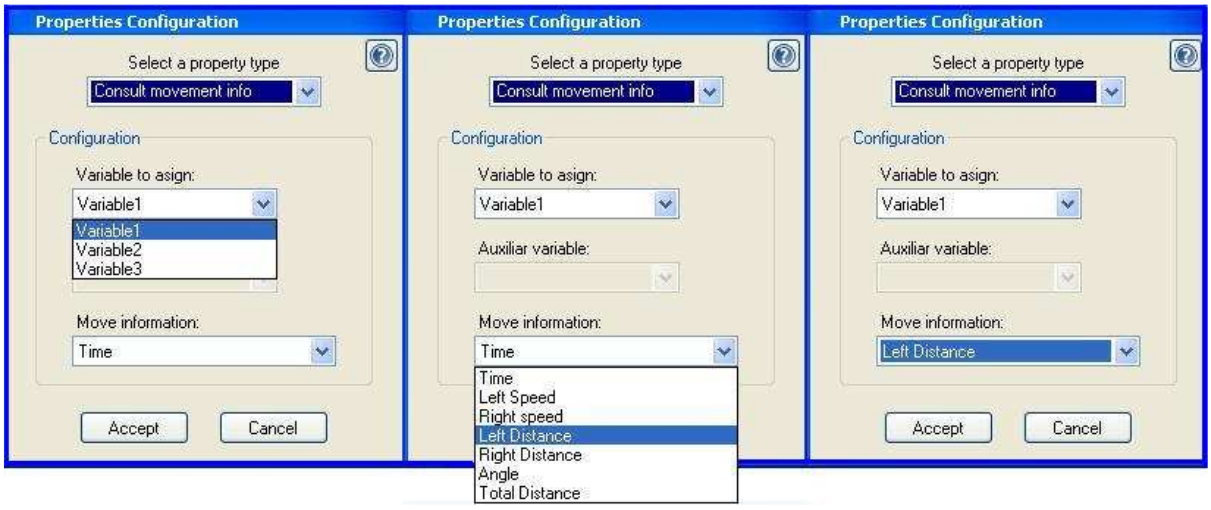


Image 80. Choose the variable and the move information

- **Reset movement data**

This module resets stored movement counters. You can select the specific pieces of data you wish to reset.

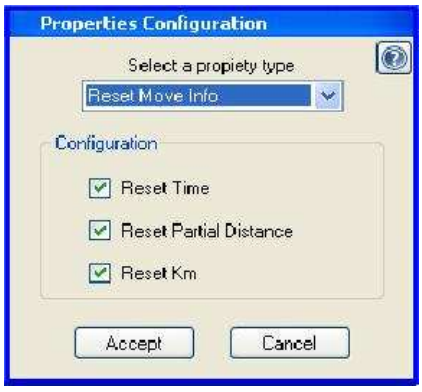


Image 81. Choose the options to be reset

RADIO FREQUENCY

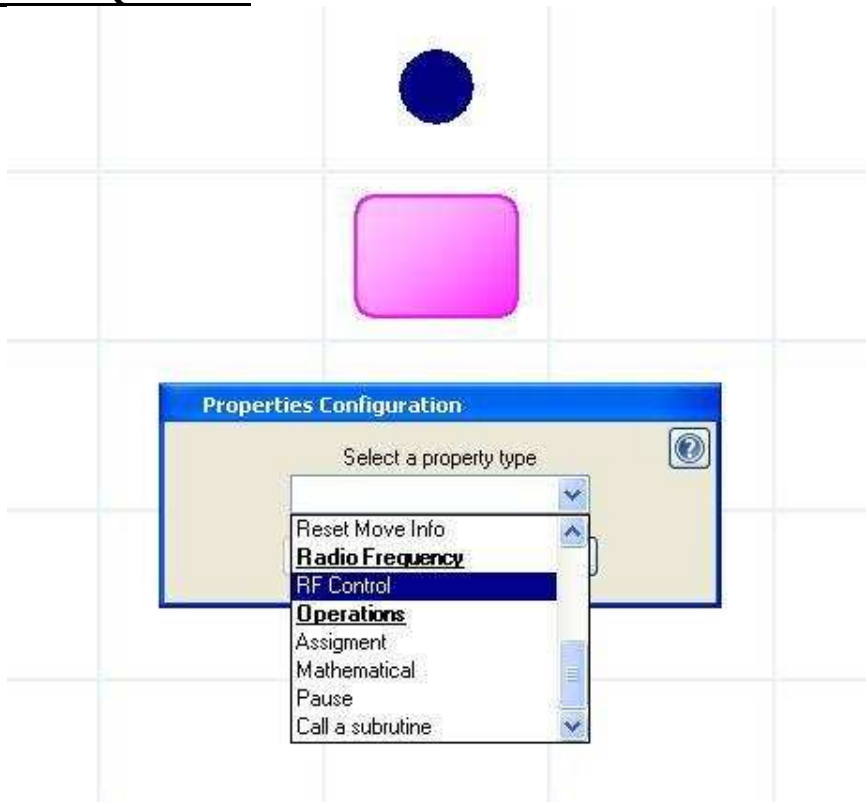


Image 82. Choose the type of module (Radio Frequency)

- **RF Control**

This activates or deactivates the RF module. When you activate this control, you must select which address the robot will use and in which channel you wish to operate. So that two mOways can communicate between each other, they must operate on the same channel.

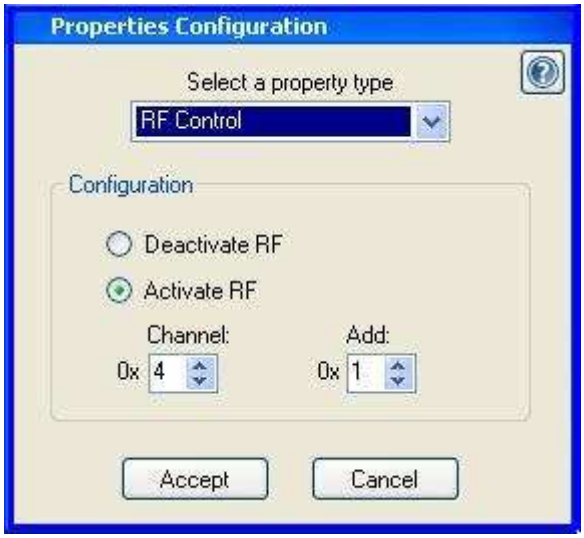


Image 83. Choose the channel and the address

OPERATIONS

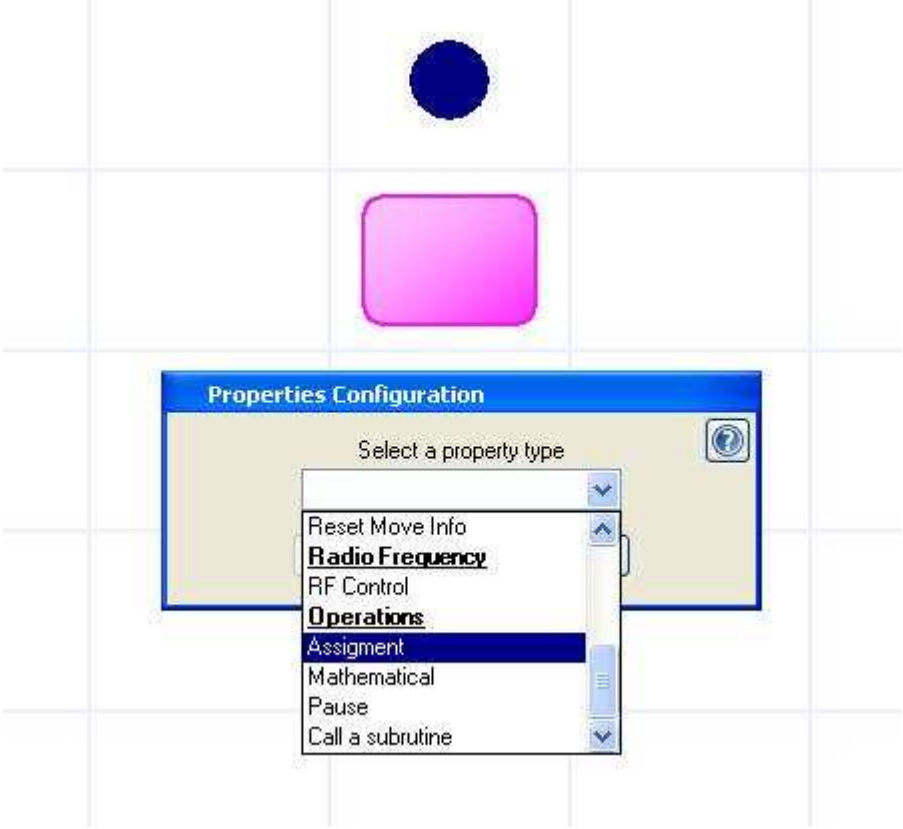


Image 84. Choose the type of module (Operations)

- **Assignment**

This function is to assign a value (a constant or variable) to a previously created variable. This variable can be used to configure different aspects of the robot.

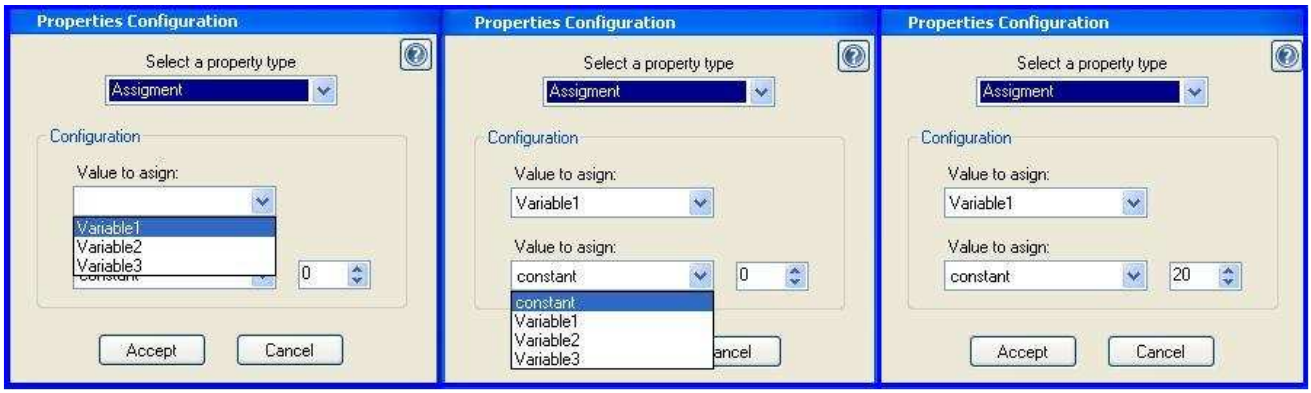
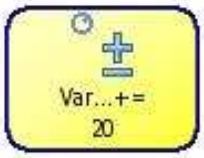


Image 85. Choose the variable and the value to assign

- **Mathematical**

This is used to carry out arithmetic operations to add or subtract to or from a variable. The first parameter must be a variable in each case and this will be used to store the result of the operation. The second operand may be a constant or a variable.



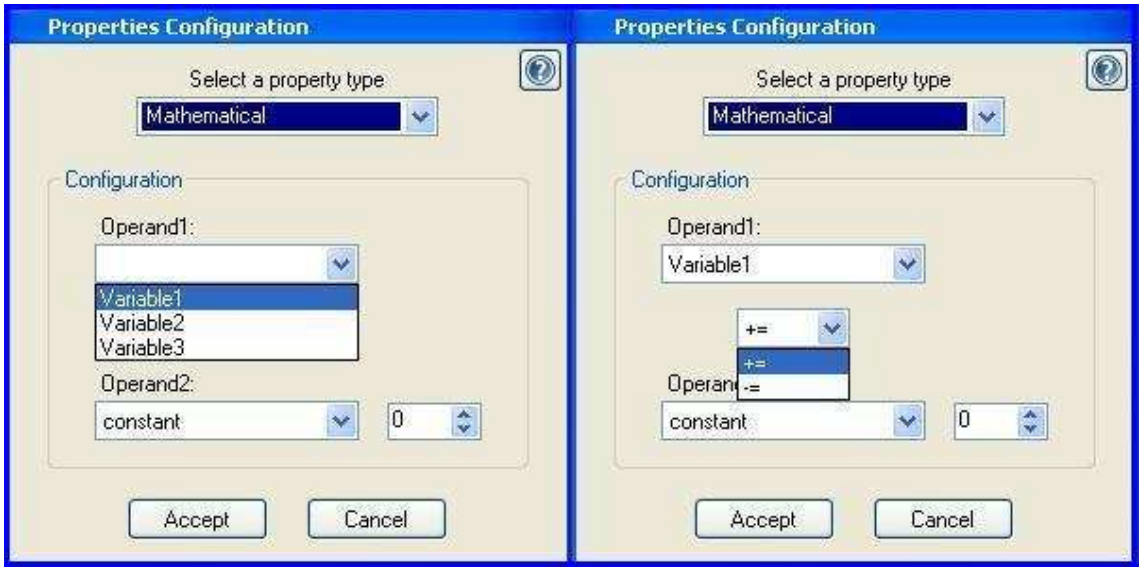


Image 86. Choose the operand1 and the operation

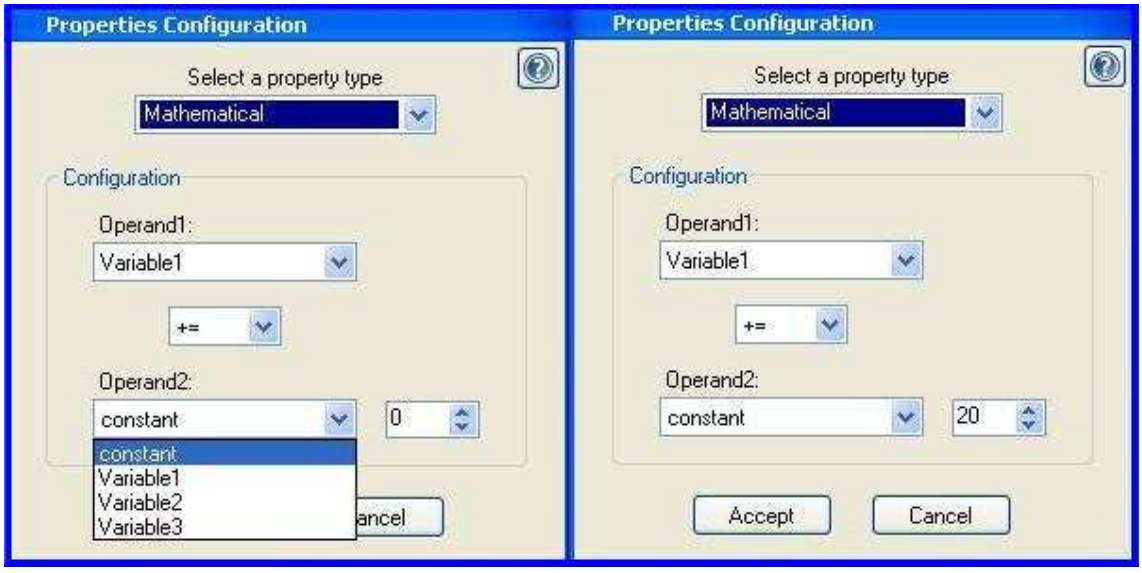
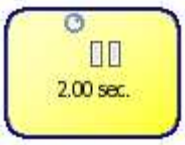


Image 87. Choose the operand2

- **Pause**

This allows you to insert a pause in the programme with a duration set in multiples of 0.05 seconds. The pause parameter may be a constant or a variable.



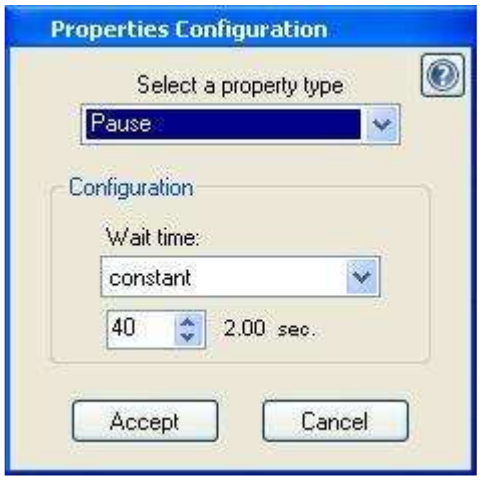


Image 88. Choose the time

- **Subroutine call**

Subroutine call, which can be reused in other mOway projects.

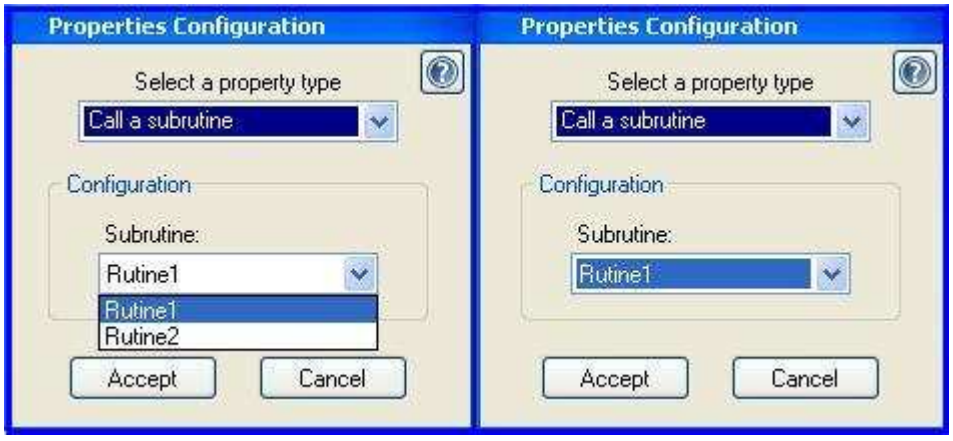


Image 89. Choose the subroutine

7.3.2. Conditionals

Conditionals are actions in which the output is important in order to operate with them: comparisons, sensor verification, etc. The following actions can be carried out with modules:

Image 90. Set of mOwayGUI conditionals

SENSORS

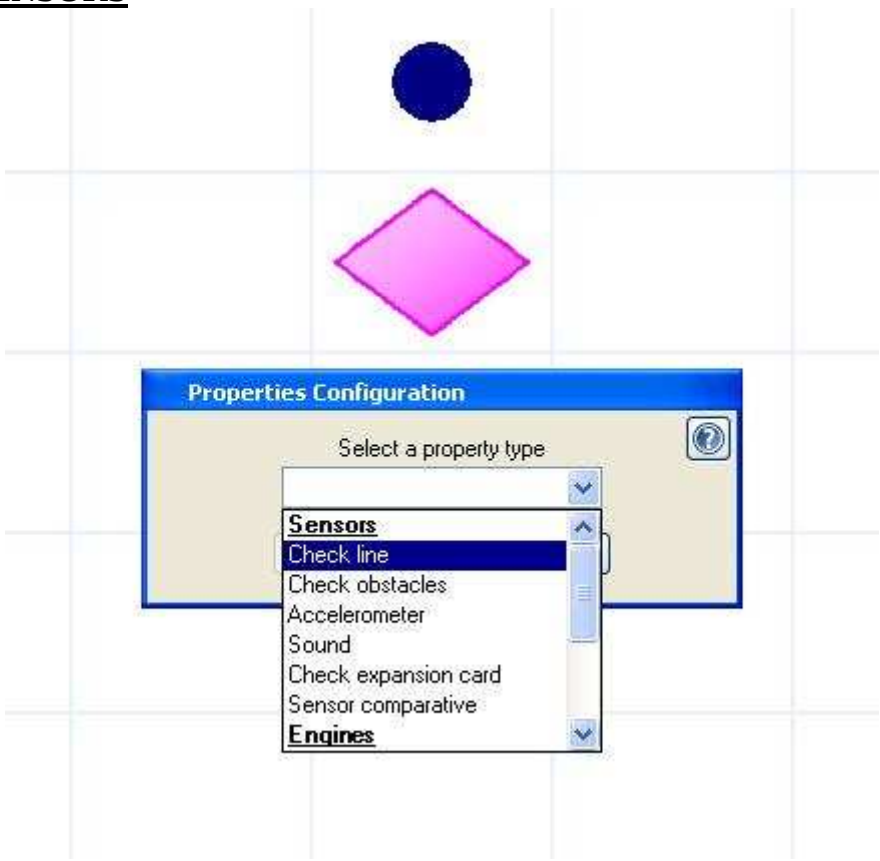


Image 91. Choose the type of conditional (Sensors)

- **Check line**

It checks the digital value of line sensors. This module is very useful for making mOway follow a line (black or white) on the floor,

detect boundaries, etc. User has to check the AND or OR boolean operation. With AND option both conditions must be true and on the other hand, with OR one of the two condition has to be true.

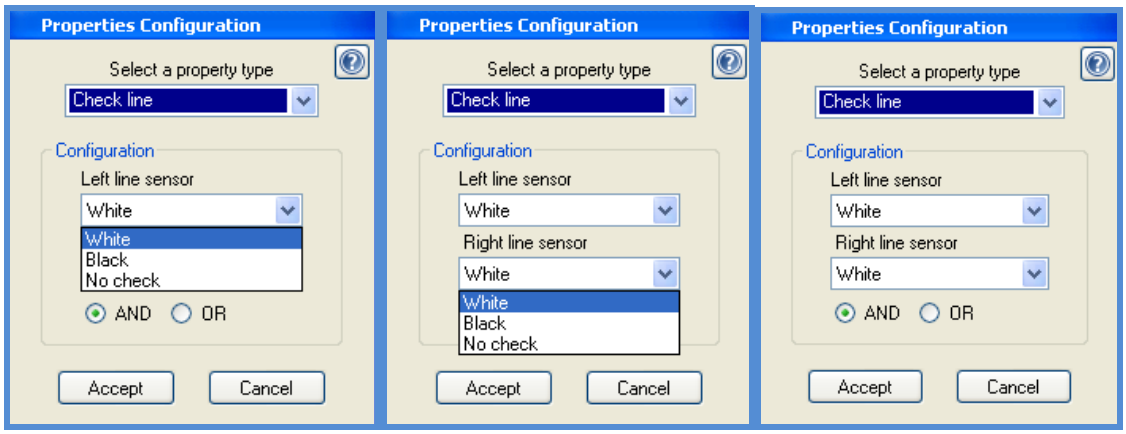


Image 92. Choose the left and the right line sensors: both have to be white

- **Check obstacles**

It checks the digital value of obstacle sensors. It is used to determine whether there is an obstacle or not in front, to the left or right. User has to check the AND or OR boolean operation. With AND both conditions must be true and on the other hand, with OR one of the two condition has to be true.

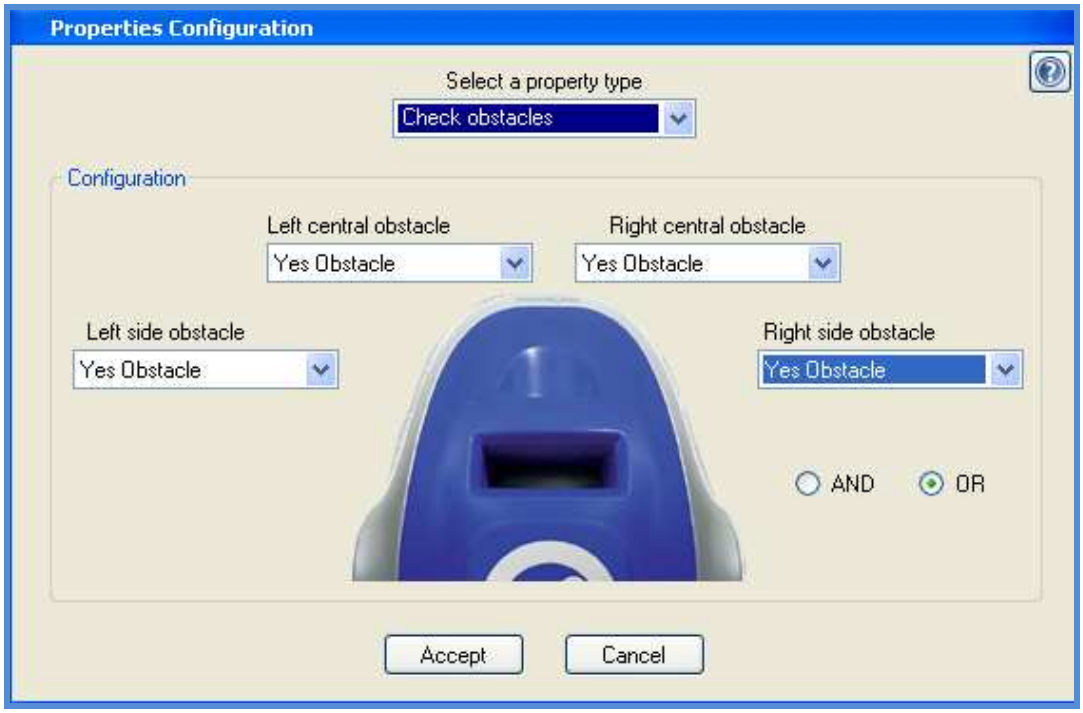


Image 93. If one of the sensors detects an obstacle: true output

- **Accelerometer**

With the accelerometer you can check if the mOway has been beaten once (tap) or twice (Double tap).

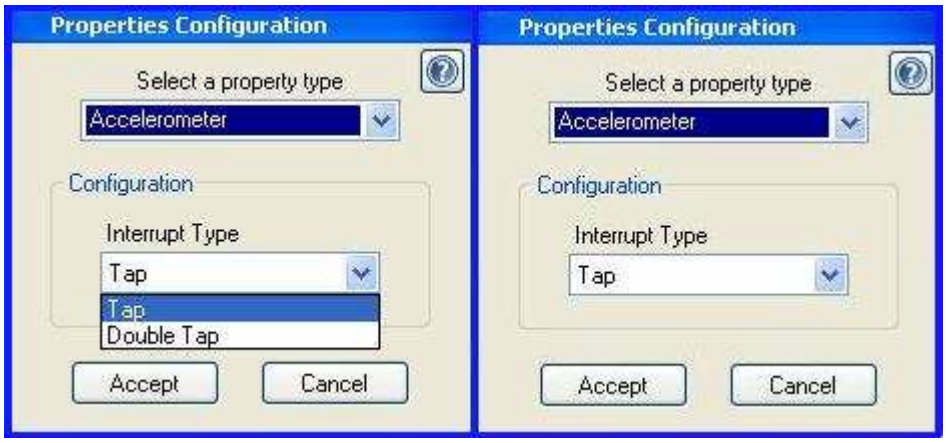


Image 94. Choose the interrupt type

- **Sound**

This sensor checks if there is a loud sound or not.

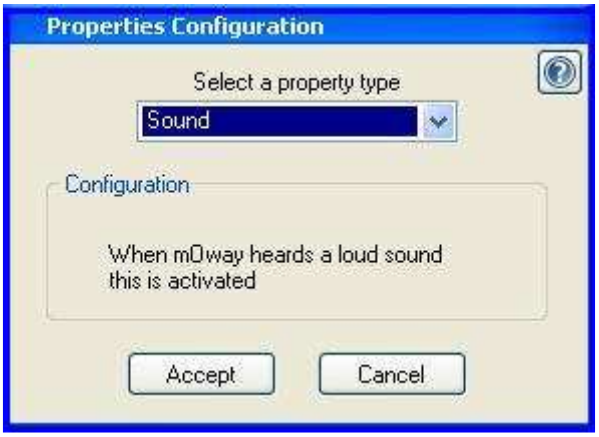


Image 95. Choose to check if there is a sound

- **Check expansion card**

This checks the digital value of the expansion connector pin.

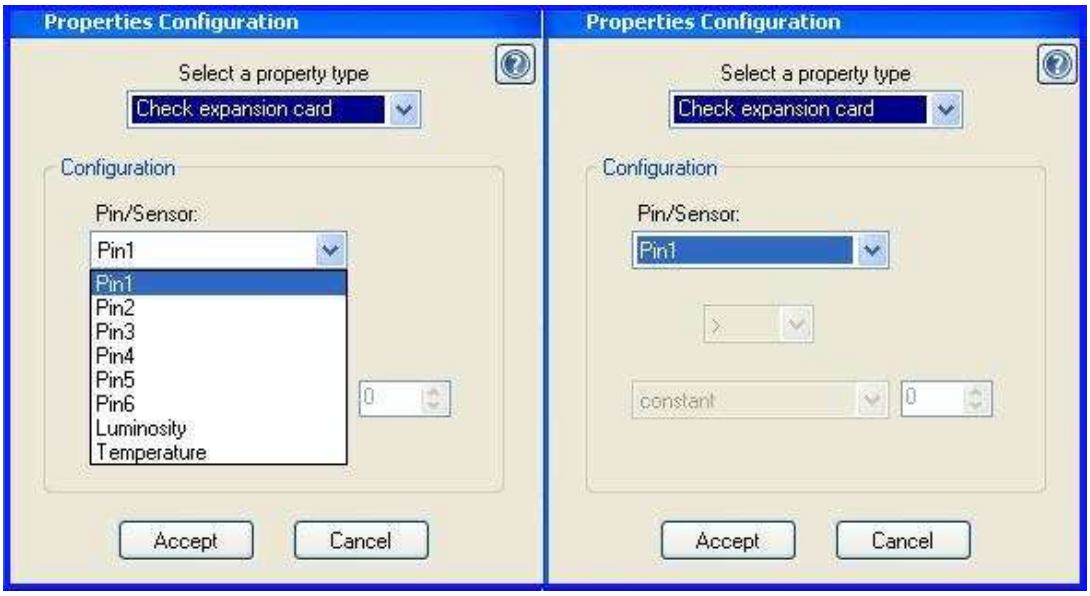


Image 96. Choose to expansion connector pin

WARNING!

Only advanced users can use the pinout configuration. Any incorrect connection of electronic elements to the expansion connector may damage the robot irreversibly.

- **Sensor comparative**

It compares the analogue value of obstacle, line, battery, temperature, accelerometer or microphone sensors. All mOway sensors return an analogue value. For example, the light sensor gives a value of 0 to 100 according to the intensity of the incident light, and obstacle sensors give a value of 0-255.

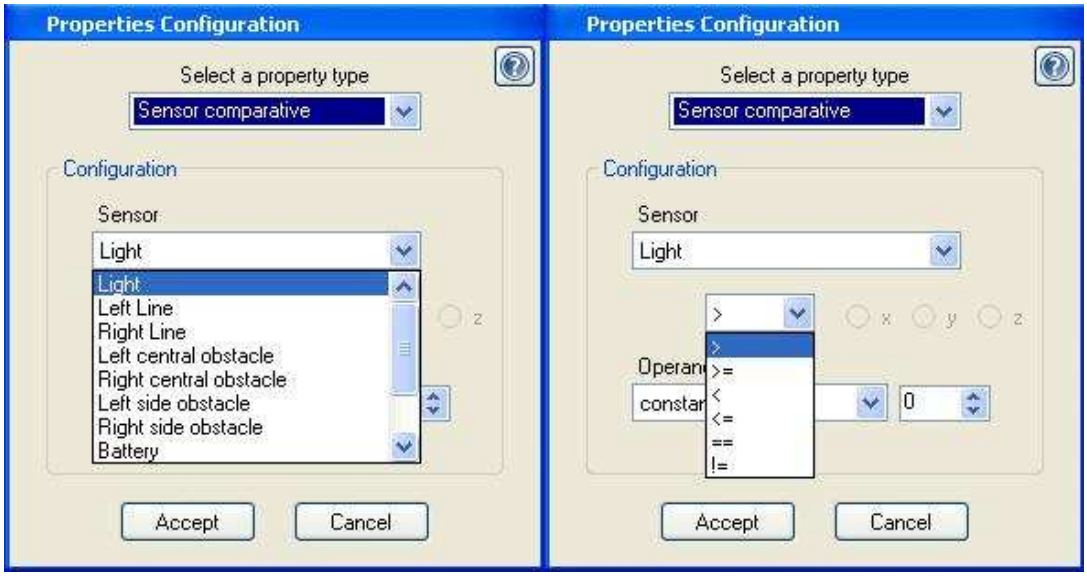


Image 97. Choose the sensor and the comparative

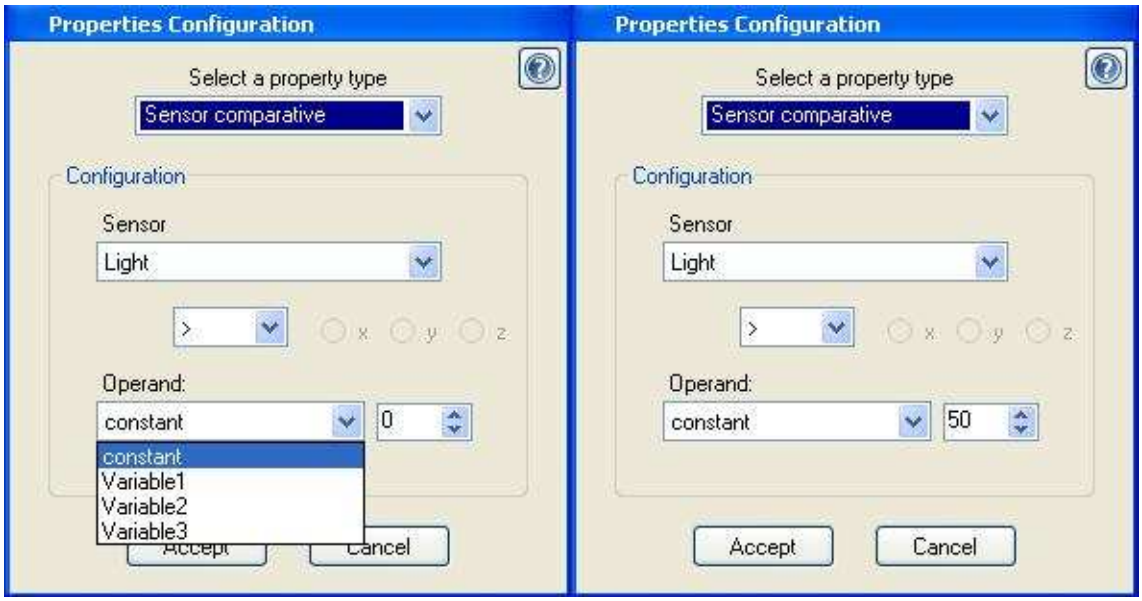


Image 98. Choose the variable to be compared

ENGINES

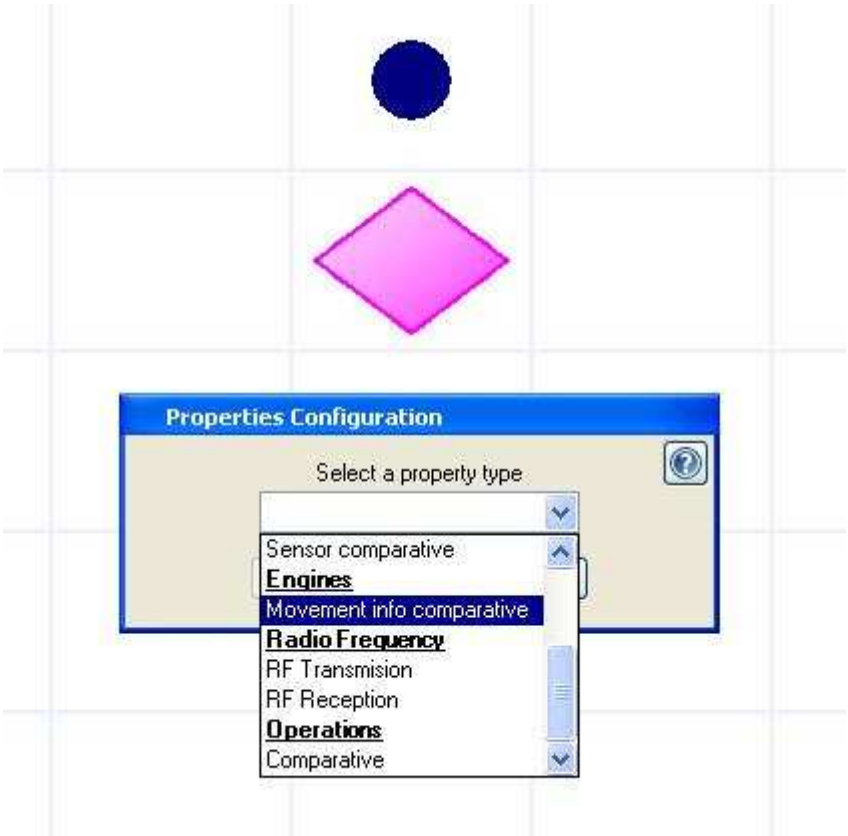


Image 99. Choose the type of conditional (Engines)

- **Movement info comparative**

A comparison is made with the information from the drive system. The drive system provides information about the total distance covered, partial distance, etc.

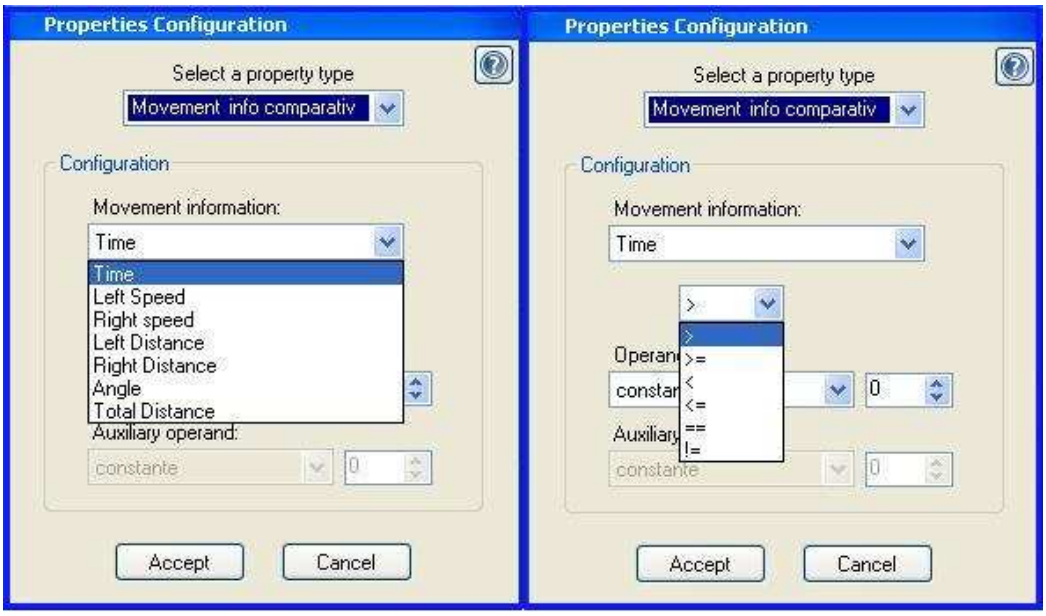


Image 100. Choose the movement information and the comparative

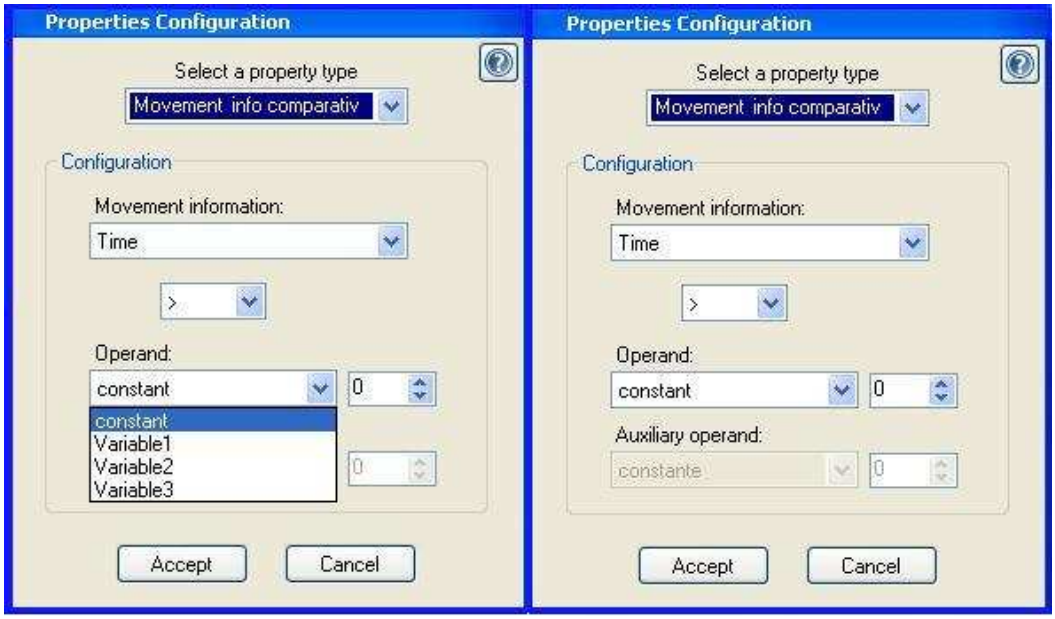


Image 101. Choose the variable to be compared

RADIO FREQUENCY

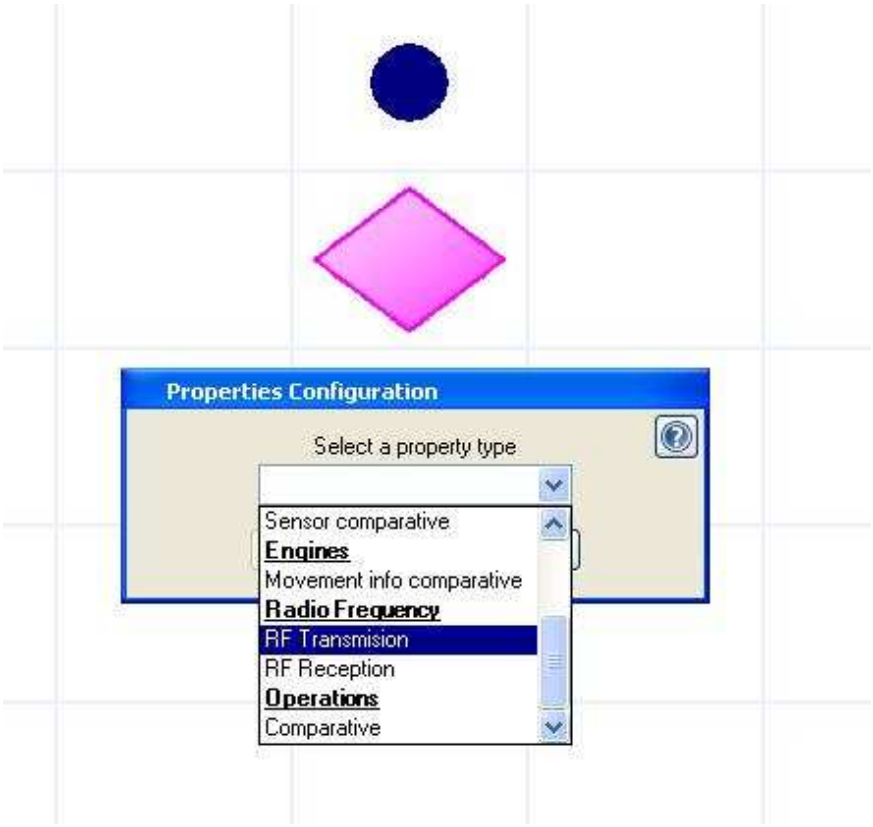


Image 102. Choose the type of conditional (Radio Frequency)

- **RF transmission**

It transmits a frame to a specific address. The address of the recipient and the data, which can consists of constants or variables, must be indicated in the frame. It must be remembered that before inserting this conditional, the module must be configured using the "RF Control" module. Remember that all the robots taking part in the RF communication must have the same channel and different addresses.

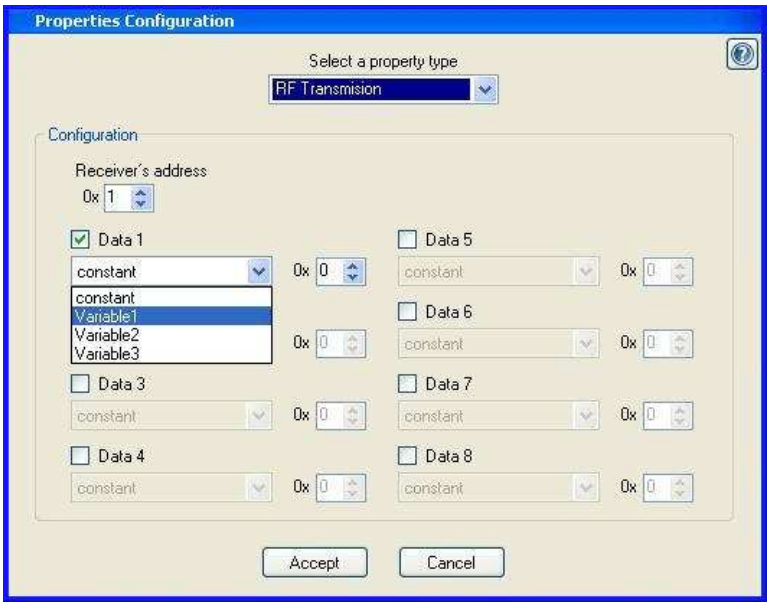


Image 103. Choose the first data to be transmitted

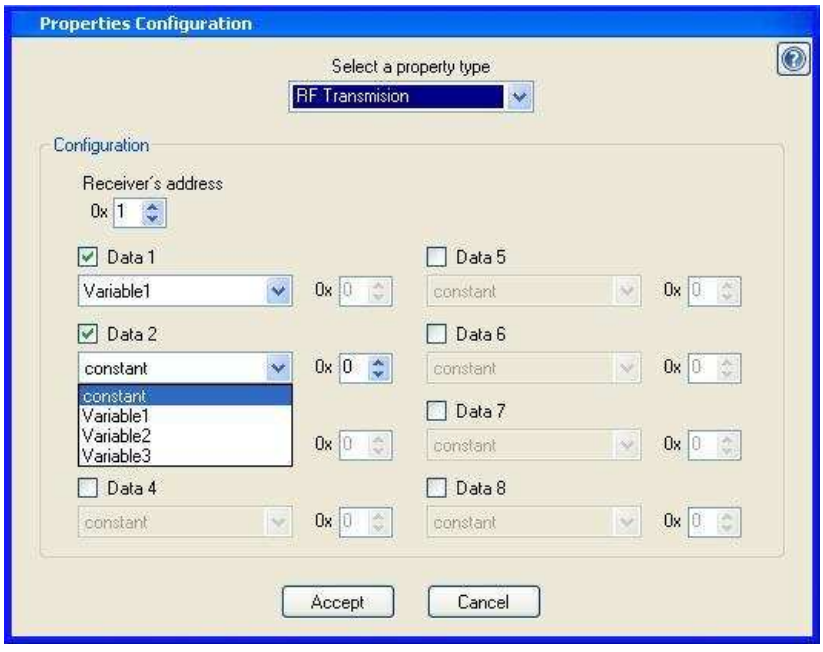


Image 104. Choose the second data to be transmitted

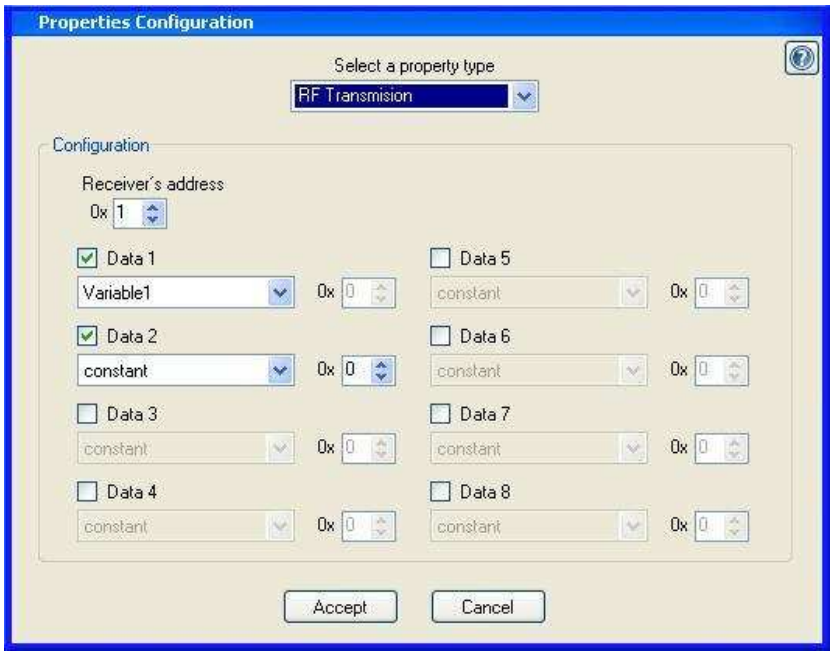


Image 105. First and second data ready to be transmitted

- **RF Reception**

It receives a frame from a specific address. It must be indicated at least two variables: one for collecting the transmitter address and the other for the data. It must be remembered that before this conditional, the module must be configured using the "RF Control" module. Remember that all robots taking part in the RF communication must have the same channel and different addresses.

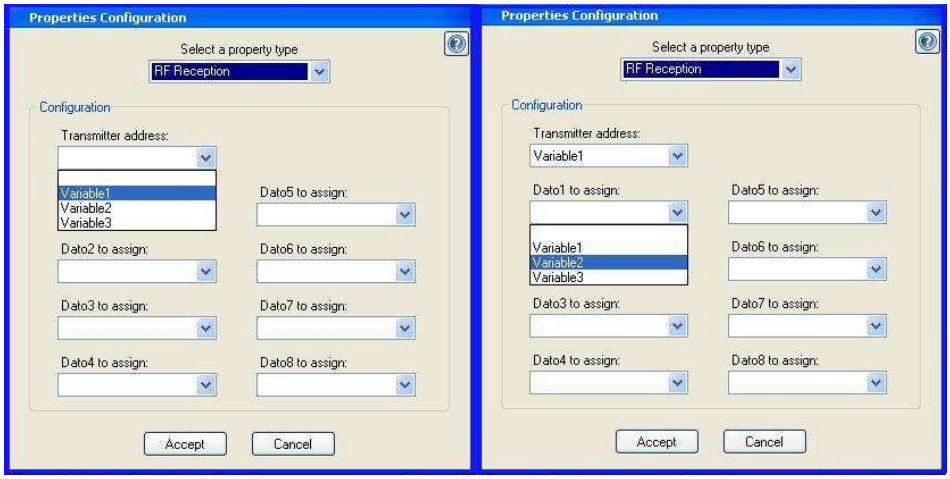


Image 106. Choose the address of the transmitter and the first data to transmit

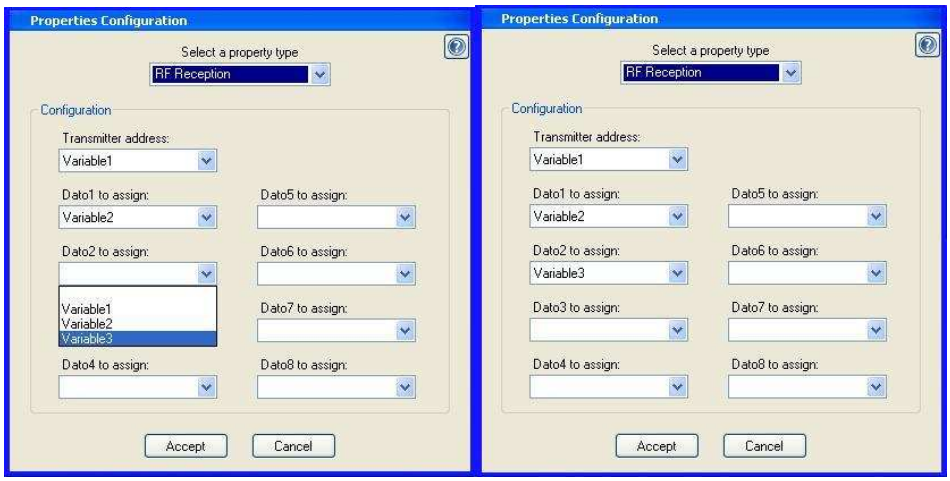


Image 107. Choose the second data to transmit

OPERATIONS

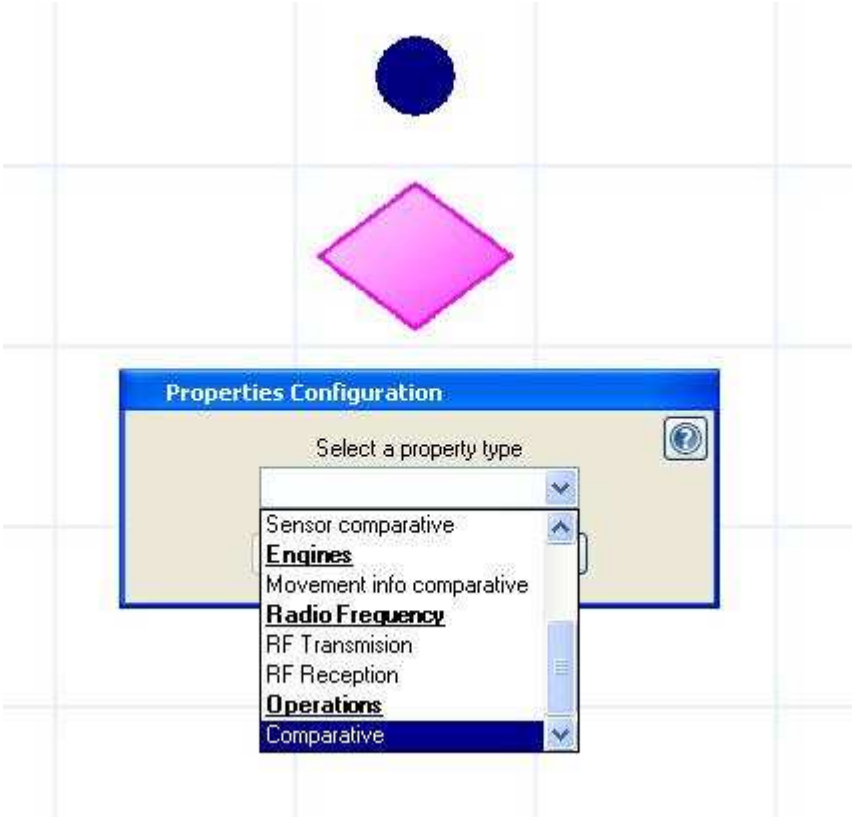


Image 108. Choose the type of conditional (Operations)

- **Comparative**

It makes a comparison on a variable. The variable can be compared with a constant or with another variable. This is very useful when comparing a variable used to carry out a mathematical operation.

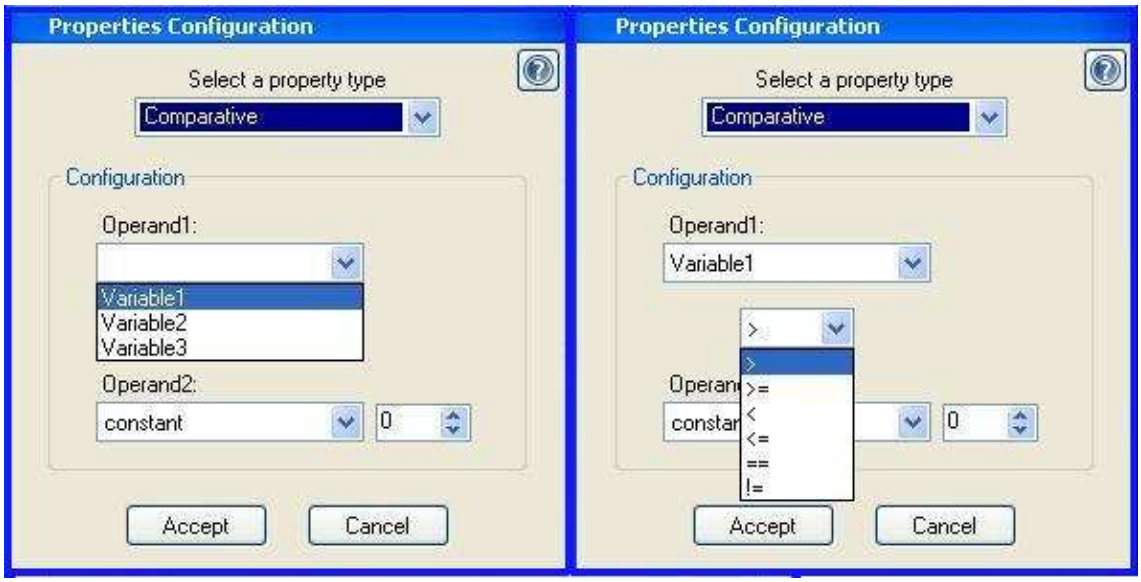


Image 109. Choose the first operand and the comparative

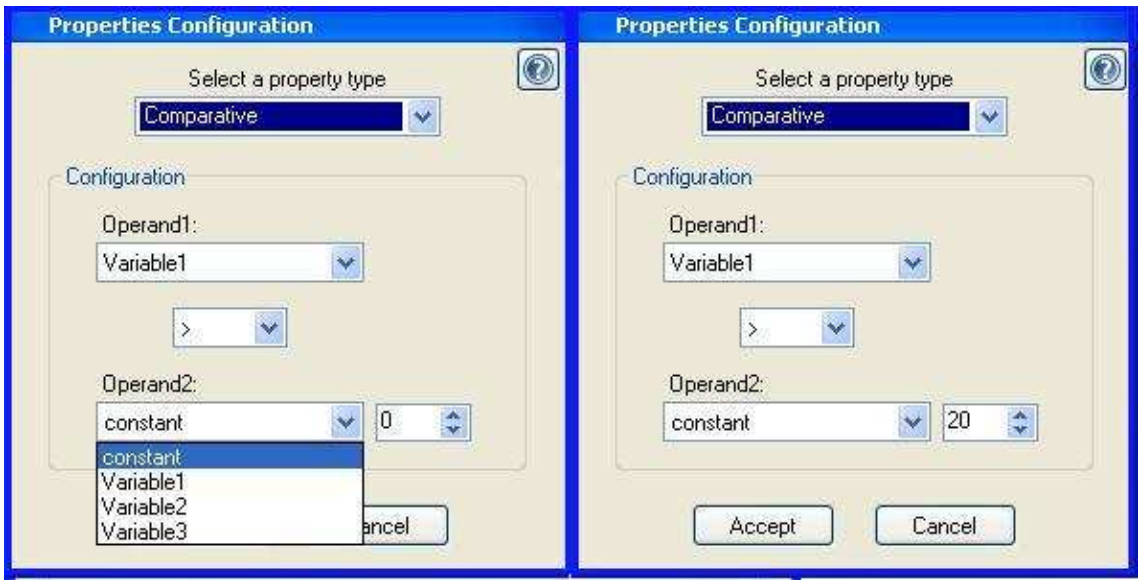


Image 110. Choose the second operand and the comparative

7.3.3. Start and End

Any program must have a Start element, but it does not need to have an End element (an infinite loop can be created).

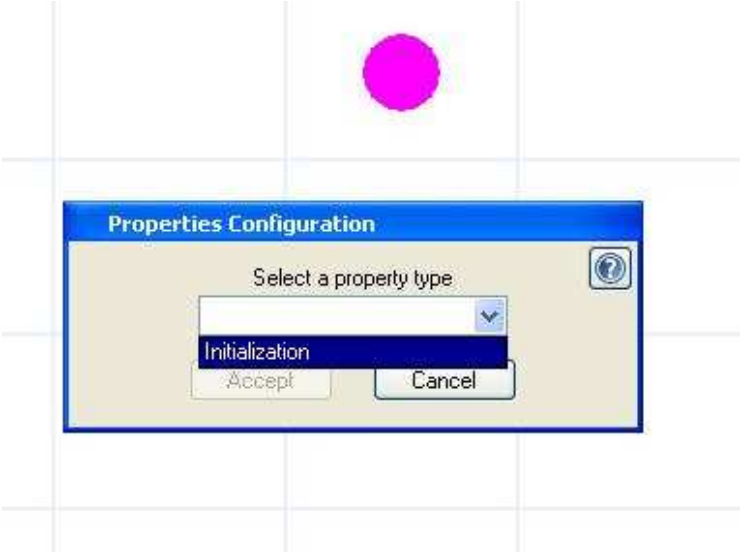


Image 111. Choose the option initialization

The Start element can initialize the variables.

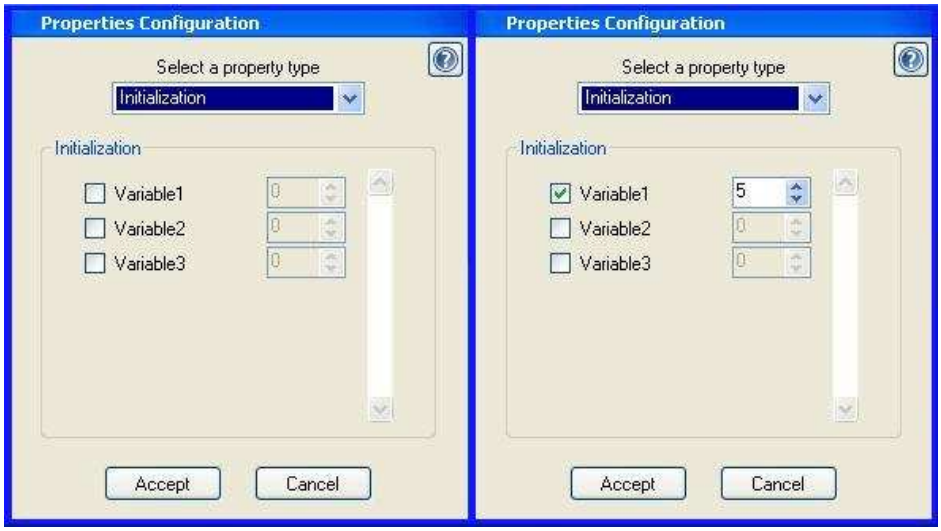
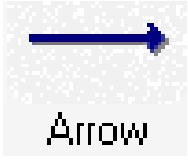


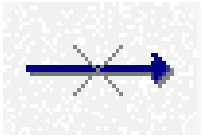
Image 112. Choose the variable and the value for the initialization

7.3.4. Arrow

Arrows are used to join Modules and Conditionals in order to create the program flow diagram. The same program indicates the user if the arrow is correctly positioned to ensure that the application operates correctly.



7.3.5. Erase Arrow



This tool is very useful to create the diagram. Erase Arrow is used to delete the link between the element and the next one. It can be done clicking on the element.

7.3.6. Subroutines

In order to simplify the diagrams and optimise the use of programme memory, reusable subroutines can be generated. In other words, if a part of the programme is repeated with great frequency, a subroutine can be created with this task, and replacing it in the main diagram with a subroutine module.

7.3.7. Recording

mOwayGUI can be used to record the diagram in the robot directly. The status of the recording process will be indicated at the top.

8. Moway RC Center

Moway RC Center is an application included in Moway's Pack to control Moway as if it was a radio control device and to monitor all the robot's sensors. This tool, which uses RF BZI-RF2GH4 modules and RFUsb (mOway Base is compatible), is very useful for all those users wishing to explore the field where the microbot will perform.

Its functioning concept is as follows: the application transmits commands by means of the USB to the RFUsb, which transmits them to Moway, where a recorded program interprets those commands (Moway_RC_Client included in Moway Pack).

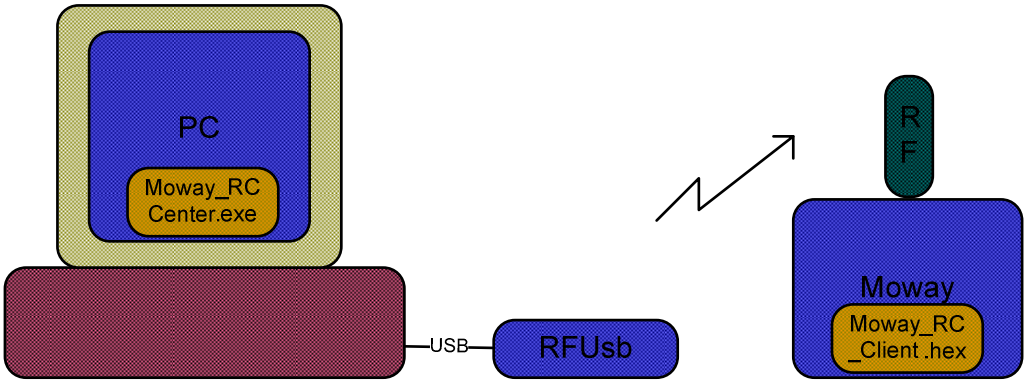


Image 113. Moway RC Diagram

Image 114. Moway RC Center

You can access the application in Window/Radio Control or using the button shown in the next image.

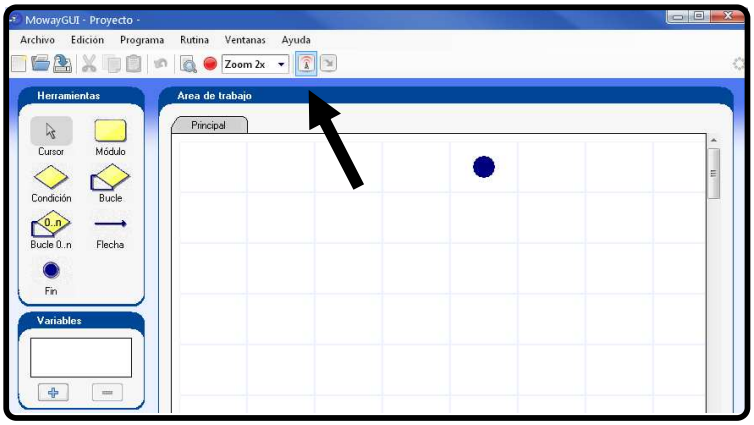


Image 115. Access from mOwayGUI

8.1. Description of the mOway RC Center

The different parts of the program are described below.

8.1.1. RF configuration

In this part the RFUsb module is configured with a 0x02 default address and 0x00 for the communications channel (default mOway RC program channel). The Radio Control's indicator (dispatch status) on the left lower part will glow red when configuring the module with the robot switched off.

Once connected, the communications channel can be changed if WI-FI, Bluetooth, Microwaves, etc. interferences are detected in this first channel. Click on the change channel button to select up to 16 channels. To change the channel the robot has to be switched on and be in communication with the RFUsb.

Every time the RFUsb is disconnected the default channel shall be 0x00.

The recommended procedure is as follows:

1) Turn the robot on
2) Connect the *RFUsb*
3) Test the channel sending mOway commands
4) If the robot does not react well change the channel and try again

8.1.2. Radio control

Once the RFUsb is connected mOway can be sent commands. The robot's movements can be controlled by means of the buttons and the keyboard.

There also are two bars to determine the speed and turning radius.

When mOway captures the transmitted data a small green indicator will light on the lower left side. On the contrary, when no data is captured its color changes to red.

8.1.3. LED

In this section Moway's four LEDs are switched on and off.

8.1.4. Speaker

In this section is checked the switched on and off of the robot´s speaker in a particular frecuency

8.1.5. Info

Displays information about ***Moway RC Center.***

8.1.6. Sensor status

This section describes the values returned by the sensors at all times (updated every second).

1) Analog value from obstacle sensors: higher when the object is closer.
2) Percentage of inciding light.
3) Values from line tracking sensors: higher when the terrain is dark.
4) Distance covered by the robot after being switched on or the distance is reset.
5) Accelerometer values.
6) Battery meter
7) Temperature of mOway

8.1.7. Keyboard control

The keyboard controls:

W-Forward
A- Left
S- Back
D-Right