

MOWAY'S **USER MANUAL**



	MOWAY	Title: mOway User Manual Rev: v3.1.2 – April 2013 Page 2 of 175
---	--------------	--

Copyright (c) 2013 Bizintek Innova, S.L.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 2.0 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Index

Index	3
1. Prologue.....	6
2. What is mOway?	8
3. Robot mOway.....	9
3.1. Processor.....	9
3.2. Drive system	10
3.3. Sensor and indicators group.....	12
3.3.1. Line sensors.....	14
3.3.2. Obstacle detection sensors	16
3.3.3. Light sensor.....	17
3.3.4. Expansion connector	17
3.3.5. Temperature sensor	18
3.3.6. Speaker.....	19
3.3.7. Microphone	19
3.3.8. Accelerometer	19
3.3.9. Battery level	20
3.3.10. Front LED.....	20
3.3.11. Top two-color LED	20
3.3.12. Brake LED.....	21
3.3.13. Free Pad.....	21
3.4. Power Supply System	22
3.5. RF module and RFUSB	22
3.5.1. Technical specifications	24
3.6. mOway Camera module	25
3.7. mOway Camera Board.....	26
3.7.1. Technical specifications	27
3.8. Wifi Module.....	29
3.8.1. Connection to the robot.....	29
3.8.2. Software	30
SPI communication	¡Error! Marcador no definido.
Technical characteristics.....	¡Error! Marcador no definido.
4. First Steps	33
4.1. mOway Pack installation	33
4.2. Download a program to mOway.....	34
4.3. RFUSB installation	35
4.4. RF modules	37
4.5. mOway Videocap drivers installation.....	38
5. Programming mOway in assembler	39
5.1. Creating a project.....	39
5.2. First program in assembler	43
5.3. Libraries	47
5.3.1. mOway's sensors library in assembly language	47
5.3.1.1. Description	48
5.3.1.2. Variables.....	48

5.3.1.3.	Functions	51
5.3.2.	mOway's motor library in assembly language.....	60
5.3.2.1.	Description	61
5.3.2.2.	Variables	62
5.3.2.3.	Functions	63
5.3.3.	BZI-RF2GH4 library in assembly language	71
5.3.3.1.	Description	71
5.3.3.2.	Variables	72
5.3.3.3.	Functions	73
5.3.3.4.	Flow diagram for sending and receiving data	79
5.3.4.	mOway Camera Board library in assembly language.....	81
5.3.4.1.	Description	81
5.3.4.2.	Variables	81
5.3.4.3.	Functions	82
6.	Programming mOway with C18 Compiler.....	85
6.1.	Creating a project.....	85
6.2.	First program in C18.....	89
6.3.	Libraries	93
6.3.1.	mOway's sensors library in C18	93
6.3.1.1.	Description	93
6.3.1.2.	Functions	94
6.3.2.	mOway's motor library C18	103
6.3.2.1.	Description	103
6.3.2.2.	Functions	104
6.3.3.	BZI-RF2GH4 library in C18	110
6.3.3.1.	Description	110
6.3.3.2.	Functions	110
6.3.3.3.	Flow diagram for sending and receiving data	116
6.3.4.	mOway Camera Board library in C18	118
6.3.4.1.	Description	118
6.3.4.2.	Functions	118
7.	MowayWorld programming.....	121
7.1.	MowayWorld workspace	121
7.1.1.	Toolbar	121
7.1.2.	Flowchart Editor.....	121
7.1.3.	Tools.....	122
7.1.4.	Properties.....	123
7.1.5.	Error List	123
7.1.6.	Arrows.....	124
7.1.7.	Language change and updates.....	125
7.2.	First program in MowayWorld	126
7.3.	Modules	130
7.3.1.	Moway Actions	130
7.3.2.	Sensors Check	134
7.3.3.	Data	136
7.3.4.	Flowchart Control	139

7.3.5.	Expansion	139
7.4.	Variables	144
7.5.	Functions / Subroutines	145
8.	Applications.....	147
8.1.	Communications Window	147
8.2.	Moway Cam.....	148
8.3.	MowayRC	150
8.3.1.	RF configuration	151
8.3.2.	Movements	151
8.3.3.	LED	151
8.3.4.	Speaker	152
8.3.5.	Sensor status	152
8.3.6.	Camera	152
8.4.	MowayServer.....	152
9.	Simulation.....	155
9.1.	Introduction.....	155
9.2.	Functioning	156
9.3.	Simulation example	160
10.	mOway Scratch	162
10.1.	Introduction	162
10.2.	Functioning.....	162
10.3.	Step by step	163
10.4.	Commands and Sensors	167
10.5.	Exercises.....	171
10.5.1.	Movement by radio control	171
10.5.2.	Radio control LEDs	172
10.5.3.	Follow the line	173

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.

	MOWAY	Title: mOway User Manual Rev: v3.1.2 – April 2013 Page 7 of 175
---	--------------	--

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

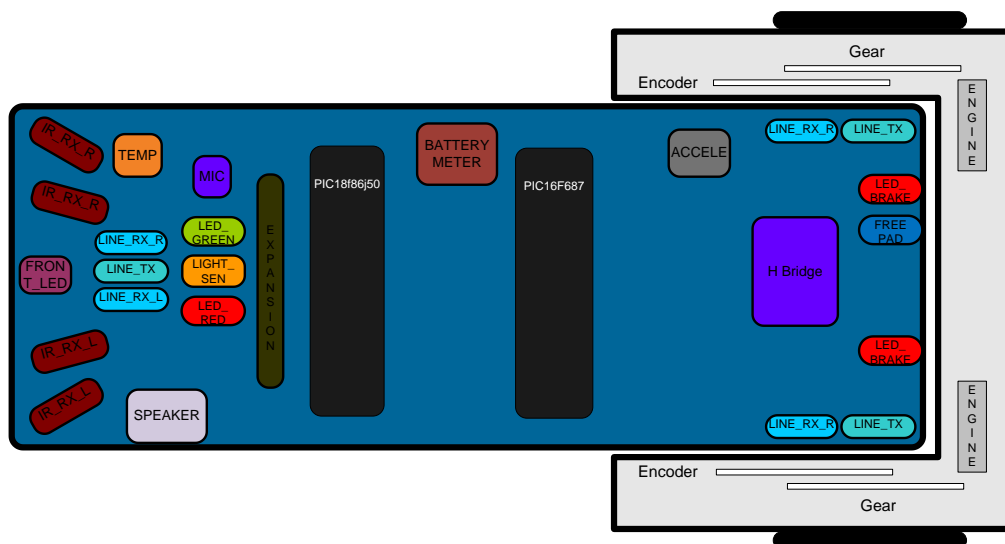


Image 1. Diagram of mOway's parts

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 signal for the SPI communication(accelerometer)
RD6	O	Clock signal 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	Temperature 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.

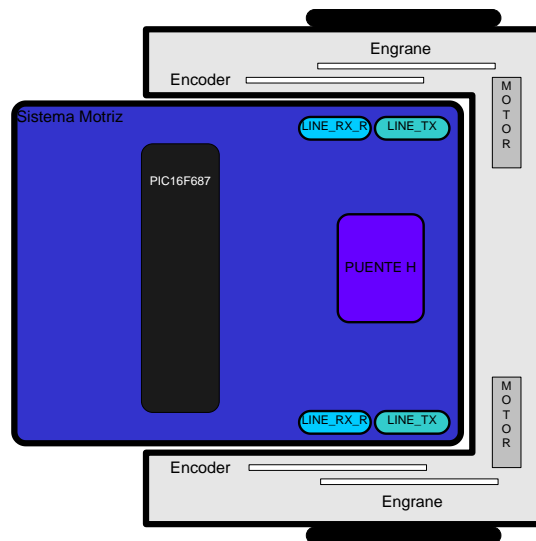


Image 2. Drive system: electronic and mechanical

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.

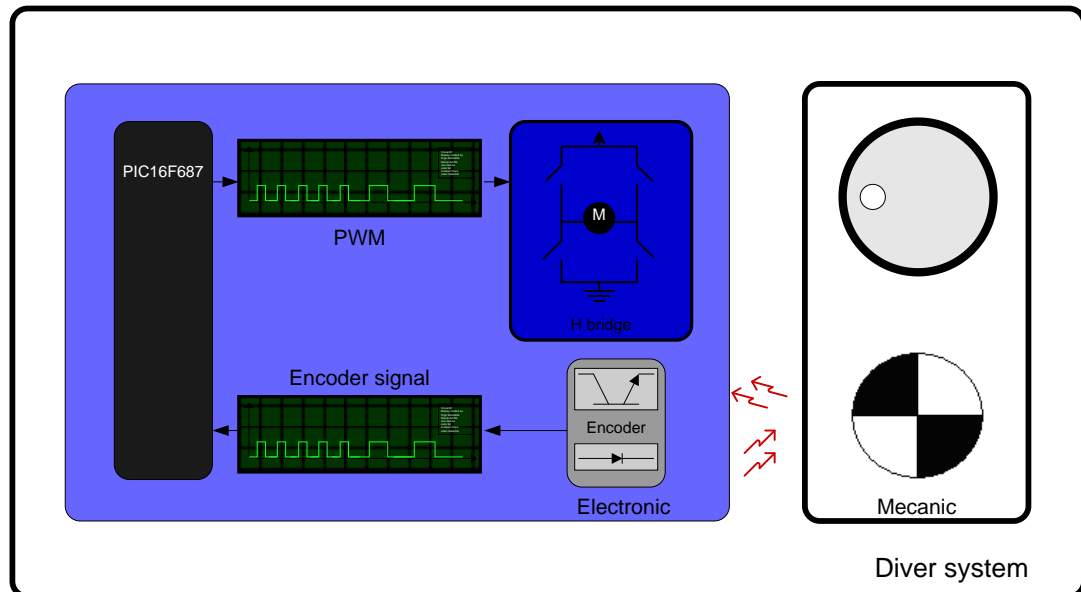


Image 3. Motor control

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.

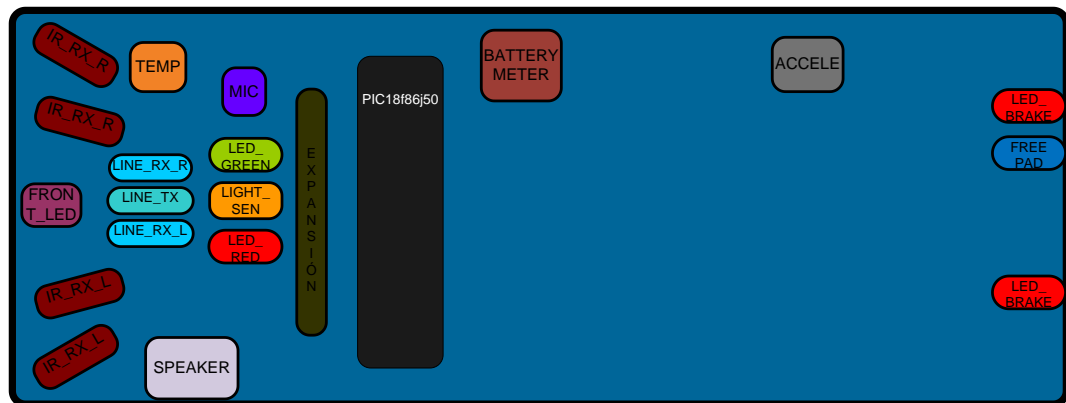


Image 4. Sensors and indicators group

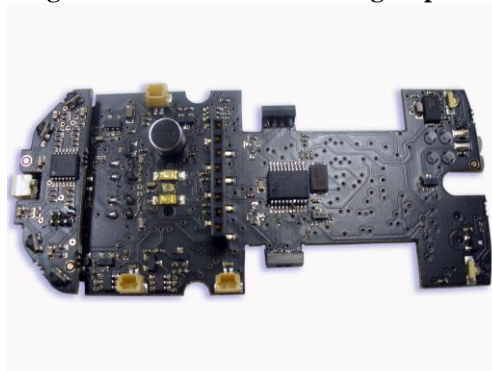


Image 5. Top-view of mOway's PCB

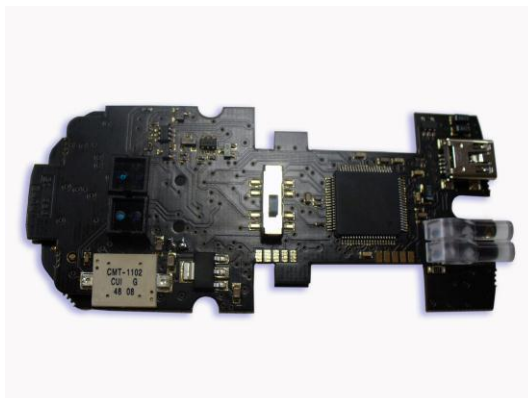


Image 6. Down-view of mOway's PCB

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.

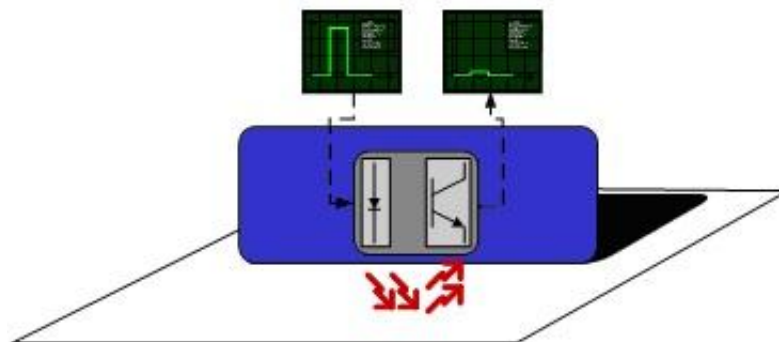


Image 7. Line tracking sensor on a clear surface.

- **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 identified¹.

¹ Due to CNY70 tolerance two different sensor can differ.

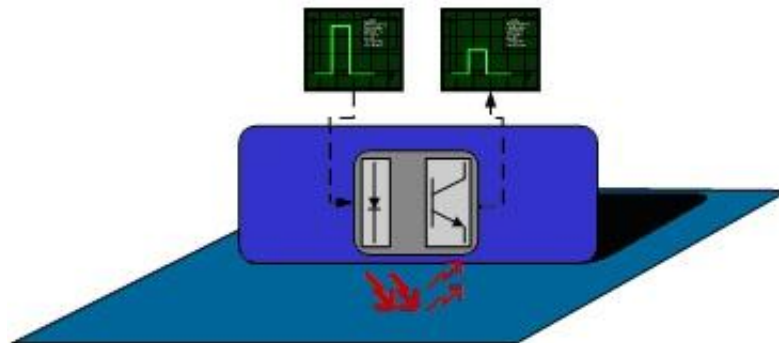


Image 8. Line tracking sensor on a colored surface.

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

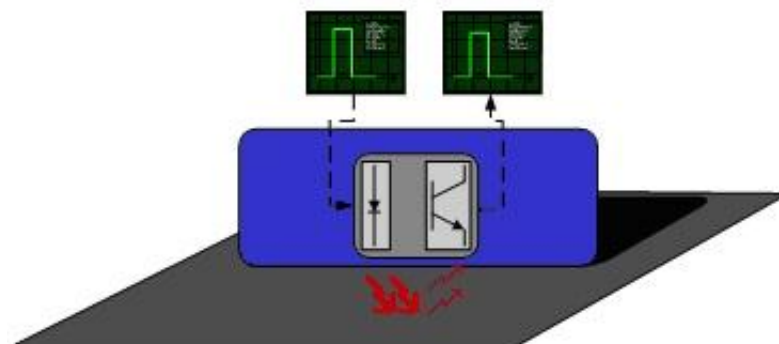


Image 9. Line tracking sensor on a dark surface.

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.

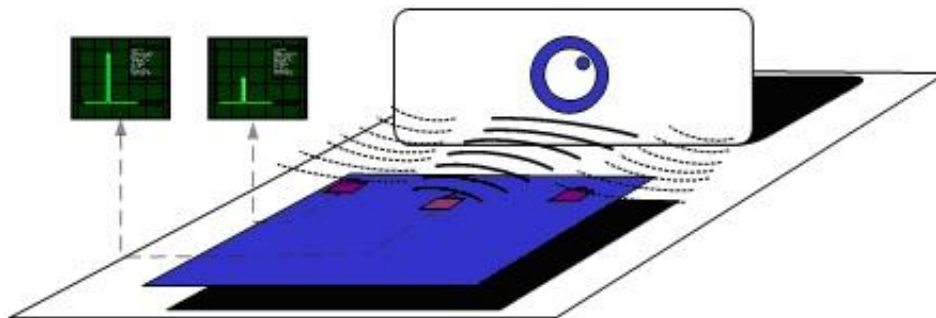


Image 11. Obstacle detection sensor

Table 4. Shock-proof sensor - PIC connections

Pin PIC	I/O	Sensor
PORTA		
RA1	I	Central right infrared receiver

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

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

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 reading³.

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

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

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



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 reading⁴.

Table 7. PIC-Temperature sensor connection

Pin PIC	I/O	Sensor
PORTH		
RH5	I	Temperature sensor

⁴ Temperature measured by the sensor can be 5°C higher than external temperature.

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.

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 outputs⁵.

Table 13. PIC-Top LED connection

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

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.



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.

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 LED⁶.

Table 15. PIC-free Pad connection

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

⁶ Advanced users only

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 robot⁷. 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 off⁸.



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

3.5. *RF module and RFUSB⁹*

RF module¹⁰ establishes communication between other mOways or with PC using RFUSB¹¹.

⁷ These LEDs can't be controlled by the user.

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

¹¹ Available in some packs



Image 18. RF module

RF module is connected in expansion connector and it is very easy to use with MowayWorld.



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	°C
Storage temperature	-40	+125	°C

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 <i>Power Down</i> mode	900	nA
Maximum frequency of the SPI bus	8	Mhz
Temperature range	-40 a +85	°C

Table 18. Pinout BZI-RF2GH4

Pins	N°	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

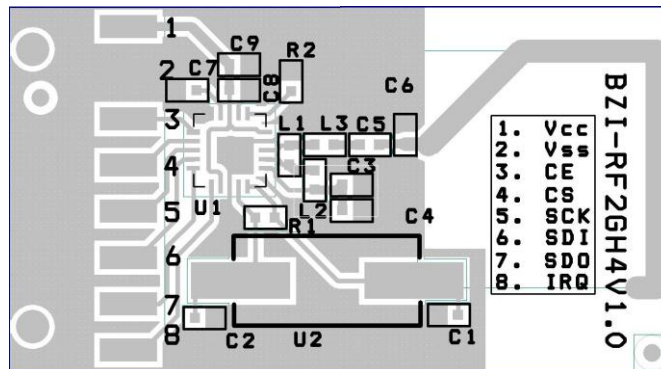


Image 20. RF Module layout

3.6. *mOway Camera module*

Thanks to the camera module (**mOway Camera Module**¹²) it is possible to display on the computer what mOway is “watching”. Camera board sends images wirelessly to the video receptor **mOway Camera Board**¹³.



Image 21. Camera module

Camera Module is connected to expansion connector of mOway robot. It is controlled by robot programming. It has a connector on the back, so that other circuitry can be attached (RF module or user circuitry).

Camera control is performed by Microchip **MCP23S08** device, which is an input/output port controlled by SPI. The basic functions are the following:

¹² Available in some packs

¹³ Available in some packs

- Turn on: When camera is on, the LED of the module is on. The camera sends images through RF to video receptor.
- Turn off: When camera is off, the LED of the module is off. The camera RF transmission ends.
- Channel change: When the camera channel changes, the LED blinks.

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.

Other wireless transmitters can affect the images quality. If so, change the channel of transmission (both camera and video receptor).

NOTE: Both camera transmission and mOway RF module transmission are in the same frequency band. So that, when camera is active, mOway RF module reception distance decreases.

3.7. *mOway Camera Board*

mOway Camera Board is the video receptor. It receives the images from the Camera Module through RF and sends them to PC through USB.



Image 22. mOway Camera Board video receptor

It has a channel selector for choosing the RF communication channel with Camera Module (channels 1 to 4). Both Camera Module and video receptor must have selected the same channel.



Image 23. Channel selector

Video receptor is connected to a USB port of the PC. First time it is connected, a message for installing drivers can appear. In this case, select the option of installing software automatically.

NOTE: While video receptor is receiving images, the casing heats up. This is a normal functioning.

3.7.1. Technical specifications

Table 19. Maximum Ratings

Parameter	Min	Max	Unit
Vdd	-0.3	5.5	V
Vss		0	V
Current		125	mA
Power dissipation		700	mW
Operating temperature	-40	+85	°C
Storage temperature	-40	+125	°C

Table 20. mOway Camera Board specifications

Parameter	Value	Unit
Minimum supply voltage	1.9	V
Maximum supply voltage	3.6	V
Current in <i>Power Down</i> mode	1	uA
Maximum frequency of the SPI bus	10	Mhz
Temperature range	-40 a +85	°C

Table 21. mOway Camera Board pinout

Pines	N°	Description
Vcc	1	Supply voltage of the module
Vss	2	GND
CE	3	Chip Enable
CS	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

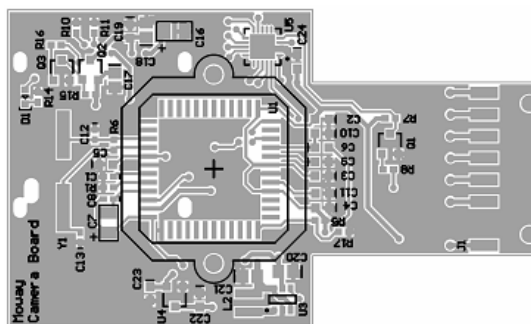


Image 24. Module layout

3.8. *Wifi Module*

Moway Wifi Module allows the robot to create a wireless net. So that the robot can communicate with other Wifi devices such as phones, tablets or PCs.

The Wifi Module allows to develop applications such as a web server embedded into the mOway robot. This server makes it possible to control the robot through user wifi devices, as well as to receive mOway sensor status in these devices.



Image 25. Moway Wifi Module and adapter

This module is based on **MRF24WB0MB** IC of Microchip. This IC is a 2.4 GHz RF transceiver, designed under IEEE 801.11b standard. mOway Wifi Module includes all the necessary elements to use this IC, so that the user does not have to design any additional circuitry.

3.8.1. *Connection to the robot*

In order to attach the Wifi Module to mOway robot is necessary to connect the adapter to the expansion connector of the robot. Then the Wifi Module can be attached to one of the two ports of the adapter.



Image 26. mOway Wifi Module and adapter

3.8.2. *Software*

The Wifi Module is managed by means of the TCP/IP stack developed by Microchip. It provides with all the required functions to implement a TCP/IP communication for the Wifi Module, as well as functions to control the module through SPI.

In order to understand how the TCP/IP stack works, an MPLAB project can be downloaded from mOway web page. This example consists of a web server embedded into the robot. The code is written in C. The mOway web server IP can be adapted in “TCPIPConfig MRF24WB0M.h” file.

In addition, the mOway Web Server can be programmed directly from the MowayWorld software.

3.8.3. *SPI communication*

Communication between mOway and Wifi Module is done through SPI (4 line SPI with interrupts). The SPI clock frequency is up to 25 MHz.

The slave SPI interface works with the Interrupt line (INT, pin 8). When data is available for mOway during operation, the INT line is asserted (logic low) by the **MRF24WB0MB** module. The INT line is de-asserted (logic high) by the **MRF24WB0MB** after the data is transferred to mOway.

Data is clocked in on the first rising edge of the clock after Chip Select (CS, pin 4) is asserted. Data is placed on the bus with most significant bit (MSb) first. The CS pin must be toggled with transfer blocks and cannot be held low permanently. The falling edge of CS is used to indicate the start of a transfer. The rising edge of CS is used to indicate the completion of a transfer.

Table 22. Wifi Module pinout

Pin	Name	Function
1	+VCC	Voltage
2	GND	Ground
3	RESET	Reset
4	CS	SPI Chip Select
5	SCK	SPI Clock
6	SDI	SPI Data Input
7	SDO	SPI Data Output
8	INT	SPI Interrupt

3.8.4. *Technical characteristics*

MRF24WB0MB IC is compatible with IEEE 802.11b/g/n. It supports the 802.1x, 802.1i security:

- WEP
- WPA-PSK
- WPA-2-PSK.

It has the radio regulation certification United States (FCC), Canada (IC), Europe (ETSI) and Japan (ARIB).

The transmission speed can be chosen between 1 Mbps or 2 Mbps. The RF range is up to 400 meters. The next image shows the radiation pattern of Wifi Module.

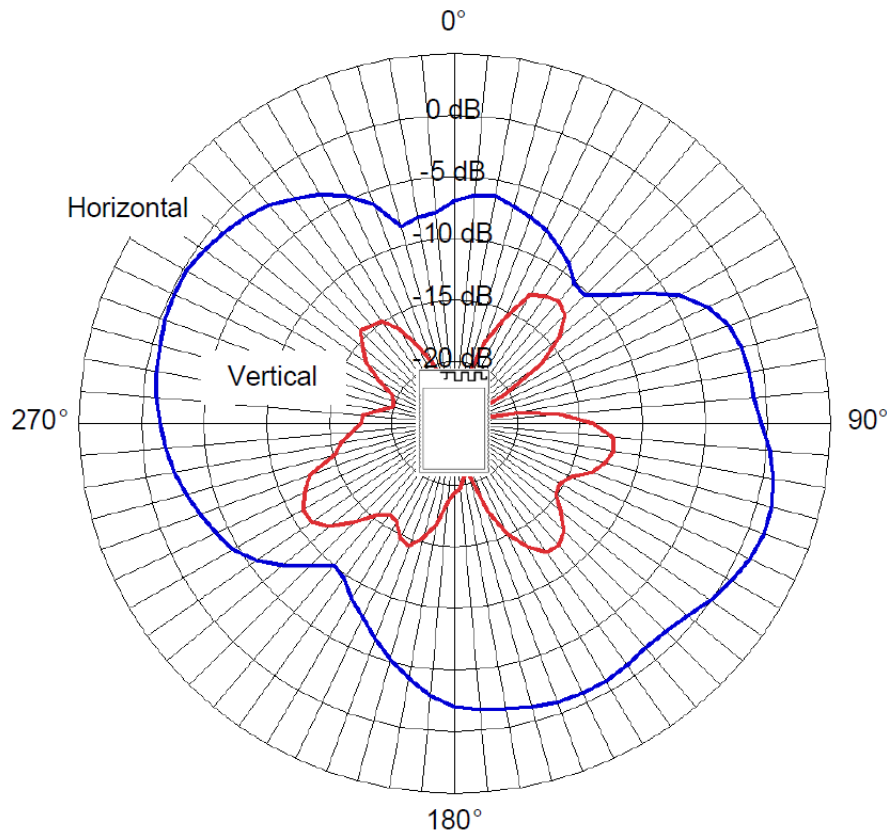


Image 27. Wifi Module radiation pattern

Table 23. Recommended operating conditions

Parameter	Min	Typical	Max	Units
Ambient temperature	-20	-	85	°C
VDD	2.70	3.3	3.63	Volts

Table 24. Current consumption (nominal conditions: 25°C, VDD=3.3V)

Parameter	Min	Typical	Max	Units
RX Mode	-	85	-	mA
TX Mode	-	154	-	mA
Sleep Mode	-	250	-	μA
Hibernate Mode	-	< 0.1	-	μA

Table 25. Receiver AC characteristics

Parameter	Min	Typical	Max	Units
Frequency	2412	-	2484	mA
Input level sensitivity (1 Mbps)	-91	-	-4	dBm
Input level sensitivity (2 Mbps)	-88	-	-4	dBm

Table 26. Transmitter AC characteristics

Parameter	Min	Typical	Max	Units
Frequency	2412	-	2484	mA
Average output power	-	+10	-	dB
Average power variation	-0.5	-	+0.5	dB

4. First Steps

4.1. *mOway Pack installation*

In mOwayPack (available in the webpage www.moway-robot.com or in the installation CD) you will find the software, mOway's libraries, test programs and documentation. 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.
- MowayWorld software.
 - This software controls all aspects of the robot: program download, battery charge control, radio control, RFUSB¹⁴ management and C or assembler programs download.
- Reference projects in assembler, C and MowayWorld.
 - Example projects to start working with mOway easily.
- RFUSB Driver
 - Driver for RFUSB that establishes the communication between robots and PC.
- mOway Camera Board Driver
 - Driver for mOway Camera Board¹⁵ that makes it possible to grab images from mOway camera to display on PC.

If a security warning message appears during installation, please click on "install driver anyway". MowayWorld software is safe.

¹⁴ Module not available in all kits

¹⁵ Module not available in all kits

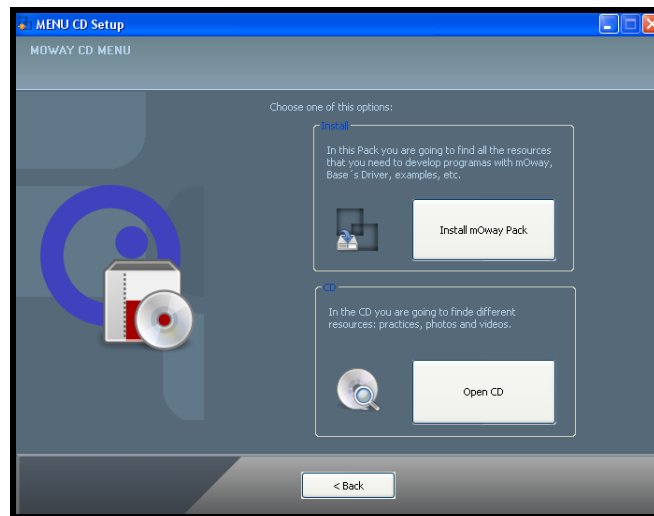


Image 28. CD menu

4.2. Download a program to mOway

Program download process is executed in MowayWorld. This application can download to the robot MowayWorld projects, assembler projects (compiled with MPLAB or 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 MowayWorld application.
- Open or create a project in MowayWorld, or import a ".hex" file from assembler or C project.
- Click on "Program mOway" icon. If a ".hex" file has been imported the download progress will start automatically.
- Disconnect the robot and turn it on.

mOwayPack provide several compiled projects to check sensors, drive system, RF communication and camera usage.

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 180°.
- mOway_first_project_C18: C18 project explained in this manual. mOway avoids obstacles rotating 180°.

4.3. *RFUSB installation*

- 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 29. Driver installation Wizard

- Now the installation process will begin.



Image 30. Windows XP driver installation

- Assistant will then indicate that the hardware is installed.



Image 31. 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 MowayWorld.
- Open *mOway_RF_send* project included in the pack.
- Click the Program button.
- Disconnect and switch the robot on.
- Configure RFUSB module using "Communications" window of MowayWorld with channel 0 and address 1.
- Check receiving data in MowayWorld.

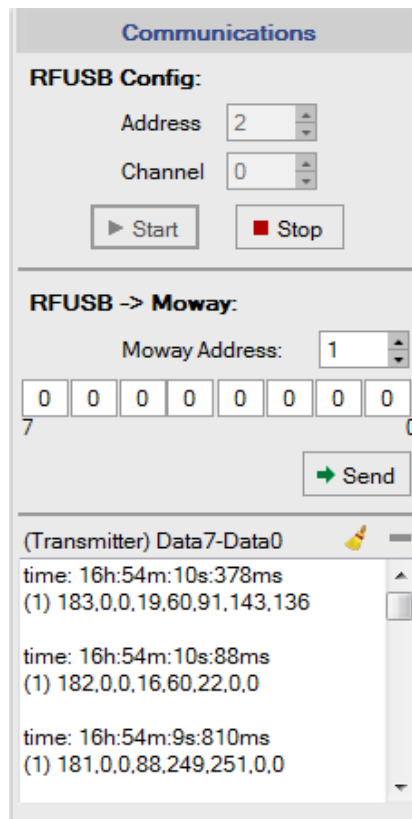


Image 32. Communications window

4.5 mOway Videocap drivers installation

As with RFUSB device, a driver that it's included in mOwayPack is required to use video capturer:

- The first time the **mOway Videocap** is connected, the PC will detect it as a new device. Driver installation runs automatically. If it doesn't, 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.
- Now the installation process will begin.
- Assistant will then indicate that the hardware is installed.
- Check if mOway's software has detected the **mOway Videocap**.

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.

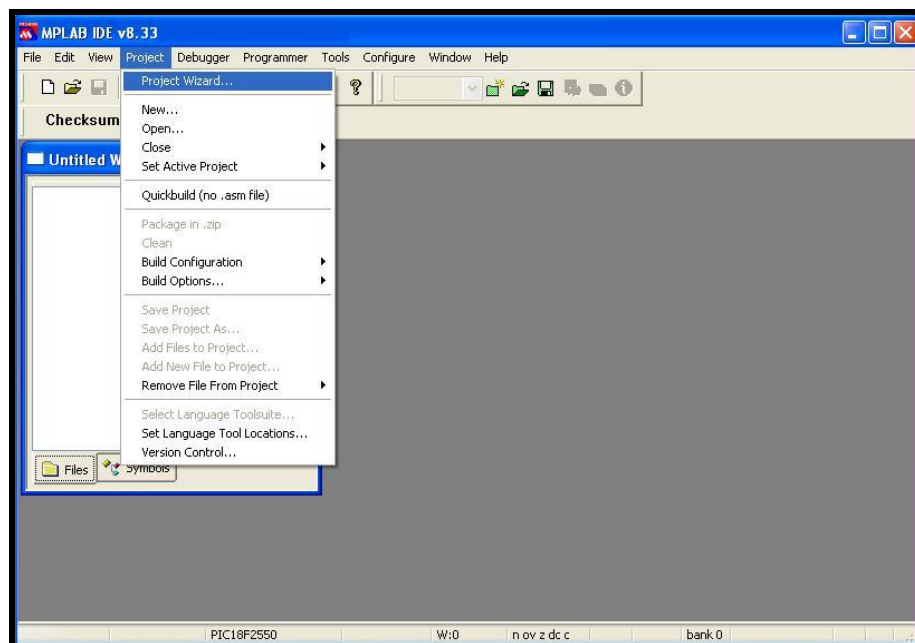


Image 33. Project Wizard

1. First select the PIC installed in mOway: PIC18F86J50

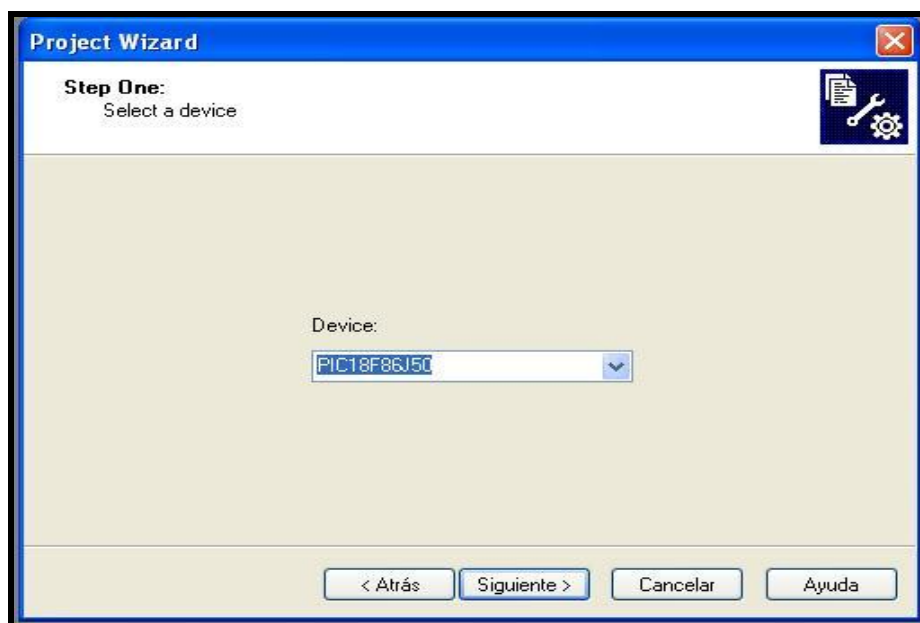


Image 34. PIC selection

2. Then select the assembly tool: MPASM.

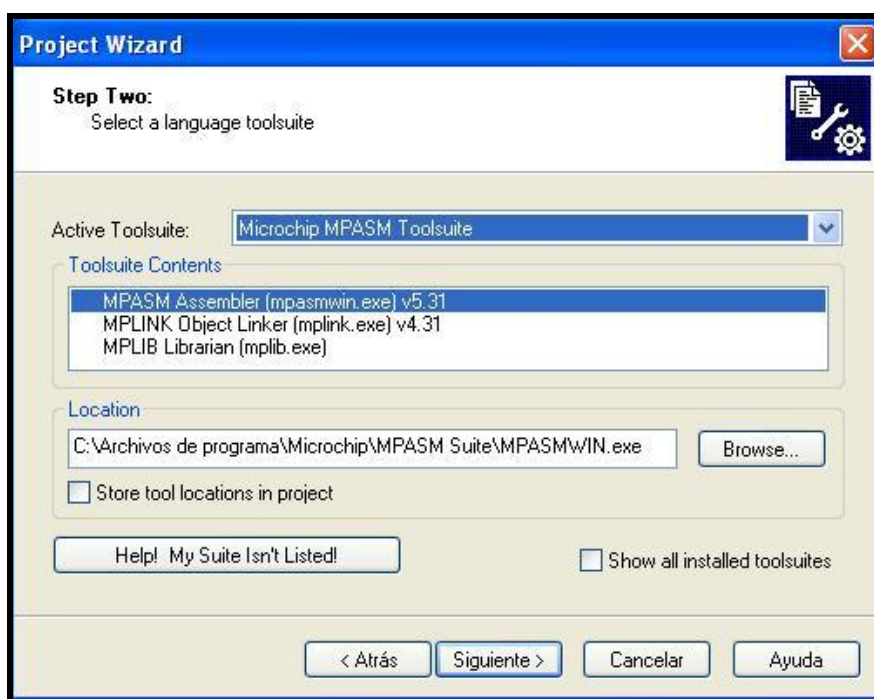


Image 35. Tool selection

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

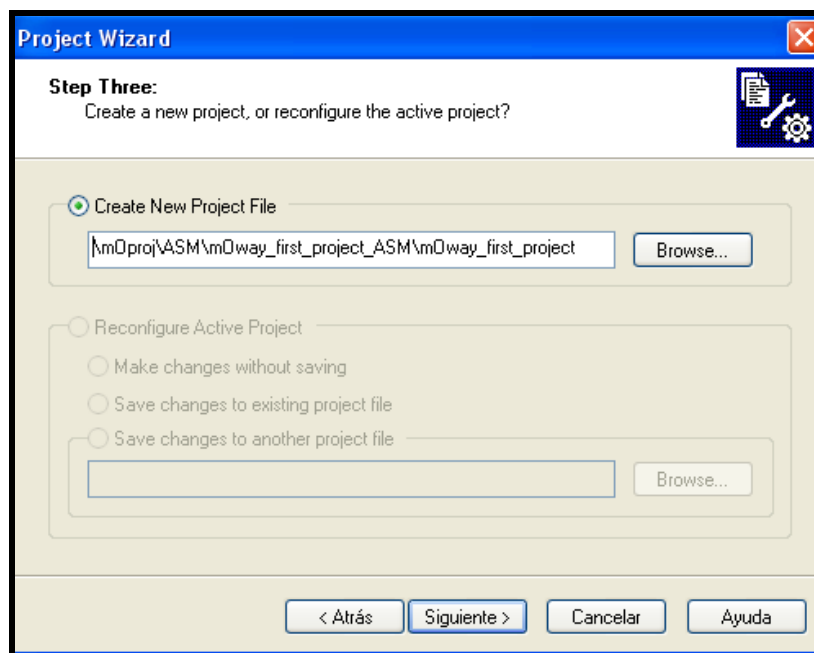


Image 36. Select name and folder

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

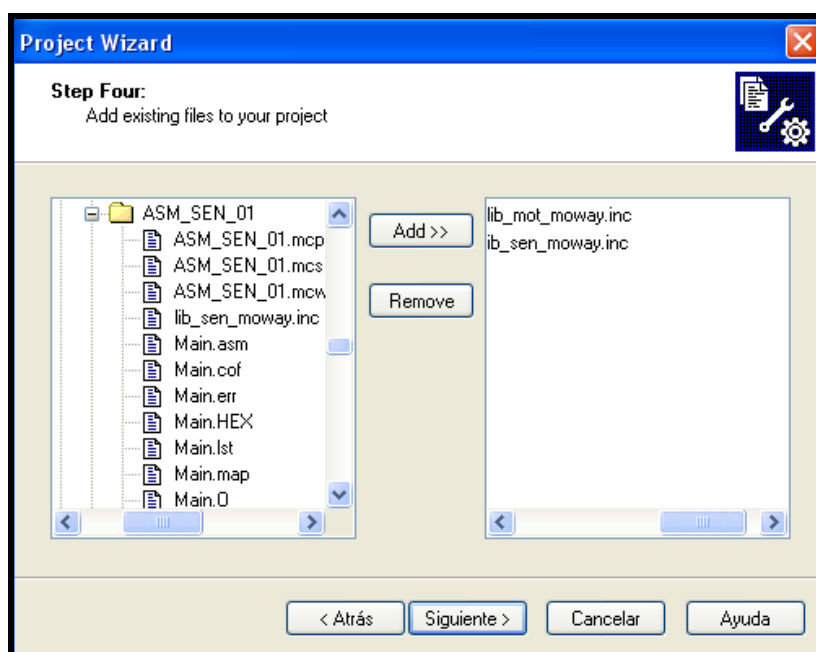


Imagen 37. Select mOway libraries

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 38. 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.

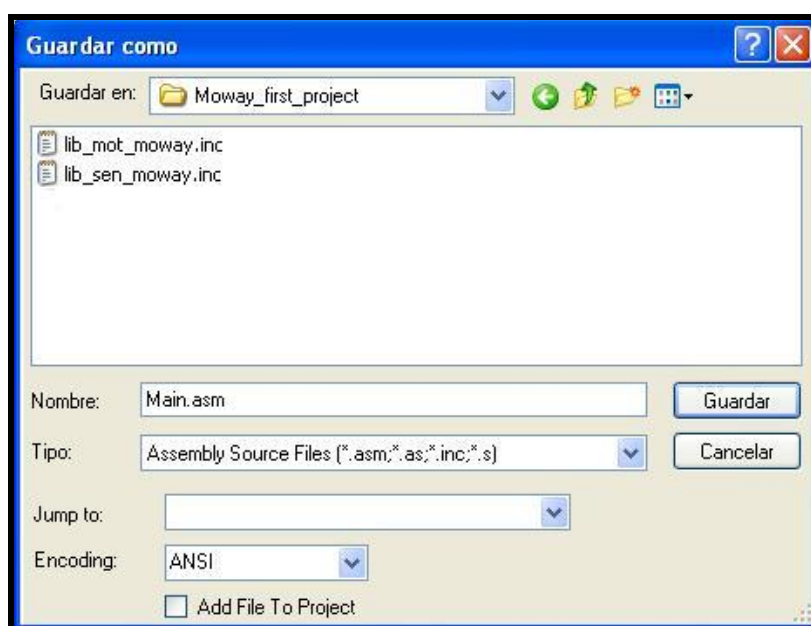


Image 39. .ASM creation

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

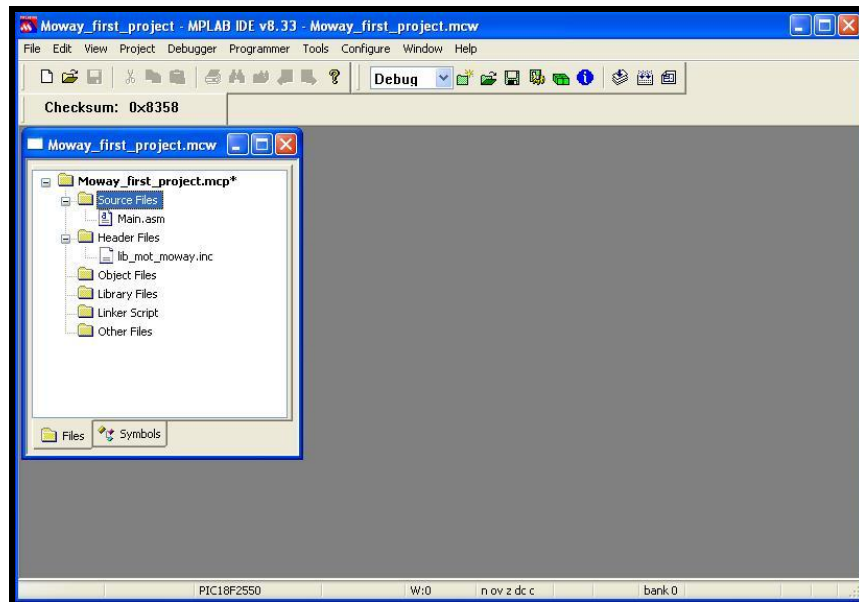


Image 40. Project with .ASM

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 MowayWorld 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 41. 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 42. 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 43. Configuration and first movement

15. When it encounters an obstacle a command is sent to rotate 180° 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 44. . 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 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_CHECK_ACCE

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

Table 28. 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 29. 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 30. 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 °C. 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 frequency, 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 31. 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 °C.
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 32. 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 signal for the SPI communication(accelerometer)
RD6		O	Clock signal 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	Temperature sensor
RH6		I	Battery measurer
RH7		I	Microphone
PORTJ			
RJ6		O	Infrared transmitter
RJ7		I/O	Free pad

SEN_LIGHT

<i>Output variables</i>	
SEN_LIGHT_P	Percentage of ambient light.

The SEN_LIGHT function captures the analog value generated by the incident 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

<i>Output variables</i>	
SEN_BATTERY_P	Percentage of battery level.

The SEN_BATTERY function captures the analog value of the battery¹⁶. 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

<i>Output variables</i>	
SEN_TEMPERATURE_C	Temperature in °C.

The SEN_TEMPERATURE function captures the analog value that depends on the temperature captured by the thermistor¹⁷. 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

<i>Output variables</i>	
SEN_MIC	Sound Intensity.

¹⁶ The output value can differ from mOwayGUI.

¹⁷ 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

<i>Output variables</i>	
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

<i>Input variables</i>	
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 33. Allowed values for SEN_SPEAKER_ON_OFF

Define	Value
SPEAKER_OFF	0
SPEAKER_ON	1
SPEAKER_TIME	2

Table 34. 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

<i>Input variable</i>	
SEN_CHECK_OBS	Which sensor must be read
<i>Output variable</i>	
SEN_OBS	Indicates if there is obstacle or not.
<i>Output</i>	
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 35. 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

<i>Input variable</i>	
SEN_CHECK_OBS	Which sensor must be read
<i>Output variable</i>	
SEN_OBS	Indicates if there is obstacle or not.
<i>Output</i>	
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.

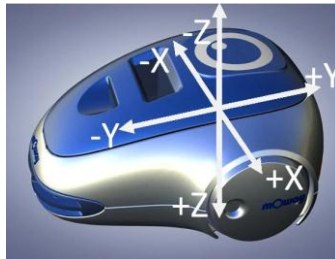
Table 36. SEN_CHECK_OBS allowed values

Define	Value
OBS_CENTER_L	0
OBS_SIDE_L	1
OBS_CENTER_R	2
OBS_SIDE_R	3

SEN_ACCE_XYZ_READ

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

SEN_ACCE_XYZ_READ returns the acceleration of the robot in the 3 axes. Resolution is $\pm 0.0156\text{G/bit}$. Value 0 is -2G and 255 is 2G.


Image 45. 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 37. SEN_CHECK_ACCE allowed values.

Define	Value
ACCE_CHECK_X	0
ACCE_CHECK_Y	1
ACCE_CHECK_Z	2

SEN_ACCE_CHECK_TAP

<i>Output variable</i>	
SEN_ACCE_TAP	Detects taps
<i>Output</i>	
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

<i>Input variable</i>	
SEN_CHECK_LINE	Which sensor must be read
<i>Output variable</i>	
SEN_LINE	Digital value of the sensor
<i>Output</i>	
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 38. SEN_CHECK_LINE allowed value

Define	Value
LINE_L	0
LINE_R	1

SEN_LINE_ANALOG

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

The SEN_LINE_ANALOG function indicates the light reflected in the optocouplers¹⁸. 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 39. 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.

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

LED_TOP_RED_ON

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.

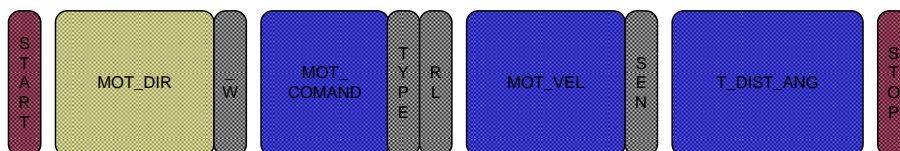


Image 46. Command format: MOT_STR, MOT_CHA_VEL

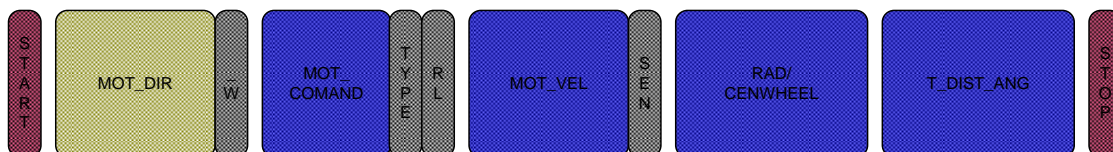


Image 47. Command format: MOT_CUR, MOT_ROT

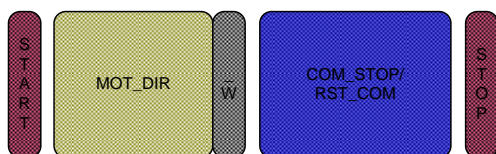


Image 48. Command format: MOT_STOP, MOT_RST

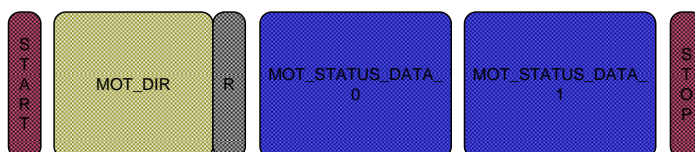


Image 49. Command format: MOT_FDBCK

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 40. Summary of assembly language functions for lib_mot_moway

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

	<i>MOT_CON</i> FWDBACK COMTYPE RL		
MOT_CUR	MOT_VEL MOT_RAD MOT_T_DIST <i>MOT_CON</i> FWDBACK COMTYPE RL	<i>MOT_STATUS</i> COMOK DWRONG	A command to execute a curve
MOT_STOP	-	<i>MOT_STATUS</i> COMOK DWRONG	A command to stop the robot
MOT_RST	MOT_RST_COM	<i>MOT_STATUS</i> 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 <i>MOT_STATUS</i> 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 41. 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

<i>Input</i>			
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	0	255
	Distance value	0	255
<i>Output variables</i>			
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

<i>Input</i>			
MOT_VEL	Desired speed	0	100
MOT_CON, FWDBACK	Movement direction	1-FWD	0-BACK



MOWAY

Title: mOway User Manual
Rev: v3.1.2 – April 2013
Page 66 of 175

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	0	255
	Distance value	0	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	0	255
	Angle value	0	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

$$MOT_T_ANG = round\left(\frac{Angle^{\circ} \times 3.33}{12^{\circ}}\right)$$

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 180°

```
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

<i>Input</i>			
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	0	255
	Distance value	0	255
<i>Output variables</i>			
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

<i>Input</i>		
MOT_RST_COM	The parameter that needs to be reset	RST_T RST_DIST RST_KM
<i>Output variables</i>		
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

<i>Input</i>		
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
<i>Output variables</i>		
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 42. Parameter resolution

Parameter	Resolution
STATUS_T	100ms/bit
STATUS_A	3.6°/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: **SNOK:** 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 43. 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 44. 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 turn, 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 45. 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.

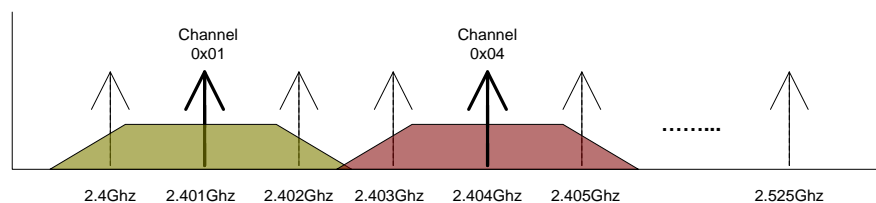


Image 50. RF channels

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

<i>Output variables</i>
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

<i>Output variables</i>	
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 be 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                                     ;Proces 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

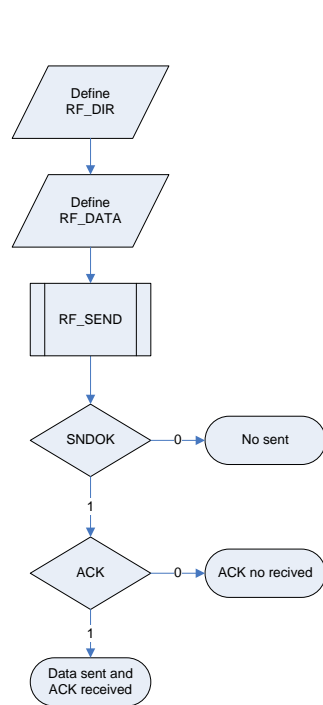


Diagram 1. Sent data in assembler

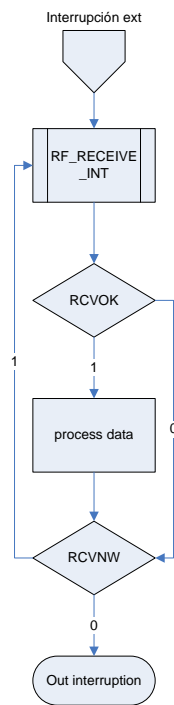


Diagram 2. Receive interruption in assembler

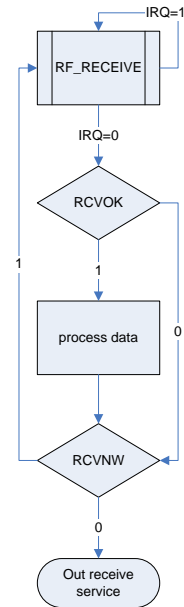


Diagram 3. Reception in assembler

5.3.4. mOway Camera Board library in assembly language

5.3.4.1. Description

This library allows to activate mOway's camera and to choose the channel to transmit the images to video capturer. Camera is controlled through SPI commands.

The camera module has been designed to make possible to use it with other modules at a time, like RF module BZI-RF2GH4. Both modules share the microcontroller SPI port. When CS pin is high, the camera module receives SPI commands while RF module is disabled. When CS pin is low, the RF module receives SPI commands while camera module is disabled.

5.3.4.2. Variables

These variables are 1 byte length.

COMMAND_CAM

This variable contains the type of the command (reading or writing) to send to the camera controller. Functions included in this library only need writing commands.

ADDRESS_CAM

This variable contains the direction of the register to read/write.

DATA_CAM

This variable contains data to send to the register defined by **ADDRESS_CAM**.

CAM_STAT

This variable indicates the camera status. If camera is activated, the value is '1'. Otherwise, the value is '0'.

CAM_CHANNEL

This variable indicates the channel in which camera transmits the images.

5.3.4.3. Functions

This library includes functions to manage camera module. A brief description of each one of these functions is given below.

Table 46. Assembler camera control functions.

Functions for the camera module	
CAM_CONFIG	Configures microcontroller SPI inputs/outputs and configures camera board controller.
CAM_ON	Activates camera
CAM_OFF	Deactivates camera
CAM_CHN_SEL	Selects the transmission channel from camera to camera capturer.
CAM_SEND_COM	Sends a command to camera controller.
CAM_SPI_WRITE	Sends a byte through SPI.

CAM_CONFIG

This function configures the parameters of SPI communication and microcontroller SPI port. It also configures input/output port of camera controller.

PIN SPI	PIN PIC
SCK	RC3
SDI	RC5
SDO	RC4
IRQ	RB0
CSN	RF2
CE	RH4

Table 47. Microcontroller SPI port configuration

CAM_SEND_COM

This function sends a command through SPI to the camera controller. The format consists of 3 bytes: command type (read/write), register and data to write in that register.

COMANDOS	
COMMAND_CAM	COM_WR to write in register COM_RD to read register
ADDRESS_CAM	IODIR_ADD to configure input/output port of camera controller. OLAT_ADD to change the level (low/high) of camera controller port.
DATA_CAM	Data to write on the register in ADDRESS_CAM

CAM_ON

This function sends a command to activate the camera. It is a writing type command to the camera controller port. The 3 bytes to send are:

BYTES	
COMMAND_CAM	COM_WR
ADDRESS_CAM	OLAT_ADD
DATA_CAM	0x22

CAM_OFF

This function sends a command to deactivate the camera. It is a writing type command to the camera controller port. The 3 bytes to send are:

BYTES	
COMMAND_CAM	COM_WR
ADDRESS_CAM	OLAT_ADD
DATA_CAM	0x00

CAM_CHN_SEL

This function sends a command to set camera transmission channel. It is a writing type command to the camera controller port.

Input variables	
CAM_CHANNEL	Camera transition channel. It must be a value from 1 to 4.

BYTES	
COMMAND_CAM	COM_WR
ADDRESS_CAM	IODIR_ADD
DATA_CAM	Channel 1: 0xD8 Channel 2: 0xD4 Channel 3: 0xCC Channel 4: 0xDC

Camera activation example:

; Camera configuration

call CAM_CONFIG

; Channel 1 selection

movlw .1

movwf CAM_CHANNEL

call CAM_CHN_SEL

; Camera activation

call CAM_ON

CAM_SPI_WRITE

This functions first sends one byte to SPI port of the microcontroller, which sends it out. Then, it reads the SPI port in case it is needed. This library only needs to send SPI commands.

<i>Input variables</i>	
BYTE_OUT	Byte to send to SPI
<i>Output variables</i>	
BYTE_IN	Byte read from SPI

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.

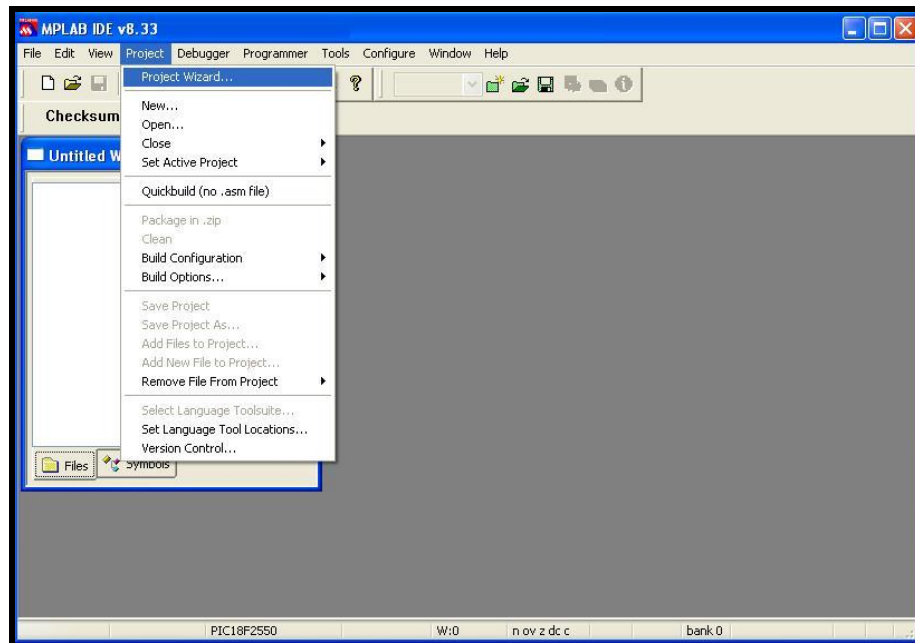


Image 51. Project Wizard

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

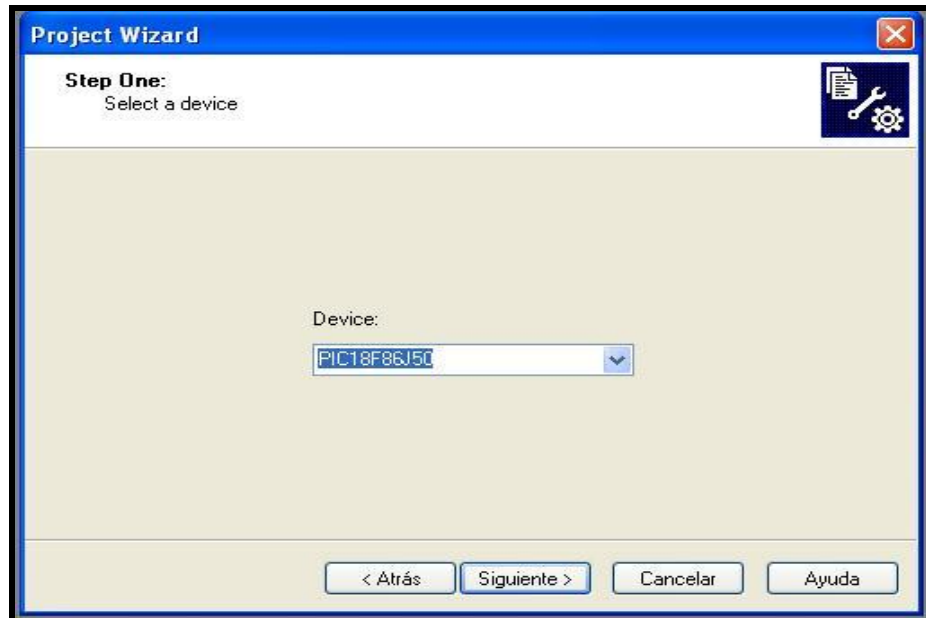


Image 52. PIC selection

2. Choose C18 C compiler.

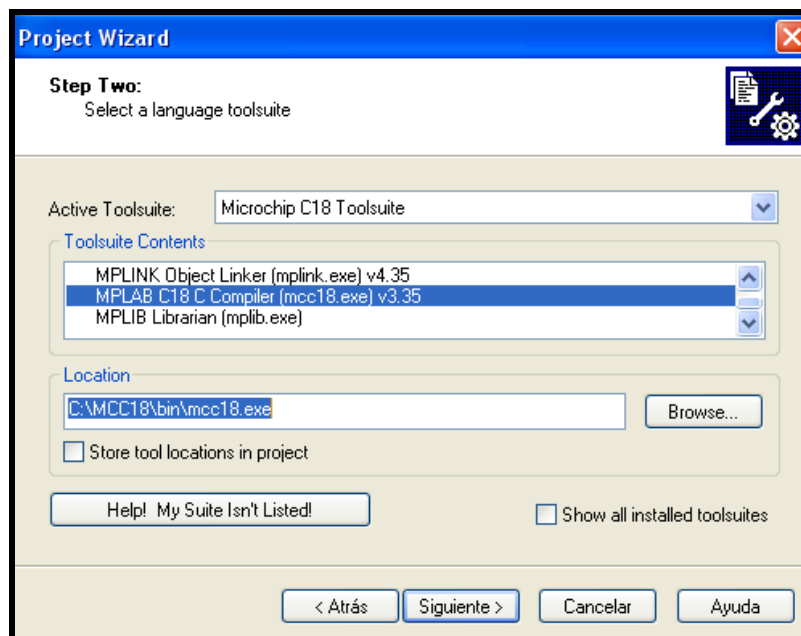


Image 53. Tool selection

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

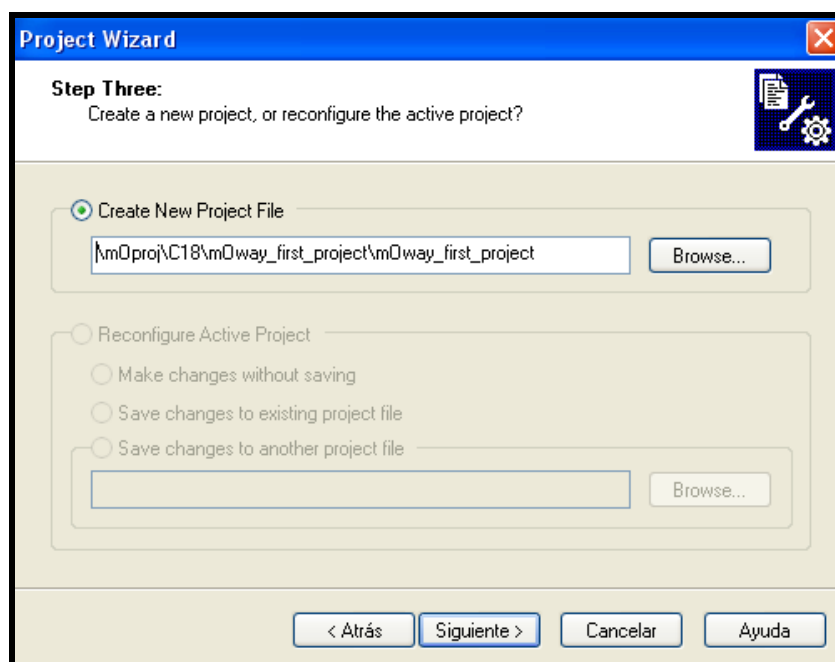


Image 54. Location

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

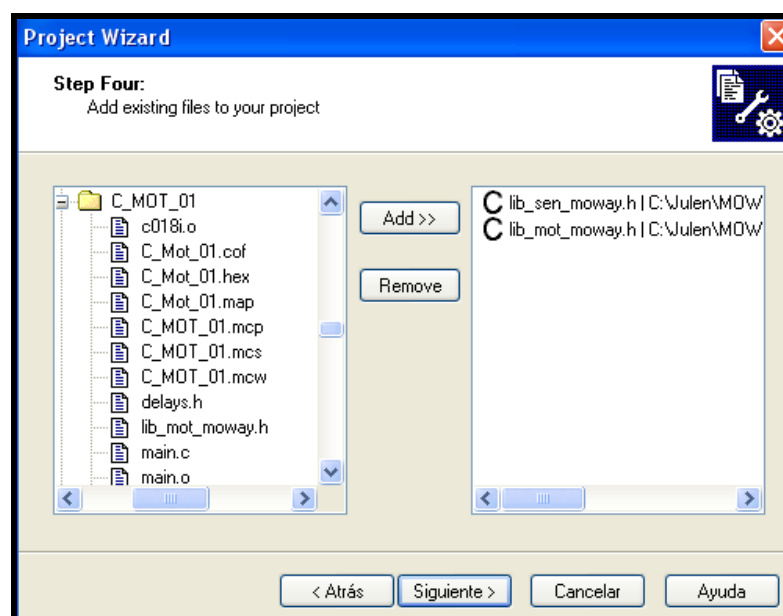


Image 55. Add libraries

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

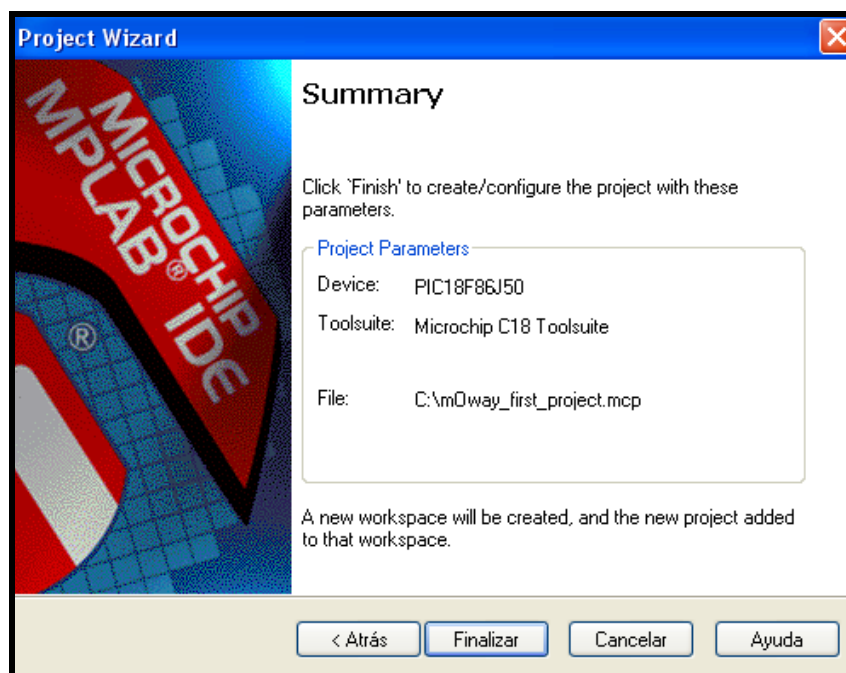


Image 56. Wizard ends

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.

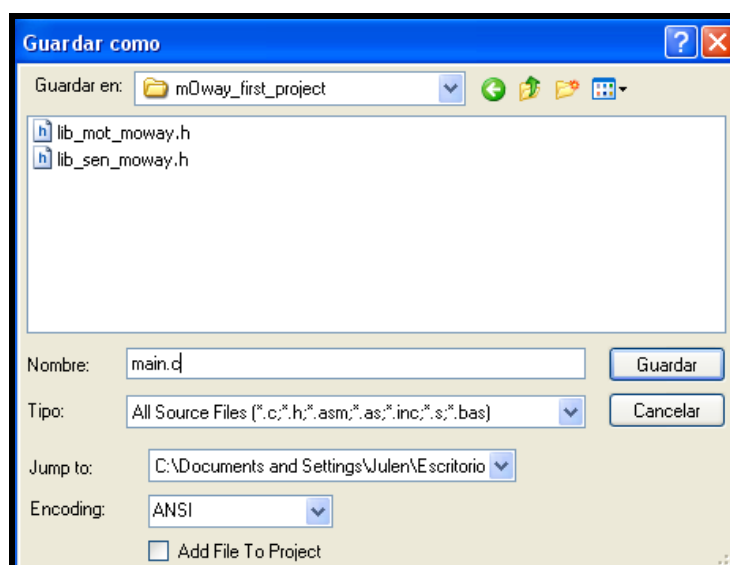


Image 57. .C creation

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.

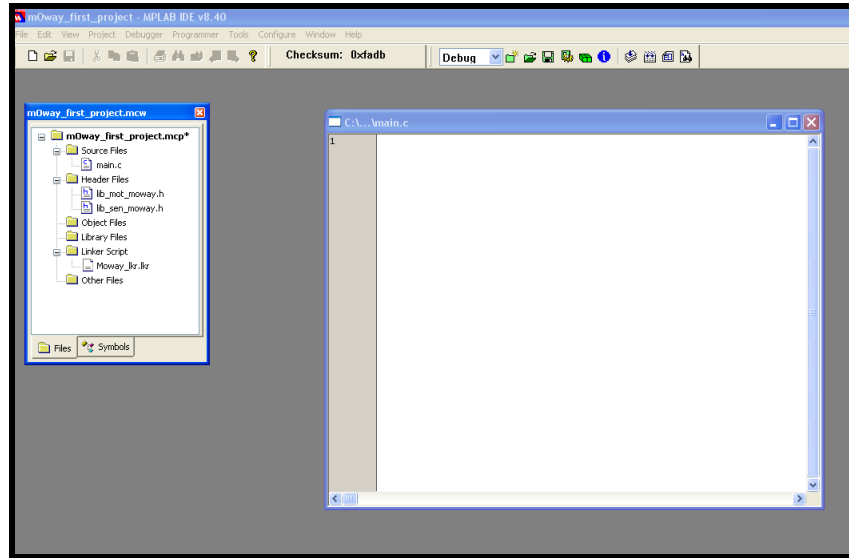


Image 58. Project with .C

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 address for the correct bootload
#define REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS 0x1008 //High priority interruption adress for t
#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 pri

//*****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 59. 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 MowayWorld 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 60. 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 61. 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 180° 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 62. 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 48. C function summary

Name	Input constants	Description
<i>void</i> SEN_CONFIGURAR(<i>void</i>)	-	Configured to use the sensors.
<i>unsigned char</i> SEN_LIGHT(<i>void</i>)	-	Reads light sensor values.
<i>unsigned char</i> SEN_BATTERY(<i>void</i>)	-	Returns the battery level.
<i>unsigned char</i> SEN_TEMPERATURE (<i>void</i>)	-	Detects the temperature in °C.
<i>unsigned char</i> SEN_MIC_ANALOG (<i>void</i>)	-	Detects sound intensity.
<i>unsigned char</i> SEN_MIC_DIG (<i>void</i>)	-	Detects if there is sound or not.
<i>unsigned char</i> SEN_SPEAKER(<i>unsigned char</i> , <i>unsigned char</i> , <i>unsigned char</i>)	SPEAKER_OFF SPEAKER_ON SPEAKER_TIME	Emits tones in a frequency between 250 Hz and 65 KHz.
<i>unsigned char</i> SEN_ACCE_XYZ_READ(<i>unsigned char</i>)	ACCE_CHECK_X ACCE_CHECK_Y ACCE_CHECK_Z	Calculates the X,Y,Z axes acceleration of mOway.
<i>unsigned char</i> SEN_ACCE_CHECK_TAP(<i>void</i>)	-	Detects if mOway has been taped.
<i>unsigned char</i> SEN_OBS_DIG(<i>unsigned char</i>)	OBS_SIDE_L OBS_CENTER_L OBS_CENTER_R OBS_SIDE_R	Detects obstacles
<i>unsigned char</i> SEN_OBS_ANALOG(<i>unsigned char</i>)	OBS_SIDE_L OBS_CENTER_L OBS_CENTER_R OBS_SIDE_R	Detects the distance to obstacles
<i>unsigned char</i> SEN_LINE_DIG(<i>unsigned char</i>)	LINE_R LINE_L	Detects dark zones (black lines)
<i>unsigned char</i> SEN_LINE_ANALOG (<i>unsigned char</i>)	LINE_R LINE_L	Detects surface colors
<i>void</i> LED_FRONT_ON(<i>void</i>)	-	Front LED on
<i>void</i> LED_BRAKE_ON(<i>void</i>)	-	Brake LED on
<i>void</i> LED_TOP_RED_ON(<i>void</i>)	-	Top red LED on
<i>void</i> LED_TOP_GREEN_ON(<i>void</i>)	-	Top green LED on
<i>void</i> LED_FRONT_OFF(<i>void</i>)	-	Front LED off
<i>void</i> LED_BRAKE_OFF(<i>void</i>)	-	Brake LED off
<i>void</i> LED_TOP_RED_OFF(<i>void</i>)	-	Top red LED off
<i>void</i> LED_TOP_GREEN_OFF(<i>void</i>)	-	Top green LED off
<i>void</i> LED_FRONT_ON_OFF(<i>void</i>)	-	Front LED blink
<i>void</i> LED_BRAKE_ON_OFF(<i>void</i>)	-	Brake LED blink
<i>void</i> LED_TOP_RED_ON_OFF(<i>void</i>)	-	Top red LED blink
<i>void</i> LED_TOP_GREEN_ON_OFF(<i>void</i>)	-	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 49. 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 signal for the SPI communication(accelerometer)
RD6	O	Clock signal 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	Temperature sensor
RH6	I	Battery measurer
RH7	I	Microphone
PORTJ		
RJ6	O	Infrared transmitter
RJ7	I/O	Free pad

unsigned char SEN_LIGHT(void)

<i>Output</i>
Percentage of ambient light.

The SEN_LIGHT function captures the analog value generated by the incident 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 incident 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 ¹⁹. 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 °C.

The SEN_TEMPERATURE function captures the analog value that depends on the temperature captured by the thermistor²⁰. 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.

¹⁹ The output value can differ from mOwayGUI

²⁰ Sensor measures mOway's temperature which can be different from ambient temperature.

- Returns amplified microphone value.

unsigned char SEN_MIC_DIG(void)

<i>Output</i>
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)

<i>Input variables</i>	
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 50. Allowed values for SEN_SPEAKER_ON_OFF

Define	Valor
SPEAKER_OFF	0
SPEAKER_ON	1
SPEAKER_TIME	2

Table 51. 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)

<i>Input variables</i>	
SEN_CHECK_OBS	Which sensor must be read
<i>Output</i>	
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 52. 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)

<i>Input variable</i>	
SEN_CHECK_OBS	Which sensor must be read
<i>Output</i>	
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 53. 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)

<i>Input variable</i>	
SEN_CHECK_ACCE	Which axis must be read
<i>Output</i>	

Acceleration value

SEN_ACCE_XYZ_READ returns the acceleration of the robot in the 3 axes. Resolution is $\pm 0.0156\text{G/bit}$. Value 0 is -2G and 255 is 2G.

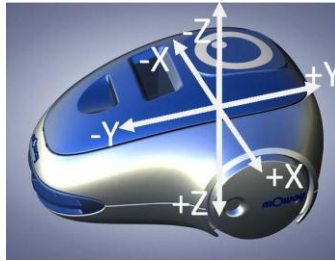


Image 63. mOway axes

- 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 54. 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)

<i>Output</i>
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)

<i>Input variable</i>	
SEN_CHECK_LINE	Which sensor must be read
<i>Output</i>	
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 55. SEN_CHECK_LINE allowed values

Define	Value
LINE_L	0
LINE_R	1

unsigned char SEN_LINE_ANALOG(unsigned char SEN_CHECK_LINE)

<i>Input variables</i>	
SEN_CHECK_LINE	Which sensor must be read
<i>Output</i>	
Analog value of the sensor	

The SEN_LINE_ANALOG function indicates the light reflected in the optocouplers²¹. 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 56. SEN_CHECK_LINE allowed values

Define	Value
LINE_L	0
LINE_R	1

void LED_BRAKE_ON(void)

²¹ Due to tolerance two different sensors can differ from each other.

Function to switches on the brake LED.

void LED_FRONT_ON(void)

Function to switches on the front LED.

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.

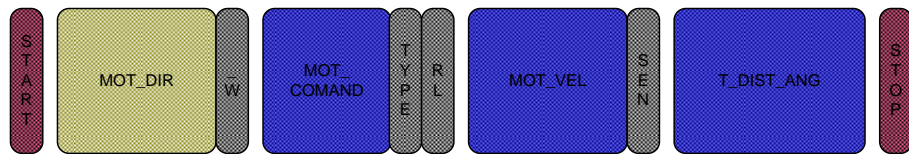


Image 64. Command format: MOT_STR, MOT_CHA_VEL

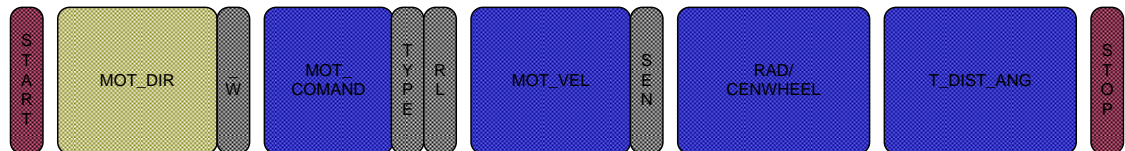


Image 65. Command format: MOT_CUR, MOT_ROT

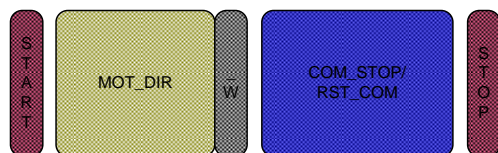


Image 66. Command format: MOT_STOP, MOT_RST



Image 67. 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 57. Summary of functions in C for lib_mot_moway

Name	Input	Return	Description
<i>void</i> MOT_CONFIG(void)	-	-	Configuration to communicate with the motors
<i>unsigned char</i> MOT_STR(<i>unsigned char, unsigned char, unsigned char, unsigned char</i>)	MOT_VEL FWDBACK COMTYPE MOT_T_DIST	0: Correct dispatch 1: Incorrect dispatch 2: Incorrect data	A command to move in a straight line
<i>unsigned char</i> MOT_CHA_VEL(<i>unsigned char, unsigned char, unsigned char, unsigned char, unsigned char</i>)	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
<i>unsigned char</i> MOT_ROT(<i>unsigned char, unsigned char, unsigned char, unsigned char, unsigned char, unsigned char</i>)	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
<i>unsigned char</i> MOT_CUR(<i>unsigned char, unsigned char, unsigned char, unsigned char, unsigned char, unsigned char</i>)	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
<i>unsigned char</i> MOT_STOP(void)		0: Correct dispatch 1: Incorrect dispatch	A command to stop the robot
<i>unsigned char</i> MOT_RST(<i>unsigned char</i>)	RST_COM	0: Correct dispatch 1: Incorrect dispatch	A command to reset the temporary variables for time and distance
<i>unsigned char*</i> 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 58. 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)

<i>Input</i>			
MOT_VEL	Desired speed	0	100
FWDBACK	Movement direction	FWD	BACK
COMTYPE	Type of command	TIME	DISTANCE
MOT_T_DIST	Time value	0	255
	Distance value	0	255
<i>Function Return</i>			
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)

<i>Input</i>			
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	0	255
	Distance value	0	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	0	255
	Angle value	0	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

$$MOT_T_ANG = round\left(\frac{Angle^{\circ} \times 3.33}{12^{\circ}}\right)$$

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 180°

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)

<i>Input</i>			
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	0	255
	Distance value	0	255
<i>Function Return</i>			
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

$$0 \leq MOT_VEL - MOT_RAD \leq 100$$

Equation 6. Condition 2 MOT_RAD
 $0 \leq MOT_VEL + MOT_RAD \leq 100$

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)

<i>Function Return</i>	
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)

<i>Input</i>		
RST_COM	The parameter that to be reset	RST_T RST_DIST RST_KM
<i>Function Return</i>		
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)**

<i>Input</i>		
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
<i>Output</i>		
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 59. Parameter resolution

Parameters	Resolution
STATUS_T	100ms/bit
STATUS_A	3.6°/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)[0];
```

```
mOway_km[1] = MOT_FDBCK(STATUS_KM)[1];
```

```
//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 60. Summary of functions in C18.

Name	Input	Return	Description
<i>void RF_CONFIG_SPI(void)</i>	-	-	Configures the inputs and outputs of the microcontroller as well as the parameters necessary to use the SPI bus.
<i>void RF_INT_EN(void)</i>	-	-	This routine enables the external interruption for the radio module in the microcontroller.

<i>unsigned char RF_CONFIG(unsigned char , unsigned char)</i>	CHANNEL ADDRESS	1: Correct configuration 0: Not configured	Configures the inputs and outputs of the microcontroller as well as the radio module parameters.
<i>unsigned char RF_ON(void)</i>	-	0: Correct activation 1: Incorrect activation	Activates the radio frequency module in watch mode.
<i>unsigned char RF_OFF(void)</i>	-	0: Correct deactivation 1: Incorrect deactivation	Deactivates the radio frequency module and leaves it in low consumption mode.
<i>unsigned char RF_SEND(unsigned char, unsigned char)</i>	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.
<i>unsigned char RF_RECEIVE(unsigned char*, unsigned char*)</i>	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 61. 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)

<i>Input variables</i>	
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).
<i>Function Return</i>	
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 62. 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.

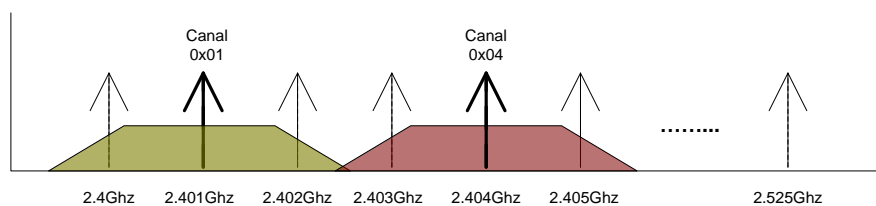


Image 68. RF channels

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	Output 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)

<i>Output variables</i>	
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.
<i>Function Return</i>	
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

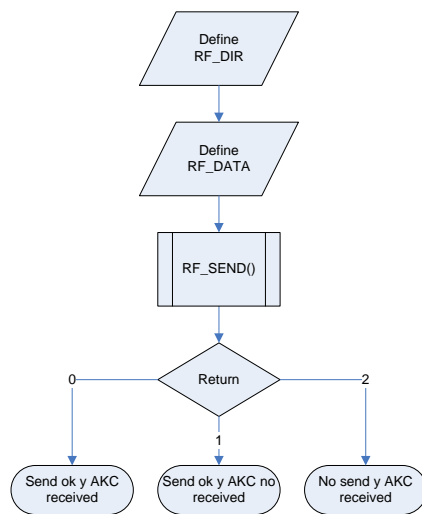


Diagram 4. Data send in C

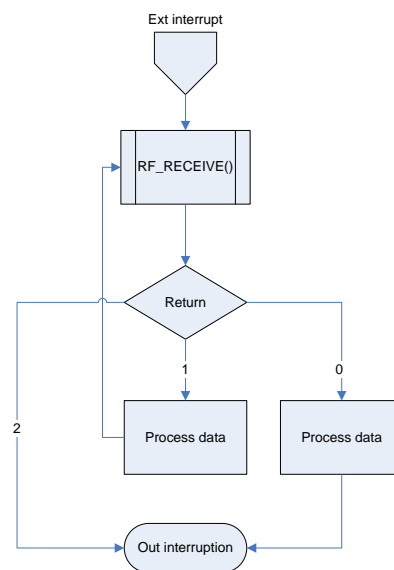
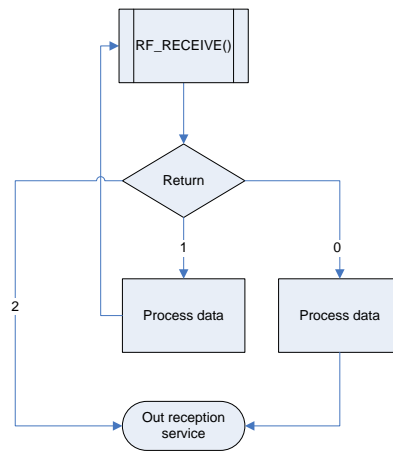


Diagram 5. Reception interruption in C

**Diagram 6. Reception in C**

6.3.4. *mOway Camera Board library in C18*

6.3.4.1. *Description*

This library allows controlling camera module.

6.3.4.2. *Functions*

This library includes functions to manage camera module. A brief description of each one of these functions is given below.

Name	Input	Return	Description
<i>void CAM_CONFIG(void)</i>	-	-	Configures microcontroller SPI inputs/outputs and configures camera board controller.
<i>void CAM_SEND_COM (unsigned char, unsigned char, unsigned char)</i>	COMMAND ADDRESS DATA	-	Sends a command to camera controller.
<i>void CAM_ON (void)</i>	-	-	Activates camera
<i>void CAM_OFF (void)</i>	-	-	Deactivates camera
<i>void CAM_CHN_SEL (unsigned char)</i>	CHANNEL	-	Selects the transmission channel from camera to camera capturer.
<i>void CAM_SPI_WRITE (unsigned char)</i>	DATA	-	Sends a byte through SPI.

Table 63. C18 functions summary.

CAM_CONFIG

This function configures the parameters of SPI communication and microcontroller SPI port. It also configures input/output port of camera controller.

PIN SPI	PIN PIC
SCK	RC3
SDI	RC5
SDO	RC4
IRQ	RB0
CSN	RF2
CE	RH4

Table 64. Microcontroller SPI port configuration

CAM_SEND_COM

This function sends a command through SPI to the camera controller. The format consists of 3 bytes: command type (read/write), register and data to write in that register.

COMMANDS	
COMMAND_CAM	COM_WR to write in register COM_RD to read register
ADDRESS_CAM	IODIR_ADD to configure input/output port of camera controller. OLAT_ADD to change the level (low/high) of camera controller port.
DATA_CAM	Data to write on the register in ADDRESS_CAM

CAM_ON

This function sends a command to activate the camera. It is a writing type command to the camera controller port. The 3 bytes to send are:

BYTES	
COMMAND_CAM	COM_WR
ADDRESS_CAM	OLAT_ADD
DATA_CAM	0x22

CAM_OFF

This function sends a command to deactivate the camera. It is a writing type command to the camera controller port. The 3 bytes to send are:

BYTES	
COMMAND_CAM	COM_WR
ADDRESS_CAM	OLAT_ADD
DATA_CAM	0x00

CAM_CHN_SEL

This function sends a command to set camera transmission channel. It is a writing type command to the camera controller port.

Input variables	
CAM_CHANNEL	Camera transmission channel. It must be a value from 1 to 4.



<i>BYTES</i>	
COMMAND_CAM	COM_WR
ADDRESS_CAM	IODIR_ADD
DATA_CAM	Channel 1: 0xD8 Channel 2: 0xD4 Channel 3: 0xCC Channel 4: 0xDC

Camera activation example:

```
// Camera configuration  
CAM_CONFIG();
```

```
// Channel 1 selection  
CAM_CHN_SEL(1);
```

```
// Camera activation  
CAM_ON();
```

CAM_SPI_WRITE

This function sends a byte from microcontroller through SPI.

<i>Input variables</i>	
DATA	Byte to send through SPI

7. MowayWorld programming

MowayWorld application makes it possible to design programs by means of flowchart diagrams, so that mOway robot can be programmed very easily. Different modules represent functions that control robot's sensors and actuators. These modules are joined by means of arrows to define the program flow. Previous programming knowledge is not essential to use flowchart programming.

7.1. *MowayWorld workspace*

The following lines describe the different parts of MowayWorld workspace.

7.1.1. *Toolbar*

Toolbar makes it possible to manage the project, edit the flowchart, create variables, program mOway robot, change MowayWorld language and many other functions.

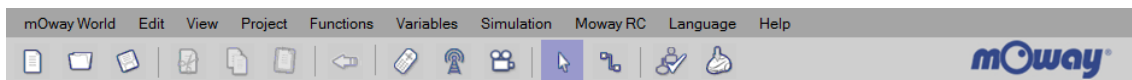


Image 69. Toolbar

7.1.2. *Flowchart Editor*

The Flowchart Editor window is where modules are placed and connected to develop the program. When a new project is created, this window is empty except for the starting point. In flow diagrams there is always a “starting point” from which the program begins.

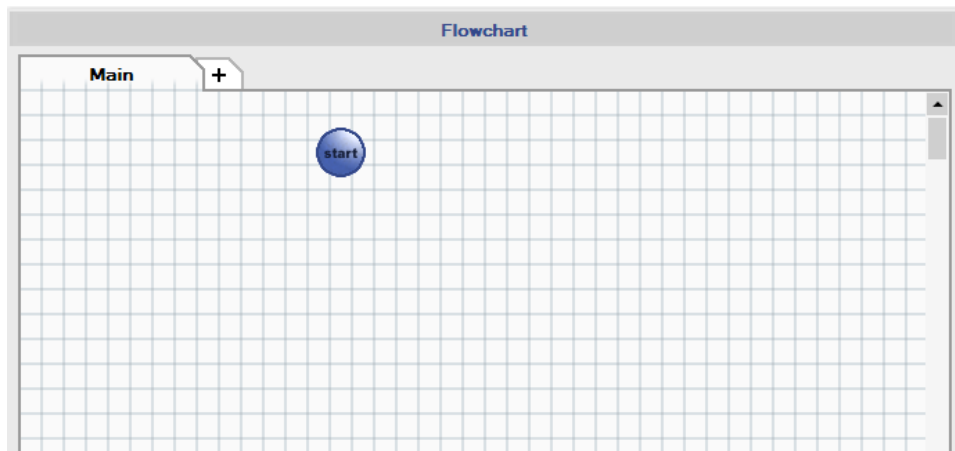


Image 70. Flowchart Editor window

7.1.3. Tools

On the left side of the workspace we can find the Tools section. Here there are all the modules to control mOway, such as movement actions, checking sensors, communication, and so on. Modules are grouped by their type of function (Actions, Sensors, Data, Flowchart Control and Expansion connector).

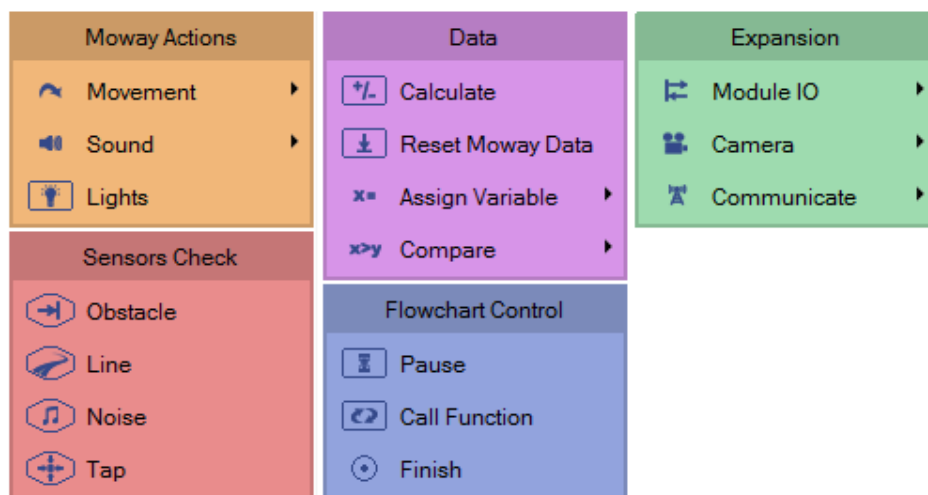
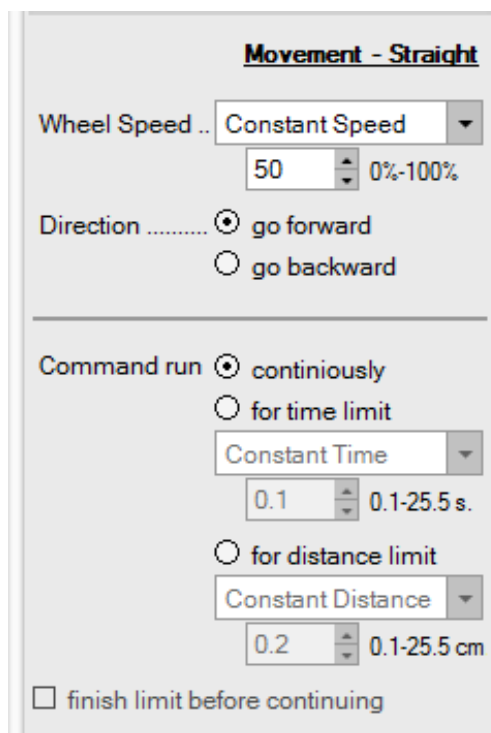


Image 71. Toolbox section

A module is inserted by just dragging and dropping it in the flowchart editor. This module function can be configured by making double click on it, or in the “Properties” window (see next chapter). Some modules are grouped by a more specific type (Movement, Sound,...). This is represented by a little black arrow in the toolbox module types.

7.1.4. Properties

When a module in Flowchart Editor is selected, the “Properties” window will appear on the right side of the workspace. This makes it easier and faster to change the modules configuration.



Movement - Straight

Wheel Speed .. Constant Speed
 50 0%-100%

Direction ☒ go forward
☐ go backward

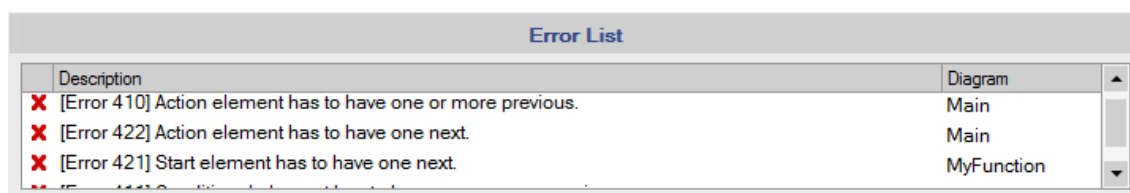
Command run ☒ continuously
☐ for time limit
 Constant Time
 0.1 0.1-25.5 s.
☐ for distance limit
 Constant Distance
 0.2 0.1-25.5 cm

☐ finish limit before continuing

Image 72. “Straight” module properties

7.1.5. Error List

Error List window shows the errors found in the diagram. It includes a description of the error (Description) and the diagram in which the error has produced (Diagram). A diagram with errors will not be downloaded to the robot.



Description	Diagram
✖ [Error 410] Action element has to have one or more previous.	Main
✖ [Error 422] Action element has to have one next.	Main
✖ [Error 421] Start element has to have one next.	MyFunction

Image 73. “Error List” window

7.1.6. Arrows

Modules execution order is established by means of arrows. In order to connect the modules, following steps must be taken:

- Place the cursor over a module until red and white marks appear.
- Click on one of these marks.
- Click on one of the next module mark.

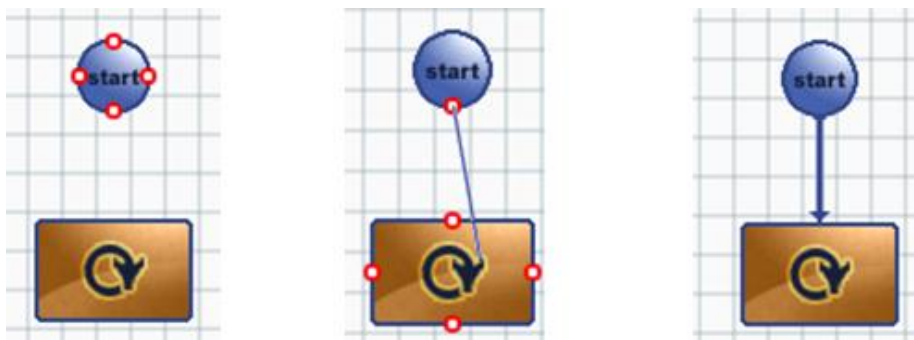


Image 74. Arrow drawing

“Connector” tool makes easier to draw arrows. It is located on Toolbar. For selecting diagram elements it is necessary to change to the default “Cursor” tool.



Image 75. Cursor tool and Connector tool icons

All the modules of the diagram must have at least one input arrow. In addition, they must have one output arrow (two output arrows in case it is a conditional module).

The conditional modules (oval shaped) have two outputs, depending on the condition. If the condition is true, the program will flow through “true” path (arrow with a green mark). If the condition is false, the program will flow through “false” path (arrow with a red cross).

In order to describe a loop, the procedure is similar but in this case the arrow starts on a module mark and finishes on another mark of the same block.

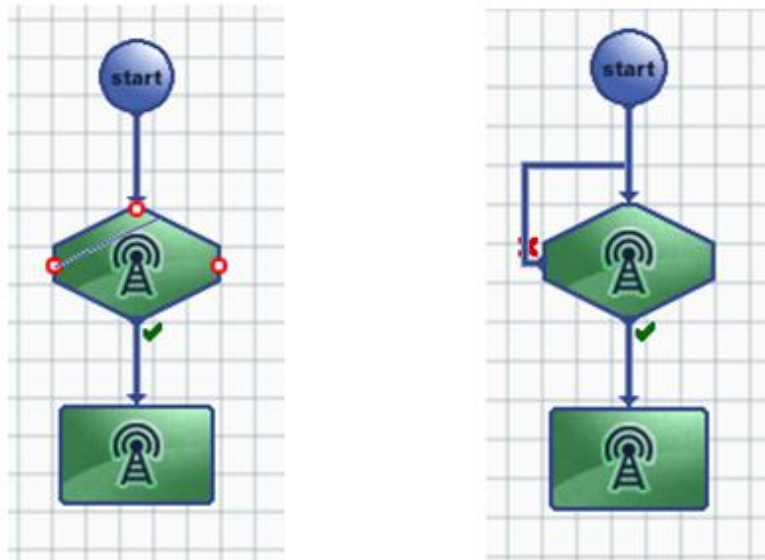


Image 76. Arrow drawing for a loop

The arrow track can be modified dragging the marks that appear when the cursor is over the arrow.

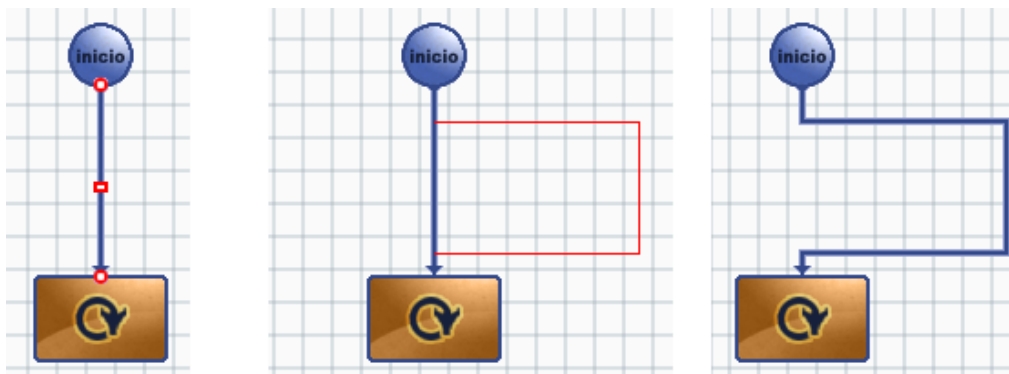


Image 77. Modifying arrow track

7.1.7. Language change and updates

The language of MowayWorld can be changed from toolbar “Language” tab. It is also possible to check if MowayWorld needs to be updated in “Help – Check for updates”.

7.2. *First program in MowayWorld*

This first basic program will make mOway avoid obstacles. Once MowayWorld is launched, save this project as “first_program”.

1. Once the project is saved, the program starts with a 2-second delay. Just drag and drop a “Pause” module and configure it with a constant value of 2 seconds.
2. The command to make green LED blink is added with the “Lights” module. Configuration of both modules are shown below.

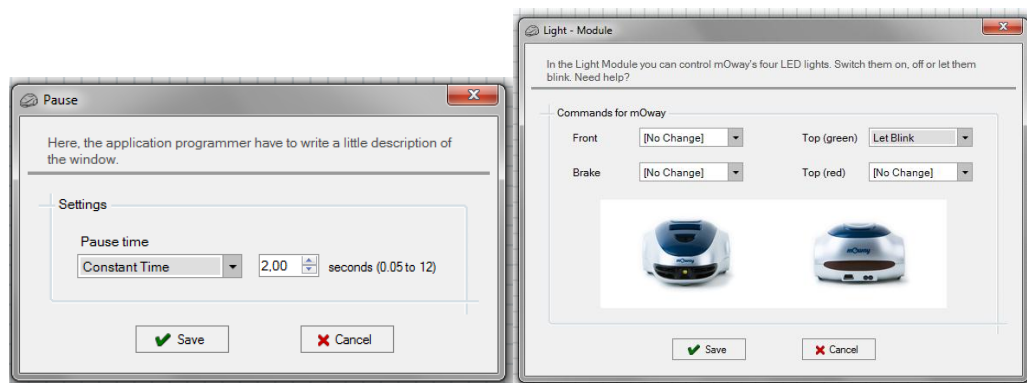


Image 78. Pause and LED configuration

3. The end of the program is added (“Finish” module) so that the application can be compiled.
4. The program is compiled and recorded into the robot clicking on “Program mOway” button in the toolbox, shown in the next image.
5. Test the program and check that after waiting 2 second the green LED blinks.

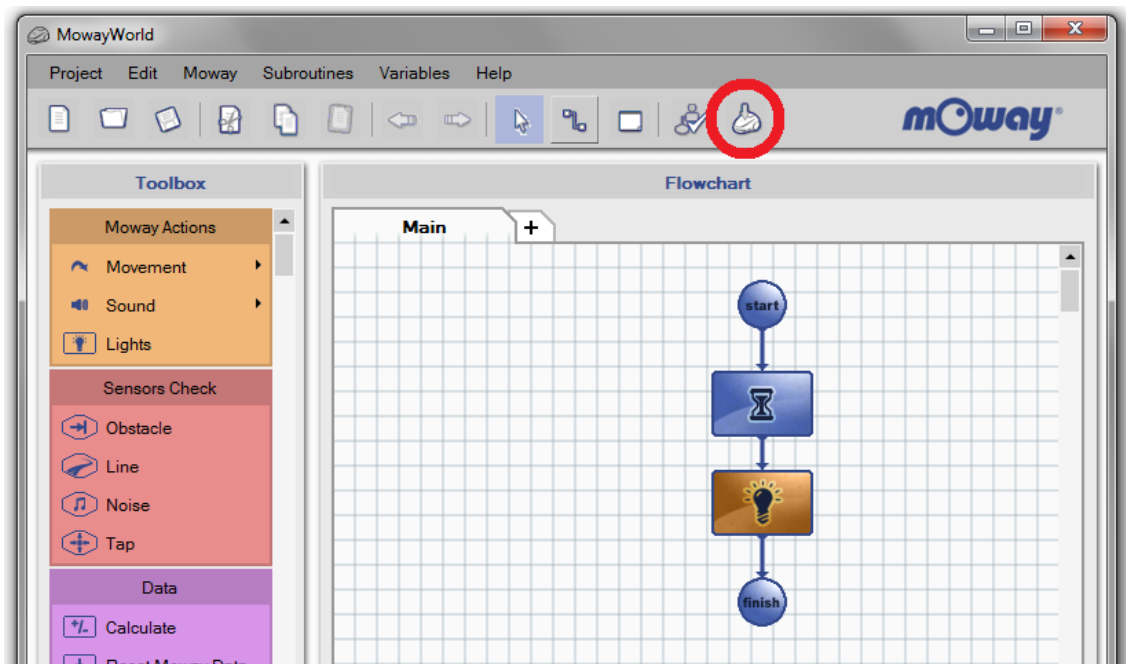


Image 79. Flowchart of the program and “Program mOway” button

6. In order to detect obstacles, four “Sensor Check Obstacle” has to be added and configured one for each obstacle sensor.

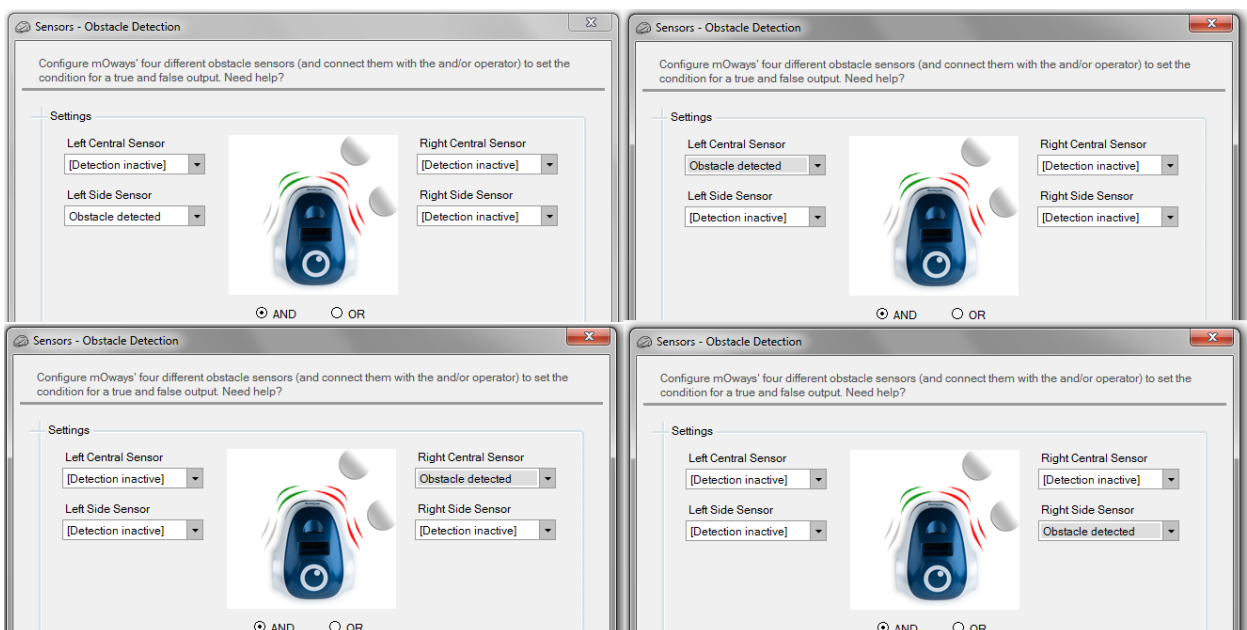


Image 80. Check Obstacles configuration

7. Condition modules have a true output and false output. When the obstacle is detected, the condition is true (green mark) and the corresponding LED turns on. If the obstacle is not detected, the condition is false (red mark) and the corresponding LED turns off. LED control is done with “Lights” modules.
8. Test the program and check that the front LEDs light up when an obstacle is detected.

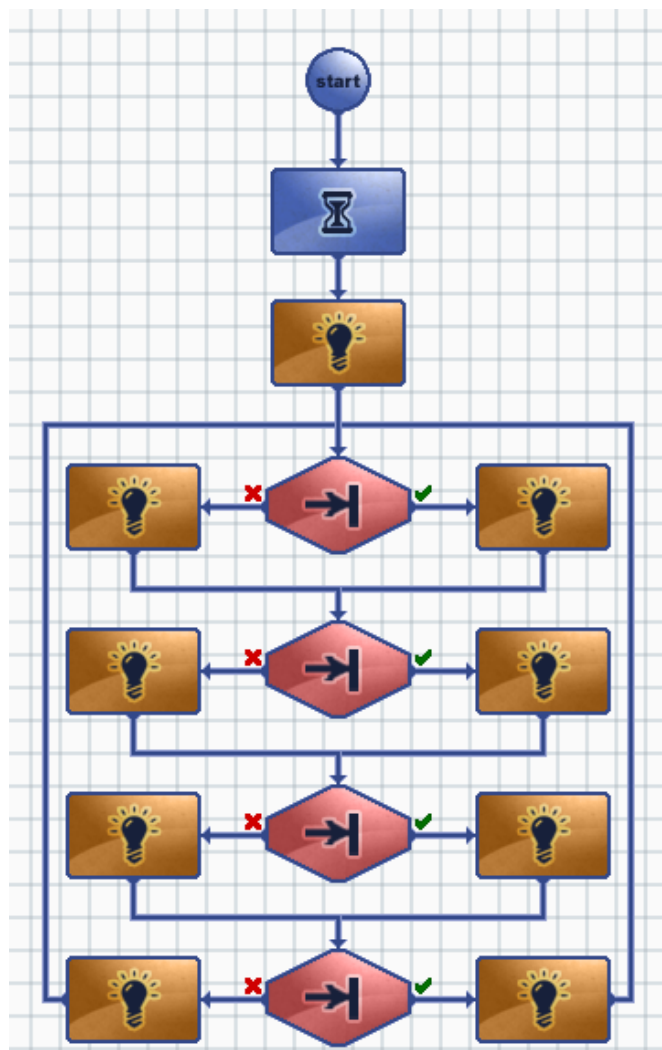


Image 81. First MowayWorld program: obstacle detection

9. We add an straight forward movement indefinitely until an obstacle is found.
10. When an obstacle is found, a command is sent to the robot to rotate 180°. The robot will continue to move in a straight line when the rotation is completed.

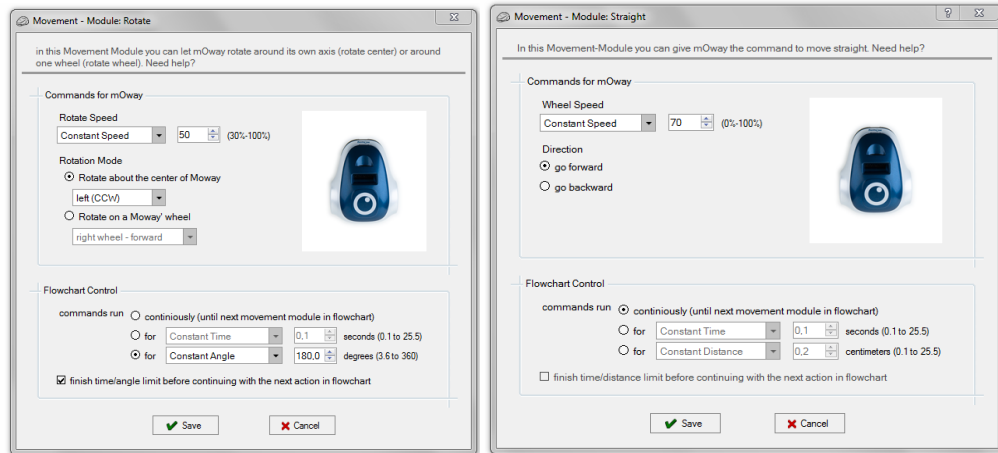


Image 82. Movement and Rotation configuration

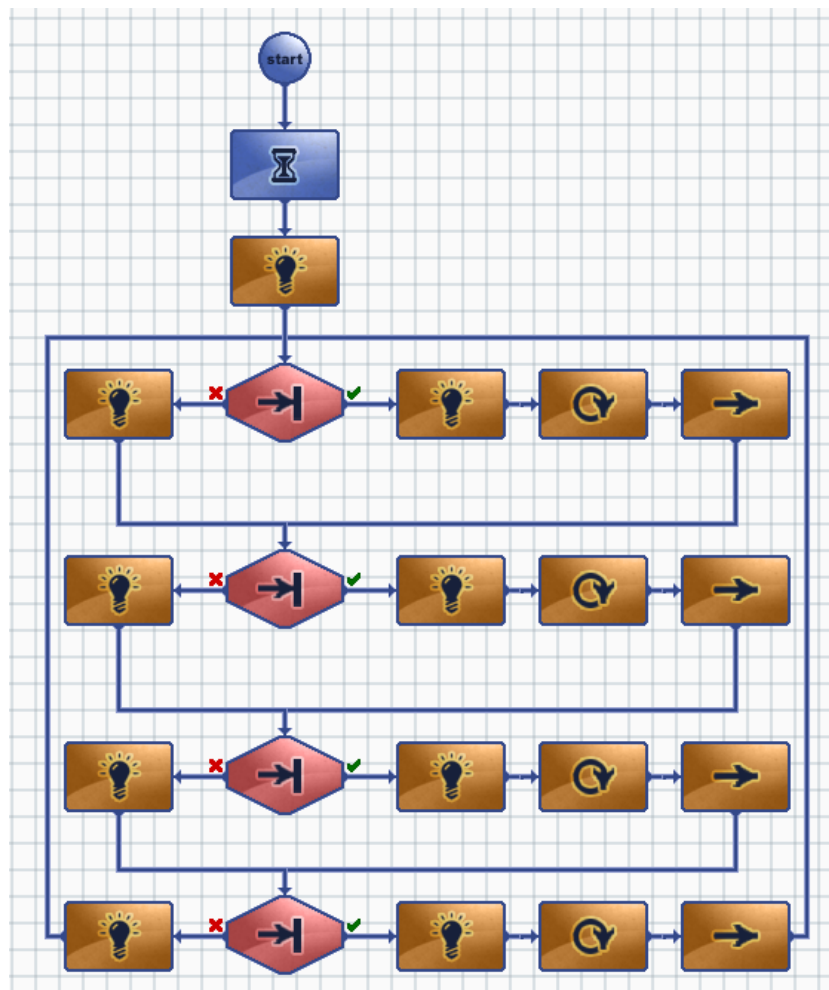


Image 83. End of first program in MowayWorld

7.3. Modules

The mOway programming consist of functions or actions that control the robot. Modules are these functions or actions that mOway can carry out: moving, reading sensors, transmission of radiofrequency messages, etc. These modules are grouped depending on their function.

Each of the modules included in MowayWorld are described below.

7.3.1. Moway Actions

This group of modules makes it possible to control mOway's actuators: motors, speaker and LEDs. Functions of these modules can be performed for a user-defined period of time or distance if the **“finish time/distance limit before continuing with the next action in flowchart”** option is selected. If this option is not selected, the function will be executed indefinitely until another module changes the current function.

- **Movement – Straight**

mOway robot has two motors, one in each wheel. These give it a great flexibility in its movements. Movement straight command makes mOway to go forward or backward describing a straight line trajectory. Speed value can be modified.

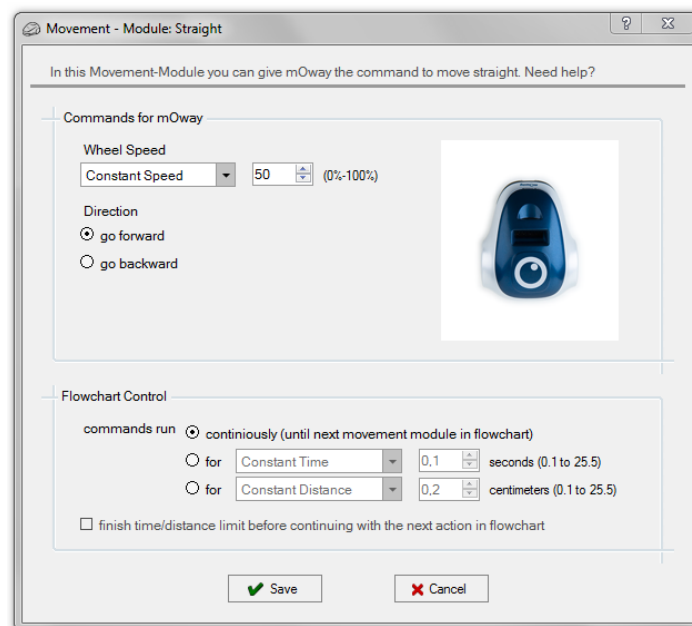


Image 84. Movement – Straight configuration window

• Movement – Free

Free movement is similar to straight movement, but in this case the speed of each motor can be adjusted separately.

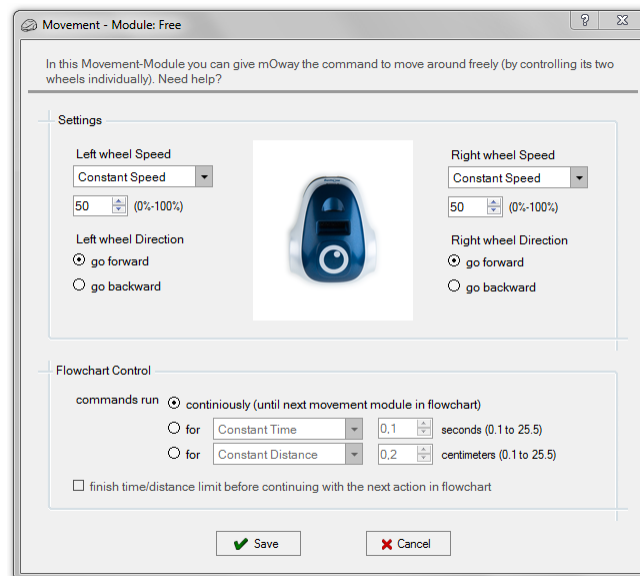



Image 85. Movement – Free configuration window

• Movement – Turn

In this function, drive system will calculate the speed of the motors in order to be able to trace a curve, indicating the speed and turning curvature.

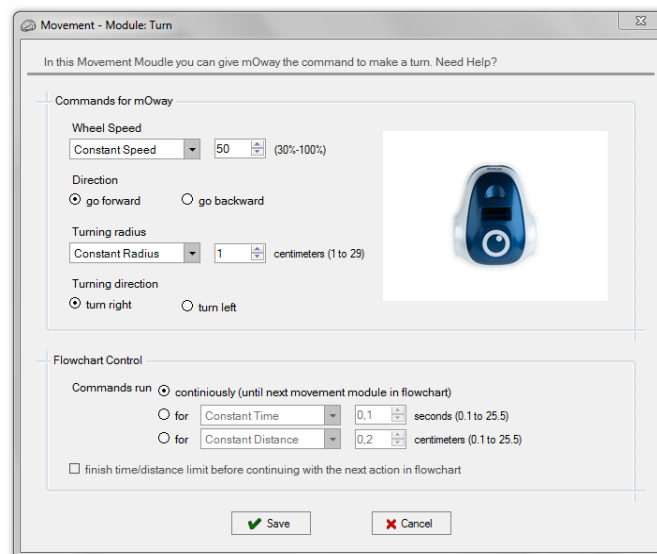



Image 86. Movement – Turn configuration window

- **Movement – Rotate**

With this command, mOway will rotate either on its centre or on one of its two wheels. Turning direction and rotation speed can be defined.

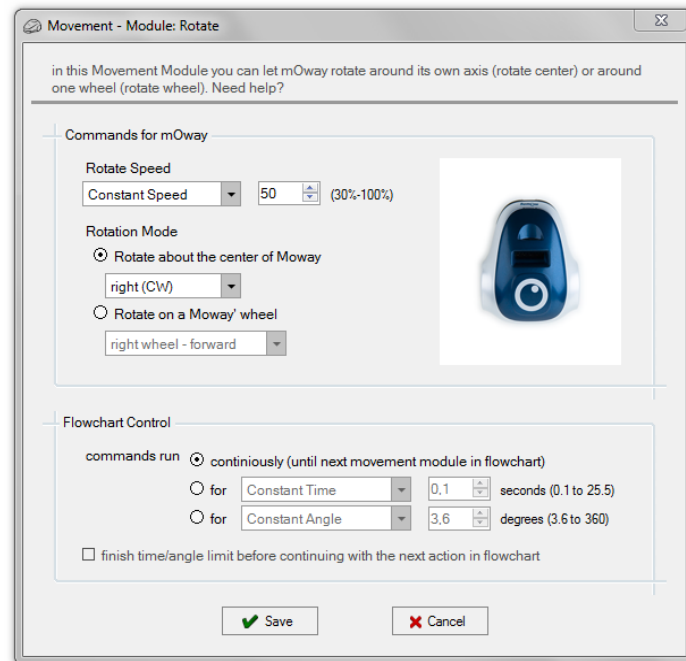


Image 87. Movement – Rotate configuration window

- **Movement – Stop**

This command stops mOway's motors.



- **Sound – Play**

This function makes mOway to emit tones from 244 Hz to 16 KHz. It is possible to define the time that the speaker will be activated.



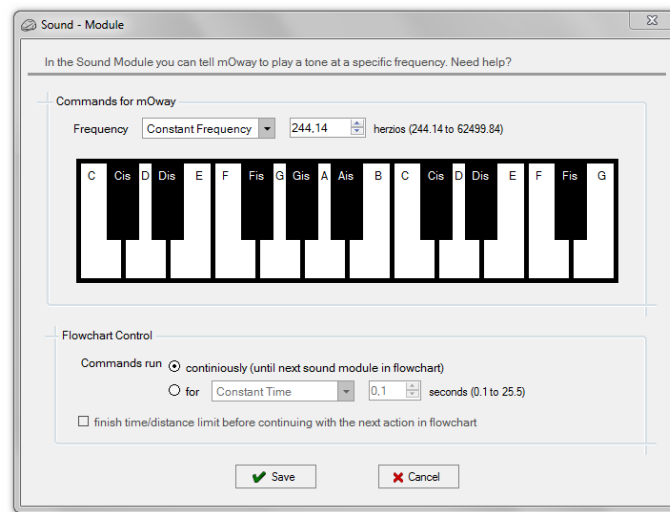


Image 88. Sound – Play configuration window

- **Sound – Stop**

Stops the speaker sound.



- **Lights**

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

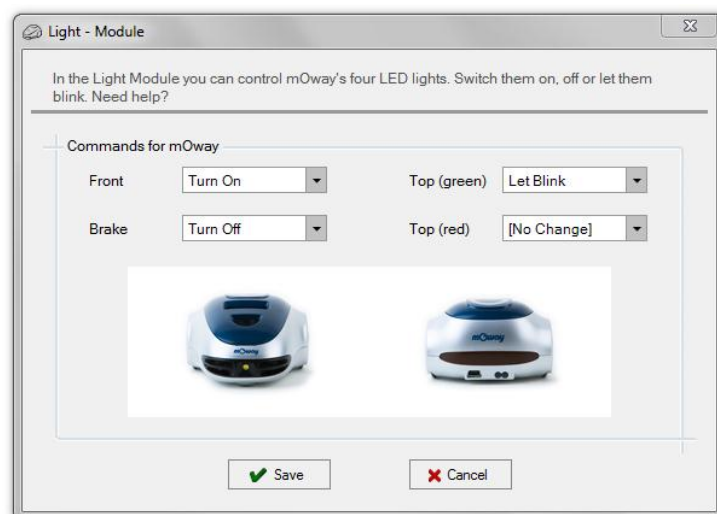


Image 89. Lights configuration window

7.3.2. Sensors Check

This group of modules makes it possible to get the mOway's sensors values. These are conditional modules, so that they have two different outputs: if the condition configured in the module is true, the output will be "true" (green mark on the flowchart). Otherwise, if the condition is false, the output will be "false" (red cross on the flowchart).

- **Obstacle**

This module checks the digital value of the four obstacle sensors. Each sensor can perform one of the following conditions:

- obstacle detected
- no obstacle detected
- detection inactive



In addition, it makes it possible to check the AND or OR boolean operation. If AND option is checked, all the conditions must be true to get a "true" output. On the other hand, if OR is checked, just one of all the condition has to be true to get a "true" output.

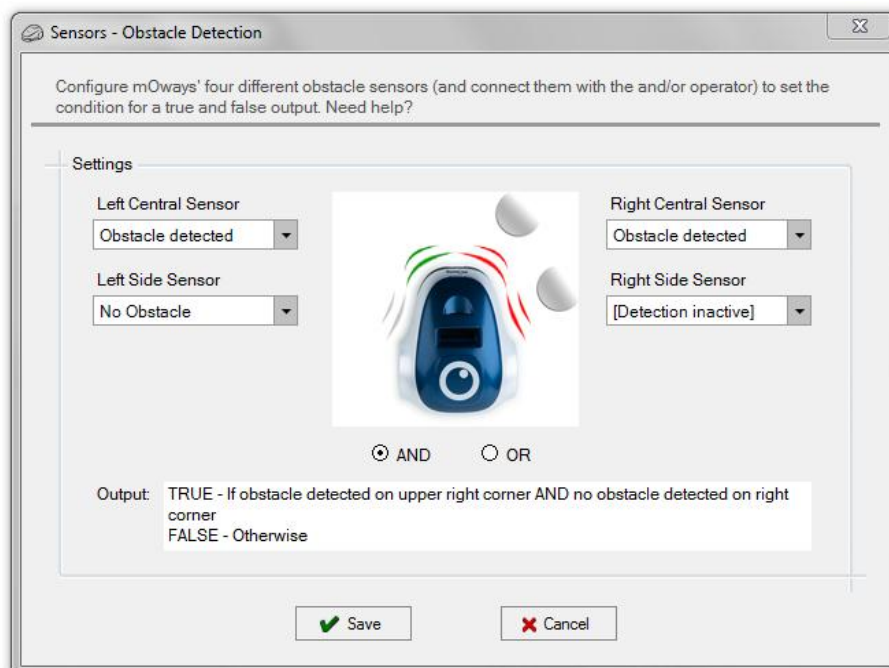


Image 90. Sensors Check - Obstacles configuration window

- **Line**

This module 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.

Each sensor can perform one of the following conditions:

- black line detection
- white line detection
- detection inactive



In addition, it makes it possible to check the AND or OR boolean operation. If AND option is checked, all the conditions must be true to get a “true” output. On the other hand, if OR is checked, just one of all the condition has to be true to get a “true” output.

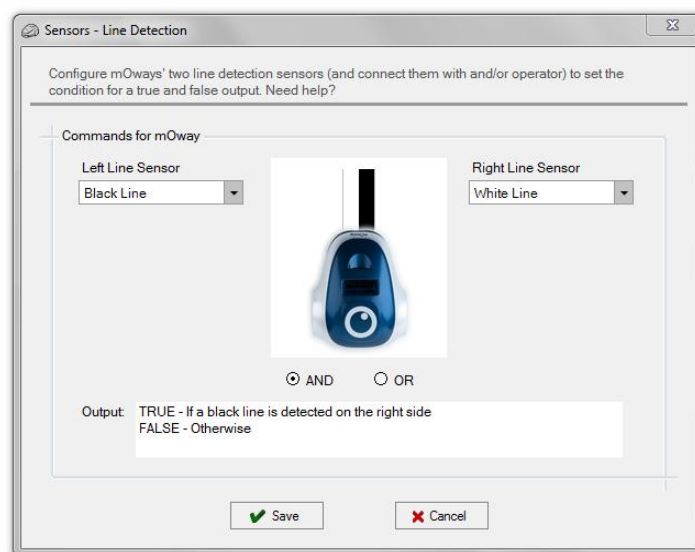


Image 91. Sensors Check – Line configuration window

- **Noise**

This module has a “true” output if mOway detects a loud sound. It has not any configuration window.



- **Tap**

This module has a “true” output if mOway detects a tap or a high acceleration. It has not any configuration window.



7.3.3. Data

This group of modules make it possible to read, write and modify data that mOway robot can provide from sensors and internal memory.

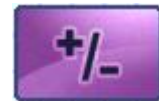
Data is stored in user-defined variables. A variable is created by clicking on “Variable” on the toolbar and then clicking on “New variable”. A name and an initial value have to be defined for this new variable (see “Variables” chapter).

Some of the modules allow to create these variables directly from their configuration window. This permits to configure the module with values that can be modified during the execution of the program.

NOTE: Each variable is stored in one byte of mOway’s internal memory, which means that the value of the variable can be from 0 to 255.

- **Calculate**

This module adds (+) or subtract (-) two values. First parameter is always a variable and it stores the operation result. Second parameter can be a constant or a variable.



- **Reset mOway Data**

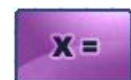
This module initializes time and distance counters stored in mOway’s internal memory.



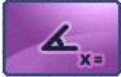


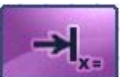
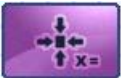

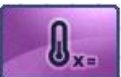
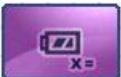
- **Assign Variable**

This group of modules makes it possible to assign a value to a user-defined variable. This value can be constant or the analog value of mOway’s sensors.

- **Value:** assigns a constant value to a variable.
- **Time:** assigns elapsed time to a variable. Value goes from 0 to 255 tenths of a second.
- **Speed:** assigns the speed value of one of the wheels to a variable. Value goes from 0 (stopped) to 100 (maximum speed).
- **Distance:** assigns distance value covered by mOway to a






variable. Value goes from 0 to 255 centimetres.

- **Angle:** assigns the value of mOway turning angle to a variable. Value goes from 0 (0 degrees) to 100 (corresponding to 360 degrees). 
- **Brightness:** assigns the light sensor value to a variable. Value goes from 0 (dark) to 100 (light). 
- **Line:** assigns one of the line sensors value to a variable. Value goes from 0 (white colour detection) to 100 (black colour detection). 
- **Obstacle:** assigns one of the obstacle sensors value to a variable. Value goes from 0 (no detection) to 100 (closest detection). 
- **Accelerometer:** assigns one of accelerometer axis value to a variable. Value goes from 0 (negative acceleration limit) to 255 (positive acceleration limit). When there is not acceleration, the value is 127 (range mid-point). 
- **Noise:** assigns the microphone value to a variable. Value goes from 0 (silence) to 100 (loud noise). 
- **Temperature:** assigns the robot temperature value to a variable. Value goes from 0°C to 255°C. 
- **Battery:** assigns battery level to a variable. Value goes from 0% to 100%. 

• Compare

This group of modules makes it possible to compare a variable or a sensor value with another value (constant value or user-variable). The following are the comparison operators: equal (==), different (<>), greater (>), greater or equal (>=), less (<), less or equal (<=).

- **Data:** compares a variable with a constant value or a user-variable. 
- **Time:** compares elapsed time. Values goes from 0.1 to 25.5 seconds. 
- **Speed:** compares one of the wheels speed value. Values 

goes from (stopped) to 100 (maximum speed).

- **Distance:** compares distance value covered by mOway to a variable. Value goes from 0 to 255 centimetres.
- **Angle:** compares the value of mOway turning angle. Value goes from 0 (0 degrees) to 100 (corresponding to 360 degrees).
- **Brightness:** compares the light sensor value. Value goes from 0 (dark) to 100 (light).
- **Line:** compares one of the line sensors value to a variable. Value goes from 0 (white colour detection) to 100 (black colour detection).
- **Obstacle:** compares one of the obstacle sensors value to a variable. Value goes from 0 (no detection) to 100 (closest detection).
- **Accelerometer:** compares one of accelerometer axis value. Value goes from -2g (negative acceleration limit) to 2g (positive acceleration limit). When there is not acceleration, the value is 0 (NOTE: g value is 9.81m/s^2).
- **Noise:** compares microphone value. Value goes from 0 (silence) to 100 (loud noise).
- **Temperature:** compares the robot temperature value. Value goes from 0°C to 255°C .
- **Battery:** compares battery level. Value goes from 0% to 100%.



7.3.4. Flowchart Control

- **Pause**

This module makes it possible to insert a pause in the program with a duration set in multiples of 0.05 seconds. The pause parameter may be a constant or a variable.



- **Call Function**

This module makes it possible to call a function or subroutine defined by user. Subroutines will be explained on chapter “Functions /Subroutines”.



- **Finish**

This module sets the program end. If the program consist of an infinite loop, this module is not necessary.



7.3.5. Expansion

This group of functions makes it possible to control the expansion connector of mOway robot, either to use radiofrequency (RF) communication, mOway camera module or user expansion modules.

WARNING!

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

- **Module IO – Configuration**

This module configures the expansion connector pins as inputs or outputs and assigns a initial value for outputs.



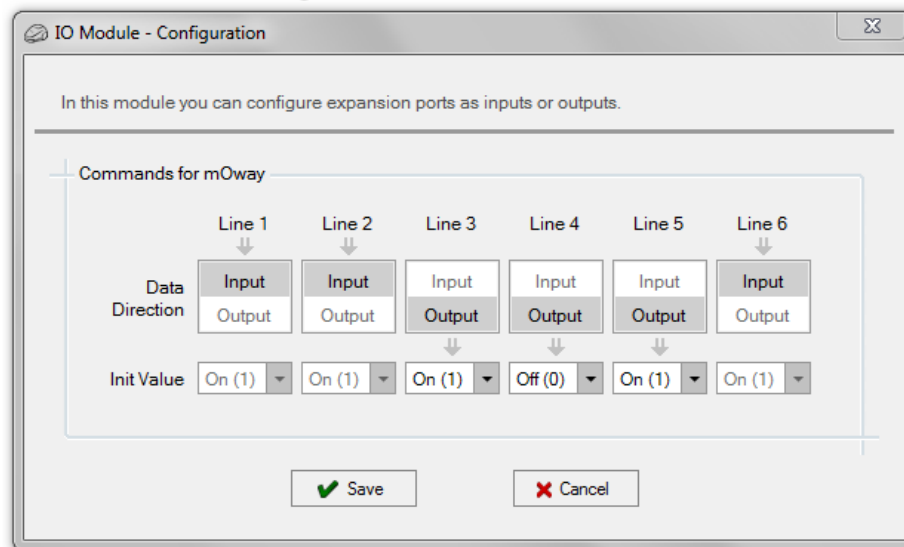


Image 92. Module IO – Configuration configuration window

- **Module IO – Set Output**

This module sets (On), resets (Off) or toggles expansion connector pins that are configured as outputs.

WARNING!

“Set Output” values only can be assigned for pins configured as outputs, leaving other lines with “No change” value. If a value is assigned to an input, this may damage the robot irreversibly.

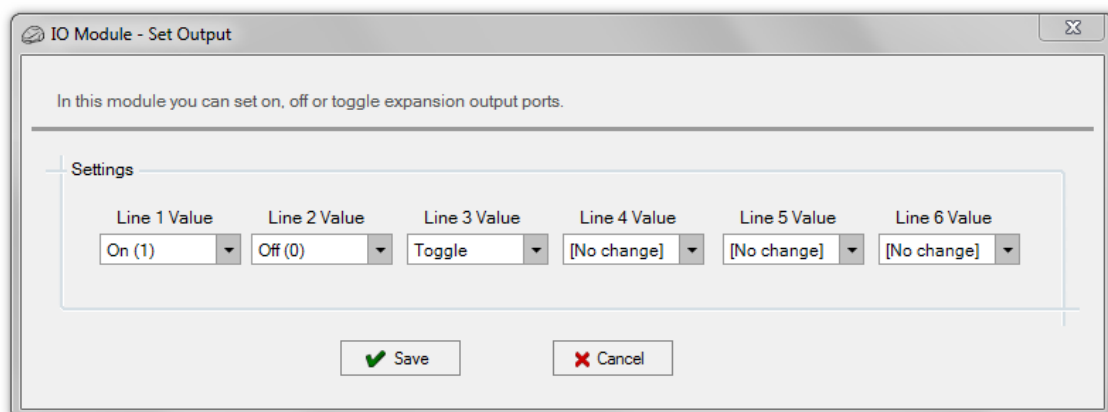


Image 93. Module IO – Set Output configuration window

- **Module IO – Check Input**

This module checks the digital value of one of the 6 pins of the expansion connector. It compares if the value of the selected pin is equal (\equiv) or different (\neq) to “on” (1) or “off” (0).



- **Camera – Play**

This module activates the mOway camera. One of the four channels has to be selected. This channel must be the same of the **Moway Camera Board** video receptor.



- **Camera – Stop**

This module turns the camera off.



- **Communicate – Start**

This module configures the robot in order to establish a radiofrequency (RF) communication. An identification for the robot and a communication channel must be selected.



For establishing communication between two mOways, they must be configured with the same channel and an unique address for each one. If the configurations is correct, the output is “true”. Otherwise, the output is “false”.

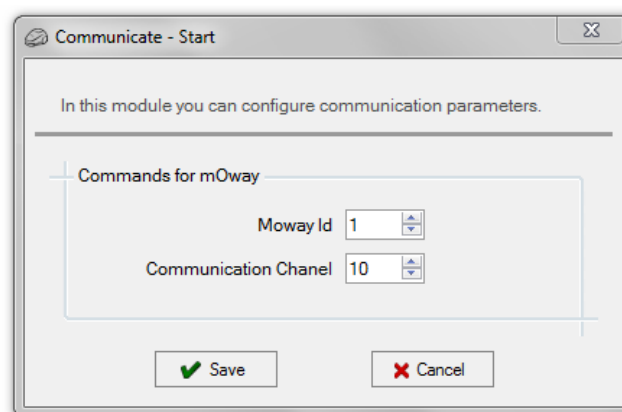


Image 94. Communicate – Start configuration window

- **Communicate – Stop**

This module stops the RF communication. It has not any configuration window.

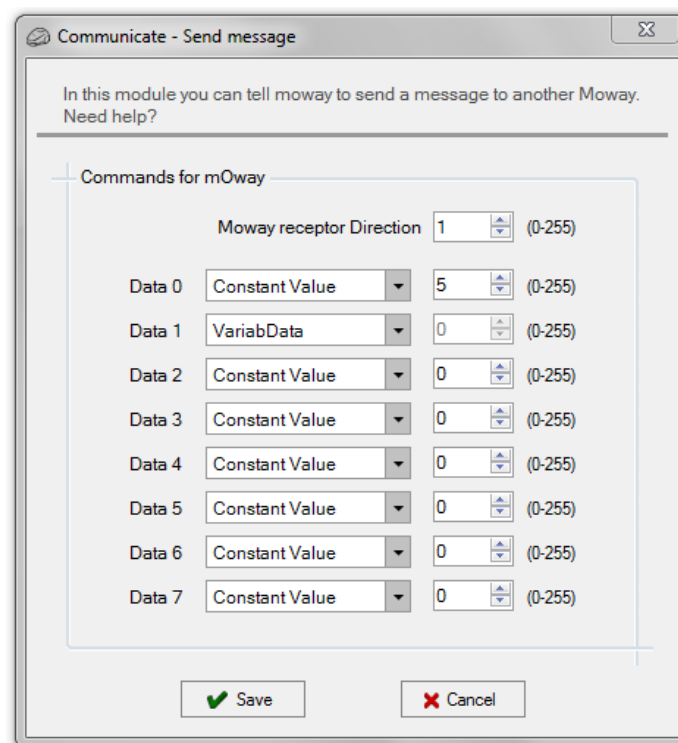


- **Communicate – Send**

This module makes it possible to transmit a frame to a specific address. The address of the receptor and the data, which can consists of constants or variables, must be indicated in the frame. If the sent data has been received by the receptor, the output is “true”. Otherwise the output is “false”.



Before using this conditional, the RF communication must be configured using the "Communicate - Start" module. All the robots taking part in the RF communication must have the same channel and different addresses.



Commands for mOway	
Moway receptor Direction	1 (0-255)
Data 0	Constant Value 5 (0-255)
Data 1	VariabData 0 (0-255)
Data 2	Constant Value 0 (0-255)
Data 3	Constant Value 0 (0-255)
Data 4	Constant Value 0 (0-255)
Data 5	Constant Value 0 (0-255)
Data 6	Constant Value 0 (0-255)
Data 7	Constant Value 0 (0-255)

Save Cancel

Image 95. Communicate – Send configuration window

• Communicate – Receive

This module makes it possible to receive 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. If the data sent by the transmitter has been received correctly, the output is “true”. Otherwise the output is “false”.



Before using this conditional, the RF communication must be configured using the "Communicate - Start" module. All the robots taking part in the RF communication must have the same channel and different addresses.

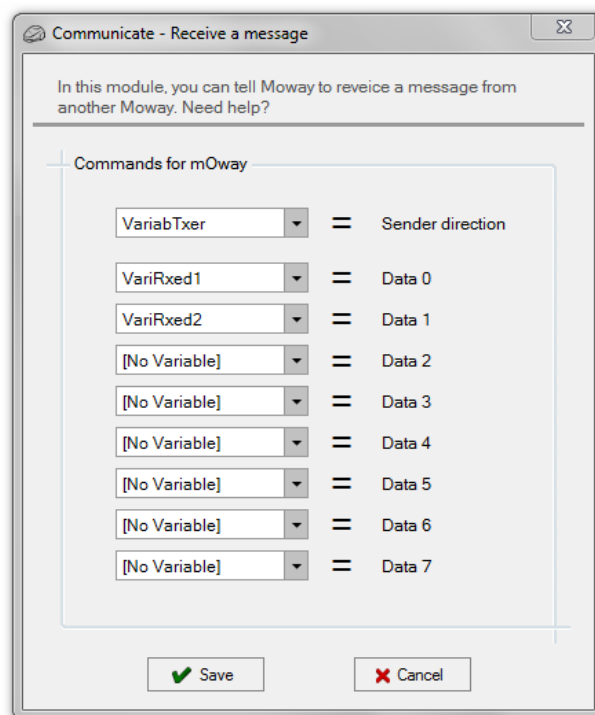


Image 96. Communicate – Receive configuration window

7.4. Variables

For more flexible programs some modules make it possible to use variables. Variables makes possible to save values into robot internal memory. A variable contains a value that can change during the execution of the program.

This is useful to read the values of the mOway's internal sensors (for example, speed value, pause time, read analog obstacle sensor values, etc.). Then these values can be compared, transmitted through RF to check sensor status, etc.

Variables can be created from some modules. They can also be defined from the toolbar, by clicking on “Variables” and then clicking on “New variable”. A name and an initial value have to be declared. The name must not have more than 14 letters.

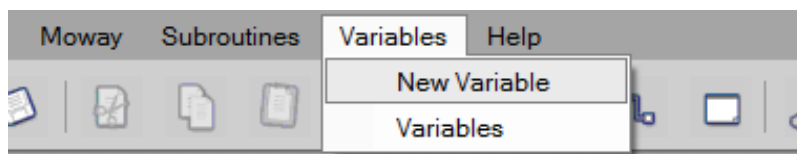


Image 97. Creation of a new variable from toolbar

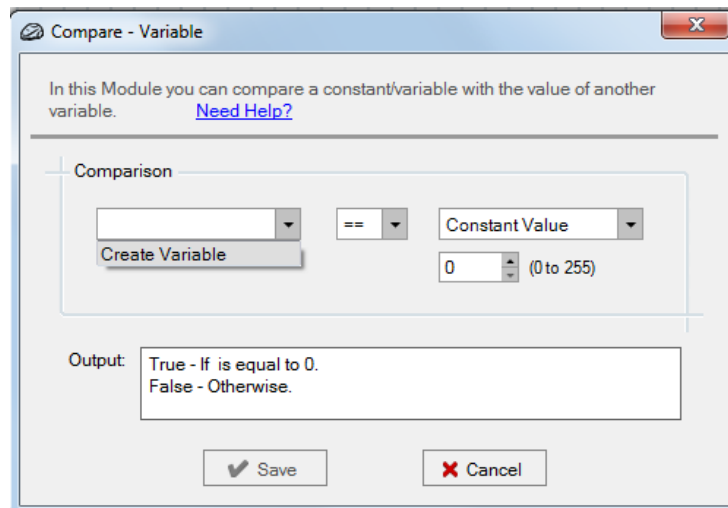


Image 98. Creation of a new variable from a module

Variables can be edited from toolbar, “Variables – Variables”.

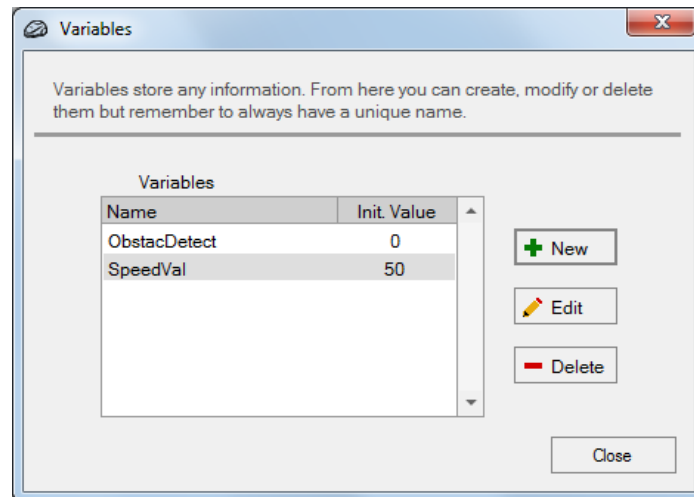


Image 99. Variables window

7.5. Functions / Subroutines

A function or subroutine is a piece of program that can be used in another program. For example, a user can develop a diagram that turns on the front LED, waits one second, then turns off the LED and finally waits one more second.

If this program is defined as a Function, it can be called from the main diagram. In order to create a new function, click on “+” icon and give it a name (for example, “MyFunction”). A new tab called “MyFunction” will appear on the top of the flowchart editor.



Image 100. Creation of a new function

The functionality described above can be developed in the “MyFunction” flowchart editor.

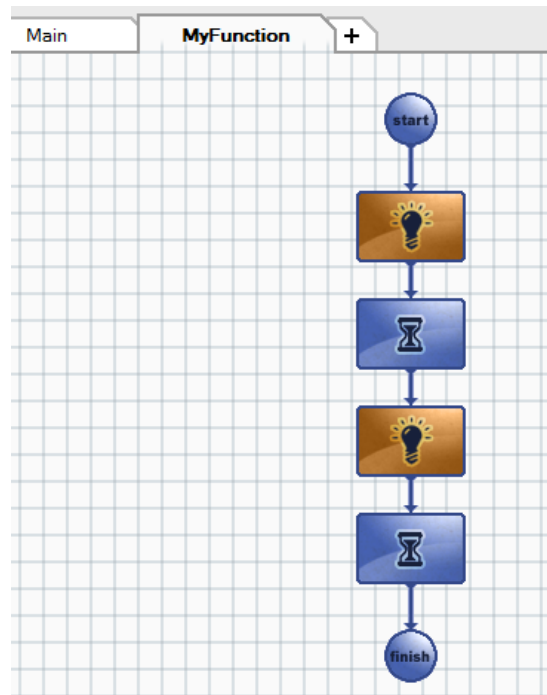


Image 101. MyFunction diagram

In order to go back to the main diagram, click on “Main” tab. There the “MyFunction” function can be called (“Flowchart Control – Call Function”). In the “Call Function” configuration window, “MyFunction” is selected.

In this example, it is called three times, so that the front LED will blink three times, once per second.

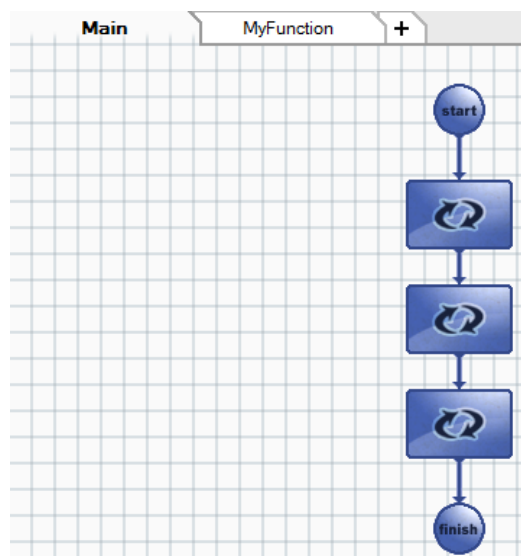


Image 102. Calling “MyFunction” three times from main diagram

8. Applications

8.1. *Communications Window*

The Communications Window makes it possible to send and receive RF messages using RFUSB device connected to PC. In order to start RF communication, push the “Communications” icon.

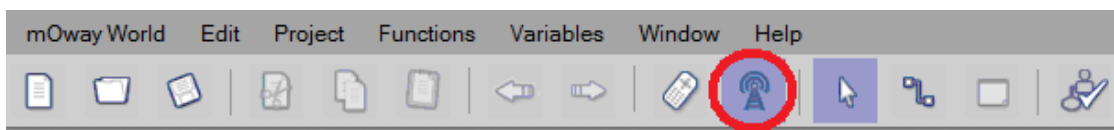


Image 103. “Communications” icon

The Communications Window will appear on the right side of the workspace. In order to start RF communication, choose an address and a RF channel for the RFUSB (“2” and “0” in this example) and push **Start** button.

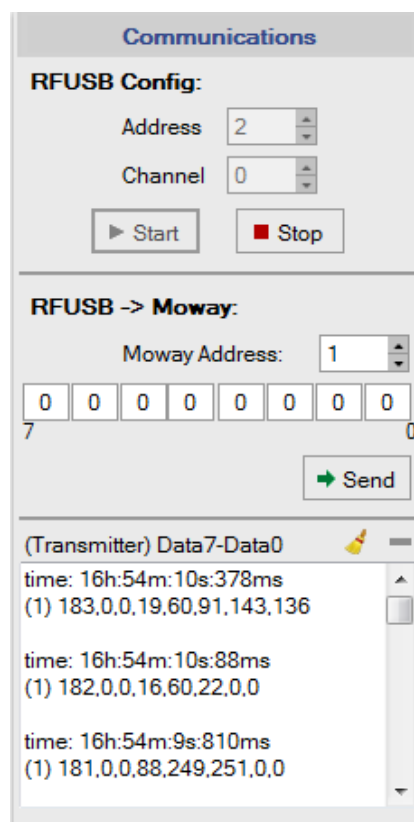


Image 104. “Communications” window

In order to send a message from PC to mOway, select mOway robot address (“1” in this example), write the data to be sent (from Data 7 to Data 0) and press **Send** button. If a “0” is selected for “Moway Address” the message will be received by all the robots near the PC.

When RFUSB receives a message from mOway, the communication window displays the time when the message was received and the data. The format of the data is:

- Address of transmitter mOway is displayed in brackets
- Data arranged from Data 7 to Data 0

8.2. *Moway Cam*

MowayCam application displays the camera images and lets the user to save a static image in a storage device connected to the computer. This application can be launched from MowayWorld toolbar (“View -> mOway Cam) or clicking this icon:



Image 105. MowayCam access from MowayWorld

Once the icon is pressed, MowayCam lateral panel appears.

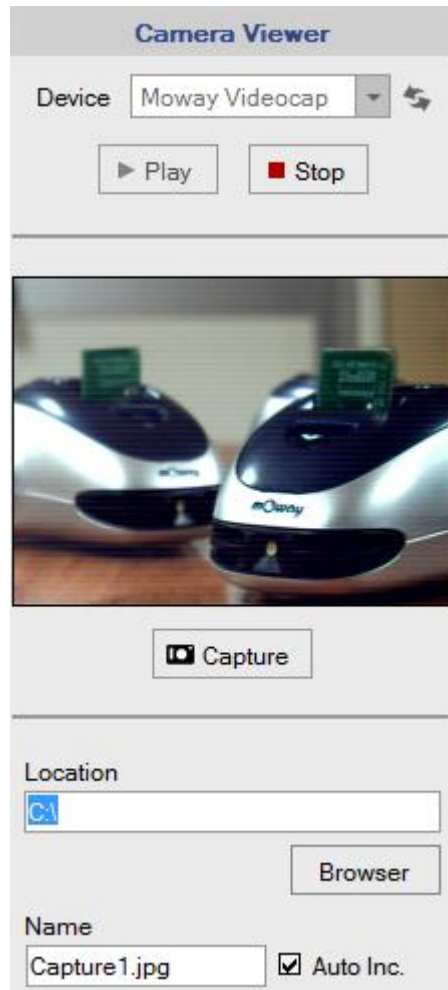


Image 106. MowayCam panel

This window displays camera images and allows the user to control the video receptor and to save a static image on PC.

- Click on **Refresh** button to see all the video devices connected to PC. Depending on the driver version, Moway video receptor may be named as “Moway Videocap”, “STK1160 Grabber” or “USB2.0 ATV”.
- Click on **Play** button to start displaying camera images.
- Click on **Stop** button to deactivate video receptor **Moway Camera Board**.
- Click on **Save** button to save the current image in the path and name displayed on that fields.

IMPORTANT: Video receptor **Moway Camera Board** MUST NOT be disconnected from USB port while MowayCam is showing camera images. If it is disconnected while showing camera images, some computers could restart. Click on Stop button or close MowayCam panel before disconnecting it.

8.3. MowayRC

MowayRC is an application 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).

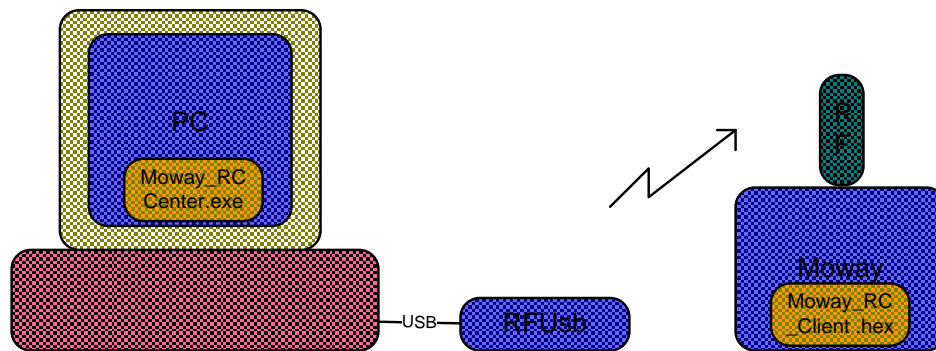


Image 107. Moway RC Diagram

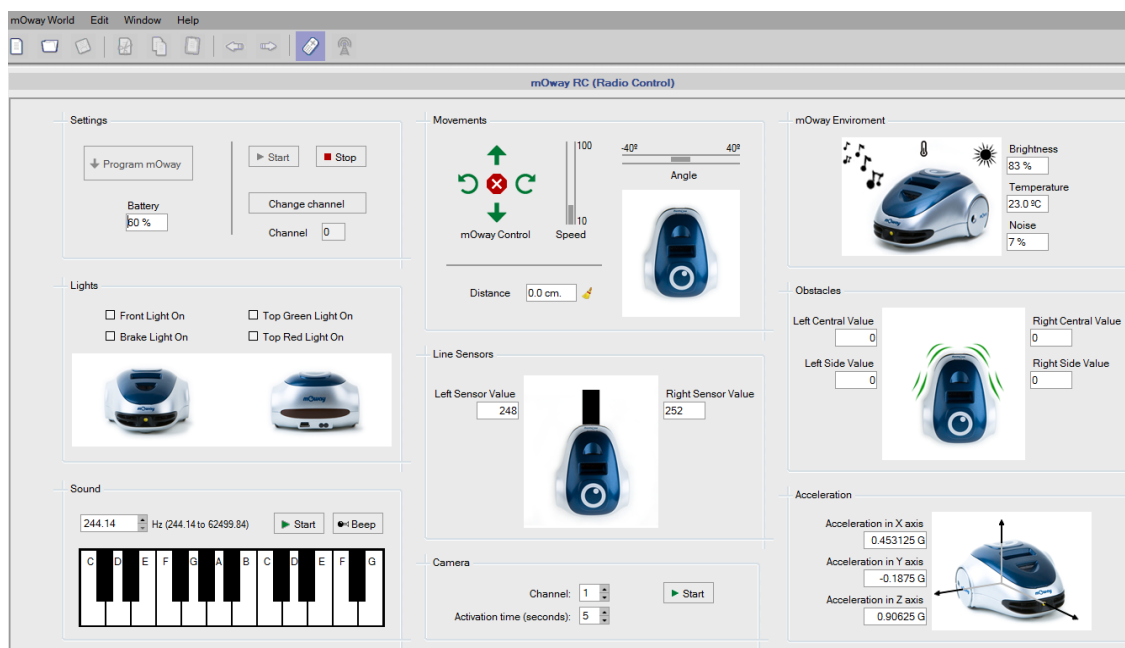


Image 108. Moway RC Center

You can access the application in “Window – Moway RC”. The program “mOway_RC_Client” is downloaded in the robot and the application starts. The fields of the RC Center will be explained on the next lines.

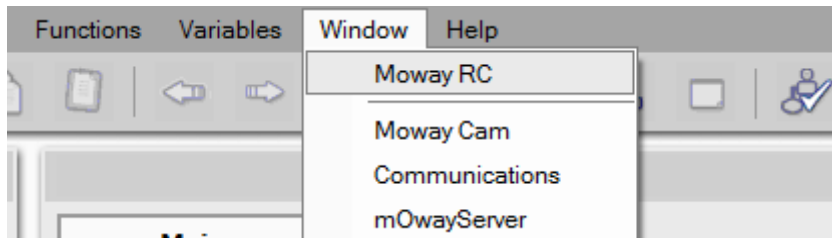


Image 109. Access to Moway RC

8.3.1. RF configuration

In this part the RFUSB module is configured with a “2” default address and “0” for the communications channel (default mOway RC program channel).

Once connected, the communications channel can be changed if WI-FI, Bluetooth, Microwaves, other mOways, 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 “0”.

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

In case the robot is not programmed with “mOway_RC_Client”, it can be programmed pressing **Program mOway** button.

8.3.2. Movements

Once the RFUSB is connected mOway can be sent commands. The robot’s movements can be controlled by means of the panel buttons or keyboard (*W, S, A, D* keys). There also are two bars to determine the speed and turning curvature. The covered distance can be reset with **Reset distance** button.

8.3.3. LED

In this section mOway’s four LEDs are switched on and off.

8.3.4. Speaker

In this section is checked the switched on and off of the robot's speaker in a particular frequency. He robot can also reproduce music notes.

8.3.5. Sensor status

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

- Line sensors: the darker is the surface, the higher is the value.
- mOway environment: brightness, temperature and noise detected by the robot.
- Obstacles: the closer is the obstacle, the higher is the value.
- Acceleration: 3 axis acceleration values.

8.3.6. Camera

It activates mOway camera on the selected channel for a maximum of 15 seconds. While the camera is on, the communication between mOway and PC stops, in order to avoid interferences. Once the activation time finishes, the communication automatically starts again.

In order to watch camera images it is necessary to launch "Camera" panel.

8.4. MowayServer

This application allows mOway to communicate with other devices, such as phones, tables and PCs through Wifi technology. These devices can connect to the net established by mOway by means of **Moway Wifi Module**²². Once they are connected, the robot can be controlled from device web browser.

The application is launched from toolbar.

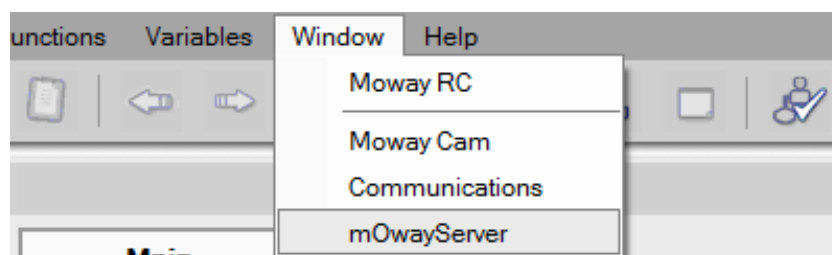


Image 110. Access to Moway Web Server

²² Module not available in all kits

The lateral panel appears.

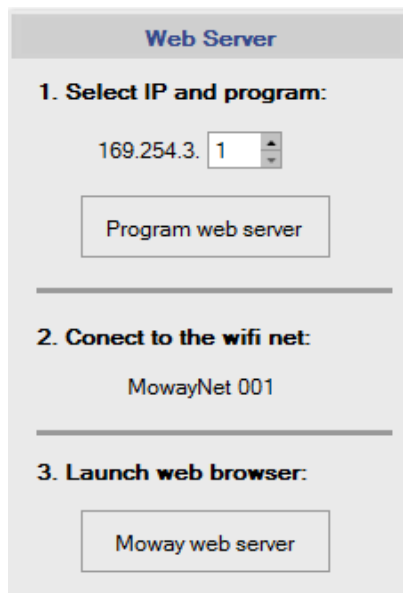


Image 111. Moway Web Server panel

In order to communicate with mOway Web Server, it is necessary to follow these steps:

- 1) Select an IP (only the last field can be modified) and press **Program web server** button.
- 2) The robot will establish an adhoc wifi net with the name MowayNet*** (***) is the selected number for IP in the previous step).
- 3) If the web server will be accessed from PC, press **Moway web server** button to launch web browser with the selected IP. If another device is used, launch web browser and introduce the selected IP.

Once this is done, the mOway Web Server will be displayed. It makes it possible to control mOway movements, LEDs and read some sensor values.

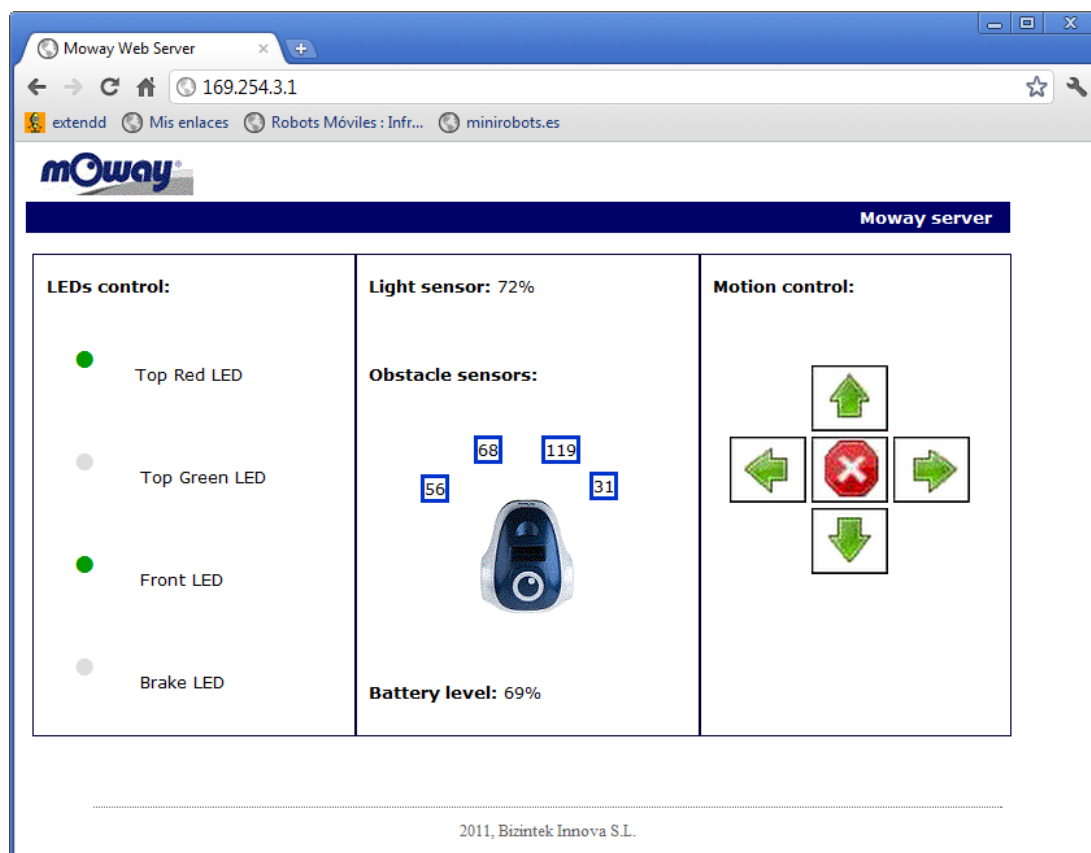


Image 112. A Moway Web Server

NOTE: Connecting to MowayNet or web server displaying can take a while, depending on the wifi connection and on the web browser.

9. Simulation

9.1. Introduction

MowayWorld simulator allows testing the working of a program before programming it in mOway robot. In this way you can detect any errors you could have in the program and fix them quickly.

The simulator is opened from the toolbar. NOTE: When you activate the simulator, the side panels remain inactive.

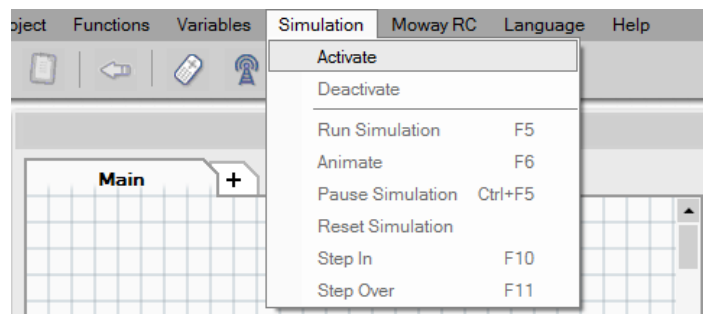








Image 113. Simulator access

By activating the simulator, it will appear in the side panel:



Image 114. Simulator side panel

The modes available in the simulator are the following:

-  **Run:** In *Run* mode, the simulator executes the program in “real time”, i.e., the speed at which the program would execute in the real robot.
-  **Animate:** In *Animate* mode the simulator executes the program step by step, pausing for a second in each block. The block that is running the simulator at each time is indicated by a yellow arrow.
-  **Pause:** Stops the simulation at the current block.
-  **Reset:** Pressing the reset button, the simulation returns to the initial state, i.e., the beginning of the program. Also resets the value of the variables.
-  **Step In:** In this mode, the simulator moves through the program step by step, stopping at each block. It also moves through the functions created by the user.
-  **Step Over:** In this mode, the simulator moves through the program step by step stopping at each block. Unlike the previous mode, the simulator does not advance into the user functions.

9.2. Functioning

In the simulator the user can vary the mOway sensors value and see how this affects the operation of the robot, depending on the program it's being simulated. The sections of the robot that can be viewed and controlled in the simulation are the following:

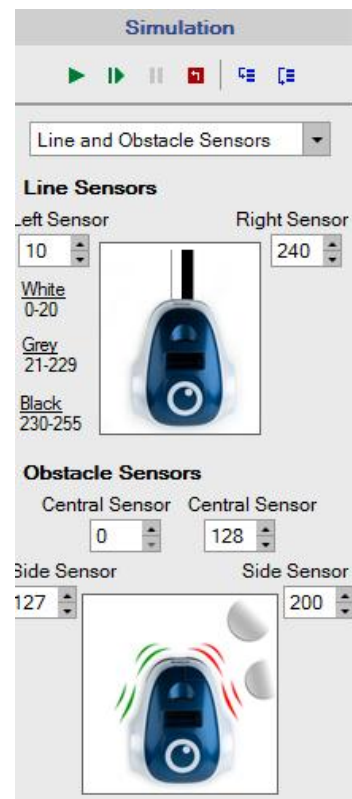
- **Movement, lights and sound:** shows the state of the motors, the state of the LEDs and the mOway speaker.
 - The rotation of the wheels is indicated by red arrows. It also reflects the distance travelled by the robot.
 - The sound emitted by the robot is indicated by the speaker icon. When it is green, it simulates that mOway beeps. When it is red, it simulates that the sound has stopped.



- Line and obstacles sensors:** simulates the change of the value of the line and obstacles sensors. The value of the simulator sensors is defined by the analog value of the real sensors of the robot. Therefore, the value of the simulated sensor are the following:

Line sensors	
Simulated line	Sensor value
White	0 to 20
Grey	21 to 229
Black	230 to 255

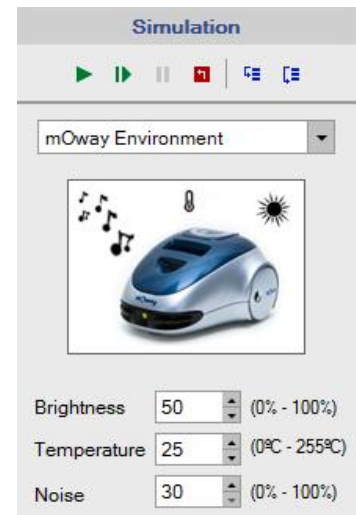
Obstacle sensors	
Simulated obstacle	Sensor value
Obstacle not detected	0 to 127
Obstacle detected	128 to 255



- **Environment:** it simulates the variation of the light and temperature sensors and the microphone.

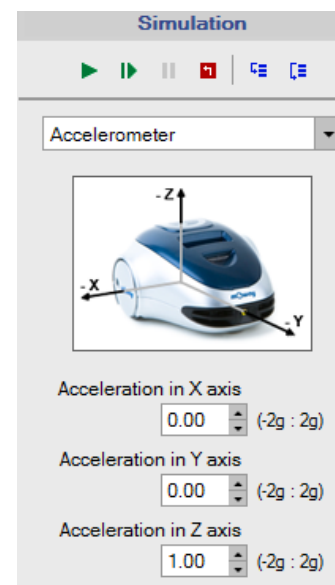
Light sensor	
Simulated environment	Sensor value
Darkness	0
High-brightness	100

Noise sensor (microphone)	
Simulated environment	Sensor value
Silence	0
Loud noise	100

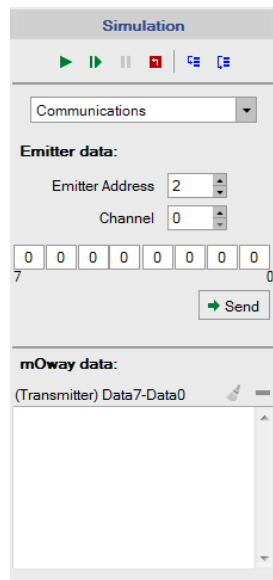


- **Accelerometer:** it simulates the variation of the accelerometer value in its 3 axis. The value of the accelerometer in each axis varies between -2g and +2g, being “g” the acceleration of gravity ($9,81\text{m/s}^2$). For example, if we simulate that mOway is on an even surface, the values are the following:

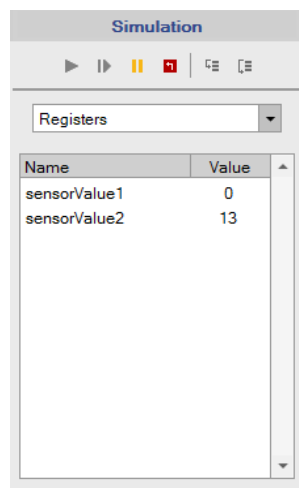
Accelerometer	
Accelerometer axis	Accelerometer value (g)
X axis	0
Y axis	0
Z axis	1.00



- **Communications:** it simulates the transmitting of RF data to the mOway robot, and also the receiving data sent by the robot. “Emitter data” field simulates the data sending to the robot. “mOway data” field shows the data sent by the simulated robot.

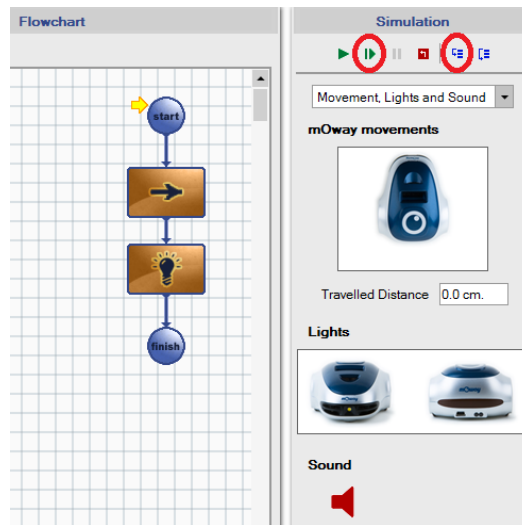


- **Registers:** it shows a list of the variables created in the program with its values at all times.

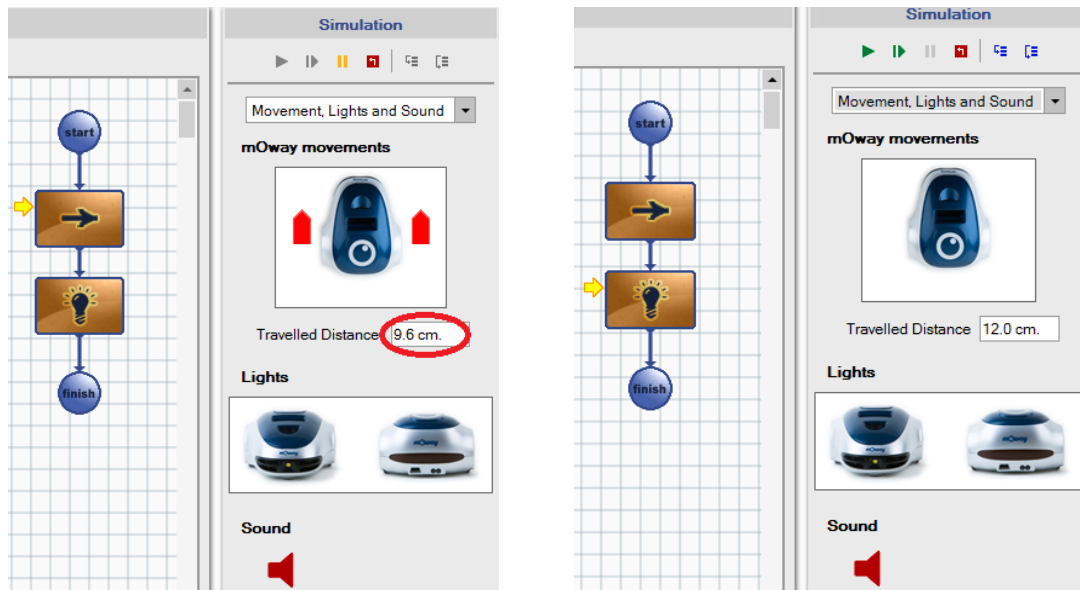


9.3. Simulation example

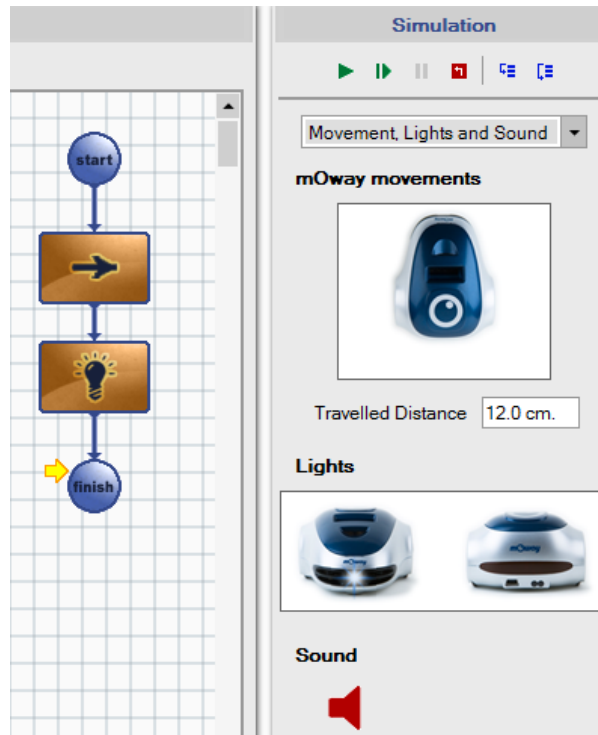
The following example shows the simulation of a program in which the robot moves straight for 12cm and then lights the front LED. Pressing the button “Animate” or “Step In”, a yellow arrow indicates where the simulator is in the diagram.



While the simulator executes the block “Straight”, it indicates the direction of the rotation of the wheels, and the travelled distance is increased.



Once the block “Straight” has finished (the robot would travel 12cm),the simulator executes the block “Lights”, and the front LED lights. This done, the simulator ends.



10. mOway Scratch

10.1. Introduction

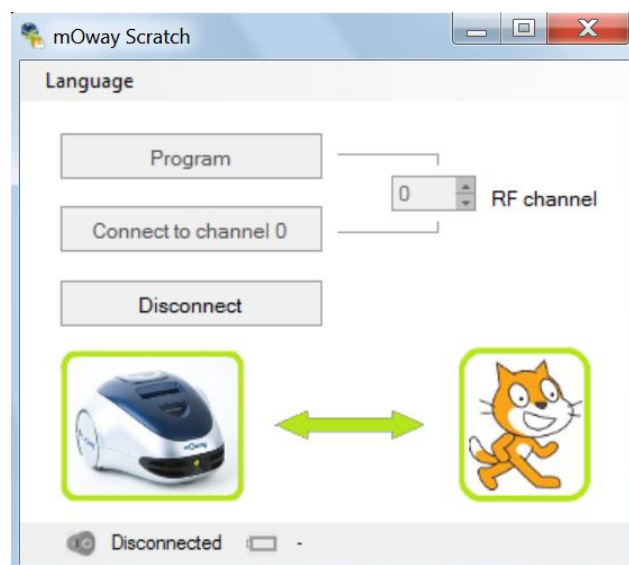
Now mOway is also compatible with Scratch. The communication between Scratch and mOway is bidirectional: Scratch receives the mOway detectors value, and at the same time mOway receives Scratch commands. This functionality uses Scratch sockets connection to obtain data from remote sensors and uses “send to all” commands for sending orders to mOway. In this case “mOway Scratch” application works as data gateway between Scratch and mOway.



The connection between mOway and Scratch is done by the mOway RF modules and the RFUSB module connected to the computer. Therefore it is necessary having the RF module placed in the mOway expansion slot and the RFUSB module connected to a computer USB port.

10.2. Functioning

The mOway-Scratch communication is activated by opening “mOway Scratch” application.



The “Program” button programs the needed firmware in mOway for a right connection with Scratch. In this case, mOway robot should be connected to a USB port. The firmware will be programmed to work in the selected channel in the “RF channel” box. Once the firmware is programmed in a certain channel, this channel should be selected when we get connected to Scratch.

To initiate communication, the “Connect to channel...” button is pressed. mOway robot should have the Scratch firmware previously programmed, the RFUSB module should be connected to PC and Scratch must be opened with a mOway sample program.

Once the connection from Scratch is started it is possible to obtain the value of any of the mOway sensors, and also it is possible to send moving orders to mOway, or switch on/off the LEDs and buzzer. To configure these actions we use variables in Scratch.

10.3. Step by step

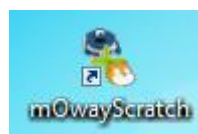
1. Connect RF module into the expansion connector of mOway robot.



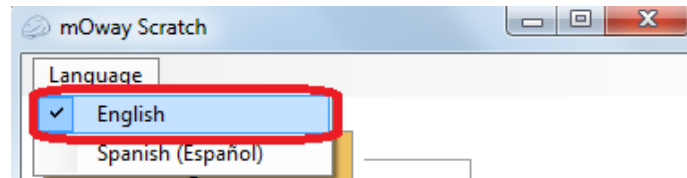
2. Connect the RFUSB to PC.



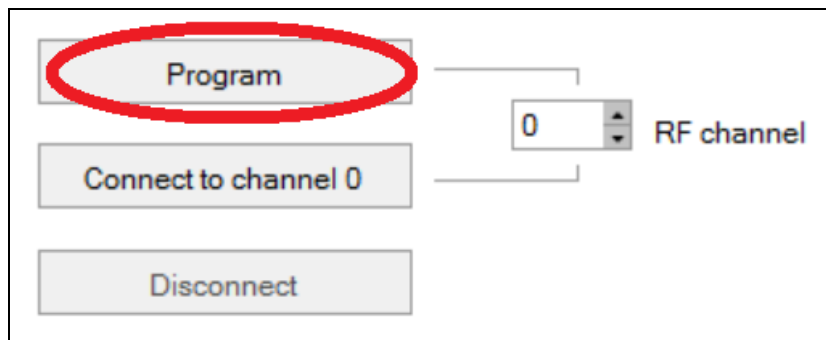
3. Connect mOway robot to PC.
4. Open “mOway Scratch” application:



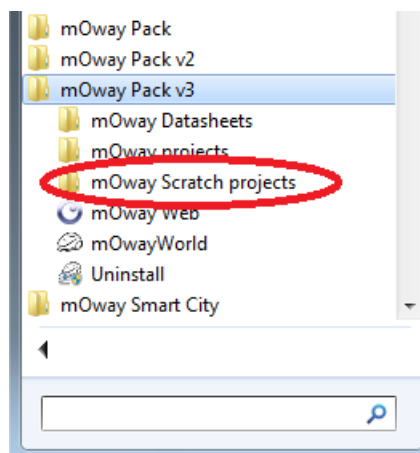
5. Select the language in which the Scratch programs will be developed (in this case, “English”).



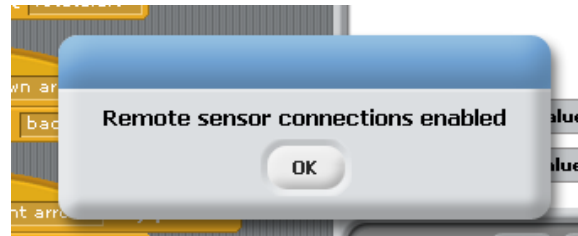
6. Press the “Program” button on the lateral panel (this is not necessary if the robot has been previously programmed in this way).



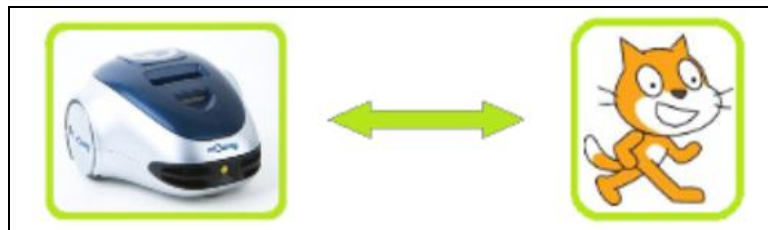
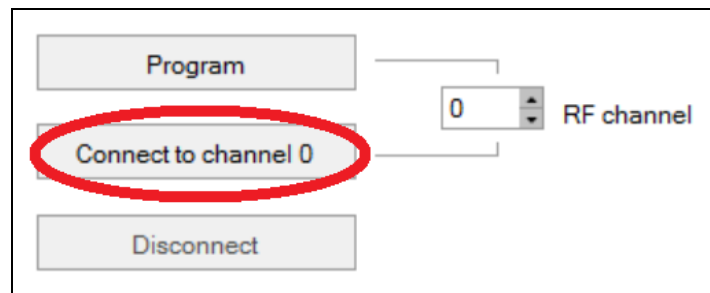
7. Once the robot has been programmed, disconnect it from the PC and switch the robot on.
8. Open one of the Scratch projects for mOway. You can open them from “Start -> All programs -> mOway Pack v3 -> mOway Scratch projects”. For example, open the “moway_RC.sb” project.



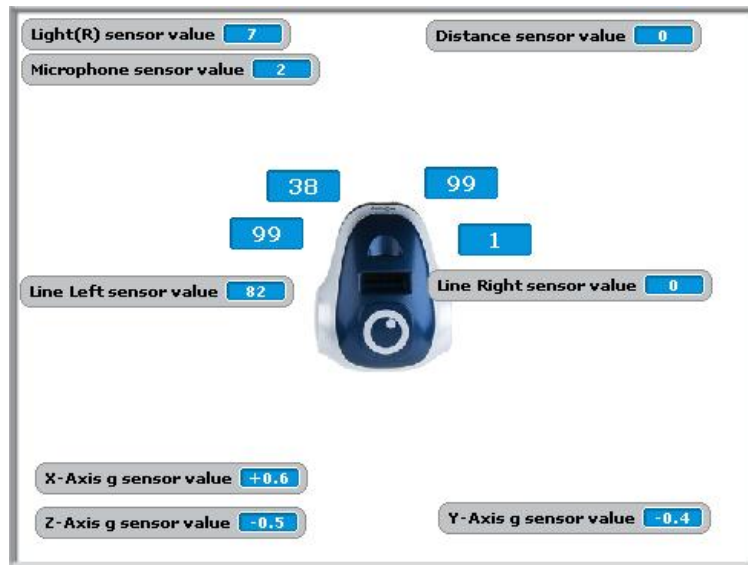
9. Press “OK”.



10. Once the Scratch project is open, press the “Start” button on the lateral panel of MowayWorld. The panel status is the following:

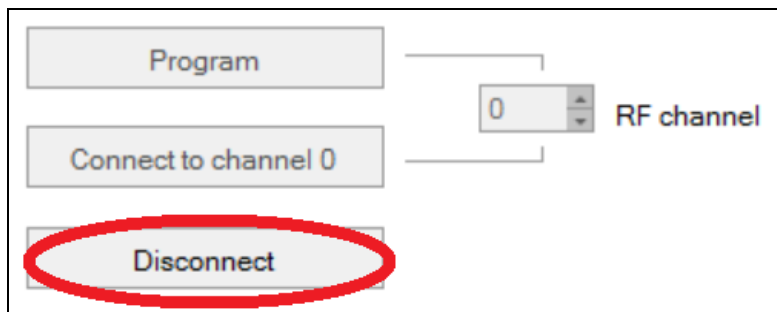


11. On the Scratch window you will see the sensor values. When the arrow keys of the keyboard are pressed, the robot will move.

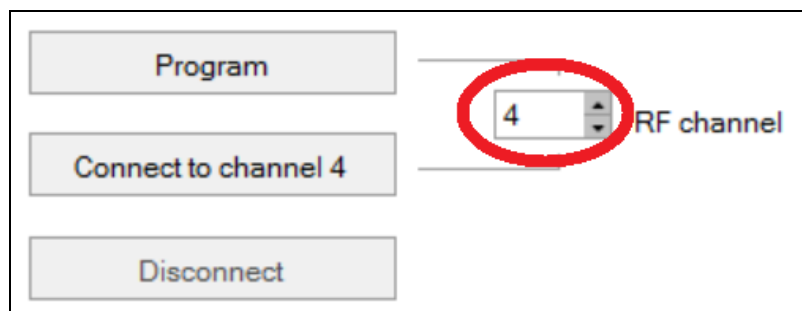


In case several robots are used in different PCs, it is necessary to assign a different channel for each one, in order to avoid interferences between them. These are the steps to follow:

1. If the communication with Scratch has been started, press the “Disconnect” button.



2. Assign a different channel to each robot (in this case, the channel 4 is selected):



3. Reprogram the robot by pressing the “Program” button.
4. Once the robot has been programmed, disconnect it from the PC and switch the robot on.
5. Press the “Connect to channel...” button.

10.4. Commands and Sensors


The actions available to run from the Scratch environment are displayed below, as well as the variables for each action. The command sending to mOway is done by using the control “broadcast” block.



For example the message “rotate(left)” would make a left rotation.

Action	Description	Used variables
go	mOway goes straight forward indefinitely, with a speed of 50%.	-
back	mOway goes straight backward indefinitely, with a speed of 50%.	-
left	mOway turns 90 degrees left.	-
right	mOway turns 90 degrees right.	-
turnaround	mOway turns around.	-
stop	mOway stops.	-
reset(distance)	To reset the total distance counter motors.	-
go(straight)	mOway goes straight at a certain speed for the “distance” distance or “time” time. If “time” and “distance” are same than zero, mOway will move indefinitely.	<div> <div>speed</div> <div>distance</div> <div>time</div> </div>
back(straight)	mOway goes back straight. It's behaviour is identical to “go(straight)” but in the opposite way.	<div> <div>speed</div> <div>distance</div> <div>time</div> </div>

go(left)	mOway moves in a curve to the left of a certain radius at a certain speed for the distance “distance” or the time “time”. If “time” and “distance” are same than zero, mOway will move indefinitely.	<div>speed</div> <div>radius</div> <div>distance</div> <div>time</div>
go(right)	mOway moves in a curve to the right. Identical but in the opposite direction of “go(left)”.	<div>speed</div> <div>radius</div> <div>distance</div> <div>time</div>
back(left)	mOway goes back in a curve to the left. Identical to “go(left)” but going backwards.	<div>speed</div> <div>radius</div> <div>distance</div> <div>time</div>
back(right)	mOway goes back in a curve to the right. Identical but in the opposite direction “back(left)”	<div>speed</div> <div>radius</div> <div>distance</div> <div>time</div>
rotate(left)	mOway rotates at a certain speed in a certain angle to the left, on its axis or on a wheel. If the angle (rotation) is same than zero it rotates indefinitely.	<div>speed</div> <div>rotation</div> <div>rotation-axis</div>
rotate(right)	mOway rotates to the right. Identical but in the opposite direction “rotate(left)”.	<div>speed</div> <div>rotation</div> <div>rotation-axis</div>
frontled(on)	Activates the front LED.	-
frontled(off)	Deactivates the front LED.	-
frontled(blink)	Blinks the front LED.	-
brakeled(on)	Activates the brake LED.	-
brakeled(off)	Deactivates the brake LED.	-
brakeled(blink)	Blinks the brake LED.	-
greenled(on)	Activates the green LED.	-
greenled(off)	Deactivates the green LED.	-
greenled(blink)	Blinks the green LED.	-
redled(on)	Activates the red top LED.	-
redled(off)	Deactivates the red top LED.	-
redled(blink)	Blinks the red top LED.	-
leds(on)	Activates all the LEDs.	-

leds(off)	Deactivates all the LEDs.	-
leds(blink)	Blinks all the LEDs.	-
buzzer(on)	Activates the buzzer with a sound in a certain frequency.	
buzzer(off)	To deactivate the buzzer.	-
melody(charge)	Sounds a melody of “charge!”.	-
melody(fail)	Sounds a melody of “failure”.	-
enclosed	Subprogram for enclosing mOway into a black circle. It goes forward when it is on a white surface and it turns around when it reaches a black line.	-
push	Subprogram for pushing objects. mOway searches the objects and pushes them.	-
defender	Subprogram for pushing objects out of a black circle. It puts together the “enclosed” and “push” subprograms.	-
linefollow(left)	Subprogram for following a line on the left side.	-
linefollow(right)	Subprogram for following a line on the right side.	-

The following table shows the variables and the values can be taken for the correct execution of the commands:

Variable	Description	Range of values
distance	Distance to cover in mm	0 – 255 mm (0 – 25,5 cm)
frequency	Signal frequency speaker	0 – 16000 Hz
radius	Radius of curvature	0 – 100 (radius + speed < 100)
rotation	Rotation angle in degrees	0 – 360 °
rotation-axis	Rotation axis	Wheel – over a wheel Any value – over the center
speed	Movement speed	30 – 100 %
time	Movement time in tenths of a second	0 – 255 (0 - 25,5 seconds)

The following table shows the mOway variables we can read in Scratch using the “sensor value” block.

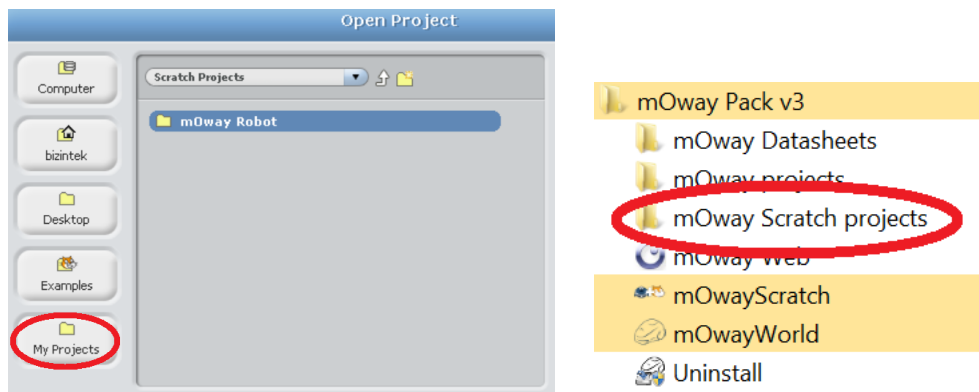


For example the previous block indicates the remaining battery level of mOway.

Variable	Description	Range of values
Obstacle Side Left	Obstacle Side Left Sensor	0 – 255
Obstacle Center Left	Obstacle Center Left	0 – 255
Obstacle Center Right	Obstacle Center Right Sensor	0 – 255
Obstacle Side Right	Obstacle Side Right Sensor	0 – 255
Line Left	Line left Sensor	0 (white) – 255 (black)
Line Right	Line right Sensor	0 (white) – 255 (black)
Light	Ambient Light Sensor	0 – 100%
Distance	Km counter in mm	-
Microphone	Noise level	0 – 255
X-Axis g	Acceleration in g in the axis X	-2.0 to 2.0 g
Y-Axis g	Acceleration in g in the axis Y	-2.0 to 2.0 g
Z-Axis g	Acceleration in g in the axis Z	-2.0 to 2.0 g
Motor End	Activated sensor when the last command sent to the motor has finished.	0 – 1

10.5. Exercises

For a better understanding of the mOway integration within the MowayWorld installer we include five sample exercises. By default they are located in this the Scratch projects folder: “*My Projects-> mOway Robot -> English*”. You can find them also in “*Start -> All programs -> mOway Pack v3 -> mOway Scratch projects*”.

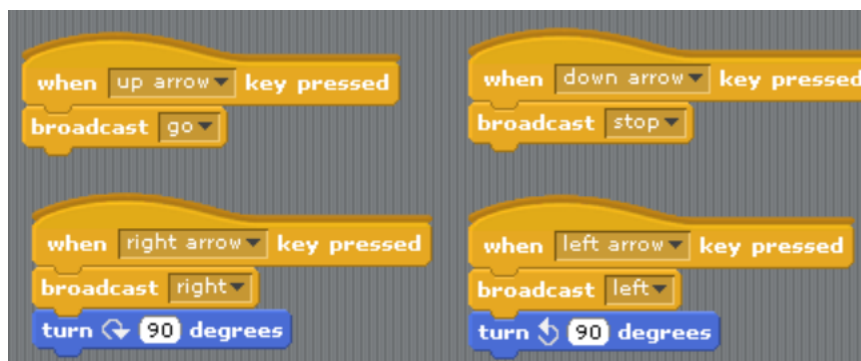


10.5.1. Movement by radio control

Every time we press one of the arrows on the keyboard, Scratch sends a movement command to mOway. For example, when the “up” arrow is pressed, the command sent is “forwards”. This makes the robot advance forwards. If the “down” arrow is pressed, the robot stops.



Similarly, when the “right” and “left” arrows are pressed, the robot turns 90° to the corresponding side. In this case, the Scratch robot will show the way the real robot turns. To do this, the “turn 90 degrees” blocks are used.



You will see that the Scratch robot copies the movements of the real robot.



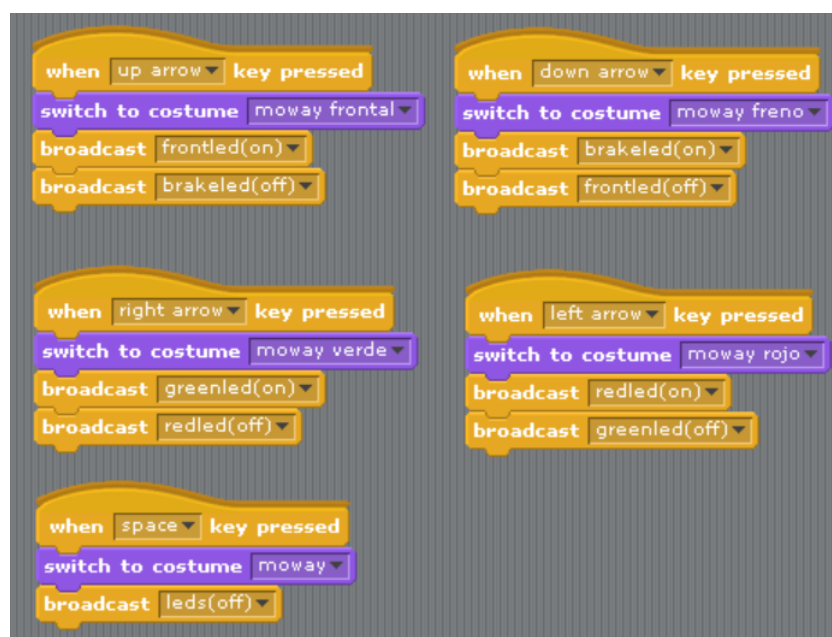
10.5.2. Radio control LEDs

The programme consists of activating the LEDs every time the corresponding key is pressed. The “up” and “down” arrows control the front and brake LEDs. The “left” and “right” arrows control the green and red LEDs.

For example, when the “up” key is pressed, the first thing it does is to change the drawing of the robot in which the front LED is on. Then, the command to turn on the front LED is sent to the real robot. Next, the command to turn off the brake LED is sent.



N.B.: At first sight, the last of these commands does not seem to be necessary, but it is useful if we have pressed the "down" arrow beforehand and the brake LED was on. In this way we ensure that every time we press the “up” and “down” arrows only one LED is left on.



The part of the programme that controls the red and green LEDs is very similar to the one explained above. Lastly, when pressing the “space” key, the robot drawing returns to what it was like initially and all the robot’s LEDs turn off.

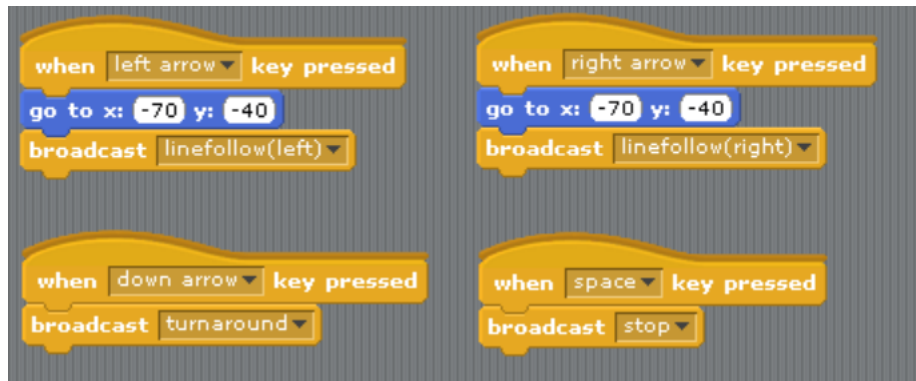


10.5.3. *Follow the line*

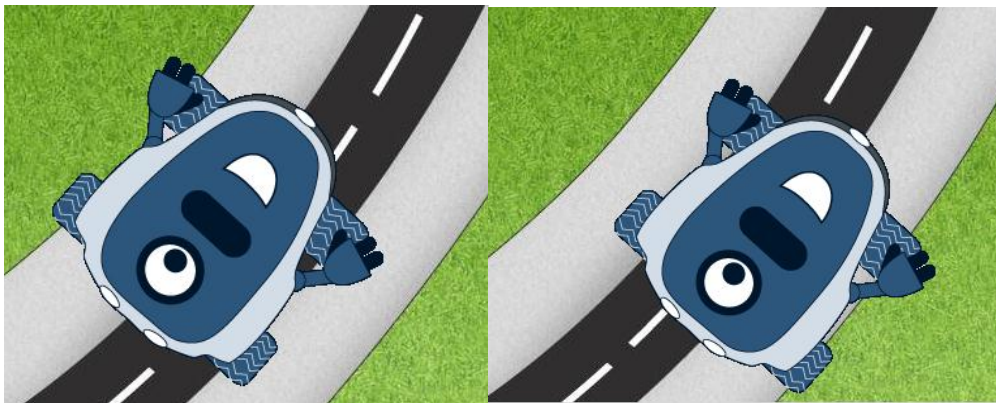
In this exercise we are going to practise the mOway “followline(left)” and “followline(right)” commands. To do this, we will need to use a track on which we will paint a black line on a white background. A number of other tracks can be downloaded from the Club mOway section at <http://moway-robot.com>.



In order to familiarise ourselves with the “followline” commands, the proposed exercise is a simple radio control which gradually changes from a “**followline(left)**” command to a “**followline(right)**” command using the direction keys. The space key stops the mOway and the back arrow sends a “turnaround” command.



In this scenario, the representation of mOway moves to the left side or right side of the line with the command change.



“followline(left)”

“followline(right)”



MOWAY

Title: mOway User Manual
Rev: v3.1.2 – April 2013
Page 175 of 175

