Sense C Coding with CARM-202

NeuLog

Sense C Coding with CARM-202

1_13

For further information contact info@neulog.com

Contents

Chapter 1 – Control and Robots

1.1 Robots

The world today is a world of embedded computer systems. We find them in media systems, watches, phones, remote control, cars, and many more electronics. A few years ago, we did not see terms such as 'wearable computing' or 'internet of things'.

Everyday a surprising new product or application appears and months later, we cannot realize how we lived without it. Modern systems are based more and more on machine learning and artificial intelligence.

The robotic systems, part of the embedded computer system, perform independent activities like search, manipulation, identification, activation, protection and so on.

Many systems combine a certain kind of artificial intelligence in operating and communication between machines.

The robotic system includes the controller, building components, wheels, gears, motors, sensors, and more.

Each robotic system includes a controller that allows it to operate in accordance with different operating programs. The robot developer writes these programs on a computer and forwards them to the controller.

Building a robotic system creates a challenge to acquire knowledge in various technology areas (electronics, computers, mechanics, electricity, etc.).

There are many types of robots such as arm robots, mobile robots, walking robots and more.

The SENSE robots are a series of robots and "brain" units for study, programming and making robots with wide variety of robot applications.

The sense autonomous is a robot which enables us to program many robot applications and functions such as movement on a line, movement along walls, tracking, AGV (Automatic Guided Vehicle), autonomic car, autonomic guard vehicle, autonomic taxi driver, environment monitoring, car manipulation and more. All these applications are described as exercises in this book.

.

1.2 Control systems

A robot is a computerized control system.

A "Control system" may be defined as a group of components, which can be operated together to control multiple variables, which govern the behavior of the system.

Examples:

- Air-conditioning systems control the temperature in the room.
- A greenhouse control system controls temperature, humidity, light, and irrigation.
- A speed control system maintains a steady motor speed regardless of the changing load on the motor.

A light control system can maintain a steady level of light, regardless of the amount of available sunlight. The control system turns lamps ON or OFF according to the requirements.

Three basic units are in every computerized control system:

1. **Input unit** – the unit that reads the system sensors like temperature, light, distance, touch switch, etc. and feeds information into the control unit.

2. **Control unit** – the "BRAIN" of the control system, which contains the system program in its memory and performs the program instructions and processes the received data.

3. **Output unit** – the unit that operates the system actuators such as motors, lamps, pump, and fan as the results of the inputs and the program "decisions".

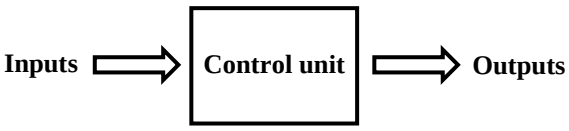


Figure 1-1

The control unit is connected to a computer for programming and downloads a program to the control unit flash memory.

Disconnecting the control unit from the computer and connecting a power source such as a battery to it will create an independent system

.

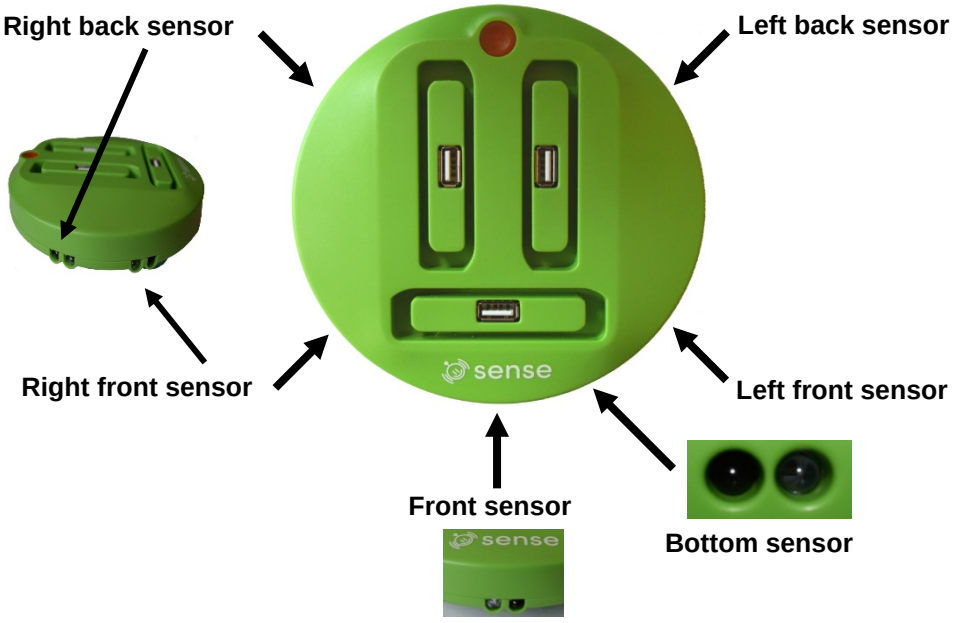
1.3 SENSE autonomous

SENSE autonomous is a mobile robot for applications such as:

- Movement along black line or white line.
- Movement along walls or in a labyrinth.
- **Autonomous vehicle** such as: AGV, autonomous car, autonomous guard vehicle, autonomous taxi driver, autonomous manipulator.
- Following a moving body holding IR transmitter using tracking module.
- **Environmental monitoring** and measurement robot with NeuLog sensors.

The **SENSE autonomous** has the following built in**:**

- Base unit
- 3 connectors for NeuLog sensors or brain units
- 5 IR range sensors
- 1 line sensor
- Pivot wheel
- 2 motors with wheels
- A controller for the base sensors, motors, and independent operation
- A flash memory for the user programs
- USB connector for connection to PC or MAC



The sense autonomous comes with an adapter for external battery. Such battery can be a standard Power Bank with USB outlet.

You may also have the NeuLog battery module BAT-202, which can be plugged directly into one of the SENSE sockets.

When connecting such battery to the sense autonomous and disconnecting it from the PC, the sense autonomous becomes an independent robot running on its internal program in its flash memory.

In this book, we shall call the SENSE autonomous in short SENSE.

1.4 CARM-202 C coding unit

The CARM-202 is a C coding unit of the Sense and Neulog series. It is based on the ARM Cortex M3 microcontroller. This microcontroller belongs to the ARM family, which is the leading family of microprocessors and microcontrollers in the world.

CARM-202 is a C language coding unit with 8 switches and 8 LEDs housed in a rigid plastic packaging and colored label.

CARM-202 can be also used as a stand-alone module for ARM microcontroller and for C language programming.

The module has two connectors for communication with NeuLog sensors or with brain I/O units. The module includes flash memory for programs.

The CARM-202 can be powered by the NeuLog battery module or by a USB power source.

Plugging CARM-202 into one of the SENSE sockets turns the SENSE to be its slave and controlled by it.

1.5 C language

C is a coding language for creating machine programs. These machine programs are fast and work directly with the system hardware components and not through interpreters as the programs above do.

Because its efficiency and simplicity, this language has become popular for developing software for microprocessors and microcontroller embedded systems (or for short, embedded systems).

The book **'C coding with CARM-202'** describes and exercises all about programming in C language with CARM-202.

It is important to exercise the **'C coding with CARM-202'** book (the first experiment at least) before exercising the experiments in this book.

Experiment 1.1 – Serial Communication

Objectives:

- Classification of communication methods.
- Serial asynchronous communication.
- UART and USART.
- ASCII code.
- Communication with PC.

Equipment required:

- Computer
- CARM-202 C coding unit

Discussion:

1.1.1 Classification of communication methods

In communication between computers, the computers are connected to each other by communication lines. At each stage of the communication, there is a transmitting computer and a receiving computer. The transmitter transmits information through an output port and the receiver receives that information through an input port.

It is possible for a transmitting computer to transmit and then switch into receiving condition and vice versa. No computer can "see" what is happening in the other computer. Computers can only read information, which is placed on their input ports. That is the reason why a part of the transferred information consists of signals concerning the status of the transmitting and the receiving computer, signals such as: "ready to receive", "receive a message", "end of message" etc.

The various communication methods are classified in three basic groups:

a) <u>Synchronous and asynchronous:</u>

In synchronous communication, the computers are connected to a mutual line, which supplies synchronization signals to them both. The synchronization signal enables the computers to know when to transmit and when to expect a message through the communication lines. Each computer, before transmitting a message, awaits the appearance of the synchronization signal, and only then starts transmitting. A computer, which is due to receive a message, awaits the appearance of the synchronization signal and only then collects the information from its input port lines.

In asynchronous communication, we circumvent the use of a synchronization pulse line and a pulse generator. On the information lines, we transmit a start signal at the beginning of each message. The receiving computer awaits the reception of such a signal. After locating it, the receiving computer collects the message, which follows that signal. This method of communication is the most commonly used.

b) Parallel and serial:

In parallel communication, we transmit the information in parallel form. A byte of 8 bits is transmitted through a cable of 8 wires. Each bit is transmitted on a separate wire simultaneously. This method requires a cable with a large number of wires.

In serial communication, we use a small number of wires. The byte is transmitted through one line, bit by bit. The transmitter and receiver must both be synchronized to the same communications frequency.

c) Polling or interrupts:

The problem in communication is in recognizing when the dialogue begins. One of the methods to overcome this obstacle is to determine one of the computers as "MASTER" and the others as "SLAVES". The master always initiates the communication. It turns to the slave and asks whether it has any information to transmit. It waits a certain time to receive a message from the slave. If the slave does not answer within that time, then the master returns to its main program.

The above procedure is performed at pre-determined regular intervals. When the slave has a message to transmit, it waits for the master to turn to it and when this happens, the slave answers by transmitting an opening message. The master reacts and the dialogue takes place. This method is called "communication by polling".

Another method to start a conversation is by interrupts. We use input ports with a strobe line (STB). When one computer wishes to talk to another, it sends a message on its own output port, together with a strobe pulse. An input port collects the message and performs an interrupt request in the receiving computer. The receiver executes the interrupt program, which handles the received message.

This method is quick and convenient although it requires the use of adequate ports and interrupt programs.

Another expression in communication is "handshake". This means that the transmitter of a message awaits an acknowledgement of it reception by the receiver. Without such acknowledgement, the transmitter does not continue with the program.

1.1.2 Serial asynchronous communication

This is the most popular method of communication in microcomputer systems. In this method, the communication line is minimal and may consist of two or three wires only. It is possible to transmit and to receive through telephone wires (with the help of an interface unit called a modem) and even through a wireless connection.

Serial communication is a method in which a byte of 8 bits is translated into a series of serial pulses, zeros and ones, which are transmitted through the communication line. The receiver knows the length of time of each pulse transmitted by the transmitter. In serial asynchronous communication, there is a problem in identifying the start of each byte. The following procedure was therefore determined.

Start bit:

The normal status of the line is "high". Before each byte which is transmitted in a serial form, a '0' bit should be transmitted for the same period of time which is required for the transmission of each of the other bits. This is called "start bit". The receiver identifies the beginning of the transmission of a character by identifying the transition from '1' to '0'.

Data bits:

At the end of the transmission of the start bit, the data bits are transmitted, one after the other. The transmission time of each bit is equal to that of the other bits. Since the receiver knows when the transmission starts, it is able to time the sampling of the data bits in order to overcome the problem of transients.

Parity bit:

Sometimes we use the eighth bit of the data bits as a parity bit, which is used by the receiver to check the accuracy of the data received by it. The value of the bit ('0' or '1') is determined according to the number of 1's in the data byte. There are two ways to determine this: "even parity" and "odd parity". In "even parity", the number of 1's, including the parity bit, should be even. For example, if there are three 1's in a byte, then the transmitter determines the parity bit as '1'. If there are four 1's, then the parity bit will be '0'.

In "odd parity", the number of 1's, including the parity bit, should be odd.

Stop bits:

At the end of each byte, bits of logical 1's are transmitted (usually 2). These bits are used to transfer the line to its normal status for a period of time, which enables the receiver to perform primary processing of the information collected by it, and to resynchronize on the beginning of the transmission of the next character.

The transmission of a single character (58H) will be as follows:

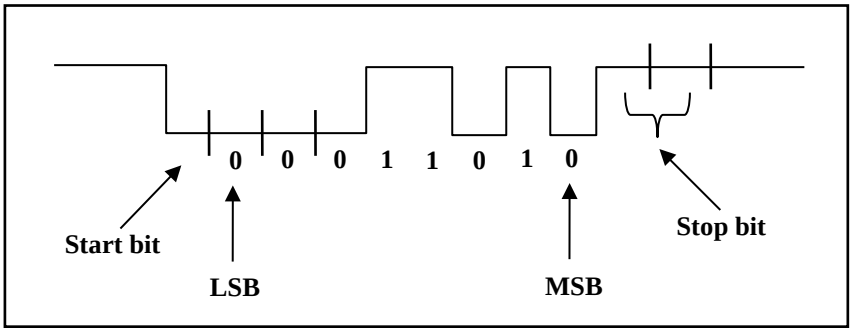


Figure 1.1-1 Transmission of the character 58H in asynchronic serial communication

The transmission rate is measured in units of baud, which are bits transmitted in a second. A different transmission is "bits per second", whereby we mean the data bits which are transmitted in one second. For example, if we transmit at a rate of 10 characters per second, the baud rate is 110. Each character requires 11 transmission bits for its transmission (including the start and stop bits). This rate is also equal to 80 bits per second (data bits).

To conclude, asynchronous serial communication is as described in following figure:

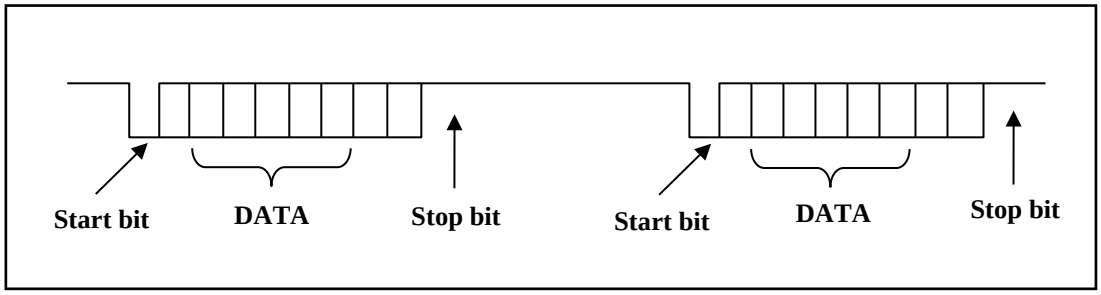


Figure 1.1-2 Description of the transmission of data in serial communication

The transmitting computer converts a character from its parallel form (as a binary number) into serial form, and then transmits it. The receiving computer translates it back from serial form into parallel form.

It is necessary for the receiver to know the transmission rate and the number of data bits in the transmitted byte (which is not always seven). It also has to know whether the eighth bit indicates even or odd parity or is insignificant, and the number of stop bits.

In communication between computers - one computer transmits and the other receives. Usually, both computers have the capability of transmitting as well as of receiving. Each computer has a TD (Transmit Data) output and a RD (Receive Data) input.

In communication, there are two major forms of connection. One is called: "Full duplex Communication".

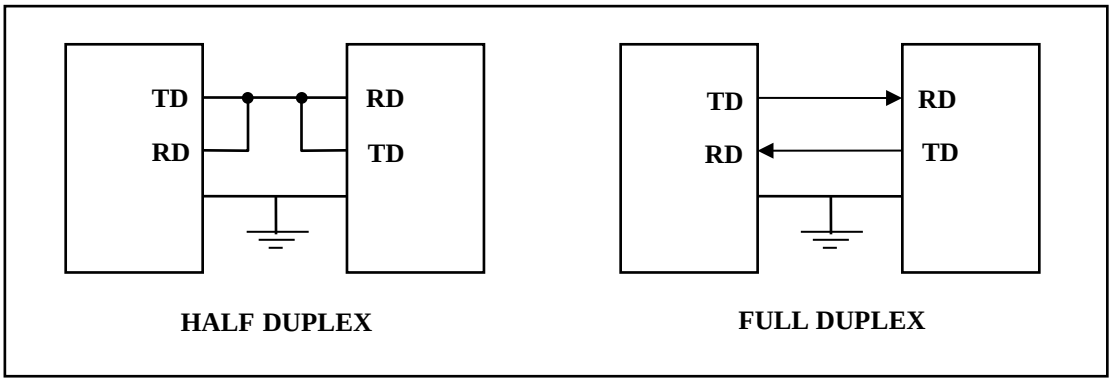


Figure 1.1-3 The forms of connection in communication

The second form of connection is called: "Half duplex Communication", and it uses only two connecting wires. In half duplex communication, a computer, which changes from receiving into transmitting status must ensure that the other computer has finished transmitting and that it has cleared the line.

Usually, at the input and in the output of a communication line, there are driving components, which enable transmission of the signals over long distances. There are different methods of connection between computers. The most popular are RS232, RS422 and 20 mA current loop.

The process of receiving the serial information and of its conversion into parallel form operates in the following manner: The receiving computer samples the RD input line and awaits the start bit, i.e. the sinking of the line to '0'. As soon as it notices this transition, it waits for a period equaling half a bit time, and then samples the line again. If the line is still '0', that means that the start bit has been received. Now it samples the line at intervals of one bit, according to the number of data bits.

While sampling, the bits are pushed one after the other (LSB is being received first) into a shift register. At the end of the process, the shift register contains the transmitted byte, which is readable in parallel form.

This process is performed with the help of a hardware device called a UART – Universal Asynchronous Receiver Transmitter.

When the UART identifies the START bit, it collects all the DATA bits into a certain buffer register and then creates an interrupt request to tell the CPU that a byte is waiting in the buffer register.

Some UARTs can work also in synchronous mode, which means, starting collecting data only after receiving a certain byte or word. They are called USART.

1.1.3 ASCII code

The ASCII code is a standard, international code used for the exchange of information between input and output units (like the various types of printers, keyboards, external memories) and the computer, as well as between computers. The name ASCII stands for: American Standard Code for Information Interchange.

Each character (letter, digit or other symbol) has been given an agreed upon binary number, by which it is represented in ASCII. For instance, if we want a printer to print the letter A, it must be fed with the binary number 01000001 or 41_{16}.

Following is a table with the various characters and their ASCII code where the ASCII code is expressed in binary, hexadecimal and decimal form.

The first 32 numbers (0-31) are used as special codes for the dialog between the computers like: Start Of Message (SOM), End Of Text (EOT), Carriage Return (CR), Line Feed (LF), etc.

Dec.	Hex.	Binary	Char.
32	20	00100000	SPACE
33	21	00100001	!
34	22	00100010	"
35	23	00100011	#
36	24	00100100	$
37	25	00100101	%
38	26	00100110	&
39	27	00100111	'
40	28	00101000	(
41	29	00101001	)
42	2A	00101010	*
43	2B	00101011	+
44	2C	00101100	,
45	2D	00101101	-
46	2E	00101110	.
47	2F	00101111	/
48	30	00110000	0
49	31	00110001	1
50	32	00110010	2
51	33	00110011	3
52	34	00110100	4
53	35	00110101	5
54	36	00110110	6
55	37	00110111	7
56	38	00111000	8
57	39	00111001	9
58	3A	00111010	:
59	3B	00111011	;
60	3C	00111100	<
61	3D	00111101	=
62	3E	00111110	>
63	3F	00111111	?

Dec.	Hex.	Binary	Char.
64	40	01000000	@
65	41	01000001	A
66	42	01000010	B
67	43	01000011	C
68	44	01000100	D
69	45	01000101	E
70	46	01000110	F
71	47	01000111	G
72	48	01001000	H
73	49	01001001	I
74	4A	01001010	J
75	4B	01001011	K
76	4C	01001100	L
77	4D	01001101	M
78	4E	01001110	N
79	4F	01001111	O
80	50	01010000	P
81	51	01010001	Q
82	52	01010010	R
83	53	01010011	S
84	54	01010100	T
85	55	01010101	U
86	56	01010110	V
87	57	01010111	W
88	58	01011000	X
89	59	01011001	Y
90	5A	01011010	Z
91	5B	01011011	[
92	5C	01011100	↓
93	5D	01011101	←
94	5E	01011110	→
95	5F	01011111	_

1.1.4 Communication with PC

The CARM-202 microcontroller has two USARTS. One is for the communication with the PC (USART2) and one for the communication with the SENSE and the NeuLog modules.

The following program outputs to the LEDs the ASCII number of the character received from the PC.

```
NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);

USART2_Interupt_Init();
__enable_irq(); //  (Enable Interrupts)
Leds_Init ();

while(1)
{
 if(Received_Flag)
  {
   Leds_Out (Received_Char);
   Received_Flag = 0;        //Clear the flag

   if (Received_Char == 'h')
   {
   PC_send( ' ' );
   PC_send( 'H' );
   PC_send( 'e' );
   PC_send( 'l' );
   PC_send( 'l' );
   PC_send( 'o' );
   PC_send( ' ' );
   PC_send( 0x0d );
   PC_send( 0x0a );

   while((USART2->SR & USART_SR_TC) == 0); //Wait until the Transmission_Complete
  USART2->SR &= ~USART_SR_TC;        //Reset the Transmission Complete flag
   }
  }
}
```

NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);
declares for the CPU where the interrupt routines are located in the memory. This table is called NVIC (Nested Vector Interrupt Control) table.

USART2_Interupt_Init(); initializes the USART2 the interrupt routine.

Interrupt routine should be enabled when we use them and the following instruction does that.

__enable_irq(); // (Enable Interrupts)
Leds_Init (); initializes the LEDs output routine.

When USART2 interrupt routine receives a character, it raises the flag '**Received_Flag**' and put the character in the '**Received_Char**' variable.

'**Received_Flag**' must be zeroed after reading the '**Received_Char**'.

When the program receives the character '**h**', it sends the word '**Hello**'.
It sends the word '**Hello**' when the **Received_Char = 'h'** (the ASCII code of the letter **h**).

The program uses the **PC_send();** that sends a character to the PC.

The program has to wait for the end of the transmission and to clear the '**Transmission Flag**' before proceeding.

The last two instructions do that.

Another option to send words. is to send them as a string with the function **Print_to_pc_msg (char msg[]);**.

This function sends the string between the brackets character after character to the PC.

The following program uses this function and send the string "Hello Robot" when the letter 'R' is pressed.

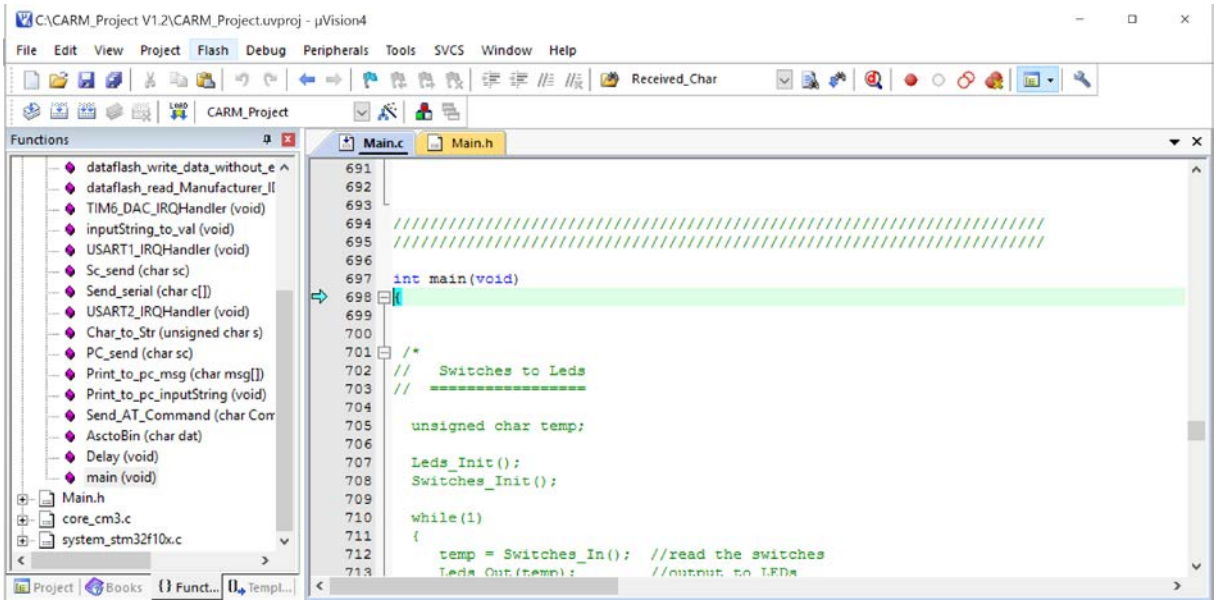
```
NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);

USART2_Interupt_Init();
 __enable_irq(); //  (Enable Interrupts)
Leds_Init ();

  while(1)
  {
   if(Received_Flag)
   {
    Leds_Out (Received_Char);
    Received_Flag = 0;        //Clear the flag
    if (Received_Char == 'R')
    {
     Print_to_pc_msg ( " Hello Robot " );
     PC_send( 0x0d );
     PC_send( 0x0a );

    while((USART2->SR & USART_SR_TC) == 0); //Wait until the Transmission_Complete
   USART2->SR &= ~USART_SR_TC;        //Reset the Transmission Complete flag
    }
   }
 }
```

Sending CR and LF are not included in this function and must be sent separately when needed.

Procedure:

1. Enter the CARM_Project library and double click on the file **CARM_Project.uvproj**.

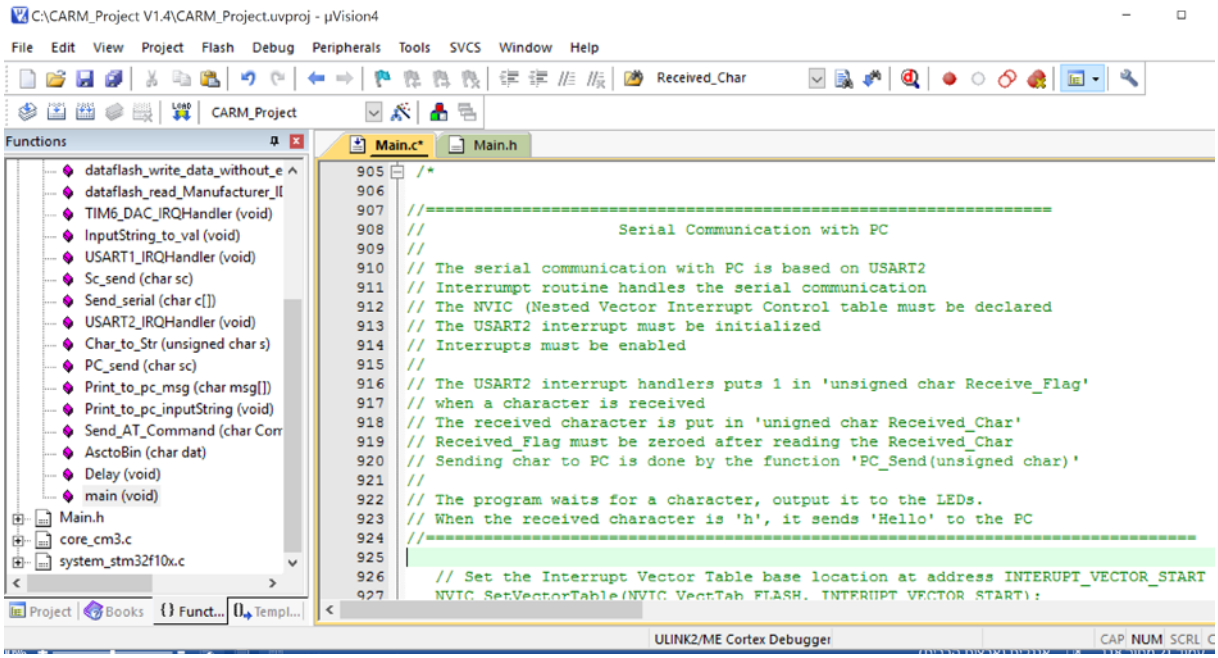
2. Check that **main.c** and **main.h** are open as in the following screen.



Observe the **main()** program.

It starts with the **Switches to LEDs** program.

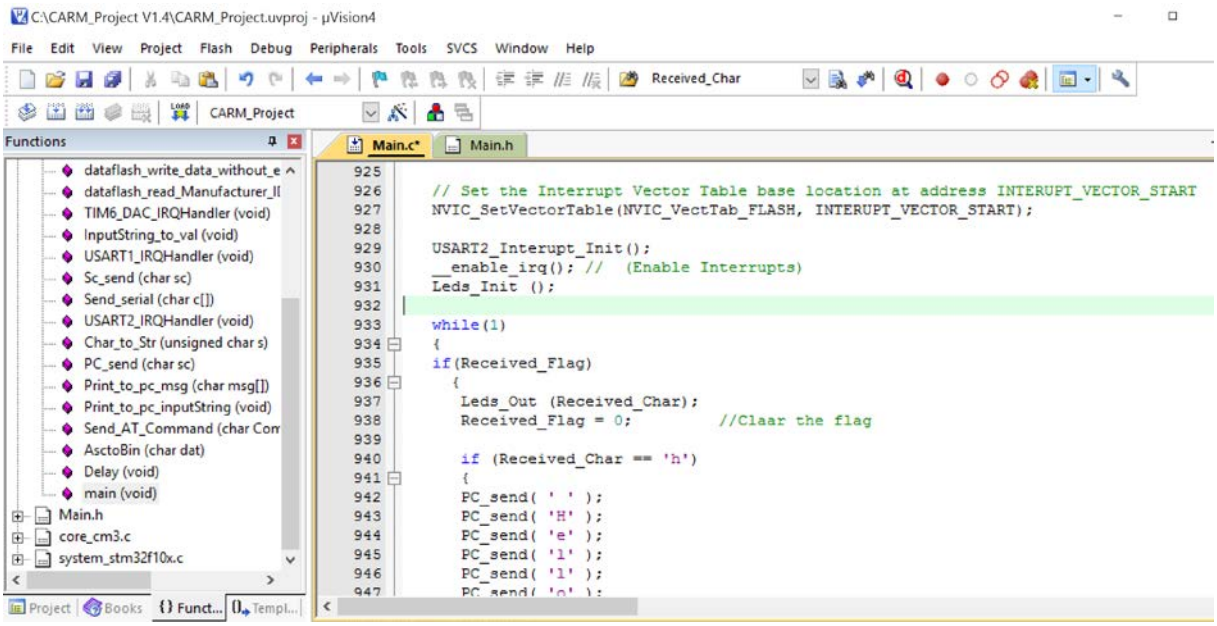
3. Scroll down the **main()** program until you get the following screen:



This section contains the **Serial Communication with PC** program.

4. Activate the program by making the two remark limit signs to remark lines by adding two slashes at the beginning of each line.

   Scroll down over the preface of the program and you get the following screen:



   Observe the program and compare it with the exercise program.

```
int main (void)
{
  NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);

  USART2_Interupt_Init();
  __enable_irq(); //  (Enable Interrupts)
  Leds_Init ();

  while(1)
  {
   if(Received_Flag)
    {
     Leds_Out (Received_Char);
     Received_Flag = 0;        //Clear the flag

     if (Received_Char == 'h')
     {
     PC_send( ' ' );
     PC_send( 'H' );
     PC_send( 'e' );
     PC_send( 'l' );
     PC_send( 'l' );
     PC_send( 'o' );
     PC_send( ' ' );
     PC_send( 0x0d );
     PC_send( 0x0a );

     while((USART2->SR & USART_SR_TC) == 0); //Wait until the Transmission_Complete flag (in status register) will be set
     USART2->SR &= ~USART_SR_TC;             //Reset the Transmission Complete flag
     }
   }
  }
}
```

Note the way the brackets are written.

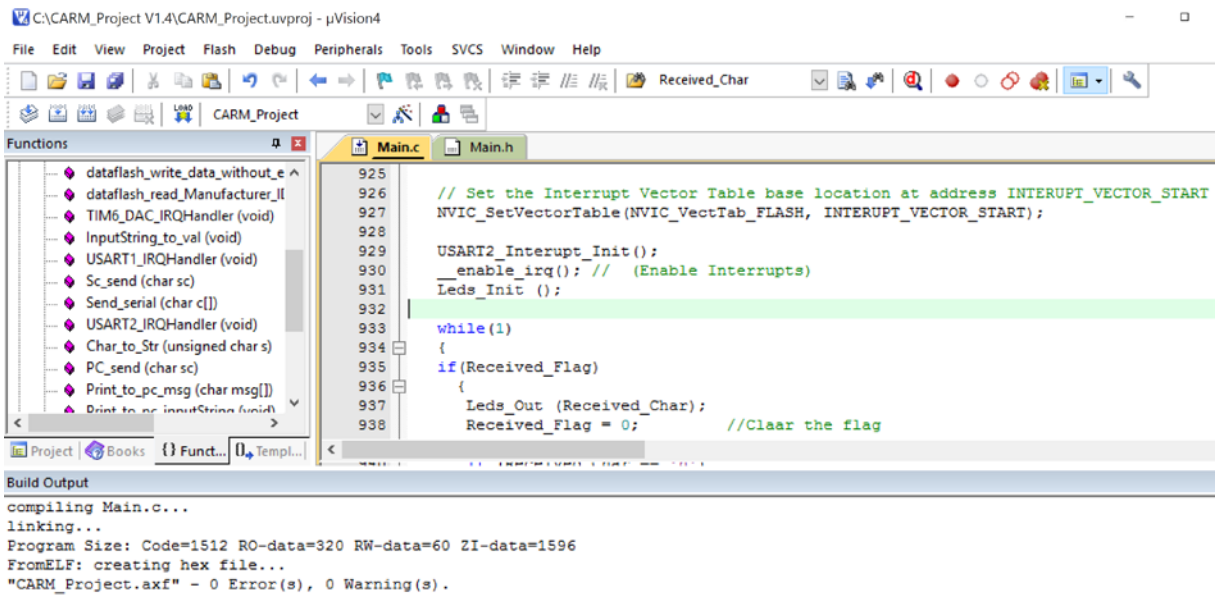
Spacing is important in order to avoid mistakes (it is not important to the compiler).

5. Save the program by clicking on the **Save** icon  .

6. Activate the compiler by clicking on the **Rebuild** icon  .

   The C compiler and linker will be activated and perform compilation, location and conversion from your file to HEX file.

7. The window presenting the program's operation and results is shown on the bottom of the screen.



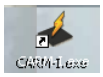
Check that there are no errors and no warnings.

8. If there are errors, repair the errors and repeat steps 5 and 6 again.

9. Connect the CARM-202 module to the PC by the USB communication cable.

10. A special interface program was prepared for downloading and running the object program in HEX format.

    Find the **CARM-I** program in the CARM exercises directory and double click on it.

    You can also click on its icon  on the desktop.

    The following screen will appear.

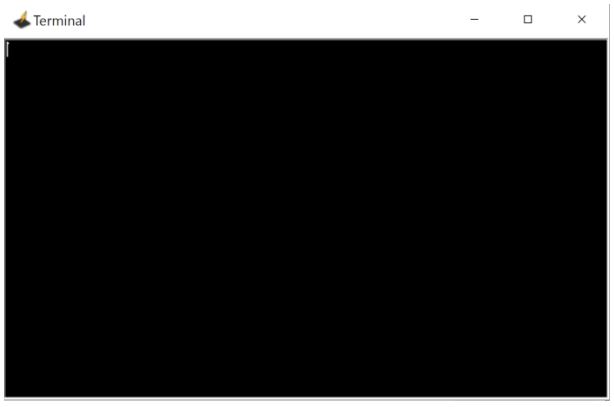
    The message **"Module is connected"** should appear.

11. Click on the **Browse**  button and use the browser to find the file **CARM_Project.hex**.

    Click on this file.

12. In order to communicate with the PC, we have to turn the PC into terminal. The terminal transmits the ASCII code of any pressed key and display the received characters.

    Click on the **Open Terminal**  button and you will get the following screen.



13. Download and run the program by clicking on the **Download**  button.

    The program downloaded into CARM-202.

14. Click on the **Run**  button to run the program.

15. Click on the terminal screen to see the blinking cursor there.

16. Click on the '**a**' key.

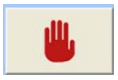
    This letter should appear on the screen.

    The '**a**' binary ASCII code (**01100001**) should appear on the LED display.

    Check that.

17. Click on other keys and see their effect on the screen and on the LEDs.

18. Click on the '**h**' key and you will get the message '**Hello**' on the screen.

19. Press the **Stop**  to stop the program's running.

20. The program is still in the CARM-202 module flash memory.

    Click on the **Run**  button to run it without downloading.

21. Click on keys and see their effect on the screen and on the LEDs.

22. Click on the '**h**' key and you will get the message '**Hello**' on the screen.

23. Press the **Stop**  to stop the program's running.

24. Change the program so it sends the word ' Robot ' when the key 'R' is pressed.

25. Save the program.

26. Compile the program and check for errors.

27. Download the program by clicking on the **Download**  button.

    You do not have to select the file **CARM_Project.hex**.

28. Click on the **Run**  button to run the program.

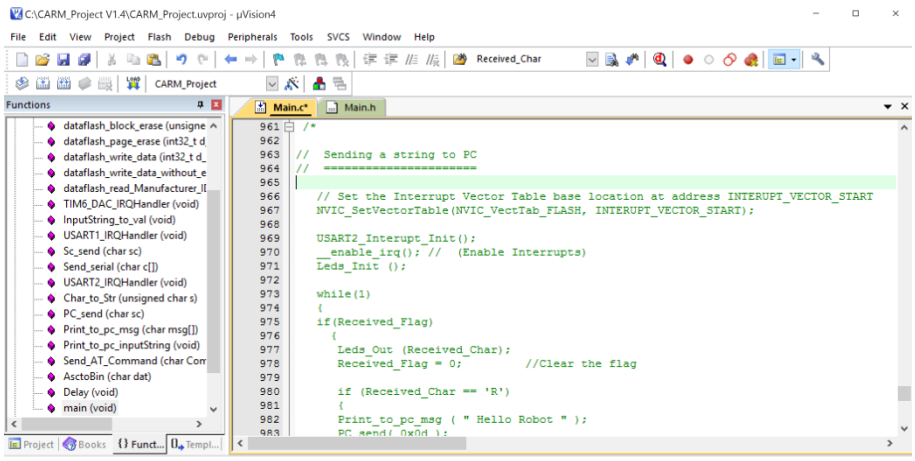
29. Press various keys including 'R' and check the program behavior.

30. Press the **Stop**  to stop the program's running.

31. Activate the **/*** ***/** signs by deleting the two slashes at the beginning of each line.

    The program turns to green.

32. Scroll down until you get the following screen:



    This section contains the **Sending a string to PC** program.

33. Activate the program by making the two remark limit signs to remark lines by adding two slashes at the beginning of each line.

34. Observe the program and compare it with the exercise program.

```
int main (void)
{
 NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);

 USART2_Interupt_Init();
  __enable_irq(); //  (Enable Interrupts)
 Leds_Init ();

   while(1)
   {
    if(Received_Flag)
    {
     Leds_Out (Received_Char);
     Received_Flag = 0;        //Clear the flag
     if (Received_Char == 'R')
     {
      Print_to_pc_msg ( " Hello Robot " );
      PC_send( 0x0d );
      PC_send( 0x0a );

     while((USART2->SR & USART_SR_TC) == 0); //Wait until the
                                    Transmission_Complete flag (in status register) will be set
    USART2->SR &= ~USART_SR_TC;        //Reset the Transmission Complete flag
     }
    }
   }
}
```

35. Save the program.

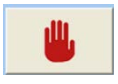
36. Compile the program and check for errors.

37. Download the program by clicking on the **Download**  button.

    You do not have to select the file **CARM_Project.hex**.

38. Click on the **Run**  button to run the program.

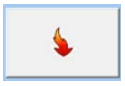
39. Press various keys including 'R' and check the program behavior.

40. Press the **Stop**  to stop the program's running.

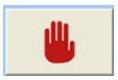
41. Use the '**Switch – Case**' instruction and creates a program that sends to the terminal a Day name when its initial letter is pressed.

42. Save the program.

43. Compile the program and check for errors.

44. Download and run the program by clicking on the **Download and Run**  button.

45. Press various keys and check the program behavior.

46. Press the **Stop**  to stop the program's running.

47. Activate the **/*** ***/** signs by deleting the two slashes at the beginning of each line.

    The program turns to green.

Experiment 1.2 – Communication with SENSE

Objectives:

- Sending commands to SENSE.
- Forward and backward.
- Changing speed.
- Turning right and left.

Equipment required:

- Computer
- SENSE autonomous
- CARM-202 C coding unit
- BAT-202 battery module

Discussion:

The previous experiment described how to send characters and strings to the PC.

Talking with the SENSE and all other brain units is done by sending strings too. The brains in these units analyze the strings and execute them. Modern systems are built this way.

The communication between CARM-202 and SENSE and brain units is done through USART1.

The commands are built as strings and we use a function that sends this string called: **Send_AT_Command();**

The following string describes the structure of a string to operate the robot motors:

"AT+SetMotor:[robo.type],[robo.ID],[num],[command]"

The square brackets mean a field with options.

robo.type options are:

Sense, **Robo206**, **RoboEx**, **BrainServo**, **BrainMotor**, **BrainArm**, **IRTrack**.

robo.ID is a number from 1 to 9.

num options are:

0 for all motors
1 for M1
2 for M2
3 for M3

command options are:

Off for stop
Cw for clockwise
Ccw for counter clockwise

The following is a command for the SENSE to move forward:

Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Cw]");

Note:
The distinction of upper case and lower case is important.

No spaces should be in the command string.

The following string describes the structure of a string to set the speed of the robot motors:

"AT+SetSpeed:[robo.type],[robo.ID],[num],[speed]"

Speed options are a number from 0 to 255.

The following is a command to set the SENSE motor number 2 speed to 250:

Send_AT_Command("AT+SetSpeed:[Sense],[1],[2],[250]");

Note:
We use the **SetSpeed** command when we want to change the speed of a motor. The robot remembers the last setup speed.

We can set each motor to a different speed.

The following program moves the SENSE forward for 2 seconds and stop.

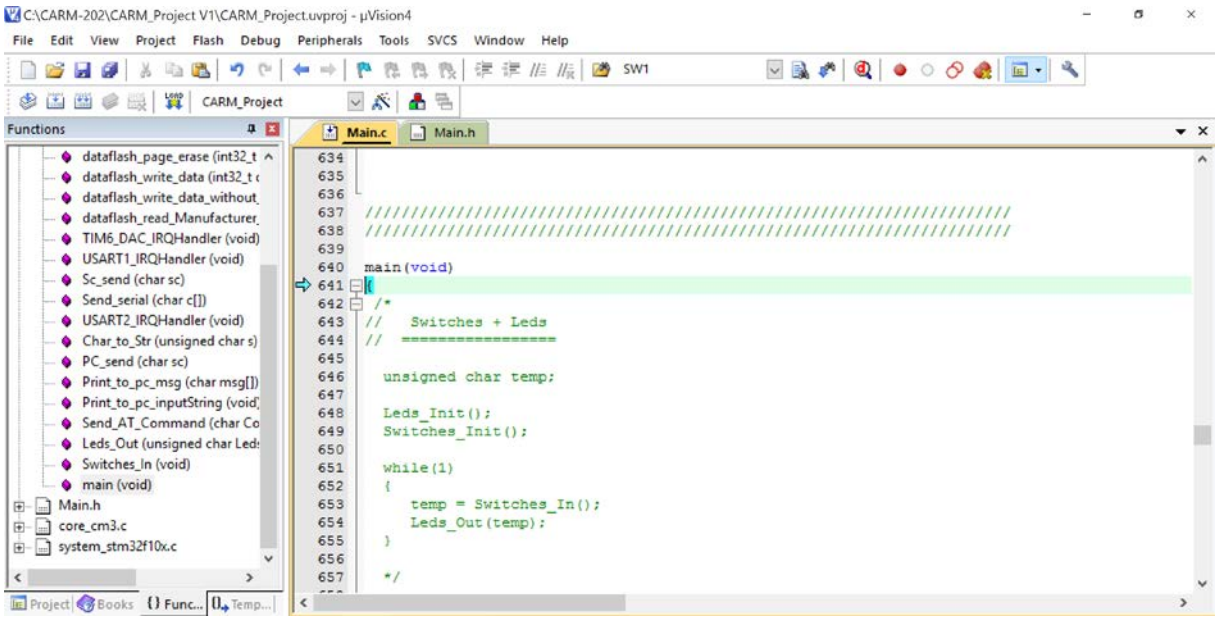
```
main (void)
{
  unsigned int    i;

  NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);
  USART1_Interupt_Init();
 __enable_irq(); //  (Enable Interrupts)

 Send_AT_Command("AT+SetSpeed:[Sense],[1],[0],[200]");     //Sense speed fast
 Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Cw]");     //Sense forward
 for (i = 1 ; i != 8000000 ; i++);
 Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Off]");     //Sense stop

End_of_program();

}
```

The function **End_of_program();** creates software reset to the CARM-202 module. It is like pressing the RST button or sending the STOP command.

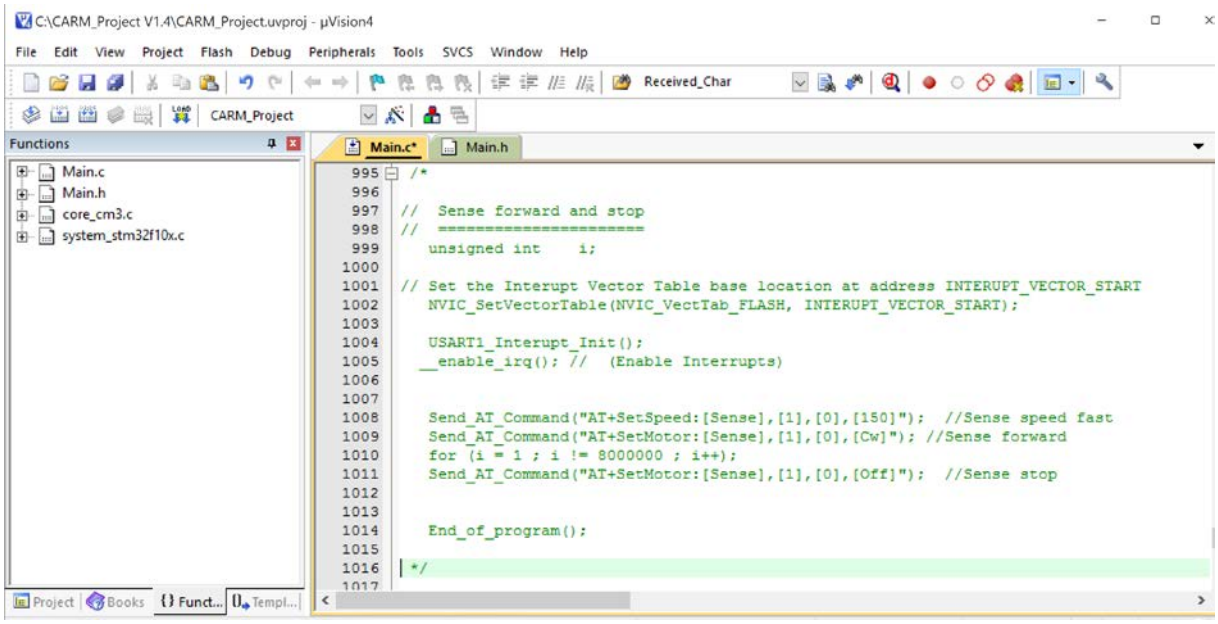
Procedure:

1. Enter the CARM_Project library and double click on the file **CARM_Project.uvproj**.

2. Check that **main.c** and **main.h** are open as in the following screen.

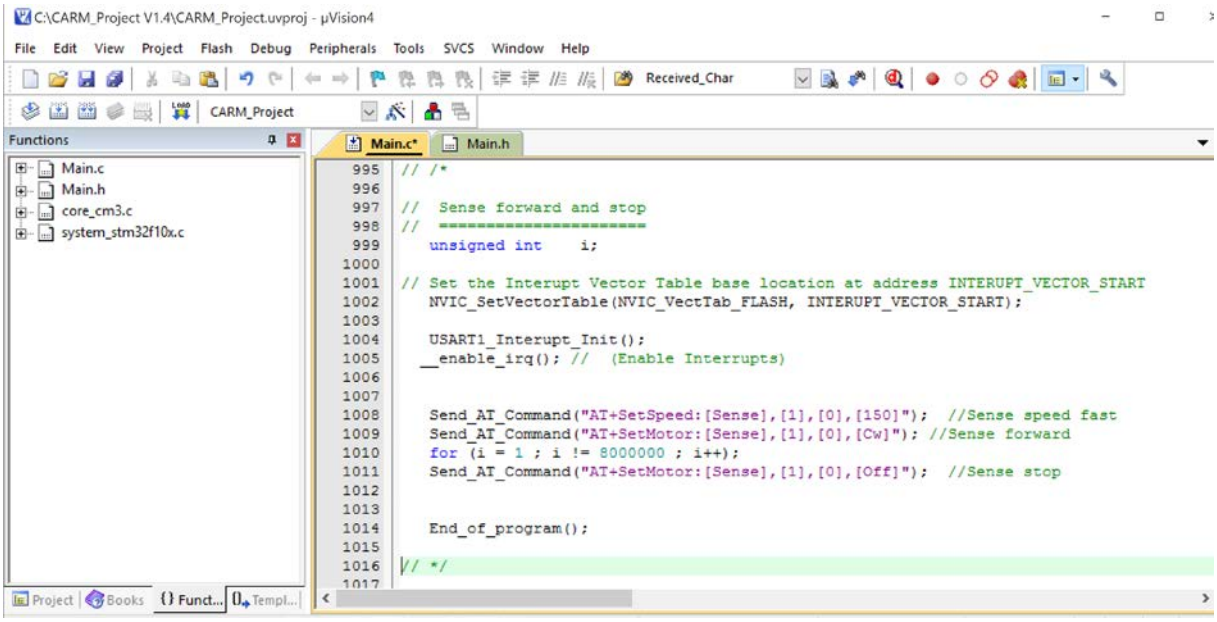


3. Scroll down the **main()** program until you get the following screen:



This section contains the **Sense forward and stop** program.

4. Activate the program by making the two remark limit signs to remark lines by adding two slashes at the beginning of each line and you get the following screen:



1.2.1 Forward and stop

1. Observe the program and compare it with the exercise program

```
int main (void)
{
 unsigned int    i;

 NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);
 USART1_Interupt_Init();
__enable_irq(); //  (Enable Interrupts)

Send_AT_Command("AT+SetSpeed:[Sense],[1],[0],[200]");    //Sense speed fast
Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Cw]");    //Sense forward
for (i = 1 ; i != 8000000 ; i++);
Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Off]");    //Sense stop

End_of_program();

}
```

2. Save the program by clicking on the **Save** icon  .

3. Activate the compiler by clicking on the **Rebuild** icon  .

   Check that there are no errors and no warnings.

4. If there are errors, repair the errors and repeat steps 2 and 3 again.

5. Connect the CARM-202 module to the PC by the USB communication cable.

6. Plug the CARM-202 module into the left socket of the SENSE.

7. Plug the BAT-202 battery module into the right socket of the SENSE.

   It does not matter where we plug the modules.

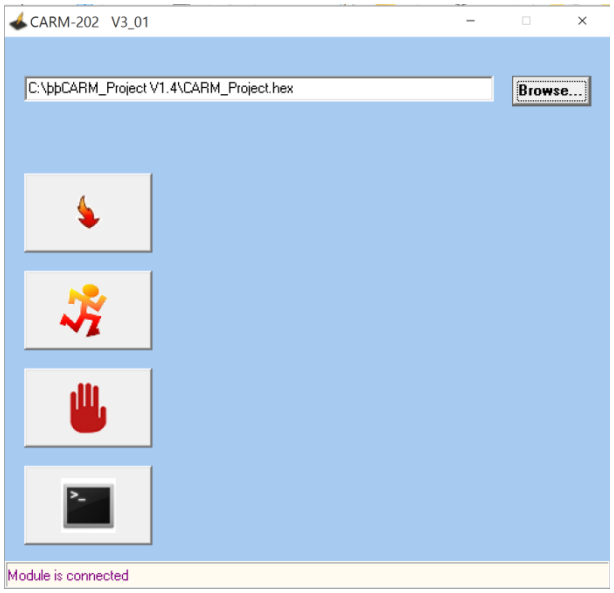
   We do so for the convenience of connecting the communication cable.

   We need the battery module, because it is not recommended too drive the SENSE by the PC USB outlet and most of the computers cannot supply enough power to CARM-202 and to the SENSE.

8. Find the **CARM-I** program in the CARM exercises directory and double click on it.

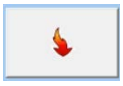
   You can also click on its icon  on the desktop.

   The following screen will appear.



9. Click on the Browse  button and use the browser to find the file **CARM_Project.hex**.

   Click on this file.

10. Download the program by clicking on the **Download**  button.

    The program downloaded into CARM-202 and runs.

11. Click on the **Run**  button to run the program.

    The SENSE should move forward for 2 seconds and stop.

12. Disconnect the communication cable from CARM-202.

13. Put the SENSE on the floor and press the RUN green button on CARM-202 panel to run the program.

    The SENSE should move forward for 2 seconds and stop.

14. Connect the communication cable to CARM-202.

1.2.2 Forward and backward

1. Change the program to the following:

```
int main (void)
{
  unsigned int    i;

  NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);
  USART1_Interupt_Init();
 __enable_irq(); //  (Enable Interrupts)

 Send_AT_Command("AT+SetSpeed:[Sense],[1],[0],[200]");     //Sense speed fast
 Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Cw]");      //Sense forward
 for (i = 1 ; i != 8000000 ; i++);
 Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Off]");     //Sense stop
 for (i = 1 ; i != 1000000 ; i++);                         //Short delay
 Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Ccw]");     //Sense backward
 for (i = 1 ; i != 8000000 ; i++);
 Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Off]");     //Sense stop

End_of_program();

}
```

   It is recommended to add short delay before changing motor direction.

2. Save the program.

3. Compile the program and check for errors.

4. Download the program by clicking on the **Download**  button.

5. Click on the **Run**  button to run the program.

   The SENSE should move forward for 2 seconds, stops, move backward for 2 seconds and stop.

1.2.3 Turning left and right

For turning, we have to address each motor separately.

We have three options:

Deviating – rotating each motor in different speed.
Turning – stopping one motor and rotating the other one
Rotating – rotating one motor forward and the other motor backward

1. Change the program to the following **turning left and right** program:

```
int main (void)
{
  unsigned int    i;

  NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);
  USART1_Interupt_Init();
 __enable_irq(); //  (Enable Interrupts)

 Send_AT_Command("AT+SetSpeed:[Sense],[1],[0],[200]");    //Sense speed fast
 Send_AT_Command("AT+SetMotor:[Sense],[1],[2],[Cw]");     //Turn left
 Send_AT_Command("AT+SetMotor:[Sense],[1],[1],[Off]");    //
 for (i = 1 ; i != 4000000 ; i++);
 Send_AT_Command("AT+SetMotor:[Sense],[1],[2],[Off]");    //Turn right
 Send_AT_Command("AT+SetMotor:[Sense],[1],[1],[Cw]");     //
 for (i = 1 ; i != 4000000 ; i++);
 Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Off]");    //Sense stop

End_of_program();

}
```

2. Save the program.

3. Compile the program and check for errors.

4. Download the program by clicking on the **Download**  button.

5. Click on the **Run**  button to run the program.

   The SENSE turns to the left for 1 second, turns to the right for 1 second, and then stops.

6. Run the SENSE on the floor without the communication cable.

7. Change the delay time for having turns of 90^{o}.

1.2.4 Rotating left and right

1. Change the program to the following **rotating left and right** program:

```
int main (void)
{
 unsigned int    i;

 NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);
 USART1_Interupt_Init();
__enable_irq(); //  (Enable Interrupts)

 Send_AT_Command("AT+SetSpeed:[Sense],[1],[0],[200]");     //Sense speed fast
 Send_AT_Command("AT+SetMotor:[Sense],[1],[2],[Cw]");     //Rotate left
 Send_AT_Command("AT+SetMotor:[Sense],[1],[1],[Ccw]");   //
 for (i = 1 ; i != 4000000 ; i++);
 Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Off]");     //Sense stop
 for (i = 1 ; i != 1000000 ; i++);
 Send_AT_Command("AT+SetMotor:[Sense],[1],[2],[Ccw]");   //Rotate right
 Send_AT_Command("AT+SetMotor:[Sense],[1],[1],[Cw]");     //
 for (i = 1 ; i != 4000000 ; i++);
 Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Off]");     //Sense stop

End_of_program();

}
```

2. Save the program.

3. Compile the program and check for errors.

4. Download the program by clicking on the **Download**  button.

5. Click on the **Run**  button to run the program.

   The SENSE rotates to the left for 1 second, rotates to the right for 1 second, and then stops.

6. Run the SENSE on the floor without the communication cable.

7. Change the delay time for having turns of 90^{o}.

1.2.5 Deviating left and right

1. Change the program to the following **deviating left and right** program:

```
int main (void)
{
 unsigned int    i;
```

```
 NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);
 USART1_Interupt_Init();
__enable_irq(); //  (Enable Interrupts)

 Send_AT_Command("AT+SetSpeed:[Sense],[1],[2],[200]");    //Motor2 speed fast
 Send_AT_Command("AT+SetSpeed:[Sense],[1],[1],[100]");    //Motor1 speed slow
 Send_AT_Command("AT+SetMotor:[Sense],[1],[2],[Cw]");     //Deviate left
 Send_AT_Command("AT+SetMotor:[Sense],[1],[1],[Cw]");     //
 for (i = 1 ; i != 4000000 ; i++);
 Send_AT_Command("AT+SetSpeed:[Sense],[1],[1],[200]");    //Motor1 speed fast
 Send_AT_Command("AT+SetSpeed:[Sense],[1],[2],[100]");    //Motor2 speed slow
 for (i = 1 ; i != 4000000 ; i++);
 Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Off]");    //Sense stop

End_of_program();

}
```

2. Save the program.

3. Compile the program and check for errors.

4. Download the program by clicking on the **Download**  button.

5. Click on the **Run**  button to run the program.

   The SENSE deviates to the left for 1 second, deviates to the right for 1 second, and then stops.

6. Run the SENSE on the floor without the communication cable.

7. Change the delay time for having turns of 90º.

1.2.6 Challenge exercises – Moving in a square

Task 1: Make a program that moves the SENSE in a 30x30 cm square until it returns to its original place.

Use the **Rotate** instructions for rotating.

Task 2: Make a program that moves the SENSE in a 30x30 cm square until it returns to its original place.

Use the **Turn** instructions for rotating.

Activate the **/*** ***/** signs by deleting the two slashes at the beginning of each line.

The program turns to green.

Experiment 1.3 – Interactive Programs

Objectives:

- Program that reacts to sensors.
- Moving the SENSE to a wall.
- Moving the SENSE to a wall and back.

Equipment required:

- Computer
- SENSE autonomous
- CARM-202 C coding unit
- BAT-202 battery module

Discussion:

In this experiment, we will move the SENSE to a wall.

We will learn how to read and react to the Front Range sensor.

A closed loop system is a control system, which reacts to sensors and switches.

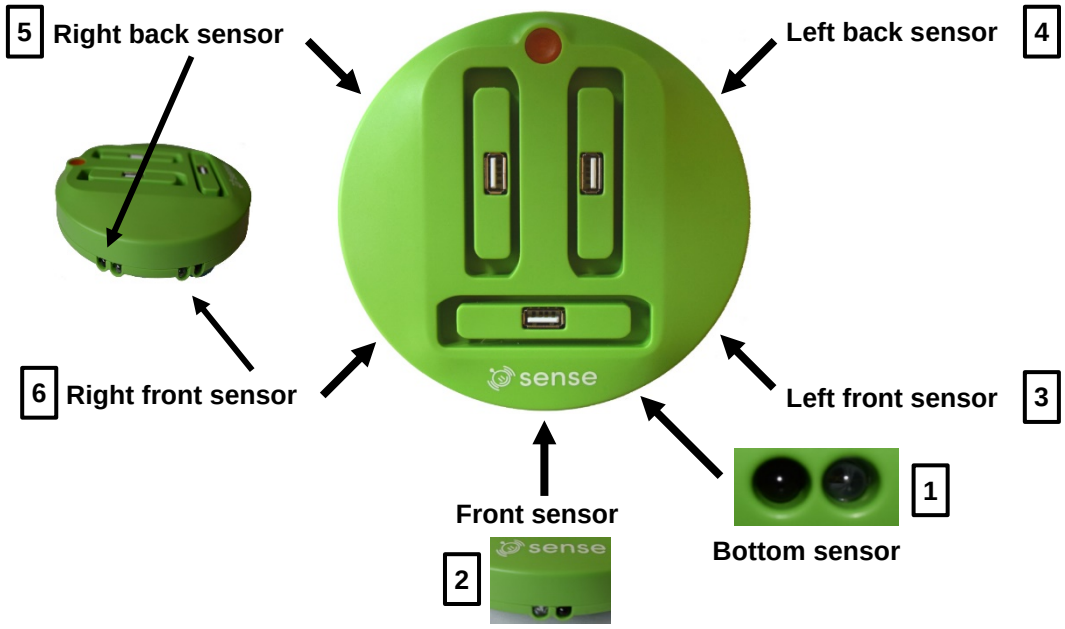
An example for closed loop system is a control system that lights up a lamp when it is dark, and turn it OFF when there is light. This system is automatically adapted to summer time (when the night is short) and to wintertime (when the night is long and starts early).

The program of closed loop system contains decision instructions such as:

'**while**', '**do - while**', '**if – then**'.

1.3.1 The SENSE sensors

The SENSE has 6 sensors. Five range sensors on its perimeter and one line detector on its bottom.



Each sensor has a number marked on the SENSE marked above.

An interactive program reacts to a value read from the sensor. Before writing a program, we have to know the required sensor value to which the program should react to.

The following program prints on the terminal screen the read values from the front sensor (2) every one second.

```
int main (void)
{
  unsigned int    i;

  NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);
   USART1_Interupt_Init();
   USART2_Interupt_Init();
   __enable_irq(); //  (Enable Interrupts)

   while(1)
   {
    //Send GetInput to Sense with ID=1 and waits for answer
    Send_AT_Command("AT+GetInput:[Sense],[1],[2]");

     if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
     {
      Print_to_pc_msg (" Front sensor = ");
      Print_to_pc_inputString ();
     }
     for (i = 1 ; i != 4000000 ; i++);
   }
 }
```

In this program, we use the two USARTs. USART1 for communication with the SENSE and the USART2 for communication with the PC.

The function **Send_AT_Command** sends through USART1 the string:

"AT+GetInput:[Sense],[1],[2]"

Sensor No. 2 is the front sensor (see above).

The function waits for the sensor value from the SENSE and puts it in a string called **inputString**.

If no answer received, the **inputString** will be "**False**".

The program checks that the received string is not "**False**".

If not, prints "**Front sensor =** " and the sensor value.

The function **Print_to_pc_inputString ();** prints the **inputString** with carriage return and line feed.

1.3.2 Moving towards a wall and stopping

The following program moves the SENSE forward and stops when the SENSE is close to a wall.

In this program, we use the variable **STOP** with the stopping value of the front sensor.

```
int main (void)
{
 float   STOP = 350;
 float   VAL;

 NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);
 USART1_Interupt_Init();
__enable_irq(); //  (Enable Interrupts)

Send_AT_Command("AT+SetSpeed:[Sense],[1],[0],[150]");    // Set Sense speed
Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Cw]");    // Sense forward

VAL = 0;
 while(VAL < STOP)
  {
   Send_AT_Command("AT+GetInput:[Sense],[1],[2]");
   if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
    {
     InputString_to_val();     //update inputStringVal
     VAL = inputStringVal;
    }
  }
Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Off]");    // Sense stop

End_of_program();
}
```

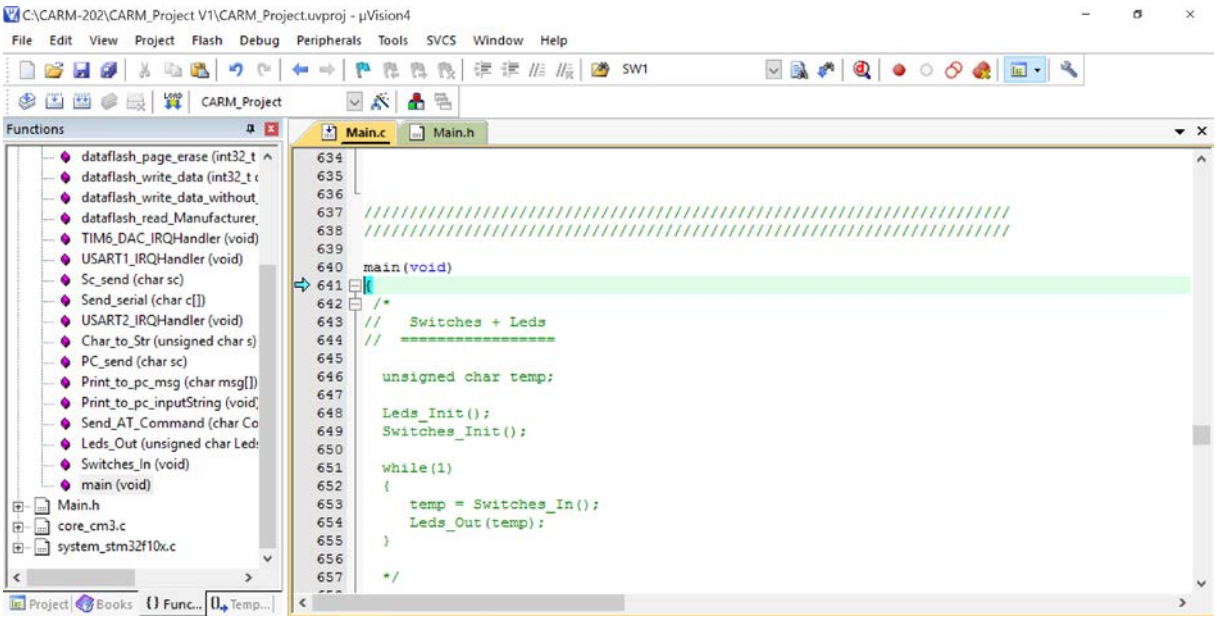
The **inputString** received from SENSE is a string and we cannot compare it with float values.

The function **InputString_to_val();** converts the **inputString** string to a float value called **inputStringVal**.

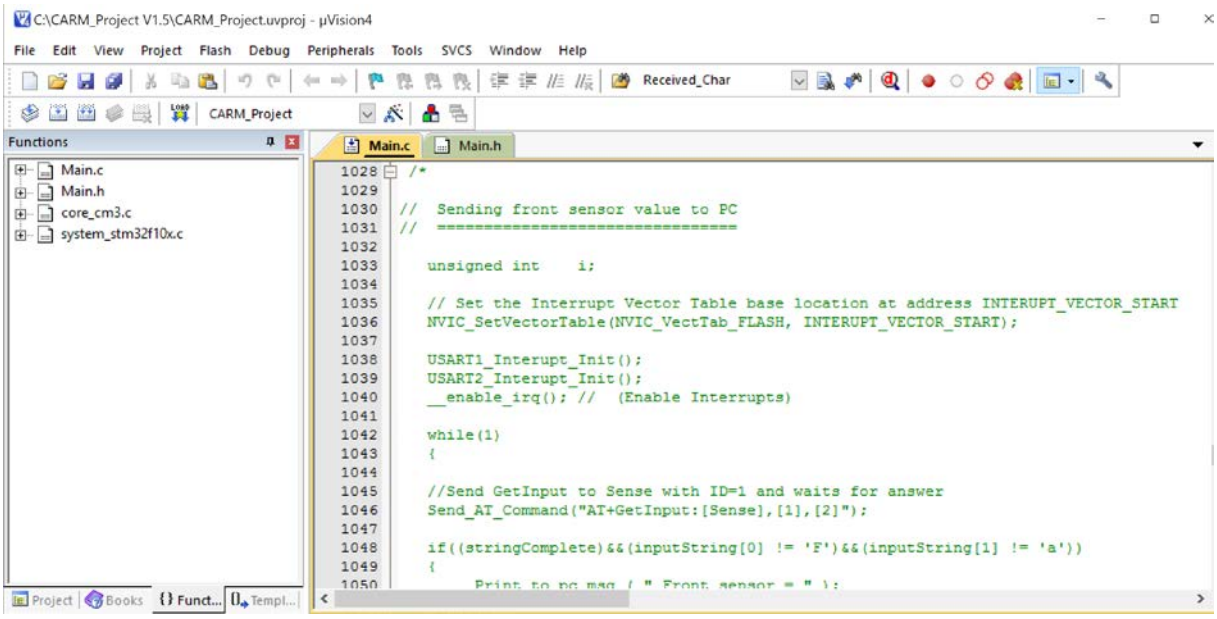
Procedure:

1. Enter the CARM_Project library and double click on the file **CARM_Project.uvproj**.

2. Check that **main.c** and **main.h** are open as in the following screen.

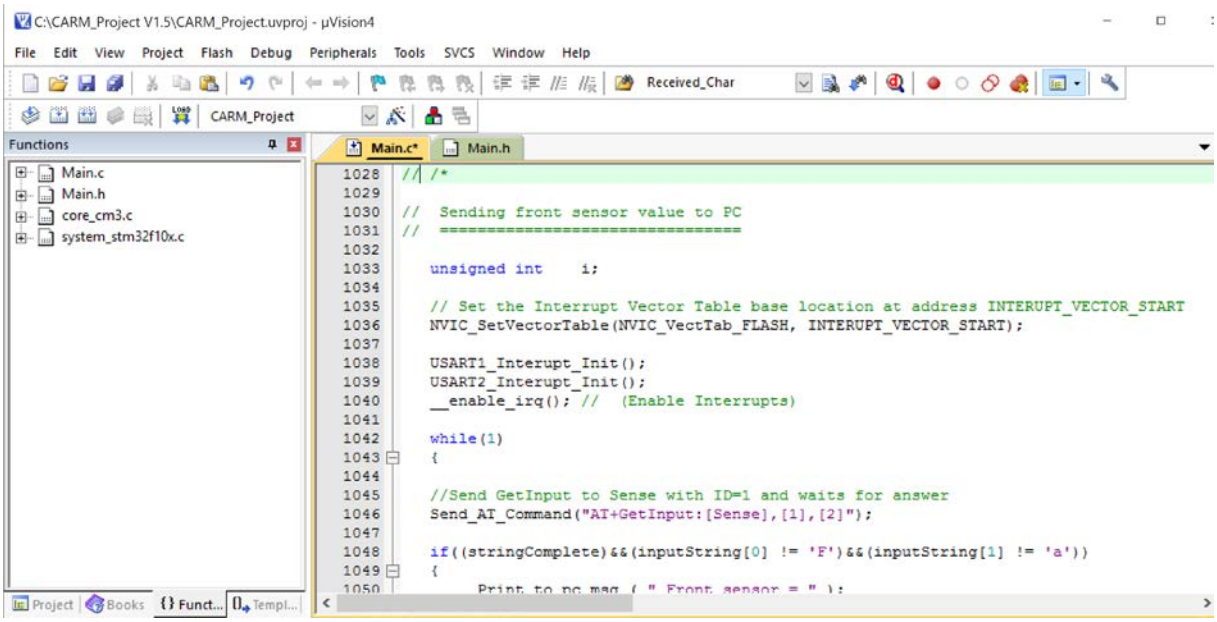


3. Scroll down the **main()** program until you get the following screen:



This section contains the **Sending front sensor value to PC** program.

4. Activate the program by making the two remark limit signs to remark lines by adding two slashes at the beginning of each line and you get the following screen:



1.3.3 Printing front sensor values

1. Observe the program and compare it with the exercise program.

```
int main (void)
{
  unsigned int    i;

 NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);
  USART1_Interupt_Init();
  USART2_Interupt_Init();
  __enable_irq(); //  (Enable Interrupts)

  while(1)
  {
   //Send GetInput to Sense with ID=1 and waits for answer
   Send_AT_Command("AT+GetInput:[Sense],[1],[2]");

    if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
    {
     Print_to_pc_msg ( " Front sensor = " );
     Print_to_pc_inputString ();
    }
    for (i = 1 ; i != 4000000 ; i++);
  }
 }
```

2. Save the program by clicking on the **Save** icon  .

3. Activate the compiler by clicking on the **Rebuild** icon  .

   Check that there are no errors and no warnings.

4. If there are errors, repair the errors and repeat steps 2 and 3 again.

5. Connect the CARM-202 module to the PC by the USB communication cable.

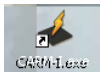
6. Plug the CARM-202 module into the left socket of the SENSE.

7. Plug the BAT-202 battery module into the right socket of the SENSE.

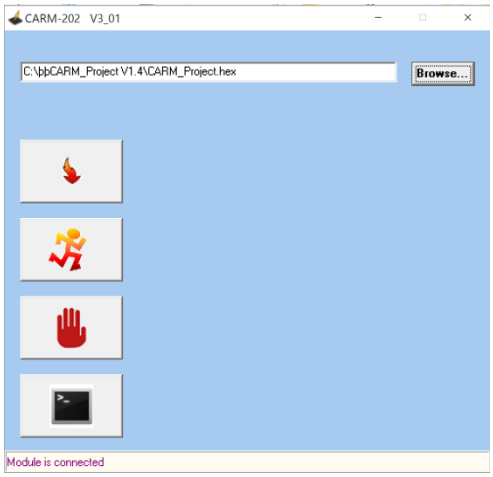
   It does not matter where we plug the modules.

   We do so for the convenience of connecting the communication cable.

8. Find the **CARM-I** program in the CARM exercises directory and double click on it.

   You can also click on its icon  on the desktop.

   The following screen will appear.

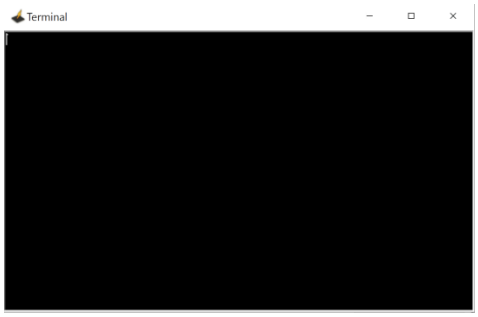


9. Click on the Browse  button and use the browser to find the file **CARM_Project.hex**.

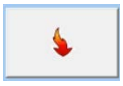
   Click on this file.

10. In order to communicate with the PC, we have to turn the PC into terminal. The terminal transmits the ASCII code of any pressed key and display the received characters.

Click on the **Open Terminal**  button and you will get the following screen.



11. Put the SENSE on the table and put a wide object 30 cm in front of it.

12. Download the program by clicking on the **Download**  button.

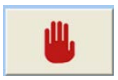
    The program downloaded into CARM-202 and runs.

13. Click on the **Run**  button to run the program.

    The front sensor values will be printed on the screen every one second.

14. Move the 'wall' object close to the SENSE and observe the displayed values.

15. Record the front sensor value in a distance of about 8 cm from the wall.

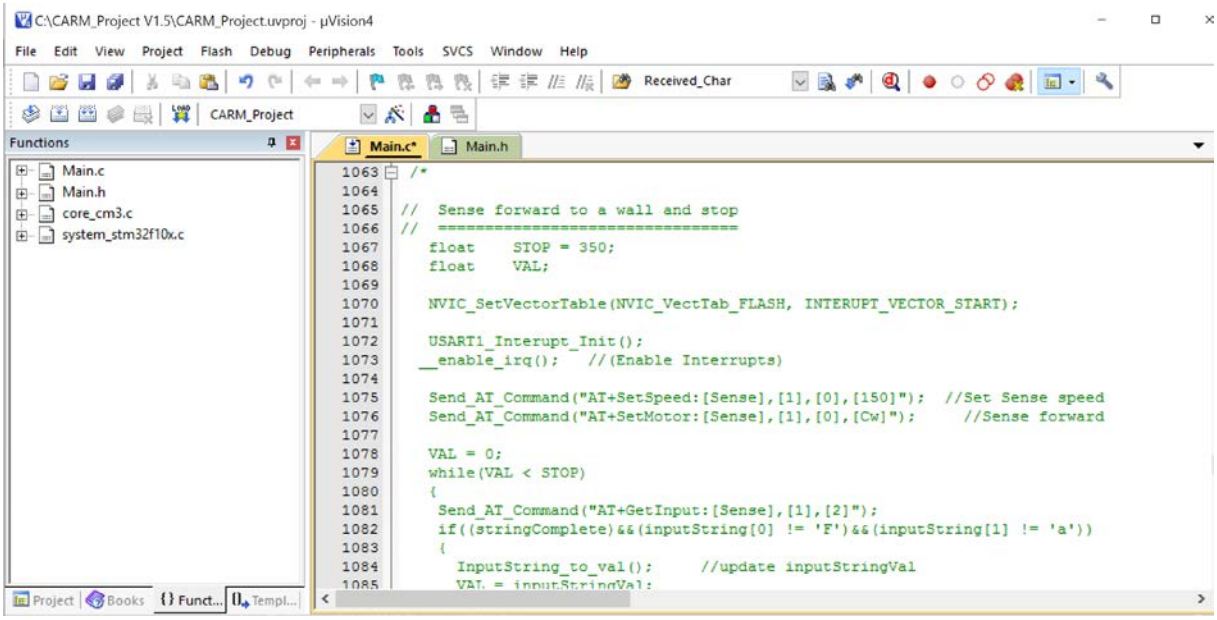
16. Press the **Stop**  to stop the program's running.

17. Activate the **/*** ***/** signs by deleting the two slashes at the beginning of each line.

    The program turns to green.

1.3.4 SENSE to a wall and stop

1. Scroll down until you get the following screen:



This section contains the **Sense forward to a wall and stop** program.

2. Activate the program by making the two remark limit signs to remark lines by adding two slashes at the beginning of each line.

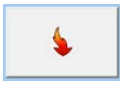
3. Observe the program and compare it with the exercise program.

```
int main (void)
{
 float   STOP = 350;
 float   VAL;

 NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);
 USART1_Interupt_Init();
__enable_irq(); //  (Enable Interrupts)

Send_AT_Command("AT+SetSpeed:[Sense],[1],[0],[150]");          //Set Sense speed
Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Cw]");           //Sense forward

VAL = 0;
 while(VAL < STOP)
  {
    Send_AT_Command("AT+GetInput:[Sense],[1],[2]");

   if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
    {
      InputString_to_val();     //update inputStringVal
      VAL = inputStringVal;
    }
  }
Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Off]");          //Sense stop

 End_of_program();
}
```

4. Save the program.

5. Compile the program and check for errors.

6. Put the SENSE on the table and put a wide object 30 cm in front of it.

7. Download the program by clicking on the **Download**  button.

   You do not have to select the file **CARM_Project.hex**.

8. Click on the **Run**  button to run the program.

   The SENSE will move forward and stop about 8 cm from the wall.

9. Change the program so the SENSE stops 5 cm in front of the wall.

10. Save the program.

11. Compile the program and check for errors.

12. Download the program by clicking on the **Download**  button.

13. Click on the **Run**  button to run the program.

    Check the SENSE movement.

14. Change the program so the SENSE stops 5 cm in front of the wall, waits two seconds and then moves backward for two seconds.

15. Save the program.

16. Compile the program and check for errors.

17. Download the program by clicking on the **Download**  button.

18. Click on the **Run**  button to run the program.

    Check the SENSE movement.

1.3.5 Endless loop

Most of the control and robotic programs are programs that run in endless loop.

1. Change the program so that the SENSE goes forward and stops when it meets the wall, goes back for 3 seconds, and forward again in endless loop.

   Add
   while (1)
   {
   Movement program
   }

2. Save the program.

3. Compile the program and check for errors.

4. Download the program by clicking on the **Download**  button.

5. Click on the **Run**  button to run the program.

   Check the SENSE movement.

6. Run the SENSE on the floor without the communication cable.

1.3.6 Challenge exercise – Moving to a wall and back

Task 1: Improve the program so that the SENSE will move between 5 cm from the wall and 10 cm from the wall very slow.

Activate the **/*** ***/** signs by deleting the two slashes at the beginning of each line.

The program turns to green.

Experiment 1.4 – Movement Along a Black Line

Objectives:

- Program that reacts to sensors.
- Moving the SENSE to a black line.
- Moving the SENSE between lines.
- Moving the SENSE along a black line.

Equipment required:

- Computer
- SENSE autonomous
- CARM-202 C coding unit
- BAT-202 battery module

Discussion:

In this experiment, we will move the SENSE to a black line and between two black lines. The position of the lines limits its motion. The robot changes direction when it finds a black line. This is an example of a system called a Manipulator.

We will learn how to read and react to the Bottom Line sensor.

1.4.1 The SENSE bottom sensor

The following program prints on the terminal screen the read values from the bottom sensor (1) every one second.

```
int main (void)
{
 unsigned int    i;

 NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);
  USART1_Interupt_Init();
  USART2_Interupt_Init();
  __enable_irq(); //  (Enable Interrupts)

  while(1)
  {
   //Send GetInput to Sense with ID=1 and waits for answer
   Send_AT_Command("AT+GetInput:[Sense],[1],[1]");
    if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
    {
     Print_to_pc_msg ( " Bottom sensor = " );
     Print_to_pc_inputString ();
    }
    for (i = 1 ; i != 4000000 ; i++);
  }
}
```

In this program we use the two UARTs. USART1 for communication with the SENSE and the USART2 for communication with the PC.

The function **Send_AT_Command** sends through USART1 the string:

"AT+GetInput:[Sense],[1],[1]"

Sensor No. 1 is the bottom sensor.

The function waits for the sensor value from the SENSE and puts it in a string called **inputString**. If no answer received, the **inputString** will be "**False**".

The program checks that the received string is not "**False**".
If not, prints "**Bottom sensor =** " and the sensor value.

The function **Print_to_pc_inputString ();** prints the **inputString** with carriage return and line feed.

1.4.2 Moving to a black line and stopping

The following program moves the SENSE forward and stops when the SENSE is on a black line.

In this program, we use the variable **STOP** with the stopping value of the front sensor.

```
int main (void)
{
 float   BLACK = 250;
 float   VAL;

 NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);
 USART1_Interupt_Init();
__enable_irq(); //  (Enable Interrupts)

Send_AT_Command("AT+SetSpeed:[Sense],[1],[0],[150]");    //Set Sense speed
Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Cw]");    //Sense forward

VAL = 900;
 while(VAL > BLACK)
  {
   Send_AT_Command("AT+GetInput:[Sense],[1],[1]");
   if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
    {
     InputString_to_val();     //update inputStringVal
     VAL = inputStringVal;
    }
  }
Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Off]");    //Sense stop

 End_of_program();
}
```

The **inputString** received from SENSE is a string and we cannot compare it with float values.

The function **InputString_to_val();** converts the **inputString** string to a float value called **inputStringVal**.

Before proceeding, print two black lines as follows:

1.4.3 Moving along a black line

To move the SENSE along a black line we use turn procedures of the SENSE.

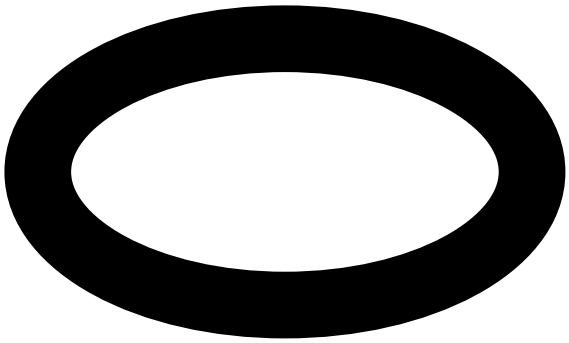
In turns, one wheel rotates and the other wheel stops. This way the SENSE still moves forward while turning.

In the main program, we do the movement according to the following idea:

Turning left until the SENSE find a black surface, and then turning right until the SENSE find a white surface.

```
int main (void)
{
 float   BLACK = 250;
 float   VAL;

 NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);
 USART1_Interupt_Init();
__enable_irq(); //  (Enable Interrupts)

Send_AT_Command("AT+SetSpeed:[Sense],[1],[0],[150]"); //Set Sense speed

VAL = 900;
 while (1)
 {
  Send_AT_Command("AT+SetMotor:[Sense],[1],[2],[Cw]");          //Turn left
  Send_AT_Command("AT+SetMotor:[Sense],[1],[1],[Off]");
  while(VAL > BLACK)
  {
   Send_AT_Command("AT+GetInput:[Sense],[1],[1]");

    if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
     {
      InputString_to_val();     //update inputStringVal
      VAL = inputStringVal;
     }
   }

  Send_AT_Command("AT+SetMotor:[Sense],[1],[1],[Cw]");          //Turn right
  Send_AT_Command("AT+SetMotor:[Sense],[1],[2],[Off]");

  while(VAL <= BLACK)
  {
   Send_AT_Command("AT+GetInput:[Sense],[1],[1]");
    if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
     {
      InputString_to_val();     //update inputStringVal
      VAL = inputStringVal;
     }
   }
 }
}
```

Before proceeding, print on a full page a black line as the following:

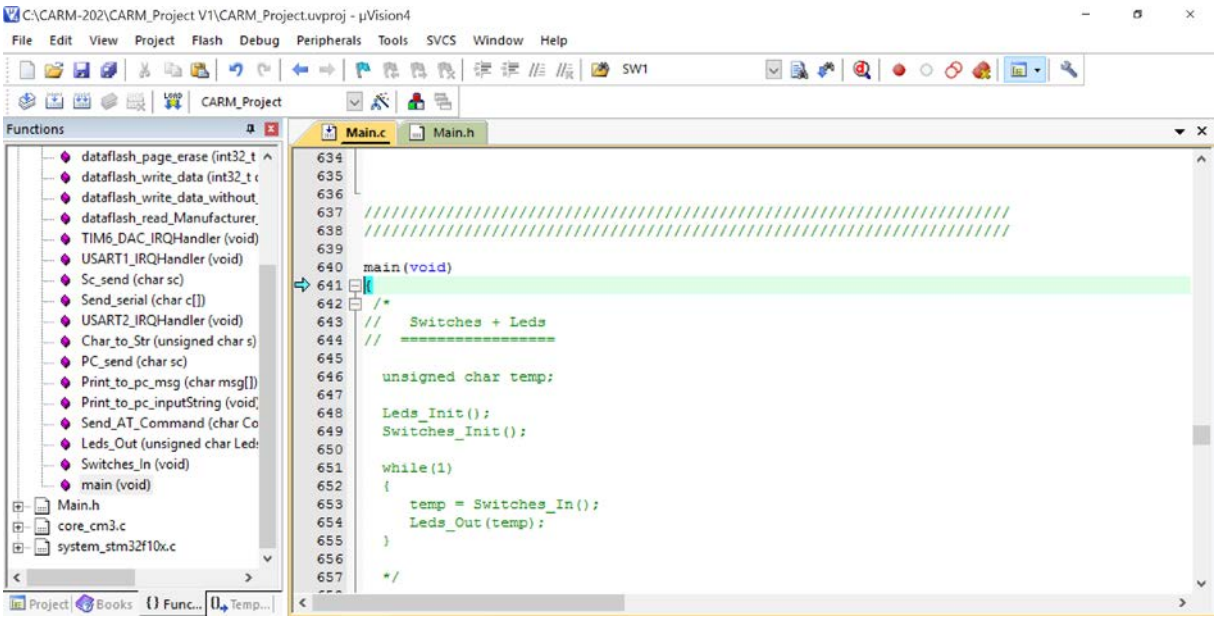


The width of the line should be at least 3 cm.

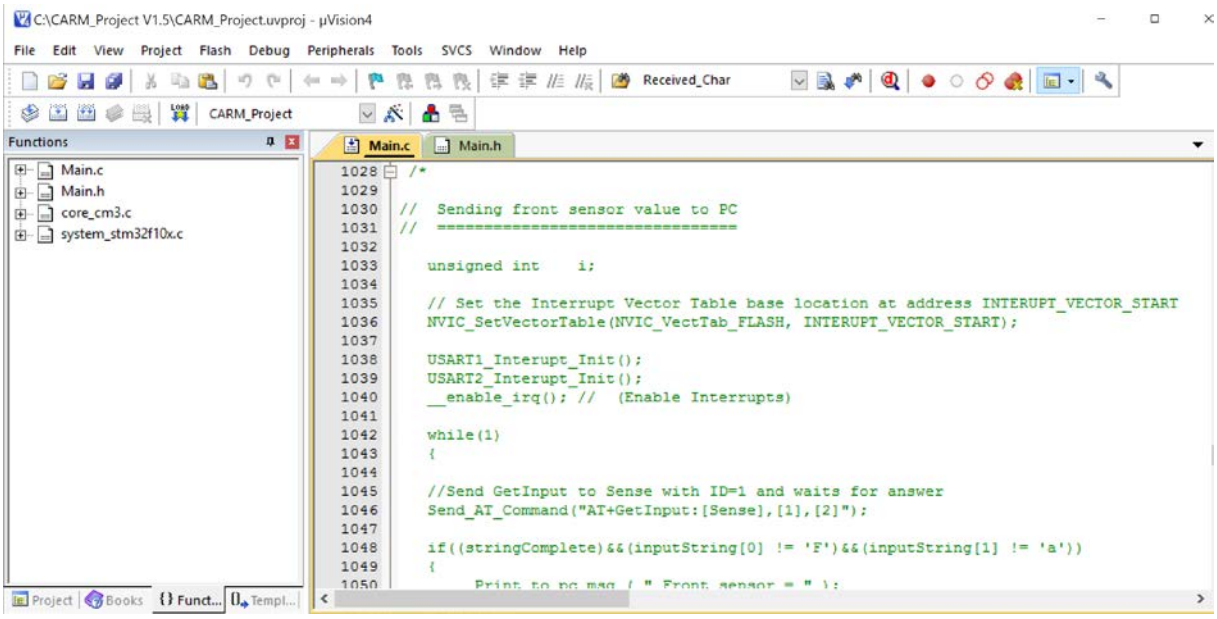
Procedure:

1. Enter the CARM_Project library and double click on the file **CARM_Project.uvproj**.

2. Check that **main.c** and **main.h** are open as in the following screen.

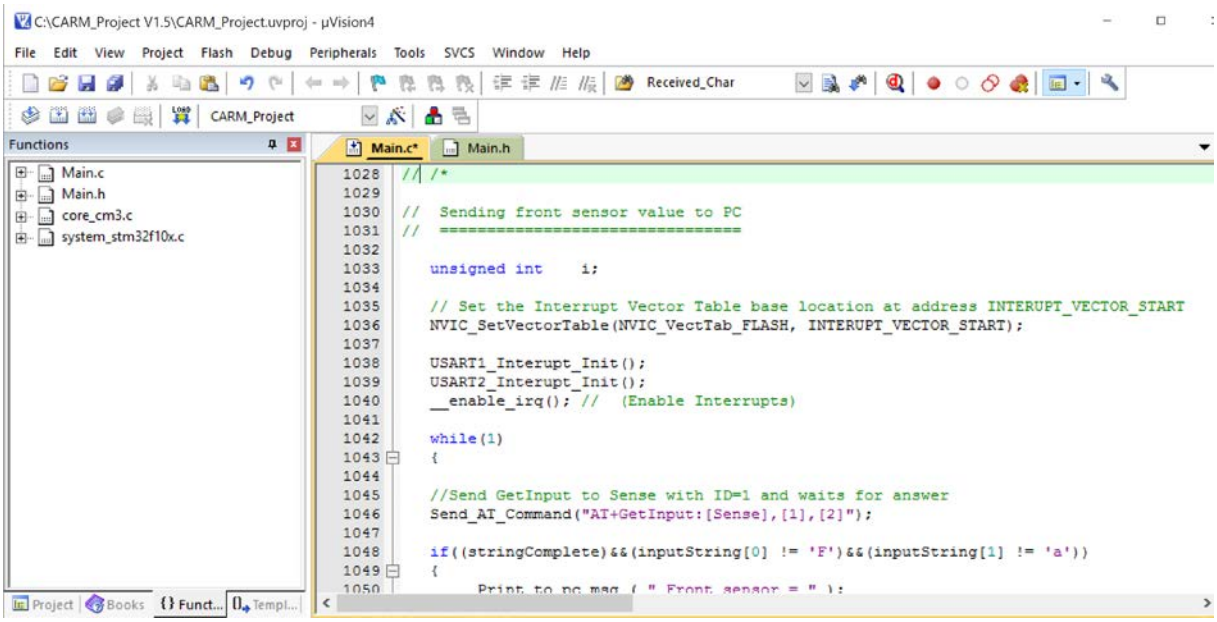


3. Scroll down the **main()** program until you get the following screen:



This section contains the **Sending front sensor value to PC** program.

4. Activate the program by making the two remark limit signs to remark lines by adding two slashes at the beginning of each line and you get the following screen:



1.4.4 Printing sensor values

1. Change the program to print the bottom sensor value. The two required changes are marked by arrows.

```
int main (void)
{
 unsigned int    i;

 NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);

  USART1_Interupt_Init();
  USART2_Interupt_Init();
  __enable_irq(); //  (Enable Interrupts)

  while(1)
  {
   //Send GetInput to Sense with ID=1 and waits for answer
   Send_AT_Command("AT+GetInput:[Sense],[1],[1]");

    if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
    {
     Print_to_pc_msg ( " Bottom sensor = " );
     Print_to_pc_inputString ();
    }
    for (i = 1 ; i != 4000000 ; i++);
  }
 }
```

2. Save the program by clicking on the **Save** icon  .

3. Activate the compiler by clicking on the **Rebuild** icon  .

   Check that there are no errors and no warnings.

4. If there are errors, repair the errors and repeat steps 2 and 3 again.

5. Connect the CARM-202 module to the PC by the USB communication cable.

6. Plug the CARM-202 module into the left socket of the SENSE.

7. Plug the BAT-202 battery module into the right socket of the SENSE.

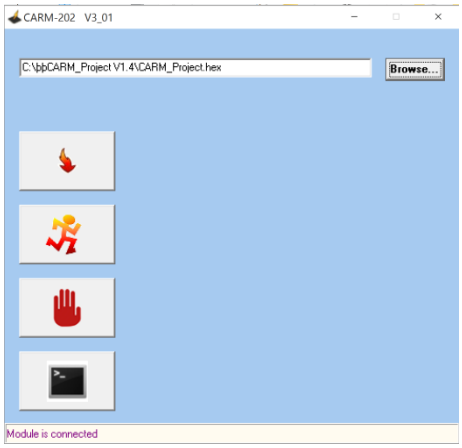
   It does not matter where we plug the modules.

   We do so for the convenience of connecting the communication cable.

8. Find the **CARM-I** program in the CARM exercises directory and double click on it.

   You can also click on its icon  on the desktop.

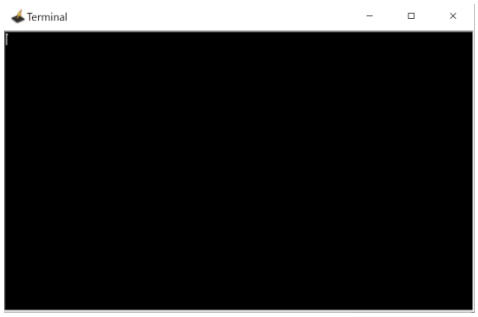
   The following screen will appear.



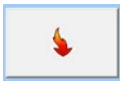
9. Click on the **Browse**  button and use the browser to find the file **CARM_Project.hex**.

   Click on this file.

10. Click on the **Open Terminal**  button and you will get the following screen.



11. Put the SENSE on white surface.

12. Download the program by clicking on the **Download**  button.

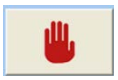
13. Click on the **Run**  button to run the program.

    The bottom sensor values will be printed on the screen every one second.

    Record the value of the sensor when the SENSE is on a white surface.

14. Put the SENSE on black surface and observe the displayed values.

15. Record the value of the sensor when the SENSE is on a black surface.

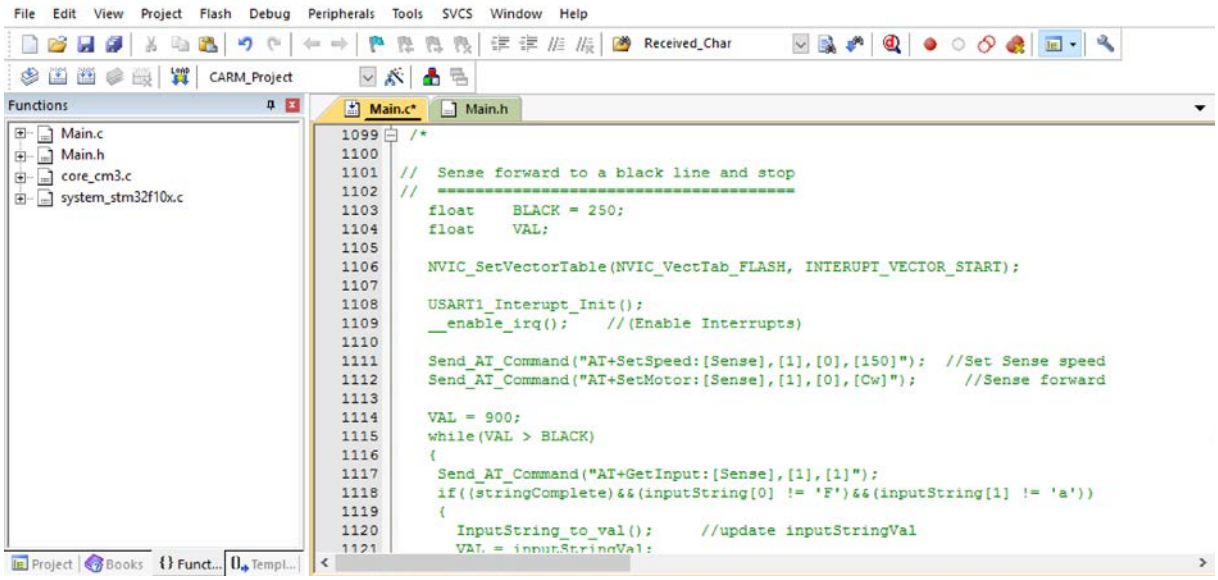
16. Press the **Stop**  to stop the program's running.

17. Activate the **/*** ***/** signs by deleting the two slashes at the beginning of each line.

    The program turns to green.

1.4.5 SENSE to a black line and stop

1. Scroll down until you get the following screen:



This section contains the **Sense forward to a black line and stop** program.

2. Activate the program by making the two remark limit signs to remark lines by adding two slashes at the beginning of each line.

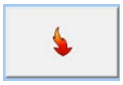
3. Observe the program and compare it with the exercise program.

```
int main (void)
{
 float   BLACK = 250;
 float   VAL;

 NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);
 USART1_Interupt_Init();
__enable_irq(); //  (Enable Interrupts)

Send_AT_Command("AT+SetSpeed:[Sense],[1],[0],[150]");          //Set Sense speed
Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Cw]");           //Sense forward

VAL = 900;
 while(VAL > BLACK)
  {
   Send_AT_Command("AT+GetInput:[Sense],[1],[1]");

   if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
    {
     InputString_to_val();    //update inputStringVal
     VAL = inputStringVal;
    }
  }
Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Off]");          //Sense stop

 End_of_program();
}
```

4. Save the program.

5. Compile the program and check for errors.

6. Put the SENSE on white surface with a black line in front.

7. Download the program by clicking on the **Download**  button.

8. Click on the **Run**  button to run the program.

   The SENSE will move forward and stop on the black line.

9. Change the program so the SENSE stops on the black line, waits two seconds and then moves backward for two seconds.

10. Save the program.

11. Compile the program and check for errors.

12. Download the program by clicking on the **Download**  button.

13. Click on the **Run**  button to run the program.

    Check the SENSE movement.

1.4.6 Moving to a black line in an endless loop

1. Change the program so that the SENSE goes forward and stops when it meets the black line, goes back for 3 seconds, and forward again in endless loop.

   Add for the endless movement the following loop:

   while (1)
   {
   Movement program
   }

2. Save the program.

3. Compile the program and check for errors.

4. Download the program by clicking on the **Download**  button.

5. Click on the **Run**  button to run the program.

   Check the SENSE movement.

1.4.7 Moving between two black lines

1. Change the program so that the SENSE goes forward and stops when it meets the black line, waits for 2 seconds, goes back until it meets the second black line, waits for 2 seconds, and forward again in endless loop.

   Note:
   When the SENSE changes direction, it should move a little without checking the bottom sensor, in order to be sure that it is out of the black line.

   Add short delay after each changing direction instruction.

2. Save the program.

3. Compile the program and check for errors.

4. Put the SENSE on white surface with two black lines, between the two lines.

5. Download the program by clicking on the **Download**  button.

6. Click on the **Run**  button to run the program.

   Check the SENSE movement.

7. Run the SENSE without the communication cable.

1.4.8 Challenge exercise – Between a wall and a line (I)

Task 1: Improve the program so that the SENSE will move between a wall in front and a black line at the back.

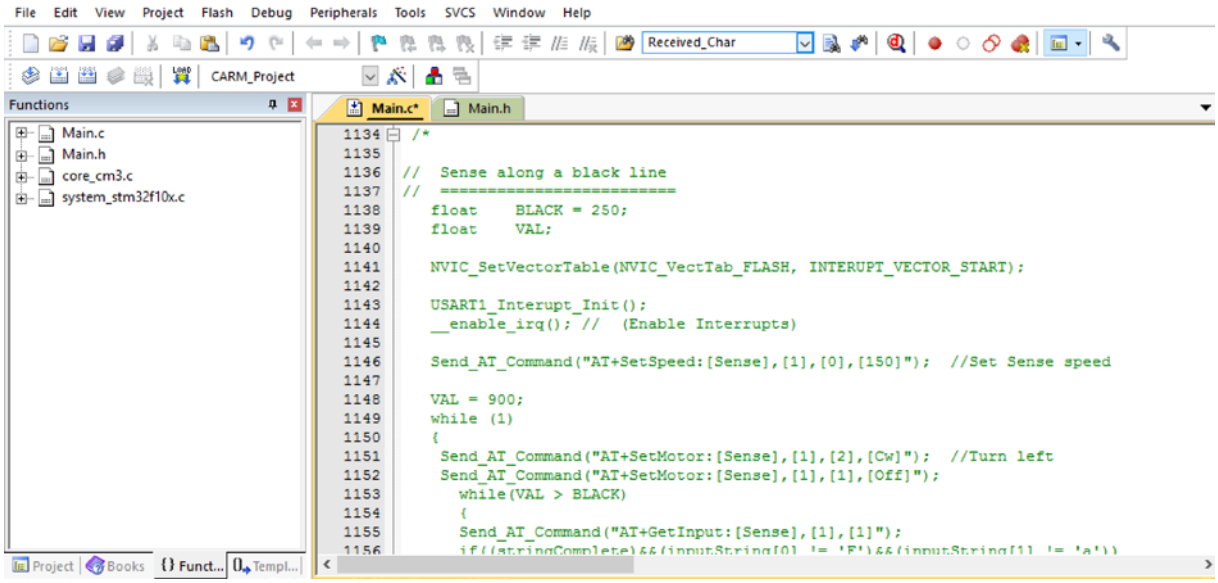
Note:
You have to use the Front sensor while moving forward. Take care for the compare sign (> or <).

Activate the **/*** ***/** signs by deleting the two slashes at the beginning of each line.

The program turns to green.

1.4.9 Moving along a black line

1. Scroll down until you get the following screen:



This section contains the **Sense along a black line** program.

2. Activate the program by making the two remark limit signs to remark lines by adding two slashes at the beginning of each line.

3. Observe the program and compare it with the exercise program.

```
int main (void)
{
 float   BLACK = 250;
 float   VAL;

 NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);
 USART1_Interupt_Init();
__enable_irq(); //  (Enable Interrupts)

Send_AT_Command("AT+SetSpeed:[Sense],[1],[0],[150]");          //Set Sense speed

VAL = 900;
 while (1)
 {
  Send_AT_Command("AT+SetMotor:[Sense],[1],[2],[Cw]");        //Turn left
  Send_AT_Command("AT+SetMotor:[Sense],[1],[1],[Off]");
  while(VAL > BLACK)
  {

   Send_AT_Command("AT+GetInput:[Sense],[1],[1]");

    if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
     {
      InputString_to_val();     //update inputStringVal
      VAL = inputStringVal;
     }
   }

  Send_AT_Command("AT+SetMotor:[Sense],[1],[1],[Cw]");        //Turn right
  Send_AT_Command("AT+SetMotor:[Sense],[1],[2],[Off]");

  while(VAL <= BLACK)
  {

   Send_AT_Command("AT+GetInput:[Sense],[1],[1]");

    if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
     {
      InputString_to_val();     //update inputStringVal
      VAL = inputStringVal;
     }
   }
 }
}
```

Note:
Pay attention to the compare signs (< and >).

4. Save the program.

5. Compile the program and check for errors.

6. Put the SENSE on white surface near the black circle.

7. Download the program by clicking on the **Download**  button.

8. Click on the **Run**  button to run the program.

   The SENSE should move along the black line.

9. Change the value of the Black variable to create smooth movement of the SENSE.

1.4.10 Moving along a black line and stop

1. We shall improve the program so the SENSE stops when you put your hand in front of it.

   Change the program to the following according to the bold sections:

```
int main (void)
{
 float    BLACK = 250;
 float     STOP = 300;
 float    VAL, FRONT;

 NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);
 USART1_Interupt_Init();
 __enable_irq(); //  (Enable Interrupts)

 Send_AT_Command("AT+SetSpeed:[Sense],[1],[0],[150]");    //Set Sense speed

 FRONT = 0;
 VAL = 900;
 while (1)
 {
    Send_AT_Command("AT+GetInput:[Sense],[1],[2]");        //Check front sensor
    if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
    {
      InputString_to_val();       //update inputStringVal
      FRONT = inputStringVal;
    }

    if (FRONT > STOP)
    {
     Send_AT_Command("AT+SetMotor:[Sense],[1],[0],[Off]");        //Stop the SENSE
     do
     {
      Send_AT_Command("AT+GetInput:[Sense],[1],[2]");   //Wait until obstacle is out
      if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
      {
       InputString_to_val();     //update inputStringVal
       FRONT = inputStringVal;
      }
     }while (FRONT > STOP);
    }
```

```
    Send_AT_Command("AT+SetMotor:[Sense],[1],[2],[Cw]");          //Turn left
    Send_AT_Command("AT+SetMotor:[Sense],[1],[1],[Off]");
    while(VAL > BLACK)
    {
     Send_AT_Command("AT+GetInput:[Sense],[1],[1]");       //Check bottom sensor
     if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
     {
      InputString_to_val();      //update inputStringVal
      VAL = inputStringVal;
     }
    }

    Send_AT_Command("AT+SetMotor:[Sense],[1],[1],[Cw]");          //Turn right
    Send_AT_Command("AT+SetMotor:[Sense],[1],[2],[Off]");
    while(VAL <= BLACK)
    {
     Send_AT_Command("AT+GetInput:[Sense],[1],[1]");       //Check bottom sensor
     if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
     {
      InputString_to_val();      //update inputStringVal
      VAL = inputStringVal;
     }
    }
  }
}
```

In every cycle, the main procedure checks the distance from the wall and calls the STOP instruction when the SENSE is close to the wall.

In control systems, we usually prefer that the OFF condition value will be different from the ON condition value. The reason is that we want to avoid having the system "bounce".

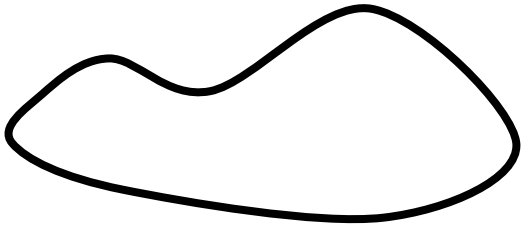
2. Put your hand in front of the SENSE, while it moves.

3. Change the values of the variables until the SENSE works well.

1.4.11 Challenge exercise – Along a complex black line

Task 1: Create different black lines for the SENSE and check its behavior. Improve the programs when needed.

Example of a complex line:



Activate the **/*** ***/** signs by deleting the two slashes at the beginning of each line.

The program turns to green.

Experiment 1.5 – Movement Along Walls

Objectives:

- Program that reacts to side sensors.
- Moving the SENSE along walls.
- Moving the SENSE along walls and stopping it.
- Moving the SENSE along walls and turning it around.

Equipment required:

- Computer
- SENSE autonomous
- CARM-202 C coding unit
- BAT-202 battery module

Discussion:

In this experiment, we will move the SENSE along walls.

We will learn how to read and react to the Front right sensor.

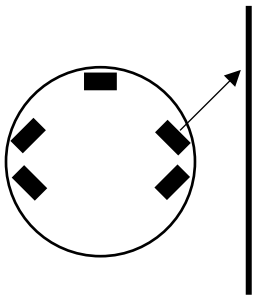
1.5.1 Movement along walls

To move the SENSE along a wall, we use the same algorithm of moving the SENSE along a black line. We use the turn commands.

- Turn left when the SENSE is too close to the wall.
- Turn right when the SENSE is far from the wall.

To go along a wall on the right, we use the front side range sensor.

The side range sensors are installed in 45^{o} to the SENSE base.



When the SENSE turns to the right, the measured distance is smaller than when it turns to the left.

Think what will happen if the range sensor is parallel to the wall.

1.5.2 The SENSE right front sensor

The following program prints on the terminal screen the read values from the bottom sensor (1) every one second.

```
int main (void)
{
  unsigned int    i;

  NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);

   USART1_Interupt_Init();
   USART2_Interupt_Init();
   __enable_irq(); //  (Enable Interrupts)

   while(1)
   {
     //Send GetSensorValue to Sense with ID=1 and waits for answer
     Send_AT_Command("AT+GetInput:[Sense],[1],[6]");

     if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
     {
      Print_to_pc_msg ( " Front right sensor = " );
      Print_to_pc_inputString ();
     }
     for (i = 1 ; i != 4000000 ; i++);
   }
 }
```

In this program we use the two UARTs. USART1 for communication with the SENSE and the USART2 for communication with the PC.

The function **Send_AT_Command** sends through USART1 the string:

"AT+GetInput:[Sense],[1],[6]"

Sensor No. 6 is the front right sensor.

The function waits for the sensor value from the SENSE and puts it in a string called **inputString**. If no answer received, the **inputString** will be "**False**".

The program checks that the received string is not "**False**".

If not, prints **" Front right sensor = "** and the sensor value.

The function **Print_to_pc_inputString ();** prints the **inputString** with carriage return and line feed.

1.5.3 Moving along walls

To move the SENSE along turn commands of the SENSE.

In turns, one wheel rotates and the other wheel stops. This way the SENSE still moves forward while turning.

In the main program, we do the movement according to the following idea:

Turning right until the SENSE is close to the wall, and then turning left until the SENSE is far from the wall.

```
int main (void)
{
  float  WALL = 300;
  float  VAL;

  NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);

  USART1_Interupt_Init();
  __enable_irq();     //(Enable Interrupts)

  Send_AT_Command("AT+SetSpeed:[Sense],[1],[0],[150]");        //Set Sense speed

  VAL = 0;
  while (1)
  {
     Send_AT_Command("AT+SetMotor:[Sense],[1],[2],[Cw]");     //Turn left
     Send_AT_Command("AT+SetMotor:[Sense],[1],[1],[Off]");
     while(VAL >= WALL)
     {
          Send_AT_Command("AT+GetInput:[Sense],[1],[6]");  //Check Front right sensor
          if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
          {
                    InputString_to_val();     //update inputStringVal
                    VAL = inputStringVal;
          }
     }

     Send_AT_Command("AT+SetMotor:[Sense],[1],[1],[Cw]");      //Turn right
     Send_AT_Command("AT+SetMotor:[Sense],[1],[2],[Off]");
     while(VAL < WALL)
     {
          Send_AT_Command("AT+GetInput:[Sense],[1],[6]");  //Check Front right sensor
          if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
          {
                    InputString_to_val();     //update inputStringVal
                    VAL = inputStringVal;
          }
     }
  }
}
```

Note:
Pay attention to the compare signs (< and >).

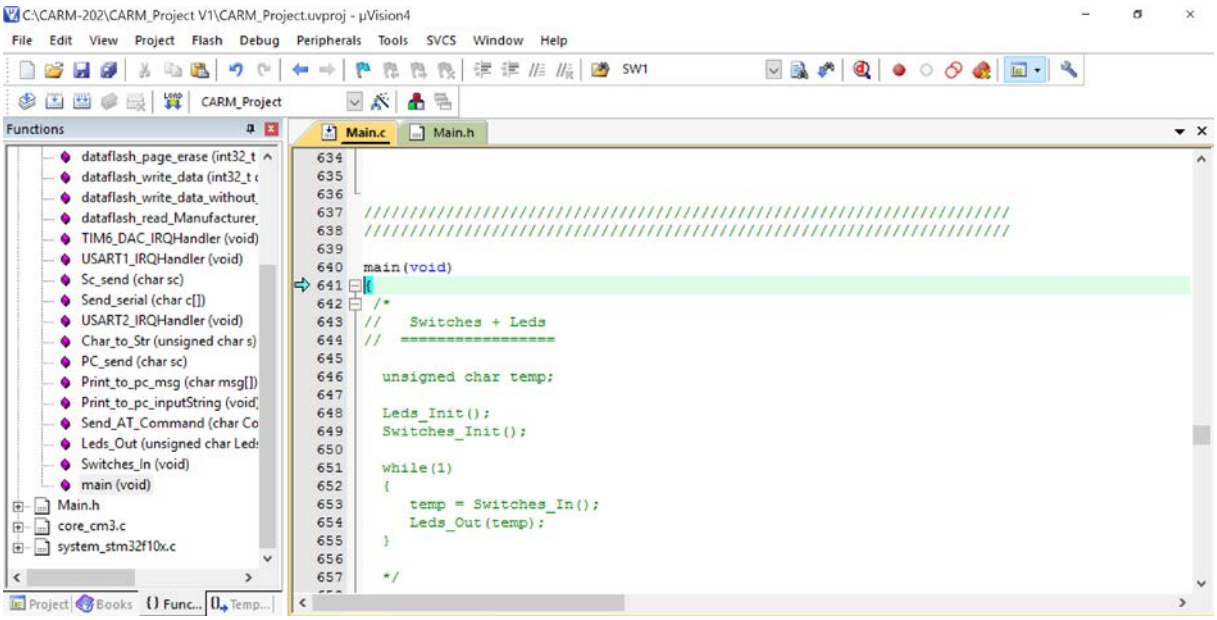
Before proceeding, prepare a box for the SENSE to go around it.

Take care that the box is not black or with dark color. White box is better.

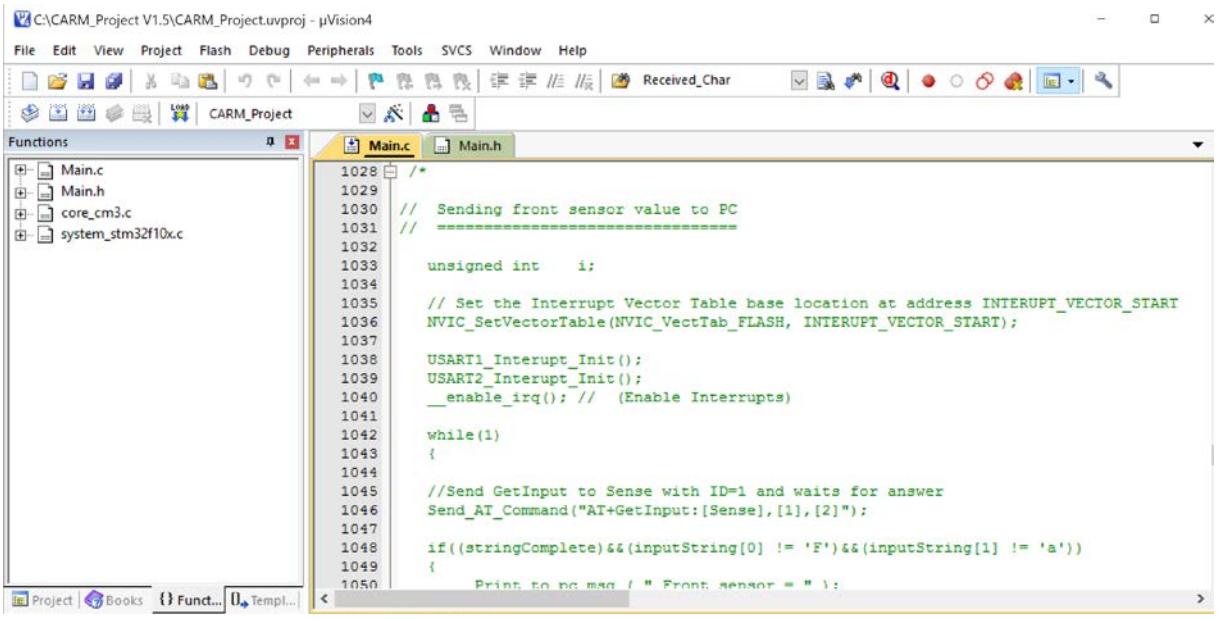
Procedure:

1. Enter the CARM_Project library and double click on the file **CARM_Project.uvproj**.

2. Check that **main.c** and **main.h** are open as in the following screen.

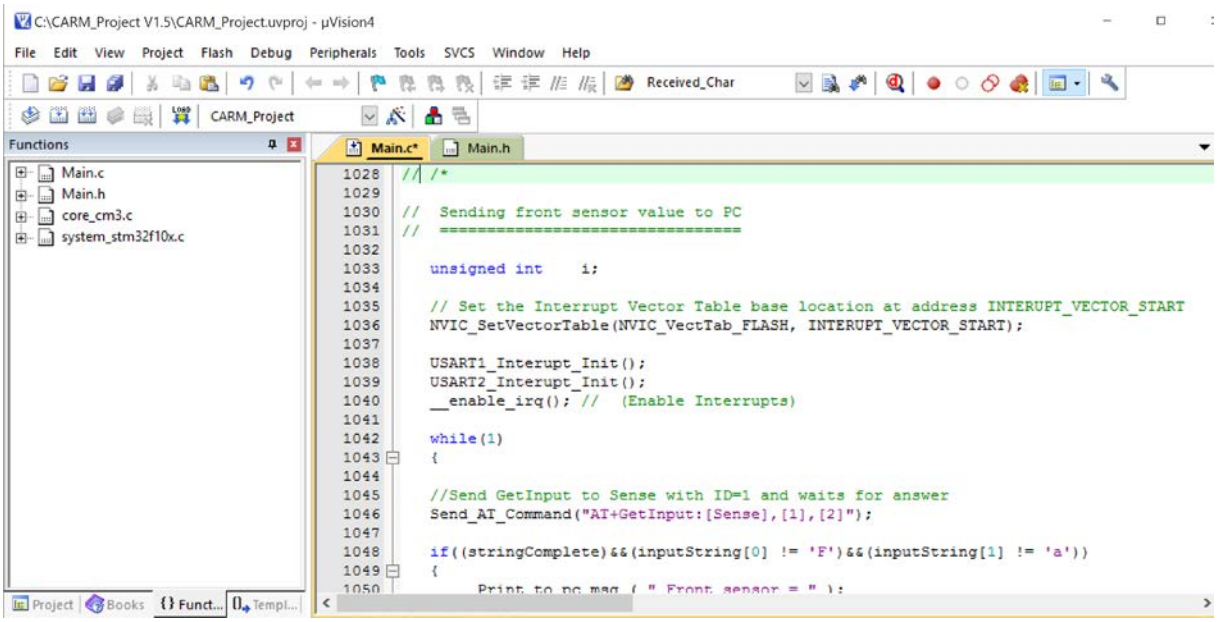


3. Scroll down the **main()** program until you get the following screen:



This section contains the **Sending front sensor value to PC** program.

4. Activate the program by making the two remark limit signs to remark lines by adding two slashes at the beginning of each line and you get the following screen:



1.5.4 Printing sensor values

1. Change the program to print the bottom sensor value. The two required changes are marked by arrows.

```
int main (void)
{
 unsigned int    i;

 NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);

  USART1_Interupt_Init();
  USART2_Interupt_Init();
  __enable_irq(); //  (Enable Interrupts)

  while(1)
  {
   //Send GetSensorValue to Sense with ID=1 and waits for answer
   Send_AT_Command("AT+GetInput:[Sense],[1],[6]");

    if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
    {
     Print_to_pc_msg ( " Front right sensor = " );
     Print_to_pc_inputString ();
    }
    for (i = 1 ; i != 4000000 ; i++);
  }
 }
```

2. Save the program by clicking on the **Save** icon  .

3. Activate the compiler by clicking on the **Rebuild** icon  .

   Check that there are no errors and no warnings.

4. If there are errors, repair the errors and repeat steps 2 and 3 again.

5. Connect the CARM-202 module to the PC by the USB communication cable.

6. Plug the CARM-202 module into the left socket of the SENSE.

7. Plug the BAT-202 battery module into the right socket of the SENSE.

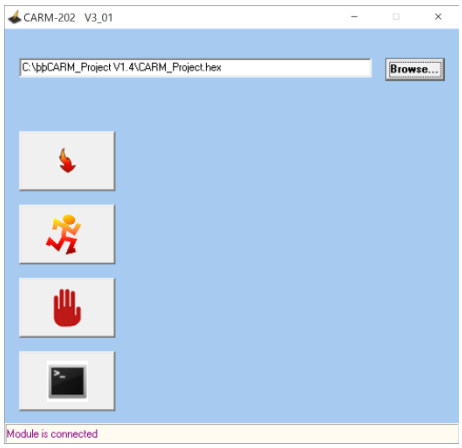
   It does not matter where we plug the modules.

   We do so for the convenience of connecting the communication cable.

8. Find the **CARM-I** program in the CARM exercises directory and double click on it.

   You can also click on its icon  on the desktop.

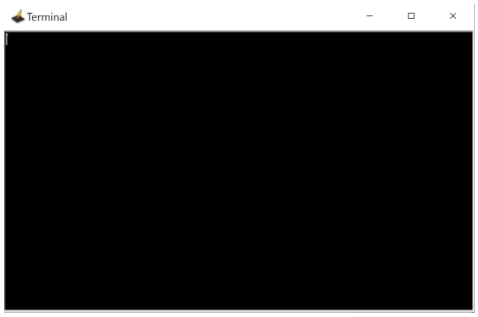
   The following screen will appear.



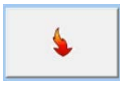
9. Click on the **Browse**  button and use the browser to find the file **CARM_Project.hex**.

   Click on this file.

10. Click on the **Open Terminal**  button and you will get the following screen.



11. Put the SENSE with its right sensors close to the box.

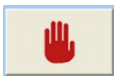
12. Download the program by clicking on the **Download**  button.

13. Click on the **Run**  button to run the program.

    The program downloaded into CARM-202 and runs.

    The front right sensor values will be printed on the screen every one second.

    Record the value of the sensor when the SENSE is 4 cm parallel to the box.

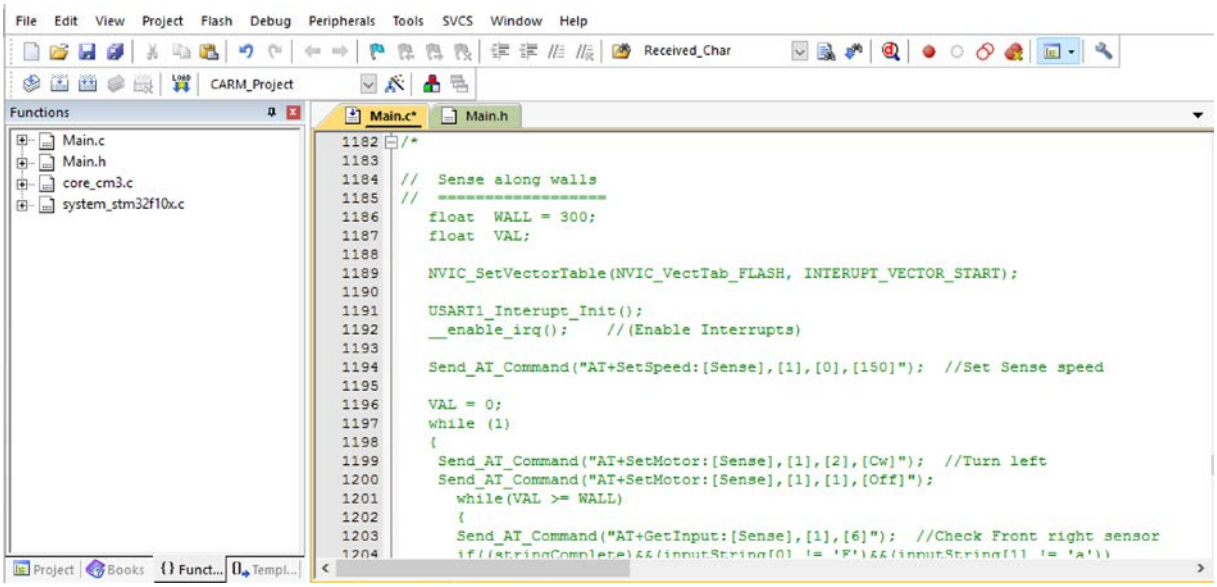
14. Press the **Stop**  to stop the program's running.

15. Activate the **/*** ***/** signs by deleting the two slashes at the beginning of each line.

    The program turns to green.

1.5.5 Moving along walls

1. Scroll down until you get the following screen:



This section contains the **Sense along walls** program.

2. Activate the program by making the two remark limit signs to remark lines by adding two slashes at the beginning of each line.

3. Observe the program and compare it with the exercise program.

```
int main (void)
{
  float  WALL = 300;
  float  VAL;

  NVIC_SetVectorTable(NVIC_VectTab_FLASH, INTERUPT_VECTOR_START);

  USART1_Interupt_Init();
  __enable_irq();     //(Enable Interrupts)

  Send_AT_Command("AT+SetSpeed:[Sense],[1],[0],[150]");          //Set Sense speed

  VAL = 0;
  while (1)
  {
     Send_AT_Command("AT+SetMotor:[Sense],[1],[2],[Cw]");       //Turn left
     Send_AT_Command("AT+SetMotor:[Sense],[1],[1],[Off]");
     while(VAL >= WALL)
     {
         Send_AT_Command("AT+GetInput:[Sense],[1],[6]");  //Check Front right sensor
         if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
         {
                  InputString_to_val();     //update inputStringVal
                  VAL = inputStringVal;
          }
     }

    Send_AT_Command("AT+SetMotor:[Sense],[1],[1],[Cw]");        //Turn right
    Send_AT_Command("AT+SetMotor:[Sense],[1],[2],[Off]");
    while(VAL < WALL)
    {
         Send_AT_Command("AT+GetInput:[Sense],[1],[6]");  //Check Front right sensor
         if((stringComplete)&&(inputString[0] != 'F')&&(inputString[1] != 'a'))
         {
                  InputString_to_val();     //update inputStringVal
                  VAL = inputStringVal;
         }
     }
  }
}
```

Note:
Pay attention to the compare signs (< and >).

4. Save the program.

5. Compile the program and check for errors.

6. Put the SENSE on the left side of the box.

7. Download the program by clicking on the **Download**  button.

8. Click on the **Run**  button to run the program.

   The SENSE should move around the box.

9. Change the value of the Black variable to create smooth movement of the SENSE.

1.5.7 Challenge exercises – Moving along walls

Task 1: Improve the program so the SENSE goes forward when it does not sense a wall on its right side.

The SENSE stops when it meets a wall, turns to the left and starts moving along this wall.

Save this program under the name **WALL2**.

Activate the **/*** ***/** signs by deleting the two slashes at the beginning of each line.

The program turns to green.

Challenge 1.6 – Counting

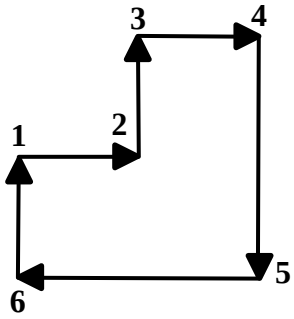
Draw block lines on a white paper.

Create a program that moves the robot through the block lines and make it stop on the fourth line.

Use variables to count the lines.

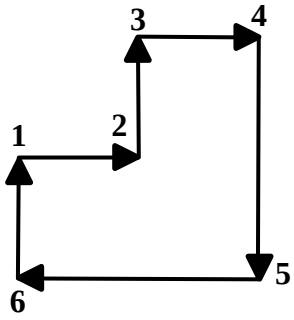
Challenge 1.7 – Automatic movement

Create a program that moves the robot according to the following figure:



Challenge 1.8 – Loops

Use loop commands to make the robot do the following cycles 3 times.



Challenge 1.9 – Loops and procedures

Convert each turn and forward movements into a procedure so the main program will have only the loop and run procedure instructions.

The program should do the same as the program in challenge 1.8.

Challenge 1.10 – "Don't touch me" robot

Create a "Do not touch me" program.

The robot should move away when you bring your hand close to one of its range sensors.

Challenge 1.11 – Robots in a convoy

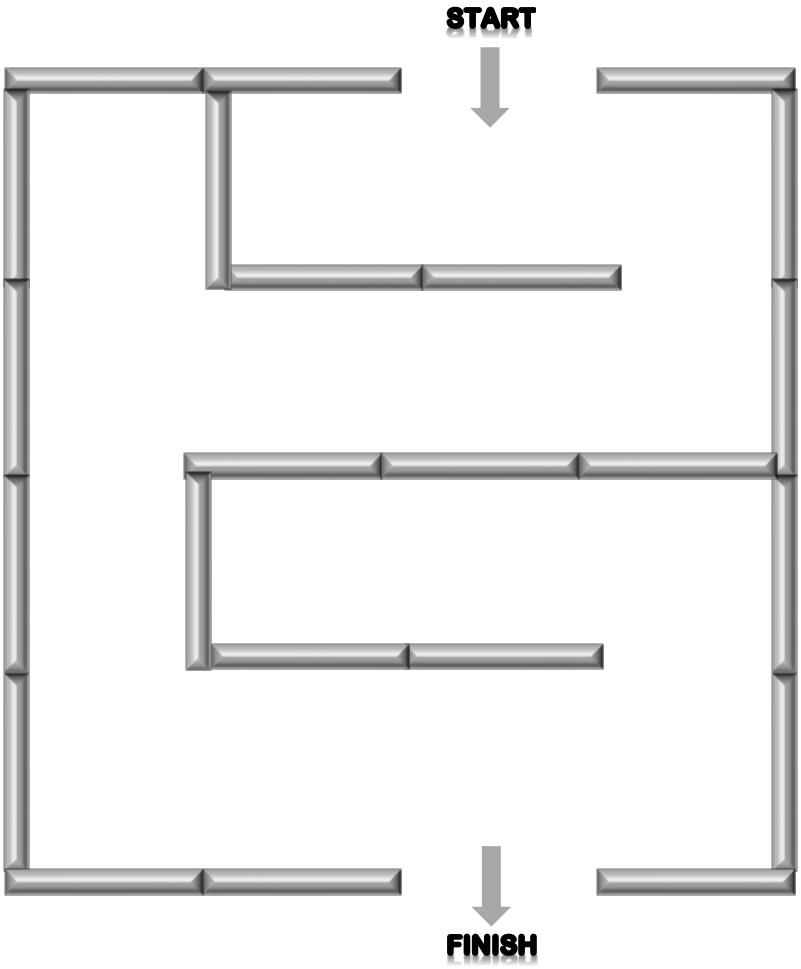
Put two robots on a black line.

The first robot should move along the black line and stop every 10 seconds.

The second robot should move along the black line and stop when it is close to the first robot.

Challenge 1.12 – Movement in a labyrinth

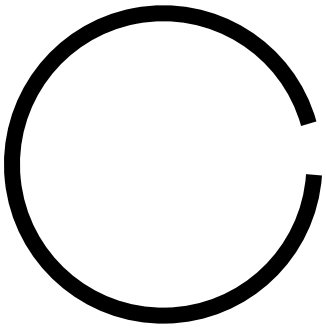
Build a labyrinth as follows:



Create a program that moves the robot from the START point to the FINISH point without touching the walls.

Challenge 1.13 – Exiting a circle

Draw a wide black line as follows:



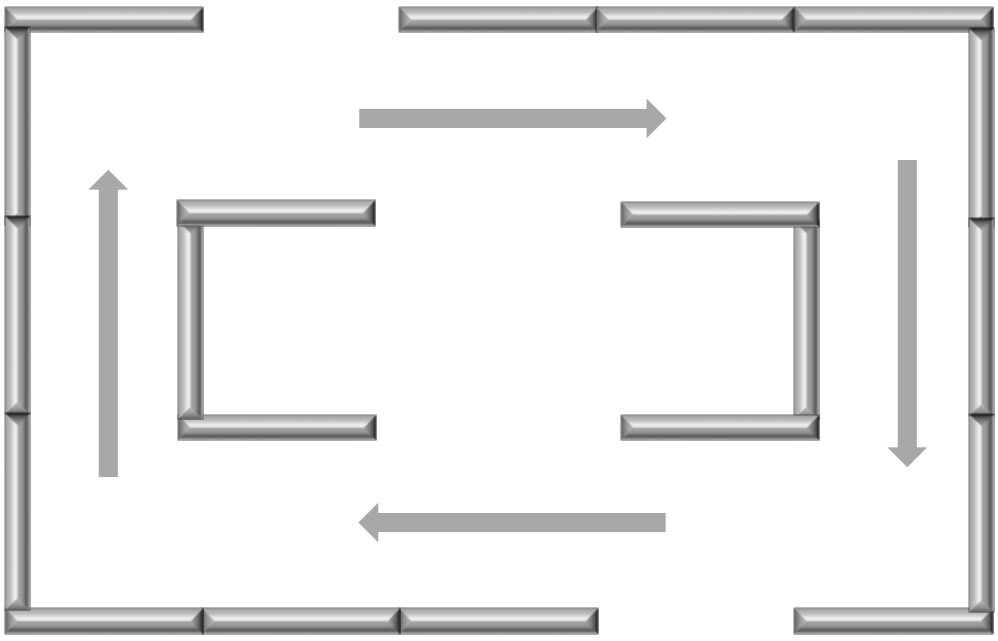
Put the robot inside the circle.

The robot should not cross the black line or move along the black line.

Create a program that makes the robot exit the circle according to the above rules.

Challenge 1.14 – Moving along corridors

Build the following corridor model:



The corridor's and the doors' widths are about 20cm.

The robot should move in the corridor without getting out through the doors.

Create a program that answers this challenge.

Chapter 2 – Brain Units

2.1 Brain units

Some of the input units can have their own "brain". The NeuLog sensors are such brain units. They send to the control unit, upon request, processed data such as: temperature (^{o}C or ^{o}F), light intensity in Lux, distance in meters, etc.

The output units can also be brain units. For example, units that control the motor speed and direction, lamp intensity, servo motor angle, etc.

These brain units are connected in a chain to the main control unit, which communicates with them through messages.

Every brain unit has an ID number. Every message from the control unit starts with ID number. Only the brain unit with this ID number interprets the message and executes it.

This system construction is the way modern systems are built, and has important advantages:

1. It creates a system with much less wires. The wires go from one module to another and not from all modules to the control unit.

2. This kind of system can easily be changed and expanded, and does not depend on the control units number of inputs and outputs.

The experiments in this chapter use the following brain units:

- NeuLog light sensor (NUL-204)
- NeuLog sound sensor (NUL-212)
- NeuLog motion sensor (NUL-213)
- NeuLog magnetic sensor (NUL-214)
- Brain tracking unit (SNS-101)
- Brain gripper arm (SNS-167)

If you do not have them, you can read about them and move to chapter 3.

Chapter 3 experiments are with The SENSE robot and battery module.

2.2 NeuLog sensors as brain units

NeuLog sensors (Neuron Logger Sensors) are also brain units. Each sensor includes a tiny computer, which samples, processes and stores the sampled data. Each probe connected to the sensor is pre-calibrated in the factory and no further calibration is required.

The data provided by the sensor is processed digital data. The sensor includes different measurement ranges. Changing the measuring range or type of processing is done simply on the computer screen with NeuLog software.

The sensors are plugged to each other with almost no limitation on the composition and number of sensors in the chain.

NeuLog has over 50 different sensors. Some sensors perform as two to three sensors.

The SENSE has three sockets for NeuLog sensors.

Experiment 2.1 – Sound Sensor

Objectives:

- The sound sensor.
- Operating the SENSE by sound.

Equipment required:

- Computer
- SENSE autonomous
- CARM-202 C coding unit
- BAT-202 Battery module
- NUL-212 NeuLog sound sensor

Discussion:

The sound sensor uses an internal microphone and special amplifier. Sound waves enter through the hole in the top of the sensor's plastic body so you should point that directly towards the sound source for best readings.

The sound sensor has two modes (ranges) of operation:

1. **Arbitrary analog units (Arb)** – An arbitrary unit indicates a number according to signal shape. At this mode, the sound is sampled and reconstructed as a signal.

2. **Decibel (dB)** – A unit of measure to show the intensity (loudness of sound). Please note that this is a logarithmic unit.

   At this mode, the wave is sampled and the average intensity (calculated by the sensor controller) is converted into dB value. 40 dB represents silence.

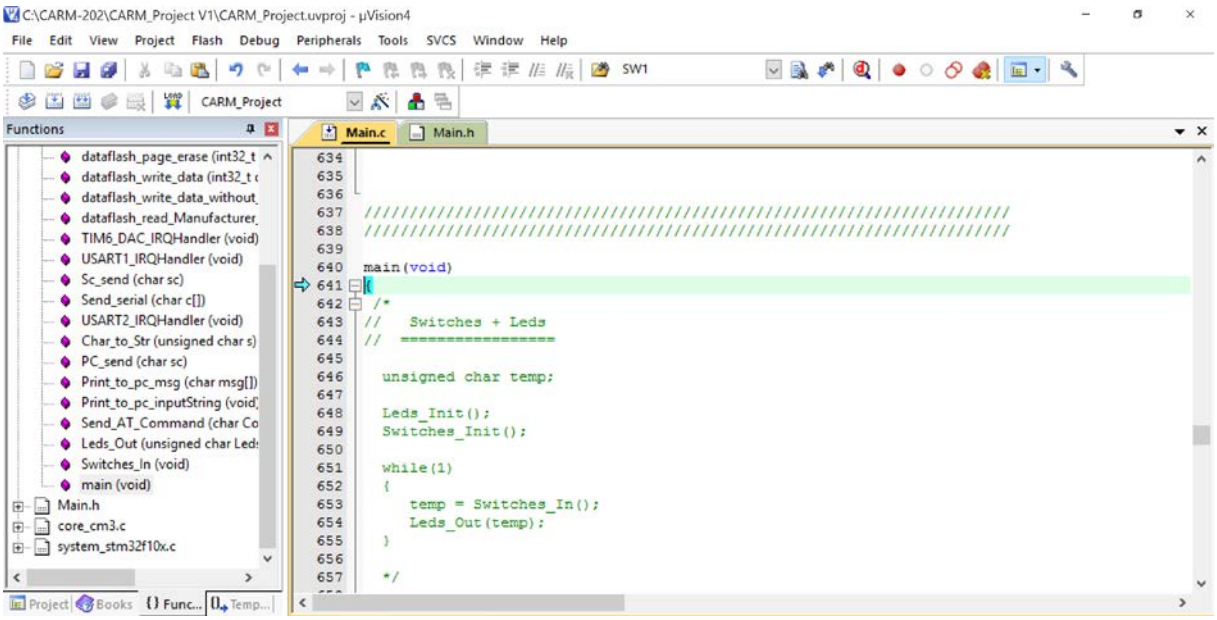
In this experiment, we shall use it at dB mode and we assume its ID is 1 as the default ID.

Selecting the range should be done by the NeuLog software.

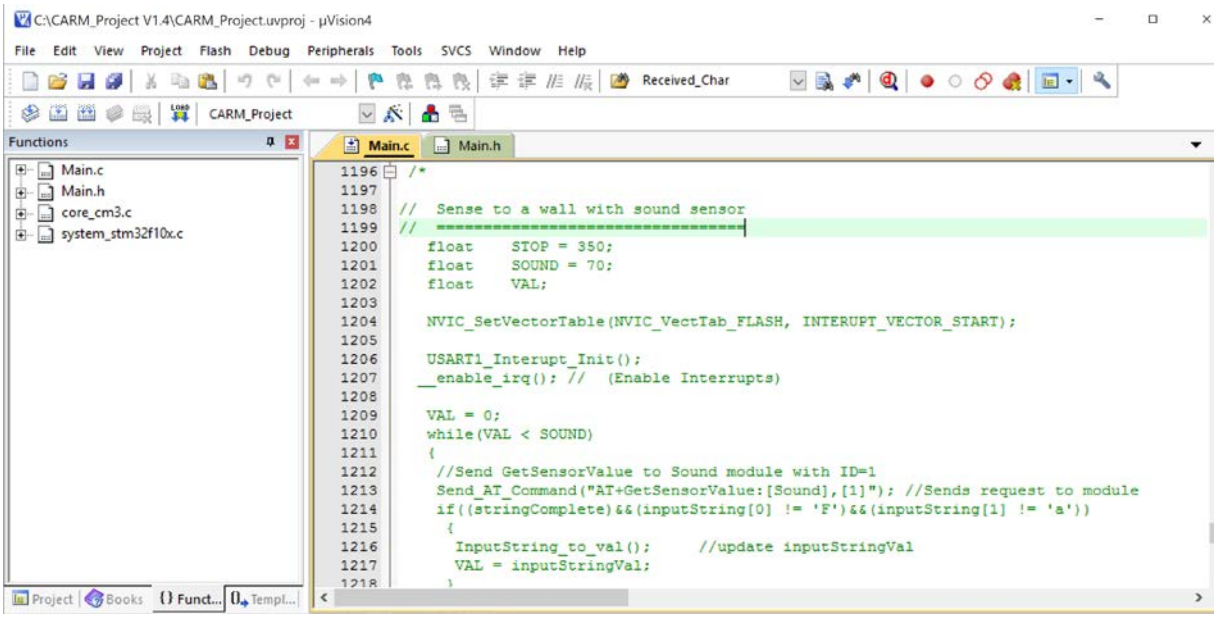
Procedure:

1. Enter the CARM_Project library and double click on the file **CARM_Project.uvproj**.

2. Check that **main.c** and **main.h** are open as in the following screen.

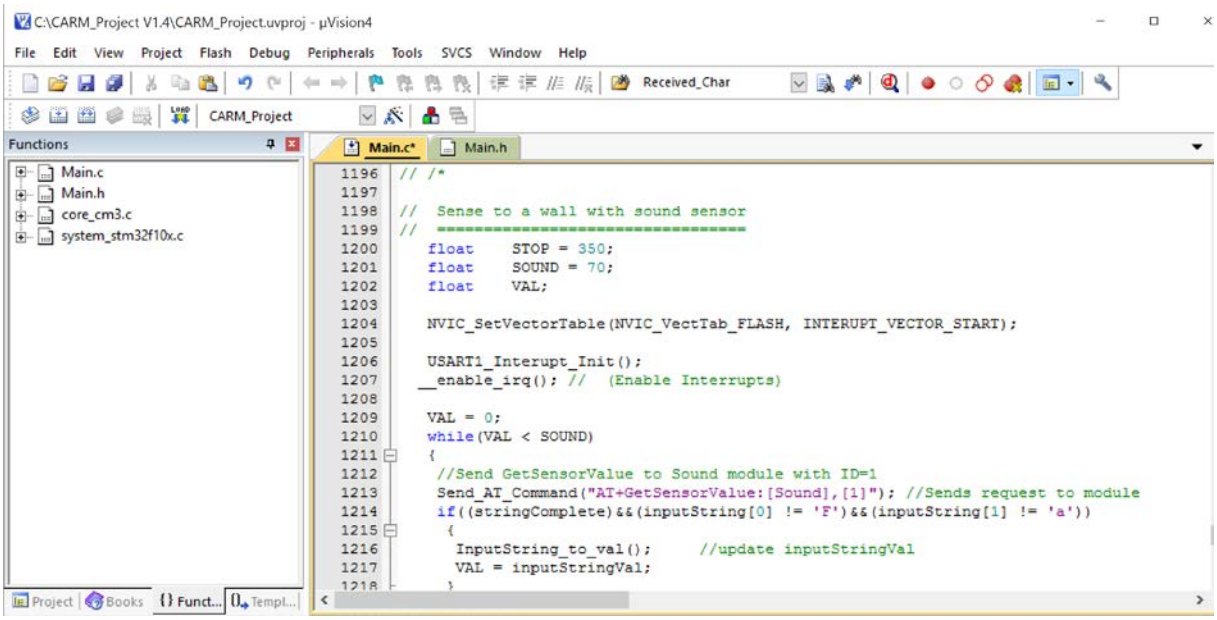


3. Scroll down the **main()** program until you get the following screen:



This section contains the **Sense to a wall with sound sensor** program.

4. Activate the program by making the two remark limit signs to remark lines by adding two slashes at the beginning of each line and you get the following screen:



5. Observe the program and make sure that you understand all of its instructions.

   Pay attention to the instruction:

Send_AT_Command("AT+GetSensorValue:[Sound],[1]"); //Sends request to module

   The **AT+GetInput:** is replace by **AT+GetSensorValue:**

6. Save the program.

7. Compile the program and check for errors.

8. Place the SENSE in front of a wall.

9. Download the program by clicking on the **Download**  button.

10. Disconnect the SENSE from the computer.

11. Press the CARM-202 **Run** button.

    The SENSE should not move.

12. Clap your hand or make high sound.

    The SENSE should move towards the wall and stop.

2.1.1 Challenge exercise – Wait for sound

Task 1: Improve the program so:

(a) The SENSE will wait for a sound above 70 dB, then moves forward until it meets a wall and then stops for 5 seconds.
(b) It will wait again for the sound, moves backward until it reaches a black line and then stops for 5 seconds.
(c) Returns to the beginning.

Activate the **/*** ***/** signs by deleting the two slashes at the beginning of each line.

The program turns to green.

Experiment 2.2 – Motion Sensor

Objectives:

- The motion sensor as distance sensor.
- Moving the robot according to the motion sensor.

Equipment required:

- Computer
- SENSE autonomous
- CARM-202 C coding unit
- BAT-202 Battery module
- NUL-213 NeuLog motion sensor

Discussion:

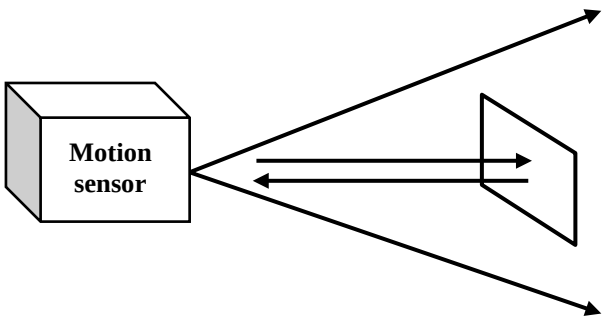
The motion sensor uses an ultrasonic transducer to both transmit an ultrasonic wave, and to measure its echo return. Objects in the range of 0.15 to 10 meters can accurately be measured to give distance, velocity, and acceleration readings using this method.

The motion sensor can collect data using the following measuring units:

- **Meters (m)** – The SI (International System of Units) distance unit
- **Meters/second (m/s)** – The SI velocity unit, which measures the distance traveled over time.
- **Meters/second^2 (m/s^2)** – The SI acceleration unit, which measures the change in velocity over time.

The motion sensor has two working ranges – one between 0.2 and 10.0 meters and one between 0.15 to 2 meters.

Ultrasonic waves are emitted from the sensor and spread out in a cone pattern at about 15° around the point of reference.



The ultrasonic transducer is a device that can convert pulse train to transmitted ultrasonic pulses. These pulses can sense and convert back to electronic pulse train by another similar ultrasonic transducer, or by itself.

The ultrasonic transducer is based on ceramic crystal, which is cut in a certain way and is placed between two metal plates. The crystal is characterized by the piezoelectric effect. Electrical field changes between the plates create mechanical vibrations in the crystal.

The crystal has a resonance frequency. The mechanical vibrations and electrical reactions depend on this resonance frequency.

Supplying pulses to the crystal of the ultrasonic transducer (in a rate according to its frequency) causes it to vibrate and to transmit these pulses as an acoustic sound. This sound cannot be heard because it is above the hearing frequency range (usually it is at 40KHz).

The acoustic sound can be converted back to electronic pulses by another ultrasonic transducer or by the transmitter when it stops transmitting. The acoustic pulses vibrate this transducer and these vibrations are turned into voltage pulses.

The speed of the ultrasonic wave is about 300 m/s because it is a sound wave.

For distance measurement, a burst of the transducer frequency wave is sent and the system measures the time between the sending and the receiving.

$S = 300 \cdot t$

Velocity is calculated by the difference between two successive distances divided by the time between the samples (according to the sampling rate).

Acceleration is calculated the difference between two successive velocities divided by the time between the samples (according to the sampling rate).

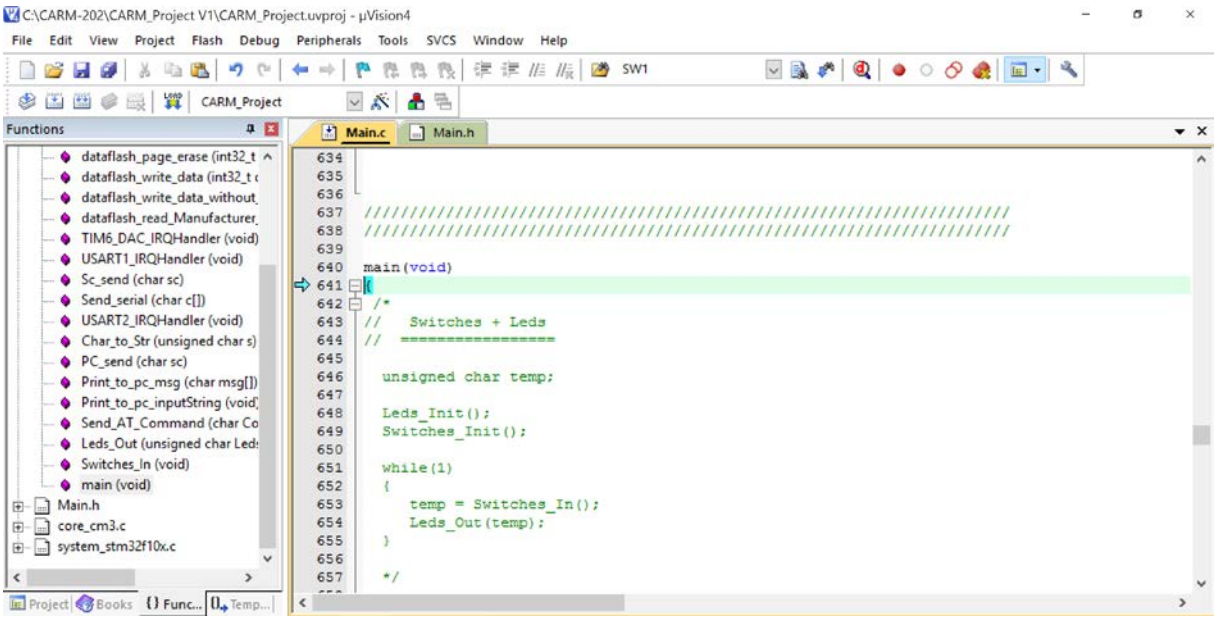
The motion sensor uses a very sophisticated method that enables it to measure long distance range with a low power of pulses.

In this experiment, we shall use it at distance range and we assume its ID is 1 as the default ID. Selecting the range should be done with the NeuLog software.

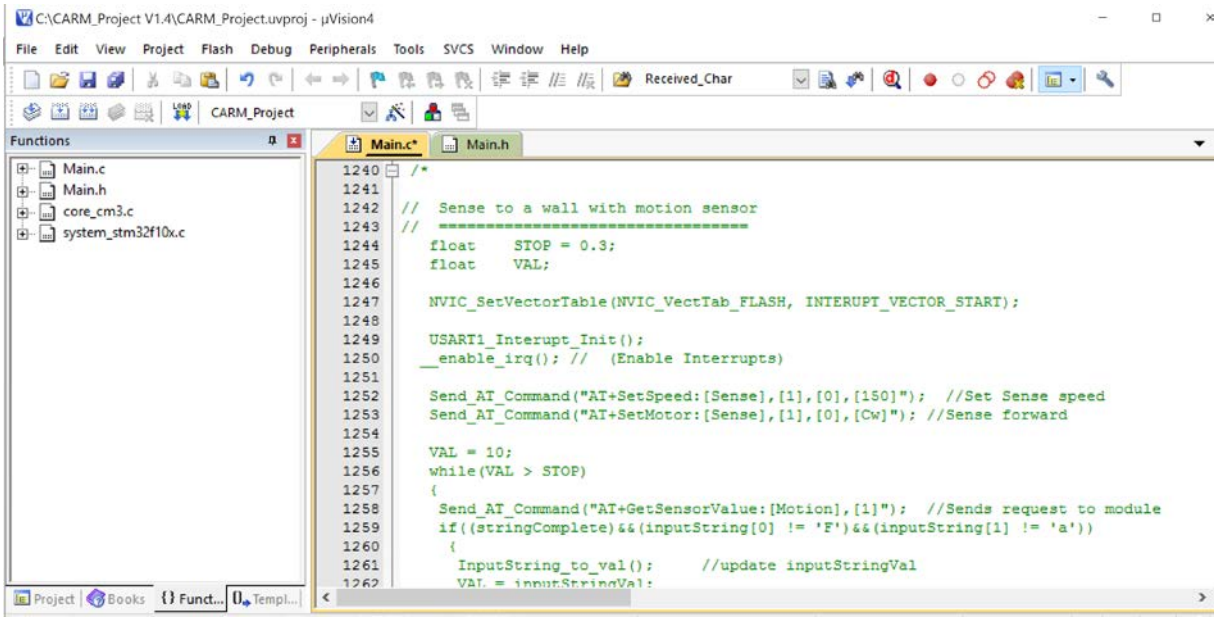
Procedure:

1. Enter the CARM_Project library and double click on the file **CARM_Project.uvproj**.

2. Check that **main.c** and **main.h** are open as in the following screen.

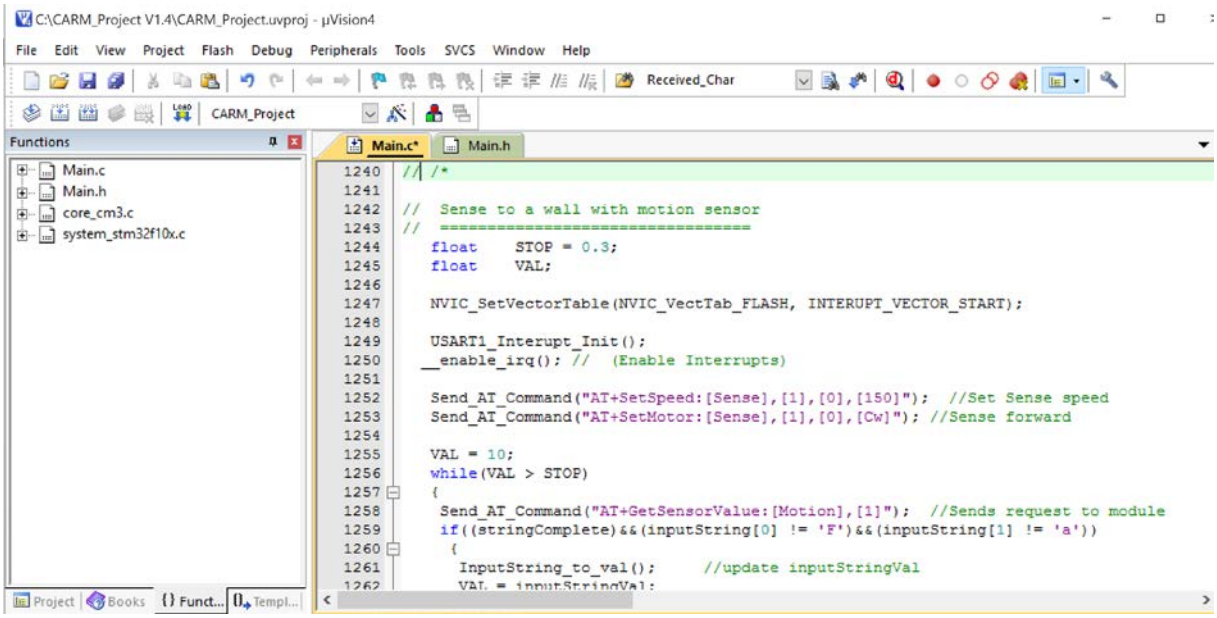


3. Scroll down the **main()** program until you get the following screen:



This section contains the **Sense to a wall with motion sensor** program.

4. Activate the program by making the two remark limit signs to remark lines by adding two slashes at the beginning of each line and you get the following screen:



5. Observe the program and make sure that you understand all of its instructions.

6. Save the program.

7. Compile the program and check for errors.

8. Plug the NeuLog motion sensor into one of the SENSE socket with its transducer directly to the front of the SENSE.

   You can plug the BAT-202 above the sensor or above the CARM-202.

9. Place the SENSE in front of a wall.

10. Download the program by clicking on the **Download**  button.

11. Disconnect the SENSE from the computer.

12. Press the CARM-202 **Run** button.

    The SENSE should move towards the wall and stop 30 cm away from it.

2.2.1 Challenge exercise – Moving in a distance range

Description: Going forward towards a wall, stop 30cm before the wall, then go backward and stop at 50cm from the wall and return.

Task 1: Improve the program so the SENSE will:

- move towards the wall,
- stop 30 cm in front of it,
- wait for 2 seconds,
- go backwards until a distance of 60cm,
- stop for 2 second,
- return to the beginning.

Activate the **/*** ***/** signs by deleting the two slashes at the beginning of each line.

The program turns to green.

Experiment 2.3 – Brain Tracking Unit

Objectives:

- The brain tracking unit.
- Moving to an IR (infrared) transmitter.
- Following an IR transmitter.

Equipment required:

- Computer
- SENSE autonomous
- CARM-202 C coding unit
- BAT-202 Battery module
- SNS-101 Brain tracking unit
- SNS-160IR transmitter

Discussion:

2.3.1 IR Transmitter

The infra-red transmitter can be plugged into any of the SENSE sockets or in the backup battery socket to be followed by the brain tracking unit.

Infrared light is transmitted from a heat source. We cannot see the IR light. The frequency of this light is a little below the red light and this is why we call it infra (before) red.

The surrounding light does not affect this light much.

2.3.2 Brain tracking unit

The brain unit, in a rigid plastic case, can be plugged into one of the SENSE sockets.

The brain tracking unit has two IR (infrared) sensors that enables it to track the IR transmitter.

The two IR sensors are at the same line with an opaque partition between them.

When IR light falls on both of them, it means that the SENSE is in front of the IR light source.

When the SENSE is at angle to the light source, the IR light will fall only on one of the IR sensors.

The third IR sensor measures the environment IR light. The brain unit controller uses this measurement to eliminate the environment light.

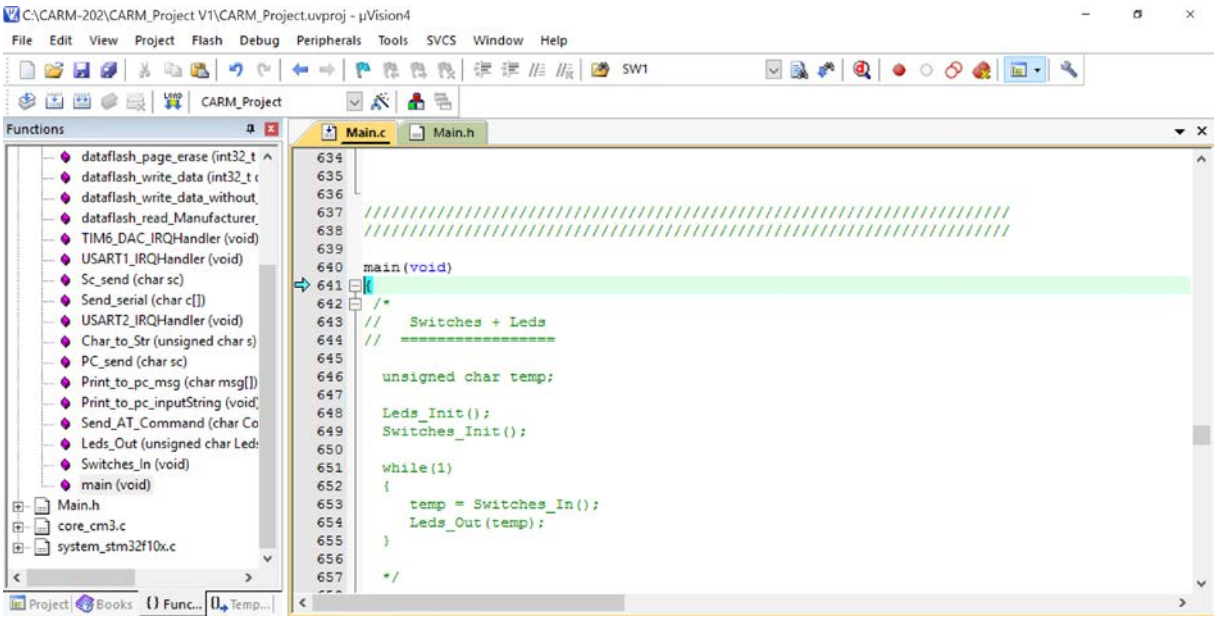
The brain unit output is a binary number that describes the detection status of an IR transmitter. This number is converted to detection results as the following:

0 – None (00) – No IR transmitter light
1 – Right (01) – IR transmitter light on the right
2 – Left (10) – IR transmitter light on the left
3 – Front (11) – IR transmitter light at front

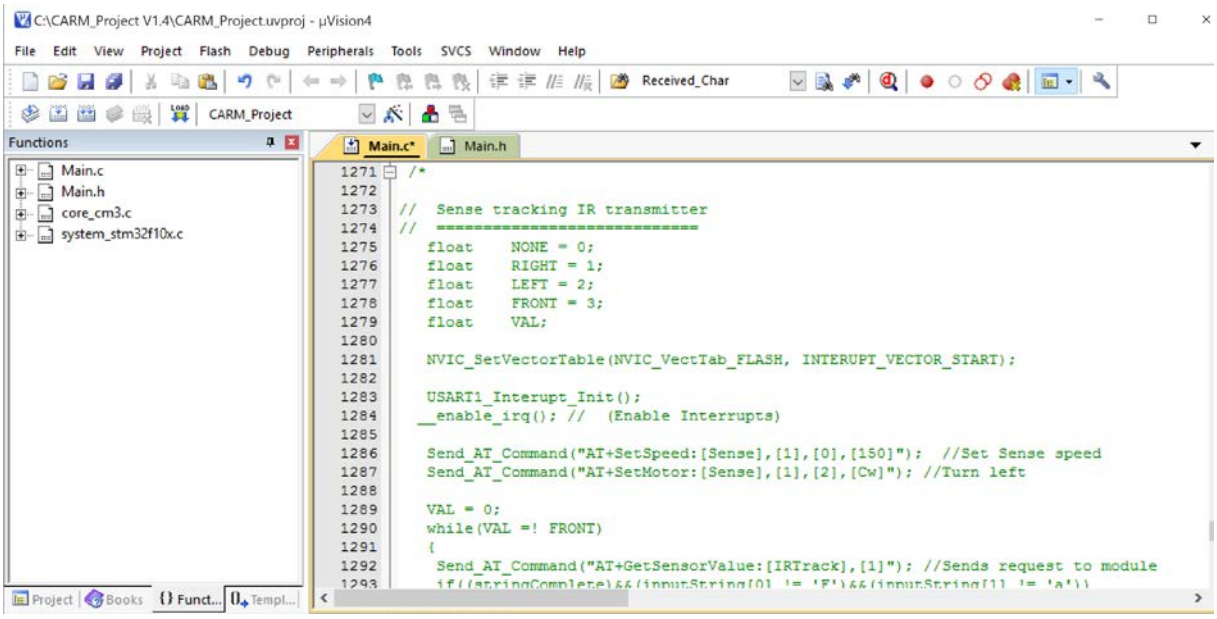
Procedure:

1. Enter the CARM_Project library and double click on the file **CARM_Project.uvproj**.

2. Check that **main.c** and **main.h** are open as in the following screen.

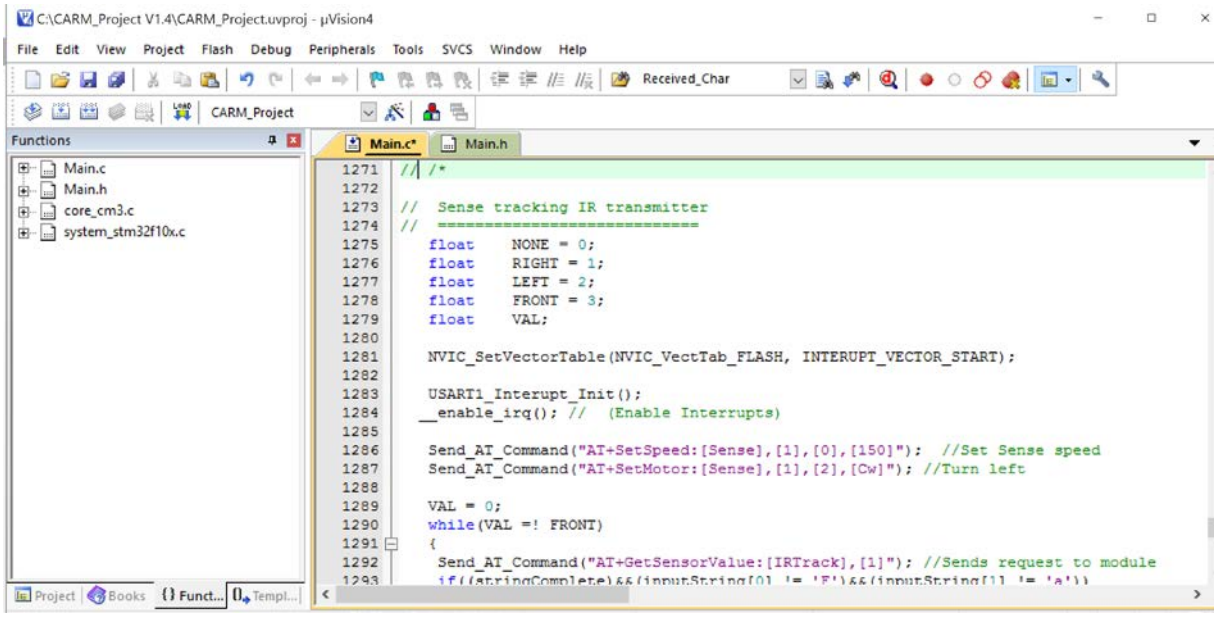


3. Scroll down the **main()** program until you get the following screen:



This section contains the **Sense tracking IR transmitter** program.

4. Activate the program by making the two remark limit signs to remark lines by adding two slashes at the beginning of each line and you get the following screen:



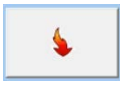
5. Observe the program and make sure that you understand all of its instructions.

6. Save the program.

7. Compile the program and check for errors.

8. Plug the brain tracking unit into the front socket of the SENSE.

9. Plug the IR transmitter into a backup battery.

10. Download the program by clicking on the **Download**  button.

11. Disconnect the SENSE from the computer.

12. Press the CARM-202 **Run** button.

    The SENSE should rotate to the left searching the IR transmitter.

13. Bring the IR transmitter to be in front of the SENSE.

    The SENSE should stop rotating.

14. Move the IR transmitter to the left and to the right.

    The SENSE should follow it.

2.3.3 Challenge exercise – Tracking a robot with IR transmitter

Task 1: Improve the program to move the SENSE towards the IR transmitter. The SENSE waits when it does not detect the IR light.

Task 2: Improve the above program and procedures so the SENSE will stop in front of the IR transmitter.

Put the IR transmitter on a box or another SENSE that can be detected by the front sensor.

Activate the **/*** ***/** signs by deleting the two slashes at the beginning of each line.

The program turns to green.

Experiment 2.4 – Brain Gripper Arm

Objectives:

- The brain gripper arm.
- Moving an object from one place to another.
- Drawing pictures with the brain gripper arm.

Equipment required:

- Computer
- SENSE autonomous
- CARM-202 C coding unit
- BAT-202 Battery module
- SNS-167 Brain gripper arm
- A wooden rod
- A marker

Discussion:

2.4.1 Brain gripper arm

The brain gripper arm has two servo motors.

One servo motor moves the gripper up and down.

The second servo motor opens and closes the gripper.

A servo motor is a motor with feedback. The feedback can be voltage according to the motor speed or the shaft angle, electrical pulses according to the motor shaft rotation and direction, and more.

Each servo motor of the gripper arm has transmission gear and potentiometer. The potentiometer consists of a variable resistor that create variable voltage according to the servo motor shaft angle.

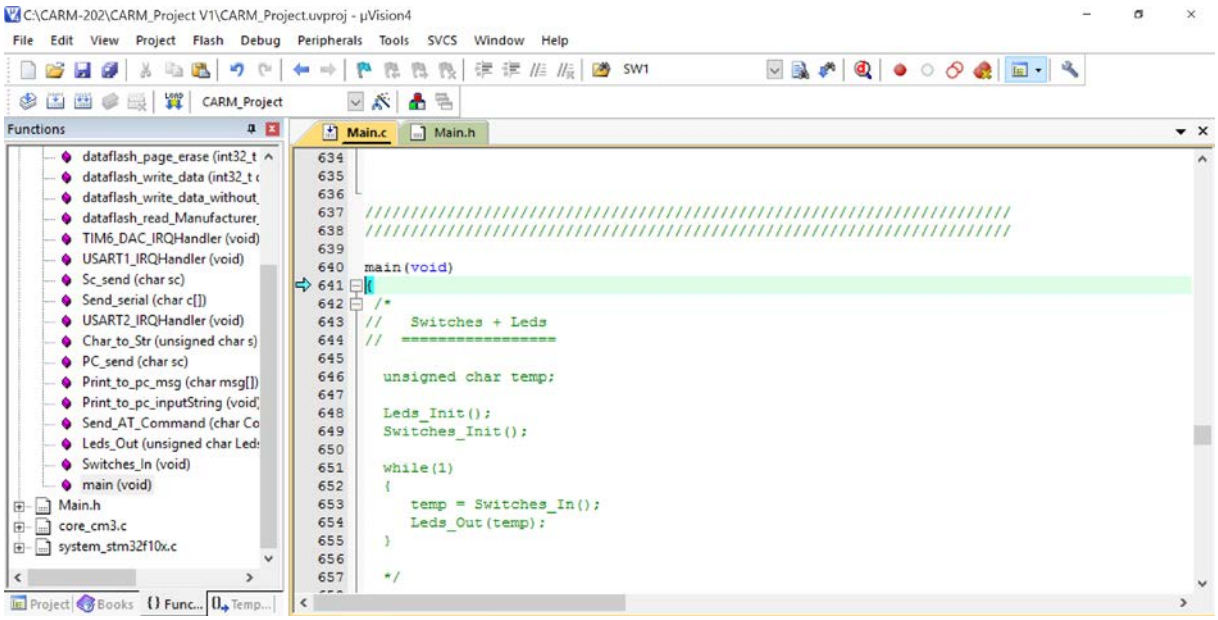
The brain controller of the gripper arm gets the required angle of the shaft. It turns the motor CW (Clock Wise) or CCW (Counter Clock Wise) until the potentiometer voltage suits this angle.

It checks the shaft angle all the time. If it changes mechanically, the controller will turn the motor ON to return the shaft to the right position.

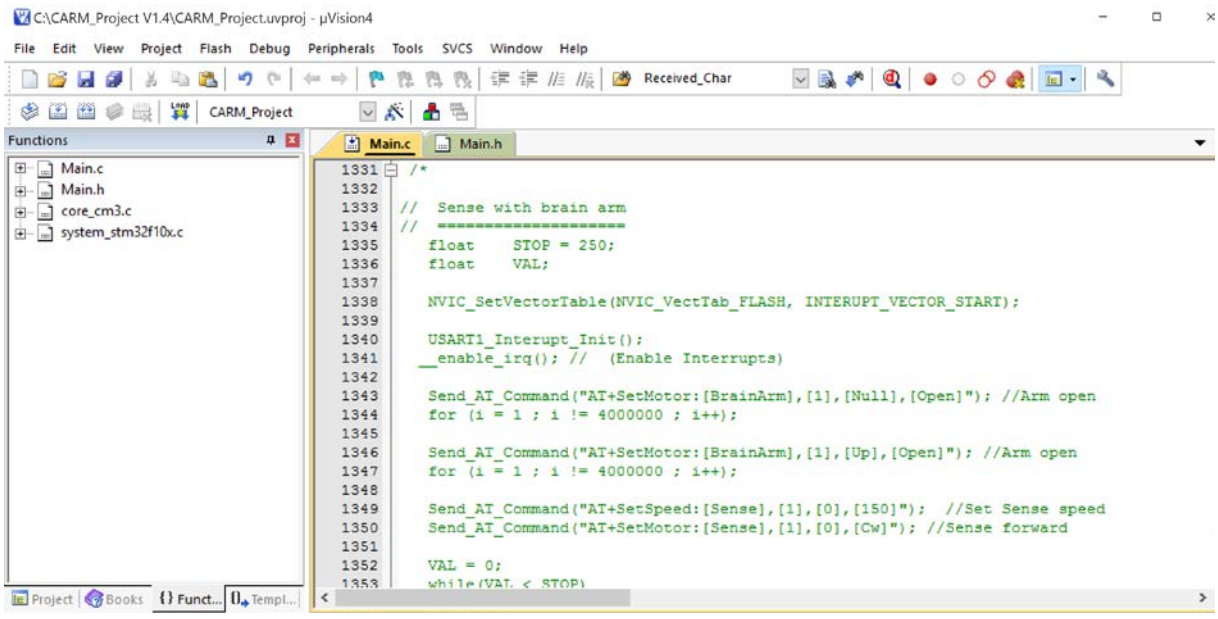
Procedure:

1. Enter the CARM_Project library and double click on the file **CARM_Project.uvproj**.

2. Check that **main.c** and **main.h** are open as in the following screen.

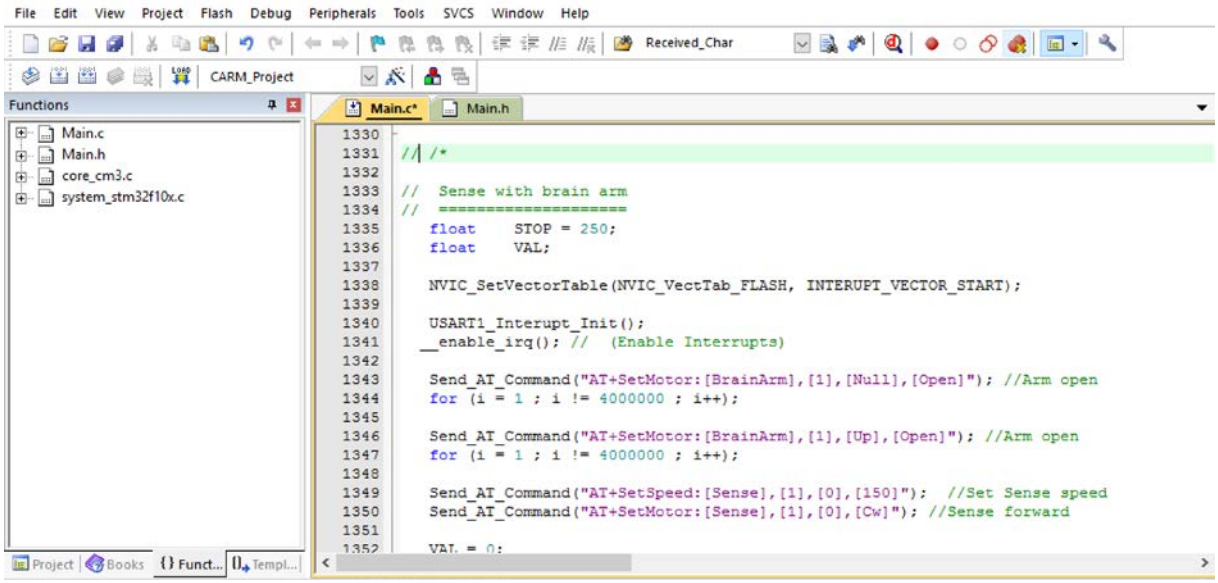


3. Scroll down the **main()** program until you get the following screen:



This section contains the **Sense with brain arm** program.

4. Activate the program by making the two remark limit signs to remark lines by adding two slashes at the beginning of each line and you get the following screen:



5. Observe the program and make sure that you understand all of its instructions.

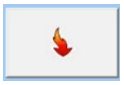
   The program should do the following:

   (a) Opens the gripper
   (b) Raises the arm
   (c) Moves the Sense forward and stops when the wooden rod between the gripper fingers
   (d) Lowers the arm to mid position
   (e) Closes the gripper
   (f) Raises the arm
   (g) Moves forward for 2 seconds
   (h) Lowers the arm to mid position
   (i) Opens the gripper
   (j) Moves backward for 2 seconds

6. Save the program.

7. Compile the program and check for errors.

8. Plug the brain gripper arm unit into the front socket of the SENSE.

9. Download the program by clicking on the **Download**  button.

10. Disconnect the SENSE from the computer.

11. Press the CARM-202 **Run** button.

    Check that the SENSE does its mission.

12. Change the program to run in endless loop.

13. Download the program, run and check the SENSE behavior.

2.4.2 Challenge exercises – The SENSE with gripper arm

Task 1: Change the last program to use functions instead of chain of instructions. Put the delays in the functions.

Task 2: Change the program to make the SENSE to rotate in about 90^{o} with the raised wooden rod before moving forward with it.

Task 3: Plug Sound sensor to the SENSE and make it wait for hand clapping before picking up the wooden rod.

Task 4: Make the gripper hold a marker manually.

Place the SENSE on wide white paper attached to the ground or to the desk.

Build some drawing programs.

1. Activate the **/*** ***/** signs by deleting the two slashes at the beginning of each line.

   The program turns to green.

Chapter 3 – Autonomous Vehicle Challenges

3.1 Autonomous vehicles

We are in the generation of autonomous vehicles, machine learning and artificial intelligence. This is the world of machines making decisions. The decisions are according to the software and programming behind. This is just the beginning.

We can understand this world and the occurring changes by trying to develop programs similar to autonomous car.

The SENSE is a tool for such challenge exercises.

This chapter introduces several of the challenge autonomous exercises. The idea is to let the user to think about algorithms and solutions to solve these challenges.

3.2 Programming tips

This chapter is built as a challenge exercise to solve with no guiding.

The Robockly and the Python are rich and powerful coding programs.

They have so any functions and options.

Start your solutions with the Robockly, but observe the Python program too.

When the screen of the Robockly becomes too loaded, move to Python.

Try to work more with functions and not with long chains of instructions.

There are multiple solutions. Try to find the most efficient way.

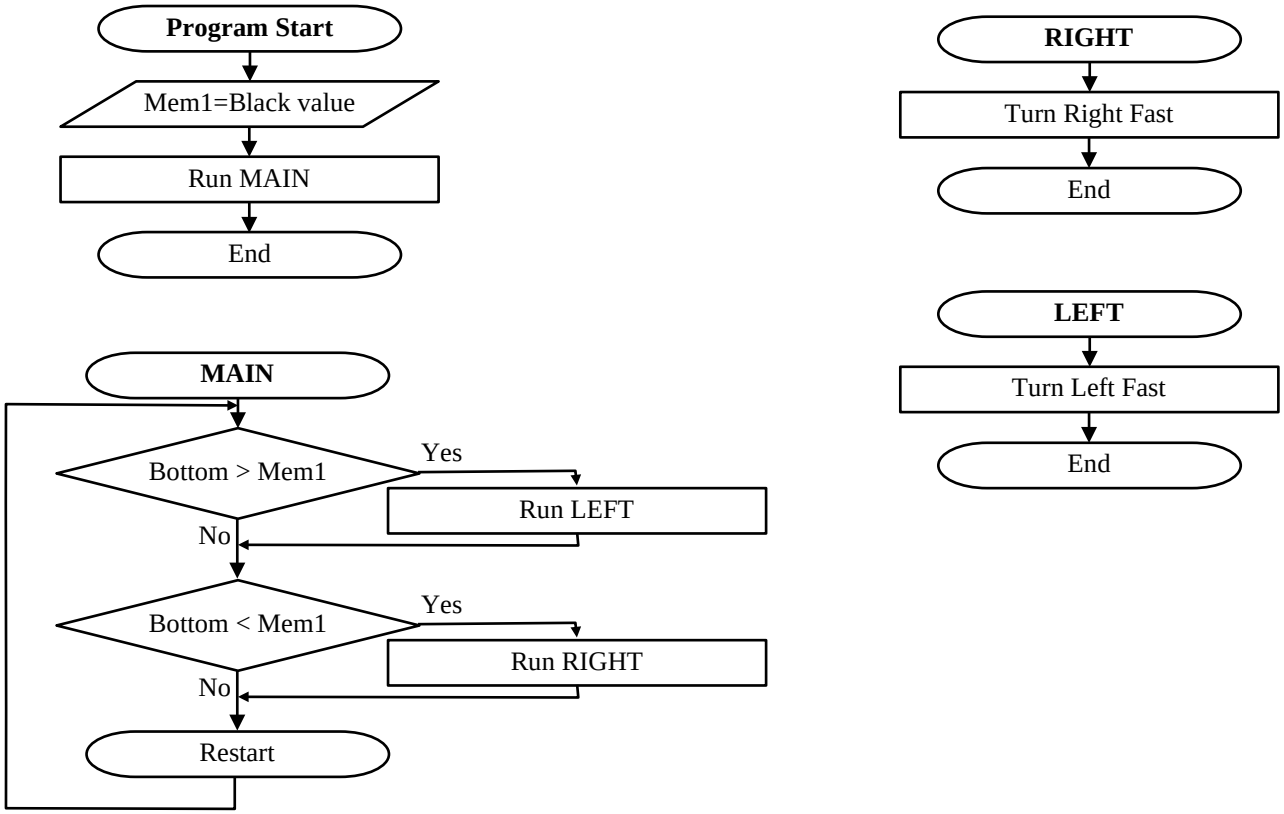
Do not be afraid to fail and to try repeatedly.

Good Luck!!

Challenge 3.1 – Along black lines

3.1.1 Left and right along a black line

The following is the flowchart of a simple movement along a black line program.



The robot moves by swinging on the edge of the black line.

Place the robot on the black line, read the bottom sensor value and set it in the program. This value may be different from one robot to another.

Convert the flowchart to a C language program.

Download, run and check to robot movement.

3.1.2 Smooth movement along a black line

In order to get a smoother movement we can replace one of the turn instructions with a deviate instruction. This depends on the robot movement direction.

When the robot moves counter clockwise, we shall replace the **Right turn** command with **Right deviate**.

When the robot moves clockwise, we shall replace the **Left turn** command with **Left deviate**.

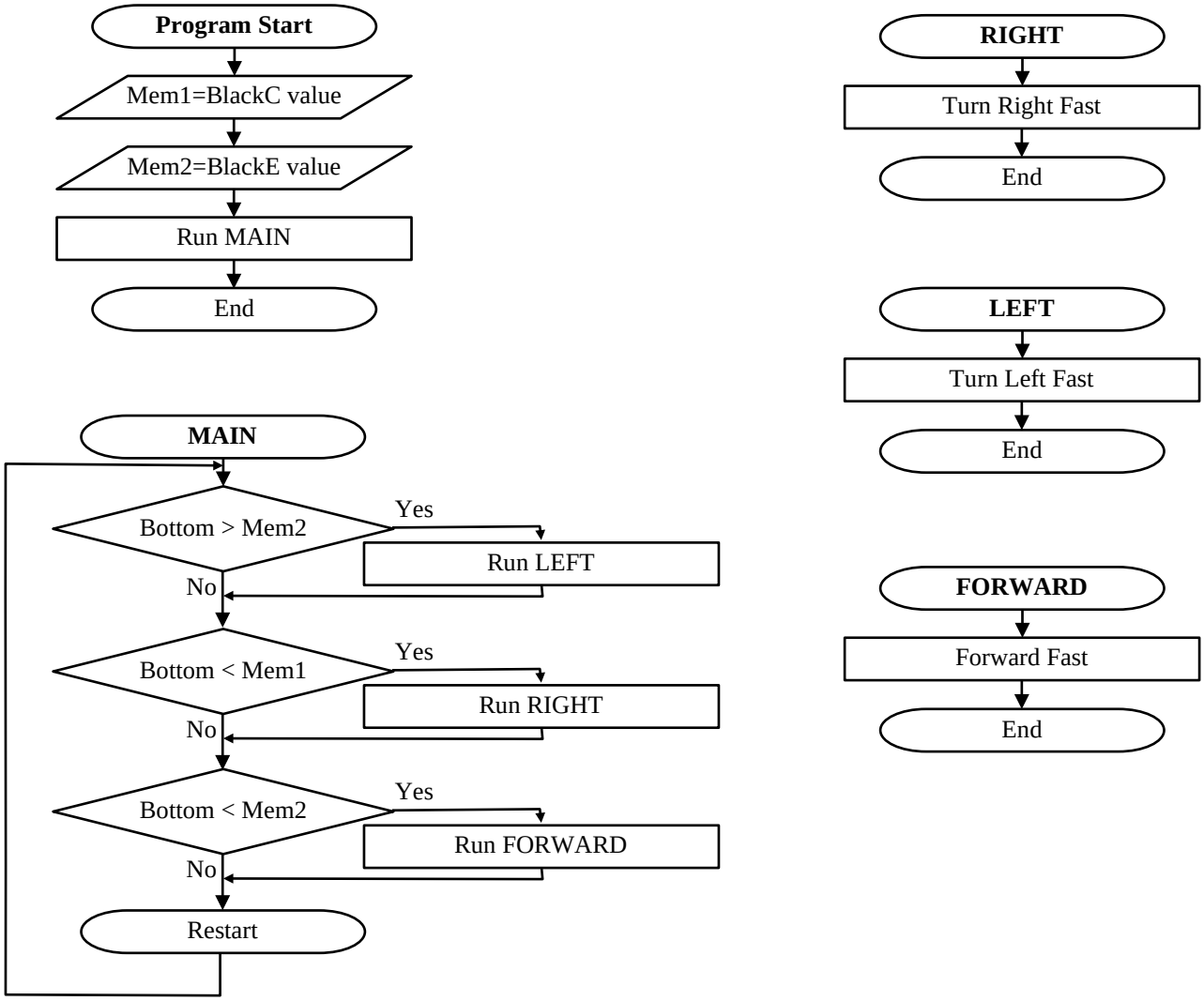
Change the program accordingly, download, run and check to robot movement.

3.1.3 Adding Forward movement

Check at **Direct** mode, the bottom sensor value when it is above the center of the black line and when it is closer to the edge of the line.

We shall call the read value at the center of the black line **BlackC** and the black value close to the edge **BlackE**.

The following program drives the robot forward when it is on the edge part of the black line.



Analyze the flowchart.

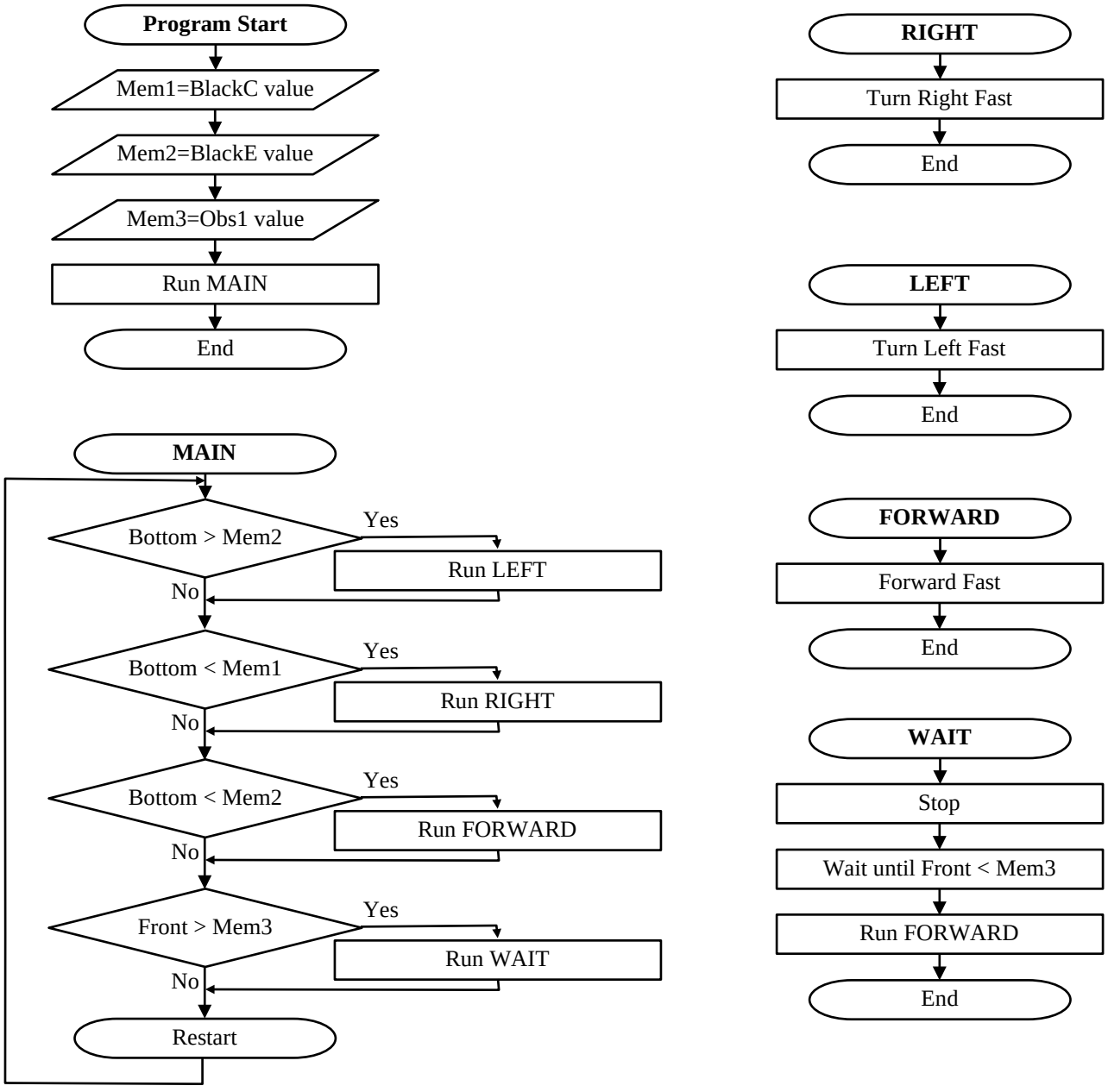
Change the program accordingly, download, run and check to robot movement.

3.1.4 Along a black line with a stop in front of an obstacle

Improve the previous program to stop in front of an obstacle until the obstacle is removed.

Put your hand in front of the SENSE and check at **Direct** mode, the front sensor value. We shall call the read value Obs1.

The following program drives the robot forward when it is on the edge part of the black line.

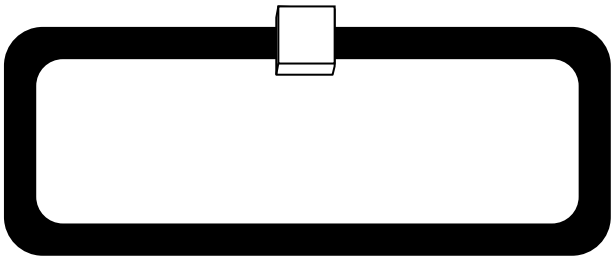


Analyze the flowchart. Change the program accordingly, download, run and check to robot movement.

Challenge 3.2 – Automatic Guided Vehicle (AGV)

An AGV is a vehicle or a cart that moves along guidelines. It is very popular in manufacturing places for transporting row materials or sub-assembly systems from one station to another.

Create the following line.



Put a small box on it as in the picture.

Write a program that moves the SENSE along the line and stops in front of the box for 5 seconds, turns around, moves on the other direction and vice versa.

The SENSE goes on the outer edge.

For this task we have two movements – clockwise and counter clockwise.

The main program should know what the current movement is. To determine that, we use what we call a flag. The main program operates the required procedure according to the value of a certain variable.

The value of this variable is changed when changing direction is needed.

Analyze the following flowchart. Memory 4 is the flag variable.

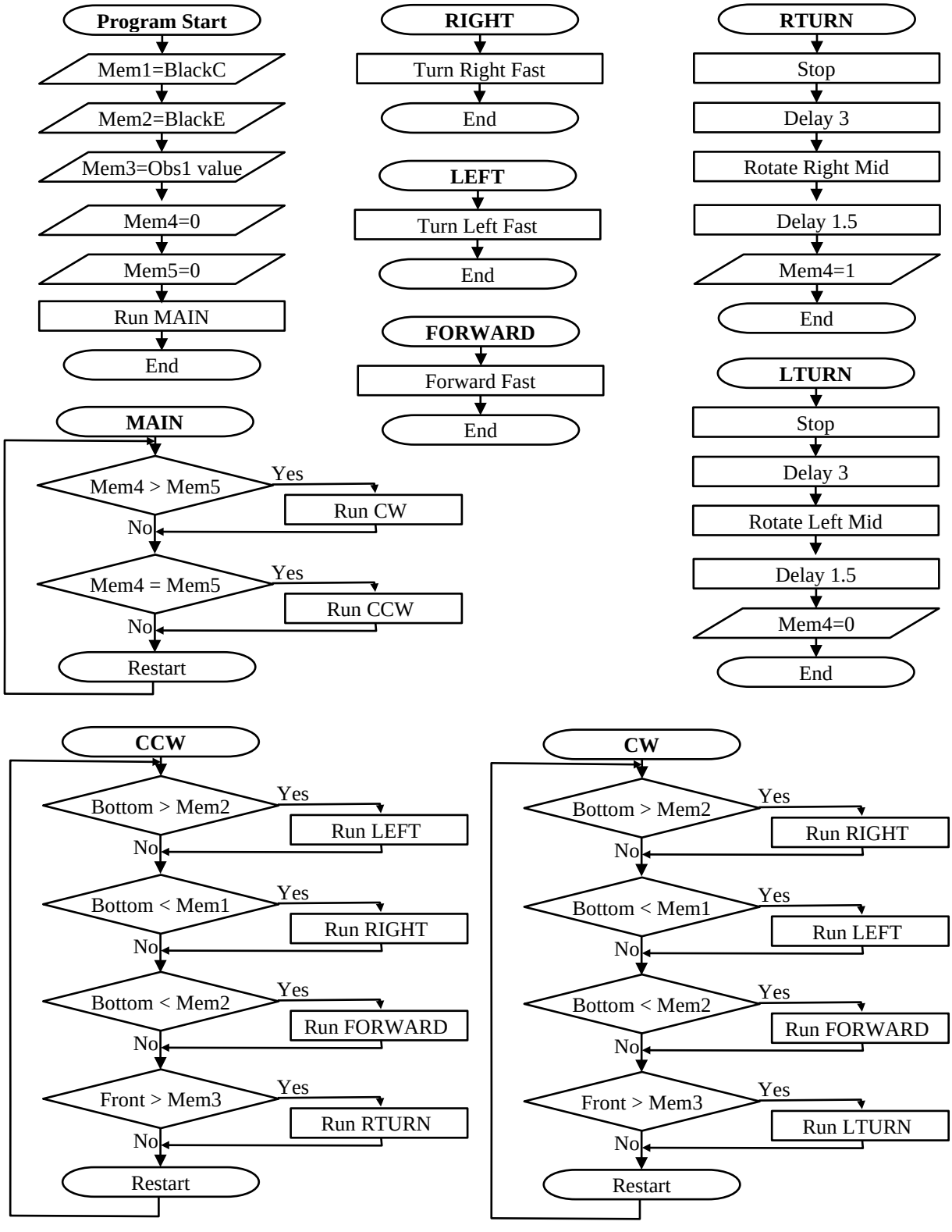
Build the program accordingly, download, run and check to robot movement.

Every robot behaves a little different.

Adapt the program to your robot sensors and behavior.

Take care to stop in front of the box in a distance that enables the robot to rotate.

AGV Program flowchart



Challenge 3.3 – AGV between stations

Create the following line with the boxes.

Write a program that moves the SENSE from one station to another along the lines in this order: 1-2-1-2-…

The Sense stops at each station and moves to the next station when you put your hand close to the right back sensor.

Hint:

The previous AGV program should answer the movement of the robot.

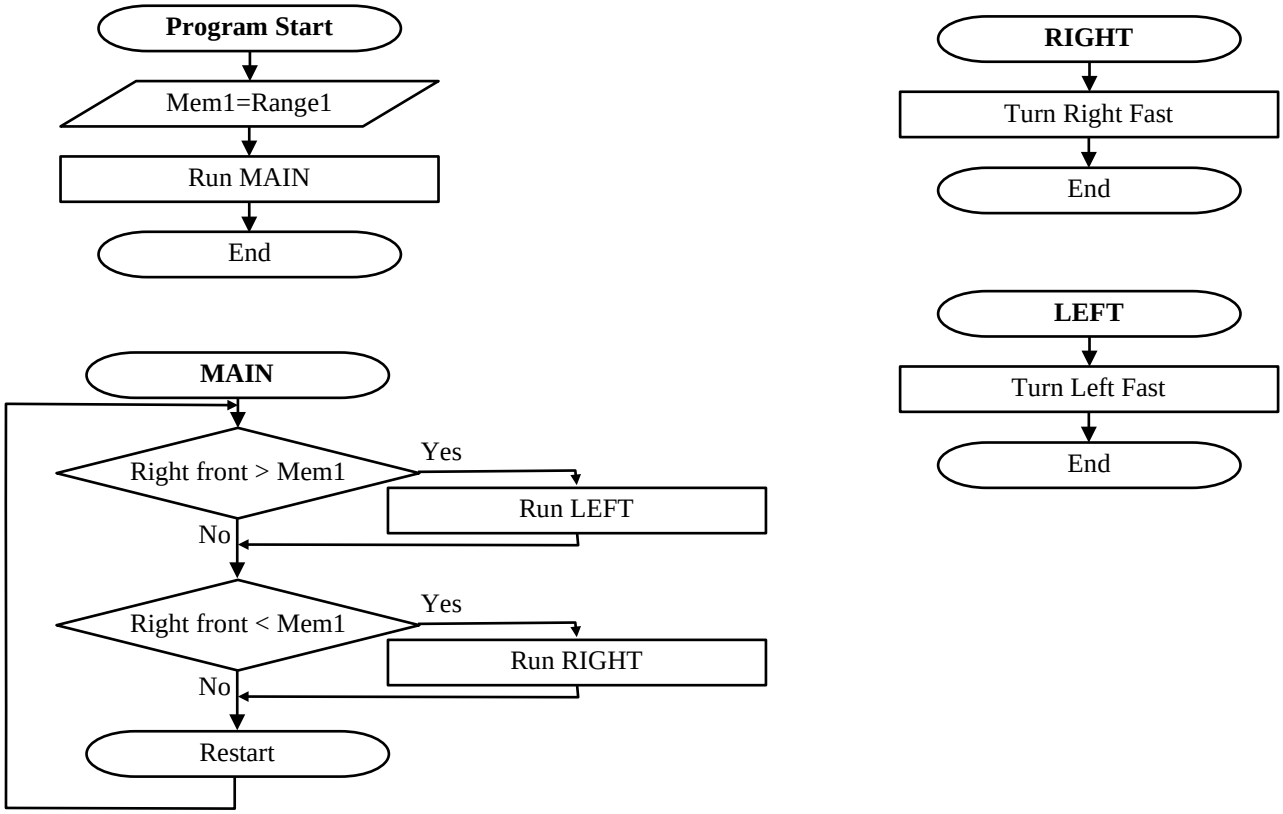
Challenge 3.4 – Along a building block

The following exercises deal with movement methods along walls and around a building block.

Use at least 40 X 40 cm box as a simulation of the building block.

3.4.1 Left and right along walls

The following is the flowchart of a simple movement along walls program.



The robot moves by swinging along a wall on its right side.

Place the robot near the box on its right side, read the right front sensor value and set it in the program. This value may be different from one robot to another.

Download, run and check the robot's movement.

3.4.2 Smooth movement along a black line

In order to get a smoother movement we can replace one of the turn instructions with a deviate instruction. This depends on the robot movement direction.

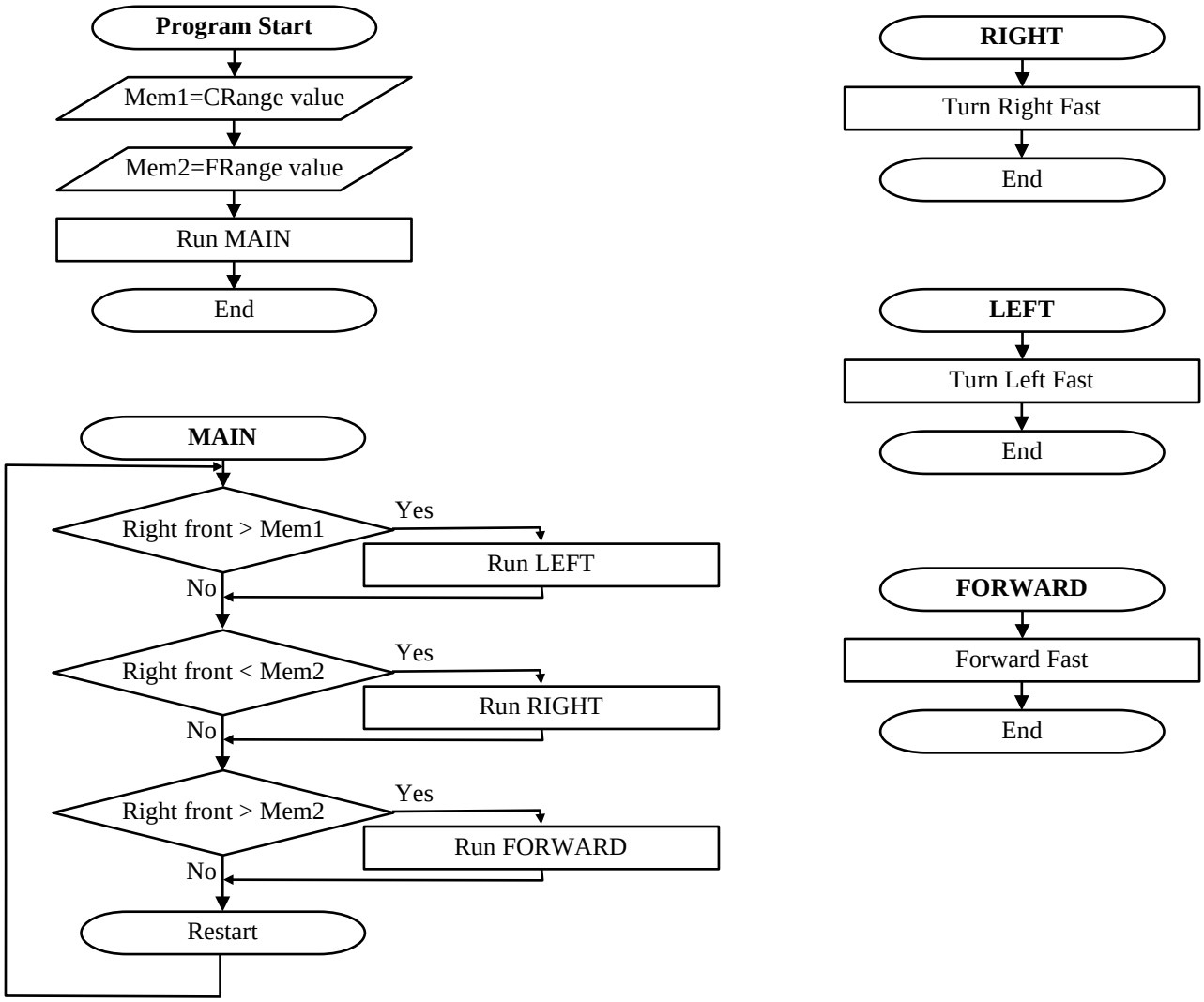
With the above program, the robot moves clockwise, we shall replace the **Left turn** command with **Left deviate**.

Change the program accordingly, download, run and check to robot movement.

3.4.3 Adding Forward movement

We can use two range values – **Crange** (close range) and **FRange** (far range).

The following program drives the robot forward when it is between CRange and FRange.



Analyze the flowchart.

We have to remember that the right front value increase when the robot come closer to the wall.

Determine the **CRange** value as the **Range1** value of the previous program.

Determine the **Frange** value as **CRange** – 10.

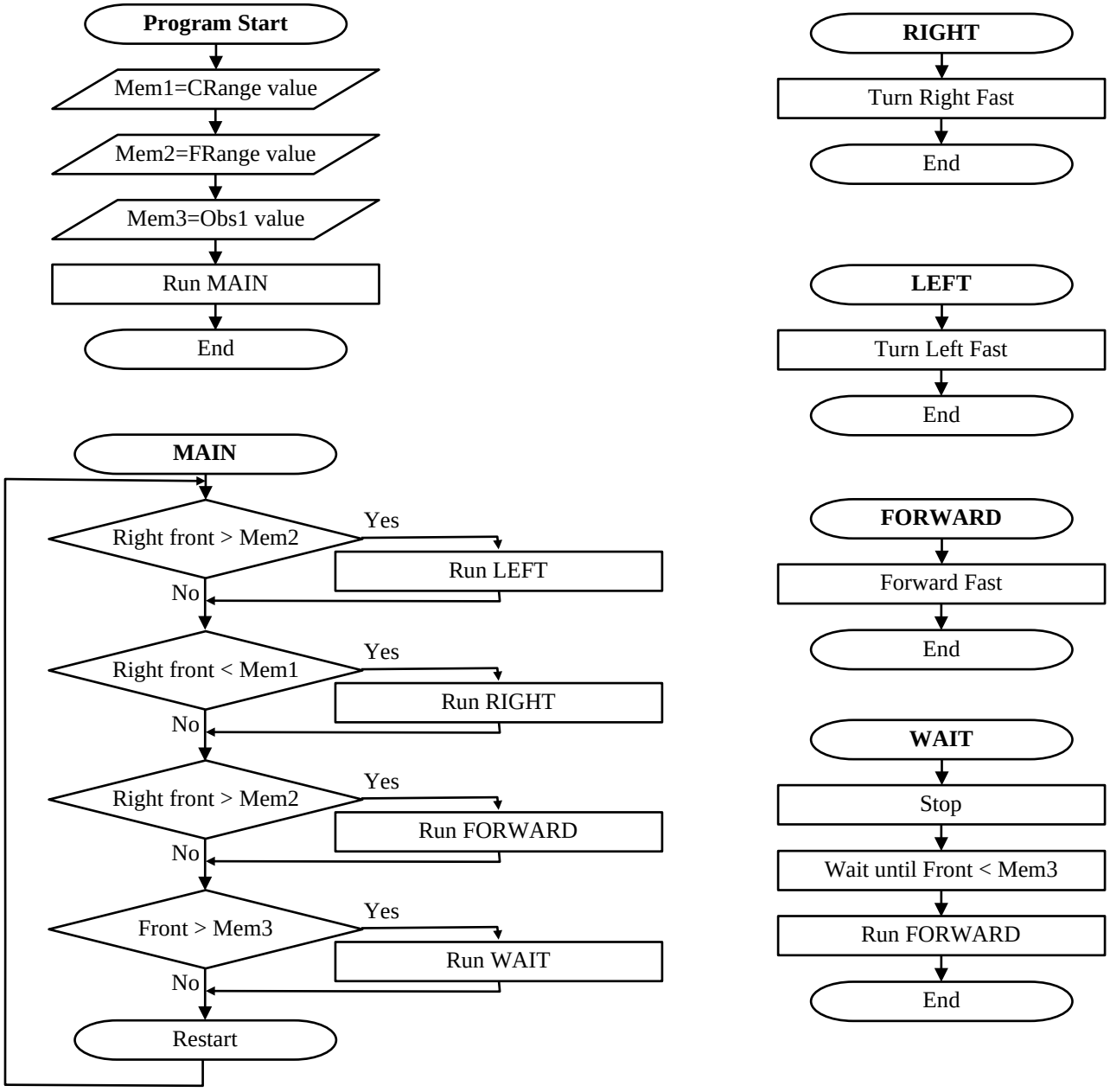
Change the program accordingly, download, run and check to robot movement.

3.4.4 Along a wall with a stop in front of an obstacle

Improve the previous program to stop in front of an obstacle until the obstacle is removed.

Put a small box in front of the SENSE and check at **Direct** mode, the front sensor value. We shall call the read value Obs1.

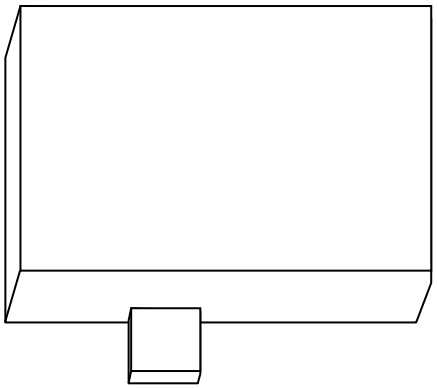
The following program drives the robot forward when it is on the edge part of the black line.



Analyze the flowchart. Change the program accordingly, download, run and check the robot's movement.

Challenge 3.5 – Along a building block and bypass cars

Put an obstacle, as described in the following picture.



Write a program, as in challenge 3.4.4, with SENSE bypassing the car. The SENSE should return to the right only after passing the car.

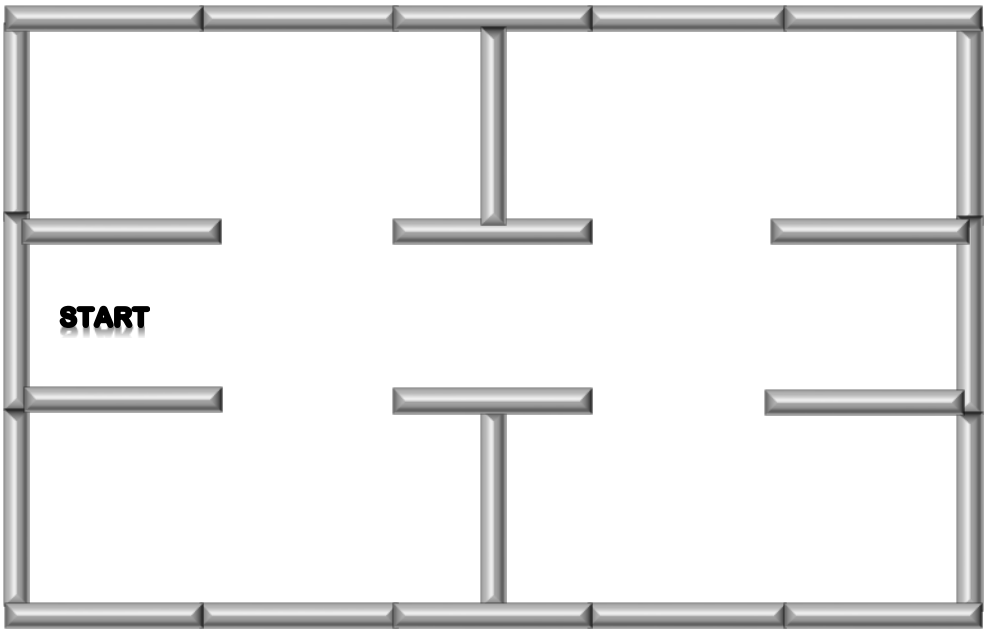
Hint:

Replace the **WAIT** procedure with **TURN** procedure.

The **TURN** procedure rotate to the left until the front sensor is below the **Obs1** value.

Challenge 3.6 – Autonomous museum guard

Build a model of a museum with rooms and corridors as follows:



Create a program that moves the robot along the walls through the museum rooms. The starting point is at the START position.

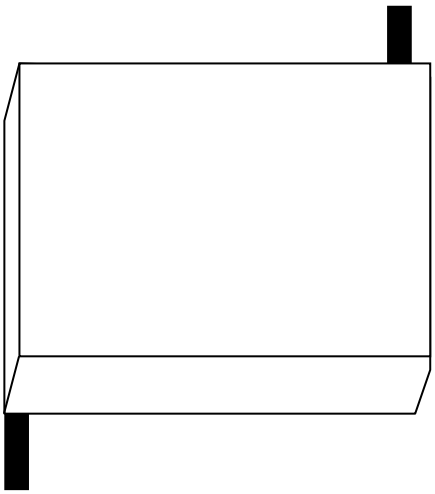
Hint:

The program of challenge 3.5 can serve as a solution for this task.

You have to adapt the memory values to the model.

Challenge 3.7 – Along a building block with stop sign

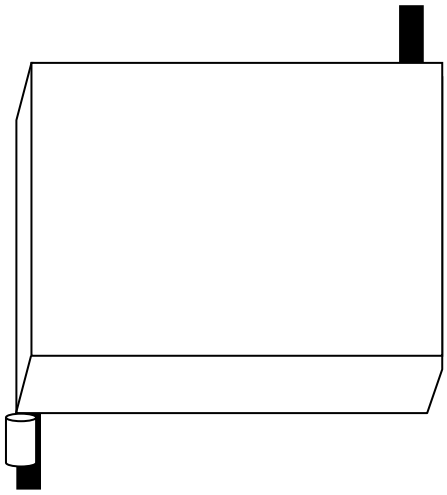
Put black lines at the corners, as described in the following picture.



Write a program, as in challenge 3.4, with SENSE stop at the black line for 3 seconds.

Challenge 3.8 – Along a building block with stop for pedestrian

Put a rod (simulates a pedestrian crossing) at one of the corners, as described in the following picture.



Write a program, as in challenge 3.4, with SENSE stop at the black line for 3 seconds.

It does not move on if an obstacle is in front of it.

The rod simulates a pedestrian.

Challenge 3.9 – Building block guard

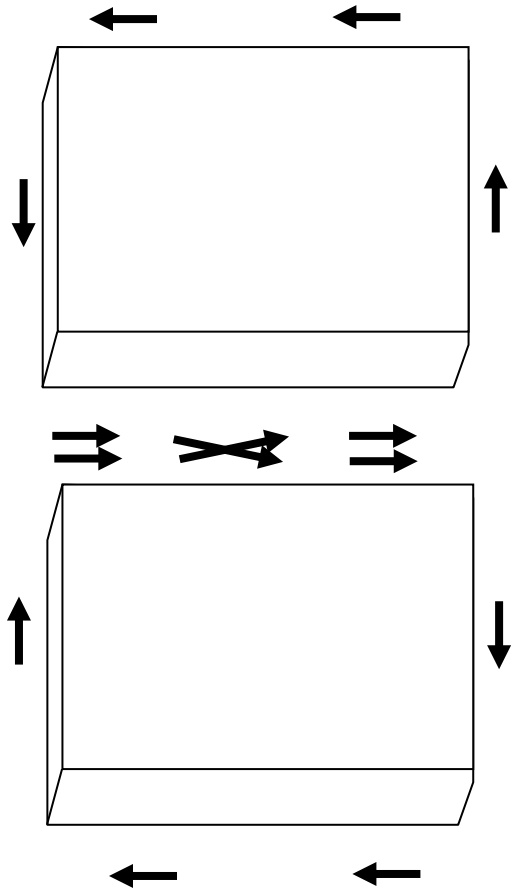
Write a program, as in challenge 3.4, with SENSE stop for 10 seconds after each round.

The program has to count the corner turns.

Hint:

- The SENSE has two range sensors on each side.
- The movement along the wall is according to the front side range sensor.
- When the robot is in a corner, the back side sensor moves away from the wall.
- The program should check the value of the back side sensor.
- When the value is low (the sensor is far from the wall), the robot should call a turn procedure with increasing the corner number.
- After counting four corners, the robot should stop for 10 seconds and then starts again.

Challenge 3.10 – Two buildings guard

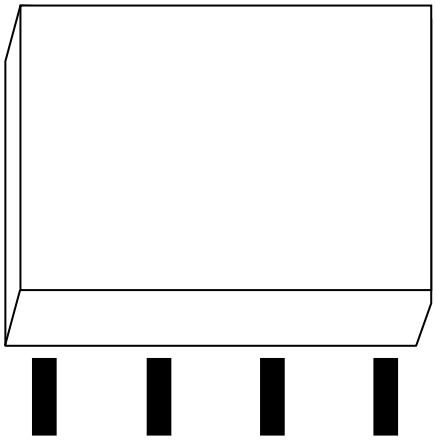


Write a program for the SENSE to go around the two buildings as in the above picture. The robot starts at the road between the two blocks.

The program has to count the corner turns and to change from moving counterclockwise around one building to clockwise around the other building.

Challenge 3.11 – Taxi driver

Put black lines along the building, as described in the following picture.

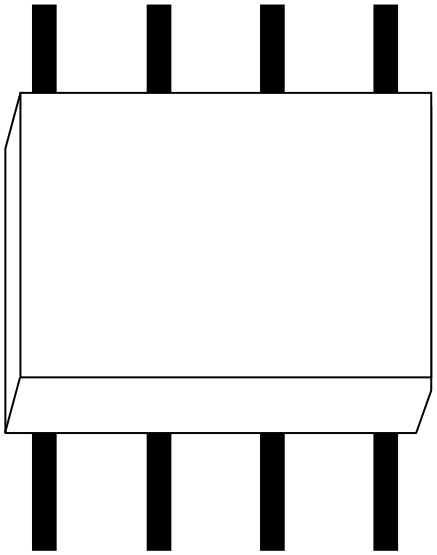


Write a program that moves the SENSE along the building and stops it on the third black line.

Start with the SENSE on the other side of the building.

Challenge 3.12 – Taxi driver with passenger

Put black lines along the building, as described in the following picture.



Write a program that moves the SENSE along the building and stops on the third black line for 5 seconds.

After that, the robot continues to the other side and stops on the second black line.

Challenge 3.13 – Home vacuum cleaner robot

Build a model of a room as follows:

Create a program that moves the robot along the walls in different distances from the walls.

At the first round, the robot will move closer to the walls and at the second round, the robot will move 8cm from the walls.