

# iRobotFramework

---



## 1. Contenido

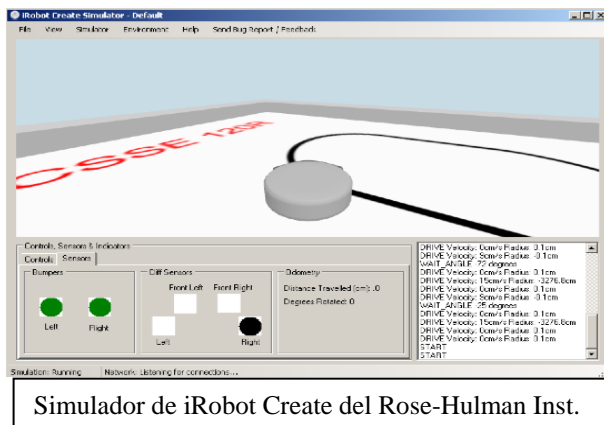
2.	Presentación .....	5
3.	Estructura del software .....	6
3.1	Portabilidad .....	6
4.	Descripción de los módulos .....	6
4.1	Instruction .....	6
4.1.1	Descripción.....	6
4.1.2	Implementación .....	6
4.1.3	Extensibilidad.....	7
4.2	ProcessorInstructionSet .....	7
4.2.1	Descripción.....	7
4.2.2	Implementación de la versión para Arduino .....	7
4.2.3	Extensibilidad.....	8
4.3	IrobotInstructionSet.....	8
4.3.1	Descripción.....	8
4.3.2	Implementación .....	8
4.3.3	Extensibilidad.....	10
4.4	SerialConnection .....	11
4.4.1	Descripción.....	11
4.4.2	Extensibilidad.....	12
4.5	IrobotConnection.....	12
4.5.1	Descripción.....	12
4.5.2	Implementación .....	12
4.5.3	Extensibilidad.....	12
5.	Instalación y uso del <i>iRobotFramework</i> .....	14
5.1	Puesta en marcha.....	14
5.2	Proceso de instalación .....	14
5.2.1	Verificaciones.....	15
5.3	Plantilla iRobotFramework Test .....	16
6.	Ejemplo de programa .....	18
6.1	Librería iRobot Connection.....	19
6.2	Cómo se consultan los sensores .....	19
6.3	Codificaciones Sensores iCreate .....	19
6.3.1	Ejemplo: .....	20
7.	Versión actual de <i>iRobotFramework</i> .....	21
7.1	Instruction.cpp.....	21
7.2	Instruction.h .....	21
7.3	IRobotInstructionSet.cpp.....	21
7.4	IRobotInstructionSet.h .....	28
7.5	IRobotConnection.cpp.....	31
7.6	IRobotConnection.h .....	35

7.7	Serial.cpp.....	37
7.8	Serial.h .....	38

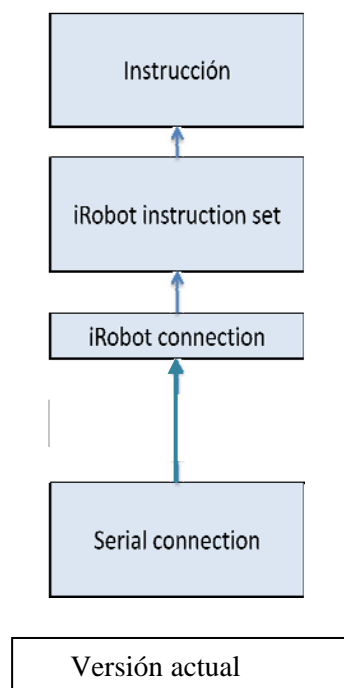
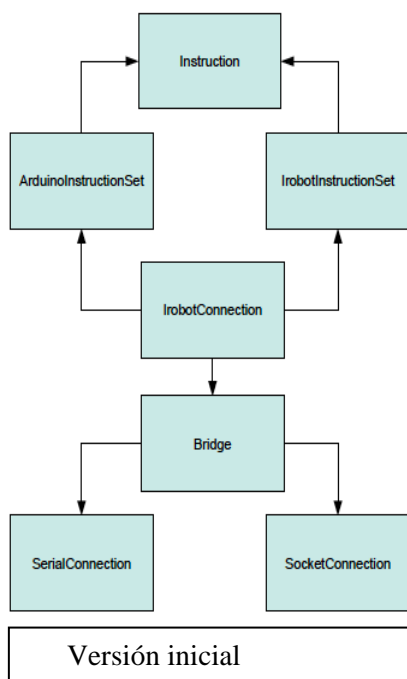
## 2. Presentación

**iRobotFramework** es una máquina virtual escrita en C/C++ que aporta un lenguaje de programación de alto nivel (LPAN) para programar el iRobot Create. Para ello, traduce las instrucciones de este lenguaje a los códigos que ejecuta el iRobot y además se encarga de conectar por línea serie el iRobot y el procesador dónde se ejecuta el programa.

**iRobotFramework** es modular y extensible, de manera que permite cambiar de plataformas (el robot móvil o el procesador) haciendo cambios sólo en determinados módulos del software. También permite modificar y extender el juego de instrucciones del robot.



Fue diseñado por Gorka Montero, en su PFC; con Arduino como procesador e iRobot Create como plataforma móvil. Inicialmente permitía usar un simulador del iRobot (desarrollado por el Rose-Hulman Inst.) y diversas conexiones serie. La versión actual, modificada por Borja Gamecho, utiliza Raspberry Pi como procesador y mantiene el iRobot Create como plataforma móvil. Además se han eliminado el simulador y la conexión vía serie mediante Bluetooth



La versión actual de las librerías iRobotConnection genera código que se ejecuta sobre Raspberry Pi. Este código traduce cada instrucción de alto nivel a comandos del iRobot. Además, abre una línea serie (entre Raspberry Pi 2 e iRobot Create) y envía por ella los códigos asociados a cada instrucción. Para ello, requiere una conexión física por línea serie entre Raspberry Pi 2 e iRobot Create

### 3. Estructura del software

El *iRobotFramework* tiene una estructura modular e independiente del robot, de forma que se pueda reutilizar la mayor cantidad posible de código si se modifica el hardware. Para ello se ha creado una aplicación de 2 capas:

La capa inferior contiene:

- **Juego de instrucciones:** Este módulo, única y exclusivamente, genera las instrucciones que el hardware (en la versión actual la Raspberry Pi 2) interpreta.
- **Conexiones:** Este módulo es el encargado de gestionar la comunicación, ofrece soporte para configurar la conexión entre el programa de control del robot y el hardware, pero no la configura.

La capa superior contiene:

- **Interfaz:** Este es el módulo encargado de unir las instrucciones con las conexiones ya configuradas. Será en este módulo en el que se añadirán las nuevas instrucciones de alto nivel que el desarrollador considere necesarias

#### 3.1 Portabilidad

Si se cambiara de robot, habría que generar un nuevo juego de instrucciones para el nuevo robot, configurar la conexión y dar soporte al nuevo juego de instrucciones en la interfaz.

Si se cambiara de microprocesador, habría que compilar el programa de control sobre la nueva plataforma. Si se desea, se puede añadir un módulo con las instrucciones de la plataforma que se vayan a usar al programar el robot (por ejemplo, instrucciones de entrada/salida). Por ejemplo, si quisiéramos añadir una nueva instrucción de configuración de la plataforma Raspberry Pi que no pueda realizarse por medio de ninguna de las instrucciones que hay actualmente en ella habría que añadir el nuevo código de instrucción al juego de instrucciones del Robot y añadir el soporte para esa nueva instrucción al programa que corre en la Raspberry Pi.

### 4. Descripción de los módulos

#### 4.1 Instruction

##### 4.1.1 Descripción

Este módulo almacena las instrucciones, junto con el tamaño en bytes de las mismas y la cantidad de bytes que se deberían recibir como respuesta. El objetivo de este módulo es marcar un estándar para los generadores de instrucciones que se van a utilizar y para los futuros generadores que se puedan crear.

##### 4.1.2 Implementación

Debido a que las instrucciones se almacenan en un char \*, y debido a que la asignación entre punteros hace que los dos punteros apunten a la misma dirección de memoria<sup>1</sup>, se ha sobrescrito el operador de asignación con uno que realice copias de valores del char \* gestionando el uso de la memoria.

---

<sup>1</sup> Si dos punteros apuntan a una misma dirección de memoria, a la hora de modificar el valor mediante uno de ellos automáticamente queda modificado el del otro, cosa que no deseamos que suceda.

```

Instruction & Instruction::operator=(const Instruction &aux )
{
if (instruction != nullptr) free(instruction);
instruction = new char[aux.length-1];
for (int i = 0; i < aux.length ; i++)
{
    instruction[i] = aux.instruction[i];
}
length = aux.length;
response = aux.response;
}

```

#### 4.1.3 Extensibilidad

A la hora de añadir más opciones a este módulo tendríamos que modificar el operador de asignación para que incluya los nuevos campos del módulo. Por ejemplo, si queremos que con las instrucciones se guarde la hora en la que son generadas en una variable llamada hora deberemos de añadir al operador de asignación la siguiente línea:

```

hora = aux.hora;

```

## 4.2 ProcessorInstructionSet

### 4.2.1 Descripción

Este módulo es el encargado de generar las instrucciones (por ejemplo de entrada/salida) específicas del procesador que utilicemos (Arduino, Raspberry Pi o cualquier otro). Puede servirnos, por ejemplo, tanto para configurar los pines de Arduino, como leer o enviar información por alguno de sus pines. En la versión actual este módulo no está activado.

### 4.2.2 Implementación de la versión para Arduino

Para facilitar en el futuro el añadir nuevas instrucciones a este generador, se ha creado el siguiente namespace:

```

namespace ArduinoInstruction{
static const char SET_PIN_MODE = 100;
static const char DIGITAL_READ = 101;
static const char DIGITAL_WRITE = 102;
static const char ANALOG_READ = 103;
static const char ANALOG_WRITE = 104;
};

```

De forma que a simple vista se sabe que los bytes correspondientes al juego de instrucciones de Arduino están en el rango [100,104] y también que byte corresponde a cada instrucción.

Por otro lado, tanto a la hora de configurar pines como a la hora de enviar y recibir información por los pines digitales, Arduino utiliza unas variables que almacenan esos valores, es por ello que se ha creado el siguiente namespace que recoge los valores especiales de Arduino:

```
namespace ArduinoData
{
static const char INPUT_MODE = 1;
static const char OUTPUT_MODE = 0;
static const char HIGH = 1;
static const char LOW = 0;
};
```

En este namespace se pueden encontrar de manera sencilla los bytes que se les ha asignado en este proyecto y que cuando Arduino los reciba en conjunción con alguna de sus instrucciones realizara las labores oportunas.

#### 4.2.3 Extensibilidad

Estos namespaces se han creado para facilitar la extensibilidad del módulo de forma que modificando alguno de los dos se podrían cambiar los bytes que corresponden a cada instrucción sin necesidad de modificar el cuerpo del mismo. Además permiten separar en varios grupos los distintos tipos de datos para así poder aprovechar mejor las funciones de auto-completado del entorno de desarrollo.

**Ejemplo:** Supongamos que queremos añadir una instrucción que configure el rango del voltaje de un pin:

```
AnalogVoltageRange(int pin, byte minVoltage, byte maxVoltage)
```

Deberemos de añadirle una variable más al namespace ArduinoInstruction:

```
static const char ANALOG_VOLTAGE_RANGE = 105;
```

### 4.3 IrobotInstructionSet

#### 4.3.1 Descripción

Este es el módulo encargado de generar todas las instrucciones que puede ejecutar el robot iRobot Create. Permite la adaptación a alto nivel de las instrucciones originales que vienen con el robot iRobot Create y que se muestran en la descripción realizada anteriormente del robot.

#### 4.3.2 Implementación

Para mayor facilidad a la hora de modificar el juego de instrucciones en la cabecera de este módulo se han creado los siguientes namespaces<sup>2</sup> encargados de almacenar los bytes correspondientes a cada instrucción en una variable que evite el tener que acudir a la tabla de instrucciones a la hora de usarla. Por otro lado permite modificar el juego de instrucciones de manera mucho más eficiente.

```
namespace iRobotInstructions{
static const char START = (char) 128;
static const char BAUD = (char) 129;
static const char CONTROL = (char) 130; //@deprecated
static const char SAFE = (char) 131;
static const char FULL = (char) 132;
static const char SPOT = (char) 134;
static const char COVER = (char) 135;
```

<sup>2</sup> Un namespace sirve para diferenciar variables de un entorno y de otro, de forma que si tenemos dos variables con nombre instrucción, una perteneciente a un namespace (standard) y otra perteneciente a un namespace (personalizada) podrían, gracias a los namespaces, usarse las 2 sin problemas. Tan solo hay que usarlas de la siguiente manera: *standard::instruccion* o *personalizada::instruccion* dependiendo de cuál de las dos queramos utilizar



```

static const char DEMO = (char) 136;
static const char DRIVE = (char) 137;
static const char LOW_SIDE_DRIVERS = (char) 138;
static const char LEDS = (char) 139;
static const char SONG = (char) 140;
static const char PLAY = (char) 141;
static const char SENSORS = (char) 142;
static const char COVER_AND_DOCK = (char) 143;
static const char PWM_LOW_SIDE_DRIVERS = (char) 144;
static const char DRIVE_DIRECT = (char) 145;
static const char DIGITAL_OUTPUTS = (char) 147;
static const char STREAM = (char) 148;
static const char QUERY_LIST = (char) 149;
static const char PAUSE_RESUME_STREAM = (char) 150;
static const char SEND_IR = (char) 151;
static const char SCRIPT = (char) 152;
static const char PLAY_SCRIPT = (char) 153;
static const char SHOW_SCRIPT = (char) 154;
static const char WAIT_TIME = (char) 155;
static const char WAIT_DISTANCE = (char) 156;
static const char WAIT_ANGLE = (char) 157;
static const char WAIT_EVENT = (char) 158;
}

```

Para dar soporte de alto nivel a las instrucciones que acceden a los distintos sensores se ha creado el siguiente namespace:

```

namespace iRobotSensors{
static const char BUMPERS_AND_WHEELDROPS = (char) 7;
static const char WALL = (char) 8;
static const char CLIFFLEFT = (char) 9;
static const char CLIFFFRONTLEFT = (char) 10;
static const char CLIFFFRONTRIGHT = (char) 11;
static const char CLIFFRIGHT = (char) 12;
static const char VIRTUALWALL = (char) 13;
static const char MOTOR_OVERCURRENTS = (char) 14;
// static const char DIRTLEFT = (char) 15;
// static const char DIRTRIGHT = (char) 16;
static const char IRBYTE = (char) 17;
static const char BUTTONS = (char) 18;
static const char DISTANCE = (char) 19;
static const char ANGLE = (char) 20;
static const char CHARGINGSTATE = (char) 21;
static const char VOLTAGE = (char) 22;
static const char CURRENT = (char) 23;
static const char TEMPERATURE = (char) 24;
static const char CHARGE = (char) 25;
static const char CAPACITY = (char) 26;
static const char WALLSIGNAL = (char) 27;
static const char CLIFFLEFTSIGNAL = (char) 28;
static const char CLIFFFRONTLEFTSIGNAL = (char) 29;
static const char CLIFFFRONTRIGHTSIGNAL = (char) 30;
static const char CLIFFRIGHTSIGNAL = (char) 31;
static const char USERDIGITAL = (char) 32;
static const char USERANALOG = (char) 33;
static const char CHARGINGSOURCES = (char) 34;
static const char OIMODE = (char) 35;
static const char SONGNUMBER = (char) 36;
static const char SONGPLAYING = (char) 37;
static const char STREAMPACKETS = (char) 38;
}

```

```
static const char VELOCITY = (char) 39;
static const char RADIUS = (char) 40;
static const char RIGHTVELOCITY = (char) 41;
static const char LEFTVELOCITY = (char) 42;
}
```

Podemos ver que los sensores correspondientes al byte 15 y al byte 16 están comentados, esto se debe a que esos bytes no corresponden a ningún sensor. En el juego de instrucciones de iRobot contiene la instrucción *wait* que hace que el robot espere a un evento concreto. Para dar soporte de alto nivel a esta instrucción se ha creado este otro namespace que relaciona los nombres de eventos con los bytes a los que corresponden:

```
namespace iRobotEvents{
static const int WHEEL_DROP = 1;
static const int FRONT_WHEEL_DROP = 2;
static const int LEFT_WHEEL_DROP = 3;
static const int RIGHT_WHEEL_DROP = 4;
static const int BUMP = 5;
static const int LEFT_BUMP = 6;
static const int RIGHT_BUMP = 7;
static const int VIRTUAL_WALL = 8;
static const int WALL = 9;
static const int CLIFF = 10;
static const int LEFT_CLIFF = 11;
static const int FRONT_LEFT_CLIFF = 12;
static const int FRONT_RIGHT_CLIFF = 13;
static const int RIGHT_CLIFF = 14;
static const int HOME_BASE = 15;
static const int ADVANCE_BUTTON = 16;
static const int PLAY_BUTTON = 17;
static const int DIGITAL_INPUT_0 = 18;
static const int DIGITAL_INPUT_1 = 19;
static const int DIGITAL_INPUT_2 = 20;
static const int DIGITAL_INPUT_3 = 21;
static const int OIMODE_PASSIVE = 22;
}
```

Para las consultas de varios sensores a la vez se ha creado el siguiente namespace que identifica los distintos grupos de sensores que ofrece el robot:

```
namespace iRobotSensorPackage {
static const char SENSORS_7_26 = (char) 0;
static const char SENSORS_DIGITAL BUMPER_CLIFF_WALL_VIRTUALWALL =
(char) 1;
static const char SENSORS_IR_BUTTONS_DISTANCE_ANGLE = (char) 2;
static const char SENSORS_BATTERY = (char) 3;
static const char SENSORS_ANALOG_CLIFF_WALL_VIRTUALWALL = (char) 4;
static const char SENSORS_35_42 = (char) 5;
static const char SENSORS_ALL = (char) 6;
}
```

### 4.3.3 Extensibilidad

A la hora de ampliar este módulo sólo habría que añadir en el namespace apropiado la información extra y en el caso de que fuera una instrucción extra, crear la función que genere la secuencia de bytes que la representan.

**Ejemplo:** Supongamos que en una actualización de firmware, iRobot añade una nueva instrucción al robot con el código 159 que usa 2 parámetros de un byte cada uno y que no genera ninguna respuesta por parte del robot. Para añadir esta nueva instrucción en el namespace iRobotInstruction añadiríamos lo siguiente:

```
static const int NUEVA_INSTRUCCION = 159;
```

Además habría que implementar la siguiente instrucción:

```
Instruction nueva(char byte1, char byte2 )
{
    int miSize = 3;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::NUEVA;
    aux.instruction[1] = byte1;
    aux.instruction[2] = byte2;
    aux.length = miSize;
    return aux;
}
```

## 4.4 SerialConnection

### 4.4.1 Descripción

Este es el módulo encargado de gestionar toda la comunicación por puerto serie.

#### *Implementación*

El módulo consta de las siguientes funciones:

```
void Open_Port(char * COMx);
```

Permite abrir el puerto serie indicado almacenado en COMx.

```
DCB Get_Configure_Port();
```

Permite obtener la variable de configuración del puerto serie.

```
DCB Configure_Port(unsigned int BaudRate, char CharParity[]);
```

Permite generar un archivo de configuración para la conexión por puerto serie.

```
int Set_Configure_Port(DCB PortDCB);
```

Asigna al puerto serie la configuración que se le pasa en la variable PortDCB

```
long Write_Port(char Data[], int SizeData);
```

Escribe en el puerto serie la cantidad de bytes indicada por SizeData que están almacenados en la variable Data.

```
long Read_Port(char *Data, int SizeData);
```

Lee del puerto serie la cantidad de bytes indicados en SizeData y los almacena en la variable Data.

```
long Getc_Port(char *Data);
```

Lee del puerto serie un carácter y lo almacena en Data[0]

```
int Kbhit_Port();
```

Indica cuantos bytes hay en el puerto de entrada.

```
int Close_Port();
```

Cierra el puerto serie.

```
int Set_Handshaking(int FlowControl);
```

Establece el flow control de la conexión.

```
int Set_Time(unsigned int Time);
```

Establece los tiempos de espera antes de decidir un timeout.

```
int Clean_Buffer();
```

Vacía el buffer de entrada

```
int Setup_Buffer(unsigned long InQueue,unsigned long OutQueue);
```

Establece la longitud del buffer a utilizar.

#### 4.4.2 Extensibilidad

Este módulo no requiere de ningún tipo de ampliación ya que en él se encuentran todas las instrucciones necesarias para tratar con cualquier tipo de puerto serie.

### 4.5 IrobotConnection

#### 4.5.1 Descripción

Este es el módulo con el que trabajan los usuarios. Los procedimientos de este módulo se encargan de pasarle al generador la instrucción que tiene que generar, de enviar la instrucción por el canal de comunicación previamente definido y de recibir la respuesta en caso de realizar alguna consulta.

#### 4.5.2 Implementación

Los constructores de la clase son los encargados de inicializar el tipo de conexión que se va a utilizar.

```
IRobotConnection::IRobotConnection(void)
{
    buffer = (char *) malloc(512*sizeof (char));
}
IRobotConnection::IRobotConnection(char * connectionType )
{
    buffer = (char *) malloc(512*sizeof (char));
}
```

Por otro lado, se ha creado una función auxiliar que convierte los números que hay almacenados en las posiciones high y low del buffer en formato integer.

```
int IRobotConnection::createInt(char* aux, int hight /*= 0*/, int low
/*= 1*/ )
{
    return (aux [low]+ (aux[hight]<<8));
}
```

#### 4.5.3 Extensibilidad

A la hora de crear nuevas instrucciones habrá que implementar las funciones correspondientes a las mismas.

**Ejemplo:** Supongamos que queremos añadir una nueva instrucción que nos permita encender o apagar un LED que está conectado a la placa Arduino en el pin 13. Tendríamos que implementar la siguiente función:

```
void cambiarEstadoLed13(bool encender)
{
  Instruction aux;
  if (encender) aux =
  ArduinoInstructionGenerator.digitalWrite(13,ArduinoData::HIGH);
  else aux =
  ArduinoInstructionGenerator.digitalWrite(13,ArduinoData::LOW);
  connection.send(aux.instruction,aux.length);
}
```

## 5. Instalación y uso del *iRobotFramework*

### 5.1 Puesta en marcha

Instalar el compilador de Raspberry Pi, versión 4.8

```
> sudo apt-get install gcc-4.8
> sudo apt-get install g++-4.8
```

Borrar los enlaces simbólicos antiguos del sistema

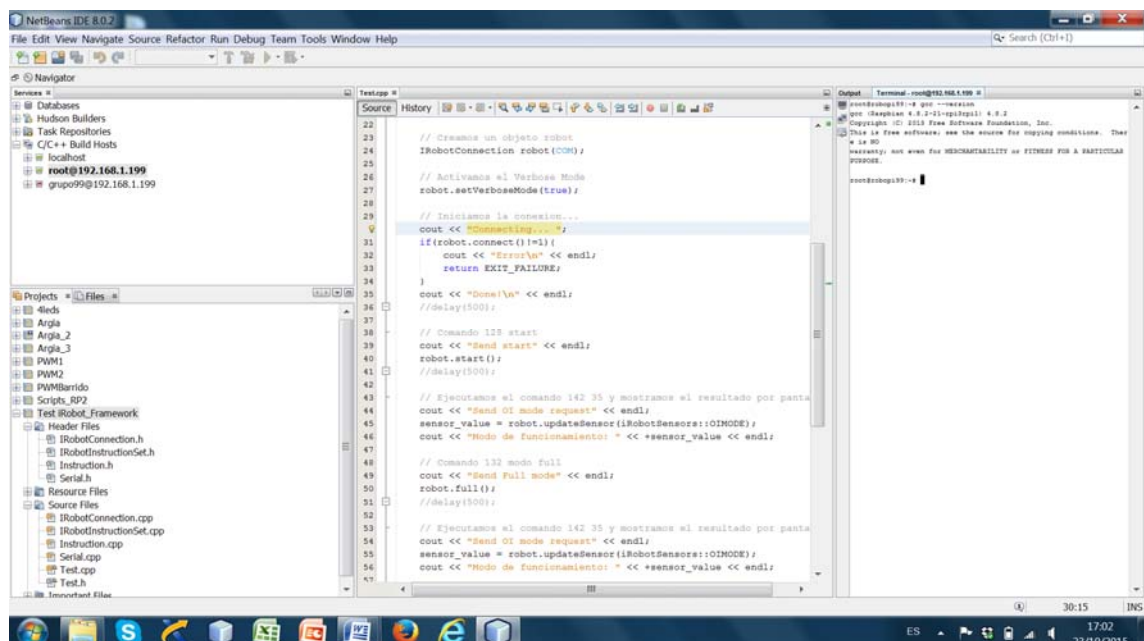
```
> sudo rm /usr/bin/gcc
> sudo rm /usr/bin/g++
```

Actualizar los enlaces simbólicos del sistema: /usr/bin/gcc y /usr/bin/g++

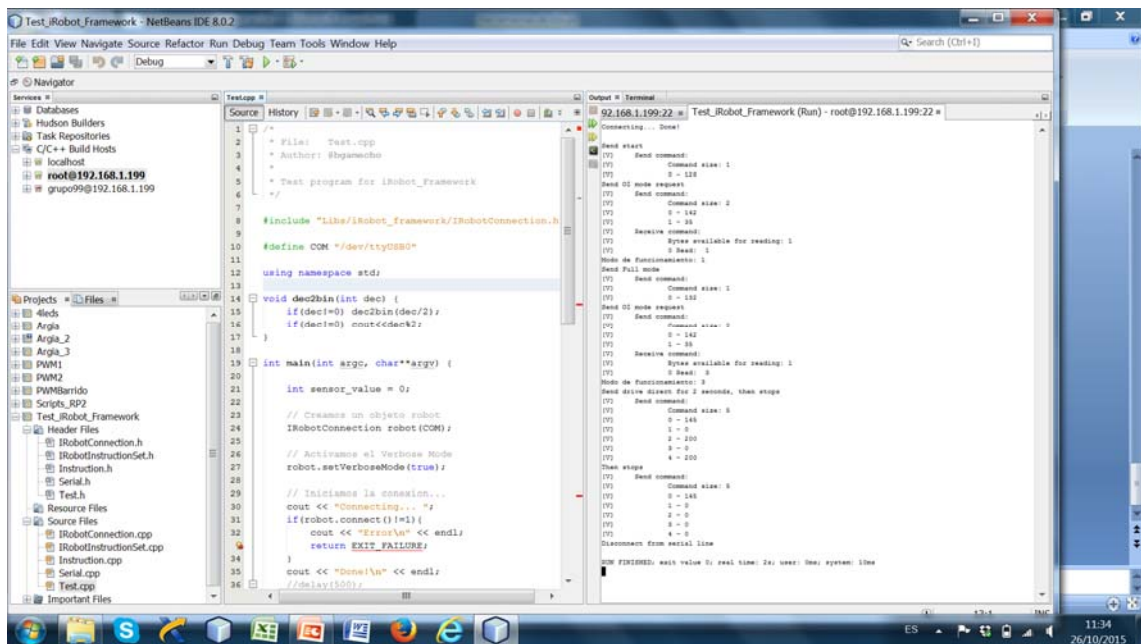
```
> sudo ln -s /usr/bin/gcc-4.8 /usr/bin/gcc
> sudo ln -s /usr/bin/g++-4.8 /usr/bin/g++
```

### 5.2 Proceso de instalación

- Abrir NetBeans
  - Importar el proyecto frameworktest.zip (**sin descomprimir**) que está en frameworktest.zip está en eGela: Práctica 3
  - seleccionar File—> Import Project —> From Zip
- frameworktest.zip contiene los ficheros:
  - IRobotConnection.h
  - IrobotInstructionSet.h
  - Instruction.h
  - Serial.h
  - Test.h
  - IRobotConnection.cpp
  - IrobotInstructionSet.cpp
  - Instruction.cpp
  - Serial.cpp
  - Test.cpp
- Compilar y ejecutar Test.cpp



La ejecución producirá:



```
1  /*
2  * File: Test.cpp
3  * Author: @bgamecho
4  *
5  * Test program for iRobotFramework
6  */
7
8  #include "iRobotFramework/iRobotConnection.h"
9
10 #define COM "/dev/ttyUSB0"
11
12 using namespace std;
13
14 void dec2bin(int dec) {
15     if(dec!=0) dec2bin(dec/2);
16     if(dec!=0) cout<<dec<<endl;
17 }
18
19 int main(int argc, char**argv) {
20     int sensor_value = 0;
21
22     // Creamos un objeto robot
23     iRobotConnection robot(COM);
24
25     // Activamos el Verbose Mode
26     robot.setVerboseMode(true);
27
28     // Iniciamos la conexión...
29     cout << "Connecting..." << endl;
30     if(robot.connect() != 1){
31         cout << "Error!\n" << endl;
32         return EXIT_FAILURE;
33     }
34     cout << "Done!\n" << endl;
35     //delay(500);
36 }
```

```
92.168.1.199:22 ~ Test_Robot_Framework (Run) - root@192.168.1.199:22 ~
Connecting... Done!
Send start
[VT] Send command: 1
[VT] 0 - 128
[VT] Send command: 1
[VT] Send mode request
[VT] Command size: 2
[VT] 0 - 142
[VT] 1 - 38
[VT] Receive command:
[VT] Bytes available for reading: 1
[VT] 0 Read: 1
[VT] Mode de funcionamiento: 1
[VT] Send Pull mode
[VT] Send command:
[VT] Command size: 1
[VT] 0 - 142
[VT] Send command:
[VT] Command size: 0
[VT] 0 - 142
[VT] 1 - 38
[VT] Receive command:
[VT] Bytes available for reading: 1
[VT] 0 Read: 0
[VT] Mode de funcionamiento: 0
[VT] Send drive disarm for 2 seconds, then stop
[VT] Send command:
[VT] Command size: 5
[VT] 0 - 145
[VT] 2 - 0
[VT] 2 - 200
[VT] 3 - 0
[VT] 4 - 200
[VT] Then stop
[VT] Send command:
[VT] Command size: 5
[VT] 0 - 145
[VT] 2 - 0
[VT] 2 - 0
[VT] 3 - 0
[VT] 4 - 0
[VT] Disconnect from serial line
[VT] NOT FINISHED: each value 0; real time: 2s; user: 0ms; system: 1ms
```

A partir de este momento ya se pueden escribir programas en C/C++ usando las instrucciones de iRobot proporcionadas por *iRobotFramework*.

### 5.2.1 Verificaciones

En caso de que encontrar problemas para ejecutar el test, conviene hacer las siguientes verificaciones:

- Si se trabaja con la cuenta root hay que verificar que está compilando para la dirección IP (root@192.168.1.XX).
- Si se trabaja con la cuenta GrupoXX, hay que verificar que está compilando para la dirección IP (grupoXX@192.168.1.XX)
- Para dar al usuario GrupoXY permiso de ejecución en el adaptador línea-serie conectado al iRobot Create, ejecutar en el terminal remoto de Raspberry Pi2:

```
> sudo usermod -aG dialout grupoXY
```

### 5.3 Plantilla *iRobotFramework Test*

El *iRobotFramework Test* sirve para probar el adecuado funcionamiento del *iRobotFramework* en una configuración específica [PC(NetBeans) – Raspberry Pi – *iRobot Create*].

También se puede usar el *iRobotFramework Test* como plantilla inicial para programar.

```
/* File:    Test.cpp
 * Author:  @bgamecho
 * Test program for iRobot_Framework */

#include "Libs/iRobot_framework/IRobotConnection.h"

#define COM "/dev/ttyUSB0"

using namespace std;

void dec2bin(int dec) {
    if(dec!=0) dec2bin(dec/2);
    if(dec!=0) cout<<dec%2;
}

int main(int argc, char**argv) {

    int sensor_value = 0;

    // Creamos un objeto robot
    IRobotConnection robot(COM);

    // Activamos el Verbose Mode
    robot.setVerboseMode(true);

    // Iniciamos la conexion...
    cout << "Connecting... ";
    if(robot.connect()!=1){
        cout << "Error\n" << endl;
        return EXIT_FAILURE;
    }
    cout << "Done!\n" << endl;
    //delay(500);

    // Comando 128 start
    cout << "Send start" << endl;
    robot.start();
    //delay(500);

    // Ejecutamos el comando 142 35 y mostramos el resultado por
    pantalla
    cout << "Send OI mode request" << endl;
    sensor_value = robot.updateSensor(iRobotSensors::OIMODE);
    cout << "Modo de funcionamiento: " << +sensor_value << endl;

    // Comando 132 modo full
    cout << "Send Full mode" << endl;
    robot.full();
    //delay(500);
```



```
// Ejecutamos el comando 142 35 y mostramos el resultado por
// pantalla
cout << "Send OI mode request" << endl;
sensor_value = robot.updateSensor(iRobotSensors::OIMODE);
cout << "Modo de funcionamiento: " << +sensor_value << endl;

// Avanzamos durante 2 segundos a 200mm/s y paramos los motores
cout << "Send drive direct for 2 seconds, then stops" << endl;
robot.driveDirect(200,200);
delay(2000);
cout << "Then stops" << endl;
robot.driveDirect(0,0);

// Desconectamos
cout << "Disconnect from serial line" << endl;
robot.disconnect();

return EXIT_SUCCESS;
}
```

## 6. Ejemplo de programa

Para programar el iRobot creamos un programa main en el que se incluye el código que controla el robot.

```
1  #include <iostream>
2  #include "../libs/IRobotConnection.h"
3  #include <dos.h>
4
5  using namespace std;
6
7  int main(int argc, char * argv[])
8  {
9
10     // Creamos un objeto robot que se conectará por el puerto x
11     IRobotConnection robot("COMx");
12
13     // Iniciamos la conexión
14     printf("Connecting... ");
15     robot.connect();
16     printf("Done!!\n");
17
18     // comando 128 start
19     robot.start();
20     Sleep(500);
21
22     // comando 132 modo full
23     robot.full();
24     Sleep(500);
25
26     ... resto de comandos ...
27
28     robot.disconnect();
29     return 0;
30 }
```

```
// Ejecutamos el comando 142 35 y mostramos el resultado por pantalla
cout << "Modo de funcionamiento: " << robot.updateSensor(iRobotSensors::OIMODE) << endl;

// Comando 132 modo full
robot.full();
Sleep(500); // Esperamos medio segundo a que cambie de modo

// Ejecutamos el comando 142 35 y mostramos el resultado por pantalla
cout << "Modo de funcionamiento: " << robot.updateSensor(iRobotSensors::OIMODE) << endl;

// Avanzamos durante 2segundos a 200mm/s y paramos los motores
robot.driveDirect(200,200);
Sleep(2000);
robot.driveDirect(0,0);
```

## 6.1 Librería iRobot Connection

Modos:

```
void connect();
void start(); // código 128
void control(); // código 130 Ya no disponible en iRobot Create
void safe(); // código 131 No recomendable
void full(); // código 132
```

Navegación:

```
void drive(int speed, int radius);
void driveDirect(int rightVelocity, int leftVelocity);
```

Sensores:

```
int updateSensor(char sensorId);
void stream(char* sensorIdList, int size);
int queryList(char * sensorIdList, int size);
void PauseResumeStream(bool bolol);
```

Scripting

```
void script(int *commandList, int size);
void playScript();
void showScript();
```

~~Espera eventos~~—[Desaconsejable usarlos, porque paran el envío de datos por la línea serie hasta que el evento se cumple]

```
void waitTime(int seconds);
void waitDistance(int mm);
void waitAngle(int degrees);
void waitEvent(int eventId);
```

Otros:

```
void leds(int ledBit, int ledColor, int ledIntensity);
void song (int songNumber, int songSize, char *song);
void playSong(int songNumber);
```

## 6.2 Cómo se consultan los sensores

```
Int value = updateSensor(char code);
Ejemplo:
    Char value = updateSensor(iRobotSensors::BUMPERS_AND_WHEELDROPS);
    printf("BUMPERS AND WHEELDROPS %s \n", value);
Codes:
iRobotSensors::CLIFFLEFT
iRobotSensors::CLIFFFRONTLEFT
iRobotSensors::CLIFFFRONTRIGHT
iRobotSensors::CLIFFRIGHT
iRobotSensors::BUMPERS_AND_WHEELDROPS
iRobotSensors::WALL
iRobotSensors::DISTANCE
iRobotSensors::ANGLE
```

## 6.3 Codificaciones Sensores iCreate

Desde updateSensor(char code); siempre obtenemos un Int (con signo), en cada caso habrá que hacer un casting al tipo de datos adecuado para poder tratar la información de los sensores correctamente.

### 6.3.1 Ejemplo:

```
Char value = updateSensor(iRobotSensors::BUMPERS_AND_WHEELDROPS);  
printf("Front Caster WheelDrop: %d\n",value & 0x10);
```

Sensor	Codificación
Bumps & WheelDrops	Máscara de bits
Wall, Cliffs x 4, Virtual Wall	1 bit value (El menos significativo)
Infrared	1 Byte [0,255]
Distance, Angle, Requested Radius	Signed 16 bit value [-32768 - 32768]
Wall Signal, Cliffs x 4,	Unsigned 16 bit [0 – 4095]
Requested Velocity x3	Signed 16 bit value [-500, 500]

## 7. Versión actual de *iRobotFramework*

### 7.1 Instruction.cpp

```
/* File:   Instruction.cpp
 * Author: Gorka Montero 2012 */

#include "Instruction.h"

Instruction::Instruction()
{
    instruction = nullptr; response = 0;
}

Instruction & Instruction::operator=(const Instruction &aux )
{
    if (instruction != nullptr) free(instruction);
    instruction = new char[aux.length-1];
    for (int i = 0; i < aux.length ; i++)
    {
        instruction[i] = aux.instruction[i];
    }
    length = aux.length;
    response = aux.response;
    return *this;
}
```

### 7.2 Instruction.h

```
/* File:   Instruction.h
 * Author: Gorka Montero 2012 */

#ifndef INSTRUCTION_H
#define INSTRUCTION_H

#pragma once
#include <malloc.h>

class Instruction{
public:
    char * instruction; //Instruction bytes
    int length;         //How many bytes compose the instruction
    int response;       //How many bytes will the instruction generate
                        as return by the machine

    Instruction();
    ~Instruction(){}
    Instruction & operator=(const Instruction &aux);
};

#endif
```

### 7.3 IRobotInstructionSet.cpp

```
/* File:   IRobotInstructionSet.cpp
 * Author: Gorka Montero 2012 */

#include "IRobotInstructionSet.h"
```

```

iRobotInstructionSet::iRobotInstructionSet(void)
{
}

iRobotInstructionSet::~~iRobotInstructionSet(void)
{
}

char iRobotInstructionSet::intHightByte(short num )
{
    return((char) (num>>8));
}

char iRobotInstructionSet::intLowByte(short num )
{
    return((char) num);
}

Instruction iRobotInstructionSet::start()
{
    int miSize = 1;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::START;
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::baud(char code )
{
    int miSize = 2;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::BAUD;
    aux.instruction[1] = code;
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::control()
{
    int miSize = 1;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::CONTROL;
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::safe()
{
    int miSize = 1;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::SAFE;
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::full()

```

```

{
    int miSize = 1;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::FULL;
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::spot()
{
    int miSize = 1;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::SPOT;
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::cover()
{
    int miSize = 1;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::COVER;
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::demo(char code )
{
    int miSize = 2;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::DEMO;
    aux.instruction[1] = code;
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::drive(int speed, int radius )
{
    int miSize = 5;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::DRIVE;
    aux.instruction[1] = intHightByte(speed);
    aux.instruction[2] = intLowByte(speed);
    aux.instruction[3] = intHightByte(radius);
    aux.instruction[4] = intLowByte(radius);
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::lowSideDrivers(char outputBit )
{
    int miSize = 2;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::LOW_SIDE_DRIVERS;
    aux.instruction[1] = outputBit;
}

```

```

        aux.length = miSize;
        return aux;
    }

Instruction iRobotInstructionSet::leds(int ledBit, int ledColor, int
ledIntensity )
{
    int miSize = 4;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::LEDS;
    aux.instruction[1] = ledBit;
    aux.instruction[2] = ledColor;
    aux.instruction[3] = ledIntensity;
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::song(int songNumber, int songSize,
char *song )
{
    int miSize = songSize+3;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::SONG;
    aux.instruction[1] = songNumber;
    aux.instruction[2] = (int)songSize/2;
    for (int i = 0; i < songSize; i++)
    {
        aux.instruction[i+3] = song[i];
    }
    aux.length = miSize;

    return aux;
}

Instruction iRobotInstructionSet::playSong(int songNumber)
{
    int miSize = 2;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::PLAY;
    aux.instruction[1] = songNumber;
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::updateSensor(char sensorId )
{
    int miSize = 2;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::SENSORS;
    aux.instruction[1] = sensorId;
    aux.response = sensorReturn(sensorId);
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::coverAndDock()
{

```



```

    int miSize = 1;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::COVER_AND_DOCK;
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::pwmLowSideDrivers(int driver2,int
driver1, int driver0 )
{
    int miSize = 4;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::PWM_LOW_SIDE_DRIVERS;
    aux.instruction[1] = driver2;
    aux.instruction[2] = driver1;
    aux.instruction[3] = driver0;
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::driveDirect(int rightVelocity, int
leftVelocity )
{
    int miSize = 5;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::DRIVE_DIRECT;
    aux.instruction[1] = intHightByte(rightVelocity);
    aux.instruction[2] = intLowByte(rightVelocity);
    aux.instruction[3] = intHightByte(leftVelocity);
    aux.instruction[4] = intLowByte(leftVelocity);
    aux.length = miSize;
    return aux;
}

int iRobotInstructionSet::sensorReturn(char sensor )
{
    if (sensor < 7) return 0;
    if (sensor < 19) return 1;
    if (sensor < 21) return 2;
    if (sensor == 21) return 1;
    if (sensor < 24) return 2;
    if (sensor == 24) return 1;
    if (sensor < 32) return 2;
    if (sensor == 32) return 1;
    if (sensor == 33) return 2;
    if (sensor < 39) return 1;
    if (sensor < 43) return 2;
    return 0;
}

Instruction iRobotInstructionSet::digitalOutputs(int outputBits )
{
    int miSize = 2;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::DIGITAL_OUTPUTS;
    aux.instruction[1] = (char) outputBits;
    aux.length = miSize;
}

```

```

        return aux;
    }

Instruction iRobotInstructionSet::stream(char* sensorIdList, int size
)
{
    int miSize = 2+ size;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::STREAM;
    aux.instruction[1] = size;
    for(int i = 0; i < size; i++)
    {
        aux.instruction[i+2] = sensorIdList[i];
        aux.response += sensorReturn(sensorIdList[i]);
    }
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::queryList(char * sensorIdList, int
size )
{
    int miSize = 2+ size;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::QUERY_LIST;
    aux.instruction[1] = size;
    for(int i = 0; i < size; i++)
    {
        aux.instruction[i+2] = sensorIdList[i];
        aux.response += sensorReturn(sensorIdList[i]);
    }
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::PauseResumeStream(bool bol )
{
    int miSize = 2;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::PAUSE_RESUME_STREAM;
    aux.instruction[1] = (char) bol;
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::sendIr(int data )
{
    int miSize = 2;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::SEND_IR;
    aux.instruction[1] = (char)data;
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::script(int *commandList, int size )
{

```

```

    int miSize = 2+ size;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::SCRIPT;
    aux.instruction[1] = size;
    for(int i = 0; i < size; i++) aux.instruction[i+2] =
commandList[i];
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::playScript()
{
    int miSize = 1;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::PLAY_SCRIPT;
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::showScript()
{
    int miSize = 1;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::SHOW_SCRIPT;
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::waitTime(int seconds )
{
    int miSize = 2;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::WAIT_TIME;
    aux.instruction[1] = seconds;
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::waitDistance(int mm )
{
    int miSize = 3;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::WAIT_DISTANCE;
    aux.instruction[1] = intHightByte(mm);
    aux.instruction[2] = intLowByte(mm);
    aux.length = miSize;
    return aux;
}

Instruction iRobotInstructionSet::waitAngle(int degrees )
{
    int miSize = 3;
    Instruction aux;
    aux.instruction = (char*)malloc(miSize * sizeof(char));
    aux.instruction[0] = iRobotInstructions::WAIT_ANGLE;
    aux.instruction[1] = intHightByte(degrees);

```

```

        aux.instruction[2] = intLowByte(degrees);
        aux.length = miSize;
        return aux;
    }

    Instruction iRobotInstructionSet::waitEvent(int eventId)
    {
        int miSize = 2;
        Instruction aux;
        aux.instruction = (char*)malloc(miSize * sizeof(char));
        aux.instruction[0] = iRobotInstructions::WAIT_EVENT;
        aux.instruction[1] = (char) eventId;
        aux.length = miSize;
        return aux;
    }

```

## 7.4 IRobotInstructionSet.h

```

/* File:   IRobotInstructionSet.h
 * Author: Gorka Montero 2012 */

#ifndef IROBOTINSTRUCTIONSET_H
#define IROBOTINSTRUCTIONSET_H

#pragma once
#include "Instruction.h"
#include <malloc.h>

//Contains the sensor opcodes that we need to use if we want to query
the sensor values
namespace iRobotSensors{
    static const char BUMPERS_AND_WHEELDROPS = (char) 7;
    static const char WALL = (char) 8;
    static const char CLIFFLEFT = (char) 9;
    static const char CLIFFFRONTLEFT = (char) 10;
    static const char CLIFFFRONTRIGHT = (char) 11;
    static const char CLIFFRIGHT = (char) 12;
    static const char VIRTUALWALL = (char) 13;
    static const char MOTOR_OVERCURRENTS = (char) 14;
    // static const char DIRTLEFT = (char) 15;
    // static const char DIRTRIGHT = (char) 16;
    static const char IRBYTE = (char) 17;
    static const char BUTTONS = (char) 18;
    static const char DISTANCE = (char) 19;
    static const char ANGLE = (char) 20;
    static const char CHARGINGSTATE = (char) 21;
    static const char VOLTAGE = (char) 22;
    static const char CURRENT = (char) 23;
    static const char TEMPERATURE = (char) 24;
    static const char CHARGE = (char) 25;
    static const char CAPACITY = (char) 26;
    static const char WALLSIGNAL = (char) 27;
    static const char CLIFFLEFTSIGNAL = (char) 28;
    static const char CLIFFFRONTLEFTSIGNAL = (char) 29;
    static const char CLIFFFRONTRIGHTSIGNAL = (char) 30;
    static const char CLIFFRIGHTSIGNAL = (char) 31;
    static const char USERDIGITAL = (char) 32;
    static const char USERANALOG = (char) 33;
    static const char CHARGINGSOURCES = (char) 34;
    static const char OIMODE = (char) 35;
}

```

```

static const char SONGNUMBER = (char) 36;
static const char SONGPLAYING = (char) 37;
static const char STREAMPACKETS = (char) 38;
static const char VELOCITY = (char) 39;
static const char RADIUS = (char) 40;
static const char RIGHTVELOCITY = (char) 41;
static const char LEFTVELOCITY = (char) 42;
}
//Contains the iRobot Create instruction opcodes
namespace iRobotInstructions{
static const char START = (char) 128;
static const char BAUD = (char) 129;
static const char CONTROL = (char) 130; //@deprecated
static const char SAFE = (char) 131;
static const char FULL = (char) 132;
static const char SPOT = (char) 134;
static const char COVER = (char) 135;
static const char DEMO = (char) 136;
static const char DRIVE = (char) 137;
static const char LOW_SIDE_DRIVERS = (char) 138;
static const char LEDS = (char) 139;
static const char SONG = (char) 140;
static const char PLAY = (char) 141;
static const char SENSORS = (char) 142;
static const char COVER_AND_DOCK = (char) 143;
static const char PWM_LOW_SIDE_DRIVERS = (char) 144;
static const char DRIVE_DIRECT = (char) 145;
static const char DIGITAL_OUTPUTS = (char) 147;
static const char STREAM = (char) 148;
static const char QUERY_LIST = (char) 149;
static const char PAUSE_RESUME_STREAM = (char) 150;
static const char SEND_IR = (char) 151;
static const char SCRIPT = (char) 152;
static const char PLAY_SCRIPT = (char) 153;
static const char SHOW_SCRIPT = (char) 154;
static const char WAIT_TIME = (char) 155;
static const char WAIT_DISTANCE = (char) 156;
static const char WAIT_ANGLE = (char) 157;
static const char WAIT_EVENT = (char) 158;
}
//Contains the event opcodes that we need to use if we want the iRobot
to wait one of them
namespace iRobotEvents{
static const int WHEEL_DROP = 1;
static const int FRONT_WHEEL_DROP = 2;
static const int LEFT_WHEEL_DROP = 3;
static const int RIGHT_WHEEL_DROP = 4;
static const int BUMP = 5;
static const int LEFT_BUMP = 6;
static const int RIGHT_BUMP = 7;
static const int VIRTUAL_WALL = 8;
static const int WALL = 9;
static const int CLIFF = 10;
static const int LEFT_CLIFF = 11;
static const int FRONT_LEFT_CLIFF = 12;
static const int FRONT_RIGHT_CLIFF = 13;
static const int RIGHT_CLIFF = 14;
static const int HOME_BASE = 15;
static const int ADVANCE_BUTTON = 16;
static const int PLAY_BUTTON = 17;
static const int DIGITAL_INPUT_0 = 18;

```

```

static const int DIGITAL_INPUT_1 = 19;
static const int DIGITAL_INPUT_2 = 20;
static const int DIGITAL_INPUT_3 = 21;
static const int OIMODE_PASSIVE = 22;
}
//Contains the sensor package opcodes that we need to use if we want
to query some sensor values at the same time
namespace iRobotSensorPackage {
static const char SENSORS_7_26 = (char) 0;
//rango 0-25 ->Actualizar 26 chars Cliff, bumper, weels,
virtual wall
static const char SENSORS_DIGITAL BUMPER_CLIFF_WALL_VIRTUALWALL =
(char) 1;
//rango 0-7 ->Actualizar 10 chars (chars 8,9 sin usar)
IR,Buttons,Distance,Angle
static const char SENSORS_IR_BUTTONS_DISTANCE_ANGLE = (char) 2;
//rango 10-15 ->Actualizar 6 chars
static const char SENSORS_BATTERY = (char) 3;
//rango 16-25 ->Actualizar 10 chars
static const char SENSORS_ANALOG_CLIFF_WALL_VIRTUALWALL = (char)
4;
//rango 27-39 ->Actualizar 14 chars
static const char SENSORS_35_42 = (char) 5;
//rango 40-51 ->Actualizar 12 chars
static const char SENSORS_ALL = (char) 6;
//Rango 0-51 ->Actualizar 52 Bytes
}

class iRobotInstructionSet
{
private:
//*****
// Method: intHightByte
// Returns: char [out] Hight byte of the number
// Parameter: short num [in] The number
//*****
char intHightByte(short num);

//*****
// Method: intLowByte
// Returns: char [out] Low byte of the number
// Parameter: short num [in] The number
//*****
char intLowByte(short num);
public:

//*****
// Method: sensorReturn
// Returns: int [out] How many bytes the sensor
generates as response
// Parameter: char sensor [in] The sensor id
//*****
int sensorReturn(char sensor);

iRobotInstructionSet(void);
~iRobotInstructionSet(void);
Instruction start();
Instruction baud(char code);
Instruction control();
Instruction safe();
Instruction full();

```

```

    Instruction spot();
    Instruction cover();
    Instruction demo(char code);
    Instruction drive(int speed, int radius);
    Instruction lowSideDrivers(char outputBit);
    Instruction leds(int ledBit, int ledColor, int ledIntensity);
    Instruction song (int songNumber, int songSize, char *song);
    Instruction playSong(int songNumber);
    Instruction updateSensor(char sensorId);
    Instruction coverAndDock();
    Instruction pwmLowSideDrivers(int driver2,int driver1, int
driver0);
    Instruction driveDirect(int rightVelocity, int leftVelocity);
    Instruction digitalOutputs(int outputBits);
    Instruction stream(char* sensorIdList, int size);
    Instruction queryList(char * sensorIdList, int size);
    Instruction PauseResumeStream(bool bol);
    Instruction sendIr(int data);
    Instruction script(int *commandList, int size);
    Instruction playScript();
    Instruction showScript();
    Instruction waitTime(int seconds);
    Instruction waitDistance(int mm);
    Instruction waitAngle(int degrees);
    Instruction waitEvent(int eventId);
};

#endif

```

## 7.5 IRobotConnection.cpp

```

/* File: IRobotConnection.cpp
 * Author: Gorka Montero 2012 */

#include "IRobotConnection.h"

using namespace std;

IRobotConnection::IRobotConnection(void)
{
    buffer = (char *) malloc(512*sizeof (char));

    connection = Serial();

    // useArduino = false;
    // simulated = true;
}

IRobotConnection::IRobotConnection(const char * connectionType )
{
    buffer = (char *) malloc(512*sizeof (char));

    connection = Serial(connectionType);
    //useArduino = false;
    //simulated = (connectionType == "sim");
}

IRobotConnection::~IRobotConnection(void)
{
}

```

```

int IRobotConnection::createInt(char* aux, int hight /*= 0*/, int low
/*= 1*/ )
{
    return (aux [low]+ (aux[hight]<<8));
}

int IRobotConnection::connect()
{
    //      if(!simulated) this->useArduino = useArduino;
    return connection.connect();
}

void IRobotConnection::disconnect()
{
    connection.disconnect();
}

void IRobotConnection::start()
{
    Instruction aux = iRobotInstructionGenerator.start();
    connection.send(aux.instruction, aux.length);
}

void IRobotConnection::baud(char code )
{
    Instruction aux = iRobotInstructionGenerator.baud(code);
    connection.send(aux.instruction, aux.length);
}

void IRobotConnection::control()
{
    Instruction aux = iRobotInstructionGenerator.control();
    connection.send(aux.instruction, aux.length);
}

void IRobotConnection::safe()
{
    Instruction aux = iRobotInstructionGenerator.safe();
    connection.send(aux.instruction, aux.length);
}

void IRobotConnection::full()
{
    Instruction aux = iRobotInstructionGenerator.full();
    connection.send(aux.instruction, aux.length);
    delay(200);
}

void IRobotConnection::spot()
{
    Instruction aux = iRobotInstructionGenerator.spot();
    connection.send(aux.instruction, aux.length);
}

void IRobotConnection::cover()
{
    Instruction aux = iRobotInstructionGenerator.cover();
    connection.send(aux.instruction, aux.length);
}

```



```

void IRobotConnection::demo(char code )
{
    Instruction aux = iRobotInstructionGenerator.demo(code);
    connection.send(aux.instruction, aux.length);
}

void IRobotConnection::drive(int speed, int radius )
{
    Instruction aux = iRobotInstructionGenerator.drive(speed,radius);
    connection.send(aux.instruction, aux.length);
}

void IRobotConnection::lowSideDrivers(char outputBit )
{
    Instruction aux =
iRobotInstructionGenerator.lowSideDrivers(outputBit);
    connection.send(aux.instruction, aux.length);
}

void IRobotConnection::leds(int ledBit, int ledColor, int ledIntensity
)
{
    Instruction aux =
iRobotInstructionGenerator.leds(ledBit,ledColor,ledIntensity);
    connection.send(aux.instruction, aux.length);
}

void IRobotConnection::song(int songNumber, int songSize, char *song )
{
    Instruction aux =
iRobotInstructionGenerator.song(songNumber,songSize,song);
    connection.send(aux.instruction, aux.length);
}

void IRobotConnection::playSong(int songNumber)
{
    Instruction aux = iRobotInstructionGenerator.playSong(songNumber);
    connection.send(aux.instruction, aux.length);
}

int IRobotConnection::updateSensor(char sensorId )
{
    Instruction aux =
iRobotInstructionGenerator.updateSensor(sensorId);
    connection.send(aux.instruction, aux.length);
    connection.receive(buffer,aux.response);
    if (aux.response == 2) return(createInt(buffer));
    return buffer[0];
}

void IRobotConnection::coverAndDock()
{
    Instruction aux = iRobotInstructionGenerator.coverAndDock();
    connection.send(aux.instruction, aux.length);
}

void IRobotConnection::pwmLowSideDrivers(int driver2,int driver1, int
driver0 )
{
    Instruction aux =
iRobotInstructionGenerator.pwmLowSideDrivers(driver2,driver1,driver0);

```

```

        connection.send(aux.instruction, aux.length);
    }

void IRobotConnection::driveDirect(int rightVelocity, int leftVelocity
)
{
    Instruction aux =
iRobotInstructionGenerator.driveDirect(rightVelocity,leftVelocity);
    connection.send(aux.instruction, aux.length);
}

void IRobotConnection::digitalOutputs(int outputBits )
{
    Instruction aux =
iRobotInstructionGenerator.digitalOutputs(outputBits);
    connection.send(aux.instruction, aux.length);
}

void IRobotConnection::stream(char* sensorIdList, int size )
{
    Instruction aux =
iRobotInstructionGenerator.stream(sensorIdList,size);
    connection.send(aux.instruction, aux.length);
}

int IRobotConnection::queryList(char * sensorIdList, int size )
{
    Instruction aux =
iRobotInstructionGenerator.queryList(sensorIdList,size);
    connection.send(aux.instruction, aux.length);
    connection.receive(buffer,aux.response);
    return aux.response;
}

void IRobotConnection::PauseResumeStream(bool bol )
{
    Instruction aux =
iRobotInstructionGenerator.PauseResumeStream(bol);
    connection.send(aux.instruction, aux.length);
}

void IRobotConnection::sendIr(int data )
{
    Instruction aux = iRobotInstructionGenerator.sendIr(data);
    connection.send(aux.instruction, aux.length);
}

void IRobotConnection::script(int *commandList, int size )
{
    Instruction aux =
iRobotInstructionGenerator.script(commandList,size);
    connection.send(aux.instruction, aux.length);
    lastScriptSize = size;
}

void IRobotConnection::playScript()
{
    Instruction aux = iRobotInstructionGenerator.playScript();
    connection.send(aux.instruction, aux.length);
}

```

```

void IRobotConnection::showScript()
{
    Instruction aux = iRobotInstructionGenerator.showScript();
    connection.send(aux.instruction, aux.length);
    connection.receive(buffer, lastScriptSize);
}

void IRobotConnection::waitTime(int seconds )
{
    Instruction aux = iRobotInstructionGenerator.waitTime(seconds);
    connection.send(aux.instruction, aux.length);
}

void IRobotConnection::waitDistance(int mm )
{
    Instruction aux = iRobotInstructionGenerator.waitDistance(mm);
    connection.send(aux.instruction, aux.length);
}

void IRobotConnection::waitAngle(int degrees )
{
    Instruction aux = iRobotInstructionGenerator.waitAngle(degrees);
    connection.send(aux.instruction, aux.length);
}

void IRobotConnection::waitEvent(int eventId )
{
    Instruction aux = iRobotInstructionGenerator.waitEvent(eventId);
    connection.send(aux.instruction, aux.length);
}

void IRobotConnection::setVerboseMode(int flag_debug){
    if(flag_debug==1){
        connection.setVerboseMode(1);
    }else{
        connection.setVerboseMode(0);
    }
}

```

## 7.6 IRobotConnection.h

```

/* File:    IRobotConnection.h
 * Author:  Gorka Montero 2012 */

#ifndefIROBOTCONNECTION_H
#defineIROBOTCONNECTION_H

#pragma once
#include <iostream>
#include "IRobotInstructionSet.h"
#include "Serial.h"

class IRobotConnection
{
private:
    // bool simulated;
    // bool useArduino;

    Serial connection;

```

```

        iRobotInstructionSet iRobotInstructionGenerator;

//If we want to implement some instruction with the Arduino board pins
we must use this instruction set
//ArduinoInstructionSet ArduinoInstructionGenerator;

        char * buffer;

//*****
// Method:      createInt
// Returns:     int      [out] integer that we want to create
// Parameter:   char * aux  [in] Buffer where the hight byte and
//              low byte of the integer are
// Parameter:   int hight  [in] Position of the hight byte in the
//              buffer
// Parameter:   int low    [in] Position of the low byte in the
//              buffer
//*****
int createInt(char* aux, int hight = 0, int low = 1);

        int lastScriptSize; //Stores the last script size sent to the robot
public:

//*****
// Method:      IRobotConnection
// Description:  Starts the connection using the simulator
//*****
IRobotConnection(void);
//*****
// Method:      IRobotConnection
// Parameter:   char * connectionType  [in] "sim" if we want to use
//              the simulator, COMx if we want to use the serial communication
//*****
IRobotConnection(const char * connectionType);
~IRobotConnection(void);

//*****
// Method:      connect
// Parameter:   bool useArduino  [in] True -> if serial
//              communication is going on it will choose Arduino config
// Description:
//              Starts the connection
//*****
int connect();
void disconnect();
void start();
void baud(char code);
void control();
void safe();
void full();
void spot();
void cover();
void demo(char code);
void drive(int speed, int radius);
void lowSideDrivers(char outputBit);
void leds(int ledBit, int ledColor, int ledIntensity);
void song (int songNumber, int songSize, char *song);
void playSong(int songNumber);
int updateSensor(char sensorId);

```

```

void coverAndDock();
void pwmLowSideDrivers(int driver2,int driver1, int driver0);
void driveDirect(int rightVelocity, int leftVelocity);
void digitalOutputs(int outputBits);
void stream(char* sensorIdList, int size);
int queryList(char * sensorIdList, int size);
void PauseResumeStream(bool bol);
void sendIr(int data);
void script(int *commandList, int size);
void playScript();
void showScript();
void waitTime(int seconds);
void waitDistance(int mm);
void waitAngle(int degrees);
void waitEvent(int eventId);

void setVerboseMode(int flag_debug);
};

#endif

```

## 7.7 Serial.cpp

```

/* File:    Serial.cpp
 * Author:  @bgamecho 2015 */

#include "Serial.h"

using namespace std;

Serial::Serial(void){
    portName = "/dev/ttyUSB0";
    flag_debug = 0;
}

Serial::Serial(const char * type){
    portName = type;
    flag_debug = 0;
}

Serial::~Serial(void){
}

int Serial::connect(){
    fd_serial = serialOpen(portName, 57600);
    if(fd_serial < 0 ){
        if(flag_debug){
            std::cout << "[V]\tError opening port: " << portName <<
std::endl;
        }
        return false;
    }
    return true;
}

int Serial::send(char *data, int size){
    int index;

    if(flag_debug){
        std::cout << "[V]\tSend command: " << std::endl;
    }
}

```

```

        std::cout << "[V]\t\tCommand size: " << size << std::endl;
    }

    for (index = 0; index < size; index++){
        serialPutchar(fd_serial, data[index]);
        if(flag_debug){
            std::cout << "[V]\t\t" << index << " - " << +data[index] <<
std::endl;
        }
        delay(10); // wait 10 ms to send the command
    }

    return 0;
}

int Serial::receive(char *buffer, size_t size){
    int index;

    if(flag_debug){
        std::cout << "[V]\tReceive command: " << std::endl;
    }
    delay(10); // wait some time before receiving data

    for (index = 0; index < (int) size; index++){
        if(flag_debug){
            int val = serialDataAvail(fd_serial) ;
            std::cout << "[V]\t\tBytes available for reading: " << val
<< std::endl;
        }

        int aux = serialGetchar(fd_serial);
        buffer[index] = (char) aux;

        if(flag_debug){
            std::cout << "[V]\t\t"<< index <<" Read: " << +aux <<
std::endl;
        }
    }

    return 0;
}

void Serial::disconnect(){
    serialClose(fd_serial);
}

void Serial::setVerboseMode(int val){
    if (val == 0){
        flag_debug = 0;
    }else{
        flag_debug = 1;
    }
}
}

```

## 7.8 Serial.h

```

/* File:    Serial.h
 * Author:  @bgamecho 2015 */

#ifndef SERIAL_H

```

```

#define SERIAL_H

#pragma once
#include <cstdlib>
#include <unistd.h>
#include <iostream>
#include <wiringPi.h>
#include <wiringSerial.h>

class Serial
{
private:
    const char * portName;
    int fd_serial;
    int flag_debug;

public:
    Serial(void);
    Serial(const char * type);
    ~Serial(void);
    int connect();
    int send(char *data, int size);
    int receive(char *buffer, size_t size);
    void disconnect();
    void setVerboseMode(int val);
};

#endif /* SERIAL_H */

```