universidad del país vasco — euskal herriko unibertsitatea

Facultad de Informática — Informatika Fakultatea

TITULACIÓN: Ingeniería Informática

Un entorno de programación multinivel y control modular de robots.

Alumno/a: D./Dña. Gorka Montero Gonzalez

Director/a: D./Dña. Edurne Larraza Mendiluze

Proyecto Fin de Carrera, julio de 2012

Resumen

En este proyecto final de carrera se van a tratar los aspectos referentes a la ampliación de robots. Para ello se utilizará una placa Arduino que se comunicará con el robot por puerto serie. Esta placa, servirá de plataforma de comunicación entre un PC y el robot, ofreciendo una interfaz del robot anterior con la capacidad de ampliación de la placa Arduino. En el transcurso del proyecto se ha realizado una capa intermedia de código C++ que gestiona el uso de la placa Arduino y del robot iRobot Create a través de la misma.

Con objeto de dar también soporte a la programación del robot iRobot Create, se ha elegido un simulador y se le ha dado soporte en la capa anteriormente citada.

Indice

1 Introducción

Este proyecto fin de carrera esta centrado en el uso del robot iRobot Create y en sus posibilidades. El éxito de estos robots en el terreno de la investigación se debe al bajo coste que tienen, pero a la hora de trabajar con ellos, sus limitaciones se descubren rápidamente.

Es bien sabido que programar requiere una cantidad considerable de ejecuciones de la aplicación por parte del desarrollador, bien sea para comprobar que todo funciona o para asegurarse de que no ha habido ningún despiste por parte del mismo y que las modificaciones realizadas hacen lo que realmente se espera que hagan. Esto, en la programación con robots, requeriría el uso del robot para cualquier comprobación, lo cual hace que el desarrollador quede pendiente de la disponibilidad del robot y de la velocidad de ejecución del mismo, ralentizándose así la velocidad de desarrollo con los mismos. Es por ello que existen los simuladores de robots y es por ello que se ha elegido un simulador del robot iRobot Create que permitirá a los futuros desarrolladores trabajar y comprobar las repercusiones de los cambios realizados en sus aplicaciones sin la necesidad de tener el robot físicamente a su disposición, permitiendo de esta manera que su velocidad de desarrollo aumente de manera considerable.

Como bien se ha dicho al comienzo de esta introducción, el robot iRobot Create es un robot bastante económico pero bastante limitado. Esto se debe a la poca cantidad de sensores y actuadores que posee. Como también se ha dicho anteriormente, cualquier persona que haya querido realizar alguna tarea especifica con el robot en cuestión, se habrá encontrado rápidamente con la gran cantidad de limitaciones que le acompañan. Este es el motivo por el cual se ha comunicado el robot con una placa Arduino por puerto serie. La placa ha sido programada con un juego de instrucciones que permitirá configurar la placa y usarla de manera remota, por medio de una conexión bluetooth.

Por último se ha realizado una capa intermedia de instrucciones de alto nivel, programada en C++, que permitirá trabajar de una manera mucho mas sencilla e intuitiva tanto con la placa Arduino, como con el robot iRobot Create. De esta manera, queda de cara a los desarrolladores un robot con las posibilidades del robot iRobot Create y la ampliabilidad de una placa Arduino. Y permitiendo que con el cambio de un parámetro se permita: Usar el simulador, usar el robot en su versión oficial o usar el robot con la placa Arduino que se le ha acoplado.

2 Objetivos

El objetivo de este proyecto es ofrecer a los futuros alumnos tanto de la asignatura de R*obótica y control industrial* como de la asignatura *Robótica, sensores y actuadores* una serie de herramientas para trabajar con el robot iRobot Create. A la hora de llevar adelante el proyecto se ha querido primar lo siguiente:

- **Reducir el tiempo de aprendizaje de los alumnos**. Para ello había que minimizar la cantidad de lenguajes de programación que se iban a utilizar. Por otro lado las instrucciones creadas deberían de ser sencillas de utilizar.
- **Aumentar el rendimiento de los alumnos.** Es por ello que se ha aportado al proyecto un simulador del robot y unas instrucciones de alto nivel para la programación del robot.
- **Permitir futuras ampliaciones.** Se ha pretendido que los posibles cambios de hardware afecten lo menos posible al software que se aporta en el proyecto.
- **Aumentar la versatilidad del robot iRobot Create.** Las cantidad de tareas que se pueden realizar con el robot de serie son muy limitadas.

3 Descripción de los elementos

3.1 Robot iRobot Create

3.1.1 Descripción

Para los que no conozcan este robot, diremos que ha sido creado por la empresa iRobot, fundada en 1990 por miembros del *instituto tecnológico de robótica de Massachusetts*. Este robot es una versión para investigadores de un famoso robot-aspiradora creado por la misma compañia, llamado *iRobot Roomba*.[1]

iRobot Create es un robot, como ya hemos dicho anteriormente, enfocado a la investigación y que ha tenido una buena acogida por parte de la comunidad de investigadores debido a su bajo coste. A continuación se muestran un par de desarrollos realizados utilizando este robot:

- En este desarrollo se le ha acoplado un dispositivo kinect al robot con objetivo de crear un grupo de robots capaces de ayudar a encontrar gente desaparecida o atrapada:

http://singularityhub.com/2010/11/17/hacked-irobot-uses-xbox-kinect-to-see-the-world-obey-your-commands-video/

- Este en cambio, es un desarrollo en el que se comunica un dispositivo Android con un iRobot Create, de forma que, aprovechando el reconocimiento de voz de Android, se pueda controlar el robot por comandos de voz:

http://www.instructables.com/id/Voice-Controlled-iRobot-Create/?ALLSTEPS

1 Para mas información acerca de los creadores visitar la siguiente dirección: http://www.irobot.com/es/explore_irobot/company_history.aspx

3.1.2 Hardware

El hardware del robot se compone de:

- 2 sensores de choque
- 4 sensores de precipicio
- 1 sensor infrarrojo
- 1 receptor de señales infrarrojas
- 2 ruedas con control de velocidad independiente
- 3 Leds
- 3 Botones
- 1 Conexión DB-25
- 1 Conexión Mini-DIN

3.1.3 Descripción de los pin del conector DB-25

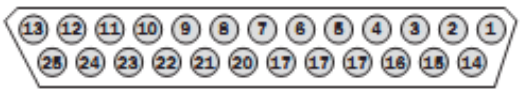


Imagen 3.2: Puerto DB-25 de iRobot Create

Pin	Name	Description
1	RXD	0 – 5V Serial input to Create
2	TXD	0 – 5V Serial output from Create
3	Power control toggle	Turns Create on or off on a low-to-high transition
4	Analog input	0 - 5V analog input to Create
5	Digital input 1	0 - 5V digital input to Create
6	Digital input 3	0 - 5V digital input to Create
7	Digital output 1	0 - 5V, 20 mA digital output from Create
8	Switched 5V	Provides a regulated 5V 100 mA supply and analog reference voltage when Create is switched on
9	Vpwr	Create battery voltage (unregulated), 0.5A
10	Switched Vpwr	Provides battery power @ 1.5 A when Create is powered on.
11	Switched Vpwr	Provides battery power @ 1.5 A when Create is powered on.
12	Switched Vpwr	Provides battery power @ 1.5 A when Create is powered on.
13	Robot charging	When Create is charging, this pin is high (5V)
14	GND	Create battery ground
15	Device Detect/Baud Rate Change Pin	0-5V digital input to Create which can also be used to change the baud rate to 19200 (see below)
16	GND	Create battery ground
17	Digital input 0	0 - 5V digital input to Create
18	Digital input 2	0 - 5V digital input to Create
19	Digital output 0	0 - 5V, 20 mA digital output from Create
20	Digital output 2	0 - 5V, 20 mA digital output from Create
21	GND	Create battery ground
22	Low side driver 0	0.5A low side driver from Create
23	Low side driver 1	0.5A low side driver from Create
24	Low side driver 2	1.5A low side driver from Create
25	GND	Create battery ground

Imagen 3.1: Descripción de los pines del puerto DB-25

Como se puede apreciar en las imágenes 3.1 y 3.2, los pines 1 y 2 del conector corresponden a una conexión serie cuyos parámetros de configuración son los siguientes:

Baudrate:57600

Byte size: 8

Parity: None

StopBits: 1

3.1.4 Juego de instrucciones

En la imagen 3.3 se muestra una tabla con el juego de instrucciones que ofrece el fabricante para trabajar con este robot. Estas instrucciones se identifican por medio de un byte, único para cada instrucción, llamado *opcode.*

Command	Opcode	Data Bytes: 1	Data Bytes: 2	Data Bytes: 3	Data Bytes: 4	Etc.
Start	128					
Baud	129	Baud Code: (0 = 11)				
Control	130					
Safe	131					
Full	132					
Spot	134					
Cover	135					
Demo	136	Demos (-1 - 9)				
Drive	137	Velocity (-500 = 500 mm/s)		Radius (-2000 = 2000 mm)		
Low Side Drivers	138	Output Bits (0 = 7)				
LEDs	139	LED Bits (0 = 10)	Power LED Color (0 = 255)	Power LED Intensity (0 = 255)		
Song	140	Song Number (0 - 15)	Song Length (1 - 16)	Note Number 1 (31 = 27)	Note Duration 1 (0 - 255)	Note Number 2, etc.
Play	141	Song Number: (0 = 15)				
Sensors	142	Packet ID: (0 = 42)				
Cover and Dock	143					
PWM Low Side Drivers	144	Low Side Driver 2 Duty Cycle (0 - 128)	Low Side Driver 1 Duty Cycle (0 - 128)	Low Side Driver 0 Duty Cycle (0 - 128)		
Drive Direct	145	Right wheel velocity (-500 = 500 mm/s)		Left wheel velocity (-500 = 500 mm/s)		
Digital Outputs	147	Output Bits (0 =7)				
Stream	148	Number of Packets	Packet ID 1 (0 = 42)	Packet ID 2, etc.		
Query List	149	Packet ID 1 (0 = 42)	Packet ID 2, etc.			
Pause/Resume Stream	150	Range: 0-1				
Send IR	151	Byte (0 - 255)				
Script	152	Script Length: (1 = 100)	Command Opcode 1	Command Data Byte 1, etc.	Command Opcode 2	Etc.
Play Script	153					
Show Script	154					
Wait Time	155	Time (0 = 255 seconds/10)				
Wait Distance	156	Distance (-32767 - 32768 mm)				
Wait Angle	157	Angle (-32767 - 32768 degrees)				
Wait Event	158	Event ID (1 to 20 and -1 to -20)				

Imagen 3.3: Juego de instrucciones de iRobot Create

Quedando las instrucciones de la siguiente manera: Supongamos que queremos hacer que el robot ejecute la demo 1 que trae preestablecida. Para ello habría que enviarle al robot por puerto serie la siguiente secuencia de bytes: 136 1

3.1.5 Organización de los sensores

En el juego de instrucciones de la imagen 3.3 se ve que los sensores del robot se pueden consultar de manera individual, o por paquetes. En las imágenes 3.4 y 3.5 se puede apreciar la agrupación de los sensores en paquetes junto con que byte corresponden a que paquete y los bytes que corresponden a cada sensor de manera individual:

Packet Membership							Name	Bytes	Value Range	Units
0	1					6	7 Bumps and Wheel Drops	1	0 - 31	
							8 Wall	1	0 - 1	
							9 Cliff Left	1	0 - 1	
							10 Cliff Front Left	1	0 - 1	
							11 Cliff Front Right	1	0 - 1	
							12 Cliff Right	1	0 - 1	
							13 Virtual Wall	1	0 - 1	
							14 Overcurrents	1	0 - 31	
							15 Unused	1	0	
							16 Unused	1	0	
		2					17 IR Byte	1	0 - 255	
							18 Buttons	1	0 - 15	
							19 Distance	2	-32768 - 32767	mm
							20 Angle	2	-32768 - 32767	mm
			3				21 Charging State	1	0 - 5	
							22 Voltage	2	0 - 65535	mV
							23 Current	2	-32768 - 32767	mA
							24 Battery Temperature	1	-128 - 127	degrees Celsius
							25 Battery Charge	2	0 - 65535	mAh
							26 Battery Capacity	2	0 - 65535	mAh

Imagen 3.4: Tabla de agrupación de sensores (parte 1)

Packet Membership						Name	Bytes	Value Range	Units
				4		27 Wall Signal	2	0 - 4095	
						28 Cliff Left Signal	2	0 - 4095	
						29 Cliff Front Left Signal	2	0 - 4095	
						30 Cliff Front Right Signal	2	0 - 4095	
						31 Cliff Right Signal	2	0 - 4095	
						32 User Digital Inputs	1	0 - 31	
						33 User Analog Input	2	0 - 1023	
						34 Charging Sources Available	1	0 - 3	
				5		35 OI Mode	1	0 - 3	
						36 Song Number	1	0 - 15	
						37 Song Playing	1	0 - 1	
						38 Number of Stream Packets	1	0 - 42	
						39 Velocity	2	-500 - 500	mm/s
						40 Radius	2	-32768 - 32767	mm
						41 Right Velocity	2	-500 - 500	mm/s
						42 Left Velocity	2	-500 - 500	mm/s

Imagen 3.5: Tabla de agrupación de sensores (parte 2)

Si se quisiera leer el estado del sensor de precipicio frontal izquierdo habría que enviar la siguiente secuencia de bytes al robot: 142 10

3.1.6 Eventos

En el juego de instrucciones de iRobot debemos destacar también la función wait, que hace que el robot no atienda a mas instrucciones hasta que suceda algún evento. Los eventos que reconoce el robot se pueden encontrar en la imagen 3.6.

Event	Number	Unsigned Equivalent of Inverse
Wheel Drop	1	255
Front Wheel Drop	2	254
Left Wheel Drop	3	253
Right Wheel Drop	4	252
Bump	5	251
Left Bump	6	250
Right Bump	7	249
Virtual Wall	8	248
Wall	9	247
Cliff	10	246
Left Cliff	11	245
Front Left Cliff	12	244
Front Right Cliff	13	243
Right Cliff	14	242
Home Base	15	241
Advance Button	16	240
Play Button	17	239
Digital Input 0	18	238
Digital Input 1	19	237
Digital Input 2	20	236
Digital Input 3	21	235
OI Mode = Passive	22	234

Imagen 3.6: Eventos de iRobot Create

A la hora de utilizar esta instrucción para hacer que el robot no atienda a mas instrucciones hasta que detecte que se ha presionado el botón de play habría que enviar la siguiente secuencia de bytes: 158 17

Para mas información visitar las siguientes direcciones:

http://www.irobot.com/filelibrary/pdfs/hrd/create/Create%20Manual_Final.pdf

http://en.wikipedia.org/wiki/IRobot_Create

3.2 Arduino

3.2.1 Descripción

Arduino es una plataforma de desarrollo de computación física (physical computing) de código abierto. En otras palabras, Arduino es una placa con un sencillo microcontrolador y un entorno de desarrollo para crear software (programas) para la placa. Las placas Arduino están basadas en los micro-controladores ATMEGA168, ATMEGA328 y ATMEGA1280.

Por otro lado el software consiste en un entorno de desarrollo que trabaja con el lenguaje de programación Processing/Wiring, ofreciendo funciones de coloreado de código y permitiendo compilar el programa y enviarlo a la placa.

Las características de esta placa son:

1. El bajo coste de la misma.
2. El hardware es abierto, pudiendo encontrarse en la misma página de Arduino los esquemas de las placas, de forma que quien quiera se pueda construir su placa, o que quien lo necesite pueda crearse una versión de la placa mas ajustada a sus necesidades.
3. El software es abierto y esta publicado también en su página web.
4. El software es ampliable a través de librerías de C++, y si se está interesado en profundizar en los detalles técnicos, se puede dar el salto a la programación en el lenguaje AVR C en el que está basado. De igual modo se puede añadir directamente código en AVR C en los programas si así se desea.
5. Es un software multiplataforma y funciona en los sistemas operativos Windows, Macintosh OSX y Linux.

A la hora de trabajar con la placa Arduino, el modo de trabajar sería el siguiente:

1. Se conecta la placa por usb a un ordenador.
2. Se ejecuta el software de desarrollo de Arduino.
3. Se escribe el código del programa que se desea enviar a la placa.
4. Se compila el programa y si se ha compilado bien, se envía el archivo compilado a la placa.
5. Automáticamente la placa se resetea y empieza a ejecutar el programa en cuestión.

A la hora de programar una placa Arduino debemos saber que todo programa de Arduino tiene dos funciones base:

- La función setup

  Esta función se ejecuta cada vez que la placa se resetea, es la encargada de configurar los pines de la placa e inicializar las variables que necesitemos inicializar.

- La función loop

  Esta función seria el equivalente a la función main de cualquier otro lenguaje de programación, con la característica de que en Arduino esta función representa un bucle infinito, de forma que todo el código que se introduzca en ella se va a estar ejecutando de manera reiterada una y otra vez.

3.2.2 El hardware

Antes de profundizar en el hardware de Arduino, debemos decir que hay varias versiones de la placa arduino. En este proyecto se hablará de dos versiones en concreto: por un lado la versión Arduino UNO, cuyo esquema hardware se puede apreciar en la imagen 3.7, y por otro lado la versión Arduino Mega, la cual se puede ver en la imagen 3.8.

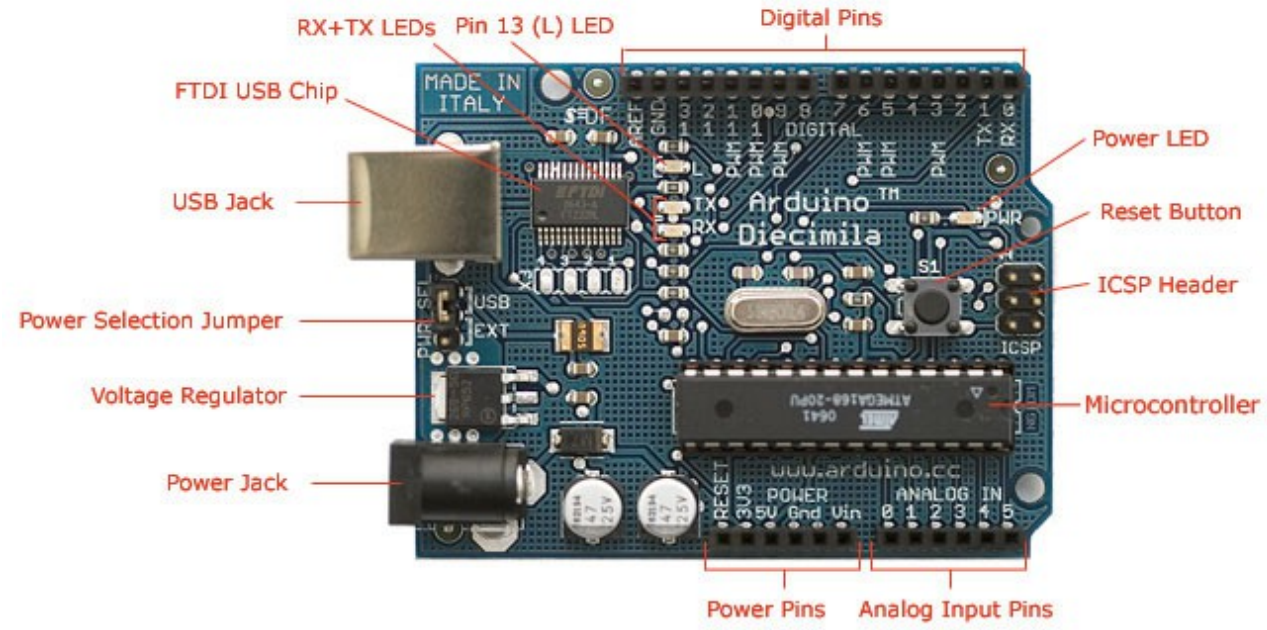


Imagen 3.7: Esquema del hardware de Arduino Diecimila (versión antigua del Arduino UNO)

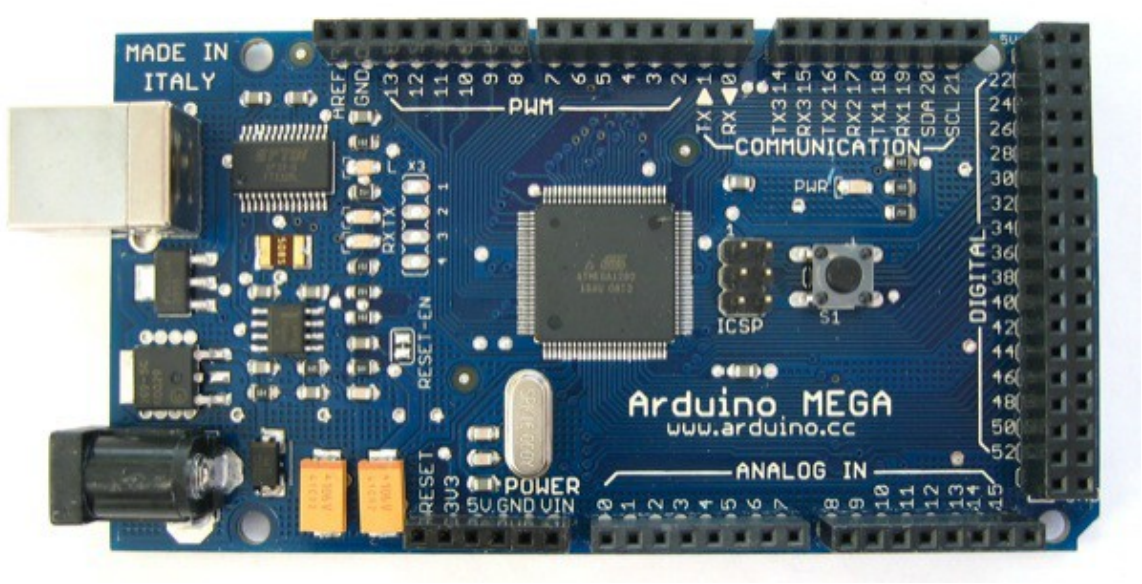


Imagen 3.8: Placa Arduino Mega

Como se puede apreciar, las diferencias entre Arduino UNO y arduino Mega radican simplemente en la cantidad de conectores que contienen, dando la segunda soporte para una cantidad de sensores o actuadores mayor que la primera. Debemos destacar también el soporte nativo para múltiples conexiones por puerto serie que ofrece la versión Mega de la placa frente a la versión UNO que solo lo ofrece para un puerto serie.

Para mas información visitar la siguiente dirección:

http://arduino.cc/es/

3.3 Visual Studio 2010

Visual Studio es un entorno de desarrollo creado por Microsoft (véase imagen 3.9) para el desarrollo de aplicaciones para Windows. Es sencillo de utilizar, ofrece muchas ayudas a la hora de programar en C++ y tiene un potente debugger que facilita la detección de errores de ejecución.

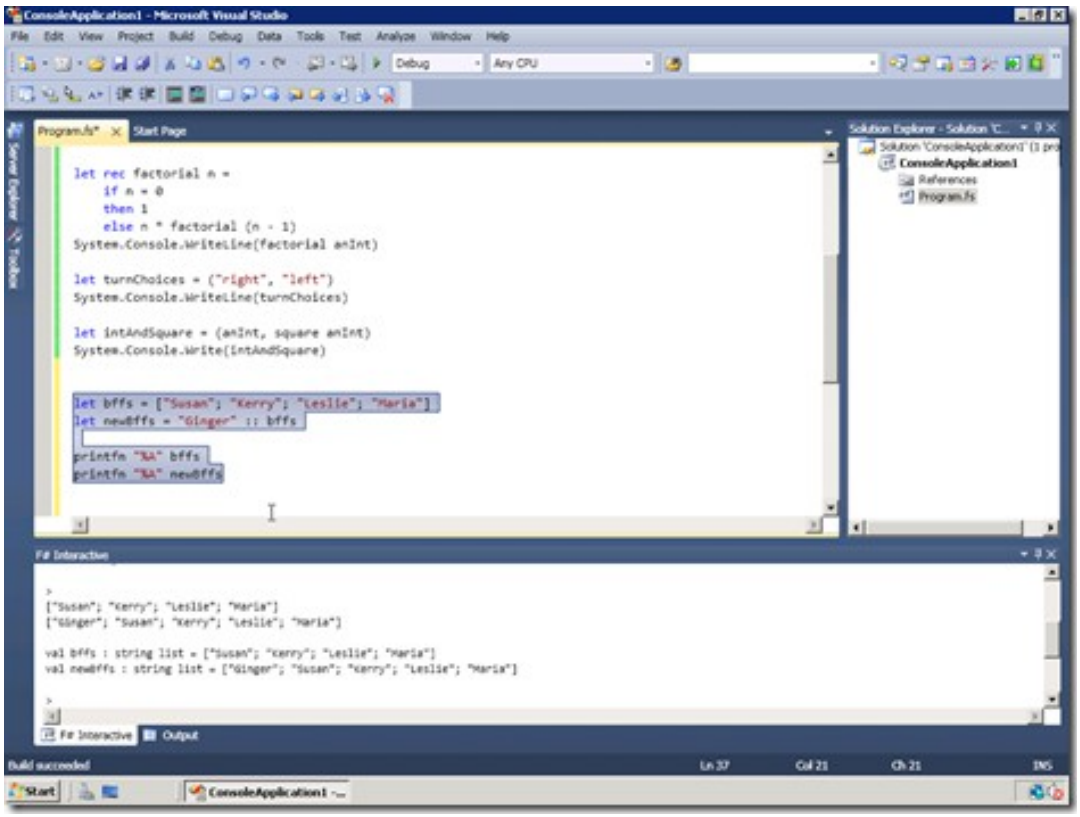


Imagen 3.9: Visual Studio 2010

Otra de las características de Visual Studio es que es completamente personalizable, permitiendo cambiar su apariencia prácticamente por completo. Además, tiene soporte para macros y sus funcionalidades se pueden ampliar por medio de Addins[2].

Para mas información visitar la dirección:

http://www.microsoft.com/visualstudio/es-es

2 Estos Addins podrán encontrarse en la siguiente dirección: http://visualstudiogallery.msdn.microsoft.com/

3.4 Simulador iRobot Create simulator

Este simulador ha sido desarrollado por **Rose-Hulman Institute of Technology - Computer Science & Software Engineering** para que fuera usado por sus alumnos.

Sus desarrolladores lo crearon para una asignatura en la que se programaba usando pyton y viene con una serie de librerías para poder comunicarse con él escritas en pyton, las cuales fueron utilizadas como guía a la hora de obtener los parámetros de conexión con el simulador.

Este simulador nos ofrece a simple vista la siguiente información del robot:

- Lista de instrucciones que ha recibido el robot.
- Estado de los sensores del robot.
- Odometría del robot.

Aparte el simulador cuenta con un editor de entornos integrado, 4 cámaras o vistas (véase figuras de la 3.11 a la 3.13), para el seguimiento de la simulación y por último opciones de aumento de la velocidad de simulación.

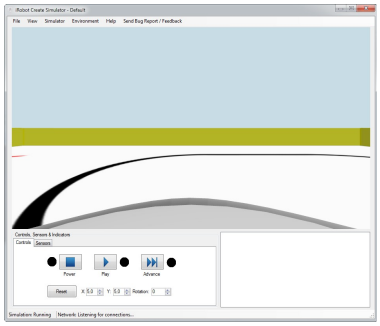


Imagen 3.11: On board view

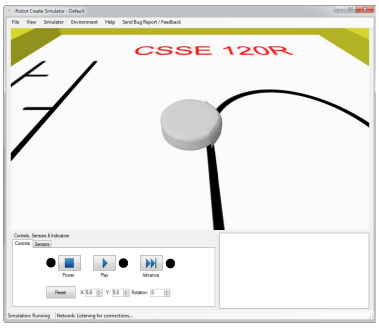


Imagen 3.10: User view

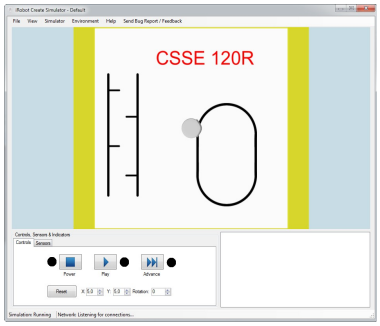
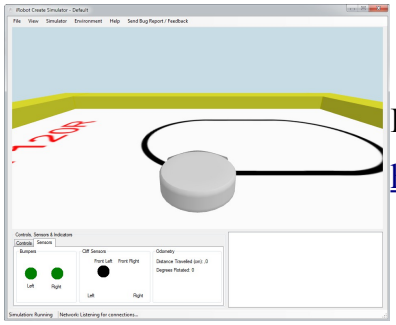


Imagen 3.12: Top view

En la imagen 3.13 se puede apreciar la vista follow view, junto con la información de los distintos sensores, la odometría del robot y las instrucciones ejecutadas (en este caso ninguna) que aparecerían en el rectángulo blanco a la izquierda de la odometría del robot.



Para mas información visitar la siguiente dirección: http://www.rose-hulman.edu/~boutell/simulator/index.html

Imagen 3.13: Follow view

3.5 Módulo bluetooth JY-MCU

El módulo JY-MCU, como puede apreciarse en la imagen 3.14, es un módulo bluetooth diseñado para trabajar con placas de prototipado[3]. Trabaja con un voltaje 5 voltios y se comunica con un puerto serie que utiliza la siguiente conexión por defecto:

Baudrate:38400

Byte size: 8

Parity: None

StopBits: 1

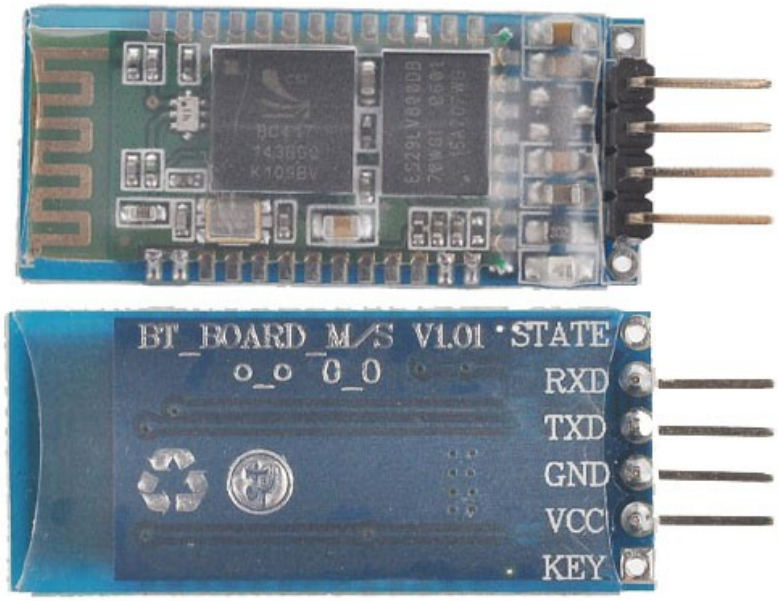


Imagen 3.14: Modulo bluetooth JY-MCU

El nombre por defecto de este dispositivo es *lindor* y su contraseña por defecto es *1234*. No obstante el baudrate al que trabaja el dispositivo, ha sido modificado a 9600 por medio de una serie de comandos de configuración del módulo[4], para ajustarlo a las restricciones de la librería softwareSerial de Arduino.

3 Una placa de prototipado es un tablero con orificios conectados eléctricamente entre si, habitualmente siguiendo patrones de líneas, en el cual se pueden insertar componentes electrónicos y cables para el armado y prototipado de circuitos electrónicos y sistemas similares

4 Los comandos para modificar la configuración del dispositivo se pueden encontrar en la siguiente dirección: http://tallerarduino.wordpress.com/2011/12/06/modulo-bluetooth-hc-06-o-gp-gc021-y-arduino/

3.6 Motivos por los que se ha elegido cada componente

3.6.1 Arduino

Se ha elegido una placa Arduino porque tiene la capacidad de ejecutar aplicaciones de manera independiente, con una velocidad de ejecución relativamente fluida y tiene un coste bastante asequible.

Otra de las ventajas de Arduino radica en que es hardware abierto, permitiendo en caso de necesidades concretas, el poder crear una placa que se ajuste mas a nuestro problema y que pueda correr el software que previamente se haya creado para Arduino.

El código open source de arduino también ha sido considerado como una ventaja muy importante ya que, queríamos una placa sobre la que tuviéramos el mayor control posible sobre ella.

Otra gran ventaja de Arduino es la comunidad que tiene desarrollando para ella, gracias a ella, existe un gran abanico de ampliaciones[5] que hay creadas para la placa que le permiten especializarse en labores muy distintas.

De todas las versiones disponibles de Arduino se ha elegido la versión mega y esto se debe principalmente a que ofrece soporte nativo para varias conexiones por puerto serie, y también, a que ofrece una mayor cantidad de pines tanto analógicos cómo digitales. No obstante, el proyecto a sido realizado con el modelo Arduino UNO ya que era el único disponible en el momento de realizarlo.

5 En esta pagina se puede encontrar una muestra de las distintas ampliaciones desarrolladas tanto por la comunidad como por los desarrolladores de Arduino: http://www.arduino.cc/playground/Main/SimilarBoards#goShie

3.6.2 Visual Studio 2010

Si bien es cierto que con este entorno de desarrollo solo podemos trabajar en un entorno Windows, también es cierto que ofrece bastantes ayudas a la hora de programar, y sobre todo ofrece un potente debugger que ayuda considerablemente a la hora de detectar errores de ejecución.

El motivo principal que nos ha llevado a elegir este entorno de desarrollo frente a otros, ha sido lo sencillo de utilizar que es. Puesto que este proyecto esta pensado para ser utilizado para impartir clase, se ha considerado que este entorno de desarrollo ofrece unas facilidades tanto para los alumnos como para los profesores que son muy interesantes.

En lo que al profesorado concierne, Visual Studio le permitirá entregar a los alumnos proyectos base con todas sus dependencias, de forma que los alumnos no pierdan tiempo instalando librerías externas ni configurando nada, y evitando los posibles errores que los alumnos puedan cometer en el proceso. Esta es una ventaja muy importante, ya que, cómo alumno que soy, el mayor problema al que me enfrentaba a la hora de tratar con el código fuente que me daban los profesores, era el ponerlo en marcha por primera vez para ver cómo funcionaba.

Por el lado del alumnado, Visual Studio les ayudará a comprender rápidamente el código fuente y los tipos de datos que se usan en él. Aparte de ello, el debugger les ayudará a detectar los errores de ejecución de una manera mas sencilla. Y las ayudas de auto completado les permitirán escribir código mas eficientemente.

Las alternativas a este entorno de desarrollo, como pueden ser, eclipse[6] o Codeblocks[7] fueron descartadas por los siguientes motivos:

Aunque el simulador elegido tenia una integración con eclipse, el plugin para desarrollar en C++ de eclipse contiene demasiados bugs, esto hacia que eclipse terminara mostrando errores en programas que realmente compilaban y se ejecutaban sin problemas, motivo mas que suficiente para descartarlo.

Codeblocks en cambio, es un entorno de desarrollo gratuito, pero ni la ayuda de autocompletado ni el debugger, se acercan por asomo a los de Visual Studio.

6 Para saber más sobre eclipse visitar la siguiente dirección: http://www.eclipse.org/
7 Para saber más sobre Codeblocks visitar la siguiente dirección: http://www.codeblocks.org/

3.6.3 Simulador iRobot Create simulator

La selección de este simulador se realizó por dos motivos:

- Su integración con eclipse, el cual en un principio iba a ser el entorno de desarrollo elegido, pero debido a lo mal que funcionaba el plugin de C++ de eclipse, que mostraba errores de código donde no los había se descartó por completo.
- Su sencillez, motivo realmente importante a la hora de ahorrar tiempo de enseñanza.

Como alternativa a este simulador, se presentaron dos candidatos: "Microsoft Robotics developer studio"[8] por un lado y "player/Stage"[9]. El primero, desarrollado por Microsoft, es aparentemente bastante potente, pero solo tiene soporte para desarrollos en C#. Y el segundo, sólo funciona en linux y al haberse elegido Visual Studio como entorno de desarrollo que sólo funciona el windows, hemos tenido que descartarlo.

8 Para más información visitar la siguiente dirección: http://www.microsoft.com/robotics/
9 Para más información visitar la siguiente dirección: http://playerstage.sourceforge.net/

4 Gestión del proyecto

4.1 Estudio de viabilidad

Antes de poner en marcha el proyecto, se estuvo estudiando la viabilidad del mismo, comprobando la posibilidad de conectar un arduino por el puerto DB-25 del iRobot Create y alimentarlo directamente desde ese mismo puerto DB-25. Una vez comprobado que era posible, fue cuando se puso en marcha el proyecto.

4.2 Análisis de riesgos

- **Aumento de jornada en mi trabajo como vigilante de seguridad.** Este riesgo estaba presente y es el que mas a afectado al desarrollo del proyecto, ya que, a causa de la crisis, se ha despedido a mucho personal de la empresa y los que quedamos estamos teniendo que meter horas extra para cubrir el trabajo. Esto ha supuesto una duplicación en la cantidad de horas que trabajaba, pasando prácticamente a realizar mensualmente una jornada completa de 160 horas. Como consecuencia de ello, se ha menguado considerablemente el tiempo que podía disponer para realizar el proyecto.
- **Avería de alguna pieza del hardware.** Trabajando con hardware, es obvio que alguna pieza puede fallar, en este proyecto las piezas mas criticas eran la placa Arduino con la que se ha realizado el proyecto, junto con el módulo bluetooth. Para poder apaliar el efecto de los riesgos, se ha solicitado a la universidad 2 placas más con sus respectivos módulos bluetooth.
- **Perdida de la información.** La perdida de información es algo muy a tener en cuenta y es por ello que se le ha dedicado una fase exclusiva en el desarrollo del proyecto, para evitar en lo máximo posible que esto pueda llegar a suceder. Para ver las medidas tomadas dirigirse al apartado 4.3 Fases del proyecto y más concrétamente a la fase número 2.
- **Posibles enfermedades.** Cualquier persona puede ponerse enfermo y no hay forma de saber cuando, lo único que se puede hacer es cuidarse lo mas posible para evitar este tipo de imprevistos.
- **Problemas con el software.** A la hora de utilizar hardware nuevo, aparece software nuevo, y como toda cosa nueva, nunca se sabe como puede funcionar. En este caso, la librería softwareSerial de Arduino esta en fase beta y como toda beta tiene sus problemas. Para evitar el uso de esta librería, se a propuesto el utilizar la versión Mega de la placa Arduino.

4.3 Fases del proyecto

Debido a la ausencia de placas arduino en el departamento, se ha utilizado una placa y un módulo bluetooth que tenia yo en casa. Las etapas del proyecto fueron las siguientes y se realizaron en el orden en que se presentan:

1. **Análisis del proyecto.** En esta etapa se creó el esquema de la aplicación, buscando un esquema lo mas ajustado a nuestras necesidades.
2. **Preparación de herramientas de control de versiones y de copias de seguridad.** En este apartado se creó un servidor de subversion[10], protegido por contraseña, para poder trabajar desde distintas ubicaciones y en distintos ordenadores sin tener problemas de potabilidad del código. También se configuró un sistema de copias de seguridad del servidor de subversion con Cobian Backup[11], programándolo para que se realizara 1 backup completo semanal, 1 backup incremental diario y para que guardara con una trazabilidad de 2 semanas, las copias de seguridad en mi cuenta particular de dropbox.
3. **Implementación del tipo de datos instruction.** En esta fase se implemento el tipo de datos instruction que sería el que se usaría para almacenar las instrucciones generadas por los generadores.
4. **Construcción del juego de instrucciones de iRobot.** En esta fase se definieron los namespaces que almacenarían la información de las tablas y se crearon las funciones que usando esos namespaces generarían las instrucciones necesarias.
5. **Búsqueda del simulador.** En esta fase se dedicó tiempo a la búsqueda y estudio de los distintos simuladores que hay para iRobot Create, terminando la fase con la selección de uno.
6. **Generación de la conexión tcp/ip y de la conexión por puerto serie.**
7. **Generación del puente de conexiones y configuración de las conexiones tcp/ip del servidor y BAM del irobot.** Usando los manuales tanto de iRobot Create como del módulo BAM y el módulo bluetooth, se obtuvieron los parámetros necesarios para configurar todas y cada una de las conexiones.
8. **Verificación de las instrucciones del juego de instrucciones de iRobot con la documentación.** En esta fase se descubrieron los problemas de usar el tipo de datos char * en una instrucción y fue donde se implemento la función que sobrescribiría el operador de asignación del tipo de datos Instruction.
9. **Pruebas con el servidor y generación de demos.** En este apartado se construyeron las funciones que descomponen un número en bytes y viceversa.

10 Para saber mas sobre subversion visitar: http://subversion.apache.org/
11 Para saber mas sobre Cobian Backup visitar: http://www.cobiansoft.com/index.htm

10. **Pruebas de conexión con arduino.** En esta fase se construyó la primera versión del programa que interpretaría el juego de instrucciones de arduino. Inicialmente este programa lo único que hacia era leer información de un puerto y enviarla por otro, y viceversa. En esta fase es donde se descubrieron las limitaciones de arduino a la hora de emular conexiones serie con sus pines digitales, las cuales impedían usar baudrates muy altos. Se comprobó que con baudrates elevados la librería que emulaba los puertos serie fallaba provocando corrupción en los datos.
11. **Pruebas con el robot físicamente.** En esta fase se comprobó que la configuración con el dispositivo BAM estaba bien conectada, se probaron instrucciones de control e instrucciones de lectura de sensores. Se realizaron las primeras pruebas de control por medio del arduino encontrando problemas a la hora de recibir información del robot.
12. **Generación del juego de instrucciones de Arduino.** En esta fase se construyó el juego de instrucciones que interpretaría la placa Arduino.
13. **Verificación del juego de instrucciones de Arduino.** En esta fase se creó la versión final del programa que correría en la placa Arduino UNO y se comprobó que se podían configurar pines digitales pasar información por ellos. Las instrucciones que corresponden a los pines analógicos, no se pudieron ni se han podido comprobar por falta de dispositivos que se conecten por ese puerto.
14. **Pruebas con el sistema completo.** En esta fase se realizaron pruebas con el sistema completo de hardware, es decir, conectando arduino al iRobot, siendo incapaces de conseguir realizar lecturas de los sensores del robot a causa de la corrupción en la emulación del puerto serie por parte de la placa Arduino.
15. **Creación de la guía de usuarios del proyecto**. En esta fase se creó la guía de usuarios que se puede encontrar en el ANEXO III
16. **Documentación del proyecto.**

5 Estructura de la aplicación

5.1 Introducción

En esta sección se va a tratar de explicar lo mas detalladamente posible el funcionamiento general de la estructura que se ha generado. La idea era tener una estructura lo mas modulada e independiente posible, de forma que se pudiera reutilizar la mayor cantidad de código posible. Para ello, se han tomado los siguientes supuestos:

- Se ha supuesto que en un futuro el robot puede ser sustituido por otro, por lo tanto se ha querido hacer tanto la estructura hardware como la estructura software lo mas independiente posible del robot.

- También se ha supuesto que en un futuro pueden aparecer otros modos de comunicación, por ello se ha querido separar la parte del robot de la parte de comunicación dentro del software de la aplicación.

Teniendo en cuenta estos dos supuestos, se ha creado una aplicación a 3 capas:

- **<u>En la capa de nivel mas bajo tenemos:</u>**
  - **Juegos de instrucciones:**

    Estos módulos, única y exclusivamente, van a generar las instrucciones que el hardware[12] va interpretar.

    Se ha realizado la separación entre ellos de esta manera, con intención de que cada pieza de hardware tenga su propio módulo de software de forma que, si esa pieza se cambia, tan solo haya que sustituir su juego de instrucciones por el de la nueva pieza.
  - **Conexiones:**

    Estos módulos son los encargados de gestionar la comunicación, ofrecen soporte para configurar la conexión entre nuestro programa y el hardware, pero no la configuran. Esta decisión se ha tomado pensando en que no haya que tocar ninguno de estos módulos aunque cambie el hardware, ya que... si no se hiciese así, en el momento en que se cambiase el hardware, habría que venir a estos módulos a modificarlos, y si se quisiera añadir un nuevo hardware, habría que añadirle el soporte en estos módulos, quedando al final las configuraciones muy esparcidas por los distintos archivos.
- **<u>En la capa intermedia tenemos:</u>**
  - **Puente de conexiones:**

    En este nivel es donde se configuran los tipos de conexión para cada dispositivo.

12 En nuestro caso el hardware va a ser, por un lado la placa arduino y por otro el robot iRobot Create.

- **<u>En la capa superior tenemos:</u>**
    - **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.

De forma que:

Si cambiáramos de robot, habría que generar un nuevo juego de instrucciones para el robot, configurar la conexión en el puente de conexiones y darle soporte al nuevo juego de instrucciones en la interfaz.

Si cambiásemos el módulo bluetooth que une la placa arduino con el robot, habría simplemente que configurar la conexión en el puente de conexiones para que de soporte al nuevo módulo y cambiar la configuración del programa que corre la placa Arduino.

Si cambiásemos de placa, habría que añadir el programa que da soporte al juego de instrucciones de arduino a la nueva placa para que pueda interpretarlo.

Si quisiéramos implementar una función que cambié el estado de un led de la placa Arduino, habría que implementarla en la interfaz usando el juego de instrucciones generado para tal propósito.

Si quisiéramos añadir una nueva instrucción de configuración de la placa arduino que no pueda realizarse por medio de ninguna de las instrucciones que hay actualmente en ella (véase por ejemplo cambiar el rango de voltaje de un pin), entonces habría que añadir el nuevo código de instrucción al juego de instrucciones de Arduino y añadir el soporte para esa nueva instrucción al programa que corre en la placa Arduino.

5.2 Esquema de los módulos

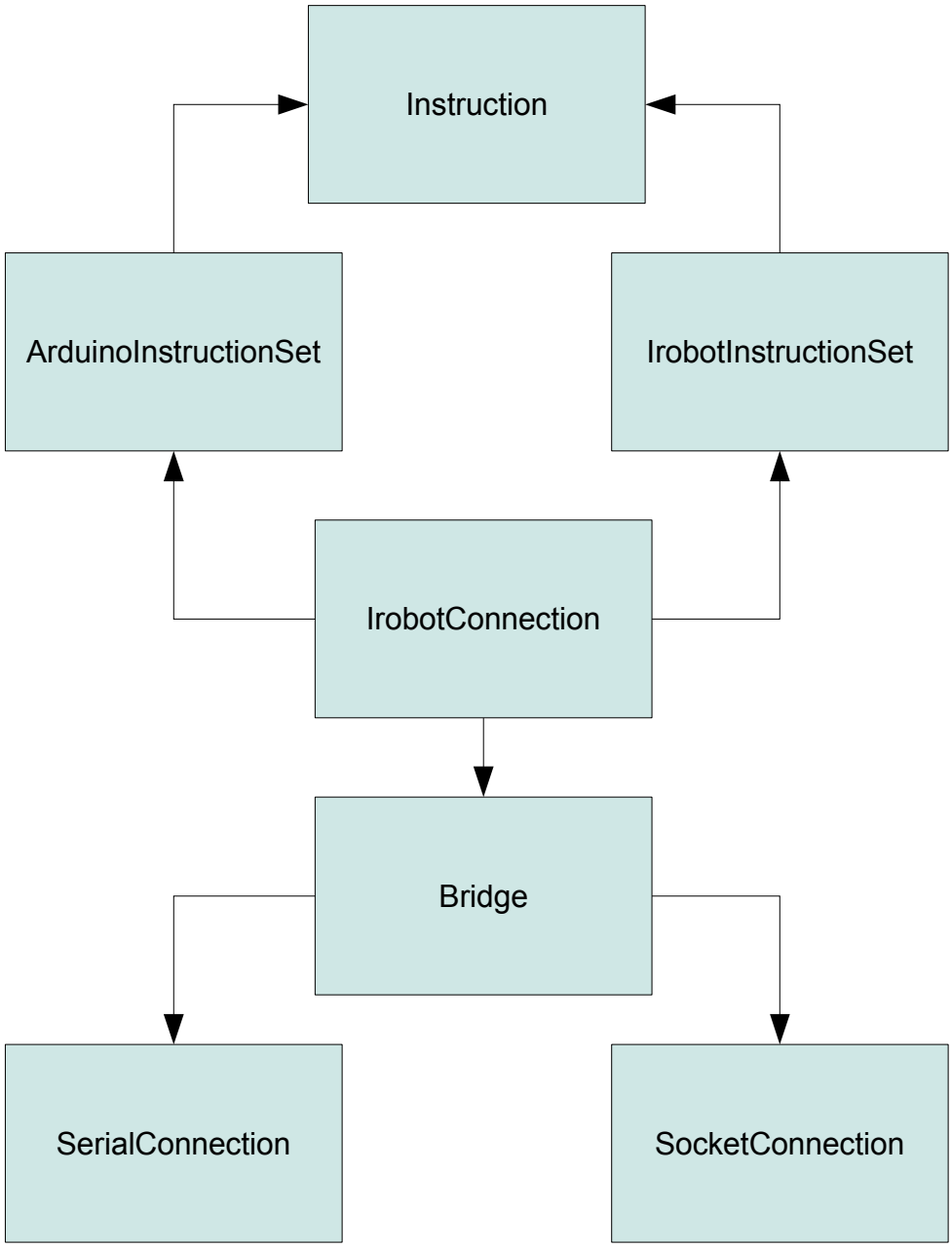


Imagen 5.1: Esquema de la aplicación

5.3 Descripción de los módulos

5.3.1 Instruction

5.3.1.1 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.

5.3.1.2 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[13], se ha sobre-escrito el operador de asignación con uno que realice copias de valores del char * gestionando el uso de la memoria.

```
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;

}
```

5.3.1.3 Ampliabilidad

A la hora de añadir mas opciones a este módulo tendríamos que modificar el operador de asignación para que incluya los nuevos campos del módulo.

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 linea:

hora = aux.hora;

13 Si dos punteros apuntan a una misma dirección de memoria a la hora de modificar el valor con uno de ellos automáticamente queda modificado el del otro cosa que no deseamos que suceda.

5.3.2 ArduinoInstructionSet

5.3.2.1 Descripción

Este módulo es el encargado de generar las instrucciones de Arduino que nos permitirán, tanto configurar los pines de la placa Arduino, como leer o enviar información por alguno de los pines.

5.3.2.2 Implementación

Para facilitar en el futuro el añadir nuevas instrucciones a este generador, se ha creado el siguiente namespace[14]:

```
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.

14 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(*standar*) 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: *standar::instruccion* o *personalizada::instruccion* dependiendo de cual de las dos queramos utilizar.

5.3.2.3 Ampliabilidad

El motivo por el cual se han creado estos namespaces es, primero para facilitar la ampliabilidad del módulo, de forma que modificando alguno de los dos se podrian cambiar los bytes que corresponden a cada instrucción sin necesidad de modificar el cuerpo del mismo, y por otro el 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 mas al namespace ArduinoInstruction:

static const char ANALOG_VOLTAGE_RANGE = 105;

5.3.3 IrobotInstructionSet

5.3.3.1 Descripción

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

5.3.3.2 Implementación

Para mayor facilidad a la hora de modificar este juego de instrucciones en la cabecera de este módulo se han creado los siguientes namespace:

- Un namespace encargado de almacenar los bytes correspondientes a cada instrucion en una variable que nos evite el tener que acudir a la tabla de instrucciones a la hora de querer usarla. Por otro lado nos permitirá modificar el juego de instrucciones de manera mucho mas 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;
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 CLIFFRIGHTTSIGNAL = (  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 podemos ver 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 adaptar 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_BATERY = (  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;
}
```

5.3.3.3 Ampliabilidad

A la hora de ampliar este módulo tan solo 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 instrucciones al robot con el código 159 con 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;
```

Aparte habría que implementar la siguiente instrucción:

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

5.3.4 SerialConnection

5.3.4.1 Descripción

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

5.3.4.2 Implementacion

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 configuracion 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 estan 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_Hands_Haking(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();`
  Vacia el buffer de entrada
- `int Setup_Buffer(unsigned long InQueue,unsigned long OutQueue);`
  Establece la longitud del buffer a utilizar.

5.3.4.3 Ampliabilidad

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.

5.3.5 SocketConnection

5.3.5.1 Descripción

Este módulo nos gestiona una comunicación tpc/ip, la cual, en nuestro caso, es necesaria para poder comunicarnos con el servidor del simulador.

5.3.5.2 Implementación

A la hora de establecer la conexión, la función encargada de ello recibe una variable de tipo SOCKADDR_IN previamente configurada con la dirección ip del servidor al que nos vamos a conectar, el puerto en el que trabaja y el tipo de socket que queremos utilizar.

Estas son las funciones que podemos encontrar en su cuerpo:

```
void connect(SOCKADDR_IN data);
int send(char * buffer, int size);
int receive(char * buffer, int size);
void closeConnection();
```

Ejemplo: Supongamos que queremos conectarnos a un servidor en la dirección **ip**: 212.12.32.42, que se comunica por el **puerto**: 30001.

Primero deberiamos inicializar nuestra variable SOCKADDR_IN config; de la siguiente manera:

memset (&config, 0, sizeof(config));

config.sin_family = AF_INET;

config.sin_port = htons(30001);

config.sin_addr.s_addr = "212.12.32.42";

Una vez inicializada la variable, para establecer la conexión, tan solo tendríamos realizar la siguiente llamada:

connect(config);

5.3.5.3 Ampliabilidad

Este módulo no requiere de ningún tipo de ampliación ya que contiene todas las funciones necesarias para configurar cualquier conexión a un servidor tcp/ip.

5.3.6 Bridge

5.3.6.1 Descripción

Este módulo es el encargado de fusionar los dos tipos de conexiones, almacena la configuración necesaria para establecer tanto la conexión con el simulador, como la conexión con el Arduino y la conexión con el módulo BAM para iRobot.

5.3.6.2 Implementación

El constructor de la clase será el que decida que tipo de conexión se va a utilizar de forma que si no se le pasa ningún parámetro o si se le pasa el string “sim” como parámetro la clase trabajará con la conexión del simulador. Si por el contrario quisiéramos que trabajara con el robot es imprescindible que le indiquemos el puerto al que queremos que se conecte en el formato “COMx” donde x es el numero de puerto.

```
Bridge::Bridge( char * type )
{
	if(!(simulated = type == "sim"))
	{
		portName = new char[strlen("\\\\.\\")+strlen(type)];
		strcpy(portName,"\\\\.\\");
		strcat(portName, type);
	}
}
```

Debido a que Windows solo reconoce los puertos [COM1,COM9] de manera automática, se ha convertido el nombre del puerto al estándar que vendría a ser \\.\COMx

En el código anterior se puede apreciar como el tipo de conexión se almacena en una variable booleana llamada simulated y en caso de que sea un puerto serie, el nombre del puerto se almacena en una variable char* llamada portName.

La función de conexión por puerto serie, nos permite introducir un parámetro booleano (por defecto false) que indica si queremos utilizar una conexión con el dispositivo BAM[15] o queremos utilizar una conexión con Arduino, esto se ha realizado de esta manera para permitir que se pueda seguir utilizando el dispositivo BAM.

15 Para mas información acerca de este dispositivo visitar la siguiente dirección: http://www.elementdirect.com/files/10542B.pdf

La función de conexión es la siguiente:

```
void Bridge::connect(bool useArduino)
{
	if(simulated)
	{
		#pragma region Connect to the simulator
		//INITIALIZE TCP/IP SIMULATION
		memset(&data, 0,sizeof(data));
		data.sin_family = AF_INET;
		data.sin_port = htons(port);
		data.sin_addr.s_addr = inet_addr(ip());
		//CONNECT TO THE SIMULATOR SERVER
		socket.connect(data);
		#pragma endregion
	}
	else
	{
		//INITIALIZE ARDUINO CONFIGURATION
		arduinoConfig.BaudRate = 9600;
		arduinoConfig.ByteSize = 8;
		arduinoConfig.Parity = NOPARITY;
		arduinoConfig.StopBits = ONESTOPBIT;
		//INITIALIZE IROBOT CONFIGURATION
		iRobotConfig.BaudRate = 57600;
		iRobotConfig.ByteSize = 8;
		iRobotConfig.Parity = NOPARITY;
		iRobotConfig.StopBits = ONESTOPBIT;
		//CHOOSE WITCH CONFIGURATION WILL BE USED
		if(useArduino) currentConfig == &arduinoConfig;
		else currentConfig = &iRobotConfig;
		//OPEN THE PORT
		serial.Open_Port(portName);
		serial.Set_Configure_Port(*currentConfig);
		serial.Set_Hands_Haking(NONE);
	}
}
```

A la hora de enviar comandos, se ha establecido que antes de enviar un comando por puerto serie, se limpiara el buffer del mismo para asegurarnos de que la única información que quedará en el buffer a la hora de leer, será la respuesta a ese comando.

5.3.6.3 Ampliabilidad

A la hora de querer utilizar una nueva conexión, habría que añadir e inicializar las variables de configuración necesarias para esta nueva conexión y modificar la función connect del módulo Bridge.

Ejemplo: Supongamos que queremos añadir una nueva conexión tcp/ip

Para darle soporte a esta conexión tendríamos que hacer los siguientes cambios:

- Añadir una variable que nos indique que queremos usar la nueva conexión:

```
bool usarNueva;
```

- Añadir una nueva variable de tipo SOCKADDR_IN y configurarla con la ip y el puerto del servidor.

```
SOCKADDR_IN nuevaConexion;
```

- La constructora la modificaríamos de la siguiente manera:

```
Bridge::Bridge( char * type )
{
	if (type == “nueva”) usarNueva = true;
	else
	if(!(simulated = type == "sim"))
	{
		portName = new char[strlen("\\\\.\\")+strlen(type)];
		strcpy(portName,"\\\\.\\");
		strcat(portName, type);
	}
}
```

- A la funcion connect añadiriamos la siguiente linea:

```
void Bridge::connect(bool useArduino)
{
if(simulated)
{
	#pragma region Connect to the simulator
	//INITIALIZE TCP/IP SIMULATION
	memset(&data, 0,sizeof(data));
	data.sin_family = AF_INET;
	data.sin_port = htons(65000);
	data.sin_addr.s_addr = inet_addr("127.0.0.1");
	//CONNECT TO THE SIMULATOR SERVER
	socket.connect(data);
	#pragma endregion
#pragma endregion
}
else
if(usarNueva)
{
	memset(&data, 0,sizeof(data));
	data.sin_family = AF_INET;
	data.sin_port = 43922;
	data.sin_addr.s_addr = "190.108.20.45";
	socket.connect(data);
}
else
{
	//INITIALIZE ARDUINO CONFIGURATION
	arduinoConfig.BaudRate = 9600;
	arduinoConfig.ByteSize = 8;
	arduinoConfig.Parity = NOPARITY;
	arduinoConfig.StopBits = ONESTOPBIT;
	//INITIALIZE IROBOT CONFIGURATION
	iRobotConfig.BaudRate = 57600;
	iRobotConfig.ByteSize = 8;
	iRobotConfig.Parity = NOPARITY;
	iRobotConfig.StopBits = ONESTOPBIT;
	//CHOOSE WITCH CONFIGURATION WILL BE USED
	if(useArduino) currentConfig == &arduinoConfig;
	else currentConfig = &iRobotConfig;
	//OPEN THE PORT
	serial.Open_Port(portName);
	serial.Set_Configure_Port(*currentConfig);
	serial.Set_Hands_Haking(NONE);
}
}
```

5.3.7 IrobotConnection

5.3.7.1 Descripción

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

5.3.7.2 Implementación

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

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


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

A la hora de establecer la conexión se le pasa el parámetro que por defecto esta inicializado a false y que indica si, en caso de utilizar la conexión de puerto serie, queremos que se use la configuración de Arduino o la configuración del dispositivo BAM.

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

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

5.3.7.3 Ampliabilidad

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 esta 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);
}
```

5.3.8 Modulo arduino

5.3.8.1 Descripción

Este módulo es un módulo escrito en proccesing y estará funcionando constantemente en la placa arduino. Este módulo se va a encargar por un lado de interpretar las funciones que genere el módulo IrobotInstructionSet, por otro lado, de pasar las instrucciones que correspondan al robot y por ultimo, de pasar la información del robot al ordenador.

5.3.8.2 Implementación

El programa en pseudo-código seria el siguiente:

```
while (1)
{
si hay informacion del robot
	{
		envia informacion al ordenador
	}

si hay información del ordenador
	{
		si es una instrucción para arduino
		{
			tratar instrucción
		}
		sino
		{
			envía instrucción al robot
		}
	}
}
```

5.3.8.3 Ampliabilidad

A la hora de realizar nuevas instrucciones habría que añadir lo pertinente en el apartado de tratar instrucciones.

6 Conclusiones y trabajo futuro

Por lo general los juegos de instrucciones que ofrecen los fabricantes de hardware no suelen ser muy amigables, requieren tanto de tablas como las de las figuras de la 3.1 a la 3.6 como de paso de parámetros poco naturales, para poder programarlos.

Véase por ejemplo la siguiente instrucción: 145 1 0 0 100. Esta instrucción cambia la velocidad de los dos motores del robot poniendo a 256mm/s el motor derecho y a 100mm/s el motor izquierdo. Como bien he dicho antes, para entender esta función es necesario acudir a la tabla y realizar los cálculos que convierten 2 bytes en un numero entero.

Viendo el ejemplo, podemos darnos cuenta de que este lenguaje ocasiona un descenso importante en la velocidad de desarrollo y aumenta considerablemente el tiempo de aprendizaje para los desarrolladores. Otro pero que tiene es el siguiente: Una instrucción suelta puede molestar, pero ¿qué pasaría a la hora de intentar encontrar un error de ejecución usando este tipo de instrucciones? Evidentemente encontrar ese error llevaría unas cantidades de tiempo impresionante.

Como bien es sabido, en docencia el tiempo es muy limitado y es por ello que surgen las capas intermedias, que nos permiten olvidarnos de las tablas ocultando los códigos y mostrándonos solo su traducción en la tabla.

Cojamos por ejemplo la instrucción anterior: 145 1 0 0 100. Usando la capa intermedia que se ha creado, nuestro código se convertiría en lo siguiente: robot.driveDirect(256,100);

Eliminando por completo la necesidad de acudir a una tabla para saber a que instrucción correspondía el código 145 y evitando, en lo que a las velocidades corresponden, tener que calcular que números son los que representan cada par de bytes. Como se puede apreciar, la capacidad de comprensión del código y de escritura del código es mas sencilla.

Las ampliabilidad tanto del hardware como del software esta pensada para que los cambios en una sección afecten lo menos posible a las otras, consiguiéndose así un mayor aprovechamiento tanto del código generado como de las piezas.

Por otro lado, el lenguaje de programación de la capa intermedia (C++) abre una puerta a un mundo de librerías que ahorrarán mucho trabajo a los desarrolladores a la hora de realizar aplicaciones complejas que involucren mas elementos, como bien podría ser el kinect que últimamente esta tan de moda.

En lo que a las posibilidades que ofrece este proyecto, debemos decir que aunque el proyecto esta centrado en el robot iRobot Create, se ha mantenido la suficiente disociación como para que la estructura hardware de arduino, pueda utilizarse para ampliar de manera sencilla cualquier sistema que posea una conexión por puerto serie. Esto quiere decir que este proyecto podría utilizarse, por ejemplo, para ampliar también el brazo robótico del laboratorio de robótica de la facultad.

7 Bibliografía

- *Manual de usuario del robot iRobot Create*
  http://www.irobot.com/filelibrary/pdfs/hrd/create/Create%20Manual_Final.pdf
- *Tutoriales de arduino*
  http://arduino.cc/es/Tutorial/HomePage
- *Página del simulador de iRobot Create*
  http://www.rose-hulman.edu/~boutell/simulator/index.html
- *Documentación winsock2 (la librería de sockets de Windows)*
  http://msdn.microsoft.com/en-us/library/windows/desktop/ms741416(v=vs.85).aspx
- *Guía de comunicación por puerto serie de Microsoft*
  http://msdn.microsoft.com/en-us/library/ff802693.aspx
- *Guía de comunicación en C++ por puerto serie con Arduino*
  http://arduino.cc/playground/Interfacing/CPPWindows
- *Guía del módulo bluetooth*
  http://tallerarduino.wordpress.com/2011/12/06/modulo-bluetooth-hc-06-o-gp-gc021-y-arduino/
- *Guía de configuración de un servidor subversion en windows*
  http://www.codinghorror.com/blog/2008/04/setting-up-subversion-on-windows.html
- *Página oficial de Cobian Backup*
  http://www.cobiansoft.com/index.htm

Anexo I (Código fuente)

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);
};
```

Instruction.cpp

```
#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;
}
```

IrobotInstructionSet.h

```
#pragma once
#include "Instruction.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 CLIFFRIGHTTSIGNAL = (	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_BATERY = (  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 generates
the sensor 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);
};
```

IrobotInstructionSet.cpp

```
#include "iRobotInstructionSet.h"
#include <malloc.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;
}
```

ArduinoInstructionSet.h

```
#pragma once
#include "Instruction.h"
//Opcodes recognized by Arduino board as instruction
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;
};

//Arduino board especial data
namespace ArduinoData
{
	static const char INPUT_MODE = 1;
	static const char OUTPUT_MODE = 0;
	static const char HIGH = 1;
	static const char LOW = 0;
};

class ArduinoInstructionSet
{

public:
	ArduinoInstructionSet(void);
	~ArduinoInstructionSet(void);


	//************************************
	// Method:    setPinMode
	// Returns:   Instruction                        [out]  Instruction that must be send
trough the Arduino connection
	// Parameter: int pin                                   [in]   Witch pin we want to set
	// Parameter: bool input                         [in]   True -> Set the pin to
INPUT_MODE (we want to use the pin to read from it)
	//************************************
	Instruction setPinMode(int pin, bool input);


	//************************************
	// Method:    digitalRead
```

```
      // Returns:   Instruction                        [out]  Instruction that must be send
through the Arduino connection
      // Parameter: int pin                                  [in]   Witch digital pin we
want to read from
      //************************************
      Instruction digitalRead(int pin);


      //************************************
      // Method:    digitalWrite
      // Returns:   Instruction                        [out]  Instruction that must be send
trough the Arduino connection
      // Parameter: int pin                                  [in]   Witch pin we want to
write to
      // Parameter: bool hight                         [in]   True -> Send digital 1 across
the pin
      //************************************
      Instruction digitalWrite(int pin, bool high);


      //************************************
      // Method:    analogRead
      // Returns:   Instruction                        [out]  Instruction that must be send
trough the Arduino connection
      // Parameter: int pin                                  [in]   Witch pin we want to
read from
      // Parameter: int numBytes                       [in]   How many bytes we want to
read, 1 or 2
      //************************************
      Instruction analogRead(int pin, bool oneByte);


      //************************************
      // Method:    analogWrite
      // Returns:   Instruction                        [out]  Instruction that must be send
trough the Arduino connection
      // Parameter: int pin                                  [in]   Witch pin we want to
write to
      // Parameter: char data                                [in]   Data to send trough the
pin
      //************************************
      Instruction analogWrite(int pin, char data);
};
```

ArduinoInstructionSet.cpp

```
#include "ArduinoInstructionSet.h"
```

```
ArduinoInstructionSet::ArduinoInstructionSet(void)
{
}

ArduinoInstructionSet::~ArduinoInstructionSet(void)
{
}

Instruction ArduinoInstructionSet::setPinMode( int pin, bool input )
{
	int miSize = 3;
	Instruction aux;
	aux.instruction = ( char * )malloc(miSize * sizeof( char ));
	aux.instruction[0] = ArduinoInstruction::SET_PIN_MODE;
	aux.instruction[1] = pin;
	aux.instruction[2] = input;
	aux.length = miSize;
	return aux;
}

Instruction ArduinoInstructionSet::digitalRead( int pin )
{
	int miSize = 2;
	Instruction aux;
	aux.instruction = ( char * )malloc(miSize * sizeof( char ));
	aux.instruction[0] = ArduinoInstruction::DIGITAL_READ;
	aux.instruction[1] = pin;
	aux.length = miSize;
	aux.response = 1;
	return aux;
}
```

```
Instruction ArduinoInstructionSet::digitalWrite( int pin, bool high )
{
	int miSize = 3;
	Instruction aux;
	aux.instruction = new char[miSize];
	aux.instruction[0] = ArduinoInstruction::DIGITAL_WRITE;
	aux.instruction[1] = pin;
	if (high)	aux.instruction[2] = ArduinoData::HIGH;
	else aux.instruction[2] = ArduinoData::LOW;
	aux.length = miSize;
	return aux;
}

Instruction ArduinoInstructionSet::analogRead( int pin, bool oneByte /*= 1*/ )
{
	int miSize = 2;
	Instruction aux;
	aux.instruction = ( char * )malloc(miSize * sizeof( char ));
	aux.instruction[0] = ArduinoInstruction::ANALOG_READ;
	aux.instruction[1] = pin;
	aux.length = miSize;
	if (oneByte) aux.response = 1;
	else aux.response = 2;
	return aux;
}

Instruction ArduinoInstructionSet::analogWrite( int pin, char data )
{
	int miSize = 3;
	Instruction aux;
	aux.instruction = new char[miSize];
	aux.instruction[0] = ArduinoInstruction::ANALOG_WRITE;
	aux.instruction[1] = pin;
	aux.instruction[2] = data;
	aux.length = miSize;
	return aux;
}
```

SerialConnection.h

```
#pragma once
#include <stdio.h>
#include <windows.h>
#define NONE      0
#define RTSCTS    1
#define HARD      1
#define XONXOFF   2
#define SOFT      2
class SerialConnection
{
private:
	HANDLE fd;
public:
	SerialConnection(void);
	~SerialConnection(void);

	//************************************
	// Method:    Open_Port
	// Parameter: char * COMx                        [in] Com port that we want to open
given in format "\\.\\COMx"
	//************************************
	void Open_Port(char * COMx);

	//************************************
	// Method:    Get_Configure_Port
	// Returns:   DCB                                [out] Current serial
connection configuration file
	//************************************
	DCB Get_Configure_Port();

	//************************************
	// Method:    Configure_Port
	// Returns:   DCB                                [out]  Generated DCB file with
desired configuration
	// Parameter: unsigned int BaudRate              [in]   Baudrate
	// Parameter: char CharParity[]                  [in]   Parity
	//************************************
	DCB Configure_Port(unsigned int BaudRate,char CharParity[]);

	//************************************
```

```
	// Method:    Set_Configure_Port
	// Returns:   int                                  [out]  Status of the operation
	// Parameter: DCB PortDCB                  [in]   Desired configuration
	//************************************
	int Set_Configure_Port(DCB PortDCB);

	//************************************
	// Method:    Write_Port
	// Returns:   long                                 [out]  How many bytes had been
written
	// Parameter: char Data[]                  [in]   Byte buffer
	// Parameter: int SizeData                 [in]   How many bytes we want to send
	//************************************
	long Write_Port(char Data[],int SizeData);

	//************************************
	// Method:    Read_Port
	// Returns:   long                                 [out]  How many bytes have been
read
	// Parameter: char * Data                  [out]  Bytes that have been read
	// Parameter: int SizeData                 [in]   How many bytes we want to read
	//************************************
	long Read_Port(char *Data,int SizeData);
	//************************************
	// Method:    Getc_Port
	// Returns:   long                                 [out]  How many bytes have been
read
	// Parameter: char * Data                  [in]   Bytes that have been read
	//************************************
	long Getc_Port(char *Data);

	//************************************
	// Method:    Kbhit_Port
	// Returns:   int                                  [out]  How many bytes are in
the incoming buffer
	//************************************
	int Kbhit_Port();

	//************************************
	// Method:    Close_Port
	// Returns:   int                                  [out]  Status of the operation
	//************************************
```

```
	int Close_Port();

	//***********************************
	// Method:    Set_Hands_Haking
	// Returns:   int                                    [out]  Status of the operation
	// Parameter: int FlowControl                  [in]   Handshaking id to be set in
the serial communication
	//***********************************
	int Set_Hands_Haking(int FlowControl);

	//***********************************
	// Method:    Set_BaudRate
	// Returns:   int                                    [out]  Status of the operation
	// Parameter: unsigned int BaudRate            [in]   Baud rate id to be set in the
serial communication
	//***********************************
	int Set_BaudRate(unsigned int BaudRate);

	//***********************************
	// Method:    Set_Time
	// Returns:   int                                    [out]  Status of the operation
	// Parameter: unsigned int Time                [in]   How much time in ms must be
wait before decide there has been a timeout
	//***********************************
	int Set_Time(unsigned int Time);

	//***********************************
	// Method:    Clean_Buffer
	// Returns:   int                                    [out]  Status of the operation
	// Description:                                                    Resets the
incoming buffer
	//***********************************
	int Clean_Buffer();

	//***********************************
	// Method:    Setup_Buffer
	// Returns:   int                                    [out]  Status of the operation
	// Parameter: unsigned long InQueue            [in]   Incoming buffer size
	// Parameter: unsigned long OutQueue     [in]   Outgoing buffer size
	//***********************************
	int Setup_Buffer(unsigned long InQueue,unsigned long OutQueue);
};
```

SerialConnection.cpp

```
#include "SerialConnection.h"

#define MAX_SIZE_BUFFER 8192

SerialConnection::SerialConnection(void)
{
	fd= INVALID_HANDLE_VALUE;
}



SerialConnection::~SerialConnection(void)
{
}

BOOL ERROR_CONFIGURE_PORT=FALSE;
void SerialConnection::Open_Port(char COMx[])
{
	HANDLE miFd;
	miFd = CreateFileA( COMx,                   // pointer to name of the file
		GENERIC_READ | GENERIC_WRITE, // access (read-write) mode
		0,                            // share mode
		NULL,                  // pointer to security attributes
		OPEN_EXISTING,                // how to create
		0,                            // file attributes
		NULL);                 // handle to file with attributes
	// to copy

	if (miFd == INVALID_HANDLE_VALUE)
	{
		printf("Error:No se puede abrir puerto: %s \n",COMx);
		system("PAUSE");
		exit(1);
	}
	fd = miFd;
}
```

```
DCB SerialConnection::Get_Configure_Port()
{
	DCB PortDCB;
	PortDCB.DCBlength = sizeof (DCB);

	if(!GetCommState (fd, &PortDCB))
	{
		printf("Error pidiendo la configuración de puerto serie.\n");
		ERROR_CONFIGURE_PORT=TRUE;
		return PortDCB;
	}
	ERROR_CONFIGURE_PORT=FALSE;
	return PortDCB;
}

DCB SerialConnection::Configure_Port(unsigned int BaudRate,char CharParity[])
{
	DCB PortDCB;
	PortDCB.DCBlength = sizeof (DCB);

	// Configuramos el tamaño del buffer de escritura/lectura
	if(!SetupComm(fd, MAX_SIZE_BUFFER, MAX_SIZE_BUFFER))
	{
		printf("Error configurando buffer\n");
		ERROR_CONFIGURE_PORT=TRUE;
		return PortDCB;
	}

	if(!GetCommState (fd, &PortDCB))
	{
		printf("Error Pidiendo configuración de Puerto\n");
		ERROR_CONFIGURE_PORT=TRUE;
		return PortDCB;
	}

	// Change the DCB structure settings.
	PortDCB.BaudRate = BaudRate;		// Current baud
	PortDCB.fBinary = TRUE;			// Binary mode; no EOF check
	//PortDCB.EofChar = 0;
	PortDCB.fErrorChar = FALSE;		// Disable error replacement.
```

```
	PortDCB.fNull = FALSE;                  // Disable null stripping.
	PortDCB.fAbortOnError = FALSE;          // Do not abort reads/writes on error.
	PortDCB.fParity = FALSE;                // Disable parity checking.

	PortDCB.fRtsControl = RTS_CONTROL_DISABLE; // RTS flow control

	PortDCB.fDtrControl = DTR_CONTROL_DISABLE; // DTR flow control type

	PortDCB.fOutxCtsFlow = FALSE;           // No CTS output flow control
	PortDCB.fOutxDsrFlow = FALSE;           // No DSR output flow control

	PortDCB.fDsrSensitivity = FALSE;        // DSR sensitivity

	PortDCB.fOutX = FALSE;                  // No XON/XOFF out flow control
	PortDCB.fInX = FALSE;                   // No XON/XOFF in flow control
	PortDCB.fTXContinueOnXoff = TRUE;       // XOFF continues Tx

	if(strncmp(CharParity,"8N1",3)==0)
	{
		PortDCB.ByteSize = 8;                   // Number of bits/bytes, 4-8
		PortDCB.Parity = NOPARITY;              // 0-4=no,odd,even,mark,space
		PortDCB.StopBits = ONESTOPBIT;          // 0,1,2 = 1, 1.5, 2
	}
	if(strncmp(CharParity,"7E1",3)==0)
	{
		PortDCB.ByteSize = 7;                   // Number of bits/bytes, 4-8
		PortDCB.Parity = EVENPARITY;            // 0-4=no,odd,even,mark,space
		PortDCB.StopBits = ONESTOPBIT;          // 0,1,2 = 1, 1.5, 2
	}
	if(strncmp(CharParity,"7O1",3)==0)
	{
		PortDCB.ByteSize = 7;                   // Number of bits/bytes, 4-8
		PortDCB.Parity = ODDPARITY;             // 0-4=no,odd,even,mark,space
		PortDCB.StopBits = ONESTOPBIT;          // 0,1,2 = 1, 1.5, 2
	}
	if(strncmp(CharParity,"7S1",3)==0)
	{
		PortDCB.ByteSize = 7;                   // Number of bits/bytes, 4-8
		PortDCB.Parity = SPACEPARITY;           // 0-4=no,odd,even,mark,space
		PortDCB.StopBits = ONESTOPBIT;          // 0,1,2 = 1, 1.5, 2
```

```
	}

	if (!SetCommState (fd, &PortDCB))
	{  // Could not configure the serial port.
		printf("Error: configurando puerto\n");
		ERROR_CONFIGURE_PORT=TRUE;
		return PortDCB;
	}

	// Configure timeouts
	COMMTIMEOUTS timeouts;
	// No timeouts
	timeouts.ReadIntervalTimeout = 0;
	timeouts.ReadTotalTimeoutMultiplier = 0;
	timeouts.ReadTotalTimeoutConstant = 0;
	timeouts.WriteTotalTimeoutMultiplier = 0;
	timeouts.WriteTotalTimeoutConstant = 0;

	if (!SetCommTimeouts(fd, &timeouts))
	{
		printf("ERROR: No se pudo poner SetCommTimeouts: %s\n",
			GetLastError());
		ERROR_CONFIGURE_PORT=TRUE;
		return PortDCB;
	}


	ERROR_CONFIGURE_PORT=FALSE;
	return PortDCB;
}

int SerialConnection::Set_Configure_Port(DCB PortDCB)
{
	// Ahora limpiamos el buffer de entrada y salida del puerto
	// y activamos la configuración del puerto.
	if (!SetCommState (fd, &PortDCB))
	{
		printf("ERROR: No se pudo poner configuración del puerto serie\n" );
		ERROR_CONFIGURE_PORT=TRUE;
		return FALSE;
```

```
	}
	ERROR_CONFIGURE_PORT=FALSE;

	return TRUE;
}

long SerialConnection::Write_Port(char Data[],int SizeData)
{
	long n;

	WriteFile(fd,			// Port handle
		Data,			// Pointer to the data to write
		(DWORD)SizeData,	// Number of bytes to write
		(DWORD*)&n,		// Pointer to the number of bytes written
		NULL);			// Must be NULL for Windows CE
	return n;
}

long SerialConnection::Read_Port(char *Data,int SizeData)
{
	long n;

	ReadFile (fd,			// Port handle
		Data,			// Pointer to the data to write
		(DWORD)SizeData,	// Number of bytes to write
		(DWORD*)(&n),		// Pointer to the number of bytes read
		NULL);			// Must be NULL for Windows CE
	return n;
}

long SerialConnection::Getc_Port(char *Data)
{
	long n;

	ReadFile(fd,Data,(DWORD)1,(DWORD*)(&n),NULL);

	return n;
}
```

```
int SerialConnection::Kbhit_Port()
{
	DWORD x;
	COMSTAT cs;
	// Actualizar COMSTAT, sirve para averiguar el número de bytes en el
	// buffer de entrada:
	ClearCommError(fd, &x, &cs);
	return cs.cbInQue;
}

int SerialConnection::Close_Port()
{
	if (fd != INVALID_HANDLE_VALUE)
	{  // Close the communication port.

		if (!CloseHandle (fd))
		{printf("Error cerrando el puerto serie\n");return FALSE;}
		else
		{fd = INVALID_HANDLE_VALUE;return TRUE;}
	}
	return FALSE;
}

int SerialConnection::Set_Hands_Haking(int FlowControl)
{
	DCB PortDCB;
	if(!GetCommState (fd, &PortDCB))
	{
		printf("Error Pidiendo configuración de puerto serie\n");
		ERROR_CONFIGURE_PORT=TRUE;
		return FALSE;
	}

	switch(FlowControl)
	{
	case 0: ///NONE
		{
			PortDCB.fOutX = FALSE;                      // No XON/XOFF out flow
control.
			PortDCB.fInX = FALSE;                       // No XON/XOFF in flow
control.
```

```
                PortDCB.fRtsControl = RTS_CONTROL_ENABLE;    // RTS flow control.

                PortDCB.fDtrControl = DTR_CONTROL_ENABLE;    // DTR flow control type.
                break;
            }
        case 1: ///RTS/CTS
            {
                PortDCB.fRtsControl = RTS_CONTROL_HANDSHAKE;// RTS flow control.
                break;
            }
        case 2: ///XON/OFF
            {
                PortDCB.fOutX = TRUE;                        // XON/XOFF out flow
control.
                PortDCB.fInX = TRUE;                         // XON/XOFF in flow
control.
                PortDCB.XonChar=0x11;                        // ASCII_XON.
                PortDCB.XoffChar=0x13;                       // ASCII_XOFF.
                PortDCB.XonLim=100;
                PortDCB.XoffLim=100;
                break;
            }
        case 3: ///DTR/DSR
            {
                PortDCB.fDtrControl = DTR_CONTROL_HANDSHAKE;// DTR flow control type.
                break;
            }
        }

        if (!SetCommState (fd, &PortDCB))
        {
            printf("ERROR: Configurando el puerto serie\n");
            ERROR_CONFIGURE_PORT=TRUE;
            return FALSE;
        }
        ERROR_CONFIGURE_PORT=FALSE;

        return TRUE;
}
```

```
int SerialConnection::Set_BaudRate(unsigned int BaudRate)
{
	DCB PortDCB;

	if(!GetCommState (fd, &PortDCB))
	{
		printf("Error Pidiendo configuración del Puerto\n");
		ERROR_CONFIGURE_PORT=TRUE;
		return FALSE;
	}

	PortDCB.BaudRate = BaudRate;		// Binary mode; no EOF check

	if (!SetCommState (fd, &PortDCB))
	{
		printf("Error configurando el BaudRate\n");
		ERROR_CONFIGURE_PORT=TRUE;
		return FALSE;
	}
	ERROR_CONFIGURE_PORT=FALSE;

	return TRUE;
}

int SerialConnection::Set_Time(unsigned int Time)
{
	COMMTIMEOUTS CommTimeouts;

	if(!GetCommTimeouts (fd, &CommTimeouts))
	{
		printf("Error obteniendo configuración time-out actual: %s\n",
			GetLastError());
		return FALSE;
	}

	// Tiempo maximo en mseg. entre caracteres consecutivos
	CommTimeouts.ReadIntervalTimeout = Time*200;

	// Time-Out=TotalTimeoutMultiplier*number_of_bytes+TotalTimeoutConstant
```

```
	// Especifique el multiplicador de tiempo fuera de lectura con el miembro
	// ReadTotalTimeoutMultiplier. En cada operación de lectura , este número
	// se multiplica por el número de bytes que la lectura espera recibir .
	CommTimeouts.ReadTotalTimeoutMultiplier = Time*100;
	// Constante a sumar al time-out total de recepción.
	CommTimeouts.ReadTotalTimeoutConstant = 0;

	// Igual que lectura.
	CommTimeouts.WriteTotalTimeoutMultiplier = Time*100;
	// Igual que lectura
	CommTimeouts.WriteTotalTimeoutConstant = 0;

	// Establecemos nuevos valores de time-out.
	if(!SetCommTimeouts (fd, &CommTimeouts))
	{
		printf("Error estableciendo nueva configuración time-out: %s\n",
			GetLastError());
		return FALSE;
	}

	return TRUE;
}

int SerialConnection::Clean_Buffer()
{
	return PurgeComm( fd , PURGE_TXABORT | PURGE_RXABORT | PURGE_TXCLEAR |
PURGE_RXCLEAR );
}

int SerialConnection::Setup_Buffer(unsigned long InQueue,unsigned long OutQueue)
{
	return SetupComm(fd,InQueue,OutQueue);
}
```

Bridge.h

```
#pragma once
#include <WinSock2.h>
#include "SocketConnection.h"
#include "SerialConnection.h"
#pragma comment(lib, "ws2_32.lib")

class Bridge
{
private:
	bool simulated; //true -> we are using the simulator
#pragma region Simulator config data
	SocketConnection socket;
	SOCKADDR_IN data;
#pragma endregion
	char * portName;
	SerialConnection serial;
	_DCB arduinoConfig;
	_DCB iRobotConfig;
	_DCB * currentConfig;

public:
	Bridge(void);
	//*************************************
	// Method:    Bridge
	// Parameter: char * type                                [in]   "sim" if we want to use
the simulator, COMx if we want to use the serial communication
	//*************************************
	Bridge(char * type);
	~Bridge(void);


	//*************************************
	// Method:    connect
	// Parameter: bool useArduino                            [in]   True -> if serial
comunication is going on it will choose Arduino config
	// Description:
	Starts the connection
	//*************************************
```

```
      void connect(bool useArduino = false);
      //***********************************
      // Method:    send
      // Returns:   int                                          [out]  How many bytes
have been send
      // Parameter: char * data                                  [in]   Byte buffer with the
bytes we want to send
      // Parameter: int size                                     [in]   Number of bytes
we want to sent
      //***********************************
      int send( char * data, int size);
      //***********************************
      // Method:    receive
      // Returns:   int                                          [out]  How many bytes
have been read
      // Parameter: char * buffer                                [out]  Buffer where the bytes
that we read will be stored
      // Parameter: size_t size                                  [in]   How many bytes we wanted
to read
      //***********************************
      int receive (char * buffer, size_t size);


      //***********************************
      // Method:    disconnect
      // Description:
      Closes the connection
      //***********************************
      void disconnect();
};
```

Bridge.cpp

```
#include "bridge.h"
#include <iostream>
using namespace std;

Bridge::Bridge(void)
{
	simulated = true;
}

Bridge::Bridge( char * type )
{
	if(!(simulated = type == "sim"))
	{
		portName = new char[strlen("\\\\.\\")+strlen(type)];
		strcpy(portName,"\\\\.\\");
		strcat(portName, type);
	}
}

Bridge::~Bridge(void)
{

}

void Bridge::connect(bool useArduino)
{
	if(simulated)
	{
		#pragma region Connect to the simulator
		//INITIALIZE TCP/IP SIMULATION
		memset(&data, 0,sizeof(data));
		data.sin_family = AF_INET;
		data.sin_port = htons(65000); //Port
		data.sin_addr.s_addr = inet_addr("127.0.0.1"); //Localhost = 127.0.0.1
		//CONNECT TO THE SIMULATOR SERVER
		socket.connect(data);
#pragma endregion
	}
	else
```

```
	{
		//INITIALIZE ARDUINO CONFIGURATION
		arduinoConfig.BaudRate = 9600;
		arduinoConfig.ByteSize = 8;
		arduinoConfig.Parity = NOPARITY;
		arduinoConfig.StopBits = ONESTOPBIT;
		//INITIALIZE IROBOT CONFIGURATION
		iRobotConfig.BaudRate = 57600;
		iRobotConfig.ByteSize = 8;
		iRobotConfig.Parity = NOPARITY;
		iRobotConfig.StopBits = ONESTOPBIT;
		//CHOOSE WITCH CONFIGURATION WILL BE USED
		if(useArduino) currentConfig = &arduinoConfig;
		else currentConfig = &iRobotConfig;
		//OPEN THE PORT
		serial.Open_Port(portName);
		serial.Set_Configure_Port(*currentConfig);
		serial.Set_Hands_Haking(NONE);
	}
}

int Bridge::send( char * buffer, int size )
{

	int n = 0;
	if (simulated)
	{
		n = socket.send(buffer,size);
	}
	else
	{
		serial.Clean_Buffer();
		n =serial.Write_Port(buffer,size);
	}
	return n;
}
```

```
int Bridge::receive( char * buffer, size_t size )
{
	int n = 0;
	if (simulated)
	{
		n = socket.receive(buffer,size);
	}
	else
	{
		char c;
		for(unsigned int i= 0; i< size;i++)
		{
			serial.Getc_Port(&c);
			buffer[i] = c;
		}
	n= size;

	}
	return n;
}

void Bridge::disconnect()
{
	if (simulated) serial.Close_Port();
	else socket.closeConnection();
}
```

IrobotConnection.h

```
#pragma once
#include "iRobotInstructionSet.h"
#include "ArduinoInstructionSet.h"
#include "bridge.h"

class IRobotConnection
{
private:
	Bridge 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 that was send to the robot
public:

	ArduinoInstructionSet * getArduinoInstructionSetGenerator()  {return &ArduinoInstructionGenerator;}
	//************************************
	// 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(char * connectionType);
	~IRobotConnection(void);


	//***********************************
	// Method:    connect
	// Parameter: bool useArduino                                          [in]   True -> if
serial comunication is going on it will choose Arduino config
	// Description:
	Starts the connection
	//***********************************
	void connect(bool useArduino = false);
	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);
};
```

IrobotConnection.cpp

```
#include "iRobotConnection.h"
#include<iostream>

using namespace std;


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

IRobotConnection::IRobotConnection( char * connectionType )
{
	buffer = (char *) malloc(512*sizeof (char));
	connection = Bridge(connectionType);
}



IRobotConnection::~IRobotConnection(void)
{
}

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

void IRobotConnection::connect(bool useArduino)
{
	connection.connect(useArduino);
}

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);
}

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);
}
```

Programa que corre en la placa Arduino

```
/*
  The circuit:
 * RX is digital pin 2 (connect to TX of other device)
 * TX is digital pin 3 (connect to RX of other device)

 created back in the mists of time
 modified 9 Apr 2012
 by Tom Igoe
 based on Mikal Hart's example

 This example code is in the public domain.

 */
#include <SoftwareSerial.h>
byte inByte, pin;
SoftwareSerial mySerial(2, 3); // RX, TX

void setup()
{
 // Open serial communications and wait for port to open:
  Serial.begin(57600);

  // set the data rate for the SoftwareSerial port
  mySerial.begin(9600);
}

void loop() // run over and over
{
  if (mySerial.available())
    {
      if (inByte > 127) //Si es una instruccion para el robot
      {
        Serial.write(inByte); //Escribimos el primer byte
        while(mySerial.available()) Serial.write(mySerial.read()); //escribimos el resto de bytes
      }
      else
      {
        switch(inByte)
        {
```

```
        case 100: //Opcode pinMode
          while(!mySerial.available());
          pin = mySerial.read();
          while(!mySerial.available());
          inByte = mySerial.read();
          if (inByte == 0) pinMode(pin,OUTPUT);
          else pinMode(pin, INPUT);
        break;
        case 101: //Opcode digital read
          while(!mySerial.available());
          pin = mySerial.read();
          Serial.write(digitalRead(pin));
          break;
        case 102: //Opcode digital write
          while(!mySerial.available());
          pin = mySerial.read();
          while(!mySerial.available());
          inByte = mySerial.read();
          if (inByte == 0) digitalWrite(pin,LOW);
          else digitalWrite(pin, HIGH);
          break;
        case 103: //Opcode analog read
          while(!mySerial.available());
          pin = mySerial.read();
          Serial.write(analogRead(pin));
          break;
        case 104: //Opcode analog write
          while(!mySerial.available());
          pin = mySerial.read();
          while(!mySerial.available());
          inByte = mySerial.read();
          analogWrite(pin,inByte);
          break;
      default:
          break;
       }
      }
    }
  if (Serial.available())
    mySerial.write(Serial.read());
}
```

Anexo II (Demos)

Demo avanzar hasta encontrar una linea negra en el suelo.

Ejecutada en el simulador:

```
void simulatorTest()
{
	IRobotConnection robot("sim");
	printf("Connecting... ");
	robot.connect();
	robot.control();
	printf("Done!!\n");
	robot.start();
	robot.driveDirect(100,100);
	while (robot.updateSensor(iRobotSensors::CLIFFFRONTLEFT) == 1)
		cout << robot.updateSensor(iRobotSensors::CLIFFFRONTLEFTSIGNAL) << endl;
	robot.driveDirect(0,0);
	cout << robot.updateSensor(iRobotSensors::CLIFFFRONTLEFTSIGNAL) << endl;
	Sleep (5000);
}
```

Ejecutar en iRobot Create con el dispositivo BAM

```
void iRobotBamTest()
{
	IRobotConnection robot("COM12");
	printf("Connecting... ");
	robot.connect();
	printf("Done!!\n");
	robot.start();
	robot.control();
	robot.driveDirect(200,200);
	while (robot.updateSensor(iRobotSensors::CLIFFFRONTLEFTSIGNAL) > 1000 )cout <<
	robot.updateSensor(iRobotSensors::CLIFFFRONTLEFTSIGNAL) << endl;
	robot.driveDirect(0,0);
	Sleep (5000);
}
```

Ejecutada en iRobot Create con el arduino conectado

```
void iRobotArduinoTest()
{
	IRobotConnection robot("COM10");
	printf("Connecting... ");
	robot.connect(true);
	printf("Done!!\n");
	robot.start();
	robot.control();
	robot.driveDirect(200,200);
	while (robot.updateSensor(iRobotSensors::CLIFFFRONTLEFTSIGNAL) > 1000 )cout << 
robot.updateSensor(iRobotSensors::CLIFFFRONTLEFTSIGNAL) << endl;
	robot.driveDirect(0,0);
	Sleep (5000);
}
```

Demo encender un led en el pin 13 de arduino

```
void ArduinoLedTest ()
{
	IRobotConnection robot("COM10");
	printf("Connecting... ");
	robot.connect(true);
	printf("Done!!\n");
	robot.getArduinoInstructionSetGenerator()->setPinMode(13,false);    //Configure the 
pin13 to output mode
	robot.getArduinoInstructionSetGenerator()->digitalWrite(13,true);   //Send digital 1 
to the pin13 (led turns on)
	Sleep (5000);
                //Wait 5 sec
	robot.getArduinoInstructionSetGenerator()->digitalWrite(13,false);  //Send digital 0 
to the pin13 (led turns off)
}
```

Anexo III (Guía de usuario)

GUIA DE USUARIO

Introducción

Esta guía esta diseñada para ayudar a poner en marcha cualquier proyecto que requiera de la utilización del iRobot Create con el modulo de Arduino.

Para quien no lo sepa, iRobot Create es un robot creado por la empresa iRobot[16] y que se ha usado mucho en investigación y docencia. En las imágenes 1,2 y 3 se puede ver el robot en cuestión, junto con los distintos componentes que lo forman.

Imagen 1: Fotografia de iRobot Create

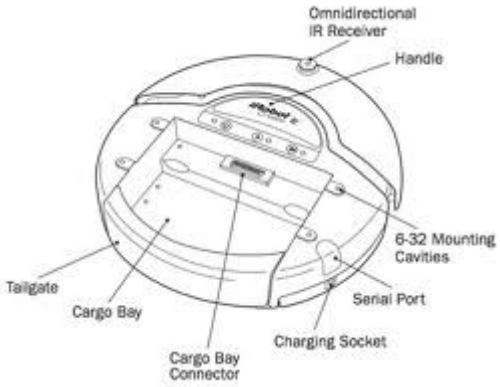


Imagen 2: Esquema superior iRobot Create

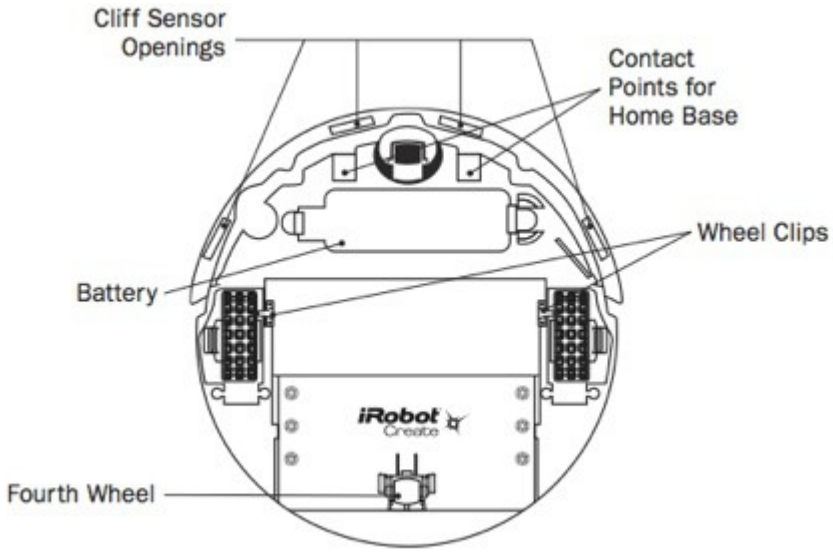


Imagen 3: Esquema inferior iRobot Create

16 http://www.irobot.com/es/

Como podemos apreciar en las imagenes 2 y 3, irobot Create dispone de los siguientes sensores y actuadores:

- 2 sensores de choque
- 4 sensores de precipicio
- 1 sensor infrarrojo
- 1 receptor de señales infrarrojas
- 2 ruedas con control de velocidad independiente

Para poder ampliar este hardware con mas sensores o actuadores, se va a utilizar su conexión DB-25 cuya descripción viene dada en la imagen 4 para conectarle una placa arduino.

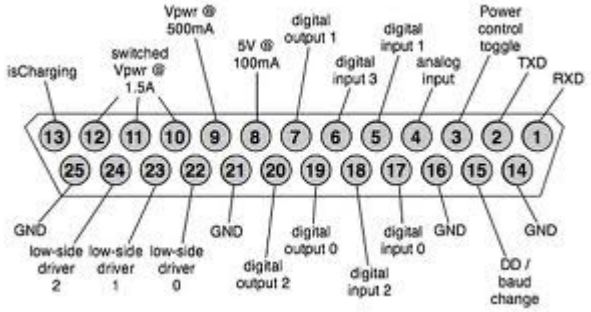
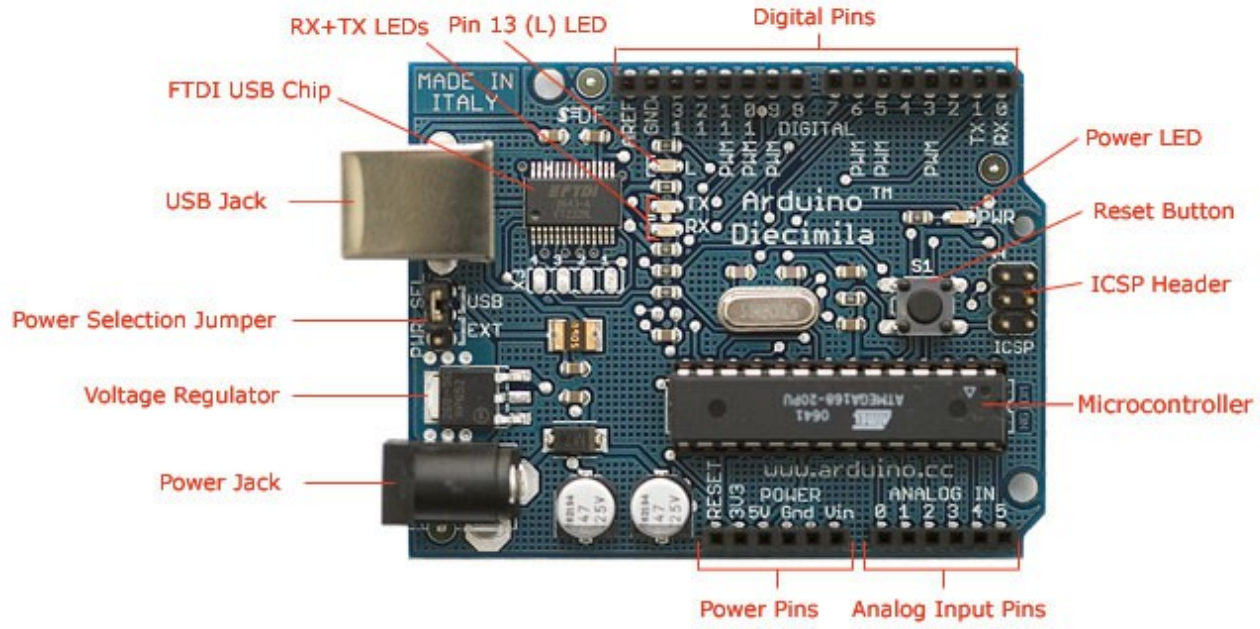


Imagen 4: Descripción de los pines

Antes de explicar como se van a conectar los componentes, veamos que es una placa Arduino. Arduino es una plataforma de desarrollo de código abierto, basada en una placa con un sencillo microcontrolador y un entorno de desarrollo para crear software (programas) para la placa.

Arduino puede ser usado para crear objetos interactivos, leyendo datos de una gran variedad de interruptores y sensores y controlar multitud de tipos de luces, motores y otros actuadores físicos. Los proyecto de Arduino pueden ser autónomos o comunicarse con un programa (software) que se ejecute en tu ordenador, en nuestro caso, usaremos su segunda opción.



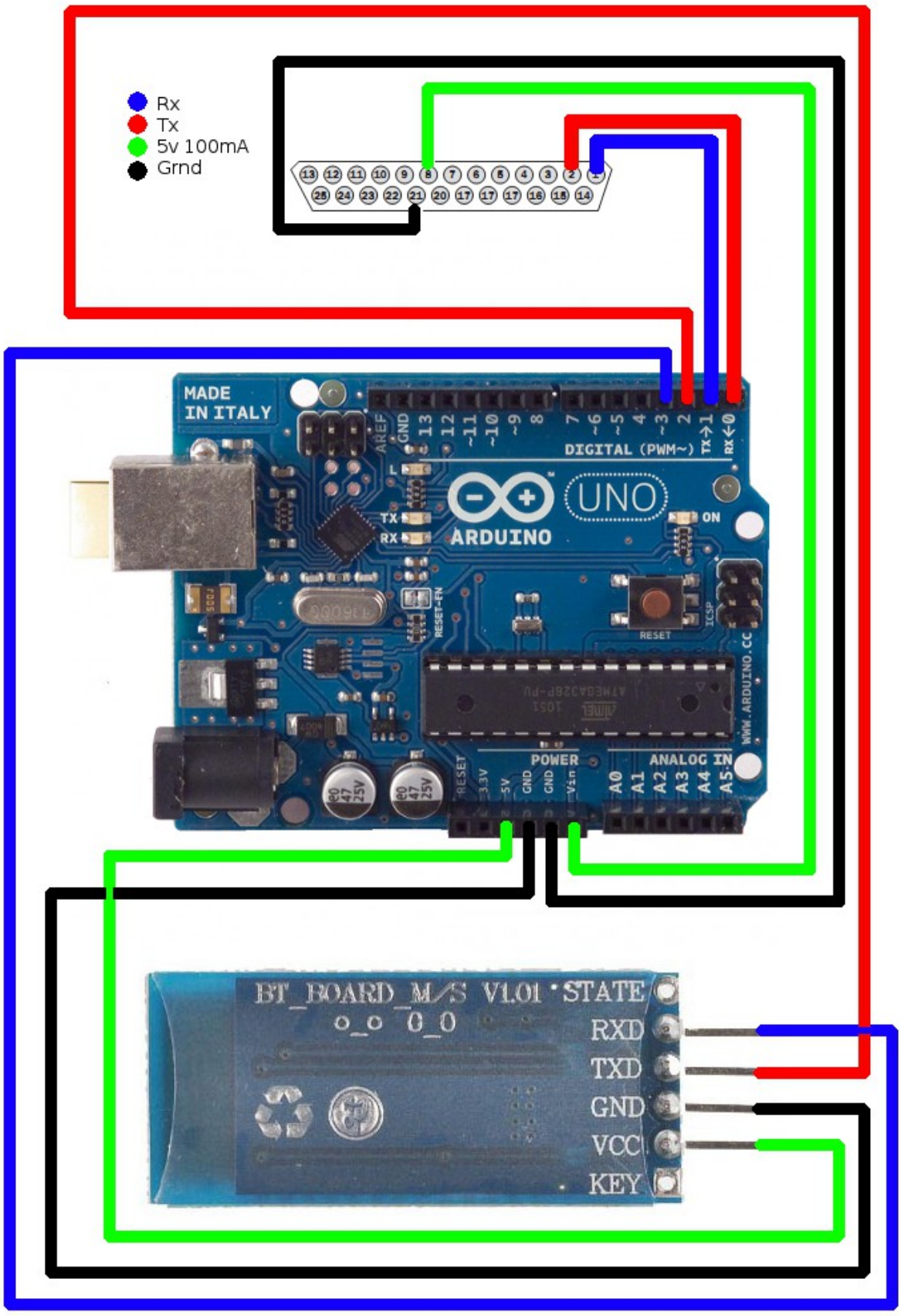
Photograph by SparkFun Electronics. Used under the Creative Commons Attribution Share-Alike 3.0 license.

Imagen 5:Descripción del hardware de Arduino Diecimila (versión antigua del Arduino UNO)

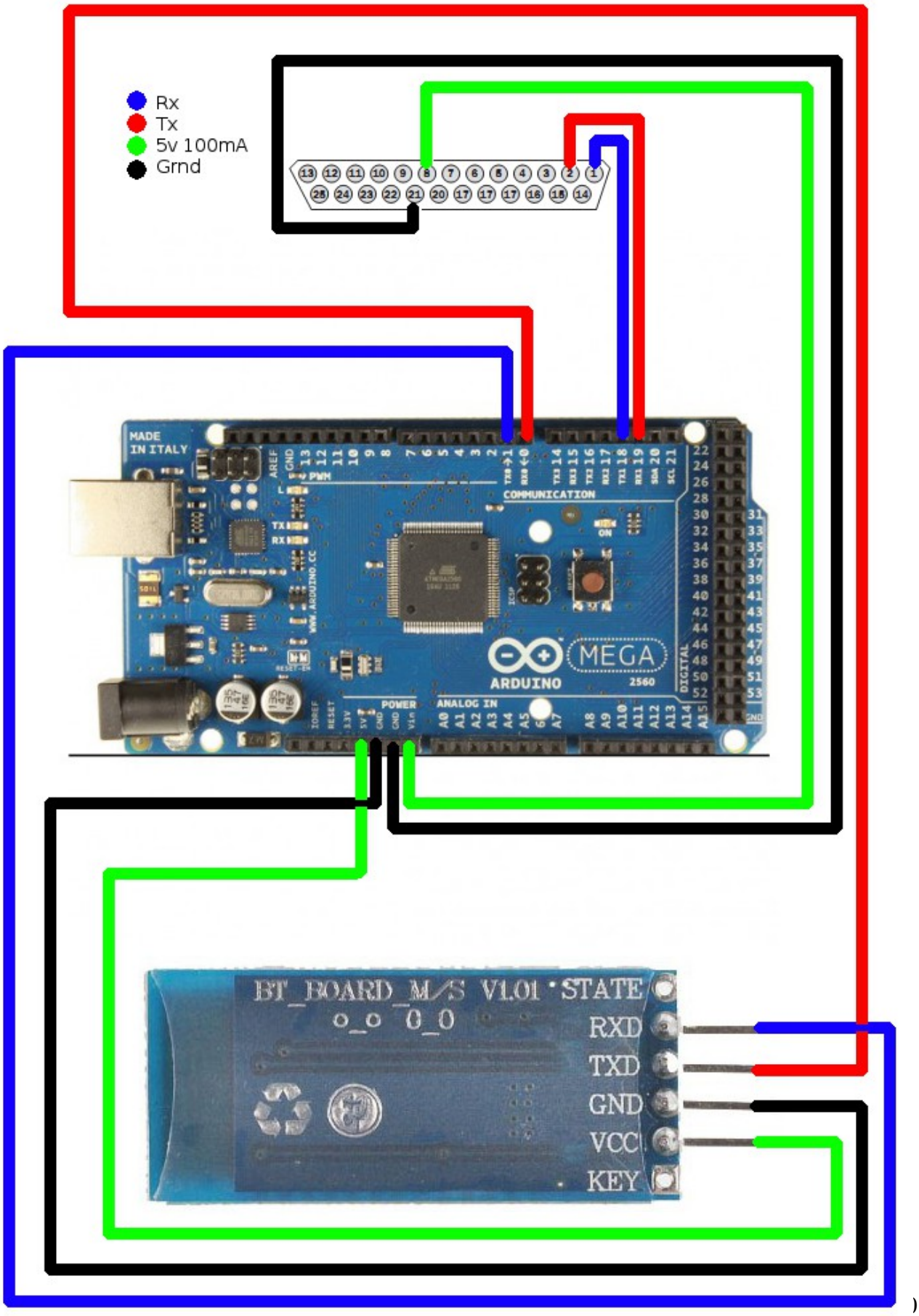
En la imagen 5 podemos observar una placa Arduino diecimila, que es la versión anterior a la placa Arduino UNO, el buque insignia de la gama Arduino. En esta imagen también se pueden apreciar los distintos tipos de pines que podemos encontrar en una placa Arduino separados en 3 bloques:

- Pines digitales: Por estos pines solo se pueden enviar y recibir voltajes de 0 o de 5 voltios, indicando 0 digital o 1 digital. Dentro de estos pines digitales, tenemos que destacar los pines 0 y 1 que están marcados como RX y TX respectivamente. Estos pines pueden trabajar como puerto serie, mas concrétamente, son una alternativa en pin al puerto serie que se genera en el puerto usb.
- Pines analógicos: Por estos pines se pueden enviar y recibir voltajes de entre 0 y 5 voltios, pudiendo hacerse lecturas numéricas
- Pines de energia: Estos pines sirven para alimentar la placa (Vin), para alimentar los sensores(3v3, 5v, Gnd) y para resetear la placa(Reset).

Esquema de conexión en placa Arduino uno:



Esquema de conexión en placa Arduino mega:



)

Configuración de la librería

A la hora de poner en marcha el proyecto, seguramente haya cambios en el hardware, esto quiere decir que las conexiones por puerto serie cambiarán, motivo por el cual se presenta aquí las partes del código a adaptar a las nuevas conexiones por puerto serie:

En la clase bridge:

```
void Bridge::connect(bool useArduino)
{	if(simulated)
	{
		#pragma region Connect to the simulator
		//INITIALIZE TCP/IP SIMULATION
		memset(&data, 0,sizeof(data));
		data.sin_family = AF_INET;
		data.sin_port = htons(65000); //Port
		data.sin_addr.s_addr = inet_addr("127.0.0.1"); //Localhost = 127.0.0.1
		//CONNECT TO THE SIMULATOR SERVER
		socket.connect(data);
		#pragma endregion
	}
	else
	{
		//INITIALIZE ARDUINO CONFIGURATION
		arduinoConfig.BaudRate = 9600;
		arduinoConfig.ByteSize = 8;
		arduinoConfig.Parity = NOPARITY;
		arduinoConfig.StopBits = ONESTOPBIT;
		//INITIALIZE IROBOT CONFIGURATION
		iRobotConfig.BaudRate = 57600;
		iRobotConfig.ByteSize = 8;
		iRobotConfig.Parity = NOPARITY;
		iRobotConfig.StopBits = ONESTOPBIT;
		//CHOOSE WITCH CONFIGURATION WILL BE USED
		if(useArduino) currentConfig == &arduinoConfig;
		else currentConfig = &iRobotConfig;
		//OPEN THE PORT
		serial.Open_Port(portName);
		serial.Set_Configure_Port(*currentConfig);
		serial.Set_Hands_Haking(NONE);
	}
```

```
}
```

Marcado en azul está la configuración para la conexión con la placa Arduino, esta conexión se hace por un dispositivo bluetooth y son los valores de configuración de este dispositivo los que tendremos que añadir en las variables indicadas.

Marcado en rojo tenemos la configuración del dispositivo BAM[17], un dispositivo que se espera sea sustituido de manera definitiva por la placa Arduino, no obstante en esta librería se le da soporte. En el caso de que ya la conexión por medio de Arduino pase a ser mas utilizada que la conexión BAM, se recomienda cambiar en el archivo bridge.h la inicialización por ausencia de la variable useArduino de la función connect a true quedando de la siguiente manera:

```
void connect(bool useArduino = true);
```

17 Para mas información acerca de este dispositivo visitar la siguiente dirección: http://www.elementdirect.com/files/10542B.pdf

A la hora de dar soporte al cambio de hardware, habrá que modificar también la configuración de las conexiones del programa que corre en la placa Arduino, para ello habrá que modificar la inicialización de las siguientes variables de su código:

```
// software serial : RX = digital pin 3, TX = digital pin 2
SoftwareSerial bluetooth(2, 3);
int iRobotRate = 57600;  //Arduino iRobot serial rate
int softwareSerialRate = 9600; // software serial doesn't support more than 9600 brate
```

Para la placa Arduino UNO y:

```
int iRobotRate = 57600;  //Arduino iRobot serial rate
int softwareSerialRate = 9600;
```

Para la placa Arduino mega.

Una vez realizados estos cambios, habría que compilar y enviar el nuevo programa a la placa. Para realizar esta labor hay que seguir el siguiente proceso:

1. Conectar la placa Arduino al pc usando el puerto USB.
2. Abrir el software de Arduino[18].
3. Cargar nuestro programa modificado en el.
4. Archivo → Cargar
5. Esperar a que termine de cargar el archivo y cerrar software de Arduino

18 El software de Arduino se puede descargar en la siguiente dirección: http://arduino.cc/en/Main/Software

Como trabajar con la librería

A la hora de trabajar con la librería, hay que añadir a nuestro programa el siguiente include:

#include “iRobotConnection.h”

Crear una variable de tipo iRobotConnection de alguna de las siguientes 2 formas:

1. `iRobotConnection variable (“sim”);` o `iRobotConnection variable;`
   Para indicar que queremos utilizar la conexión con el simulador.

2. iRobotConnection variable(“COM1”);
   Para indicar que queremos usar la conexión por puerto serie usando el puerto COM1

Una vez decidido el tipo de conexión que vamos a utilizar hay que establecerla, para ello se usa la opción:

1. variable.connect(); o variable.connect(false);
   Para indicar que en caso de utilizar puerto serie estamos conectándonos por medio de Arduino.
2. variable.connect(true);
   Para indicar que en caso de utilizar puerto serie estamos conectándonos por medio del dispositivo BAM.

Tras tener la conexión establecida tendremos que indicar al robot que queremos empezar a trabajar con los comandos:

`variable.start();` → Imprescindible para hacer cualquier cosa con el robot.

`variable.control();` → Imprescindible para poder utilizar los motores del robot.

A la hora de realizar una consulta de algún sensor hay que utilizar el namespace iRobotSensor de forma que la consulta del sensor seria la siguiente:

variable.updateSensor(iRobotSensors::distance);

El simulador

El funcionamiento del simulador es bastante sencillo, basta con ejecutar el archivo "*Roomba Simulator.exe*" para ponerlo en marcha. Una vez puesto en marcha, tan solo habrá que ejecutar el código deseado.

Los parámetros de conexión del servidor del simulador son:

ip: localhost (127.0.0.1)

puerto: 65000

sin family: AF_INET

protocolo: TCP/IP

A la hora de trabajar con el simulador podemos cambiar la posición del robot, para ello basta con poner las coordenadas en las que queremos que se posicione el robot y darle al botón *reset.*

Aviso: El botón reset reinicia el servidor del simulador, esto quiere decir que si hay algún programa ejecutándose quedara desconectado del simulador y habrá que reiniciarlo.

Otra cosa que nos permite realizar el simulador es cambiar la vista de la simulación, lo cual puede hacerse desde el menú *view → Camera mode.* Aparte de cambiar la vista de la simulación, nos permite cambiar la velocidad de ejecución desde el menú *Simulator → Running Speed.*

Aparte de todo esto, se nos permite editar el escenario con un editor que viene incorporado, pudiendo añadirse lineas e incluso obstáculos al entorno en el que se moverá el robot.

A la hora de crear nuevos entornos debido a un bug que tiene el editor de entornos hay que realizar los siguientes pasos:

1. Recomiendo crear la imagen con algún editor de imágenes y guardarla en formato png.
2. Desde el editor de entornos del simulador, que se encuentra en menú *environment → Environment editor,* podemos cargar la imagen anteriormente dibujada usando el boton *load image*.
3. Ponemos los obstaculos que queramos añadir, si queremos añadir alguno, y le damos al boton *save env.*

4. Vamos a la carpeta donde guarda el simulador los entornos que se encuentra en *%programfiles%\iRobot Create Simulator\Resources\Environments* y ahi buscamos el archivo que almacena el entorno que hemos creado. Este archivo es un archivo de tipo zip y contiene en su interior los siguientes archivos Environment.xml y floor.bmp.
5. Debido a que el simulador no reconoce los archivos en formato bmp tenemos que coger la imagen que habíamos dibujado en formato png, renombrarla a floor.png y guardarla dentro del zip.

6. Abrimos con un editor el archivo Environment.xml
7. Modificamos el apartado donde pone “floor.bmp” y lo sustituimos por “floor.png”.

   <environment width="20" height="20" name="Camino" image="floor.png">

   Los valores de width y height indican el tamaño del entorno. Si queremos que sea mas grande o mas pequeño aquí podemos modificarlo.

   En el apartado <robot x="1,0" y="1,0" rotation="0" /> esta la inicializacion del robot, si queremos que cuando se cargue el entorno el robot este en unas coordenadas concretas y que tenga una rotación concreta modificando estos valores podremos guardarlo.
8. Renombramos el archivo de imagen png a *floor.png*
9. Comprimimos en un zip el archivo floor.png y el archivo Environment.xml que hemos modificado.

Con esto ya se podrá cargar nuestro entorno en el simulador.

Como ampliar la librería

Esta librería esta creada a 3 capas:

- En la capa de nivel mas bajo tenemos:
- Juegos de instrucciones:

  Estos módulos, única y exclusivamente, van a generar las instrucciones que el hardware[19] va interpretar.

  Se ha realizado la separación entre ellos de esta manera, con intención de que cada pieza de hardware tenga su propio modulo de software de forma que si esa pieza se cambia, tan solo haya que sustituir su juego de instrucciones por el de la nueva pieza.
- Conexiones:

  Estos módulos son los encargados de gestionar la comunicación, ofrecen soporte para configurar la conexión entre nuestro programa y el hardware, pero no la configuran. Esta decisión se ha tomado pensando en que no haya que tocar ninguno de estos módulos aunque cambie el hardware, ya que... si se cambiase el hardware, habría que venir a estos módulos a modificarlos, y si se quisiera añadir un nuevo hardware, habría que añadirle el soporte en estos módulos, quedando al final las configuraciones muy esparcidas por los distintos archivos.
- En la capa intermedia tenemos:
- Puente de conexiones:

  En este nivel es donde se configuran los tipos de conexión para cada dispositivo.
- En la capa superior tenemos:
- 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.

De forma que:

Si cambiáramos de robot, habría que generar un nuevo juego de instrucciones para el robot, configurar la conexión en el puente de conexiones y darle soporte al nuevo juego de instrucciones en la interfaz.

Si cambiásemos el modulo bluetooth que une la placa arduino con el robot, habría simplemente que configurar la conexión en el puente de conexiones para que de soporte al nuevo modulo y cambiar la configuración del programa que corre la placa Arduino.

19 En nuestro caso el hardware va a ser, por un lado la placa arduino y por otro el robot iRobot Create.

Si cambiásemos de placa, habría que añadir el programa que da soporte al juego de instrucciones de arduino a la nueva placa para que pueda interpretarlo.

Si quisiéramos implementar una función que cambié el estado de un led de la placa Arduino, habría que implementarla en la interfaz usando el juego de instrucciones generado para tal propósito.

Si quisiéramos añadir una nueva instrucción de configuración de la placa arduino que no pueda realizarse por medio de ninguna de las instrucciones que hay actualmente en ella (véase por ejemplo cambiar el rango de voltaje de un pin), entonces habría que añadir el nuevo código de instrucción al juego de instrucciones de Arduino y añadir el soporte para esa nueva instrucción al programa que corre en la placa Arduino.

Ejemplos de programas

Programa que hace que el robot avance hasta encontrar una linea negra

Versión Arduino

```
void iRobotarduinoTest()
{
	IRobotConnection robot("COM10");
	printf("Connecting... ");
	robot.connect(true);
	printf("Done!!\n");
	robot.start();
	robot.control();
	robot.driveDirect(200,200);
	while (robot.updateSensor(iRobotSensors::CLIFFFRONTLEFTSIGNAL) > 1000 )cout << 
robot.updateSensor(iRobotSensors::CLIFFFRONTLEFTSIGNAL) << endl;
	robot.driveDirect(0,0);
	Sleep (5000);
}
```

Versión iRobot Create con dispositivo BAM

```
void iRobotBamTest()
{
	IRobotConnection robot("COM12");
	printf("Connecting... ");
	robot.connect();
	printf("Done!!\n");
	robot.start();
	robot.control();
	robot.driveDirect(200,200);
	while (robot.updateSensor(iRobotSensors::CLIFFFRONTLEFTSIGNAL) > 1000 )cout << 
robot.updateSensor(iRobotSensors::CLIFFFRONTLEFTSIGNAL) << endl;
	robot.driveDirect(0,0);
	Sleep (5000);
}
```

Versión Simulador

```
void simulatorTest()
{
	IRobotConnection robot("sim");
	printf("Connecting... ");
	robot.connect();
	robot.control();
	printf("Done!!\n");
	robot.start();
	robot.driveDirect(200,200);
	while (robot.updateSensor(iRobotSensors::CLIFFFRONTLEFT) == 1)cout << 
robot.updateSensor(iRobotSensors::CLIFFFRONTLEFTSIGNAL) << endl;
	robot.driveDirect(0,0);
	cout << robot.updateSensor(iRobotSensors::CLIFFFRONTLEFTSIGNAL) << endl;
	Sleep (5000);
}
```

Programa que hace que se encienda un led conectado en el pin 13 de la placa Arduino

```
void ArduinoLedTest ()
{
	IRobotConnection robot("COM10");
	printf("Connecting... ");
	robot.connect(true);
	printf("Done!!\n");
	robot.getArduinoInstructionSetGenerator()->setPinMode(13,false);    //Configure the 
pin13 to output mode
	robot.getArduinoInstructionSetGenerator()->digitalWrite(13,true);   //Send digital 1 
to the pin13 (led turns on)
	Sleep (5000);
                //Wait 5 sec
	robot.getArduinoInstructionSetGenerator()->digitalWrite(13,false);  //Send digital 0 
to the pin13 (led turns off)
}
```