

UNIVERSIDAD DE GUADALAJARA
CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS



Interfaz y Biblioteca para el control y administración de
sensores en una plataforma robótica

Proyecto de titulación que presenta

Omar Alejandro Rodríguez Rosas

para obtener el grado de Licenciado en Ingeniería en Computación

Directora:

Dra. Nancy Guadalupe Arana Daniel

Asesora:

Dra. Alma Yolanda Alanis García

2014, GUADALAJARA, JALISCO; MÉXICO



UNIVERSIDAD DE GUADALAJARA
CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS
SECRETARÍA ACADÉMICA
COORDINACIÓN DE INGENIERÍA EN COMPUTACIÓN

CUCEI/CDCOM/202/14
Código: 005351278

C. RODRIGUEZ ROSAS OMAR ALEJANDRO
EGRESADO DE LA CARRERA INGENIERÍA EN COMPUTACIÓN
P R E S E N T E

Por este conducto le damos a conocer el Dictamen emitido por el Comité de Titulación de Ingeniería en Computación con relación a su solicitud de aprobación de modalidad y opción de titulación, conforme al Reglamento General de Titulación de la Universidad de Guadalajara:

Artículo 12, Modalidad de Investigación y Estudios de Posgrado,
Opción V. Diseño o Rediseño de Equipo, Aparato o Maquinaria
Y conforme al reglamento particular del Centro Universitario de Ciencias Exactas e Ingenierías

Artículo 12, Modalidad de Investigación y Estudios de Posgrado
Opción V. Diseño o Rediseño de Equipos, Aparatos, Maquinaria, Proceso o Sistema de
Computación y / o informática

Con el título: **“Interfaz y biblioteca para el control y administración de sensores en una plataforma robótica.”**

Con base en lo anterior este Comité emite el siguiente:

DICTAMEN APROBADO

Que queda asentado en el acta de sesión con fecha 16 de Junio del 2014, con el folio No. 87/13, que se autoriza el cambio de nombre del presente trabajo de “Desarrollo de bibliotecas para la administración de sensores de un robot móvil Qbot en ROS” por “Interfaz y biblioteca para el control y administración de sensores en una plataforma robótica”

Este Comité designa a la Dra. Nancy Guadalupe Arana Daniel como Directora.

Con base al procedimiento académico-administrativo de titulación del CUCEI y el acta de sesión, se le otorga una prórroga de 6 meses para la presentación del trabajo.

ATENTAMENTE
“PIENSA Y TRABAJA”
Guadalajara, Jal; Junio 24 del 2014



MTRO. JUAN JOSÉ LÓPEZ CISNEROS
PRESIDENTE DEL COMITÉ DE TITULACIÓN
INGENIERÍA EN COMPUTACIÓN

c.c. Dra. Nancy Guadalupe Arana Daniel como Directora
JJLC/ylsa



UNIVERSIDAD DE GUADALAJARA

Centro Universitario de Ciencias Exactas e Ingenierías

Secretaría Académica / Coordinación de la Carrera de Ingeniería en Computación

COMPROBANTE ACADEMICO

El Comité de Titulación de la Carrera de Ingeniería en Computación, hace constar que el (la) egresado (a):

RODRIGUEZ ROSAS OMAR ALEJANDRO código 005351278 Folio 87/13

Egresado (a) del Plan Semestral en Sistema de Créditos, ha cumplido con los requisitos académicos que le dan derecho a solicitar ceremonia de titulación. La modalidad y opción que le han sido aprobadas se indica en la siguiente tabla:

Artículo 9 Desempeño Académico Sobresaliente	I.- Excelencia Académica II.- Titulación por Promedio
Artículo 10 Exámenes	I.- Examen Global Teórico Práctico
	II.- Examen Global Teórico
	III.- Examen General de Certificación Profesional
	IV.- Examen de Capacitación Profesional
Artículo 11 Producción de Materiales Educativos	I.- Guías Comentadas o Ilustradas
	II.- Paquete Didáctico
Artículo 12 Investigación y Estudios de Posgrado	I.- Cursos de Maestría o Doctorado en IES de reconocido Prestigio
	II.- Trabajo Monográfico de Actualización
	III.- Seminario de Investigación
	IV.- Seminario de Titulación
	V.- Diseño o Rediseño de Equipos, Aparatos, Maquinaria, proceso o Sistema de Computación y/o Informática.
Artículo 14 Tesis, Tesina e Informes	X I.- Tesis
	II.- Tesina
	III.- Informe de Prácticas Profesionales

Y han sido designados los Profesores: Dra. Nancy Guadalupe Arana Daniel como Presidente, a la Dra. Alma Yolanda Alanís García como Secretario y al Dr. Edwin Christian Becerra Alvarez como Vocal.

Presidente, Secretario y Vocal del jurado respectivamente.

Miembros del Comité para realizar la ceremonia.

ATENTAMENTE
"PIENSA Y TRABAJA"
Guadalajara, Jal; Junio 24 del 2014



JL COMITÉ DE TITULACIÓN
MTRO. JUAN JOSE LÓPEZ CISNEROS EN COMPUTACIÓN
Presidente del Comité de Titulación de Ingeniería en Computación

DECLARATORIA

El que suscribe este documento, Omar Alejandro Rodríguez Rosas, declaro que el trabajo titulado “Interfaz y Biblioteca para el control y administración de sensores en una plataforma robótica”, es de autoría propia y no ha sido presentado total ni parcialmente en ningún otro lugar. Que además no contiene material que pertenezca a otra persona.



FIRMA

Omar Alejandro Rodríguez Rosas

La que suscribe, Dra. Nancy Guadalupe Arana Daniel, directora del proyecto de Diseño o rediseño de equipo, aparato o maquinaria titulado “Interfaz y Biblioteca para el control y administración de sensores en una plataforma robótica”, certifico que he revisado tanto el formato como el contenido de este trabajo. Por lo anterior, autorizo la impresión de este documento para su defensa, de conformidad con lo estipulado en el capítulo V, artículo 23 y capítulo VI, artículo 24 del Reglamento General de Titulación de la Universidad de Guadalajara, así como en el capítulo V, artículos 23 y 24 del Reglamento de Titulación del Centro Universitario de Ciencias Exactas e Ingenierías.



FIRMA

Dra. Nancy Guadalupe Arana Daniel



COMITÉ DE TITULACIÓN
INGENIERÍA EN COMPUTACIÓN



UNIVERSIDAD DE GUADALAJARA

Centro Universitario de Ciencias Exactas e Ingenierías

Secretaría Académica / Coordinación de la Carrera de Ingeniería en Computación

CUCEI/CDCOM/315/13
Código: 005351278

C. RODRÍGUEZ ROSAS OMAR ALEJANDRO
EGRESADO DE LA CARRERA INGENIERÍA EN COMPUTACIÓN
P R E S E N T E

Por este conducto le damos a conocer el Dictamen emitido por el Comité de Titulación de Ingeniería en Computación con relación a su solicitud de aprobación de modalidad y opción de titulación, conforme al Reglamento General de Titulación de la Universidad de Guadalajara:

Artículo 12, Modalidad de Investigación y Estudios de Posgrado,
Opción V. Diseño o Rediseño de Equipo, Aparato o Maquinaria
Y conforme al reglamento particular del Centro Universitario de Ciencias Exactas e Ingenierías

Artículo 12, Modalidad de Investigación y Estudios de Posgrado
Opción V. Diseño o Rediseño de Equipos, Aparatos, Maquinaria, Proceso o Sistema de
Computación y / o informática

Con el título: **“Desarrollo de bibliotecas para la administración de sensores de un robot móvil Qbot en ROS.”**

Con base en lo anterior este Comité emite el siguiente:

DICTAMEN APROBADO

Que queda asentado en el acta de sesión con fecha 24 de junio del 2013, con el folio No. 87/13, y en la que este Comité designa a la Dra. Nancy Guadalupe Arana Daniel como Director.

Con base al procedimiento académico-administrativo de titulación del CUCEI se le otorga el plazo de un año a partir de la fecha de su dictaminación para la presentación del trabajo.

ATENTAMENTE
“PIENSA Y TRABAJA”
Guadalajara, Jal; Junio 26 del 2013

MTRO. JUAN JOSÉ LÓPEZ CISNEROS
PRESIDENTE DEL COMITÉ DE TITULACIÓN

**COMITÉ DE TITULACIÓN
INGENIERÍA EN COMPUTACIÓN**



c.c. Dra. Nancy Guadalupe Arana Daniel como Director
JJLC/ylsa

X

Índice general

Índice de figuras	XXI
1. Introducción	1
2. Marco teórico	5
2.1. El iRobot Create®	5
2.2. El estándar RS-232	6
2.3. El estándar USB 2.0	7
2.3.1. Protocolo de comunicación USB	9
2.3.2. Conexión, desconexión y enumeración de dispositivos	11
2.3.3. La clase HID	11
3. Especificación Tecnológica-Metodológica	13
3.1. Estructura organizacional del hardware	13
3.2. Estructura organizacional del software	14
3.2.1. Capa de comunicación	14
3.2.2. Capa de abstracción	16
3.2.3. Capa de aplicación	17
3.3. Tecnologías a utilizar	17
3.3.1. Microcontrolador PIC18F4550	17
3.3.2. La arquitectura x86	17
3.3.3. c++ 11	18
4. Implementación de hardware	19
4.1. Alimentación eléctrica	19
4.2. Circuito para la conexión serial RS-232	19
4.3. Circuito para la comunicación USB 2.0	22

5. Implementación de Software	25
5.1. Firmware del microcontrolador	25
5.1.1. Configuración del oscilador para la comunicación USB 2.0	25
5.1.2. Descriptores USB	26
5.1.3. Configuración de puertos	30
5.1.4. Lectura de sensores externos	31
5.2. Interacción con el driver HID	34
5.3. Comunicación mediante el estándar RS-232	34
5.4. Comunicación serial en Linux	34
5.4.1. Formato básico de instrucción	36
5.4.2. Instrucciones de lectura	37
5.4.3. Instrucciones de streaming	39
5.4.4. Ordenamiento de bytes y tipos de datos	41
6. Pruebas y Análisis de resultados	45
7. Manual de Usuario	49
7.1. La placa fenólica	49
7.2. Conexiones físicas	49
7.3. Dependencias de software	50
7.3.1. Recompilación de la biblioteca con g++	51
7.4. Uso de UDG_Create de manera local	55
7.5. Uso de UDG_Create como biblioteca estándar	56
7.6. La Clase Create	56
7.6.1. Tipos de datos y enumeraciones	57
7.6.1.1. enum errorCode	57
7.6.1.2. enum modes	57
7.6.1.3. enum baudCode	57
7.6.1.4. enum chargingstates	58
7.6.1.5. enum infraredbytechars	58
7.6.1.6. enum VerbosityLevels	59
7.6.1.7. t_verbosity	59
7.6.1.8. enum BoolSigned	59
7.6.1.9. boolSigned	59
7.6.1.10. NUMBER_OF_SENSORS	60
7.6.1.11. int16	60
7.6.2. Miembros (privados)	60
7.6.2.0.1. std::string portName	60

7.6.2.0.2.	int mode	60
7.6.2.0.3.	bool charging	60
7.6.2.0.4.	int portDescriptor	60
7.6.2.0.5.	int baudRate	60
7.6.2.0.6.	bool bumpRight	61
7.6.2.0.7.	bool bumpLeft	61
7.6.2.0.8.	bool wheelDropRight	61
7.6.2.0.9.	bool wheelDropLeft	61
7.6.2.0.10.	bool wheelDropCaster	61
7.6.2.0.11.	bool wall	61
7.6.2.0.12.	bool cliffLeft	61
7.6.2.0.13.	bool cliffFrontLeft	61
7.6.2.0.14.	bool cliffFrontRight	61
7.6.2.0.15.	bool cliffRight	61
7.6.2.0.16.	bool virtualWall	61
7.6.2.0.17.	bool ld0	61
7.6.2.0.18.	bool ld1	62
7.6.2.0.19.	bool ld2	62
7.6.2.0.20.	bool rightWheel	62
7.6.2.0.21.	bool leftWheel	62
7.6.2.0.22.	unsigned char infraredbyte	62
7.6.2.0.23.	bool advancebtn	62
7.6.2.0.24.	bool playbtn	62
7.6.2.0.25.	int distance	62
7.6.2.0.26.	int angle	62
7.6.2.0.27.	unsigned char chargingstate	62
7.6.2.0.28.	int voltage	62
7.6.2.0.29.	int current	63
7.6.2.0.30.	unsigned char batterytemperature	63
7.6.2.0.31.	int batterycharge	63
7.6.2.0.32.	int batterycapacity	63
7.6.2.0.33.	int wallsignal	63
7.6.2.0.34.	int cliffls	63
7.6.2.0.35.	int cliffls	63
7.6.2.0.36.	int clifffrs	63
7.6.2.0.37.	int cliffrs	63
7.6.2.0.38.	bool digitalinput0	63
7.6.2.0.39.	bool digitalinput1	64

7.6.2.0.40. bool digitalinput2	64
7.6.2.0.41. bool digitalinput3	64
7.6.2.0.42. bool baudchangerate	64
7.6.2.0.43. int cargoanalogsignal	64
7.6.2.0.44. bool homebase	64
7.6.2.0.45. bool internalcharger	64
7.6.2.0.46. unsigned char oimode	64
7.6.2.0.47. unsigned char songnumber	64
7.6.2.0.48. bool songplaying	64
7.6.2.0.49. unsigned char streampackets	65
7.6.2.0.50. int reqvelocity	65
7.6.2.0.51. int reqradius	65
7.6.2.0.52. int reqrvelocity	65
7.6.2.0.53. int reqlvelocity	65
7.6.2.0.54. bool streamingState	65
7.6.2.0.55. bool externalSensorsEnabled	65
7.6.2.0.56. t_verbosity robotVerbosity	65
7.6.3. Funciones	65
7.6.3.1. Publicas	65
7.6.3.1.1. Create()	65
7.6.3.1.2. Create(std::string _portName, t_verbosity verbosityLevel)	66
7.6.3.1.3. ~Create()	66
7.6.3.1.4. std::string getPortName()	66
7.6.3.1.5. void start()	66
7.6.3.1.6. void baud(unsigned char baudRate)	66
7.6.3.1.7. void control()	66
7.6.3.1.8. void safe()	67
7.6.3.1.9. void full()	67
7.6.3.1.10. void spot()	67
7.6.3.1.11. void cover()	67
7.6.3.1.12. void coverAndDock()	67
7.6.3.1.13. void demo(unsigned char demo)	67
7.6.3.1.14. void drive(int velocity, int radius)	67
7.6.3.1.15. void driveDirect(int rightVelocity, int leftVelocity)	68
7.6.3.1.16. void leds(unsigned char bit,unsigned char color, unsigned char intensity)	68

7.6.3.1.17. void digitalOutputs(unsigned char output-Bits)	68
7.6.3.1.18. void pwmLowSideDrivers(unsigned char dir-ver1, unsigned char driver1, unsigned char driver0)	69
7.6.3.1.19. void lowSideDrivers(unsigned char bits) . . .	69
7.6.3.1.20. void sendIr(unsigned char byteValue)	69
7.6.3.1.21. void song(unsigned char,unsigned char,...) .	69
7.6.3.1.22. void playSong(unsigned char songNumber) . .	70
7.6.3.1.23. int sensors(unsigned char idPacket)	70
7.6.3.1.24. int getSizePacket(int idPacket)	70
7.6.3.1.25. void stream(unsigned char* destinationBuf-fer,void* thread,int n,...)	70
7.6.3.1.26. void pauseResumeStream(bool streamState)	71
7.6.3.1.27. void script(unsigned char n,...);	71
7.6.3.1.28. void playScript()	71
7.6.3.1.29. void showScript()	71
7.6.3.1.30. void waitTime(unsigned char time)	72
7.6.3.1.31. void waitDistance(int distance)	72
7.6.3.1.32. void waitAngle(int angle)	72
7.6.3.1.33. void waitEvent(unsigned char event)	72
7.6.3.1.34. char* charMode(int mode)	72
7.6.3.1.35. int getBaudCode(int baudCode)	73
7.6.3.1.36. bool getBumpRight()	73
7.6.3.1.37. bool getBumpLeft()	73
7.6.3.1.38. bool getWheelDropRight()	73
7.6.3.1.39. bool getWheelDropLeft()	73
7.6.3.1.40. bool getWheelDropCaster()	73
7.6.3.1.41. bool getWallSeen()	73
7.6.3.1.42. bool getCliffLeft()	73
7.6.3.1.43. bool getCliffFrontLeft()	74
7.6.3.1.44. bool getCliffFrontRight()	74
7.6.3.1.45. bool getCliffRight()	74
7.6.3.1.46. bool getVirtualWall()	74
7.6.3.1.47. bool getLd0()	74
7.6.3.1.48. bool getLd1()	74
7.6.3.1.49. bool getLd2()	74
7.6.3.1.50. bool getRightWheel()	75

7.6.3.1.51. bool getLeftWheel()	75
7.6.3.1.52. unsigned char getInfraredByte()	75
7.6.3.1.53. bool getAdvanceBtn()	75
7.6.3.1.54. bool getPlayBtn()	75
7.6.3.1.55. int getDistance()	75
7.6.3.1.56. int getAngle()	75
7.6.3.1.57. unsigned char getChargingState()	76
7.6.3.1.58. int getVoltage()	76
7.6.3.1.59. int getCurrent()	76
7.6.3.1.60. unsigned char getBatteryTemperature() . .	76
7.6.3.1.61. int getBatteryCharge()	76
7.6.3.1.62. int getBatteryCapacity()	77
7.6.3.1.63. int getWallSignal()	77
7.6.3.1.64. int getCliffLS()	77
7.6.3.1.65. int getCliffFLS()	77
7.6.3.1.66. int getCliffFRS()	77
7.6.3.1.67. int getCliffRS()	77
7.6.3.1.68. bool getDigitalInput0()	77
7.6.3.1.69. bool getDigitalInput1()	78
7.6.3.1.70. bool getDigitalInput2()	78
7.6.3.1.71. bool getDigitalInput3()	78
7.6.3.1.72. bool getBaudRateChange()	78
7.6.3.1.73. int getCargoAnalogSignal()	78
7.6.3.1.74. bool getHomeBase()	78
7.6.3.1.75. bool getInternalCharger()	78
7.6.3.1.76. unsigned char getOIMode()	79
7.6.3.1.77. unsigned char getSongNumber()	79
7.6.3.1.78. bool getSongPlaying()	79
7.6.3.1.79. unsigned char getStreamPackets()	79
7.6.3.1.80. int getRequestedVelocity()	79
7.6.3.1.81. int getRequestedRadius()	79
7.6.3.1.82. int getRequestedRVelocity()	79
7.6.3.1.83. int getRequestedLVelocity()	80
7.6.3.1.84. bool getStreamingState()	80
7.6.3.1.85. void getExternalSensors(int[NUMBER_OF_SENSORS] sensors)	80
7.6.3.1.86. bool getExternalSensorsEnabledStatus() . .	80
7.6.3.1.87. void setVerbosity(t_verbosity)	80

7.6.3.1.88. int getExternalNthSensor(int n)	80
7.6.3.1.89. int toLittleEndian(unsigned char* source,int nbytes,int* destination,boolSigned sign) . .	81
7.6.3.2. Privadas	81
7.6.3.2.1. void error(int error,void* info)	81
7.6.3.2.2. void printRobotMessage(const char* message,...)	81
7.6.3.2.3. int readBumpsAndWheelDrops(unsigned char *data)	81
7.6.3.2.4. int readWall(unsigned char *data)	82
7.6.3.2.5. int readCliffLeft(unsigned char *data)	82
7.6.3.2.6. int readCliffFrontLeft(unsigned char *data)	82
7.6.3.2.7. int readCliffFrontRight(unsigned char *data)	82
7.6.3.2.8. int readCliffRight(unsigned char *data)	82
7.6.3.2.9. int readVirtualWall(unsigned char *data) . .	83
7.6.3.2.10. int readLSDriverAndWheelO(unsigned char *data)	83
7.6.3.2.11. int readInfraredByte(unsigned char *data) . .	83
7.6.3.2.12. int readButtons(unsigned char *data)	83
7.6.3.2.13. int readDistance(unsigned char *data)	84
7.6.3.2.14. int readAngle(unsigned char *data)	84
7.6.3.2.15. int readChargingState(unsigned char *data)	84
7.6.3.2.16. int readVoltage(unsigned char *data)	84
7.6.3.2.17. int readCurrent(unsigned char *data)	84
7.6.3.2.18. int readBatteryTemperature(unsigned char *data)	85
7.6.3.2.19. int readBatteryCharge(unsigned char *data)	85
7.6.3.2.20. int readWallSignal(unsigned char *data) . .	85
7.6.3.2.21. int readCliffLS(unsigned char *)data	85
7.6.3.2.22. int readCliffFLS(unsigned char *data)	85
7.6.3.2.23. int readCliffFRS(unsigned char *data)	86
7.6.3.2.24. int readCliffRS(unsigned char *data)	86
7.6.3.2.25. int readDigitalInputs(unsigned char *data) . .	86
7.6.3.2.26. int readCargoAnalogSignal(unsigned char *data)	86
7.6.3.2.27. int readChargingSources(unsigned char *data)	86
7.6.3.2.28. int readOIMode(unsigned char *data)	87
7.6.3.2.29. int readSongNumber(unsigned char *data) . .	87

7.6.3.2.30. int readSongPlaying(unsigned char *data)	87
7.6.3.2.31. int readStreamPackets(unsigned char *data)	87
7.6.3.2.32. int readReqVelocity(unsigned char *data)	87
7.6.3.2.33. int readReqRadius(unsigned char *data)	88
7.6.3.2.34. int readReqRVelocity(unsigned char *data)	88
7.6.3.2.35. int readReqLVelocity(unsigned char *data)	88
7.6.3.2.36. int updateSensor(unsigned char packetID, int size)	88
7.6.3.2.37. void commonInitializationProcedures(string port,bool auto)	89
7.6.3.2.38. int16ToInt16(int integer)	89
8. Conclusiones y trabajo futuro	91
Bibliografía	91
9. Anexos	97
9.1. main.c	97
9.2. DataTypes.h	128
9.3. ExternalSensors.cpp	129
9.4. ExternalSensors.h	130
9.5. SerialPort.c	130
9.6. SerialPort.h	138
9.7. UDG_Create.cpp	139
9.8. UDG_Create.h	184
9.9. USBLayer.cpp	191
9.10. USBLayer.h	196

Índice de figuras

1.1.	El Qbot de Quanser	3
2.1.	El Create [®] de iRobot	6
2.2.	Distribución de pines en el estándar RS-232 para los conectores DB-9 hembra y DB-25 macho	8
2.3.	Protocolo de comunicación mediante RS-232 para una transmisión de 8 bits de datos sin bit de paridad	9
2.4.	Topología de bus USB	10
3.1.	Diagrama a bloques de la estructura del hardware	14
3.2.	Diagrama del modelo en capas del software	15
4.1.	Diagrama de entradas/salidas del puerto DB-25 en el iRobot Create [®]	20
4.2.	Regulación de volaje mediante un L7805CV	21
4.3.	Diagrama eléctrico del circuito para la comunicación serial RS-232	23
4.4.	Diagrama eléctrico del circuito para la comunicación USB	24
5.1.	Importancia del correcto manejo de signos y orden de bytes. El ejemplo I) muestra algunas de las posibles interpretaciones para un entero de 16 bits sin signo, solo el inciso c) es correcto. En el ejemplo II) se muestran posibles interpretaciones para un entero negativo, solo el inciso d) es correcto	42
6.1.	Comparación entre distintos modelos de programación. a) iRobot Create [®] y UDG_Create. b) Quanser Qbot y Simulink	46
6.2.	Ambiente de programación y ejecución del Qbot con Simulink	46
6.3.	Ambiente de programación y ejecución para el Create [®] con UDG_Create	47
6.4.	Comparación de configuración entre los robots Qbot y Create [®]	48
7.1.	Esquema de las terminales en la placa fenólica	50

7.2.	Placa fenólica para el uso de UDG_Create	51
7.3.	Conektor DB-25 en la bahía de carga del Create®	52
7.4.	Conexión de los sensores externos mediante un cable USB	52
7.5.	Conektor DB-9 para la comunicación serial	52
7.6.	Terminales para cables recomendadas	53
7.7.	Conexión de sensores externos	54
7.8.	Conexión de la placa fenólica a la PC	54
7.9.	Modo de uso recomendado	55

Capítulo 1

Introducción

Desde sus inicios a principios de la década de 1960, la importancia de la Robótica y los Sistemas Inteligentes como áreas de investigación ha crecido de manera exponencial de tal forma que hoy en día, ambas disciplinas juegan un papel de gran importancia en campos tan diversos como la educación, la salud, la manufactura, el cuidado del hogar, el control de inventarios , la seguridad, el rescate, operaciones militares, entre otros [1].

Como era de esperarse, este crecimiento se vió reflejado en la industria como un incremento en la demanda de profesionistas capacitados en dichas disciplinas[2]. Sin embargo, en años recientes diversas universidades de los Estados Unidos y la Unión Europea han experimentado una disminución en el interés por carreras afines [3][4] lo que sugiere dificultades para satisfacer esta demanda a mediano plazo.

Buscando subsanar por lo menos de manera parcial esta brecha en la oferta-demanda educacional, estudios recientes han demostrado cómo el estudio de la robótica en etapas tempranas de la educación (digase preparatoria) ayuda a disminuir el rechazo hacia las ciencias matemáticas, de la ingeniería y la computación que muchos alumnos perciben como difíciles, altamente demandantes y en general indeseables[5].

Si bien es cierto que la robótica forma parte del mapa curricular en muchas universidades, el trabajo con robots reales supone una tarea abrumadora para la mayoría de los estudiantes a nivel de licenciatura lo que sumado a los altos costos del equipo de laboratorio lo convierte en un privilegio reservado generalmente para los investigadores y alumnos de posgrado[6].

Por fortuna, en años recientes se han lanzado al mercado diferentes plataformas, frameworks y APIs tales como los Lego Mindstorms, los MIT Handyboards, los Rug Warriors, entre otros [7] proveyendo a estudiantes y profesores de herramientas para lograr una introducción sencilla a temas básicos de robótica y una transición suave

hacia tópicos más complejos. Este tipo de plataformas han sido bien recibidas en gran medida gracias a su bajo precio, su robustez y su corto ciclo de desarrollo haciéndolos ideales para el trabajo en proyectos de licenciatura o incluso preuniversitarios.

Las nuevas posibilidades ofrecidas por esos sistemas de bajo costo han sido exploradas a profundidad en un gran número de investigaciones educativas y han probado ser un recurso invaluable para el salón de clases ya que ayudan a los estudiantes a desarrollar habilidades como la descomposición de problemas, la creación de procedimientos, el trabajo multidisciplinario en equipo[7] y diseño de software en un ambiente controlado y divertido [5] volviéndolos conscientes del equipo disponible en los laboratorios y brindándoles la confianza, conocimientos y experiencia necesarios para proponer futuros trabajos de posgrado[6].

En el caso concreto del Laboratorio de Sistemas Inteligentes en el Centro Universitario de Ciencias Exactas e Ingenierías de la Universidad de Guadalajara, la experiencia con este tipo de sistemas comenzó en el año 2010 con la adquisición de 2 robots Qbot (figura 1.1), un modelo comercializado por la empresa canadiense Quanser y basado en la plataforma Create® de iRobot (ver 2.1) que añade a esta última cinco sensores de distancia infrarrojos, 3 ultrasónicos y una cámara web que le permiten una mayor flexibilidad y lo hacen apto para proyectos más avanzados. Además, el control del robot y la administración de sus sensores puede llevarse a cabo mediante Matlab/Simulink a través QuaRC -el bloque control para sistemas en tiempo real de Quanser-OpenCV, QMRFC, Roomba, entre otros. [8]

Sin embargo, a pesar de que el Qbot permite el prototipado rápido de muchas aplicaciones también posee serios inconvenientes que lo hacen poco apropiado para el desarrollo de proyectos más formales. El primero de ellos es la comunicación inalámbrica por medio del protocolo TCP que se utiliza como forma de contacto principal entre la PC y el robot, el cual es conocido por generar considerables retardos debidos a las limitaciones de ancho de banda y al preprocessamiento de seguridad adicional propios del protocolo[9]. El segundo tiene que ver con los problemas inherentes al uso de Matlab/Simulink para el control de sistemas en tiempo real como la inflexibilidad del código[10] y el no determinismo temporal[11] además de la naturaleza considerablemente más lenta de los lenguajes de programación interpretados (comparados con otros lenguajes compilados como c/c++).

Estas consideraciones nos llevaron a buscar otras alternativas que permitieran un desarrollo más eficiente y flexible. El primer enfoque para ello fue interceptar el proceso de transmisión de datos mediante un sniffer TCP y aplicar ingeniería inversa a este proceso con la finalidad de sustituir el código generado en Simulink por el propio. Aunque factible, dado que los procedimientos específicos de comunicación PC-robot

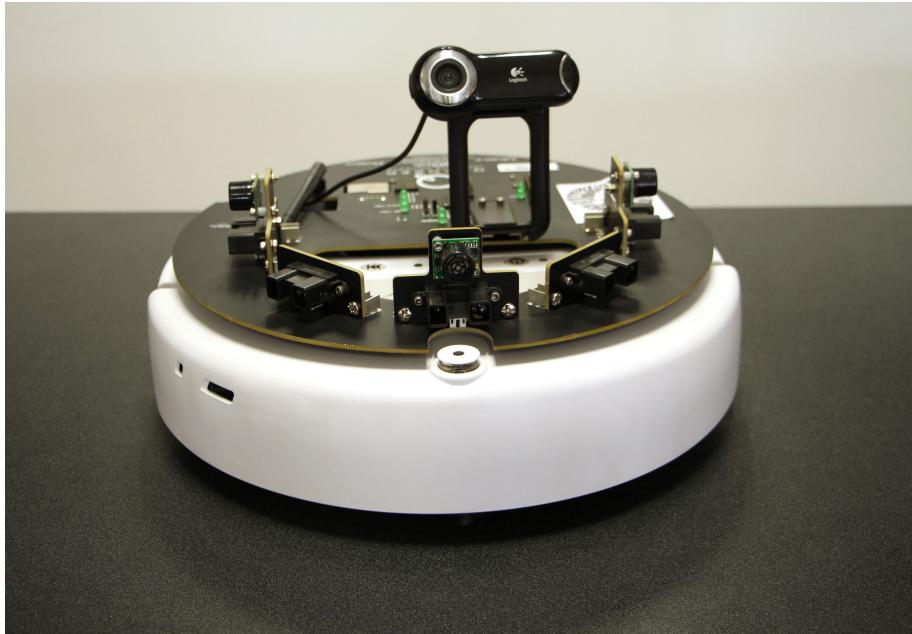


Figura 1.1: El Qbot de Quanser

del Qbot son de código cerrado y no documentados, esta idea fue descartada debido a la cantidad de trabajo que implicaba, las posibles implicaciones legales respecto al uso de licencias y a que, en el mejor de los casos solo se resolvía el segundo de los dos problemas anteriormente presentados pero no el desempeño general del sistema entorpecido por la comunicación inalámbrica. Por ello, se optó por retirar el hardware de control de sensores y comunicación del Qbot construido alrededor de una microcomputadora Gumstix (de arquitectura ARM) y sustituirlo por un diseño de hardware a medida directamente sobre el iRobot Create® que constituye la base del Qbot.

A pesar de que existen algunas bibliotecas de control para el iRobot Create® (ver 2.1) ninguna de ellas satisface del todo las necesidades de los trabajos de investigación del laboratorio.

Por esta razón, el objetivo primordial de este trabajo es desarrollar la biblioteca UDG_Create, un *framework* integral con un funcionamiento a bajo nivel pero con funciones y abstracciones de hardware de alto nivel para el manejo del robot que mediante el uso de c++ como lenguaje de programación principal y un sistema operativo basado en linux permita un desarrollo flexible y eficiente eliminando prácticamente en su totalidad los costos de licencias de software. Además, gracias su implementación

alrededor de un sistema de arquitectura x86, se expanden aún más las capacidades del iRobot Create® aumentando considerablemente su poder de computo y su compatibilidad con un sinfín de aplicaciones y dispositivos de hardware diseñados para los procesadores con este tipo de set de instrucciones.

Capítulo 2

Marco teórico

2.1. El iRobot Create®

Este robot, basado en la aspiradora robótica autónoma Roomba (ver 2.1) y comercializado por la empresa iRobot fue concebido como un kit de desarrollo que permite la implementación de algoritmos y diseño de nuevos comportamientos de manera sencilla manteniendo los aspectos mecánicos y electrónicos de la operación transparentes al usuario[12]. Posee 10 modos de demostración que pueden utilizarse directamente de fábrica, pero también puede ser manejado por una PC o microcontrolador a través de un puerto serial dedicado con un conector mini-DIN 7 o mediante un conector DB-25.

Está provisto de 20 sensores entre los que se incluyen detectores de bordes, un receptor infrarrojo, un parchoque de 4 estados, motores independientes para cada una de sus dos ruedas móviles, tres LED's indicadores, tres botones programables, una salida digital de niveles TTL de 3 bits y 3 salidas para PWM que permiten la adición de electrónica adicional como brazos robóticos o actuadores[13][14].

En el ámbito académico este robot ha sido utilizado como base para investigaciones en temas como telepresencia [15], mapeo tridimensional [16][17] y localización [18][19] demostrando sus capacidades como una plataforma asequible, robusta y fácil de usar.

Como muestra de su popularidad cabe mencionar la presencia de drivers, modelos de simulación y bibliotecas de control en importantes entornos como The Player Project [20], Universal Real-time Behavior Interface (URBI) [21], Microsoft Robotics Studio [22], Webots [22] y el Robot Operating System (ROS)[23].



Figura 2.1: El Create® de iRobot

2.2. El estándar RS-232

El estándar para comunicación serial RS-232, también conocido como EIA/TIA RS-232C, define el cableado, conectores, niveles de voltaje, temporización de señales y protocolo de intercambio de información entre un Dispositivo Terminal de datos (DTE) y un Dispositivo de Comunicación de datos (DCE) mediante una transmisión serial de datos binarios.

Una implementación completa del estándar utiliza un conector de 25 terminales DB-25, pero también son comunes versiones reducidas de 9 e implementaciones mínimas de 4, en los que se asigna una función y posición a cada terminal (ver figura 2.2). La comunicación es asíncrona (pero síncrona a nivel de bit) en modos simplex, half duplex y full duplex.

A pesar de no estar considerado en la especificación del estándar, es muy común conectar dos dispositivos DTE (por ejemplo dos computadoras personales) directamente entre ellos en lo que se denomina una conexión de módem nulo. Para lograrlo se utilizan por lo general cables cruzados cuya organización interna suele variar de una implementación a otra.

Para la transferencia de datos binarios se establecen niveles de voltaje en lógica negativa con amplios margenes de tolerancia que le brindan una gran robustez. Durante

la transmisión, un 0 binario será representado mediante un voltaje de entre +5v y +15v mientras que un 1 se enviará con un nivel de entre -5v y -15v. Para la recepción estos niveles varían ligeramente ya que un voltaje de entre +3v y +13v se interpretará como un 0 binario y entre -3v y -13v como un 1.

El protocolo de comunicación señala que, para la transmisión, en estado ocioso se envían bits en nivel alto de manera continua. El comienzo de la comunicación se indica mediante un bit de inicio que es siempre un nivel bajo seguido de los entre 5 y 8 bits de datos. El fin de la transmisión se especifica mediante un nivel alto que debe durar por lo menos $100\mu s$ para después iniciar una nueva transmisión o volver al estado ocioso.

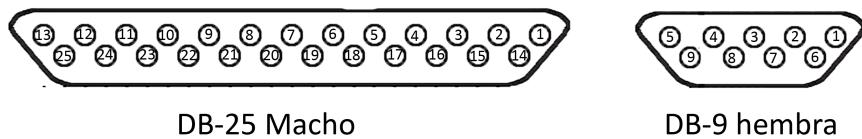
La recepción comienza con el dispositivo esperando a que la línea pase a un nivel bajo. Al ser detectado , el receptor esperá $51\mu s$ de manera que se encuentre a la mitad del bit para descartar que el cambio de nivel haya sido ocasionado por ruido. Posteriormente esperará $104\mu s$ más hasta encontrarse a la mitad del primer bit de datos. El resto de las lecturas se hacen de la misma manera hasta encontrar el bit de parada (ver 2.3)[24].

2.3. El estándar USB 2.0

El bus universal en serie (USB por sus siglas en inglés) es un estándar que define el cableado y protocolos de comunicación entre una computadora y sus periféricos. La especificación original publicada por primera vez en 1996 buscaba facilitar la interconexión entre las computadoras personales y los teléfonos, brindar flexibilidad y reconfigurabilidad a los sistemas de computo mediante software amigable y hardware *plug-and-play* paliando la indisponibilidad, en aquel entonces, de puertos bidireccionales de bajo costo y velocidad media no ligados a un dispositivo en particular.

Un tiempo después, en el año 2000, el aumento en la velocidad de procesamiento de las computadoras y las crecientes capacidades de los periféricos motivaron la definición de una nueva especificación USB 2.0 en la que se aumentaba la tasa máxima de transferencia del bus de 12Mb/s a 480Mb/s proporcionando soporte para voz, audio y video y agregando algunas clases de dispositivos, pero manteniendo a la vez el bajo costo, flexibilidad y compatibilidad total con versiones anteriores.

Esta especificación, USB 2.0, es la que se utiliza a lo largo de este trabajo.



DB-25	DB-9	Nombre	Dirección	Descripción
1	-	-	-	Tierra protegida
2	3	TD	SALIDA	Trasmisión de datos (Tx, TxD)
3	2	RD	ENTRADA	Recepción de datos (Rx, RxD)
4	7	RTS	SALIDA	Petición de envío
5	8	CTS	ENTRADA	Transmisión lista
6	6	DSR	ENTRADA	Datos listos
7	5	SGND	-	Tierra de señal/retorno común
8	1	CD	ENTRADA	Detección de portadora (DCD)
9	-	-	-	Reservado
10	-	-	-	Reservado
11	-	-	-	No asignado
12	-	SDCD	ENTRADA	Detección de portadora secundaria
13	-	SCTS	ENTRADA	Transmisión lista secundaria
14	-	STD	SALIDA	Transmisión de datos secundaria
15	-	DB	SALIDA	Reloj de transmisión (TCLK, TxCLK)
16	-	SRD	ENTRADA	Recepción de datos secundaria
17	-	DD	ENTRADA	Reloj de recepción(RCLK)
18	-	LL	-	Bucle de retorno local
19	-	SRTS	SALIDA	Petición de envío secundaria
20	4	DTR	SALIDA	Terminal de datos lista
21	-	RL/SQ	-	Detector de calidad de señal/bucle de retorno remoto
22	9	RI	ENTRADA	Indicador de anillo
23	-	CH/CI	SALIDA	Selector de tasa de señal
24	-	DA	-	Reloj auxiliar(ACLK)
25	-	-	-	No asignado

Figura 2.2: Distribución de pines en el estándar RS-232 para los conectores DB-9 hembra y DB-25 macho

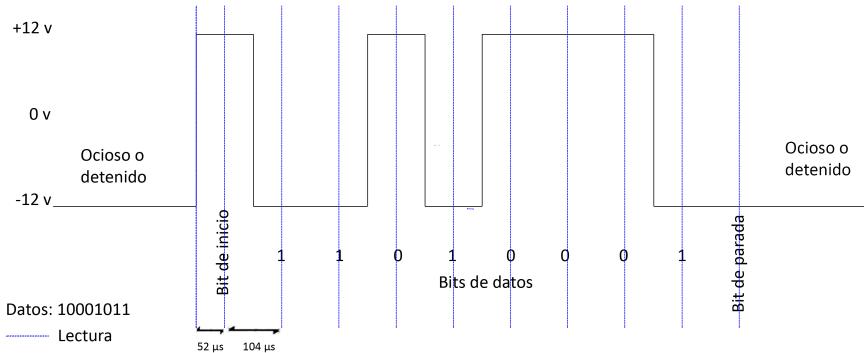


Figura 2.3: Protocolo de comunicación mediante RS-232 para una transmisión de 8 bits de datos sin bit de paridad

2.3.1. Protocolo de comunicación USB

Un sistema USB está compuesto por un *host* principal, concentradores y dispositivos conectados con una topología de estrella escalonada (ver 2.4) en la que el centro de cada estrella es un concentrador y el resto de las conexiones son entre el host principal y un concentrador o dispositivo, o bien, entre un concentrador y otro concentrador o dispositivo.

Debido a los retrasos de propagación del cableado y las limitaciones de tiempo que establece el propio protocolo, el número de niveles en la topología está limitado a siete incluyendo al host, además de que el último escalón no puede contener un concentrador o dispositivo compuesto. Los tipos de transferencias de datos definidas en el estándar son:

Transferencias de control: Se utilizan para configurar dispositivos recién conectados o para implementar funcionalidades muy específicas.

Transferencias por volumen: Se refiere a grandes cantidades de datos secuenciales, por ejemplo, los provenientes de una cámara digital.

Transferencias por interrupción: Están generalmente asociadas con funcionalidades de notificación de eventos que se presentan en cualquier momento.

Transferencias Isócronas: Consiste en la creación, entrega y consumo de datos en tiempo real a una tasa constante. Se utiliza para aplicaciones sensibles al tiempo como lo es la transmisión de voz. Algunos mecanismos de seguridad e integridad de datos como la retransmisión no se aplican a este tipo de transferencia.

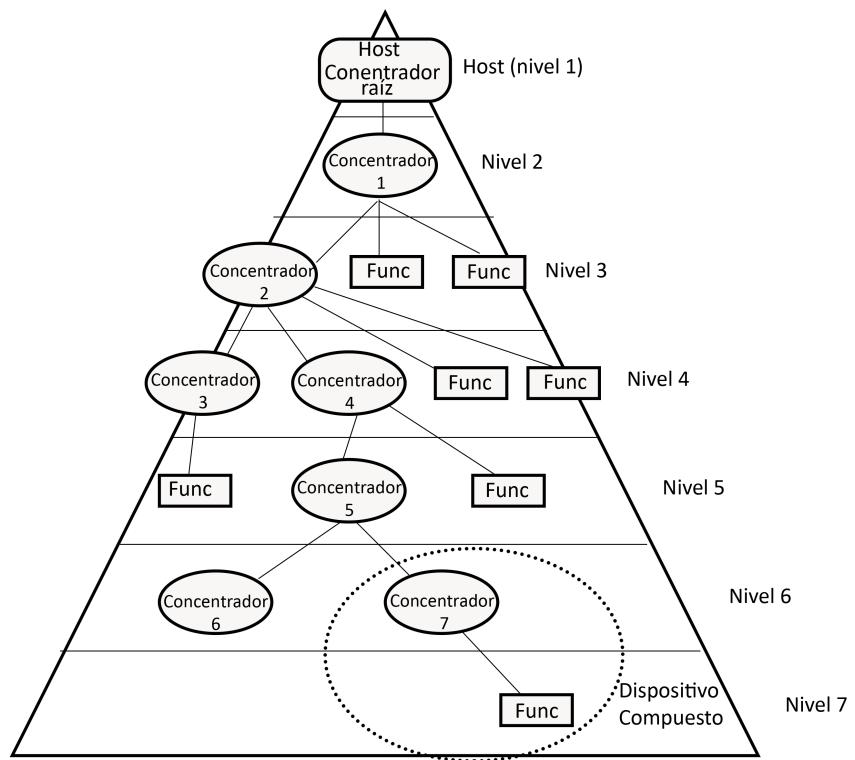


Figura 2.4: Topología de bus USB

2.3.2. Conexión, desconexión y enumeración de dispositivos

Dado que se permite la conexión y desconexión de dispositivos en cualquier momento, una enumeración de estos y sus respectivos buses se lleva a cabo de manera permanente.

Cada concentrador contiene bits de estado mediante los que se reporta la conexión o desconexión de un dispositivo y que son constantemente consultados por el *host*. Si un nuevo dispositivo se ha conectado se habilitan los puertos y se dirigen algunos mensajes de control a las direcciones predeterminadas. El *host* entonces asigna una dirección al dispositivo, determina su tipo y dirige el resto de la rutina de control al *endpoint* 0 de esta dirección. Si el dispositivo es un concentrador se repite lo anterior por cada uno de los elementos conectados a él. Para la desconexión simplemente se deshabilita el puerto y se envía una notificación al *host*[25].

2.3.3. La clase HID

Las configuraciones de control que se llevan a cabo al conectar el dispositivo dependerán de sus funcionalidades. Por ello, USB define ciertas clases de dispositivos, cada una de las cuales describe las configuraciones y operaciones comunes entre ellos. Una clase de particular importancia es la de Dispositivos de Interfaz Humana (HID por sus siglas en inglés), que se utiliza a lo largo del presente trabajo.

Esta clase fue originalmente diseñada, como lo sugiere su nombre, para su implementación en dispositivos usados por los seres humanos para interactuar con las computadoras tales como ratones, teclados, controles para juegos, etc. Sin embargo, debido a lo genérico de sus configuraciones , cualquier dispositivo puede ser utilizado como HID siempre y cuando se apegue a la lógica de la clase que es, por cierto, sumamente flexible.

Esta clase posee la ventaja de no requerir la programación de drivers adicionales ya que estos son proporcionados por la mayoría de los sistemas operativos modernos.

Capítulo 3

Especificación Tecnológica-Metodológica

En este capítulo se presenta una descripción a alto nivel de la estructura y flujo de ejecución de la biblioteca tanto en el hardware como en el software así como una breve descripción de algunas de las tecnologías utilizadas para su implementación.

3.1. Estructura organizacional del hardware

La figura 3.1 muestra un diagrama a bloques que ilustra la estructura y el flujo de la comunicación entre los distintos elementos que conforman el sistema.

iRobotCreate: la plataforma robótica que se comunica con el resto del sistema mediante el estándar RS-232 a través de un puerto DB-25 con dos terminales dedicadas a ello.

Conversión RS-232/TTL: Este módulo funciona como intermediario y se encarga de la conversión de niveles de voltaje RS-232 (hasta +/- 15v) a niveles TTL (hasta +/- 5v) y viceversa. Está construido a partir de un circuito integrado MAX232.

USB HID: Es un bloque basado en un microcontrolador PIC18F4550 cuya función es convertir señales tanto analógicas como digitales provenientes de sensores externos y transmitirlas mediante un flujo de bytes a la computadora.

Sensores externos: La implementación específica de este módulo dependerá de la aplicación y comprende todos aquellos sensores ajenos al robot que expanden sus capacidades para reconocer su entorno. El diseño actual posee 8 terminales para la conexión de sensores digitales (de niveles TTL) y otras 13 para sensores analógicos cuya salida se encuentre entre 0 y 5v.

Computadora x86: Es una de las piezas centrales del sistema y se encarga, además

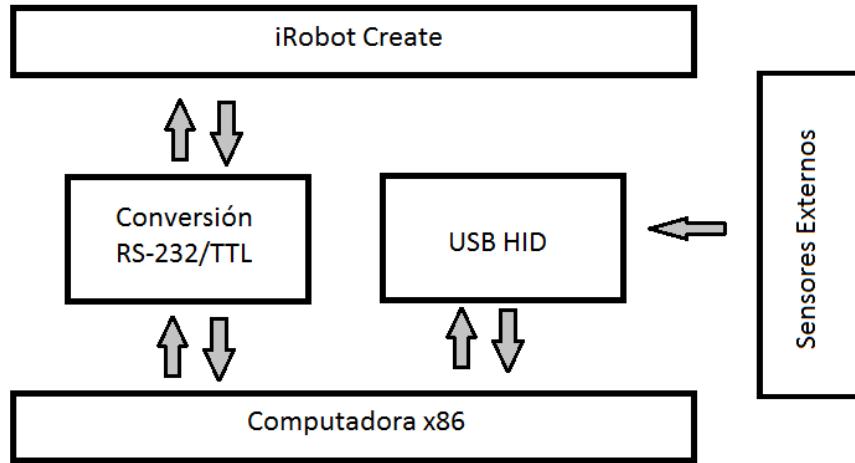


Figura 3.1: Diagrama a bloques de la estructura del hardware

de servir como vínculo entre el resto de los bloques, de coordinar y procesar el flujo de información proveniente del robot durante su operación.

3.2. Estructura organizacional del software

A fin de conseguir un diseño flexible, robusto y altamente portátil, dado el gran número de sistemas operativos compatibles con la arquitectura x86 y las diferencias entre ellos, fué necesario crear un diseño modular a base de capas que permitiera la independencia de sistemas operativos, plataformas e incluso lenguajes de programación. El esquema de la figura 3.2 ilustra este diseño.

3.2.1. Capa de comunicación

La comunicación directa con el robot se lleva a cabo en esta capa a través de descriptores de hardware e instrucciones de lectura y escritura operadas directamente sobre ellos. Es aquí donde tiene lugar el verdadero intercambio de datos y debido a su cercanía con el hardware debe ser reimplementada para cada sistema operativo. El siguiente pseudocódigo ejemplifica el tipo de función que se puede encontrar a este nivel:



Figura 3.2: Diagrama del modelo en capas del software

```
Escribir_a_puerto_Serial(BYTE[] datos)
{
    descriptor_puerto <- abrir(
        DIRECCION_FISICA_DEL_PUERTO)
    Escribir( descriptor_puerto , datos)
    cerrar( descriptor_puerto )
}
```

3.2.2. Capa de abstracción

Su propósito es proveer funciones genéricas, humanamente legibles e independientes de implementación para operaciones de entrada/salida comunes. La abstracción de los componentes de hardware del robot (sensores, LED's, motores, etc) también se lleva a cabo en este nivel. Así, esta capa se convierte en una interfaz que permite la interacción entre las capas superior e inferior. Para su implementación, siempre que el hardware y el lenguaje de programación lo permitan, se puede sacar gran ventaja del uso del paradigma orientado a objetos. Una típica implementación en la capa de abstracción luce como el siguiente pseudocódigo:

```
Clase Robot
{
    ENTERO sensor1
    ENTERO modo
    ENTERO rueda_izquierda

    ( ... )

    Avanzar( ENTERO velocidad )
    {
        buffer <- [ CODIGO_AVANZAR, velocidad .msb ,
                    velocidad .lsb ]
        Escribir_a_puerto_Serial( buffer )
    }
}
```

3.2.3. Capa de aplicación

Capa a nivel de usuario del programa. Aquí son descritas todas las funcionalidades del robot para una aplicación en específico. Todas las operaciones a bajo nivel deben ser transparentes para el programador. Por ejemplo:

```
main( )
{
    [ Clase ] Robot robot <- new Robot( )
    si( no_hay_obstaculos )
        robot .Avanzar( 100 )
}
```

3.3. Tecnologías a utilizar

3.3.1. Microcontrolador PIC18F4550

El PIC18F4550 de Microchip es un microcontrolador de bajo costo y consumo. Para este trabajo se utiliza en su encapsulado PDIP de 40 terminales que proporciona cuatro puertos de entrada/salida de entre siete y ocho bits digitales y analógicos, además de un convertidor analógico-digital de 13 canales con resolución de 10 bits. Una de sus características más sobresalientes es el soporte para USB 2.0 de baja (1.5 Mbps) o alta (12Mbps) velocidad, transferencias de control isocronas, por interrupción y por volumen (*bulk*), con hasta 32 *endpoints* y regulador de voltaje integrado. [26]

3.3.2. La arquitectura x86

La especificación para esta arquitectura fue creada por Intel en 1978 y ha sido el estándar para las computadoras personales durante los últimos 30 años [30] aunque también se encuentra presente en otros dispositivos como plataformas embebidas, SBC y móviles. Esto proporciona grandes ventajas como la generalizada familiaridad de la mayor parte de los programadores con la arquitectura y la gran disponibilidad de sistemas operativos, software y hardware compatibles. Además, las reducciones de tamaño y consumo eléctrico de las Unidades de Procesamiento Gráfico (GPU) tanto integradas como discretas propician la alineación de la robótica y sistemas inteligentes con la creciente popularidad de la programación gráfica y de propósito general en paralelo [31], territorio virtualmente inexplorado hasta el momento.

3.3.3. c++ 11

Se trata de un nuevo estándar del lenguaje c++ que introduce algunos cambios al núcleo del lenguaje y adiciones a la biblioteca estándar. Además de mejorar el rendimiento en tiempo de compilación y ejecución, una de sus mayores contribuciones es el soporte para la programación multihilo que se utiliza en este trabajo para el manejo de funciones bloqueantes (por ejemplo la lectura del puerto serial) mediante clases y funciones especializadas en ello.

Capítulo 4

Implementación de hardware

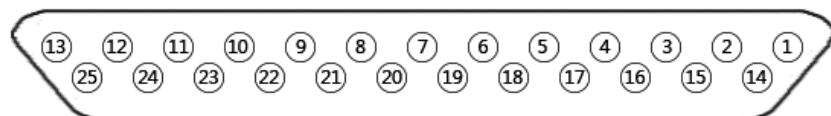
4.1. Alimentación eléctrica

Uno de los más importantes aspectos a considerar durante el diseño de un circuito como este es la alimentación eléctrica. Dado que la mayoría de los componentes que lo conforman operan al mismo voltaje y a fin de evitar el uso fuentes de energía externas al sistema, se optó por utilizar como fuente primaria la batería propia del robot.

El iRobot Create[®], es capaz de alimentar circuitos externos mediante salidas de volaje dedicadas a través de su conector DB-25 (ver 4.1) que pueden proporcionar 5v y máximos de 100mA o 1.5A cuando el robot está encendido [12]. Para brindar más flexibilidad al diseño, se han elegido las terminales 9 y 25 como fuente de alimentación que corresponden a las salidas Vpwr y GND de la batería del robot. Los 18v provenientes de la batería son reducidos a 5v mediante un regulador de voltaje positivo L7805CV que admite cargas de hasta 1.5A y cuyo diagrama de conexión se muestra en la figura 4.2.

4.2. Circuito para la conexión serial RS-232

Como se mencionó anteriormente, la comunicación entre la computadora de arquitectura x86 y el iRobot Create[®] se lleva a cabo mediante una conexión serial RS-232 pero que debido a limitaciones de hardware debe ser convertida a niveles TTL antes de ser utilizada. Esto se logra mediante el circuito integrado MAX232 cuyos capacitores externos llevan a cabo las multiplicaciones y divisiones de voltaje necesarias para convertir los hasta +/- 15v definidos en el estándar a los +/- 5v



Pin	Name
1	Serial RXD
2	Serial TXD
3	Power Control Toggle
4	Analog input
5	Digital input 1
6	Digital input 3
7	Digital output 1
8	Switched 5v
9	Vpwr
10	Switched Vpwr
11	Switched Vpwr
12	Switched Vpwr
13	Robot charging
14	GND
15	Device detect/Baud rate change
16	GND
17	Digital input 0
18	Digital input 2
19	Digital output 0
20	Digital output 2
21	GND
22	Low side driver 0
23	Low side driver 1
24	Low side driver 2
25	GND

Figura 4.1: Diagrama de entradas/salidas del puerto DB-25 en el iRobot Create®

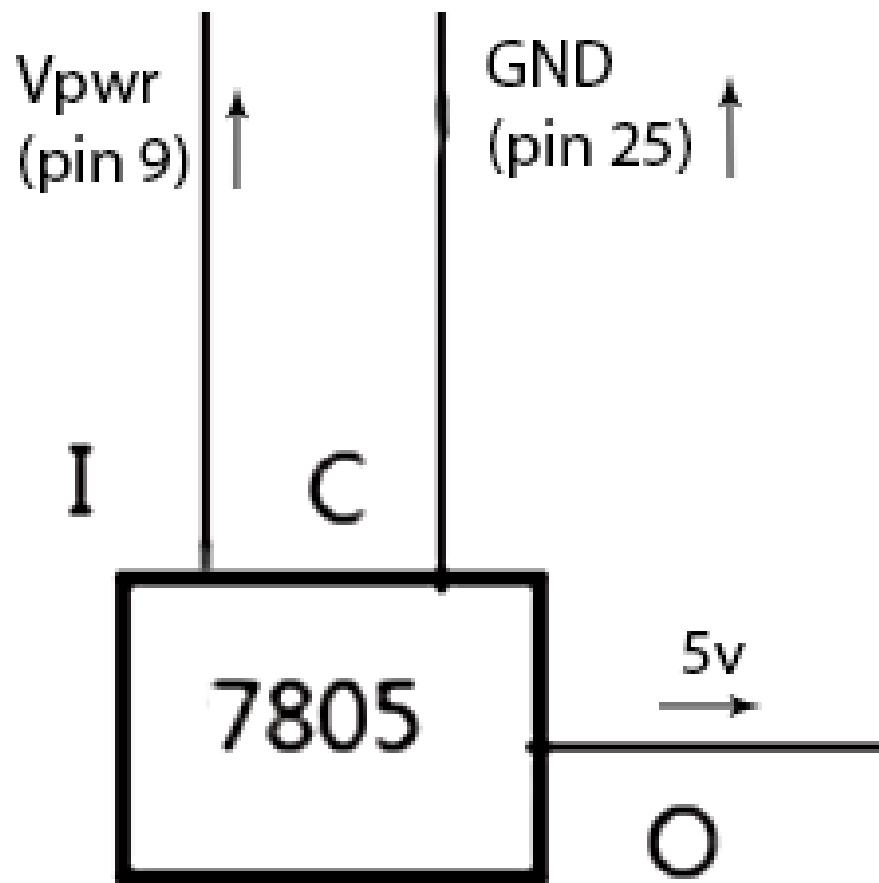
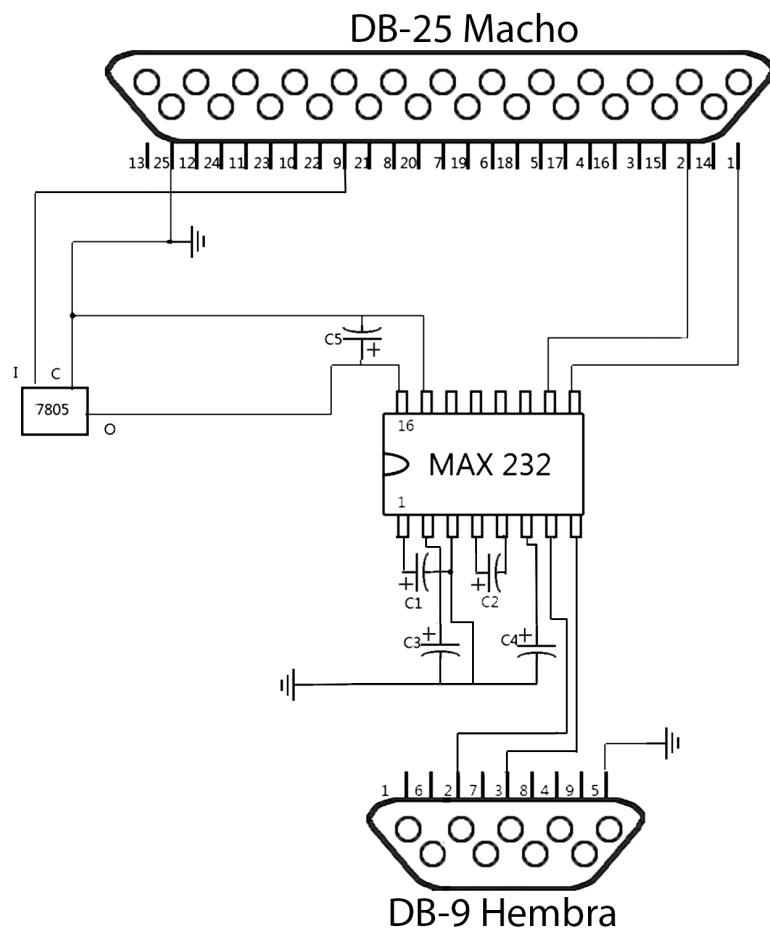


Figura 4.2: Regulación de volaje mediante un L7805CV

requeridos por el robot. En la figura 4.3 se muestran los circuitos para la conversión de voltajes y la comunicación serial. El conector DB-9 hembra se acopla a la computadora x86 y el DB-25 macho se conecta al Create®.

4.3. Circuito para la comunicación USB 2.0

En la figura 4.4 se muestra un diagrama esquemático de las conexiones eléctricas necesarias para la comunicación USB. La parte central de este sistema la conforma el microcontrolador PIC18F4550. El resistor R1 se utiliza a manera de *pull-up* para evitar la reinicialización del microcontrolador. Los resistores del R2 al R24, se utilizan como *pull-down* para forzar un nivel lógico bajo cuando los sensores externos no están conectados. Los resistores R25 y R26 disminuyen la corriente que llega a los diodos LED D1 y D2 utilizados para indicar el estado general del sistema. El oscilador XTAL1 provee el pulso de reloj principal para la ejecución del programa registrado en el microncontrolador y está además conectado a los capacitores Ca y Cb para disminuir el ruido en la señal del mismo. El capacitor Cc es un requisito de hardware establecido por el fabricante *Microchip* para la comunicación USB. Los pines 23 y 24 se conectan a las terminales D- y D+ del conector USB macho respectivamente y es a través de ellas que se transporta el flujo de datos entre el microcontrolador y la computadora x86.



*C1, C2, C3, C4, C5 = $1\mu F$

Figura 4.3: Diagrama eléctrico del circuito para la comunicación serial RS-232

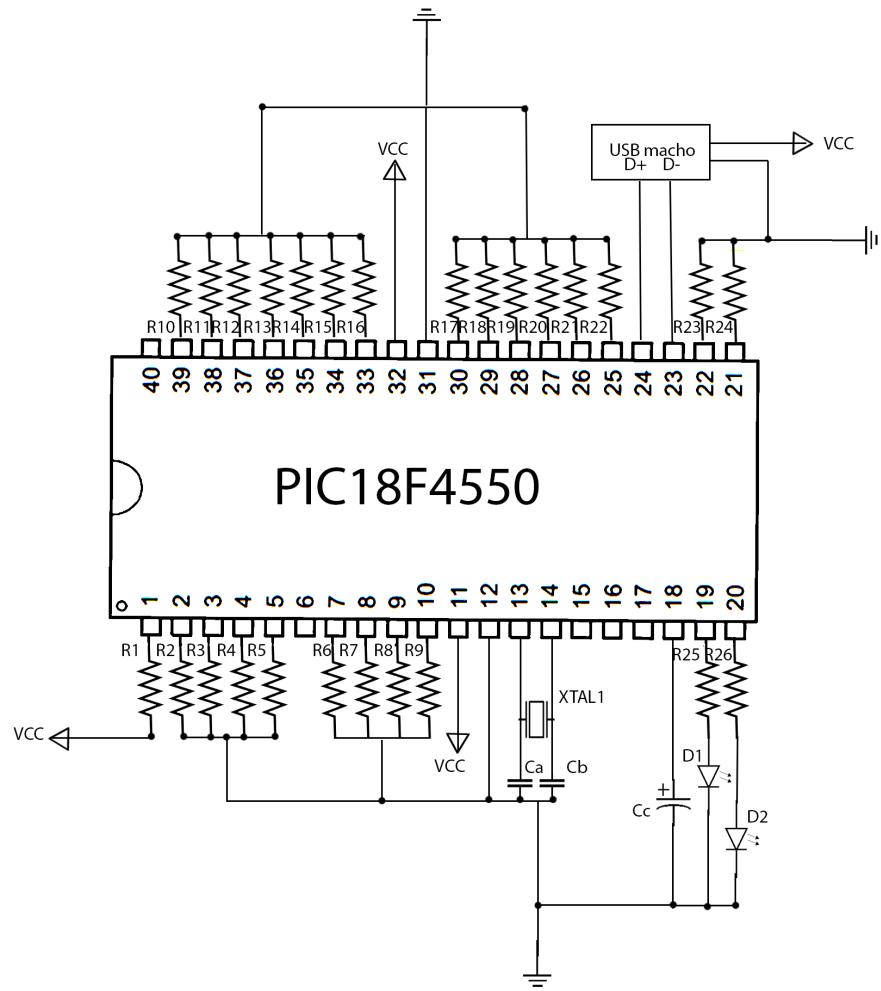


Figura 4.4: Diagrama eléctrico del circuito para la comunicación USB

Capítulo 5

Implementación de Software

El software de este proyecto se divide en dos grandes partes. Una de ellas es el firmware del microcontrolador, codificado en lenguaje c en el ambiente MPLAB y compilado con c18 de microchip. El otro es el software de la biblioteca UDG_Create programado en c/c++ y compilado en g++.

5.1. Firmware del microcontrolador

Para facilitar la implementación de dispositivos que cumplan con el estandar USB, Microchip, fabricante del microcontrolador PIC18F4550, proporciona el *MCHPF-SUSB Firmware Framework*, un conjunto de ejemplos en los que se implementan distintas funcionalidades relativas a este estándar. El firmware para este trabajo está basado en el ejemplo *USB Device - HID - Simple Custom Demo - C18 - PIC-DEM FSUSB* de dicho framework. Algunas de sus principales características se describen a continuación.

5.1.1. Configuración del oscilador para la comunicación USB 2.0

La comunicación USB de velocidad completa en el PIC18F4550 requiere utilizar un pulso de reloj de 48MHz que debe ser tomada del oscilador primario y que en general será alcanzada mediante pre y post escaladores cuyas configuraciones por software dependerán del hardware disponible.

La configuración de reloj a utilizar, para un cristal de 4MHZ es:

```

//Sin pre-escala para entrada de PLL
#pragma config PLLDIV = 1
//PLL = 96MHz /2
#pragma config CPUDIV = OSC1_PLL2
//Usar PLL/2 como reloj USB
#pragma config USBDIV = 2
//Habilitar oscilador de alta velocidad y PLL
#pragma config FOSC = HSPLL_HS

```

De esta forma obtenemos una frecuencia de 4MHz a la entrada del PLL y 48 MHz a la salida del mismo que serán utilizados como reloj para los módulos USB. En el caso particular del PIC18, durante la operación USB, otros periféricos pueden operar a velocidades menores por lo que esto no representará un problema de compatibilidad.

5.1.2. Descriptores USB

En los sistemas basados en USB, el intercambio de datos se hace a través de *endpoints* mediante mensajes llamados reportes, conformados de manera específica y cuya organización está definida en estructuras llamadas descriptores.

Los descriptores comienzan con su propio tamaño en bytes, seguidos de su tipo (necesario para ser interpretado correctamente por el *host*) y después una serie de campos específicos para cada descriptor. Por ejemplo:

```

typedef struct{ BYTE bLength; // Tamano en bytes
    BYTE bDescriptorType; // tipo
    bcdUSB; //numero de release
    BYTE bDeviceClass; //clase de dispositivo
    bDeviceSubClass; //sub-clase de dispositivo
    BYTE bDeviceProtocol; //codigo de protocolo
    BYTE bMaxPacketSize0; //Tamano maximo para EP0
    WORD idVendor; //ID de fabricante
    WORD idProduct; //ID de producto
    WORD bcdDevice; //numero de release
    BYTE iManufacturer; // indice de descriptor
    BYTE iProduct; // descriptor de producto
    BYTE iSerialNumber; //Numero de serie
    BYTE bNumConfigurations; //numero de configuraciones
}device_descriptor;

```

Definición de cadenas

Las cadenas de caracteres a las que tendrá acceso el dispositivo USB deben ser almacenadas como estructuras de datos que incluyen tamaño, tipo de estructura y los datos como tales dentro de una zona de memoria ROM determinada. Las 3 cadenas necesarias son las siguientes:

```
//Descriptor de codigo de lenguaje
ROM struct{BYTE bLength;BYTE bDscType;WORD string [1];} sd000
    ={
        sizeof(sd000),USB_DESCRIPTOR_STRING,{0x0409
    }};

//Descriptor de cadena de fabricante
ROM struct{BYTE bLength;BYTE bDscType;WORD string [25];} sd001
    ={
        sizeof(sd001),USB_DESCRIPTOR_STRING,
        {'M','i','c','r','o','c','h','i','p',' ',' ',
         'T','e','c','h','n','o','l','o','g','y',' ',' ','I','n','c','.'}
    };

//Descriptor de cadena de producto
ROM struct{BYTE bLength;BYTE bDscType;WORD string [20];} sd002
    ={
        sizeof(sd002),USB_DESCRIPTOR_STRING,
        {'C','r','e','a','t','e',' ','R','o','b','o','t',' ','S','e',
         'n','s','o','r','s'}};
    }
```

Estas cadenas son a su vez almacenadas en otra estructura de datos que las agrupa. Las referencias futuras a las cadenas (en los descriptores de dispositivo y de configuración) se hacen en relación a la siguiente definición:

```
//Arreglo de descriptores de cadena
ROM BYTE *ROM USB_SD_Ptr[]=
{
    (ROM BYTE *)sd000 ,
    (ROM BYTE *)sd001 ,
    (ROM BYTE *)sd002
};
```

Descriptor de dispositivo

Los campos identificador de fabricante (0x04D8) e identificador de dispositivo (0x003F)

en este descriptor son de especial importancia porque es a través de ellos que iniciaremos la comunicación entre el driver HID del Sistema Operativo y el microcontrolador.

```
ROM USB_DEVICE_DESCRIPTOR device_dsc=
{
    0x12, //tamano en bytes
    USB_DESCRIPTOR_DEVICE, //tipo de dispositivo
    0x0200, // version USB
    0x00, // Codigo de clase
    0x00, // codigo de subclase
    0x00, //codigo de protocolo
    USB_EP0_BUFF_SIZE, //Tamano max para EP0 (8)
    0x04D8, //ID fabricante
    0x003F, //ID producto
    0x0002, //Release dispositivo
    0x01, //cadena de fabricante
    0x02, //cadena de producto
    0x00, //numero de serie
    0x01 // posibles configuraciones
};
```

Descriptor de configuración

Define la única configuración posible para este dispositivo. Se encapsula en una sola estructura de datos toda la información necesaria para la operación USB incluyendo el protocolo de comunicación y clase de dispositivo a utilizar (HID), longitud, tipo y formato de los reportes así como la dirección, longitud y tipo de uso de los endpoints de entrada y salida.

```
ROM BYTE configDescriptor []={

    0x09, //Tamano de descriptor
    USB_DESCRIPTOR_CONFIGURATION, // Tipo de descriptor (0
    x02)
    0x29,0x00, //Longitud de datos de
    configuracion
    1, //Numero de interfaces
    1, //Indice de la configuracion
    0, //Indice de la cadena de
    configuracion
    _DEFAULT | _SELF, //Atributos ,
```

```

50,                                // max consumo de potencia (2X
mA)

/* Descriptor de Interfaz */
0x09, // sizeof(USB_INTF_DSC),      // Tamano
USB_DESCRIPTOR_INTERFACE,           // Tipo de descriptor (0x04
)
0,                                     // Numero de interfaz
0,                                     // Numero de configuracion
    alternativo
2,                                     // Numero de endpoints
HID_INTF,                            // Codigo de clase HID (0
x03)
0, // Codigo de subclase
0, // codigo de protocolo
0, // Indice de cadena

/* Descriptor de Reporte HID*/
0x09, // sizeof(USB_HID_DSC)+3,      // Tamano de descriptor
DSC_HID,                           // Tipo de descriptor (0
x21)
0x11, 0x01,                          // version de especificacion HID
(1.11)
0x00,                               // Codigo de pais (0x00 = no
soportado)
HID_NUM_OF_DSC,                      // Numero de descriptores
(1)
DSC_RPT,                            // Tipo de descriptor (0x22,
reporte)
HID_RPT01_SIZE, 0x00, //tamano de descriptor de reporte

/* Descriptor endpoint IN */
0x07, /* sizeof(USB_EP_DSC) */
USB_DESCRIPTOR_ENDPOINT,           // Descriptor de endpoint (0
x05)
HID_EP | _EP_IN,                  // Direccion de endpoint (1|0
x80)
_INTERRUPT,                        // Atributos (0x03)

```

```

0x40,0x00 ,           //Tamano
0x01 ,                //Intervalo

/* Descriptor endpoint OUT*/
0x07, //tamano de descriptor
USB_DESCRIPTOR_ENDPOINT, //Descriptor de endpoint (0x05
)
HID_EP | _EP_OUT,      // Direccion de endpoint (1|0
x00)
_INTERRUPT,            //Atributos (0x03)
0x40,0x00 ,           //Tamano
0x01                  //Intervalo
};


```

Todas las configuraciones posibles deben ser almacenadas en un arreglo. Toda referencia a estas configuraciones debe hacerse de acuerdo a la siguiente definición:

```

ROM BYTE *ROM USB_CD_Ptr []=
{
    (ROM BYTE *ROM)&configDescriptor1
};


```

5.1.3. Configuración de puertos

Los cuatro puertos de entrada/salida del PIC18F4550 son configurados mediante los registros *LATX* y *TRISX* donde *X* corresponde al identificador del puerto (A,B,C o D) y es a través de ellos que se lee/escribe su valor y se determina su dirección. Otros registros de control específicos de cada puerto definirán su funcionalidad. Para este trabajo, una parte de los puertos A y B (terminales 2, 3, 4, 5, 7, 8, 9, 10, 33, 34, 35, 36 y 37) se destinan a la conversión de señales analógicas provenientes de sensores externos a datos digitales en cuyo caso se omite la configuración del registro TRISX ya que se incluye en ADCON1.PCFGX. Los dos bits restantes del puerto B (terminales 38 y 39) se utilizan para leer las señales digitales de carga y poder del robot.

```

//B5 (RobotOn) como entrada
TRISBbits.TRISB5=1;
//B6 (RobotCharging) como entrada
TRISBbits.TRISB6=1;


```

```
//AN0 – AN12 como entradas analogicas , se omite TRISX
ADCON1bits.PCFG3=0;
ADCON1bits.PCFG2=0;
ADCON1bits.PCFG1=0;
ADCON1bits.PCFG0=0;
```

Los bits 0 y 1 del puerto D se conectan a dos diodos LED que se utilizan como indicadores de estado por lo que son configurados como pines digitales de salida. El resto de los bits del puerto D así como los bits 6 y 7 del C se usan como entradas para sensores digitales.

```
TRISDbits.TRISD0 = 0; //Salida LED
TRISDbits.TRISD1 = 0; //Salida LED
TRISDbits.TRISD2 = 1; //entrada para sensor digital
TRISDbits.TRISD3 = 1; //entrada para sensor digital
TRISDbits.TRISD4 = 1; //entrada para sensor digital
TRISDbits.TRISD5 = 1; //entrada para sensor digital
TRISDbits.TRISD6 = 1; //entrada para sensor digital
TRISDbits.TRISD7 = 1; //entrada para sensor digital

TRISCbits.TRISC6 = 1; //entrada para sensor digital
TRISCbits.TRISC7 = 1; //entrada para sensor digital
```

5.1.4. Lectura de sensores externos

La lectura de sensores externos se lleva a cabo dentro de un bucle infinito al principio del cual, mediante una operación de lectura bloqueante se esperan reportes del host USB principal. Cuando el identificador de resorte es 0x37 se inicia la lectura de los sensores externos los cuales son convertidos a información digital y empaquetados como un arreglo de bytes en el que la primer posición es un eco del identificador de reporte recibido, posteriormente se almacenan 13 pares de bytes correspondientes a la conversión de los sensores analógicos (byte mas significativo en la posición de memoria más baja) y por último 8 bits de sensores digitales. Este arreglo se envía como respuesta al host USB.

La lectura de los sensores analógicos requiere un proceso de conversión digital. Los registros asociados a la configuración del convertidor analógico-digital del microcontrolador son ADRESH y ADRESL que almacenan el resultado de la conversión y los registros de control ADCON0, ADCON1 y ADCON2.

Para la configuración inicial del registro ADCON0 solamente fijaremos en alto el bit 0, correspondiente a la habilitación/deshabilitación del módulo de conversión. El resto de los bits (selección de canal e inicio de conversión) se establecerán en el momento de iniciar una conversión nueva.

Los bits 4 y 5 del registro ADCON1 (VCFG1:VCFG0) indican el voltaje analógico de referencia para la conversión, los cuales serán 0 para que se utilicen los voltajes VSS y VDD. El resto de los bits de este registro (PCFG3:PCFG0) se fija a 0 durante la configuración de puertos.

El bit 7 del registro ADCON2 es un bit de justificación, esto es, especifica el desplazamiento del resultado de 13 bits dentro del registro de 16 bits que forman en conjunto ADRESH y ADRESL de manera que, de ser necesario se desprecien los bits menos significativos o viceversa. Debido a que los bytes alto y bajo del resultado de la conversión se tratarán por separado, se eligió la justificación a la derecha, es decir, el bit ADFM en nivel alto. Los bits 3, 4 y 5 corresponden al tiempo de adquisición mínimo para la conversión, el tiempo que se debe esperar para que el capacitor CHOLD se cargue hasta el voltaje de entrada. Ya que desconocemos a priori la impedancia total que tendrán los sensores, asignamos el valor más alto posible ($20T_{AD}$) a fin de no perder precisión en las mediciones. Por último, los bits 0, 1 y 2 nos permiten elegir la fuente de reloj para el módulo, que será $F_{OSC}/4$.

```
ADCON0=0x01 ;
ADCON1=0x00 ;
ADCON2=0x3C ;
ADCON2bits.ADFM = 1;
```

Para llevar a cabo la conversión, con la ayuda de un contador controlado por un ciclo for, modificamos los bits de selección de canal CHS3:CHS0 en el registro ADCON0 y fijamos el bit GO a 1. Posteriormente debemos esperar al cambio de estado del bit GO indicando que la conversión ha finalizado. Entonces almacenamos el contenido de ADRESH y ADRESL en posiciones contiguas del buffer de salida y continuamos con la siguiente conversión.

```
unsigned char i ;
ToSendDataBuffer [0] = 0x37; //Eco
for (i=0;i<13;i++)
{
    ADCON0=(i<<2)+1; //cambio de canal
    ADCON0bits.GO = 1; //inicio de conversion
    while(ADCON0bits.GO); //esperar fin de conversion
```

```

    ToSendDataBuffer [2*i+1] = ADRESL;           //LSB de
    resultado
    ToSendDataBuffer [2*i+2] = ADRESH;           //MSB de
    resultado
}

```

Debido a que los bits asignados a la lectura de sensores digitales no son contiguos y se encuentran de hecho distribuidos en más de un puerto, estos deben ser analizados bit por bit y, en caso de existir un nivel alto de voltaje en ellos, asignar el valor correspondiente al dicho bit mediante un operador OR a una variable originalmente inicializada en 0.

```

ToSendDataBuffer [27] = 0x00;

if (PORTDbits.RD7==1)
{
    ToSendDataBuffer [27]|=0x80;
}
if (PORTDbits.RD6==1)
{
    ToSendDataBuffer [27]|=0x40;
}
if (PORTDbits.RD5==1)
{
    ToSendDataBuffer [27]|=0x20;
}
if (PORTDbits.RD4==1)
{
    ToSendDataBuffer [27]|=0x10;
}
if (PORTCbits.RC7==1)
{
    ToSendDataBuffer [27]|=0x08;
}
if (PORTCbits.RC6==1)
{
    ToSendDataBuffer [27]|=0x04;
}
if (PORTDbits.RD3==1)
{
    ToSendDataBuffer [27]|=0x02;
}

```

```

{
    ToSendDataBuffer [27] |=0x02 ;
}
if (PORTDbits.RD2==1)
{
    ToSendDataBuffer [27] |=0x01 ;
}

```

5.2. Interacción con el driver HID

Como se menciono anteriormente, una de las ventajas de desarrollar dispositivos USB de la clase HID, es el hecho de que la mayoría de los sistemas operativos modernos poseen por defecto un driver compatible. Sin embargo, es necesaria la creación de un software de nivel intermedio que permita la interacción entre el programa de usuario y el driver a nivel de kernel. En Ubuntu y la mayoría de los sistemas basados en Linux esto es posible gracias a la API definida en el archivo de encabezado linux/hiddev.h.

5.3. Comunicación mediante el estándar RS-232

Enviar instrucciones al iRobot Create[®] implica escribir códigos de operación a través del puerto serial de la computadora x86 siguiendo la sintaxis establecida en la *Create Open Interface*. La biblioteca UDG_Create proporciona una interfaz de alto nivel que permite a los desarrolladores escribir código en c++ sin preocuparse por especificaciones eléctricas o intercambio de datos a bajo nivel. Las siguientes secciones proporcionan un panorama general del intercambio de instrucciones y datos entre el Create[®] y la computadora.

5.4. Comunicación serial en Linux

Las operaciones de lectura y escritura del puerto serial, por estar fuertemente ligadas al hardware, son dependientes del sistema operativo. Para la implementación de la biblioteca UDG_Create, basada en la distribución de linux Ubuntu 12.10, se utiliza la API de UNIX definida en el archivo de encabezado termios.h y los tipos de datos declarados en fcntl.h.

Tal como es costumbre en sistemas basados en UNIX, estas operaciones se ejecutan mediante las instrucciones `read()` y `write()` operadas sobre un archivo de enlace simbólico asociado con el respectivo puerto serial, al que se hace referencia mediante una variable entera que funge como descriptor de archivo. La apertura de este descriptor se efectua mediante:

```
int portDescriptor = open(portName, O_RDWR | O_NOCTTY | O_NDELAY);
```

donde `O_RDWR`, `O_NOCTTY` y `O_NDELAY` indican la apertura en modo de lectura y escritura, la no asignación del puerto como terminal de control del proceso y el establecimiento de operaciones no bloqueantes respectivamente. La variable `portName` es una cadena de caracteres que contiene el nombre del enlace simbólico a utilizar, el cual debe ser proporcionado por el usuario.

La biblioteca UDG_Create proporciona además un modo de apertura automático en el que se itera a través de los primeros 50 enlaces simbólicos con prefijo `ttyS` (`ttyS1`, `ttyS1`, `ttyS2`, etc) comunmente asociados al puerto RS-232 y los primeros 10 enlaces con prefijo `ttyUSB` utilizados para la comunicación serial mediante adaptadores USB en aquellos sistemas que carecen de un conector DB-9 macho.

El resto de las configuraciones se efectuan utilizando la estructura llamada `termios` como se muestra en la siguiente porción de código:

```
struct termios configuration;
//ignorar break, deshabilitar traducciones entre CR y NL,
deshabilitar bits de paridad, deshabilitar control de flujo
configuration.c_iflag &= ~(IGNBRK | BRKINT | ICRNL |
                           INLCR | PARMRK | INPCK | ISTRIP | IXON);
configuration.c_oflag=0;
//deshabilitar echo, modo canonico, procesamiento
//de entradas personalizado
configuration.c_lflag &= ~(ECHO | ECHONL | ICANON | IEXTEN |
                           ISIG);
//fijar mascara de caracteres de 8 bits
configuration.c_cflag &= ~(CSIZE | PARENB);
configuration.c_cflag |= CS8;
//read() se considera terminado despues de 1 caracter
configuration.c_cc[VMIN] = 1;
//read() se considera terminado despues de 0 decimas de
//segundo
configuration.c_cc[VTIME] = 0;
```

```
//fijar baudrate de 57600
cfsetispeed(&configuration, B57600) < 0 ||
cfsetospeed(&configuration, B57600);
//aplicar cambios de manera inmediata
tcsetattr(portDescriptor, TCSANOW, &configuration);
```

Una vez realizadas estas configuraciones, leer y escribir al puerto serial se logra mediante:

```
write(portDescriptor, buffer, numberOfBytes);
read(portDescriptor, buffer, numberOfBytes);
```

5.4.1. Formato básico de instrucción

Como se mencionó anteriormente, para la Create Open Interface, una instrucción consiste en un flujo de datos que inicia con un código de operación correspondiente a una instrucción y es seguido, de requerirlo, por un número determinado de bytes de datos, que deben ser escritos a través de un puerto serial RS-232.

```
CodigoOp [ DatosByte1      DatosByte2  DatosByte3 ...  DatosByteN ]
tal como se ejemplifica en el siguiente pseudocódigo para una instrucción genérica
TIPO.RETORNO Instruccion(BYTE datos1, DWORD datos2 )
{
    BYTE buffer[3]  <-  [ CODIGO_OP, datos1, datos2.msb,
                           datos2.lsb ]
    Escribir_Serial( PUERTO, buffer, size(buffer) )
}
```

Cabe resaltar que independientemente del tipo de datos de los parámetros que pudieran recibirse como argumentos a alto nivel, estos siempre serán divididos y enviados como bytes independientes al robot junto con el resto de los datos y el código de operación.

Un claro ejemplo de ello es la función waitDistance() que indica al robot que debe esperar hasta que cierta distancia, definida por la variable entera *distance*, sea recorrida antes de recibir otra instrucción. Su implementación en lenguaje c++ es la siguiente:

```
void Robot::waitDistance(int distance)
{
```

```

if(modo == FULL || modo == SAFE || modo == PASSIVE)
{
    int16 data1 = toInt16(distance);
    unsigned char ins[3]{0x9C,data1.H,data1.L};
    WriteToSerial(descriptorPuerto ,ins ,3 );
}
else
    error(MODO_INVALIDO_INSTRUCCION,(char*)"
        Funcion WAITDISTANCE");
}

```

Podemos apreciar como el código de operación correspondiente a la operación Wait Distance de la Create Open Interface, es enviado al puerto serial al principio del buffer de escritura seguida por los dos bytes que en conjunto conforman un entero de 16 bits representando la distancia a esperar.

5.4.2. Instrucciones de lectura

Algunas instrucciones (por ejemplo, las consultas de los sensores integrados) requieren que la computadora, además de enviar la instrucción correspondiente, espere una respuesta por parte del robot, la cual debe ser recibida e interpretada adecuadamente. El pseudocódigo para una típica instrucción de lectura es:

```

TIPORETORNO instrucion_lectura( REF INTEGER sensor )
{
    BYTE[2] instrucion <- {CODIGO_OP, datos}
    Escribir_Serial(PUERTO, instrucion ,2)
    BYTE datos[N]
    Leer_Serial(PUERTO, datos , N)
    Ajustar_Orden_bytes( datos , N, sensor , SIGNO )
}

```

Puede observarse como la respuesta del robot debe ser leída, almacenada en un bufer intermedio y despues ajustada a las convenciones de ordenamiento de bytes y tipos de datos de la arquitectura x86. La función encargada de ello (de la cual se habla a detalle en 5.4.4) es Ajustar_Orden_bytes().

Para ejemplificar este tipo de instrucción el siguiente segmento de código muestra el procedimiento para la lectura de paquetes de los sensores integrados del Create®. La

implementación final en c++, para facilitar la reutilización de código, está subdividida en 3 funciones:

```

int Create::sensors(unsigned char idPacket)
{
    if( idPacket >=0 && idPacket <=42)
    {
        unsigned char ins [2]{0x8E,idPacket };
        WriteToSerial( descriptorPuerto ,ins ,2) ;
        int sizePacket = getSizePacket(idPacket) ;
        return actualizaSensor(idPacket ,sizePacket) ;
    }
    else
        error(FUERA.RANGO,( char*)" Funcion SENSORS" );
}

int Create::actualizaSensor(unsigned char idPaquete , int
                           sizeInstruccion)
{
    unsigned char ins2[sizeInstruccion];
    usleep(120000);
    ReadFromSerial(descriptorPuerto ,ins2 ,sizeInstruccion
                   );
    int ret = -65536;
    int offset=0;
    switch(idPaquete)
    {
        //...
        case 8:
            ret = readWall(ins2+offset);
            if( idPaquete < 7)
            {
                offset+= getSizePacket(8);
            }
            if( idPaquete == 8)
                break;
        //...
    }
}

```

```

    return ret;
}

int Create::readWall(unsigned char *ins2)
{
    wall = ins2[0];
    int retValue;
    return toLittleEndian(ins2,1,&retValue,UNSIGNED);
}

```

La función Create::sensors() se encarga de escribir el código de operacion correspondiente a la lectura de paquetes y el identificador del sensor a leer. Esta a su vez llama a la función actualizaSensor(), la cual recibe como parametros el ID del paquete de sensores (puede ser más de uno) y el numero de bytes que se espera recibir como respuesta. En esta función se recibe un arreglo de bytes proveniente del robot mediante ReadFromSerial() que representa el valor del sensor solicitado. Por ultimo este arreglo de bytes se opera mediante una función específica de cada sensor (en el ejemplo readWall()) en la que se convierten a tipos de datos nativos de c++ y se realizan los ajustes de orden de bytes, signo y tamaño de datos mediante toLittleEndian().

5.4.3. Instrucciones de streaming

Para mejorar el desempeño de ambientes con capacidades limitadas de procesamiento en tiempo real (por ejemplo, aquellos basados en redes inalámbricas) el Create® es capaz de enviar paquetes con datos provenientes de sus sensores cada 15 milisegundos. El manejo correcto de esta información supone ciertas dificultades que pueden ser abordadas siguiendo tres pasos fundamentales. El primero de ellos es cambiar el estado de la bandera lógica encargada de controlar el streaming de los datos. Esta operación es propia del robot Create® y se describe en la siguiente función.

```

void Create::pauseResumeStream(bool streamState)
{
    if(mode == PASSIVE || mode == SAFE || mode == FULL)
    {
        unsigned char ins[2] = {0x96,streamState};
        WriteToSerial(portDescriptor,ins,2);
    }
}

```

```

        streamingState = streamState;
    }
else
    error (INVALID_INSTRUCTION_MODE, ( char *)"
        Function PAUSERESUMESTREAM" );
}

```

De acuerdo a las especificaciones de la Create Open Interface, el contenido del flujo de bytes que se recibe como respuesta dependerá de la información solicitada por el usuario, por ello deberá ser cuidadosamente manejada e interpretada. La inicialización del flujo y el nuevo hilo puede hacerse como sigue:

```

void Create :: stream (unsigned char* destinationBuffer ,void*
thread ,int n,...)
{
    std :: thread *t =(std :: thread *) t ;
    va_list args ;
    va_start (args ,n) ;
    int bytesToRead=3+n;
    //Probar funcion 4/7
    if (n>=0 && n<=43)
    {
        if (mode==PASSIVE || mode == SAFE || mode == FULL)
        {
            unsigned char* ins = new unsigned char [n+2];
            ins [0] = 0x94 ;
            ins [1] =n;

            for (int i = 2;i<n+2;i++)
            {
                ins [i] = (unsigned char) va_arg (args ,
                    int);
                bytesToRead+=getSizePacket (ins [i]) ;
            }
            WriteToSerial (portDescriptor ,ins ,n+2);
            streamingState = true;
            //Read packets
            sleep (1);
        }
    }
}

```

```

        *t= std :: thread ( ThreadedReadStream ,
            portDescriptor , this , destinationBuffer ,
            bytesToRead) ;
                //t . join () ;
}
else
{
    error (INVALID_INSTRUCTION_MODE, ( char* )"
        Function STREAM" );
}
else
{
    error (OUT_OF_RANGE, ( char* )" Function STREAM" );
}
}

```

Ya que en general se recomienda que las operaciones de lectura a puerto serial sean bloqueantes, lo mejor es manejar el flujo de bytes en un hilo de ejecución separado sacando ventaja de las capacidades multi-hilo en los lenguajes de programación, procesadores y sistemas operativos modernos. La interpretación de los datos recibidos debe hacerse en el cuerpo del nuevo hilo:

```

void ThreadedReadStream( int portDescriptor , Create* r ,
    unsigned char* buffer , int numberofBytes)
{
    while( r->getStreamingState() )
    {
        usleep(15000);
        ReadFromSerial( portDescriptor , buffer ,
            numberofBytes);
    }
}

```

5.4.4. Ordenamiento de bytes y tipos de datos

Una de las mayores dificultades que surgen al intentar controlar este robot mediante una computadora x86 es lidiar con las diferencias entre las maneras de representar los datos como se ilustra en la figura 5.1. El iRobot Create® opera usando enteros con y sin signo de 8 y 16 bits recibidos uno a uno en orden *Big Endian*, esto es, con el byte más significativo primero. Por otro lado, la arquitectura x86 utiliza *Little*

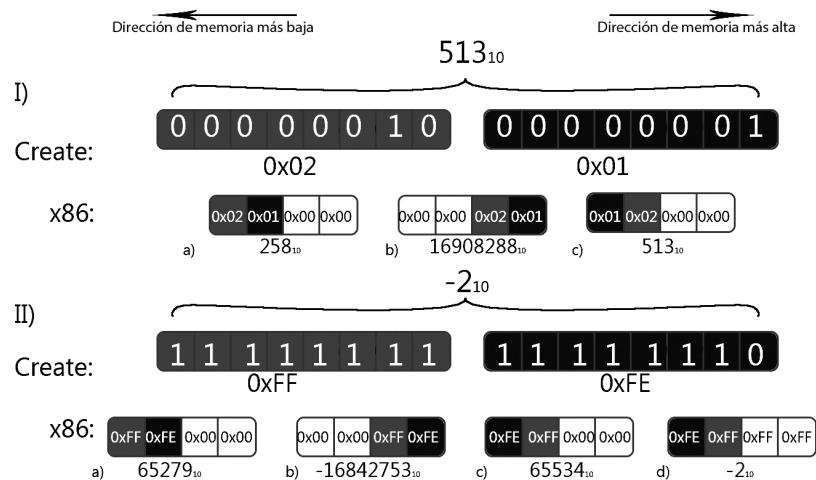


Figura 5.1: Importancia del correcto manejo de signos y orden de bytes. El ejemplo I) muestra algunas de las posibles interpretaciones para un entero de 16 bits sin signo, solo el inciso c) es correcto. En el ejemplo II) se muestran posibles interpretaciones para un entero negativo, solo el inciso d) es correcto

Endian, es decir que los bytes menos significativos se encuentran en las posiciones de memoria más bajas, con tamaños de datos variables dependiendo del lenguaje de programación, capacidades de hardware y compilador utilizado. De ahí la necesidad de utilizar una función que permita hacer ajustes en las representaciones en memoria de los datos de manera transparente al programador y sin perder portabilidad. Para ello se propone la siguiente función:

```

int Create :: toLittleEndian( unsigned char* source , int nbytes
    , int* destination , boolSigned sign )
{
    char* tmp;

    if( sign == UNSIGNED)
    {
        *destination = 0;
    }
    else
    {
        if( source[0]&0x80) //if it is negative (2's
            complement)
        {
            *destination = -1;//0xFFFFFFFF... only
            1's
        }
        else
        {
            *destination = 0;
        }
    }

    tmp = (char*) destination;
    for(int i =nbytes -1; i>=0;i--)
    {
        *(tmp+(( nbytes-1) - i )) = source [ i ];
    }
}

```

```
return *destination;  
}
```

En el código anterior, la variable *source* es un apuntador al arreglo de bytes que contiene la información proveniente del robot, *nbytes* el número de bytes que deben ser considerados como parte del valor, *destination* es un apuntador a la variable entera donde será almacenado de manera definitiva el resultado y *sign* una variable lógica que nos indica si *source* debe ser considerado una cifra con o sin signo.

El primer paso es establecer el valor inicial de la variable *destination* que dependerá del signo esperado del resultado. En caso de tratarse de un número sin signo, el valor de *destination* se inicializa a 0. Si es un número con signo debe verificarse el bit más significativo del byte más significativo en el arreglo *source* que representa el bit de signo en ambas arquitecturas, mediante la operación *source[0] & 0x80*. Cuando este bit es 0, significa que el número es positivo y el valor de *destination* debe inicializarse a 0. En caso contrario, cuando el número es negativo, dado que ambas arquitecturas utilizan complemento a 2 para la representación de números negativos, el valor de *destination* se inicializa a -1, de tal manera que todos sus bits sean 1 independientemente del tamaño en bytes asignado a las variables enteras por el compilador/sistema operativo.

El siguiente paso es asignar un apuntador a un tipo de dato de un byte de longitud (*unsigned char*) a la misma dirección de memoria que *destination*. Por último, con la ayuda de un ciclo *for* se invierte el orden de los bytes en *source* y se almacenan en *destination*.

Capítulo 6

Pruebas y Análisis de resultados

A fin de medir las capacidades y limitaciones de la biblioteca UDG_Create, se diseñó un experimento para compararlo directamente contra el Qbot de Quanser. El programa creado para dicha prueba utiliza los sensores en el parachoques del Create® para detectar objetos en su camino y esquivarlos. La ejecución inicia con el robot avanzando en linea recta hacia el frente hasta que un objeto sea detectado, si el parachoques registra un impacto por el lado izquierdo, girará aproximadamente 38° hacia la derecha. De manera similar, cuando el impacto ocurre por el lado derecho, el giro se efectuará hacia el lado izquierdo. En el caso de un choque frontal, la dirección del giro se determina de forma aleatoria.

Ambos robots fueron colocados en un recorrido con obstáculos con las mismas posiciones de inicio de manera que nos permitiera obtener una evaluación cualitativa de su comportamiento bajo las mismas condiciones de operación (ver figura 6.1). El Qbot fué programado utilizando simulink y sus respectivos bloques de control de Quanser, mientras que el software del Create® fue codificado en c++ utilizando las instrucciones de la biblioteca UDG_Create (ver 6.3).

El comportamiento de ambos robots durante la prueba fue prácticamente idéntico, sin embargo, algunas diferencias significativas tanto cualitativas como cuantitativas (que se resumen en 6.4) fueron observadas. En el caso del Qbot, para esta aplicación debido a la complejidad de los protocolos de comunicación inalámbricos y el manejo de sistemas en tiempo real por parte de simulink, una gran cantidad de archivos deben ser generados y copiados a la tarjeta Gumstix integrada del robot para ser compilados en ella. Este proceso incluyendo la compilación tomó un promedio de 65 segundos (porogramas más complejos desarrollados en el laboratorio toman hasta



Figura 6.1: Comparación entre distintos modelos de programación. a) iRobot Create® y UDG_Create. b) Quanser Qbot y Simulink

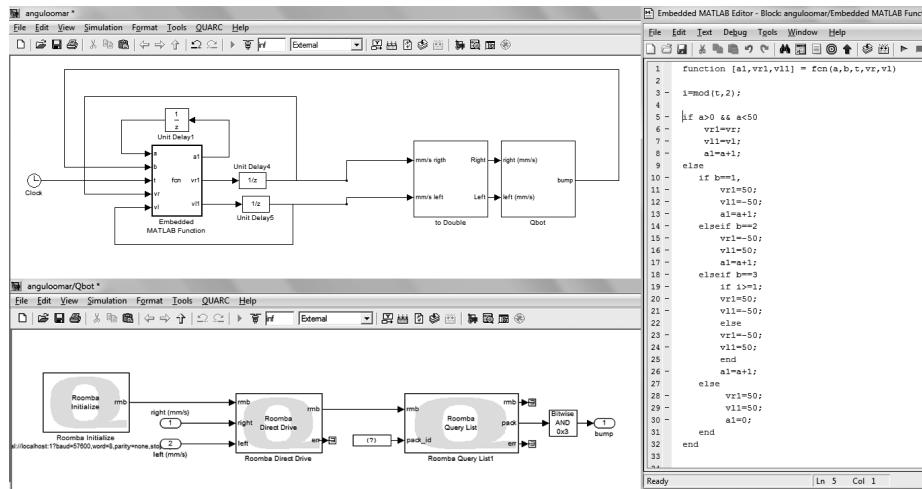


Figura 6.2: Ambiente de programación y ejecución del Qbot con Simulink

```

root@omar-ubuntu:/home/omar/Desktop/robotLibraryRelease# ./main
Trying to open /dev/ttyUSB0
Serial port successfully opened
serial port successfully configured
successfully started libusb session
Error initializing device handle
USB external sensors not enabled!
Going straight
Turning randomly
Turning clockwise
Turning anti-clockwise
Turning clockwise

```

Figura 6.3: Ambiente de programación y ejecución para el Create® con UDG_Create

30 minutos), es decir, 6500 % más lento que la implementación en c++.

En cuanto al desempeño, una de las principales ventajas de utilizar lenguajes compilados como c++ en lugar de interpretados como matlab/simulink, es su potencial para ejecutarse hasta 500 veces más rápido (comparado con código de matlab/simulink puro)[33]. Esta disimilitud crece aun más cuando, a diferencia del Qbot, se proporciona un enlace cableado entre el robot y la PC. Además, al utilizar software libre para el desarrollo, se evita la compra de las licencias de Matlab/Simulink y QuaRC que pueden llegar a ser muy costosas. Por último, los tiempos de desarrollo y depuración se pueden reducir drásticamente gracias a la familiaridad generalizada de los alumnos con la arquitectura y metodologías de programación y el uso de una biblioteca amigable con el usuario con una curva de aprendizaje no muy pronunciada.

	Quanser Qbot	iRobot Create
Tiempo de compilación promedio(segundos)	65	1
Dependencias	Matlab, Simulink, QuaRC, Roomba simulink blocks, HIL, Microsoft c++ compiler	compilador c++11 (g++)
Numero de archivos fuente	28	1
Tamaño del ejecutable	82 KB	172.2 KB
Sistema operativo a bordo	Open Embedded Linux	Ubuntu 13.10 64 bits
Sistema operativo de PC	Windows XP 32 bits	
Arquitectura de CPU a bordo	ARM	x86
Arquitectura de CPU de PC	x86	
Hardware a bordo	Tarjeta Gumstix Verdex XL6P , 600 MHz, 128MB RAM, 32MB Flash	Intel core i3-3217U CPU, 1.80GHz × 4 , 4GB RAM
Hardware de PC	Intel core i7-3517U, 1.90 GHz × 8, 8GB RAM	

Figura 6.4: Comparación de configuración entre los robots Qbot y Create®

Capítulo 7

Manual de Usuario

7.1. La placa fenólica

Las figuras 7.1 y 7.2 muestran respectivamente una visión esquemática y física de la placa fenólica para la conexión del robot y los sensores externos con la PC. Posee 13 terminales para sensores analógicos con salidas de 0 a 5v, 8 bits para sensores digitales, un conector DB-9 para comunicación serial con la PC mediante el estándar RS-232, un conector USB 2.0 para la lectura de los sensores externos desde la PC y un conector DB-25 para el intercambio de información con el Create® desde el conector de su bahía de carga.

Para utilizar la biblioteca UDG_Create, la conexión USB es opcional. Cuando no se requiera el uso de sensores externos es posible sustituir la placa fenólica por el cable serial propio del Create® o cualquier otro circuito de conversión TTL/RS-232.

7.2. Conexiones físicas

Para la correcta ejecución de los programas creados con la biblioteca UDG_Create es necesario realizar algunas conexiones físicas entre los distintos componentes de hardware.

1. Conecte el cable DB-25 macho de la placa fenólica al puerto DB-25 hembra en la bahía de carga del Create® como se muestra en la figura 7.3.
2. Conecte un cable USB al puerto correspondiente de la placa fenólica como se muestra en la figura 7.4.
3. Conecte un cable serial al conector DB-9 de la placa fenólica como se ilustra en la

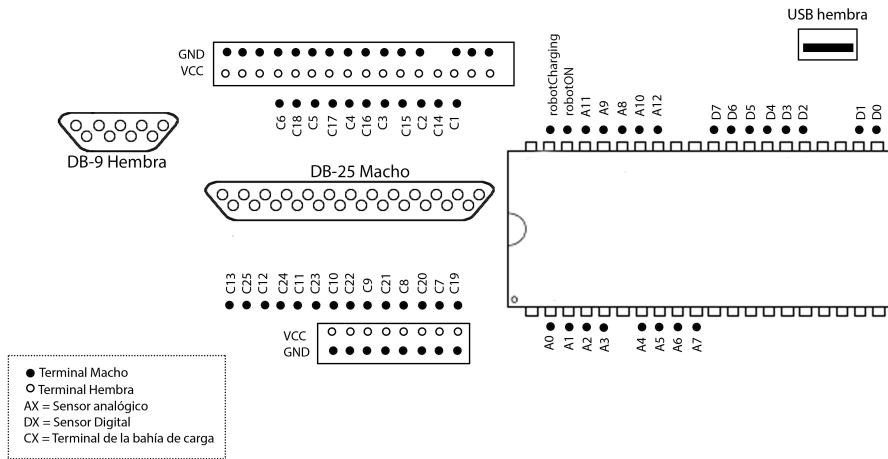


Figura 7.1: Esquema de las terminales en la placa fenólica

figura 7.5. Es posible utilizar un convertidor serial/usb si su PC no posee un puerto serial.

4. Para conectar sensores externos, tanto analógicos como digitales, se recomienda que estos tengan terminales acopladas a las puntas. Por seguridad, la terminal VCC del sensor debe ser macho y el resto hembras (ver figura 7.6). Conecte las terminales de alimentación de los sensores a la placa fenólica. El identificador de sensor en el software será interpretado de acuerdo a la posición de la conexión física de la terminal de señal. Esto se ilustra en la figura 7.7.

5. Conecte el otro extremo de los cables serial y USB a la PC (ver figura 7.8).

6. Una vez realizadas estas conexiones, el robot está listo para usarse. Para aprovechar al máximos las capacidades de la biblioteca UDG_Create se recomienda utilizar una computadora portátil y montarla junto con los sensores sobre el robot como se muestra en la figura 7.9, sin embargo, otras técnicas como la utilización de hardware intermedio para la comunicación Wi-Fi o Zigbee son también posibles.

7.3. Dependencias de software

Para utilizar la biblioteca UDG_Create es necesario contar con un sistema operativo basado en linux para la arquitectura x86. Durante la ejecución se requieren permisos de root para la lectura de los sensores externos USB. Para recompilar la biblioteca es necesario contar con un compilador que cumpla con el estándar c++

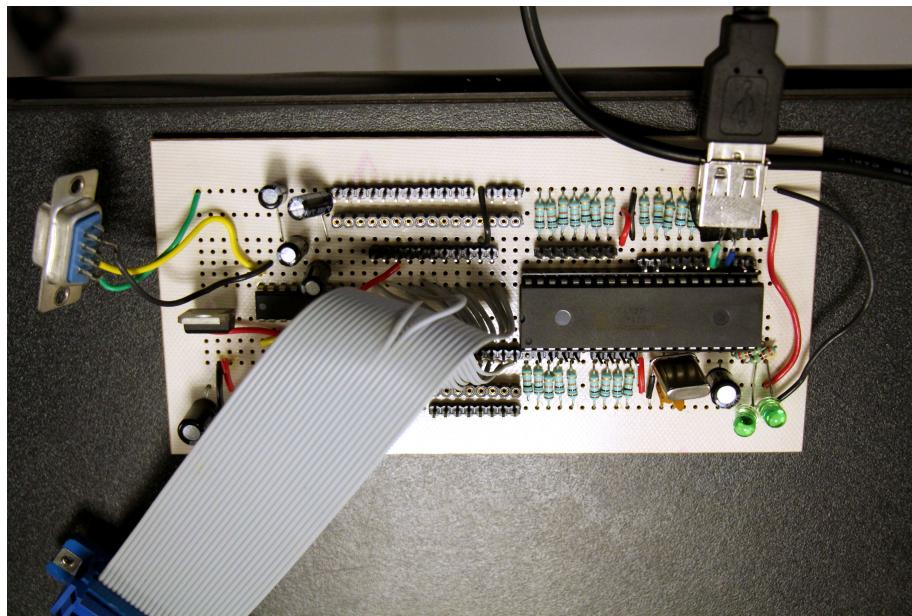


Figura 7.2: Placa fenólica para el uso de UDG_Create

11. En esta guía de usuario se utilizan Ubuntu 13.10 y g++ 4.7.

7.3.1. Recompilación de la biblioteca con g++

La biblioteca UDG_Create se distribuye como una paquete precompilado. Si necesita recompilarla puede hacerlo de manera automática ejecutando el script *rebuildLibrary.sh* contenido en la carpeta del código fuente. Se recomienda utilizar los siguientes comandos:

```
cd <directorio_del_codigo_fuente>
sh rebuildLibrary.sh
```

Para compilaciones personalizadas se requiere llevar a cabo los siguientes pasos:
Compilar los archivos fuente (sin ligado).

```
g++ -c -std=c++11 USBLayer.cpp ExternalSensors.cpp UDG_Create.cpp SerialPort.c1
```

Empaqueando del código objeto.

```
ar rvs UDG_Create.a ExternalSensors.o UDG_Create.o SerialPort.o USBLayer.o
```

¹Para versiones de g++ 4.3 y posteriores pero inferiores a 4.7 utilizar -std=c++0x en lugar de -std=c++11

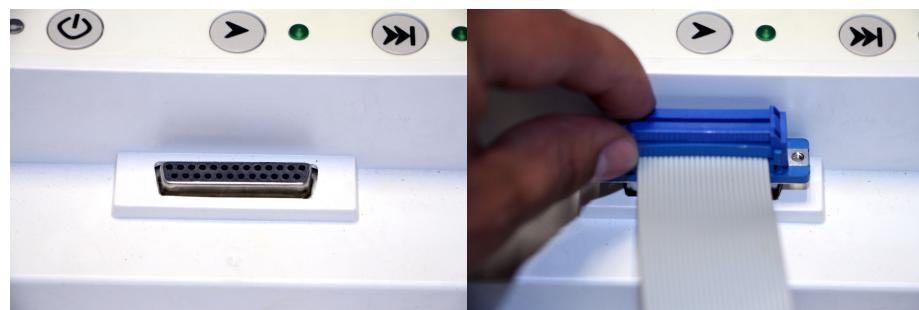


Figura 7.3: Conector DB-25 en la bahía de carga del Create®

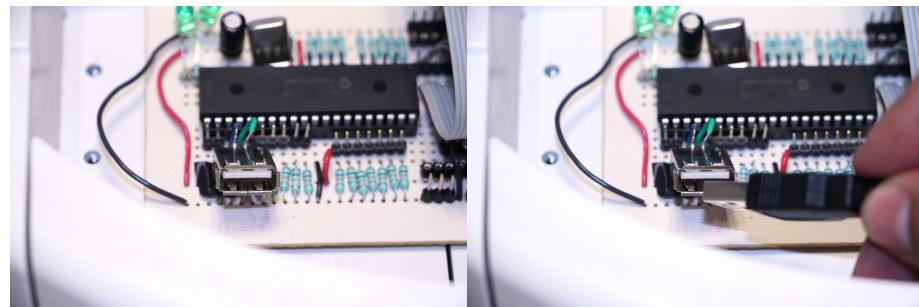


Figura 7.4: Conexión de los sensores externos mediante un cable USB

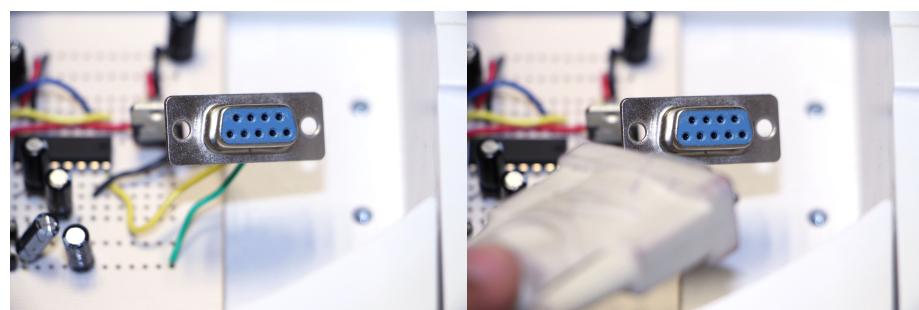


Figura 7.5: Conector DB-9 para la comunicación serial

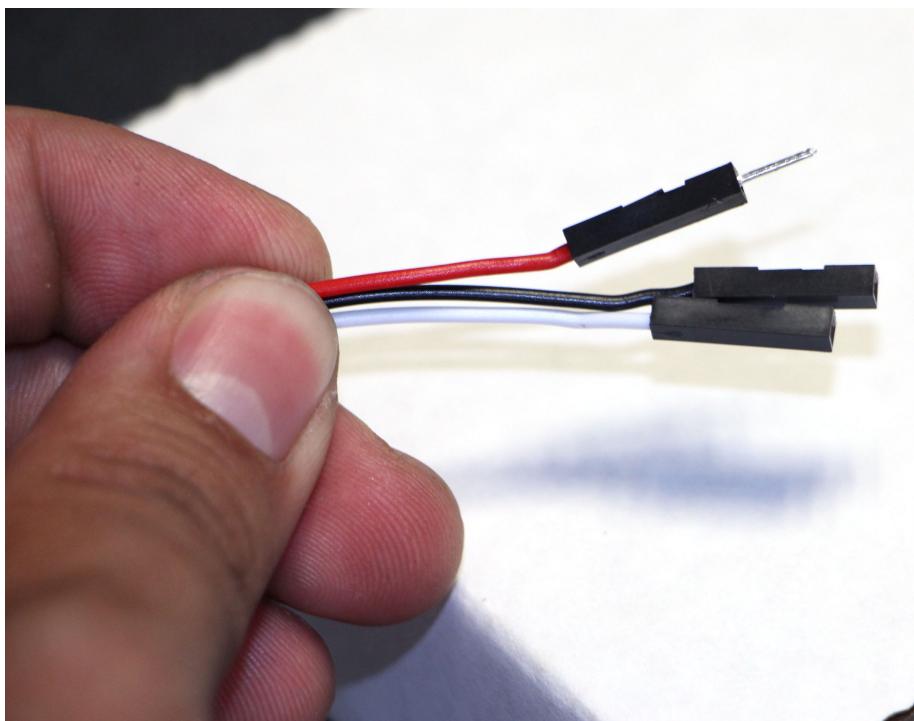


Figura 7.6: Terminales para cables recomendadas

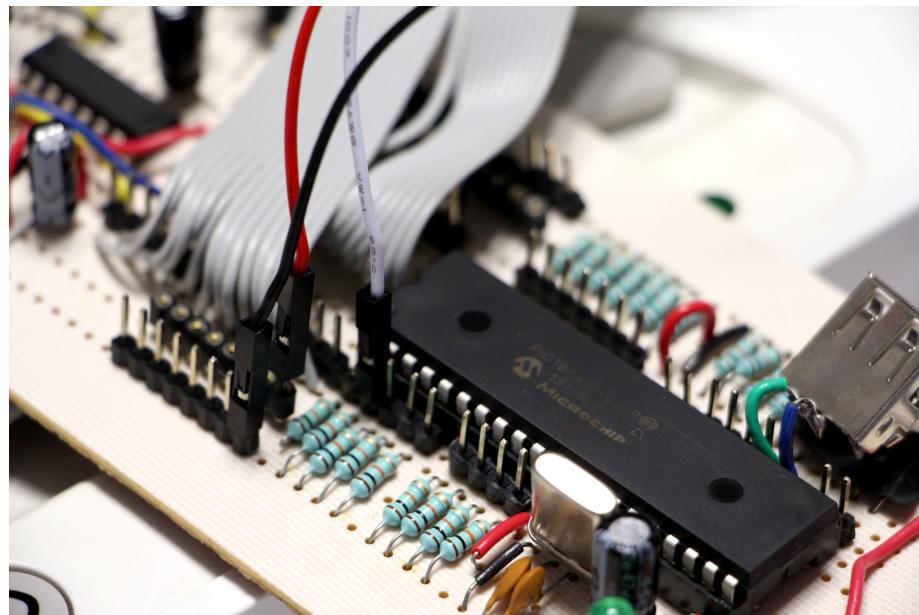


Figura 7.7: Conexión de sensores externos



Figura 7.8: Conexión de la placa fenólica a la PC



Figura 7.9: Modo de uso recomendado

De manera opcional, para utilizar UDG_Create.h como archivo de encabezado estandar debe agregar el prefijo lib al nombre del paquete.

```
ar rvs libUDG_Create.a ExternalSensors.o UDG_Create.o SerialPort.o USBLayer.o
```

Por ultimo, si asi lo desea, puede eliminar los archivos intermedios generados durante el proceso.

```
rm *.o
```

7.4. Uso de UDG_Create de manera local

Copie los archivos UDG_Create.h y UDG_Create.a al directorio del código fuente de su proyecto.

Asegurese de que su programa incluya a la biblioteca como `#include "UDG_Create.h"` (con comillas).

Incluya UDG_Create.a a su comando de compilación para realizar el ligado utilizando la bilioteca. Por ejemplo, para el programa:

Listing 7.1: main.cpp

```
#include "UDG_Create.h"
int main()
{
    Create robot;
    robot.demo(9);
    return 0;
}
```

Puede utilizar:

g++ main.cpp UDG_Create.a

7.5. Uso de UDG_Create como biblioteca estándar

Copiar UDG_Create.h a uno de los directorios por defecto para archivos de encabezado (por ejemplo /usr/include/) o usar la opción -I de g++ para especificar la ruta al momento de compilar.

Copiar libUDG_Create.a a uno de los directorios por defecto de c++ para bibliotecas (por ejemplo /usr/lib/) o usar la opción -L de g++ para especificar la ruta en el momento de la compilación. La biblioteca debe incluirse en el código fuente como `#include <UDG_Create.h>` (con paréntesis angulares).

Agregue -lUDG_Create a su comando de compilación para realizar el ligado utilizando la biblioteca. Por ejemplo para el programa

Listing 7.2: main.cpp

```
#include <UDG_Create.h>
int main()
{
    Create robot;
    robot.demo(9);
    return 0;
}
```

puede usar: *g++ main.cpp -lUDG_Create*

7.6. La Clase Create

La clase de c++ Create, esta contenida en el archivo UDG_Create.h y en ella se describe la abstracción de los sensores, actuadores y otros componentes del robot además de todas las operaciones necesarias para la inicialización y operación de los módulos RS-232 y USB.

7.6.1. Tipos de datos y enumeraciones

7.6.1.1. enum errorCode

Define los códigos de error para distintas condiciones de falla durante la ejecución de un programa.

INVALID_DEMO (0x00): El número de modo de desmostración seleccionado no existe.

INVALID_BAUDRATE (0x01): Se ha seleccionado una velocidad de transmisión no válida.

INVALID_INSTRUCTION_MODE (0x02): Se ha intentado ejecutar una instrucción en un modo que no lo permite.

OUT_OF_RANGE (0x03): El parámetro seleccionado está fuera del rango permitido.

INVALID_MODE (0x04): Se ha seleccionado un modo de operación inexistente.

7.6.1.2. enum modes

Enlista los distintos modos en los que puede operar el robot.

OFF: Robot apagado.

PASSIVE: En el modo pasivo es posible ejecutar los programas de demostración así como leer y escribir información de sensores, pero no se permite ejecutar comandos de actuadores.

SAFE: En el modo seguro se tiene control total del robot, pero este regresa a modo pasivo si se detecta un borde (por el que el robot pudiera caer), se solicita un radio de giro menor al radio del robot o se conecta el cargador.

FULL: Se tiene control absoluto del robot.

7.6.1.3. enum baudCode

Enumera las velocidades de transmisión soportadas por el Create®. Las constantes tienen un nombre de la forma BAUDX en donde X representa la velocidad de transmisión en baudios.

BAUD300 (0x00).

BAUD600 (0x01).

BAUD1200 (0x02).

BAUD2400 (0x03).

BAUD4800 (0x04).

BAUD9600 (0x05).

BAUD14400 (0x06).
BAUD19200 (0x07).
BAUD28800 (0x08).
BAUD38400 (0x09).
BAUD57600 (0x0A).
BAUD115200 (0x0B).

7.6.1.4. enum chargingstates

Describe los posibles estados de carga en los que pudiera encontrarse el robot. La documentación del fabricante no proporciona información adicional respecto a estas condiciones, se asumen autoexplicativas.

NOT_CHARGING (0x00).
RECONDITIONING_CHARGING (0x01).
FULL_CHARGING (0x02).
TRICKLE_CHARGING (0x03).
WAITING (0x04).
CHARGING_FAULT_CONDITION (0x05).

7.6.1.5. enum infraredbytechars

Define los posibles valores que pudieran recibirse a través del receptor infrarrojo provenientes del control remoto, la base de carga, otros robots o aplicaciones externas. La documentación del fabricante no proporciona información adicional por lo que los valores se asumen autoexplicativos.

IRLEFT (0x81).
IRFORWARD (0x82).
IRRIGHT (0x83).
IRSPOT (0x84).
IRMAX (0x85).
IRSMALL (0x86).
IRMEDIUM (0x87).
IRLARGE (0x88).
IRPAUSE (0x89).
IRPOWER (0x8A).
IRARC_FORWARD_LEFT (0x8B).
IRARC_FORWARD_RIGHT (0x8C).

IRDRIVE_STOP (0x8D).
IRSENDALL (0x8E).
IRSEEKDOCK (0x8F).
IRRESERVED (0x90).
IRRED (0x91).
IRGREEN (0x92).
IRFORCEFIELD (0x93).
IRREDGREEN (0x94).
IRREDFORCEFIELD (0x95).
IRGREENFORCEFIELD (0x96).
IRREDGREENFORCEFIELD (0x97).

7.6.1.6. enum VerbosityLevels

VERBOSITY_NORMAL (0x00): Los mensajes del robot se escriben a la salida estándar.

VERBOSITY_FILE (0x01): Los mensajes del robot se escriben a un archivo.

VERBOSITY_OFF (0x02): No se escribe ningun mensaje.

VERBOSITY_NUMBER_OF_LEVELS (0x03): El número total de niveles.

7.6.1.7. t_verbosity

Tipo de dato enumerado que corresponde a los valores de VerbosityLevels.

7.6.1.8. enum BoolSigned

SIGNED (0x01).
UNSIGNED (0x02).

7.6.1.9. boolSigned

Tipo de dato enumerado que corresponde a los valores de BoolSigned y se utiliza para indicar si los elementos en un arreglo de bytes debe ser considerado como un número con o sin signo.

7.6.1.10. NUMBER_OF_SENSORS

Describe el máximo número de sensores externos que pueden conectarse a la interfaz USB.

7.6.1.11. int16

Entero de 16 bits, utilizado para garantizar su tamaño independientemente del compilador. Su definición exacta es:

```
typedef struct int_16
{
    unsigned char H; //High byte
    unsigned char L; //Low byte
} int16;
```

7.6.2. Miembros (privados)

7.6.2.0.1. std::string portName

Una cadena que indica el nombre del enlace simbólico asociado a la comunicación serial con el robot.

7.6.2.0.2. int mode

Un valor definido en la enumeración *oimodes* que indica el modo de operación actual del robot.

7.6.2.0.3. bool charging

Una variable lógica que indica si el robot está siendo cargado.

7.6.2.0.4. int portDescriptor

El descriptor del puerto asociado a la comunicación serial con el robot.

7.6.2.0.5. int baudRate

Un valor perteneciente a la enumeración *baudCode* que indica la velocidad de transmisión utilizada para la comunicación serial.

7.6.2.0.6. bool bumpRight

Una variable lógica que indica si el lado derecho del parachoques está siendo presionado.

7.6.2.0.7. bool bumpLeft

Una variable lógica que indica si el lado izquierdo del parachoques está siendo presionado.

7.6.2.0.8. bool wheelDropRight

Una variable lógica que indica si el sensor de caída de la rueda derecha fué activado.

7.6.2.0.9. bool wheelDropLeft

Una variable lógica que indica si el sensor de caída de la rueda izquierda fué activado.

7.6.2.0.10. bool wheelDropCaster

Una variable lógica que indica si el sensor de caída de la rueda caster fué activado.

7.6.2.0.11. bool wall

Una variable lógica que indica si una pared ha sido detectada.

7.6.2.0.12. bool cliffLeft

Una variable lógica que indica si un borde ha sido detectado del lado izquierdo.

7.6.2.0.13. bool cliffFrontLeft

Una variable lógica que indica si el sensor de bordes frontal izquierdo ha sido activado.

7.6.2.0.14. bool cliffFrontRight

Una variable lógica que indica si el sensor de bordes frontal derecho ha sido activado.

7.6.2.0.15. bool cliffRight

Una variable lógica que indica si un borde ha sido detectado del lado derecho.

7.6.2.0.16. bool virtualWall

Una variable lógica que indica si una pared virtual ha sido detectada.

7.6.2.0.17. bool ld0

Una variable lógica que indica si el *low side driver* 0 está encendido.

7.6.2.0.18. bool ld1

Una variable lógica que indica si el *low side driver* 1 está encendido.

7.6.2.0.19. bool ld2

Una variable lógica que indica si el *low side driver* 2 está encendido.

7.6.2.0.20. bool rightWheel

Una variable lógica que indica si el sensor de sobrecorriente de la rueda derecha ha sido activado.

7.6.2.0.21. bool leftWheel

Una variable lógica que indica si el sensor de sobrecorriente de la rueda izquierda ha sido activado.

7.6.2.0.22. unsigned char infraredbyte

Una variable de un byte que almacena la información leída a través del receptor infrarrojo.

7.6.2.0.23. bool advancebtn

Una variable lógica que indica si el botón *advance* del robot está siendo presionado.

7.6.2.0.24. bool playbtn

Una variable lógica que indica si el botón *play* del robot está siendo presionado.

7.6.2.0.25. int distance

Una variable entera que indica la distancia recorrida desde la última vez que se consultó.

7.6.2.0.26. int angle

Una variable entera que indica el ángulo rotado desde la última vez que se consultó.

7.6.2.0.27. unsigned char chargingstate

Un miembro de la enumeración *chargingstates* que describe las condiciones actuales de la carga de la batería.

7.6.2.0.28. int voltage

Una variable entera que representa el voltaje en mV en la batería del robot.

7.6.2.0.29. int current

La corriente en mA que entra (valores positivos) o sale (valores negativos) de la batería del robot.

7.6.2.0.30. unsigned char batterytemperature

La temperatura de la batería del robot en grados celsius.

7.6.2.0.31. int batterycharge

La carga de la batería del robot en mAh.

7.6.2.0.32. int batterycapacity

La capacidad aproximada de la batería en mAh.

7.6.2.0.33. int wallsignal

Una variable entera que representa la magnitud de la lectura en el sensor de paredes.

7.6.2.0.34. int cliffls

Una variable entera que representa la magnitud de la lectura en el sensor de bordes izquierdo.

7.6.2.0.35. int clifffls

Una variable entera que representa la intensidad de la lectura en el sensor de bordes frontal izquierdo.

7.6.2.0.36. int clifffrs

Una variable entera que representa la intensidad de la lectura en el sensor de bordes frontal derecho.

7.6.2.0.37. int cliffrs

Una variable entera que representa la intensidad de la lectura en el sensor de bordes derecho.

7.6.2.0.38. bool digitalinput0

Una variable lógica que indica si existe un nivel de voltaje alto en la entrada digital 0.

7.6.2.0.39. bool digitalinput1

Una variable lógica que indica si existe un nivel de voltaje alto en la entrada digital 1.

7.6.2.0.40. bool digitalinput2

Una variable lógica que indica si existe un nivel de voltaje alto en la entrada digital 2.

7.6.2.0.41. bool digitalinput3

Una variable lógica que indica si existe un nivel de voltaje alto en la entrada digital 3.

7.6.2.0.42. bool baudchangerate

Una variable lógica que indica si hay un nivel de voltaje alto en la terminal correspondiente al cambio de velocidad de transmisión (terminal 15).

7.6.2.0.43. int cargoanalogsignal

Una variable entera que representa el valor de la señal analógica en la terminal 4 del robot.

7.6.2.0.44. bool homebase

Una variable lógica que indica si existe una conexión entre el robot y la base de carga.

7.6.2.0.45. bool internalcharger

Una variable lógica que indica si el cargador interno está conectado.

7.6.2.0.46. unsigned char oimode

Representa el modo de operación actual del robot como un miembro de la enumeración *oimodes*.

7.6.2.0.47. unsigned char songnumber

Indica el número de la canción actualmente seleccionada.

7.6.2.0.48. bool songplaying

Indica si hay una canción siendo tocada actualmente.

7.6.2.0.49. unsigned char stremppackets

El número de sensores o paquetes de sensores solicitados como parte de un flujo.

7.6.2.0.50. int reqvelocity

La velocidad de las ruedas solicitada en la última llamada a una instrucción *drive*.

7.6.2.0.51. int reqradius

El radio de giro solicitado en la última llamada a una función *drive*.

7.6.2.0.52. int reqrvelocity

La velocidad de la rueda derecha solicitada en la última llamada a una función *drive*.

7.6.2.0.53. int reqlvelocity

La velocidad de la rueda izquierda solicitada en la última llamada a una función *drive*.

7.6.2.0.54. bool streamingState

Una variable lógica que indica si un flujo de paquetes de sensores se está transmitiendo actualmente.

7.6.2.0.55. bool externalSensorsEnabled

Una variable lógica que indica si los sensores externos conectados por usb están actualmente habilitados.

7.6.2.0.56. t_verbosity robotVerbosity

Indica el tipo de mensajes de depuración que se mostrarán en pantalla.

7.6.3. Funciones

7.6.3.1. Publicas

7.6.3.1.1. Create() Es el método constructor por defecto, inicializa el puerto serial utilizando los primeros enlaces símbolicos para puerto serial y USB disponibles.

7.6.3.1.2. Create(std::string _portName, t_verbosity verbosityLevel)

_portName: El nombre del enlace símbolico a utilizar para la comunicación serial.

verbosityLevel: El tipo de mensajes de depuración a recibir durante la ejecución. Los valores disponibles para este parametro son

VERBOSITY_NORMAL: Los mensajes se imprimen a la salida estándar.

VERBOSITY_OFF: No se imprimen mensajes.

7.6.3.1.3. ~Create()

Es el método destructor por defecto, libera todos los recursos solicitados durante la ejecución.

7.6.3.1.4. std::string getPortName()

Regresa un valor de tipo std::string conteniendo el nombre del enlace simbólico utilizado para la inicialización del puerto serial.

7.6.3.1.5. void start()

Prepara al robot para recibir instrucciones por el puerto serial y lo pone en modo pasivo (*passive*).

7.6.3.1.6. void baud(unsigned char baudRate)

baudRate: La nueva velocidad de transmisión de acuerdo a los valores establecidos en la Create Open Interface.

Modifica por software la velocidad de transmisión (baud rate) utilizado para la comunicación serial. Se recomienda utilizar los miembros de la enumeración baudRate de la forma BAUDx en donde x representa la velocidad.

7.6.3.1.7. void control()

Pone al robot en modo seguro (safe). Esta función se conserva para mantener compatibilidad con Roomba, se recomienda el uso de la función safe() en su lugar para el Create®.

7.6.3.1.8. void safe()

Pone al robot en modo seguro (safe).

7.6.3.1.9. void full()

Pone al robot en modo íntegro (full).

7.6.3.1.10. void spot()

Ejecuta la demostración “spot demo” en la que el robot se mueve formando una espiral. Al terminar de ejecutar la instrucción el Create® pasa a modo pasivo (passive).

7.6.3.1.11. void cover()

Ejecuta la demostración “cover” de la Create Open Interface en el que se cubre el área de una habitación.

7.6.3.1.12. void coverAndDock()

Ejecuta la demostración “cover and dock” de la Create Open Interface.

7.6.3.1.13. void demo(unsigned char demo)

demo: El número de la demostración a ejecutar.

Ejecuta la demostración indicada por *demo*. El número de demo debe estar entre 1 y 9.

7.6.3.1.14. void drive(int velocity, int radius)

speed: La velocidad promedio a la que girarán las ruedas del robot en mm/s.

radius: Radio en milímetros del giro del robot medido desde el centro del círculo de giro hasta el centro del Create®.

Controla las ruedas del robot y lo hace avanzar usando la velocidad y radio específicos. Un radio negativo genera un giro en el sentido de las manecillas del reloj y uno positivo hacia el lado contrario. Valores positivos de velocidad impulsan al robot

hacia adelante mientras que los negativos lo hacen hacia atrás. Solo los 16 bits menos significativos de los parámetros de entrada son considerados.

7.6.3.1.15. void driveDirect(int rightVelocity, int leftVelocity)

rightVelocity: La velocidad de la rueda derecha.

leftVelocity: La velocidad de la rueda izquierda.

Hace al robot avanzar utilizando las velocidades especificadas para las ruedas izquierda y derecha. Valores positivos representan movimiento hacia adelante mientras que los negativos lo hacen hacia atrás. Solo los 16 bits menos significativos del entero con signo son considerados.

7.6.3.1.16. void leds(unsigned char bit,unsigned char color, unsigned char intensity)

bit: Indica el LED a encender. Los valores posibles son:

0 = ninguno.

2 = Play.

8 = Advance.

10 = Play y Advance.

color: Indica el color que mostrará el LED *Power*. El 0 representa color verde y 255 rojo. Valores intermedios mostrarán tonos intermedios (amarillo, naranja, etc).

intensity: La intensidad con la que encenderá el LED *Power*.

Enciende o apaga los diodos LED seleccionados del Create® permitiendo modificar el color e intensidad del LED *Power*.

7.6.3.1.17. void digitalOutputs(unsigned char outputBits)

outputBits: El valor de salida deseado.

Establece el valor indicado en las 3 terminales de salida del conector DB-25.

Precaución: Cuando se enciende el robot, las salidas digitales permanecen en nivel alto durante 3 segundos.

7.6.3.1.18. void pwmLowSideDrivers(unsigned char dirver1, unsigned char driver1, unsigned char driver0)

driver2: El ciclo de trabajo para la salida PWM en la terminal 24 del conector DB-25 con un valor de entre 0 y 128.

driver1: El ciclo de trabajo para la salida PWM en la terminal 22 del conector DB-25 con un valor de entre 0 y 128.

driver0: El ciclo de trabajo para la salida PWM en la terminal 23 del conector DB-25 con un valor de entre 0 y 128.

Proporciona el ciclo de trabajo especificado en las salidas de PWM del robot. Este debe ser indicado como un número entero entre 0 y 128, por ejemplo, para indicar un ciclo de trabajo de 25 % en alguna de las salidas se escoje un 32.

7.6.3.1.19. void lowSideDrivers(unsigned char bits)

bits: Indica los bits a activar.

Activa los bits seleccionados de los *low side drivers*.

7.6.3.1.20. void sendIr(unsigned char byteValue)

byteValue: El valor del byte a enviar.

Esta función envía a un LED infrarrojo conectado a la terminal 23, el byte indicado en el formato esperado por el receptor infrarrojo del robot.

7.6.3.1.21. void song(unsigned char,unsigned char,...)

Permite almacenar una canción en la memoria del robot para ser tocada posteriormente. La lista de parametros de entrada es de tamaño variable y debe apagarse al siguiente formato:

song(numeroDeCancion,numeroDeNotas, nota1,duracion1, nota2, duracion2,...,notaN, duracionN)

numeroDeCancion debe tener un valor de entre 0 y 15. *numeroDeNotas* debe estar entre 1 y 16. La altura de *notaX* se apega al esquema de numeración MIDI, pero solo

en el rango de entre 31 y 127, cualquier otro valor se considerará como un silencio. El valor de *duracionX* puede estar entre 0 y 255, representando incrementos de 1/64 de segundo.

7.6.3.1.22. void playSong(unsigned char songNumber)

songNumber: El número de canción a tocar.

Toca la canción previamente grabada mediante la instrucción *song*. Se debe esperar a que termine una canción antes de poder tocar otra.

7.6.3.1.23. int sensors(unsigned char idPacket)

idPacket: El identificador del paquete solicitado.

Permite obtener el valor del sensor o paquete de sensores seleccionado y actualizar su valor en la variable correspondiente. El valor de retorno es el del último sensor leído.

7.6.3.1.24. int getSizePacket(int idPacket)

packet: El identificador del paquete solicitado.

Regresa el número de bytes que se esperan del sensor o paquete de sensores especificado.

7.6.3.1.25. void stream(unsigned char* destinationBuffer,void* thread,int n,...)

destinationBuffer: Una apuntador a un buffer de bytes en donde se almacenará la respuesta a la petición de paquetes.

thread: Este parámetro se conserva para mantener compatibilidad con una versión anterior de la biblioteca y será eliminado a corto plazo.

n: El número de sensores o paquetes de sensores solicitados. Debe estar entre 0 y 42.

... : Una lista de tamaño variable conteniendo los identificadores (entre 0 y 42) de los sensores o paquetes de sensores solicitados.

Esta función indica al robot que debe enviar un flujo de bytes con las lecturas de los sensores solicitados cada 15 ms a través de la conexión serial. Esta información es almacenada en el buffer proporcionado en el formato:

19 nbytes IDPaquete1 DatosPaquete1 IDPaquete2 DatosPaquete2 ... IDPaqueteN
DatosPaqueteN Checksum

Para más información consulte la instrucción “stream” de la Create Open Interface.

7.6.3.1.26. **void pauseResumeStream(bool streamState)**

streamState: una variable lógica que indica si el flujo debe estar activado o desactivado.

Permite detener o reiniciar el flujo de bytes conteniendo el valor de los sensores solicitados mediante la función stream, sin tener que reiniciar la lista.

7.6.3.1.27. **void script(unsigned char n,...);**

n: El número de instrucciones en el script. Debe estar en el rango entre 1 y 100.

Permite almacenar ua serie de instrucciones a manera de un script para ser ejecutadas posteriormente. Recibe como primer parametro el número de instrucciones que contendrá el script seguido de una lista de longitud variables en los que se especifican los bytes correspondientes a los códigos de operación de las instrucciones y bytes de datos de acuerdo a la Create Open Interface.

script(4, instrucion1, datos1, instrucion2, datos2);

7.6.3.1.28. **void playScript()**

Ejecuta el último script guardado mediante la instrucción *script()*.

7.6.3.1.29. **void showScript()**

Imprime en pantalla el ultimo script almacenado mediante la instrucción *script()* en su representación como numeros enteros.

7.6.3.1.30. void waitTime(unsigned char time)

time: El tiempo a esperar en décimas de segundo con una resolución de 15ms.

Instruye al robot para esperar el tiempo especificado durante el cual no podrá modificar su estado ni recibir cualquier tipo de estímulo o señal.

7.6.3.1.31. void waitDistance(int distance)

distance: La distancia a esperar expresada en milímetros entre -32767 y 32768.

Instruye al robot para esperar a que la distancia indicada haya sido recorrida. Si avanza hacia adelante o las ruedas son giradas de manera pasiva en cualquier dirección la distancia se incrementa, al avanzar hacia atrás se decrementa. Durante este tiempo el robot no podrá modificar su estado ni recibir cualquier tipo de estímulo o señal.

7.6.3.1.32. void waitAngle(int angle)

angle: El ángulo a girar en grados entre -32767 y 32768.

Hace al robot esperar hasta haber rotado el ángulo especificado. Cuando el giro se hace en el sentido de las manecillas del reloj el ángulo se decrementa mientras que en la dirección contraria se incrementa. Durante este tiempo el robot no podrá modificar su estado ni recibir cualquier tipo de estímulo o señal.

7.6.3.1.33. void waitEvent(unsigned char event)

event: Identificador del evento esperado como se especifica en la Create Open Interface, con un rango de -20 a -1 y 1 a 20.

Instruye al robot para esperar a que el evento especificado suceda. Durante este tiempo no podrá modificar su estado ni recibir cualquier tipo de estímulo o señal.

7.6.3.1.34. char* charMode(int mode)

mode: Representación como número entero del modo actual (ver enum modes).

Obtiene una representación como cadena de caracteres del modo de operación actual.

7.6.3.1.35. int getBaudCode(int baudCode)

baudCode: Velocidad de transmisión en su representación como número entero como se especifica en la Create Open Interface (ver enum baudCode).

Regresa el valor entero correspondiente a la velocidad de transmisión.

7.6.3.1.36. bool getBumpRight()

Regresa el estado del parachoques derecho como un valor lógico. Verdadero significa que el parachoques ha sido presionado.

7.6.3.1.37. bool getBumpLeft()

Regresa el estado del parachoques izquierdo como un valor lógico. Verdadero significa que el parachoques ha sido presionado.

7.6.3.1.38. bool getWheelDropRight()

Indica con una variable lógica si la rueda derecha ha caído.

7.6.3.1.39. bool getWheelDropLeft()

Indica con una variable lógica si la rueda izquierda ha caído.

7.6.3.1.40. bool getWheelDropCaster()

Indica con una variable lógica si la rueda caster ha caído.

7.6.3.1.41. bool getWallSeen()

Regresa el estado del sensor de paredes como un valor lógico. Verdadero significa que una pared ha sido detectada.

7.6.3.1.42. bool getCliffLeft()

Regresa el estado del sensor de bordes izquierdo como un valor lógico. Verdadero significa que un borde ha sido detectado.

7.6.3.1.43. bool getCliffFrontLeft()

Regresa el estado del sensor de bordes frontal izquierdo como un valor lógico. Verdadero significa que un borde ha sido detectado.

7.6.3.1.44. bool getCliffFrontRight()

Regresa el estado del sensor de bordes frontal derecho como un valor lógico. Verdadero significa que un borde ha sido detectado.

7.6.3.1.45. bool getCliffRight()

Regresa el estado del sensor de bordes derecho como un valor lógico. Verdadero significa que un borde ha sido detectado.

7.6.3.1.46. bool getVirtualWall()

Regresa el estado del detector de pared virtual como un valor lógico. Verdadero significa que una pared virtual ha sido detectada.

7.6.3.1.47. bool getLd0()

Regresa el estado del *lowside driver 0* como un valor lógico. Verdadero significa que está activado.

7.6.3.1.48. bool getLd1()

Regresa el estado del *lowside driver 1* como un valor lógico. Verdadero significa que está activado.

7.6.3.1.49. bool getLd2()

Regresa el estado del *lowside driver 2* como un valor lógico. Verdadero significa que está activado.

7.6.3.1.50. bool getRightWheel()

Regresa el estado del sensor de sobrecorriente de la rueda derecha como un valor lógico. Verdadero significa que un existe sobrecorriente.

7.6.3.1.51. bool getLeftWheel()

Regresa el estado del sensor de sobrecorriente de la rueda izquierda como un valor lógico. Verdadero significa que un existe sobrecorriente.

7.6.3.1.52. unsigned char getInfraredByte()

Regresa un byte que representa la información recibida a través del sensor infra rojo. Un valor de 255 significa que no se ha recibido información.

7.6.3.1.53. bool getAdvanceBtn()

Regresa el estado del botón “advance” como un valor lógico. Verdadero significa que el botón está presionado.

7.6.3.1.54. bool getPlayBtn()

Regresa el estado del botón “play” como un valor lógico. Verdadero significa que el botón está presionado.

7.6.3.1.55. int getDistance()

Regresa la distancia recorrida desde la última vez que se ejecutó la instrucción como un valor entero de 16 bits con signo (de -32768 a 32767). Los movimientos hacia adelante se manifiestan como números positivos y hacia atrás como negativos. El resultado final será la suma de todos los movimientos.

7.6.3.1.56. int getAngle()

Regresa el ángulo que se ha rotado desde la última vez que se ejecutó la instrucción como un valor entero de 16 bits con signo (de -32768 a 32767). Los movimientos en el sentido de las manecillas del reloj se manifiestan como números negativos y hacia el lado contrario como positivos. El resultado final será la suma de todos los

movimientos.

7.6.3.1.57. `unsigned char getChargingState()`

Regresa el estado de carga del robot como un byte con uno de los siguientes valores²:

- NOT_CHARGING (0x00).
- RECONDITIONING_CHARGING (0x01).
- FULL_CHARGING (0x02).
- TRICKLE_CHARGING (0x03).
- WAITING (0x04).
- CHARGING_FAULT_CONDITION (0x05).

7.6.3.1.58. `int getVoltage()`

Regresa el voltaje de la batería en mV con un rango de entre 0 y 65535.

7.6.3.1.59. `int getCurrent()`

Regresa la corriente del robot en mA como un entero en un rango de -32768 a 32767. Las corrientes negativas indican un flujo desde la batería (como en la operación normal) y las positivas hacia la batería (como durante la carga).

7.6.3.1.60. `unsigned char getBatteryTemperature()`

Regresa un byte correspondiente a la temperatura de la batería en grados centígrados en un rango de -128 a 127.

7.6.3.1.61. `int getBatteryCharge()`

Regresa un entero correspondiente a la carga de la batería en mAh en un rango de 0 a 65535.

²La documentación de la Create Open Interface no especifica detalles sobre estos estados

7.6.3.1.62. int getBatteryCapacity()

Regresa un entero correspondiente a la capacidad aproximada de la batería en mAh con un rango de 0 a 65535. Esta estimación puede no ser precisa cuando se utilizan baterías alcalinas.

7.6.3.1.63. int getWallSignal()

Regresa un entero correspondiente a la intensidad de la señal del sensor de paredes en un rango de 0 a 4095.

7.6.3.1.64. int getCliffLS()

Regresa un entero correspondiente a la intensidad de la señal del sensor de bordes izquierdo en un rango de 0 a 4095.

7.6.3.1.65. int getCliffFLS()

Regresa un entero correspondiente a la intensidad de la señal del sensor de bordes frontal izquierdo en un rango de 0 a 4095.

7.6.3.1.66. int getCliffFRS()

Regresa un entero correspondiente a la intensidad de la señal del sensor de bordes frontal derecho en un rango de 0 a 4095.

7.6.3.1.67. int getCliffRS()

Regresa un entero correspondiente a la intensidad de la señal del sensor de bordes derecho en un rango de 0 a 4095.

7.6.3.1.68. bool getDigitalInput0()

Regresa el valor de la entrada digital 0 como una variable lógica. Verdadero representa un nivel alto de voltaje.

7.6.3.1.69. bool getDigitalInput1()

Regresa el valor de la entrada digital 1 como una variable lógica. Verdadero representa un nivel alto de voltaje.

7.6.3.1.70. bool getDigitalInput2()

Regresa el valor de la entrada digital 2 como una variable lógica. Verdadero representa un nivel alto de voltaje.

7.6.3.1.71. bool getDigitalInput3()

Regresa el valor de la entrada digital 3 como una variable lógica. Verdadero representa un nivel alto de voltaje.

7.6.3.1.72. bool getBaudRateChange()

Regresa el valor de la terminal 15 en el conector DB-25 del robot como una variable lógica. Verdadero representa un nivel alto de voltaje.

7.6.3.1.73. int getCargoAnalogSignal()

Regresa un valor entero de 10 bits (entre 0 y 1023) representando el nivel de voltaje de 0 a 5v presente en la terminal 4 del puerto DB-25 del Create®.

7.6.3.1.74. bool getHomeBase()

Regresa un valor lógico que indica si la base del Create® está disponible como fuente de carga.

7.6.3.1.75. bool getInternalCharger()

Regresa un valor lógico que indica si el cargador interno está disponible como fuente de carga.

7.6.3.1.76. `unsigned char getOIMode()`

Regresa uno de los siguientes valores de la enumeracion *oimodes*:

OIOFF (0x00): El robot se encuentra apagado.

OIPASSIVE (0x01): El robot se encuentra en modo pasivo.

OISAFE (0x02): El robot se encuentra en modo seguro.

OIFULL (0x03): El robot se encuentra en modo íntegro.

7.6.3.1.77. `unsigned char getSongNumber()`

Regresa el número de la canción actualmente seleccionada.

7.6.3.1.78. `bool getSongPlaying()`

Regresa verdadero si hay una canción tocando o falso si no la hay.

7.6.3.1.79. `unsigned char getStreamPackets()`

Regresa el número de sensores o paquetes de sensores solicitados para un flujo. Este valor de retorno estará entre 0 y 43.

7.6.3.1.80. `int getRequestedVelocity()`

Regresa un entero entre -500 y 500 representando la velocidad más reciente solicitada mediante un comando *drive* en mm/s.

7.6.3.1.81. `int getRequestedRadius()`

Regresa un entero entre -32768 y 32767 representando el radio más reciente solicitando mediante un comando *drive* en mm.

7.6.3.1.82. `int getRequestedRVelocity()`

Regresa un entero entre -500 y 500 representando la velocidad más reciente solicitada para la rueda derecha mediante un comando *drive* en mm/s.

7.6.3.1.83. int getRequestedLVelocity()

Regresa un entero entre -500 y 500 representando la velocidad más reciente solicitada para la rueda izquierda mediante un comando *drive* en mm/s.

7.6.3.1.84. bool getStreamingState()

Regresa un valor lógico que indica si actualmente se transmite un flujo de sensores o no.

7.6.3.1.85. void getExternalSensors(int[NUMBER_OF_SENSORS] sensors)

sensors: Un arrreglo de NUMBER_OF_SENSORS (14 en la implementación actual) enteros pasado por referencia para almacenar los resultados.

Instruye al microcontrolador conectado por USB realizar las lecturas y conversiones de los sensores externos, almacenar los resultados en *sensors* y enviarlos de regreso.

7.6.3.1.86. bool getExternalSensorsEnabledStatus()

Regresa una variable lógica indicando si el uso de los sensores externos está habilitado.

7.6.3.1.87. void setVerbosity(t_verbosity)

t_verbosity: El identificador del tipo de mensajes a mostrar.

Permite modificar la forma en que se muestran los mensajes de depuración durante la ejecución del programa. Los valores que puede tomar son:

VERBOSITY_NORMAL: Los mensajes se imprimen a la salida estándar.

VERBOSITY_OFF: No se imprimen mensajes.

7.6.3.1.88. int getExternalNthSensor(int n)

n: El número de sensor solicitado.

Obtiene el valor del enésimo sensor externo conectado al microcontrolador USB.

7.6.3.1.89. int toLittleEndian(unsigned char* source,int nbytes,int* destination,boolSigned sign)

source: Un arreglo de bytes que contiene los datos a convertir.

nbytes: Tamaño en bytes de los datos a convertir.

destination: Un apuntador a entero contenido la dirección de la variable en donde se almacenará el resultado final de la conversión.

sign: Una variable lógica que indica si los datos a convertir tienen o no signo.

Convierte el arreglo de bytes que recibe como entrada a una variable entera compatible con la arquitectura x86.

7.6.3.2. Privadas**7.6.3.2.1. void error(int error,void* info)**

error: Un miembro de la enumeración errorCodes que identifica el tipo de error.

info: Información adicional que puede ser utilizada por el programador para la depuración del código.

Imprime de manera estandarizada los errores detectados en tiempo de ejecución.

7.6.3.2.2. void printRobotMessage(const char* message,...)

message: El mensaje a imprimir, acepta los mismos indicadores de formato que printf. Internamente se usa como primer parametro de vprintf.

... Una lista de parametros opcional que se pasara como segundo argumento a vprintf.

Es un *wrapper* alrededor de la función vprintf que provee una interfaz para imprimir mensajes durante la ejecución.

7.6.3.2.3. int readBumpsAndWheelDrops(unsigned char *data)

data: La lectura de sensores provenientes del robot.

Convierte la lectura del parachoques y sensores de caída de las ruedas a un formato compatible con la arquitectura x86, la almacena en las variables *bumpRight*, *bumpLeft* , *wheelDropRight*, *wheelDropLeft* y *wheelDropCaster*. Su valor de retorno es un entero cuyos 5 bits menos significativos representan, en ese orden, el estado del parachoques derecho, parachoques izquierdo y sensores de caída de las ruedas

derecha, izquierda y caster.

7.6.3.2.4. int readWall(unsigned char *data)

data: La lectura del sensor de paredes proveniente del robot.

Convierte la lectura del sensor de paredes a un formato compatible con la arquitectura x86 y la almacena en la variable *wall* que es también el valor de retorno de la función.

7.6.3.2.5. int readCliffLeft(unsigned char *data)

data: La lectura del sensor de bordes izquierdo proveniente del robot.

Convierte la lectura del sensor de bordes izquierdo a un formato compatible con la arquitectura x86 y la almacena en la variable *cliffLeft* que es también su valor de retorno.

7.6.3.2.6. int readCliffFrontLeft(unsigned char *data)

data: La lectura del sensor de bordes frontal izquierdo proveniente del robot.

Convierte la lectura del sensor de bordes frontal izquierdo a un formato compatible con la arquitectura x86 y la almacena en la variable *cliffFrontLeft* que es también su valor de retorno.

7.6.3.2.7. int readCliffFrontRight(unsigned char *data)

data: La lectura del sensor de bordes frontal derecho proveniente del robot.

Convierte la lectura del sensor de bordes frontal derecho a un formato compatible con la arquitectura x86 y la almacena en la variable *cliffFrontRight* que es también su valor de retorno.

7.6.3.2.8. int readCliffRight(unsigned char *data)

data: La lectura del sensor de bordes derecho proveniente del robot.

Convierte la lectura del sensor de bordes derecho a un formato compatible con la arquitectura x86 y la almacena en la variable *cliffRight* que es también su valor de retorno.

7.6.3.2.9. **int readVirtualWall(unsigned char *data)**

data: La lectura del sensor de pared virtual proveniente del robot.

Convierte la lectura del sensor de pared virtual a un formato compatible con la arquitectura x86, la almacena en la variable *virtualWall* que es también su valor de retorno.

7.6.3.2.10. **int readLSDriverAndWheelO(unsigned char *data)**

data: La lectura del paquete de sensores *Low Side Driver and Wheel Overcurrents* proveniente del robot.

Convierte la lectura del paquete de sensores *Low Side Driver and Wheel Overcurrents* a un formato compatible con la arquitectura x86 y la almacena en las variables ld1, ld0, ld2, rightWheel y leftWheel correspondientes a los *low side drivers* 1, 0 y 2 y los sensores de sobrecorriente derecho e izquierdo respectivamente. Estos valores son empaquetados en un solo byte en ese orden (siendo ld0 el bit menos significativo) que se utiliza como valor de retorno.

7.6.3.2.11. **int readInfraredByte(unsigned char *data)**

data: La lectura del receptor infrarrojo proveniente del robot.

Convierte la lectura del receptor infrarrojo a un formato compatible con la arquitectura x86 y la almacena en la variable *infraredbyte* que es también su valor de retorno.

7.6.3.2.12. **int readButtons(unsigned char *data)**

data: La lectura del estado de los botones programables proveniente del robot.

Convierte la lectura del estado de los botones del Create® a un formato compatible con la arquitectura x86 y la almacena en las variables *playbtn* y *advancebtn* que son empaquetadas como el bit 0 (menos significativo) y 2 respectivamente de una varia-

ble entera que es el valor de retorno.

7.6.3.2.13. int readDistance(unsigned char *data)

data: El valor de la distancia recorrida proveniente del robot.

Convierte la lectura de distancia recorrida a un formato compatible con la arquitectura x86, la almacena en la variable *distance* que es también su valor de retorno.

7.6.3.2.14. int readAngle(unsigned char *data)

data: La lectura del ángulo rotado proveniente del robot.

Convierte la lectura del ángulo rotado desde la última vez que se ejecutó la instrucción a un formato compatible con la arquitectura x86 y la almacena en la variable *distance* que es también su valor de retorno.

7.6.3.2.15. int readChargingState(unsigned char *data)

data: La lectura del estado de carga proveniente del robot.

Convierte la lectura del estado de carga a un formato compatible con la arquitectura x86 y la almacena en la variable *chargingstate* que es también su valor de retorno.

7.6.3.2.16. int readVoltage(unsigned char *data)

data: La lectura del voltaje de la batería proveniente del robot.

Convierte la lectura del voltaje de la batería a un formato compatible con la arquitectura x86 y la almacena en la variable *voltage* que es también su valor de retorno.

7.6.3.2.17. int readCurrent(unsigned char *data)

data: La lectura de la corriente que consume o recibe el robot.

Convierte la lectura de corriente a un formato compatible con la arquitectura x86 y la almacena en la variable *current* que es también su valor de retorno.

7.6.3.2.18. int readBatteryTemperature(unsigned char *data)

data: La lectura del temperatura de la batería del robot.

Convierte la lectura de la temperatura de la batería a un formato compatible con la arquitectura x86 y la almacena en la variable *batterytemperature* que es también su valor de retorno.

7.6.3.2.19. int readBatteryCharge(unsigned char *data)

data: La lectura del la carga de la batería proveniente del robot.

Convierte la lectura de la carga de la batería a un formato compatible con la arquitectura x86 y la almacena en la variable *batterycapacity* que es también su valor de retorno.

7.6.3.2.20. int readWallSignal(unsigned char *data)

data: La lectura del sensor de paredes proveniente del robot.

Convierte la lectura del sensor de paredes a un formato compatible con la arquitectura x86 y la almacena en la variable *wallsignal* que es también su valor de retorno.

7.6.3.2.21. int readCliffLS(unsigned char *)data

data: La lectura del sensor de bordes izquierdo proveniente del robot.

Convierte la lectura del sensor de bordes izquierdo a un formato compatible con la arquitectura x86 y la almacena en la variable *cliffls* que es también su valor de retorno.

7.6.3.2.22. int readCliffFLS(unsigned char *data)

data: La lectura del sensor de bordes frontal izquierdo proveniente del robot.

Convierte la lectura del sensor de bordes frontal izquierdo a un formato compatible con la arquitectura x86 y la almacena en la variable *cliffcls* que es también su valor de retorno.

7.6.3.2.23. int readCliffFRS(unsigned char *data)

data: La lectura del sensor de bordes frontal derecho proveniente del robot.

Convierte la lectura del sensor de bordes frontal derecho a un formato compatible con la arquitectura x86 y la almacena en la variable *clifffrs* que es también su valor de retorno.

7.6.3.2.24. int readCliffRS(unsigned char *data)

data: La lectura del sensor de bordes derecho proveniente del robot.

Convierte la lectura del sensor de bordes derecho a un formato compatible con la arquitectura x86 y la almacena en la variable *cliffrs* que es también su valor de retorno.

7.6.3.2.25. int readDigitalInputs(unsigned char *data)

data: La lectura de las entradas digitales y el bit de cambio de velocidad de transmisión del robot.

Convierte la lectura de las entradas digitales y el bit de cambio de velocidad de transmisión a un formato compatible con la arquitectura x86 y la almacena en las variables *digitalinput0*, *digitalinput1*, *digitalinput2*, *digitalinput3* y *baudchangerate* que se empaquetan en ese orden como un entero siendo *digitalinput0* el bit menos significativo, que se convierte en valor de retorno.

7.6.3.2.26. int readCargoAnalogSignal(unsigned char *data)

data: La lectura de la entrada analógica proveniente del robot.

Convierte la lectura de la entrada analógica del robot a un formato compatible con la arquitectura x86 y la almacena en la variable *cargoanalogsignal* que es también su valor de retorno.

7.6.3.2.27. int readChargingSources(unsigned char *data)

data: La lectura de las fuentes de carga disponibles proveniente del robot.

Convierte la lectura de las fuentes de carga disponibles a un formato compatible con la arquitectura x86 y la almacena en las variables *internalcharger* y *hombase* que

se empaquetan como los bits 0 y 1 respectivamente de una variable entera que es también su valor de retorno.

7.6.3.2.28. `int readOIMode(unsigned char *data)`

data: La lectura del modo de operación proveniente del robot.

Convierte la lectura del modo de operación a un formato compatible con la arquitectura x86 y la almacena en la variable *oimode* que es también su valor de retorno.

7.6.3.2.29. `int readSongNumber(unsigned char *data)`

data: La lectura del número de canción proveniente del robot.

Convierte la lectura del número de canción a un formato compatible con la arquitectura x86 y la almacena en la variable *songnumber* que es también su valor de retorno.

7.6.3.2.30. `int readSongPlaying(unsigned char *data)`

data: La lectura del estado de ejecución de canción proveniente del robot.

Convierte la lectura del estado de ejecución de canción a un formato compatible con la arquitectura x86 y la almacena en la variable *songplaying* que es también su valor de retorno.

7.6.3.2.31. `int readStreamPackets(unsigned char *data)`

data: El número de sensores o paquetes de sensores solicitados mediante la instrucción *stream* proveniente del robot.

Convierte la lectura del número de sensores solicitados a un formato compatible con la arquitectura x86 y la almacena en la variable *streampackets* que es también su valor de retorno.

7.6.3.2.32. `int readReqVelocity(unsigned char *data)`

data: La lectura de la velocidad solicitada proveniente del robot.

Convierte la lectura de la última velocidad solicitada a un formato compatible con la arquitectura x86 y la almacena en la variable *reqvelocity* que es también su valor de retorno.

7.6.3.2.33. int readReqRadius(unsigned char *data)

data: La lectura del radio de giro solicitado al robot.

Convierte la lectura del último radio de giro solicitado a un formato compatible con la arquitectura x86 y la almacena en la variable *reqradius* que es también su valor de retorno.

7.6.3.2.34. int readReqRVelocity(unsigned char *data)

data: La lectura de la última velocidad de la rueda derecha solicitada proveniente del robot.

Convierte la lectura de la velocidad de la rueda derecha a un formato compatible con la arquitectura x86 y la almacena en la variable *reqrvelocity* que es también su valor de retorno.

7.6.3.2.35. int readReqLVelocity(unsigned char *data)

data: La lectura de la última velocidad de la rueda izquierda solicitada proveniente del robot.

Convierte la lectura de la velocidad de la rueda izquierda a un formato compatible con la arquitectura x86 y la almacena en la variable *reqlvelocity* que es también su valor de retorno.

7.6.3.2.36. int updateSensor(unsigned char packetID, int size)

packetID: El identificador del sensor o paquete de sensores que se actualizará.

size: El tamaño en bytes de los datos del sensor o paquete de sensores a actualizar.

Actualiza el valor del sensor o paquete de sensores especificados por *packetID*.

7.6.3.2.37. void commonInitializationProcedures(string port,bool auto)

port: El nombre del enlace simbólico que representa el puerto serial que se comunicará con el robot.

auto: Indica si se debe o no buscar el enlace simbólico que representa al puerto serial de manera automática.

Efectúa todas las operaciones comunes a los distintos métodos constructores.

7.6.3.2.38. int16 toInt16(int integer)

integer: El valor a convertir.

Realiza una conversión de una variable entera a una estructura que representa un entero de 16 bits.

Capítulo 8

Conclusiones y trabajo futuro

A lo largo de este trabajo se diseñó e implementó UDG_Create, una biblioteca que permite el control a bajo nivel de un robot Create® mediante una interfaz de alto nivel en c++, volviendo la mayoría de las operaciones mecánicas, eléctricas y de manejo de buffers prácticamente transparentes al programador.

En pruebas experimentales se demostró que es posible desarrollar las mismas aplicaciones que con otros frameworks de desarrollo comerciales como el Qbot de Quanser con la eficiencia de los lenguajes a bajo nivel, eliminando la necesidad de licencias y resolviendo la inflexibilidad del código propio del uso de software propietario.

Como trabajo futuro se plantea la posibilidad de su implementación en otros lenguajes de programación, sistemas operativos y arquitecturas, de tal forma que se expandan las posibilidades y se acorten los ciclos de trabajo, adaptándose a las nuevas tecnologías y paradigmas de programación sin perder su eficiencia y flexibilidad.

Bibliografía

- [1] A. Lazinica, *Mobile Robots*. Advanced Robotic Systems International, 2006.
- [2] D. Stites, “Demand for math, science jobs only expected to grow.” [En línea] Disponible en http://www.lansingstatejournal.com/article/20130826/BUSINESS/308260003/Demand-math-science-jobs-only-expected-grow?nclick_check=1, Ago 2013. Revisado 12 de Septiembre de 2013.
- [3] J. Vegso, “Interest in cs as a major drops among incoming freshmen,” May 2005. Computing Research News, Número 3, volumen 17.
- [4] K. Davidson, “Fewer us students training for science careers.” , SFGate, [En línea] Disponible en <http://www.sfgate.com/science/article/Fewer-U-S-students-training-for-science-careers-2760252.php>, May 2004. Revisado 12 de septiembre de 2013.
- [5] A. Salamon, S. Kupersmith, and D. Houston, “Inspiring future young engineers through robotics outreach.” [En línea] Disponible en <http://www.atl.external.lmco.com/papers/1559.pdf>. Revisado 2 de Septiembre de 2013.
- [6] J. Challinger, “Efficient use of robots in the undergraduate curriculum.” [En línea] Disponible en <http://citeseerx.ist.psu.edu/viewdoc/download?jsessionid=5D968BDFF4FE0A87D3D68B1B69B4FFEF?doi=10.1.1.110.4757&rep=rep1&type=pdf>, 2005. Revisado 12 de septiembre de 2013.
- [7] M. Goldweber, C. Congdon, B. Fagin, D. Hwang, and F. Klassner, “The use of robots in the undergraduate curriculum: Experience reports.” [En línea] Disponible en <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.23.651&rep=rep1&type=pdf>. Revisado 2 de Septiembre de 2013.
- [8] R. Huq, H. Lacheray, C. Fulford, D. Wight, and J. Apkarian, “Qbot: An educational mobile robot controlled in matlab simulink environment.” [En

- [8] [En línea] Disponible en <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5090152&tag=1&url=http%3A%2F%2Fieeexplore.ieee.org%2Fstamp%2Fstamp.jsp%3Ftp%3D%26arnumber%3D5090152%26tag%3D1>, May 2009. Revisado 19 de Septiembre de 2013.
- [9] J. Nilsson, *Real-Time Control Systems with Delays*. Lund: Department of Automatic Control Lund Institute of Technology, 1998.
- [10] C.-J. Sjöstedt, “Modelling of displacement compressors using matlab/simulink software.” [En línea] Disponible en <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.149.8012&rep=rep1&type=pdf>, August 2004. Revisado 13 de Septiembre de 2013.
- [11] D. Henriksson, A. Cervin, and K.-E. Årzén, “Truetime: Real-time control system simulation with matlab/simulink.” [En línea] Disponible en <http://lup.lub.lu.se/luur/download?func=downloadFile&recordId=536781&fileId=625643>, 2003. Revisado 13 de Septiembre de 2013.
- [12] iRobot Corporation, “irobot create owner’s guide.” [En línea] Disponible en http://www.irobot.com/filelibrary/create/Create%20Manual_Final.pdf, 2006. Revisado 15 de enero 2014.
- [13] iRobot Corporation, “iRobot create open interface.” [En línea] Disponible en http://www.irobot.com/filelibrary/pdfs/hrd/create/Create%20open%20Interface_v2.pdf, 2006. Revisado 25 de Septiembre de 2013.
- [14] iRobot Corporation, “Create product page details.” [En línea] Disponible en <http://www.irobot.com/en/us/learn/Educators/Create/Details.aspx>, 2013. Revisado 13 de Septiembre de 2013.
- [15] J. C. Lee, “Low-cost video chat robot.” [En línea] Disponible en <http://procrastineering.blogspot.mx/2011/02/low-cost-video-chat-robot.html>, Feb 2011. Revisado 17 Septiembre de 2013.
- [16] R. El-laithy, J. Huang, and M. Yeh, “Study on the use of microsoft kinect for robotics applications.” [En línea] Disponible en http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6236985&tag=1, April 2012. Revisado 17 de Septiembre de 2013.
- [17] A. Saenz, “Hacked irobot uses xbox kinect to see the world, obey commands.” [En línea] Disponible en <http://singularityhub.com/2010/11/17/>

- hacked-irobot-uses-xbox-kinect-to-see-the-world-obey-your-commands-video/, November 2010. Revisado 17 de Septiembre de 2013.
- [18] D. Houston and W. C. Regli, “Low-cost localization for educational robotic platforms via an external fixed-position camera.” [En línea] Disponible en <http://aaaipress.org/Papers/Workshops/2008/WS-08-02/WS08-02-008.pdf>, 2008. Revisado 17 de Septiembre de 2013.
- [19] M. Kuipers, “Localization with the irobot create.” [En línea] Disponible en <http://www.cse.sc.edu/files/reu/2008papers/KuipersPaper.pdf>, 2008. Revisado 17 de Septiembre de 2013.
- [20] Pharos Testbed Wiki Community, “How to use player to control the irobot create.” [En línea] Disponible en http://pharos.ece.utexas.edu/wiki/index.php/How_to_use_Player_to_Control_the_iRobot_Create, December 2010. Revisado 18 de septiembre de 2013.
- [21] UrbiForge, “I robot.” [En línea] Disponible en <http://www.urbiforge.org/index.php/Robots/IRobot>, May 2010. Revisado 18 de Septiembre de 2013.
- [22] Microsoft Corporation , “irobot create and roomba.” [En línea] Disponible en <http://msdn.microsoft.com/en-us/library/bb483025.aspx>, 2012. Revisado 18 de septiembre de 2013.
- [23] ROS.org, “irobot_create_2_1.” [En línea] Disponible en http://wiki.ros.org/irobot_create_2_1, Feb 2007. Revisado 18 de Septiembre de 2013.
- [24] Omega Corporation , “The rs-232 standard.” [En línea] Disponible en <http://www.omega.com/techref/pdf/RS-232.pdf>, 2013. Revisado 20 de septiembre de 2013.
- [25] Compaq Computer Corporation , Hewlett-Packard Company, Intel Corporation , Lucent Technologies Inc , Microsoft Corporation , NEC Corporation and Koninklijke Philips Electronics N.V. , “Universal serial bus specification.” [En línea] Disponible en http://www.usb.org/developers/docs/usb20_docs/usb_20_070113.zip, Abr 2000. Revisado 20 de enero de 2014.
- [26] Microchip Technology Inc., “Pic18f2550/2550/4455/4550 data sheet.” [En línea] Disponible en <http://ww1.microchip.com/downloads/en/DeviceDoc/39632e.pdf>, 2009. Revisado 26 de Enero 2014.

- [27] iRobot Corporation, “iRobot’s online store create product page.” [En línea] Disponible en <http://store.irobot.com/family/index.jsp?categoryId=2591511&s=A-ProductAge>, 2013. Revisado 13 de Septiembre de 2013.
- [28] H. Karaoguz, “Webots guide.” [En línea] Disponible en <http://www.isl.ee.boun.edu.tr/courses/ee451/projects/webotsdoc.pdf>. Revisado 18 de Septiembre de 2013.
- [29] Willow Garage , “Turtlebot.” [En línea] Disponible en <http://www.willowgarage.com/turtlebot>, 2011. Revisado 18 de Septiembre de 2013.
- [30] M. Smith, “Arm vs x86 processors: What’s the difference?.” [En línea] Disponible en <http://www.brighthub.com/computing/hardware/articles/107133.aspx>, May 2011. Revisado 19 de Septiembre de 2013.
- [31] C. Nugteren, G.-J. v. d. Braak, and H. Corporaal, “Future of gpgpu micro-architectural parameters.” [En línea] Disponible en http://parse.ele.tue.nl/system/attachments/39/original/DATE_Future_of_GPGPU_Micro-Architectural_Parameters.pdf, 2013. Revisado 20 de Septiembre de 2013.
- [32] Intel Corporation , “Intel architecture software developer’s manual volume 2: Instruction set reference.” [En línea] Disponible en <http://download.intel.com/design/intarch/manuals/24319101.pdf>, 1999. Revisado 25 de Septiembre de 2013.
- [33] T. Andrews, “Computation time comparison between matlab and c++ using launch windows.” [En línea] Disponible en <http://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1080&context=aerosp>, 2012. Revisado 14 de Octubre de 2013.
- [34] USB Implementers Forum , “Device class definition for human interface devices (hid).” [En línea] Disponible en http://www.usb.org/developers/devclass_docs/HID1_11.pdf, Jun 2001. Revisado 22 de Enero de 2014.

Capítulo 9

Anexos

En esta sección se presentan los archivos de código fuente más relevantes. Se invita al lector a consultar el disco adjunto a este trabajo para encontrar los archivos de proyecto, dependencias y el resto del código.

9.1. main.c

```
/*
 ****
FileName:      main.c
Dependencies:  See INCLUDES section
Processor:     PIC18, PIC24, dsPIC, and PIC32 USB
               Microcontrollers
Hardware:      This demo is natively intended to be used on
               Microchip USB demo
               boards supported by the MCHPFSUSB stack.
               See release notes for
               support matrix. This demo can be modified
               for use on other
               hardware platforms.
Complier:      Microchip C18 (for PIC18), XC16 (for PIC24/
               dsPIC), XC32 (for PIC32)
Company:       Microchip Technology, Inc.

Software License Agreement:
```

The software supplied herewith by Microchip Technology Incorporated (the "Company") for its PIC(R) Microcontroller is intended and supplied to you, the Company's customer, for use solely and exclusively on Microchip PIC Microcontroller products. The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved.

Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

File Description:

Change History:

Rev Description

1.0 Initial release

2.1 Updated for simplicity and to use common

coding style

2.7b *Improvements to USBCBSendResume(), to make it easier to use.*

2.9f *Adding new part support*

```

 **** coding style ****
2.7b Improvements to USBCBSendResume(), to make it easier
      to use.
2.9f Adding new part support
***** coding style *****

#ifndef MAIN_C
#define MAIN_C

/** INCLUDES
***** coding style *****
#include "./USB/usb.h"
#include "HardwareProfile.h"
#include "./USB/usb_function_hid.h"

/** CONFIGURATION
***** coding style *****
#pragma config PLLDIV    = 1           //No
                           divide 4MHz input
#pragma config CPUDIV    = OSC1_PLL2 // [OSC1/OSC2 Src
                           : /1]/[96MHz PLL Src: /2]
#pragma config USBDIV    = 2           // Clock source
                           from 96MHz PLL/2
#pragma config FOSC      = HSPLL_HS // HS oscillator ,
                           PLL enabled, HS used by usb
#pragma config FCMEN     = OFF         // Fail safe clock
                           monitor
#pragma config IESO      = OFF         // Internal/external
                           switch over
#pragma config PWRT      = OFF         // Power up timer
#pragma config BOR       = ON          // Brown out reset
#pragma config BORV     = 3
#pragma config VREGEN    = ON          // USB Voltage
                           Regulator
#pragma config WDT       = OFF
#pragma config WDTPS    = 32768
#pragma config MCLRE    = ON
```

```

        #pragma config LPT1OSC = OFF
        #pragma config PBADEN = OFF
//      #pragma config CCP2MX = ON
        #pragma config STVREN = ON
        #pragma config LVP = OFF
//      #pragma config ICPRT = OFF           // Dedicated In-
Circuit Debug/Programming
        #pragma config XINST = OFF          // Extended
Instruction Set
        #pragma config CP0 = OFF
        #pragma config CP1 = OFF
//      #pragma config CP2 = OFF
//      #pragma config CP3 = OFF
        #pragma config CPB = OFF
//      #pragma config CPD = OFF
        #pragma config WRT0 = OFF
        #pragma config WRT1 = OFF
//      #pragma config WRT2 = OFF
//      #pragma config WRT3 = OFF
        #pragma config WRIB = OFF          // Boot Block
Write Protection
        #pragma config WRTC = OFF
//      #pragma config WRTD = OFF
        #pragma config EBTR0 = OFF
        #pragma config EBTR1 = OFF
//      #pragma config EBTR2 = OFF
//      #pragma config EBTR3 = OFF
        #pragma config EBTRB = OFF

        //high power Timer1 mode
/** VARIABLES
*****
*#pragma udata

#pragma udata USB_VARIABLES=0x480

```

```
unsigned char ReceivedDataBuffer[64] ;
unsigned char ToSendDataBuffer[64];

#pragma udata

USB_HANDLE USBOutHandle = 0;      //USB handle. Must be
                                  initialized to 0 at startup.
USB_HANDLE USBInHandle = 0;       //USB handle. Must be
                                  initialized to 0 at startup.
BOOL blinkStatusValid = TRUE;

/** PRIVATE PROTOTYPES
 ****
void BlinkUSBStatus(void);
BOOL Switch2IsPressed(void);
BOOL Switch3IsPressed(void);
void ProcessIO(void);
void Initialize(void);
void YourHighPriorityISRCode();
void YourLowPriorityISRCode();
void USBCBSendResume(void);
WORD_VAL ReadPOT(void);

/** VECTOR REMAPPING
 ****
#if defined(_18CXX)
```

```

//On PIC18 devices, addresses 0x00, 0x08, and 0x18
//are used for
//the reset, high priority interrupt, and low
//priority interrupt
//vectors. However, the current Microchip USB
//bootloader
//examples are intended to occupy addresses 0x00–0
//x7FF or
//0x00–0xFFFF depending on which bootloader is used.
//Therefore,
//the bootloader code remaps these vectors to new
//locations
//as indicated below. This remapping is only
//necessary if you
//wish to program the hex file generated from this
//project with
//the USB bootloader. If no bootloader is used,
//edit the
//usb_config.h file and comment out the following
//defines:
//#define PROGRAMMABLE_WITH_USB_HID_BOOTLOADER
//#define
PROGRAMMABLE_WITH_USB_LEGACY_CUSTOM_CLASS_BOOTLOADER

```

```

#define REMAPPED_RESET_VECTOR_ADDRESS
0x00
#define
REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS 0
x08
#define
REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS 0
x18

```

```
#pragma code REMAPPED_HIGH_INTERRUPT_VECTOR =
    REMAPPED_HIGH_INTERRUPT_VECTOR_ADDRESS
void Remapped_High_ISR (void)
{
    _asm goto YourHighPriorityISRCode _endasm
}
#pragma code REMAPPED_LOW_INTERRUPT_VECTOR =
    REMAPPED_LOW_INTERRUPT_VECTOR_ADDRESS
void Remapped_Low_ISR (void)
{
    _asm goto YourLowPriorityISRCode _endasm
}

#pragma code

/* ***** I'm not using interrupts but i leave the
   next code section as a PLACEHOLDER
   ***** since it might be useful later
*****
//These are your actual interrupt handling routines.
#pragma interrupt YourHighPriorityISRCode
void YourHighPriorityISRCode()
{
    //Check which interrupt flag caused the
    //interrupt.
    //Service the interrupt
    //Clear the interrupt flag
    //Etc.
#if defined(USB_INTERRUPT)
    USBDeviceTasks();
#endif

}      //This return will be a "retfie fast", since
       //this is in a #pragma interrupt section
#pragma interruptlow YourLowPriorityISRCode
void YourLowPriorityISRCode()
```

```

{
    //Check which interrupt flag caused the
    //interrupt.
    //Service the interrupt
    //Clear the interrupt flag
    //Etc.

}      //This return will be a "retfie", since this
       is in a #pragma interruptlow section

#endif

/** DECLARATIONS
 ****
#pragma code

/*
 ****
 * Function:          void main(void)
 *
 * PreCondition:     None
 *
 * Input:            None
 *
 * Output:           None
 *
 * Side Effects:    None
 *
 * Overview:         Main program entry point.
 *
 * Note:             None
 ****
 */

unsigned char counter=0;

```

```
void main(void)
{
    Initialize();
    USBDeviceInit();
    USBDeviceAttach();
    LATDbits.LATD0 = 1; //turn on LEDon LED
    while(1)
    {
        ProcessIO(); //this is where everything
                      happens
    }
}

void Initialize(void)
{
    //Initialize all of the LED pins
    //mInitAllLEDs();
    TRISDbits.TRISD0 = 0; //output for a LED
    TRISDbits.TRISD1 = 0; //output for a second LED
    TRISDbits.TRISD2 = 1; //input for a digital sensor
    TRISDbits.TRISD3 = 1; //input for a digital sensor
    TRISDbits.TRISD4 = 1; //input for a digital sensor
    TRISDbits.TRISD5 = 1; //input for a digital sensor
    TRISDbits.TRISD6 = 1; //input for a digital sensor
    TRISDbits.TRISD7 = 1; //input for a digital sensor

    TRISCbits.TRISC6 = 1; //input for a digital sensor
    TRISCbits.TRISC7 = 1; //input for a digital sensor

    //B5 (RobotOn) as input
    TRISBbits.TRISB5=1;
    //B6 (RobotCharging) as input
```

```
TRISBbits.TRISB6=1;

ADCON1bits.PCFG3=0;
ADCON1bits.PCFG2=0;
ADCON1bits.PCFG1=0;
ADCON1bits.PCFG0=0;

// Initialize I/O pin and ADC settings to collect
potentiometer measurements
TRISAbits.TRISA0=1;
ADCON0=0x01;
ADCON2=0x3C;
ADCON2bits.ADFM = 1;

// initialize the variable holding the handle for the
last
// transmission
USBOutHandle = 0;
USBInHandle = 0;

blinkStatusValid = TRUE;
} //end UserInit

/*
*****  

* Function: void ProcessIO(void)
*  

* PreCondition: None
*  

* Input: None
*
```



```

        manually control them now.
if (mGetLED_1() == mGetLED_2())
{
    mLED_1_Toggle();
    mLED_2_Toggle();
}
else
{
    if (mGetLED_1())
    {
        mLED_2_On();
    }
    else
    {
        mLED_2_Off();
    }
} */
break;
case 0x81: //Get robotOn and RobotCharging
//Check to make sure the endpoint/buffer is
//free before we modify the contents
if (!HIDTxHandleBusy(USBInHandle))
{
    ToSendDataBuffer[0] = 0x81;
    if (PORTBbits.RB5 == 1)
        //RobotOn is high
    {
        ToSendDataBuffer[1]
            = 0x01;
    }
    else
        //RobotOn is Low
    {
        ToSendDataBuffer[1]
            = 0x00;
    }
}

```

```

        }
        if(PORTBbits.RB6 == 1)                                // RobotCharging is high
        {
            ToSendDataBuffer [2] = 0x01;
        }
        else                                                 // RobotCharging is Low
        {
            ToSendDataBuffer [2] = 0x00;
        }
        //Prepare the USB module to
        //send the data packet to
        //the host
        USBInHandle = HIDTxPacket(HID_EP,(BYTE*)
            &ToSendDataBuffer [0],64);
    }
    break;

case 0x37: //get sensors
{
    //Check to make sure the endpoint/buffer
    //is free before we modify the contents
    if (!HIDTxHandleBusy(USBInHandle))
    {
        // w = ReadPOT();
        unsigned
        char i;
        ToSendDataBuffer
            [0] = 0x37
}

```

```

;           //  

Get each  

ADC  

channel  

coonversion

for ( i=0;i <13; i++ )  

{  

    ADCON0=(i  

        <<2)+1;  

    //  

    increment  

    CHS  

    ADCON0bits.  

    GO = 1;  

while (ADCON0bits.GO  

) ;  

    ToSendDataBuffer  

        [2 * i +1] =  

        ADRESL;  

        //  

        Measured  

        analog  

        voltage  

        LSB  

    ToSendDataBuffer  

        [2 * i +2] =  

        ADRESH;  

        //  

        Measured  

        analog  

        voltage  

        MSB  

}
//The  

digital  

inputs are

```

```

        stored at
        byte 28
        as a
        single
        byte
//We clean
        the byte
ToSendDataBuffer
[27] = 0
x00;

if(PORTDbits
    .RD7==1)
{
    ToSendDataBuffer
    [27]|=0
    x80
    ;
}
if(PORTDbits
    .RD6==1)
{
    ToSendDataBuffer
    [27]|=0
    x40
    ;
}
if(PORTDbits
    .RD5==1)
{
    ToSendDataBuffer
    [27]|=0
    x20
    ;
}
if(PORTDbits
    .RD4==1)
{

```

```

   ToSendDataBuffer
    [27]|=0
    x10
    ;
}

if(PORTCbits
    .RC7==1)
{
   ToSendDataBuffer
    [27]|=0
    x08
    ;
}

if(PORTCbits
    .RC6==1)
{
   ToSendDataBuffer
    [27]|=0
    x04
    ;
}

if(PORTDbits
    .RD3==1)
{
   ToSendDataBuffer
    [27]|=0
    x02
    ;
}

if(PORTDbits
    .RD2==1)
{
   ToSendDataBuffer
    [27]|=0
    x01
    ;
}

```

```

        //Prepare the USB
        //module to send the
        //data packet to
        //the host
    USBInHandle = HIDTxPacket(HID_EP
        ,(BYTE*)&ToSendDataBuffer
        [0],64);
    }
}
break;
}
//Re-arm the OUT endpoint, so we can receive the
//next OUT data packet
//that the host may try to send us.
USBOutHandle = HIDRxPacket(HID_EP, (BYTE*)&
    ReceivedDataBuffer, 64);
    counter++;
    if(counter == 0)
        LATDbits.LATD1 ^= 0x01; //toggle LEDdata
        LED
}

} //end ProcessIO

/*
*****
* Function:          void BlinkUSBStatus(void)
*
* PreCondition:     None
*
* Input:            None
*
* Output:           None

```

```

*
* Side Effects :      None
*
* Overview :          BlinkUSBStatus turns on and off LEDs
*                      corresponding to the USB device state.
*
* Note :               mLED macros can be found in
*                      HardwareProfile.h
*                      USBDeviceState is declared and updated
*                      in
*                      usb_device.c.
*****
*/
void BlinkUSBStatus(void)
{
    // No need to clear UIRbits.SOFIF to 0 here.
    // Callback caller is already doing that.
    static WORD led_count=0;

    if(led_count == 0)led_count = 10000U;
    led_count--;

#define mLED_Both_Off()           {mLED_1_Off();mLED_2_Off()
    () ;}
#define mLED_Both_On()           {mLED_1_On();mLED_2_On()
    ;}
#define mLED_Only_1_On()          {mLED_1_On();mLED_2_Off()
    () ;}
#define mLED_Only_2_On()          {mLED_1_Off();mLED_2_On()
    () ;}

    if(USBSuspendControl == 1)
    {
        if(led_count==0)
        {
            mLED_1_Toggle();
            if(mGetLED_1())

```

```
{  
    mLED_2_On() ;  
}  
else  
{  
    mLED_2_Off() ;  
}  
}//end if  
}  
else  
{  
    if( USBDeviceState == DETACHED_STATE)  
    {  
        mLED_Both_Off() ;  
    }  
    else if( USBDeviceState == ATTACHED_STATE)  
    {  
        mLED_Both_On() ;  
    }  
    else if( USBDeviceState == POWERED_STATE)  
    {  
        mLED_Only_1_On() ;  
    }  
    else if( USBDeviceState == DEFAULT_STATE)  
    {  
        mLED_Only_2_On() ;  
    }  
    else if( USBDeviceState == ADDRESS_STATE)  
    {  
        if( led_count == 0)  
        {  
            mLED_1_Toggle() ;  
            mLED_2_Off() ;  
        } //end if  
    }  
    else if( USBDeviceState == CONFIGURED_STATE)  
    {  
        if( led_count==0)
```

```

{
    mLED_1_Toggle() ;
    if (mGetLED_1() )
    {
        mLED_2_Off() ;
    }
    else
    {
        mLED_2_On() ;
    }
} //end if
}

} //end BlinkUSBStatus

// ****
// ***** USB Callback Functions ****
// ****

// The USB firmware stack will call the callback functions
// USBCBxxx() in response to certain USB related
// events. For example, if the host PC is powering down, it
// will stop sending out Start of Frame (SOF)
// packets to your device. In response to this, all USB
// devices are supposed to decrease their power
// consumption from the USB Vbus to <2.5mA* each. The USB
// module detects this condition (which according
// to the USB specifications is 3+ms of no bus activity/SOF
// packets) and then calls the USBCBSuspend()

```

```
// function. You should modify these callback functions to
// take appropriate actions for each of these
// conditions. For example, in the USBCBSuspend(), you may
// wish to add code that will decrease power
// consumption from Vbus to <2.5mA (such as by clock
// switching, turning off LEDs, putting the
// microcontroller to sleep, etc.). Then, in the
// USCBWakeFromSuspend() function, you may then wish to
// add code that undoes the power saving things done in the
// USBCBSuspend() function.

// The USBCBSendResume() function is special, in that the
// USB stack will not automatically call this
// function. This function is meant to be called from the
// application firmware instead. See the
// additional comments near the function.

// Note *: The "usb_20.pdf" specs indicate 500uA or 2.5mA,
// depending upon device classification. However,
// the USB-IF has officially issued an ECN (engineering
// change notice) changing this to 2.5mA for all
// devices. Make sure to re-download the latest
// specifications to get all of the newest ECNs.

/*
 ****
 * Function:          void USBCBCheckOtherReq(void)
 *
 * PreCondition:     None
 *
```

```

* Input: None
*
* Output: None
*
* Side Effects: None
*
* Overview: When SETUP packets arrive from the host , some
  firmware must
  appropriately to
  the SETUP packets
  USB "chapter 9" (as
  the official USB
  others may be
  that is being
  class device needs
  "GET REPORT" type of
  is not a standard
  therefore not
  this request should
  firmware , such as
  that contained in usb_function_hid.c .
*
* Note: None
*****
 */

```

```

void USBCBCheckOtherReq(void)
{
    USBCheckHIDRequest() ;
}//end

/*
 ****
 * Function:           void USBCBInitEP(void)
 *
 * PreCondition:      None
 *
 * Input:              None
 *
 * Output:             None
 *
 * Side Effects:       None
 *
 * Overview:           This function is called when the device
 *                      becomes
 *                      initialized , which occurs after the host
 *                      sends a
 *                      SET-CONFIGURATION (
 *                      wValue not = 0) request. This
 *                      callback function
 *                      should initialize the endpoints
 *                      for the device 's
 *                      usage according to the current
 *                      configuration .
 *
 * Note:               None
 ****
 */
void USBCBInitEP(void)
{

```

```

//enable the HID endpoint
USBEEnableEndpoint(HID_EP,USB_IN_ENABLED|USB_OUT_ENABLED|
    USB_HANDSHAKE_ENABLED|USB_DISALLOW_SETUP);
//Re-arm the OUT endpoint for the next packet
USBOutHandle = HIDRxPacket(HID_EP,(BYTE*)&
    ReceivedDataBuffer,64);
}

/*
*****
* Function:          void USBCBSendResume(void)
*
* PreCondition:     None
*
* Input:            None
*
* Output:           None
*
* Side Effects:    None
*
* Overview:         The USB specifications allow some types
*                   of USB
*                   peripheral devices
*                   to wake up a host PC (such
*                   as if it is in a low
*                   power suspend to RAM state).
*                   This can be a very
*                   useful feature in some
*                   USB applications,
*                   such as an Infrared remote
*                   control receiver.
*                   If a user presses the "power"
*                   button on a remote
*                   IR receiver can
*                   detect this signalling, and then

```

* send a USB "command"
 * to the PC to wake up.
 *
 * The USBCBSendResume
 * () "callback" function is used
 * to send this special
 * USB signalling which wakes
 * up the PC. This
 * application firmware
 * function will only
 * be able to wake up the host if
 * all of the below are true:
 *
 * 1. The USB driver
 * used on the host PC supports
 * the remote
 * wakeup capability.
 * 2. The USB
 * configuration descriptor indicates
 * the device
 * is remote wakeup capable in the
 * bmAttributes
 * field.
 * 3. The USB host PC
 * is currently sleeping,
 * and has
 * previously sent your device a SET
 * FEATURE
 * setup packet which "armed" the
 * remote
 * wakeup capability.
 *
 * If the host has not armed the device to
 * perform remote wakeup,
 * then this function will return without
 * actually performing a


```

*
*               The modifiable section in this routine
*       may be changed
*               to meet the application needs. Current
*       implementation
*               temporary blocks other functions from
*       executing for a
*               period of ~3–15 ms depending on the core
*       frequency.
*
*               According to USB 2.0 specification
*       section 7.1.7.7,
*               "The remote wakeup device must hold the
*       resume signaling
*               for at least 1 ms but for no more than
*       15 ms."
*
*               The idea here is to use a delay counter
*       loop, using a
*               common value that would work over a wide
*       range of core
*               frequencies.
*
*               That value selected is 1800. See table
*       below:
*

```

	Core Freq (MHz)	MIP	RESUME
Signal Period (ms)			

*	48	12	1.05
*	4	1	12.6

* These timing could be incorrect when
using code

```

*           optimization or extended instruction
*           mode,
*           or when having other interrupts
*           enabled.
*           SIM's Stopwatch
*           Make sure to verify using the MPLAB
*           and verify the actual signal on an
*           oscilloscope.
*****
*/
void USBCBSendResume(void)
{
    static WORD delay_count;

    //First verify that the host has armed us to perform
    //remote wakeup.
    //It does this by sending a SETFEATURE request to
    //enable remote wakeup,
    //usually just before the host goes to standby mode (
    //note: it will only
    //send this SETFEATURE request if the configuration
    //descriptor declares
    //the device as remote wakeup capable, AND, if the
    //feature is enabled
    //on the host (ex: on Windows based hosts, in the device
    //manager
    //properties page for the USB device, power management
    //tab, the
    ///"Allow this device to bring the computer out of
    //standby." checkbox
    //should be checked).
    if(USBGetRemoteWakeUpStatus() == TRUE)
    {
        //Verify that the USB bus is in fact suspended,
        //before we send
        //remote wakeup signalling.
        if(USBIIsBusSuspended() == TRUE)
        {

```

```

USBMaskInterrupts() ;

//Clock switch to settings consistent with
normal USB operation.
// USBCBWakeFromSuspend();
USBsuspendControl = 0;
USBBusIsSuspended = FALSE; //So we don't
execute this code again,
//until a new
suspend condition
is detected.

//Section 7.1.7.7 of the USB 2.0 specifications
indicates a USB
//device must continuously see 5ms+ of idle on
the bus, before it sends
//remote wakeup signalling. One way to be
certain that this parameter
//gets met, is to add a 2ms+ blocking delay here
(2ms plus at
//least 3ms from bus idle to USBIsBusSuspended()
== TRUE, yields
//5ms+ total delay since start of idle).
delay_count = 3600U;
do
{
    delay_count--;
} while(delay_count);

//Now drive the resume K-state signalling onto
the USB bus.
USBResumeControl = 1; // Start RESUME
signaling
delay_count = 1800U; // Set RESUME line
for 1-13 ms
do
{
    delay_count--;

```

```

}while( delay_count );
USBResumeControl = 0;           //Finished driving
                                resume signalling

USBUnmaskInterrupts();
}

}

/*
*****
* Function:      BOOL USER_USB_CALLBACK_EVENT_HANDLER(
*                  USB_EVENT event, void *pdata, WORD
*                  size)
*
* PreCondition:  None
*
* Input:         USB_EVENT event - the type of event
*                void *pdata - pointer to the event data
*                WORD size - size of the event data
*
* Output:        None
*
* Side Effects: None
*
* Overview:     This function is called from the USB
*                stack to
*                notify a user application that a USB
*                event
*                occurred. This callback is in interrupt
*                context
*                when the USB_INTERRUPT option is
*                selected.
*
* Note:          None

```

```
*****
 */
BOOL USER_USB_CALLBACK_EVENT_HANDLER( int event , void *pdata ,
WORD size )
{
    switch( event )
    {
        case EVENT_TRANSFER:
            //Add application specific callback task or
            //callback function here if desired.
            break;
        case EVENT_SOF:
            //USBCB_SOF_Handler();
            break;
        case EVENT_SUSPEND:
            //USBCBSuspend();
            break;
        case EVENT_RESUME:
            //USBCBWakeFromSuspend();
            break;
        case EVENT_CONFIGURED:
            USBCBInitEP();
            break;
        case EVENT_SET_DESCRIPTOR:
            //USBCBStdSetDscHandler();
            break;
        case EVENT_EP0_REQUEST:
            USBCBCheckOtherReq();
            break;
        case EVENT_BUS_ERROR:
            //USBCBErrorHandler();
            break;
        case EVENT_TRANSFER_TERMINATED:
            //Add application specific callback task or
            //callback function here if desired.
            //The EVENT_TRANSFER_TERMINATED event occurs
            //when the host performs a CLEAR
```

```

//FEATURE (endpoint halt) request on an
//application endpoint which was
//previously armed (UOWN was = 1). Here would
//be a good place to:
//1. Determine which endpoint the transaction
//   that just got terminated was
//   on, by checking the handle value in the
//   *pdata.
//2. Re-arm the endpoint if desired (typically
//   would be the case for OUT
//   endpoints).
break;
default:
    break;
}
return TRUE;
}

/** EOF main.c
*****

```

9.2. DataTypes.h

```

#ifndef DATATYPES_H
#define DATATYPES_H
/*typedef enum VerboseLevels {VERBOSITY_NORMAL,
    VERBOSITY_FILE, VERBOSITY_OFF, VERBOSITY_NUMBER_OF_LEVELS}
    t_verbose;
typedef enum BoolSigned {SIGNED, UNSIGNED} boolSigned;

typedef struct int_16
{
    unsigned char H; //High byte
    unsigned char L; //Low byte
} int16; */
#endif

```

9.3. ExternalSensors.cpp

```
#include "USBLayer.h"
#define VID 0X04D8
#define PID 0X003F

bool InitializeSensors()
{
    if (initializeUSB(VID,PID) < 0)
    {return false;}
    else
    {return true;}
}

bool InitializeSensors(int vid,int pid)
{
    if (initializeUSB(vid,pid) < 0)
    {return false;}
    else
    {return true;}
}

void GetSensors(int destinationArray [NUMBER_OF_SENSORS])
{
    unsigned char buffer [65];
    GetUSBData(buffer);
    int j=0;
    for (int i=1 ; i<27;i+=2)
    {
        destinationArray [j]=(buffer [i+1]<<8)+buffer [
            i];
        j++;
    }
    destinationArray [13] = (int)buffer [27];
}

int GetNthSensor(int n)
{
```

```

if(n < 0 || n > 13)
{return -1;}
int array [NUMBER_OF_SENSORS];
GetSensors(array);
return array[n];
}

void CleanExternalSensors()
{
    CloseUSB();
}

void SetSensorVerbosity(t_verbosity level)
{
    SetUSBVerbosity(level);
}

```

9.4. ExternalSensors.h

```

#ifndef EXTERNALSENSORS_H
#define EXTERNALSENSORS_H
//#include "DataTypes.h"
#include "UDG_Create.h"
bool InitializeSensors();
bool InitializeSensors(int VID, int PID);
void GetSensors(int destinationArray[13]);
int GetNthSensor(int n);
void CleanExternalSensors();
void SetSensorVerbosity(t_verbosity level);

#endif

```

9.5. SerialPort.c

*/*Author: Omar Alejandro Rodrguez Rosas
based on the examples at [http://en.wikibooks.org/wiki/
Serial_Programming/termios](http://en.wikibooks.org/wiki/Serial_Programming/termios)*

```
*/  
#include "SerialPort.h"
```

```
t_verbosity SerialPortVerbosity;
```

```
/*
```

string PrintError(int errorCode)

errorCode: an "Error" enumeration member that represents a particular error condition

Prints a message describing a particular error code

```
*/
```

```
void PrintError(int errorCode)
{
    if(SerialPortVerbosity == VERBOSITY.NORMAL)
    {
        printSerialMessage("E% :", errorCode);

        switch(errorCode)
        {
            case ERROR_OPEN:
                printSerialMessage(" An error has
                                  occurred while trying to open a
                                  port.\n");
                break;

            case ERROR_SET_SPEED:
                printSerialMessage(" An error has
                                  occurred while setting the serial
                                  port baud rate\n");
                break;
        }
    }
}
```

```

        case ERROR_CONFIGUREPORT:
            printSerialMessage(" An error has
                occurred while setting the serial
                port configuration\n");
            break;

        case ERROR_WRITE:
            printSerialMessage(" An error has
                occurred while writing to the
                serial port\n");
            break;

        case ERROR_READ:
            printSerialMessage(" An error has
                occurred while reading from the
                serial port\n");
            break;
    }
}

/*

```

int OpenPort(const char PortName)*
Receives a char string representing the name of the port to
open (i.e. "/dev/tty0")
and returns an int descriptor of such port. Returns -1 if
the port can't be opened.

Additional info:

O_RDWR: Opens the port for reading and writing
O_NOCTTY: The port never becomes the controlling terminal of
 the process.
O_NDELAY: Use non-blocking I/O. On some systems this also
 means the RS232 DCD signal line is ignored.

```
/*
int OpenPort(const char* portName)
{   int portDescriptor = open(portName, O_RDWR | O_NOCTTY |
O_NDELAY);

    if (portDescriptor < 0)
    {
        PrintError(ERROR_OPEN);
        printSerialMessage("Port: %s\n", portName);
    }
    else
    {
        printSerialMessage("Successfully opened %s\n"
                           , portName);
    }

    return portDescriptor;
}

int AutoOpenPort()
{
    string portPrefix("/dev/ttyUSB");
    int portDescriptor = -1;
    string portName;
    for (int i = 0; i < 10 ; i++)
    {
        portName = portPrefix+to_string(i);
        portDescriptor = open(portName.c_str() ,
                             O_RDWR | O_NOCTTY | O_NDELAY);
        printSerialMessage("Trying to open %s\n",
                           portName.c_str());
        if (portDescriptor > 0)
        {
            printSerialMessage("Successfully
                               opened %s\n", portName.c_str());
            return portDescriptor;
        }
    }
}
```

```

portPrefix = "/dev/ttyS";
for (int i =0; i< 50 ; i++)
{
    portName = portPrefix+to_string(i);
    portDescriptor = open(portName.c_str() ,
        O_RDWR | O_NOCTTY | O_NDELAY);
    printSerialMessage("Trying to open %s\n",
        portName.c_str());
    if(portDescriptor > 0)
    {
        printSerialMessage("Successfully
            opened %s\n",portName.c_str());
        return portDescriptor;
    }
}

printSerialMessage("No serial port could be opened \
n");
return portDescriptor;

}
/*
```

int ConfigurePort(int portDescriptor)
PortDescriptor: An int descriptor for the selected port

Sets all the needed configurations for serial communication.
Returns -1 if something
goes wrong

```
*/
int ConfigurePort(int portDescriptor)
{
int returnValue;
```

```

struct termios configuration;
//=====Configure port
=====

//more info http://homepages.cwi.nl/~aeb/linux/man2html/man3
//termios.3.html

/*IGNBRK:      Ignore break conditions
 BRKINT:      flushes the queue when a brake is received
 ICRNL:       Translate CR to NL
 INLCR:       Translate NL to CR
 PARMRK:      Mark parity errors
 INPCK:       Parity checking
 ISTRIP:      Strip off eight bit
 IXON:        XON/XOFF flow control
*/
//Disables those options
memset(&configuration, 0, sizeof(configuration));
configuration.c_iflag &= ~(IGNBRK | BRKINT | ICRNL |
                           INLCR | PARMRK | INPCK | ISTRIP | IXON);
configuration.c_oflag=0;
/*ECHO: echo input characters
 ECHONL: echoes the NL character if ICANON is set
 ICANON: Canonical mode
 IEXTEN: implementation-defined input processing
 ISIG: generate INTR, QUIT, SUSP, or DSUSP signals
*/
configuration.c_lflag &= ~(ECHO | ECHONL | ICANON | IEXTEN |
                           ISIG);
/*CSIZE: Character size mask. Values are CS5, CS6, CS7, or
 CS8
PARENB: parity generation on output and parity checking for
input.
CS8: See CSIZE
*/
configuration.c_cflag &= ~(CSIZE | PARENB);
configuration.c_cflag |= CS8;
/*VMIN: minimum number of bytes received before read()
returns

```

```

VTIME: Tentshs of second before consider a read() as
finished
*/
configuration.c_cc[VMIN] = 1;
configuration.c_cc[VTIME] = 0;

returnValue = cfsetispeed(&configuration, B57600) < 0 ||
    cfsetospeed(&configuration, B57600);
if(returnValue < 0)
{
    PrintError(ERROR_SET_SPEED);
    return returnValue;
}
/*TCSANOW: Apply immediately
*/
returnValue = tcsetattr(portDescriptor, TCSANOW, &
    configuration);
if(returnValue < 0)
{
    PrintError(ERROR_CONFIGUREPORT);
    printSerialMessage(" Serial Port Succesfully configured\n");
    return returnValue;
}

/*


---


int WriteToSerial(int portDescriptor, char* buffer, int
numberOfBytes)

writes numberOfBytes bytes from buffer to portDescriptor
port


---


*/
int WriteToSerial(int portDescriptor, unsigned char* buffer,
    int numberOfBytes)
{
    int returnValue;

```

```
    returnValue=write( portDescriptor ,buffer ,
                      numberOfBytes );
    if( returnValue<0)
    { PrintError(ERROR_WRITE) ;}
    return returnValue;
}
int ReadFromSerial( int portDescriptor ,unsigned char* buffer ,
                     int numberOfBytes )
{
int returnValue;
    returnValue=read( portDescriptor ,buffer ,numberOfBytes
                      );
    if( returnValue<0)
    { PrintError(ERROR_READ) ;}
    return returnValue;
}

void StartThreadedRead( int portDescriptor ,unsigned char*
                        buffer ,int numberOfBytes )
{
    std :: thread t(ThreadedRead ,portDescriptor ,buffer ,
                    numberOfBytes );
t.join();

}

void ThreadedRead( int portDescriptor ,unsigned char* buffer ,
                   int numberOfBytes )
{
    while(1)
    {
        if( read( portDescriptor ,buffer ,numberOfBytes )
            >0)
            cout<<*buffer ;
    }
}

void printSerialMessage( const char* message ,... )
```

```

{
    if( SerialPortVerbosity == VERBOSITY_NORMAL )
    {
        va_list arguments;
        va_start( arguments , message );
        vprintf( message , arguments );
        va_end( arguments );
    }
}

void SetSerialPortVerbosity( t_verbosity level )
{
    SerialPortVerbosity = level ;
}

```

9.6. SerialPort.h

```

#ifndef SERIALPORT_H
#define SERIALPORT_H
#define ERROR -1

#include <termios.h>           //UNIX API for terminal I/O
#include <fcntl.h>               // Constant declarations for termios
                                // functions
#include <unistd.h>             //close() function
#include <iostream>
#include <cstdlib>
#include <stdio.h>
#include <string.h>              //needed for memset
#include <thread>
#include <stdarg.h>
#include "UDG_Create.h"
using namespace std;

void PrintError( int errorCode );
int OpenPort( const char* portName );
int ConfigurePort( int portDescriptor );

```

```

int WriteToSerial(int portDescriptor ,unsigned char* buffer ,
    int numberofBytes);
int ReadFromSerial(int portDescriptor ,unsigned char* buffer ,
    int numberofBytes);
void StartThreadedRead(int portDescriptor ,unsigned char*
    buffer ,int numberofBytes);
void ThreadedRead(int portDescriptor ,unsigned char* buffer ,
    int numberofBytes);
void printSerialMessage(const char* message ,...);
void SetSerialPortVerbosity(t_verbosity level);
int AutoOpenPort();
enum Errors
{
    ERROR_OPEN,
    ERROR_SET_SPEED,
    ERROR_CONFIGURE_PORT,
    ERROR_WRITE,
    ERROR_READ
};

};

#endif

```

9.7. UDG_Create.cpp

```

#include "UDG_Create.h"
#include<thread>
#include "ExternalSensors.h"
#include "SerialPort.h"

using namespace std;

////////////////////////////////////////////////////////////////////////NOT PART OF CREATE CLASS
*****
//This function will save the numberofBytes bytes in buffer
when requested

```

```

//It's the user's responsability to give the proper
interpretation to the raw data
void ThreadedReadStream(int portDescriptor , Create* r ,
unsigned char*buffer ,int numberOfBytes)
{
    while(r->getStreamingState() )
    {
        usleep(15000);
        ReadFromSerial(portDescriptor , buffer ,
                        numberOfBytes);
    }
}
//*****
*****
```

```

int16 Create :: toInt16(int integer )
{
    int16 data;
    unsigned char temp[2];
    memcpy(temp,&integer ,2 );
    data.H = temp[1];
    data.L = temp[0];
    return data;
```

```

}
```

```

Create :: Create()
{
    commonInitializationProcedures( string("nothing") ,
                                    true);
}
```

```

Create :: Create( string _portName , t_verbosity level)
{
    setVerbosity( level);
    portName = _portName;
```

```
    commonInitializationProcedures( _portName , false ) ;  
}  
  
void Create :: commonInitializationProcedures( string _portName  
    , bool automaticInitialization )  
{  
  
    if( automaticInitialization )  
    {  
        portDescriptor = AutoOpenPort() ;  
  
    }  
    else  
    {  
        portName = _portName ;  
        portDescriptor = OpenPort( portName . c _str () ) ;  
    }  
    ConfigurePort( portDescriptor ) ;  
    streamingState = false ;  
    externalSensorsEnabled = InitializeSensors() ; //<—  
    USB  
    if ( ! externalSensorsEnabled )  
    { cout << "USB external sensors not enabled!" << endl ; }  
    start() ;  
}  
Create :: ~ Create ()  
{  
    if ( externalSensorsEnabled )  
    { CleanExternalSensors() ; }  
}  
  
string Create :: getPortName()  
{  
    return portName ;  
}  
  
bool Create :: isCharging()
```

```
{  
    //TODO: implement charging state by hardware  
    return charging;  
}  
  
void Create::start()  
{  
    unsigned char ins = 0x80;  
    WriteToSerial(portDescriptor,&ins,1);  
    mode = PASSIVE;  
}  
  
void Create::baud(unsigned char baudCode)  
{  
    if(baudCode>=0 && baudCode<=11)  
    {  
        unsigned char ins[2]{0x81,baudCode};  
        WriteToSerial(portDescriptor,ins,2);  
        //according to documentation you must wait  
        //100ms before sending another instruction  
        usleep(100000);  
        baudRate = getBaudCode(baudCode);  
    }  
    else  
        error(INVALID_BAUDRATE,&baudCode);  
}  
  
void Create::control()  
{  
    safe();  
}  
  
void Create::safe()  
{  
    unsigned char ins = 0x83;  
    WriteToSerial(portDescriptor,&ins,1);  
    mode = SAFE;  
}
```

```
void Create::full()
{
    unsigned char ins = 0x84;
    WriteToSerial(portDescriptor,&ins,1);
    mode = FULL;
}

void Create::spot()
{
    unsigned char ins = 0x86;
    WriteToSerial(portDescriptor,&ins,1);
    mode = PASSIVE;
}

void Create::cover()
{
    unsigned char ins = 0x87;
    WriteToSerial(portDescriptor,&ins,1);
    mode = PASSIVE;
}

void Create::demo(unsigned char demo)
{
    if(demo>0 && demo<=9)
    {
        unsigned char ins[2] = {0x88,demo};
        WriteToSerial(portDescriptor,ins,2);
        mode = PASSIVE;
    }
    else
        error(INVALID_DEMO,&demo);
}

void Create::drive(int velocity,int radius)
{
    if(mode == SAFE || mode == FULL)
    {
```

```

        int16 data1 =.toInt16(velocity);
        int16 data2 =.toInt16(radius);
        unsigned char ins[5] = {0x89, data1.H, data1.L
                               , data2.H, data2.L};
        WriteToSerial(portDescriptor, ins, 5);
    }
    else
        error(INVALID_INSTRUCTION_MODE, (char*)"Function DRIVE");
}

void Create::lowSideDrivers(unsigned char driverBits)
{
    //Probar funcion 1/7
    if(mode == SAFE || mode == FULL)
    {
        if(driverBits >=0 && driverBits <=7)
        {
            unsigned char ins[2] = {0x8A,
                                   driverBits};
            WriteToSerial(portDescriptor, ins, 2);
        }
        else
            error(OUT_OF_RANGE, (char*)"Function LOWSIDEDRIVERS");
    }
    else
        error(INVALID_INSTRUCTION_MODE, (char*)"Function LOWSIDEDRIVERS");
}

void Create::leds(unsigned char bit, unsigned char color,
                  unsigned char intensity)
{
    if(mode == SAFE || mode == FULL)
    {
        if(bit == 2 || bit == 8 || bit == 10 || bit
            == 0)

```

```

    {
        unsigned char ins[4]{0x8B, bit, color,
                             intensity};
        WriteToSerial(portDescriptor, ins, 4);
    }
    else
    {
        error(OUT_OF_RANGE, (char*)" Function
              LEDS");
    }
}
else
{
    string auxString(charMode(mode));
    auxString += " in Function LEDS";
    error(INVALID_INSTRUCTION_MODE, const_cast<
          char*>(auxString.c_str()));
}
}

```

```

        ins [ i+1 ] = ( unsigned char ) va_arg (
            args , int ) ;
    }
    WriteToSerial( portDescriptor , ins , n*2+3 );
}
else
{
    error ( OUT_OF_RANGE, ( char * ) " Function song(
        unsigned char , unsigned char , ... ) " );
}

void Create :: playSong ( unsigned char songNumber )
{
    if ( mode == SAFE || mode == FULL )
    {
        if ( songNumber >= 0 && songNumber <= 15 )
        {
            unsigned char ins [ 2 ] { 0x8D , songNumber
                };
            WriteToSerial( portDescriptor , ins , 2 );
        }
        else
        {
            error ( OUT_OF_RANGE, ( char * ) " Function
                PLAYSONG" );
        }
    }
    else
    {
        string auxString ( charMode ( mode ) );
        auxString += " in Function PLAYSONG" ;
        error ( INVALID_INSTRUCTION_MODE , const_cast<
            char*>( auxString . c_str ( ) ) );
    }
}

int Create :: sensors ( unsigned char idPacket )
{
    if ( idPacket >= 0 && idPacket <= 42 )

```

```
{  
    unsigned char ins[2]{0x8E,idPacket};  
    WriteToSerial(portDescriptor,ins,2);  
    int sizePacket = getSizePacket(idPacket);  
    return updateSensor(idPacket,sizePacket);  
}  
else  
    error(OUT_OF_RANGE,(char*)"Function SENSORS"  
        );  
}  
  
int Create::getSizePacket(int idPacket)  
{  
    int sizePacket = 0;  
    switch(idPacket)  
    {  
        case 0:  
            sizePacket = 26;  
            break;  
  
        case 1:  
            sizePacket = 10;  
            break;  
  
        case 2:  
            sizePacket = 6;  
            break;  
  
        case 3:  
            sizePacket = 10;  
            break;  
  
        case 4:  
            sizePacket = 14;  
            break;  
  
        case 5:  
            sizePacket = 12;
```

```
        break;

    case 6:
        sizePacket = 52;
        break;

    case 7:case 8:case 9:case 10:case 11:case
        12:case 13:
    case 14:case 15:case 16:case 17:case 18:case
        21:case 24:
    case 32:case 34:case 35:case 36:case 37:case
        38:
        sizePacket = 1;
    break;
    case 19:case 20:case 22:case 23:case 25:case
        26:case 27:case 28:
    case 29:case 30:case 31:case 33:case 39:case
        40:case 41:case 42:
        sizePacket = 2;
    break;
}
return sizePacket;
}

void Create::coverAndDock()
{
    unsigned char ins = 0x8F;
    WriteToSerial(portDescriptor,&ins,1);
    mode = PASSIVE;
}

void Create::pwmLowSideDrivers(unsigned char lowSideDriver2,
    unsigned char lowSideDriver1,unsigned char lowSideDriver0)
{
    //Probar funcion 2/7
    if(mode == SAFE || mode == FULL)
    {
```

```

if( lowSideDriver2 <=128 && lowSideDriver2 >=0
    &&
    lowSideDriver1 <=128 && lowSideDriver1 >=0 &&
    lowSideDriver0 <=128 && lowSideDriver0 >=0)
{
    unsigned char ins [4] = {0x90 ,
        lowSideDriver2 , lowSideDriver1 ,
        lowSideDriver0 };
    WriteToSerial( portDescriptor , ins ,4 );
}
else
    error (OUT_OF_RANGE, ( char* )" Function
        PWMLOWSIDEDRIVERS" );
}
else
    error (INVALID_INSTRUCTION_MODE, ( char* )"
        Function PWMLOWSIDEDRIVERS" );
}

void Create::driveDirect(int rightVelocity ,int leftVelocity )
{
    if(mode == SAFE || mode == FULL)
    {
        int16 data1 =.toInt16(rightVelocity);
        int16 data2 =.toInt16(leftVelocity);
        unsigned char ins [5] = {0x91 ,data1.H,data1.L
            ,data2.H,data2.L};
        WriteToSerial( portDescriptor ,ins ,5 );
    }
    else
        error (INVALID_INSTRUCTION_MODE, ( char* )"
            Function DRIVEDIRECT" );
}

void Create::digitalOutputs(unsigned char outputBits)
{
    //Probar funcion 3/7
    if(mode == SAFE || mode == FULL)

```

```

{
    if( outputBits>=0 && outputBits<=7)
    {
        unsigned char ins [ 2 ] = {0x93 ,
                                    outputBits};
        WriteToSerial( portDescriptor ,ins ,2 ) ;
    }
    else
        error(OUT_OF_RANGE,( char *)" Function
              DIGITALOUTPUTS" );
}
else
    error( INVALID_INSTRUCTION_MODE,( char *)" Function
          DIGITALOUTPUTS" );
}

//TODO: I probably need to re-implement this
/*void Create::stream( vector<unsigned char> packets )
{
    //Probar funcion 4/7
    if(mode==PASSIVE || mode == SAFE || mode == FULL)
    {
        unsigned char* ins = new unsigned char[
            packets.size() +2];
        ins[0] = 0x94;
        ins[1] = packets.size();
        int size = packets.size();
        for( int i = 2;i<size ;i++)
        {
            ins[ i ] = packets[ i -2];
        }
        WriteToSerial( portDescriptor ,ins ,size+2);
        streamingState = true;
        //Read packets
        sleep(1);
        std::thread t( ThreadedReadStream ,
                      portDescriptor ,ins ,8 );
        t.join();
    }
}
```

```

        }
    else
        error( INVALID_INSTRUCTION_MODE, ( char* )"
            Funcion STREAM");
}*/



void Create::stream( unsigned char* destinationBuffer ,void*
    thread ,int n,...)
{
    std :: thread *t =(std :: thread*) t ;
    va_list args ;
    va_start( args ,n );
    int bytesToRead=3+n;
    //Probar funcion 4/7
    if(n>=0 && n<=43)
    {
        if(mode==PASSIVE || mode == SAFE || mode == FULL)
        {
            unsigned char* ins = new unsigned char[n+2];
            ins [0] = 0x94;
            ins [1] =n;

            for( int i = 2;i<n+2;i++)
            {
                ins [i] = (unsigned char)va_arg( args ,
                    int);
                bytesToRead+=getSizePolicy( ins [i] );
            }
            WriteToSerial( portDescriptor ,ins ,n+2);
            streamingState = true;
            //Read packets
            sleep(1);
            *t= std :: thread( ThreadedReadStream ,
                portDescriptor ,this ,destinationBuffer ,
                bytesToRead);
                //t.join();
        }
    else

```

```

        error (INVALID_INSTRUCTION_MODE, ( char *)"
            Function STREAM" );
    }
else
{
    error (OUT_OF_RANGE, ( char *)" Function DIGITALOUTPUTS" )
        ;
}
}

void Create :: queryList ( vector<unsigned char> packets )
{
    //Probar funcion 5/7
    if ( mode == PASSIVE || mode == SAFE || mode == FULL )
    {
        if ( packets . size () >=1 && packets . size () <=255 )
        {
            unsigned char* ins = new unsigned
                char [ packets . size () +2];
            ins [ 0 ] = 0x95;
            ins [ 1 ] = packets . size ();
            int size = packets . size ();
            for ( int i = 2; i < size ; i++)
            {
                ins [ i ] = packets [ i -2];
            }
            WriteToSerial ( portDescriptor , ins ,
                size +2);
            //Read packet
        }
    }
    else
        error (OUT_OF_RANGE, ( char *)" Function
            QUERYLIST" );
}
else
    error (INVALID_INSTRUCTION_MODE, ( char *)"
        Function QUERYLIST" );
}

```

```
}
```

```
void Create::pauseResumeStream(bool streamState)
{
    if(mode == PASSIVE || mode == SAFE || mode == FULL)
    {
        unsigned char ins[2] = {0x96, streamState};
        WriteToSerial(portDescriptor, ins, 2);
        streamingState = streamState;
    }
    else
        error(INVALID_INSTRUCTION_MODE, (char*)"Function PAUSERESUMESTREAM");
}

void Create::sendIr(unsigned char byteValue)
{
    //Probar funcion 6/6
    if(mode == SAFE || mode == FULL)
    {
        unsigned char ins[2] = {0x97, byteValue};
        WriteToSerial(portDescriptor, ins, 2);
    }
    else
        error(INVALID_INSTRUCTION_MODE, (char*)"Function SENDIR");
}

void Create::script(vector<unsigned char> instructions)
{
    if(mode == PASSIVE || mode == SAFE || mode == FULL)
    {
        if(instructions.size()>=1 && instructions.size()<=100)
        {
            unsigned char* ins = new unsigned char[instructions.size() + 2];
            ins[0] = 0x98;
            ins[1] = instructions.size();
            for(int i = 0; i < instructions.size(); i++)
                ins[i + 2] = instructions[i];
            WriteToSerial(portDescriptor, ins, ins[1] + 2);
            delete []ins;
        }
    }
}
```

```

        ins[0] = 0x98;
        ins[1] = instructions.size();
        int size = instructions.size();
        for(int i = 2; i < size; i++)
        {
            ins[i] = instructions[i - 2];
        }
        WriteToSerial(portDescriptor, ins,
                      size + 2);
    }
    else
    { error(OUT_OF_RANGE, (char*)"Function SCRIPT"
          );}
}
else
{ error(INVALID_INSTRUCTION_MODE, (char*)"Function
      SCRIPT");}

}

void Create::script(unsigned char n, . . .)
{
    va_list args;
    va_start(args, n);
    if(mode == PASSIVE || mode == SAFE || mode == FULL)
    {
        if(n >= 1 && n <= 100)
        {
            unsigned char* ins = new unsigned char[n + 2];
            ins[0] = 0x98;
            ins[1] = n;
            for(int i = 2; i < n; i++)
            {
                ins[i] = (unsigned char)
                    va_arg(args, int);
            }
        }
    }
}
```

```

        WriteToSerial( portDescriptor ,ins ,n +
                      2) ;
    }
    else
    { error (OUT_OF_RANGE,( char*)" Function  SCRIPT"
              );}
}
else
{ error (INVALID_INSTRUCTION_MODE,( char*)" Function
          SCRIPT" );}

}

void Create :: playScript()
{
    if(mode == PASSIVE || mode == SAFE || mode == FULL)
    {
        unsigned char ins = 0x99;
        WriteToSerial( portDescriptor ,&ins ,1) ;
    }
    else
    { error (INVALID_INSTRUCTION_MODE,( char*)" Function
          SCRIPT" );}
}

void Create :: showScript()
{
    if(mode == PASSIVE || mode == SAFE || mode == FULL)
    {
        unsigned char ins = 0x9A;
        WriteToSerial( portDescriptor ,&ins ,1) ;
        unsigned char ins2 [101];
        usleep (120000);
        ReadFromSerial( portDescriptor ,ins2 ,101) ;
        int size = (int)ins2 [0];
        for(int i=0;i<size ;i++)
        {
            printRobotMessage(” % ” ,(int)ins2 [ i
                ]);
        }
    }
}
```

```

        }

        printRobotMessage(”\n”);

        if( streamingState )
            streamingState = false;
    }
    else
    { error( INVALID_INSTRUCTION_MODE,( char *)" Function
        SCRIPT" );
}

void Create::waitTime(unsigned char time)
{
    if( mode == PASSIVE || mode == SAFE || mode == FULL)
    {
        unsigned char ins[2]{0x9B,time};
        WriteToSerial( portDescriptor ,ins ,2 );
    }
    else
    { error( INVALID_INSTRUCTION_MODE,( char *)" Function
        SCRIPT" );
}

void Create::waitDistance(int distance)
{
    if( mode == FULL || mode == SAFE || mode == PASSIVE)
    {
        int16 data1 =.toInt16( distance );
        unsigned char ins[3]{0x9C,data1.H,data1.L};
        WriteToSerial( portDescriptor ,ins ,3 );
    }
    else
        error( INVALID_INSTRUCTION_MODE,( char *)"
            Function WAITDISTANCE" );
}

void Create::waitAngle(int angle)
{

```

```

if( mode == FULL || mode == SAFE || mode == PASSIVE)
{
    int16 data1 = toInt16( angle );
    unsigned char ins[3]{ 0x9D, data1.H, data1.L };
    WriteToSerial( portDescriptor , ins ,3 );
}
else
    error( INVALID_INSTRUCTION_MODE, ( char* )"
        Function WAITANGLE" );
}

void Create::waitEvent( unsigned char event )
{
    if( mode == FULL || mode == SAFE || mode == PASSIVE)
    {
        if( event >= -20 && event <=20)
        {
            unsigned char ins[2]{ 0x9E, event };
            WriteToSerial( portDescriptor , ins ,3 );
        }
        else
        { error( OUT_OF_RANGE, ( char* )" Function
            WAITEVENT" ); }
    }
    else
        error( INVALID_INSTRUCTION_MODE, ( char* )"
            Function WAITEVENT" );
}

void Create::error( int nError ,void *extra )
{
    int* extraAp = ( int* )extra ;
    char* extraInfoString = ( char* )extra ;
    if( robotVerbosity == VERBOSITY_NORMAL)
    {
        switch( nError )
        {
            case INVALID_DEMO:

```

```

        cout<<" Invalid Demo(1-9):\t"<*
                extraAp<<endl;
break;
case INVALID_BAUDRATE:
        cout<<" Invalid Baud code (0-11)\t"
                <<extraAp<<endl;
break;
case INVALID_INSTRUCTION_MODE:
        cout<<"The instruction cannot be
                executed in this operation mode:\t"
                <<extraInfoString<<endl;
break;
case OUT_OF_RANGE:
        cout<<" Parameter is out of range in
                "\t<<extraInfoString<<endl;
break;
case INVALID_MODE:
        cout<<" Selected mode doesn't exist
                in charMode function"\t<<endl;
break;
//case EXTRENAL_SENSORS_ERROR:
//        cout<<"Hubo un problema al
//                inicializar los sensores externos.
//                Continuando con los sensores desactivados
//                "\t<<endl;
//break;
default:
        cout<<" Unknown error"\t<<endl;
    }
}

int Create::getBaudCode(int baudCode)
{
    //functions returns 0 if invalid
    int tmp = 0;
    switch(baudCode)
    {

```

```
case BAUD300:  
    tmp = 300;  
break;  
case BAUD600:  
    tmp = 600;  
break;  
case BAUD1200:  
    tmp = 1200;  
break;  
case BAUD2400:  
    tmp = 2400;  
break;  
case BAUD4800:  
    tmp = 4800;  
break;  
case BAUD9600:  
    tmp = 9600;  
break;  
case BAUD14400:  
    tmp = 14400;  
break;  
case BAUD19200:  
    tmp = 19200;  
break;  
case BAUD28800:  
    tmp = 28800;  
break;  
case BAUD38400:  
    tmp = 38400;  
break;  
case BAUD57600:  
    tmp = 57600;  
break;  
case BAUD115200:  
    tmp = 115200;  
break;  
default:  
    error (INVALID_BAUDRATE,&baudCode);
```

```

        }
        return tmp;
    }

char* Create :: charMode(int mode)
{
    //La funcion regresa el cuando es incorrecto el modo
    char * tmp = (char*)"OFF";
    switch(mode)
    {
        case OFF:
            tmp = (char*)"OFF";
            break;
        case FULL:
            tmp = (char*)"FULL";
            break;
        case PASSIVE:
            tmp = (char*)"PASSIVE";
            break;
        case SAFE:
            tmp = (char*)"SAFE";
            break;
        default:
            error(INVALID_MODE, (char*)" ");
    }
    return tmp;
}

-----
//Metodos get
-----

bool Create :: getBumpRight()
{
    return bumpRight;
}

bool Create :: getBumpLeft()

```

```
{  
    return bumpLeft;  
}  
  
bool Create::getWheelDropRight()  
{  
    return wheelDropRight;  
}  
  
bool Create::getWheelDropLeft()  
{  
    return wheelDropLeft;  
}  
  
bool Create::getWheelDropCaster()  
{  
    return wheelDropCaster;  
}  
  
bool Create::getWallSeen()  
{  
    return wall;  
}  
  
bool Create::getCliffLeft()  
{  
    return cliffLeft;  
}  
  
bool Create::getCliffFrontLeft()  
{  
    return cliffFrontLeft;  
}  
  
bool Create::getCliffFrontRight()  
{  
    return cliffFrontRight;  
}
```

```
bool Create::getCliffRight()
{
    return cliffRight;
}

bool Create::getVirtualWall()
{
    return virtualWall;
}

bool Create::getLd0()
{
    return ld0;
}

bool Create::getLd1()
{
    return ld1;
}

bool Create::getLd2()
{
    return ld2;
}

bool Create::getRightWheel()
{
    return rightWheel;
}

bool Create::getLeftWheel()
{
    return leftWheel;
}

//Unused bytes 15-16.
```

```
unsigned char Create::getInfraredByte()
{
    return infraredbyte;
}

bool Create::getAdvanceBtn()
{
    return advancebtn;
}

bool Create::getPlayBtn()
{
    return playbtn;
}

int Create::getDistance()
{
    return distance;
}

int Create::getAngle()
{
    return angle;
}

unsigned char Create::getChargingState()
{
    return chargingstate;
}

int Create::getVoltage()
{
    return voltage;
}

int Create::getCurrent()
{
    return current;
```

```
}
```

```
unsigned char Create::getBatteryTemperature()
{
    return batterytemperature;
}
```

```
unsigned char Create::getBatteryCharge()
{
    return batterycharge;
}
```

```
unsigned char Create::getBatteryCapacity()
{
    return batterycapacity;
}
```

```
int Create::getWallSignal()
{
    return wallsignal;
}
```

```
int Create::getCliffLS()
{
    return cliffls;
}
```

```
int Create::getCliffFLS()
{
    return clifffls;
}
```

```
int Create::getCliffFRS()
{
    return clifffrs;
}
```

```
int Create::getCliffRS()
```

```
{  
    return cliffrs;  
}  
  
bool Create::getDigitalInput0()  
{  
    return digitalinput0;  
}  
  
bool Create::getDigitalInput1()  
{  
    return digitalinput1;  
}  
  
bool Create::getDigitalInput2()  
{  
    return digitalinput2;  
}  
  
bool Create::getDigitalInput3()  
{  
    return digitalinput3;  
}  
  
bool Create::getBaudRateChange() //DeviceDetect/  
    BaudRateChange  
{  
    return baudchangerate;  
}  
  
int Create::getCargoAnalogSignal()  
{  
    return cargoanalogsignal;  
}  
  
bool Create::getHomeBase()  
{  
    return homebase;
```

```
}
```

```
bool Create::getInternalCharger()
{
    return internalcharger;
}
```

```
unsigned char Create::getOIMode()
{
    return oimode;
}
```

```
unsigned char Create::getSongNumber()
{
    return songnumber;
}
```

```
bool Create::getSongPlaying()
{
    return songplaying;
}
```

```
unsigned char Create::getStreamPackets()
{
    return stremapackets;
}
```

```
int Create::getRequestedVelocity()
{
    return reqvelocity;
}
```

```
int Create::getRequestedRadius()
{
    return reqradius;
}
```

```
int Create::getRequestedRVelocity()
```

```
{  
    return reqrvelocity;  
}  
  
int Create::getRequestedLVelocity()  
{  
    return reqlvelocity;  
}  
  
bool Create::getStreamingState()  
{  
    return streamingState;  
}  
  
//  
//Packet methods  
//  
  
int Create::readBumpsAndWheelDrops(unsigned char *ins2)  
{  
  
    bumpRight = ins2[0] & 0x01;  
    bumpLeft = ins2[0] & 0x02;  
    wheelDropRight = ins2[0] & 0x04;  
    wheelDropLeft = ins2[0] & 0x08;  
    wheelDropCaster = ins2[0] & 0x10;  
  
    int retValue;  
    return toLittleEndian(ins2,1,&retValue,UNSIGNED);  
}  
  
int Create::readWall(unsigned char *ins2)  
{  
    wall = ins2[0];  
    int retValue;  
    return toLittleEndian(ins2,1,&retValue,UNSIGNED);  
}
```

```
int Create::readCliffLeft(unsigned char* ins2)
{
    cliffLeft = ins2[0];
    int retValue;
    return toLittleEndian(ins2,1,&retValue,UNSIGNED);
}

int Create::readCliffFrontLeft(unsigned char*ins2)
{
    cliffFrontLeft = ins2[0];
    int retValue;
    return toLittleEndian(ins2,1,&retValue,UNSIGNED);
}

int Create::readCliffFrontRight(unsigned char*ins2)
{
    cliffFrontRight = ins2[0];
    int retValue;
    return toLittleEndian(ins2,1,&retValue,UNSIGNED);
}

int Create::readCliffRight(unsigned char*ins2)
{
    cliffRight = ins2[0];
    int retValue;
    return toLittleEndian(ins2,1,&retValue,UNSIGNED);
}

int Create::readVirtualWall(unsigned char*ins2)
{
    virtualWall = ins2[0];
    int retValue;
    return toLittleEndian(ins2,1,&retValue,UNSIGNED);
}

int Create::readLSDriverAndWheelO(unsigned char*ins2)
{
    ld1 = ins2[0] & 0x01;
```

```
ld0 = ins2[0] & 0x02;
ld2 = ins2[0] & 0x04;
rightWheel = ins2[0] & 0x08;
leftWheel = ins2[0] & 0x10;
int retValue;
return toLittleEndian(ins2,1,&retValue,UNSIGNED);
}

//Two unused bytes 15-16

int Create::readInfraredByte(unsigned char *ins2)
{
    infraredbyte = ins2[0];
    int retValue;
    return toLittleEndian(ins2,1,&retValue,UNSIGNED);
}

int Create::readButtons(unsigned char *ins2)
{
    playbtn = ins2[0] & 0x01;
    advancebtn = ins2[0] & 0x04;
    int retValue;
    return toLittleEndian(ins2,1,&retValue,UNSIGNED);
}

int Create::readDistance(unsigned char *ins2)
{
    distance = 0;
    return toLittleEndian(ins2,2,&distance,SIGNED);
}

int Create::readAngle(unsigned char *ins2)
{
    angle = 0;
    return toLittleEndian(ins2,2,&angle,SIGNED);
}

int Create::readChargingState(unsigned char *ins2)
```

```
{  
    chargingstate = ins2[0];  
    int retValue;  
    return toLittleEndian(ins2,1,&retValue,UNSIGNED);  
}  
  
int Create::readVoltage(unsigned char *ins2)  
{  
    voltage = 0;  
    return toLittleEndian(ins2,2,&voltage,UNSIGNED);  
}  
  
int Create::readCurrent(unsigned char *ins2)  
{  
    current=0;  
    return toLittleEndian(ins2,2,&current,SIGNED);  
}  
  
int Create::readBatteryTemperature(unsigned char *ins2)  
{  
    batterytemperature = ins2[0];  
    int retValue;  
    return toLittleEndian(ins2,1,&retValue,SIGNED);  
}  
  
int Create::readBatteryCharge(unsigned char *ins2)  
{  
    batterycharge=0;  
    return toLittleEndian(ins2,2,&batterycharge,UNSIGNED);  
}  
  
int Create::readBatteryCapacity(unsigned char *ins2)  
{  
    batterycapacity=0;  
    return toLittleEndian(ins2,2,&batterycapacity,UNSIGNED);  
}
```

```
int Create::readWallSignal(unsigned char *ins2)
{
    wallsignal=0;
    return toLittleEndian(ins2,2,&wallsignal,UNSIGNED);
}

int Create::readCliffLS(unsigned char *ins2)
{
    cliffls=0;
    return toLittleEndian(ins2,2,&cliffls,UNSIGNED);
}

int Create::readCliffFLS(unsigned char *ins2)
{
    clifffls=0;
    return toLittleEndian(ins2,2,&clifffls,UNSIGNED);
}

int Create::readCliffFRS(unsigned char *ins2)
{
    clifffrs=0;
    return toLittleEndian(ins2,2,&clifffrs,UNSIGNED);
}

int Create::readCliffRS(unsigned char *ins2)
{
    cliffrs=0;
    return toLittleEndian(ins2,2,&cliffrs,UNSIGNED);
}

int Create::readDigitalInputs(unsigned char *ins2)
{
    digitalinput0 = ins2[0] & 0x01;
    digitalinput1 = ins2[0] & 0x02;
    digitalinput2 = ins2[0] & 0x04;
    digitalinput3 = ins2[0] & 0x08;
    baudchangerate = ins2[0] & 0x10;
    int retValue;
```

```
    return toLittleEndian(ins2,1,&retValue,SIGNED);  
}  
  
int Create::readCargoAnalogSignal(unsigned char *ins2)  
{  
    cargoanalogsignal=0;  
    return toLittleEndian(ins2,2,&cargoanalogsignal,  
        UNSIGNED);  
}  
  
int Create::readChargingSources(unsigned char *ins2)  
{  
    internalcharger = ins2[0] & 0x01;  
    homebase = ins2[0] & 0x02;  
    int retValue;  
    return toLittleEndian(ins2,1,&retValue,UNSIGNED);  
}  
  
int Create::readOIMode(unsigned char *ins2)  
{  
    oimode = ins2[0];  
    int retValue;  
    return toLittleEndian(ins2,1,&retValue,UNSIGNED);  
}  
  
int Create::readSongNumber(unsigned char *ins2)  
{  
    songnumber = ins2[0];  
    int retValue;  
    return toLittleEndian(ins2,1,&retValue,UNSIGNED);  
}  
  
int Create::readSongPlaying(unsigned char *ins2)  
{  
    songplaying = ins2[0];  
    int retValue;  
    return toLittleEndian(ins2,1,&retValue,UNSIGNED);  
}
```

```
int Create::readStreamPackets(unsigned char *ins2)
{
    streampackets = ins2[0];
    int retValue;
    return toLittleEndian(ins2,1,&retValue,UNSIGNED);
}

int Create::readReqVelocity(unsigned char *ins2)
{
    reqvelocity=0;
    return toLittleEndian(ins2,2,&reqvelocity,SIGNED);
}

int Create::readReqRadius(unsigned char *ins2)
{
    reqradius=0;
    return toLittleEndian(ins2,2,&reqradius,SIGNED);
}

int Create::readReqRVelocity(unsigned char *ins2)
{
    reqrvelocity=0;
    return toLittleEndian(ins2,2,&reqrvelocity,SIGNED);
}

int Create::readReqLVelocity(unsigned char *ins2)
{
    reqlvelocity=0;
    return toLittleEndian(ins2,2,&reqlvelocity,SIGNED);
}

int Create::updateSensor(unsigned char packetID, int
instructionSize)
{
    unsigned char ins2[instructionSize];
    usleep(120000);
    ReadFromSerial(portDescriptor,ins2,instructionSize);
```

```
int ret = -65536;
int offset=0;
switch( packetID )
{
    case 0: case 1: case 6: case 7: //packets
        0,1,6 and 7 start here
        ret = readBumpsAndWheelDrops( ins2+
            offset );
        if( packetID < 7 )
        {
            offset+= getSizePacket (7);
        }
        if( packetID == 7 )
            break;
    case 8:
        ret = readWall( ins2+offset );
        if( packetID < 7 )
        {
            offset+= getSizePacket (8);
        }
        if( packetID == 8 )
            break;
    case 9:
        ret = readCliffLeft( ins2+offset );
        if( packetID < 7 )
        {
            offset+= getSizePacket (9);
        }
        if( packetID == 9 )
            break;
    case 10:
        ret = readCliffFrontLeft( ins2+offset
            );
        if( packetID < 7 )
        {
            offset+= getSizePacket (10);
        }
        if( packetID == 10 )
```

```
        break;
case 11:
    ret = readCliffFrontRight(ins2+
        offset);
    if(packetID < 7)
    {
        offset+= getSizePacket(11);
    }
    if(packetID == 11)
        break;
case 12:
    ret = readCliffRight(ins2+offset);
    if(packetID < 7)
    {
        offset+= getSizePacket(12);
    }
    if(packetID == 12)
        break;
case 13:
    ret = readVirtualWall(ins2+offset);
    if(packetID < 7)
    {
        offset+= getSizePacket(13);
    }
    if(packetID == 13)
        break;
case 14:
    ret = readLSDriverAndWheelO(ins2+
        offset);
    if(packetID < 7)
    {
        offset+= getSizePacket(14);
    }
    if(packetID == 14)
        break;
case 15:
case 16:
```

```

/* Unused bytes: Two unused bytes are
   sent after the overcurrent
   byte when the requested packet is 0,
   1, or 6. The value of the
   two unused bytes is always 0.*/

if(packetID < 7)
{
    offset+= getSizePacket(16)+
        getSizePacket(15);
}
if(packetID == 15||packetID == 16||
    packetID == 1) //Ends packet 1
    break;
case 2: case 17: //packet 2 and 17 start here
    ret = readInfraredByte(ins2+offset);
    if(packetID < 7)
    {
        offset+= getSizePacket(17);
    }
    if(packetID == 17)
        break;
case 18:
    ret = readButtons(ins2+offset);
    if(packetID < 7)
    {
        offset+= getSizePacket(18);
    }
    if(packetID == 18)
        break;
case 19:
    ret = readDistance(ins2+offset);
    if(packetID < 7)
    {
        offset+= getSizePacket(19);
    }
    if(packetID == 19)

```

```

break;
case 20:
    ret = readAngle( ins2+offset );
    if( packetID < 7)
    {
        offset+= getSizePacket(20);
    }
    if( packetID == 20|| packetID == 2) //  

        Packet 2 ends here
        break;
case 3:case 21:
    ret = readChargingState( ins2+offset )
    ;
    if( packetID < 7)
    {
        offset+= getSizePacket(21);
    }
    if( packetID == 21)
        break;
case 22:
    ret = readVoltage( ins2+offset );
    if( packetID < 7)
    {
        offset+= getSizePacket(22);
    }
    if( packetID == 22)
        break;
case 23:
    ret = readCurrent( ins2+offset );
    if( packetID < 7)
    {
        offset+= getSizePacket(23);
    }
    if( packetID == 23)
        break;
case 24:
    ret = readBatteryTemperature( ins2+
        offset );

```

```

if( packetID < 7)
{
    offset+= getSizePacket(24);
}
if( packetID == 24)
    break;
case 25:
    ret = readBatteryCharge( ins2+offset )
    ;
    if( packetID < 7)
    {
        offset+= getSizePacket(25);
    }
    if( packetID == 25)
        break;
case 26:
    ret = readBatteryCapacity( ins2+
        offset );
    if( packetID < 7)
    {
        offset+= getSizePacket(26);
    }
    if( packetID == 26 || packetID == 0 ||
        packetID == 3) //Ends packets 0
        and 3
        break;
case 4: case 27:
    ret = readWallSignal( ins2+offset );
    if( packetID < 7)
    {
        offset+= getSizePacket(27);
    }
    if( packetID == 27)
        break;
case 28:
    ret = readCliffLS( ins2+offset );
    if( packetID < 7)
    {

```

```
        offset+= getSizePacket(28);
    }
    if(packetID == 28)
        break;
case 29:
    ret = readCliffFLS(ins2+offset);
    if(packetID < 7)
    {
        offset+= getSizePacket(29);
    }
    if(packetID == 29)
        break;
case 30:
    ret = readCliffFRS(ins2+offset);
    if(packetID < 7)
    {
        offset+= getSizePacket(30);
    }
    if(packetID == 30)
        break;
case 31:
    ret = readCliffRS(ins2+offset);
    if(packetID < 7)
    {
        offset+= getSizePacket(31);
    }
    if(packetID == 31)
        break;
case 32:
    ret = readDigitalInputs(ins2+offset)
        ;
    if(packetID < 7)
    {
        offset+= getSizePacket(32);
    }
    if(packetID == 32)
        break;
case 33:
```

```

ret = readCargoAnalogSignal( ins2+
    offset );
if( packetID < 7)
{
    offset+= getSizePacket (33);
}
if( packetID == 33)
    break;
case 34:
    ret = readChargingSources( ins2+
        offset );
if( packetID < 7)
{
    offset+= getSizePacket (34);
}
if( packetID == 34 || packetID == 4)
    //Ends packet 4
    break;
case 5: case 35:           //Packet 5 starts
    ret = readOIMode( ins2+offset );
    if( packetID < 7)
{
    offset+= getSizePacket (35);
}
if( packetID == 35)
    break;
case 36:
    ret = readSongNumber( ins2+offset );
    if( packetID < 7)
{
    offset+= getSizePacket (36);
}
if( packetID == 36)
    break;
case 37:
    ret = readSongPlaying( ins2+offset );
    if( packetID < 7)
{

```

```
        offset+= getSizePacket(37);
    }
    if(packetID == 37)
        break;
case 38:
    ret = readStreamPackets(ins2+offset)
    ;
    if(packetID < 7)
    {
        offset+= getSizePacket(38);
    }
    if(packetID == 38)
        break;
case 39:
    ret = readReqVelocity(ins2+offset);
    if(packetID < 7)
    {
        offset+= getSizePacket(39);
    }
    if(packetID == 39)
        break;
case 40:
    ret = readReqRadius(ins2+offset);
    if(packetID < 7)
    {
        offset+= getSizePacket(40);
    }
    if(packetID == 40)
        break;
case 41:
    ret = readReqRVelocity(ins2+offset);
    if(packetID < 7)
    {
        offset+= getSizePacket(41);
    }
    if(packetID == 41)
        break;
case 42:
```

```
    ret = readReqLVelocity( ins2+offset ) ;
    if( packetID < 7 )
    {
        offset+= getSizePacket( 42 ) ;
    }
    if( packetID == 42 || packetID==5 ||
        packetID == 6 ) //Ends packets 5
        and 6
        break ;
    }

    return ret ;
}

int Create::getExternalSensors( int destinationArray [
    NUMBER_OF_SENSORS] )
{
    GetSensors( destinationArray ) ;
}

int Create::getExternalNthSensor( int n )
{
    GetNthSensor( n ) ;
}

bool Create::getExternalSensorsEnabledStatus()
{
    return externalSensorsEnabled ;
}

void Create::setVerbosity( t_verbosity level )
{
    SetSensorVerbosity( level ) ;
    SetSerialPortVerbosity( level ) ;
    robotVerbosity = level ;
}
void Create::printRobotMessage( const char* message , . . . )
{
```

```

if( robotVerbosity == VERBOSITY_NORMAL)
{
    va_list arguments;
    va_start(arguments, message);
    vprintf(message, arguments);
    va_end(arguments);
}
}

int Create::toLittleEndian( unsigned char* source, int nbytes
    , int* destination, boolSigned sign)
{
    char* tmp;

    if(sign == UNSIGNED)
    {
        *destination = 0;

    }
    else
    {
        if(source[0]&0x80) //if it is negative (2's
            complement)
        {
            *destination = -1;//0xFFFFFFFF... only
            1's
        }
        else
        {
            *destination = 0;
        }
    }

    tmp = (char*) destination;
    for(int i = nbytes - 1; i >= 0; i--)
    {

```

```

*(tmp+((nbytes-1) - i)) = source [ i ];

}

return *destination;
}

```

9.8. UDG_Create.h

```

#ifndef UDG_CREATE_H
#define UDG_CREATE_H

#include<string>
#include<sstream>
#include<vector>
#include<stdio.h>
#include <iostream>
#include <unistd.h>
#include <stdarg.h>
#include <string.h>
using namespace std;
#include <math.h>

enum errorCodes {INVALID_DEMO, INVALID_BAUDRATE,
    INVALID_INSTRUCTION_MODE, OUT_OF_RANGE, INVALID_MODE/* ,
    EXTRENAL_SENSORS_ERROR*/};
enum modes{OFF, PASSIVE, SAFE, FULL};
enum baudCode{BAUD300, BAUD600, BAUD1200, BAUD2400, BAUD4800,
    BAUD9600, BAUD14400, BAUD19200, BAUD28800, BAUD38400, BAUD57600
    , BAUD115200};
enum chargingstates {NOT_CHARGING, RECONDITIONING_CHARGING,
    FULL_CHARGING, TRICKLE_CHARGING, WAITING,
    CHARGING_FAULT_CONDITION};

```

```

enum oimodes{OIOFF, OIPASSIVE, OISAFE, OIFULL};
enum infraredbytechars{IRLEFT=129,IRFORWARD,IRRIGHT,IRSPOT,
    IRMAX,IRSMALL,IRMEDIUM,IRLARGE,IRPAUSE,IRPOWER,
    IRARC_FORWARD_LEFT,
    IRARC_FORWARD_RIGHT,IRDIVESTOP,
    IRSENDALL,IRSEEKDCK,IRRESERVED,
    IRRED,IRGREEN,IRFORCEFIELD,IRREDGREEN
    ,IRREDFORCEFIELD,IRGREENFORCEFIELD,
    IRREDGREENFORCEFIELD};

typedef enum VerbosityLevels {VERBOSITY_NORMAL,
    VERBOSITY_FILE,VERBOSITY_OFF, VERBOSITY_NUMBER_OF_LEVELS}
    t_verbosity;
typedef enum BoolSigned {SIGNED, UNSIGNED} boolSigned;
#define NUMBER_OF_SENSORS 14

typedef struct int_16
{
    unsigned char H; //High byte
    unsigned char L; //Low byte
} int16;

class Create
{
public:
    Create();
    Create(string);
    Create(string portName1,t_verbosity);
    ~Create();
    string getPortName();
    bool isCharging();
    void start();
    void baud(unsigned char);
    void control();
    void safe();
    void full();
}

```

```
void spot();
void cover();
void demo(unsigned char);
void drive(int ,int );
void lowSideDrivers(unsigned char);
void leds(unsigned char ,unsigned char ,
          unsigned char );
void song(unsigned char ,unsigned char ,... );
void playSong(unsigned char );
int sensors(unsigned char );
int getSizePacket(int );
void coverAndDock();
void pwmLowSideDrivers(unsigned char ,
                       unsigned char ,unsigned char );
void driveDirect(int ,int );
void digitalOutputs(unsigned char );
//void stream(vector<unsigned char>);
void stream(unsigned char* destinationBuffer
            ,void* thread ,int n,... );
//void ThreadedReadStream(int ,char* ,int );
void queryList(vector<unsigned char>);
void pauseResumeStream(bool );
void sendIr(unsigned char );
void script(vector<unsigned char>);
void script(unsigned char ,... );
void playScript();
void showScript();
void waitTime(unsigned char );
void waitDistance(int );
void waitAngle(int );
void waitEvent(unsigned char );
void error(int ,void* );
int getBaudCode(int );
char* charMode(int );
//void ThreadedReadStream(int portDescriptor
//                        ,unsigned char* buffer ,int numberOfBytes);
///////////////
//Get Prototypes
```

```
///////////
bool getBumpRight() ;
bool getBumpLeft() ;
bool getWheelDropRight() ;
bool getWheelDropLeft() ;
bool getWheelDropCaster() ;
bool getWallSeen() ;
bool getCliffLeft() ;
bool getCliffFrontLeft() ;
bool getCliffFrontRight() ;
bool getCliffRight() ;
bool getVirtualWall() ;
bool getLd0() ;
bool getLd1() ;
bool getLd2() ;
bool getRightWheel() ;
bool getLeftWheel() ;
//Unused bytes 15–16.
unsigned char getInfraredByte() ;
bool getAdvanceBtn() ;
bool getPlayBtn() ;
int getDistance() ;
int getAngle() ;
unsigned char getChargingState() ;
int getVoltage() ;
int getCurrent() ;
unsigned char getBatteryTemperature() ;
unsigned char getBatteryCharge() ;
unsigned char getBatteryCapacity() ;
int getWallSignal() ;
int getCliffLS() ;
int getCliffFLS() ;
int getCliffFRS() ;
int getCliffRS() ;
bool getDigitalInput0() ;
bool getDigitalInput1() ;
bool getDigitalInput2() ;
bool getDigitalInput3()
```

```

bool getBaudRateChange(); //DeviceDetect/
    BaudRateChange
int getCargoAnalogSignal();
bool getHomeBase();
bool getInternalCharger();
unsigned char getOIMode();
unsigned char getSongNumber();
bool getSongPlaying();
unsigned char getStreamPackets();
int getRequestedVelocity();
int getRequestedRadius();
int getRequestedRVelocity();
int getRequestedLVelocity();
bool getStreamingState();
int getExternalSensors(int [NUMBER_OF_SENSORS
    ]);
int getExternalNthSensor(int);
bool getExternalSensorsEnabledStatus();
void setVerbosity(t_verbosity);
void printRobotMessage(const char* message
    , ...);
int toLittleEndian(unsigned char* source, int
    nbytes, int* destination, boolSigned sign);

private:
    /////////////////////
    //Read Prototypes
    /////////////////////
int readBumpsAndWheelDrops(unsigned char *);
int readWall(unsigned char *);
int readCliffLeft(unsigned char *);
int readCliffFrontLeft(unsigned char *);
int readCliffFrontRight(unsigned char *);
int readCliffRight(unsigned char *);
int readVirtualWall(unsigned char *);
int readLSDriverAndWheelO(unsigned char *);
    //Two unused bytes 15-16
int readInfraredByte(unsigned char *);
int readButtons(unsigned char );

```

```
int readDistance(unsigned char *);
int readAngle(unsigned char *);
int readChargingState(unsigned char *);
int readVoltage(unsigned char *);
int readCurrent(unsigned char *);
int readBatteryTemperature(unsigned char *);
int readBatteryCharge(unsigned char *);
int readBatteryCapacity(unsigned char *);
int readWallSignal(unsigned char *);
int readCliffLS(unsigned char *);
int readCliffFLS(unsigned char *);
int readCliffFRS(unsigned char *);
int readCliffRS(unsigned char *);
int readDigitalInputs(unsigned char *);
int readCargoAnalogSignal(unsigned char *);
int readChargingSources(unsigned char *);
int readOIMode(unsigned char *);
int readSongNumber(unsigned char *);
int readSongPlaying(unsigned char *);
int readStreamPackets(unsigned char *);
int readReqVelocity(unsigned char *);
int readReqRadius(unsigned char *);
int readReqRVelocity(unsigned char *);
int readReqLVelocity(unsigned char *);
int updateSensor(unsigned char, int);
void commonInitializationProcedures(string,
                                     bool);
string portName;
int mode;
//TODO: implement charging by hardware
bool charging;
int portDescriptor;
int baudRate;

bool bumpRight;
bool bumpLeft;
bool wheelDropRight;
bool wheelDropLeft;
```

```
bool wheelDropCaster;
bool wall;
bool cliffLeft;
bool cliffFrontLeft;
bool cliffFrontRight;
bool cliffRight;
bool virtualWall;
bool ld0;
bool ld1;
bool ld2;
bool rightWheel;
bool leftWheel;
//Unused bytes 15–16.
unsigned char infraredbyte;
bool advancebtn;
bool playbtn;
int distance;
int angle;
unsigned char chargingstate;
int voltage;
int current;
unsigned char batterytemperature;
int batterycharge;
int batterycapacity;
int wallsignal;
int cliffls;
int clifffls;
int clifffrs;
int cliffrs;
bool digitalinput0;
bool digitalinput1;
bool digitalinput2;
bool digitalinput3;
bool baudchangerate;
int cargoanalogsignal;
bool homebase;
bool internalcharger;
unsigned char oimode;
```

```

    unsigned char songnumber;
    bool songplaying;
    unsigned char streampackets;
    int reqvelocity;
    int reqradius;
    int reqrvelocity;
    int reqlvelocity;
    bool streamingState;
    bool externalSensorsEnabled;
    t_verbosity robotVerbosity;
    int16.toInt16(int integer);
};

#endif

```

9.9. USBLayer.cpp

```

/* This is the host side for the microcontroller's USB
   firmware
This file is loosely based on "USB and PIC: quick guide to
an USB HID framework"
by Alberto Maccioni, available at http://openprog.altervista
.org/USB-firm-eng.html
*/
#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <linux/hiddev.h>
#include <string.h>
#include <unistd.h>
#include <ctype.h>
#include <fcntl.h>
#include "USBLayer.h"

#define BUFFER_SIZE 64
#define REPORT_SIZE 64
#define SLEEP_MILISECONDS 100

```

```

int fd = -1;
char devicePath[256];
unsigned char inBuffer[BUFFER_SIZE];
unsigned char outBuffer[BUFFER_SIZE];

//these structs hold information about the reports
struct hiddev_report_info inReportInfo;
struct hiddev_report_info outReportInfo;
//information about the endpoint usage
struct hiddev_usage_ref_multi inUsage;
struct hiddev_usage_ref_multi outUsage;

struct hiddev_devinfo deviceInfo;

t_verbosity USBVerbosity = VERBOSITY_NORMAL;

int initializeUSB(int VID, int PID)
{
    int i;
    int MAX_DESCRIPTORS = 50;
    printUSBMessage(" Searching Create's Sensors USB
                    Interface (VID 0x%04X and PID 0x%04X)\n", VID, PID)
    ;
    for( i = 0; i < MAX_DESCRIPTORS ; i++)
    {
        sprintf(devicePath, "/dev/usb/hiddev%d", i);
        fd = open(devicePath, O_RDONLY);
        if(fd >= 0) //if file was opened properly
        {
            ioctl(fd, HIDIOCGDEVINFO, &
                  deviceInfo);
            if(deviceInfo.vendor == VID &&
                deviceInfo.product == PID)
            {
                printf(" Device found: %s\n",
                       devicePath);
                break;
            }
        }
    }
}

```

```
        }
```

```
    }
```

```
    }
```

```
    if ( i >= MAX_DESCRIPTORS)
```

```
{
```

```
    printUSBMessage("USB sensors not found!\n");
```

```
    return -1;
```

```
}
```

```
}
```

```
void GetUSBData(unsigned char* destinationBuffer)
```

```
{
```

```
    //Lots of info about the next part here http://www.
```

```
wetlogic.net/hiddev/
```

```
    //Now that usb port is open, this is the actual
```

```
    initialization
```

```
    outReportInfo.report_type = HID_REPORT_TYPE_OUTPUT;
```

```
    outReportInfo.report_id = HID_REPORT_ID_FIRST;
```

```
    outReportInfo.num_fields = 1;
```

```
    inReportInfo.report_type = HID_REPORT_TYPE_INPUT;
```

```
    inReportInfo.report_id = HID_REPORT_ID_FIRST;
```

```
    inReportInfo.num_fields = 1;
```

```
    outUsage.uref.report_type = HID_REPORT_TYPE_OUTPUT;
```

```
    outUsage.uref.report_id = HID_REPORT_ID_FIRST;
```

```
    outUsage.uref.field_index = 0;
```

```
    outUsage.uref.usage_index = 0;
```

```
    outUsage.num_values = REPORT_SIZE;
```

```

inUsage. uref. report_type = HID_REPORT_TYPE_INPUT;
inUsage. uref. report_id = HID_REPORT_ID_FIRST;
inUsage. uref. field_index = 0;
inUsage. uref. usage_index = 0;
inUsage. num_values = REPORT_SIZE;

/*
HIDIOCSUSAGE: Sets the value of a usage in an output
report.
HIDIOCSREPORT: Instructs the kernel to send a report
to the device. This report can be filled in by
the user through HIDIOCSUSAGE calls (below) to
fill in individual usage values in the report
before sending the report in full to the device.

HIDIOCSUSAGE: Returns the value of a usage in a
hiddev_usage_ref structure. The usage to be
retrieved can be specified as above, or the user
can choose to fill in the report_type field and
specify the report_id as HID_REPORT_ID_UNKNOWN. In
this case, the hiddev_usage_ref will be filled in
with the report and field infomation associated
with this usage if it is found.
HIDIOCGREPORTInstructs the kernel to get a feature
or input report from the device, in order to
selectively update the usage structures (in
contrast to INITREPORT).

*/
outUsage.values[0] = 0x37; //GET_SENSORS_REPORT_CODE
//0x37 is the report code to read all of the PIC'
s ADC's

//write

```

```
    ioctl(fd,HIDIOCSUSAGES, &outUsage); //set the output
                                             report
    ioctl(fd,HIDIOCSREPORT, &outReportInfo); //send the
                                              output report
    usleep(SLEEP_MILISECONDS*1000);
    //read
    ioctl(fd,HIDIOCGUSAGES, &inUsage); //set the input
                                             report
    ioctl(fd,HIDIOCGREPORT, &inReportInfo); //get the
                                              input report

    //printf("-->");
    for(int i = 1 ; i < REPORT_SIZE ; i++)
    {destinationBuffer[i] = inUsage.values[i];}

    printf("— %X —", destinationBuffer[28]);

}

void CloseUSB()
{
    close(fd);
}

void printUSBMessage(const char* message,...)
{
    if( USBVerbosity == VERBOSITY_NORMAL)
    {
        va_list arguments;
        va_start(arguments, message);
        vprintf(message, arguments);
        va_end(arguments);
    }
}

void SetUSBVerbosity(t_verbosity level)
{
    if(level < VERBOSITY_NUMBER_OF_LEVELS)
```

```
{  
    USBVerbosity = level;  
}  
}
```

9.10. USBLayer.h

```
#ifndef USBLAYER_H  
#define USBLAYER_H  
#define GET_SENSORS_REPORT_CODE 0x37  
  
#include <errno.h>  
#include <signal.h>  
#include <string.h>  
#include <stdio.h>  
  
#include <unistd.h>  
#include <stdarg.h>  
#include "UDG_Create.h"  
  
int initializeUSB(int VID, int PID);  
void printUSBMessage(const char* message, ...);  
void GetUSBData(unsigned char* destinationBuffer);  
void CloseUSB();  
void printUSBMessage(const char* message, ...);  
void SetUSBVerbosity(t_verbosity level);  
#endif
```