



Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

**DESARROLLO E IMPLEMENTACIÓN DE UN ACTUADOR PARA UN SISTEMA
ROBÓTICO LEGO® UTILIZANDO PROTOCOLO DE COMUNICACIÓN I²C**

Darling Anabella Luarte Picén

Asesorado por el Dr. Juan Carlos Córdova Zeceña

Guatemala, mayo de 2013

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



FACULTAD DE INGENIERÍA

**DESARROLLO E IMPLEMENTACIÓN DE UN ACTUADOR PARA UN SISTEMA
ROBÓTICO LEGO® UTILIZANDO PROTOCOLO DE COMUNICACIÓN I²C**

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

DARLING ANABELLA LUARTE PICÉN

ASESORADO POR EL Dr. JUAN CARLOS CÓRDOVA ZECEÑA

AL CONFERÍRSELE EL TÍTULO DE

INGENIERA ELECTRÓNICA

GUATEMALA, MAYO DE 2013

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA



NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Murphy Olympto Paiz Recinos
VOCAL I	Ing. Alfredo Enrique Beber Aceituno
VOCAL II	Ing. Pedro Antonio Aguilar Polanco
VOCAL III	Inga. Elvia Miriam Ruballos Samayoa
VOCAL IV	Br. Walter Rafael Véliz Muñoz
VOCAL V	Br. Sergio Alejandro Donis Soto
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

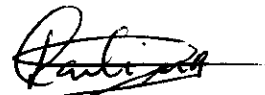
DECANO	Ing. Murphy Olympto Paiz Recinos
EXAMINADOR	Ing. Carlos Eduardo Guzmán Salazar
EXAMINADOR	Ing. Enrique Edmundo Ruiz Carballo
EXAMINADOR	Ing. Armando Alonso Rivera Carrillo
SECRETARIA	Inga. Marcia Ivónne Véliz Vargas

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

DESARROLLO E IMPLEMENTACIÓN DE UN ACTUADOR PARA UN SISTEMA ROBÓTICO LEGO® UTILIZANDO PROTOCOLO DE COMUNICACIÓN I²C

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha 15 de mayo de 2009.



Darling Anabella Luarte Picén



Guatemala, 11 de febrero de 2013

FACULTAD DE INGENIERIA

Ingeniero Carlos Eduardo Guzmán Salazar
Coordinador del Área de Electrónica
Escuela de Ingeniería Mecánica-Eléctrica
Facultad de Ingeniería
USAC

Estimado Ingeniero Guzmán.

Deseo hacer del conocimiento de la Escuela de Ingeniería Mecánica-Eléctrica que, como asesor de la estudiante *Darling Anabella Luarte Picén*, quien se identifica con carné 200212556, he leído, revisado y corregido su trabajo de graduación titulado *Desarrollo e implementación de un actuador para un sistema robótico LEGO® utilizando protocolo de comunicación I²C* y considero que el mismo llena o excede los requisitos en cuanto a contenido, alcance y realización requeridos de trabajos de esta naturaleza.

El contenido teórico expuesto en su trabajo de graduación está respaldado por la realización práctica de un actuador inteligente que se interconecta con un sistema robótico LEGO mediante protocolo I²C; constituye así un ejemplo de diseño en el que se demuestra la capacidad de la estudiante para aplicar los conocimientos, habilidades y buenas prácticas adquiridas en nuestra Facultad y en nuestra Escuela, sin contar también la utilidad que dicha tesis pueda proporcionar a futuros estudiantes interesados en temas de robótica, de comunicaciones, o de sistemas embebidos inteligentes.

Por lo tanto declaro que apruebo el trabajo de graduación de la estudiante *Darling Anabella Luarte Picén* titulado *Desarrollo e implementación de un actuador para un sistema robótico LEGO® utilizando protocolo de comunicación I²C* y firmo la presente para que la interesada haga uso de la misma en los trámites de graduación que correspondan.

Atentamente,

Dr. Juan Carlos Córdova Zeceña



UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA



ACULTAD DE INGENIERIA

Ref. EIME 10. 2013.

Guatemala, 22 de FEBRERO 2013.

Señor Director
Ing. Guillermo Antonio Puente Romero
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC.

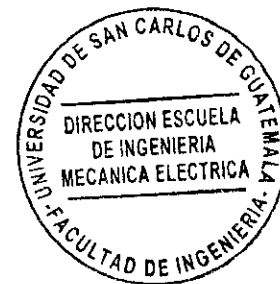
Señor Director:

**Me permito dar aprobación al trabajo de Graduación titulado:
“DESARROLLO E IMPLEMENTACIÓN DE UN ACTUADOR
PARA UN SISTEMA ROBÓTICO LEGO® UTILIZANDO
PROTOCOLO DE COMUNICACIÓN I²C.”, de la estudiante
Darling Anabella Luarte Picén, que cumple con los requisitos
establecidos para tal fin.**

Sin otro particular, aprovecho la oportunidad para saludarle.

Atentamente,
DID Y ENSEÑAD A TODOS

Ing. Carlos Eduardo Guzmán Salazar
Coordinador Área Electrónica



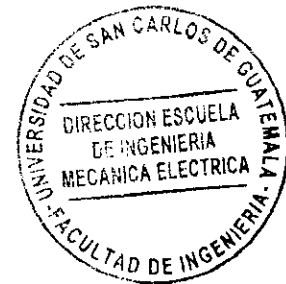
CEGS/sro



REF. EIME 11. 2013.

El Director de la Escuela de Ingeniería Mecánica Eléctrica, después de conocer el dictamen del Asesor, con el Visto Bueno del Coordinador de Área, al trabajo de Graduación del estudiante; DARLING ANABELLA LUARTE PICÉN titulado: “DESARROLLO E IMPLEMENTACIÓN DE UN ACTUADOR PARA UN SISTEMA ROBÓTICO LEGO® UTILIZANDO PROTOCLO DE COMUNICACIÓN PC.”, procede a la autorización del mismo.

Ing. Guillermo Antonio Puente Romero



GUATEMALA, 12 DE MARZO 2013.

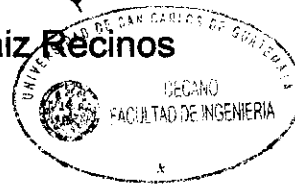


Ref. DTG.335-2013

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica, al trabajo de graduación titulado: **DESARROLLO E IMPLEMENTACIÓN DE UN ACTUADOR PARA UN SISTEMA ROBÓTICO LEGO® UTILIZANDO PROTOCOLO DE COMUNICACIÓN I²C**, presentado por la estudiante universitaria **Darling Anabella Luarte Picén**, autoriza la impresión del mismo.

IMPRÍMASE.

Ing. Murphy Olympo Paiz Recinos
Decano



Guatemala, mayo de 2013

/cc

ACTO QUE DEDICO A:

- Dios** Por ser mí guía y apoyo en los momentos más importantes de mi vida para concluir con éxito mis estudios superiores.
- Mis padres** Rolando Raúl Luarte Gómez y Rosa Clemencia Picén Díaz (q.e.p.d.). Su amor será siempre mi inspiración.
- Mis hermanos** Wendy, Juan Carlos, Christiam Luarte Picén, por brindarme su apoyo y cariño en todo momento.
- Mis familiares** Especialmente a mis tías Alicia, Elida, Olivia y Elvira Picén, por su apoyo y amor incondicional. A mis primos, en especial a Jéssica Oliva, José María Campos, Claudia y Leticia Marín, por el gran apoyo y cariño brindado.
- Mi novio** Mynor Marroquín, por su cariño, paciencia y apoyo. Gracias por estar conmigo y ser como eres.
- Mis amigos** Por su gran apoyo y amistad, Dios los puso en mi camino en el momento y hora precisa. Gracias por su cariño.

AGRADECIMIENTOS A:

La Universidad de San Carlos de Guatemala	Alma máter, donde se desarrollaron mis pensamientos académicos.
Facultad de Ingeniería	Por brindarme los conocimientos que me permiten desarrollarme como profesional y de esta forma contribuir a la sociedad.
Mis catedráticos	En especial a los ingenieros Enrique Ruiz, Carlos Guzmán y al Dr. Juan Carlos Córdova, por sus conocimientos y por ser ejemplo para mi vida profesional.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES	VII
RESUMEN	XIII
OBJETIVOS.....	XV
INTRODUCCIÓN	XVII
1. PLATAFORMA LEGO® MINDSTORMS® NXT	1
1.1. Introducción	1
1.2. Características principales de hardware.....	2
1.2.1. Puertos de salida	3
1.2.2. Puertos de entrada	6
1.2.3. Puertos con interfaz I ² C	8
1.3. Arquitectura del bloque NXT.....	10
1.3.1. Sistema de archivos	11
1.3.2. Interfaces digitales.....	12
1.4. Bloque programable NXT	13
1.4.1. Programación en software NXT-G.....	13
1.4.1.1. Entorno de programación NXT-G	14
1.4.2. Lenguaje de programación LabVIEW	20
1.4.3. Programación en LabVIEW para NXT-G	23
1.5. Sumario	27
2. ACTUADOR PINZA INTELIGENTE	29
2.1. Propuesta del actuador pinza inteligente.....	29
2.2. Estructura general del sistema	30
2.2.1. Sistema mecánico y eléctrico	30

2.2.1.1.	Servomotor	31
2.2.1.2.	Control.....	31
2.2.1.3.	Funcionamiento.....	32
2.2.1.4.	Modulación por ancho de pulso PWM.....	33
2.2.2.	Sistema de control.....	34
2.2.2.1.	Microcontrolador PIC.....	34
2.2.2.1.1.	Características	35
2.2.2.1.2.	El oscilador.....	36
2.2.2.1.3.	Puertos de E/S	37
2.2.2.1.4.	Módulos.....	38
2.2.2.2.	Herramienta de programación.....	41
2.2.2.2.1.	Proceso de desarrollo ...	41
2.2.2.2.2.	Programa MPLAB	42
2.2.3.	Sistema de comunicación.....	43
2.2.3.1.	Bus I ² C	43
2.2.3.2.	Transferencia de datos.....	45
2.2.3.3.	Protocolo de comunicación I ² C.....	46
2.2.3.4.	Formatos de transferencia de datos	48
2.2.3.5.	PIC16F877A: módulo MSSP	53
2.2.3.5.1.	Registros	54
2.2.3.5.2.	Operación.....	56
2.2.3.5.3.	Modo esclavo	57
2.2.3.5.4.	Direccionamiento.....	57
2.2.3.5.5.	Recepción	58
2.2.3.5.6.	Transmisión.....	58
2.3.	Elementos de programación del módulo bloque Pinza NXT en LabVIEW y Toolkit para LEGO® MINDSTORMS® NXT	59
2.3.1.	Dispositivo digital E/S.....	61

	2.3.1.1.	Configuración del puerto E/S.....	61
	2.3.1.2.	Comunicación de baja velocidad	62
	2.3.1.3.	Secuencia de control	67
	2.3.1.4.	Conexión de ícono.....	67
	2.3.1.5.	Edición de ícono	69
	2.3.2.	Creación de nuevos bloques de programa	69
	2.3.2.1.	Ícono Pinza NXT	71
	2.3.2.2.	Ícono de arrastre.....	72
	2.3.2.3.	Bloque Pinza NXT.vi.....	72
	2.3.2.4.	Bloque Pinza NXTSub.vi.....	75
	2.3.2.5.	Configuración vi	76
	2.3.2.6.	Otros archivos de soporte.....	77
	2.3.2.7.	Importar y exportar.....	77
3.		DISEÑO E IMPLEMENTACIÓN DEL ACTUADOR PINZA INTELIGENTE.....	79
	3.1.	Diseño de hardware.....	79
	3.1.1.	Descripción de los componentes empleados.....	80
	3.1.1.1.	Microcontrolador PIC16F877A.....	80
	3.1.1.2.	Bus I ² C.....	82
	3.1.1.3.	Servomotor	83
	3.1.1.4.	Sistema de alimentación.....	85
	3.1.1.5.	Pinza.....	86
	3.2.	Diseño de software.....	86
	3.2.1.	Protocolo lógico de comunicación I ² C: bloque NXT - PIC16F877A.....	87
	3.2.1.1.	Comandos para pinza inteligente NXT.....	91
	3.2.2.	Funciones desempeñadas por PIC16F877A	96

3.2.2.1.	Configuración módulo MSSP	96
3.2.2.2.	Diseño de programación: comunicación I ² C.....	100
3.2.2.3.	Generación de la señal PWM.....	106
3.2.2.3.1.	Configuración módulo TMR0	109
3.2.2.3.2.	Configuración módulo TMR1	114
3.2.2.4.	Memoria EEPROM.....	116
3.2.2.5.	Diseño de programación	118
3.2.2.5.1.	Declaración de variables.....	121
3.2.2.5.2.	Ingreso de datos a memoria EEPROM.....	122
3.2.2.5.3.	Configuración: módulos y variables.....	123
3.2.2.5.4.	Inicialización_pinza	124
3.2.2.5.5.	Interrupción	125
3.3.	Diseño del módulo: bloque Pinza NXT.....	134
3.3.1.	Creación de un bloque simple	135
3.3.2.	Diseño de programación parte lógica.....	136
3.3.2.1.	Pinza NXT.vi.....	136
3.3.2.2.	Pinza NXTSub.vi	140
3.3.3.	Diseño de programación parte visual	148
3.3.3.1.	Config Pinza NXT.vi	148
3.3.3.2.	Draw Pinza NXT.vi	153
3.3.3.3.	Drawers.dat.....	156
3.3.3.4.	Importación.....	158

4.	ANÁLISIS DE FUNCIONAMIENTO.....	161
4.1.	Conectividad.....	161
4.2.	Pruebas lógicas.....	162
4.3.	Pruebas mecánicas.....	164
4.4.	Determinación de características.....	165
4.4.1.	Rango de apertura.....	165
4.4.2.	Linealidad.....	165
4.4.3.	Capacidad de agarre.....	171
4.4.4.	Resumen de características.....	171
4.5.	Mejoras.....	172
	CONCLUSIONES.....	175
	RECOMENDACIONES.....	177
	BIBLIOGRAFÍA.....	179
	APÉNDICES.....	181

ÍNDICE DE ILUSTRACIONES

FIGURAS

1.	Señal PWM para diferentes niveles de potencia.....	5
2.	Comportamiento: voltaje vrs corriente de sensor de 4,5V.....	5
3.	Intervalos de lectura.....	7
4.	Esquema de conexiones internas del puerto de entrada.....	9
5.	Navegación en el menú del bloque NXT.....	12
6.	Entorno de programación software NXT-G.....	14
7.	Pantalla de inicio software NXT-G.....	17
8.	Entorno software NXT-G.....	17
9.	Entorno de funciones del controlador.....	18
10.	Bloques Display y Time en área de trabajo.....	19
11.	Panel de configuración bloque Display.....	20
12.	Panel frontal software LabVIEW.....	21
13.	Diagrama de bloques software LabVIEW.....	22
14.	Funciones del software LabVIEW.....	24
15.	Función NXT Display y Wait.....	25
16.	Información de cada terminal de los objetos NXT Display y Wait.....	26
17.	Programa ejemplo desarrollado en LabVIEW.....	27
18.	Actuador pinza inteligente: diagrama en bloques.....	29
19.	Servomotor HS-422 marca Hitec.....	31
20.	Tren de impulsos para control de un servomotor.....	33
21.	PIC16F877A empaquetado de 40 pines.....	36
22.	Esquema de un oscilador de cristal TTL.....	37
23.	Estructura de un bus I ² C.....	44

24.	Transferencia de un bit en el bus I ² C	45
25.	Condiciones <i>start</i> y <i>stop</i>	46
26.	Protocolo de comunicación I ² C	47
27.	Transferencia de datos completa.....	49
28.	Maestro escribe datos en el esclavo.....	49
29.	Maestro lee datos del esclavo.....	50
30.	Formato de transferencia combinado	51
31.	Trama de transmisión de <i>bytes</i> combinado	52
32.	Trama de transmisión de <i>bytes</i> no combinado	53
33.	Diagrama de bloques I ² C modo esclavo	54
34.	Funciones de NXT Toolkit en LabVIEW.....	59
35.	Funciones de NXT Toolkit: Invoke Node y Property Node.....	60
36.	Configuración: función Property Node clase NXTInput.....	62
37.	Función Invoke Node clase NXTCommLSCheckStatus	64
38.	Función Invoke Node clase NXTCommLSWrite	65
39.	Función Invoke Node clase NXTCommLSRead	66
40.	Ubicación del control Sequence Flow.....	67
41.	Conexiones del ícono y los diferentes esquemas de conexión.....	68
42.	Editor de ícono.....	69
43.	Opciones de New NXT Block Wizard.....	70
44.	Carpeta Pinza NXT	71
45.	Carpeta MyBlock: íconos	72
46.	Panel frontal: programa Pinza NXT.vi.....	73
47.	Carpeta Drawerimages	74
48.	Diagrama de bloques: programa Pinza NXT.vi.....	74
49.	Diagrama de bloques: programa Pinza NXTSub.vi.....	75
50.	Config Pinza NXT.vi.....	76
51.	Actuador pinza inteligente.....	79
52.	Diagrama circuital del actuador pinza inteligente.....	81

53.	Circuito integrado impreso	82
54.	Parte posterior del actuador pinza inteligente	83
55.	Dimensiones del servomotor Hitec.....	84
56.	Regulador MC78M05 y sus terminales	85
57.	Pinza.....	86
58.	Mapa de memoria para un dispositivo digital externo	88
59.	Diagrama de flujo del programa del microcontrolador módulo MSSP modo I ² C esclavo	101
60.	Secuencia de pulsos PWM.....	107
61.	Diagrama de flujo rutina EEPROM_LeeDatos.....	118
62.	Diagrama de flujo general	120
63.	Diagrama de flujo inicialización_pinza.....	124
64.	Diagrama de flujo de la rutina interrupción	126
65.	Diagrama de flujo TMR0_interrupción.....	127
66.	Diagrama de flujo TMR1_interrupción.....	129
67.	Diagrama de flujo de rutina I ² C_interrupción.....	130
68.	Diagrama de flujo de la rutina verificación_var_comando	132
69.	Flujo grama de rutina posicionar_pinza.....	133
70.	Diagrama de flujo rutina escritura_I ² C.....	134
71.	Diagrama de bloque Pinza NXT.vi	137
72.	Diagrama de bloques Pinza NXT.vi.....	140
73.	Diagrama de bloques Pinza NXTSub.vi	140
74.	Diagrama de flujo programa Pinza NXTSub.vi.....	142
75.	Diagrama de flujo: verificación de comandos	144
76.	Diagrama de bloques de la programación comunicación I ² C	145
77.	Diagrama de bloques Pinza NXTSub.vi	147
78.	Panel frontal del programa Config Pinza NXT.vi	148
79.	Diagrama de bloques Config Pinza NXT.vi	149
80.	Funciones utilizadas en Config Pinza NXT.vi.....	150

81.	Panel frontal Config Pinza NXT.vi.....	151
82.	Diagrama de bloques programación Config Pinza NXT.vi	152
83.	Panel frontal Draw Pinza NXT.vi.....	153
84.	Diagrama de bloques Draw Pinza NXT.vi.....	154
85.	Programación en diagrama de bloques Draw Pinza NXT.vi	155
86.	Opciones de la función Skin.....	155
87.	Programa Draw Pinza NXT.vi	156
88.	Importación del nuevo bloque desde NXT-G	158
89.	Bloque Pinza NXT desde software NXT-G	159
90.	Cable UTP de 30 cm de longitud	161
91.	Conectores RJ12 conectados al cable UTP.....	162
92.	Programa: comandos.....	163
93.	Programa: verificación de comando de acción	164
94.	Programa aperturar pinza	166
95.	Longitud de apertura vrs posición de apertura.....	168
96.	Diferencia entre el valores reales y valores teóricos.....	170

TABLAS

I.	Características de hardware del bloque NXT	2
II.	Puerto de salida	4
III.	Puerto de entrada.....	6
IV.	Archivos y extensiones soportados por el bloque NXT	11
V.	Registro TMR0: bits de configuración.....	39
VI.	Registro TMR1: bits de configuración.....	41
VII.	Protocolo lógico de comunicación de LEGO	51
VIII.	Códigos de error: NXTCommLSCheckStatus	64
IX.	Códigos de error NXTCommLSWrite	65
X.	Códigos de error NXTCommLSRead	66

XI.	Características físicas del actuador pinza inteligente	80
XII.	Características del servomotor Hitec	83
XIII.	Relación de ancho de pulso vrs ángulo	85
XIV.	Protocolo lógico de comunicación I ² C, sensor ultrasónico de LEGO.....	90
XV.	Tercer grupo de comandos sensor ultrasónico.....	91
XVI.	Primer grupo de comandos para el actuador pinza inteligente..	92
XVII.	Comando estado	93
XVIII.	Segundo grupo de comandos.....	94
XIX.	Comandos tercer grupo	95
XX.	Comandos para el actuador pinza inteligente.....	95
XXI.	Registro INTCON.....	97
XXII.	Registro PIE1.....	98
XXIII.	Puerto C.....	98
XXIV.	Registro SSPCON	99
XXV.	Estados de los bits SSMPM3:SSPM0	100
XXVI.	Registro SSPSTAT	102
XXVII.	Manejo del servomotor y pinza	107
XXVIII.	Rango de asignación para carga TMR0	111
XXIX.	Registro OPTION_REG.....	113
XXX.	Rango de operación del divisor de frecuencia.....	114
XXXI.	Registro T1CON	115
XXXII.	Carga TMR1	115
XXXIII.	Registro EECON1.....	117
XXXIV.	Valores de entrada y salida métodos NXTComm - LSWrite/LSRead.....	141
XXXV.	Resumen de comandos: actuador pinza inteligente	143
XXXVI.	Valores asignados a cada comando	152
XXXVII.	<i>Pin-in y pin-out</i>	162

XXXVIII.	Comandos enviados desde el bloque NXT hacia el actuador pinza inteligente	163
XXXIX.	Resultado de pruebas de movimiento del actuador pinza inteligente.....	165
XL.	Tabulación de datos experimentales.....	166
XLI.	Posición de apertura – longitud de apertura.....	167
XLII.	Tabulación de datos reales, teóricos y desviación porcentual.....	170
XLIII.	Resultados de pruebas de peso.....	171
XLIV.	Características del actuador pinza inteligente	172

RESUMEN

El presente trabajo de graduación pretende demostrar cómo pueden expandirse las capacidades de entradas y salidas de un sistema inteligente como el LEGO® MINDSTORMS® NXT, empleando una de sus características más importantes: el bus de comunicación I²C.

El trabajo comprende el diseño de hardware y de software del nuevo actuador, el cual es llamado pinza inteligente.

En el primer capítulo se presentan las principales características del dispositivo programable de LEGO® MINDSTORMS® NXT llamado bloque NXT, donde se exterioriza su arquitectura, hardware y software.

En el segundo capítulo se expone la propuesta de diseño del nuevo actuador pinza inteligente, desarrollando los sistemas utilizados: sistema de control y de comunicación; como también, los elementos de programación necesarios a emplear en software LabVIEW.

En el tercer capítulo se lleva el diseño a la implementación, en esta sección se expone el diseño de hardware: microcontrolador PIC16F877A, bus I²C, servomotor, sistema de alimentación; diseño de software: protocolo lógico de comunicación entre bloque NXT y nuevo actuador; y el diseño de programación de un bloque programa en software LabVIEW importado a software NXT-G.

En el cuarto capítulo se realiza el análisis de funcionamiento del nuevo actuador pinza inteligente, se verifica su conectividad, linealidad y características físicas de desempeño.

OBJETIVOS

General

Diseñar un actuador inteligente que interactúe mediante protocolo de comunicación I²C con un sistema robótico comercialmente disponible y amplíe las capacidades de éste.

Específicos

1. Diseñar un actuador, en la forma de una pinza inteligente, que se adapte y responda a las instrucciones de un controlador de la serie LEGO® MINDSTORMS® NXT.
2. Utilizar protocolo I²C como medio de comunicación entre el controlador y el actuador, empleando un microcontrolador para su interfaz.
3. Crear los módulos de software necesarios que permitan incorporar al actuador dentro de programas que puedan ser cargados y ejecutados por el controlador del sistema LEGO® MINDSTORMS® NXT.
4. Convertir el diseño en un actuador real que efectivamente cumpla con su propósito.

INTRODUCCIÓN

El propósito de este trabajo de graduación es diseñar e implementar un actuador inteligente para un sistema robótico comercialmente disponible como el LEGO® MINDSTORMS® NXT, utilizando los recursos y características propias de su sistema, principalmente la comunicación I²C.

El nuevo actuador inteligente tiene la forma de una pinza y su sistema se divide en tres elementos importantes, los cuales son: hardware, software y la interfaz de comunicación I²C.

Para el diseño de hardware se emplea un microcontrolador, un servomotor y una pinza mecánica; el dispositivo central del sistema hardware es el microcontrolador y, para ello se ha empleado el PIC16F877A debido a que facilita la interfaz de comunicación hacia el bloque NXT y el manejo del servomotor por medio de sus diferentes módulos, además proporciona suficiente espacio de memoria para el almacenamiento de programas y elementos de configuración.

Del PIC16F877A se utilizan los registros que manejan la memoria EEPROM para el almacenamiento y envío de información al bloque NXT, el módulo MSSP que maneja el puerto serial de comunicación I²C para la interfaz de comunicación y los módulos TMR0 y TMR1, que son utilizados para generar pulsos PWM y ser enviados hacia el servomotor.

Para el diseño e implementación de software se emplea el software LabVIEW donde se utiliza la herramienta Wizard, para la generación de los

diferentes VI's del nuevo módulo, los cuales son programados e importados al software NXT-G.

Para lograr la comunicación lógica entre el bloque NXT y el PIC16F877A se diseña un protocolo lógico de comunicación que sigue las pautas del fabricante LEGO, este protocolo se materializa a través del proceso físico mediante la comunicación I²C del que dispone tanto el bloque NXT como el PIC16F877A.

1. PLATAFORMA LEGO® MINDSTORMS® NXT

1.1. Introducción

El sistema LEGO® MINDSTORMS® NXT es la última versión de juego robótico que la empresa LEGO ha lanzado al mercado. Su elemento central es el bloque programable NXT, parte que contiene la lógica y electrónica; con capacidad de almacenamiento, carga y ejecución de programas.

El bloque NXT es La base principal del proyecto, ya que sus características de hardware y software son de fácil acceso para el investigador. Una de las características principales del bloque son sus puertos de entrada los cuales son de interfaz digital que soportan la comunicación I²C.

Una peculiaridad del bloque NXT es su programación que puede realizarse en diversos lenguajes de bajo o alto nivel, gráficos o textuales, sin embargo, solo abarcaremos dos; el software oficial de LEGO (MINDSTORMS NXT-G) donde se programara el nuevo actuador pinza inteligente y el software LabVIEW donde se creará el módulo del actuador pinza inteligente. Cabe mencionar que ambos software fueron desarrollados por National Instruments y son fundamentales en el desarrollo del proyecto.

En las próximas secciones se detallarán las principales características del bloque NXT, el software de programación y la importancia de estos para la implementación del nuevo actuador.

1.2. Características principales de hardware

El bloque NXT contiene un microprocesador Atmel ARM7 de 32 bits a 48 MHz, un coprocesador Atmel AVR de 8 bits a 8 MHz, cuenta con una unidad de comunicación inalámbrica Bluetooth (especificación v2.0 EDR) de 8 bits a 26 MHz; comunicación por cable usando tecnología USB (estándar 2.0) que soportan tasas de transferencia de datos de hasta 2,1 y 12 Mbits/s respectivamente.

El bloque NXT cuenta con 4 puertos de entrada y 3 de salida con conexión RJ12 (conectores telefónicos de 6 hilos), que permiten conectar sensores tanto digitales como analógicos para sus puertos de entrada y diferentes tipos de actuadores (normalmente servomotores), para sus puertos de salida. Además el bloque NXT cuenta con una pantalla LCD gráfica de 100x64 pixeles que se puede utilizar en modo gráfico para dibujar figuras, un altavoz con un canal de sonido con 8 bits de resolución capaz de generará tonos en el rango de 2 a 16 KHz y 4 botones que permiten interactuar con el bloque NXT.

En la tabla I se presenta el resumen de características de hardware.

Tabla I. **Características de hardware del bloque NXT**

Procesador principal:	Atmel® 32-bit ARM® processor, AT91SAM7S256
	256 KB <i>flash</i>
	64 KB RAM, 48 MHz
Co-procesador:	Atmel® 8-bit AVR processor, ATmega48
	4 KB <i>flash</i>
	512 Byte RAM
	8 MHz
	CSR BlueCore™ 4 v2.0 + Sistema EDR

Continuación de la tabla I.

Unidad de comunicación inalámbrica Bluetooth	Soporta el perfil de puerto serial (SPP)
	47 KByte RAM internos
	8 MBit <i>flash</i> externos
	26 MHz
	Velocidad del puerto (12 Mbit/s)
Comunicación vía USB 2,0	Con interfaz de 6 hilos y soporte de conexiones AD y DA
4 puertos de entrada	1 puerto de alta velocidad, IEC 61158 tipo 4/EN 50170 compatible
3 puertos de salida	Interfaz de 6 hilos y soporte para lectura desde <i>encoders</i>
Pantalla	100 x 64 pixel LCD pantalla gráfica blanco & negro
	Área de visión: 26 X 40,6 mm
Altavoz	Con canal de salida de sonido con 8-bit de resolución
	Soporta tasas de muestreo de 2-16 KHZ
4 botones de goma	Para la interfaz con el usuario
Alimentación	6 baterías AA
	Se recomienda utilizar pilas alcalina
	También se encuentra disponible una batería de ion de litio de 1 400 mAh
Conectores	De 6 hilos estándar industrial, RJ12 con ajuste a la derecha

Fuente: LEGO Group. LEGO® MINDSTORMS® NXT Hardware Developer kit. p. 3.

1.2.1. Puertos de salida

El bloque NXT cuenta con 3 puertos de salida nombrados A, B y C; utilizan una interfaz digital de 6 hilos, característica implementada para que los dispositivos puedan enviar y recibir información; y así evitar el uso de un puerto adicional. En la tabla II se detalla el color y el nombre de cada uno de los pines del puerto.

Tabla II. **Puerto de salida**

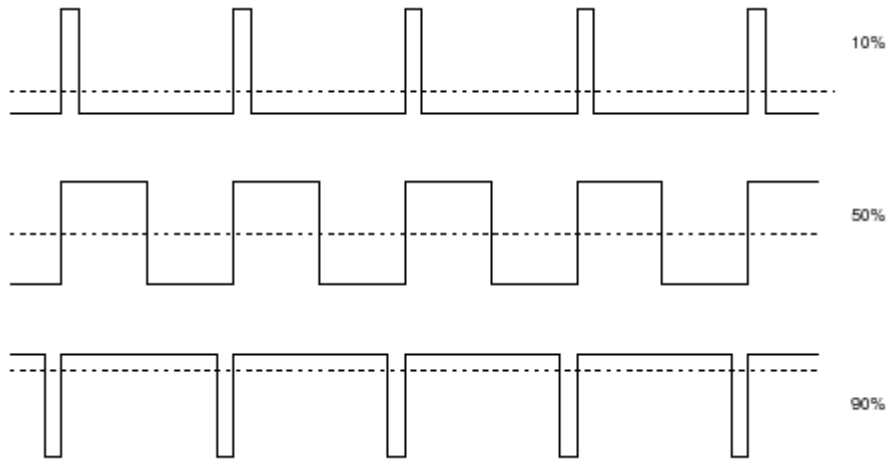
Pin	Color	Nombre
1	Blanco	MA0
2	Negro	MA1
3	Rojo	GND
4	Verde	4.3V POTENCIA
5	Amarillo	TACHO0
6	Azul	TACHO1

Fuente: GASPERI, Michael; HURBAIN, Philippe. Extrem NXT Extending the LEGO® MINDSTORMS® NXT to the Next Level. p. 30.

MA0 y MA1 proporcionan una señal de salida PWM para controlar los actuadores (motores), su voltaje máximo de salida es igual al voltaje de las baterías (9 voltios batería estándar), esta señal es manejada por un controlador de motores llamado puente H (IC LB1836 y LB1930M), el cual suministra una corriente continua de 700mA, para cada puerto con un pico de salida que puede llegar hasta un amperio aproximadamente. El controlador cuenta con una terminal de protección integrada, si demasiada potencia es constantemente drenada desde el bloque NXT, el controlador automáticamente modificará la salida de la corriente. En la figura 1 se muestra diferentes señales PWM.

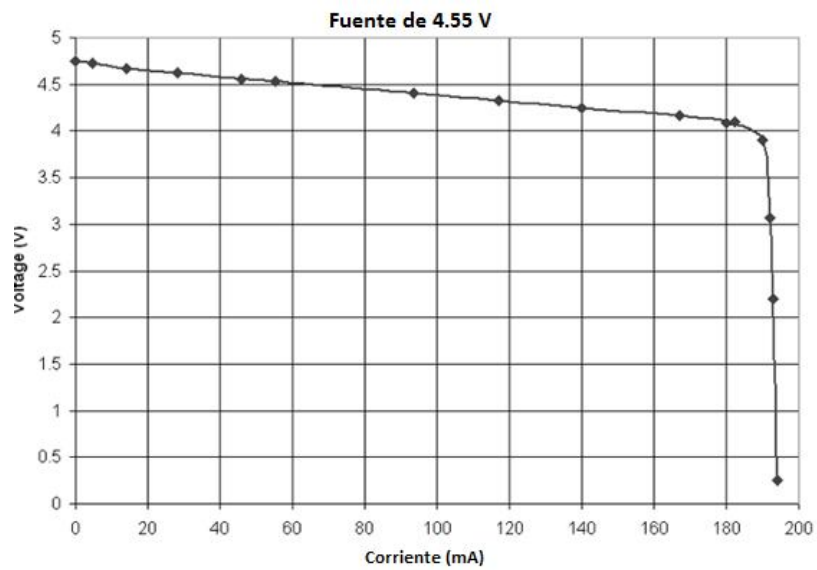
El pin de potencia está conectado a una fuente de 4,3 voltios, internamente todos los pines de potencia de los puertos de entrada y salida están conectados a esta fuente, la cual provee una corriente máxima de aproximadamente 180 mA, esto significa que cada puerto puede drenar aproximadamente 20 mA. Si más potencia es drenada, el total de corriente de salida disminuirá automáticamente sin advertencia. Si en caso la señal de potencia es cortocircuitada a tierra, el bloque NXT automáticamente se reiniciará. En la figura 2 se muestra este comportamiento.

Figura 1. **Señal PWM para diferentes niveles de potencia**



Fuente: <http://www.best-microcontroller-projects.com/pwm-pic.html>. Consulta: enero de 2013.

Figura 2. **Comportamiento: voltaje vrs corriente de sensor de 4,5V**



Fuente: GASPERI, Michael; HURBAIN, Philippe. Externe NXT Extending the LEGO® MINDSTORMS® NXT to the Next Level. p. 45.

Los pines tach0 y tach1 son señales de entrada, las cuales internamente tienen un disparador de Schmitt conectado a los pines de entrada del procesador ARM7. Con el firmware estándar estas dos señales son usadas para contar los números de pulsos desde los motores NXT y detectar si el motor está girando a la derecha o a la izquierda.

1.2.2. Puertos de entrada

El bloque NXT cuenta con 4 puertos de entrada y al igual que los puertos de salida, estos utilizan una interfaz digital de 6 hilos. A diferencia de los puertos de salida, los puertos de entrada están nombrados por numerales: puerto 1, 2, 3 y 4. Los cuatro puertos permiten al bloque NXT medir diferentes parámetros físico.

La conexión de los puertos de entrada puede ser digital o analógica, por lo que es posible extender la capacidad de entradas de sensores y actuadores. En la tabla III se detalla el color y el nombre de cada uno de los pines del puerto de salida.

Tabla III. Puerto de entrada

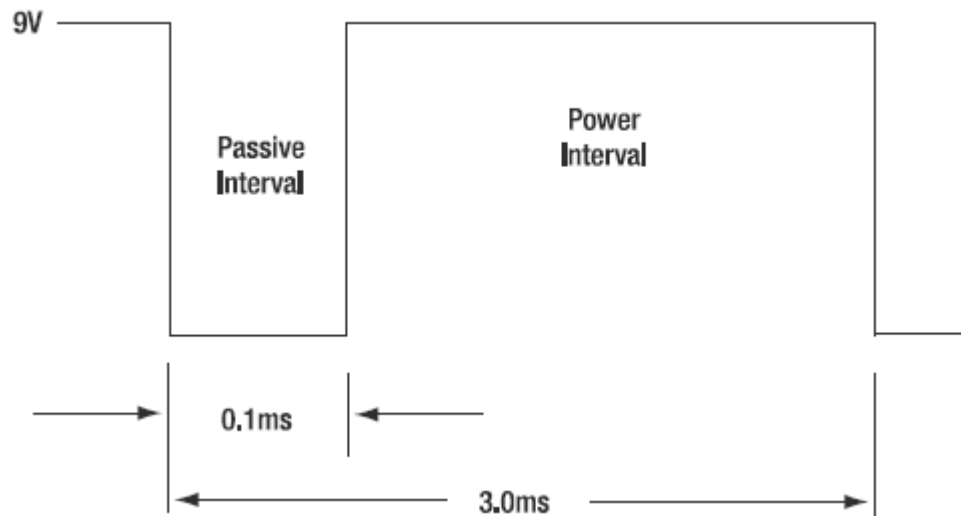
Pin	Color	Nombre
1	Blanco	ANA
2	Negro	GND
3	Rojo	GND
4	Verde	4.3 V Potencia
5	Amarillo	DIGIAI0
6	Azul	DIGIAI1

Fuente: GASPERI, Michael; HURBAIN, Philippe. Externe NXT Extending the LEGO® MINDSTORMS® NXT to the Next Level. p. 27.

El pin ANA puede usarse como entrada analógica o como fuente de 9V. Al utilizar el pin como entrada analógica, la señal es conectada a un convertidor A/D de 10-bits con un rango de entrada de 0 a 5V, el cual es trasladado a un valor digital de 0 a 1 023. La señal de entrada A/D es muestreada cada 3 milisegundos (ms). El pin está conectado internamente a 5V a través de una resistencia *pull-up* de 10KΩ. La frecuencia de muestreo para todos los sensores analógicos es de 333 Hz.

El pin ANA, como fuente de 9V, es utilizado por los sensores del modelo anterior (LEGO® MINDSTORMS® RCX). Los sensores necesitan de un capacitor que cargue en un intervalo 3ms. En la figura 3 se muestran los dos intervalos de tiempo, que se manejan el de lectura y el de carga.

Figura 3. **Intervalos de lectura**



Fuente: GASPERI, Michael; HURBAIN, Philippe. Exteme NXT Extending the LEGO® MINDSTORMS® NXT to the Next Level. p. 131.

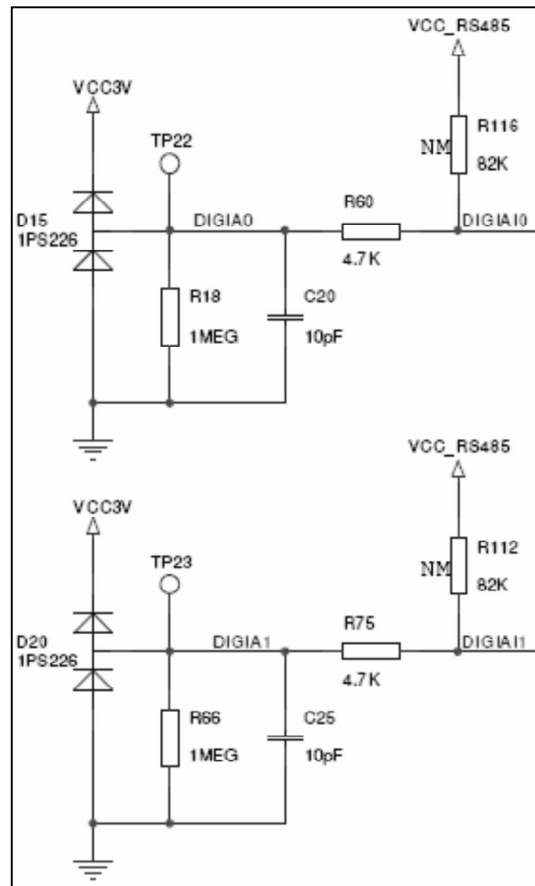
Los pines digitales I/O digital0 y digital1 son de 3,3V, usados para comunicación digital y están conectados directamente al microprocesador del bloque NXT, estos pines son, principalmente, utilizados para la comunicación I²C. Cuando estos pines son utilizados como salida el límite de su voltaje es de 3,3V, si bien son utilizados como entrada el bloque NXT cuenta con circuitos de protección para prevenir cualquier sobre voltaje. Los circuitos incluyen un resistor de 4,7K Ω conectado en serie con el puerto, de esa manera si el voltaje de los sensores es muy alto, la corriente será baja evitando dañar los circuitos electrónicos.

1.2.3. Puertos con interfaz I²C

El bloque NXT cuenta con 4 canales de comunicación I²C (Interconexión de Circuitos Integrados), uno por cada puerto de entrada. La comunicación digital I²C permite la expansión de puertos de entradas y de salidas, la capacidad máxima por cada puerto es de 127 puertos lógicos.

La comunicación I²C permite trabajar los dispositivos de dos maneras: como maestros o como esclavos. El bloque NXT fue diseñado para trabajar como dispositivo maestro, esto significa que él controla el flujo de datos en cada uno de sus canales. Para la comunicación es necesario utilizar tan solo dos líneas activas y una de tierra o referencia, los datos son transmitidos de forma serial, un bit por vez. Una línea envía y recibe la información (los datos SDA) y la otra provee la sincronización (pulsos de reloj SCL). En la figura 4 se muestra el esquema de las conexiones internas de un puerto de entrada del bloque NXT.

Figura 4. Esquema de conexiones internas del puerto de entrada



Fuente: LEGO Group. LEGO® MINDSTORMS® NXT Hardware Developer kit. p. 9.

Parámetros importantes extraídos del documento LEGO Group: LEGO® MINDSTORMS® NXT Hardware Developer kit:

- El puerto de entrada tiene una resistencia de 4,7 K Ω en serial con la línea de señal.
- El puerto de entrada no cuenta con una resistencia *pull-up* internamente. Para la comunicación I²C es necesario que el dispositivo externo la

contenga; se recomienda una resistencia de 82 kΩ como *pull-up* en la línea de datos (SDA) y la línea de reloj (SCL), este valor de resistencia es para trabajar propiamente con el bloque NXT. Si se conectan múltiples dispositivos en el bus I²C, la resistencia de *pull-up* debe seguir siendo de 82kΩ, luego solo uno de los dispositivos debe tener esta resistencia.

- DIGIx0 (pin 5) es la señal de reloj (SCL) y DIGIx1 (pin 6) es la señal de datos (SDA) para la comunicación I²C.
- Las señales digitales entrada/salida en el bloque NXT no pueden abrirse directamente para drenar corriente. El bloque NXT manejará las señales digitales de entrada/salida en nivel alto o bajo dependiendo del evento generado.
- La comunicación I²C es serial y maneja una tasa de velocidad de 9 600 bits/s.
- Cada canal tiene un búfer de entrada y de salida de 16 *byte*. Por lo tanto un máximo de 16 *bytes* pueden ser enviados y recibidos durante cada ciclo de comunicación de datos.

1.3. Arquitectura del bloque NXT

Se describe a continuación la característica de algunos elementos:

Conectores eléctricos: son similares a un conector JR12 de teléfono, pero el cerrojo no está en el centro; se encuentra en la parte izquierda previniendo utilizar el cable de la señal telefónica, si este fuera accidentalmente conectado.

Memoria *flash*: la memoria *flash* del bloque NXT tiene la capacidad de conservar el firmware y todos sus archivos, incluso cuando se retiran las baterías durante un periodo prolongado. La memoria *flash* puede almacenar hasta 256 KB de datos.

1.3.1. Sistema de archivos

El sistema de archivos del bloque NXT puede almacenar hasta 64 archivos. El firmware permite a los usuarios crear y borrar los archivos, cambiar el nombre y modificarlos. La extensión de los archivos consta de 3 caracteres separados del nombre por un punto. El nombre del archivo en si puede ser de hasta 15 caracteres. En la tabla IV se muestra los archivos soportados y sus respectivas extensiones.

Tabla IV. **Archivos y extensiones soportados por el bloque NXT**

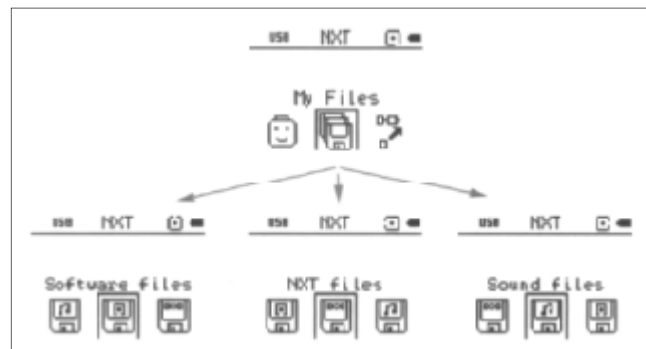
Tipos de Archivos	Extensiones
Archivos de datos	.rdt
Archivos ejecutables y programas de pruebas	.rxex, .rtm
Archivos de ícono	.ric
Archivos ocultos de menú	.rms
Archivos de programas	.rpg
Archivos de sonido	.rso
Archivos ocultos del sistema	.sys
Archivos ocultos temporales	.tmp

Fuente: ASTOLFO, Dave; FERRARI, Mario; FERRARI, Giulio. Building Robots with LEGO® MINDSTORMS® NXT. p.120.

En el bloque NXT la interfaz de usuario muestra estos archivos en la forma de una estructura de menú que comienza con mis archivos, y luego se ramifican

en archivos de software, archivos NXT, archivos de sonido y así sucesivamente. En la figura 5 se muestran los archivos almacenados.

Figura 5. **Navegación en el menú del bloque NXT**



Fuente: ASTOLFO, Dave; FERRARI, Mario; FERRARI, Giulio. Building Robots with LEGO® MINDSTORMS® NXT. p. 102.

1.3.2. Interfaces digitales

El bloque NXT cuenta con las interfaces I²C, USB y Bluetooth. La interfaz USB se utiliza para comunicarse con una PC con Windows o Macintosh.

La interfaz de comunicación I²C es utilizada por el sensor ultrasónico, incluido en el kit original de LEGO, pero también puede ser utilizada por otros dispositivos desarrollados por terceros los cuales pueden ser conectados físicamente al bloque NXT. Cabe mencionar que la interfaz de comunicación I²C tiene la característica de permitir que muchos más sensores y actuadores puedan ser conectados al bloque NXT, compensando así la limitación en puertos de entrada.

La comunicación inalámbrica Bluetooth del bloque NXT puede funcionar tanto en modo maestro o en modo esclavo; pero en la conexión a un PC siempre trabajará en modo esclavo. El bloque NXT se puede conectar de forma inalámbrica a otros tres dispositivos al mismo tiempo, pero sólo puede comunicarse con un dispositivo a la vez, un bloque deberá ser configurado en modo maestro y el resto en modo esclavo. Esta funcionalidad ha sido implementada con el perfil de puerto serial (SPP), que se considera un puerto serial inalámbrico.

1.4. Bloque programable NXT

La programación para el bloque NXT es comúnmente realizada en un computador mediante un software. El lenguaje de programación oficial utilizado es LEGO® MINDSTORMS® NXT-G, comúnmente conocido tan solo como NXT-G.

Este lenguaje describe en forma visual elementos que se arrastran y sueltan, buscando formar una sucesión gráfica de instrucciones que describirán el funcionamiento de un sistema cualquiera.

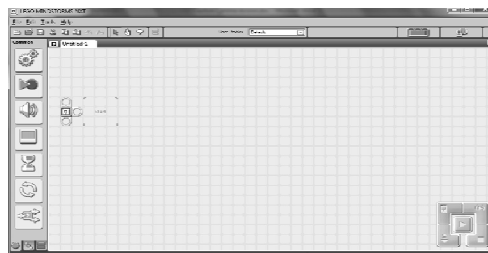
Existen otros lenguajes de programación, por ejemplo: Robot C, NXC, pbLua, Lejos NXJ, LabVIEW, etc., para programar el bloque NXT, en este trabajo se utilizarán el lenguaje de programación oficial NXT-G y LabVIEW.

1.4.1. Programación en software NXT-G

El software NXT-G está disponible para Windows XP y Mac y se ha desarrollado para LEGO por National Instruments, una compañía especializada en entornos de prueba automatizados en software de instrumentación virtual.

NXT-G es un lenguaje de programación que utiliza flujos de datos, los programas se modelan como gráficos dirigidos de los datos que fluyen entre las operaciones. Este modelo utiliza notación gráfica, que comprende bloques o módulos de programas, donde se pueden visualizar las diferentes operaciones. También cuenta con vigas de secuencia, que controlan el flujo del programa. Las vigas indican la secuencia en que los bloques de programas están conectados entre sí.

Figura 6. **Entorno de programación software NXT-G**



Fuente: elaboración propia.

Los programas de NXT-G se crean mediante arrastrar y soltar los diferentes tipos de bloques de programas en el área de trabajo y conectar sus terminales mediante cables de datos. Antes de descargar un programa al bloque NXT, el programa automáticamente se compila en código ensamblador que puede ser ejecutado por el firmware. El sistema LEGO® MINDSTORMS® NXT® permite crear nuevos bloques de programa, además de los que inicialmente vienen en el software, los cuales pueden ser fácilmente importados.

1.4.1.1. Entorno de programación NXT-G

De forma predeterminada el software NXT-G ofrece una gran variedad de bloques de programa los cuales se detallan a continuación.

- Bloques de control de salida: controlan los dispositivos de salida tales como motores, sensores de sonido y pantalla LCD. Los bloques de control de salida son: de motor, de movimiento, sensor de rotación, de sonido y de visualización.
- Bloques de control de entrada: controlan los dispositivos de entrada, entre ellos están: de sensor de contacto, de sensor de sonido, de sensor de luz y de sensor ultrasonido.
- Bloques de comunicación: envían y reciben mensajes utilizando la comunicación Bluetooth. Entre ellos están: bloque envía mensaje, y el que recibe mensaje.
- Bloques de programas de flujo: el control de flujo general de programas se logra mediante el uso de los siguientes bloques: de espera, de bucle y de interruptor.
- Otros bloques: hay algunos que no se ajustan a las categorías anteriores, los cuales son: bloque temporizador, de grabación/reproducción, personalizados y de variables.

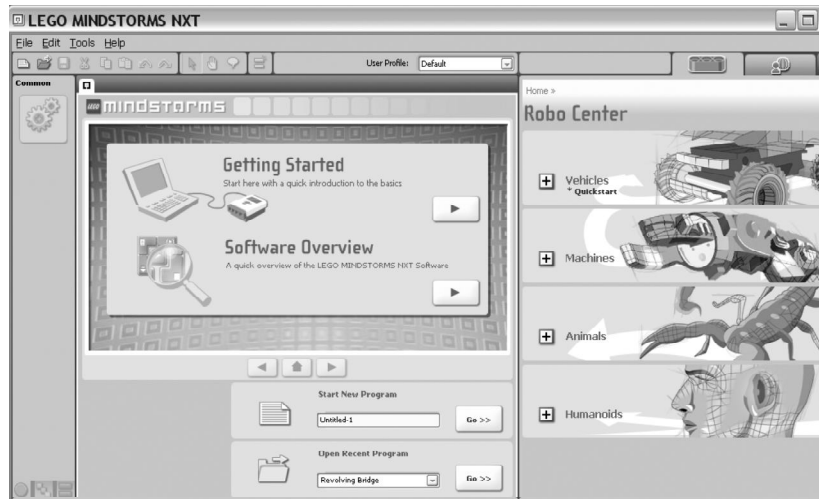
NXT-G ofrece el concepto de cables de datos, los elementos del modelo son visualizados como líneas. Cada bloque de programa cuenta con un centro de información de datos y es compartida mediante sus puertos lógicos de entrada/salida. Si un bloque necesita información de otro, es necesario cablear los puertos compatibles para el traslado de la información.

Entorno de desarrollo integrado (IDE): una vez que se instale e inicie el software de NXT-G, se puede ver el IDE, como se muestra en la figura 7.

El IDE se compone de seis áreas diferentes que le permiten realizar todas las tareas que se requieren para la programación de programas las cuales son:

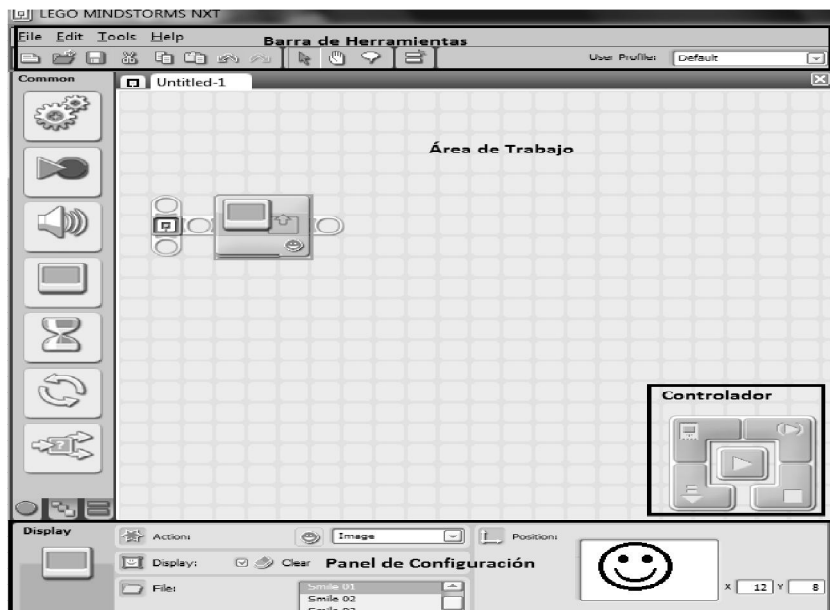
- Barra de herramientas: el área que le permite activar los comandos más utilizados, por ejemplo: cargar o guardar un programa.
- Programación de la paleta: contiene los diferentes tipos de bloques.
- Área de trabajo: es el área donde se colocan los bloques para desarrollar los programas.
- Panel de configuración: es el área donde se configura cada bloque.
- Controlador: herramienta para compilar, descargar y ejecutar programas.
- Robo center: ofrece una guía para la construcción paso a paso de diferentes prototipos.

Figura 7. **Pantalla de inicio software NXT-G**



Fuente: elaboración propia.

Figura 8. **Entorno software NXT-G**



Fuente: elaboración propia.

Para programar en NXT-G, solo se debe de arrastrar y soltar los bloques que se utilizarán en el área de trabajo, luego se debe configurar sus parámetros para que estos puedan proporcionar y/o recibir la información necesaria, para su funcionamiento dentro del programa, si el bloque así lo requiere y, por último cablear de ser necesario, los puertos de entradas y salidas para que estos puedan procesar la información.

Al tener listo el programa y tener el bloque NXT encendido y conectado a la PC, solo es necesario hacer un clic derecho con el *mouse* en el botón Download and run, del controlador que se encuentra en la parte inferior derecha del área de trabajo.

Figura 9. **Entorno de funciones del controlador**

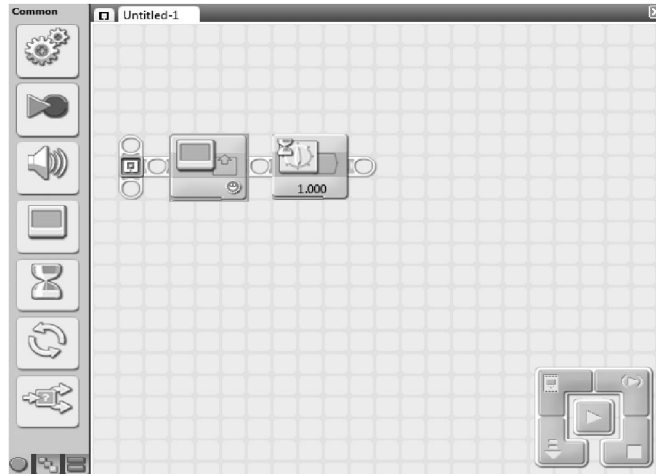


Fuente: elaboración propia.

Se presenta a continuación un programa de ejemplo desarrollado en el software NXT-G, para su fácil comprensión. El programa mostrará una imagen en la pantalla LCD del bloque NXT durante 30 segundos.

Primero: se lleva el bloque Display y el bloque Time al área de trabajo, y se selecciona uno de ellos mediante un clic derecho con el ratón.

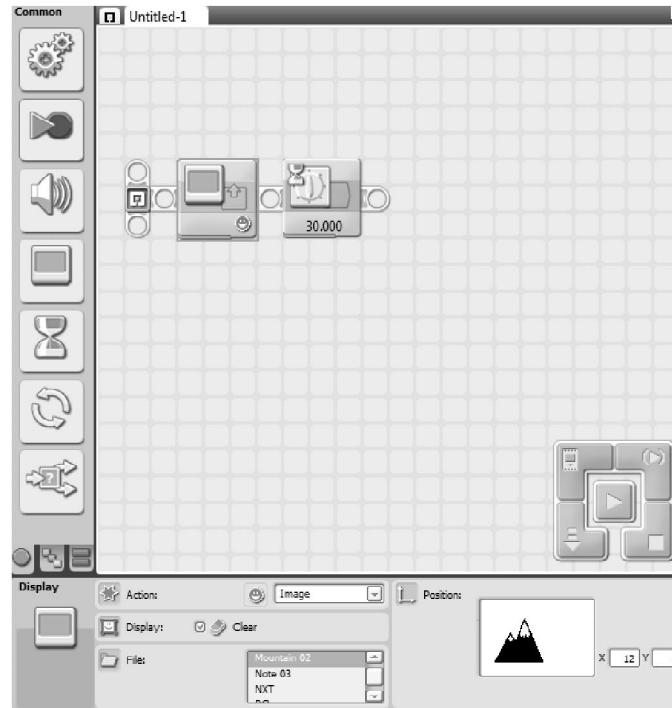
Figura 10. **Bloques Display y Time en área de trabajo**



Fuente: elaboración propia.

Segundo paso: se configura el bloque Display a modo tal que se muestre una imagen de montañas, luego se configura el bloque Time para que la imagen se muestre por 30 segundos. Al llegar a este punto se tiene el programa ejemplo finalizado. En la figura 11 se muestra la configuración del bloque Display.

Figura 11. **Panel de configuración bloque Display**



Fuente: elaboración propia.

Para cargar el programa en el bloque NXT, es necesario que esté conectado con el computador, ya sea vía USB o Bluetooth; luego solo es de dar un clic derecho en el controlador en el botón Download and run, para que el software compile, cargue y ejecute el programa en el bloque NXT.

1.4.2. Lenguaje de programación LabVIEW

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) es un lenguaje y a la vez un entorno de programación gráfico, su revolucionario ambiente de desarrollo gráfico con funciones integradas permite realizar

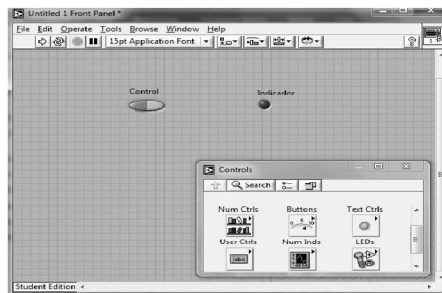
adquisición de datos, control de instrumentos, análisis de mediciones y presentaciones de datos.

Los programas desarrollados en LabVIEW se denominan Instrumentos Virtuales (VI's), porque su apariencia y funcionamiento imitan los de un instrumento real. Los VIs tiene una parte interactiva con el usuario y otra parte de código fuente

Todos los VI's tienen un panel frontal, un diagrama de bloques y paletas de funciones que contienen las opciones que se emplean para crear y modificar los VI's, estos últimos aceptan parámetros procedentes de otros VI's.

Panel Frontal: es la interfaz gráfica del VI con el usuario. Esta interfaz recoge las entradas procedentes del usuario y representa las salidas proporcionadas por el programa. Un panel frontal está formado por una serie de botones, pulsadores, gráficos, etc. Cada uno de ellos puede ser definido como un control o un indicador. Los primeros sirven para introducir parámetros al VI, mientras que los indicadores se emplean para mostrar los resultados producidos, ya sean datos adquiridos o resultados de alguna operación.

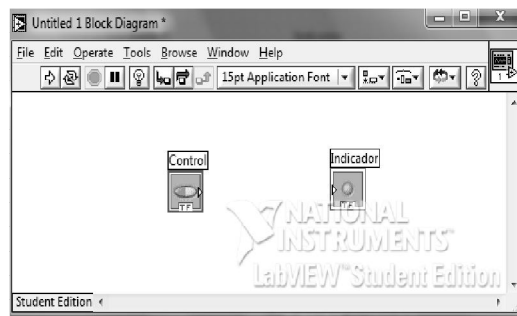
Figura 12. **Panel frontal software LabVIEW**



Fuente: elaboración propia.

Diagrama de bloques: constituye el código fuente del VI, es donde se realiza la implementación del programa del VI para controlar o realizar cualquier proceso de las entradas y salidas que se crearon en el panel frontal. El diagrama de bloques incluye funciones y estructuras integradas en las librerías que incorpora LabVIEW. Los controles e indicadores que se colocaron previamente en el panel frontal, se materializan en un diagrama de bloques con sus terminales de entradas y salidas. En la figura 13 se muestra el diagrama de bloques con el indicador y controlador previamente colocados en el panel frontal.

Figura 13. **Diagrama de bloques software LabVIEW**



Fuente: elaboración propia.

Paletas: las de LabVIEW proporcionan las herramientas que se requieren para crear y modificar tanto el panel frontal como el diagrama de bloques. Existen las siguientes paletas:

- Herramientas (Tools Palette)
- Controles (Controls Palette)
- Funciones (Functions Palette)

En esta sección se asume que el lector tiene los conocimientos básicos para la programación en LabVIEW.

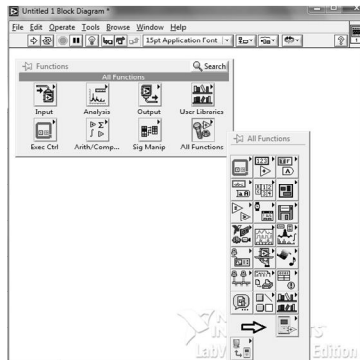
1.4.3. Programación en LabVIEW para NXT-G

El software NXT-G fue desarrollado en la versión 7.1 de LabVIEW, y por ello se encuentra disponible la herramienta LabVIEW Toolkit para LEGO® MINDSTORMS® NXT, donde se proporciona paletas de funciones para la programación del bloque NXT, este módulo se puede bajar en la siguiente pagina <http://zone.ni.com/devzone/cda/tut/p/id/4435> (última consulta enero de 2013).

La herramienta de LabVIEW no solo permite desarrollar programas para este bloque NXT, sino que, también permite desarrollar nuevos bloques que pueden ser exportados al NXT-G.

Para programar el bloque NXT en labVIEW es necesario que se baje y cargue el archivo Toolkit para LEGO® MINDSTORMS® NXT, luego al tenerlo instalado en LabVIEW aparecerá en la paleta de controles de todas las funciones en la parte inferior derecha, la cual se muestra en la figura 14.

Figura 14. **Funciones del software LabVIEW**



Fuente: elaboración propia.

Para programar en LabVIEW se puede comenzar a partir del panel frontal o diagrama de bloques. En primer lugar, se definirán y seleccionarán los controles e indicadores que se emplearán para introducir los datos por parte del usuario.

Una vez colocados todos los objetos necesarios, se debe pasar a la ventana *block diagram* (diagrama de bloques), donde se realiza la programación. Al estar en esta ventana, se encuentran se visualizan las terminales correspondientes a los objetos previamente colocados el panel frontal.

Si en un caso se requiere información sobre algún objeto (variable, función, etc.), que se quiera utilizar en la programación, simplemente se habilita en el menú Help la opción Show Help, con lo que al colocar el cursor del ratón sobre un elemento aparece una ventana con información relativa a este objeto (parámetros de entrada y salida).

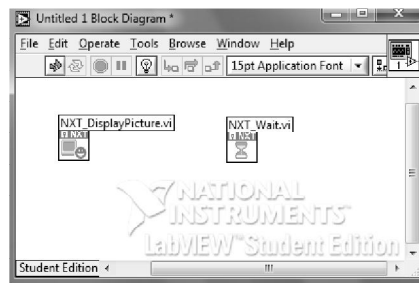
Para cargar el programa en el bloque NXT, se debe de ejecutar el módulo NXT Terminal, el cual se encuentra en la barra de herramientas en la opción Tools > NXT module > NXT Terminal. La opción NXT Terminal proporciona las opciones siguientes:

- Compilar, cargar y ejecutar programa
- Compilar y cargar
- Depurar
- Abortar

Para ejemplificar lo antes expuesto se desarrollará el mismo programa realizado en la sección anterior, con la diferencia que ahora se desarrollará en LabVIEW.

Primero: ubicarse en el diagrama de bloques, seleccionar en la paleta de controles NXT Toolkit las funciones de NXT library > Display > Display picture y colocar el control en el área de trabajo. Luego realizar el mismo procedimiento, pero seleccionar la función Wait. En la figura 15 se muestra los dos objetos colocados en el área de trabajo de LabVIEW.

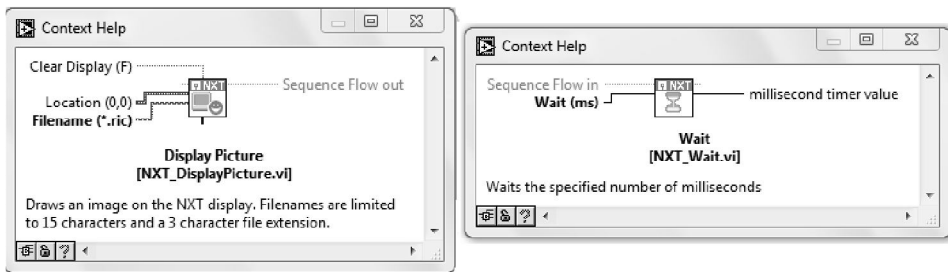
Figura 15. **Función NXT Display y Wait**



Fuente: elaboración propia.

Se hace uso de la ventana de ayuda, donde se observan las terminales de entra y salida de cada objeto colocados en el área de trabajo.

Figura 16. **Información de cada terminal de los objetos NXT Display y Wait**

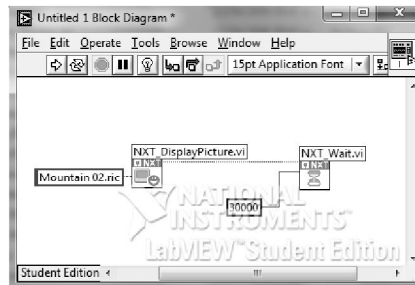


Fuente: elaboración propia.

Segundo: se procede a conectar cada objeto:

- Para el bloque Display se crea una constante en la terminal Filename (*.ric) colocando el nombre del archivo Mountain 02 con extensión .ric, este archivo se encuentra en la ubicación siguiente: LEGO software > LEGO MINDSTORMS NXT > engine > pictures donde se encuentran todos los archivos con extensión .ric y son reconocidos en LabVIEW.
- Para el bloque Wait se crea una constante con valor de 30 000 ms que equivale a 30 segundos.

Figura 17. Programa ejemplo desarrollado en LabVIEW



Fuente: elaboración propia.

Luego de tener el programa desarrollado se procede a guardarlo con el nombre: programa ejemplo.

Tercero: se ejecuta el módulo NXT Terminal y teniendo conectado el bloque NXT al computador se procede a dar clic en la opción: compilar, cargar y ejecutar; y con ello el programa ejemplo se ha cargado y ejecutado en el bloque NXT.

1.5. Sumario

En este capítulo se desarrollaron las principales características de hardware y software del bloque NXT, se indicó que este es un bloque programable con la capacidad de almacenar, cargar y ejecutar programas.

Cabe mencionar que el bloque NXT es la principal herramienta para el desarrollo del nuevo actuador pinza inteligente, los puntos principales de sus características tratadas en el capítulo son:

- Los puertos de entrada del bloque NXT son de interfaz digital de 6 hilos y soportan el bus I²C, lo que permite comunicar microcontroladores y sus periféricos al sistema, esta característica incentiva al investigador a desarrollar nuevos actuadores y sensores que pueden ser conectados a él. El bus I²C permite que el bloque NXT no esté limitado a cuatro puertos de entrada/salida.
- Los software NXT-G y LabVIEW son lenguajes de alto nivel, ambos soportados por el bloque NXT, software independientes uno del otro; sin embargo, los módulos creados en el programa LabVIEW deben ser exportados al programa NXT-G. Ambos lenguajes de programación se utilizarán para el desarrollo del nuevo módulo pinza.

2. ACTUADOR PINZA INTELIGENTE

2.1. Propuesta del actuador pinza inteligente

La propuesta es diseñar un actuador inteligente que se adapte y responda a las instrucciones del controlador bloque NXT de la serie LEGO® MINDSTORMS® NXT.

En la figura 18 se presenta el diagrama general propuesto para el diseño del actuador pinza inteligente:

Figura 18. **Actuador pinza inteligente: diagrama en bloques**



Fuente: elaboración propia.

Bloque NXT: manejará el actuador pinza por medio de uno de sus puertos de entrada y a través de la integración de un nuevo módulo en el software NXT-G.

Interfaz de comunicación: el bloque NXT enviará las instrucciones a seguir en uno de sus puertos de entrada donde se entablará la comunicación a través del bus I²C con el microcontrolador PIC, una vez entablada esta comunicación, el PIC interpretará y dará respuesta a lo solicitado.

Microcontrolador PIC: su función es entablar la comunicación I²C con el bloque NXT, controlar el servomotor mediante pulsos PWM y monitorear la posición actual de la pinza; si el PIC recibe la instrucción, por ejemplo: de abrir la pinza; este deberá de enviar los pulsos correspondientes al servomotor para que posicione la pinza en el lugar indicado y guardar su posición actual.

Servomotor: este dispositivo es controlado por los pulsos generados por el PIC, más adelante se describirá su manejo y características.

Pinza: elemento final del sistema, el cual tiene como objetivo el agarre de un objeto cualquiera.

2.2. Estructura general del sistema

El sistema se compone de 4 elementos importantes:

- Mecánico
- Eléctrico
- Control
- Comunicación

2.2.1. Sistema mecánico y eléctrico

El servomotor es básicamente, un actuador mecánico el cual está basado en un motor y un conjunto de engranajes que permiten multiplicar el torque del sistema final, en este el sistema final es la pinza; también posee elementos internos de control para monitorear de manera constante su posición; es por ello que este dispositivo se encuentra en el sistema mecánico, como también el eléctrico.

Por lo anterior, se abarcará un poco más sobre el funcionamiento y control de este dispositivo en los siguientes incisos.

2.2.1.1. Servomotor

Un servomotor está equipado con un motor de corriente continua y un mecanismo servo para un control preciso de la posición angular. Se muestra en la figura 19 el servo motor a utilizar.

Los servomotores, por lo general, tienen un límite de rotación de 90 a 180 grados, pero pueden ser modificados para tener un giro libre de 360 grados, como un motor estándar. Los servomotores no giran continuamente, su rotación es restringida entre ángulos fijos.

Figura 19. **Servomotor HS-422 marca Hitec**



Fuente: http://www.emucit.com/urun/hitec_hs-422_deluxe_servo. Consulta: enero de 2013.

2.2.1.2. Control

Para controlar un servo motor se debe aplicar un pulso de duración y frecuencia específica. Todos los servos disponen de tres cables, uno para la alimentación (Vcc), uno para la masa o tierra (Gnd), y otro para aplicar un tren

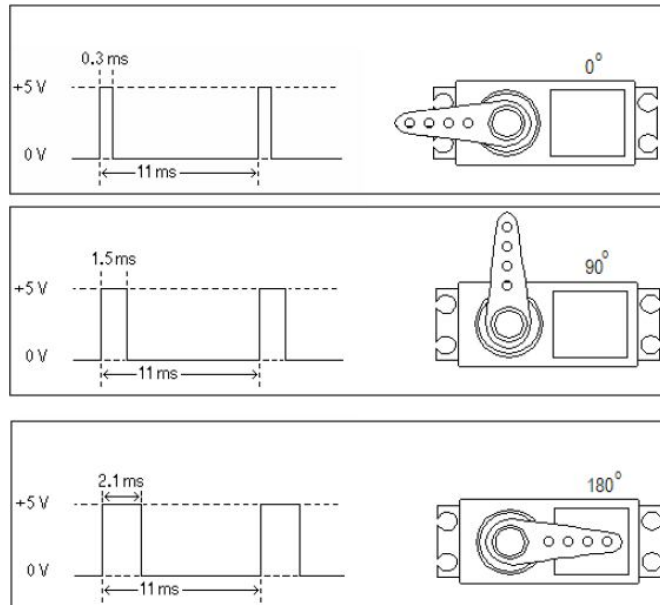
de pulsos, que hace que el circuito de control diferencial interno ponga el servomotor en la posición indicada.

2.2.1.3. Funcionamiento

La tensión de alimentación de los servos suele estar comprendida entre los 4 y 8 voltios. El control de posición lo efectúa el servomotor internamente, mediante un potenciómetro conectado mecánicamente al eje de salida y controlando una señal PWM (modulación de ancho de pulso) interna; para así compararla con la señal de entrada PWM externa mediante un circuito diferencial, y modifica la posición del eje de salida hasta que los valores se igualen y el servomotor pare en la posición indicada.

La duración de los pulsos indica el ángulo de giro del motor, como se muestra en la figura 20. Cada servomotor tiene sus márgenes de operación, que corresponden con el ancho del pulso máximo y mínimo que el servo entiende, y en principio, mecánicamente no puede sobrepasar; estos valores varían dependiendo del modelo de servomotor utilizado.

Figura 20. **Tren de impulsos para control de un servomotor**



Fuente:CapítuloV.http://catarina.udlap.mx/u_dl_a/tales/documentos/lep/arreortua_n_a/capitulo5.pdf. Consulta: enero de 2013.

En la actualidad, son muchos los fabricantes de servomotores de los cuales se pueden mencionar: Futuba, Hitec, Airtronics y radios JR, etc. Sus servos son los mismos a excepción de algunas diferencias de interfaz como los colores de alambre, tipo de conector, etc.

2.2.1.4. **Modulación por ancho de pulso PWM**

La modulación por ancho de pulso de una señal es una técnica en la que se modifica el ciclo de trabajo de una señal periódica, por ejemplo: una señal cuadrada; ya sea para transmitir información a través de un canal de comunicación o para controlar la cantidad de energía que se envía a una carga.

El ciclo de trabajo de una señal periódica es el ancho relativo de su parte positiva en relación con el período de la señal. Expresado matemáticamente se tiene:

$$C = \frac{t}{P}$$

Donde:

- C = ciclo de trabajo
- t = tiempo en que la función es positiva (ancho del pulso)
- P = período de trabajo

Para el presente caso, la señal PWM se utilizará para controlar el servomotor, ya que modifica su posición de acuerdo al ancho de pulso enviado a cada cierto período. Esta señal es generada por el microcontrolador PIC.

2.2.2. Sistema de control

Para el sistema de control del nuevo actuador se empleará un microcontrolador PIC, debido a que este tiene la capacidad de comunicarse con el bloque NXT vía I²C, de manejar señales PWM y, además cuenta con capacidad de memoria para grabar información del posicionamiento del nuevo actuador.

2.2.2.1. Microcontrolador PIC

Un microcontrolador es un circuito integrado programable que contiene todos los componentes necesarios para controlar el manejo de una tarea

determinada. En su interior se encuentran las tres unidades funcionales de una computadora: CPU, memoria y unidades de entrada/salida.

La utilización de un microcontrolador en un circuito, reduce el tamaño y número de componentes, y por ello disminuye el volumen y peso del dispositivo.

El microcontrolador es comúnmente llamado PIC y se ha escogido para el proyecto el PIC 16F877A de Microchip, por las siguientes razones:

- Costo accesible
- Fácil de encontrar en electrónicas locales
- Sus características cumplen con las necesidades del proyecto
- Manuales ampliamente explicados
- Fácil de programar

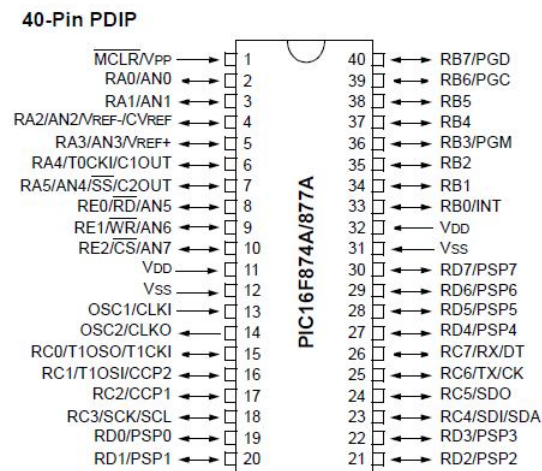
2.2.2.1.1. Características

Las principales características del PIC 16F877A son las siguientes:

- Frecuencia de operación en DC- 20 MHz.
- Hasta 8k x 14 bits de memoria flash de programa.
- 368 *bytes* de memoria de datos (RAM).
- 256 *bytes* de memoria de datos EEPROM.
- 33 puertos de entrada/salida.
- 3 timer: 2 de 8 y 1 de 16 bits.
- 2 módulos CCP (Captura, Comparador, PWM).
- Comunicación serial: MSSP (Master Synchronous Serial Port), USART (Universal Synchronous Asynchronous Receiver Transmitter).
- Comunicación paralelo: PSP.

- 1 convertidor A/D de 10-bits (8 canales).
- 2 comparadores analógicos.
- Set de 33 instrucciones.
- Voltaje de operación VDD de 2.5 hasta 5.5 V.
- Disponible en empaquetado de 40 pines PDIP (Plastic Dual In Line Package).

Figura 21. **PIC16F877A empaquetado de 40 pines**



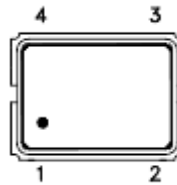
Fuente: Microchip Technology Inc. Microchip PIC16F87XA Data Sheet. p. 5.

2.2.2.1.2. El oscilador

El oscilador le indica al PIC la velocidad de trabajo, este genera una onda cuadrada de alta frecuencia que se utiliza como señal de reloj para sincronizar todas las operaciones del sistema. En el PIC16F877A los pines OSC1/CLK1 y OSC2/CLK0 son las líneas utilizadas para este fin.

La señal de reloj para el PIC16F877A será a través de un oscilador de cristal TTL de 20 MHz, el cual se conecto al pin 13 del PIC a utilizar. Las conexiones de este tipo de oscilador se muestran en la figura 22.

Figura 22. **Esquema de un oscilador de cristal TTL**



Fuente: elaboración propia.

Donde:

- Pin 1: no se conecta
- Pin 2: es masa o tierra (GND)
- Pin 3: salida
- Pin 4: alimentación de 5V (Vcc)

2.2.2.1.3. Puertos de E/S

El PIC16F877A tiene 5 puertos de entrada/salida denominados PORTA, PORTB, PORTC, PORTD y PORTE. Cada puerto puede ser configurado como entrada o como salida por medio de software, cada línea del puerto es independiente una de otra.

Las líneas de cada puerto son capaces de entregar niveles TTL cuando la tensión de alimentación aplicada en VDD es de 5V.

2.2.2.1.4. Módulos

El PIC cuenta con variedad de módulos, aquí solo se abarcarán los utilizados para el desarrollo y diseño del nuevo actuador, los cuales son: TIMER y MSSP.

El módulo MSSP se detallará en la sección de Sistemas de Comunicación, aquí se detalla el módulo TIMER.

- Módulo TIMER: el PIC16F877A cuenta con 3 módulos para controlar intervalos de tiempo, reciben el nombre de TIMER (temporizadores) los cuales son:
 - TIMER0
 - TIMER1
 - TIMER2

Los módulos TIMER pueden trabajar como temporizadores o contadores, para el presente proyecto se utilizará el módulo TIMER0 y TIMER1, como temporizadores para generar una señal PWM y poder así controlar el servomotor.

- TIMER0: el PIC16F877A posee un temporizador/contador de 8 bits que actúa de dos maneras diferentes:
 - ❖ Como contador de sucesos: que están representados por los impulsos que se aplican al pin RA4/T0CKI/C1OUT. Al llegar al valor FFh se desborda el contador y, con el siguiente impulso pasa a 00h, advirtiendo esta circunstancia

mediante la activación de un señalizador y/o provocando una interrupción.

- ❖ Como temporizador: cuando se carga en el registro un valor inicial se incrementa con cada ciclo de instrucción ($F_{osc}/4$) hasta que se desborda, pasa de FFh a 00h y activa la señalización y/o provoca una interrupción.

El actuar de una u otra forma depende del bit TOSC del registro OPTION, por ejemplo:

- ✓ Si TOSC =1, el TMR0 actúa como contador
- ✓ Si TOSC =0, el TMR0 actúa como temporizador

Los registros asociados con el TMR0 se presentan en la tabla V.

Tabla V. **Registro TMR0: bits de configuración**

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
01h,101h	TMR0	Timer0 Module Register								xxxx xxxx	uuuu uuuu
0Bh,8Bh,10Bh,18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
81h,181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Legend: x = unknown, u = unchanged, - = unimplemented locations read as '0'. Shaded cells are not used by Timer0.

Fuente: Microchip Technology Inc. PIC16F87XA Data Sheet. p. 56.

- TIMER1: consiste de 2 registros de 8 bits TMR1H y TMR1L, los cuales son de lectura y escritura. El par de registros del TMR1 incrementan de 0000h a FFFFh.
 - ❖ Al igual que el TMR0, el TMR1 puede operar como temporizador o contador, el modo de operación es determinado por el bit de control TMR1CS del registro T1CON.
 - ❖ En modo de temporizador, el TIMER1, incrementa en cada ciclo de instrucción y en modo de Contador incrementa en cada flanco de subida de la entrada de un reloj externo.
 - ❖ La interrupción del TMR1 puede ser habilitada/deshabilitada mediante el bit TMR1IE del registro PIE1, si es habilitada la interrupción al ocurrir un desbordamiento el bit TMR1IF del registro PIR1 que actúa como bandera, se pondrá en 1 y debe de ser limpiado por medio de software.

Los registros asociados con el TMR1 se presentan en la tabla VI.

Tabla VI. **Registro TMR1: bits de configuración**

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh, 8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF ⁽¹⁾	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE ⁽¹⁾	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000	--uu uuuu

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the Timer1 module.

Note 1: Bits PSPIE and PSPIF are reserved on the 28-pin devices; always maintain these bits clear.

Fuente: Microchip Technology Inc. Microchip PIC16F87XA Data Sheet. p. 62.

2.2.2.2. Herramienta de programación

El PIC dispone de una memoria de programación interno donde se almacena el programa que lo controla y el cual consiste en un código hexadecimal. El programa de control se graba en la memoria de programa mediante un equipo físico llamado grabador o quemador, el cual se conecta a una PC a través de un puerto serial, paralelo o USB.

2.2.2.2.1. Proceso de desarrollo

Para el proceso de desarrollo de una aplicación basada en un microcontrolador se siguen las siguientes etapas:

- Desarrollo de software: corresponde a la escritura y compilación/ensamblaje del programa que regirá las acciones del PIC. El

programa puede desarrollarse de las siguientes dos maneras: lenguaje bajo nivel (ensamblador) y de alto nivel.

- Proceso de grabación: se refiere a utilizar un programa en la PC que toma el código ensamblado (.hex, bin, etc.), para cargarlo a través de un puerto de la PC al dispositivo grabador que lo escribe en la memoria del microcontrolador.

2.2.2.2. Programa MPLAB

El programa ensamblador se encarga de traducir los nemónicos y símbolos alfanuméricos del programa escrito en ensamblador a código máquina, para que pueda ser interpretado y ejecutado por el PIC.

El entorno de MPLAB IDE (Entorno de Desarrollo Integrado) es un software que se ejecuta bajo el sistema operativo de Windows, el cual es muy fácil de aprender y de utilizar. MPLAB permite editar el archivo fuente del proyecto y también ensambla y simularla en pantalla la evolución, tanto de memoria de datos RAM, como la ROM, los registros SFR, etc., según progresa la ejecución del programa.

Características del software MPLAB:

- Incorpora todas las utilidades necesarias para la realización de cualquier proyecto basado en microcontroladores.
- Permite editar el archivo fuente en lenguaje ensamblador y además de ensamblarlo y simularlo en pantalla.

- Depurador de código fuente muestra las instrucciones en ensamblador conforme se va ejecutando, permite la ejecución paso a paso, por rutina y puntos de ruptura en el programa.
- El programa es completamente interactivo, lo que permite modificar cualquier registro o localidad de memoria en cualquier momento.
- Es un software gratuito, se puede bajar en la dirección internet del fabricante microchip www.microchip.com. Última consulta: enero de 2013.

2.2.3. Sistema de comunicación

En el sistema de comunicación se utiliza el protocolo de comunicación I²C, en las siguientes secciones se describirá a detalle de las características y manejo de este protocolo, así como también, los diferentes formatos.

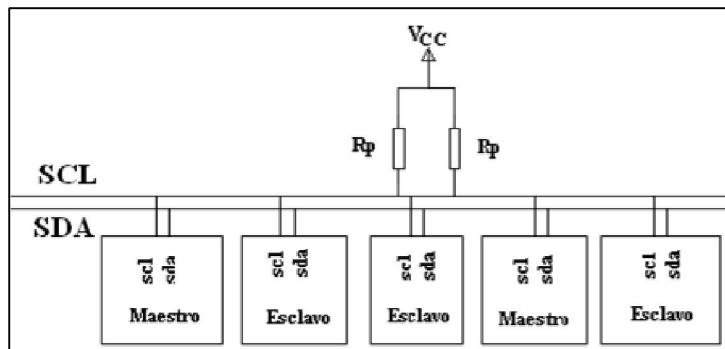
2.2.3.1. Bus I²C

Bus serie desarrollado en los años 80's por Philips Semiconductors (ahora NXP Semiconductors), que actualmente es utilizado en una variedad de arquitecturas de control y además está integrado en más de 1 000 diferentes circuitos, el bus requiere de poco hardware y un mínimo cableado que incorpora una serie de procedimientos para la correcta comunicación entre los componentes del sistema.

El bus está formado por dos líneas llamadas dato serial (SDA) y reloj serial (SCL), la transferencia de datos es bidireccional y su tasa puede llegar arriba de los 100kbit/s en modo estándar.

Las líneas SDA y SCL están polarizadas a positivo mediante una resistencia de *pull-up* de forma que en reposo están a nivel alto. El bus es manejado por el dispositivo maestro el cual genera la señal SCL y controlan la comunicación y el dispositivo esclavo responde a peticiones del maestro. En la figura 23 se muestra la conexión de varios dispositivos en el bus.

Figura 23. Estructura de un bus I²C



Fuente: http://www.nxp.com/documents/user_manual/UM10204.pdf. Consulta: enero de 2013.

El bus se basa en tres señales:

- SDA por la cual viajan los datos entre los dispositivos.
- SCL por la cual transitan los pulsos de reloj que sincronizan el sistema.
- GND (masa) interconectada entre todos los dispositivos enganchados al bus.

Los dispositivos conectados al bus mantiene un protocolo de comunicaciones del tipo maestro/esclavo. Las funciones del maestro y del esclavo se diferencian en:

- El circuito maestro inicia y termina la transferencia de información, además de controlar la señal de reloj.
- El esclavo es el circuito direccionado por el maestro.

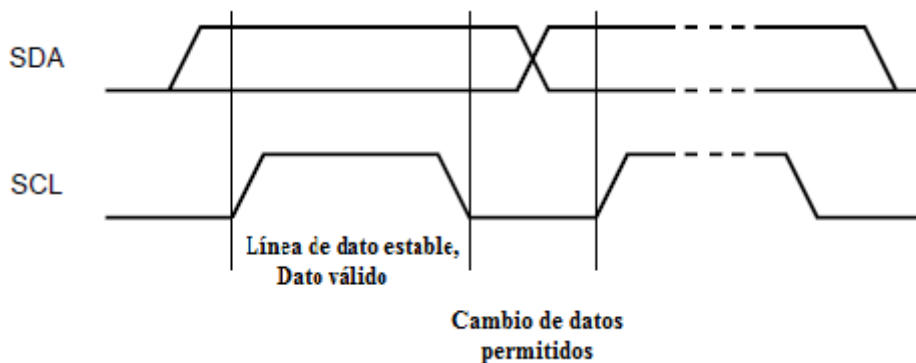
Para el presente proyecto, el bloque NXT es el dispositivo maestro y el PIC como el esclavo.

Cada dispositivo conectado al bus I²C es reconocido por una única dirección que lo diferencia del resto de los circuitos conectados.

2.2.3.2. Transferencia de datos

Para transferir un bit por la línea de datos SDA, debe ser generado un pulso de reloj por la línea SCL. Los bits de datos transferidos por la línea SDA deben mantenerse estables mientras la línea SCL esté a nivel alto. El estado de la línea SDA sólo puede cambiar cuando la línea SCL está a nivel bajo.

Figura 24. **Transferencia de un bit en el bus I²C**

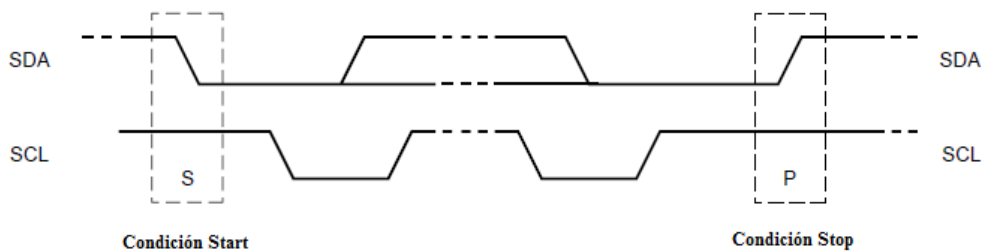


Fuente: http://www.nxp.com/documents/user_manual/UM10204.pdf. Consulta: enero de 2013.

- Condición de *start* y *stop*

Para indicar que el bus está libre las líneas de reloj (SCL) y de datos (SDA) deben estar a nivel alto. Una vez que se ha verificado esto, el transmisor procederá a enviar un bit en cada pulso de reloj.

Figura 25. **Condiciones *start* y *stop***



Fuente: http://www.nxp.com/documents/user_manual/UM10204.pdf. Consulta: enero de 2013.

La condición de *start* indica el comienzo de la transferencia de datos, la línea SDA debe estar en flanco de bajada, mientras que la línea SCL permanece a nivel alto.

La condición de *stop* indica el fin de la transferencia de datos, la línea SDA debe estar en flanco de subida mientras que SCL permanece a nivel alto.

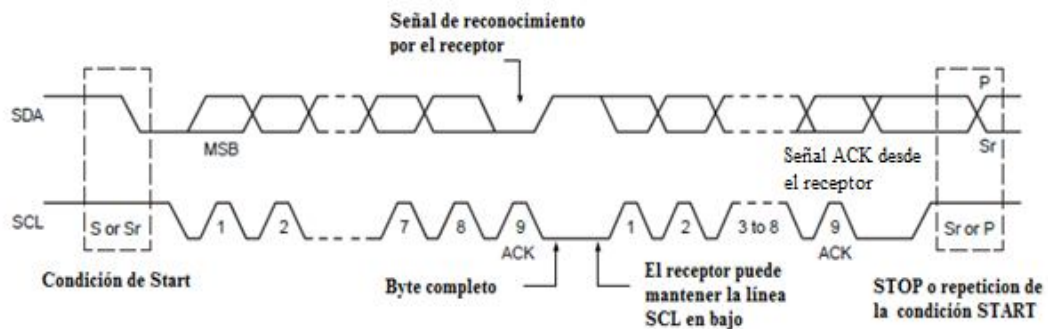
2.2.3.3. Protocolo de comunicación I²C

Para iniciar una comunicación entre dispositivos conectados al bus I²C se debe respetar un protocolo.

Cada dato enviado por la línea SDA debe tener una longitud de un *byte*. El número de *bytes* que se puede enviar no tiene restricción. El *byte* de datos se transfiere empezando por el bit 7 que es de mayor peso, denominado MSB (*most significant bit*).

Una vez que se ha transmitido el *byte* el receptor deberá mandar un bit de reconocimiento (*acknowledgement*) en el noveno pulso de reloj. El receptor ejecuta este reconocimiento poniendo la señal SDA a un nivel bajo. Cada grupo de *byte* debe ser asentido.

Figura 26. **Protocolo de comunicación I²C**



Fuente: http://www.nxp.com/documents/user_manual/UM10204.pdf. Consulta: enero de 2013.

El bit de reconocimiento ACK es obligatorio en la transferencia de datos. El pulso de reloj SCL correspondiente al bit de reconocimiento que es generado por el maestro. El transmisor deja libre la línea SDA (la pone en alta impedancia o a nivel alto). El receptor ha de hacer que la línea SDA pase a nivel bajo estable durante el periodo alto del noveno pulso de reloj SCL.

Si el esclavo – receptor que está direccionado no desea recibir más *bytes*, el esclavo no genera el bit ACK en el último *byte* quedando la línea SDA a nivel

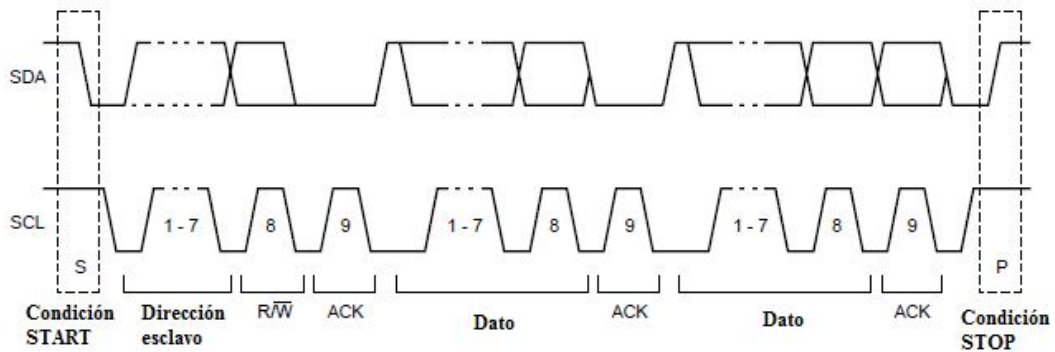
alto, lo cual es detectado por el maestro que puede generar la condición *stop* o *start*, para aborta la transferencia de datos anterior.

Si un maestro – receptor está recibiendo datos de un esclavo – transmisor, debe generar un bit ACK tras cada *byte* recibido. Para finalizar la transferencia de datos no debe generar el bit ACK tras el último *byte* enviado por el esclavo. El esclavo – transmisor debe dejar libre la línea de datos SDA, para permitir que el maestro genere la condición de *stop* o de *start*.

2.2.3.4. Formatos de transferencia de datos

Las transferencias de datos siguen el formato que se muestra en la figura 27. Después de la condición de *start* (S), se envía una dirección del dispositivo esclavo. Esta dirección tiene una longitud de 7 bits y es seguido de un octavo bit el cual indica la dirección de los datos (R /W) un cero indica una transmisión de escritura y un 1 indica una solicitud de datos de lectura. Una transferencia de datos siempre se termina por una condición de parada (P) generados por el dispositivo maestro. Sin embargo, si un maestro todavía desea comunicarse en el bus, se puede generar la condición *start* repetida (Sr) y enviará la dirección de otro dispositivo esclavo sin generar una condición de parada. Varias combinaciones de formatos de lectura/escritura son posibles dentro de dicha transferencia.

Figura 27. **Transferencia de datos completa**

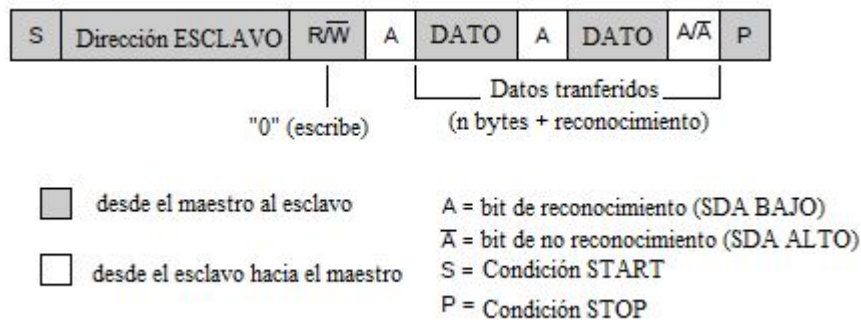


Fuente: http://www.nxp.com/documents/user_manual/UM10204.pdf. Consulta: enero de 2013.

Los posibles formatos de transferencia son:

- 1° Maestro-emisor: transmite al esclavo-receptor. Si el bit 8 es de escritura ($R/W=0$), la secuencia se muestra en la figura 28.

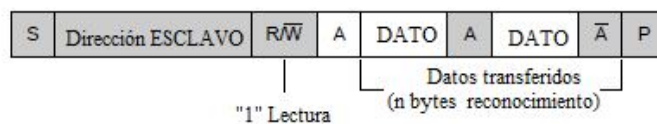
Figura 28. **Maestro escribe datos en el esclavo**



Fuente: http://www.nxp.com/documents/user_manual/UM10204.pdf. Consulta: enero de 2013.

- 2° Maestro-receptor: lee de un esclavo-emisor inmediatamente después del primer *byte*. Si el bit 8 es de lectura, (R/W), la secuencia se muestra en la siguiente figura.

Figura 29. **Maestro lee datos del esclavo**

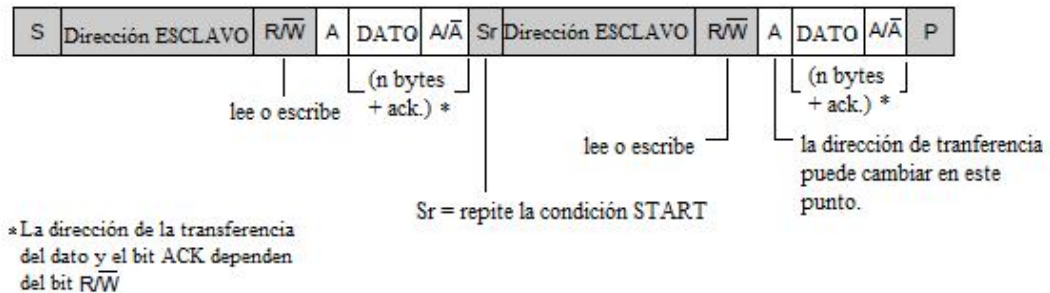


Fuente: http://www.nxp.com/documents/user_manual/UM10204.pdf. Consulta: enero de 2013.

- 3° Formato combinado: una transferencia de datos siempre acaba con una condición de *stop* generada por el maestro. Sin embargo, si un maestro todavía desea comunicarse con el bus, puede generar repetidamente condiciones de *start* y direccionar a otro esclavo sin generar, primero la condición de *stop*, ver figura 30. Durante un cambio de dirección dentro de una transferencia, la condición de *start* y la dirección del esclavo se repiten, pero con el bit R/W invertido.

Varias combinaciones de lectura y escritura son posibles dentro de una misma transferencia de datos.

Figura 30. **Formato de transferencia combinado**



Fuente: http://www.nxp.com/documents/user_manual/UM10204.pdf. Consulta: enero de 2013.

Para comprender la diferencia entre utilizar el formato de transferencia combinado y los formatos maestro escribe datos en el esclavo y maestro lee datos desde el esclavo, se presenta a continuación el protocolo lógico de comunicación propio de LEGO.

- Formato de transferencia combinado: en la tabla VIII se muestra la secuencia de *bytes* que transmite el bloque NXT.

Tabla VII. **Protocolo lógico de comunicación de LEGO**

	Transmitted from NXT			
Command	Byte 0	Byte 1	Byte 2	Lenght
Constants				
Read version	Device address	0x00	R + 0x03	8

Fuente: The LEGO Group. LEGO® MINDSTORMS® NXT Hardware Devoloper Kit. p.10.

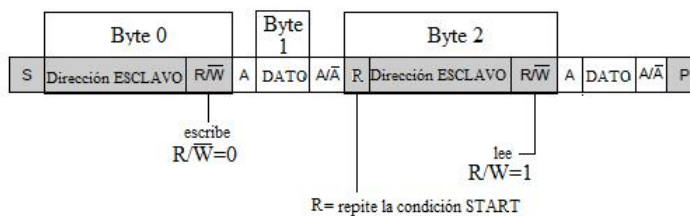
El primer *byte* (0) trasmitido desde el bloque NXT es la dirección del dispositivo esclavo junto con el bit R/W=0. La dirección del esclavo para este

ejemplo es b'0000001, por consiguiente el *byte* transmitido tiene el valor b'00000010 (0x02)

El segundo *byte* (1) transmitido es el comando *read version* que tiene el valor asignado de b'00000000 (0x00).

El tercer *byte* (2) transmitido R+0x03, indica que el bloque NXT generará una condición de *start* nuevamente (Sr) y enviará en el siguiente *byte*, en este caso en el *byte* 2, la dirección del dispositivo esclavo con el bit R/W=1. La letra R significa *repeated start*. El significado de R+0x03 es: condición Sr más dirección del dispositivo esclavo junto con el bit R/W=1. Por consiguiente el valor del *byte* a transmitir es b'00000011 (0x03) que es la dirección del dispositivo esclavo=b'0000001 que son los primeros 7 bits más el bit R/W=1; el resultado es b'00000011 (0x03). En la figura 31 se muestra el formato de transmisión.

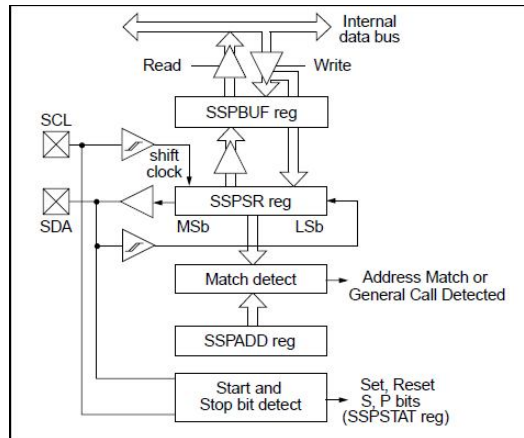
Figura 31. Trama de transmisión de *bytes* combinado



Fuente: elaboración propia.

- Formato maestro escribe/lee al dispositivo esclavo: para realizar la secuencia presentada en el inciso anterior utilizaremos primero el formato maestro escribe datos al esclavo y luego la secuencia maestro lee datos desde el esclavo, la secuencia se muestra en la figura 32.

Figura 33. Diagrama de bloques I²C modo esclavo



Fuente: Microchip Technology Inc. PICmicro Mid-Rang MCU Family Reference Manual. p. 279.

Dos líneas son usadas para la transferencia de datos, el pin (18 – RC3/SCK/ SCL) SCL, el cual es el reloj; y el pin (23 – RC4/SDI/SDA) SDA el de datos. Estas líneas son automáticamente configuradas cuando el módulo MSSP I²C esclavo es habilitado. Sin embargo, antes de habilitar el módulo, las líneas del puerto deben ser configuradas como entradas o salidas a través del registro TRISC < 4:3 > bits.

2.2.3.5.1. Registros

El módulo MSSP dispone de 6 registros asociados:

- MSSP *control register1* (SSPCON1)
- MSSP *control register2* (SSPCON2)
- MSSP *status register* (SSPSTAT)
- MSSP *serial receive/transmit buffer* (SSPBUF)
- MSSP *shift register* (SSPSR) – No es directamente accesible

- MSSP *address register* (SSPADD)

Los registros SSPCON1 y SSPCON2 son de control y permite la escritura/lectura de sus bits.

Registro SSPSTAT indica el estado en modo de operación I²C, los últimos 6 bits menos significativos son únicamente de lectura, mientras que los dos bits más significativos son de escritura/lectura.

El registro SSPSR es el registro de desplazamiento de datos de entrada o salida, este registro no es accesible.

El registro SSPBUF es el registro búfer de *bytes* de datos, utilizado para el envío o recepción de datos.

El registro SSPADD contiene la dirección designada al dispositivo esclavo cuando se configura el MSSP en modo esclavo I²C. Cuando el MSSP está configurado en modo maestro, los siete bits menos significativos del registro actúan como el valor de recarga del generador de velocidad de transmisión en baudios.

En la operación de recepción, el registro SSPRS y SSPBUF ambos crean un doble búfer, cuando SSPSR recibe un *byte* completo, este es transferido al registro SSPBUF y la interrupción del registro SSPIF se pone en 1. En la transmisión, el registro SSPBUF no es doble búfer.

2.2.3.5.2. Operación

Las funciones del módulo MSSP son habilitadas colocando a uno el bit 5 del registro SSPEN.

El registro SSPCON permite el control de operación del modo I²C, son cuatro bits (3:0) los configurados para operar en uno de los siguientes modos:

- I²C modo maestro, reloj esta dado por OSC/4 (SSPADD + 1).
- I²C modo esclavo utilizando 7 bits para asignación de la dirección.
- I²C modo esclavo utilizando 10 bits para asignación de la dirección.
- I²C modo esclavo utilizando 7 bits para asignación de la dirección con *start* y *stop* bits de interrupción.
- I²C modo esclavo utilizando 10 bits para asignación de la dirección con *start* y *stop* bits de interrupción.
- I²C.
- I²C Firmware modo maestro controlado, esclavo libre.

En la selección de cualquier modo I²C con el bit puesto a uno del registro SSPEN, fuerza los pines SCL y SDA a ser salidas de drenador abierto, siempre y cuando estos pines estén programados como entradas mediante el establecimiento de los bits apropiados del registro TRISC. Para garantizar el correcto funcionamiento del módulo, resistencias *pull-up* externas deben ser colocadas en los pines SCL y SDA.

2.2.3.5.3. Modo esclavo

En modo esclavo, las líneas SCL y SDA deben ser configuradas como entradas (TRISC <4:3>). El módulo MSSP anulará el estado de la entrada cuando los datos de salida sean requeridos (esclavo- trasmisor).

Cuando se da una coincidencia entre la dirección que aparece en el bus y la que tiene asignada el dispositivo esclavo, el hardware generará de manera automática el bit de reconocimiento (ACK negado) sobre la línea SDA, lo mismo sucederá tras haber detectado coincidencia en la dirección después de cada dato que se complete salvo la bandera BF del registro SSPSTAT <0> (indicación de búfer lleno) o SSPOV del registro SSPCON <6> (desbordamiento) estén activos.

2.2.3.5.4. Direccionamiento

Luego de habilitar el módulo MSSP éste se queda esperando por una condición de *start*. Tras ésta, se introducen 8 bits en el registro SSPSR.

Se comparan 7 de los bits recibidos SSPSR <7:1> con la dirección almacenada en el registro SSPADD y si coincide y, además los bits BF y SSPOV están a un nivel 0, se dan los siguiente eventos:

- El registro SSPBUF se carga con el contenido del registro SSPSR.
- Cuando el búfer está lleno, el bit BF se pone a 1 en el 8° flanco de bajada en la línea SCL.
- Se genera un pulso de reconocimiento ACK negado.

- El bit de bandera del módulo MSSP del registro SSPIF (PIR<3>) se pone a 1 (la interrupción es generada si fue habilitada) en el 9° flanco de bajada de la línea SCL.

2.2.3.5.5. Recepción

Se produce cuando hay una coincidencia de dirección y además, el bit R/W negado es 0 (el que acompaña a la dirección). En ese caso el bit R/W negado del registro SSPSTAT se pone a 0, se genera el pulso de reconocimiento en la línea SDA y la dirección recibida se carga en el registro SSPBUF (si no se dio una condición de desbordamiento con BF ó SSPOV a 1).

2.2.3.5.6. Transmisión

Se da si hay una coincidencia con la dirección en el bus y, además el bit R/W negado es 1 (el cual es el último bit del *byte*). En ese caso el bit R/W negado del registro SSPSTAT se pone a 1 y se genera el pulso de reconocimiento en la línea SDA y la dirección recibida se carga en el registro SSPBUF.

Se retiene la generación de reloj del dispositivo maestro, porque el dispositivo esclavo mantiene la línea SCL a 0 mientras prepara el envío del dato. El dato que el esclavo debe enviar lo carga en el registro SSPBUF, que a su vez, carga al registro SSPSR. Para liberar la línea SCL, se debe poner el bit CKP del registro SSPCON <4> a 1 por software, ya que se puso a 0 por hardware donde se indica que está listo el dato a enviar.

Si al finalizar el envío de un *byte* por parte del dispositivo esclavo, el maestro no sitúa el pulso de reconocimiento en la línea SDA (no hay ACK

negado), se interpreta que se acabó la transferencia de datos, se resetea la lógica del esclavo y queda a la espera de una nueva condición de *start*.

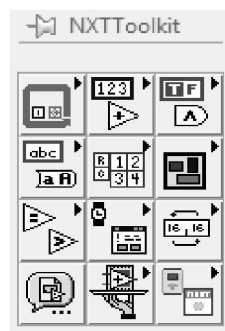
2.3. Elementos de programación del módulo bloque Pinza NXT en LabVIEW y Toolkit para LEGO® MINDSTORMS® NXT

Labview desarrolló el Toolkit para LEGO® MINDSTORMS® NXT para poder programar el bloque NXT. El Toolkit puede ser descargado en la página siguiente:

<http://zone.ni.com/devzone/cda/tut/p/id/4435>, última fecha de consulta: octubre de 2012.

El Toolkit proporciona cuatro nuevas clases de números de referencia genéricos (refnums) y un conjunto de propiedades y/o métodos de cada clase de refnums. Cada propiedad/método ligeramente corresponde a una pieza particular de la interfaz de E/S proporcionada para el intérprete de bytecode del NXT.

Figura 34. Funciones de NXT Toolkit en LabVIEW



Fuente: elaboración propia.

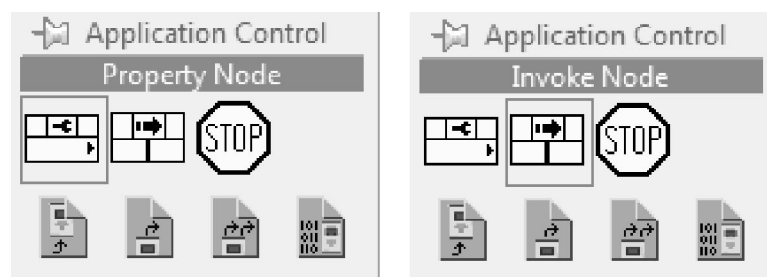
El Toolkit ofrece las siguientes cuatro nuevas clases de Refnums genéricos:

- NXTInput
- NXTOutput
- NXTOutputMulti
- NXTSyscall

Los Refnums de NXTInput, NXTOutput, NXTOutputMulti son utilizados con el control de aplicación Nodo de Propiedad (Property Node), para las interfaces de dispositivos de entrada y salida.

Los Refnums de NXTSyscall se utilizan con el control de aplicación Invocación de Nodo (Invoke Node), para los métodos a nivel de sistema y poder acceder a varias características del firmware del NXT. Cada método que se utiliza corresponde a una llamada de una función interna del sistema la cual coincide con el nombre del método. La mayoría de los métodos de llamada al sistema proporcionan códigos de condición de valores.

Figura 35. **Funciones de NXT Toolkit: Invoke Node y Property Node**



Fuente: elaboración propia.

El NXTSyscall cuenta con los siguientes métodos:

- NXTCommLSCheckStauts
- NXTCommLSWrite
- NXTCommLSRead

Existen muchos más, pero los anteriormente mencionados son los que se utilizan para el desarrollo del bloque Pinza NXT.

2.3.1. Dispositivo digital E/S

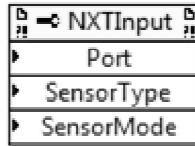
Para la programación de dispositivos digitales es necesario habilitar y configurar el puerto a utilizar, para ello se utiliza la interfaz NXTInput y para acceder al subsistema de comunicación I²C se utiliza la interfaz NXTSyscall. En particular el puerto puede ser configurado como entrada analógica o digital.

2.3.1.1. Configuración del puerto E/S

Para la programación del puerto, en primer lugar se debe crear la aplicación de control Property Node, luego se debe configurar la clase NXTInput. Seguido de ello se deben añadir los siguientes elementos:

- Port: sirve para especificar el puerto a utilizar
- Sensor Type: sirve para especificar el tipo de sensor a utilizar
- SensorMode: influye en el valor de escala.

Figura 36. **Configuración: función Property Node clase NXTInput**



Fuente: elaboración propia.

El firmware soporta los siguientes dos valores del elemento Sensor Type para utilizar dispositivos digitales:

- LOWSPEED: se utiliza si el dispositivo no requiere 9 voltios.
- LOWSPEED_9V: se utiliza si el dispositivo requiere 9 voltios, este es utilizado por el sensor ultrasónico NXT.

2.3.1.2. Comunicación de baja velocidad

Para comunicarse con el dispositivo a través de la comunicación I²C es necesario acceder al subsistema de comunicación I²C del NXT firmware. Se utiliza Refnums NXTSyscall con Invoke Node, para exponer la interface del sistema de comunicación.

El Toolkit proporciona los siguientes tres métodos para la comunicación digital de baja velocidad:

- NXTCommLSWrite
- NXTCommLSCheckStatus
- NXTCommLSRead

El firmware del NXT es el responsable de controlar las operaciones de lectura y escritura en los búferes de cada puerto. El bloque NXT será siempre el dispositivo maestro para la comunicación I²C, y los tres métodos de NXTSyscall habilitan al programador el acceso a los búferes.

Una llamada a NXTCommLSWrite constituye el inicio de una operación asíncrona entre el bloque NXT y un dispositivo digital, de tal manera que el programa sigue funcionando mientras que el firmware gestiona el envío de *bytes* del búfer de escritura y lee la respuesta de *bytes* provenientes del dispositivo. Debido a que el bloque NXT es el dispositivo maestro, también debe especificar el número de *bytes* que se espera del dispositivo esclavo en respuesta a cada operación de escritura.

Después de haber iniciado una transacción de escritura con NXTCommLSWrite, es recomendable utilizar la llamada a NXTCommLSCheckStaus en un bucle While para comprobar el estado del puerto. Si NXTCommLSCheckStaus devuelve un código de estado igual a cero, el búfer de lectura tiene *bytes* disponibles; el sistema está listo para el uso de la llamada de NXTCommLSRead, para copiar los datos desde el búfer de lectura en su propio búfer.

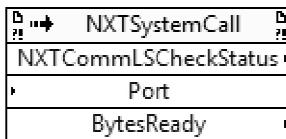
Tomar en cuenta que ninguna de estas llamadas puede devolver varios códigos de estado en cualquier momento. Un código de estado de cero significa que el puerto está inactivo y la última transacción (si existe) no produjo ningún error. Código de estado negativo indica que ha ocurrido un error, mientras que el código de estado positivo indica que una transacción está en curso en el puerto especificado.

- **NXTCommLSCheckStatus**

Este método de sistema de llamada chequea el estatus de la comunicación I²C en el puerto especificado por el programador. Si la última operación en este puerto fue satisfactoria la operación `NXTCommLSWrite` solicitó datos de respuesta de un dispositivo. El `BytesReady` indica el número de *bytes* en el búfer de lectura interna.

Si el valor de retorno es cero, el puerto está libre y la última operación no causó ningún error.

Figura 37. **Función Invoke Node clase NXTCommLSCheckStatus**



Fuente: elaboración propia.

Este método puede retornar los siguientes códigos diferentes de cero

Tabla VIII. **Códigos de error: NXTCommLSCheckStatus**

Decimal	Hexadecimal	Nombre	Significado
32	0x20	STAT_CONMM_PENDING	El puerto está ocupado realizando una transacción
-35	0xDD	ERR_COMM_BUS_ERR	La ultima transacción fallo
-33	0xDF	ERR_COMM_CHAN_INVALID	Puerto especificado está fuera del rango
-32	0xE0	ERR_COMM_CHAN_NOT_READY	Puerto especificado no está correctamente configurado

Fuente: LabVIEW. LabVIEW Toolkit for LEGO® MINDSTORMS® NXT Programming Guide.

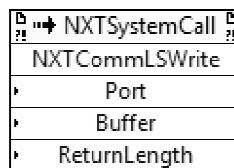
p.17.

- NXTCommoLSWrite

Este método de llamada al sistema copia los datos del búfer de entrada a un búfer de escritura interno e instruye al NXT firmware, para realizar una transacción enviando al búfer de escritura el dato que se enviará al dispositivo, y luego lee el *byte* ReturnLength de nuevo en el búfer de lectura interna.

Si el valor de retorno es cero, el método empezó una transacción de comunicación con éxito. Se utiliza el método NXTCommLSCheckStatus, para monitorear el estado de la transacción.

Figura 38. **Función Invoke Node clase NXTCommLSWrite**



Fuente: elaboración propia.

Este método puede retornar los siguientes valores diferentes a cero:

Tabla IX. **Códigos de error NXTCommLSWrite**

Decimal	Hexadecimal	Nombre	Significado
-33	0xDF	ERR_COMM_CHAN_INVALID	El puerto especificado esta fuera de rango
-32	0xE0	ERR_COMM_CHAN_NOT_READY	El puerto está ocupado o no está correctamente configurado
-19	0xED	ERR_INVALID_SIZE	El tamaño del búfer o ReturnLength superó los 16 <i>bytes</i> máximos permitidos.

Fuente: LabVIEW. LabVIEW Toolkit for LEGO® MINDSTORMS® NXT Programming Guide.

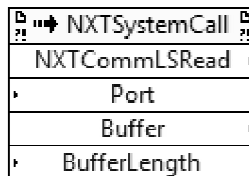
p.18.

- **NXTCommLSRead**

Este método de llamada al sistema copia los *bytes* de BufferLengh desde el búfer de lectura interna para el almacenamiento interno de la información leída desde el dispositivo.

Si el valor de retorno es cero, la operación de lectura logró leer el dato y el búfer contendrá todos los *bytes* disponibles en el búfer interno. Las sucesivas llamadas al método NXTCommoLSRead leerá un nuevo dato cada vez.

Figura 39. **Función Invoke Node clase NXTCommLSRead**



Fuente: elaboración propia.

Este método puede retornar los siguientes valores diferentes a cero

Tabla X. **Códigos de error NXTCommLSRead**

Decimal	Hexadecimal	Nombre	Significado
32	0x20	STAT_COMM_PENDING	El puerto está ocupado para realizar una transacción
-35	0xDD	ERR_COMM_BUS_ERR	La ultima transacción fue fallida
-33	0xDF	ERR_COMM_CHAN_INVALID	El puerto especificado esta fuera de rango
-32	0xE0	ERR_COMM_CHAN_NOT_READY	El puerto especificado no está correctamente configurado

Fuente: LabVIEW. LabVIEW Toolkit for LEGO® MINDSTORMS® NXT Programming Guide.

2.3.1.3. Secuencia de control

NXT Toolkit proporciona un control especial de tipo booleano llamado Sequence Flow. En la programación de un nuevo bloque de programa para el NXT-G es esencial que se coloquen dos controles: uno de salida y el otro de entrada; el nombre de cada uno debe comenzar como Sequence Flow In/Out. Es necesario para el programa debido a que si la secuencia de flujo booleano en un VI esté conectado a otro VI del programa, éste sea capaz de controlar el flujo de secuencia del programa principal.

Para ubicar el control Sequence Flow se debe estar en el panel frontal, clic derecho, All Controls, NXT Toolkit, Sequence Flow.

Figura 40. Ubicación del control Sequence Flow



Fuente: elaboración propia.

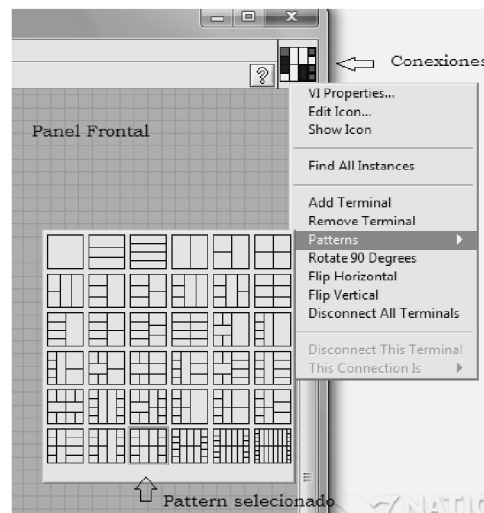
2.3.1.4. Conexión de ícono

A cada programa realizado se le asocia un ícono, el cual debe tener un patrón de conexión. Cada variable de entra/salida debe tener una conexión a una terminal del patrón seleccionado. Para realizar las conexiones se

selecciona con el botón derecho del *mouse* en el ícono del panel frontal ubicado en la parte superior derecha y se selecciona la opción Show Connector, luego se realizan las conexiones con las variables entrada/salida ubicadas en el panel frontal. Se recomienda conectar las entradas a la izquierda y las salidas a la derecha.

LabVIEW tiene diferentes patrones de conexión, para visualizar estas opciones solamente es de dar un clic al botón derecho del *mouse* y escoger la opción Patterns, sin embargo, el programa para el nuevo bloque programable únicamente trabaja con el patrón selección, el cual se muestra en la figura 41.

Figura 41. **Conexiones del ícono y los diferentes esquemas de conexión**



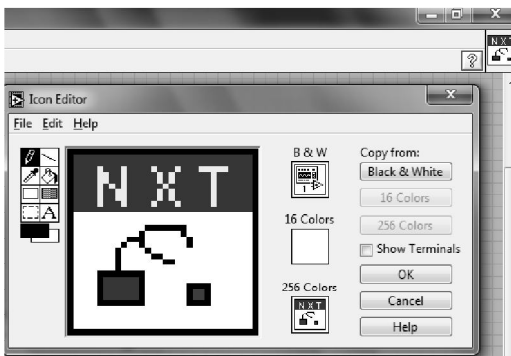
Fuente: elaboración propia.

Se debe de tomar en cuenta que la función Generic Refnum Name no se conecta a ningún conector del ícono.

2.3.1.5. Edición de ícono

Para editar el ícono se debe seleccionar Edit Icon haciendo clic con el botón derecho del *mouse* en el ícono del panel frontal. En este editor se puede dibujar el ícono deseado por el programador para identificar gráficamente el elemento creado.

Figura 42. Editor de ícono



Fuente: elaboración propia.

2.3.2. Creación de nuevos bloques de programa

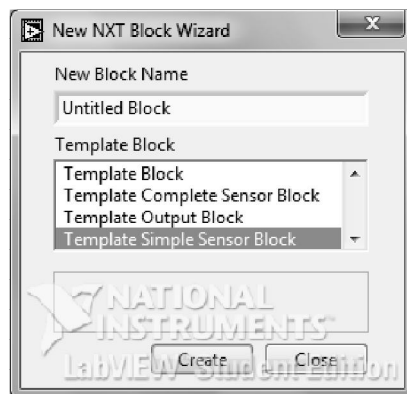
En el capítulo 1 se mencionó que el entorno del software NXT-G fue desarrollado en LabVIEW 7.1, y se debe de tener esta versión para añadir nuevos bloques de programa.

El software puede ser adquirido desde el sitio web de National Instruments <https://lumen.ni.com/nicif/us/infoLEGOlvcd/content.xhtml>, ultima fecha de consulta: octubre de 2012. Cabe mencionar que la empresa National Instruments solicita una serie de datos para poder enviar vía correo el software al nuevo programador.

LabVIEW ofrece la creación de un conjunto de plantillas VI para la creación de nuevos bloques las cuales son generadas para ser fácilmente exportadas al software NXT-G. Para crear un nuevo bloque hay que colocarse en la barra de herramientas >> Tools >> NXT Module >> New Block Wizard. En el menú aparecen cuatro tipos de plantillas:

- Template Block
- Template Complete Sensor Block
- Template Output Block
- Template Simple Sensor Block

Figura 43. **Opciones de New NXT Block Wizard**

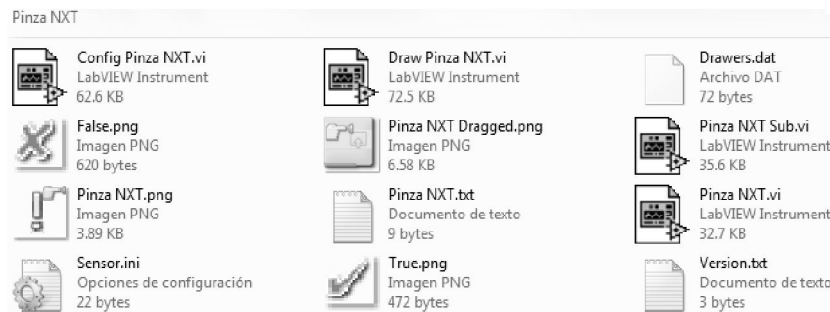


Fuente: elaboración propia.

Para el desarrollo del nuevo bloque se escogerá la última plantilla Template Simple Sensor Block, donde solicita que se agregue el nombre para el nuevo bloque programa, el cual se llamará Pinza NXT, y luego se deberá indicar la ubicación donde se guardarán los programas VI's que son generados.

El software generará una carpeta con el nombre Pinza NXT en la dirección indicada por el programador y dentro de éste se encontrarán los siguientes VI's generados, los cuales son mostrados en la figura 44.

Figura 44. **Carpeta Pinza NXT**



Fuente: elaboración propia.

El número total de archivos creados son 12, Config Pinza NXT.vi y Draw Pinza NXT.vi son utilizados por el entorno de programación de NXT-G, para configurar y dibujar el bloque; Pinza NXT.vi y Pinza NXTSub.vi programas del bloque que son ejecutados dentro del bloque NXT. El resto de los archivos son usados para graficación y soporte.

2.3.2.1. **Ícono Pinza NXT**

La función Wizard, automáticamente crea una imagen, la cual asocia al nuevo bloque para ser usado cuando se selecciona el bloque programa desde la paleta del programa NXT-G. Se tiene más opciones para cambiar esta imagen, las cuales pueden encontrarse en la siguiente carpeta: \LEGO software\LEGOMINDSTORMS NXT\engine\icons\MyBlock, ver figura 45; para realizar el cambio de imagen se debe copiar de la carpeta antes mencionada y

pegar en el archivo creado por Wizard, para el presente caso, en Pinza NXT, seguido a ello la nueva imagen deberá tener el mismo nombre del archivo.

Figura 45. **Carpeta MyBlock: íconos**



Fuente: elaboración propia.

2.3.2.2. **Ícono de arrastre**

Este es una imagen de arrastre, la cual es visible cuando se selecciona el bloque programa desde la paleta del programa NXT-G hacia el área de trabajo. Como es un archivo con extensión .png se puede cambiar la imagen por otra como en el caso del ícono.

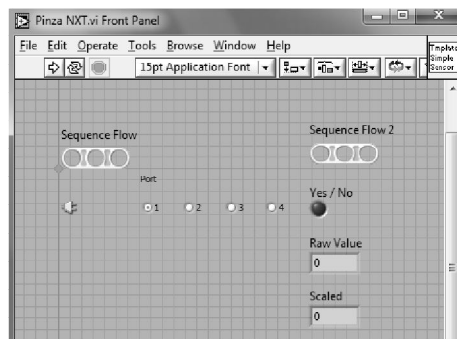
2.3.2.3. **Bloque Pinza NXT.vi**

El programa Pinza NXT.vi es cargado y ejecutado dentro del bloque NXT y contiene los siguientes elementos:

- Sequence Flow
- Sequence Flow 2
- Port
- Variable de tipo booleano llamada Yes/No

- Variable de tipo entero llamada Raw Value
- Variable de tipo entero sin signo llamada Scale

Figura 46. **Panel frontal: programa Pinza NXT.vi**



Fuente: elaboración propia.

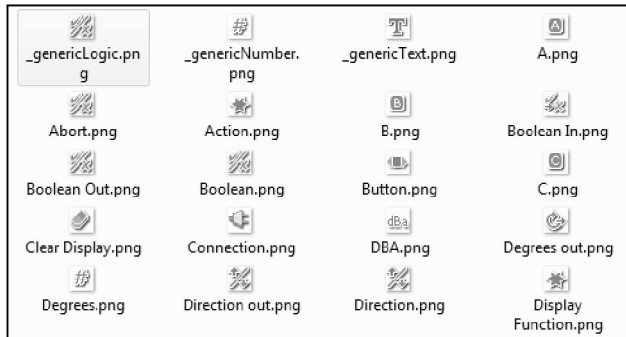
El programa Pinza NXT.vi puede ser cambiado por el programador a su conveniencia, por ejemplo: se puede agregar y/o quitar elemento de entrada y salida, cambiar el ícono y sus conexiones, cambiar el nombre de los elemento, etc.

Es importante saber que cada elemento de entrada y salida es asociado a un ícono. La imagen del ícono que se asocia es a base del nombre que éste tiene, el ícono es pequeño y aparecerá en la parte inferior del bloque programa cuando se active la visualización de sus elementos de entrada y salida en el programa NXT-G. Por ejemplo, si el nombre de la variable es igual al nombre del archivo de uno de los íconos que se encuentran en el directorio:

\\LEGOSoftware\LEGOMINDSTORMSNXT\engine\editorVIs\resources\bloc kImages\drawerImages. El editor de NXT-G lo mostrará y de no ser así, mostrará un ícono genérico basado en el tipo de variable. Ejemplo: la variable

booleano, tienen un ícono asociado por su nombre, el cual se muestra en la figura 47.

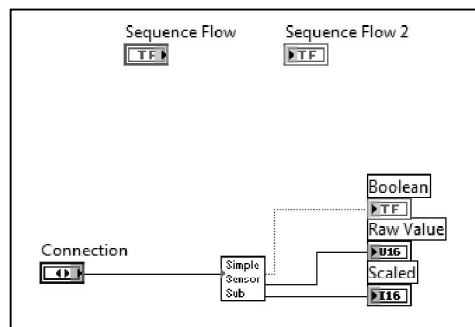
Figura 47. **Carpeta Drawerimages**



Fuente: elaboración propia.

El programa Pinza NXT.vi en el diagrama de bloques muestra el elemento Simple Sensor Sub el cual es mostrado en la figura 48, este es el nombre del ícono y no el nombre del programa al cual hace referencia. El programa realmente llama al programa Pinza NXTSub.vi.

Figura 48. **Diagrama de bloques: programa Pinza NXT.vi**



Fuente: elaboración propia.

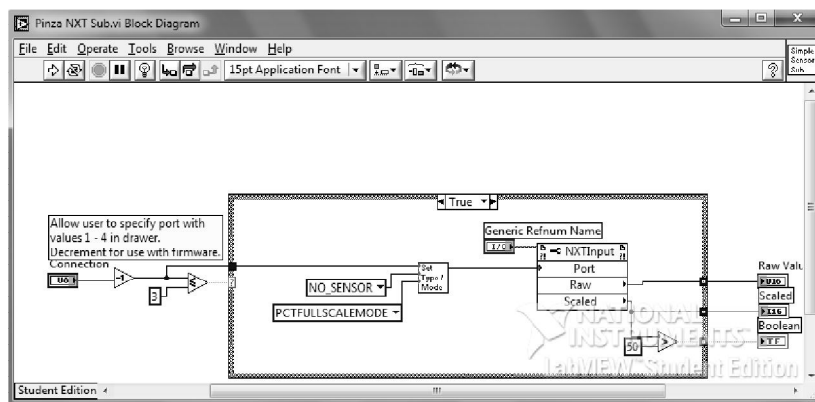
Es importante mencionar que el software de NXT-G solo soporta los siguientes tipos de datos: numéricos, booleanos y de cadena.

2.3.2.4. Bloque Pinza NXTSub.vi

En el panel frontal del programa sub.vi muestra los controles de entrada y salida, los cuales son los mismos controles vistos en el programa principal Pinza NXT.vi.

Si bien se observa el diagrama de bloques, Wizard automáticamente genera un pequeño, pero importante programa, el cual se compone de la programación de la entrada del puerto y hace una llamada a otro VI el cual es proporcionado por el Toolkit, el cual se utiliza para cambiar el tipo y modo del sensor. Además combina la función Property Node para cambiar los valores, así como la lógica y espera hasta que el cambio haya sido reconocido por el sistema operativo del bloque NXT.

Figura 49. Diagrama de bloques: programa Pinza NXTSub.vi



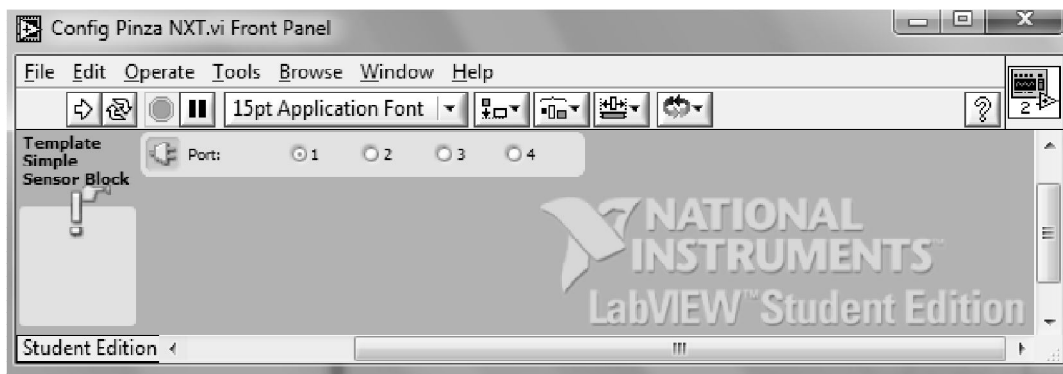
Fuente: elaboración propia.

Por supuesto, el programa generado por Wizard puede ser modificado a la conveniencia del programador e inclusive utilizar algunos elementos ya creados, por ejemplo, el bloque Set Tip/Mode.

2.3.2.5. Configuración vi

Wizard crea el VI de configuración, el cual es utilizado para la presentación visual del bloque programa en el panel de configuración en el editor de NXT-G. El programa puede ser modificado por el programador, es importante saber que el nombre de las variables asociadas al programa Pinza NXTSub.vi deben ser los mismos en el programa configuración, para que cada cambio realizado en el editor de NXT-G tenga efecto al programa principal.

Figura 50. **Config Pinza NXT.vi**



Fuente: elaboración propia.

Es importante mencionar que solo los objetos en la esquina superior izquierda del panel frontal aparecerán en el editor de NXT-G.

2.3.2.6. Otros archivos de soporte

Los siguientes archivos son creados:

- True y False con extensión .png: son usados por los bloques para indicar el estado del sensor, estos archivos no serán utilizados, pero tampoco se eliminarán.
- Pinza NXT.txt y Version.txt: contiene el nombre y versión del bloque programa, y dentro de estos archivos se encuentra el nombre Pinza NXT y la versión 1.0.
- Drawers.dat: es un archivo opcional que controla el orden de las variables que son desplegados al utilizar el bloque en la parte inferior creado dentro del editor de NXT-G.
- Sensor.ini: archivo necesario si se desea que la interfaz del bloque creado trabaje con un Loop/Wait For/Switch en el editor NXT-G.

2.3.2.7. Importar y exportar

Para importar los bloques creados dentro del software MINDSTORMS, es necesario instalar el archivo Dynamic Block Update, el cual puede ser descargado en la siguiente página:

<http://community.LEGOeducation.us/media/p/1482.aspx>. Fecha de última consulta: octubre de 2011.

Pasos para importar el nuevo bloque:

- Iniciar el software NXT-G y crear un nuevo programa.
- En la barra de herramientas seleccionar Tools >> Block Import And Export Wizard.
- Seleccionar Browse y la carpeta donde se encuentra el nuevo bloque.
- Seleccionar el bloque que se desea importar, si éste no se puede seleccionar en el cuadro de lista de importación; posiblemente una parte del bloque está generando un error y por ello no es posible importarlo, o bien la carpeta del bloque a importar no contiene todos los archivos necesarios o el nombre de los archivos no son los adecuados, el programador deberá verificar cada uno de estos detalles.
- Se elige la paleta a la que se desea agregar el bloque.
- Para finalizar se debe seleccionar: importar.

Las imágenes asociadas al nuevo bloque no aparecerán hasta que se reinicie el software NXT-G, luego de ser importado.

3. DISEÑO E IMPLEMENTACIÓN DEL ACTUADOR PINZA INTELIGENTE

3.1. Diseño de hardware

En el diseño de hardware se trabaja en el sistema del microcontrolador, el cual se encarga de recibir las señales del bus I²C, almacenar información y generar la señal PWM para mover el servomotor, mientras que el servomotor y pinza deben responder a este sistema.

Figura 51. **Actuador pinza inteligente**



Fuente: elaboración propia, con el apoyo del material adquirido.

En la figura 51 se muestra la arquitectura física que tendrá el nuevo actuador. Su característica física se presentan en la tabla XI.

Tabla XI. **Características físicas del actuador pinza inteligente**

Dimensiones de caja	Largo	9,5 cm
	Ancho	10,5 cm
	Alto	5,5
Peso		0,30 Lb

Fuente: elaboración propia.

3.1.1. Descripción de los componentes empleados

En esta sección se describirán las conexiones realizadas de los diferentes componentes físicos de hardware utilizados, para lograr el funcionamiento del actuador pinza inteligente.

3.1.1.1. Microcontrolador PIC16F877A

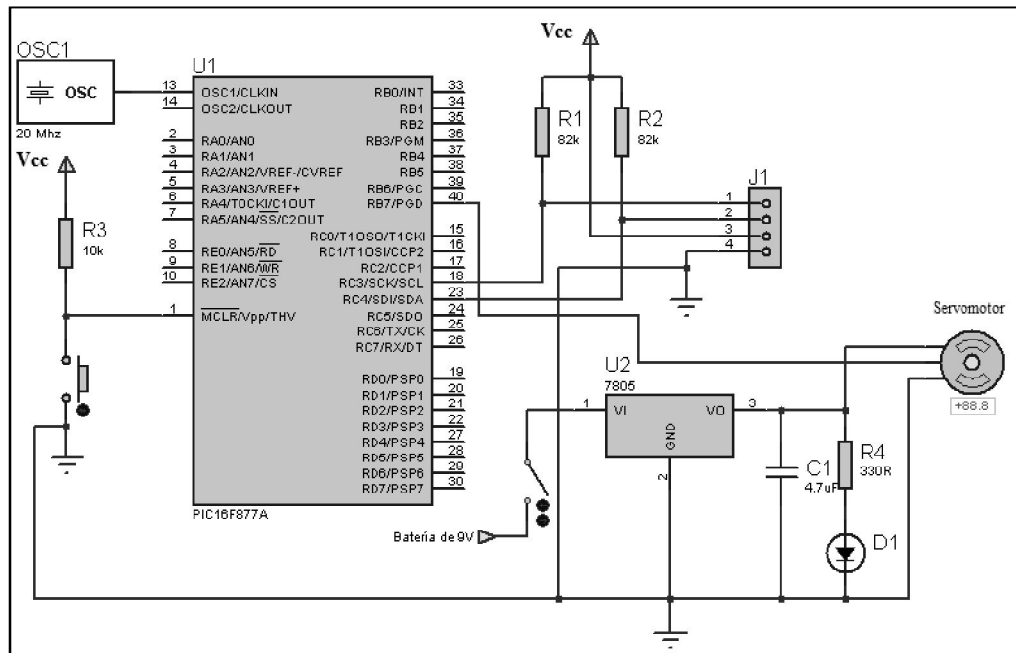
El hardware del sistema del microcontrolador contempla los siguientes elementos:

- Bus I²C: se utilizan dos resistencias de 82K Ω y son conectadas como resistencias *pull-up*.
- Señal PWM: la cual es generada en el pin RB7, debe ser conectada a la señal de entrada del servomotor.
- Circuito de *reset*: utiliza una resistencia de 10 K Ω y está conectada entre un *push botton* y Vcc. Esto permite la inicialización del sistema al momento de realizar pruebas.

- Generador de frecuencia: se utiliza un cristal de cuarzo que genera una frecuencia de trabajo de 20 MHz.
- Regulador de voltaje: señal que alimentará al servomotor.

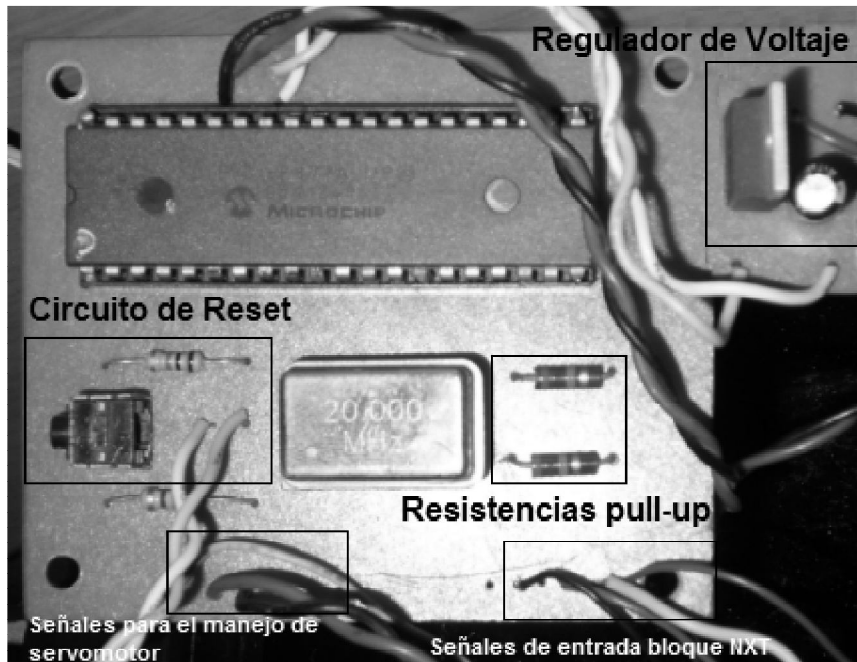
En la figura 52 se muestran las conexiones del microcontrolador PIC16F877A con sus elementos y en la figura 53 se muestra el circuito integrado impreso.

Figura 52. Diagrama circuital del actuador pinza inteligente



Fuente: elaboración propia.

Figura 53. **Circuito integrado impreso**



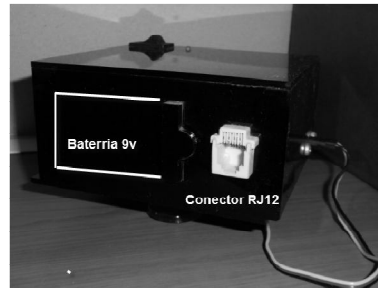
Fuente: elaboración propia.

3.1.1.2. **Bus I²C**

El circuito del bus está compuesto por las señales del puerto C del microcontrolador pin 18 y 23, los cuales proporcionan las señales SCL y SDA respectivamente, estas señales están conectadas a las resistencias *pull-up* R1 y R2 ambas de 82 K Ω (valor sugerido en especificaciones del bloque NXT, ver capítulo 1).

Para conectar las líneas de entrada del bloque NXT con el nuevo actuador se utiliza un conector RJ12, el cual está internamente conectado con las líneas SCL, SDA, Vcc y GND del microcontrolador. Es importante mencionar que el bloque NXT alimenta al PIC mediante la señal Vcc.

Figura 54. **Parte posterior del actuador pinza inteligente**



Fuente: elaboración propia, con el apoyo del material adquirido.

3.1.1.3. **Servomotor**

El servomotor seleccionado es un Hitec modelo HS-422, cuyas características técnicas son:

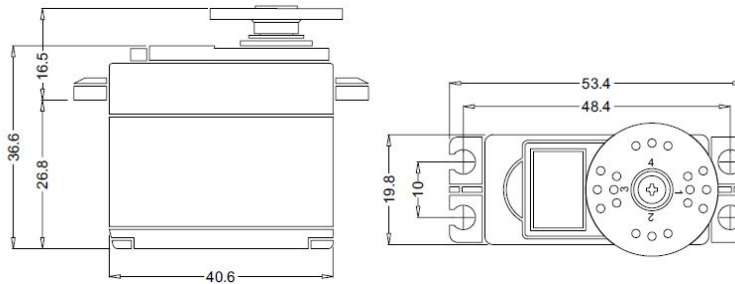
Tabla XII. **Características del servomotor Hitec**

Servo Hitec HS422	
Sistema de control	Control por anchura de pulso. 1,5 ms al centro
Tensión de funcionamiento	4,8V a 6 V
Velocidad a 6V	0,16 Seg /60 grados sin carga
Fuerza a 6V	4,1 Kg · cm
Corriente en reposo	8 mA
Corriente en funcionamiento	150 mA sin carga
Corriente máxima	1100 mA
Zona neutra	8 µsec
Dimensiones	40,6 x 19,8 x 36,6 mm
Peso	45,5 g
Rodamiento principal	Metálico
Engranajes	Plástico
Longitud del cable	300 mm

Fuente: Manual de especificaciones. <http://www.robotshop.com/content/PDF/hs422-31422S.pdf>.

Consulta: enero de 2013.

Figura 55. **Dimensiones del servomotor Hitec**



Fuente: Manual de especificaciones. <http://www.robotshop.com/content/PDF/hs422-31422S.pdf>.

Consulta: enero de 2013.

Terminales del servomotor:

- Vcc,: cable de alimentación, su color es rojo
- GND: cable de masa o negativo, su color es negro
- Señal: cable donde se aplica la señal PWM, su color es amarillo

La señal de los pulsos será generada por el microcontrolador a través del puerto B <7>.

El servomotor requiere un voltaje de 4,8 a 6 voltios pico a pico del pulso de onda cuadrada. La variación del pulso es de 0,9 ms a 2,1 ms donde 1,5 ms ubica al servomotor en el centro. Su frecuencia de trabajo es de 50 Hz.

Tabla XIII. **Relación de ancho de pulso vrs ángulo**

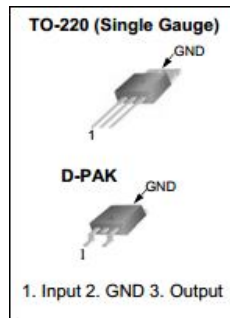
Tiempo en ms	Ángulo en °
0,9	0
1,5	90
2,1	180

Fuente: elaboración propia.

3.1.1.4. **Sistema de alimentación**

Para energizar al servomotor se utiliza una batería de 9 voltios, se regula este voltaje por medio del circuito regulador, donde se utiliza el regulador MC78M05 de 5 voltios.

Figura 56. **Regulador MC78M05 y sus terminales**



Fuente: Manual de especificaciones. <http://www.fairchildsemi.com/ds/MC/MC78M05.pdf>.

Consulta: 15 de marzo de 2012.

Características principales del regulador:

- Corriente de salida 0,5 A
- Voltaje de salida 5V

- Protegido en sobrecarga térmica
- Integrado de 3 terminales

En el circuito de conexión se tiene una batería de 9v, un *swich* de encendido apagado, un regulador y un capacitor utilizado para eliminar ruido. La resistencia R4 y el led D1 del diagrama circuital, se han colocado para tener un indicador visual del encendido/apagado del servomotor.

3.1.1.5. Pinza

Pinza pequeña con una apertura máxima es de 3.3 centímetro. Su diseño le permite la adaptación de servomotores Futuba o Hitec. Eje. HS-322, HS-422, SA-325, SA-422, etc.

Figura 57. **Pinza**



Fuente: <http://www.trossenrobotics.com/store/p/5561-Little-Gripper-Kit-no-servos-.aspx>.

Consulta: enero de 2013.

3.2. Diseño de software

El software del sistema está constituido por dos programas, uno reside en el microcontrolador y el otro reside en el bloque NXT.

En esta sección se comienza a definir la comunicación entre el bloque NXT y el actuador pinza inteligente, para luego poder programar los diferentes módulos del microcontrolador. Posteriormente, se procede a realizar la programación del módulo o bloque programa Pinza NXT en el software de LabVIEW.

La programación del microcontrolador se realiza en el software MPLAB, donde se emplean las siguientes aplicaciones:

- MPLAB editor: permite editar, lincar y simular
- MPASM: permite compilar y generar el fichero de código ensamblado

3.2.1. Protocolo lógico de comunicación I²C: bloque NXT - PIC16F877A

Al emplear la comunicación I²C, se encuentran múltiples maneras de definir e implementar la funcionalidad para leer y escribir datos desde y hacia un dispositivo externo.

LEGO ha caracterizado los dispositivos externos como áreas externas de memoria donde se puede leer o escribir datos. En la figura 58 se presenta el mapa de memoria utilizada por LEGO.

Se observa que LEGO tiene tres grupos de comandos, los cuales son:

- Constates: los ubica en la posición 0h, los datos esperados son información propia del dispositivo externo, valores que no cambian y/o son constantes.

- Variables: inician en la posición 40h, los datos esperados son valores que representan, por ejemplo: mediciones hechas por el dispositivo externo, el valor a recibir es variante.
- Comandos: inician en la posición 80h, este grupo indica al dispositivo externo realizar alguna orden de acción y no espera recibir información de retroalimentación. Si es necesario se puede enviar *bytes* de información.

Figura 58. **Mapa de memoria para un dispositivo digital externo**

Command	Transmitted from NXT			Length
	Byte 0	Byte 1	Byte 2	
		Addr.		
Constants				
Read version	Device address	0x00	R + 0x03	8
Read product ID	Device address	0x08	R + 0x03	8
Read sensor type	Device address	0x10	R + 0x03	8
Read factory zero (Cal 1)	Device address	0x18	R + 0x03	1
Read factory scale factor (Cal 2)	Device address	0x19	R + 0x03	1
Read factory scale divisor	Device address	0x1A	R + 0x03	1
Read measurement units	Device address	0x1B	R + 0x03	7
Variables				
Read variable 1	Device address	0x40	R + 0x03	1
Read variable 2	Device address	0x41		
...				
Commands				
Command 1	Device address	0x80	0xXX	
Command 2	Device address	0x81		
...				

Fuente: The LEGO Group. LEGO® MINDSTORMS® NXT Hardware Developer Kit. p.10.

El formato de comunicación I²C utilizado es continuo y su secuencia es la siguiente:

- *Byte 0*: envío de la dirección del dispositivo más el bit R/W = 0.
- *Byte 1*: envío de comando según mapa de memoria definido por LEGO.
- *Byte 2*: continúa la comunicación I²C enviando el pulso *repeat start* (R) y envía nuevamente la dirección mas el bit R/W =1.

En la tabla XIV se presenta parte del formato utilizado por el sensor ultrasónico de LEGO, con el fin de ejemplificar lo anterior y utilizar este formato para el nuevo actuador pinza inteligente.

El sensor ultrasónico de LEGO tiene asignada la dirección 01h.

- Para el primer grupo, el sensor define los valores a enviar luego de recibir el comando, por ejemplo, si el bloque NXT transmite por el bus I²C la dirección 01h y el comando Versión (valor 00h), el bloque está solicitando al dispositivo externo su versión y esperará la respuesta de 8 *bytes* de longitud.
- El segundo grupo de comandos representan los valores que varían, pues son utilizados para realizar algún tipo de medición, por ejemplo, el bloque NXT envía el valor de comando 40h, el sensor ultrasónico de LEGO atenderá la petición enviando el valor actual del intervalo de medición continua.
- Para el tercer grupo de comandos del sensor ultrasónico a manera de ejemplificar se presentan alguno de ellos en la tabla XV.

Tabla XIV. **Protocolo lógico de comunicación I²C, sensor ultrasónico de LEGO**

Command	Transmitted from NXT			Length	Received in NXT								Comment	
	Byte 0	Byte 1	Byte 2		Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7		Byte 8
	Addr.													
Constants														
Read version	0x02	0x00	R + 0x03	8	0x56	0x31	0x2E	0x30	0x00					V1.0
Read product ID	0x02	0x08	R + 0x03	8	0x4C	0x45	0x47	0x4F	0x00					LEGO
Read sensor type	0x02	0x10	R + 0x03	8	0x53	0x6F	0x6E	0x61	0x72	0x00				Sonar
Read factory zero (Cal 1)	0x02	0x11	R + 0x03	1	0x00									
Read factory scale factor (Cal 2)	0x02	0x12	R + 0x03	1	0x01									
Read factory scale divisor	0x02	0x13	R + 0x03	1	0x0E									
Read measurement units	0x02	0x14	R + 0x03	7	0x31	0x30	0x45	0x2D	0x32	0x6D	0x00			10E-2m
Variables														
Read continuous measurements interval	0x02	0x40	R + 0x03	1	Interval									
Read command state	0x02	0x41	R + 0x03	1	Command state									
Read Measurement Byte 0	0x02	0x42	R + 0x03	1	Result 1									
Read Measurement Byte 1	0x02	0x43	R + 0x03	1	Result 2									
.....														
Commands														
Off Command	0x02	0x41	0x00											
Single shot command	0x02	0x41	0x01											
Continuous measurement command (default)	0x02	0x41	0x02											
.....														

Fuente: The LEGO Group. LEGO® MINDSTORMS® NXT Ultrasonic Sensor I²C Communication Protocol. p. 2.

Tabla XV. Tercer grupo de comandos sensor ultrasónico

Comandos	Transmite el NXT			Observaciones
	Byte 0	Byte 1	Byte 2	
Apagar	0X02	0x41	0X00	
Captura simple	0X02	0x41	0X01	Modo que le permite al sensor tomar medición cada vez que el comando enviado al sensor.
Medición continua	0X02	0x41	0X02	Modo donde el sensor continuamente toma nuevas mediciones
Evento de captura	0X02	0x41	0X03	Modo donde el sensor mide si hay otros sensores ultrasónicos dentro del área.

Fuente: elaboración propia.

3.2.1.1. Comandos para pinza inteligente NXT

Para controlar el nuevo dispositivo se considera lo siguiente:

- Definir su dirección

El rango disponible de direcciones es de 0 a 127 direcciones (7 bits asignados para este propósito), sin embargo, la dirección 01h pertenece al sensor ultrasónico de LEGO, por lo tanto se puede elegir cualquier dirección del rango mencionado exceptuando la dirección 01h. Para el actuador pinza inteligente se define la dirección '1010'b y al agregar el bit R/W = 0 se tiene el valor binario a cargar '00010100' (14 h).

- Definir los grupos de comandos constantes, variables y comandos del nuevo actuador pinza inteligente.

- Primer grupo de comandos: para este grupo de comandos constantes se definen en la tabla XVI. El bloque NXT transmite:
 - ❖ El *byte 0*: contiene la dirección y el bit R/W del protocolo de comunicación I²C, el cual se envía con valor de 0. El valor en binario a enviar es '0010100'= 14 h.
 - ❖ El *byte 1*: contiene el valor de memoria correspondiente al comando solicitado según mapeo de la figura 58, comando que solicita información al actuador pinza inteligente.
 - ❖ El *byte 2*: se genera el pulso de *repeat start* y se envía el *byte 2* el cual contiene la dirección y bit R/W del protocolo de comunicación I²C, el cual se envía con valor de 1. El valor binario es '0010101'= 15 h.

El bloque NXT espera recibir, dependiendo del comando enviado, una determinada longitud de *bytes*.

Tabla XVI. **Primer grupo de comandos para el actuador pinza inteligente**

Comando Constante	Transmite el NXT			Longitud a Recibir (Byte)	Recibe el NXT								Comentario
	Byte 0	Byte 1	Byte 2		Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	
Versión	0x14	0x00	R + 0x15	8	0x56	0x31	0x2E	0x30	0x00	0x00	0x00	0x00	V1,0
ID del producto	0x14	0x08	R + 0x15	8	0X54	0X45	0X53	0X49	0X53	0X2D	0X31	0X32	TESIS-12
Tipo de sensor	0x14	0x10	R + 0x15	8	0X50	0X49	0X4E	0X5A	0X41	0X49	0X4E	0X54	PINZAINIT

Continuación de la tabla XVI.

Factor cero	0x14	0x18	R + 0x15	1	0x00								
Factor de escala													
	0x14	0x19	R + 0x15	1	0x00								
Factor de escala de división	0x14	0x1A	R + 0x15	1	0x00								
Unidad de medida	0x14	0x1B	R + 0x15	7	0x4E	0x2F	0x41	0x70	0x6C	0x69	0x63		N/Aplic

Fuente: elaboración propia.

- Segundo grupo de comandos: para el segundo grupo se definen los siguientes comandos variables, los cuales el actuador pinza inteligente responderá lo siguiente:
 - ❖ Comando de estado: éste comando se define para verificar el estado actual del actuador pinza inteligente, los posibles estados se describen en la tabla XVII.

Tabla XVII. **Comando estado**

Significado	Estado
Operación realizada con éxito	0x00
Estado ocupado	0x01
Error, operación no realizada	0x02

Fuente: elaboración propia.

- ❖ Comando posición actual: este solicita información de la posición actual del actuador pinza inteligente, el resultado que obtendrá será un valor que variará de 0 a 100, es un valor a dimensional donde cero indicará que la pinza está

completamente cerrada y el valor 100 indicará que está completamente abierta.

Tabla XVIII. **Segundo grupo de comandos**

Comandos Variables	Transmite el NXT			Longitud a Recibir (Byte)	Recibe el NXT
	Byte 0	Byte 1	Byte 2		Byte 0
Comando de estado	0x14	0x40	R + 0x15	1	Estado
Posición actual	0x14	0x41	R + 0x15	1	Resultado

Fuente: elaboración propia.

- Tercer grupo de Comandos: para este grupo se definen los siguientes comandos:
 - ❖ Comando posicionar pinza: éste indicará a la pinza que debe abrir o cerrar la pinza a una posición deseada por el usuario, este valor varía de 0 a 100.
 - ❖ Comando inicializa pinza: éste indicará que la pinza debe colocarse en su apertura máxima (totalmente abierta).
 - ❖ La secuencia de los *bytes* a transmitir se muestra en la tabla XIX y el resumen de los comandos definidos se muestran en la tabla XX.

Tabla XIX. Comandos tercer grupo

Comandos	Transmite el NXT			Longitud a recibir (Byte)	Observaciones
	Byte 0	Byte 1	Byte 2		
Posicionar pinza	0x14	0x81	Dato	0	Byte 2 contendrá la posición a la cual se desea abrir la pinza
Inicializar pinza	0x14	0x82		0	Posiciona la pinza a su máxima apertura, totalmente abierta.

Fuente: elaboración propia.

Tabla XX. Comandos para el actuador pinza inteligente

Comandos	Transmite el NXT				Longitud a Recibir (Byte)	Recibe el NXT							Comentarios
	Byte 0	Byte 1	Byte 2			Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	
Constantes													
Versión	0x14	0x00	R + 0x15	8	0x56	0x31	0x2E	0x30	0x00	0x00	0x00	0x00	V1,0
ID del producto	0x14	0x08	R + 0x15	8	0x54	0x45	0x53	0x49	0x53	0x2D	0x31	0x32	TESIS-12
Tipo de sensor	0x14	0x10	R + 0x15	8	0x50	0x49	0x4E	0x5A	0x41	0x49	0x4E	0x54	PINZAINTE
Factor cero	0x14	0x18	R + 0x15	1	0x00								
Factor de escala del elemento	0x14	0x19	R + 0x15	1	0x00								
Factor de escala de división	0x14	0x1A	R + 0x15	1	0x00								
Unidad de medida	0x14	0x1B	R + 0x15	7	0x4E	0x2F	0x41	0x70	0x6C	0x69	0x63		N/Aplic
Variables													
Comando de estado	0x14	0x40	R + 0x15	1	Estado								
Posición actual	0x14	0x41	R + 0x15	1	Resultado								
Comandos													
Posicionar pinza a X	0x14	0x81	Dato										
Inicializar pinza	0x14	0x82											

Fuente: elaboración propia.

3.2.2. Funciones desempeñadas por PIC16F877A

Este elemento es el encargado del manejo del nuevo actuador pinza inteligente, y principalmente realiza las siguientes funciones:

- Atender el bus I²C.
- Establecer la comunicación con el bloque NXT mediante el protocolo lógico de comunicación.
- Envío de pulsos PWM al servomotor.
- Guardar información en memoria.

Todo el programa ensamblador se ha estructurado en funciones y en subrutinas de atención a la interrupción para una mejor organización del código, éste es presentado en el apéndice 2.

La lista de módulos configurados son los siguientes: módulo MSSP, TMR0, TMR1 y EEPROM, los cuales se detallan a continuación.

3.2.2.1. Configuración módulo MSSP

Para configurar el módulo MSSP en modo esclavo I²C, deben ser configurados algunos bits de los registros INTCON, PIE1, PORT C, SSPADD y SSPCON, además se limpiarán los registros SSPSTAT y PIR1 para eliminar información errónea que estos registros puedan contener al inicio de la ejecución del programa.

- Del registro INTCON

Registro que controla las interrupciones provocadas por diferentes registros y eventos. En la tabla XXII se muestran los bits de configuración.

Tabla XXI. **Registro INTCON**

INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
bit 7							bit 0

Fuente: Microchip Technology Inc. PIC16F87XA Data Sheet. p. 24.

Se habilita el bit PEIE, bit de interrupción de periféricos internos del microcontrolador que no controla el registro INTCON. Registro INTCON <6>, configurado en la línea número 341 del programa PIC_NXT, ver apéndice 2.

- Del registro PIE1

Registro que permite o prohíbe las interrupciones provocadas por los periféricos internos del microcontrolador, los cuales no están contemplados en el registro INTCON.

Tabla XXII. **Registro PIE1**

PIE1 REGISTER (ADDRESS 8Ch)								
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	
PSPIE ⁽¹⁾	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	
bit 7								bit 0

Fuente: Microchip Technology Inc. PIC16F87XA Data Sheet. p. 25.

Se habilita el Bit SSPIE, bit de habilitación de interrupción del puerto serie síncrono. Registro PIE1<3>, configurado en la línea número 323 del programa PIC_NXT, ver apéndice 2.

- Del registro TRISC/ PORTC

Las líneas SCL y SDA deben ser configuradas como entradas, para lo cual se escriben unos en los bits correspondientes. Registro TRISC <4:3>, configurado en la línea número 293 del programa PIC_NXT, ver apéndice 2.

Tabla XXIII. **Puerto C**

PORTC FUNCTIONS		
Name	Bit#	Function
RC0/T1OSO/T1CKI	bit 0	Input/output port pin or Timer1 oscillator output/Timer1 clock input.
RC1/T1OSI/CCP2	bit 1	Input/output port pin or Timer1 oscillator input or Capture2 input/ Compare2 output/PWM2 output.
RC2/CCP1	bit 2	Input/output port pin or Capture1 input/Compare1 output/ PWM1 output.
RC3/SCK/SCL	bit 3	RC3 can also be the synchronous serial clock for both SPI and I ² C modes.
RC4/SDI/SDA	bit 4	RC4 can also be the SPI data in (SPI mode) or data I/O (I ² C mode).
RC5/SDO	bit 5	Input/output port pin or Synchronous Serial Port data output.
RC6/TX/CK	bit 6	Input/output port pin or USART asynchronous transmit or synchronous clock.
RC7/RX/DT	bit 7	Input/output port pin or USART asynchronous receive or synchronous data.

Fuente: Microchip Technology Inc. PIC16F87XA Data Sheet. p. 47.

- Del registro SSPADD

Registro donde debe ser ingresada la dirección del dispositivo esclavo, se asigna el valor binario '00010100' (14h). Registro configurado en la línea número 321 del programa PIC_NXT, ver apéndice 2.

- Del registro SSPCON

Registro de control donde se le indica al microcontrolador cómo trabajará el módulo MSSP.

Tabla XXIV. **Registro SSPCON**

SSPCON1: MSSP CONTROL REGISTER 1 (I ² C MODE) (ADDRESS 14h)							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7						bit 0	

Fuente: Microchip Technology Inc. PIC16F87XA Data Sheet. p. 73.

Los siguientes bits son configurados:

Bit SSPEN, bit de habilitación de puerto serie síncrono que permite habilitar las líneas SCL y SDA del puerto C, para que trabajen como puertos serie síncrono.

Bit CKP, bit que activa el reloj en modo esclavo.

Bits SSPM, bits que seleccionan el modo de trabajo del puerto serial síncrono, en la tabla XXVI se presentan los diferentes modos de trabajo. Se

selecciona en modo esclavo I²C, con asignación de 7 bits para la dirección del dispositivo esclavo.

La configuración del registro SSPCON se encuentra configurado en la línea número 318 del programa PIC_NXT, ver apéndice 2.

Tabla XXV. **Estados de los bits SSMPM3:SSPM0**

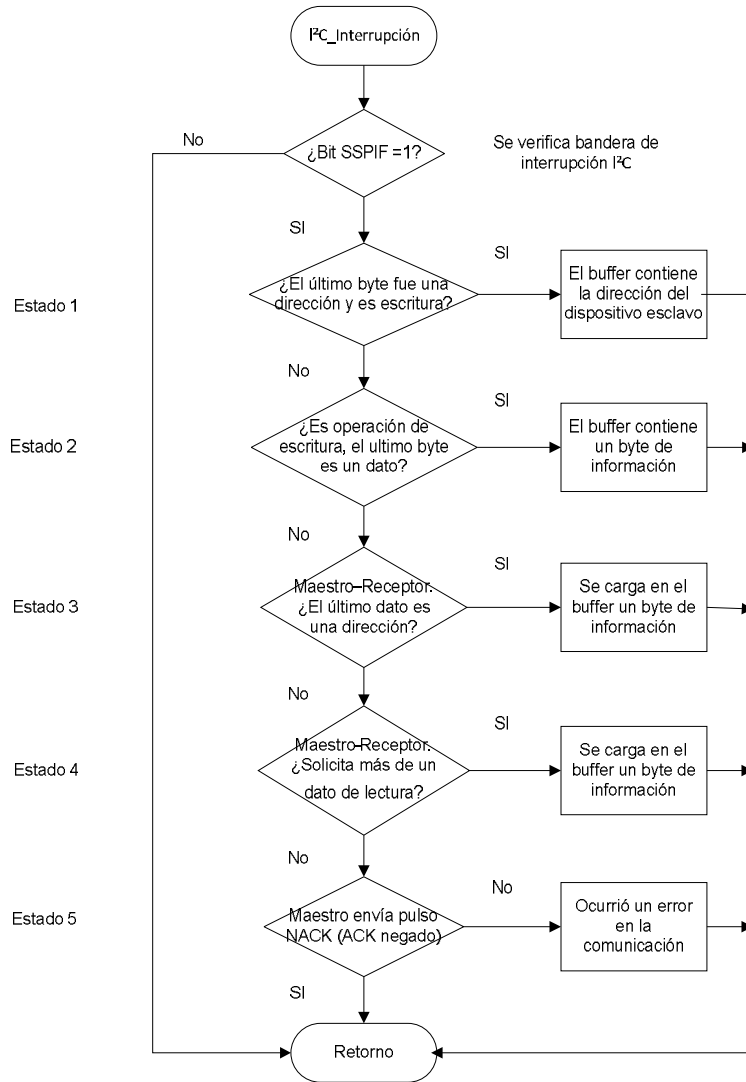
bit 3-0	SSPM3:SSPM0: Synchronous Serial Port Mode Select bits
	1111 = I ² C Slave mode, 10-bit address with Start and Stop bit interrupts enabled
	1110 = I ² C Slave mode, 7-bit address with Start and Stop bit interrupts enabled
	1011 = I ² C Firmware Controlled Master mode (Slave Idle)
	1000 = I ² C Master mode, clock = Fosc/(4 * (SSPADD + 1))
	0111 = I ² C Slave mode, 10-bit address
	0110 = I ² C Slave mode, 7-bit address
	Note: Bit combinations not specifically listed here are either reserved or implemented in SPI mode only.

Fuente: Microchip Technology Inc. PIC16F87XA Data Sheet. p. 73.

3.2.2.2. **Diseño de programación: comunicación I²C**

Luego de configurar los registros para poder utilizar el bus I²C, se presenta el diagrama de flujo de la comunicación en el que se puede observar la secuencia de rutinas que se ejecutarán para la comunicación I²C.

Figura 59. **Diagrama de flujo del programa del microcontrolador módulo MSSP modo I²C esclavo**



Fuente: elaboración propia.

El programa está diseñado para actuar al presentarse algún evento del bus I²C.

- Rutina de interrupción

Rutina que verifica si la interrupción ha sido provocada por el bus I²C, el registro que se verifica es el registro PIR1 bit SSPIF <3>, si este bit está en 1, es porque la interrupción se ha provocado a causa del puerto serial síncrono. Es importante limpiar esta bandera para verificar las futuras interrupciones (líneas 113 y 115, programa PIC_NXT, ver apéndice 2).

Para las siguientes rutinas se hace referencia a las observaciones encontradas en el manual AN734 de microchip:

- Rutina dirección

- Estado 1: Maestro - transmite: el último *byte* fue una dirección y es escritura.

El dispositivo maestro ha empezado la comunicación de escritura inicializando una condición de *start* o *repeat start* en el bus, envía en el *byte* la dirección del dispositivo esclavo. El bit menos significativo del *byte* enviado es 0, lo que indica que el dispositivo maestro desea escribir un dato al esclavo.

Los bits del registro SSPSTAT tendrán los siguientes valores:

Tabla XXVI. **Registro SSPSTAT**

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SSPSTAT	SMP	CKE	D/Ā	P	S	R/W	UA	BF

Fuente: Microchip Technology Inc. PIC16F87XA Data Sheet. p. 81.

S = 1: condición *start* ha ocurrido
R/W = 0: dispositivo maestro envía dato a dispositivo esclavo
D/A = 0: el último *byte* fue una dirección
BF = 1: el búfer está lleno

En esta rutina se verifica la dirección enviada por el bloque NXT, si pertenece al actuador pinza inteligente. Luego de verificar, se limpia el búfer.

La rutina dirección se encuentra programada en la rutina estado1, línea 122, programa PIC_NXT, ver apéndice 2.

- Rutina de escritura
 - Estado 2: maestro - transmite, operación de es escritura, el último *byte* fue un dato.

Luego de haber enviado la dirección (estado 1), el dispositivo maestro puede enviar uno o más *bytes* de datos al esclavo. El registro de estado tendrá los siguientes valores:

S = 1: condición *start* ha ocurrido
R/W = 0: dispositivo maestro envía dato a dispositivo esclavo
D/A = 1: el último *byte* fue un *byte* de dato
BF = 1: el búfer está lleno

En esta rutina, el microcontrolador guarda el dato recibido por el bloque NXT.

La rutina escritura se encuentra programada en la rutina estado2 línea 132, programa PIC_NXT, ver apéndice 2.

- Rutina de lectura
 - Estado 3: maestro - receptor, el último dato es una dirección

El dispositivo maestro inicializa el bus con una condición de *start* o *repeat start* enviando la dirección del dispositivo esclavo e indicándole en el bit menos significativo que es una operación de lectura. El registro de estado reflejará los siguientes valores:

S = 1: condición *start* ha ocurrido

R/W = 1: dispositivo maestro leerá datos del dispositivo esclavo

D/A = 0: el último *byte* fue una dirección

BF = 0: el búfer está vacío

En este estado, el búfer se cargará con el primer *byte* de dato solicitado por el dispositivo maestro y será enviado por el bus.

En esta rutina y en la siguiente, el microcontrolador envía el dato solicitado por el dispositivo maestro.

La rutina lectura se encuentra programada en la rutina estado3 línea 151, programa PIC_NXT, ver apéndice 2.

- Rutina de lectura de datos

El dispositivo Maestro solicita más de un dato de lectura.

- Estado 4: maestro - receptor, el último *byte* es un dato

Este estado ocurre cada vez que el dispositivo maestro, previamente ha leído un *byte* de dato desde el dispositivo esclavo y desea leer otro *byte*. El registro de estado tendrá los siguientes valores:

S = 1: condición *start* ha ocurrido

R/W = 1: dispositivo maestro leerá datos del dispositivo esclavo

D/A = 1: el último *byte* enviado es un *byte* de dato

BF = 0: el búfer está vacío

La rutina lectura de datos se encuentra programada en la rutina estado4 línea 173, programa PIC_NXT, ver apéndice 2.

- Rutina de verificación ACK

- Estado 5: maestro envía pulso NACK (ACK negado)

Este estado ocurre cuando el dispositivo maestro ha enviado el pulso NACK en respuesta que ha recibido el dato desde el esclavo de forma correcta. Esta acción indica que el maestro no desea leer más datos del dispositivo esclavo. El registro de estado tendrá en sus bits los siguientes valores:

S = 1: condición *start* ha ocurrido

R/W = 0: bit R/W es reseteado por la lógica del esclavo

D/A = 1: el último *byte* enviado es un *byte* de dato

BF = 0: el búfer está vacío

Si por el contrario, no se recibe el pulso NACK, el microcontrolador indicará que ha ocurrido algún error en la transmisión, este proceso ayuda al programador a la hora de realizar pruebas.

La rutina de verificación ACK se encuentra programada en la rutina estado5 línea 132, programa PIC_NXT, ver apéndice 2.

- Retorno

El retorno es una rutina propia del PIC, al ser llamada en el programa, automáticamente limpia los bits de la interrupción, carga los valores iniciales de la pila y se retorna al programa donde se generó la interrupción (línea 110, programa PIC_NXT, ver apéndice 2)

3.2.2.3. Generación de la señal PWM

Para poder mover el servomotor es necesario que el microcontrolador genere una señal PWM a una frecuencia de 50 Hz, en un principio se estudio como alternativa utilizar el módulo CCP modo PWM, pero este módulo no trabaja frecuencias por debajo de 1,2 KHz. Los cálculos, además de realizarlos manualmente, también se pueden realizarse en la siguiente página de internet: <http://www.micro-examples.com/public/microex-navig/doc/097-pwm-calculator.html>, página consultada en mayo de 2012; es por ello que se generará la señal con los módulos de TIMER 0 y 1 del microcontrolador, se analizará su comportamiento.

Los valores a recibir desde el bloque NXT son:

- 0 equivale a tener la pinza cerrada
- 100 equivale a tener la pinza completamente abierta

Valores del servomotor obtenidos al realizar una prueba del manejo entre pinza y servomotor son:

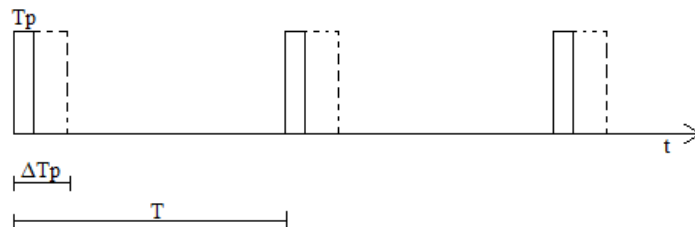
Tabla XXVII. **Manejo del servomotor y pinza**

Angulo (grados)	Tiempo (milisegundos)	Posición
180	2,1	Pinza completamente cerrada
0	0,9	Pinza completamente abierta

Fuente: elaboración propia.

El tiempo del pulso varía de 0,9 a 2,1 milisegundos para que el microcontrolador pueda generar la señal se parametrizará este intervalo de la siguiente forma:

Figura 60. **Secuencia de pulsos PWM**



Fuente: elaboración propia.

Donde:

- T_p : es el tiempo de pulso
- ΔT_p : tiempo el cual el pulso es variable
- T : periodo (1/frecuencia servomotor)

Parametrizando a 1 *byte* los valores de tiempo:

- X (1 *byte*): T_p [0,9:2,1] ms
- 0 → Tiempo máximo del pulso
- 255 → Tiempo mínimo del pulso

La relación es inversamente proporcional, así que se podrá analizar con la ecuación de una recta:

$$T_p = a - b x$$

Si cuando T_p es máximo $x = 0$ se tiene como resultado:

$$T_{pmax} = a$$

Ahora bine si cuando T_p es mínimo $x = 255$, despejando b :

$$T_{min} = a - b * 255$$

$$b = \frac{T_{pmax} - T_{pmin}}{255}$$

El tiempo del pulso sería:

$$T_p = T_{pmax} - \frac{T_{pmax} - T_{pmin}}{255} * x$$

$$T_p = 0,0021 - \frac{0,0021 - 0,0009}{255} * x$$

Esta ecuación se utilizará al realizar el cálculo de los valores que tomará el registro TMR0.

3.2.2.3.1. Configuración módulo TMR0

El módulo TMR0 será el encargado de generar el pulso en alto y el módulo TMR1 se encargará del pulso en bajo.

El módulo TMR0 se emplea como temporizador para determinar intervalos de tiempo concretos. Para calcular el tiempo que le toma incrementar el registro TMR0 se utilizará la siguiente ecuación:

$$\text{Tiempo de incremento} = 4 * \frac{1}{f} * cm$$

Donde:

f : es la frecuencia del oscilador, la señal del reloj es de 20 Mhz

cm: ciclo máquina, unidad básica de tiempo que utiliza el microcontrolador

Se tiene:

$$\text{Tiempo de incremento} = 4 * \frac{1}{20 \text{ Mhz}} * 1 = 0,2 \mu\text{s}$$

El registro TMR0 se incrementa cada 0,02μs.

Para calcular los tiempos del temporizador se tiene la siguiente ecuación:

$$\text{Temporización} = T_{cm} * \text{prescaler} * (256 - \text{carga TMR0})$$

Donde:

- Temporización: es el tiempo del pulso en nivel alto.
- T_{cm} : es el período de un ciclo máquina, este dato fue previamente calculado y es igual a 0,2 μs.
- *Prescaler*: es el rango de divisor de frecuencia elegido.
- (256 – carga TMR0): es el número total de pulsos a contar por el TMR0 antes de desbordarse en la cuenta ascendente. carga TMR0 es el valor cargado inicialmente en el TMR0.

$$\text{Temporización} = 0,2\mu\text{s} * \text{prescaler} * (256 - \text{carga TMR0})$$

Para la selección del *prescaler* se generó todos los valores de cada *prescaler* con ayuda de un programa de cómputo y se obtienen valores positivos utilizando el *prescaler* con la relación 1:64, a continuación se presentan en la siguiente tabla XXVIII.

- L : valor a recibir desde el bloque NXT
- Tp: tiempo del pulso, su rango de valores es [0,9 : 2,1] en milisegundos

Tabla XXVIII. **Rango de asignación para carga TMR0**

		Carga TMR0
L	 Tp (s)	 64
0	0,0009	186
1	0,00091	185
2	0,00092	184
3	0,00094	183
4	0,00095	182
5	0,00096	181
6	0,00097	180
7	0,00098	179
8	0,001	178
9	0,00101	177
10	0,00102	176
11	0,00103	175
12	0,00104	174
13	0,00106	174
14	0,00107	173
15	0,00108	172
16	0,00109	171
17	0,0011	170
18	0,00112	169
19	0,00113	168
20	0,00114	167
21	0,00115	166
22	0,00116	165
23	0,00118	164
24	0,00119	163

		Carga TMR0
L	 Tp (s)	 64
34	0,00131	154
35	0,00132	153
36	0,00133	152
37	0,00134	151
38	0,00136	150
39	0,00137	149
40	0,00138	148
41	0,00139	147
42	0,0014	146
43	0,00142	145
44	0,00143	144
45	0,00144	144
46	0,00145	143
47	0,00146	142
48	0,00148	141
49	0,00149	140
50	0,0015	139
51	0,00151	138
52	0,00152	137
53	0,00154	136
54	0,00155	135
55	0,00156	134
56	0,00157	133
57	0,00158	132
58	0,0016	131

		Carga TMR0
L	 Tp (s)	 64
68	0,00172	122
69	0,00173	121
70	0,00174	120
71	0,00175	119
72	0,00176	118
73	0,00178	117
74	0,00179	116
75	0,0018	115
76	0,00181	114
77	0,00182	114
78	0,00184	113
79	0,00185	112
80	0,00186	111
81	0,00187	110
82	0,00188	109
83	0,0019	108
84	0,00191	107
85	0,00192	106
86	0,00193	105
87	0,00194	104
88	0,00196	103
89	0,00197	102
90	0,00198	101
91	0,00199	100
92	0,002	99

Continuación de la tabla XXVIII.

25	0,0012	162	59	0,00161	130	93	0,00202	99
26	0,00121	161	60	0,00162	129	94	0,00203	98
27	0,00122	160	61	0,00163	129	95	0,00204	97
28	0,00124	159	62	0,00164	128	96	0,00205	96
29	0,00125	159	63	0,00166	127	97	0,00206	95
30	0,00126	158	64	0,00167	126	98	0,00208	94
31	0,00127	157	65	0,00168	125	99	0,00209	93
32	0,00128	156	66	0,00169	124	100	0,0021	92
33	0,0013	155	67	0,0017	123			

Fuente: elaboración propia.

La configuración para TMR0 es la siguiente:

- Del registro INTCON

Bit GIE: bandera de habilitación global del permiso de interrupción, se borra automáticamente cuando se reconoce una interrupción, la cual evita que se produzca otra interrupción mientras se está atendiendo a la primera. Al retornar de la interrupción con la instrucción `retfie`, el bit GIE se vuelve a activar poniéndose a 1.

Bit TMR0IE: autoriza la interrupción por desbordamiento del TMR0

Bit TMR0IF : bandera de interrupción del TMR0, indica que se ha producido un desbordamiento del TMR0, que ha pasado de `b'11111111'` a `b'00000000'`, este bit es un indicador, al activarse ocurre una interrupción y debe ser limpiada (rutina Estado_TMR0 línea 263, programa PIC_NXT, ver apéndice 2).

La configuración del registro INTCON se encuentra en la línea número 341 del programa PIC_NXT, ver apéndice 2.

- Del registro OPTION_REG

Registro que gobierna el comportamiento del TMR0. Los bits utilizados por el TMR0 son:

Tabla XXIX. **Registro OPTION_REG**

OPTION_REG REGISTER							
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

Fuente: Microchip Technology Inc. PIC16F87XA Data Sheet. p. 54.

Bits PS2:PS0: seleccionan el rango con el que actúa el divisor de frecuencia, la tabla XXX muestra las posibles configuraciones, en este caso es b '101' *prescaler* 1:64.

Bit PSA asigna el divisor de frecuencia a utilizar.

Bit T0CS selecciona la fuente de la señal de entrada del TMR0 a utilizar.

La configuración de los bits de este registro se muestra en la rutina configuración línea 297, programa PIC_NXT, ver apéndice 2.

Tabla XXX. **Rango de operación del divisor de frecuencia**

PS2:PS0: Prescaler Rate Select bits		
Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Fuente: Microchip Technology Inc. PIC16F87XA Data Sheet. p. 54.

3.2.2.3.2. Configuración módulo TMR1

TMR1 es el encargado de generar el pulso en bajo y la frecuencia de trabajo es de 50 Hz, por consiguiente la ecuación para el cálculo de temporización de TMR1, es la siguiente:

$$\text{Temporización} = 0,2\mu\text{s} * \text{prescaler} * (65\ 536 - \text{carga TMR1})$$

La ecuación utilizada es la misma que se utilizó para el cálculo de TMR0 solo que el valor 256 se ha remplazado por el valor 65 536, el módulo de TMR1 tiene 2 registros TMR1H y TMR1L de 8 bits cada uno.

El valor a cargar a TMR1 es:

$$\text{Carga TMR1} = 65\ 536 - \frac{\text{temporización}}{(0,2\mu * \text{prescaler})}$$

- Del registro T1CON

Tabla XXXI. **Registro T1CON**

T1CON: TIMER1 CONTROL REGISTER (ADDRESS 10h)							
U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{T1SYNC}$	TMR1CS	TMR1ON
bit 7						bit 0	

Fuente: Microchip Technology Inc. PIC16F87XA Data Sheet. p. 57.

Bit TMR1ON: inicia el conteo del temporizador, este bit se habilita de forma conveniente en la ejecución del programa. Se configura en la rutina inicialización_pinza, línea 337, programa PIC_NXT, ver apéndice 2.

Bit TMR1CS: selecciona la fuente de señal de entrada del TMR1.

Bits T1CKPS1:T1CKPS0: seleccionan el rango con el que actúa el divisor de frecuencia, en la tabla XXXIII muestra las posibles configuraciones.

Tabla XXXII. **Carga TMR1**

T1CKPS1	T1CKPS0	Prescaler	Temporización	Carga TMR1
0	0	1:1	0,02	-34464
0	1	1:2	0,02	15536
1	0	1:4	0,02	40536
1	1	1:8	0,02	53036

Fuente: elaboración propia.

Se puede elegir los *prescaler* 2, 4 y 8 para la generación del periodo de frecuencia de la señal PWM, se elige el *prescaler* 1:8.

El tiempo de temporización para el registro TMR1 será de 1/50 Hz, el valor a cargar es 53 036 (valor decimal).

La configuración de este registro se encuentra en la rutina inicialización_pinza, línea 332, programa PIC_NXT, ver apéndice 2.

3.2.2.4. Memoria EEPROM

El propósito de utilizar la memoria EEPROM del microcontrolador es para almacenar los valores de la tabla XXIX que son los valores a cargar en TMR0, los cuales se utilizan para generar el ancho de pulso adecuado.

- **Escritura**

Para cargar estos valores en la memoria EEPROM se utiliza la directiva DE. Al utilizarla se debe fijar el origen en 2100h, para su uso con grabadores. El programa de MPLAB permite su uso (línea 28, programa PIC_NXT, ver apéndice 2).

A continuación se muestra un ejemplo de cómo utilizar la directiva DE, la rutina guarda los datos automáticamente en la memoria EEPROM.

- **ORG 0x2100:** se fija la posición de origen. El dato a grabar se posiciona en la dirección de memoria 0x00h.
- **DE "V1.0",0.1:** si se desea guardar alguna palabra, ésta debe ser encerrada entre comillas, para este ejemplo se colocó la palabra V1.0 y adicional se guarda el valor decimal 1.

- Lectura

Para tener acceso a los datos guardados en memoria se utilizan los registros siguientes:

- Registro EEDATA: contiene el *byte* de dato que se ha leído de la memoria EEPROM.
- Registro EEADR: contiene el *byte* de dirección de la memoria EEPROM a la que se desea acceder para leer o escribir un dato.
- Registro EECON1: es de control de lectura y escritura, los bits de este definen el modo de funcionamiento de la memoria EEPROM. Los bits RD y WR indican respectivamente lectura o escritura. No hay que ponerlos con el valor de 0 sólo a 1, se borran automáticamente cuando la operación de lectura o escritura ha sido completada. El bit EEPGD debe ser puesto a 0 para indicar al microcontrolador que se desea seleccionar la memoria EEPROM para tener acceso de lectura o escritura.

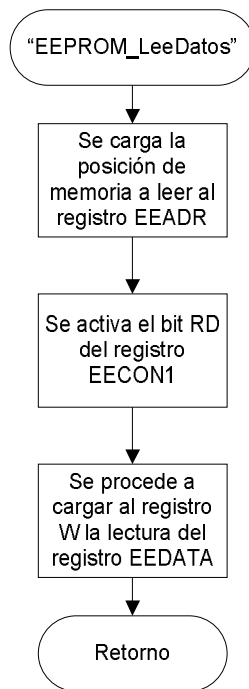
Tabla XXXIII. **Registro EECON1**

EECON1 REGISTER (ADDRESS 18Ch)							
R/W-x	U-0	U-0	U-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	—	—	—	WRERR	WREN	WR	RD
bit 7				bit 0			

Fuente: Microchip Technology Inc. PIC16F87XA Data Sheet. p. 34.

Se muestra en la figura 61 el diagrama de flujo de la rutina de acceso a los datos guardados en la memoria EEPROM.

Figura 61. **Diagrama de flujo rutina EEPROM_LeeDatos**



Fuente: elaboración propia.

La configuración de estos registros se muestra en la rutina EEPROM_LeeDatos línea 347, programa PIC_NXT, ver apéndice 2.

3.2.2.5. Diseño de programación

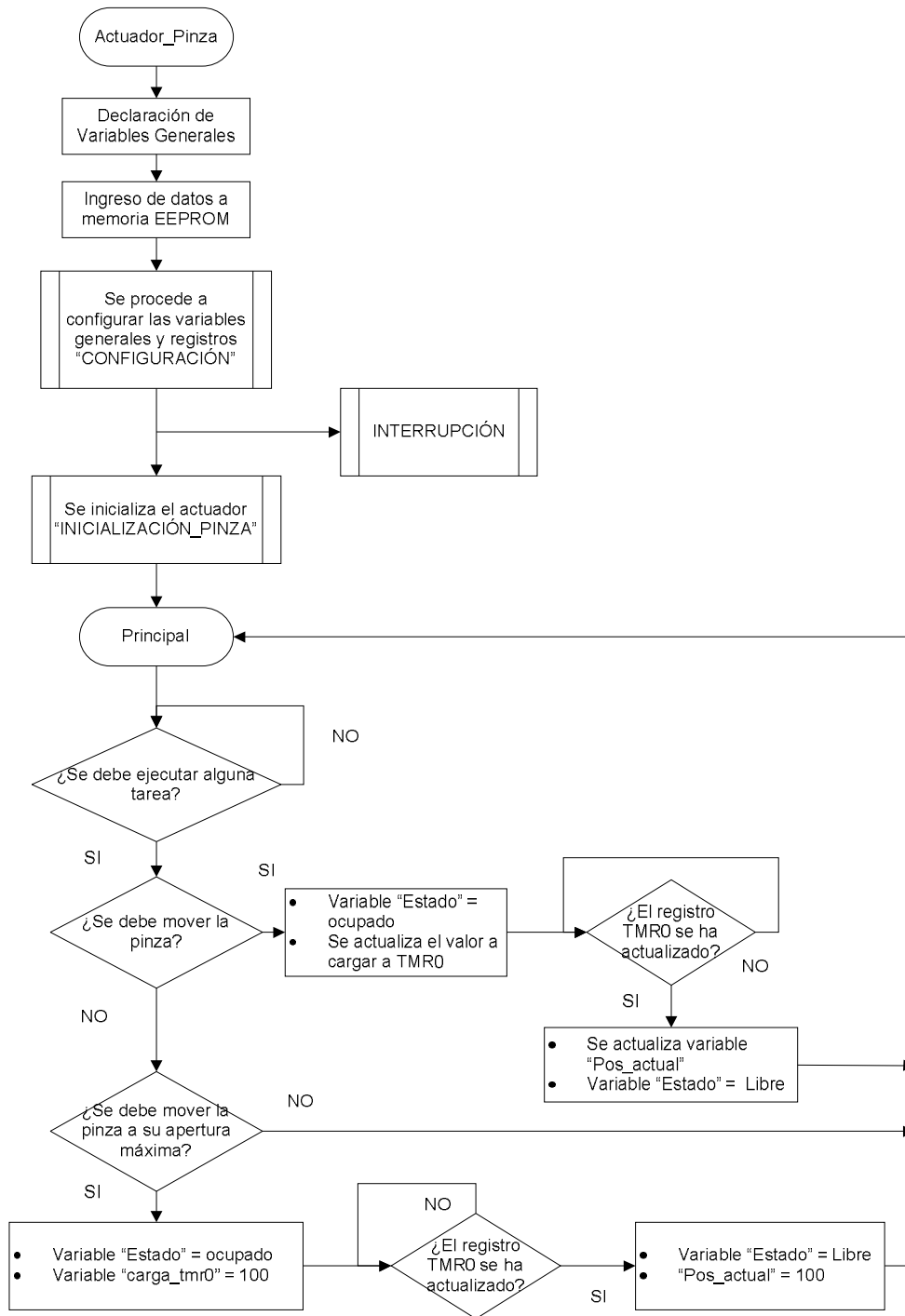
Para el diseño del programa se toma como base el protocolo lógico de comunicación I²C, el cual fue estructurado en secciones anteriores.

Para ello se toma la consideración de que hay dos tipos de datos principales a recibir desde el bloque NXT:

- El *byte*: que contiene el comando: el comando está representado por un número hexadecimal y está seccionado por grupos de rangos de memoria, estos grupos se definen en el programa como solicitud de información que es constante (ej. Versión), comandos constantes; solicitud de información que es variable (ej. Posición actual), comandos variables y solicitud de una acción la cual se tiene como un comando (ej. Inicializar pinza), comando.
- El *byte* dato: *byte* que contiene el valor al cual se debe de posicionar la pinza, este varía de 0 a 100.

A continuación se presenta el diagrama de flujo general del programa, en donde se pudo observar la secuencia de tareas y subrutinas que se ejecutarán, las cuales se describirán de forma separada.

Figura 62. Diagrama de flujo general



Fuente: elaboración propia.

El código del programa se encuentra en el apéndice 2 y se llama PIC_NXT, en las siguientes secciones se hace referencia a éste.

3.2.2.5.1. Declaración de variables

Parte del programa donde se declaran las variables globales a utilizar, las principales se detallan a continuación mostrando su posición de memoria:

- flag_vc (29h): variable utilizada como bandera, los bits 7,6 y 0 son utilizados para determinar acciones mediante su activación en el desarrollo de cada proceso del programa.
- Byte_1 (2fh): variable utilizada para guardar de forma temporal el *byte* de comando o dato recibido desde el bloque NXT.
- Temp (22h): variable utilizada para realizar comparaciones al entrar en la interrupción provocada por el módulo MSSP modo I²C
- Carga_tmr0 (2e): variable utilizada para guardar el valor del *byte* de dato, la cual tendrá de forma indirecta el valor a cargar al registro TMR0.
- Var_comando (28h): variable utilizada para guardar el *byte* que contiene el comando.
- Pos_pinza (2dh): variable utilizada para guardar el valor de la posición deseada de apertura de la pinza.

- Estado (40h): contiene el valor del estado actual de la pinza, esta variable se ha colocado convenientemente en la dirección de memoria 40h.
- Pos_actual (41h): contiene el valor de la posición actual de apertura de la pinza, esta variable se ha colocado convenientemente en la dirección de memoria 41h.

La declaración de estas variables se encuentra a partir de la línea 5 del programa.

3.2.2.5.2. Ingreso de datos a memoria EEPROM

Proceso mediante el cual se procede a ingresar los valores constantes que el bloque NXT solicitará por medio de su comando constante. El primer dato que se ingresa se mapea en la posición de memoria 0x00.

ADDR	Valor
00h	V1.0
08h	TESIS-12
10h	PINZAINIT
1Bh	N/Apric

También se ingresan los valores de TMR0 en la posición de memoria 100(2164h), ver código del programa línea 30.

3.2.2.5.3. Configuración: módulos y variables

En esta sección del programa se configuran los módulos MSSP, TMR0, TMR1 y puertos de entrada y salida. La rutina en el código del programa PIC_NXT se llama configuración y se encuentra en la línea 289.

Se limpian registros y variables globales:

- Registros: INTCON, PORTB, PORTC, PIR1, TMR1H, TMR1L, SSPSTAT.
- Variables: flag, byte_1, temp, data_ee_addr, data_ee_dato, var_comando, flag_vc, pos_pinza, estado, pos_actual.

A continuación se presenta la configuración de los registros y el número de línea del código de programa.

Línea 293 b '00011000' → PORTC

Línea 295 b '00000000' → PORTB

Se configuran los módulos TMR0 y TMR1

Línea 297 b '00000101' → OPTION_REG

Línea 324 b '00001001' → PIE1

Línea 310 b '01100100' → carga_tmr0

Se configura el módulo MSSP

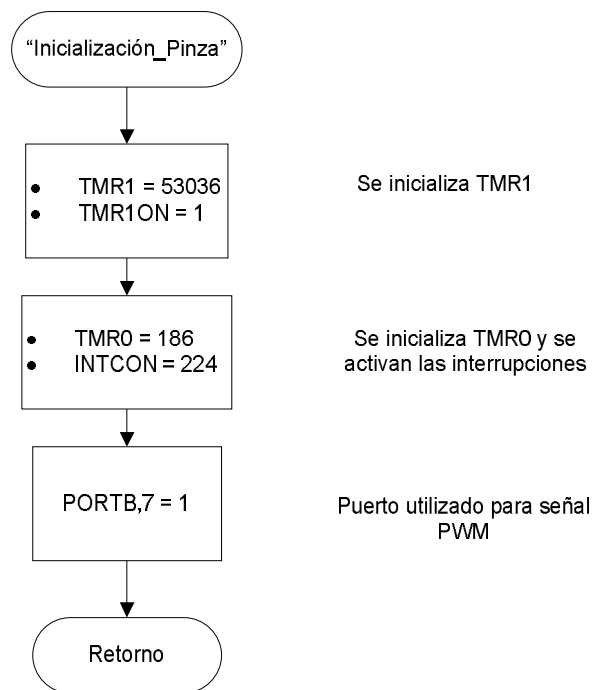
Línea 318 b '00110110' → SSPCON

Línea 321 b '00010100' → SSPADD

3.2.2.5.4. Inicialización_pinza

Proceso encargado de colocar la pinza completamente abierta. A continuación se presenta el diagrama de flujo.

Figura 63. Diagrama de flujo inicialización_pinza



Fuente: elaboración propia.

Esta rutina activa los temporizadores TMR1 y TMR0, para generar la señal PWM con un pulso de 0,9 ms y frecuencia de 50 Hz.

Además se activan las interrupciones por medio del registro INTCON bits GIE, PEIE y TMR0IE. Se procede a colocar el puerto B bit 7 a un valor igual a 1, pin donde se generará la señal PWM.

Esta rutina se encuentra la línea 329 del programa PIC_NXT.

- Principal

Rutina principal en la cual se verifica si en algún momento se recibió un comando para realizar alguna acción. Las dos acciones posibles son:

- Mover la pinza a X apertura
- Inicializar pinza

Se utiliza para la comparación la variable flag_vc,6 y para verificar que acción se debe realizar la variable var_comando. La rutina se encuentra en la línea 50 del código del programa PIC_NXT.

3.2.2.5.5. Interrupción

En esta rutina, en primer lugar se procede a guardar el valor inicial de los registros W, STATUS, PCLATH y FSR, esto con el objetivo de no perder la información que estos registros contienen en el desarrollo del programa principal.

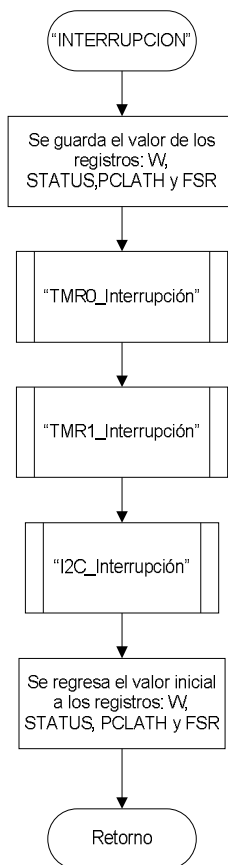
Luego se verifica la causa de la interrupción por medio de las rutinas:

- TMR0_interrupción
- TMR1_interrupción

- I²C_interrupción

El diagrama de flujo se presenta en la figura 64 y la rutina se encuentra en la línea 89 del código del programa PIC_NXT.

Figura 64. **Diagrama de flujo de la rutina interrupción**

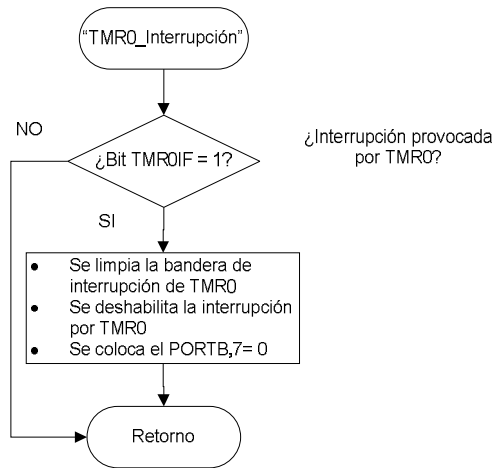


Fuente: elaboración propia.

- TMR0_interrupción

Rutina que se encarga en verificar si la interrupción fue provocada por el desbordamiento del registro TMR0, si el registro INTCON bit TMR0IF fue puesto a uno, se procede a limpiar el bit del registro INTCON: TMR0IF y se deshabilita la interrupción de TMR0 por medio del bit TMR0IE; luego se procede a poner el bit 7 del puerto B en 0, el cual es el pulso en bajo de la señal PWM. El diagrama de flujo se presenta en la figura 65 y la rutina se encuentra en la línea 263 del código del programa PIC_NXT.

Figura 65. **Diagrama de flujo TMR0_interrupción**



Fuente: elaboración propia.

- TMR1_interrupción

Subrutina encargada de verificar si la interrupción fue provocada por el desbordamiento del par de registros de TMR1, de ser afirmativo se procede a restablecer e iniciar nuevamente el temporizador.

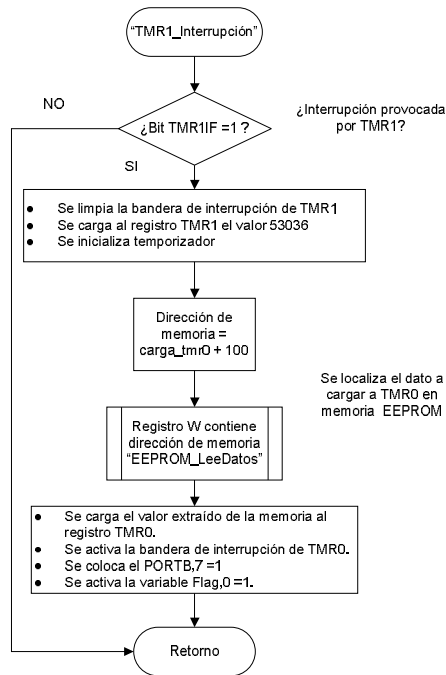
Para cargar el valor correcto del registro TMR0, se utiliza la variable carga_tmr0, quien tendrá el valor de la posición al cual debe moverse la pinza (rango de variación 0 a 100); el valor a cargar al registro TMR0 se encuentra en memoria EEPROM (rango de dirección 100 a 200), por consiguiente la dirección de memoria que contiene el valor a cargar se obtiene con el resultado de la suma del valor de la variable carga_tmr0 + 100.

La dirección de memoria se carga al registro W y seguido de esto se llama a la rutina EEPROM_LeeDatos, rutina que fue ampliamente explicada en la sección 3.2.2.4; el valor devuelto por la rutina es el valor de carga del registro TMR0.

Por último se activan el bit TMR0IE para iniciar el conteo de TMR0 y se coloca en 1 el bit 7 del puerto B, para iniciar nuevamente el pulso en alto de la señal PWM.

El código de esta rutina se encuentra en la línea 263 del programa PIC_NXT.

Figura 66. Diagrama de flujo TMR1_interrupción

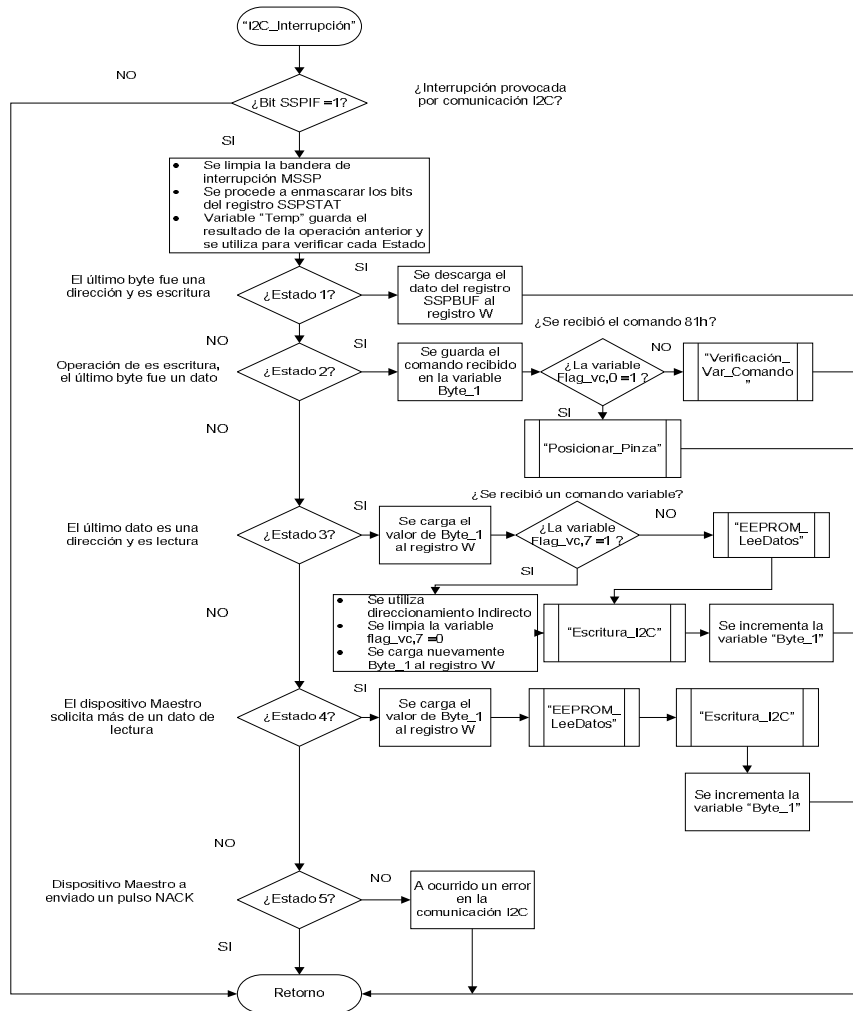


Fuente: elaboración propia.

○ I²C_interrupción

La rutina se encarga de recibir y enviar la información al dispositivo maestro por medio de la comunicación I²C utilizando el protocolo ya definido. Se presenta en la figura 67 el diagrama de flujo de la rutina I²C_interrupción.

Figura 67. Diagrama de flujo de rutina I²C_interrupción



Fuente: elaboración propia.

La rutina comprende los siguientes pasos:

Se verifica si la interrupción fue provocada por una comunicación I²C, de ser afirmativo el primer paso, se procede a verificar cada estado.

Si se recibió un comando constate, se procede a enviar la información requerida accedendo a la memoria EEPROM, para esta acción se utiliza el direccionamiento indirecto.

Si se recibió un comando variable, se accesa a la memoria de las variables globales estado o pos_actual del microcontrolador y se procede a enviar el dato ahí almacenado. El valor hexadecimal del comando variable es igual a la dirección de memoria de las variables mencionadas, las cuales son actualizadas en la rutina principal.

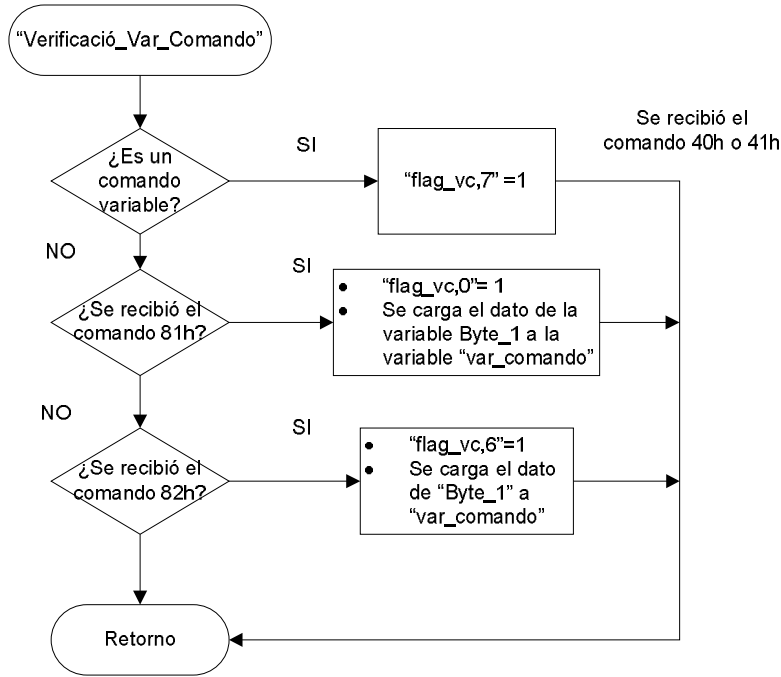
Si se recibió un comando, se guarda la información recibida, se activa la variable flag_vc,6 y se procede a realizar lo solicitado en la rutina principal.

La rutina se muestra en la línea 112 del código del programa PIC_NXT. A continuación se describen las subrutinas utilizadas en la interrupción I²C.

❖ Verificación_var_comando

Rutina utilizada para verificar si el comando recibido es un comando variable o comando para realizar una acción. La variable utilizada para indicar a los demás procesos sobre esta información es la variable flag_vc mediante la activación de determinado bit. Se almacena el *byte* comando recibido en la variable var_comando. La rutina se muestra en la línea 214 del código del programa PIC_NXT.

Figura 68. Diagrama de flujo de la rutina verificación_var_comando



Fuente: elaboración propia.

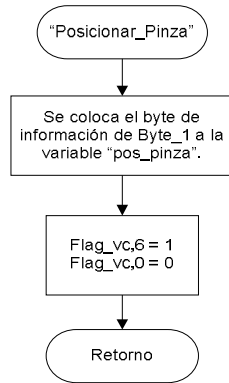
❖ Posicionar_pinza

Rutina que actualiza la variable pos_pinza, mediante la acción de cargarle la información que contiene la variable Byte_1. La bandera flag_vc bits 6 y 0 son puestos a 1 y 0 respectivamente.

En esta rutina solo se actualiza la información de las variables, ya que serán llaves en otra parte del programa.

La rutina se muestra en la línea 254 del programa PIC_NXT.

Figura 69. **Flujo grama de rutina posicionar_pinza**



Fuente: elaboración propia.

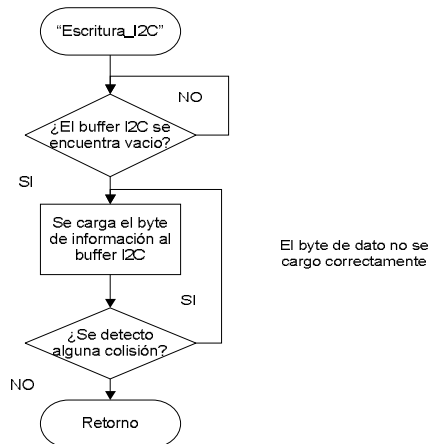
❖ Escritura_I²C

Esta rutina se utiliza para cargar información al búfer utilizado para enviar información por el bus I²C.

Se asume que el registro W antes de entrar a esta rutina tiene cargado el dato de información a enviar.

La rutina se muestra en la línea 201 del código del programa PIC_NXT.

Figura 70. Diagrama de flujo rutina escritura_I2C



Fuente: elaboración propia.

3.3. Diseño del módulo: bloque Pinza NXT

En el capítulo 2 se desarrolló ampliamente el tema de los elementos de programación necesarios para la creación de un nuevo módulo en el software LabVIEW. En esta sección se realizará el diseño del nuevo módulo que manejará el actuador pinza inteligente, el cual se llamará de aquí en adelante: bloque Pinza NXT.

Para programar el bloque, se recomienda utilizar la versión 7.1 de LabVIEW, debido a que el lenguaje NXT-G está basado en esta versión. En este proyecto se utiliza la versión estudiantil 7.1, el cual puede ser solicitado en la página oficial de National Instrumentes www.ni.com (última consulta: octubre de 2013).

En la programación del bloque es importante mencionar el tipo de dato de variables que soporta el Toolkit para LEGO® MINDSTORMS® NXT los cuales son:

- Números enteros con y sin signo, longitud 8, 16 y 32 bits.
- Booleanos.
- *String*.
- Arreglo que contenga enteros, booleanos, strings y *clusters*. Soporte de una dimensión.
- *Clusters* que contengan enteros, booleanos, *strings*, arreglos de una dimensión u otros *clusters*.

3.3.1. Creación de un bloque simple

Para empezar se creará por medio de la opción de Wizard un módulo simple, el cual será modificado según las necesidades a cubrir para el manejo del actuador pinza inteligente.

Al nuevo bloque se le asigna el nombre Pinza NXT; los principales archivos generados son los siguientes:

- Pinza NXT.vi
- Pinza NXTSub.vi
- Draw Pinza NXT.vi
- Config Pinza NXT.vi
- Drawer.dat

La programación se compone de dos partes que son: programación que comprende la lógica y la parte visual que se presenta al usuario.

La parte lógica es programada en los VI's Pinza NXT y Pinza NXTSub, mientras que la de programación visual es en los VI's Config Pinza NXT y Draw Pinza NXT.

En las siguientes secciones primero, se desarrolla la programación de lógica y luego se procederá con la parte visual.

3.3.2. Diseño de programación parte lógica

En esta sección se abarcará la programación en el software de LabVIEW describiendo las diferentes variables y funciones utilizadas para el nuevo actuador pinza inteligente.

3.3.2.1. Pinza NXT.vi

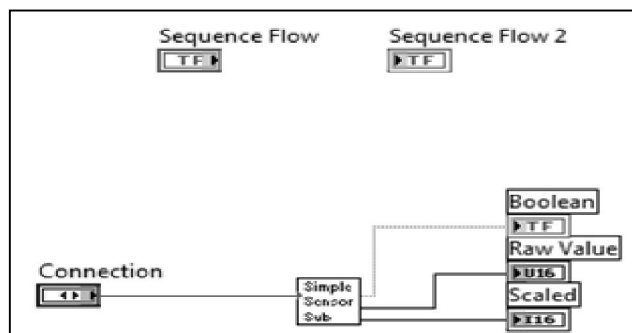
Para el desarrollo del programa, primero es necesario identificar las variables de entrada y salida que tendrá el programa, las cuales son:

- Variables de entrada
 - Puerto: variable que tendrá el valor del número de puerto al cual estará conectado el dispositivo esclavo.
 - Dirección: variable que tendrá el valor de la dirección del dispositivo esclavo.
 - Comando: variable que solo aceptará valores de 0 a 10, cada valor numérico tendrá un valor hexadecimal asociado el cual

representa cada uno de los comandos presentados en la tabla XXXV.

- Longitud de apertura: variable que tendrá el valor que indica la posición a la cual debe ser posicionado el actuador pinza inteligente, su rango de operación es de 0 a 100.
- Variables de salida
 - Estado actual: variable que indicará el estado actual del dispositivo.
 - Posición actual: variable que indicará la posición actual del actuador pinza inteligente.
 - Indicador de error: variable de tipo booleana, la cual indicará si ha ocurrido algún problema en la ejecución del programa.

Figura 71. Diagrama de bloque Pinza NXT.vi



Fuente: elaboración propia.

Luego de haber definido las variables de entrada y salida, se procede a modificar el programa que inicialmente LabVIEW ha generado.

El programa inicialmente tiene creado lo siguiente:

- Variables Sequence Flow (entrada) y Sequence Flow2 (salida) ambas de tipo booleano, propias del compilador de MINDSTORMS, se emplean para la conexión de SubVI's que necesitan ejecutarse secuencialmente. Estas variables no serán modificadas.
- Variable Connection donde se ingresa el puerto a utilizar. Esta variable no será modificada.
- Variable de salida Yes/No (de tipo booleano), Raw Value y Scaled (ambas de tipo entero).
- Llamada a la subrutina Pinza NXTSub.vi, nombre visible Simple Sensor Sub.

El diagrama de bloques creado por LabVIEW se muestra en la figura 71, ahora se procederá a modificarlo de la siguiente manera:

Se crean las variables de entrada y salida: dirección, comando y longitud de apertura; todas de tipo de dato numérico de 8 bits, para la variable puerto se utiliza la ya creada Connection. Las variables de salida son: estatus de tipo *string*, posición actual de tipo numérico de 8 bits e indicador de error de tipo booleano.

Se utiliza el programa Set Type/Mode, el cual se encuentra inicialmente en Pinza NXTSub.vi. Lo único que se realiza es llamar este programa desde Pinza NXT.vi y no desde Pinza NXTSub.vi como inicialmente se encuentra creada la programación del VI. Además se cambiarán sus controles de entrada: en Type se le configura LOWSPEED y en Mode se le asigna RAWMODE.

Se realiza la llamada del programa Pinza NXTSub.vi el cual tiene el nombre en recuadro Simple Sensor Sub, este nombre se procede a cambiarlo por Pinza NXTSub. Este programa se encargará de realizar las operaciones de lectura y escritura I²C.

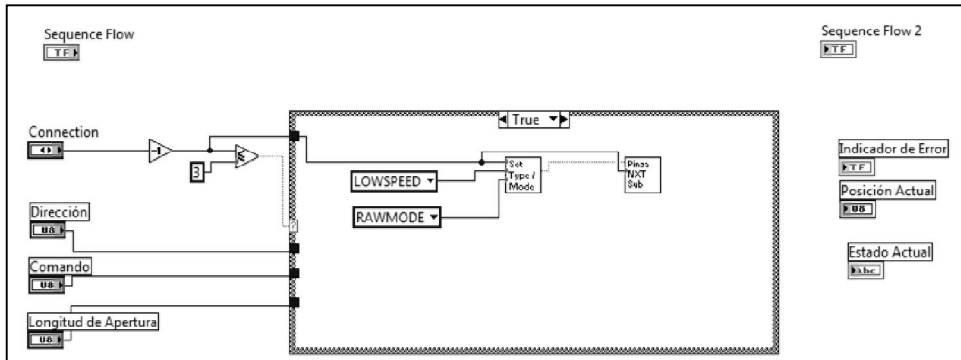
Se colocará una estructura de condición True/False para verificar el valor de la variable puerto, si este valor es menor o igual a cuatro procederá a seguir la secuencia del programa y si es falso no se ejecutará ninguna instrucción.

Se procede a realizar las conexiones de ícono para cada una de las variables y se coloca un Patterns de 12 casillas, de no hacer este requerimiento el programa dará error en su ejecución.

Luego de realizar los pasos anteriores el programa queda como se muestra en la figura 72.

Como se puede observar las conexiones hacia el programa Pinza NXTSub.vi no se han realizado debido a que aún no se han declarado sus variables de entrada y salida, lo cual se procede a realizar en la siguiente sección.

Figura 72. Diagrama de bloques Pinza NXT.vi

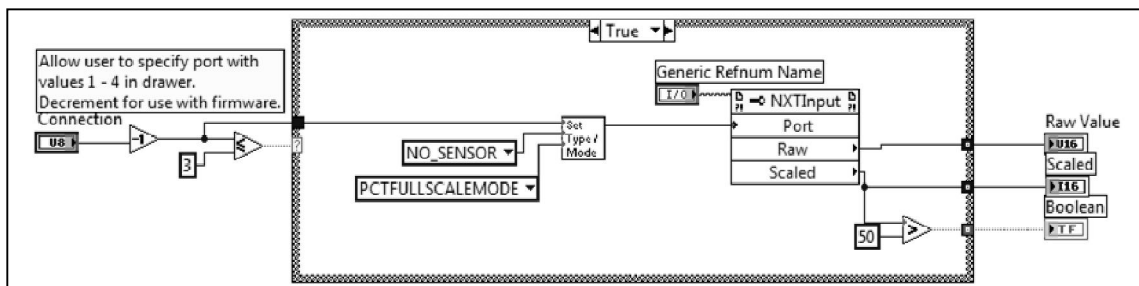


Fuente: elaboración propia.

3.3.2.2. Pinza NXTSub.vi

Inicialmente al abrir el diagrama de bloque del programa Pinza NXTSub.vi se encuentra el siguiente código.

Figura 73. Diagrama de bloques Pinza NXTSub.vi



Fuente: elaboración propia.

Para empezar se elimina todo lo que se encuentra en la estructura True/False y luego se procede a crear las mismas variables de entradas y salidas con el mismo tipo de datos del programa Pinza NXT.vi.

El programa Pinza NXTSub.vi se encargará de la comunicación I²C, para ello es necesario utilizar las clases mencionadas en el capítulo dos sección 2.4, en particular la clase NXTSyscall, método NXTCommLSCheckStatus, NXTCommLSWrite y NXTCommLSRead.

Tabla XXXIV. **Valores de entrada y salida métodos NXTComm - LSWrite/LSRead**

Parametros de NXTCommLSWrite														
<table border="1"> <tr> <td>↳ NXTSystemCall</td> <td>↳</td> </tr> <tr> <td>NXTCommLSWrite</td> <td>↳</td> </tr> <tr> <td>Port</td> <td>↳</td> </tr> <tr> <td>Buffer</td> <td>↳</td> </tr> <tr> <td>ReturnLength</td> <td>↳</td> </tr> </table>	↳ NXTSystemCall	↳	NXTCommLSWrite	↳	Port	↳	Buffer	↳	ReturnLength	↳	Port	U8	Entrada	Puertos, [0,3]
	↳ NXTSystemCall	↳												
	NXTCommLSWrite	↳												
	Port	↳												
Buffer	↳													
ReturnLength	↳													
Buffer	U8 matriz	Entrada	Hasta 16 bytes para escribir al dispositivo											
Return Length	U8	Entrada	Número de bytes que se eseran del dispositivo en respuesta a la escritura de datos en el buffer, maximo 16.											
Parametros NXTCommLSRead														
<table border="1"> <tr> <td>↳ NXTSystemCall</td> <td>↳</td> </tr> <tr> <td>NXTCommLSRead</td> <td>↳</td> </tr> <tr> <td>Port</td> <td>↳</td> </tr> <tr> <td>Buffer</td> <td>↳</td> </tr> <tr> <td>BufferLength</td> <td>↳</td> </tr> </table>	↳ NXTSystemCall	↳	NXTCommLSRead	↳	Port	↳	Buffer	↳	BufferLength	↳	Port	U8	Entrada	Puertos, [0,3]
	↳ NXTSystemCall	↳												
	NXTCommLSRead	↳												
	Port	↳												
Buffer	↳													
BufferLength	↳													
Buffer	U8 matriz	Salida	Número de bytes de lectrua											
Buffer Length	U8	Entrada	Número de bytes a leer en el buffer, número de bytes limitado a la dispinibilidad del buffer de lectrua interna.											

Fuente: elaboración propia.

Para el diseño del programa se toman las siguientes consideraciones:

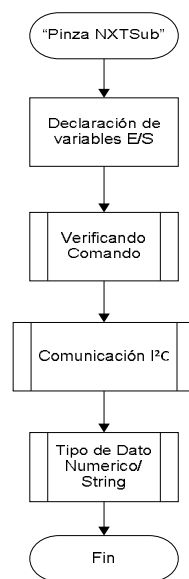
Verificar el valor de la variable comando, para poder decidir el proceso de operación a realizar. Este proceso deberá de indicar si es una operación de escritura o lectura y si se espera respuesta por parte del dispositivo esclavo.

Se diseña la programación de comunicación I²C mediante la creación de Invoke Node clase NXTSyscall.

Se ha definido que las variables de salida son de tipo entero para posición actual y *string* para estatus de salida, por lo que se debe diseñar la programación a manera de proporcionar los tipos de datos correctos a las variables antes mencionadas.

Para desarrollar el programa se presenta primero el diagrama de flujo en la figura 74.

Figura 74. **Diagrama de flujo programa Pinza NXTSub.vi**



Fuente: elaboración propia.

Se detalla a continuación cada bloque de programación:

Declaración de variables E/S: son creadas las variables de entrada y salida, las cuales son las mismas que fueron creadas en el programa Pinza NXT.vi.

- Verificando comando

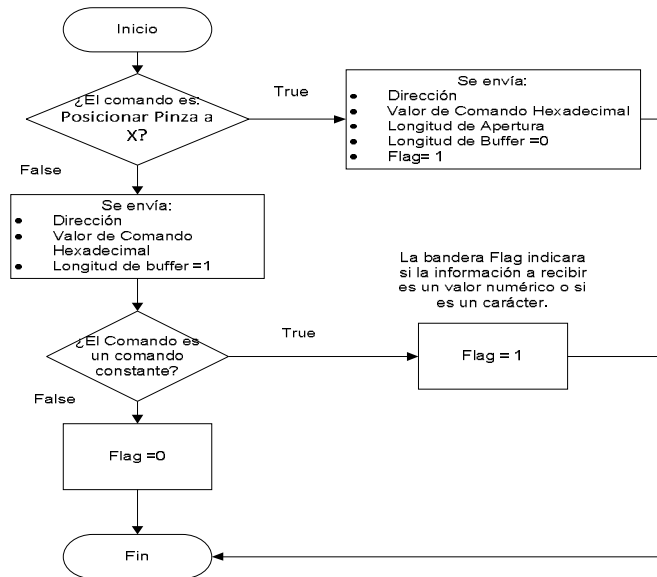
La rutina se programa a base de estructuras *true/false*. Se presenta en la tabla XXXV la trama de *bytes* que el bloque NXT debe de transmitir en el bus I²C y la longitud de *bytes* a recibir y/o enviar.

Tabla XXXV. **Resumen de comandos: actuador pinza inteligente**

	NXT Transmite			
<i>Read</i>	<i>Byte 0</i>	<i>Byte 1</i>	<i>Byte 2</i>	<i>Bytes a recibir</i>
Nombre comando	Dirección + bit R/W	Comandos		
Versión	0x14	0x00		8
ID del producto	0x14	0x08		8
Tipo de sensor	0x14	0x10		8
Factor cero	0x14	0x18		1
Factor de escala del elemento	0x14	0x19		1
Factor de escala de división	0x14	0x1A		1
Unidad de medida	0x14	0x1B		7
Comando de estado	0x14	0x40		1
Posición actual	0x14	0x41		1
<i>Write</i>				
Posicionar pinza a X	0x14	0x81	Longitud de apertura	0
Inicializar pinza	0x14	0x82		0

Fuente: elaboración propia.

Figura 75. Diagrama de flujo: verificación de comandos



Fuente: elaboración propia.

Mediante este programa se determina qué comando se enviará por el bus I²C y además, determina la longitud del búfer, dato que será enviado a NXTComm LSWrite/LSRead.

Como se observa en el diagrama de flujo de la figura 75, esta parte de código proporciona a los métodos NXTComm -LSWrite/LSRead la siguiente información: dirección del dispositivo esclavo, el comando en número hexadecimal y la longitud del búfer de retorno y de envío.

Mediante la bandera llamada flag indica qué tipo de datos se recibirá, si es un valor numérico como resultado o si es una cadena de datos de información.

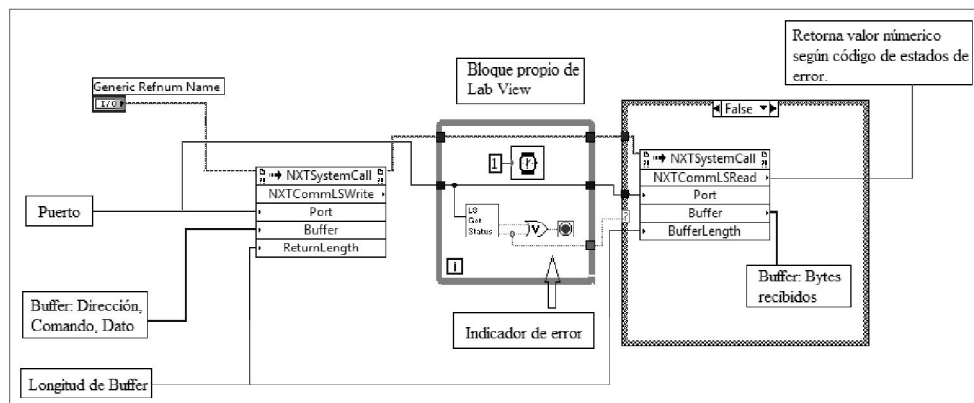
- Comunicación I²C

Rutina encargada de realizar la comunicación I²C por medio de la clase NXTSystem Call métodos NXTCommLSCheckStatus, NXTCommLSWrite y NXTCommLSRead.

Básicamente se colocarán dos Invoke Node, uno para NXTCommLSWrite y el otro para NXTCommLSRead. Seguidamente se creará la variable Generic Refnum Name, la cual se encuentra en el menú de LabVIEW en control de aplicaciones, esta variable se interconectará en cada Invoke Node, si esta variable no es creada e interconectada, el programa no podrá ser ejecutado pues dará error.

Adicionalmente se utilizará una rutina propiamente de LabVIEW, la cual se presenta en la figura 76. Esta rutina básicamente realiza la verificación de la comunicación I²C por medio NXTCommLSCheckStatus, verifica si no ha ocurrido algún evento que afecte la ejecución de la rutina.

Figura 76. Diagrama de bloques de la programación comunicación I²C



Fuente: elaboración propia.

Especificaciones del método NXTCommLSWrite:

- Port: espera un valor numérico, este valor se ha limitado de 1 a 4.
- Búfer (*buffer*): este parámetro espera secuencias de *bytes* a ser transmitidos en el bus, se envía la siguiente trama: dirección del dispositivo, valor hexadecimal del comando y el valor decimal de la variable posición de apertura si fuera el caso.
- ReturnLength: por medio de este parámetro se le indica al programa si debe prepararse para recibir información del dispositivo esclavo, aquí se indica cuántos *bytes* se esperan recibir.

Especificaciones del método NXTCommLSRead

- NXTCommLSRead: por medio de este indicador es posible saber si ha ocurrido algún error en la comunicación. Si el valor del indicador es diferente de 0, indica que ha ocurrido algún evento que perjudico la comunicación.
- Port: espera un valor numérico, éste se ha limitado de 1 a 4.
- Búfer: proporciona los valores recibidos desde el dispositivo maestro.
- BufferLength: por medio de este parámetro se le indica al programa si debe prepararse para recibir información del

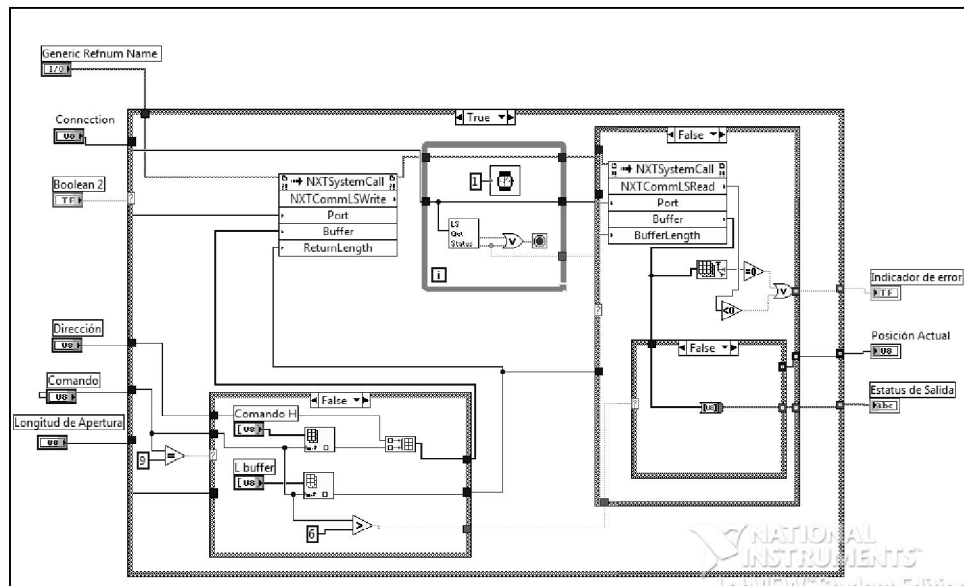
dispositivo esclavo, aquí se indica cuántos *bytes* se esperan recibir.

- Tipo de dato *string*/Numérico

En esta rutina se transforma según sea el caso el valor recibido en el búfer a un valor numérico o a una cadena de caracteres.

En la figura 77 se presenta el diagrama de bloques de la programación de Pinza NXTSub.vi en LabView.

Figura 77. Diagrama de bloques Pinza NXTSub.vi



Fuente: elaboración propia.

Para finalizar la programación lógica, se debe de realizar la conexión de ícono en el programa Pinza NXTSub.vi y luego realizar las conexiones que quedaron pendientes en el programa Pinza NXT.vi.

3.3.3. Diseño de programación parte visual

Los programas Config Pinza NXT.vi y Draw Pinza NXT.vi son programas que serán modificados para presentar al usuario un entorno visual agradable a la hora que este se ejecute en el programa NXT-G.

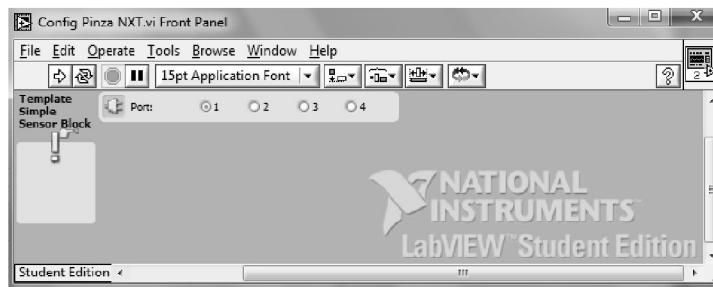
El programa Config Pinza NXT.vi es el que define lo que aparece en el panel de configuración en la parte inferior del software NXT-G, el programa Draw Pinza NXT.vi se encarga del formato fuente de los campos de cada carácter.

3.3.3.1. Config Pinza NXT.vi

El VI de configuración proporciona las funciones necesarias para la actualización de valores y lectura de datos del sensor y traslada estos valores al programa Pinza NXTSub.vi.

Al abrir el programa Config Pinza NXT.vi se encuentra en el panel frontal lo siguiente:

Figura 78. **Panel frontal del programa Config Pinza NXT.vi**



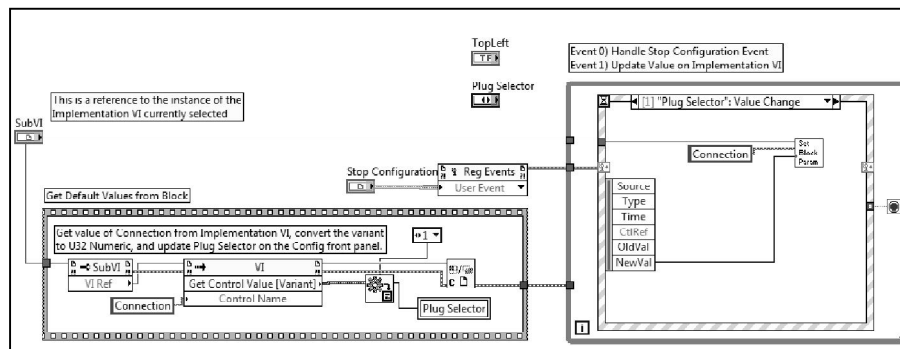
Fuente: elaboración propia.

Los elementos automáticamente creados son:

- Port: control de puerto, este no será modificado.
- Template Simple Sensor Block: elemento visual modificable, se cambiará por Pinza NXT.
- Ícono: elemento visual el cual consiste de la imagen de un signo de exclamación seguido por la imagen de una mano apuntando con su dedo índice un *push botton*. Esta imagen se remplazará.
- Recuadro de color amarillo: este recuadro se modificará.

Se abre el diagrama de bloques del programa Config Pinza NXT.vi se tiene la programación siguiente:

Figura 79. Diagrama de bloques Config Pinza NXT.vi

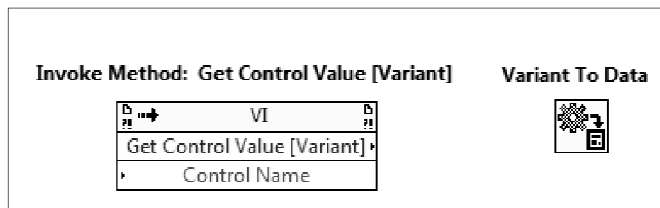


Fuente: elaboración propia.

El programa está diseñado para poder operar los valores introducidos desde este programa y ser trasladados al aplicativo Pinza NXTSub.vi. El programa utiliza el método VI Get Control Value [Variant] Method para obtener el valor del control denominado dato variante y con la función Variant To Data

convierte los datos de la variante a un tipo de datos que la LabView pueda mostrar y/o procesar.

Figura 80. **Funciones utilizadas en Config Pinza NXT.vi**



Fuente: elaboración propia.

Si se observa la única variable de control de entrada creada es Port, este se refleja tanto en el panel frontal y en el diagrama de bloques. Primero se declara la variable la cual en este caso se llama Plug Selector (dato variante), este dato corresponderá a la variable de control Connection del programa Pinza NXTSub.vi y es declarada como variable constante y se conecta en Control Name.

El siguiente bloque se observa la estructura de selección, parte encargada de actualizar la información en la aplicación Pinza NXTSub.vi.

Ahora se modificará el programa según la conveniencia:

Se declararán las variables de control: dirección, comando, etc., es importante denotar que el nombre debe ser igual al declarado en aplicativo Pinza NXTSub.vi.

Para cada variable se le asigna un Invoke Method y una función Variar To Data y se conecta a cada función el control/indicador constante de cada variable.

En estructura de selección se añaden opciones para cada nueva variable creadas.

Se modifica el panel frontal. En la figura 81 se muestra el panel frontal del programa Config Pinza NXT.vi.

Figura 81. **Panel frontal Config Pinza NXT.vi**



Fuente: elaboración propia.

Al diseñar el programa la variable longitud de apertura, el usuario tendrá dos formas de introducir el valor a esta variable, una forma será en deslizar la barra indicadora y la otra forma será introduciendo el valor al recuadro siguiente. Es importante mencionar que para ello se utilizan dos variables de control, una que representa al control deslizable y el otro al control del recuadro.

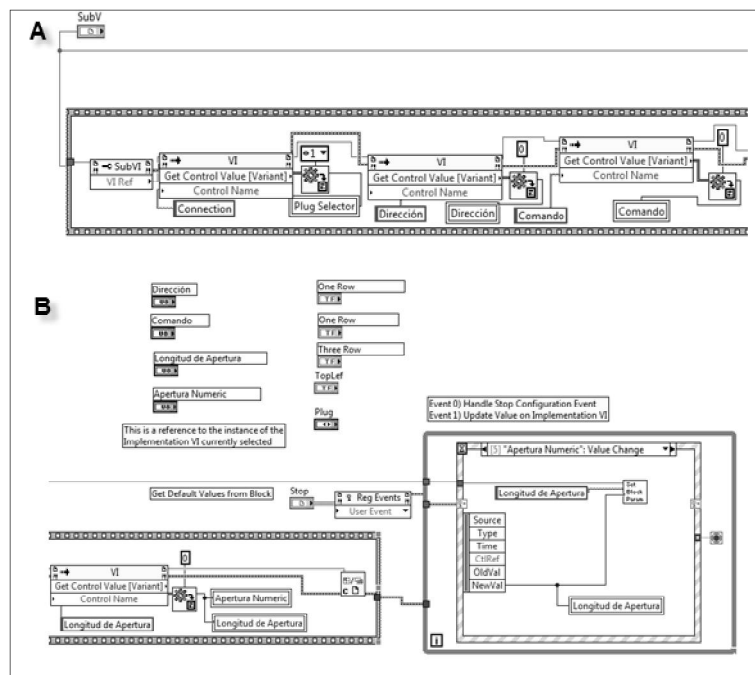
Para la variable comando se colocó un control en forma de listado de opciones para el usuario. Esta variable por cada elección le corresponde internamente un valor numérico decimal, estos valores son reflejados en la tabla XXXVI. En la figura 82 se presenta la programación en diagrama de bloques.

Tabla XXXVI. Valores asignados a cada comando

Valor Numérico	Comando	Valor Hexadecimal
0	Versión	0x00
1	ID del producto	0x08
2	Tipo de sensor	0x10
3	Factor cero	0x18
4	Factor de escala	0x19
5	Factor de división	0x1A
6	Unidad de medida	0x1B
7	Estado actual	0x40
8	Posición actual	0x41
9	Aperturar	0x81
10	Inicializar	0x82

Fuente: elaboración propia.

Figura 82. Diagrama de bloques programación Config Pinza NXT.vi



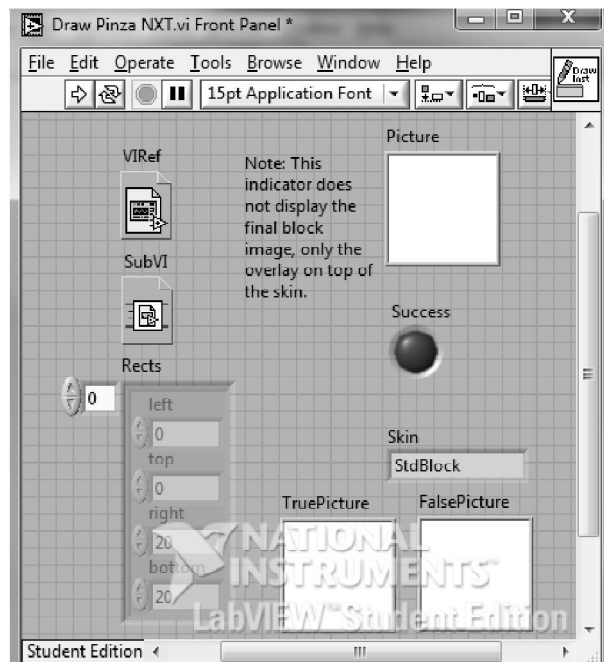
Fuente: elaboración propia.

3.3.3.2. Draw Pinza NXT.vi

Draw VI se encarga de actualizar la imagen del bloque creado en el diagrama de bloques del software NXT-G.

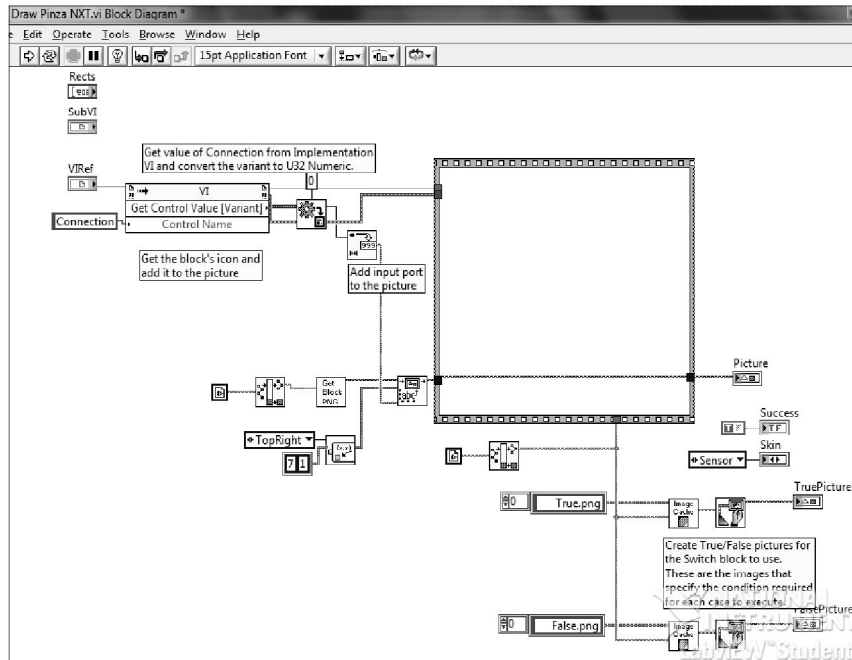
Al abrir el programa Draw Pinza NXT.vi, se abre la ventana del panel frontal el cual se muestra en la figura 83. Esta parte del programa no será modificada.

Figura 83. Panel frontal Draw Pinza NXT.vi



Fuente: elaboración propia.

Figura 84. Diagrama de bloques Draw Pinza NXT.vi



Fuente: elaboración propia.

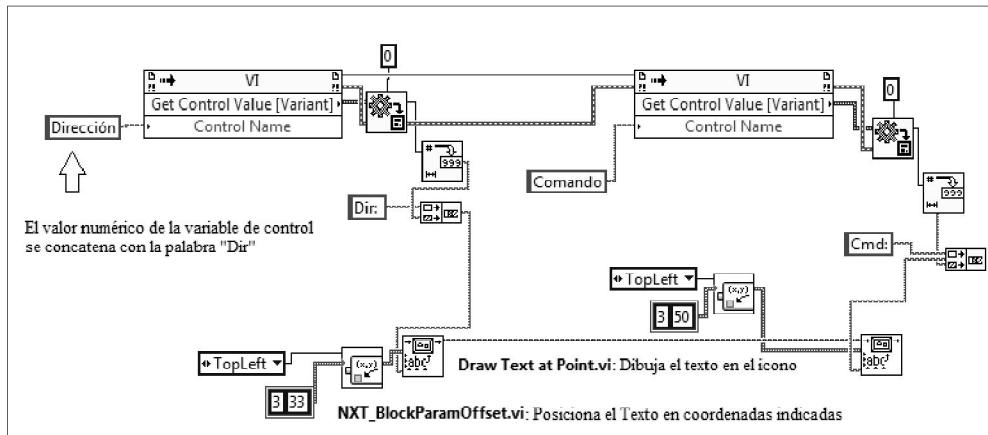
El diagrama de bloques se programará para que se muestre además del puerto, la dirección del dispositivo esclavo y el valor del comando al ejecutar el módulo del bloque programa Pinza NXT.vi en el software NXT-G al ser seleccionado y colocado en la zona de trabajo de este programa.

La manera de lograr que se visualice lo antes mencionado se deberá de agregar el siguiente grupo de funciones:

- Get Control Value [variant]
- Variant To Data
- Draw text at point.vi
- NXT_BlockParamOffset.vi

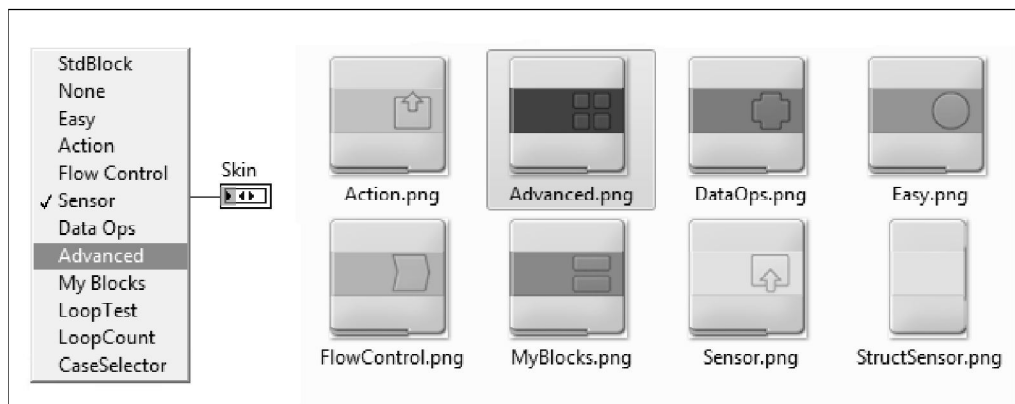
Además se seleccionará en la función Skin la opción Advanced. En las figuras 85, 86 y 87 se muestra la programación realizada.

Figura 85. Programación en diagrama de bloques Draw Pinza NXT.vi



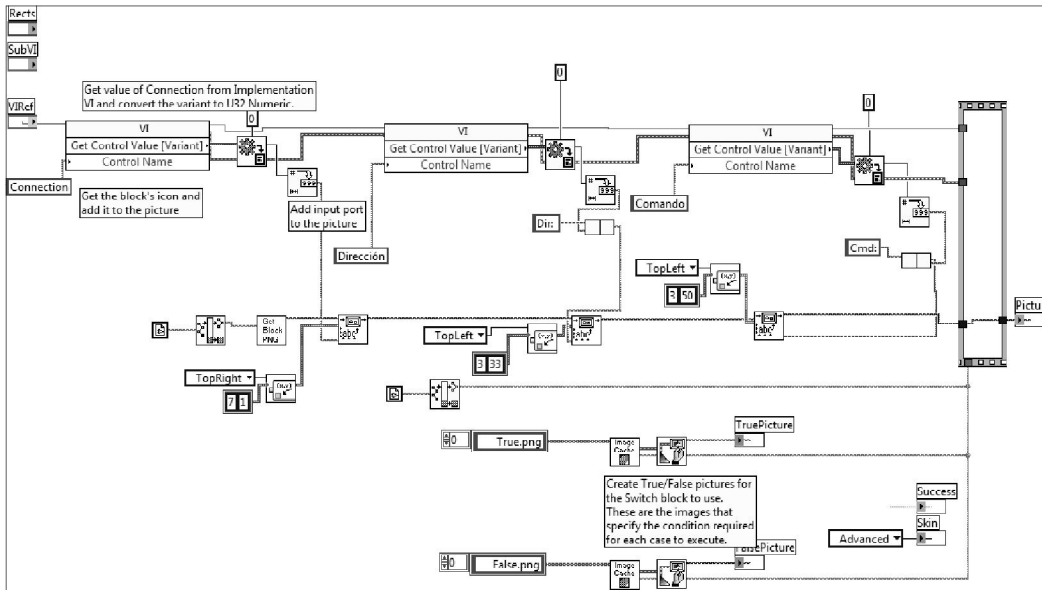
Fuente: elaboración propia.

Figura 86. Opciones de la función Skin



Fuente: elaboración propia.

Figura 87. Programa Draw Pinza NXT.vi



Fuente: elaboración propia.

3.3.3.3. Drawers.dat

Este archivo describe el orden en el cual el software NXT-G muestra los controles e indicadores de la aplicación Pinza NXT.vi cuando el bloque es colocado en el área de trabajo y se despliega la parte inferior.

NXT-G, automáticamente incluye los controles e indicadores que se encuentran conectados al ícono, los únicos tipos de datos soportados por este software son: numérico, booleano y *string*.

El archivo Drawers.dat puede ser modificado utilizando un editor de texto, se emplea el programa: bloc de notas para abrir este archivo.

En el archivo se encuentra lo siguiente:

- [Drawers]
- Connection
- Boolean
- Scaled
- Raw value
- [OpenOnDrop]
- Boolean

Donde aparece [Drawers], en la siguiente línea: se encuentran los controles e indicadores de la aplicación Pinza NXT.vi, esta sección determina el orden en el cual serán desplegados. Es importante mencionar que para desplegar estos controles es necesario dar doble clic en la parte inferior del bloque programa.

Donde aparece [OpenOnDrop], en la siguiente línea: se listan los controles e indicadores que se deseen mostrar inmediatamente al colocar el bloque programa en el área de trabajo.

NXT-G despliega por defecto la imagen, asocia el tipo de dato del control o indicador, las imágenes se encuentran disponibles en la siguiente carpeta:

LEGOMINDSTORMSNXT/engine/editorsVIs/resources/blockimages/drawe
rImages.

El archivo Drawer.dat se modifica colocando los controles e indicadores, comenzando con las variables de entrada y luego las de salida, solo se utiliza la opción [Drawers]. El archivo queda de la siguiente manera:

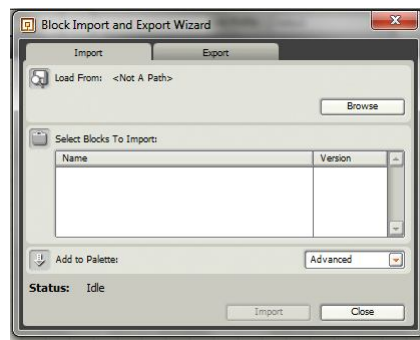
- [Drawers]
- Connection
- Dirección
- Comando
- Longitud de apertura
- Estado actual
- Posición actual
- Indicador de error
- [OpenOnDrop]

3.3.3.4. Importación

Para importar el nuevo bloque programa se debe arrancar el software NXT-G ubicarse en Tools.

Luego seleccionar la opción: Block Import And Export Wizard, después aparece el siguiente diálogo:

Figura 88. Importación del nuevo bloque desde NXT-G

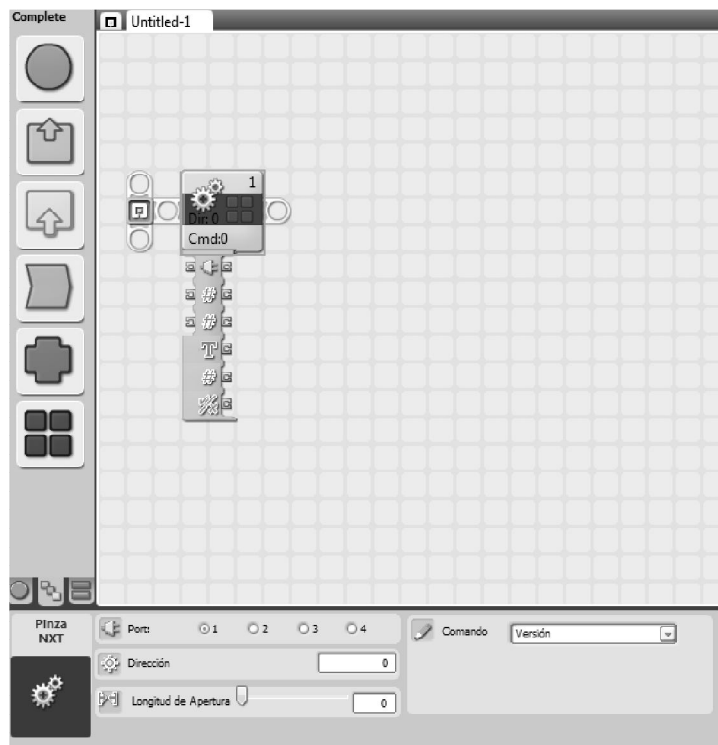


Fuente: elaboración propia.

- Browse: seleccionar el directorio donde fueron creados los diferentes ficheros y aplicaciones.
- Select Blocks To Import: luego del paso anterior aparece el archivo, este debe ser nuevamente seleccionado.
- Add To Palette: seleccionar en qué paleta se colocará el bloque Pinza NXT, se deja Advanced.
- Import: seleccionar para que se realice la importación.

Al ser importado el bloque, ubicarse en la paleta Advanced y se encuentra el bloque programa Pinza NXT.

Figura 89. **Bloque Pinza NXT desde software NXT-G**



Fuente: elaboración propia.

En la figura 89, el bloque fue seleccionado y colocado en el área de trabajo. Los valores pueden ser introducidos en la parte de configuración o por medio de las conexiones que se presentan en la parte inferior del bloque.

4. ANÁLISIS DE FUNCIONAMIENTO

En este capítulo se realizan diferentes pruebas para determinar las características y funcionamiento del nuevo actuador. Para ello se establece la conectividad entre bloque NXT y actuador pinza, se verifica la respuesta del actuador al enviar los diferentes comandos y, por último se realizan pruebas para determinar la capacidad máxima de agarre.

4.1. Conectividad

El bloque NXT utiliza sus propios conectores los cuales son muy parecidos a los conectores RJ12, con la variante que la pestaña de anclaje se encuentra en el lado derecho. En la elaboración del cable de conexión se ha utilizado un cable UTP de 6 hilos de 30 cm de longitud y en ambos extremos del cable se ha colocado un conector RJ12, y se ha eliminado la pestaña de anclaje del conector un de sus extremos. En las figuras 90 y 91 se muestra el cable de conexión y la modificación realizada.

Figura 90. **Cable UTP de 30 cm de longitud**



Fuente: elaboración propia, con apoyo de un cable UTP adaptado para conectar al bloque NXT.

Figura 91. **Conectores RJ12 conectados al cable UTP**



Fuente: elaboración propia, con apoyo de conectores RJ12 adaptados para conectar al bloque NXT.

En la tabla XXXVII se muestra la conexión de pines de entrada y salida del bloque NXT y el actuador pinza inteligente.

Tabla XXXVII. Pin-in y pin-out

Pin	Bloque NXT	Pinza inteligente
1	ANA	SIN CONECCIÓN
2	GND	SIN CONECCIÓN
3	GND	GND
4	VCC	VCC
5	DIGIAI0	SCL
6	DIGIAI1	SDA

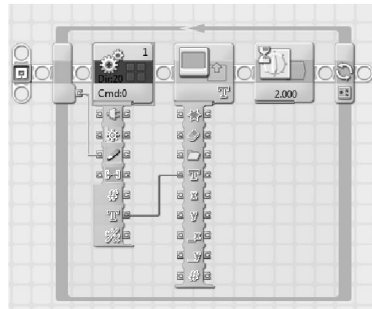
Fuente: elaboración propia.

4.2. Pruebas lógicas

El objetivo de esta sección es verificar si el actuador responde correctamente a los diferentes comandos (previamente programados en el capítulo 3) y para ello se procede a ejecutar el siguiente programa, el cual envía los comandos que se muestran en la tabla XXXVIII y se comprueba la

respuesta del actuador pinza mediante la visualización del mensaje de texto desplegado en la pantalla LCD del bloque NXT.

Figura 92. Programa: comandos



Fuente: elaboración propia.

Luego de ejecutar el programa, la respuesta obtenida y visualizada desde la pantalla LDC por cada instrucción de comando ver la tabla XXXVIII:

Tabla XXXVIII. Comandos enviados desde el bloque NXT hacia el actuador pinza inteligente

Bloque NXT comando	Respuesta desde actuador pinza	Validación
Versión	V1.0	ok
Id del producto	TESIS-12	ok
Sensor	PINZAIN	ok
Factor cero	-	-
Factor de escala	-	-
Factor de división	-	-
Unidad de medida	N/Aplic	ok

Fuente: elaboración propia.

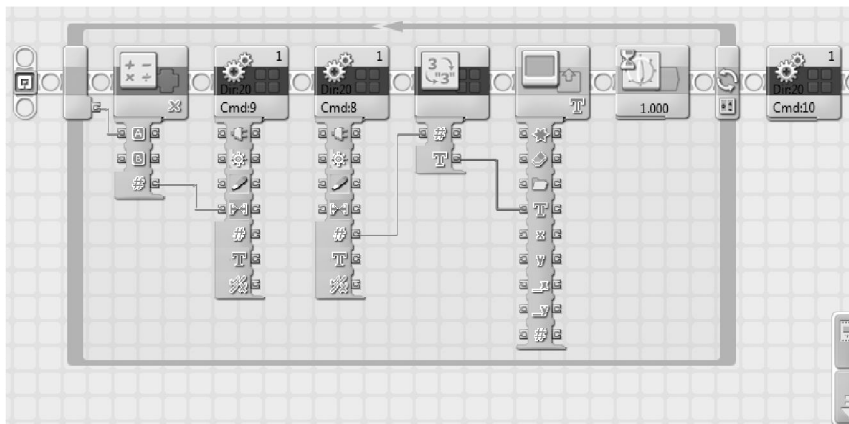
4.3. Pruebas mecánicas

En esta sección se verificarán los comandos que accionan el movimiento del actuador pinza, como también el comando de estado.

El programa verificación_comandos es utilizado para verificar los siguientes comandos: posicionar pinza, posición actual e inicializar pinza.

El programa genera tres movimientos diferentes para aperturar la pinza y se le solicita el estado actual de posicionamiento en cada movimiento realizado, el resultado es reflejado en la pantalla LCD del bloque NXT y luego se le solicita al actuador pinza inteligente colocarse en su posición inicial (completamente abierta).

Figura 93. Programa: verificación de comando de acción



Fuente: elaboración propia.

En la tabla XXXIX se muestran los resultados obtenidos al ejecutar el programa previamente descrito.

Tabla XXXIX. **Resultado de pruebas de movimiento del actuador pinza inteligente**

Comando	Posición de apertura	Bloque NXT visualización en pantalla LCD	Acción de respuesta actuador pinza
Apertura	0	0	ok
Apertura	10	10	ok
Apertura	20	20	ok
Inicializar pinza	100	-	ok

Fuente: elaboración propia.

4.4. Determinación de características

En esta sección se determinarán las características del nuevo actuador pinza inteligente mediante las diferentes pruebas de funcionamiento, utilizando el programa NXT-G y el bloque programable NXT.

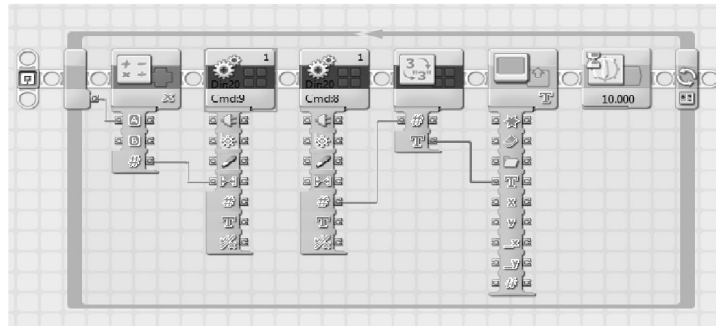
4.4.1. Rango de apertura

Se ejecuta varias veces (5), el programa utilizado en la sección 4,3 y se determina el rango de apertura de la pinza tomando la medida en cada prueba realizada, la cual es de 0 a 29 milímetros con un rango de incerteza de +/- 1 mm.

4.4.2. Linealidad

Se procede a generar un programa que dé 11 movimientos para la apertura de la pinza. El programa se presenta en la figura 94.

Figura 94. Programa aperturar pinza



Fuente: elaboración propia.

El programa aperturar pinza generará 11 movimientos para abrir la pinza y en cada movimiento se detendrá durante 10 segundos, el retraso es utilizado para realizar la medición de la longitud de apertura que en ese momento tendrá la pinza, la primera posición que tomará la pinza será el de cerrar la pinza, posición 0.

Luego de correr el programa se procede a tabular los datos.

Tabla XL. Tabulación de datos experimentales

No.	Posición de Apertura	L1 (mm)	L2 (mm)	L3 (mm)	L4 (mm)	L5(mm)
1	10	1	1	1	1	1
2	20	3	3	3	3	3
3	30	6	5	5	5	5
4	40	8	7	8	7	8
5	50	11	11	11	11	11
6	60	14	14	14	14	14
7	70	18	17	18	17	18
8	80	21	21	21	22	22
9	90	25	25	26	26	26
10	100	29	29	29	29	29

Fuente: elaboración propia.

Proceder a calcular los valores medios y el valor de incerteza mediante las siguientes ecuaciones:

Media aritmética

(ecuación 1)

$$x = \frac{1}{n} \sum_{i=1}^n Li$$

Incerteza de la medida

(ecuación 2)

$$\Delta x = \frac{1}{n} \sum_{i=1}^n |xi - Li|$$

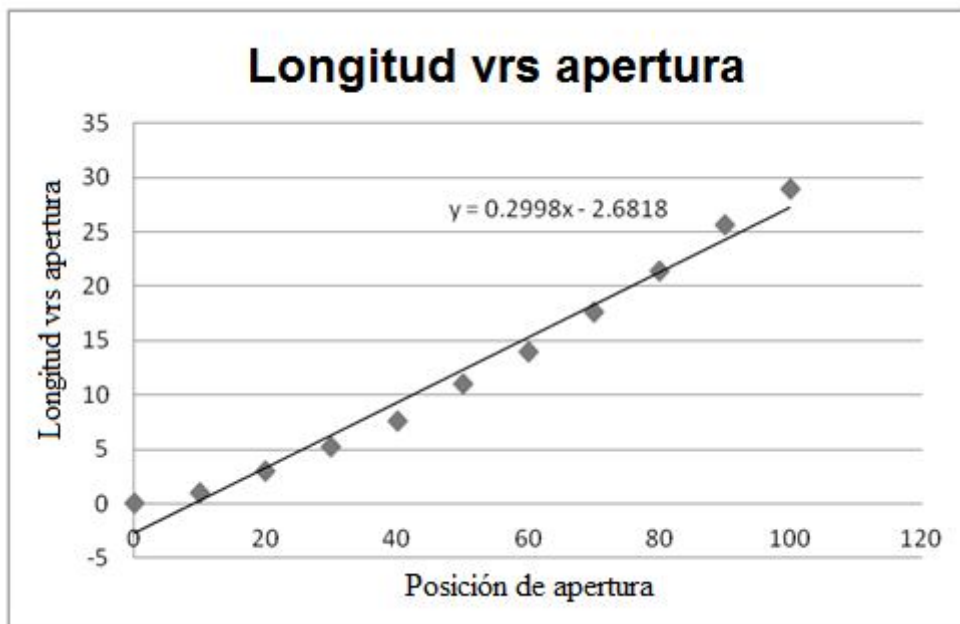
Tabla XLI. **Posición de apertura – longitud de apertura**

No.	Posición de Apertura	L (mm)	Valor medio de L (mm)	ΔL (mm)
1	0	0	0	0
2	10	1	1	0
3	20	3	3	0
4	30	5,2	5,2	0,32
5	40	7,6	7,6	0,48
6	50	11	11	0
7	60	14	14	0
8	70	17,6	17,5	0,48
9	80	21,4	21,4	0,48
10	90	25,6	25,6	0,48
11	100	29	29	0

Fuente: elaboración propia.

Luego de tabular los datos, se procede a utilizar herramienta de cómputo para graficar los datos y determinar la ecuación que describe el movimiento; la gráfica se presenta en la figura 95.

Figura 95. **Longitud de apertura vrs posición de apertura**



Fuente: elaboración propia.

El movimiento del servomotor es lineal, pero al ensamblar la pinza los primeros datos no siguen una tendencia lineal. Sin embargo, se observa que gran parte de la trayectoria es lineal y por conveniencia se procede a linealizar los datos para obtener la ecuación de una recta.

Ecuación de una recta:

$$y = ax + b$$

Donde:

a = la pendiente

b = una constante

y = la longitud de apertura

x = la posición de apertura

Para el cálculo de la recta se utilizarán los puntos en los cuales el comportamiento sea lo más próximo a una recta.

Utilizando los siguientes valores como punto inicial $y_0 = 5,2$ y $x_0 = 30$ y como punto final se tienen los valores $y_1 = 29$ y $x_1 = 100$, para el cálculo de la pendiente:

$$a = \frac{y_1 - y_0}{x_1 - x_0}$$

$$a = \frac{29 - 5,2}{100 - 30} = 0,34$$

Utilizando el punto final y el valor de la pendiente para el cálculo de la constante, se tiene:

$$b = y_1 - ax_1$$

$$b = 29 - (0,34 * 100) = -5$$

La ecuación de la recta es:

$$y = 0,34x - 5$$

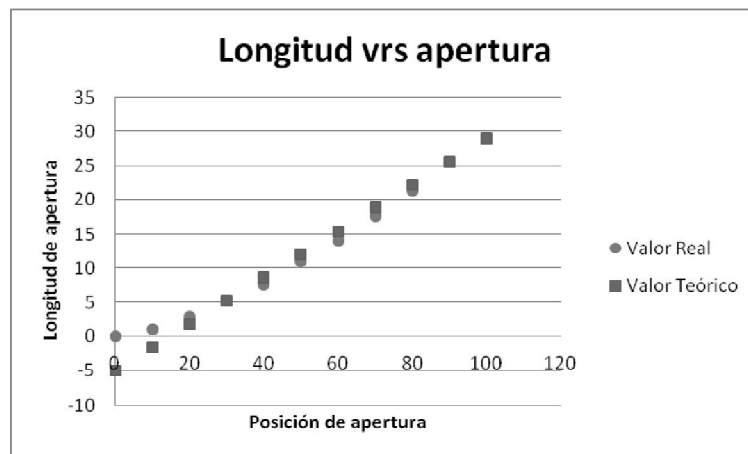
Se procede a calcular el valor de la desviación porcentual, la cual indica el margen de error respecto al valor medio de cada muestra.

Tabla XLII. **Tabulación de datos reales, teóricos y desviación porcentual**

No.	Posición de Apertura	Valor medio de L (mm)	Valor Teórico	DP%
1	0	0	-5,0	-200,0
2	10	1	-1,6	-866,7
3	20	3	1,8	50,0
4	30	5,2	5,2	0,0
5	40	7,6	8,6	-12,3
6	50	11	12,0	-8,7
7	60	14	15,4	-9,5
8	70	17,5	18,8	-7,2
9	80	21,4	22,2	-3,7
10	90	25,6	25,6	0,0
11	100	29	29,0	0,0

Fuente: elaboración propia.

Figura 96. **Diferencia entre el valores reales y valores teóricos**



Fuente: elaboración propia.

Las causas por las cuales el movimiento no es 100 por ciento lineal son: torque que ejerce los engranajes del servomotor hacia engranaje de la pinza y el mecanismo propio de la pinza.

4.4.3. Capacidad de agarre

Para determinar la capacidad de agarre del nuevo actuador se procede a realizar varias pruebas donde la pinza deberá sostener diferentes objetos de distinto peso y tamaño, debido a que no se cuenta con un objeto que se le pueda variar su peso.

Tabla XLIII. **Resultados de pruebas de peso**

Objeto	Peso (g)	Posición pinza	Resultado
1	4,5	0	Realizado
2	50	60	Realizado
3	100	77	Realizado
4	150	85	No realizado

Fuente: elaboración propia.

El resultado de las pruebas demuestran que el actuador pinza no puede sostener objetos que tengan un peso mayor de 150 gramos.

4.4.4. Resumen de características

En esta sección se presenta un resumen de las características más importantes del nuevo actuador pinza inteligente.

Tabla XLIV. **Características del actuador pinza inteligente**

Actuador pinza	
Dimensiones de estructura	10,5 x 9,5 x 5,5 cm
Tensión de funcionamiento	4,8V a 6 V
Peso	0,14 kg
Rango de longitud de apertura	0 a 29 mm
Rango de apertura	0 a 100
Fuerza máxima	1,47 N
Corriente en reposo	8 mA
Corriente en funcionamiento	150 mA sin carga
Corriente máxima	1100 mA

Fuente: elaboración propia.

4.5. Mejoras

A continuación se presentan algunas mejoras que pueden ser implementadas al prototipo:

- Emplear circuitos integrados de superficie para reducir las dimensiones y ocupación del circuito integrado impreso.
- Utilizar PIC's de menor cantidad de pines, por ejemplo, PIC16f873A, PIC16f876A, los cuales son de 28 pines, lo importante es que el PIC soporte el módulo MSSP.
- Emplear sensores de tacto para mejorar el sistema de agarre de la pinza, al colocarle estos sensores el sistema detectará los objetos que estén cerca de cada placa de la pinza y se ajustará automáticamente para el agarre del objeto.

- Modificar el tamaño, peso y estructura física del prototipo de modo tal, que pueda ensamblarse a los elementos propios de LEGO.

CONCLUSIONES

1. El bloque NXT no está limitado a cuatro puertos digitales de entrada/salida, su característica de diseño permite la expansión de hasta 127 puertos más por cada puerto digital mediante el protocolo de comunicación I²C.
2. En la creación y diseño del nuevo actuador fue necesario utilizar un microcontrolador que cumpliera con las características del manejo de puerto serial, comunicación I²C modo esclavo, almacenamiento de información, generación de señales PWM, para que fuese capaz de interactuar con el bloque NXT.
3. La programación del bloque NXT pudo realizarse mediante el software de LabVIEW y LEGO® MINDSTORMS® NXT-G, sin embargo, para programar el nuevo actuador desde NXT-G fue necesario crear un módulo de programación desde LabVIEW, la forma fácil es utilizando la herramienta Wizard que genera todos los archivos necesarios del módulo, estos archivos fueron modificados acorde a las necesidades de la aplicación propia del actuador pinza.
4. El prototipo actuador pinza creado es funcional y sí cumple con los objetivos propuestos, sin embargo, para llevarlo a un nivel competitivo de venta necesita de mejoras tanto en su estructura física, como también en su diseño de software y hardware.

RECOMENDACIONES

1. Al desarrollar actuadores/ sensores inteligentes para LEGO® MINDSTORMS® NXT observar las reglas de protocolo de comunicación propias de LEGO, para la interoperabilidad de ambos dispositivos.
2. El Toolkit para LEGO® MINDSTORMS® NXT de LabVIEW funciona para las versiones de LabVIEW 7.1.x, 8.0, 8.2, 8.5.x, 8.6.x cuando se está utilizando Windows. Se debe ver actualizaciones en la página oficial de National Instruments www.ni.com.
3. Para nuevas versiones de LEGO® MINDSTORMS® NXT se debe descargar la última actualización de firmware y las actualizaciones de software, las cuales se encuentran en la página oficial MINDSTORMS.com.

BIBLIOGRAFÍA

1. ASTOLFO, Dave, et al. *Building robots with LEGO® MINDSTORMS® NXT*. Burlington, MA: Syngress, 2007. 447 p.
2. GASPERI, Michael, et al. *Extreme NXT Extending the LEGO® MINDSTORMS® NXT to the Next Level*. New York: Technology in Action Press, 2007. 286 p.
3. _____. *LabVIEW for LEGO® MINDSTORMS® NXT*. USA: National Technology and Science Press, 2008. 376 p.
4. The LEGO Group. *MINDSTORMS® Hardware Developer kit*, [en línea] <http://ebookbrowse.com/lego-mindstorms-nxt-hardware-developer-kit-pdf-d1384528>. [Consulta: 15 de mayo de 2013].
5. Microchip. *AN734*. USA: Microchip Technology, 2000. 14 p.
6. _____. *PICmicro™ Mid-Range MCU Family Reference Manual*. USA: Microchip Technology, 1997. 688 p.
7. _____. *PIC16F87XA Data Sheet 28/40/44- Pin Enhanced Flash Microcontrollers*. USA: Microchip Technology, 2003. 234 p.
8. PREDKO, Myke. *Programming and customizing the Pic microcontroller*. 2a ed. New York: Tab Books, 2007. 1184 p.

9. Philips Semiconductors. *Application Note I²C bus AN10216-01I²C*, NXP.
USA: Philips, 2003. 51 p.

10. _____. *I²C-bus specification and user manual UM10204*, NXP.
USA: Philips, 2012. 50 p.

APÉNDICES

1. Diagrama eléctrico
2. Programa PIC_NXT
3. Programas del módulo Pinza NXT
4. Programa robot pinza

Apéndice 2. Programa PIC_NXT

```

1  __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _HS_OSC;
2  LIST          P=16F877A
3  INCLUDE<P16F877A.INC>
4
5  valdir                equ  0x21                ; Valor de la dirección del esclavo.
6  temp                  equ  0x22
7  WREGsave              equ  0x23
8  STATUSsave           equ  0x24
9  FSRsave               equ  0x25
10 PCLATHsave           equ  0x26
11
12 var_comando           equ  0x28                ; Contiene el comando recibido
13                       ; por i2c.
14 flag_vc               equ  0x29
15 data_ee_a             equ  0x2a
16 data_ee_d             equ  0x2b
17 flag                  equ  0x2c
18 pos_pinza             equ  0x2d                ; Posición deseada.
19 carga_tmr0            equ  0x2e
20 Byte_1                equ  0x2f
21 Estado                equ  0x40
22 Pos_actua             equ  0x41
23
24
25
26
27          ORG          0X2100
28 DE          V1.0,.0,.0,.0,0,"TESIS-12PINZAIN",.0,.0,.0,"N/Aplic"
29          ORG          0X2164
30 DE          .94,.95,.96,.97,.98,.99,.100,.101,.102,.103,.103,.104,.105,.106,.107,.108,.109,.110,.
111,.112,.113,.113,.114,.115,.116,.117,.118
31

```

Continuación del apéndice 2.

32		ORG	0X217F	
33	DE		.119,.120,.121,.122,.123,.124,.124,.125,.126,.127,.128,.129,.130,.131,.132,.133, 134,.135,.135,.136,.137,.138,.139,.140,.141,.142	
34				
35		ORG	0X2199	
36	DE		.143,.144,.145,.145,.146,.147,.148,.149,.150,.151,.152,.153,.154,.155,.156,.156, 157,.158,.159,.160,.161,.162,.163,.164	
37				
38		ORG	0x21B1	
39	DE		.165,.166,.166,.167,.168,.169,.170,.171,.172,.173,.174,.175,.176,.177,.177,.178, 179,.180,.181,.182,.183,.184,.185,.186	
40				
41		ORG	0x00	
42		goto	INICIO	
43		ORG	0x04	
44		goto	INTERRUPCION	
45				
46	INICIO			
47		call	CONFIGURACION	
48		call	INICIALIZACION_PINZA	
49				
50	PRINCIPAL			
51		bcf	STATUS,RP0	
52		btfs	flag_vc,6	
53		call	IDENTIFICAR_COMANDO	
54		goto	PRINCIPAL	
55				
56				
57	IDENTIFICAR_COMANDO			
58		movlw	0x81	
59		xorwf	var_comando,w	
60		btfs	STATUS,Z	; Compara si es el estado 1.
61		goto	comando_82	; No, sigue al otro estado.
62				
63		movlw	0x01	
64		movwf	Estado	
65		movf	pos_pinza,W	

Continuación del apéndice 2.

```
66      movwf   carga_tmr0
67      btfss  flag,0           ; Ya se tiene cargado en el TMR0 el valor.
68      goto   $-1
69      movf   pos_pinza,w
70      movwf  Pos_actual
71      movlw  0x00
72      movwf  Estado
73      bcf   flag_vc,6
74      bcf   flag,0
75      return
76
77 comando_82
78
79      movlw  0x01
80      movwf  Estado
81      movlw  d'100'
82      movwf  carga_tmr0
83      movwf  Pos_actual
84      clrf  Estado
85      bcf   flag_vc,6
86      return
87
88
89 INTERRUPCION
90      bcf   STATUS,RP0
91      movwf WREGsave
92      swapf STATUS,W
93      movwf STATUSsave
94      movf  PCLATH,w
95      movwf PCLATHsave
96      movf  FSR,w
97      movwf FSRsave
98      call  TMR0_interrupcion
99      call  TMR1_interrupcion
100     call  I2c_interrupcion
101
```

Continuación del apéndice 2.

102		movf	FSRsave,w	
103		movwf	FSR	
104		movf	PCLATHsave,w	
105		movwf	PCLATH	
106		swapf	STATUSsave,w	
107		movwf	STATUS	
108		swapf	WREGsave,f	
109		swapf	WREGsave,w	
110		retfie		
111				
112	i2c_interrupcion			
113		btss	PIR1,SSPIF	; ¿Es una mssp interrupción?
114		return		; No.
115		bcf	PIR1,SSPIF	
116		bsf	STATUS,RP0	
117		movf	SSPSTAT,W	
118		bcf	STATUS,RP0	
119		andlw	b'00101101'	
120		movwf	Temp	; Para chequear la comparación.
121				
122	estado1:			; Operación de escritura, el último byte fue una dirección.
123		bcf	STATUS,RP0	
124		movlw	b'00001001'	
125		xorwf	temp,W	
126		btss	STATUS,Z	; Compara si es el estado 1.
127		goto	estado2	; Si no, sigue al otro estado.
128				
129		movf	SSPBUF,W	; Toma la dirección y la desecha.
130		return		
131				
132	estado2:			; operación de escritura, último byte fue un dato
133				
134		bcf	STATUS,RP0	
135		movlw	b'00101001'	; Búfer está lleno.
136		xorwf	temp,W	
137		btss	STATUS,Z	; ¿Es estado 2?
138		Goto	estado3	; No, pasar al siguiente estado.

Continuación del apéndice 2.

139			
140			
141	bcf	STATUS,RP0	;cambio de banco
142	movf	SSPBUF,W	
143	movwf	Byte_1	
144	btss	flag_vc,0	
145	goto	\$+3	
146	call	Posicionar_pinza	
147	return		
148	call	Verificacion_var_comando	
149	return		
150			
151	estado3:		; El último dato es una dirección y es lectura, manda el primer byte.
152			
153	movlw	b'00001100'	
154	xorwf	temp,w	
155	btss	STATUS,Z	
156	goto	state4	
157			
158	bcf	STATUS,RP0	
159	movf	Byte_1,w	
160	btss	flag_vc,7	
161	goto	\$+5	
162	movwf	FSR	
163	movf	INDF,W	
164	bcf	flag_vc,7	
165	goto	\$+3	
166	movf	Byte_1,w	
167	call	EEPROM_LeeDatos	
168	call	escritura2C	
169	incf	Byte_1,f	
170	return		
171			
172			
173	estado4:		; Estado 4 es para seguir mandando datos al maestro.
174			
175	bcf	STATUS,RP0	

Continuación del apéndice 2.

176	movlw	b'00101100'	
177	xorwf	temp,W	
178	btfs	STATUS,Z	
179	goto	state5	
180			
181	bcf	STATUS,RP0	
182	movf	Byte_1,w	
183	call	EEPROM_LeeDatos	
184	call	escritural2C	
185	incf	Byte_1,f	
186	return		
187			
188	estado5:		
189			
190	movlw	b'00101000'	
191	xorwf	temp,W	; Un NACK fue recibido en la transmisión.
192	btfs	STATUS,Z	
193	goto	I2CErr	
194	return		
195	I2CErr		; Si no pasamos de este estado, ha sucedido un error.
196	bcf	STATUS,RP0	
197	;movlw	b'10101010'	
198	goto	\$; Ha ocurrido un error. Línea de uso del programador (línea deshabilitada).
199	return		
200			
201	escritural2C::		
202			
203	bsf	STATUS,RP0	
204	btfs	SSPSTAT,BF	
205	goto	escritural2C:	
206	bcf	STATUS,RP0	
207	seguir_escrib::		
208	movwf	SSPBUF	
209	btfs	SSPCON,WCOL	
210	goto	seguir_escrib:	
211	bsf	SSPCON,CKP	
212	Return		

Continuación del apéndice 2.

213				
214	Verificacion_var_comando			
215				
216		movlw	0x40	
217		xorwf	Byte_1,w	
218		btsss	STATUS,Z	; Lee estado actual de la pinza.
219		goto	Estado_41	
220				
221		bsf	flag_vc,7	
222		return		
223				
224	Estado_41			
225		movlw	0x41	; Lee posición actual.
226		xorwf	Byte_1,w	
227		btsss	STATUS,Z	
228		goto	Estado_81	
229				
230		bsf	flag_vc,7	
231		return		
232				
233	Estado_81			
234		movlw	0x81	
235		xorwf	Byte_1,w	
236		btsss	STATUS,Z	
237		goto	Estado_82	
238				
239		bsf	flag_vc,0	
240		movf	Byte_1,w	
241		movwf	var_comando	
242		return		
243				
244	Estado_82			
245		movlw	0x82	
246		xorwf	Byte_1,w	
247		btsss	STATUS,Z	
248		return		
249		Movf	Byte_1,w	; Byte_1 es constante.

Continuación del apéndice 2.

250	movwf	var_comando	
251	bsf	flag_vc,6	
252	return		
253			
254	Posicionar_pinza		
255	bcf	STATUS,RP0	
256	bcf	STATUS,RP1	
257	movf	Byte_1,w	
258	movwf	pos_pinza	
259	bsf	flag_vc,6	
260	bcf	flag_vc,0	
261	return		
262			
263	TMR0_interrupcion		
264	btfss	INTCON,TMR0IF	
265	return		
266	bcf	INTCON,TMR0IF	
267	bcf	INTCON,TMR0IE	
268	bcf	PORTB,7	
269	return		
270	TMR1_interrupcion		
271	btfss	PIR1,TMR1IF	
272	return		
273	bcf	PIR1,TMR1IF	
274	movlw	0x2c	
275	movwf	TMR1L	; TMR1 carga datos.
276	movlw	0xcf	
277	movwf	TMR1H	
278	bsf	T1CON,TMR1ON	
279	movf	carga_tmr0,w	; Inicia conteo tmr01.
280	addlw	0.1	
281	movwf	data_ee_addr	
282	call	EEPROM_LeeDatos	
283	movwf	TMR0	
284	bsf	INTCON,TMR0IE	
285	bsf	PORTB,7	
286	bsf	flag,0	

Continuación del apéndice 2.

287	return	
288		
289	CONFIGURACION	
290	bsf	STATUS,RP0
291	bsf	PCON,NOT_POR
292	movlw	b'00011000'
293	movwf	PORTC
294	clrf	PORTD
295	clrf	PORTB
296	movlw	b'00000101'
297	movwf	OPTION_REG
298	bcf	STATUS,RP0
299	movlw	b'00010100'
300	movwf	Valdir
301	clrf	flag_vc
302	clrf	Byte_1
303	clrf	data_ee_addr
304	clrf	Flag
305	clrf	var_comando
306	clrf	PORTB
307	clrf	PORTD
308	clrf	carga_tmr0
309	movlw	d'100'
310	movwf	carga_tmr0
311	clrf	Pos_actual
312	clrf	Estado
313	clrf	pos_pinza
314	clrf	PIR1
315	clrf	TMR1H
316	clrf	TMR1L
317	movlw	0x36
318	movwf	SSPCON
319	movf	valdir,W
320	bsf	STATUS,RP0
321	movwf	SSPADD
322	clrf	SSPSTAT
323	Movlw	b'00001001'

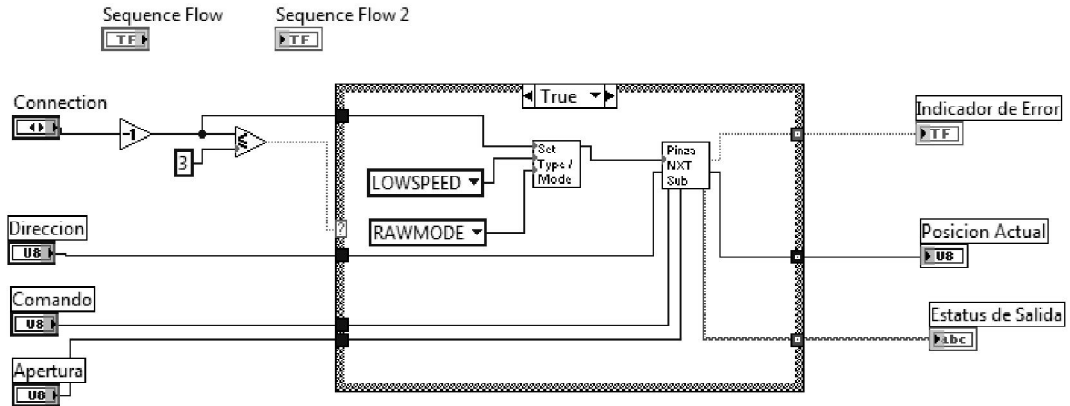
Continuación del apéndice 2.

324	movwf	PIE1
325	bcf	STATUS,RP0
326	clrf	INTCON
327	return	
328		
329	INICIALIZACION_PINZA	
330	bcf	STATUS,RP0
331	movlw	b'110000'
332	movwf	T1CON
333	movlw	0x2c
334	movwf	TMR1L
335	movlw	0xcf
336	movwf	TMR1H
337	bsf	T1CON,TMR1ON
338	movlw	d'186'
339	movwf	TMR0
340	movlw	b'11100000'
341	movwf	INTCON
342	bsf	PORTB,7
343	movlw	d'100'
344	movwf	Pos_actual
345	return	
346		
347	EEPROM_LeeDatos	
348	movwf	data_ee_addr
349	bsf	STATUS,RP1
350	bcf	STATUS,RP0
351	movwf	EEADR
352	bsf	STATUS,RP0
353	bcf	EECON1,EEPGD
354	bsf	EECON1,RD
355	bcf	STATUS,RP0
356	movf	EEDATA,W
357	bcf	STATUS,RP1
358	bcf	STATUS,RP0
359	return	
360	END	

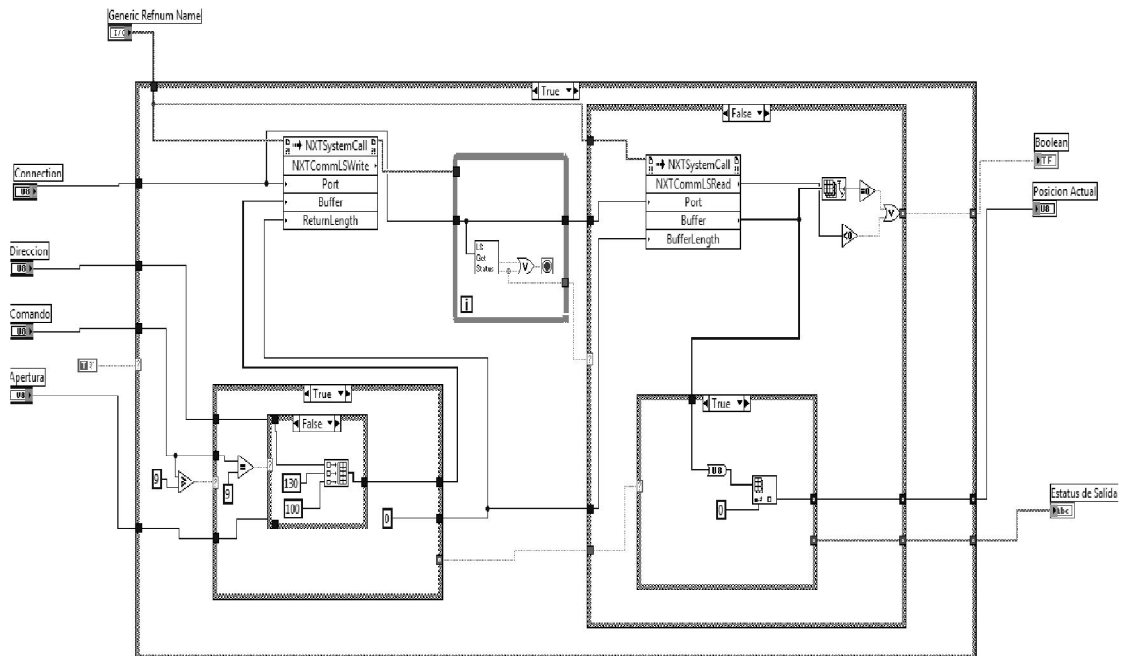
Fuente: elaboración propia.

Apéndice 3. Programas del módulo Pinza NXT

Pinza NXT.vi

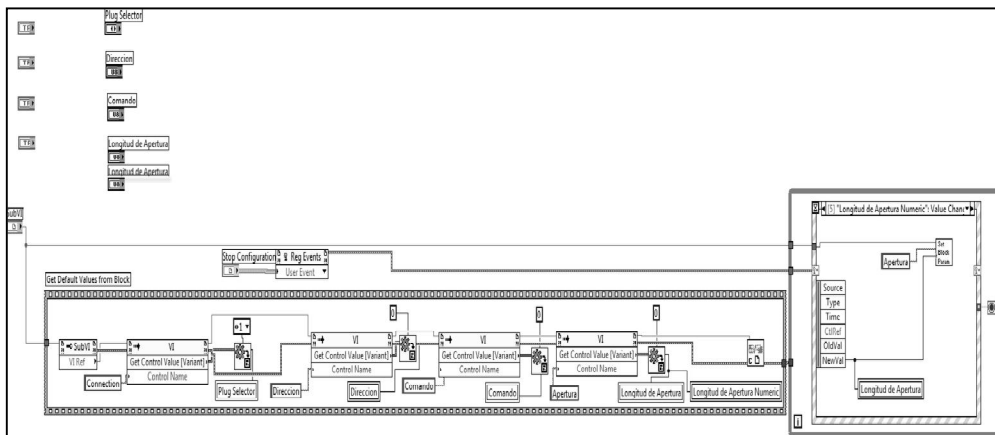


Pinza NXTSub.vi

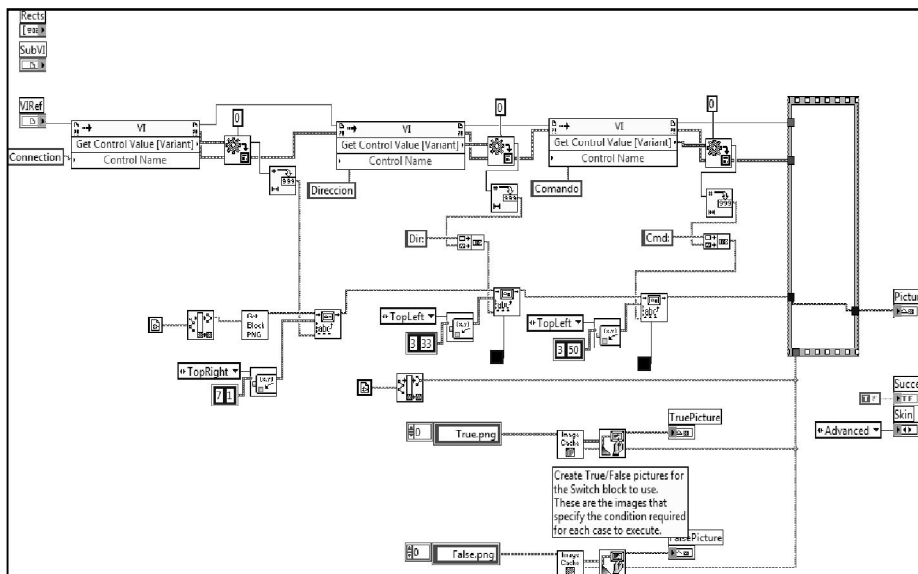


Continuación del apéndice 3.

Config Pinza NXT.vi



Draw Pinza NXT.vi



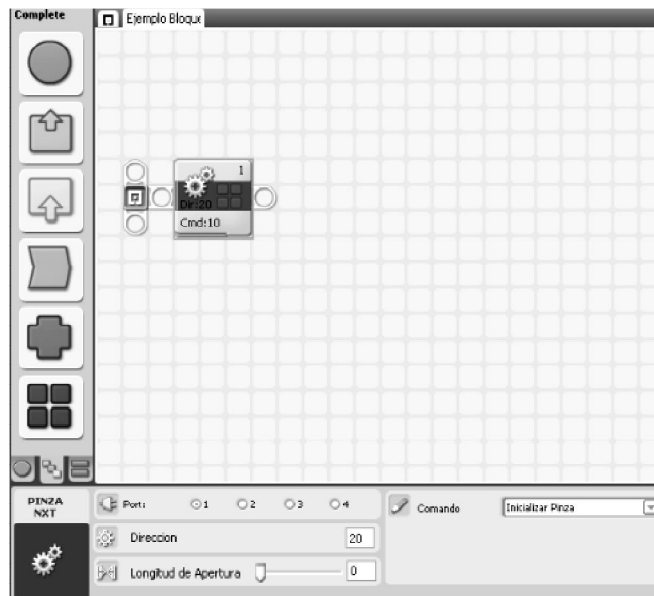
Fuente: elaboración propia.

Apéndice 4. Programa robot pinza

El programa robot pinza tiene como objetivo utilizar el nuevo actuador junto con el bloque NXT, básicamente el programa trata de mover al robot una pequeña distancia para agarrar un objeto y moverlo de su lugar.

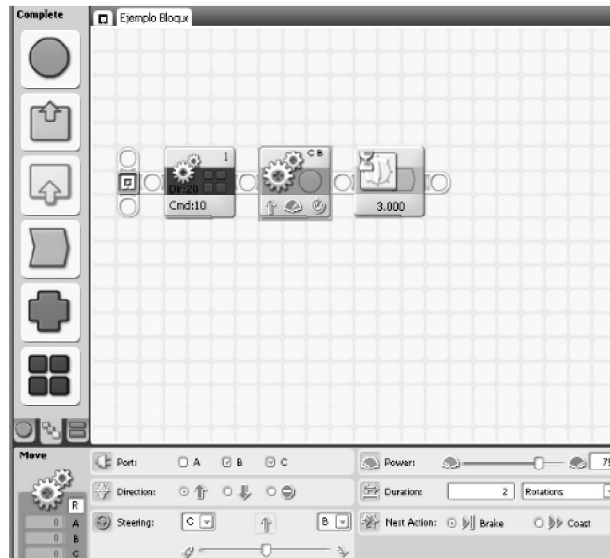
El programa está hecho en el software NXT-G y la secuencia de cada acción es la siguiente:

Iniciar el nuevo actuador pinza, colocándole al módulo su dirección y escogiendo el comando inicializar (posición completamente abierta).

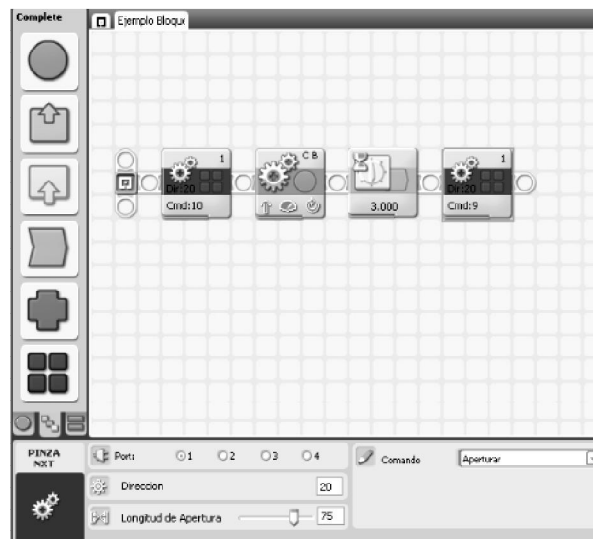


Para mover el robot se utilizará el módulo *move*, donde se escoge la opción de mover los servomotores hacia adelante y darán únicamente dos revoluciones antes de detenerse (distancia de recorrido corta). Luego el robot se detendrá por 3 segundos.

Continuación del apéndice 4.

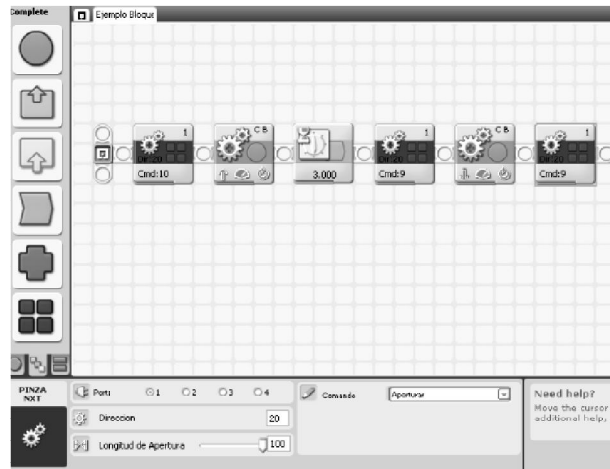


En esta posición el robot se encuentra justo delante de su objetivo, por lo que se programa la pinza para que se cierre, posicionándola a un 75 por ciento de su apertura total. Este último movimiento hará que la pinza agarre el objeto.



Luego que el robot tenga asegurado el objeto retrocederá y lo soltará.

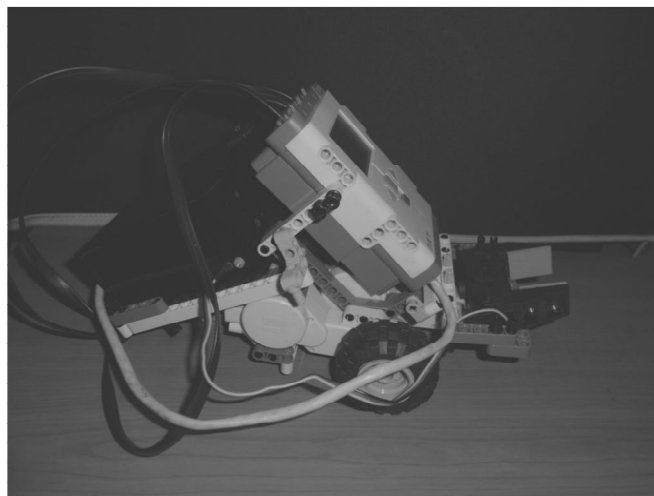
Continuación del apéndice 4.



Ya teniendo el programa listo, se carga al bloque NXT y se ejecuta el programa.

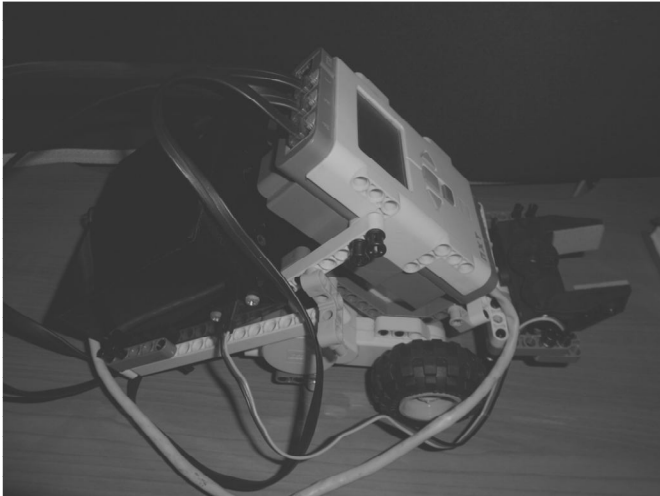
A continuación se presenta la secuencia de movimientos a través de fotografías que fueron tomadas mientras el robot se encontraba en movimiento.

El actuador pinza se ha iniciado

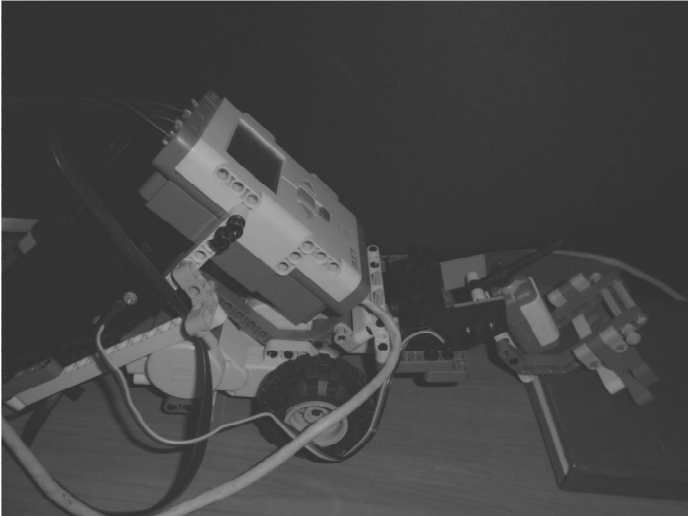


Continuación del apéndice 4.

El robot se mueve hacia el objeto

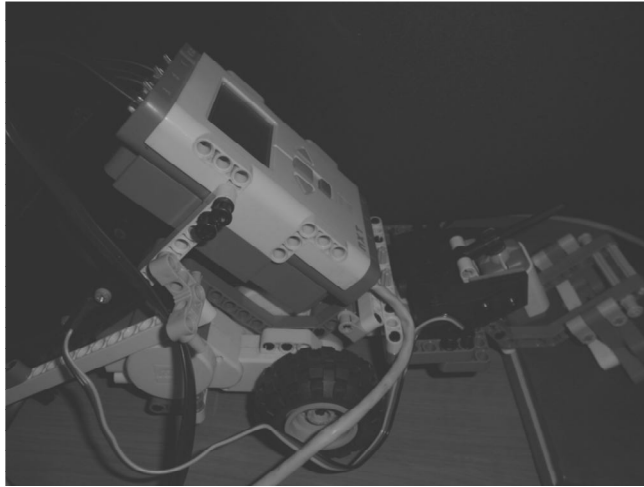


El robot se encuentra justo enfrente de su objetivo

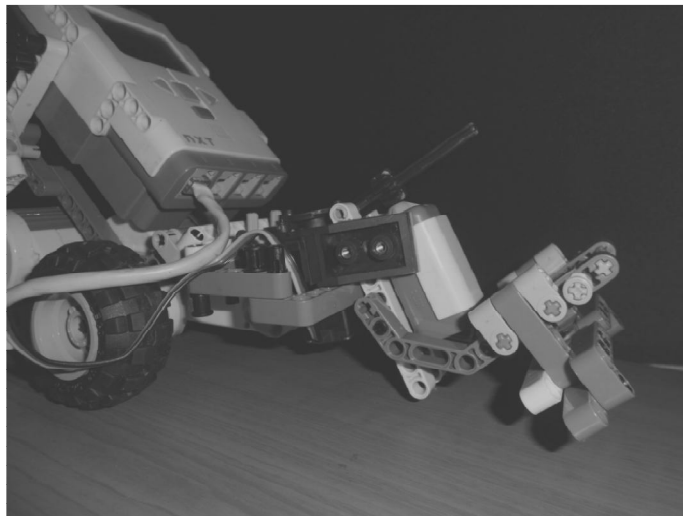


Continuación del apéndice 4.

El robot procede a cerrar la pinza, logrando agarrar el objeto

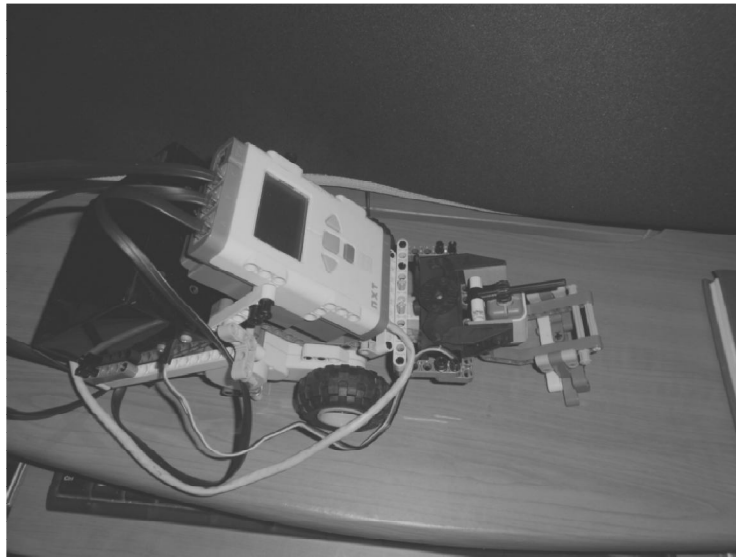


El robot procede a retroceder sosteniendo el objeto



Continuación del apéndice 4.

El robot suelta el objeto



Fuente: elaboración propia, con el apoyo de las herramientas de LEGO® MINDSTORMS®
NXT.