**UNIVERSIDAD AUTÓNOMA DE QUERÉTARO**

**FACULTAD DE INGENIERÍA**

# Desarrollo e Implementación de software para programación de movimientos a robot Bioloid Premium

**TESIS**

Que como parte de los requisitos para obtener el título de

Ingeniero Electromecánico, L. T. Mecatrónica


Presenta:

**Salvador Martínez Cruz**

Dirigido por:

**Dr. Gerardo Israel Pérez Soto, Director**

**Dr. Luis Alberto Morales Hernández, Co-director**


San Juan del Río, Querétaro

Junio 2016

UNIVERSIDAD AUTÓNOMA DE QUERÉTARO

FACULTAD DE INGENIERÍA

CAMPUS SAN JUAN DEL RÍO

# DESARROLLO E IMPLEMENTACIÓN DE SOFTWARE PARA PROGRAMACIÓN DE MOVIMIENTOS A ROBOT BIOLOID PREMIUM

T E S I S

Como parte de los requisitos para obtener el título de

## INGENIERO ELECTROMECÁNICO

Línea Terminal

**MECATRÓNICA**

Presenta

**SALVADOR MARTÍNEZ CRUZ**

Dirigido por:

**DR. GERARDO ISRAEL PÉREZ SOTO**

**S I N O D A L E S:**

**DR. GERARDO ISRAEL PÉREZ SOTO (12950)**
**Presidente** _____

**DR. LUIS ALBERTO MORALES HERNÁNDEZ (6284)**
**Secretario** _____

**DR. JUAN PRIMO BENITEZ RANGEL (2088)**
**Vocal** _____

**DR. MARTÍN VALTIERRA RODRÍGUEZ (11869)**
**Suplente** _____

San Juan del Río, Qro., Agosto 2016

**********

## DEDICATORIA

Para mi madre y hermanos que siempre confiaron en mí y me apoyaron en todo momento.

## AGRADECIMIENTOS

Agradezco por el apoyo en el financiamiento para la compra de los robots utilizados en este proyecto al convenio PEI 2016 (232143) así como por la beca proporcionada para mi uso personal.

A todos los que me ayudaron a desarrollar este trabajo como para la revisión y corrección del documento final.

Al Dr. Gerardo Israel Pérez Soto por confiar en mí y sobre todo por su gran apoyo para la realización de este trabajo.

A mi familia por estar conmigo durante todo este tiempo impulsándome a seguir adelante depositando toda su confianza en cada reto que se me presenta.

## TABLA DE CONTENIDO

## ÍNDICE DE FIGURAS

### 2. FUNDAMENTACIÓN TEÓRICA



### 3. DESCRIPCIÓN DEL SOFTWARE



### 4. IMPLEMENTACIÓN DEL SOFTWARE

**APÉNDICES**

**Apéndice 1. Metodología para la instalación de los programas empleados**



**Apéndice 2. Manual de usuario**

# ÍNDICE DE TABLAS

## 2. FUNDAMENTACIÓN TEÓRICA



## 3. DESCRIPCIÓN DEL SOFTWARE

# 1. INTRODUCCIÓN

En esta sección se describen los fundamentos de los que se partió para realizar este proyecto.

## 1.1. Antecedentes

En esta sección se presentan los trabajos más importantes que se han publicado relacionados con el presente trabajo.

Se han desarrollado diversos trabajos con o para robots del tipo humanoide en los últimos años.

Según Pfeiffer (2011), Un robot humanoide es robot antropomorfo que, además de imitar la apariencia humana, imita algunos aspectos de su conducta. Él presenta un proyecto en el que se desarrolla un guiado gestual de un robot humanoide mediante el uso de un sensor Kinect.

Perea (2010) desarrolla una tarjeta de sensorización y control aplicable para robot humanoide.

Pardos (2005) menciona que los humanos crean entornos adecuados para ser habitados por ellos mismos, por lo que un robot humanoide es un instrumento muy bien adaptado para proporcionar muchos servicios a las personas. Sin embargo, todavía nos encontramos lejos de una producción comercial masiva de humanoides fiables y útiles para la sociedad. Una de las principales razones que justifican la situación actual es el formidable desafío computacional que presentan estos sistemas mecánicos, debido a la complejidad dada por el gran número de restricciones y grados de libertad. En su trabajo Algoritmos de Geometría Diferencial para la Locomoción y Navegación Bípedas de Robots Humanoides.

También se han desarrollado diversos proyectos de programación usando precisamente plataforma Eclipse.

Eclipse es una plataforma universal para la integración de herramientas de desarrollo. Es de arquitectura abierta y extensible basada en componentes añadidos (plug-ins). Inicialmente desarrollado por OTI e IBM evolución en un

proyecto de código abierto. El sistema de ayuda de la plataforma Eclipse define puntos de extensión con los cuales podemos contribuir. Uno de los aspectos menos documentados es la producción de ayuda en forma dinámica (Vanrell, 2006).

El amplio ecosistema de proyectos creado alrededor de la plataforma Eclipse está propiciando que se convierta en una tecnología ampliamente usada para implementar las ideas propuestas en el enfoque del Desarrollo de Software Dirigido por Modelos (DSDM). Son varias las tecnologías disponibles en Eclipse para realizar transformaciones entre modelos (Sánchez, 2007).

Valverde et al (2007) desarrolló un OOWS Suite: Un Entorno de desarrollo para Aplicaciones Web basado en MDA en plataforma Eclipse en el cual menciona que un modelador basado en Eclipse que permite la edición visual de modelos OOWS.

Además de estos trabajos se han realizado muchos proyectos de investigación en los que se utilizan los servomotores Dynamixel en sus diversas versiones.

Rodríguez (2010) desarrolló un proyecto de Tele-operación en Línea de un Humanoide Virtual Mediante la Manipulación de un Humanoide Real, el cual trabaja con servomotores Dynamixel AX-12.

Se sabe que los robots pueden tener diferentes configuraciones en las cuales también se han desarrollado diversos trabajos. Por ejemplo Rodríguez (2015) desarrolló un Procesamiento digital de señales en FPGA para análisis de vibraciones en robots industriales. Entre otros puntos menciona que los robots son la principal herramienta en las líneas de producción de una empresa de manufactura, por esta razón estos robots deben de ser monitoreados constantemente para verificar su desempeño el cual se reflejará en las piezas que estos estén manufacturando.

## 1.2. Descripción del problema

A continuación se describe el problema del que se parte para llevar a cabo este proyecto.

El software utilizado en la mayoría de los trabajos desarrollados con el robot Bioloid Premium y controlador CM-530 es el que maneja el fabricante, RoboPlus. El cual presenta ciertas desventajas cuando se desea programar sólo movimientos.

- Para programar cualquier movimiento es necesario cargar al robot dos archivos entregados por RoboPlus. Lo que hace más difícil y tardado la programación de cualquier movimiento.
- Es difícil de usar debido a que tiene características muy particulares como el hecho de que los movimientos y el código se generan en archivos separados.
- Toma mucho tiempo programar cualquier movimiento en RoboPlus, alrededor de 10 min mínimo.
- Si sólo se desean mover ciertos motores se tiene que especificar en el programa que los demás no se moverán.

## 1.3. Objetivos

En esta sección se describen los objetivos del proyecto.

### 1.3.1. Objetivo General

Desarrollar un software que permita la programación del Robot Bioloid Premium mediante un lenguaje de programación en C.

### 1.3.2. Objetivos Particulares.

- Implementar en el software desarrollado, la selección de los servomotores en los que se implementarán los movimientos, la velocidad del movimiento y la selección del correspondiente ID de cada uno de los servomotores.
- El software se podrá implementar en cualquier configuración del kit Bioloid Premium con controlador CM-530.

- Desarrollar una interfaz de usuario que facilite la programación del Robot Bioloid Premium.

## 1.4. Hipótesis

Con el desarrollo del presente trabajo, se podrá realizar la programación del robot de manera más específica, es decir, teniendo control de selección de servomotores, velocidad e identificación de los mismos.

## 1.5. Justificación

A partir del 2013, la Facultad de Ingeniería e la UAQ ha participado en "FIRA RoboWorld Cup & Congress" de la Federación Internacional de la Asociación de Robot–soccer (FIRA, por sus siglas en inglés), donde se ha posicionado siempre en los tres primeros lugares. En las primeras competencias la programación del robot se realizó directamente en el software de fabricante, la cual se trabaja como programación punto a punto, es decir, se lleva la extremidad del robot al punto deseado y se graba la posición del servomotor. Posteriormente, se realizaron algunos programas en software matemático donde se obtenían las variables articulares a partir de una trayectoria especificada, sin embargo los valores de estas variables se tenían que pasar al software del fabricante. Actualmente, se cuenta con algunos desarrollos de software para la implementación de trayectorias del bípedo, sin embargo, son específicamente para el controlador CM-510 [7], empleando ATMEL Studio [8], para las versiones anteriores del robot Bioloid Premium.

Recientemente, la Facultad de Ingeniería, Campus San Juan del Río, adquirió dos robots Bioloid Premium, sin embargo, estos siendo la nueva versión, cuentan con un controlador CM-530, del cual no se tiene ningún desarrollo previo y además, no funciona la interfaz con el software empleado hasta el momento, ATMEL-Studio. De aquí que, el desarrollo del presente trabajo de tesis permitirá

programar el robot mediante un lenguaje de programación en C, lo que permitirá realizar trabajos posteriores relacionados con la planificación de trayectorias y optimización de las mismas.

# 2. FUNDAMENTACIÓN TEÓRICA

En esta sección se hace una descripción básica de los conceptos teóricos y las herramientas físicas y de programación que se utilizaron para llevar a cabo este proyecto.

## 2.1 Robot Bioloid Premium

La empresa surcoreana ROBOTIS [1] ofrece como uno de sus principales productos el kit educacional de Robótica “BIOLOID Premium", el cual emplea bloques modulares de servomotores de corriente directa (DC, por sus siglas en inglés). Con diferentes grados de libertad véase Figura 2.1. Actualmente, este kit se ofrece con los siguientes componentes:

- Controlador: CM-530 1 pieza.
- Servomotores Dynamixel del modelo AX-12A: 18 piezas.
- Giróscopo de 2 ejes: 1 pieza.
- Sensor infrarrojo: 2 piezas.
- RC-100A (Control remoto): 1 pieza.
- Set de cubierta para humanoide.
- Batería tipo Li-Po (Polímero de Litio) a 11.1 Vdc, 1000 mA/PCM: 1 pieza.
- Cargador de batería: 1 pieza.
- Libro \Quick Start Book" (se incluyen 3 tipos de robots humanoides como ejemplos): 1 pieza.
- Destornillador, Sujeta-cables: 1 pieza.
- Disco de RoboPlus: 1 pieza.

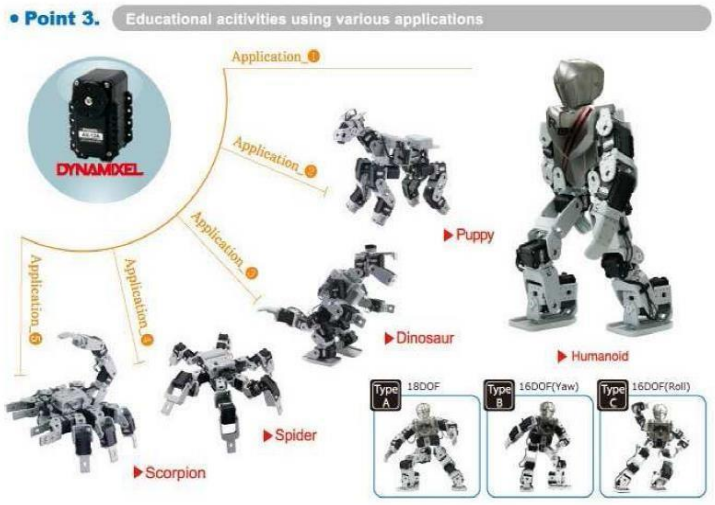



Figura 2.1 Diferentes configuraciones de ensamble para el kit BIOLOID Premium.

### 2.1.1. Controlador CM-530

En esta sección, se describe el controlador CM-530, el cual forma parte del kit Bioloid Premium.

La descripción de los componentes del controlador se muestra en la Figura 2.2.

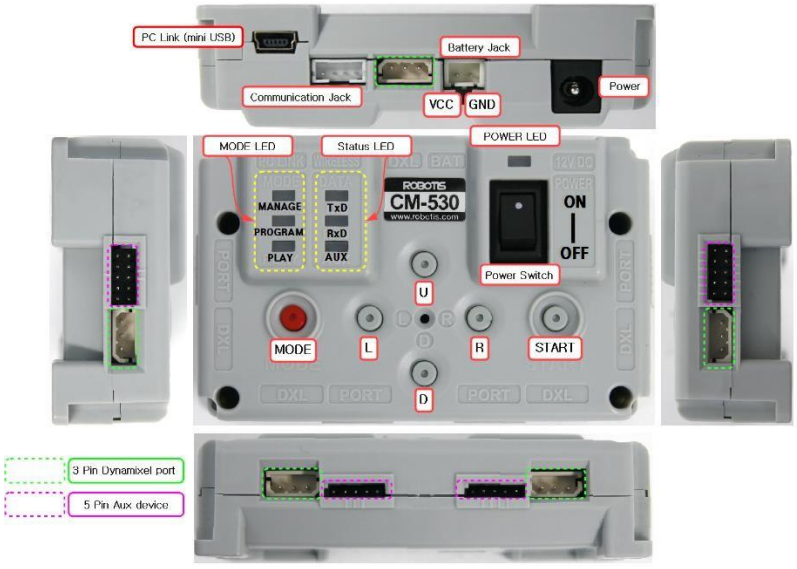



Figura 2.2 Controlador CM-530.

Las funciones de cada componente del controlador se muestran en la Tabla 2.1.

| | |
|---|---|
| PC Link (mini USB) | Se usa para conectar el cable serial hacia el CM-530 y PC vía USB. |
| Communication Device Connection Jack | Usado para la comunicación inalámbrica mediante los módulos ZIG-110A, BT-110A, IR. |
| Battery Jack | Se usa para la conexión de la batería. |
| Power Jack | Se usa para conectar el suplemento de energía SMPS. |
| Power LED | Estado del led encendido o apagado. |
| Power Switch | Usado para encender o apagar el robot. |
| MODE Button | Usado para cambiar el modo de operación del CM-530. |
| START Button | Usado para iniciar la ejecución de los programas. |
| U / L / D / R Button | Usados para propósitos de entrada cuando un programa está corriendo. |
| AX/MX Serise Bus Port | Usado para conectar los AX Dynamixel en una conexión en cadena. |

Tabla 2.1 Descripción de componentes del controlador CM-530.

El uso de este controlador como sistema de procesamiento permite hacer uso de dos ambientes de desarrollo integrado (IDE, por sus siglas en inglés) para la elaboración de algoritmos. Estos son:

- RoboPlus Task, donde se ofrece un estilo de programación muy similar al tipo Lenguaje C.

- Lenguaje C Embebido, a través de la programación en Eclipse Kepler.

### 2.1.2. Servomotor Dynamixel AX-12A

Los servomotores Dynamixel AX-12A están diseñados principalmente para aplicaciones de robótica y mini robótica. Cada servomotor está micro controlado de tal manera que se puede tener acceso a diferentes variables: velocidad rotacional, temperatura, carga y voltaje.

Los actuadores para robot de la serie Dynamixel son actuadores inteligentes y modulares que incorporan un moto-reductor, un motor de DC de precisión y un circuito de control con funcionalidad en red, todo en un solo paquete. A pesar de su tamaño compacto, puede producir un gran torque. También cuenta con la capacidad de detectar y actuar sobre condiciones internas como los cambios en la temperatura interna o el voltaje de alimentación. Algunas características de interés son también:

**Control de precisión:** Posición y velocidad pueden ser controlados con una resolución de 1024 pasos.

**Manejo confiable:** El grado de precisión puede ser ajustado y especificado en el control de posición.

**Retroalimentación:** Retroalimentación para la posición y velocidad angular, así como en el torque de carga.

**Sistema de alarma:** Los actuadores Dynamixel pueden alertar al usuario cuando los parámetros se desvían de los rangos predefinidos, por ejemplo, la temperatura interna, torque, voltaje, etcétera, y pueden manejar el problema automáticamente, al deshabilitar el torque por ejemplo.

**Comunicación:** El cableado es fácil de realizar a través de conectores especiales y son capaces de soportar velocidades de comunicación más allá de 1 Mbps.

**Control distribuido:** Se puede definir la posición, velocidad, confiabilidad y el torque con un solo comando, además el procesador principal puede llegar a controlar varios Dynamixel incluso con muy pocos recursos.

**Plástico ingenieril:** El cuerpo principal del motor está hecho con plástico ingenieril de alta calidad que le permite manejar cargas con un alto torque.

**Ejes con cojinetes:** Un cojinete es usado al final del eje para asegurar una degradación con altas cargas externas.

**LED de estado:** El LED puede indicar un estado de error al usuario.

**Marcos:** Se incluyen marcos bisagra y un marco de montura lateral.

Estos servomotores están protegidos contra sobre voltaje, sobrecalentamiento además de condiciones predefinidas de error. La versión AX-12A es la más nueva del modelo AX-12 con el mismo comportamiento, pero con un diseño externo más avanzado.

**Las especificaciones son las siguientes:**

| | |
|---|---|
| Peso: | 53.5 g (AX-12/AX-12), 54.6g (AX-12A) |
| Dimensiones: | 32 mm * 50mm * 40mm |
| Relación de reducción en el engranaje: | 254:1 |
| Torque máximo: | 15 kg-f*cm (12.0V, 1.5A) |
| Velocidad sin carga: | 114 rpm (12V) |
| Rango de movimiento: | 0 a 300° Giro continuo |
| Temperatura de trabajo: | -5 °C hasta 70 °C |
| Voltaje de trabajo: | 9-12 V (se recomiendan 11.1 V) |
| Señal de comando: | Paquete de datos digitales. |
| Tipo de protocolo: | Comunicación serial asíncrona tipo half duplex (8bits, 1 stop, sin paridad) |
| Identificador: | 0 a 253 |
| Velocidad de comunicación: | Desde 7343bps hasta 1Mbps |
| Retroalimentación: | Posición, temperatura, carga, voltaje de entrada, etc. |
| Material de construcción: | Plástico |

#### 2.1.2.1. Serie de instrucciones para transferir datos al Dynamixel AX-12A

La Serie de instrucciones es el paquete enviado por el controlador principal a las unidades Dynamixel para enviar comandos.

La estructura de la Serie de instrucciones es como la siguiente:

0xFF, 0xFF, ID, LONGITUD, INSTRUCCÓN, PARÁMETRO, PARÁMETRO N, SUMA DE COMPROBACIÓN

Los significados de cada byte de definición de paquetes son como los siguientes:

0xFF 0xFF; Los dos indican el inicio de un paquete entrante (START).

ID; La identificación única de una unidad Dynamixel. Hay 254 valores de ID disponibles, que van desde 0X00 a 0xFD.

Longitud; Longitud del paquete, donde su valor es "Número de parámetros (N) + 2"

INSTRUCCIÓN; La instrucción para el actuador Dynamixel a realizar.

Parámetro N Se utiliza si existe información adicional que se necesita para ser enviados aparte de la propia instrucción.

Suma de verificación; El método de cálculo para la 'Suma de verificación' es como el siguiente.

Suma de verificación = NOT (ID + Longitud+ Instrucción + Parámetro1 + ... + Parámetro N)

Si el valor calculado es mayor que 255, el byte inferior se define como la suma de comprobación valor. ~ Representa la operación lógica NOT.

### 2.1.3. Programación

En la mayoría de los trabajos realizados hasta la fecha con los robots humanoide BIOLOID Premium tipo A existe una común particularidad sobre la

manera de programar los movimientos, la cual consiste en lo siguiente; con el robot conectado a la PC mediante una conexión USB se logra la comunicación entre el robot y el software que maneja el fabricante para programarlo [2], con esto, se pueden tomar lecturas en tiempo real de la posición de cada uno de los motores del robot, entonces se da una posición inicial al robot y se guardan los valores de la posición de cada uno de los motores, los cuales tienen la libertad de girar en un rango de $0^0$ a $300^0$, después se coloca el robot en la siguiente posición, se guardan de nuevo los valores y así sucesivamente hasta terminar con el número total de posiciones, entonces se define un tiempo de ejecución entre cada una de las posiciones y se reproducen todas sucesivamente como un movimiento continuo en cada uno de los motores que tiene el robot, en este caso cada motor actúa como un grado de libertad en el mecanismo completo. Después se guarda el archivo, se descarga el código al robot y éste ejecutará el movimiento cuando se le dé la indicación; ya sea al encenderlo, con un botón, mediante la señal de uno de sus sensores, etc. Esto depende de la forma en que éste haya sido programado.

## 2.2 RoboPlus

Los diferentes robots que se pueden generar con el kit BIOLOID Premium se programa mediante movimientos y comportamientos. Los programas *Motion y Task* sirven para estas dos tareas respectivamente. Para controlar y monitorear el robot en línea desde una laptop o PC, se conecta el ordenador al CM-530. Los programas *Manger y Terminal* pueden interactuar con el CM-530 o con los Dynamixel de esta forma. Un quinto programa: Dynamixel wizard, permite la interacción con los Dynamixel conectados en cadena, sin pasar por el CM-530 con un cable especial. Los programas mencionados forman parte de la suite RoboPlus cuya ventana principal se muestra en la Figura 2.3. A continuación se describe con un poco más de detalle los programas *Task y Motion*.


Figura 2.3 Ventana principal de RoboPlus.

***RoboPlus Task*** maneja el entorno de programación por medio de código; el lenguaje que utiliza es C embebido. Éste permite declarar variables tanto locales como globales, crear ciclos o toma de decisiones y realizar operaciones aritméticas.

Esta herramienta se complementa con el *RoboPlus Motion* dado que por medio de este se pueden mandar llamar las páginas de movimiento para lograr que el robot pueda moverse o pueda realizar una rutina sin necesidad de estar conectado a la computadora dejando al robot en espera de una señal de arranque (botón, sonido, control remoto, etc.).

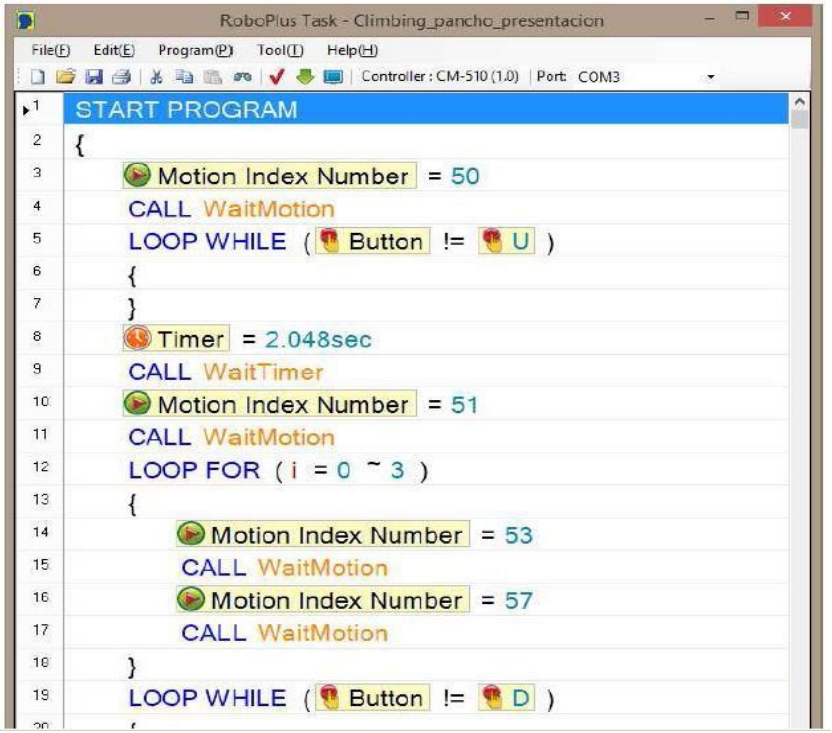



Figura 2.4 Código en RoboPlus Task.

***RoboPlus Motion*** crea y edita archivos que contienen la información que especifica el movimiento del robot. Estos archivos tienen la extensión "*.mtn".

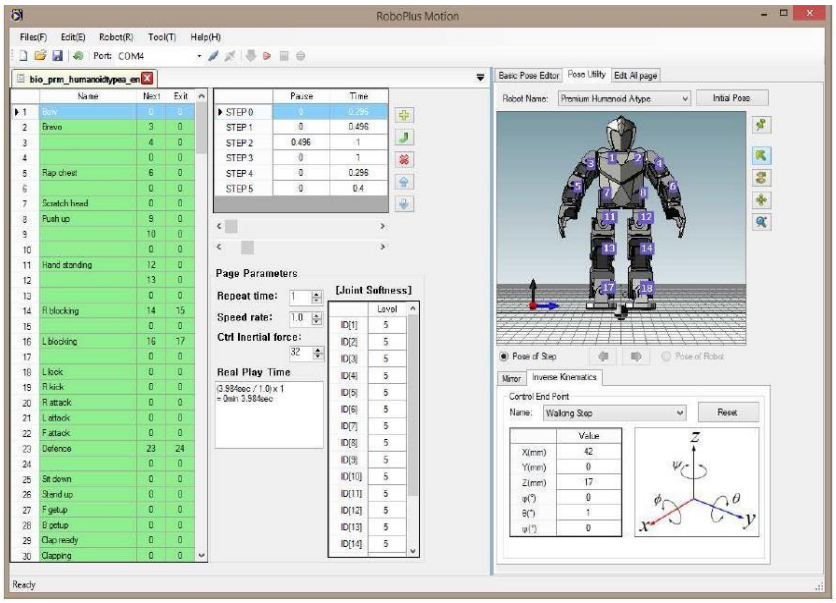



Figura 2.5 Ventana principal de RoboPlus Motion.

Al inicio del archivo se especifica el número de motores activos con que cuenta un robot dado; esto se hace por medio de un conjunto de 25 valores. Para el caso del humanoide BIOLOID Premium Tipo A el archivo *bio prm humanoidtypea en.mt*n (movimientos programados de fábrica) especifica: enable = 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0, lo que indica la presencia de N=18 servomotores Dynamixel con número de ID del 1 al 18. Es importante notar que para otros robots que es posible armar con el kit se contará con un número de motores diferente y que los vectores compliance y pose (que se explican enseguida) tienen 25 valores de los cuales solo se especifican los que corresponden con un valor presente especificado por el enable.

La información que contiene enable puede expresarse matemáticamente como:

$$e_i = \begin{cases} 1 \text{ Motor con ID } i \text{ presente} \\ 0 \text{ Motor sin ID } i \text{ presente} \end{cases}$$

$\text{con } i \in \{1, 2 \ldots, 25\}.$

El número máximo de motores presentes en caso específico de la lógica de programación del kit Bioloid es 25 y el vector enable denotado por e tiene 25 valores que indican si el motor con la ID correspondiente al subíndice está conectado a la unidad de control CM-530.

El movimiento de un robot se especifica mediante páginas, que están delimitadas por las palabras page *begin y page end*. El límite de páginas es de 255 y se enumeran consecutivamente a partir del 0. El archivo “*.mtn" contiene las 255 páginas aunque pueden existir páginas vacías. La página contiene el campo *name* cuyo valor puede ser nulo o una cadena de caracteres que especifique el nombre de la página, un campo compliance que especifica la rigidez que tendrá cada servomotor correspondiente al índice i y que puede variar de 1 a 7; un valor típico utilizado en los movimientos pregrabados por el fabricante es 5. El campo *play param* especifica 5 parámetros del movimiento contenido en la página. Estos 5 parámetros se especifican a continuación:

**Next:** un número (de 1 a 255) correspondiente a una página existente dentro del mismo archivo y que será reproducida enseguida de la página actual. En caso de que este parámetro sea 0, el movimiento del robot se detiene al finalizar la página en curso.

**Exit:** este parámetro indica la página en la que finaliza el movimiento al presionar stop.

**Repeat Time**: es el número de veces que se repite el movimiento especificado en una página.

**Speed rate:** indica la rapidez con la que el movimiento descrito en una página será realizado.

**Inertial Force:** es un parámetro que incrementa o disminuye la velocidad, manipulando la aceleración.

El campo paso contiene de 1 a 7 vectores de 25 valores que indican las posiciones deseadas de los motores (de 0 a 1024), y dos parámetros:

**Pause:** Es el tiempo que durará la pose del robot, es decir el tiempo que se mantendrá estático el robot una vez alcanzadas las posiciones angulares de los motores.

**Time:** El tiempo que tomará para que todos los motores lleguen a la posición deseada.

Los valores de pausa y tiempo correspondiente a cada vector presente en un paso los cuales se suman junto con los parámetros *repeat time y speed rate* permitiendo calcular el tiempo real que llevará cada página.

Robotis, propone esta estructura de archivo para almacenar los posibles movimientos del robot. Para generar estos archivos se puede trabajar en línea con el robot real o solo con los valores de los ángulos y una visualización 3D del prototipo de robot adecuado (humanoide en este caso).

## 2.3 Herramientas de programación con Lenguaje C

A continuación se describen las diferentes herramientas de programación con Lenguaje C empleadas en el presente trabajo de tesis, para más información véase [“Lenguaje C”, **Bonet Esteban**].

### 2.3.1. Lenguaje C

El lenguaje C es un lenguaje para programadores en el sentido de que proporciona una gran flexibilidad de programación y una muy baja comprobación de incorrecciones, de forma que el lenguaje deja bajo la responsabilidad del programador acciones que otros lenguajes realizan por sí mismos. Así, por ejemplo, C no comprueba que el índice de referencia de un vector (llamado array en la literatura informática) no sobrepase el tamaño del mismo; que no se escriba en zonas de memoria que no pertenecen al área de datos del programa, etc. El lenguaje C es un lenguaje estructurado, en el mismo sentido que lo son otros lenguajes de programación tales como el lenguaje Pascal, el Ada o el Modula-2, pero no es estructurado por bloques, o sea, no es posible declarar subrutinas (pequeños trozos de programa) dentro de otras subrutinas, a diferencia de como sucede con otros lenguajes estructurados tales como el Pascal.

### 2.3.2. Tipos de datos

En C, toda variable, antes de poder ser usada, debe ser declarada, especificando con ello el tipo de dato que almacenara. Toda variable en C se declara de la forma:

< tipo de dato > < nombre de variable > [nombre de variable];

En C existen cinco tipos de datos según puede verse en la tabla siguiente:

| **Tipo de dato** | **Descripción.** |
|---|---|
| char | Carácter o entero pequeño (byte) |
| int | Entero |
| float | Punto flotante |

| | |
|---|---|
| double | Punto flotante (mayor rango que *float*) |
| void | Sin tipo (uso especial) |

Tabla 2.2 Tipos de datos.

### 2.3.3. Operadores aritméticos

Los operadores aritméticos existentes en C son, ordenados de mayor a menor precedencia:

| | |
|---|---|
| **++** | Incremento |
| -- | Decremento |
| + | Suma |
| - | Resta |
| * | Multiplicación |
| / | División |
| % | Módulo |

Tabla 2.3 Operadores aritméticos.

### 2.3.4. Operadores relaciónales y lógicos.

Los operadores relaciónales y lógicos de C, ordenados de mayor a menor prioridad son:

| | | | |
|---|---|---|---|
| ! | NOT | == | Igual |
| > | Mayor que | != | No igual |
| >= | Mayor o igual | && | AND |
| < | Menor que | \|\| | OR |
| <= | Menor o igual | | |

Tabla 2.4 Operadores relacionales y lógicos.

### 2.3.5. Operadores sobre bits

El lenguaje C posee operadores que actúan a nivel de bits sobre los datos, estos operadores son:

| Operador | Nombre | Operación |
|---|---|---|
| ~ | NOT | Complemento a uno (NOT) |
| << | Desplazamiento izquierda | Desplazamiento izquierda |
| >> | Desplazamiento derecha | Desplazamiento derecha |
| & | AND | Y |
| ^ | XOR | O exclusivo (XOR) |
| \| | OR | O |

Tabla 2.5 Operadores sobre bits.

### 2.3.6. Sentencias de control y Bucles

El concepto de sentencia en C es igual que el de otros muchos lenguajes. Por sentencia se entiende en C cualquier instrucción simple o bien, cualquier conjunto de instrucciones simples que se encuentren encerradas entre los caracteres {y}, que marcan respectivamente el comienzo y el final de una sentencia.

La forma general de la **sentencia if** es:
*if (condición) Sentencia; else Sentencia;*

Siendo el *else* opcional. Si la condición es verdadera se ejecuta la sentencia asociada al *if,* en caso de que sea falsa la condición se ejecuta la sentencia asociada al *else* (si existe el else).

La sintaxis del **bucle for** es:

*for (inicialización,condición,incremento) sentencia;*

En primer lugar, conviene destacar el hecho de la gran flexibilidad del *bucle for* de C. En C, el *bucle for* puede no contener inicialización, condición o incremento, o incluso pueden no existir dos e incluso las tres expresiones del

bucle. El bucle for se ejecuta siempre que la condición sea verdadera, es por ello que puede llegar a no ejecutarse.

La sintaxis del **bucle while** es:

*while (condición) sentencia;*

Donde la sentencia puede no existir (sentencia vacía), pero siempre debe existir la condición. El *bucle while* se ejecuta mientras la condición sea verdad.

Al contrario que los bucles for y while que comprueban la condición en lo alto de la misma, el **bucle do/while** comprueba la condición en la parte baja del mismo, lo cual provoca que el bucle se ejecute como mínimo una vez. La sintaxis del bucle do/while es:

*do sentencia; while (condición);*

El bucle do/while se ejecuta mientras la condición sea verdad.

Las sentencias de control **break y continue** permiten modificar y controlar la ejecución de los bucles anteriormente descritos. La sentencia break provoca la salida del bucle en el cual se encuentra y la ejecución de la sentencia que se encuentra a continuación del bucle. La sentencia continue provoca que el programa vaya directamente a comprobar la condición del bucle en los bucles while y do/while, o bien, que ejecute el incremento y después compruebe la condición en el caso del bucle for.

### 2.3.7. Funciones

Las funciones son simulares a las de cualquier otro lenguaje, pero, tal y como citamos en la introducción, al no ser un lenguaje estructurado por

bloques, no es posible declarar funciones dentro de otras funciones. El formato general de una función de C es:

*tipo nombre(lista de parámetros)*
*{*
*cuerpo de la función }*

### 2.3.8. El preprocesador

En un programa escrito en C, es posible incluir diversas instrucciones para el compilador dentro del código fuente del programa. Estas instrucciones dadas al compilador son llamadas directivas del preprocesador y, aunque realmente no son parte del lenguaje C, expanden el ámbito del entorno de programación de C.

El preprocesador, definido por el standard ANSI de C, contiene las siguientes directivas:

| | | | |
|---|---|---|---|
| #if | #ifdef | #ifndef | #else |
| #elif | #endif | #include | #define |
| #undef | #line | #error | #pragma |

Tabla 2.6 Directivas del preprocesador.

**Directiva #define;** La directiva define se usa para definir un identificador y una cadena que el compilador sustituirá por el identificador cada vez que se encuentre en el archivo fuente. El standard ANSI llama al identificador "nombre de macro" y al proceso de sustitución "sustitución de macro". Por ejemplo:

*#define TRUE 1 #define FALSE 0*

El compilador, cada vez que vea el identificador *TRUE*, lo sustituirá por el valor *1*, e igual con *FALSE*. El uso más común de la directiva *#define* es la definición de valores constantes en el programa, tamaños de arrays, etc.

Una característica que posee la directiva *#define* es que el "nombre de macro" puede contener argumentos. Cada vez que el compilador encuentra el "nombre de macro", los argumentos reales encontrados en el programa reemplazan los argumentos asociados con el nombre de macro.

**Directiva #include;** La directiva *#include* fuerza al compilador a incluir otro archivo fuente en el archivo que tiene la directiva *#include*, y a compilarlo. El nombre del archivo fuente a incluir se colocara entre comillas dobles o entre paréntesis de ángulo.

Los archivos incluidos mediante *#include* pueden a su vez poseer otras directivas *#include*. La diferencia existente entre encerrar el archivo entre paréntesis de ángulo o entre comillas dobles, es que, en el primer caso, se busca el archivo en los directorios de la línea de órdenes de compilación, y después en los directorios standard de C, pero nunca en el directorio de trabajo; y en el segundo caso el primer sitio donde se busca el archivo a incluir es en el directorio actual de trabajo, pasándose, caso de no haber sido encontrado, a buscar en los mismos sitios que el caso anterior.

## 2.4 Interfaz Eclipse Kepler

Es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo,

también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent o Azureus.

Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas.

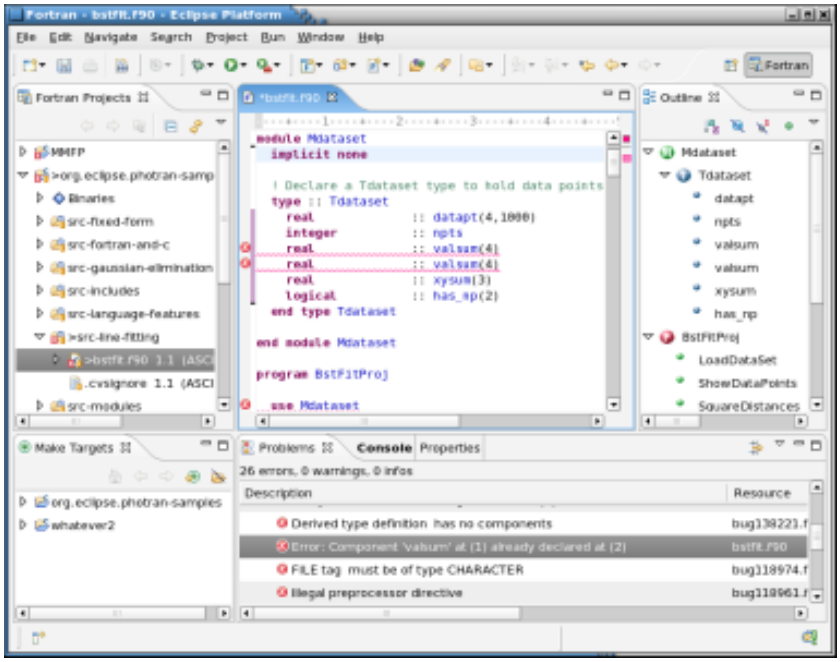



Figura 2.6 Interfaz Eclipse Kepler.

### 2.4.1. Arquitectura Eclipse

El entorno de desarrollo integrado (IDE) de Eclipse emplea módulos (en inglés plug-in) para proporcionar toda su funcionalidad al frente de la plataforma de cliente enriquecido, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. Adicionalmente a permitirle a Eclipse extenderse usando otros lenguajes de programación como son C/C++ y Python, permite a Eclipse trabajar con lenguajes para procesado de texto como LaTeX, aplicaciones en red como Telnet y Sistema de gestión de base de datos. La arquitectura plug-in permite escribir cualquier extensión deseada en el ambiente, como sería Gestión de la configuración. Se provee soporte para Java y CVS en el SDK de

Eclipse. Y no tiene por qué ser usado únicamente con estos lenguajes, ya que soporta otros lenguajes de programación.

La definición que da el proyecto Eclipse acerca de su software es: "una especie de herramienta universal - un IDE abierto y extensible para todo y nada en particular".

En cuanto a las aplicaciones clientes, Eclipse provee al programador con frameworks muy ricos para el desarrollo de aplicaciones gráficas, definición y manipulación de modelos de software, aplicaciones web, etc. Por ejemplo, GEF (*Graphic Editing Framework - Framework* para la edición gráfica) es un plugin de Eclipse para el desarrollo de editores visuales que pueden ir desde procesadores de texto *wysiwyg* hasta editores de diagramas UML, interfaces gráficas para el usuario (GUI), etc. Dado que los editores realizados con GEF "viven" dentro de Eclipse, además de poder ser usados conjuntamente con otros plug-in, hacen uso de su interfaz gráfica personalizable y profesional.

El SDK de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java. Esto permite técnicas avanzadas de refactorización y análisis de código. Mediante diversos plug-in estas herramientas están también disponibles para otros lenguajes como C/C++ (Eclipse CDT) y en la medida de lo posible para lenguajes de script como PHP o JavaScript.

### 2.5. Visual Studio

En esta sección se hace una breve descripción de la plataforma de programación Visual Studio, para obtener más información visitar [Visual Studio Product Updates Blog (2014-11-12). "Visual Studio 2015 Preview".]

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas operativos Windows. Soporta múltiples lenguajes de programación tales como C++, C#, Visual Basic

.NET, F#, Java, Python, Ruby, PHP; al igual que entornos de desarrollo web como ASP.NET MVC, Django, etc., a lo cual sumarle las nuevas capacidades online bajo Windows Azure en forma del editor Monaco.

Visual Studio permite a los desarrolladores crear sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión .NET 2002). Así se pueden crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos, consolas, etc.

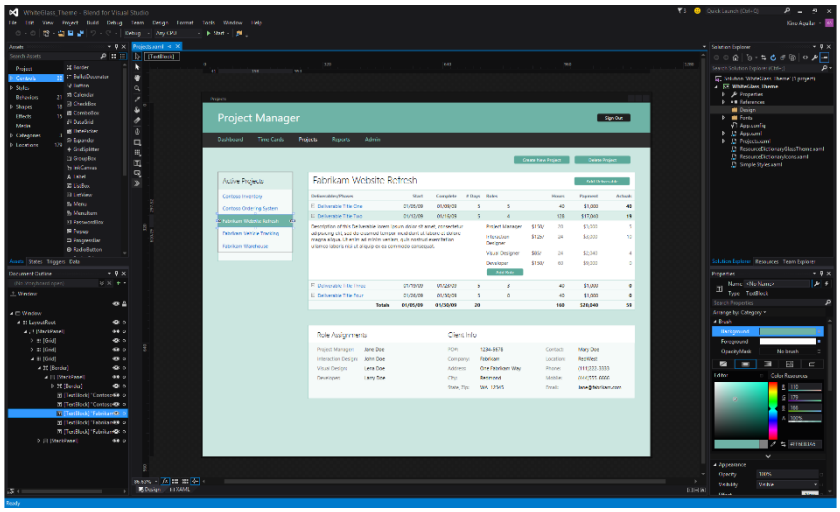



Figura 2.7 Interfaz de Visual Studio.

# 3. DESCRIPCIÓN DEL SOFTWARE

En esta sección se explica el proceso seguido para llevar a cabo este proyecto. Para poder comenzar con el desarrollo del software se deben tener instalados correctamente el programa Eclipse Kepler con los complementos para lenguaje C/C++ (Véase Apéndice 1).

La metodología que se sigue para el desarrollo del presente trabajo de Tesis se muestra en la Figura 3.1.

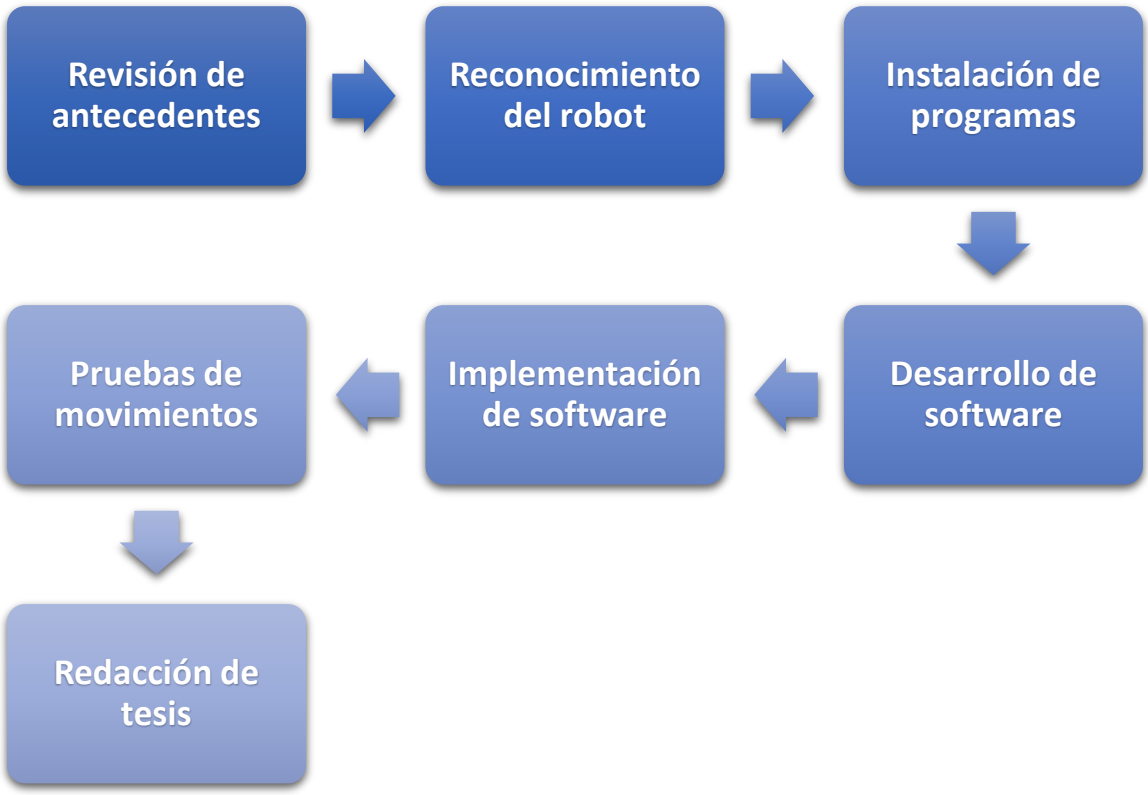



Figura 3.1 Diagrama a boques de la metodología para llevar a cabo el proyecto de tesis.

Cada etapa presentada en la Figura 3.1 se describe a continuación.

**Revisión de antecedentes:** en esta parte se lleva a cabo la búsqueda de información que esté relacionada con el presente trabajo de tesis. Entre los temas más importantes están; aplicaciones en robots humanoide, trabajos desarrollados en plataforma Eclipse y trabajos desarrollados utilizando servomotores Dynamixel AX-12A.

**Reconocimiento del robot:** en esta sección se lleva a cabo el análisis del funcionamiento de cada componente del robot Bioloid Premium, incluyendo; servomotores Dynamixel AX-12A, controlador CM-530 y los sensores adicionales del kit, como lo son; el sensor giróscopo, sensor de infrarrojos, sensor de distancia DMS y control remoto, estos sensores se muestran en la Figura 3.2. Dentro de las características más importantes que se deben analizar en el controlador es la velocidad a la que se transfieren los datos y el tipo de comunicación (Véase sección 2.1.1).

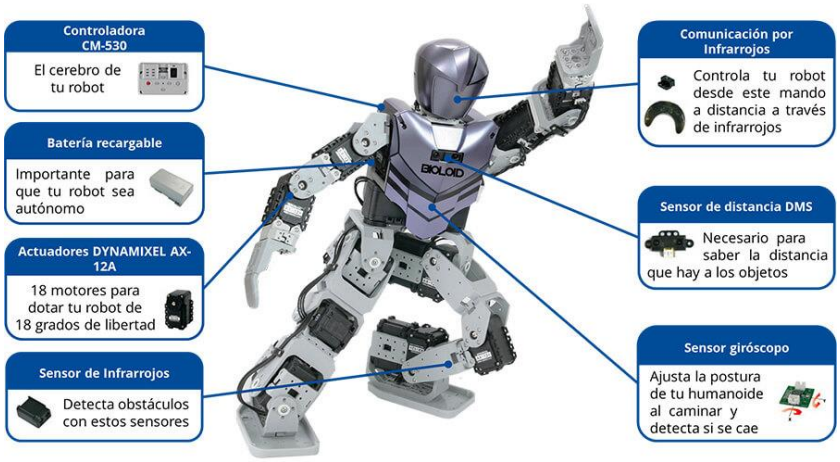



Figura 3.2 Componentes del kit de robótica Bioloid Premium.

**Instalación de programas:** se realiza la correcta instalación de los programas utilizados; plataforma Eclipse Kepler, RoboPlus, Visual Studio (Véase Apéndice 1).

**Desarrollo de software:** con base en el objetivo principal del presente proyecto de tesis se crea un nuevo proyecto en plataforma Eclipse Kepler que incluye las librerías necesarias para los servomotores Dynamixel AX-12A y para el

controlador CM-530, en el que se describe el código en Lenguaje C/C++ para dejar un programa sencillo de usar en el que se pueda programar movimientos al Robot Bioloid Premium de una forma rápida y sencilla. Esta parte de la metodología es la parte medular del presente trabajo desarrollado.

**Implementación de software:** Una vez terminado el programa se procede a cargarlo al robot, para ello se construye el proyecto dando clic en el botón "*Build*" de la interfaz de Eclipse y esto genera un archivo *.hex* que es el archivo que contiene el código máquina[1] para el controlador. Ahora se abre el programa RoboPlus en la parte de "*Terminal*" y se conecta el robot por medio del puerto serial USB, entonces se da clic en "*transmitir archivo*" y se busca el archivo *.hex* para que finalmente se dé clic en "*abrir*" y listo, el programa creado se ha cargado al controlador. Este proceso se explica en el diagrama de la Figura 3.3.

1 El lenguaje máquina es el único lenguaje que puede ejecutar una computadora, es específico en cada arquitectura, es un código que es interpretado directamente por el microprocesador, está compuesto por un conjunto de instrucciones ejecutadas en secuencia que representan acciones que la máquina podrá tomar.

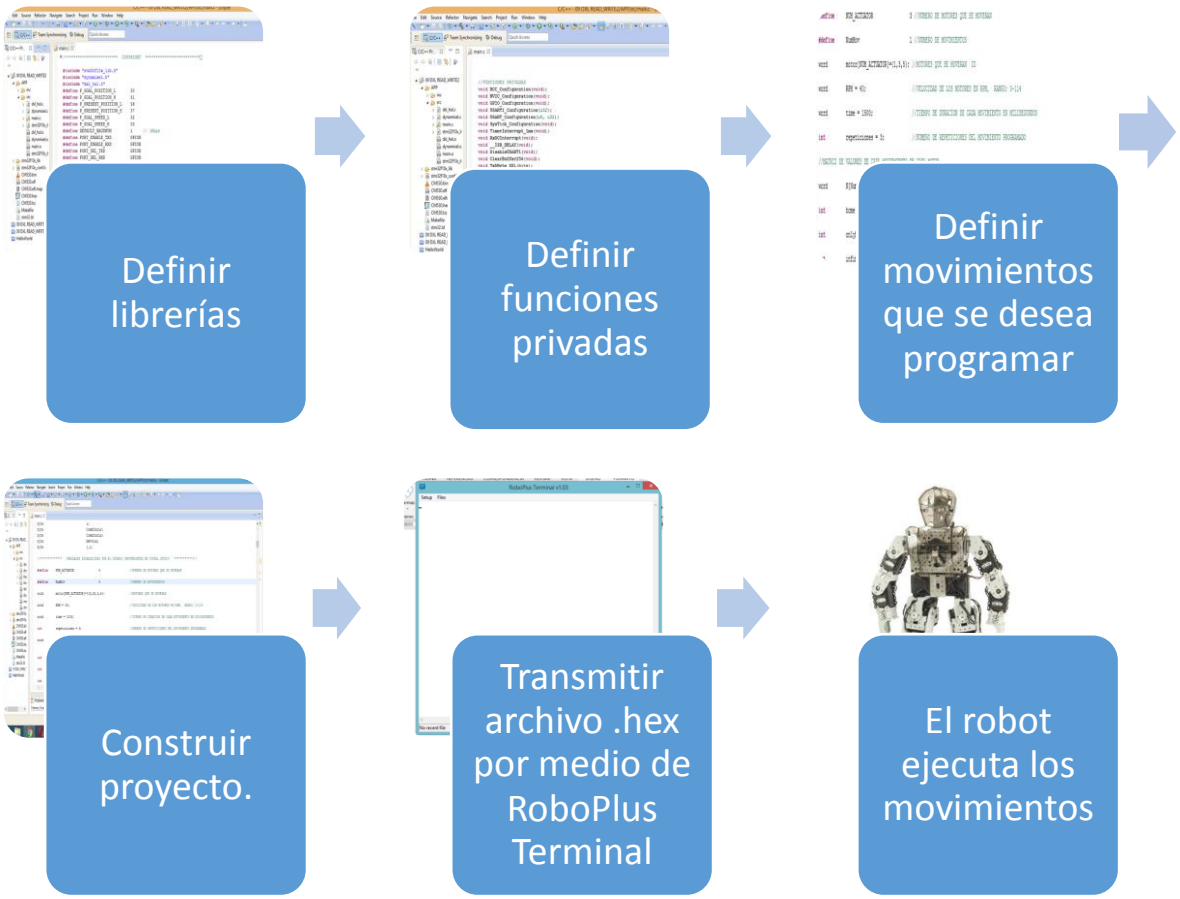



Figura 3.3 Diagrama a bloques de la implementacion del software.

Cada etapa del diagrama a bloques mostrado en la Figura 3.3 se describe más a detalle a partir de la sección 3.1.

**Pruebas de movimientos:** se hacen varias pruebas en las que se dan valores de entrada al programa desarrollado y se le carga al robot para que finalmente éste ejecute los movimientos de acuerdo a los valores descritos por el usuario.

**Redacción de Tesis:** a lo largo del desarrollo del presente trabajo de Tesis se redacta y se toma evidencia de lo que se hace para que al final se pueda sintetizar.

## 3.1. Descripción de librerías

Las librerías de un programa son un conjunto de funciones, que a diferencia de un programa de Lenguaje C, no contiene la función "*main()*" [9].

En el presente proyecto de Tesis las librerías principales utilizadas son las que se muestran a continuación.

- *#include "stm32f10x_lib.h"*
- *#include "dynamixel.h"*
- *#include "dxl_hal.h"*

Se utilizaron estas librerías desarrolladas por el fabricante del controlador [1] porque permiten la correcta comunicación entre el código descrito en Lenguaje C y el controlador CM-530, y con los actuadores del Robot, los motores Dynamixel AX-12A.

Las librerías *"dynamixel.h"* y *"dxl_hal.h"* se utilizan para poder enviar y recibir información de los motores Dynamixel AX-12A, dentro de ellas están definidas varias funciones que permiten leer y escribir datos de acuerdo al tipo de comunicación de los actuadores Dynamixel AX-12A (Véase sección 2.2.1.1). A continuación se muestran las funciones más utilizadas en este proyecto que contienen estas librerías.

- *dxl_write_word (a, b, c);*

Esta función sirve para mandar datos a los motores. El primer parámetro (a) indica el ID del motor al que se envía la información, el segundo (b) indica la acción que se desea ejecutar en el actuador. El tercero (c) indica la magnitud de la acción que se desea ejecutar.

- *dxl_read_word (a, b);*

Esta función sirve para leer valores de los motores. El primer parámetro (a) indica el ID del motor al que se envía la información, el segundo (b) indica el parámetro que se desea leer del actuador.

Posterior a declarar estas librerías se definen las constantes globales usando las directivas del preprocesador (Véase sección 2.3.8) como se muestra en la Tabla 3.1.

| **Directiva** | **Nombre** | **Valor** |
|---|---|---|
| #define | P_GOAL_POSITION_L | 30 |
| #define | P_GOAL_POSITION_H | 31 |
| #define | P_PRESENT_POSITION_L | 36 |
| #define | P_PRESENT_POSITION_H | 37 |
| #define | P_GOAL_SPEED_L | 32 |
| #define | P_GOAL_SPEED_H | 33 |
| #define | DEFAULT_BAUDNUM | 1 |
| #define | PORT_ENABLE_TXD | GPIOB |
| #define | PORT_ENABLE_RXD | GPIOB |
| #define | PORT_DXL_TXD | GPIOB |
| #define | PORT_DXL_RXD | GPIOB |
| #define | P_MOVING | 46 |
| #define | P_MOVING | 46 |
| #define | PIN_ENABLE_TXD | GPIO_Pin_4 |
| #define | PIN_ENABLE_RXD | GPIO_Pin_5 |
| #define | PIN_DXL_TXD | GPIO_Pin_6 |
| #define | PIN_DXL_RXD | GPIO_Pin_7 |
| #define | PIN_PC_TXD | GPIO_Pin_10 |
| #define | PIN_PC_RXD | GPIO_Pin_11 |
| #define | USART_DXL | 0 |
| #define | USART_PC | 2 |
| #define | Word | u16 |
| #define | Byte | u8 |

Tabla 3.1 Definiciones globales del código.

Este tipo de declaración de valores permite un panorama más claro de lo que se está haciendo en el código, ya que para poner los valores que se encuentran

en la columna del lado derecho sólo se escribe el nombre de la columna de en medio y se hace la sustitución automáticamente.

### 3.2. Definición de variables privadas

Las variables privadas sólo pueden ser accedidas desde dentro de la misma clase. Todo intento de llamarlas desde la una instancia de la misma es en vano. Mantener variables/funciones privadas permiten tener un mayor control sobre la clase, sobre el modo como procesa sus métodos, como maneja sus variables, etc. [10].

Las funciones privadas utilizadas en el presente proyecto de Tesis se describen en la Tabla 3.1. Estas funciones fueron desarrolladas por la empresa fabricante del controlador [1].

| | |
|---|---|
| 1 | void RCC_Configuration(void); |
| 2 | void NVIC_Configuration(void); |
| 3 | void GPIO_Configuration(void); |
| 4 | void USART1_Configuration(u32); |
| 5 | void USART_Configuration(u8, u32); |
| 6 | void SysTick_Configuration(void); |
| 7 | void TimerInterrupt_1ms(void); |
| 8 | void RxD0Interrupt(void); |
| 9 | void __ISR_DELAY(void); |
| 10 | void DisableUSART1(void); |
| 11 | void ClearBuffer256(void); |
| 12 | void TxDByte_DXL(byte); |
| 13 | byte RxDByte_DXL(void); |
| 14 | void TxDString(byte*); |
| 15 | void TxDWord16(word); |
| 16 | void TxDByte16(byte); |
| 17 | void TxDByte_PC(byte); |
| 18 | void mDelay(u32); |
| 19 | void StartDiscount(s32); |
| 20 | byte CheckTimeOut(void); |

Tabla 3.2 Funciones privadas utilizadas en el proyecto.

Estas funciones tienen la función de configurar internamente el controlador de acuerdo a lo que se esté programando. Para desarrollar cualquier proyecto con este controlador en esta plataforma son indispensables.

### 3.2.1. Descripción del código

En esta sección se explican las partes más importantes del código descrito en Lenguaje C que permiten el correcto funcionamiento del programa, esta es la parte del proyecto en la que se alcanzan los objetivos planteados.

El funcionamiento del programa depende solamente de los valores de entrada que son ingresados directamente por el usuario como se muestra en la Figura 3.3.

```
1  #define     NUM_ACTUATOR              3 //NUMERO DE MOTORES QUE SE MOVERAN
2  #define     NumMov                    1 //NUMERO DE MOVIMIENTOS
3  word        motor[NUM_ACTUATOR]={1,3,5}; //MOTORES QUE SE MOVERAN  ID
4  word        RPM = 40;                  //VELOCIDAD DE LOS MOTORES EN RPM,  RANGO; 0-114
5  word        time = 1500;               //TIEMPO DE DURACION DE CADA MOVIMIENTO EN MILISEGUNDOS
6  int         repeticiones = 3;          //NUMERO DE REPETICIONES DEL MOVIMIENTO PROGRAMADO
   //MATRIZ DE VALORES DE CADA MOVIEMIENTO DE CADA MOTOR
7  word        N[NumMov][NUM_ACTUATOR]={512,512,512};
8  int         home = 0;                  //INDICA SI EL MOVIMIENTO INICIARA CON LA POSICION DE HOME (0-1)
9  int         onlyhome = 0;              //INDICA SI EL ROBOT SOLO TOMARA LA POSICION DE HOME Y NADAMAS (0-1)
10 int         infinito = 0;              //INDICA SI EL NUMERO DE REPETICIONES DEL MOVIMIENTO SERA INFINITO (0-1)
```

Figura 3.4 Valores de entrada descritos por el usuario.

La Figura 3.4 muestra los parámetros de configuración con los que el programa desarrolla el código máquina que se carga al controlador del robot, de tal forma que éste ejecute los movimientos esperados. A continuación se describe cómo se lleva a cabo este proceso.

Una vez que están definidos los valores de entrada, el código realiza la interpretación éstos, de tal forma que el controlador del robot reciba las instrucciones correctas para ejecutar los movimientos deseados.

Se crearon dos funciones principales para el desarrollo de este proyecto.

- *void MovCiclico(int counter)*
- *Home(void)*

En la función "*MovCiclico*" se recibe la variable "*counter*" tipo entero (Véase sección 2.3.2) que determina el número de repeticiones que ejecuta robot con los movimientos programados. En esta función se describen los movimientos que ejecuta el robot de forma sincronizada, esto es, cada motor toma la posición asignada en un mismo periodo de tiempo.

```
void MovCiclico(int counter)
{
    word   SPEED   = RPM*256/57;   4
                for( i=0; i<NUM_ACTUATOR; i++ )   1
                {
                        id[i] = motor[i];
                }
```

En esta parte del código se declara la función y dentro de ésta se convierte el valor de velocidad angular determinado por el usuario a un valor con una rango de 0 a 1024, es el rango de valores que debe recibir el controlador para velocidad angular de los motores. También se asignan los valores a un vector auxiliar creado para guardar los valores de los motores que el usuario desea mover.

```
byte j =0;
    byte cuenta= 0;
```

```
while(1)
{
        bMoving = dxl_read_byte( BROADCAST_ID, P_MOVING );

dxl_write_word(BROADCAST_ID,P_GOAL_SPEED_L,SPEED);
        dxl_set_txpacket_id(BROADCAST_ID);
        dxl_set_txpacket_instruction(INST_SYNC_WRITE);
        dxl_set_txpacket_parameter(0, P_GOAL_POSITION_L);
        dxl_set_txpacket_parameter(1, 2);
```

En esta parte de la función primero se declaran dos variables auxiliares tipo *Byte "counter"* y *"j"* que sirven para realizar el conteo de las repeticiones de los movimientos programados al robot por el usuario. Después se describe el código para el envío de los parámetros correspondientes a los Dynamixel AX-12A para el movimiento sincronizado de motores (Véase sección 2.2.1) en el que se utilizan algunos de los valores constantes declarados con las directivas del preprocesador (Véase sección 2.3.8).

```
    for( i=0; i<NUM_ACTUATOR; i++ )   1
    {
dxl_set_txpacket_parameter(2+3*i, id[i]);
dxl_set_txpacket_parameter(2+3*i+1, dxl_get_lowbyte(N[j][i]));
dxl_set_txpacket_parameter(2+3*i+2, dxl_get_highbyte(N[j][i]));   7
                                                    }
                                        mDelay(time);
dxl_set_txpacket_length((2+1)*NUM_ACTUATOR+4);   1
                                                    dxl_txrx_packet();
```

En esta parte de la función se asignan directamente los valores de posición que se definen dentro de una matriz "*N[j][i]*" así como el número de motores que se

van a mover "*NUM_ACTUATOR*", y se define el tiempo de duración de cada movimiento "*time*". Todos estos parámetros están definidos directamente por el usuario, y ésta es la parte del programa donde se describe la interpretación de lo que el usuario desee programar al robot.

```
cuenta++;
        if(j==NumMov-1)
                j=0;
        else
                j++;
        if(cuenta>=counter*NumMov+1)
                {
        break;
                }
```

Por último se describe la parte de la función en la que se define el número de veces que se repetirá el movimiento. Éste, también está definido por el usuario, como se observa en la parte del último *"if"* donde se condiciona el número de veces que se repite el bucle *"While"* de acuerdo a la variable auxiliar "*counter*" y a la variable definida por el usuario "*NumMov*".

Esta es una de las dos funciones principales que describen el funcionamiento de este proyecto la otra es la que se describe en la sección 3.3 que es la función donde se definen los valores para la posición de "*home".*

### 3.3. Definición de valores para posición de HOME para configuración de Humanoide Tipo A.

Dentro del programa el usuario tiene la opción de mover el robot en una posición inicial "*Home*" (Véase sección 3.2.1), ésta es una posición inicial en la que el robot se encuentra de pie y listo para caminar como se muestra en la Figura 3.5.

Para esto se definen valores de posición a cada uno de los actuadores del Bioloid Premium Tipo A, sabiendo que el actuador toma una posición que puede variar dentro de su rango de movimiento ($0^0$-$300^0$) ejecutándose con un valor declarado en el programa en un rango (0-1023) que es equivalente a la posición que toma.

En el presente proyecto de Tesis se utilizó un Robot en la configuración tipo A, con dos grados de libertad más ya que se le incorporó un sistema de visión artificial, por lo cual en la Tabla 3.3 se muestran los 20 valores asignados a los 20 motores que tiene en total el Robot.

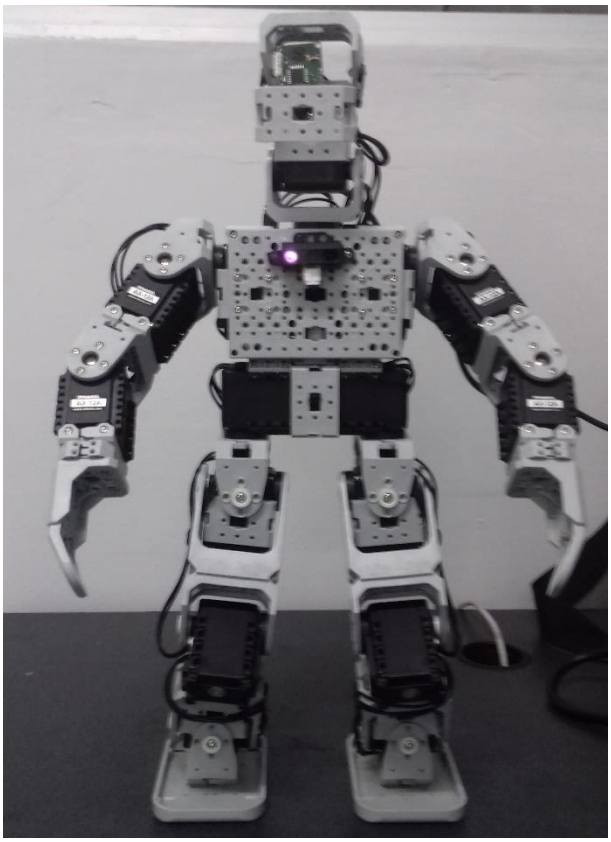


Figura 3.5 Posición de Home para el robot Bioloid Tipo A.

Para lograr esta posición se tiene que asignar los valores mostrados en la Tabla 3.3 a los actuadores correspondientes del robot.

| **ID** | **Nombre** | | **Valor** | |
|---|---|---|---|---|
| *1* | T1 | = | 235 | ; |

| | | | | |
|---|---|---|---|---|
| *2* | T2 | = | 788 | ; |
| *3* | T3 | = | 279 | ; |
| *4* | T4 | = | 744 | ; |
| *5* | T5 | = | 462 | ; |
| *6* | T6 | = | 561 | ; |
| *7* | T7 | = | 358 | ; |
| *8* | T8 | = | 666 | ; |
| *9* | T9 | = | 513 | ; |
| *10* | T10 | = | 522 | ; |
| *11* | T11 | = | 342 | ; |
| *12* | T12 | = | 681 | ; |
| *13* | T13 | = | 241 | ; |
| *14* | T14 | = | 781 | ; |
| *15* | T15 | = | 646 | ; |
| *16* | T16 | = | 377 | ; |
| *17* | T17 | = | 513 | ; |
| *18* | T18 | = | 522 | ; |
| *19* | T19 | = | 512 | ; |
| *20* | T20 | = | 512 | ; |

Tabla 3.3 Valores de posición de los respectivos motores para que el robot adopte la posición de "*Home*".

La forma en que se describe el posicionamiento de un motor del robot es como el siguiente ejemplo (Véase sección 2.1.2 y sección 3.1).

```
dxl_write_word( 1, P_GOAL_SPEED_L, 400);
dxl_write_word( 1, P_GOAL_POSITION_L, T1);
```

En la cual el primer renglón le define al Dynamixel la velocidad a la que se moverá en un rango de 0 – 512, es la máxima cantidad de valores que se pueden representar con 9 bits, que corresponde proporcionalmente a una velocidad en un rango de operación de 0 – 114 RPM (revoluciones por minuto).

El segundo renglón define el valor que tomará de posición el motor 1 en este caso y como se puede ver se le asigna el valor declarado como Home para el actuador 1 que corresponde a la variable T1. Por lo que dentro del programa se describe una función con el nombre "*Home*" y cada vez que se mande llamar el

robot tomará ésta posición con la velocidad establecida por el usuario (Véase sección 2.3.7).

### 3.4. Descripción de interfaz de usuario

La interfaz gráfica de usuario, conocida también como GUI (*graphical user interface),* es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador.

El sistema de definición de datos para programación de Robot desarrollado en este trabajo requiere ingresar sólo valores numéricos, para hacerlo de manera sencilla se desarrolló una interfaz de usuario de tal manera que el usuario final no requiere conocimientos de programación. A continuación, se describe la ventana que conforma la interfaz de usuario.

Para el desarrollo de esta interfaz se sigue el siguiente procedimiento.

En Visual Studio se crea un nuevo proyecto de visual *C++ Windows Form Application* que es el tipo de proyecto permite crear ejecutables con programación orientada a objetos.

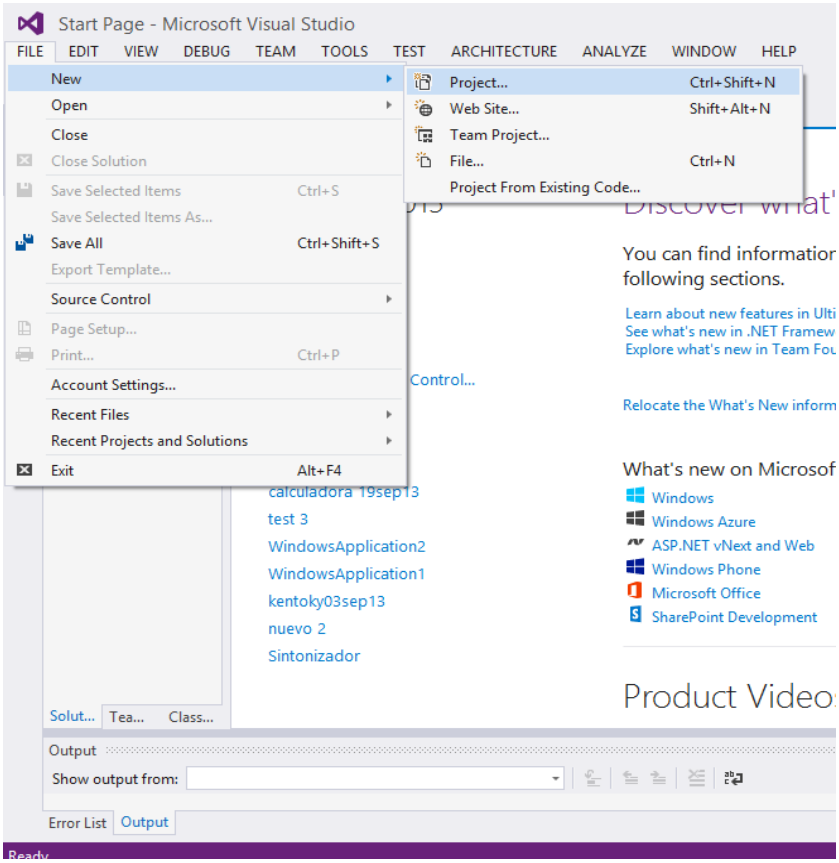



Figura 3.6 Se crea un nuevo proyecto en Visual Studio.

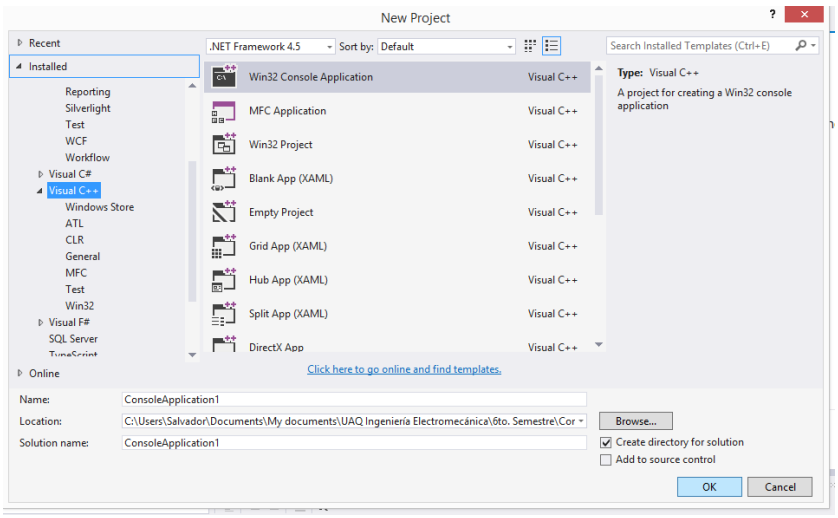



Figura 3.7 Se determina el tipo de proyecto (*Visual C++ Windows Form Application*).

Ahora se agregan todos los objetos necesarios para la interfaz como lo son cajas de texto, etiquetas, botones, entre otras.

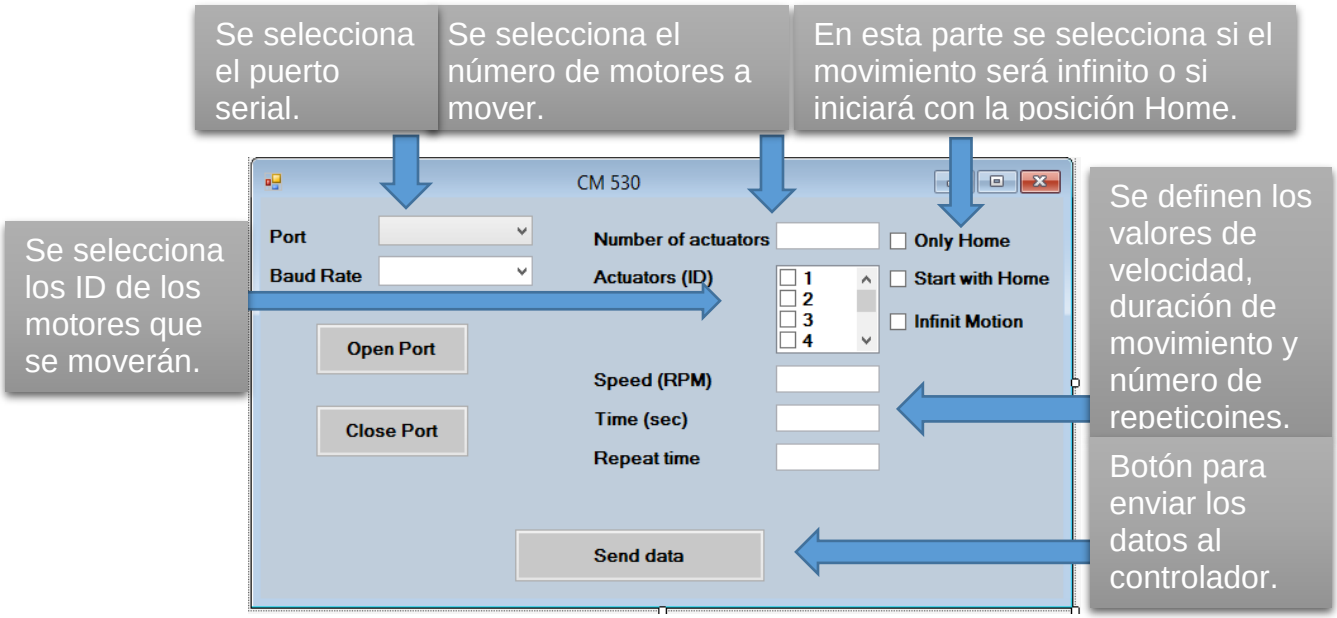



Figura 3.8 Interfaz del software con objetos agregados.

Como se observa en la figura 3.7 para poder enviar los datos desde esta interfaz se tiene que abrir el puerto serial, elegir el valor de Baud Rate[2] al que trabaja el controlador y una vez conectado el controlador CM-530 con la computadora se procede a asignar los valores necesarios para mover el robot de acuerdo a lo que se desee para que finalmente presionando el botón “Send data” el robot reciba la información necesaria y ejecute los comandos asignados.

2
Número de bits por segundo.

# 4. IMPLEMENTACIÓN DEL SOFTWARE

En esta sección se describe la experimentación llevada a cabo para probar el software desarrollado, se explica brevemente la forma en que se implementó y se muestran de forma detallada los resultados obtenidos.

## 4.1. Descarga del programa al Controlador CM-530

La descarga del programa al controlador es la transmisión del código máquina generado por el proyecto en Eclipse, el archivo que contiene este código se genera en la carpeta del proyecto y tiene formato *.hex*. Para poder realizar esta parte del proyecto se debe tener previamente instalado el software RoboPlus (Véase Apéndice 1).

Para que el robot ejecute los movimientos descritos por el usuario (Véase sección 3.2.1) es necesario descargar el programa al controlador, para esto se sigue el siguiente procedimiento.

Se abre el software RoboPlus en la ventana *Terminal* (Véase sección 2.2) como se muestra en la Figura 4.1.

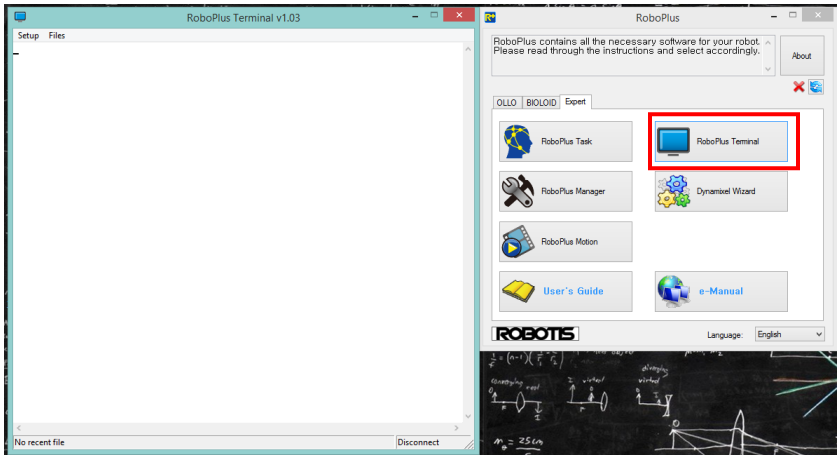



Figura 4.1 Ventana de RoboPlus Terminal.

Se conecta el controlador del Robot por medio del puerto USB y se realiza la comunicación Controlador – PC. Antes de transmitir el archivo se limpia la

información que tiene cargada el controlador. En la Figura 4.2 se muestra este procedimiento.

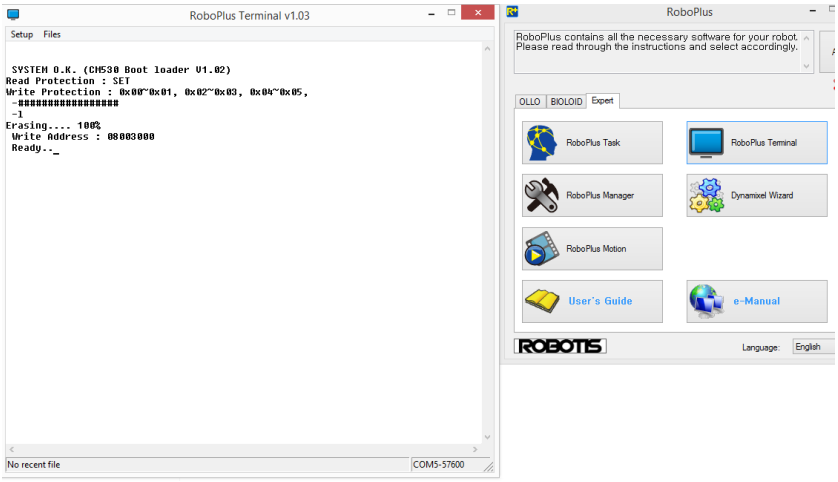



Figura 4.2 Conexión Controlador – PC.

Por último se transmite el archivo *.hex* como se muestra en la Figura 4.3.

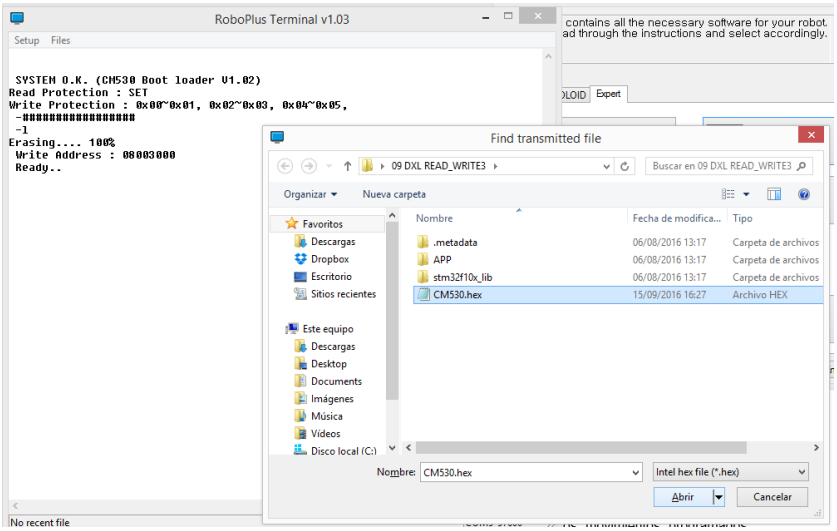



Figura 4.3 Transmisión de archivo *.hex* a controlador CM-530.

Una vez que el programa ha sido transmitido de forma exitosa, se ejecuta el programa con el Robot. Hay dos formas distintas de hacerlo; una es sin desconectarlo de la terminal de RoboPlus se escribe la letra “G” (abreviación de la

palabra “*Go*”, del inglés “Ir”) y se presiona la tecla *Enter*. Inmediatamente el Robot comienza a hacer los movimientos programados. La otra opción es con el Robot desconectado de la Terminal de RoboPlus apagándolo y encendiéndolo, en cuanto el Robot se enciente inmediatamente ejecuta los movimientos programados. Respetando los valores que se muestran en la Figura 3.3 como lo son; velocidad 4, número de motores 1, ID de los motores 3, posiciones 1, número de movimientos 2 y número de repeticiones 6 así como los valores de las variables Home 8, Onlyhome 9, Infinito 10 descritos por el usuario. Se debe tomar en cuenta que si la variable Infinito se activa, el programa ignora el número de repeticiones que se hayan predefinido ya que tiene prioridad la variable Infinito sobre la variable repeticiones.

La comprobación del funcionamiento del software desarrollado en el presente trabajo se muestra en la sección 4.2.

## 4.2. Ejemplos de programación de movimientos en el robot Bioloid Premium

En esta sección se muestran dos ejemplos de rutinas de movimientos programadas al Robot utilizando el software desarrollado en el presente proyecto de tesis.

Ya comprobado el funcionamiento del software se hicieron varias pruebas, definiendo valores de entrada y cargando el programa al Robot. A continuación se muestran 2 ejemplos de movimientos del Robot Bioloid Premium en configuración tipo A más dos grados de libertad.

### 4.2.1. Rutina *Saludar*

• Se programa un movimiento en el que el Robot saluda con las dos manos 4 veces. Para esto se definen los valores de entrada como se muestra en la Figura 4.4. Se puede ver que se deben mover en total 6 motores,

correspondientes a los brazos del Robot. El número de movimientos es 6 por lo que la matriz NP en este caso tiene un tamaño de 6x6 (renglón x columna), cada renglón de la matriz representa una posición diferente para cada motor que se desee mover y cada columna representa los diferentes motores que se moverán en el orden respectivo que se definieron de izquierda a derecha. Además se programa el movimiento con la variable *Home* igual a uno, quiere decir que está activada, entonces el Robot antes de comenzar con el movimiento programado adquiere la posición *Home* mostrada en la Figura 4.5.

Como esta es una rutina de prueba se utilizó el simulador de RoboPlus *Motion* para obtener los valores de posición de cada motor, es decir, se diseñó la rutina en este programa para poder comparar la ejecución con la rutina simulada.

Dado que RoboPlus maneja valores en un rango 0 – 1024 que es equivalente de forma proporcional al rango de trabajo de los actuadores Dynamixel $0^0$ - $300^0$ se hizo la multiplicación de cada valor por la cantidad proporcional (regla de tres) como se muestra a continuación.

$$Valor\ grados\ =\ Valor\ Bits\ *\frac{75}{256} \qquad \textit{(Ec. 4.1)}$$

Con esto se pudieron ingresar los valores en grados para cada movimiento de cada motor.

```
#define     NUM_ACTUATOR                6                   //NUMERO DE MOTORES QUE SE MOVERAN

#define     NumMov                      6                   //NUMERO DE MOVIMIENTOS

word        motor[NUM_ACTUATOR]={19,20,5,6};                //MOTORES QUE SE MOVERAN

word        RPM = 40;                                       //VELOCIDAD DE LOS MOTORES EN RPM,  RANGO; 0-114

word        time = 1000;                                    //TIEMPO DE DURACION DE CADA MOVIMIENTO EN MILISEGUNDOS

int         repeticiones = 4;                               //NUMERO DE REPETICIONES DEL MOVIMIENTO PROGRAMADO
//MATRIZ DE VALORES DE CADA MOVIEMIENTO DE CADA MOTOR EN GRADOS
word dNP[NumMov][NUM_ACTUATOR]={
                {   98.14453125 ,   171.3867188 ,   178.4179688 ,   171.3867188 ,   178.4179688 ,   98.14453125 },
                {   201.5625    ,   125.390625  ,   122.4609375 ,   125.390625  ,   122.4609375 ,   201.5625    },
                {   81.73828125 ,   90.234375   ,   90.52734375 ,   90.234375   ,   90.52734375 ,   81.73828125 },
                {   217.96875   ,   215.0390625 ,   213.5742188 ,   215.0390625 ,   213.5742188 ,   217.96875   },
                {   135.3515625 ,   47.16796875 ,   138.5742188 ,   47.16796875 ,   138.5742188 ,   135.3515625 },
                {   164.3554688 ,   253.125     ,   159.9609375 ,   253.125     ,   159.9609375 ,   164.3554688 }};
int         home = 1;                                       //INDICA SI EL MOVIMIENTO INICIARA CON LA POSICION DE HOME (0-1)

int         onlyhome = 0;                                   //INDICA SI EL ROBOT SOLO TOMARA LA POSICION DE HOME Y NADAMAS (0-1)

int         infinito = 0;                                   //INDICA SI EL NUMERO DE REPETICIONES DEL MOVIMIENTO SERA INFINITO (0-1
```

Figura 4.4 Valores de entrada para la rutina “saludar” del Robot.

Se carga el código al Robot y se observa que el Robot ejecuta los movimientos de forma correcta. La secuencia de movimientos se muestra en la Figura 4.5.

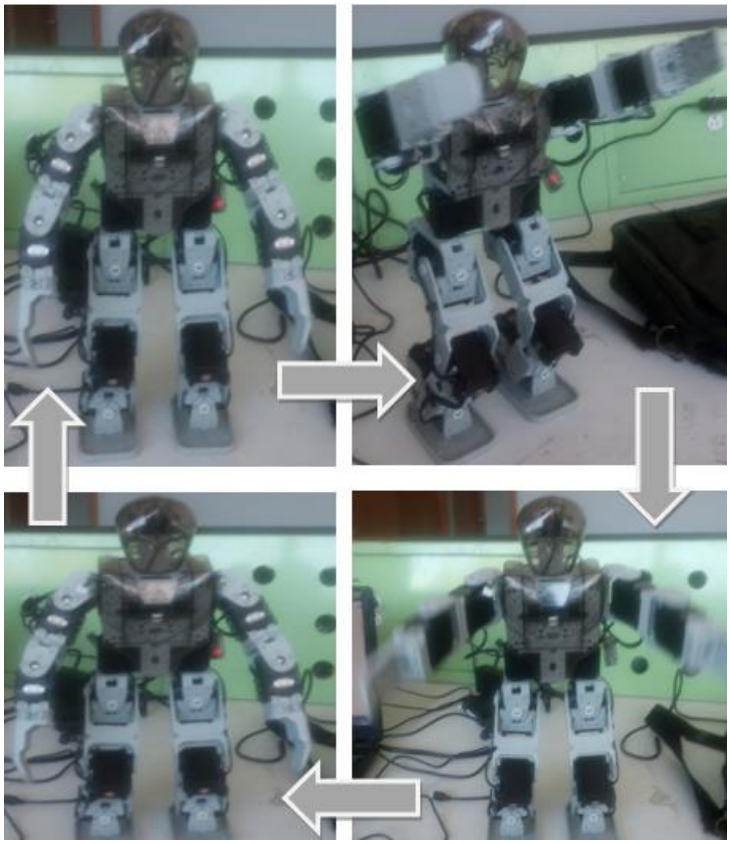


Figura 4.5 Diagrama de secuencia del Robot ejecutando la rutina “saludar”.

### 4.2.2. Rutina *Caminado*

• La segunda rutina programada al Robot utilizando el software desarrollado en este trabajo es la rutina de caminado para la cual se definieron los valores de entrada que se muestran en la Figura 4.6. Se observa que en esta rutina de movimientos están involucrados los 20 motores que contiene la configuración del Robot utilizado para el presente trabajo de tesis.

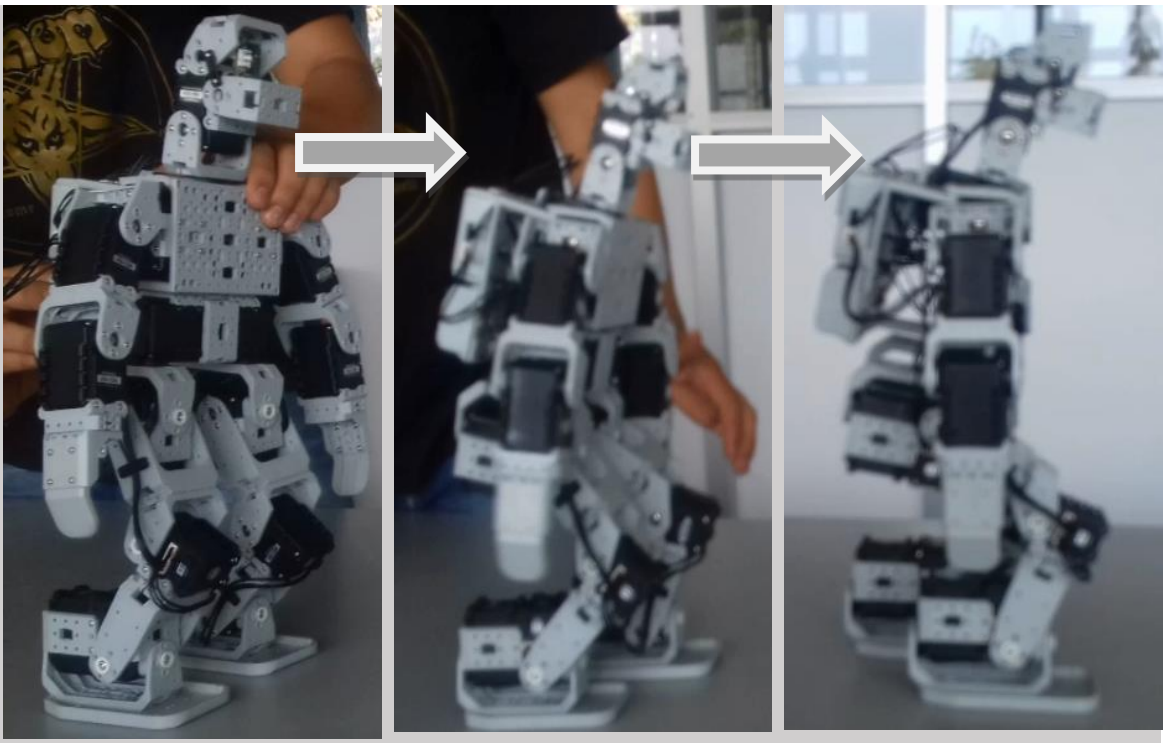
Debido a que la matriz de movimientos NP en este caso es muy grande (20x28) se decidió llenarla de valores más abajo en el código. Esta matriz de valores se obtuvo de la misma forma que para la rutina de saludo descrita anteriormente, los valores de movimiento se obtuvieron de las rutinas de prueba que maneja el fabricante del Robot. Y se convirtieron a grados de la misma forma que para la rutina de saludo, usando la ecuación 4.1.

```
#define     NUM_ACTUATOR            20 //NUMERO DE MOTORES QUE SE MOVERAN
#define     NumMov                  28 //NUMERO DE MOVIMIENTOS
word        motor[NUM_ACTUATOR]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20}; //MOTORES QUE SE MOVERAN  ID
word        RPM = 80;                   //VELOCIDAD DE LOS MOTORES EN RPM,  RANGO; 0-114
word        time = 80;              //TIEMPO DE DURACION DE CADA MOVIMIENTO EN MILISEGUNDOS
int         repeticiones = 3;           //NUMERO DE REPETICIONES DEL MOVIMIENTO PROGRAMADO
//MATRIZ DE VALORES DE CADA MOVIEMIENTO DE CADA MOTOR
word        NP[NumMov][NUM_ACTUATOR];
int         home = 1;                   //INDICA SI EL MOVIMIENTO INICIARA CON LA POSICION DE HOME (0-1)
int         onlyhome = 0;               //INDICA SI EL ROBOT SOLO TOMARA LA POSICION DE HOME Y NADAMAS (0-1)
int         infinito = 0;               //INDICA SI EL NUMERO DE REPETICIONES DEL MOVIMIENTO SERA INFINITO (0-1)
```

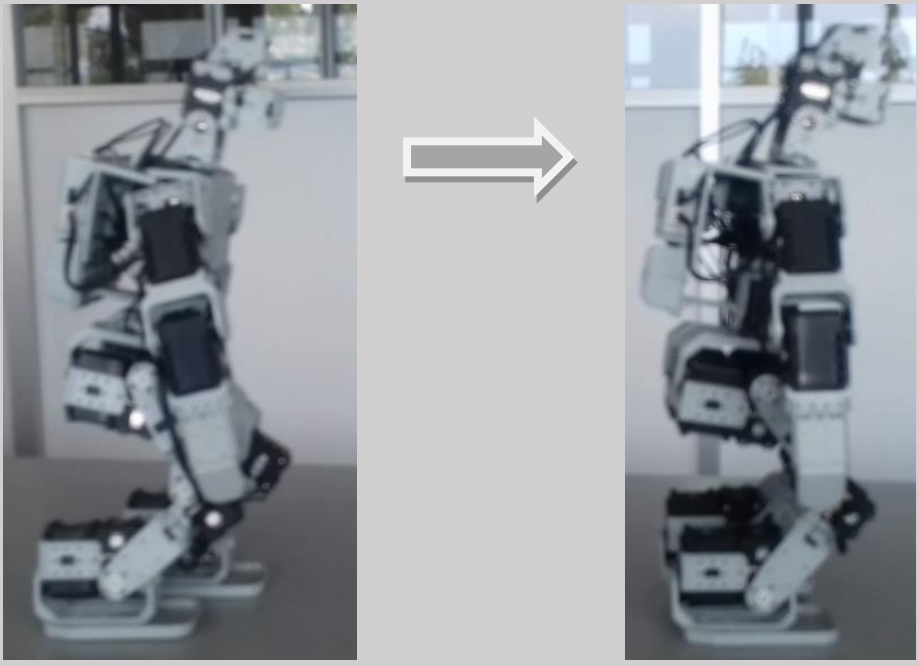
Figura 4.6 Valores de entrada para rutina de caminado en Robot Bioloid Premium tipo A.

Los movimientos programados en esta rutina son 28 y se requiere del movimiento de los 20 motores del Robot para llevarla a cabo como se muestra en la Figura 4.6. Además, se define que se repita el movimiento 3 veces por lo que el Robot dará 9 pasos en total ya que la matriz que se ingresa contiene 3 pasos programados. A diferencia de la rutina de saludo el tiempo de duración de cada movimiento se modificó a 80 ms, y la velocidad se aumentó a 80 RPM. También se declaró la variable *Home* con valor igual a uno para que el robot inicie y termine con esta posición.

En la Figura 4.7 se muestra el diagrama de secuencia de cómo el Robot ejecuta la rutina de caminado.



a) b) c)



d) e)

Figura 4.7 Diagrama de secuencia del Robot ejecutando la rutina de caminado.

A continuación se describe más a detalle la secuencia de movimientos mostrada en la Figura 4.7.

Una vez cargado el programa se hace la prueba de movimientos. En cuanto se enciende el Robot éste adquiere la postura de *Home* (Véase sección 3.3) como se muestra en la Figura 4.7 a), entonces comienza a caminar y da los primeros tres pasos (Figura 4.7 b)), es ahí cuando termina de ejecutar completamente los movimientos descritos en la matriz de movimientos (Figura 4.6). Pero como tiene programado repetir el movimiento 3 veces antes de terminar de moverse continúa con los siguientes 3 pasos como se observa en la Figura 4.7 c). Repite esta parte del movimiento una vez más (Figura 4.7 d)) y por último adquiere nuevamente la posición de Home para quedar estático como se muestra en la Figura 4.7 e).

# 5. CONCLUSIONES Y TRABAJO FUTURO

Con base en los resultados obtenidos con las pruebas de movimientos realizadas, se puede observar que el uso del software desarrollado en el presente trabajo de tesis ofrece múltiples ventajas respecto al software que maneja el fabricante RoboPlus.

## 5.1. Conclusiones

Es más fácil programar movimientos al Robot Bioloid Premium con el software desarrollado en el presente trabajo que con RoboPlus ya que los valores que se ingresan son magnitudes físicas como RPM (revoluciones por minuto), grados, segundos, etc. Además para realizar repeticiones de movimientos no es necesario describir sentencias de programación, sólo se debe escribir el número de repeticiones que se desea.

El software se puede aplicar a cualquier configuración del kit Bioloid Premium incluyendo todas las configuraciones que maneja el fabricante o incluso cualquiera que se le ocurra al usuario ya que puede conectar en serie un máximo de 254 actuadores Dynamixel y el software los reconoce inmediatamente y puede trabajar sin problemas.

Para agregar o quitar motores a la configuración del Robot Bioloid Premium con RoboPlus se debe seguir un procedimiento relativamente largo para cada uno de ellos, en cambio con el software desarrollado en el presente trabajo de tesis no hace falta hacer nada más que conectar o desconectar los motores que el usuario desee, el software inmediatamente reconoce la modificación.

El tiempo que toma programar un movimiento en este software es mucho más rápido que en RoboPlus ya que puede tomar alrededor de un minuto lo que lo hace cerca de diez veces más rápido y con menos pasos.

## 5.2. Resultados

Se cuenta con un programa que facilita la programación de movimientos al Robot Bioloid Premium con controlador CM-530 en cualquiera de sus configuraciones.

El programa permite modificar de forma directa los motores que se desean mover, la velocidad a la que se desea el movimiento, el número de repeticiones de cada movimiento, el número de movimientos, la duración de cada movimiento y la posición de cada uno de los motores en cada movimiento.

Se desarrolló una interfaz gráfica de usuario que facilitará el envío de los datos al controlador.

## 5.3. Trabajo a futuro

El programa sólo está hecho para programar movimientos al Robot, pero es muy importante para el uso de esta tecnología crear movimientos autónomos por lo que dentro del trabajo a futuro en este programa está facilitar la incorporación de sensores como los que contiene el kit; sensor ultrasónico, sensor infrarrojo, cámaras, etc. (Véase sección 3).

Se desarrolló la interfaz de usuario para facilitar la programación de movimientos al Robot, pero aún falta hacer la comunicación entre la interfaz y el controlador.

## REFERENCIAS


[1] http://en.robotis.com/index/company_01.php

[2] Software RoboPlus

[3] [“Lenguaje C”, Bonet Esteban].

[4] [Visual Studio Product Updates Blog (2014-11-12). "Visual Studio 2015 Preview".]

[5] Ing. Mauricio Ramos Vázquez, Análisis cinemático de posición y velocidad de un bípedo humanoide, 2015.

[6] http://support.robotis.com/en/product/auxdevice/controller/cm530.htm

[9] Alberto Pacheco, "Funciones y Librerías" Programación C, 2007

[10] http://www.cristalab.com/programacion-orientada-objetos/ambitos-public-private-protected-local/

Pardos Gotor, J. M. (2005). Algoritmos de geometría diferencial para la locomoción y navegación bípedas de robots humanoides: Aplicación al robot RH0.

Pfeiffer, S. (2011). Guiado gestual de un robot humanoide mediante un sensor" kinect".

Perea, I., Puente, S., Candelas, F. A., & Torres, F. (2010). Nueva tarjeta de sensorización y control para robot humanoide. XXXI Jornadas de Automática. Jaén.

Pardos Gotor, J. M. (2005). Algoritmos de geometría diferencial para la locomoción y navegación bípedas de robots humanoides: Aplicación al robot RH0.

Vanrell, J. A., & Vaucheret, C. A. (2006). Implementando ayuda dinámica en la plataforma eclipse. In VIII Workshop de Investigadores en Ciencias de la Computación.

Sánchez, E. A., Merin, G., & Muñoz, J. (2007). Hacia un Marco Práctico de Evaluación de Tecnologías de Transformación de Modelos en la Plataforma Eclipse. Actas de Talleres de Ingeniería del Software y Bases de Datos, 1(6), 81.

Valverde, F., Valderas, P., & Fons, J. (2007). OOWS Suite: Un Entorno de Desarrollo para Aplicaciones Web basado en MDA. In CIbSE (pp. 253-266).

Rodríguez-Vega, D. A., Zaldívar-Colado, U., Nuñez-Nalda, J. V., & Briseño-Cerón, A. Teleoperación en Línea de un Humanoide Virtual Mediante la Manipulación de un Humanoide Real. In Memorias del XIV Congreso Mexicano de Robótica, BUAP, Puebla, México (pp. 24-26).

Rodríguez Doñate, Carlos,  (2015) Procesamiento digital de señales en FPGA para análisis de vibraciones en robots industriales.

## APÉNDICES

### Apéndice 1. Metodología para la instalación de los programas empleados.

#### 1. Instalación de Java

El programa Java es un programa gratuito que se puede descargar desde la página. https://www.java.com/es/download/. Se selecciona el adecuado de acuerdo al sistema operativo de cada computadora y se instala.

#### 2. Instalación de Eclipse Kepler

La plataforma Eclipse se descarga de la página https://eclipse.org/downloads/ en la que se debe elegir la versión Kepler que es la utilizada para llevar a cabo este proyecto.

Una vez descargado se descomprime la carpeta y se pega directamente en Disco Local /C.

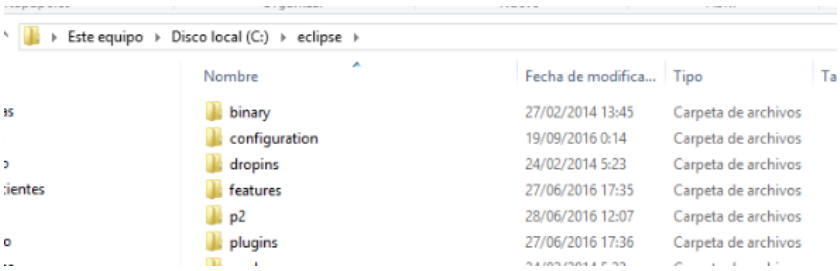



Figura 1 La carpeta se ubica en Disco Local /C.

Se abre la carpeta donde se instaló Java -> después la carpeta *bin* y se busca el archivo *javaw.exe.* Se copia este archivo.

Se abre la carpeta de eclipse ubicada en disco local C y se pega el archivo *javaw.exe.*

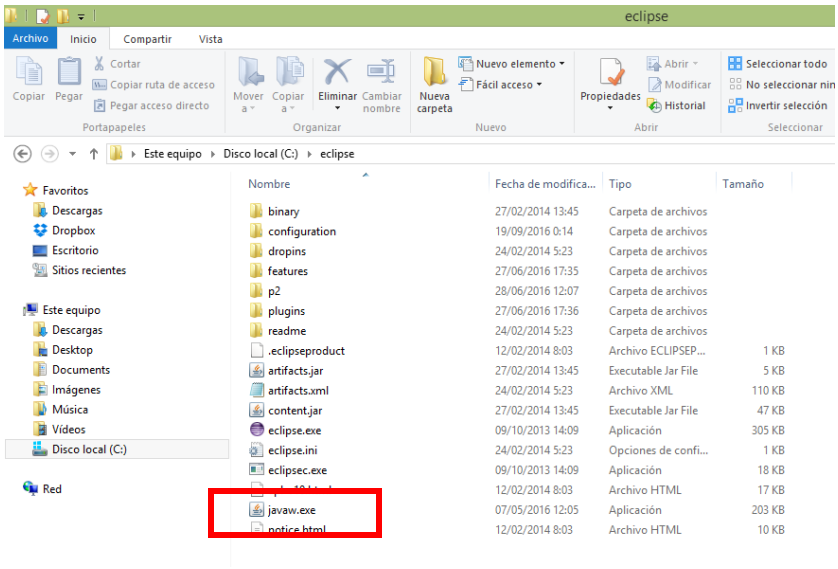



Figura 2 Se ubica dentro de la carpeta Eclipse el archivo javaw.exe.

Ahora se ejecuta el archivo *Eclipse.exe* cada que se desee trabajar en Eclipse y ya se podrá abrir sin problema.

Una vez que ya se puede trabajar en Eclipse correctamente se instalan los complementos para Lenguaje C/C++ dando clic en la ficha *Ayuda -> Instalar Nuevo software*.

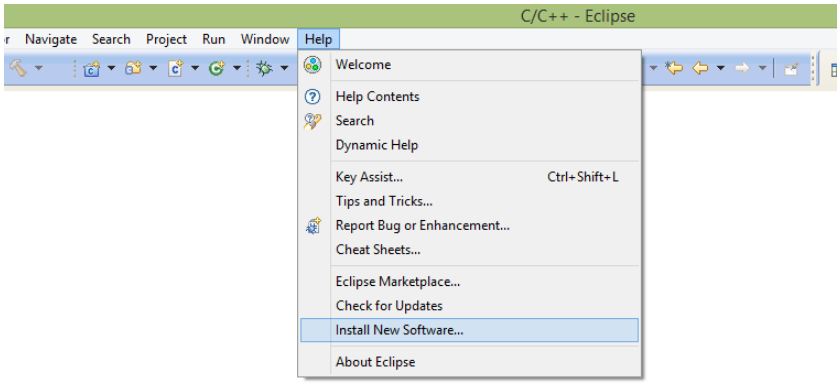



Figura 3 Se instalan los complementos para lenguaje C/C++.

Después en la barra que dice *Trabajar con* buscar C/C++ y seleccionarlo.

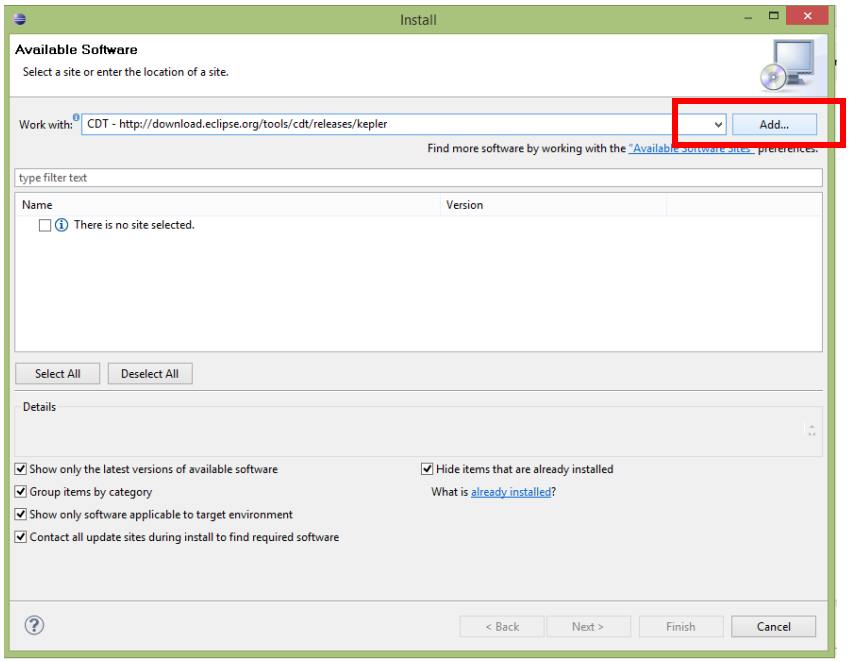



Figura 4 Se agregan los complementos C/C++.

Por último dar clic en *Agregar* y listo, ahora se puede trabajar con proyectos en Lenguaje C/C++.

### 3. Instalación de compilador MinGW

Para poder realizar proyectos se debe tener instalado el compilador correcto, en este caso MinGW, se descarga desde la página http://www.mingw.org/ y se selecciona de acuerdo al sistema operativo de cada computadora.

Una vez descargado se instala. Posterior a esto se debe modificar una variable de entorno llamada *Path* para direccionar el compilador para esto abrimos Panel de Control -> Sistema -> Configuración Avanzada del Sistema -> Variables de Entorno -> Variables del Sistema.

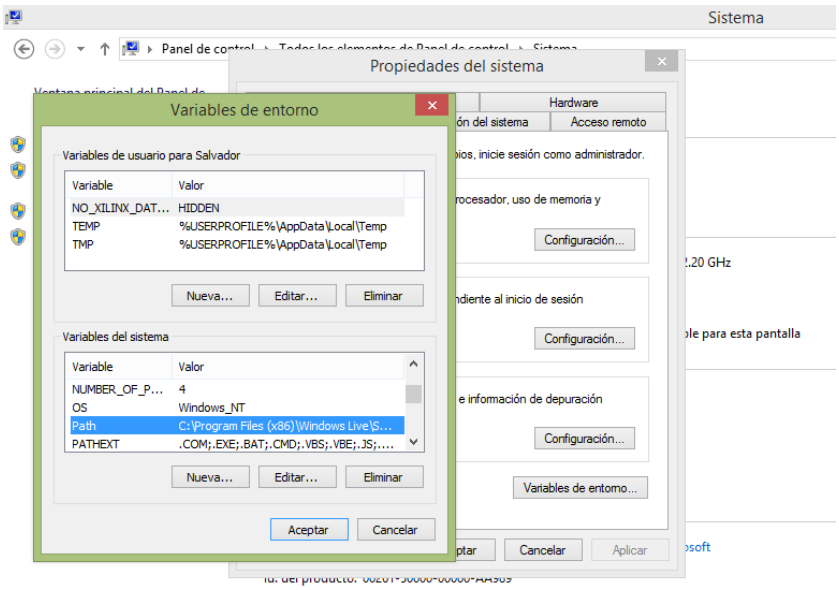



Figura 5 Se modifica la variable de entorno Path.

Ahora buscar la variable *Path* y seleccionarla, dar clic en Editar y hasta el final del texto del valor de la variable pegar la siguiente dirección: **“*;C:\MinGW\bin*”**

Ahora solo dar clic en Aceptar -> Aceptar -> Aceptar ya está correctamente instalado el compilador. Ahora ya se pueden compilar los proyectos en lenguaje C desde Eclipse Kepler.

## 4. Instalación de RoboPlus

El programa RoboPlus se puede descargar directamente de la página de Robotis http://www.robotis.com/xe/1132559 y simplemente se instala.

## Apéndice 2. Manual de usuario.

Para poder hacer uso del software se deben tener instalados los programas necesarios (Véase Apéndice 1).

### 1. Agregar el proyecto a Eclipse.

La carpeta que contiene el proyecto se debe ubicar en el escritorio de la computadora para que las direcciones internas coincidan y no marque error al compilar.

Se abre la plataforma Eclipse Kepler y se agrega el proyecto dando clic en Nuevo -> Agregar proyecto con código existente -> seleccionar la carpeta -> Abrir.

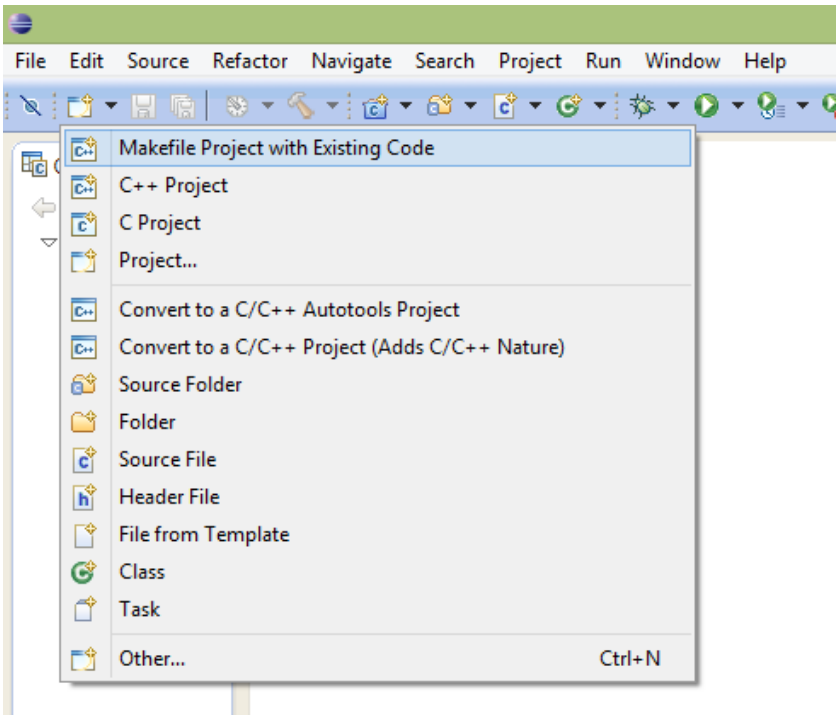
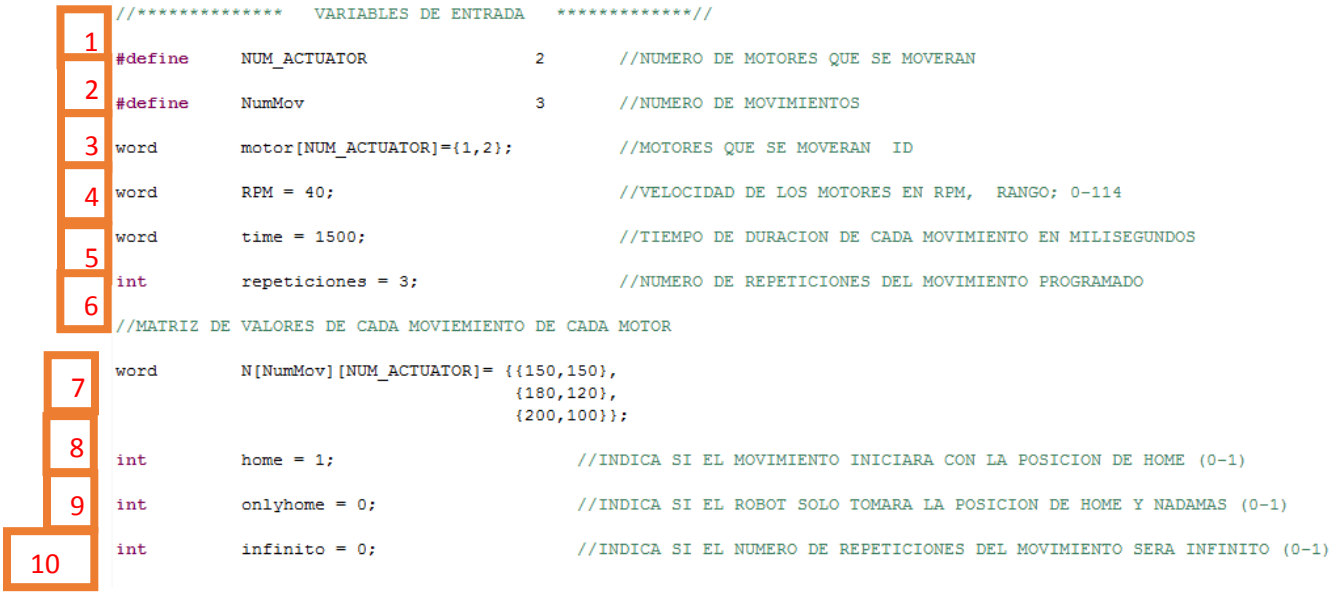



Figura 1 Se agrega el proyecto a la plataforma Eclipse Kepler.

## 2. Programación de movimientos

Ahora se abre el archivo main.c que se encuentra ubicado en la carpeta Proyecto/App/scr/main.c y se debe ubicar la parte que dice VARIABLES DE ENTRADA. Es aquí donde se describen los movimientos al robot.




```
//**************   VARIABLES DE ENTRADA   *************//
#define     NUM_ACTUATOR                2       //NUMERO DE MOTORES QUE SE MOVERAN
#define     NumMov                      3       //NUMERO DE MOVIMIENTOS
word        motor[NUM_ACTUATOR]={1,2};          //MOTORES QUE SE MOVERAN  ID
word        RPM = 40;                           //VELOCIDAD DE LOS MOTORES EN RPM,  RANGO; 0-114
word        time = 1500;                        //TIEMPO DE DURACION DE CADA MOVIMIENTO EN MILISEGUNDOS
int         repeticiones = 3;                   //NUMERO DE REPETICIONES DEL MOVIMIENTO PROGRAMADO
//MATRIZ DE VALORES DE CADA MOVIEMIENTO DE CADA MOTOR
word        N[NumMov][NUM_ACTUATOR]= {{150,150},
                                      {180,120},
                                      {200,100}};

int         home = 1;                       //INDICA SI EL MOVIMIENTO INICIARA CON LA POSICION DE HOME (0-1)
int         onlyhome = 0;                   //INDICA SI EL ROBOT SOLO TOMARA LA POSICION DE HOME Y NADAMAS (0-1)
int         infinito = 0;                   //INDICA SI EL NUMERO DE REPETICIONES DEL MOVIMIENTO SERA INFINITO (0-1)
```

Figura 2 Ventana donde se ingresan los parámetros para los movimientos del Robot.

1 En el primer renglón se define el número de motores que se desea mover.

2 En el segundo renglón se define el número de movimientos que se desee programar para cada motor.

3 En el tercer renglón se escriben los ID (Número de identificación de cada motor, lo traen impreso en la parte de atrás) de los actuadores que se deseen mover.

NOTA; es importante tener en cuenta el orden en que se ingresan los ID ya que en ese orden se ingresarán las diferentes posiciones para cada uno de ellos.

NOTA; si el número de actuadores definido en el primer renglón no coincide con la cantidad de ID descritos el robot no ejecutará ninguna acción.

4 En el cuarto renglón se define la velocidad a la que girarán los actuadores, ésta puede variar en un rango de 0 – 114 RPM.

NOTA; si se ingresan valores de velocidad negativos el Robot no se moverá, y si se ingresan valores mayores a los de su rango el Robot se moverá a su máxima velocidad.

5 En el quinto renglón se define el tiempo de duración de cada movimiento de los motores en milisegundos (se eligió en ms para tener una mayor resolución de tiempo).

6 En el renglón sexto se define el número de repeticiones que se desee que el Robot haga los movimientos programados.

Nota; si se desea que el Robot sólo haga el movimiento una vez y no lo repita, entonces la variable repeticiones debe valer uno.

7 En el renglón séptimo se definen los valores de posición en grados en un rango de $0^0$-$300^0$ (este rango de posición se ilustra en la Figura 3) para cada uno de los motores. Cada columna representa los movimientos para cada uno de los motores definidos de izquierda a derecha respectivamente como se definieron en el renglón 3. Y cada renglón representa las diferentes posiciones que tomará cada motor en cada movimiento.

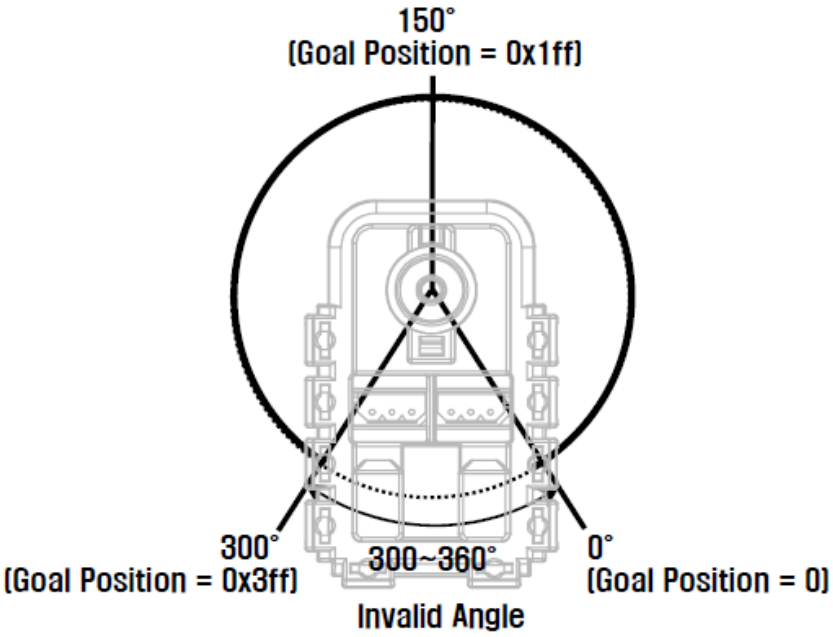



Figura 3 Rango de posición que puede tomar el actuador Dynamixel.

NOTA; si los valores de posición que se definan se encuentran fuera del rango de trabajo el motor se puede dañar.

NOTA; si la matriz de movimientos descrita no coincide con el tamaño definido por el número de movimientos y por el número de actuadores el programa marcará error y no compilará.

8 En el octavo renglón se define si se activa o no la variable *Home*

*Home = 1; Activada*

*Home = 0; Desactivada*

Si se encuentra activada, el Robot antes de iniciar los movimientos adquiere la posición Home (Véase sección 3.3) y después los ejecuta, una vez que termina se queda en esta posición.

Si se encuentra desactivada los únicos motores que se moverán serán los definidos en el renglón 3 y los demás se quedarán estáticos en la posición en la que se encuentren en el momento de ejecutar la rutina.

NOTA; si se asigna un valor diferente de uno o de cero la variable se encontrará desactivada.

9 En el renglón noveno se asigna el valor de la variable *Onlyhome* que representa si el Robot solamente toma su posición inicial y nada más. Si esta variable está activada (=1) el Robot ignora todos los demás movimientos y sólo toma su posición de *Home* a la velocidad descrita y no hace nada más.

*Onlyhome = 1; Activada*
*Onlyhome = 0; Desactivada*

NOTA; el uso de la posición *Home* sólo aplica para la configuración humanoide tipo A con 18, 19 o 20 servomotores en los que los ID se encuentren ensamblados en el orden que marca el manual del fabricante. Si no se cuenta con un Robot armado en alguna de estas configuraciones se recomienda NO USAR esta posición para evitar daños físicos al equipo.

NOTA; si se asigna un valor diferente de uno o de cero la variable se encontrará desactivada.

10 En el renglón décimo se define si el número de movimientos programados en la matriz de movimientos se repite de forma indefinida, es decir, sin parar. Por medio de la variable *infinito.*

*Infinito = 1; Activada*
*Infinito = 0; Desactivada*

NOTA; si esta variable se encuentra activada el Robot ignora el valor de la variable *Repeticiones.* Tiene prioridad la variable *Infinito.*

NOTA; si se asigna un valor diferente de uno o de cero la variable se encontrará desactivada.

### 3. Construcción de proyecto

Una vez programados los movimientos deseados se debe generar el archivo .hex que es el que contiene el código máquina para posteriormente cargarlo al controlador.

Se da clic en Guardar todo 1 -> clic en botón Construir. 2

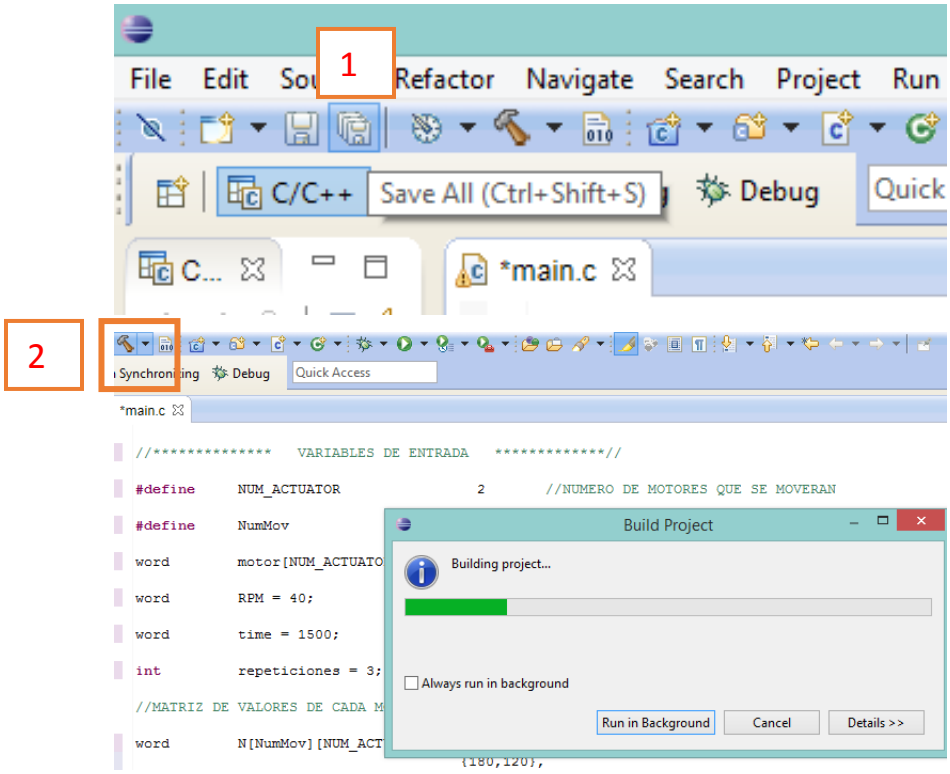



Figura 4 Construcción de proyecto.

### 4. Cargar programa al controlador

Con el controlador CM-530 apagado y conectado a la PC por medio del puerto serial se abre la terminal de RoboPlus.

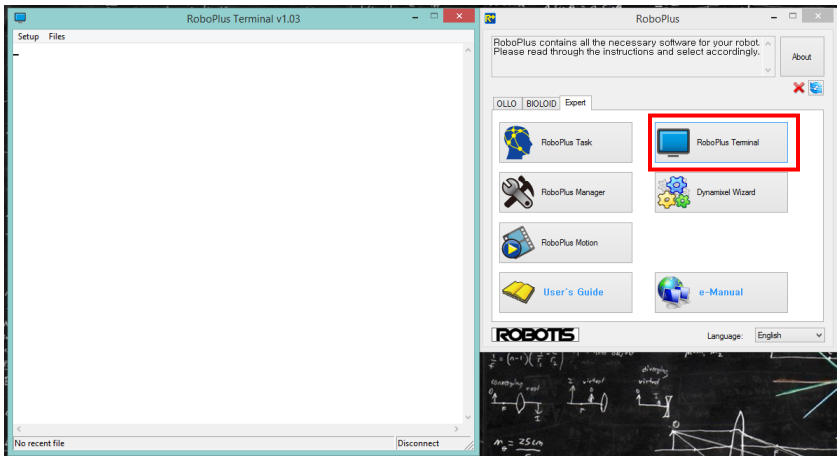



Figura 5 Ventana de RoboPlus Terminal.

Se conecta el controlador del Robot por medio del puerto USB y se realiza la comunicación Controlador – PC. Para esto se debe dar clic en Configuración -> Conectar -> Conectar. Ahora se mantiene presionado el símbolo # (gato) y se enciende el Robot.

Antes de transmitir el archivo se debe limpiar la información que tiene cargada el controlador. Para esto se escribe la letra "*L*" (abreviación de "*Load*", del Inglés, Cargar) y presionar tecla *Enter* como se muestra en la Figura 6.

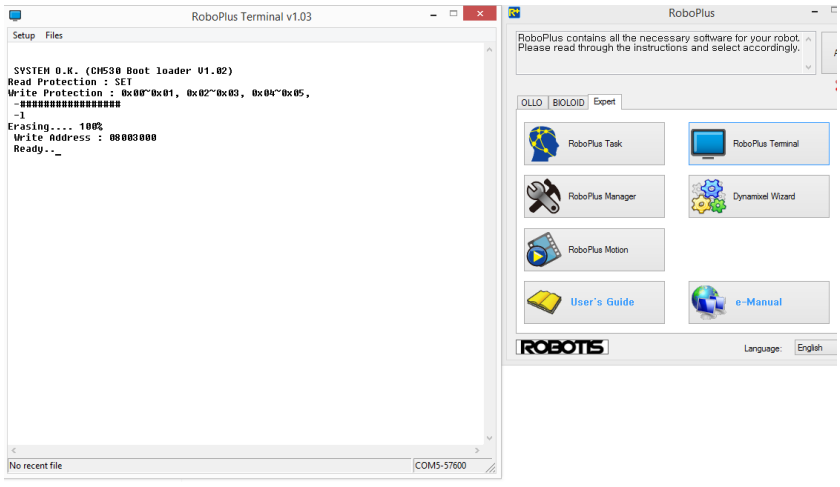



Figura 6 Conexión Controlador – PC.

Ahora se transmite el archivo .hex. Para ello se da clic en la ficha Archivo -> Transmitir archivo. Se busca el archivo en formato .hex que se encuentra en el menú principal de la carpeta del proyecto y se selecciona dando doble clic.

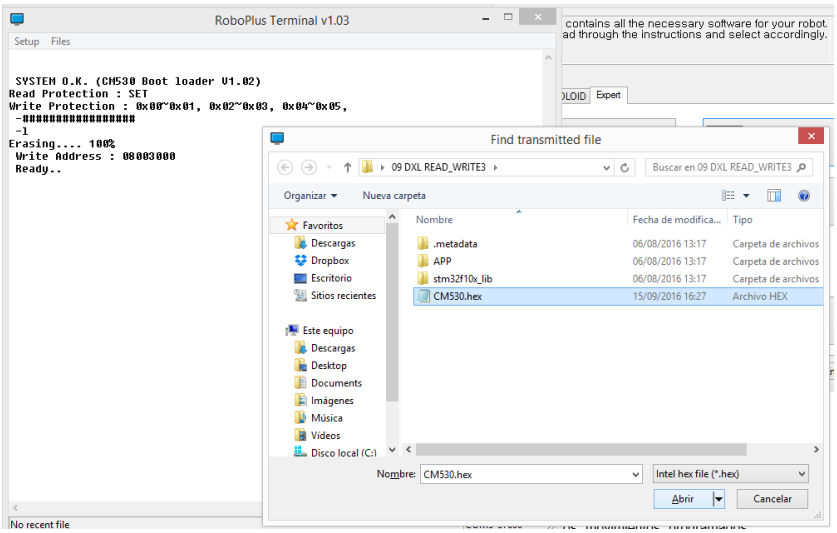



Figura 7 Transmisión de archivo *.hex* a controlador CM-530.

Una vez que el programa ha sido transmitido de forma exitosa, se ejecuta el programa con el Robot. Hay dos formas distintas de hacerlo; una es sin desconectarlo de la terminal de RoboPlus se escribe la letra “G” (abreviación de la palabra “*Go*”, del inglés “Ir”) y se presiona la tecla *Enter*. Inmediatamente el Robot comienza a hacer los movimientos programados. La otra opción es con el Robot desconectado de la Terminal de RoboPlus apagándolo y encendiéndolo, en cuanto el Robot se enciente inmediatamente ejecuta los movimientos programados.