Programación de gaits y adquisición de datos para robots serpiente

Vivian Dayana Fernández García

Director de Trabajo de Grado:

Kamilo Melo
Departamento de Electrónica,
Pontificia Universidad Javeriana

Carrera de Ingeniería Electrónica
Pontificia Universidad Javeriana
Bogotá D.C., Colombia

20 de junio de 2013

Índice general

Capítulo 1

Introducción

Durante años, los robots modulares han servido como plataformas para experimentar con diferentes esquemas de locomoción. Teniendo en cuenta la clase de robot modular que se está utilizando [1], algunos robots modulares han sido desarrollados para lograr locomoción. Particularmente, el estudio de la locomoción en serpientes biológicas ha colaborado con la selección de estrategias adecuadas para lograr mover estos robots de una manera efectiva [2]. Es por esto que el uso de los robots serpiente se ha venido incrementando debido al potencial que estos tienen de moverse a través de diferentes ambientes y la capacidad que tienen para desempeñar ciertas tareas específicas como la inspección de terrenos con escombros, evaluación de terrenos con riesgo de dispositivos explosivos [3], inspeccionar dentro y fuera de un tubo [4], rescate de victimas de un desastre [5], entre otras. Dentro de los robots modulares se encuentran los *Modular Snake Robots* o MSR los cuales son de particular interés debido al potencial que estos robots poseen para atravesar diferentes obstaculos, razón por la cuál la locomoción de estos robots debe ser modelada para no solo entender la fisica del movimiento del robot, sino también, para poder realizar control sobre los diferentes esquemas de locomoción posibles, con el fin de hacer estos robots cada vez más autónomos [6].

1.1. Objetivos

1.1.1. Objetivo general

Implementar un conjunto de funciones en un lenguaje de programación de alto nivel, que permitan la programación de movimientos (scripted gaits) y adquisición de variables de los sensores de un Modular Snake Robot.

1.1.2. Objetivos específicos

- Implementar un conjunto de funciones en un lenguaje de programación de alto nivel, que permitan el envío de datos de posición a un conjunto de 16

servomotores Dinamixel AX-12.

- Implementar un conjunto de funciones en un lenguaje de programación de alto nivel, que permitan la recepción de datos provenientes de los sensores integrados a los servomotores Dinamixel AX-12.

- Implementar un conjunto de funciones en un lenguaje de programación de alto nivel, que permitan almacenar en una estructura de datos la información proveniente de los servomotores Dinamixel AX-12.

- Determinar un protocolo de pruebas para verificar el funcionamiento de las funciones implementadas.

1.2. Glosario

- Configuración: Conjunto de variables que especifican la ubicación y orientación de todos los puntos de un robot. Al conjunto de todas las posibles configuraciones se le denomina *espacio de configuración*.

- Gait: En la literatura relacionada con robotica, los *gaits* son los cambios de configuración que reflejan macroscopicamente cambios en la forma del robot, con los que se logra locomoción. Tambien llamados esquemas de locomoción [7]. Hace referencia al tipo de esquema de locomoción que el robot ejecuta al moverse.

- Grado de Libertad (DOF): Es una de las direcciones en las cuales un cuerpo es capaz de moverse. También está relacionado con la dirección hacia la cual puede girar un cuerpo.

- Kit de Desarrollo de Software (*Software Development Kit* SDK): Conjunto de herramientas de desarrollo de software (librerias pre-compiladas) que permiten al programador utilizar estas librerias para embeberlas en otro lenguaje de programación.

- Interfaz de Programación de Aplicaciones (*Application Programming Interface* API): Libreria de programación estándar que hace parte del *Software Development Kit* SDK que es utilizada para desarrollar software y realizar acciones de control sobre los servomotores *Dynamixel*.

- Efector Final: Nombre con el que se denomina a la herramienta que usualmente se ubica en el extremo de un robot manipulador y con la que se efectua la tarea.

- *Front End*: Parte del software que tiene como función interactuar con el usuario, es decir, es el responsable de recolectar la información introducida por el usuario y almacenarla para ser utilizada por el software desarrollado.

- *Middleware*:Software de conectividad que consiste en un conjunto de funciones que permiten a una aplicación interactuar o comunicarse con otras aplicaciones.

Capítulo 2

Marco Teórico

Para un mejor entendimiento de los desarrollos que se reportan en este documento de trabajo de grado, este capitulo pretende introducir al lector en algunos terminos, y conceptos provenientes de desarrollos previos por otros autores, asi como de la descripción de algunas de las herramientas de desarrollo usadas en este trabajo.

2.1. *Modular Snake Robot*

En un trabajo previo, realizado por la Ingeniera Laura Paéz y dirigido por el Ingeniero Kamilo Melo, se presentó el robot modular Lola-OP^{TM} [8]; dicha plataforma de hardware abierto es la que se usará, para los objetivos de este trabajo. En este robot diferentes *gaits* pueden ser programados [4], sin embargo, para el objeto de este trabajo, los métodos de control del robot están enfocados hacia una clase determinada de *gaits* y se expondrán en el desarrollo del trabajo. El robot, mostrado en la figura 2.1 está compuesto por una serie de n 1-DOF modulos interconectados entre ellos de manera serial, con torsión de 90 en sus ejes de rotación. Cada módulo está compuesto por servomotores Dinamixel AX-12 [4]. Ahora, para describir el software utilizado para controlar el robot se explican las herramientas utilizadas para enviar información a los servomotores y para crear la interfaz de usuario por medio de la cual se configura el robot. Lograr programar y registrar secuencias de configuraciones posibles del robot es el objeto de este trabajo.

2.1.1. Actuadores

Los actuadores utilizados por el robot Lola-OP^{TM}son los servomotores dynamixel AX-12. Estos servomotores tienen un microcontrolador interno que se caracteriza por ser capaz de entender 50 comandos diferentes (como posición angular, velocidad rotacional), entre los que se pueden escribir o leer parámetros que definen el comportamiento del servomotor. El microcontrolador puede

Figura 2.1: Robot Modular Lola-OP^{TM}

entregar un estado (*status*) de parámetros como temperatura interna, velocidad, voltaje y posición, entre otros. Estos servomotores pueden autoprotegerse cuando detectan sobre-voltaje, sobre-calentamiento o condiciones de error. Cuando los servomotores detectan un error lo muestran mediante el encendido de un led que este posee. Tienen una resolución de 1024 pasos, que corresponden a un intervalo de posiciones angulares entre 0° y 300° y así mismo, una velocidad con un intervalos de 1024 pasos. Es posible conectar hasta 254 servomotores AX-12 en cadena [9].

2.2. Herramientas de Software

2.2.1. Dynamixel SDK

Dynamixel (SDK) es un conjunto de herramientas de software dentro de las cuales se encuentra un API (ver glosario), que permiten al programador comunicarse con los servomotores y de la misma manera recibir datos de los mismos. A esto se le denomina *middleware* (ver figura 2.2). Con el API se tiene una libreria estandar por medio de la cual se pueden controlar los servomotores. Esta libreria llamada `dinamixel.h` permite programar de una forma más sencilla en lenguaje C funciones de uso general como lo son controlar la comunicaciòn entre el computador y los servomotores, escribir y leer datos [10].Las funciones con las que cuenta la libreria se pueden agrupar de la siguiente manera:

- Procedimiento de control de dispositivo (*Device Control Method*): Son funciones que permiten controlar la iniciación y la finalización de los dispositivos de comunicación. Estas funciones son importantes para establecer la comunicación entre el computador y el robot.

- Procedimiento de establecer/obtener paquetes (*Set/Get Packet Method*): Son funciones que permiten armar y desarmar paquetes de instrucciones para ser enviadas a los servomotores. Un paquete de instrucciones cambia su estructura de acuerdo a la instrucción que se desea ejecutar; dentro de las funciones basicas se encuentran escribir un dato en el servomotor y leer un dato del mismo. Un paquete de instrucciones se compone de Id (número) del actuador, longitud del paquete a transmitir, la instrucción que se desea ejecutar, y los parametros sobre los que se realizará la instrucción (leer o escribir) [11].

- Procedimiento de comunicación del paquete (*Packet Communication Method*): Son funciones que permiten enviar y recibir el paquete de instrucciones. Tambien existen funciones que permiten determinar el estado de la comunicación del paquete, es decir, permite saber si la comunicación fue exitosa o si se presentó algun error.

- Procedimiento de comunicación de alto nivel (*High Communication Method*): Son funciones que facilitan el proceso de armar el paquete de instrucciones de uso frecuente, es decir, por ejemplo, para escribir una posición a un servomotor no es necesario armar todo un paquete de instrucciones sino que, utilizando estas funciones simplemente se llama la función y se escribe el Id del servomotor y la posición que este debe tomar.

- Procedimiento de utilidad (*Utility Method*): Son funciones útiles que son comunmente utilizadas. Por ejemplo si se quiere obtener un dato de los servomotores, esta información se compone de dos bytes. Una de las funciones que se encuentra en este grupo permite armar la palabra que se compone de los dos bytes y de esta manera se obtiene el dato deseado.

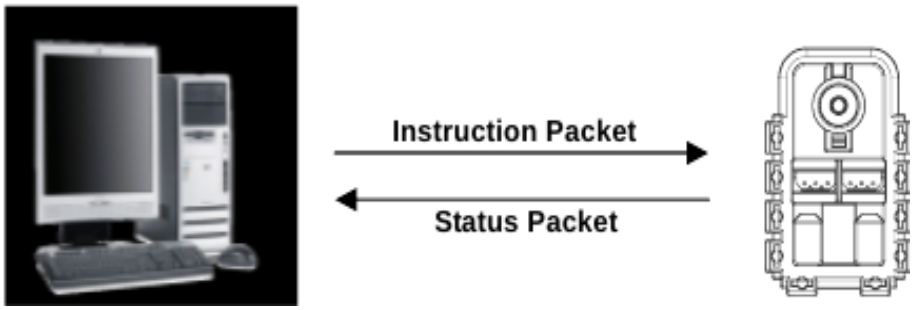

Figura 2.2: Intercambio de Información entre el PC y los Servomotores

2.2.2. Qt Creator

Qt creator es una herramienta utilizada para desarrollar aplicaciones con interfaces gráficas de usuario. Es multiplataforma, es decir, se pueden programar

aplicaciones ejecutables en diferentes sistemas operativos. El lenguaje propio de este programa es C++, sin embargo puede ser utilizado con otros lenguajes de programación por medio de *bindings* (adaptaciones de la biblioteca a un lenguaje de programación diferente al originario). Para el codigo escrito en este trabajo se utilizó el lenguaje de programación C++ por ser orientado a objetos, como se vera mas adelante. Qt Creator ha sido utilizado previamente para el desarrollo de interfaces de control para robots serpiente; dichas interfaces se pueden observar en [12].

2.3. *Scripted Gaits*

El profesor Howie Choset del *Robotic Institute de Carnegie Mellon University* (CMU), en su trabajo [7] clasifica los esquemas de locomoción de un MSR en *parametrized gaits* y *scripted gaits*. Los *parametrized gaits* son aquellos que pueden ser descritos por medio de funciones sinusoidales o cíclicas. En un trabajo previo [9], se pueden observar algunos ejemplos de *parametrized gaits* ejecutados en diferentes ambientes por el robot que se utiliza en este trabajo de grado [13]. Por otra parte los *scripted gaits* (los cuales son relevantes para este desarrollo reportado en este documento) son aquellos que no pueden ser generados con una función analítica, sino que son generados mediante los cambios de configuración de robot, debido a la asignación paso a paso de los angulos de juntura asociados al conjunto de modulos que conforman el robot, teniendo así conocimiento de la postura del robot, es decir, el robot puede ser controlado y puede moverse a la posición o forma deseada asignando estas variables de juntura. Un robot de este tipo, esta conformado por segmentos unidos por junturas (ver figura 2.1). A cada juntura se le asigna un ángulo y el conjunto completo de ángulos que representan al robot se conoce como configuración [14]. Los cambios de configuración llevan a movimientos y existen diversos ejemplos de este tipo de movimientos que se pueden ver en [7]. Finalmente, para este trabajo se tuvieron en cuenta unicamente *scripted gaits* .

Los *scripted gaits* son muy útiles debido a que permiten programar diferentes configuraciones en especial para ser utilizadas en las transiciones que impliquen cambios de configuración, es decir, por medio de los *scripted gaits* se puede pasar de la ejecución de un *parametrized gait* a otro buscando una mayor efectividad en los movimientos del robot teniendo en cuenta varios factores como velocidad, energía y estabilidad, entre otros. En un trabajo paralelo a este, los estudiantes Diego Roa y Sebastian Herrera se encuentran analizando especialmente la estabilidad mecánica del robot para diferentes configuraciones programadas y utilizarán los *scripted gaits* junto con el resultado de este trabajo con el objetivo de lograr establecer configuraciones que permitan el uso de los extremos del robot como efector final. Estos son algunos ejemplos de la importancia que tiene este tipo de esquemas de locomoción, y es por esto que se han venido realizando trabajos previos en los cuales se desarrollaron una serie de movimientos simulados que permiten al robot ejecutar tareas específicas. Dichos estudios previos son de gran relevancia para este trabajo de grado ya que en ellos se hizo un

análisis del movimiento de cada uno de los modulos del robot serpiente y como se comportaban estos al unirlos para formar una cadena articulada. Por medio de los *scripted gaits* es posible confirmar las investigaciones hechas previamente en cuanto al movimiento de los modulos articulados del robot. La importancia y justificación de este trabajo es que al programar *scripted gaits* se tienen resultados experimentales que permiten comparar los resultados obtenidos teoricamente y los resultados obtenidos experimentalmente en [15], [16] y [17].

Capítulo 3

Descripción General

Para describir los desarrollos que se mostrarán más adelante, se realizará una breve descripción del trabajo efectuado reportado en este documento a partir del diagrama de bloques general del proyecto que se observa a continuación.

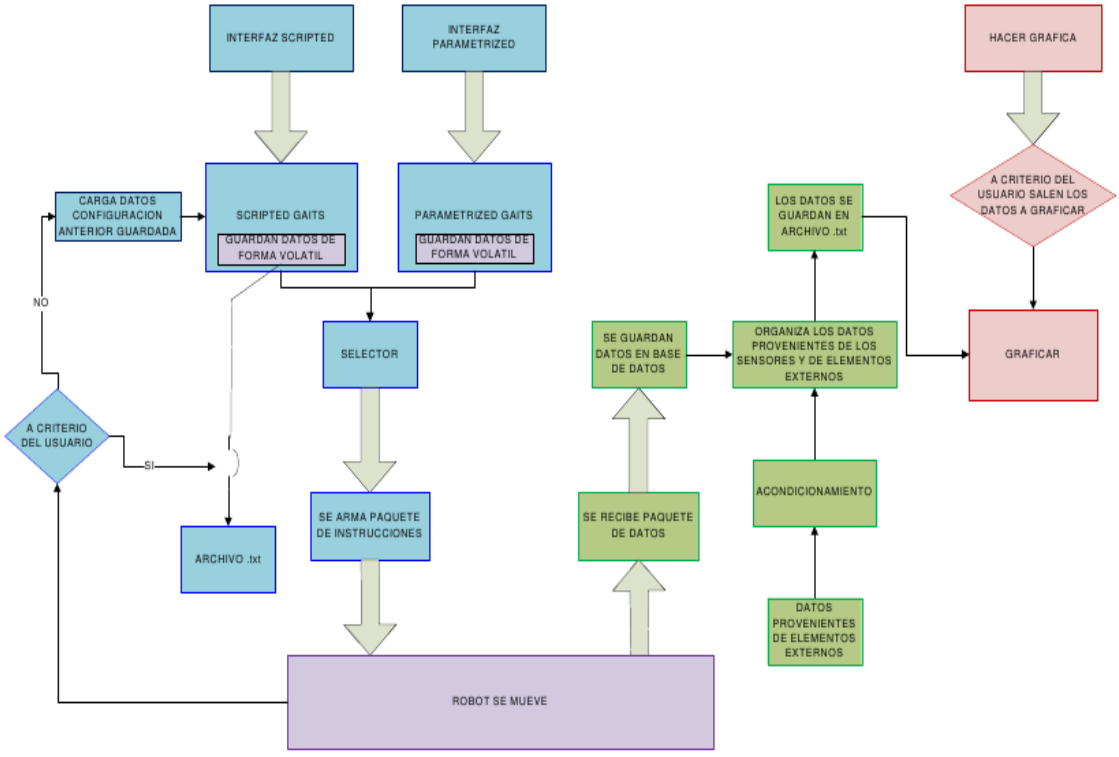

Figura 3.1: Diagrama de Bloques General

En el diagrama de la figura 3.1 se observa la descripción general del desarrollo de software propuesto para este trabajo. Para controlar el robot serpiente se utilizan 3 interfaces: *parametrized gaits*, *scripted gaits* y la interfaz que permite graficar datos. La primera interfaz, *parametrized gaits* fue desarrollada en un trabajo previo bajo la dirección del director de este trabajo de grado [9] y dada su arquitectura, fue posible integrarla a este; sin embargo, en este documento, el tema se centrará en las otras dos. Las interfaces desarrolladas en este trabajo de grado son la interfaz de *scripted gaits* y la interfaz utilizada para graficar los datos obtenidos. Es importante tener en cuenta que estas interfaces funcionan como *front ends* para todo el conjunto de funciones que corresponden al desarrollo propuesto en los objetivos de este trabajo.

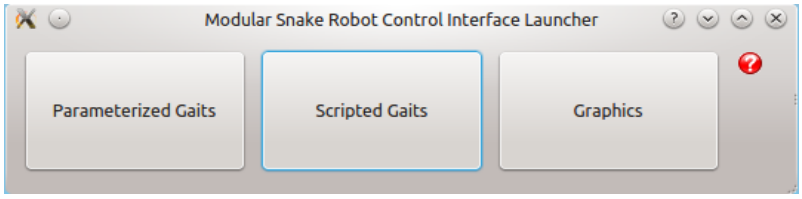

Figura 3.2: Lanzador de Aplicaciones (*Application Launcher*)

Figura 3.3: *About* de la Interfaz Principal

Debido a que son 3 interfaces, se desarrolló una interfaz principal (*application*

launcher) en la cual se escoge a que interfaz se quiere acceder (Figura 3.2). Esta interfaz tiene las 3 opciones de acceso al control del robot serpiente nombradas anteriormente. Además tiene un botón (*About*), que despliega una ventana nueva la cual se puede observar en la figura 3.3. En esta ventana se encuentra la información del programa, como lo es el nombre de la interfaz, la versión de la interfaz, los autores y la licencia bajo la cual fue liberada, tratandose de una aplicación de codigo abierto.

3.1. Descripción de la Operación de la Interfaz

Una vez el usuario ha escogido la interfaz que utilizará, comienza todo el proceso descrito en el diagrama de flujo de la figura 3.1. Si el usuario escoge la interfaz de *scripted gaits* se lanza la interfaz de usuario la cual se observa en la figura 4.1. Esta interfaz de usuario posee unos campos donde se introducen los datos, en este caso, los ángulos deseados. Una vez el usuario ha hecho esto solo debe presionar un botón de ejecución que se denomina *move* por medio del cual el robot comienza a moverse hasta adquirir la configuración programada. Todo este proceso se describirá con detalle más adelante en el capitulo 4.

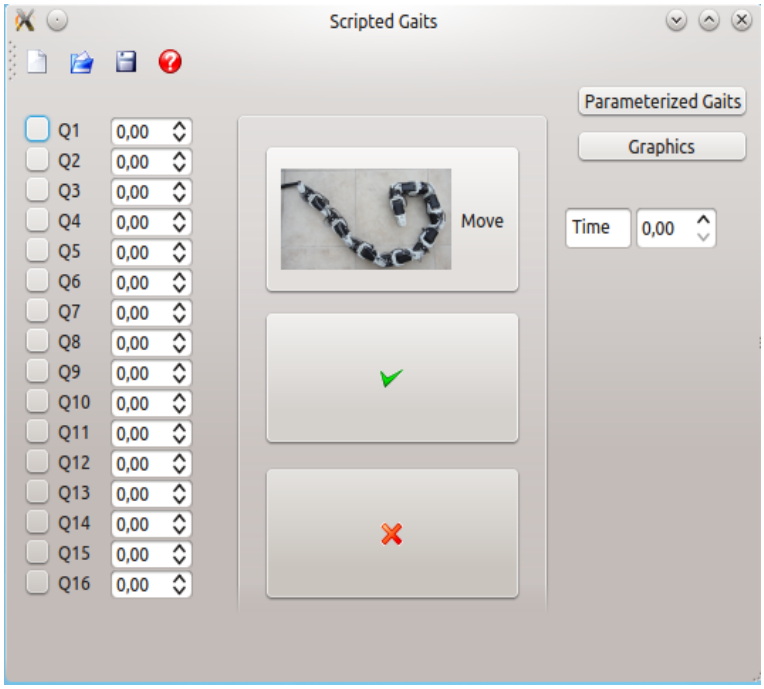

Figura 3.4: Interfaz *Scripted Gaits*

Lo primero que ocurre al presionar el boton de ejecución *move* es que los datos recibidos se almacenan de forma volatil internamente en un arreglo, por

medio del cual son manipulados para lograr transformar la información ingresada por el usuario en información que pueda ser leída por los servomotores (este es el objetivo del *middleware*). Para lograr esto, se utiliza una función que esta descrita por la ecuación 3.1, la cual permite calcular el ángulo a enviar a los servomotores dada la corrección relacionada con el valor del ángulo y el valor digital que ha de programarse realmente en el motor, debido a las especificaciones del fabricante; también se utilizan las funciones de SDK explicadas anteriormente en el marco teorico que son principalmente para obtener comunicación y flujo de información entre el robot y el pc de control.

$$servo_{ang} = (2 * user_{ang} * (-1{,}711) + 512) \tag{3.1}$$

En la ecuación 3.1 $servo_{ang}$ es el valor del ángulo que es enviado a los servomotores y $user_{ang}$ es el valor de ángulo ingresado por el usuario.

Esta función fue encontrada luego de un análisis realizado al inicio de este trabajo a partir de la equivalencia entre el angulo de usuario y el angulo del servomotor (ver figura 3.5).

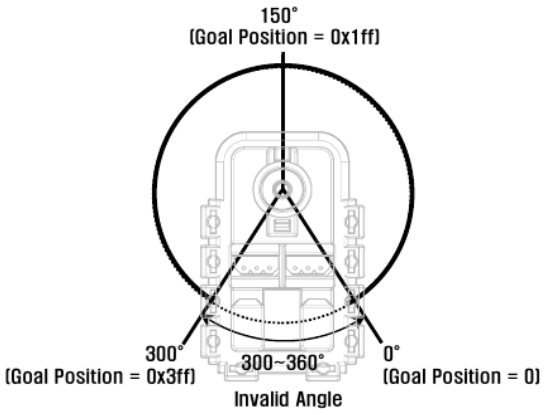

Figura 3.5: Angulos de los Servomotores [11]

Debido a que los angulos en los servomotores varían entre 0° y 300° lo que equivale en decimal entre 0 y 1024, lo primero que se hizo fue hallar el intervalo de angulo (la resolución) como se muestra a continuación

$$Resolucion = 300°/1024 Resolucion = 0{,}294° \tag{3.2}$$

Despues de obtener este resultado, y teniendo en cuenta que los valores decimales van de 0 a 1024, se calculo el valor del angulo de los servomotores en grados multiplicando el valor decimal por el intervalo de angulo

Angulo del Servomotor en grados = (0,294)∗(*Angulo Decimal Servomotor*)

Una vez calculados los angulos en grados del servomotor, se calcularon los angulos correspondientes al usuario. Este proceso partió de la base que el angulo

0 del servomotor equivale al angulo 150 del usuario, lo que significa que entre los dos angulos hay una diferencia de 150°. Con base en esto se halló la ecuación 3.3

$$user_{ang} = (150 - servo_{ang}) \tag{3.3}$$

Luego de esto, sabiendo que la relación entre los ángulos es lineal, se planteo la ecuación de la linea recta 3.4 para de esta manera lograr llegar a la relación entre los ángulos del servomotor y los ángulos del usuario. Para lograr esto se hizo una gráfica de los ángulos del usuario en grados versus los ángulos del servomotor en decimal (ver figura 3.6)para de esta manera sacar la pendiente (m). Ese valor de pendiente sería el factor que convierte de grados a decimal.

$$y = (m * x + b) \tag{3.4}$$

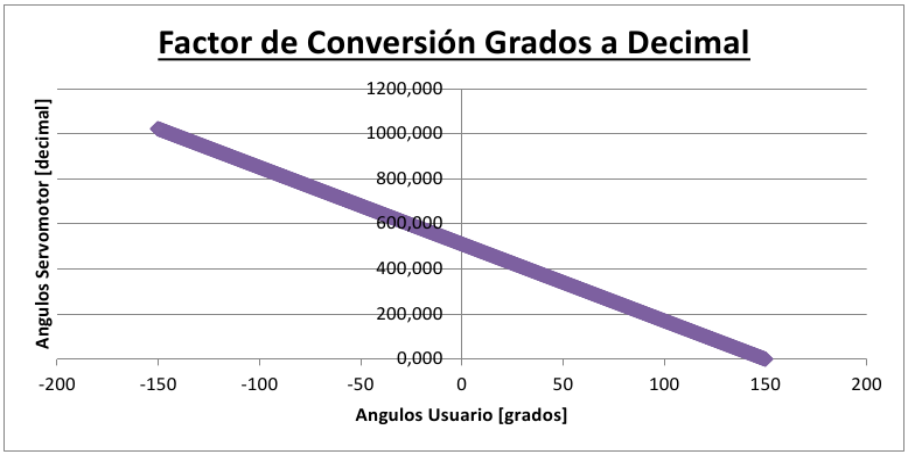

Figura 3.6: Grafica de Conversión Grados a Decimal

Teniendo la grafica se calculó la pendiente y despues de una regresión se obtuvo que el valor de la pendiente es $m = 3{,}42222$. De la misma manera se halló el punto de corte: $b = 150°$ y su equivalente en decimal es 512. Una vez hallados estos valores, la ecuación de la conversión está completa, donde y es el ángulo del servomotor y x es el ángulo del usuario. De esta manera se obtuvo la ecuación 3.1.

Luego de utilizar la función matemática descrita por la ecuación 3.1, se utiliza una función de software, la cual está descrita en el marco teorico, que permite armar o estructurar el paquete de instrucciones y una vez está listo, este se envía a los servomotores. Luego de esto, los servomotores reciben la información y el robot se mueve a sus posiciones deseadas.

Siguiendo con la descripción de la operación de la interfaz, haciendo uso del diagrama de bloques mostrado en la figura 3.1, una vez el robot se mueve, la interfaz permite al usuario manipular los datos suministrados inicialmente. Para

esto la interfaz cuenta con dos opciones: guardar los datos iniciales o descartar los datos (ver diagrama de la figura 3.1). Esto es debido a que el objetivo principal de la interfaz es ayudar al usuario al diseño de gaits. Cuando el usuario está a gusto con la configuración obtenida al enviar los angulos iniciales, es posible guardar estos ángulos en una base de datos la cual puede ser consultada en cualquier momento y sirve de registro de secuencias de movimiento que corresponden a gaits diseñados. Para guardar los datos el usuario solo debe presionar un botón y de esta manera los datos son trasladados automaticamente a la base de datos. Para este caso se escogió crear un archivo `.txt` para funcionar como base de datos, en el cual se organizan los datos que son escogidos para ser almacenados. Cuando el usuario no está a gusto con la configuración programada en el robot, los datos no son almacenados y el robot vuelve automaticamente a la ultima posición guardada; en este momento el usuario debe introducir nuevos ángulos comenzando de nuevo el proceso de *scripted gaits* tal como se muestra en el diagrama de la figura 3.1.

Por otro lado, existe un proceso de adquisición de datos dentro del programa global que puede observarse en el diagrama de la figura 3.1 en color verde, el cual consiste en obtener información de los servomotores, que es útil para el diseñador de *gaits* para entender el comportamiento del robot en diferentes ambientes y situaciones [4]. Este proceso empieza una vez se ha movido el robot, ya que en ese momento por medio del paquete de instrucciones (nombre dado por el fabricante al paquete conformado por la información enviada al robot), se envía al robot la orden de adquirir datos provenientes de los sensores internos de los motores los cuales vienen instalados de fabrica. Los datos recibidos son la velocidad, temperatura, torque y la posición. Luego de que el robot recibe la orden de mandar información, esta es recibida por el computador por medio del paquete de estados (nombre dado por el fabricante al paquete conformado por la información recibida del robot), en el cual esta almacenada toda la información solicitada. Despues de recibida, la información es desempaquetada y enviada a la base de datos en donde se organiza en columnas toda la información recibida. Además es posible almacenar datos externos que en trabajos futuros será necesario almacenar, provenientes de sensores externos como giroscopios, acelerómetros, entre otros, para lo cual esta arquitectura de software puede ser utilizada [6].

Una vez estos datos estan debidamente almacenados y organizados, quedan guardados esperando ser utilizados para graficarse o analizarse. Cuando el usuario escoge la interfaz de graficar, puede graficar los datos deseados, ya sean todas las variables, o sea solo una. Lo unico que debe hacer es indicar que archivo desea graficar y la interfaz lo hará automaticamente. Para esta interfaz se utilizó el llamado a una aplicación *open source* llamada KST que permite graficar en tiempo real; de esta manera al presionar el boton *graphics* automaticamente se ejecuta este programa. Esta parte de graficar los datos será explicada a mayor profundidad más adelante en el capítulo 5.

Capítulo 4

Interfaz Scripted Gaits

Como se mencionó anteriormente, el objetivo de este trabajo es implementar un conjunto de funciones que permitan la programación de movimientos *scripted gaits* y adquisición de variables de los sensores de un *modular snake robot*. Para corroborar el funcionamiento del software fue necesario desarrollar una interfaz que permite y ayuda al usuario a diseñar *scripted gaits*. Dicha interfaz se muestra en la figura 4.1.

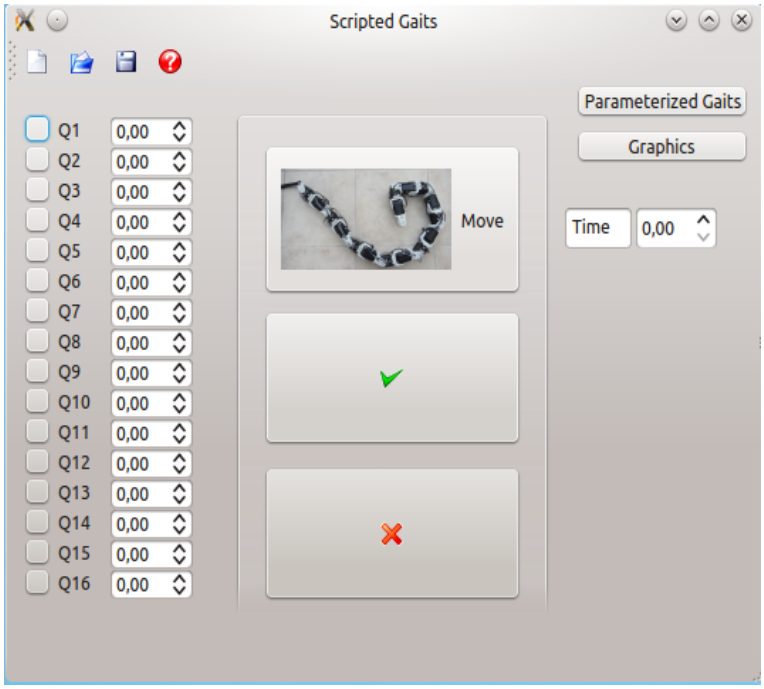

Figura 4.1: Interfaz *Scripted Gaits*

4.1. Descripción de los Componentes de la Interfaz

Al presionar el botón de *scripted gaits* en la interfaz principal (*application launcher*) (ver figura 3.2) se despliega la interfaz mostrada en la figura 4.1. Esta interfaz posee diferentes funciones que facilitan el diseño y la programación de este tipo de *gaits* y ademas permite hacer uso de herramientas para experimentar con estos movimientos.

Para empezar, la interfaz tiene 16 casillas numeradas desde Q1 hasta Q16, como se observa en la figura 4.2, que permiten al usuario seleccionar los modulos del robot que desea activos; de esta manera los modulos que estan activos son los que se van a mover en el robot, mientras que, los inactivos no se mueven y no hacen parte de la configuración del robot.

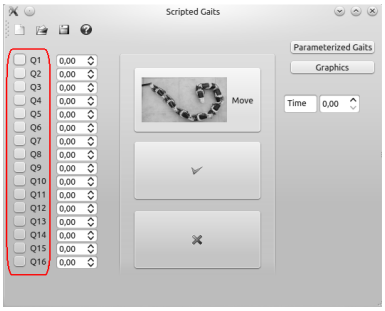

Figura 4.2: Casillas de activación de modulos

Al frente de estas casillas se encuentran 16 celdas, como se observa en la figura 4.3, en las cuales se ingresa el valor del ángulo que se desea enviar al robot. Si la casilla de activación no se encuentra activa, no importa que valor de angulo se ingrese en la celda correspondiente, el robot no se va a mover. Estas celdas reciben valores de ángulo enteros o flotantes.

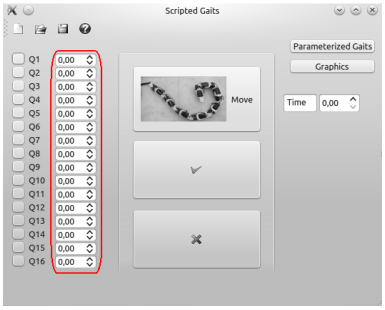

Figura 4.3: Celdas en las cuales se ingresa el valor de angulo

En el lado derecho la interfaz tiene 3 botones: el primero tiene un icono en

forma de robot serpiente y es el boton denominado *Move*, el segundo tiene un simbolo de visto bueno verde (llamado de ahora en adelante "OK") y el otro boton tiene una equis roja (llamado de ahora en adelante cancelado). Estos botones pueden observarse en la figura 4.4. El boton *Move* es utilizado para mover el robot, es decir, al presionar este boton ocurre todo el proceso descrito en el capitulo 3 acerca de la operación del programa en el diseño de *gaits* obteniendo así el movimiento del robot. El boton de visto bueno verde, por su parte, es utilizado para guardar los angulos deseados, es decir, una vez el robot se ha ubicado en la configuración deseada, el usuario puede determinar si la configuración cumple con las expectativas para el fin deseado. Si el usuario aprueba dicha configuración debe presionar el boton de ok y automaticamente se guardan los valores de los 16 ángulos en un archivo `.txt` y de esta manera los ángulos pueden ser cargados en cualquier momento posterior. El tercer y ultimo botón es una equis roja o un simbolo de cancelado el cual descarta las configuraciones que no son utiles para el usuario; de esta manera partiendo del proceso descrito anteriormente, si la configuración obtenida no cumple con las expectativas del usuario este debe presionar el boton de cancelar y el valor de los ángulos se eliminará cargando en la interfaz el valor de los ángulos de la ultima configuración guardada.

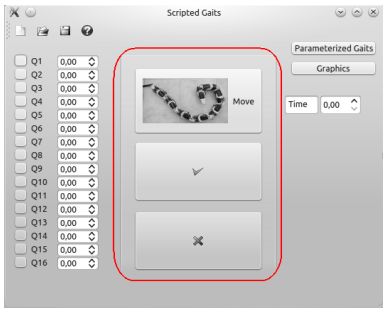

Figura 4.4: Botones de funcionamiento de la Interfaz

La interfaz también cuenta con una barra de herramientas o *Toolbar* la cual está compuesta por 4 botones: archivo nuevo o *New File*, abrir archivo o *Open File*, guardar archivo o *Save File* y el *About* de la interfaz. Esta barra de herramientas se puede observar en la figura 4.5.

El boton *New File* es el boton que crea una nueva sesión de la interfaz, es decir, al presionar este botón la interfaz aparece como se muestra al ejecutar por primera vez el programa: las casillas numeradas desde Q1 a Q16 aparecen sin seleccionar, las celdas donde se introducen los valores de angulo para cada servomotor aparecen en 0 y la casilla *Time*, cuya función se explicará al final de este capitulo, también aparece en 0. El botón *Open File*, por su parte, despliega una ventana por medio de la cual es posible elegir cualquier archivo `.txt` para ser leido. Cuando se escoge el archivo deseado y se presiona *Open*, el programa lee el archivo `.txt` teniendo en cuenta el tiempo ingresado por el usuario en la casilla

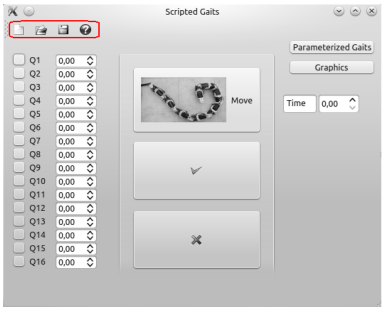

Figura 4.5: Barra de Herramientas de la Interfaz

Figura 4.6: *About* de la Interfaz *Scripted Gaits*

Time y carga los datos a la interfaz; además realiza todo el proceso de convertir y mandar los angulos a los servomotores sin necesidad de presionar el botón *Move*. Por otro lado, el botón *Save File* permite al usuario guardar todos los datos adquiridos durante una sesión para luego utilizarlos en el momento que lo desee. Al presionar este botón, se despliega una ventana que permite darle nombre al archivo que contiene todos los datos obtenidos de los servomotores. De esta manera el usuario puede reutilizar estos datos posteriormente para analizarlos con el fin de entender el movimiento del robot. Finalmente el botón del *About* despliega una ventana nueva la cual se puede observar en la figura 4.6. En esta

ventana se encuentra la información del programa, como lo es el nombre de la interfaz, la versión de la interfaz, los autores y la licencia bajo la cual fue liberada, tratandose de una aplicación de codigo abierto.

Finalmente la interfaz tiene en la parte derecha 2 botones y una celda en la cual es posible introducir valores llamada *Time*. Estos se pueden observar en las figuras 4.7 y 4.8. Adicionalmente y recordando que esta interfaz de *scripted gaits* hace parte de todo un framework de control, los botones de *Parametrized Gaits* y *Graphics* despliegan la interfaz de *Parametrized Gaits*, creada en un trabajo previo por Laura Paez [9], y ejecutan automaticamente el programa para graficar en tiempo real KST, respectivamente. Por ultimo se encuentra la celda en la cual es posible introducir valores llamada *Time*. Los valores introducidos en esta celda determinan el tiempo que tarda el programa en leer cada linea de texto existente en un archivo txt, el cual es leido y cargado a la interfaz utilizando el boton *Open* de la barra de herramientas como se mencionó anteriormente.

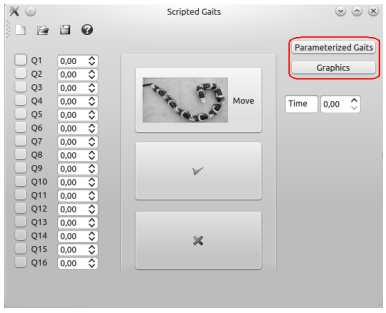

Figura 4.7: LLamado a las otras Interfaces

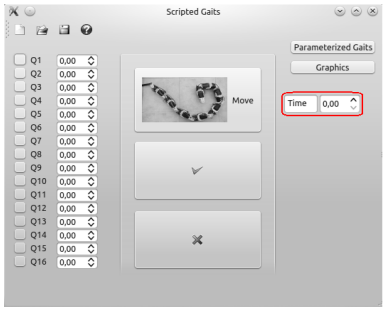

Figura 4.8: Time de la Interfaz

4.2. Descripción del conjunto de funciones de la interfaz

Para que todo el proceso descrito anteriormente funcione como es debido, tal como se explicó, el software se desarrolló por medio de la programación

orientada a objetos en el lenguaje de programación de alto nivel C++.

4.2.1. Clases

El codigo del conjunto de funciones que soportan la interfaz de *Scripted Gaits* como *front end* está conformado por dos clases las cuales se explican a lo largo de esta sección con sus respectivos atributos y métodos. Este codigo se puede ver en el anexo 1.

Clase Configuration

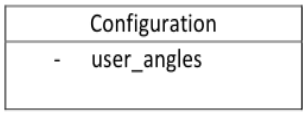

Figura 4.9: Diagrama de la clase Configuration

La clase `Configuration` es la clase diseñada para manipular la información ingresada por el usuario, es decir, los valores de ángulo ingresados por el usuario son manipulados por medio del objeto `config` perteneciente a esta clase. El objeto `config` posee unos atributos por medio de los cuales cumple con la función requerida que es recibir y guardar los ángulos ingresados por el usuario. El atributo de la clase `Configuration` se denomina `user_angles`. Este atributo se utiliza para almacenar y manipular los valores de ángulo ingresados por el usuario.

Clase Package

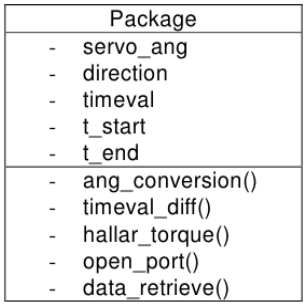

Figura 4.10: Diagrama de la clase Package

La clase `Package` es la clase diseñada para manejar toda la comunicación entre el robot y el computador, es decir, en esta clase se construye y envía el paquete de instrucciones al robot por medio del cual se especifica la configuración deseada; además, por medio de esta se realiza la adquisición de datos

provenientes del robot, los cuales serán objeto de análisis por parte del usuario. El objeto de esta clase se denomina `pack`.

Los atributos del objeto `pack` son:

- `servo_ang`: Este atributo se utiliza para almacenar los valores de ángulo que van a ser enviados al robot después de pasar por su respectiva conversión la cual se logra por medio de uno de los metodos de esta clase que se explicará más adelante en este capitulo.
- `direction`: Este atributo permite saber en que dirección se mueve cada servomotor del robot. Es muy importante ya que dependiendo de este valor se ajusta el valor de los datos provenientes del robot.
- `timeval`: Este atributo es utilizado para almacenar el tiempo de rutina que toma el robot.
- `t_start`: Este atributo es utilizado para almacenar los tiempos de referencia para calcular el tiempo de rutina que toma el robot.
- `t_end`: Este atributo es utilizado para almacenar los tiempos de referencia para calcular el tiempo de rutina que toma el robot.

Por otra parte, los métodos del objeto `pack` se muestran a continuación.

- `ang_conversion`: Este método se utiliza para hacer la conversión de ángulo ingresado por el usuario al valor a enviar a los servomotores, ya que el angulo 0° del usuario equivale a 150° (valor en decimal 512) en los servomotores (ver figura 4.11).

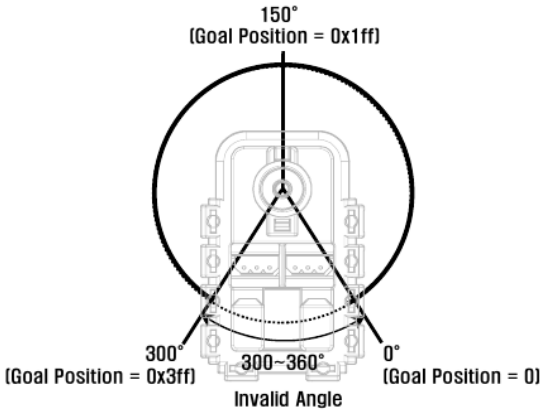

Figura 4.11: Equivalencia de Angulos en los Servomotores [11]

Esta conversión se realiza por medio de la ecuación 4.1 la cual fue explicada en la descripción general del proyecto.

$$servo_{ang} = (2 * user_{ang} * (-1{,}711) + 512) \tag{4.1}$$

- `timeval_diff:` Este método se utiliza para establecer la cantidad de tiempo que tarda el robot en realizar una determinada secuencia de movimiento, utilizando los atributos `t_start` y `t_end` a manera de *timer*, ya que al hacer este conjunto de funciones portable a varios sistemas operativos y computadores diferentes, no se puede confiar en el temporizador de la CPU. Cuando el robot va a empezar la rutina, `t_start` guarda la hora actual del reloj de la CPU del PC de control; luego de terminar la rutina, el atributo `t_end` guarda la hora actual del mismo reloj. Luego de esto se realiza una resta de `t_end` y `t_start` para saber cual fue el tiempo total en milisegundos (ms) que tardo el robot en ejecutar determinada secuencia de movimiento.
- `torque:` Es un método cuya función es hallar la carga LOAD total del robot dependiendo de la dirección de este, es decir, si la dirección va contra las manecillas del reloj, la dirección es 0; si por el contrario la dirección va en el sentido de las manecillas del reloj, la dirección es 1. Esto se observa en la figura 4.12.

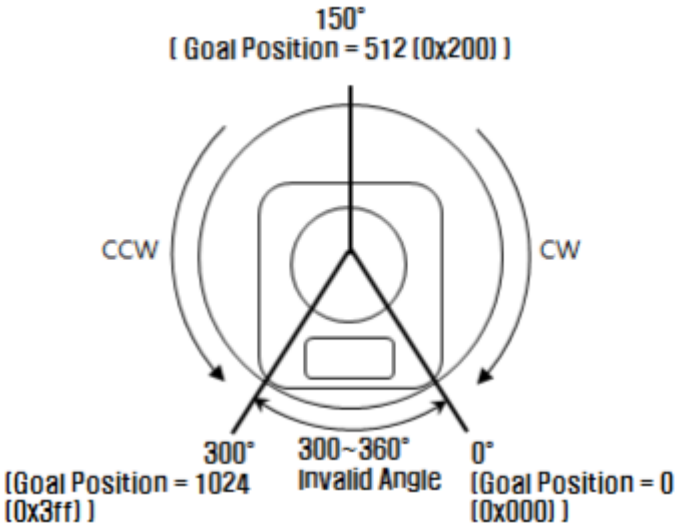

Figura 4.12: Dirección del giro de un servomotor [11]

En caso de que la dirección sea 0 quiere decir que el valor de la carga LOAD esta entre 0 y 1024 por lo tanto la función retornará el mismo valor; en caso de que la dirección sea 1 quiere decir que el valor esta entre 1024 y 2047, por lo tanto la función retornará la resta del valor actual con 1024.

Es importante tener en cuenta que la carga LOAD representa una medida indirecta del torque la cual viene dada por el fabricante [18]. Esta variable da un indicio de que tanto torque aplica el servomotor a los engranajes pero no es una medida directa de torque sobre los ejes [19].

- `open_port:` Este método permite establecer la comunicación entre el PC de control y el *modular snake robot* mediante `dxl_initialize(devIndex,baudnum),` una función del API de dynamixel. Además en este método se construye el paquete de instrucciones a enviar mediante funciones tomadas del API, las cuales fueron explicadas en el capitulo 2, y una vez se termina de construir el paquete de instrucciones se envía utilizando la función `dxl_txrx_packet().` Las funciones utilizadas por este método provienen del API de dynamixel y están explicadas en el marco teórico del presente documento.
- `data_retrieve:` En este método se crea un archivo de texto llamado `datos_recibidos.txt` en donde se guardan los datos provenientes del robot. Para adquirir los datos se envía un paquete de instrucciones especificando la función a leer y también se especifican los datos a leer. Para este caso es la posición, la velocidad, la carga (medida indirecta del torque ejercido por el motor) y la temperatura. Una vez enviado este paquete, el robot retorna los datos pedidos y los guarda en el archivo de texto `datos_recibidos.txt`. Tanto para enviar datos al robot como para recibir información se utilizan las funciones disponibles en el API de dynamixel las cuales fueron explicadas en el capitulo 2.

El codigo escrito para estas clases, atributos y métodos puede ser observado en el anexo 1.

4.3. Sumario

Como se mencionó en la introducción el primer objetivo específico de este trabajo de grado es implementar un conjunto de funciones en un lenguaje de programación de alto nivel, que permitan el envío de datos de posición a un conjunto de 16 servomotores Dinamixel AX-12. Este objetivo se explica en el capitulo 4 en la sección 4.2 en la cual se habla del envío de datos de posición a los servomotores Dinamixel AX-12 utilizando el método `open_port` , implementado en el codigo en el anexo 1.

El segundo objetivo específico, por su parte, es implementar un conjunto de funciones en un lenguaje de programación de alto nivel, que permitan la recepción de datos provenientes de los sensores integrados a los servomotores Dinamixel AX-12. Este objetivo se explica en el capitulo 4 en la sección 4.2 en la cual se habla de la recepción de datos provenientes de los sensores de los servomotores Dinamixel AX-12 utilizando el método `data_retrieve`, implementado en el codigo en el anexo 1.

Por otro lado, el tercer objetivo específico es implementar un conjunto de funciones en un lenguaje de programación de alto nivel, que permitan almacenar en una estructura de datos la información proveniente de los servomotores

Dinamixel AX-12. Este objetivo se explica en el capitulo 4 en la sección 4.2 en la cual se explica como se almacenan los datos recibidos de los servomotores en una estructura de datos llamada `datos_recibidos.txt`, implementado en el codigo en el anexo 1.

Finalmente el cuarto objetivo específico es determinar un protocolo de pruebas para verificar el funcionamiento de las funciones implementadas. Este objetivo se desarrollará más adelante en el capitulo 6.

Al cumplir estos objetivos se cumple el objetivo general que es implementar un conjunto de funciones en un lenguaje de programación de alto nivel, que permitan la programación de movimientos (scripted gaits) y adquisición de variables de los sensores de un Modular Snake Robot.

Capítulo 5

Graphics

Como se menciono en el capitulo anterior, las interfaces de control del robot serpiente son 3: *Parametrized Gaits*, la cual fue diseñada en un trabajo previo por Laura Paez [9]; *Scripted Gaits*, la cual fue desarrollada en este trabajo y explicada en los capitulos anteriores; y *Graphics*, la cual se desarrolló mediante la utilización de una aplicación *open source* llamada KST que permite graficar en tiempo real.

KST es una herramienta que permite la visualización de datos en tiempo real y además cuenta con una funcionalidad integrada de análisis de datos. La versión de KST utilizada en este trabajo es la 2.0.3, licenciada bajo GPL y está disponible para diferentes plataformas como lo son Microsoft Windows, Linux, Mac OSX. Mas información sobre esta herramienta se puede ver en [20].

5.1. Descripción del Funcionamiento de KST

Para acceder a esta herramienta, el usuario debe presionar el boton llamado *Graphics* que se encuentra en el lanzador de aplicaciones (ver figura 3.2) o el boton que se encuentra en la interfaz *scripted gaits* (ver figura 4.7).

Al presionar una de estas dos opciones se ejecuta el programa KST y se abre la ventana que se muestra en la figura 5.1.

Una vez aparece la pantalla de inicio de KST el usuario debe introducir los datos que desea graficar. Para lograr esto debe presionar el boton *Data Wizard* (ver figura 5.2) por medio del cual se despliega otra ventana (ver figura 5.3) en la cual se debe introducir el archivo que contiene los datos que se desea graficar. Este archivo esta ubicado en la carpeta *MSR-built-desktop* la cual es creada por *Qt Creator* y es en esa carpeta donde se guardan los archivos de datos provenientes del software desarrollado. Al seleccionar el archivo aparece un campo debajo el cual dice el tipo de archivo, para este caso es ASCII (lo que significa que es un archivo de texto) y al lado derecho se encuentra un boton de configuración en el cual se configura el formato en el cual se encuentra el archivo de datos insertado. Generalmente la configuración que esta por defecto es la

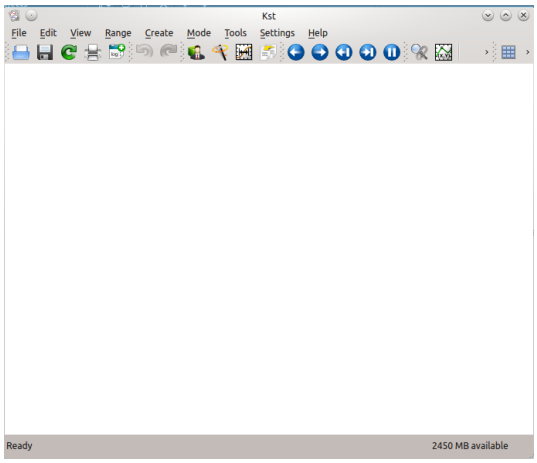

Figura 5.1: Pantalla Inicio KST

adecuada para que KST lea el archivo seleccionado sin ningun inconveniente, sin embargo, es importante saber que si se presenta algún inconveniente se puede solucionar revisando la configuración predeterminada.

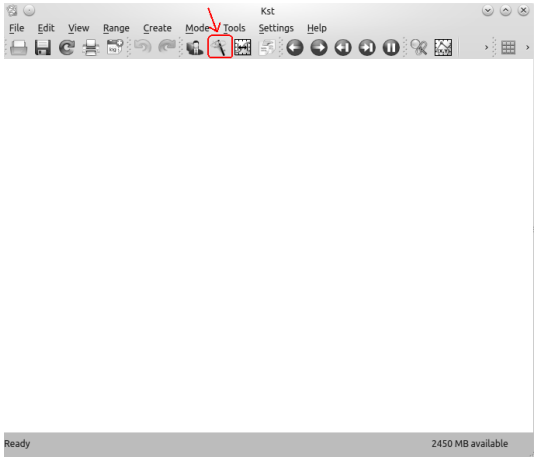

Figura 5.2: Boton Data Wizard KST

Una vez se especifica el archivo que posee los datos a graficar, el cual puede estar ubicado en cualquier path del computador (por defecto de la aplicación, los archivos de texto se encuentran en la carpeta *MSR-built-desktop* la cual es creada por *Qt Creator*), se debe presionar el boton *Next*. Al hacer esto, aparece una nueva ventana (ver figura 5.4) en la cual se debe especificar las columnas que se desea graficar. Tambien aparece un termino INDEX el cual es un vector creado por KST que puede ser utilizado como datos del eje X cuando el archivo

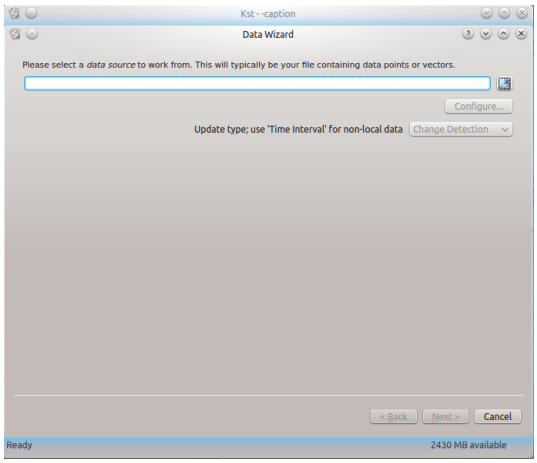

Figura 5.3: Data Wizard KST: Especificación de Archivo a Graficar

de texto no proporciona el vector de datos para el eje X. El vector INDEX contiene enteros de 0 hasta N-1 donde N es el numero de filas que tiene el archivo de texto. El vector INDEX no debe ser seleccionado para ser graficado en ningun caso, ya que ha sido creado para asignarle los valores al eje X de la gráfica, mientras que si deben ser seleccionadas las columnas del archivo de texto.

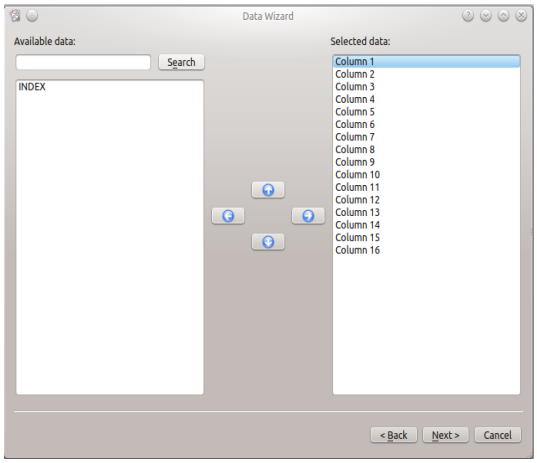

Figura 5.4: Data Wizard KST: Elección de Columnas a Graficar del Archivo de Texto

Luego de haber seleccionado las columnas a graficar, se debe presionar el boton *Next*. Al hacer esto, aparece otra ventana en la cual es posible configurar la presentación de los datos (ver figura 5.5), es decir, los parametros para

generar la grafica. El primer campo se denomina *Start* y hace referencia a la fila en el archivo de la cual se empiezan a leer los datos. El siguiente campo se denomina *Range*, que es el numero total de filas a leer del archivo. Tambien se encuentra una casilla denominada *Count from end* la cual lee el numero de filas especificadas en *Range* comenzando desde el final del archivo de texto. Se debe seleccionar el campo que dice *Create XY plots* y luego de esto se debe especificar que el eje X se genere a partir del vector INDEX.

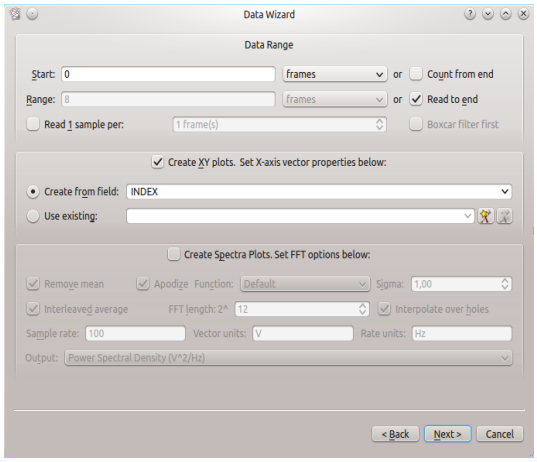

Figura 5.5: Data Wizard KST: Especificación de Parámetros para Crear la Gráfica

Una vez se han configurado estos parametros, se debe presionar el boton *Next* lo cual genera una nueva ventana (ver figura 5.6) en la cual se configuran los parametros de presentación de la grafica como lo son el estilo de la curva, el titulo de la grafica, los nombres de los ejes X y Y, en donde poner las graficas, es decir, si se generan las graficas de todas las columnas en una sola grafica o por separado.

Una vez se han configurado todos los parametros se presiona el boton *Finish*. Al presionar este boton se genera automaticamente la grafica. En la figura 5.7 se puede ver un ejemplo de la grafica de las 16 columnas en una sola ventana y en la figura 5.8 se puede ver un ejemplo de la grafica de una sola columna. Los datos utilizados para generar estas gráficas fueron obtenidos de los servomotores de movimientos realizados.

En los ejemplos anteriores, los datos tomados para el eje X de las gráficas son generados por el vector INDEX. Esto es porque algunos archivos de texto no tienen una columna que represente el vector que debe graficarse en el eje X y cuando esto sucede, KST crea el vector INDEX para que funcione como el vector a graficar en el eje independiente. Sin embargo es posible utilizar otro archivo donde se tengan datos que conformen este vector y especificarlo en el campo llamado *use existing* (ver figura 5.9).

Para este trabajo de grado se introduce la base de datos en la cual se al-

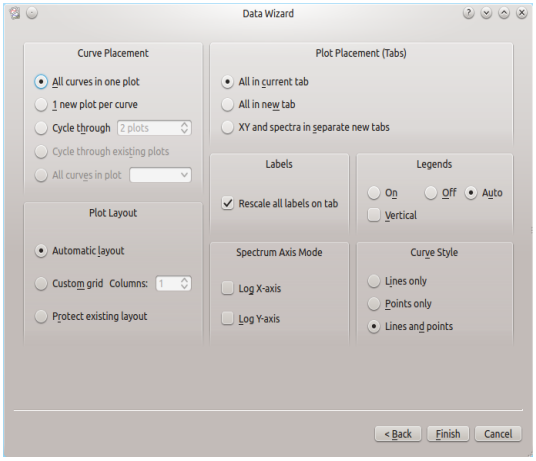

Figura 5.6: Data Wizard KST: Especificación de Parámetros para la Parte Estetica de la Gráfica

macenan los datos obtenidos de los servomotores, que en este caso se llama `datos_recibidos.txt`, y se grafican contra el tiempo. Esta base de datos no contiene el vector que se debe asignar al eje X, por lo tanto utilizamos el vector INDEX para graficar las diferentes variables versus el tiempo. Como se puede ver KST es un software muy útil para graficar y analizar diferentes bases de datos. Es por esto que se utilizó para este trabajo de grado ya que para realizar el análisis de los datos adquiridos de los servomotores se necesita graficar las diferentes variables contra el tiempo para entender el comportamiento del robot.

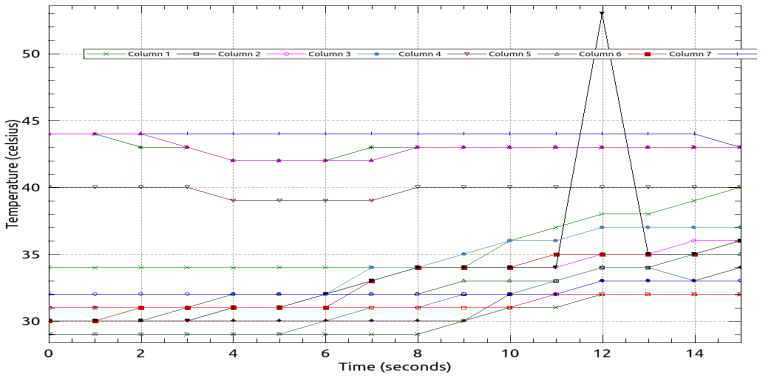

Figura 5.7: Grafica 16 Columnas KST. Temperatura en 16 servomotores

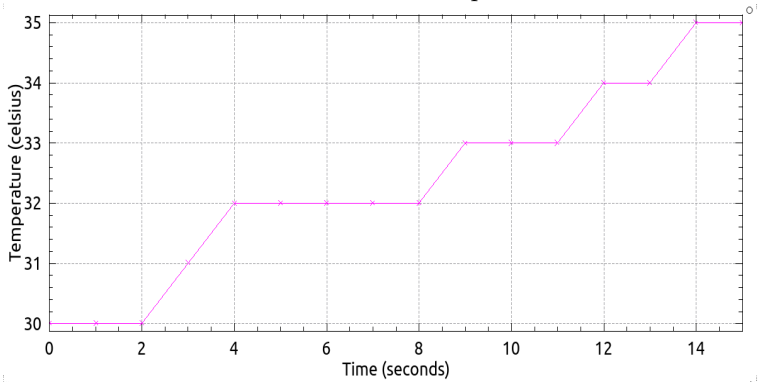

Figura 5.8: Grafica 1 Columna KST. Temperatura en un servomotor

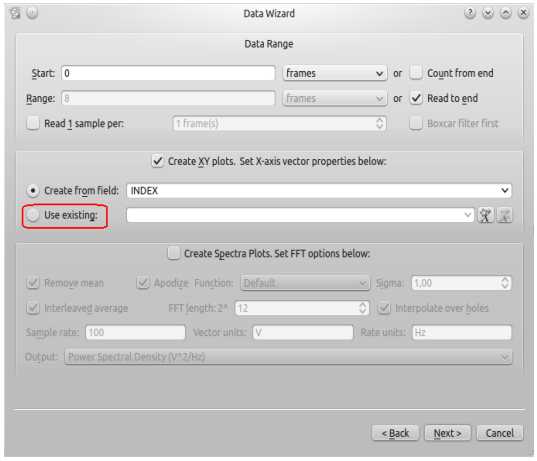

Figura 5.9: Data Wizard KST: Especificación Ubicación Vector INDEX

Capítulo 6

Protocolo de Pruebas

En este capitulo se documenta el proceso de experimentación. Para esto se definen los parámetros que se usan para medir el desempeño del robot. Luego de esto se establece el protocolo de pruebas que se desea implementar para comprobar el correcto funcionamiento del software desarrollado en el presente trabajo de grado, se ejecutan las pruebas y se analizan los resultados.

6.1. Diseño, Ejecución y Resultados del Protocolo de Pruebas

Para diseñar el protocolo de pruebas a implementar es necesario hacer un recuento de los objetivos propuestos en este trabajo de grado ya que el protocolo tiene como objetivo demostrar el correcto funcionamiento del software desarrollado.

Las pruebas propuestas a ejecutar se observan a continuación.

1. Datos recibidos por la interfaz (permite introducir ángulos escogidos por el usuario para ser ejecutados por el robot).
2. Envío de una trama de datos.
3. Recepción de datos provenientes de los servomotores.
4. Ejecución del programa que permite graficar los datos para su posterior análisis.
5. Analizar el esfuerzo realizado por el MSR a partir de la carga LOAD recibida de los servomotores.

a. Datos recibidos por la interfaz

Para comprobar el correcto funcionamiento del software desarrollado fue necesario desarrollar una interfaz que permite y ayuda al usuario a diseñar

scripted gaits. Es por esto que es importante asegurar el correcto funcionamiento de dicha interfaz. Para esto se debe realizar una prueba del ingreso de datos elegidos por el usuario para ser enviados a los servomotores.

La forma de verificar que la interfaz de *Scripted Gaits* recibe efectivamente los datos y los almacena para ser enviados a los servomotores, es introducir una serie de datos y al presionar el botón *Move*, que es el botón con el cual se empieza todo el proceso descrito en los capítulos anteriores, se revisa la salida del código utilizando una función que esta escrita en C++ (y se encuentra dentro de la librería `iostream`) la cual permite mostrar los datos que maneja el codigo. Esto se logra mediante el uso de un comando llamado `cout` el cual se encarga de la salida de datos del programa. De esta manera es posible conocer el valor de ángulos ingresados al programa. El codigo que hace esto se puede observar en el anexo 1.

Para ejecutar la prueba es necesario escoger un valor de ángulo para los 16 servomotores. Para este caso se escogieron los siguientes valores:

$AngulosIngresados = [10\ 10\ 10\ 10\ 10\ 10\ 10\ 10\ 10\ 10\ 10\ 10\ 10\ 10\ 10\ 10]$

Se introducen los valores de ángulo en la interfaz como se muestra en la figura 6.1. Luego de esto se presiona el botón *Move* y despues se revisa el reporte que entrega el programa.

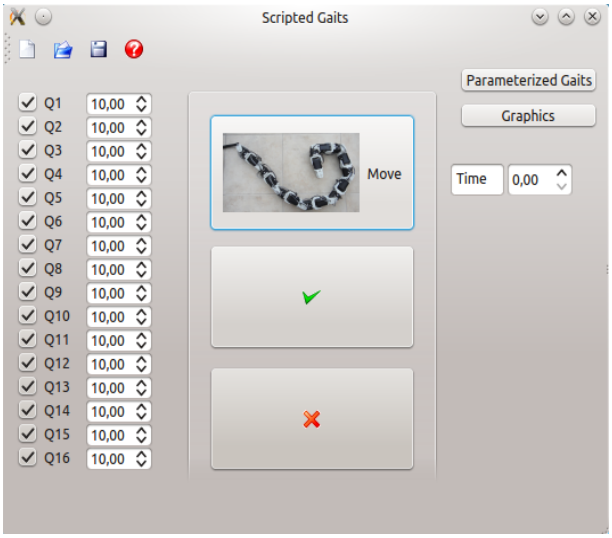

Figura 6.1: Ingreso de Datos en la Interfaz

Una vez se tienen los datos proporcionados por el programa, se verifica que la interfaz recibe efectivamente los datos y los almacena para ser enviados a los servomotores mediante la revisión y la comparación de los ángulos ingresados a la interaz con los ángulos arrojados por el programa. Si el valor de ángulo arrojado por el código es el mismo que el usuario introdujo se comprueba que la interfaz si recibe y envía los ángulos correctamente.

Los valores obtenidos mediante el reporte del programa se pueden ver en la figura 6.2 donde el numero ingresado corresponde al numero que está al lado de la palabra Entero y el equivalente a enviar al robot es el numero que se encuentra en la linea siguiente del numero ingresado. Este numero se obtiene utilizando la ecuación explicada en los capitulos anteriores.

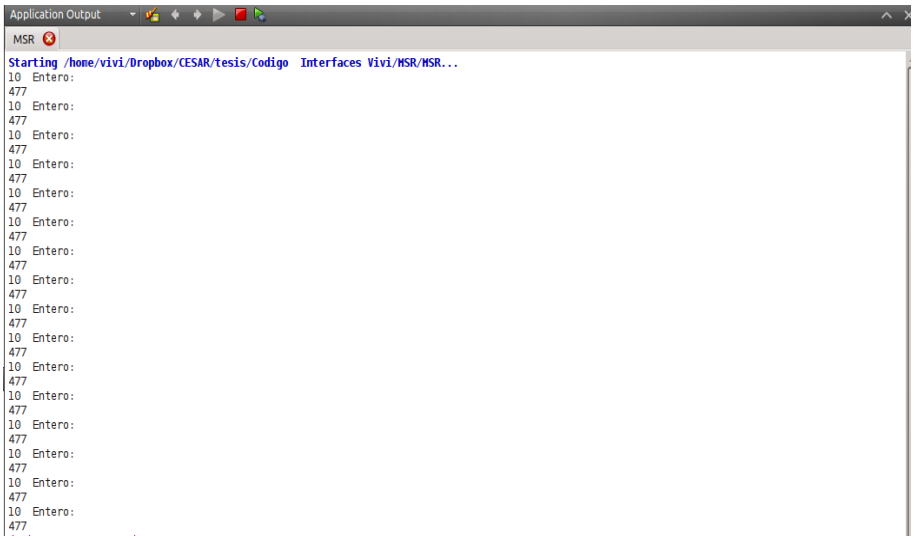

Figura 6.2: Reporte Generado por el Programa

Teniendo ya los datos obtenidos por el reporte generado por el programa, se compararan con los datos ingresados por el usuario(figura 6.1) comprobando así que son los mismos. Con esta prueba quedo demostrado que la interfaz recibe y almacena correctamente los datos ingresados por el usuario para luego ser procesados y enviados a los servomotores del *Modular Snake Robot*.

b. Envío de una trama de datos

Con esta prueba se pretende comprobar el correcto funcionamiento del software desarrollado en relación al primer objetivo específico del presente trabajo de grado. La forma de verificar dicho funcionamiento es introducir la trama de datos que se desea enviar al robot para que este lo ejecute y se presiona el botón *Move*, con el cual inicia todo el proceso explicado en los capitulos anteriores. Al hacer esto el robot debe moverse y de esta manera se revisan los valores de angulo que efectuó el robot.

Para ejecutar esta prueba se diseñó un movimiento paso a paso con el cual se mueven los servomotores uno a uno con el fin de llegar a la configuración deseada. Los valores de ángulo enviados a los servomotores se muestran a continuación.

$AngulosIngresados = [-45\ -45\ -45\ -45\ -45\ -45\ -45\ -45\ -45\ -60\ -60\ 45\ -30\ 45\ -45\ -90]$

Estos angulos fueron ingresados (como se muestra en la figura 6.3) y enviados, presionando el botón *Move*, uno a uno al robot para así verificar el movimiento de cada servomotor.

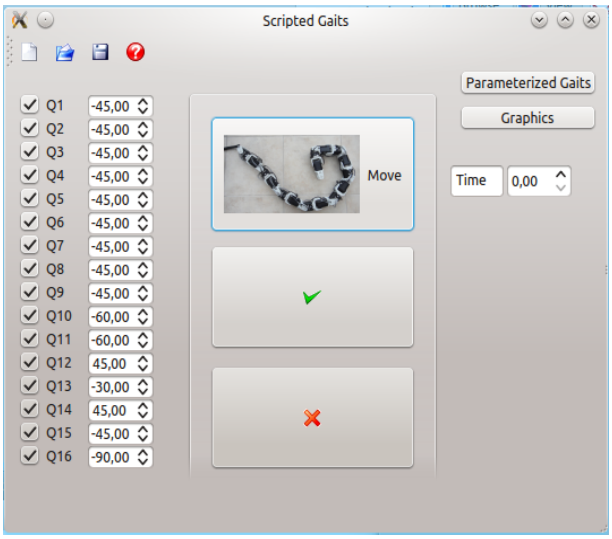

Figura 6.3: Ingreso de Datos en la Interfaz

El movimiento realizado por el robot paso a paso se puede observar en el video anexo 2. Al observar el video se muestra claramente el movimiento de cada servomotor y es coherente con el movimiento que se espera. En la figura 6.4 se observa la configuración final obtenida al introducir la secuencia de ángulos seleccionada.

Figura 6.4: Configuración Obtenida

El movimiento programado en esta prueba se ejecutó a diferentes velocidades, es decir, se enviaron valores de ángulo uno a uno a los servomotores como se mencionó anteriormente, se enviaron valores de ángulo para dos servomotores a la vez (ver video anexo 3) y finalmente se enviaron valores de ángulo para cuatro servomotores a la vez (ver video anexo 4). Al observar los videos se ve la diferencia de velocidad cuando se mueven varios servomotores a la vez.

Al observar la configuración final obtenida se puede concluir que el software desarrollado permite el envío de tramas de datos al robot modular Lola-OP^{TM}y de esta manera se logra dar una configuración al *Modular Snake Robot*.

c. Recepción de datos provenientes de los servomotores

El objetivo de esta prueba es asegurar el funcionamiento de la recepción de datos provenientes de los servomotores. Para comprobar el correcto funcionamiento del software desarrollado en relación con el segundo objetivo del presente trabajo de grado es necesario hacer una prueba de recepción de datos provenientes de los servomotores Dynamixel del MSR.

La forma de verificar la recepción de datos provenientes de los servomotores, es introducir una serie de datos y presionar el botón *Move*. Al hacer esto el robot se mueve a la configuración programada por el usuario y el software desarrollado pide internamente los datos de temperatura, carga, velocidad y posición. Estos datos se almacenan en una base de datos (la cual hace referencia al tercer objetivo específico del presente trabajo de grado) que en este caso es un archivo `.txt`.

Para ejecutar la prueba se utiliza la misma configuración programada para comprobar el envío de datos al robot. A continuación se muestran los valores de angulo utilizados.

$AngulosIngresados = [-45\ -45\ -45\ -45\ -45\ -45\ -45\ -45\ -45\ -60\ -60\ 45\ -30\ 45\ -45\ -90]$

Se introducen los valores de ángulo en la interfaz como se muestra en la figura 6.3. Luego de esto se presiona el botón *Move* y se espera que el robot ejecute los movimientos programados. Luego se revisa la base de datos que para este caso se denomina `datos_recibidos.txt` y está ubicada en la carpeta en la cual se encuentra guardado el software desarrollado. Una vez ubicada la base de datos, se revisa y se verifica que se encuentren los valores correspondientes a cada servomotor. La figura 6.5 muestra la base de datos en la cual se almacenan los datos provenientes de los servomotores.

En la figura se puede ver un texto conformado por varias columnas. En cada columna se tiene un dato proveniente de los servomotores. La primera columna es el valor del id de cada servomotor, la segunda columna es la dirección hacia la cual gira el servomotor, la tercera columna es la carga (LOAD) presente en los servomotores, la cuarta columna es la velocidad a la cual se movió cada servomotor, la quinta columna es la temperatura presente en los servomotores, la sexta columna es el valor de la posición de los servomotores sin convertir, es decir, sin ser pasados por la ecuación expuesta en capitulos anteriores, y la septima y ultima columna es el valor de la posición de los servomotores

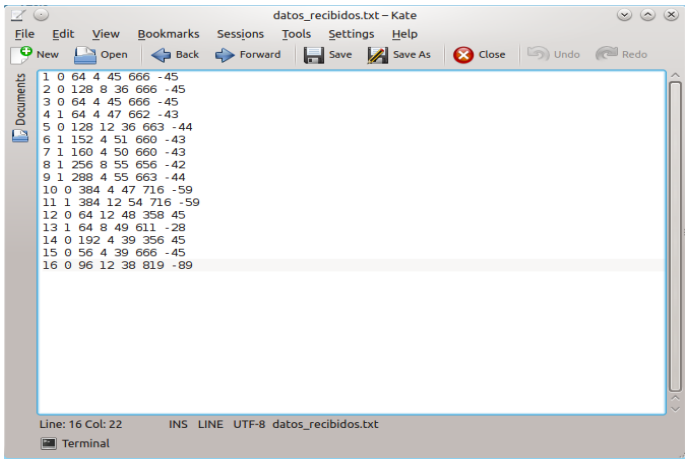

Figura 6.5: Archivo de Texto que funciona como Base de Datos para el Registro de la información proveniente del Robot

convertido, es decir, despues de utilizar la ecuación para saber el valor de ángulo en el rango de valores del usuario.

Los parámetros registrados en la base de datos son importantes para monitorear el funcionamiento de los servomotores del MSR. La dirección es importante debido a que dependiendo de su valor, se determina el valor de la carga (LOAD). Si el valor detectado de dirección es 0 significa que el servomotor giró en contra de las manecillas del reloj, mientras que si el valor de dirección es 1 significa que el servomotor giró en la dirección de las manecillas del reloj. Esto es mirando el servomotor de frente tal como se muestra en la figura 6.6. Con base en esta información se puede saber el valor de la carga ya que si la dirección es 0 el valor de la carga esta entre 0 y 1024, mientras que si la dirección es 1 el valor de la carga esta entre 1024 y 2047.

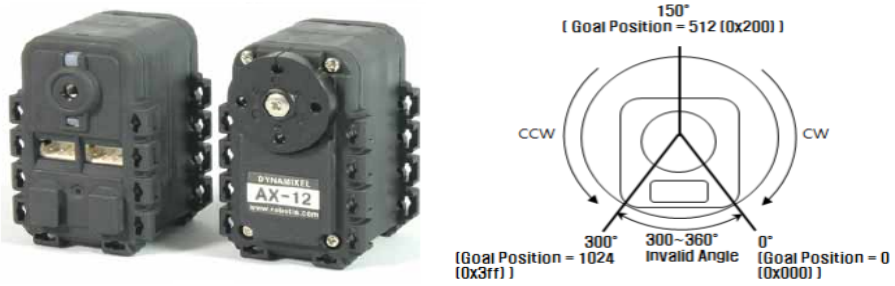

Figura 6.6: A la izquierda está el Servomotor Dynamixel AX-12. A la derecha funcionamiento del Servomotor AX-12

Es importante recordar que la carga LOAD representa una medida indirecta

del torque la cual viene dada por el fabricante [18]. Esta variable da un indicio de que tanto torque aplica el servomotor a los engranajes pero no es una medida directa de torque sobre los ejes [19].

Al observar la figura 6.5 se puede ver que se reciben los datos provenientes de los servomotores y se almacenana en la base de datos designada para esto. Una de las maneras que puede utilizarse para comprobar que los datos recibidos son correctos es coger el valor de ángulo ingresado por el usuario y compararlo con la posición recibida de los servomotores. En el archivo se puede ver que hay algunos valores de ángulo que no concuerdan por 1 o 2 grados de error del valor ingresado por el usuario. Esto se debe al funcionamiento interno de los servomotores ya que existe una curva llamada *Compliance* la cual describe la relación entre el torque de salida y la posición del motor (ver figura 6.7). *Compliance Margin* es el error entre la posición objetivo y la posición actual. Entre mayor sea el valor de ángulo de la posición objetivo, mayor es el error, es decir, es lineal. *Slope Margin* establece el nivel de torque cerca a la posición objetivo.

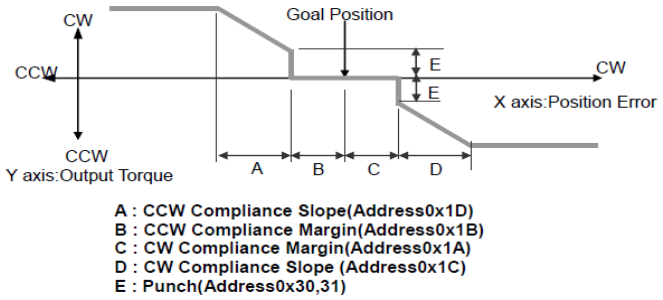

Figura 6.7: Compliance Curve. Curva proporcionada por Robotis

En la tabla de la figura 6.8 se hace una comparación entre el valor de ángulo ingresado por el usuario y el valor de ángulo que retorna el robot para los 16 servomotores en una prueba típica. Adicional se calcula el error en cada caso.

d. Ejecución del programa que permite graficar los datos para su posterior análisis

El objetivo de esta prueba es demostrar la aplicación del software desarrollado en el presente trabajo de grado. Esto se hace utilizando el programa kst el cual se ejecuta al presionar el botón *Graphics* en el lanzador de aplicaciones y se introduce el archivo de texto creado en base a la base de datos que almacena los datos provenientes de los servomotores. Luego de esto se genera la gráfica.

Para ejecutar la prueba se utilizan los datos adquiridos con la configuración programada en la prueba anterior. Para este caso la variable a graficar es la temperatura que manejaron los 16 servomotores durante la ejecución del movimiento completo. Estos valores se escogen y se almacenan en otro archivo de texto que sera el ingresado en kst para ser graficado. El archivo a graficar se denomina

DATOS ENVIADOS (POSICIÓN DE LOS SERVOMOTORES)	DATOS RECIBIDOS (POSICIÓN DE LOS SERVOMOTORES)	ERROR (±3°)
-45°	-45°	0°
-45°	-45°	0°
-45°	-45°	0°
-45°	-43°	2°
-45°	-44°	1°
-45°	-43°	2°
-45°	-43°	2°
-45°	-42°	3°
-45°	-44°	1°
-60°	-59°	1°
-60°	-59°	1°
45°	45°	0°
-30°	-28°	2°
45°	45°	0°
-45°	-45°	0°
-90°	89°	1°

Figura 6.8: Comparación entre los ángulos enviados y los ángulos recibidos moviendose de 0 a 45 grados

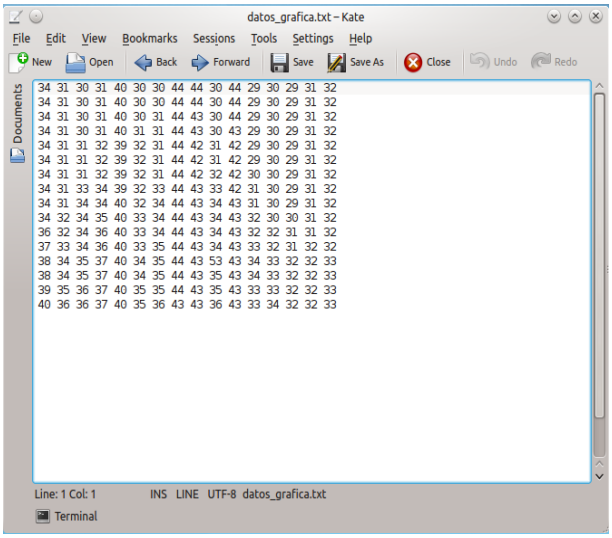

Figura 6.9: Archivo a ingresar en KST

`datos_grafica.txt` y se puede ver en la figura 6.9. Este archivo se encuentra ubicado en la carpeta donde se guarda el software desarrollado.

Una vez se tiene listo el archivo de texto se importa en KST y se genera la grafica de la temperatura con respecto al tiempo. La gráfica resultante se puede observar en la figura 6.10

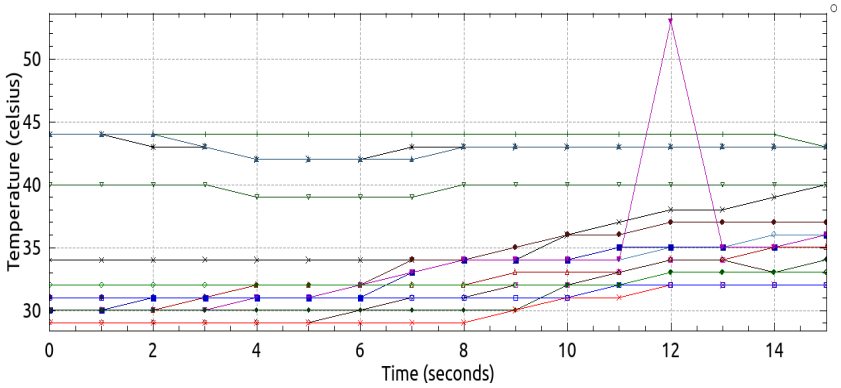

Figura 6.10: Gráfica de la Temperatura de los 16 Servomotores con respecto al Tiempo

En la gráfica se observa la temperatura de cada servo durante el tiempo de ejecución de todo el movimiento. Generar esta gráfica es útil para el programador de *gaits* ya que puede observar el comportamiento de las diferentes variables de los servomotores durante la ejecución de un movimiento programado. Debido a esto el software desarrollado en este trabajo de grado permite graficar datos utilizando el programa KST.

e. Análisis del esfuerzo realizado por el MSR a partir de la carga LOAD recibida de los servomotores

El objetivo de esta prueba es demostrar la utilidad del software desarrollado en el presente trabajo de grado. Esto se hace mediante la utilización de los datos provenientes de los servomotores del MSR para realizar un análisis de la cantidad de energía gastada por el robot y de esta manera definir en qué movimientos tiene que hacer un mayor esfuerzo.

Partiendo de la base que el software permite recibir y almacenar los datos provenientes de los servomotores del MSR, como se mostró en las pruebas anteriores, se definen dos movimientos a realizar de dos maneras diferentes pero con el mismo objetivo. Para este caso se escoge el movimiento de subir escaleras pero con dos alturas diferentes. Una vez definidos se programan en la interfaz para que sean ejecutados por el MSR. Luego de que el MSR termine de ejecutar los movimientos, se cogen los datos recibidos de los movimientos y se grafican.

Una vez se tiene la gráfica de los dos movimientos se puede analizar qué tanta energía invierte el robot en cada movimiento. Esto se hace partiendo de los resultados obtenidos de la carga (LOAD) que utiliza cada actuador durante la ejecución del movimiento. La carga (LOAD) es adimensional y es una repre-

sentación del torque. A pesar de que este parámetro no tiene unidades es ideal para realizar la comparación entre los dos movimientos ejecutados por el robot.

Para ejecutar esta prueba se creó un archivo de texto con la secuencia de datos a ingresar y se utilizó la opción con la que cuenta la interfaz *Scripted Gaits* diseñada en el presente trabajo de grado y explicada en el capítulo 4 que permite manipular el tiempo de lectura de un archivo `.txt` mediante la casilla *time*.

El primer escalon utilizado tiene una altura de 5,5cm y la secuencia de datos ingresada se muestra en la figura 6.11.

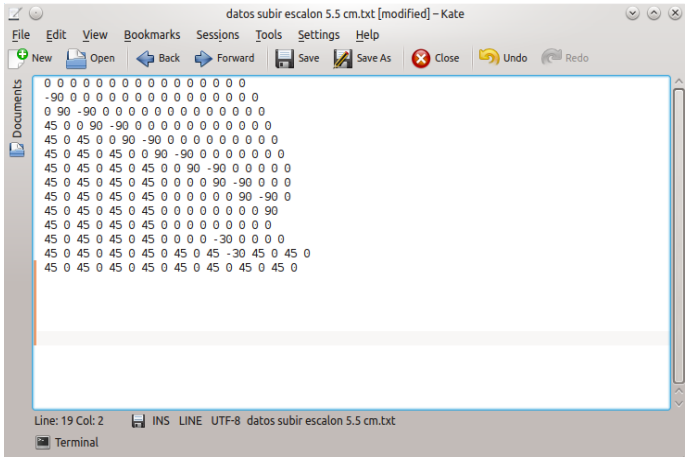

Figura 6.11: Archivo a ingresar a la interfaz

Antes de ingresar los datos a la interfaz se debe especificar el tiempo de lectura que se desea utilizar en la casilla *time*. Este tiempo esta en segundos y admite el ingreso tanto de numeros enteros como decimales. Para esta prueba se utilizó 1 segundo, 2 segundos y 5 segundos. Una vez especificado este tiempo se ingresaron los datos a la interfaz y se observó el movimiento del robot en cada tiempo especificado (ver videos anexos 5,6 y 7). Una secuencia del movimiento del robot se puede observar en la figura 6.12.

La base de datos obtenida se muestra en el anexo 8. Con base en esa base de datos se generaron las gráficas correspondientes a los valores de carga (LOAD) obtenidos de los 16 servomotores (ver figura 6.13).

Al observar la figura se puede ver la variación de carga en cada servomotor en el transcurso del movimiento. En la figura 6.14 se muestra una gráfica que recopila los datos obtenidos por los servomotores durante la ejecución del movimiento.

Al observar la gráfica se puede decir que los valores de carga obtenidos de los servomotores a lo largo de la ejecución del movimiento de subir un escalon de 5,5cm oscilan entre 0 y 350. Para poder sacar unidades y poder decir que es una medida real, hay que hacer una conversión, pero para comparar dos secuencias como lo es en este caso, es posible hacerlo con estos valores.

Figura 6.12: Ejecución del movimiento con un escalon de 5cm

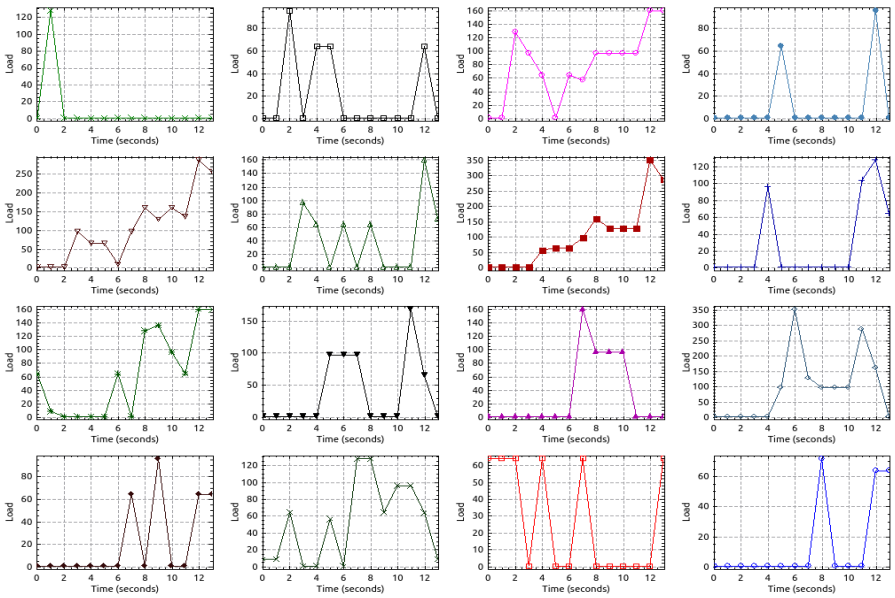

Figura 6.13: Gráficas del Tiempo vs la carga para los 16 servomotores al subir un escalon de 5,5cm

El segundo escalon utilizado tiene una altura de 13cm y la secuencia de datos ingresada para ejecutar el movimiento se muestra en el anexo 9.

Antes de ingresar los datos a la interfaz se debe especificar el tiempo de lectura que se desea utilizar en la casilla *time*. Este tiempo esta en segundos por lo cual para esta prueba se utilizó 1 segundo y 2 segundos. Una vez especificado

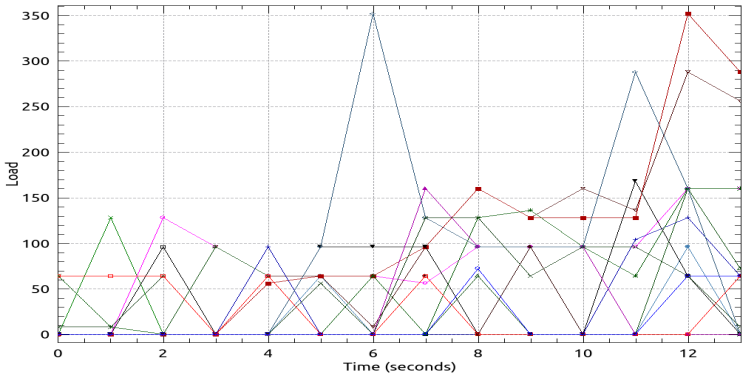

Figura 6.14: Gráfica del Tiempo vs la carga presente en los 16 servomotores al subir un escalon de 5,5cm

este tiempo se ingresaron los datos a la interfaz y se observó el movimiento del robot en cada tiempo especificado (ver videos anexos 10 y 11). Una secuencia del movimiento del robot se puede observar en la figura 6.17.

La base de datos obtenida se muestra en el anexo 12. Con base en esa base de datos se generaron las gráficas correspondientes a los valores de carga (LOAD) obtenidos en los 16 servomotores (ver figura 6.15.)

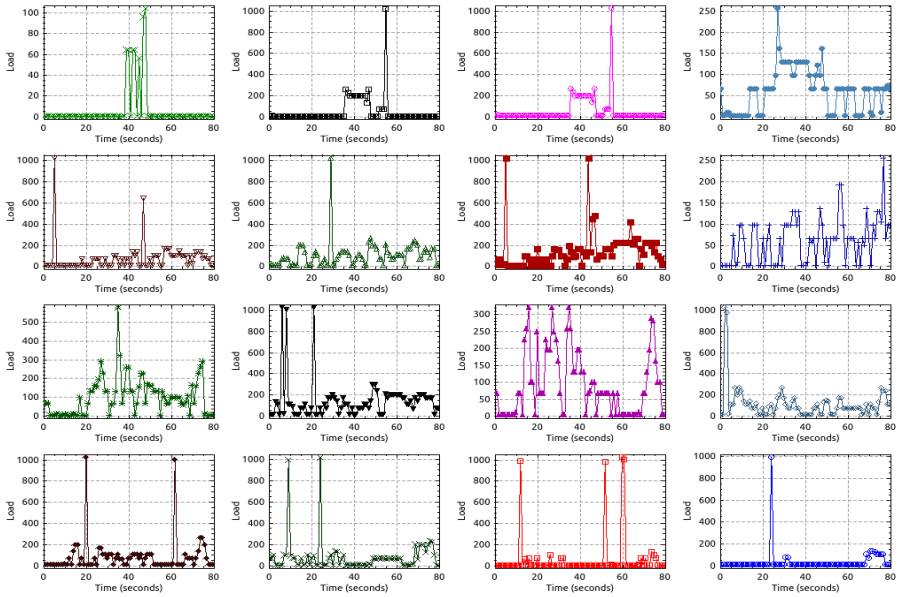

Figura 6.15: Gráficas del Tiempo vs la carga para los 16 servomotores al subir un escalon de 13cm

Al observar la figura se puede ver la variación de carga en cada servomotor en el transcurso del movimiento. En la figura 6.16 se muestra una gráfica que recopila los datos obtenidos por los servomotores durante el movimiento.

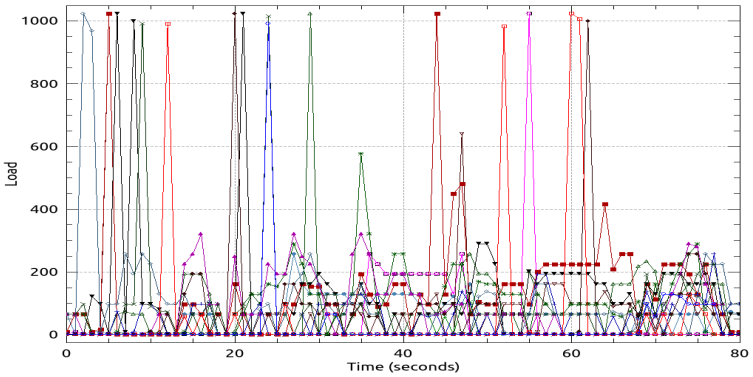

Figura 6.16: Gráfica del Tiempo vs la carga presente en los 16 servomotores al subir un escalon de 13cm

Al observar la gráfica se puede decir que los valores de carga obtenidos de los servomotores a lo largo de la ejecución del movimiento de subir un escalon de 13cm oscilan entre 0 y 1023.

Haciendo una comparación de las gráficas obtenidas a partir de los datos provenientes del robot modular Lola-OP^{TM}se puede observar que los valores de carga registrados al subir un escalon de 5,5cm llegan hasta 350 mientras que los valores de carga registrados al subir un escalon de 13cm llegan hasta 1023, es decir, la carga registrada al ejecutar el movimiento del escalon de 13cm es 2.92 mas grande que la carga registrada al ejecutar el movimiento del escalon de 5,5cm. Esto significa que el robot tiene que hacer un mayor esfuerzo para subir el escalon 13cm, es decir, entre más alto el escalon que suba, mayor es el esfuerzo que debe hacer para cumplir su objetivo.

Figura 6.17: Ejecución del movimiento con un escalon de 13cm

Capítulo 7

Conclusiones

El trabajo desarrollado y documentado en este libro mostró la implementación de funciones de software que permiten el envío de datos a un *Modular Snake Robot* logrando así el diseño de diferentes *gaits* con el objetivo de asignar al robot la configuración deseada. También se mostró la implementación de funciones de software que permiten la adquisición de datos provenientes del *Modular Snake Robot* y su posterior almacenamiento en una base de datos creada solo para este fin.

Adicional a esto, se diseñó una interfaz de control que funciona como *Front End* para las funciones de software desarrolladas en el presente trabajo la cual facilita al usuario el diseño y programación de *gaits* que se obtienen mediante la asignación de valores de ángulos de juntura. También queda disponible un software que permite la generación de archivos que contienen información sobre los datos adquiridos de los servomotores Dynamixel AX-12 durante la ejecución de *gaits*. Las variables que se adquieren de los servomotores en el presente trabajo de grado son la temperatura, la velocidad, la carga (LOAD) y la posición. Estos datos ayudan a entender el comportamiento del robot y es por esto que son utiles para la etapa de análisis de datos.

Dentro de las pruebas diseñadas e implementadas en el protocolo de pruebas se encuentra el desarrollo de un análisis del esfuerzo hecho por los servomotores a partir de la lectura de una de las variables almacenada en la base de datos generada a paritr de la ejecución de un movimiento. Despues de hacer este análisis se puede afirmar que el esfuerzo que hace el robot en un *gait* como el de subir escaleras depende de la altura presente en cada escalon ya que, entre más alto sea el escalón, mayor será el esfuerzo que tiene que hacer el robot para cumplir su objetivo. Con el desarrollo de este análisis se demuestra la utilidad del software desarrollado ya que puede ser utilizado en diversas aplicaciones que utilicen funciones a trozos en un *Modular Snake Robot*.

Con las pruebas desarrolladas se corrobora el funcionamiento del sistema implementado en el presente trabajo de grado. Sin embargo, es importante tener en cuenta que no era la intención de este trabajo en ningun caso, comprobar ninguna hipótesis relacionada con dinámica o física. Los resultados aquí obtenidos

pueden ser utilizados a futuro por expertos para comprobar sus teorías.

Este trabajo de grado contribuyó con la investigación que se viene haciendo sobre los robots modulares, más especificamente, sobre los robots serpiente, mediante el desarrollo del software que permite diseñar *scripted gaits* los cuales son útiles para desarrollar tareas específicas. Adicional a esto, el software aqui desarrollado es el primer paso para una futura integración que involucre todas las interfaces de control de un *Modular Snake Robot*, para así crear un gran *framework* [6] que incluya no solo las interfaces de control, sino también trabajos que se estan desarrollando actualmente. De esta manera se tendrían todos los elementos que permiten experimentar con un *Modular Snake Robot* en un solo *framework*.

Capítulo 8

Lista de Anexos

1. Anexo 1: Codigo desarrollado para la Programación y Adquisición de datos para robots serpiente.
2. Anexo 2: Video paso a paso obtenido en la prueba de envío de una trama de datos.
3. Anexo 3: Video obtenido enviando una trama de datos a 2 servomotores a la vez.
4. Anexo 4: Video obtenido enviando una trama de datos a 4 servomotores a la vez.
5. Anexo 5: Video subiendo escalon de 5,5cm a 1 segundo.
6. Anexo 6: Video subiendo escalon de 5,5cm a 2 segundos.
7. Anexo 7: Video subiendo escalon de 5,5cm a 5 segundos.
8. Anexo 8: Base de datos obtenida de subir un escalon de 5,5cm.
9. Anexo 9: Secuencia de datos ingresada para subir un escalon de 13cm.
10. Anexo 10: Video subiendo escalon de 13cm a 1 segundo.
11. Anexo 11: Video subiendo escalon de 13cm a 2 segundos.
12. Anexo 12: Base de datos obtenida de subir un escalon de 13cm.

Índice de figuras

Bibliografía

[1] M. Yim, D. Duff, and K. Roufas, "Polybot: a modular reconfigurable robot," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 1, 2000, pp. 514–520 vol.1.

[2] S. Hirose, *Biologically Inspired Robots: Snake-like Locomotors and Manipulators*. Oxford University Press, 1993.

[3] K. Melo, L. Paez, M. Hernandez, A. Velasco, F. Calderon, and C. Parra, "Preliminary studies on modular snake robots applied on deminig tasks," in *IEEE IX Latin American Robotics Symposium and IEEE Colombian Conference on Automatic Control*, Oct. 2011.

[4] K. Melo and L. Paez, "Modular Snake Robot Gaits on Horizontal Pipes," in *Intelligent Robots and Systems (IROS), 2012 IEEE International Conference on*, october 2012, pp. 3099–3104.

[5] T. Kamegawa, T. Yamasaki, H. Igarashi, and F. Matsuno, "Development of the snake-like rescue robot," in *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, vol. 5, april-1 may 2004, pp. 5081–5086 Vol.5.

[6] K. Melo, J. Leon, J. Monsalve, V. Fernández, and D. Gonzalez, in *System Integration (SII), 2012 IEEE/SICE International Symposium on*, Dec.

[7] M. Tesch, K. Lipkin, I. Brown, R. L. Hatton, A. Peck, J. Rembisz, and H. Choset, "Parameterized and scripted gaits for modular snake robots," *Advanced Robotics*, vol. 23, no. 9, pp. 1131–1158, 2009.

[8] "Robotsource," fecha última visita: noviembre 15 de 2012. [Online]. Available: www.robotsource.org

[9] L.Paez, *Modular snake-like hyper-redundant mechanism for tubular locomotion analysis*. Pontificia Universidad Javeriana, 2011.

[10] "Robotis," fecha última visita: agosto 28 de 2012. [Online]. Available: www.robotis.com

[11] "Robotis-support," fecha última visita: julio 10 de 2012. [Online]. Available: support.robotis.com

[12] K. Melo, L. Paez, A. Polo, and C. Parra, "Gait Programming and Data Acquisition User Interfaces, for Modular Snake Robots," in *Informatics in Control, Automation and Robotics*, ser. Lecture Notes in Electrical Engineering, D. Yang, Ed. Springer Berlin Heidelberg, 2012, vol. 133, pp. 113–117, 10.1007/978-3-642-25992-0_15.

[13] K. Melo, L. Paez, and C. Parra, "Indoor and outdoor parametrized gait execution with modular snake robots," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, may 2012, pp. 3525–3526.

[14] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. John Wiley & Sons, Inc.

[15] A. Trujillo, J. Igua, K. Melo, and C. Parra, "Dinámica Pasiva para una Cadena Articulada Coplanar de 6 DOF," in *Proc. IEEE IX Latin American Robotics Symposium and IEEE Colombian Conference on Automatic Control*, Oct. 2011.

[16] K. Melo, A. Velasco, and C. Parra, "Motion analysis of an ellipsoidal kinematic closed chain," in *Robotics Symposium, 2011 IEEE IX Latin American and IEEE Colombian Conference on Automatic Control and Industry Applications (LARC)*, oct. 2011, pp. 1–6.

[17] K. Melo and A. Velasco, "Motion analysis of a wheel-like articulated closed chain," in *ANDESCON, 2010 IEEE*, sept. 2010, pp. 1–6.

[18] K.Melo, *Motion Planning for Modular Snake Robots*. Pontificia Universidad Javeriana, 2012.

[19] E. Tira-Thompson, "Digital servo calibration and modeling."

[20] "Kst-visualize your data," fecha última visita: febrero 23 de 2013. [Online]. Available: kst-plot.kde.org