Software controlador de bajo nivel para el robot Lola-OP™

Alvaro Julian Torres Di Zeo

Director de trabajo de grado:
Kamilo Melo Ph.D.

Pontificia Universidad Javeriana
Facultad de Ingeniería
Departamento de Electrónica
02 de Diciembre de 2013

Contenido

Índice de figuras

Abreviaturas

MSR	**M**odular **S**nake **R**obot
IMU	**I**nertial **M**easurement **U**nit
HLC	**H**igh **L**evel **C**ontroller
LLC	**L**ow **L**evel **C**ontroller

Nota de publicaciones

El presente trabajo ha contribuido en las siguientes publicaciones.

- J. Leon, J. Monsalve, A. Di Zeo, L. Paez, and K. Melo. “Open Modular Snake Robot Software Architecture”. In VIII Workshop on Software Development and Integration in Robotics (SDIR VIII), IEEE International Conference on Robotics and Automation (ICRA), 2013, pages 84–86, may 2013
- K. Melo, D. Roa, V. Gudavarthi, Y. Bello, A. Di Zeo, J. Leon, and L. Paez, “Modular Snake Robots. Research on Locomotion and Low Cost Open Hard-Software Platforms, Development and lntegration”, in Robotics for Risky Environments - Extreme Robotics, Proceedings of the 7th International Workshop IARP-RISE-ER2013, E. Yurevich and I. Baudoin, Eds. Central Research Institute of Robotics and Technical Cybernetics, 2013, vol. 69, pp. 391-397.
- K. Melo, J. Leon, A. Di Zeo, V. Rueda, D. Roa, M. Parraga, D. Gonzalez, and L. Paez, “The Modular Snake Robot Open Project: Turning Animal Functions into Engineering Tools”, in 11th IEEE International Symposium on Safety Security and Rescue Robotics (SSRR), oct 2013.

Capítulo 1

Introducción

Un robot serpiente constituido por módulos, llamado en la literatura *modular snake robot* [1-6] o MSR como se hará referencia en este trabajo, es un sistema robótico biomórfico construido al concatenar múltiples módulos, cuya estructura mecánica se asemeja al cuerpo de las serpientes. Las principales cualidades que le diferencian de otros tipos de robot son: la reducida longitud de su sección transversal respecto a la de su sección longitudinal, lo que le permite moverse y maniobrar en espacios estrechos [1], y la versatilidad, otorgada por su diseño modular, para adoptar un amplio rango de posturas [2], posibilitando diversos esquemas de locomoción, cada uno conseguido al variar su forma secuencialmente.

Desde hace algunos años los *snake robots* han ganado un interés creciente por parte de numerosos grupos de investigación alrededor del mundo [2][7-11], motivados por el potencial de estos sistemas para ejecutar tareas como la búsqueda y rescate en escenarios de desastre, la inspección y reparación de tuberías, el reconocimiento y desactivación de explosivos, entre otras [3].

El grupo KM-RoBoTa[1] ha llevado a cabo varios trabajos de investigación y desarrollo sobre *modular snake robots* dentro del proyecto Lola-OP™, una plataforma robótica de hardware abierto, sobre la cual se realizan principalmente experimentos de locomoción y para la cual se han desarrollado herramientas de simulación, control y análisis [4].

Fig. 1.1. Plataforma robótica Lola-OP™ [2]

1 www.km-robota.com

2 Fotografía tomada de www.km-robota.com

La arquitectura de hardware actual de Lola-OP™ requiere de un sistema PC externo al robot para generar las instrucciones que éste ejecutará. Estas son enviadas a un puerto USB conectado al robot mediante una interfaz que emula un puerto serial. Con el fin de lograr una plataforma robótica totalmente portable, y eliminar los problemas de movilidad asociados al cableado, se debe integrar la unidad de procesamiento a la estructura mecánica. Este trabajo de grado es una aproximación a esa necesidad.

Debido a que casi la totalidad de los MSR desarrollados hasta ahora son prototipos, generalmente se realiza el control del robot desde un sistema PC para simplificar el proceso de experimentación. Existen numerosos SBC (*Single Board Computer*), como el conocido *Beagleboard* [12], cuyo tamaño y peso permitirían su instalación en el robot, pero que carecen de las interfaces necesarias (puertos, etapa de potencia) para realizar la conexión con los motores de forma simple. Para integrar una de estas plataformas al robot seria indispensable construir hardware adicional, lo que demandaría un elevado costo en tiempo y dinero.

Recientemente fue lanzada la tarjeta CM 900 [13], una plataforma de hardware abierto producida por la compañía *Robotis Inc.*, quien fabrica también los actuadores que utiliza el robot Lola-OP™. *Robotis Inc*. ha creado la comunidad *Robotsource* (a la cual pertenece KM-RoBoTa)[1] para que grupos de investigación en robótica de todo el mundo puedan intercambiar conocimientos y dar a conocer sus proyectos. La tarjeta CM 900 fue lanzada como un proyecto en el que se busca que los grupos dentro de la comunidad *Robotsource* sean quienes evalúen su funcionalidad, y eventualmente contribuyan a su desarrollo al constituir una fuente importante de retroalimentación. Por esto, *Robotis Inc.* puso a disposición de la comunidad una primera versión de la tarjeta.

La tarjeta CM 900 cuenta con todo el hardware necesario para manejar los servomotores a bordo del robot, además cuenta con diversas interfaces que podrían usarse en futuros proyectos para expandir las capacidades de la plataforma robótica. Adicionalmente, el fabricante provee (sin costo) un SDK (Software Development Kit) especifico para esta tarjeta.

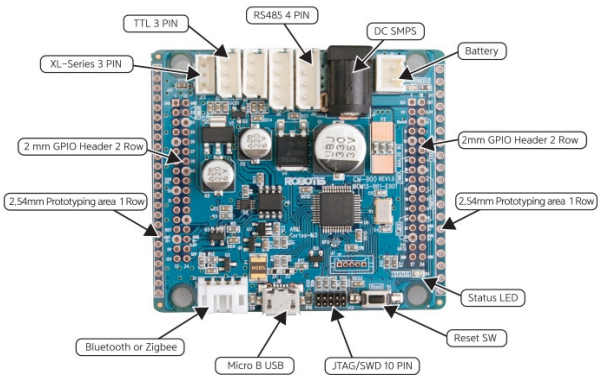

Fig. 1.2. Tarjeta CM 900

1 http://www.robotsource.org/bs/bd.php?bt=forum_LolaOP
2 Fotografía tomada de www.robotis-shop-en.com/shop/step1.php?number=992

A pesar de la versatilidad que otorga la tarjeta CM 900, la capacidad de procesamiento que esta plataforma puede proveer, al estar basada en un microcontrolador (pese a ser de 32 bits), podría limitar el desarrollo de futuras aplicaciones de software para Lola-OP™. Como solución, en este trabajo se han dividido las tareas de procesamiento, dejando el control de bajo nivel a cargo de la tarjeta CM 900, que por las características de su licenciamiento y sus prestaciones de conectividad, tamaño y peso, resulta ideal para el proyecto. Las aplicaciones de alto nivel han sido delegadas a un sistema con mayor capacidad de computo, que puede o no, en un futuro estar a bordo.

1.1. Objetivos

1.1.1. Objetivo general

Implementar el control de bajo nivel del robot Lola-OP™, en el sistema de desarrollo de 32 bits CM-900.

1.1.1. Objetivos específicos

- Determinar los requerimientos de tiempos de transmisión y recepción de datos de actuadores y sensores pertenecientes al robot Lola-OP™.
- Implementar sobre un (1) sistema de desarrollo de 32 bits CM 900, un conjunto de controladores de dispositivos para 16 servomotores *Dynamixel AX-12* y una (1) unidad de medición inercial (IMU).
- Manejar excepciones localmente en el sistema de desarrollo CM 900, generadas por alarmas y mensajes de error de los dispositivos del robot Lola-OP™.
- Desarrollar una API para la creación de funciones (instrucciones) de alto nivel para controlar la plataforma robótica desde la unidad de procesamiento principal.
- Desarrollar un programa de demostración de locomoción empleando el conjunto de controladores de dispositivos implementados.

Capitulo 2

Descripción del hardware

La plataforma robótica Lola-OP™ esta compuesta por 16 servomotores Dynamixel AX-12 y una (1) unidad de medición inercial, los criterios bajo los cuales fue diseñado este robot y los criterios de selección de sus componentes se describen extensamente en trabajos anteriores [1-6]. Ademas, en este trabajo se ha incorporado una (1) tarjeta de desarrollo CM 900 a la plataforma robótica. En este capitulo se describe los diferentes componentes que hacen parte de este robot.

2.1. Tarjeta CM 900

Esta tarjeta es un proyecto en desarrollo de hardware abierto en el que se pretende cubrir las necesidades en cuanto a conectividad, procesamiento y autonomía de plataformas robóticas que hagan uso de la serie de servomotores *Dynamixel*, producidos por la compañía *Robotis Inc*, la misma que ha desarrollado la tarjeta CM 900. En esta sección se describen las características mas relevante de este sistema.

2.1.1. Microcontrolador

La tarjeta CM 900 esta basada en el microcontrolador STM32F103C8, el cual incorpora el procesador de 32 *bits ARM Cortex M-3* operando a una frecuencia de 72 *MHz*. Los periféricos incluidos en este microcontrolador se presentan en la figura 2.1.[1]

Peripheral		STM32F103C8
Flash - Kbytes		64
SRAM - Kbytes		20
Timers	General-purpose	3
	Advanced-control	1
Communication	SPI	2
	I^2C	2
	USART	3
	USB	1
	CAN	1
GPIOs		37
12-bit synchronized ADC Number of channels		2 10 channels
CPU frequency		72 MHz
Operating voltage		2.0 to 3.6 V

Fig. 2.1. Características del microcontrolador STM32F103C8

Entre estos componentes es especialmente relevante para este trabajo el numero de UARTs del que dispone, ya que el control de los dispositivos a bordo del robot requiere de estas interfaces. Como mínimo se requieren dos UARTs, una para los actuadores y otra para el sensor inercial. Una tercera UART queda disponible, la cual fue de gran utilidad para depurar el software desarrollado

1 Tomada de *STM Medium-density performance line ARM-based 32-bit M* http://www.st.com/web/catalog/mmc/FM141/SC1169/SS1031/LN1565/P

para este microcontrolador, al hacer posible el envió de datos que permitieran conocer el estado de las variables del programa de control de bajo nivel.

2.1.2. Interfaces de comunicación

La tarjeta CM 900 cuenta con interfaces de comunicaciones de distinto tipo, entra las que se soportan sin necesidad de hardware adicional se encuentran: *USB*, *RS485*, *TTL Dynamixel*, *Bluetooth* y *Zigbee.* En este trabajo fueron usadas la interfaz *USB* en la comunicación con el sistema de control principal y la interfaz *TTL Dynamixel* en la comunicación con los actuadores. En el caso de la comunicación con la *IMU* fue necesario hacer uso del área *de prototipado*, la cual consiste en puntos de conexión que llevan directamente a los pines del microcontrolador, para poder conectarla a una de las UARTs de éste.

2.1.3. Etapa de potencia

La tarjeta requiere de un voltaje de operación de 5 *V*, el cual puede ser provisto por la conexión *USB*. Pero para el manejo de la potencia requerida por los actuadores es necesario conectar a la tarjeta CM 900 una fuente de voltaje de 12 *V*, el cual sera regulado por la etapa de potencia de ésta y entregado a los actuadores a través del conector usado por la interfaz *TTL Dynamixel.*

2.1.4. Ambiente de desarrollo *Open CM9*

El fabricante de la tarjeta CM 900 proporciona un ambiente de desarrollo para ésta llamado *Open CM9*, el cual esta basado en *Processing*. Este *IDE* (*Integrated Development Enviroment*) es presentado como una herramienta de desarrollo que permite la programación de la tarjeta CM 900 de forma fácil a través de un puerto *USB*. También es compatible con las librerías del proyecto *Arduino* y su intención es que pueda ser usado por personas sin conocimientos avanzados de programación.

Inicialmente se planeó trabajar con este *IDE* debido a las ventajas que ofrecían las funciones de alto nivel de las que dispone, pero después de realizar algunas pruebas con las rutinas ejemplo incluidas en él, se decidió descartar esa opción ya que el proceso de programación del microcontrolador era tedioso debido a los constantes errores que se presentaban en él. Ademas, el tamaño de los programas compilados usando este *IDE* era excesivamente grande, llegando a ocupar la mayoría de los ejemplos, los cuales son relativamente simples, cerca de un tercio de la memoria flash del microcontrolador. De esto ultimo surgió la preocupación de que debido a la magnitud del proyecto el ejecutable resultante fuera demasiado grande para la memoria del microcontrolador.

Por lo anterior se decidió realizar la programación de la tarjeta CM 900 usando el dispositivo *ST-LINK*, el cual a través de una interfaz *SWD* (*Serial Wire Debug*) es capaz de acceder a la memoria *flash* del microcontrolador, y almacenar en esta el programa. Ademas se empleo la versión

de evaluación del ambiente de desarrollo *µVision* 4.7 el cual provee herramientas profesionales de desarrollo de software embebido, estas son: editor para el lenguaje de programación C, compilador, ensamblador, encadenador y depurador. La única restricción de la versión de evaluación es un limite de 32 *KB* en el ejecutable generado a partir de los archivos fuente, esto no fue un problema en el desarrollo de este trabajo ya que el tamaño final del ejecutable fue cercano a 14 *KB*.

2.2. Actuadores *Dynamixel AX-12*

A continuación se listan las características mas relevantes de estos actuadores:

- Cuentan con un microcontrolador a bordo que hace posible la configuración de los distintos parámetros que determinan su comportamiento.
- Mediante un protocolo de comunicaciones propio de los actuadores es posible enviar instrucciones a estos y recibir información de estatus. Este protocolo se describe en la sección 3.1.
- La conexión de múltiples actuadores se realiza siguiendo la topología *Daisy Chain*
- La interfaz de comunicaciones serial soporta velocidades de transmisión de hasta 1 *Mbps.*
- La resolución del control de posición y velocidad es de 8 *bits*, es decir, estos parámetros pueden tomar 1024 valores diferentes.

2.3. Sistema de medición inercial *Razor SEN-10736*

El sistema de medición inercial *Razor SEN-10736* consiste en una tarjeta que incluye tres sensores distintos cuya salida es digital, y a partir de los cuales provee el valor de los parámetros de orientación azimut, elevación y alabeo a través de la interfaz serial de un microcontrolador que también hace parte de este sistema. La figura 2.2[1] muestra una fotografía de esta tarjeta en la que

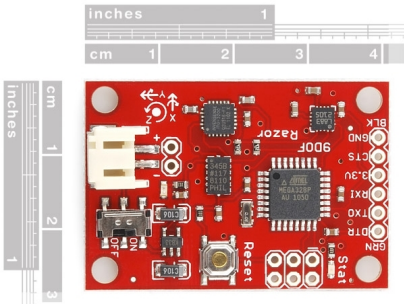

Fig 2.2. Sistema de medición inercial Razor SEN-10736

1 Fotografia tomada de https://www.sparkfun.com/products/10736

pueden apreciarse sus dimensiones. A continuación se describen cada uno de los componentes de este sistema.

2.3.1. Acelerómetro

Corresponde al circuito integrado ADXL345, el cual es capaz de realizar mediciones, en tres ejes, de aceleración estática (gravedad) y dinámica (producida por choques o movimiento) con una resolución de 13 *bits*. Esta información puede ser adquirida a través de una interfaz de comunicaciones *I2C* o *SPI*.

2.3.2. Giroscopio

El circuito integrado ITG-3200 es un giroscopio de tres ejes cuya salida es digital. Cuenta con una resolución de 16 *bits* y las mediciones realizadas por este sensor pueden ser obtenidas a través de una interfaz *I2C*.

2.3.3. Magnetómetro

El sensor HMC5883L es un magnetómetro de tres ejes cuya resolución es de 5 *mGauss*, su salida es digital y maneja el protocolo de comunicaciones *I2C*.

2.3.4. Microcontrolador

El microcontrolador presente en la tarjeta *Razor SEN-10736* es el ATmega328, el cual viene programado con el *bootloader Arduino*, por lo que es posible modificar su *firmware* mediante un puerto serial usando el mismo *IDE* que empelan las tarjetas de desarrollo *Arduino*.

Este microcontrolador se encarga de recibir la salida de los tres sensores descritos arriba, por medio de una interfaz *I2C*, y procesar esa información para obtener los parámetros de orientación azimut, elevación y alabeo, los cuales pueden ser adquiridos por un dispositivo externo a través de una UART del microcontrolador, el protocolo de comunicaciones empleado para realizar la solicitud de los datos es descrito en la sección 3.2.

Capitulo 3

Requerimientos de tiempos de transmisión y recepción de datos de los actuadores y sensores a bordo del robot

Los actuadores *Dynamixel AX-12* disponen de un microcontrolador integrado cuya función es el manejo del servomotor y la adquisición de datos de los sensores que contiene el actuador. A su vez, la unidad de medición inercial *Razor* cuenta con un microcontrolador encargado de la adquisición de datos del giroscopio, acelerómetro y magnetómetro, y del procesamiento de esta información. Tanto los actuadores como el sensor inercial hacen uso de una interfaz de comunicación serial que les permite recibir instrucciones de un controlador externo y enviar información de estatus a este. Gracias a ello, el manejo de estos dispositivos se reduce en gran medida a la implementación de los protocolos usados por cada uno de ellos.

En este capitulo se describen brevemente los protocolos de comunicaciones empleados por los dispositivos que componen al robot y se analizan los requerimientos de tiempo presentes en la comunicación con estos dispositivos.

3.1. Comunicación con los actuadores Dynamixel AX-12

Estos actuadores emplean un protocolo basado en paquetes dentro del cual existen dos posibles tipos de paquete: de instrucción y de estatus. Los paquetes de instrucción, son recibidos por los actuadores y contienen comandos que estos deben ejecutar. Los paquetes de estatus, enviados por los actuadores, contienen la respuesta a un paquete de instrucción.

Como se menciono el control de movimiento y la comunicación están a cargo de un microcontrolador. En éste han sido definidos registros para cada una de las variables que pueden ser controladas en los actuadores. Para modificar u obtener el valor de una de estas variables se de debe escribir o leer, respectivamente, el registro correspondiente.

3.1.1. Formato de los paquetes

Ambos tipos de paquete están precedidos por dos caracteres de inicio. Además incluyen un encabezado, el cual contiene la dirección o identificador del actuador al que corresponde el paquete, la longitud del paquete y un ultimo campo que corresponde a la instrucción que debe ser ejecutada (en el caso de un paquete de instrucción), o a un código de error (cuando el paquete es de estatus). El paquete puede contener también información adicional, llamada parámetros de paquete, necesaria para la ejecución de una instrucción o que corresponde a información que le fue solicitada al actuador. Finalmente se incluye una suma de comprobación para permitir la detección de errores en el paquete.

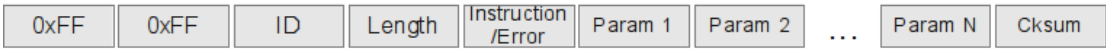

Fig. 3.1. Estructura de los paquetes usados por los actuadores Dynamixel AX-12

3.1.2. Tiempos de transferencia

Según la estructura de los paquetes descrita, la cual se muestra en la figura 3.1, la longitud mínima de un paquete, caso en el que no se requieren parámetros, es de 6 *bytes* (dos *bytes* de inicio y cuatro *bytes* que corresponden a los campos *ID*, *Length* e *Instruction/Error*). Cuando se incluyen parámetros la longitud es igual a *6 + N bytes*, donde *N* es el numero de parámetros que contiene el paquete. La transferencia de los datos es asíncrona y cada *byte* es enviado usando un solo bit de parada. Ya que no se envía el bit de paridad, por cada *byte* transmitido deben ser enviados 10 *bits* (1 *bit* de inicio + 8 *bits* del dato + 1 *bit* de parada).

La velocidad de transferencia máxima soportada por los actuadores es de 1 *Mbps*, esta fue la velocidad que se decidió usar en la implementación de este trabajo con el fin de contar con tiempos de transferencia mínimos. A esa velocidad, se requieren 10 μs para completar la transferencia de un dato. De lo anterior puede hallarse el tiempo que requiere la transferencia de un paquete de tamaño arbitrario 6 + *N*, donde *N* es el numero de parámetros del paquete.

$$t_{txpkg} = (6+N)\cdot 10\mu\, s \tag{1}$$

Además de la velocidad de transmisión debe considerarse el retardo de retorno (*return delay time*). Este parámetro corresponde al tiempo que deben esperar los actuadores para iniciar la transmisión de un paquete de estatus, después de haber recibido uno de instrucción. Su valor es configurable y puede estar entre *1* y *500* μs. Según esto, se tiene que el tiempo de ida y vuelta (*round trip time*) de un paquete sera igual a la suma del tiempo de transmisión del paquete de instrucción, el retardo de retorno, y el tiempo de transmisión del paquete de estatus.

$$RTT = t_{tx\,inst} + t_{rd} + t_{tx\,stat} \tag{2}$$

También en el caso del retardo de retorno se han configurado los actuadores para minimizar el tiempo de la comunicación; el valor que se ha elegido para este parámetro es de 1 μs. De este modo el tiempo de ida y vuelta (*round trip time*) de un paquete sera aproximadamente el tiempo de transmisión del paquete de instrucción más el tiempo de transmisión del paquete de estatus.

$$RTT = t_{tx\,inst} + t_{tx\,stat} + 1\,\mu s \simeq t_{tx\,inst} + t_{tx\,stat} \tag{3}$$

Ahora se considerará el tiempo que requiere el envió de paquetes de instrucción específicos y la recepción del paquete de estatus correspondiente.

Para modificar una variable de uno de los actuadores es necesario enviar a éste un paquete de instrucción de escritura, el cual incluirá como parámetros: la dirección del registro sobre el que se va escribir, el *byte* menos significativo y el *byte* más significativo del valor de la variable. Este paquete tiene tres parámetros (N = 3), y su tamaño total es de 6 *bytes* + 3 *bytes* = 9 *bytes*.

Ante el paquete de instrucción el actuador responderá con un paquete de estatus que confirma la recepción de la instrucción. El paquete de estatus no requiere de parámetros, por lo tanto su tamaño es de 6 *bytes*. Las ecuaciones (4), (5) y (6) corresponden, respectivamente, al tiempo de transmisión del paquete de instrucción, el tiempo de transmisión de un paquete de estatus, y el tiempo que transcurre desde el inicio de la transmisión, del paquete de instrucción, hasta el fin de la recepción del paquete de estatus.

$$t_{tx\,inst} = 9 \cdot 10\,\mu s = 90\,\mu s \tag{4}$$

$$t_{tx\,stat} = 6 \cdot 10\,\mu s = 60\,\mu s \tag{5}$$

$$RTT_{Write} \simeq t_{tx\,inst} + t_{tx\,stat} = 90\,\mu s + 60\,\mu s = 150\,\mu s \tag{6}$$

En el caso de una operación de lectura se debe incluir en el paquete de instrucción la dirección del registro correspondiente y la longitud de los datos que deben ser leídos. Un paquete de este tipo tiene un tamaño de *8 bytes*. Por otro lado, el paquete de estatus contiene como parámetros el valor del registro especificado en el paquete de instrucción. Este valor puede tener un tamaño de uno o dos *bytes*, suponiendo que el tamaño del dato es de 2 *bytes* (peor caso) la longitud total del paquete de estatus es también de 8 *bytes*.

Tiempos de transmisión cuando se envía un paquete de instrucción solicitando el valor de un parámetro cuyo tamaño es de 2 *bytes:*

$$t_{tx\,inst} = 8 \cdot 10\,\mu s = 80\,\mu s \tag{7}$$

$$t_{tx\,stat} = 8 \cdot 10\,\mu s = 80\,\mu s \tag{8}$$

$$RTT_{Read} \simeq t_{tx\,inst} + t_{tx\,stat} = 80\,\mu s + 80\,\mu s = 160\,\mu s \tag{9}$$

3.2. Comunicación con el sensor inercial

La información provista por los diferentes sensores que componen la IMU es adquirida por el microcontrolador de a bordo, a partir de esta información los parámetros azimut (*yaw*), elevación (*pitch*) y alabeo (*roll*) son calculados, éstos determinan la orientación del sistema. Cada parámetro es almacenado en una variable de tipo flotante, de manera que su tamaño equivale a cuatro *bytes*.

Para que un sistema externo pueda acceder a estos valores debe enviar un comando al microcontrolador de la IMU. El comando esta conformado por un bit de inicio (# o 0x23 en el código ASCII) seguido por la letra f (0x66 en el código ASCII), esta secuencia es interpretada como una solicitud de trama o *frame*. Cada trama contiene el valor de los tres parámetros de orientación divididos en doce *bytes* (cuatro por cada parámetro) y esta es enviada a través de una interfaz serial al dispositivo que la solicitó.

3.2.1. Tiempos de transferencia

La UART del microcontrolador a bordo de la IMU opera a un *baud rate* de 57600 *bps*, y cada dato se envía usando un solo bit de parada y no se incluye el bit de paridad. El tiempo que se requiere para obtener los parámetros de orientación es igual al tiempo de transmisión de catorce *bytes* (dos del comando y doce del *frame*), además de un bit de inicio y otro de parada por cada *byte*. En total deben enviarse 140 bits para completar la transferencia. El tiempo que transcurre desde el inicio de la transmisión del comando de solicitud hasta el final de la recepción de la trama es calculado en (10).

$$RTT = \frac{10 * 14}{57600} = 2.43\,ms \tag{10}$$

Capitulo 4

Controladores de dispositivos

Al integrar la tarjeta de desarrollo CM 900 a la plataforma robótica Lola-OP™ se busca delegar a ésta las funciones de control de bajo nivel. Estas funciones comprenden principalmente: el manejo de la comunicación con los dispositivos que componen al robot, que como se mencionó en el capitulo anterior es fundamental en el control de los mismos; el manejo de excepciones y errores que puedan surgir; y dotar al sistema de capacidades de temporización.

Por otro lado, el control de alto nivel, relacionado con la locomoción del robot, esta a cargo de un sistema externo con mayores capacidades de procesamiento, el cual se soporta en la tarjeta CM 900 para llevar a cabo el control de movimiento del robot. De lo anterior surge la necesidad de que ambos sistemas puedan comunicarse entre si.

Para dar solución a estos requerimientos fue definida una arquitectura de software en base a la cual se implementó el control de bajo nivel y el manejo de la comunicación con el sistema de control de principal. En este capitulo se presenta esta arquitectura y los desarrollos realizados en torno a ella.

4.1. Arquitectura de software del controlador de bajo nivel

En la figura 4.1 se muestra el diagrama de bloques de la arquitectura de software del controlador de bajo nivel. A continuación se describe cada uno de los módulos que la componen.

- ***Interfaz USB:*** Este módulo se encarga del manejo del protocolo USB, usado para establecer la comunicación con el sistema de control principal, el cual puede ser un sistema PC o un SBC (*Single Board Computer*).
- ***Protocolo de Comunicaciones LLC-HLC:*** Este módulo se encarga de la construcción, interpretación y validación de los paquetes de datos definidos por el protocolo de comunicaciones desarrollado para este trabajo, el cual permite la transferencia eficaz de información entre el sistema de control principal y la tarjeta CM 900. En la sección 4.2. se describe en detalle este protocolo.

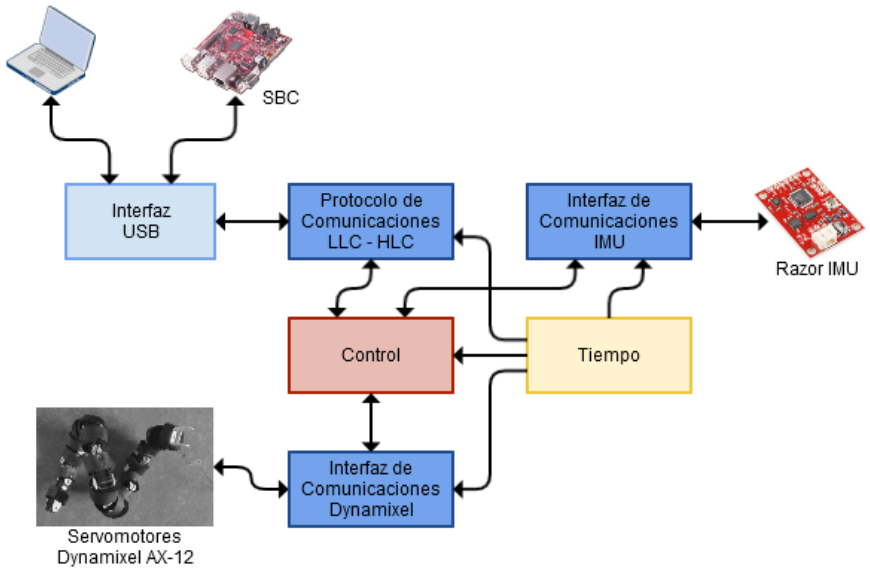

Fig. 4.1. Arquitectura de software del controlador de bajo nivel

- ***Interfaz de Comunicaciones IMU:*** Este módulo tiene a cargo el control sobre la comunicación con la unidad de medición inercial *Razor*, permitiendo a otros módulos obtener el valor de las mediciones realizadas por esta.
- ***Interfaz de Comunicaciones Dynamixel:*** Su función es la construcción, interpretación y validación de paquetes de datos, propios del protocolo usado por estos actuadores, permitiendo el envió de instrucciones a los servomotores y la recepción de información de estatus desde estos.
- ***Tiempo:*** Este módulo provee funciones de temporización al sistema, permitiendo a otros módulos establecer periodos y timeouts, y verificar la ocurrencia de estos eventos.
- ***Control:*** Básicamente este módulo se encarga de ejecutar las instrucciones recibidas desde el sistema de control principal, haciendo uso de los módulos que manejan la comunicación con los dispositivos a bordo del robot. También usa estos módulos para recopilar información de estatus que posteriormente es enviada al sistema de control principal.

4.2. Protocolo de comunicaciones LLC-HLC

El protocolo de comunicaciones, usado por la aplicación de alto nivel y el controlador de bajo nivel, emplea paquetes de datos de tamaño variable para transferir información de un sistema a otro. Además de estos paquetes se emplean secuencias de caracteres que sirven como comandos en el control de la comunicación y la ejecución del control de movimiento.

Este protocolo sigue un modelo maestro-esclavo donde es posible el intercambio de estos roles mediante el uso de un *token*, es decir, de un comando que indica a un dispositivo que se le ha entregado el control sobre la comunicación; quien es maestro puede enviar instrucciones y comandos, por otro lado el esclavo solo puede enviar confirmaciones y solicitudes de reenvió. Inicialmente el controlador principal actúa como maestro.

En esta sección se describe el formato de los paquetes y comandos, y la manera en que se transfieren.

4.2.1. Formato de los paquetes

Cada paquete incluye tanto información de control de la comunicación (inicio de paquete, numero de secuencia, tamaño de paquete y suma de comprobación) como información necesaria para clasificar el contenido del paquete. En la figura 4.2 se muestra el formato general de los paquetes usados por el protocolo de comunicaciones, en el cual se especifica el tamaño de cada uno de sus campos.

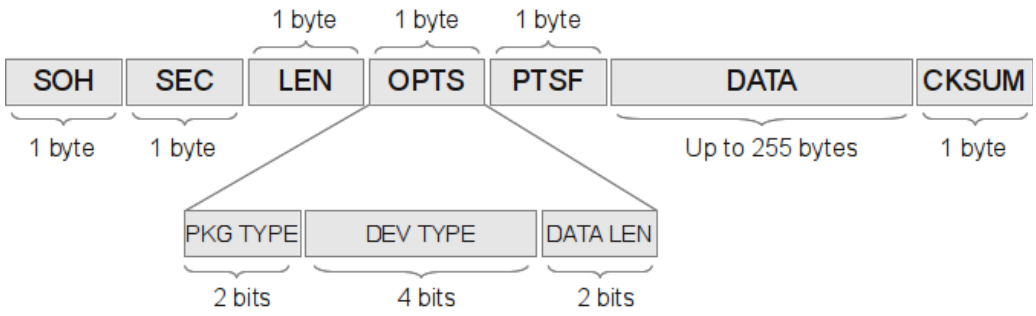

Fig. 4.2. Formato general de los paquetes del protocolo LLC-HLC

Todos los paquetes incluyen los siguientes campos:

- ***SOH:*** El carácter SOH (0x01) es usado para indicar el inicio de un paquete.
- ***SEC:*** Numero de secuencia.
- ***LEN:*** Tamaño o longitud del paquete.
- ***OPTS:*** Opciones del paquete.
  - ***PKG TYPE:*** Tipo de paquete, indica si el paquete contiene información de configuración, instrucciones que deban ser ejecutadas por los actuadores, información proveniente de los sensores del robot o un mensaje de error.
  - ***DEV TYPE:*** Especifica a cual de los dispositivos controlados (IMU o servomotores) hace referencia la información contenida en el paquete.
  - ***DATA_LEN:*** Indica el tamaño de cada uno de los datos incluidos en el paquete, estos pueden ser de 8, 16, 32 o 64 bits.
- ***PTSF (Packet Type Specific Field):*** Campo específico de cada tipo de paquete.

- ***CKSUM:*** Suma de verificación. Se calcula como:

$$CKSUM = LEN + OPTS + PTSF + DATA \tag{1}$$

Si el resultado es mayor a *0xFF* se debe tomar el byte menos significativo.

4.2.2. Tipos de paquetes

Los tipos de paquetes que se describen a continuación pueden ser clasificados según el sistema que los construye y envía.

El controlador principal genera y envía paquetes de configuración y de instrucción para ordenar al controlador de bajo nivel ciertas acciones. Por otro lado, el controlador de bajo nivel genera y envía paquetes de estatus y de error para entregar información al controlador principal del estado de los dispositivos del sistema.

Paquetes de configuración

Estos paquetes son usados para definir y modificar la configuración global, o de un dispositivo en particular, del sistema de control de bajo nivel. Configuración en este contexto se refiere a la definición de los parámetros bajo los cuales se realizará el control de los dispositivos. Por ejemplo, uno de estos paquetes puede contener la frecuencia a la cual se deben enviar instrucciones de movimiento a los servomotores o podría contener las direcciones de los parámetros que deben ser leídos de los registros de control de los actuadores, en la sección 4.3 se describe en detalle la totalidad de los parámetros de configuración. A continuación se muestra el formato de este tipo de paquete.

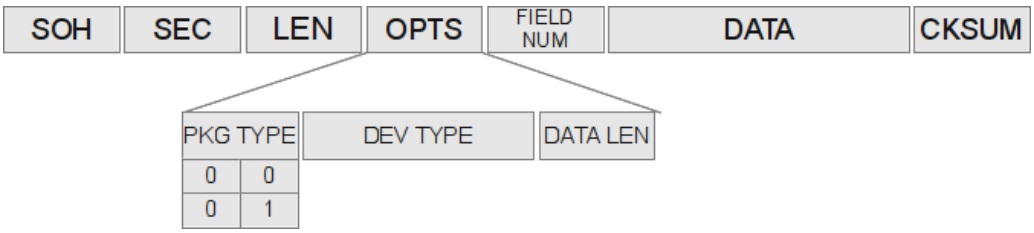

Fig. 4.3. Formato de los paquetes de configuración

Dos tipos diferentes de paquete de configuración fueron definidos:

- ***Paquete de configuración global:*** Este tipo de paquete se utiliza para definir o modificar la configuración global del sistema. Los parámetros que pueden ser definidos o modificados son:
  - Periodo de ejecución de instrucciones.
  - Cantidad de dispositivos de tipo diferente que deban ser controlados.

El campo *DEV TYPE* es ignorado cuando se recibe este tipo de paquete.

- ***Paquete de configuración de dispositivo:*** Son usados para definir o modificar la configuración de un tipo de dispositivo en particular, especificado por el campo *DEV TYPE*. Los parámetros de configuración que pueden ser definidos o modificados son:
  - Cantidad de instancias del dispositivo.
  - Cantidad de parámetros que serán controlados (posición, velocidad, etc) mediante paquetes de instrucción y sus respectivos identificadores. Se soporta el control de hasta dos parámetros.
  - Cantidad de parámetros que deberán ser leídos (temperatura, posición, etc.) y enviados al controlador principal mediante paquetes de estatus, sus respectivos identificadores y la frecuencia a la que debe realizarse la lectura de cada parámetro. Se soporta la lectura de hasta cuatro parámetros para el total de dispositivos que deban ser controlados.

Nota: La restricción impuesta a la cantidad de parámetros controlados y leídos se debe a la necesidad de establecer condiciones bajo las cuales se garantice al usuario la consistencia entre la ejecución de una instrucción y el momento que fue definido para su ejecución.

El campo *FIELD NUM* dentro de un paquete de configuración indica cual de los parámetros de configuración será definido o modificado.

Paquetes de instrucción

Este tipo de paquete es usado por el controlador principal para indicar, al controlador de bajo nivel, que se debe realizar un cambio en el valor de uno de los parámetros control de un tipo de dispositivo. Por ejemplo, un paquete de instrucción podría especificar que para los actuadores (tipo de dispositivo) se debe modificar la velocidad angular (parámetro), y en el paquete se incluirá el valor de la velocidad para cada una de las instancias del dispositivo. En la figura 4.4 se muestra el formato de los paquetes de instrucción.

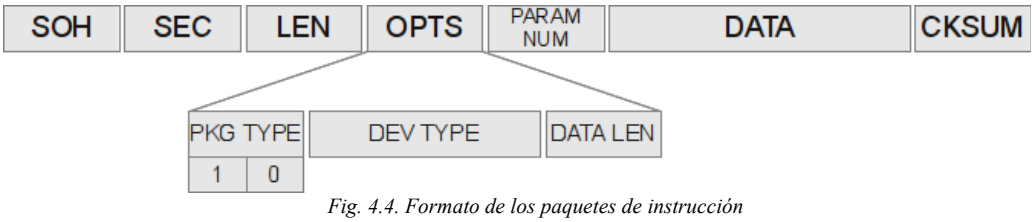

Fig. 4.4. Formato de los paquetes de instrucción

El campo *PARAM NUM* identifica el parámetro de control que debe ser modificado. En la sección 4.3 se enumeran los parámetros de control específicos de los actuadores que componen al robot Lola-OP™, y se aclaran las diferencias entre los parámetros de configuración y los parámetros de control.

Paquetes de estatus

Son usados por el controlador de bajo nivel para enviar información adquirida de los dispositivos del sistema al controlador principal. La figura 4.5 muestra el formato de estos paquetes.

En un paquete de estatus se envía el valor de un parámetro de control para todas las instancias de un tipo de dispositivo determinado. Por ejemplo, un paquete de estatus puede contener los valores de la temperatura (parámetro) de los actuadores (tipo de dispositivo).

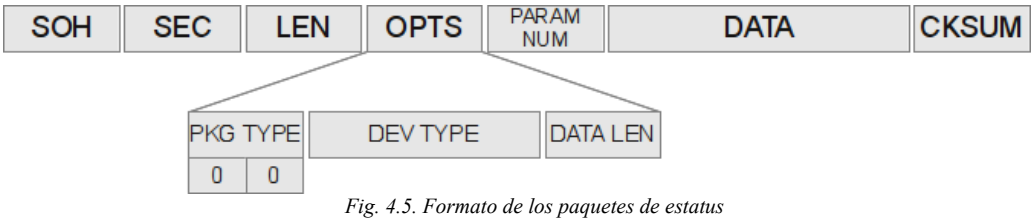

Fig. 4.5. Formato de los paquetes de estatus

El campo *PARAM NUM* identifica el parámetro de control al cual se refieren los datos contenidos en el paquete.

Paquetes de error

En caso de que el controlador de bajo nivel detecte un problema con alguno de los dispositivos del sistema, un paquete de error sera construido y enviado al controlador principal. En la figura 4.6 se muestra el formato de este tipo de paquete.

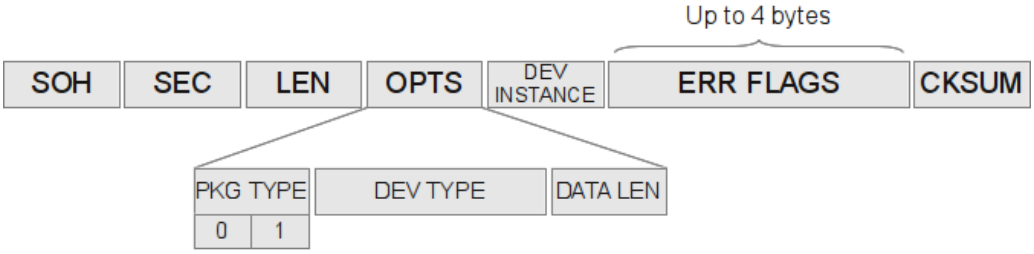

Fig. 4.6. Formato de los paquetes de error

El campo *DEV INSTANCE* indica cual de las instancias del dispositivo, especificado por el campo *DEV TYPE*, presenta el problema.

El campo *ERR FLAGS* especifica que tipo de problema ha ocurrido, su tamaño puede ser de hasta 4 *bytes* y cada uno de los bits que contiene hace referencia a un tipo de error en particular. En el capitulo 5 se describen los posibles errores, tanto de los actuadores como del sensor inercial, que pueden ser reportados.

4.2.3. Comandos

Además de paquetes de datos, el protocolo soporta secuencias de caracteres que indican una acción de control sobre la comunicación o el control de movimiento del robot.

Todas los comandos tienen un tamaño de tres *bytes* y hacen uso del siguiente formato :

<*Identificador*> ~ <*Identificador*>

Nota: ~ representa la operación complemento a uno.

La secuencia inicia con el carácter # (0x23 en el código ASCII) seguido por un carácter identificador del comando (una letra) y el complemento de éste para hacer posible la detección de errores.

Los comandos soportados son los siguientes :

- ***Reinicio:*** Identificado con la letra *r* (0x72 en el código ASCII). Indica al controlador de bajo nivel que debe prepararse para configurar nuevamente el sistema.
- ***Inicio:*** Identificado con la letra *s* (0x73 en el código ASCII). Una vez finaliza la fase de configuración se utiliza este comando para indicar al controlador de bajo nivel el inicio de la fase de ejecución de movimiento.
- ***Pausa:*** Identificado con la letra *p* (0x70 en el código ASCII). Se usa para indicar al controlador de bajo nivel que el envío de paquetes de instrucción sera suspendido. Para reanudar el envío de estos paquetes se debe enviar primero el comando de inicio.
- ***Token:*** Identificado con la letra *t* (0x74 en el código ASCII). Como se mencionó al inicio de este capitulo el protocolo sigue un modelo de maestro-esclavo; el comando *token* se emplea para que un dispositivo pueda ceder a otro el control sobre la comunicación, es decir, indica un cambio de roles.

4.2.4. Transferencia de datos

La transferencia de comandos y paquetes se realiza a través de un puerto serial emulado mediante una interfaz USB 2.0. Se ha decidido trabajar con esta interfaz por conveniencia en cuanto a la conectividad física entre el sistema de control de alto nivel (*PC* o *SBC*) y la tarjeta CM 900.

Cuando uno de los dispositivos recibe un paquete a través de la interfaz USB, éste examina su contenido para verificar que el numero de secuencia, el tamaño y la suma de comprobación sean correctos, en caso de que no se encuentren errores el receptor devuelve el carácter *ACK* (0x06 en el código ASCII) como confirmación, de lo contrario envía el carácter *NAK* (0x15) y queda a la espera de la retransmisión del paquete.

Lo mismo ocurre en el caso de los comandos: cuando un dispositivo recibe un comando verifica

que este sea correcto comparando el identificador del comando con su complemento, si la secuencia es valida devuelve una confirmación (*ACK*), de lo contrario envía una solicitud de retransmisión (NA*K*) del comando.

4.3. Parámetros de configuración y control

Los términos configuración y control se utilizan para diferenciar los parámetros involucrados en el control de bajo nivel del robot. El primero se refiere a los parámetros bajo los cuales se lleva a cabo el control de los dispositivos que componen al robot. El otro hace referencia a parámetros propios de los dispositivos que son controlados. En esta sección se listan los parámetros pertenecientes a ambas categorías y se describen los más relevantes.

4.3.1. Parámetros de configuración

Como se menciono estos parámetros determinan la manera en que el sistema de bajo nivel ejerce el control sobre los dispositivos pertenecientes al robot. Los parámetros de configuración pueden dividirse en parámetros globales y parámetros de dispositivo.

Parámetros globales

El valor de estos parámetros determina aspectos generales del control de bajo nivel. Los parámetros globales son los siguientes:

- ***Periodo:*** Durante la fase de ejecución de movimiento esta variable indica el periodo de ejecución de los paquetes de instrucción que son recibidos.
- ***Numero de dispositivos:*** Se refiere al numero de dispositivos de tipo diferente que deben ser controlados. Por ejemplo, si el robot esta compuesto solo por motores y un sensor inercial, el numero de dispositivos de tipo diferente sera dos. Se soporta el control de hasta dos dispositivos de tipo diferente..

Parámetros de dispositivo

Para cada tipo de dispositivo que hace parte del robot es necesario definir los siguientes parámetros:

- ***Identificador del tipo de dispositivo:*** A cada uno de los tipos de dispositivo que componen el robot se le asigna un identificador único, este permite al sistema de bajo nivel elegir el conjunto de rutinas apropiado para manejarlo.

- ***Numero de instancias del dispositivo:*** Indica el numero de instancias de un tipo de dispositivo presentes en el robot. Se soportan hasta 16 instancias de un mismo tipo de dispositivo.
- ***Identificador de instancias:*** Este parámetro corresponde a un arreglo que contiene los identificadores correspondientes a cada una de las instancias de un tipo de dispositivo.
- ***Numero de parámetros de escritura:*** Corresponde al numero de parámetros de un tipo de dispositivo que deben ser modificados dentro de un mismo periodo. Se soporta el control de hasta dos parámetros.
- ***Parámetros de escritura:*** Es un arreglo que contiene la dirección de los parámetros de un tipo de dispositivo que deben ser modificados dentro de un mismo periodo.
- ***Numero de parámetros de lectura:*** Es el numero de parámetros de un tipo de dispositivo que deben ser leídos cada cierto numero de periodos, especificado por el parámetro *periodo de parámetros de lectura*. Se soporta la lectura de hasta cuatro parámetros para el total de dispositivos de tipo diferente que deban ser controlados.
- ***Parámetros de lectura:*** Arreglo con las direcciones de los parámetros de un tipo de dispositivo que deben ser leídos.
- ***Periodo de parámetros de lectura:*** Arreglo que contiene el numero de periodos que deben transcurrir para que se realice la lectura de un parámetro. A cada *parámetro de lectura* le corresponde un elemento de este arreglo.

Nota: El valor de los parámetros *numero de dispositivo* y *numero de instancias del dispositivo* han sido limitados a las cantidades correspondientes de estos parámetros en el robot Lola-OP™, ya que en torno a las necesidades de este sistema ha sido desarrollado el control de bajo nivel. Cualquier intento por portar los desarrollos logrados en este trabajo a otra plataforma robótica que sobrepase estos limites debe evaluar si se superan las capacidades del sistema de control.

Nota: La restricción impuesta a la cantidad de parámetros controlados y leídos se debe a la necesidad de establecer condiciones bajo las cuales se garantice al usuario la consistencia entre la ejecución de una instrucción y el momento que fue definido para su ejecución.

4.3.2. Parámetros de control

Cada tipo de dispositivo que es controlado cuenta con un conjunto de parámetros que pueden ser modificados o leídos. A estos parámetros se les ha llamado parámetros de control. En esta sección se enumeran los parámetros de control de los dispositivos que actualmente hacen parte del robot Lola-OP™, estos son: servomotores Dynamixel AX-12 y el sistema de medición inercial Razor SEN-10736.

Servomotores Dynamixel AX-12

Como se mencionó en la sección 3.1 estos actuadores cuentan con un microcontrolador en el que ha sido definido un conjunto de registros que contienen el valor de los parámetros de control de los

dispositivos. En la figura 4.7 se muestran los parámetros de control de los actuadores.

En la tabla de parámetros de control que proporciona Robotis Inc., compañía que fabrica los actuadores, se encuentra la dirección, el nivel de acceso y el valor inicial correspondiente a cada parámetro. Nótese que el valor de algunos parámetros es almacenado en dos posiciones de memoria, donde una de ellas contiene la parte baja y la otra la parte alta.

Los parámetros especialmente relevantes para el control de bajo nivel son los siguientes:

- ***Baud Rate:*** Define la velocidad de la comunicación con el dispositivo. Se ha decidido trabajar con el máximo *baud rate* soportado por los actuadores, cuyo valor es de 1 *Mbps*, con el fin de minimizar el tiempo de transferencia de datos. Es importante que el usuario defina este parámetro antes de ejecutar el control de bajo nivel, de lo contrario no sera posible establecer la comunicación con los actuadores.
- ***Return Delay Time:*** Cuando uno de estos actuadores recibe un paquete de instrucción, el tiempo de espera antes de devolver el paquete de estatus correspondiente esta dado por el valor de este parámetro. El controlador de bajo nivel considera este parámetro para establecer un tiempo de espera máximo en la recepción de la respuesta a un paquete de instrucción. El tiempo de espera puede tomar valores entre 1 μs y 500 μs.
- ***Status Return Level:*** Este parámetro determina en que casos se debe devolver un paquete de estatus, más precisamente, ante que tipos de instrucción se debe enviar una respuesta. Los posibles casos son: responder a todas las instrucciones, responder solo a instrucciones de lectura, y no enviar una respuesta a ninguna de las instrucciones recibidas.
  En el control de bajo nivel es imprescindible la recepción de la confirmación de los paquetes de escritura, ya que son la única garantía de que la instrucción fue ejecutada, por ello los actuadores son configurados por el controlador de bajo nivel para que respondan a cualquier tipo de instrucción. Esto se realiza en la fase de configuración.

Unidad de medición inercial Razor SEN-10736

Los parámetros de control de este dispositivo son: azimut (*yaw*), elevación (*pitch*) y alabeo (*roll*), cada uno tiene un tamaño de 4 *bytes*. Cada uno de estos parámetros es calculado por el microcontrolador a bordo de la IMU a partir de la información adquirida de los distintos sensores que la componen. Los tres parámetros son enviados al sistema de bajo nivel cuando se recibe una solicitud de *frame* o trama. Debido a que no es posible solicitar el valor de cada uno de los parámetros por separado, el control de bajo nivel los considera como un único parámetro que ha sido llamado *orientación*, su tamaño es de 12 *bytes*.

	Address	Item	Access	Initial Value
EEPROM Area	0(0X00)	Model Number(L)	RD	12(0x0C)
	1(0X01)	Model Number(H)	RD	0(0x00)
	2(0X02)	Version of Firmware	RD	?
	3(0X03)	ID	RD,WR	1(0x01)
	4(0X04)	Baud Rate	RD,WR	1(0x01)
	5(0X05)	Return Delay Time	RD,WR	250(0xFA)
	6(0X06)	CW Angle Limit(L)	RD,WR	0(0x00)
	7(0X07)	CW Angle Limit(H)	RD,WR	0(0x00)
	8(0X08)	CCW Angle Limit(L)	RD,WR	255(0xFF)
	9(0X09)	CCW Angle Limit(H)	RD,WR	3(0x03)
	10(0x0A)	(Reserved)	-	0(0x00)
	11(0X0B)	the Highest Limit Temperature	RD,WR	85(0x55)
	12(0X0C)	the Lowest Limit Voltage	RD,WR	60(0X3C)
	13(0X0D)	the Highest Limit Voltage	RD,WR	190(0xBE)
	14(0X0E)	Max Torque(L)	RD,WR	255(0XFF)
	15(0X0F)	Max Torque(H)	RD,WR	3(0x03)
	16(0X10)	Status Return Level	RD,WR	2(0x02)
	17(0X11)	Alarm LED	RD,WR	4(0x04)
	18(0X12)	Alarm Shutdown	RD,WR	4(0x04)
	19(0X13)	(Reserved)	RD,WR	0(0x00)
	20(0X14)	Down Calibration(L)	RD	?
	21(0X15)	Down Calibration(H)	RD	?
	22(0X16)	Up Calibration(L)	RD	?
	23(0X17)	Up Calibration(H)	RD	?
RAM Area	24(0X18)	Torque Enable	RD,WR	0(0x00)
	25(0X19)	LED	RD,WR	0(0x00)
	26(0X1A)	CW Compliance Margin	RD,WR	0(0x00)
	27(0X1B)	CCW Compliance Margin	RD,WR	0(0x00)
	28(0X1C)	CW Compliance Slope	RD,WR	32(0x20)
	29(0X1D)	CCW Compliance Slope	RD,WR	32(0x20)
	30(0X1E)	Goal Position(L)	RD,WR	[Addr36]value
	31(0X1F)	Goal Position(H)	RD,WR	[Addr37]value
	32(0X20)	Moving Speed(L)	RD,WR	0
	33(0X21)	Moving Speed(H)	RD,WR	0
	34(0X22)	Torque Limit(L)	RD,WR	[Addr14] value
	35(0X23)	Torque Limit(H)	RD,WR	[Addr15] value
	36(0X24)	Present Position(L)	RD	?
	37(0X25)	Present Position(H)	RD	?
	38(0X26)	Present Speed(L)	RD	?
	39(0X27)	Present Speed(H)	RD	?
	40(0X28)	Present Load(L)	RD	?
	41(0X29)	Present Load(H)	RD	?
	42(0X2A)	Present Voltage	RD	?
	43(0X2B)	Present Temperature	RD	?
	44(0X2C)	Registered Instruction	RD,WR	0(0x00)
	45(0X2D)	(Reserved)	-	0(0x00)
	46[0x2E)	Moving	RD	0(0x00)
	47[0x2F)	Lock	RD,WR	0(0x00)
	48[0x30)	Punch(L)	RD,WR	32(0x20)
	49[0x31)	Punch(H)	RD,WR	0(0x00)

Fig. 4.7. Parámetros de control Dynamixel AX-12

4.4. Lógica de control

En esta sección se describe la lógica involucrada en las fases de configuración y ejecución de movimiento, y los mecanismos utilizados por el controlador de bajo nivel en el manejo de los dispositivos que componen al robot.

4.4.1. Ventanas de tiempo

En la sección 4.3 se habló de ciertos parámetros que rigen la manera en que se ejerce el control de bajo nivel. Uno de ellos es llamado *periodo*, este determina la frecuencia a la cual se ejecutan las instrucciones recibidas desde el controlador principal, y representa además el tamaño de las ventanas de tiempo dentro de las cuales es necesario ejecutar un cierto numero de acciones, ésta cantidad está determinada por la configuración del sistema y puede variar entre ventanas de tiempo diferentes.

Estas acciones consisten en la lectura o escritura de parámetros de control de los dispositivos que componen al robot. Mediante los parámetros de configuración: 1. *numero de parámetros de escritura*, 2. *parámetros de escritura*, 3. *numero de parámetros de lectura*, 4. *parámetros de lectura* y 5. *periodo de parámetros de lectura*, el controlador de bajo nivel puede conocer que acciones deben llevarse a cabo dentro de una ventana de tiempo en particular.

En cada ventana de tiempo son ejecutadas todas las acciones de escritura, pero la ejecución de una acción de lectura dentro de cierta ventana de tiempo depende del parámetro *periodo de parámetros de lectura*, cuyos valores indican cuantas ventanas de tiempo deben transcurrir para que se realice la lectura de cada parámetro. El siguiente ejemplo intenta aclarar el concepto de ventanas de tiempo usado en este trabajo:

El sistema de control de bajo nivel ha sido configurado con los siguientes valores:

- *Periodo = T*
- *Numero de parámetros de escritura = 1*
- *Parámetros de escritura = {param_wr}*
- *Numero de parámetros de lectura = 2*
- *Parámetros de lectura = {param_rd0, param_rd1}*
- *Periodo de parámetros de lectura = {2, 3}*

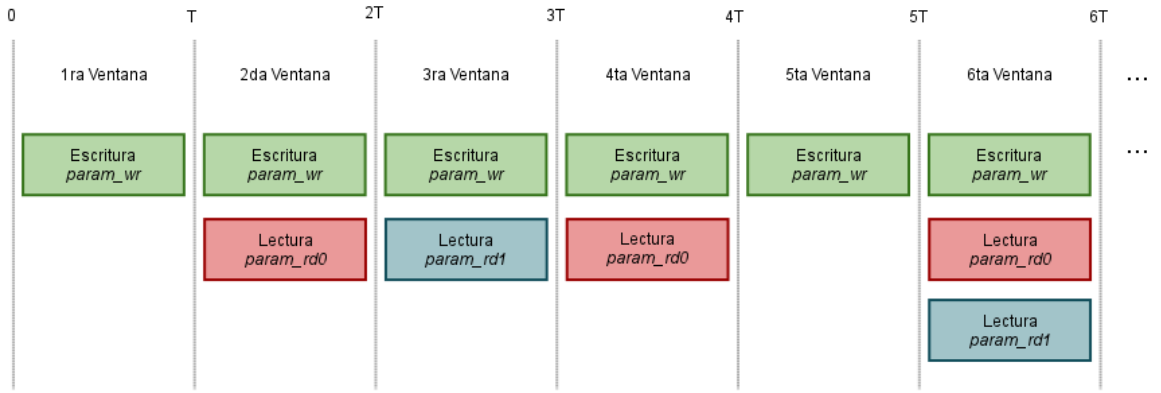

Fig. 4.8. Diagrama de ventanas de tiempo

La figura 4.8 muestra las acciones que son ejecutadas dentro de múltiples ventanas de tiempo según la configuración definida para este ejemplo. En el diagrama las acciones de escritura se ejecutan en todas las ventanas de tiempo, es decir, con cada periodo que transcurre. En cambio la ejecución de las acciones de lectura se da solo en las ventanas de tiempo que correspondan; el parámetro *param_rd0* cuyo periodo de es igual a dos es leído en cada ventana cuyo numero es par, de forma similar el parámetro *param_rd1* con periodo igual a tres es leído en las ventanas cuyo numero es múltiplo de tres.

4.4.2. Fase de configuración

Después de recibir un *comando de reinicio*, descrito en la sección 4.2.3, comienza a la fase de configuración. En esta fase el controlador principal envía *paquetes de configuración global* y *paquetes de configuración de dispositivo* al controlador de bajo nivel, el cual almacena y organiza la información contenida en ellos dentro de sus estructuras de control.

Una vez se ha obtenido el valor de todos los parámetros de configuración, descritos en la sección 4.3.1, el sistema de control pasa a comprobar la presencia, en las interfaces de comunicación correspondientes, de todas las instancias de los dispositivos que, según los parámetros de configuración, debe controlar. Para ello envía paquetes de *ping* a estos y espera su respuesta. En caso de que luego de varios intentos uno de los dispositivos no responda, el controlador de bajo nivel reportara la ausencia de éste enviando un paquete de error al control principal. Si la verificación concluye sin problemas, el control de bajo nivel queda a la espera del *comando de inicio*, el cual indica el paso a la fase de ejecución de movimiento.

4.4.3. Fase de ejecución de movimiento

El sistema de control de bajo nivel ha sido diseñado teniendo en mente su aplicación en robots móviles con el fin de brindar mayor autonomía a este tipo de sistemas. La plataforma robótica

Lola-OP™ pertenece a esa categoría y fue concebida para desplazarse en terrenos de diversa naturaleza.

Cuando se introduce este robot en el entorno en el que operará, la postura de éste depende en gran medida de la superficie y la configuración inicial adoptada al reposar sobre dicho terreno. Debido a esto la ejecución de la primera instrucción de movimiento, recibida desde el controlador principal, podría requerir de variaciones grandes en los ángulos de juntura del robot, si este es el caso y el movimiento es llevado a cabo de manera rápida los actuadores sufrirían un deterioro significativo.

Previendo esta situación el controlador de bajo nivel fue desarrollado de modo que la primera instrucción de movimiento se ejecute de forma "suave". Para ello se envían paquetes de escritura a los actuadores con el fin de modificar la velocidad angular a la que operan. El valor que se le da a este parámetro fue elegido para que en el peor de los casos (cuando la variación de un angulo es máxima) la ejecución total del movimiento dure cinco segundos. Dada la experiencia previa con este tipo de plataformas y actuadores, esto es lo bastante lento como para garantizar que no se producirán daños en éstos.

Luego de alcanzar la posición inicial se modifica nuevamente la velocidad angular para que esta tome su valor máximo. Esto fue decidido así en base a los resultados de experimentos llevados a cabo en trabajos anteriores, en los cuales el robot mostró un buen desempeño, en cuanto a locomoción, al configurar sus actuadores para trabajar a la velocidad máxima. Además de ajustar la velocidad el controlador de bajo nivel da inicio al conteo de los periodos que indicaran el momento en el que serán ejecutadas las acciones de escritura, el valor de este parámetro es definido durante la fase de configuración. Finalmente se lleva a cabo la lectura de los parámetros cuyo *periodo de lectura* coincida con la primera ventana de tiempo.

La figura 4.9 muestra de manera general la lógica que sigue el controlador de bajo nivel, una vez ha concluido la inicialización descrita anteriormente, a lo largo de la fase de ejecución de movimiento. A continuación se describe la lógica en este diagrama.

Nota: En esta sección se omite la descripción del manejo de los errores que puedan surgir durante la fase de ejecución, este tema es cubierto en el capitulo 5.

Gracias a que el módulo que maneja el protocolo de comunicaciones con el sistema de control principal descrito en la sección 4.1 cuenta con dos *buffers* de entrada, el sistema bajo nivel puede disponer simultáneamente del paquete de instrucción de la ventana de tiempo en la que se encuentra y el que corresponde a la ventana de tiempo siguiente. Ésto con el fin de que el paquete de instrucción correspondiente a una ventana de tiempo determinada pueda estar disponible para ser ejecutado de inmediato una vez ésta inicia. Teniendo esto en cuenta, el control de bajo nivel espera un nuevo paquete de instrucción antes de iniciar cada ventana de tiempo, si el paquete es recibido y esta ya ha iniciado, un paquete de error es enviado al controlador principal indicando que no fue posible cumplir con el periodo definido. En el caso contrario se procede a ejecutar la instrucción recibida.

La instrucción de escritura es enviada a los actuadores y se espera su confirmación. Si no se detectan problemas en la comunicación y en el paquete de estatus recibido no se reportan errores, el controlador de bajo nivel determina si la instrucción que fue enviada corresponde a un movimiento, si es así esperará a que éste concluya leyendo periódicamente el registro *moving* de la tabla de control de los actuadores, si otro tipo de instrucción fue enviada, el paso anterior se omite. Luego se verifica si quedan instrucciones de escritura pendientes y se repite este proceso.

Cuando todas las instrucciones de escritura son atendidas se verifica si debe realizarse la lectura de algún parámetro, si es así se envía un paquete de lectura a los dispositivos y se espera su respuesta. Si no se presentan problemas en la comunicación ni se reportan errores en los paquetes de respuesta, la información contenida en ellos es enviada al control principal. Este proceso es repetido hasta que todas las acciones de lectura correspondientes a la ventana de tiempo actual sean ejecutadas. Finalmente el controlador de bajo nivel da por terminada la ventana de tiempo actual y prepara las variables implicadas en la fase de ejecución (banderas, contadores y timeouts) para dar inicio a una nueva ventana de tiempo.

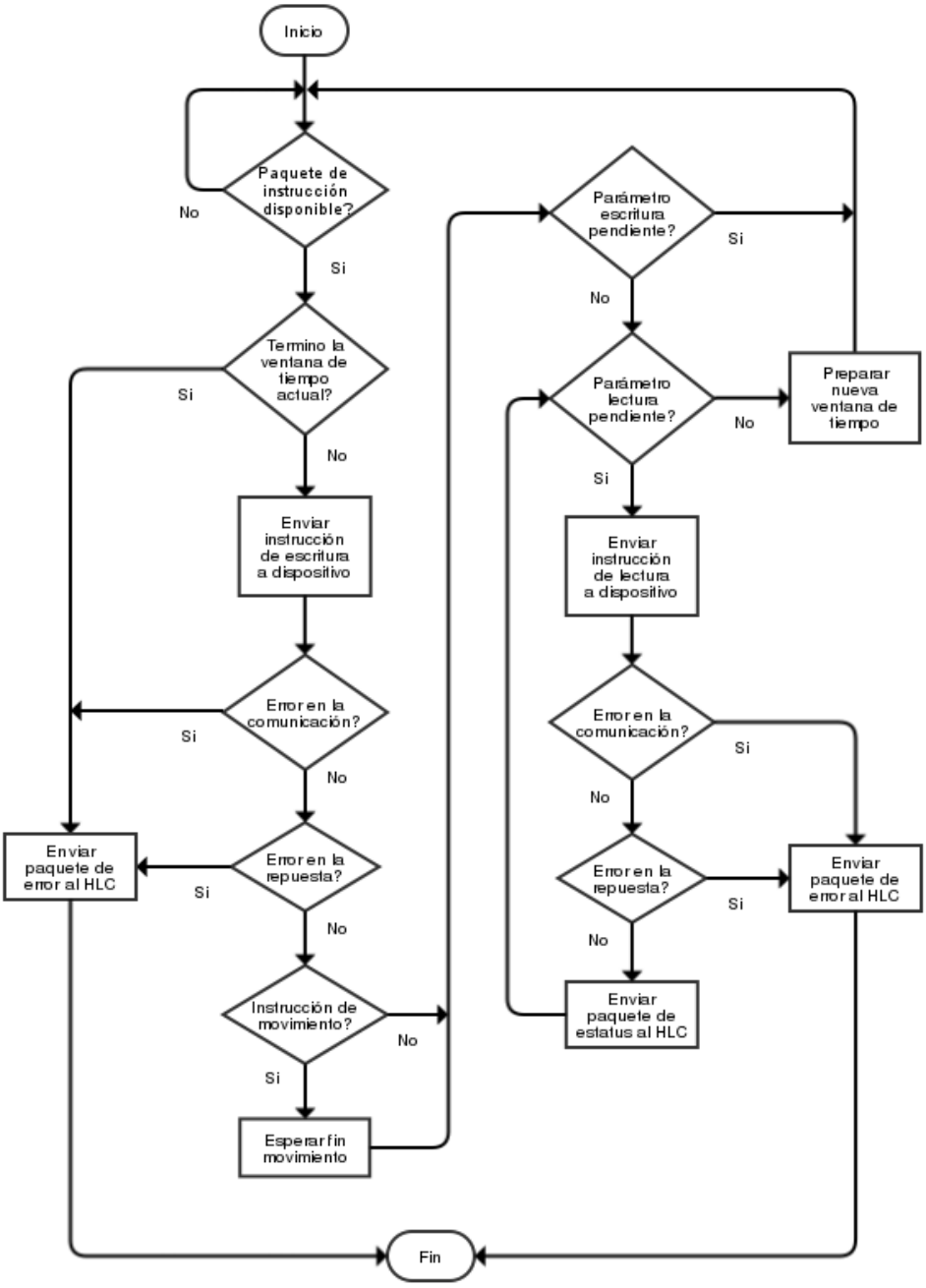

Fig. 4.9. Diagrama de flujo de la fase de ejecución de movimiento

Capitulo 5

Manejo local de excepciones

En este capitulo se describen los errores que pueden surgir en el control de los dispositivos que componen al robot Lola-OP™ y la manera en que son manejados por el sistema de bajo nivel. Estos errores se clasifican en dos categorías: errores en la comunicación y errores de dispositivos.

5.1. Errores en la comunicación

Corresponden a los errores que se producen durante la transferencia de información entre el sistema de control de bajo nivel y los dispositivos controlados. Estos errores son introducidos por el medio físico a través del cual se lleva a cabo la comunicación, y se deben a la presencia de ruido electromagnético y a problemas de conductividad en el cableado.

5.1.1. Perdida de datos

Cuando un paquete de instrucción es enviado a uno de los dispositivos controlados la información esperada como respuesta puede ser nula o encontrarse incompleta. Debido a que la UART del microcontrolador a bordo de la tarjeta CM 900 hace uso del método de *oversampling* para adquirir y validar las señales presentes en el puerto de entrada, un nivel alto de ruido electromagnético en el medio de transmisión puede provocar que los datos recibidos sean descartados, al detectarse variaciones importantes de la señal durante el tiempo en que cada *bit* es recibido.

La perdida de datos puede presentarse también por problemas en el cableado que puedan impedir la transmisión de las señales necesarias para la transferencia de información.

Por esto, una vez se envía un paquete de instrucción un *timeout* es iniciado y con cada *byte* recibido su valor es restablecido. Esto permite al controlador de bajo nivel detectar la perdida de un dato al verificar si el *timeout* ha finalizado, el cual es activado mientras no se haya recibido el total de los datos esperados, este valor es calculado por el controlador de bajo nivel antes del envío del paquete de instrucción.

5.1.2. Errores en la suma de comprobación y en el tipo de instrucción

Los actuadores *Dynamixel AX-12* tienen la capacidad de reportar, al controlador de bajo nivel, errores detectados por estos. Esto lo hacen mediante un campo en los paquetes de estatus destinado a ello, en donde cada bit representa uno de los errores que pueden ser detectados.

Uno de estos errores corresponde al originado por paquetes de instrucción en los que la suma de comprobación que estos incluyen no coincide con la calculada por el actuador, esto podría deberse a ruido electromagnético presente en el medio de transmisión. Otro caso que es interpretado como un error es el de un paquete cuyo tipo de instrucción no coincide con ninguno de los tipos soportados por estos actuadores, este caso también se considera un error en la comunicación ya que de ocurrir ha de ser porque el paquete fue alterado durante su transmisión, debido a que la construcción de los paquetes ha sido cuidadosamente implementada y validada, y por esto no existe la posibilidad de que sea un error del controlador de bajo nivel.

Estos errores son fácilmente detectados por el controlador de bajo nivel mediante la inspección del campo de error en los paquetes de estatus.

5.1.3. Manejo de los errores en la comunicación

Todos los posibles errores relacionados con la comunicación con los dispositivos se manejan de la misma manera: en caso de un problema con uno de los dispositivos el controlador de bajo nivel reenvía a éste la instrucción, si el problema persiste, y supera tres reintentos, un paquete de error es enviado al controlador de principal.

5.2. Errores de dispositivo

Hacen referencia a los errores propios de cada uno de los tipos de dispositivo que componen al robot. En esta sección se describen errores que pueden surgir en los actuadores *Dynamixel AX-12* y el sensor inercial *Razor SEN-10736.*

5.2.1. Servomotores Dynamixel AX-12

Como se mencionó en la sección anterior estos actuadores emplean uno de los campos de los paquetes de estatus, que envían como respuesta a paquetes de instrucción, para informar al controlador de bajo nivel que se ha producido un error. Además de los errores en la suma de comprobación y en el tipo de instrucción, clasificados como errores en la comunicación, los

actuadores son capaces de detectar los siguientes problemas:

- Niveles de voltaje de polarización fuera de los limites de operación recomendados.
- Un paquete con una instrucción de movimiento en el cual el valor del ángulo especificado se encuentra fuera de los limites permitidos.
- Sobrecalentamiento, la temperatura interna de los actuadores sobrepasa el valor máximo de operación.
- Un paquete con una instrucción de escritura cuyo valor se encuentra fuera de los limites definidos para el parámetro que debe ser modificado.
- Sobrecarga, la carga aplicada a un actuador supera el torque máximo del que este dispone.

5.2.2. Sistema de medición inercial Razor SEN-10736

Este sistema puede presentar errores en la lectura de los diferentes sensores que lo componen. Si esto ocurre el microcontrolador que pertenece a la IMU enviara una trama con el formato *!ERR: reading <sensor>* donde *<sensor>* corresponde al sensor que no fue posible leer.

5.2.3. Manejo de los errores de dispositivo

Los errores de dispositivo constituyen problemas que no pueden ser superados por el controlador de bajo nivel, por esto una vez ocurren son reportados de inmediato al controlador principal mediante paquetes de error.

Capitulo 6

Control de alto nivel

Entre los objetivos de este trabajo se encuentra el desarrollo de una API que permita la creación de instrucciones de alto nivel, las cuales serian usadas para controlar la plataforma robótica desde la unidad de procesamiento principal, estas instrucciones de alto nivel corresponden a los paquetes de datos empleados por el protocolo LLC-HLC, descrito en la sección 4.2.

Los desarrollos conseguidos en este trabajo superan el alcance de este objetivo y ofrecen no solo un conjunto de funciones que permiten crear estos paquetes de datos, sino toda una aplicación que otorga al usuario una capa de abstracción sobre el control de los dispositivos que componen al robot. En este capitulo se describe esa aplicación y la manera en que fue implementada.

En la figura 6.1 se muestra el diagrama de bloques de la arquitectura de software de la aplicación de alto nivel.

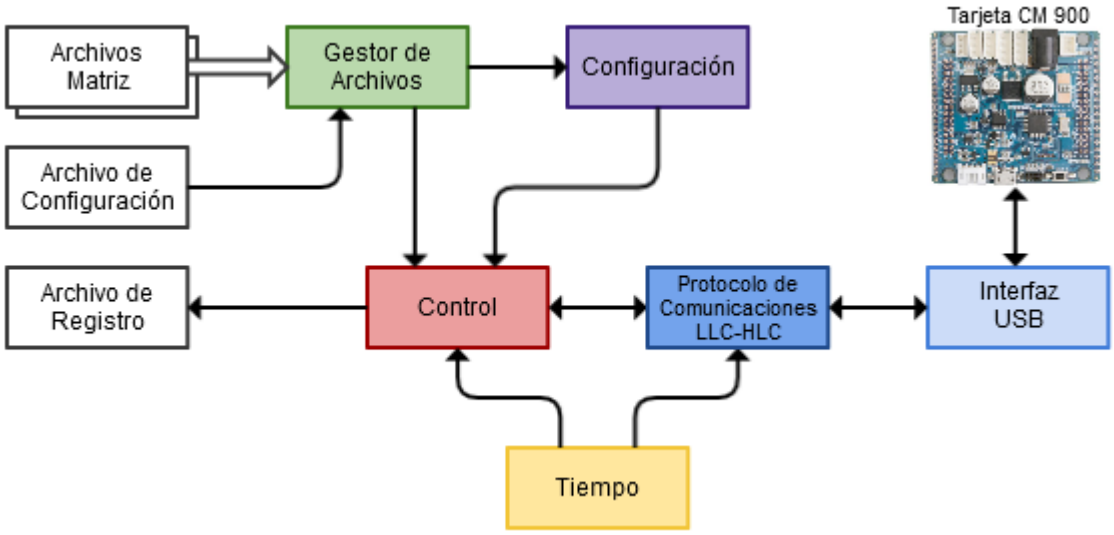

Fig. 6.1. Arquitectura de software de la aplicación de alto nivel

6.1. Archivos matriz, de configuración y de registro

Los bloques *archivos matriz*, *archivo de configuración* y *archivo de registro* no son módulos de software, sino archivos de texto plano (**.txt*) que contienen información relacionada con el control de los dispositivos que pertenecen al robot. Los primeros son entradas del sistema y el ultimo es una de las salidas.

La razón para trabajar con archivos de texto es la facilidad que ofrecen al usuario para visualizar su contenido y modificarlo de forma simple. Por ejemplo, un usuario puede conocer completamente la configuración del sistema de control con solo leer el *archivo de configuración*, y puede también modificar en este los parámetros de configuración según sea necesario. En esta sección se describe la función de cada uno de estos archivos y el formato de su contenido.

- ***Archivos matriz:*** Cada uno de estos archivos contiene una matriz cuyos elementos corresponden al valor de uno de los parámetros de control, definidos en la sección 4.3.2. Las filas de esta matriz corresponden a la ventana de tiempo en la que el parámetro debe ser modificado, y las columnas a cada una de las instancias del dispositivo. El formato del contenido de estos archivos es el siguiente:

$$\begin{matrix} P_{11} & P_{12} & \cdots & P_{1N} \\ P_{21} & P_{22} & \cdots & P_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ P_{M1} & P_{M2} & \cdots & P_{MN} \end{matrix}$$

  Donde P corresponde al valor del parámetro de una de las instancias de un dispositivo para una determinada ventana de tiempo; N y M son, respectivamente, el numero de instancias del dispositivo y el total de las ventanas de tiempo.

  Estos archivos son usados por la aplicación de alto nivel para construir los *paquetes de instrucción*, descritos en la sección 4.2.2, que son enviados al controlador de bajo nivel.

- ***Archivo de configuración:*** En este archivo se almacenan los parámetros de configuración descritos en la sección 4.3.1. La información en estos archivos es extraída por la aplicación de alto nivel y enviada al controlador de bajo nivel mediante paquetes de configuración, descritos en la sección 4.2.1.

  Según el siguiente formato el usuario debe especificar todos los parámetros bajo los cuales se llevara acabo el control de los dispositivos que conforman al robot.

```
PROGRAM_PARAM = {
   PERIOD = <Time value>
```

```
   REPEAT = <How many times are executed matrix files>
   DEV_NUM = <Number of devices that are controlled>
   ANG_UNIT = <Angle units; radians or degrees>
}

<Device name> = {
   TYPE = <Actuator or sensor>
   NUM = <Number of the device instances>
   ID = {<Array of device instances ids>}
   FILE_NUM = <Number of parameters to be written>
   FILE_NAME = {<Array with the names of the matrix files>}
   PARAM_NUM = <Number of parameters that will be read>
   PARAM = {<Array with the names of the read parameters and
   their reading period>}
}
```

Todo archivo de configuración contiene una estructura llamada *PROGRAM_PARAM* que contiene los *parámetros de configuración globales (PERIOD* y *DEV_NUM)* del controlador de bajo nivel. Además esta estructura contiene los parámetros *REPEAT* y *ANG_UNIT*; el primero indica el numero de veces que deben ser ejecutados los archivos matriz y el segundo la unidad (radianes o grados) de los archivos matriz que contengan ángulos.

Además de la estructura *PROGRAM_PARAM*, el archivo de configuración debe contener al menos una estructura de tipo *DEVICE*. Este tipo de estructura contiene la información de configuración para un tipo especifico de dispositivo; a esta información se le ha llamado *parámetros de dispositivo* en la sección 4.3.1.

El siguiente es un ejemplo de archivo de configuración usado para llevar a cabo el control de los dispositivos del robot Lola-OP™.

```
PROGRAM_PARAM = {
  PERIOD = 100
  REPEAT = 0
  DEV_NUM = 2
  ANG_UNIT = RAD
}

AX-12 = {
  TYPE = ACTUATOR
  NUM = 16
  ID = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}
  FILE_NUM = 1
  FILE_NAME = {goal_position}
  PARAM_NUM = 2
  PARAM = {goal_position = 1, present_temperature = 10}
```

```
}

RAZOR_IMU = {
  TYPE = SENSOR
  NUM = 1
  ID = {0}
  PARAM_NUM = 1
  PARAM = {RPY = 3}
}
```

En este ejemplo fue definida la siguiente configuración para llevar a cabo el control de los dispositivos del robot:

- El periodo de ejecución de las filas de los archivos matriz es de 100 *ms.*
- Los archivos matriz son ejecutados completamente un numero indefinido de veces, ya que se ha asignado al parámetro *REPEAT* el valor cero.
- El numero de dispositivos de tipo diferente (*Dynamixel AX-12* y *Razor IMU*) que sera controlado es igual a dos.
- Las unidades de los archivos matriz que contengan valores de ángulos estarán en radianes.
- Para los servomotores, identificados con el nombre *AX-12*, se especifica que serán controladas 16 instancias de este tipo de dispositivo; sus respectivos identificadores toman valores desde 1 hasta 16. El numero de parámetros que serán modificados con cada periodo esta dado por el valor de *FILE_NUM*, en este caso es solo uno, que corresponde a la posición o angulo destino (*goal_position*). De manera similar, el numero de parámetros que pueden ser leídos dentro de una ventana de tiempo es determinado por *PARAM_NUM*; las lectura de la posición destino y la temperatura serán realizadas, respectivamente, cada 100 *ms* y 1 *s*.
- En el caso del sensor inercial, identificado con el nombre *RAZOR_IMU*, se ha definido una sola instancia del mismo, cuyo identificador es 0. Por tratarse de un sensor no se modifica ninguno de sus parámetros mediante archivos matriz. La lectura del parámetro llamado RPY (*Roll*, *Pitch* y *Yaw*) sera realizada cada 300 *ms*.

- ***Archivo de registro:*** En este archivo se almacena el valor de las lecturas realizadas en cada una de las ventanas de tiempo que serán ejecutadas. El archivo esta organizado de tal manera que la información de cada parámetro leído se encuentra precedida por el numero de la ventana de tiempo en la cual fue realizado, el identificador del dispositivo al que corresponde y su dirección o identificador de parámetro. El formato de usado para especificar el valor de los parámetros leídos dentro de cada ventana de tiempo se muestra a continuación:

```
<Time slot num>:
<Dev X ID>-<Param A addr>: A0     A1     …     AN
<Dev Y ID>-<Param B addr>: B0     B1     …     BM
…
```

En este ejemplo X y Y representan dispositivos de tipo diferente; A y B se refieren al valor de dos parámetros diferentes; N y M corresponden al numero de instancias de X y Y respectivamente. Cada uno de los valores de cada parámetro esta relacionado a una de las instancias del dispositivo, el orden de estos valores coincide con el del arreglo de *identificadores de instancia* en el archivo de configuración.

6.2. Módulos de software de la aplicación de alto nivel

En esta sección se describen los módulos que hacen parte de la arquitectura de software de la aplicación de alto nivel, la cual se muestra en la figura 6.1.

- ***Gestor de archivos:*** Se encarga de extraer la información contenida en el archivo de configuración y los archivos matriz. Debido a que la información en estos archivos consiste en texto plano es necesario, ademas del manejo de sus formatos, interpretar estos caracteres para obtener el valor que representan, de modo que esa información pueda ser usada por otros módulos.
- ***Configuración:*** Este módulo tiene como función organizar la información que se encuentra en el archivo de configuración, adquirida a través del *gestor de archivos*, en estructuras de datos que serán usadas por el módulo de control.
- ***Control:*** En la fase de configuración del controlador de bajo nivel, descrita en la sección 4.4.2, este módulo se encarga de organizar la información contenida en el archivo de configuración y solicitar a *protocolo LLC-HLC* el envió de ésta.
  Durante la fase de ejecución, descrita en la sección 4.4.3, el módulo de control obtiene, a través del *gestor de archivos*, en cada ventana de tiempo una nueva linea de los archivos matriz que fueron definidos, y solicita el envió de los paquetes de instrucción correspondientes al sistema de bajo nivel. También evalúa que parámetros deben ser leídos, según el archivo de configuración, en una ventana de tiempo determinada y almacena la información correspondiente en el archivo de registro siguiendo el formato descrito en la sección anterior.
- ***Protocolo de Comunicaciones LLC-HLC:*** Este módulo se encarga de la construcción, interpretación y validación de los paquetes de datos definidos por el protocolo de comunicaciones desarrollado para este trabajo, descrito en la sección 4.2, el cual permite la transferencia eficaz de información entre el sistema de control principal y la tarjeta CM 900.
- ***Interfaz USB:*** Este módulo se encarga del manejo de un puerto serie emulado mediante una interfaz USB, gracias al cual se establece la comunicación con el sistema de control de bajo nivel.

- ***Tiempo:*** Este módulo provee funciones de temporización al sistema, permitiendo a otros módulos establecer periodos y timeouts, y verificar la ocurrencia de estos eventos.

Capitulo 7

Rutina de demostración de locomoción

Cuando a la tarjeta CM 900 no le es posible establecer la comunicación con el sistema de control principal, mediante la interfaz USB, esta pasa a ejecutar un conjunto de instrucciones de movimiento almacenadas en la memoria del microcontrolador. A ese conjunto de instrucciones se le ha denominado rutina de demostración. En este capitulo se describe el mecanismo que utiliza el control de bajo nivel para reconocer que la rutina de demostración ha de ser efectuada, y la lógica que sigue durante su ejecución.

A pesar de que no es posible conocer los resultados de la ejecución de una rutina de demostración (debido a que no se dispone de comunicación con el control de alto nivel), la lógica de control usada en este caso es una versión simplificada de la lógica descrita en secciones anteriores, por ello una vez ésta ultima es validada se espera que su versión simplificada sea correcta también, ésta ademas ha sido depurada mediante un puerto serie del microcontrolador.

Cabe aclarar que el valor practico de estas rutinas es mostrar el potencial en cuanto a autonomía al integrar la tarjeta CM 900 a la plataforma robótica, ya que prueba como ésta sin necesidad de un sistema externo puede ser controlada.

7.1. Fase de configuración

Una vez comienza el programa de control de bajo nivel, el cual se ejecuta sobre la tarjeta CM 900, éste realiza la inicialización de los módulos de software que componen al sistema, la cual consiste principalmente en la asignación de valores iniciales a los parámetros de control de estos módulos. Por ejemplo, para el módulo *Protocolo de Comunicaciones LLC-HLC* se define, en esta primera fase, el tamaño del *buffer* usado para almacenar temporalmente la información recibida, el valor inicial de los indices necesarios para acceder a ese *buffer* y las banderas de software empleadas por ese módulo en particular.

En el caso del módulo *Interfaz USB* parte de la inicialización consiste en esperar a que se establezca la comunicación con el controlador principal, revisando constantemente el estado del periférico que lleva a cabo el manejo de este protocolo. Un *timeout* de 10 segundos ha sido definido

para permitir al control de bajo nivel reconocer que no es posible comunicarse con el controlador principal. Cuando se ha vencido este tiempo se lleva a cabo la fase de configuración.

El valor de los *parámetros de configuración*, descritos en la sección 4.3.1, que serán usados en la ejecución de la rutina de demostración se encuentran definidos en el archivo fuente principal (*main.c*). Por ésto la fase de configuración se limita a la inicialización de la estructura de datos que contiene estos parámetros, es decir, en esta fase se asigna a los campos de esta estructura el valor, definido previamente en el archivo fuente principal, de los diferentes *parámetros de configuración*.

7.2. Fase de ejecución de movimiento

Los valores de ángulo que serán enviados a los actuadores se encuentran almacenados en una matriz de dimensiones *N* x *M*, donde *N* corresponde al numero total de pasos de tiempo que requiere la rutina de demostración, y *M* es igual al numero de instancias del dispositivo que sera controlado. Esta matriz es definida como una variable en el archivo fuente principal (*main.c*) del programa de control de bajo nivel, esta variable cumple la misma función que los *archivos matriz*, descritos en la sección 6.1. A modo de ejemplo se muestra a continuación la declaración e inicialización de una matriz genérica de ángulos llamada *demo*.

```
unsigned char demo[<time step num (N)>][<actuators num (M)>] =
{
    {<value 00>, <value 01>, ... , <value 0M>},
    {<value 10>, <value 11>, ... , <value 1M>},
    ...
    {<value N0>, <value N1>, ... , <value NM>}
};
```

Durante la fase de ejecución de movimiento el controlador de bajo nivel recorre esta matriz, y a partir de sus elementos construye paquetes de instrucción que son enviados a los actuadores; con cada periodo que transcurre los paquetes de instrucción correspondientes a una fila de la matriz son enviados a los actuadores. En la figura 7.1 se muestra el diagrama de flujo de la fase de ejecución de movimiento.

La lectura de *parámetros de control* no esta soportada durante la ejecución de la rutina de demostración, esto es debido a que los valores de las lecturas que fueran realizadas no podrían ser obtenidos por el usuario al no disponerse de la comunicación con el control principal, y carecerían de utilidad.

7.3. Manejo de excepciones

En caso de que ocurran errores relacionados con la comunicación con un dispositivo, descritos

en la sección 5.1, el controlador de bajo nivel reenvía a éste la instrucción, si el problema persiste, y supera tres reintentos, se da inicio al parpadeo del *LED* de estatus presente en la tarjeta CM 900, usado como indicador de un problema cuando se ejecuta una rutina de demostración.

Si ocurre un error de dispositivo, descritos en la sección 5.2, se da inicio de inmediato al parpadeo del *LED* de estatus.

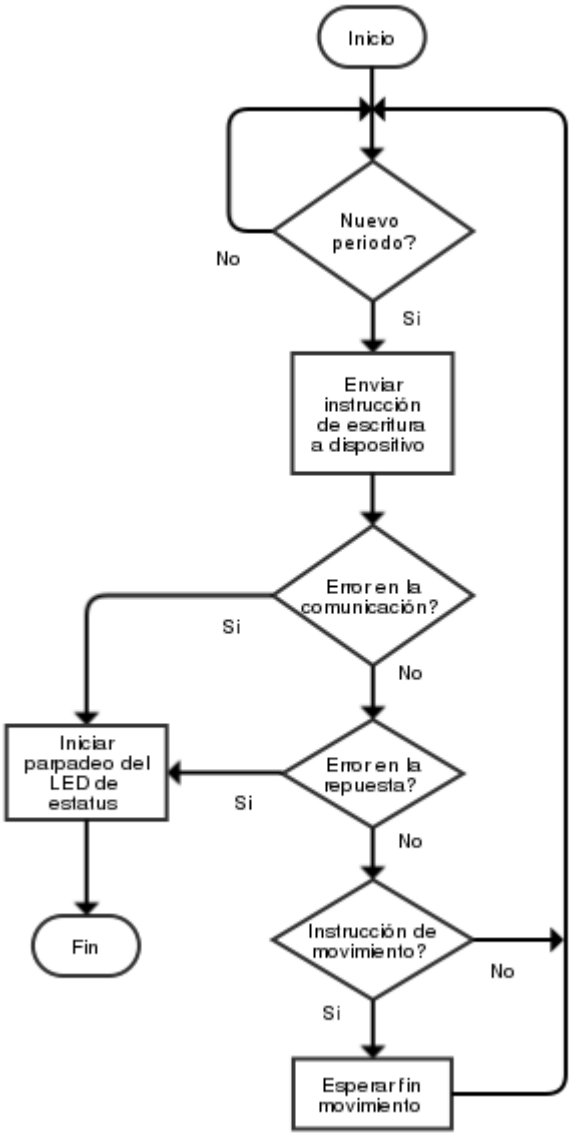

Fig. 7.1. Diagrama de flujo de la fase de ejecución de movimiento

Capitulo 8

Problemas conocidos

La tarjeta CM 900 es actualmente un proyecto en desarrollo de hardware abierto, liderado por Robotis Inc, su fabricante, al cual contribuyen grupos de investigación en robótica de todo el mundo, entre ellos KM-RoBoTa, evaluando la funcionalidad de la tarjeta. Debido a que este sistema se encuentra aún en desarrollo existe la posibilidad de encontrarse con problemas inherentes a su diseño. Así ha ocurrido con una de las interfaces de comunicación de esta tarjeta. En este capitulo se describe este problema y otros más, derivados de este, y la solución provisional que fue implementada en este trabajo.

8.1. Interfaz de comunicaciones Dynamixel

Para llevar a cabo la transferencia de paquetes de instrucción y estatus, con los actuadores Dynamixel AX-12, se requiere de una interfaz serial *half-duplex* que opere usando niveles TTL (0 y 5 V). El microcontrolador a bordo de la tarjeta CM 900 dispone de UARTs *full-duplex* que manejan niveles CMOS (0 y 3.3 V) en el puerto de salida, y soportan tanto niveles CMOS como TTL en el puerto de entrada. Por ello esta tarjeta dispone de hardware adicional que permite convertir los niveles de voltaje y definir el sentido de la comunicación. En la figura 8.1[2] se muestra el circuito presente en la tarjeta CM 900 encargado de esas funciones.

Este módulo consiste en un *buffer 3-state* que maneja niveles TTL a la salida y soporta tanto niveles TTL como CMOS a la entrada, y un filtro *EMI* (*Electromagnetic Interference*) conectado a los puertos de la UART. El *buffer 3-state* se encarga de la conversión de los niveles de voltaje y de establecer el sentido de la comunicación, mientras que el filtro *EMI* tiene como función minimizar las componentes de alta frecuencia (ruido) que puedan aparecer en las señales eléctricas involucradas en la comunicación.

Durante el desarrollo de este trabajo se produjo repentinamente un fallo permanente en la tarjeta CM 900, el cual impide la comunicación con los actuadores. El fallo consiste en la imposibilidad de

2 Imagen tomada de *CM-900 REV 1.01 Schematic*, disponible en: https://github.com/robotis-pandora/ROBOTIS_CM9_Series

la interfaz de comunicaciones Dynamixel, cuyo circuito se muestra en la figura 8.1, para propagar las señales presentes en sus entradas a las salidas correspondientes.

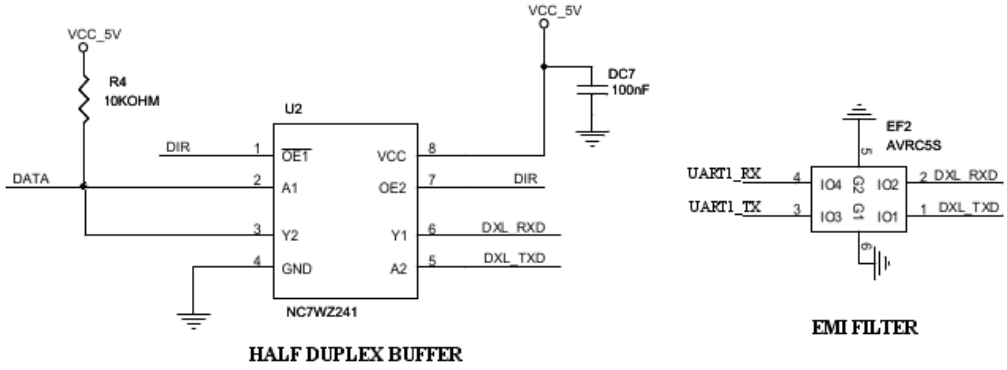

Fig. 8.1. Interfaz de comunicaciones Dynamixel en la tarjeta CM 900

Al inicio de septiembre de 2013 Robotis Inc, quien fabrica las tarjetas CM 900, publico en foros de la comunidad Robotsource la presencia de este problema en todas sus tarjetas y anuncio el lanzamiento de su reemplazo, en el cual el problema es corregido. A continuación se copian textualmente los comentarios publicados por representantes de Robotis Inc en donde se refieren al problema.

> "***3-state dual buffer IC in TTL circuit is much vulnerable to any electric noise generated sudden and ramdomly, especially ESD or surge****...etc. But RS-485 circuit is okay, it is better robust than TTL circuit. So, If you use RS-485, it is not problem but if not,* ***I do not recommand to use CM-900 for making robot****. We almost finished next cm-9 serise(CM-9.04) as you know, it has a TTL circuit fixed old problem,yes it do not provide RS-485 pins, only TTL 3pin, but RS-485 expantion board will be ready at same time."*

> *"The square CM-900 had a known hardware issue with the "TTL" dynamixel bus.* ***The buffer IC used to convert the full-duplex 3.3V UART of the STM32 into the half-duplex 5V UART of the "TTL" dynamixel bus was not as robust as expected and would become damaged under some conditions****. If the board works in all ways except for a non-responsive dynamixel bus, then it is very likely that the buffer IC has been damaged. The buffer IC can be replaced using a hot-air rework station and the correct replacement part, but most people do not have access to that equipment. The replacement part number is found in the CM-900 BOM excel file on the CM9 github repository (part should be NC7WZ241K8X in US-8 package). It is about $0.53USD in single quantities from a US supplier, but shipping might be expensive."*

Estos comentarios e información adicional acerca del problema pueden encontrarse en los siguientes enlaces:

- http://www.robotsource.org/bs/bd.php?bt=forum_CM9DeveloperWorld&bt_id=462#c_466

- http://www.robotsource.org/bs/bd.php?bt=forum_CM9DeveloperWorld&bt_id=491

Como una medida temporal, que permitiera continuar con el desarrollo del proyecto de forma rápida, se realizó el montaje de un circuito equivalente al de la figura 8.1 en una *protoboard*. El diagrama del circuito utilizado se muestra en la figura 8.2.

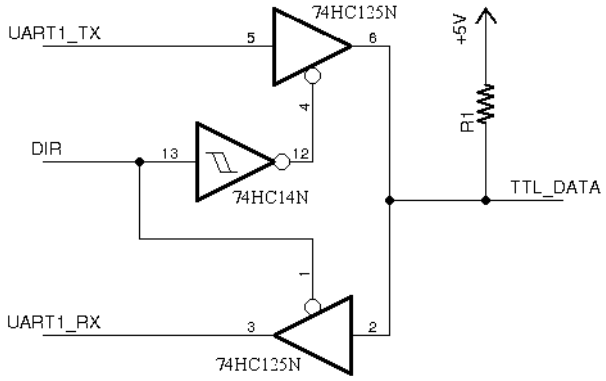

Fig. 8.2. Circuito de reemplazo para la interfaz de comunicaciones Dynamixel

En este circuito se emplea un *buffer 3-state* y una compuerta inversora, ambos pertenecientes a la familia lógica 74HC, la cual se caracteriza por manejar niveles de voltaje TTL y ofrecer altas velocidades de conmutación, este ultimo aspecto es relevante para posibilitar la comunicación con los actuadores a un *baud rate* de 1 *Mbps*.

La solución presentada es en gran medida rudimentaria ya que el circuito usado como reemplazo no incluye un filtro *EMI* y, como se mencionó, el montaje fue llevado a cabo sobre una *protoboard*, por lo que fue necesario utilizar cables de una longitud considerable para realizar la conexión entre este circuito y la UART del microcontrolador. Por todo esto debe tenerse en cuenta que el circuito de reemplazo es altamente vulnerable al ruido electromagnético, y no puede considerarse como una solución definitiva.

En la tarjeta CM 904, sucesora de la CM 900, se ha corregido este problema. Gracias a que todo el código desarrollado en este trabajo esta escrito en el lenguaje de programación C, migrar de una tarjeta a otra es una tarea simple que no requiere más que algunos ajustes a las funciones de software que manipulan directamente el hardware. A mediados de noviembre del 2013 fue solicitada la importación de dos tarjetas CM 904, se espera disponer de ellas antes de la fecha de sustentación de este trabajo para que sea posible implementar sobre ellas los desarrollos alcanzados en este trabajo.

8.2. Interferencia electromagnética en la interfaz Dynamixel

A partir del problema descrito en la sección anterior surgieron otros problemas adicionales. El mas relevante de ellos es la alta vulnerabilidad de la interfaz de comunicaciones Dynamixel ante la interferencia electromagnética. Lo cual hizo muy frecuente la ocurrencia de errores en la suma de comprobación de los paquetes de estatus, e incluso la ausencia de estos ante paquetes de instrucción, de lo cual se infiere que estos últimos no llegaron a ser recibidos. Este problema llega a ser critico durante la ejecución de movimiento de los actuadores, los cuales se convierten en una fuente importante de ruido electromagnético.

Según lo anterior, se hizo necesario adaptar el módulo de software *interfaz de comunicaciones Dynamixel* para que fueran considerados los distintos errores que se presentan constantemente, como consecuencia del problema de hardware, en la comunicación con los actuadores, y que fueron identificados obteniendo mediante software el valor de algunos parámetros relativos a la comunicación, entre ellos: el numero total de datos recibidos y el valor de cada uno de estos.

Uno de estos errores se debía a la conmutación de los circuitos lógicos presentes en el reemplazo de la interfaz de comunicaciones Dynamixel. Luego de finalizar el envió de un paquete de instrucción es necesario invertir el sentido de la comunicación para permitir la recepción del paquete de estatus correspondiente. Esta transición provocaba que la UART reconociera la señal eléctrica resultante como el primer dato recibido del paquete de estatus, lo que normalmente invalidaba el paquete que se esperaba. Para evitar este efecto fue necesario definir en el módulo encargado del manejo del protocolo que este primer dato fuera ignorado. Ademas este problema imponía el aumento del valor del parámetro *status return delay* (propio de los actuadores) para garantizar que el primer dato del paquete de estatus no se viera alterado por el cambio en el sentido de la comunicación, implicando un aumento en el tiempo de espera de cada paquete de estatus.

Los efectos negativos de la interferencia electromagnética en la interfaz Dynamixel pueden resumirse en los siguientes puntos:

- Incremento del numero promedio de intentos necesarios para conseguir la transferencia efectiva de un paquete de instrucción y su respectivo paquete de estatus.
- Aumento del tiempo de espera del paquetes de estatus luego del envió de un paquete de instrucción.

Es importante tener en cuenta que aunque aquellos efectos puedan parecer despreciables si se consideran individualmente, estos se presentan con frecuencia cada vez que se hace necesario el intercambio de información entre el controlador de bajo nivel y los actuadores, lo que resulta en un impacto apreciable en el tiempo total que debe invertirse en la comunicación con estos dispositivos.

8.3. Periodo de ejecución de movimiento

Los problemas descritos en la sección anterior tienen un impacto importante en el sistema de control de bajo nivel, en cuanto a su capacidad para ejecutar de manera oportuna las instrucciones recibidas desde el control principal.

Numerosas pruebas de locomoción fueron realizadas con el robot Lola-OP™ con el fin de validar los desarrollos llevados a cabo en este trabajo. Estas pruebas consistían en la ejecución de un archivo matriz cuyos valores corresponden a los ángulos de las junturas de los módulos del robot. Múltiples archivos matriz fueron empleados, entre los cuales la diferencia de angulo entre filas consecutivas era variable, siguiendo siempre un mismo patrón de movimiento o *gait.* Entre una prueba y otra también se modificaba el periodo de ejecución de las filas del archivo matriz.

Mediante esas pruebas se pudo verificar el correcto funcionamiento de los módulos involucrados tanto en el sistema de control principal, como en el de bajo nivel. Por otro lado, al evaluar el periodo mínimo bajo el cual el sistema estaba en capacidad de responder oportunamente ante un determinado archivo matriz, se encontró que el rendimiento en relación a esa capacidad era más bien pobre.

Para el caso de un archivo matriz con diferencias de angulo menores a 5 grados entre lineas consecutivas, se encontró que el periodo mínimo de ejecución es cercano a los 300 *ms*. Para esa diferencia de angulo relativamente pequeña, el valor del periodo resulta demasiado alto y en consecuencia el desplazamiento del robot se realiza muy lentamente.

Al aumentar la diferencia de angulo, sin sobrepasar los 15 grados, entre lineas consecutivas en el archivo matriz, el periodo mínimo de ejecución aumento a un valor cercano a 700 *ms.* Nuevamente la relación periodo y diferencia de angulo no es buena, y el desplazamiento del robot continua siendo lento. En este caso se observo ademas la presencia de un mayor numero de errores en la comunicación con los motores, debida a niveles de ruido mayores provocados por estos. La acumulación de retardos causados por esos errores, hace posible apreciar un desfase en el inicio del movimiento entre los módulos del robot.

Capitulo 9

Conclusiones

Los desarrollos logrados constituyen una herramienta que proporciona un control total en el manejo de los dispositivos que componen al robot Lola-OP™. Esta herramienta representa una contribución importante al desarrollo de esta plataforma, ya que no solo permite la movilidad del robot sino también provee un mecanismo con el cual puede determinarse si su funcionamiento es correcto, es decir, hace posible conocer bajo que condiciones los dispositivos que componen al robot presentan una respuesta correcta y oportuna. Ademas, posibilita la obtención del valor de los diferentes parámetros que determinan el comportamiento del robot, permitiendo su análisis en el proceso de experimentación con el robot.

Aunque el proyecto esta dirigido al robot Lola-OP™ en particular, la arquitectura de software implementada en este trabajo puede usarse en el control de los dispositivos pertenecientes a otras plataformas robóticas, lo cual requeriría unicamente del desarrollo de los módulos que interactúan directamente con estos dispositivos. Ademas, gracias a que el software de control de bajo nivel ha sido escrito completamente en *C* estándar, no solo es posible extender este trabajo a otras plataformas robóticas sino también a otros sistemas de procesamiento, simplemente modificando las rutinas encargas del manejo de hardware. Por lo anterior pueden considerarse como portables y escalables los desarrollos conseguidos en torno a este trabajo.

En trabajos anteriores se ha llevado a cabo un análisis extenso del rendimiento de la plataforma robótica Lola-OP™ [1-6]. En ellos se han alcanzado tiempos de ejecución relativamente pequeños (cercanos a los 100 *ms*) con diferencias de angulo grandes entre pasos sucesivos (hasta 37°). Por otro lado, al implementar el control de bajo nivel tal y como ha sido planteado en este trabajo se han obtenido resultados inferiores, llegando a alcanzar un máximo de 10° entre pasos sucesivos con un periodo de 100 *ms*.

Por ende, en cuanto a locomoción, el rendimiento del sistema de control de bajo nivel no es comparable con el obtenido en trabajos anteriores, en los que se anteponía la velocidad del robot a la certeza de su correcto funcionamiento. En este trabajo se ha hecho gran énfasis en garantizar que cada una de las acciones que deban ser llevadas a cabo por los dispositivos pertenecientes al robot sean efectivamente ejecutadas por estos. Al conseguir un control total sobre estos dispositivos no solo se obtiene una plataforma robótica mas robusta sino también una mayor comprensión del hardware involucrado en ella, de manera que puedan ser reconocidas nuevas necesidades y tomar

medidas ante ellas, las cuales habrían sido difícilmente identificadas sin llevar a cabo primero un trabajo similar al presentado en este documento.

A partir de la experiencia de este trabajo se ha identificado la necesidad de optimizar la comunicación con los actuadores con el fin de obtener mejores resultados en la locomoción del robot. Para lograrlo se proponen tres alternativas:

1. Distribuir la comunicación entre múltiples puertos independientes asignando a cada uno de ellos una fracción del numero de actuadores presentes en el robot. Esto podría implementarse al replicar el modulo *Interfaz de Comunicaciones Dynamixel* para que cada una de sus instancias manejara un puerto diferente. Ademas seria necesario diseñar hardware que permita a esos puertos establecer la comunicación con los actuadores, ya que estos manejan una comunicaciones *half-duplex* con niveles TTL.
2. Evaluar la posibilidad de reducir el numero de módulos que conforman el robot, de modo que se requieran tiempos menores en la comunicación para lograr la ejecución de una fila de un archivo matriz.
3. Reemplazar los actuadores actuales por unos con mayores prestaciones, como los *Dynamixel MX*, los cuales admiten velocidades de transferencia de hasta 4 *Mbps*. Ante esta posibilidad habría que determinar si la relación costo beneficio es favorable para el proyecto.

Estas alternativas no son mutuamente excluyentes por lo que podrían ser implementadas conjuntamente.

Bibliografía

[1] 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.

[2] 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.

[3] K. Melo, L. Paez, M. Hernandez, A. Velasco, F. Calderon, and C. Parra, "Preliminary studies on modular snake robots applied on de-mining tasks," in Robotics Symposium, 2011 IEEE IX LatinAmerican and IEEE Colombian Conference on Automatic Control and Industry Applications (LARC), oct. 2011, pp. 1–6.

[4] Melo, K.; Leon, J.; Monsalve, J.; Fernandez, V.; Gonzalez, D., "Simulation and control integrated framework for modular snake robots locomotion research," System Integration (SII), 2012 IEEE/SICE International Symposium on , vol., no., pp.523,528, 16-18 Dec. 2012

[5] K. Melo, M. Hernandez, and D. Gonzalez, "Parameterized Space Conditions for the Definition of Locomotion Modes in Modular Snake Robots," in Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on, dec. 2012.

[6] K. Melo and L. Paez, "Modular Snake Robots on Horizontal Pipes," in Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, Oct. 2012.

[7] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. Chirikjian, "Modular Self-Reconfigurable Robot Systems [Grand Challenges of Robotics]," Robotics Automation Magazine, IEEE, vol. 14, no. 1, pp. 43–52, march 2007.

[8] S. Hirose and H. Yamada, "Snake-like robots [Tutorial]," Robotics Automation Magazine, IEEE, vol. 16, no. 1, pp. 88–98, march 2009.

[9] S. Ma, "Analysis of snake movement forms for realization of snake-like robots," in Roboticsand Automation, 1999. Proceedings. 1999 IEEE International Conference on, vol. 4, 1999, pp. 3007–3013 vol.4.

[10] H. Fukushima, M. Tanaka, T. Kamegawa, and F. Matsuno, "Path-tracking control of a snake-like robot using screw drive mechanism," in Intelligent Robots and Systems, 2008. IROS 2008.IEEE/RSJ International Conference on, sept. 2008, pp. 1624–1629.

[11] P. Liljeback, "Modelling, Development, and Control of Snake Robots," Ph.D. dissertation, NTNU, Norwegian University of Science and Technology, Feb. 2010.

[12] www.beagleboard.org

[13] "CM 900 Specification Sheet" en www.robotsource.org/, consultado el 28 de noviembre de 2013