UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

Proyecto Fin de Carrera

Estudio y análisis de las comunicaciones en el robot Scorbot-ER 4u.

Desarrollo de un driver e implementación para su uso en interfaz web.

Para la obtención del título de
Ingeniero en Informática
Titulación: Ingeniería en Informática

Autor: Yeray Miranda Betancor
Tutor: Pedro Medina Rodríguez
Marzo, 2017

Proyecto Fin de Carrera

Escuela de Ingeniería Informática

Título del proyecto

Estudio y análisis de las comunicaciones en el robot Scorbot-ER 4u. Desarrollo de un driver e implementación para su uso en interfaz web.

Autor: Yeray Miranda Betancor

Tutor: Pedro Medina Rodríguez

Resumen

Durante el proyecto se ha desarrollado un driver para controlar el robot Scorbot-ER 4u obteniendo la información relativa al control del mismo a través del estudio de los mensajes enviados en el puerto USB al controlador mediante un proceso de captura y decodificación. Una vez desarrollado el driver se han creado una serie de aplicaciones a modo de herramientas para demostrar su funcionamiento mediante diferentes pruebas y se ha creado además una interfaz web y un servidor que permiten su uso remoto. Estas pruebas han confirmado que el driver desarrollado consigue operar de forma correcta el robot.

Abstract

During this proyect we've developed a driver to control the Scorbot-ER 4u robot obtaining all the information about its control through the study of the messages sent through the USB port after capturing and decoding them. Once the driver has been developed we've also created a series of programs to serve as a toolset to test it and created a web interface and server that allow a remote use. These tests have confirmed that the driver is capable of correctly operating with the robot.

ÍNDICE

ÍNDICE DE FIGURAS

1. Introducción

La idea de desarrollar un driver para el brazo robótico Scorbot-ER 4u surgió a raíz de la imposibilidad de utilizar el mismo accediendo directamente a sus motores. Esta limitación no supone muchos problemas a nivel usuario o para actividades educativas que no requieran entrar en detalle de cinemáticas ya sean directas o inversas del mismo pero sí supone un inconveniente a la hora de trabajar con el mismo por ejemplo en un programa externo.

Antes de empezar a analizar el funcionamiento a nivel software se debe analizar el robot con el que vamos a trabajar: el Scorbot-ER 4u, este robot cuenta con un controlador dotado de entradas / salidas tanto digitales como analógicas así como de la posibilidad de controlar hasta 8 motores, 6 de los cuales se encuentran en el brazo robótico. Nos proporciona además una fuente de 12 voltios, aunque esta última es irrelevante a nivel software.

FIGURA 1 - CONTROLADOR DEL SCORBOT-ER 4U

Estos motores disponen de codificadores incrementales, por lo tanto, no codifican valores absolutos sino relativos al momento en que encendemos el robot donde todos indican un valor 0.

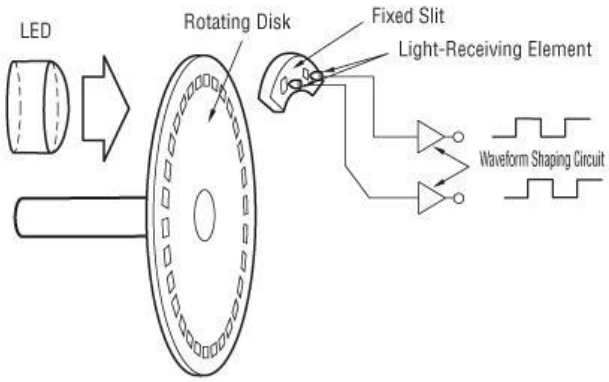


FIGURA 2 - CODIFICADOR INCREMENTAL, FUENTE: ORIENTALMOTOR.COM

FIGURA 3 - BRAZO ROBOT DEL SCORBOT-ER 4U

Esto explica la necesidad de realizar una llamada a Home, una operación de sincronización, por parte de Scorbase antes de operar con el mismo. La llamada a la función Home hace que el robot empiece a buscar diferentes marcas a través de microswitches de los que dispone en los ejes con el objetivo de empezar siempre la operación desde el mismo punto y por lo tanto proporcionar repetitividad en su funcionamiento.

Otra particularidad de este robot es que no disponemos de una tableta para mover sus ejes como sí tenemos en el Rhino por lo que el movimiento del mismo se hace a través del propio programa Scorbase por lo que habrá que implementar también este tipo de controles en nuestro driver.

Por lo tanto el objetivo es llegar a controlar y comunicarnos con el robot pero a diferencia del programa original Scorbase proporcionando además un control directo de los ejes y la utilización del robot en entornos fuera de Scorbase.

La solución propuesta fue la de crear un driver propio que nos permitiese acceder a los elementos básicos a la hora de controlar un robot, el estado de los codificadores motores y la posibilidad de mover los mismos. A la hora de afrontar el reto de desarrollar un driver para el robot se disponía de dos métodos para realizar el estudio de su funcionamiento original, el primero el desensamblado y estudio a nivel software del driver proporcionado por Intelitek y el segundo el estudio de los mensajes transmitidos por USB entre el programa Scorbase y el robot. Se ha optado por la segunda ya que resulta más interesante a la hora de analizar los mensajes y porque así no

dependeremos de partes del código original como podría haber sido el caso si no entendemos exactamente lo que representa cada parámetro y cómo trabajar con él. A través de la captura de mensajes USB se ha ido descubriendo el formato de los mensajes transmitidos, sus campos y posibles valores mediante el desarrollo de pruebas que han ido mostrando el funcionamiento de las funciones incluidas en Scorbase. También nos hemos encontrado con otros problemas como la sincronización o el control directo de los ejes que se han solucionado a nivel computacional con el uso de hilos independientes para la sincronización y de control como veremos más adelante.

Además del desarrollo del driver se han creado también una serie de aplicaciones para operar con el robot. La primera es una aplicación simple que se ha ido ampliando a medida que se incluían funciones en el driver y por lo tanto ha sido útil también durante el proceso de testeo y depuración, nos permite operar la pinza, mover una cantidad de pasos predefinida el motor que deseemos y trabajar con las entradas y salidas entre otras operaciones. La segunda es un intérprete que como su nombre indica se dedica exclusivamente a leer y enviar mensajes de un terminal al robot siempre y cuando sean mensajes válidos. Finalmente se ha desarrollado un conector PHP/C que nos permite a través de una interfaz web realizada en HTML5 con la librería Jquery UI interactuar con un intérprete como el visto anteriormente, esta aplicación web permite operar con el robot de manera parecida a como operamos con el robot a través del programa original Scorbase.

1.1 La Robótica

La robótica se define como una rama de la tecnología que se encarga del diseño, construcción y operación del hardware que compone un robot, así como del estudio de sus aplicaciones en diferentes campos tanto de las ciencias como de la industria.

La robótica ha avanzado mucho desde sus inicios, viendo unos ejemplos se ve que su evolución ha sido muy rápida y tiende a serlo cada vez más en los últimos años. El primer robot industrial de la historia, el Unimate, desarrollado por George Devol gracias a la ayuda proporcionada por Joseph Engelberger, considerado el padre la robótica, en 1954 y vendido a General Motors en el año 1961 era utilizado con el fin de mover las piezas metálicas que salían de las forjas, sin embargo automatizar un proceso aun siendo tan simple como ese le permitió a la empresa tomar ventaja respecto a todos sus competidores en el sector, debido en gran medida a que podía acelerar la producción hasta los 110 vehículos por hora, más del doble que la competencia.

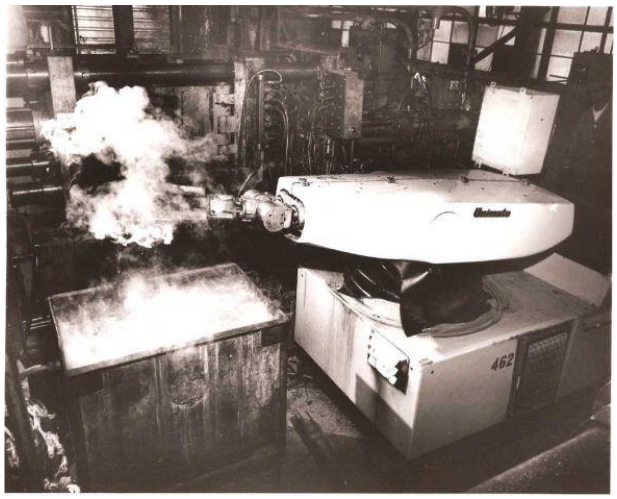


FIGURA 4 - ROBOT UNIMATE TRABAJANDO CON UNA PIEZA DE METAL SALIDA DE LA FORJA.

Últimamente se debate sobre el abuso que podemos llegar a realizar de los robots, muchos son los que se preguntan, si el desarrollo llega a niveles que permitan que un robot realice de una manera más segura, eficiente y barata una tarea, ¿para qué emplear a humanos para esas tareas? Algunos profesores y científicos ya han expresado su preocupación. El profesor Moshe Vardi de la Universidad de Rice en Texas ha declarado que podríamos llegar a niveles de paro del 50% en los próximos 30 años si en un futuro se emplean robots para actividades que ahora mismo realizan humanos. Este tipo de robots autónomos no necesitan operadores humanos en muchos casos, ya que una vez programados podrían realizar una tarea simple repetitivamente necesitando únicamente mantenimiento. Aun siendo autónomos también es cierto que para un grupo de robots suele haber al menos un técnico siempre presente para prevenir/solucionar fallos o realizar el mantenimiento de los mismos.

Un ejemplo de este tipo de robots y su funcionamiento son los que vemos por ejemplo en una planta de montaje de vehículos donde muchas tareas, desde la soldadura hasta el pintado son realizadas casi exclusivamente (a excepción de algunas marcas de lujo) por robots.

FIGURA 5 - FÁBRICA DE VEHÍCULOS DONDE LAS SOLDADURAS SON REALIZADAS POR BRAZOS ROBÓTICOS.

Otro aspecto en el que sí hay más unanimidad sobre su impacto positivo es que la robótica nos puede facilitar tareas que debido a problemas de escala o precisión pueden ser resueltos mejor con ayuda de un robot aunque en estos casos, no sean autónomos al 100%. Es el caso de robots utilizados en el campo médico como es el da Vinci, un robot compuesto por dos componentes donde el controlador es utilizado a modo de "paleta" proporcionando a un cirujano la posibilidad de realizar operaciones a través de incisiones muchísimo más pequeñas que las que serían necesarias utilizando métodos tradicionales, estos terminales también permiten que dos cirujanos trabajen juntos, lo que supone una gran ventaja. Proporciona además al cirujano la posibilidad de utilizar hasta 4 brazos con diferentes herramientas o para sujetar tejidos y la gran ventaja de tener una cámara dentro del paciente que puede observar mientras maneja los brazos del robot. En una laparoscopia estándar el cirujano debe manejar herramientas largas, perdiendo así fuerza y precisión mientras debe levantar la vista continuamente para observar a través de un monitor la imagen de lo que está haciendo en cada momento.

Es indudable que en este aspecto la robótica puede ser un gran aliado y proporcionar herramientas para facilitar y mejorar el trabajo de los médicos.

FIGURA 6 - DOS CIRUJANOS UTILIZAN TERMINALES PARA TRABAJAR JUNTOS CON EL ROBOT DA VINCI.

FIGURA 7 - BRAZOS DEL ROBOT DA VINCI.

1.2 El protocolo USB

Breve historia del protocolo USB

El protocolo USB nació en 1994 aunque no fue lanzado al mercado hasta el año 1996 y fue diseñado con la intención de proporcionar un puerto que permitiese conectar diferentes dispositivos hardware a ordenadores utilizando una interfaz común. Su principal objetivo era el de ser un puerto universal que se convirtiera en un estándar entre los fabricantes y que a la vez fuera capaz de proporcionar no sólo una interfaz de comunicación sino poder suministrar también voltaje a los dispositivos conectados de manera que se eliminase la dependencia de los cargadores.

En la siguiente tabla podemos ver las diferentes versiones del protocolo USB que se han implementado a lo largo de los años y su velocidad de transferencia.

Versión	Velocidades soportadas	Fecha de lanzamiento
1.0	12 Mbps	Enero 1996
1.1	1.5 , 12 Mbps	Septiembre 1998
2.0	1.5, 12 , 480 Mbps	Abril 2000
3.0	1.5, 12, 480 Mbps y 4.8 Gbps	Agosto 2008

Como vemos existe una retro compatibilidad entre las diferentes versiones ya que cada una soporta el modo de velocidad ofrecido por la anterior.

El estándar define no sólo el protocolo de comunicación sino también la fisonomía del puerto y de los cables.

Actualmente existen 8 tipos de conectores.

Fuente: www.l-com.com

FIGURA 8 - CONECTORES USB

Limitaciones del protocolo USB

El protocolo USB aparte de su velocidad (según la versión) tiene otras dos limitaciones:

- El número máximo de dispositivos que se pueden conectar al bus.
- La longitud del cable entre el dispositivo y el puerto.

Actualmente se pueden conectar a un puerto USB hasta 127 dispositivos. Esto se debe a que en el protocolo USB la dirección del paquete está compuesta por 7 bits. En cuanto a la longitud del cable normalmente se establece una longitud máxima de 3 metros para dispositivos que utilizan el protocolo USB 1.0 y de 5 metros a partir de la versión USB 2.0

Las limitaciones en cuanto a la longitud del cable vienen impuestas por el propio diseño del protocolo y no por las características del cable, interferencias, etc. Se debe a que cuando se diseñó se llegó a la conclusión de que un dispositivo que iba a ser utilizado en un ordenador no iba a estar nunca más lejos de esos 3-5 metros.

La limitación en la longitud del cable se lleva a cabo mediante el uso de un delay máximo en la interfaz y no puede ser solucionada utilizando cables de mejor calidad o más aislados sino utilizando hubs o extensores usb. Esto se debe a que el delay de la señal se controla entre cada segmento del cable USB y no sobre el total.

El bus USB está limitado a 5 hubs por lo tanto utilizando cable de 5 metros en la práctica la longitud máxima para un cable USB será de 30 metros.

Alimentación a través de USB

El protocolo USB proporciona corriente a los dispositivos conectados de manera que éstos no necesiten de un cargador externo. Sin embargo esta corriente es de baja intensidad y por lo tanto no siempre se podrá alimentar cualquier tipo de dispositivo satisfactoriamente.

Un ejemplo son los discos duros externos de gran capacidad que aún a día de hoy siguen necesitando un cargador externo para funcionar o cables especiales que permiten conectar más de un puerto USB al dispositivo para tomar corriente de dos puertos.

FIGURA 9 - CABLE DOBLE ALIMENTACION

La corriente proporcionada por un Host USB viene definida por unidades de carga o unit loads y por cuántas de éstas es capaz de proporcionar. Con el paso del tiempo se ha ido incrementando la corriente proporcionada por el conector desde que apareciera el primer conector USB capaz de proporcionar 100mA de unidad de carga hasta la última versión USB 3.0 que ya es capaz de proporcionar una unidad de carga de 150mA.

Además del valor máximo de una unidad de carga también se define cuántas de éstas se pueden obtener del host. Las primeras versiones del protocolo hasta la 2.0 podían consumir 5 unidades de carga del host, teniendo en cuenta que cada unidad proporciona 100mA podemos consumir hasta 500mA. de un mismo host. En cuanto a la versión 3.0 además de proporcionar una unidad de carga mayor, de 150mA, también es capaz de proporcionar hasta 6 de ellas por lo que la corriente máxima que nos proporciona un host USB 3.0 es de 900mA.

Los dispositivos al conectar siempre requerirán una unidad de carga al estar en modo low-power siempre que se conecta un nuevo dispositivo y luego durante la configuración si son dispositivos que requieren más corriente, es decir, trabajan en modo high-power, el host les proporcionará hasta 500mA o 900mA en el caso de USB 3.0.

En el caso del controlador del Scorbot se trata de un dispositivo auto alimentado al disponer de su propia fuente de alimentación y por lo tanto no hace uso del puerto USB para alimentarse sino únicamente para transmitir datos por lo que hará uso de una única unidad de carga.

Configuración del dispositivo

Los dispositivos USB conectados al bus deben configurarse antes de realizar cualquier tipo de transferencia de datos.

El dispositivo dispone de una o más configuraciones de las que una es seleccionada al conectar. Esta configuración indicará requerimientos de corriente y ancho de banda, además cada configuración puede definir múltiples interfaces que a su vez definen una serie de endpoints.

Todas las interfaces que proporciona una configuración están disponibles una vez establecida la misma. Una vez configurado y seleccionado el interface con el que el driver va a trabajar éste dispondrá de una serie de endpoints sobre los que realizar las transferencias.

Los endpoints proporcionan acceso unidireccional a través de búfers con el dispositivo.

Toda la información relativa a las configuraciones se encuentra en los descriptores a los que podemos acceder a través de peticiones de tipo Get Descriptor al dispositivo.

Descriptores

- Descriptor de dispositivo: Todo dispositivo tiene un único device descriptor y éste almacena información como la versión del protocolo que utiliza, identificadores del producto y su fabricante, y el número de configuraciones de las que dispone.

- Descriptor de configuración: Contiene el número de interfaces de la configuración, si se soporta o no la funcionalidad de suspensión y requerimientos de energía.

- Descriptor de interfaz: Clase y subclase de la interfaz, donde se definen una serie de drivers no específicos que pueden ser utilizados por todos los dispositivos, por ejemplo un pen drive no requiere que el fabricante cree un driver ya que existe una clase que define el funcionamiento de todos los dispositivos de este tipo. Además de definir clases y subclases de control de dispositivos comunes también indicará el número de configuraciones alternativas para la interfaz y endpoints.

- Descriptor de endpoint: El descriptor del endpoint indicará la dirección, sentido de transferencia y tipo del mismo. Además nos proporciona otros datos de interés como el tamaño máximo de paquete y la frecuencia para el polling si es un dispositivo que funciona mediante interrupciones.

- Descriptor de string: Contiene información relativa al dispositivo en un formato legible.

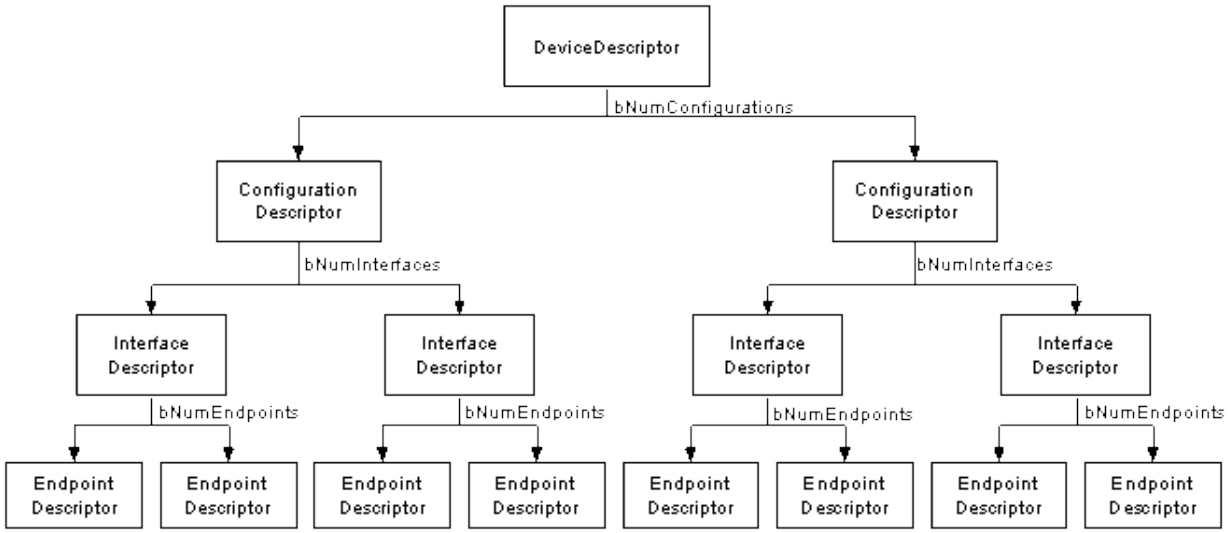


FIGURA 10 - DESCRIPTOR USB, FUENTE: HTTP://WWW.BEYONDLOGIC.ORG

Transferencia a través del bus USB

Las transferencias a través del bus USB se realizan enviando y recibiendo paquetes que podemos agrupar en 4 categorías.

1. **Token**: Indican al dispositivo USB de que el host desea comunicarse para realizar transferencia de lectura, escritura o control.

2. **Data**: Hay dos tipos de paquetes de datos que permiten transferencias de hasta 1024 bytes. Los tamaños máximos de datos en estos mensajes es de 8 bytes para dispositivos de baja velocidad y 1023/1024 para full y high speed respectivamente.

3. **Handshake**: Paquetes simples envían únicamente un identificador. Existen tres tipos ACK para indicar que se ha recibido correctamente, NAK para indicar que no es posible el envío o recepción de datos y STALL para indicar que se requiere la intervención del Host.

4. **Start of frame**: Consiste en un número de 11 bits envíado por el host para sincronizar con los dispositivos. Se envía cada 1ms. Aunque en los dispositivos High speed se divide en 8 microframes de 125 uS. Estos 8 microframes tendrán el mismo número de frame que aumenta cada microsegundo.

Los formatos de estos 4 tipos de mensajes son los siguientes:

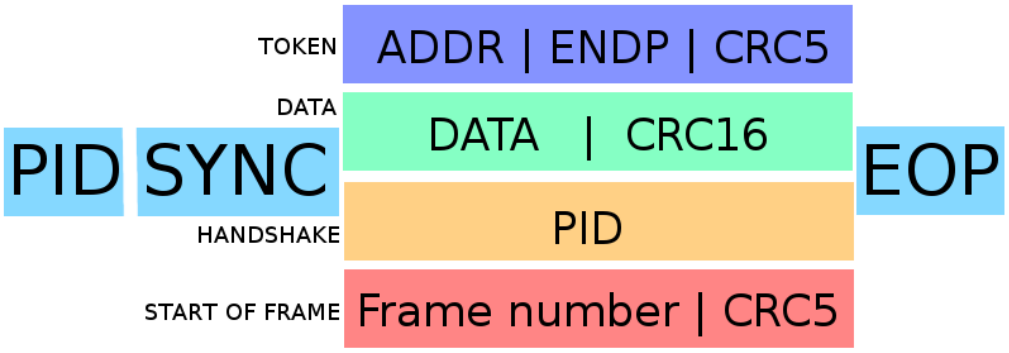


FIGURA 11 - FORMATOS MENSAJE USB

Como vemos todos los mensajes tienen en común los campos de sincronización (Sync), el identificador del paquete (PID) y el fin del mismo (EOP).

ENDPOINTS

Los endpoints actúan como búfers unidireccionales donde enviar y recibir paquetes que pueden ser de cualquiera de los 4 tipos vistos anteriormente. Estos paquetes se envían a través de dichos endpoints que pueden ser de entrada o salida y se mantienen en el búfer de manera que podamos leerlos en cuanto el software lo desee.

Por ejemplo, si se envía un paquete al endpoint 4 de un dispositivo ya configurado se quedará en el búfer de entrada del endpoint 4 del mismo hasta que éste lo lea y a su vez nos responderá a través del endpoint 4 pero en el búfer de salida donde se almacenará hasta ser leído.

PIPES

Los pipes se dividen en dos tipos y representan una conexión lógica para la transferencia de datos a nivel de software que además tiene asociada la configuración actual que se ha establecido entre el host y un endpoint.

Un pipe es en esencia un endpoint ya configurado.

- **Stream pipes** que no tienen formato USB definido pueden enviar/leer datos de los mismos aunque con una dirección predefinida y de forma secuencial. Soportan modos de transferencia bulk, asíncrono e interrupt.

- **Message pipes** tienen un formato definido y a diferencia de los stream pipes son controlados siempre por el host. Sólo soportan transferencias de tipo control.

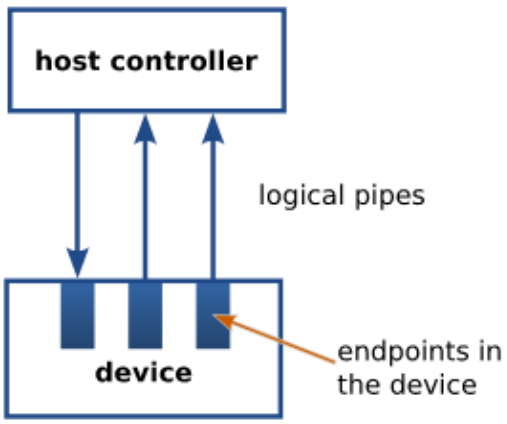


FIGURA 12 - PIPES Y ENDPOINTS, FUENTE: WIKIMEDIA COMMONS.

1.3 Estado del arte

El robot Scorbot-ER 4U nos proporciona por parte del fabricante INTELLITEK el driver original para controlar la comunicación entre el computador y el controlador del robot, además nos proporciona el programa Scorbase para operar con el robot.

El driver está definido por un archivo .inf que contiene la información referente al dispositivo.

No se nos proporciona acceso a las funciones del robot desde entornos externos a Scorbase de manera que estamos limitados a este programa para operar con el robot. Scorbase nos proporciona 3 niveles de funcionamiento, las funciones aparecen agrupadas por niveles complejidad.

También se nos permite el movimiento de cualquiera de los ejes a través de unos controles sencillos mediante una paleta de control digital.

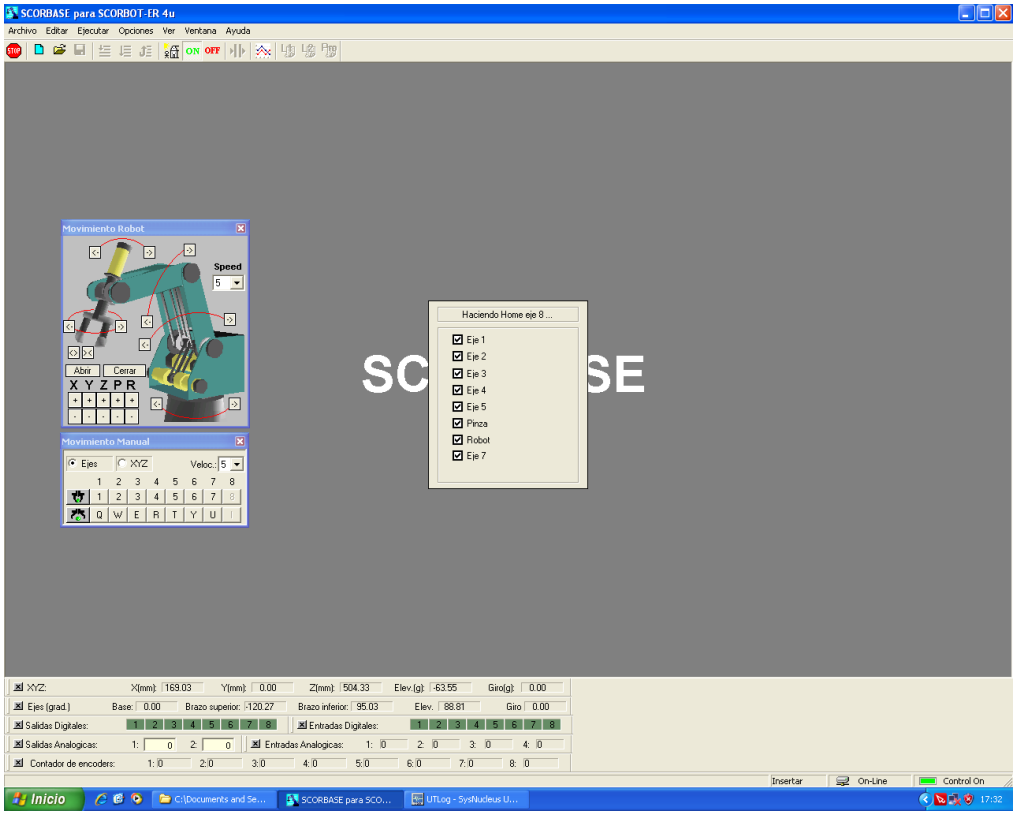


FIGURA 13 - CONTROLES E INFORMACIÓN EN SCORBASE

Finalmente, el aspecto más importante y por el cual se ha desarrollado este driver es que no podemos acceder a los motores del robot directamente, sino a través de las funciones Scorbase y por lo tanto nos limita a la hora de trabajar con el robot de manera directa o desarrollando cinemáticas propias (cambio del actuador, de algún elemento del brazo) si cambiáramos algún parámetro físico del robot.

1.4 Objetivos

Los objetivos del proyecto se han dividido en varias etapas:

ETAPA 1: Estudio y análisis detallado del protocolo del bus USB.

1. Estudio de la arquitectura del bus USB.

2. Evaluación y selección de herramientas para el análisis USB.

3. Instalación y puesta a punto de las herramientas, aprendizaje de las mismas y análisis de la información de interés en cada tipo de transferencia.

ETAPA 2: Análisis de las comunicaciones entre el robot y el ordenador.

4. Análisis de flujo de información entre el dispositivo robot y el ordenador con el fin de averiguar la estructura que presentan las transferencias.

5. Identificación de los diferentes tipos de comandos, primitivas de control y/o respuestas por parte del robot. Verificación de funcionamiento.

6. Desarrollo, depuración y puesta a punto de una librería lo suficientemente completa para poder operar con el robot.

7. Ensayo, depuración y verificación del correcto funcionamiento de las comunicaciones utilizando un driver alternativo al del fabricante.

ETAPA 3: Aplicaciones

8. En base a los resultados anteriores diseñar e implementar un interfaz gráfico que proporcione al usuario desde un ordenador la posibilidad de programar el robot y acceder a las diferentes funcionalidades del nuevo controlador.

2. Metodología

La metodología empleada ha sido la metodología de prototipado debido a la facilidad de dividir en funciones pequeñas nuestro proyecto y probar por separado los diferentes componentes que forman el software que vamos a desarrollar. Además nos proporcionaba una manera sencilla de saber en una fase todavía inicial del proyecto, si el mismo era factible, y por lo tanto se podía continuar con su desarrollo. Esto se puede ver en el ejemplo de la primera función desarrollada, nos ha permitido verificar que podíamos conectarnos, luego se ha ampliado a mantener la conexión abierta y finalmente en una última iteración incluso se ha encargado del movimiento en algunos casos.

2.1 Metodología de prototipado

La metodología empleada ha sido el prototipado debido a las siguientes características que presenta el proyecto:

- Múltiples funciones pequeñas que se pueden probar por separado.
- Independencia entre funciones, exceptuando la de conexión, permite probar de forma separada.
- Al tratarse de un driver no tenemos la posibilidad de simplemente fijar unos requisitos de usuario respecto al sistema que vamos a desarrollar más allá de permitir el uso del dispositivo.

Por lo tanto, una vez establecido un diseño global para todo el proyecto, seguiremos el mismo tras la fase de análisis con el objetivo de mantener la coherencia entre funciones, seguir unas pautas lógicas y de programación similares y por lo tanto lograr una integración mejor al preparar el sistema final.

Las fases a seguir para el desarrollo de cada función serán:

- Análisis de especificaciones del prototipo
- Diseño del prototipo
- Desarrollo del prototipo
- Refinamiento/optimización del prototipo desarrollado
- Verificación de funcionamiento para el prototipo

Cuando una función pase todas estas fases la consideraremos terminada y podrá ser añadida a la librería que compone el driver.

El siguiente diagrama de creación de prototipos muestra las fases y como se desarrolla el proceso iterativo que nos permite ir mejorando el prototipo hasta obtener un producto que satisfaga los requisitos definidos por nuestro análisis.

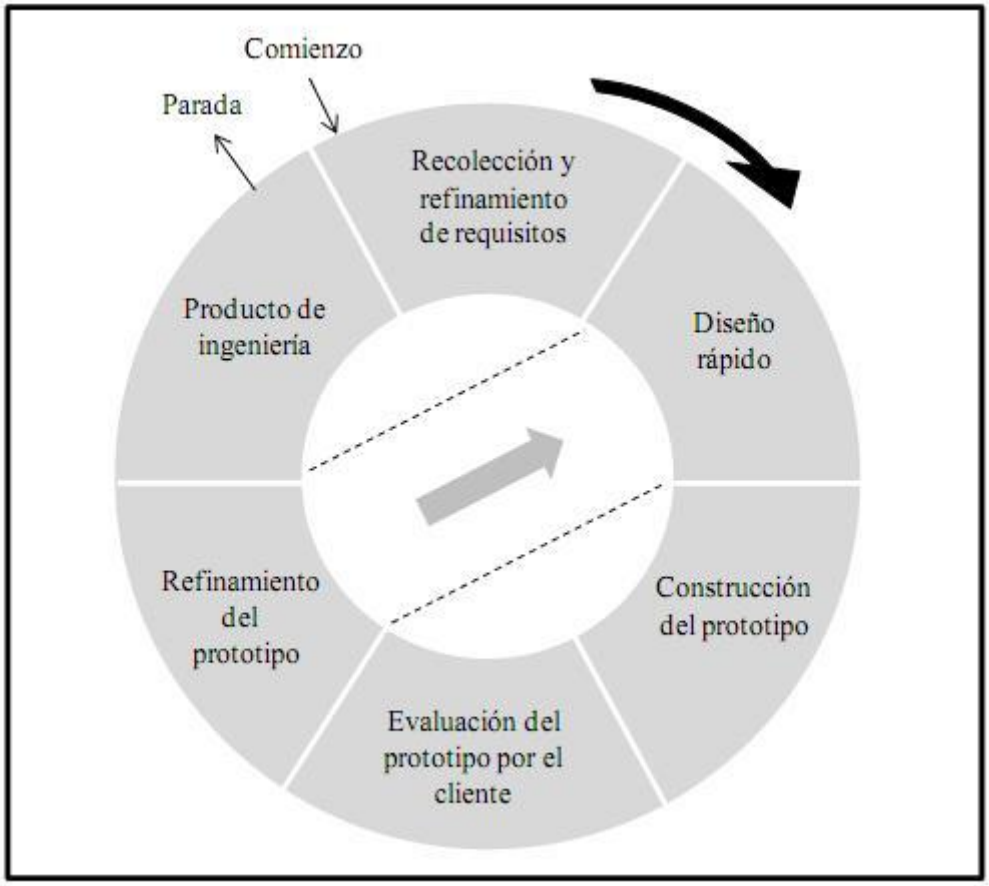


FIGURA 14 - DIAGRAMA DE CREACIÓN DE PROTOTIPOS

2.2 Recursos necesarios

Tanto para el desarrollo del driver como de la interfaz y el resto de aplicaciones han sido necesarios los siguientes recursos hardware y software:

Hardware

- Controlador robot, Scorbot-ER 4u Controller.
- Ordenador Personal Pentium IV con conexión a internet.
- Brazo robótico, Scorbot-ER 4u Arm.
- Cinta transportadora, Rhino Robots Inc.
- Sensor infrarrojo, Sensor Module Rhino Robots Inc.
- Fuente de tensión
- Voltímetro

Software

- Sistema operativo Windows XP
- Programa de captura de datos USB, USBTRACE
- DEV-CPP para programación
- Librería LibUSB.
- Netbeans 4.5 desarrollo interfaz web.
- Librería JQUERY UI
- Servidor Web APACHE.
- Google Drive
- Apache OpenOffice Writer 4.1.1
- GIMP 3.0

2.3 Temporalización

El proyecto se ha dividido en las siguientes fases:

FASE 1: Estudio y análisis detallado del protocolo del bus USB.

- Estudio de la arquitectura del bus USB.
- Evaluación y selección de herramientas de análisis para el bus USB.
- Instalación y puesta a punto de las herramientas: Aprender su uso y a analizar e identificar la información de interés para el estudio de las transferencias.

90 HORAS

FASE 2: Aplicación de técnicas de ingeniería inversa en el estudio y análisis de las comunicaciones entre el robot y el computador.

- Análisis del flujo de información entre robot y computador. Averiguar la estructura de las transferencias.
- Identificación de los diferentes tipos de comandos y/o respuestas del robot.
- Desarrollo, depuración y puesta a punto de una librería lo suficientemente completa para permitir el uso del robot.
- Ensayo, depuración y verificación del correcto funcionamiento de las comunicaciones con un driver alternativo al del fabricante.

350 HORAS

FASE 3: Aplicación: Diseño de un interfaz gráfico para la programación del robot.

- Análisis de requisitos.
- Diseño de la interfaz de usuario.
- Implementación.
- Experimentos, resultados y evaluación del sistema.

250 HORAS

FASE 4: Elaboración de la documentación y presentación del proyecto.

150 HORAS

TOTAL: 840 HORAS

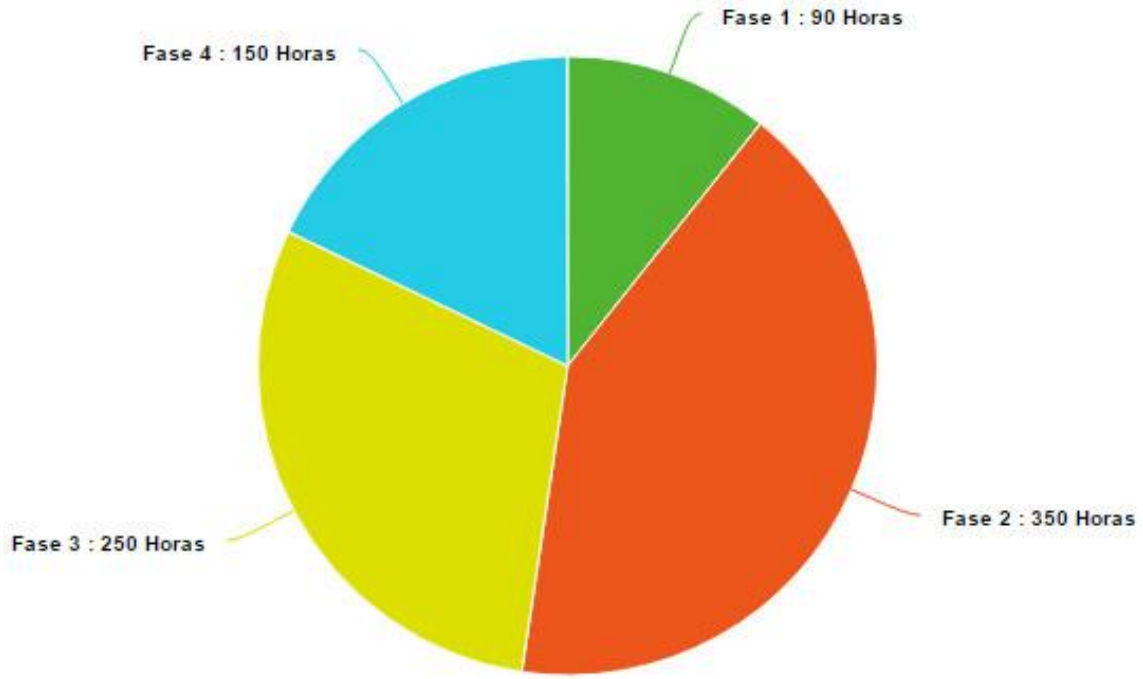


FIGURA 15 - DIVISION TEMPORAL DEL PROYECTO

El tiempo dedicado al proyecto en total ha sido de unas 840 horas y podemos ver que gran parte del mismo ha sido empleado en las dos fases más costosas del proyecto, el análisis y el desarrollo.

Resumen

Se ha establecido que la metodología a utilizar va a ser la de prototipado debido a que es la que más posibilidades nos da de verificar de forma rápida y eficaz si una función en la que estamos trabajando es capaz de cumplir los requisitos que exigimos de ella, que a su vez vienen basados en el análisis de las funciones que el propio Scorbase nos proporciona.

Además se ha verificado que disponemos de todos los recursos y realizado una temporalización del proyecto para dividirlo en 4 fases bien diferenciadas.

3. Análisis

Durante la fase de análisis vamos a estudiar la comunicación con el controlador del robot que realiza la aplicación Scorbase. Como requisitos previos vamos a repasar el uso de la librería USB que hemos elegido para comunicarnos con el controlador robot, LibUsb, y además ver sus diferentes modos de transferencia.

Una vez conectados al robot comienza el estudio exhaustivo de todos los mensajes que sé envían Scorbase y el controlador para lo cual haremos uso de una aplicación de captura de datos transferidos a través de los puertos USB del computador.

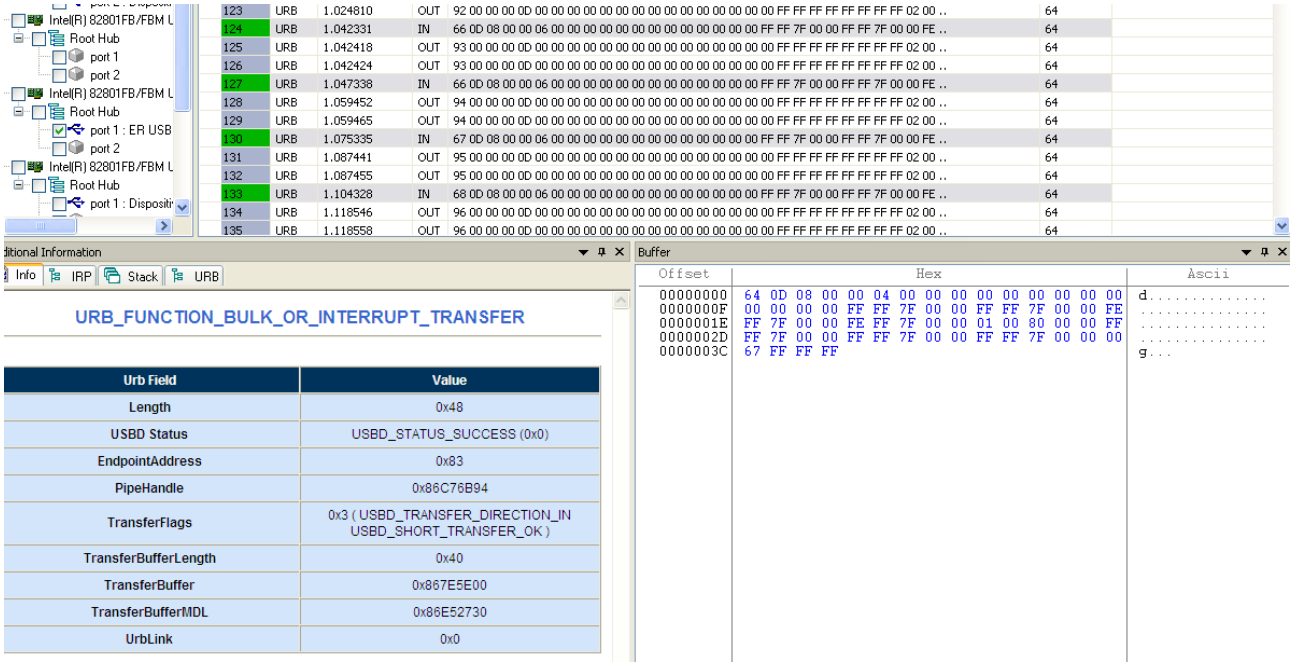


FIGURA 16 - CAPTURA DE DATOS USB

Como vemos el programa nos muestra una lista de dispositivos, del que seleccionamos el controlador y una vez iniciada la captura disponemos de una lista de mensajes de los cuales podemos ver tanto los detalles como el contenido del búfer.

Una vez finalizada esta fase de análisis y estudio de los mensajes podremos encarar el desarrollo de nuestras funciones a fin de conseguir operar con el robot satisfactoriamente.

3.1 Uso de la librería LibUsb

A continuación se explica el funcionamiento de la librería LibUsb y cómo hacemos uso de la misma para interactuar con el dispositivo USB.

Se requiere haber instalado previamente la librería y configurado el filtro (detalles en anexo). Una vez realizados estos dos sencillos pasos ya podemos acceder a las funciones que nos proporciona esta librería, para lo que únicamente tenemos que incluir en nuestro proyecto el fichero cabecera **lusb0_usb.h** y al compilar enlazar con la librería **/lib/gcc/libusb.a**

A continuación vamos a repasar los pasos para conectar con un dispositivo usb ya filtrado a través del siguiente ejemplo:

```
//Puntero a estructura de buses.
struct usb_bus *busses;
//Inicializamos la librería.
usb_init();
//Buscamos los buses del sistema.
usb_find_busses();
//Se buscan los dispositivos sobre los buses del sistema.
usb_find_devices();
//Cargo los resultados en una estructura.
busses = usb_get_busses();
//Estructuras para el manejo de los dispositivos encontrados.
struct usb_bus *bus;
struct usb_device *dev;
//Abrimos el primer dispositivo, será siempre el nuestro ya que sólo estamos filtrando este dispositivo.
bus=busses;
dev=bus->devices;
//Puntero al dispositivo y lo abrimos.
struct usb_dev_handle *handle;
handle=usb_open(dev);
//Aplicamos la configuración al dispositivo.
if(usb_set_configuration(handle,1)>0){
        printf("Error de configuración.");
        return -1;
}
//Reclamamos la interfaz para poder utilizarla.
if(usb_claim_interface(handle,0)>0){
        printf("Error de interfaz.");
        return -1;
}
//En esta zona trabajaríamos con el dispositivo
//Liberamos la interfaz.
usb_release_interface(handle,0)
//Cerramos la conexión usb.
usb_close();
```

3.2 Diagrama de conexión al dispositivo

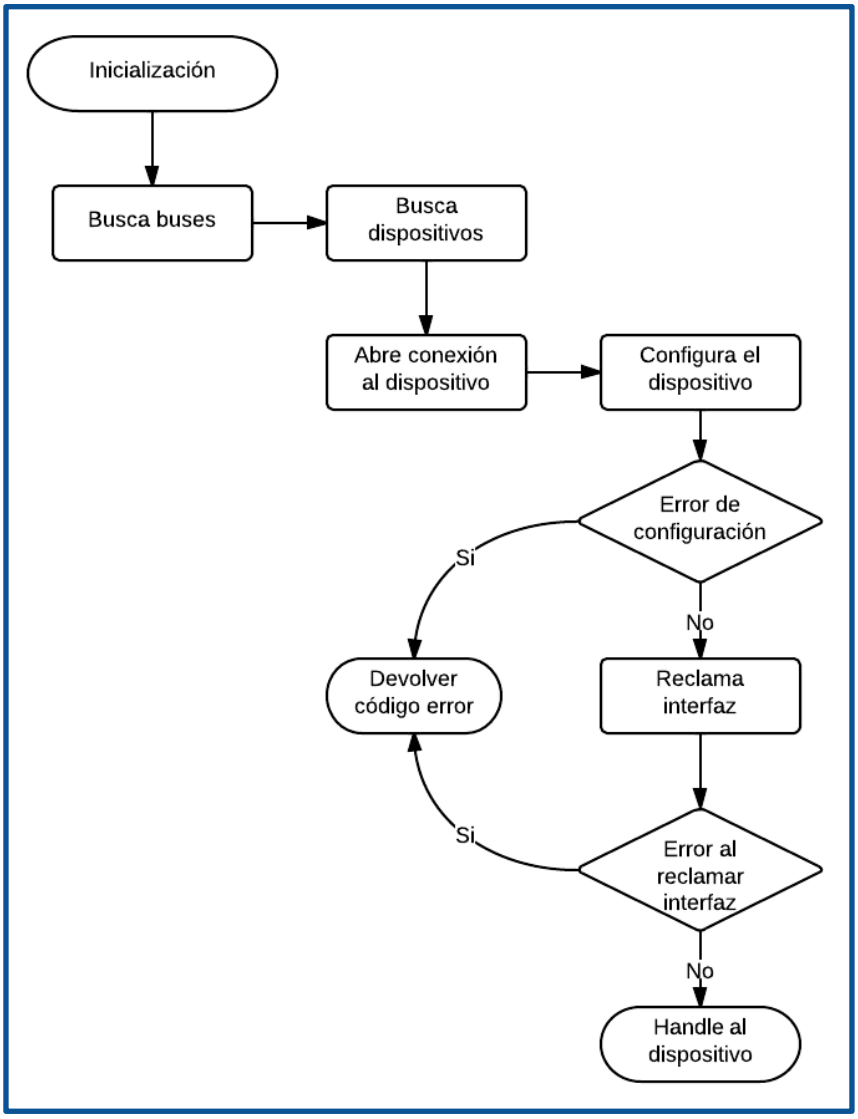


FIGURA 17 - DIAGRAMA CONEXIÓN

Podemos apreciar como durante los dos últimos pasos nos aseguramos de que configuramos y tomamos correctamente el control de la interfaz.

No debemos confundir estas funciones de conexión de LibUSB con las funciones de conexión al controlador propiamente dicho. Estas nos proporcionan un handle al dispositivo a bajo nivel y nuestra librería luego lo utilizará para comunicarse y efectivamente enviar los comandos tanto de conexión como de sincronización, así como los utilizados para dar órdenes al controlador.

3.3 Comunicación con LibUsb

'LibUsb' nos proporciona varias funciones para una vez establecida la conexión con el dispositivo poder realizar transferencias de datos.

Las funciones para el envío y recepción de mensajes se distinguen por el modo de transferencia: Interrupt o Bulk. Vamos a repasar sus características brevemente.

Modo Bulk

Las transferencias en modo bulk son utilizadas cuando los datos que se van a transmitir son intensivos y la latencia no es un factor determinante.

Un ejemplo de transferencia Bulk es la transferencia de archivos a una memoria USB. Se trata de un modo de transferencia que no garantiza la latencia aunque sí la transferencia de los datos a través de la detección de errores y el reenvío que garantiza la entrega.

La detección de errores se lleva a cabo a través de la inclusión de un campo CRC16. El algoritmo de detección de errores mediante redundancia cíclica tendrá por lo tanto un polinomio de 17 bits. Concretamente en las transferencia USB se utiliza el protocolo CRC-16-IBM.

Este tipo de transferencia está limitada a dispositivos de velocidades full-speed y high-speed pudiendo enviar paquetes de datos con un tamaño máximo de 64 y 512 bytes respectivamente.

Modo Interrupt

A diferencia de otros dispositivos donde son estos los que generan la interrupción, en el protocolo USB este tipo de transferencia requiere que el host realice un poll sobre el dispositivo y que el mismo indique que va a transmitir datos.

Este tipo de transmisión es la utilizada por dispositivos como teclados y ratones y la comunicación se produce a través de un pipe unidireccional de manera periódica.

El tamaño máximo de un paquete varía según la velocidad del dispositivo conectado y son las siguientes: 8 bytes para un dispositivo low-speed, 64 para un dispositivo full-speed y 1024 para un dispositivo high-speed.

Funciones proporcionadas por libUSB

A continuación repasamos las funciones que nos proporciona libUSB a la hora de realizar estos dos tipos de transferencia. Las dos funciones disponibles nos proporcionan la capacidad de realizar lecturas y escrituras en cada uno de los modos y todas requieren los mismos parámetros.

Modo Bulk

```
int usb_bulk_write(usb_dev_handle *dev, int ep, char *bytes, int size, int timeout);
int usb_bulk_read(usb_dev_handle *dev, int ep, char *bytes, int size, int timeout);
```

Modo interrupt

```
int usb_interrupt_write(usb_dev_handle *dev, int ep, char *bytes, int size, int timeout);
int usb_interrupt_read(usb_dev_handle *dev, int ep, char *bytes, int size, int timeout);
```

Los parámetros para llamar a estas funciones son los siguientes:

- **usb_dev_handle *dev:** puntero al dispositivo.
- **int ep:** endpoint al que va destinado la transferencia.
- **char *bytes:** vector de bytes que se van a escribir o sobre los que volcar una lectura.
- **int size:** número de bytes a enviar/leer.
- **int timeout:** timeout en milisegundos de la operación.

El valor de retorno es un entero con el número de bytes leídos en caso de éxito y < 0 en caso de error.

En el caso de nuestra librería, para la comunicación con el controlador del Scorbot vamos a utilizar el modo bulk.

Las variables y parámetros que se utilizarán serán:

```
char temp[64]={0};
int bytesread = usb_bulk_read(handle, 0x83, temp, 64, 1000);
```

Donde el parámetro temp está definido como un vector de 64 bytes inicializados cero y bytesread es nuestra variable con el valor de retorno de la indicando bytes leídos.

Conexión y sincronización con el controlador

El primer paso antes de empezar a desarrollar las funciones que van a componer nuestra librería para el robot Scorbot-ER 4u es conectarnos al mismo. Para ello vamos a capturar la secuencia de comandos que envía Scorbase para conectarse al robot, en este caso no nos importa analizar exhaustivamente qué significa cada mensaje de los que se envían siempre y cuando podamos conectarnos y mantener la conexión abierta.

Una captura durante el proceso de conexión nos indica que para conectarse al robot Scorbase intercambia con el mismo una serie de mensajes. Por lo tanto y ya que el análisis de los mismos no nos va a ayudar a la hora de desarrollar nuestras funciones vamos a simplemente reproducirlos al conectar nosotros con el fin de que el controlador logre conectar con nuestro programa satisfactoriamente.

Una vez enviada la misma secuencia que envía Scorbase hemos conseguido conectar con el controlador, sin embargo nos encontramos con un problema ya que cada vez que establecemos la conexión y estamos en estado ocioso esperando para dar alguna orden el controlador se desconecta. Este error también nos lo comunica a través del cambio de color del led de estado que pasa de verde a naranja.

Aunque Scorbase esté en estado ocioso siempre existe un intercambio de mensajes:

Seq	Type	Time	I/O	Buffer Snippet
862	URB	6.608649	OUT	20 00 00 00 0D 00 00 00 00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00 01 00 00 00 FF FF ..
863	URB	6.608668	OUT	20 00 00 00 0D 00 00 00 00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00 01 00 00 00 FF FF ..
864	URB	6.624518	IN	F3 0D 08 00 00 01 00 00 00 00 00 00 00 00 00 00 00 01 00 80 00 00 00 00 80 00 00 FF ..
865	URB	6.637911	OUT	21 00 00 00 0D 00 00 00 00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00 01 00 00 00 FF FF ..
866	URB	6.637924	OUT	21 00 00 00 0D 00 00 00 00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00 01 00 00 00 FF FF ..
867	URB	6.655511	IN	F4 0D 08 00 00 01 00 00 00 00 00 00 00 00 00 00 00 01 00 80 00 00 00 00 80 00 00 FF ..
868	URB	6.669161	OUT	22 00 00 00 0D 00 00 00 00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00 01 00 00 00 FF FF ..
869	URB	6.669176	OUT	22 00 00 00 0D 00 00 00 00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00 01 00 00 00 FF FF ..
870	URB	6.686506	IN	F5 0D 08 00 00 01 00 00 00 00 00 00 00 00 00 00 00 01 00 80 00 00 00 00 80 00 00 FF ..
871	URB	6.700411	OUT	23 00 00 00 0D 00 00 00 00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00 01 00 00 00 FF FF ..
872	URB	6.700422	OUT	23 00 00 00 0D 00 00 00 00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00 01 00 00 00 FF FF ..
873	URB	6.717504	IN	F6 0D 08 00 00 01 00 00 00 00 00 00 00 00 00 00 00 01 00 80 00 00 00 00 80 00 00 FF ..
874	URB	6.731660	OUT	24 00 00 00 0D 00 00 00 00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00 01 00 00 00 FF FF ..
875	URB	6.731669	OUT	24 00 00 00 0D 00 00 00 00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00 01 00 00 00 FF FF ..
876	URB	6.748505	IN	F8 0D 08 00 00 01 00 00 00 00 00 00 00 00 00 00 00 01 00 80 00 00 00 00 80 00 00 FF ..
877	URB	6.748589	OUT	25 00 00 00 0D 00 00 00 00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00 01 00 00 00 FF FF ..
878	URB	6.748595	OUT	25 00 00 00 0D 00 00 00 00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00 01 00 00 00 FF FF ..

FIGURA 18 - TRÁFICO USB EN ESTADO OCIOSO

Vimos como antes se capturaron una serie de transmisiones tras hacer Home y con todas las entradas/salidas desactivadas para evitar que otros datos ocupasen el búfer haciéndonos más complicada la tarea de descubrir cómo se sincronizaban Scorbase y el robot.

Vimos que los mensajes que enviamos en estado ocioso tenían la siguiente estructura:

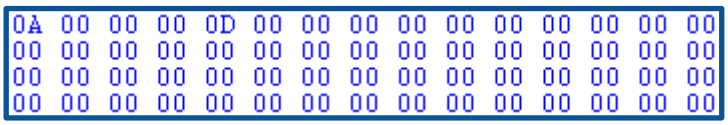


FIGURA 19 - ESTRUCTURA DE UN MENSAJE USB DE SALIDA

Se ha podido imitar el comportamiento de estos mensajes para no perder la conexión con el controlador creando un array de 64 bytes con la misma estructura que se ha utilizado como mensaje de salida al controlador. Tras realizar múltiples lecturas y estudiar anteriormente su estructura llegamos a la conclusión de que lo único que ocurre es que se aumenta en +1 el valor del primer byte con cada mensaje hasta el valor 0xFF donde se empieza otra vez de cero.

Ahora se nos presenta otro problema porque la entrada/salida con el controlador es bloqueante y no nos permite comunicarnos con el controlador de manera continua (necesario a la hora de enseñarle las posiciones) o esperar una entrada del usuario para un movimiento discreto ya que durante la lectura de la misma se perdería la conexión.

La solución es una vez establecida la conexión crear un hilo independiente que se encargue de estos mensajes de sincronización evitando así el bloqueo. Este hilo se encargará de ir aumentando el byte que sincroniza con el controlador y del movimiento continuo como veremos posteriormente.

Por ejemplo, en el caso de que queramos activar una salida digital la función debe comportarse de la siguiente manera:

Manipular el búfer en el hilo de sincronización

Se envía el mensaje

Devolver el búfer al estado ocioso

Esta solución ha sido probada y funciona satisfactoriamente manteniendo abierta la conexión y sin perder el control del robot.

El hilo de sincronización

Utilizando Windows el hilo se ha creado como una función y se ha lanzado como un hilo independiente utilizando la función createThread de la que disponemos en la librería windows.h.

```
HANDLE hThread;
printf("Lanzando HILO...");
DWORD a;
hThread=CreateThread(NULL,0,idleThread,handle,0,&a);
```

El tercer parámetro contiene el nombre de nuestra función, idleThread, de la cual detallaremos el funcionamiento a continuación. Vamos a dividir la función en dos partes para así ver cómo se encargará de tanto del problema de la sincronización con el controlador como del movimiento continuo para alguno de los ejes.

Nuestra primera necesidad y más importante que debe cubrir esta función es la de no perder la sincronización y por lo tanto el control del robot. Este primer objetivo tiene fácil solución, como veremos a continuación lo único que exige el controlador para mantener la conexión abierta es que enviemos los mensajes con “identificadores” en el primer byte que sean secuenciales lo que haremos será aumentar el valor del mismo tras cada escritura.

En cuanto al desbordamiento del byte al llegar al valor 0xFF se ha observado que vuelve a empezar de cero y no tiene ningún efecto sobre ningún otro parámetro.

Como vemos en el código de la misma es una función muy básica pero sin la cual la conexión al controlador se perdería continuamente.

```
DWORD WINAPI idleThread(LPVOID lpParam){
   array[4]=0x0d;
   while(1){
      usb_bulk_write((struct usb_dev_handle*)lpParam,0x02,array,64,1000);
      array[0]=array[0]+1;
      Sleep(Retardo);
      usb_bulk_read((struct usb_dev_handle*)lpParam, 0x83, temp, 64, 1000);
   }
};
```

Esta función aun así no soluciona todos los problemas ya que todavía no se encarga de mantener la conexión abierta durante el envío de comandos. Sin embargo lo que sí es capaz de hacer es mantener la conexión abierta y ya no nos desconectamos del controlador.

Movimiento continuo de los ejes

El movimiento a través de un mando, interfaz web o programa es algo que todo robot debe proporcionar al usuario ya que es la única manera eficiente de enseñar posiciones nuevas a un robot. Podríamos ir dando pequeños incrementos en los ejes y llegar a una posición final deseada pero esto sería poco práctico y de ahí que sea necesario el poder controlar el movimiento de los ejes de forma directa.

Al realizar un movimiento, por ejemplo, de 100 pasos en el eje 1 siempre sabemos de qué posición vamos a partir y a cual tenemos que llegar. Conocemos los valores de inicio y finalización, lo que no ocurre con los movimientos continuos donde únicamente conocemos la posición de partida.

Se ha decidido incluir el control directo de los ejes en la función del hilo independiente ya que siempre está activa para mantener la conexión abierta y por lo tanto nos permite actualizar el estado del movimiento con cada mensaje que se envía.

Sin entrar en los detalles del movimiento, que veremos más adelante, podemos ver la parte donde se controla este movimiento y que hace uso de dos variables globales para determinar si el usuario ha iniciado un movimiento continuo de un eje y en qué dirección.

A continuación vamos a ver las partes más relevantes de la función:

```
if(movContinuo!=0){
      codMotor = 12 + (movContinuo-1)*4;
      ...
      if(movContinuoDir > 0){
         posSalida += vel[velFactor];
         if(posSalida>65535){
            posSalida -= 65535;
            array[codMotor+2]=0;
            array[codMotor+3]=0;
         }
      }else{
         posSalida -= vel[velFactor];
         if(posSalida<0){
            posSalida += 65535;
            array[codMotor+2]=0xff;
            array[codMotor+3]=0xff;
         }
      }
...
```

Aquí se verá si la variable global movContinuo está activa y por lo tanto se debe mover el robot en este modo, además de ser así también se calcula el código del motor que se mueve y la dirección.

Una vez conocida tanto la dirección como el motor que se va a mover (en ocasiones dos como en el caso de roll e inclinación de la pinza) podemos enviar estos parámetros a través del búfer de salida al dispositivo:

```
//Colocamos nuestros valores de salida en el búfer.
    hexa = (char*)&posSalida;
    array[codMotor]=hexa[0];
    array[codMotor+1]=hexa[1];
    if(movContinuo > 3){
       hexa2 = (char*)&posSalida2;
       array[codMotor+4]=hexa2[0];
       array[codMotor+5]=hexa2[1];
    }
```

En el siguiente ciclo de escritura al dispositivo el búfer ya contendrá los nuevos valores, produciendo el movimiento.

Debido a que esta función es la que también mantiene la conexión abierta no hay necesidad de enviar los valores explícitamente una vez se encuentran en el búfer ya que esta función nunca deja de comunicarse con el controlador.

3.4 Estudio de los mensajes USB

Los mensajes USB que vamos a analizar tienen un campo de datos de 64 bytes, como nos indicó el descriptor del dispositivo y por lo tanto es en estos bytes donde tendremos que encontrar tanto los diferentes códigos de operación como los parámetros.

Para empezar a comprender cómo se organiza el contenido de estos mensajes y descifrar sus campos vamos a comenzar con el tipo de transferencia más simple posible realizando capturas de datos de la comunicación entre controlador y PC cuando el robot está conectado y se ha tomado control del mismo mediante la función control-on. En esta primera captura todavía no se habrá hecho Home.

A partir de ahora nos referiremos a los mensajes que van del PC al controlador como mensajes de salida y a los que provienen del controlador como mensajes de entrada.

3.5 Estudio de transferencia en estado ocioso

Una vez inicializado Scorbase y nuestra aplicación de captura USB vamos a ordenar a Scorbase tomar el control del robot y permanecer en estado ocioso. Si todo ha ocurrido normalmente se iluminarán en verde los dos leds de estado y conexión del controlador.

Realizamos una serie de capturas y extraemos el siguiente mensaje:

```
0A 00 00 00 0D 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

FIGURA 20 - MENSAJE DE SALIDA

Este mensaje de salida se repite de forma ininterrumpida durante toda la conexión con el controlador. Se puede apreciar como prácticamente todo el mensaje está compuesto por ceros y únicamente encontramos dos bytes con un valor diferente en el primer y quinto byte del mismo. Múltiples capturas han confirmado que éste es el mensaje de salida que empieza a enviar Scorbase una vez conectado y en estado ocioso.

Se han estudiado y analizado multitud mensajes y la conclusión es que se trata de algún tipo de mensaje de sincronización que simplemente mantiene la conexión abierta ya que el primer byte de la secuencia de datos aumenta con cada respuesta enviada por el controlador. Esta sincronización sólo se da cuando hemos tomado el control del robot y por lo tanto el no enviar estos mensajes supone la pérdida del control y el tener que realizar un nuevo control-on.

La siguiente parte del mensaje que nos interesa es donde encontramos el byte 0x0D que se mantiene sin cambios durante toda la conexión. A pesar de no saber su finalidad todavía se ha apreciado que siempre se da el mismo byte en dicha posición durante el estado ocioso del robot.

Hemos visto que el mensaje de salida no varía mucho en estado ocioso, a excepción del primer byte, vamos a analizar ahora un mensaje de entrada:

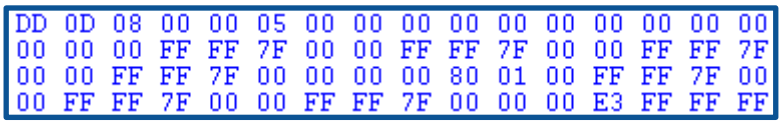


FIGURA 21 - MENSAJE DE ENTRADA

En este mensaje ocurre lo mismo con el primer byte, incrementa con cada respuesta al igual que pasaba con el primer byte del mensaje de salida, sin embargo, en este caso también ocurre con el sexagésimo primer byte del mensaje (0xE3). El resto de bytes no varían y su contenido de momento nos supone una incógnita.

Como vemos los primeros bytes tanto de salida como de entrada indican una secuencia numérica que se incrementa de uno en uno. Además podemos apreciar que este mismo byte también se encuentra al final de la secuencia de datos en las respuestas. Aparte de estos números tenemos una secuencia de bytes que parece ser la misma en todos los mensajes. Al no estar haciendo nada con el robot podemos suponer que es su funcionamiento normal.

¿ Cambiarán estos mensajes al hacer el Home ? Esta es la única manera que tenemos ahora mismo de comparar dos estados ociosos del robot sin intentar tocar nada más que pudiera complicarnos más descifrar los mensajes. Haremos el Home del robot llamando a su correspondiente función en Scorbase y volveremos a realizar una captura de los datos para comparar con el estado ocioso pre-Home que ya estudiamos.

Las capturas obtenidas han sido las siguientes:

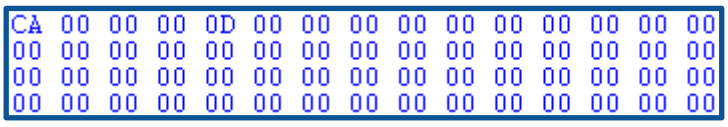


FIGURA 22 - MENSAJE DE SALIDA 2

Se puede ver que el mensaje de salida sigue conteniendo el valor 0x0D en el quinto byte y al igual que ocurría antes de hacer HOME sigue enviando la secuencia numérica en el primer byte. Esto nos supondrá una ventaja a la hora de empezar a trabajar con el robot y mantener una conexión al mismo debido a la simplicidad de estos mensajes, eso sí, no debemos olvidar que se trata de mantener abierta una conexión con el robot en estado ocioso por lo que aun siendo una parte importante para poder empezar a programar un controlador no es más que el principio y mantener una conexión abierta es lo mínimo que se puede pedir al driver.

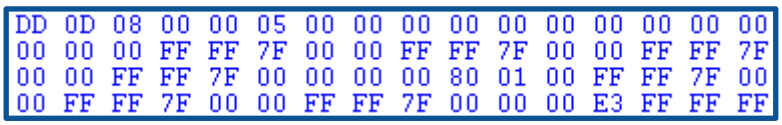


FIGURA 23 - MENSAJE DE ENTRADA 2

En el caso de la entrada desde el controlador se mantienen los valores incrementales que se mantienen al igual que en el caso de la captura previa a realizar Home. Los cambios se aprecian en los siguientes bytes donde vemos que han aparecido, aunque no alineados, un patrón recurrente de bytes, se trata de la secuencia FF FF F7. Aparece en 8 ocasiones, si pensamos en 8 campos o elementos de un robot que pudieran ser iguales tras un Home podemos pensar en los motores, el robot dispone de 6 y además el controlador permite añadir dos más a través de su controlador. Aunque es demasiado pronto para suponer que estos datos están asociados a la codificación de los motores se tendrá en cuenta para futuras pruebas cuando se siga profundizando en el estudio de los mensajes.

Resumen

En esta primera fase de estudio se ha descubierto la existencia de mensajes de sincronización entre el controlador y programa Scorbase con el fin de mantener la conexión abierta ya que solo dejan de transmitirse al desconectar el robot. Además se ha apreciado como en estado ocioso el quinto byte del mensaje mantiene el valor 0x0D tanto antes como después de realizar el HOME del robot. Finalmente hemos visto cómo unos valores que aparentemente no tenían sentido en el momento previo a realizar el Home del robot repiten una secuencia una vez se ha realizado el mismo y que se repite ocho veces por lo que podría estar asociada a los codificadores motores.

3.6 Estudio de transferencia de datos, E/S digital

Vamos a iniciar nuestro análisis del búfer una vez hemos visto cómo mantener la conexión al robot abierta. Utilizaremos una operación sencilla y que no haga uso de los motores ya que su codificación en los mensajes probablemente sea más compleja de lo que podría ser una operación que trabaje con las salidas y entradas digitales. Además de no involucrar el movimiento de motores este tipo de operaciones debe ser la más simple de analizar debido a la propia naturaleza de los valores que presumiblemente va a tomar el parámetro.

Salida Digital

Para analizar el comportamiento del controlador vamos a crear utilizando Scorbase un pequeño programa que se encargue de activar y desactivar de manera cíclica la salida digital 1 una vez cada segundo:

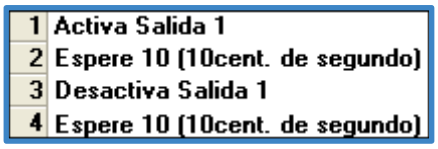


FIGURA 24 - PROGRAMA DE PRUEBA PARA LA SALIDA 1

Debemos tener en cuenta que la gran mayoría de los mensajes son de sincronización como vimos antes y debemos filtrar los paquetes interceptados que nos interesan. La captura se realizó en varias ocasiones con el fin de identificar el patrón de la función que activa y desactiva las salidas digitales.

Una vez analizado el tráfico se han extraído los siguientes mensajes:

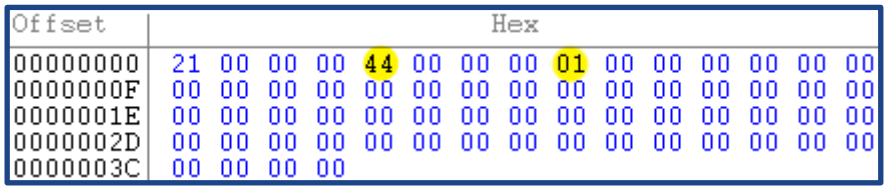
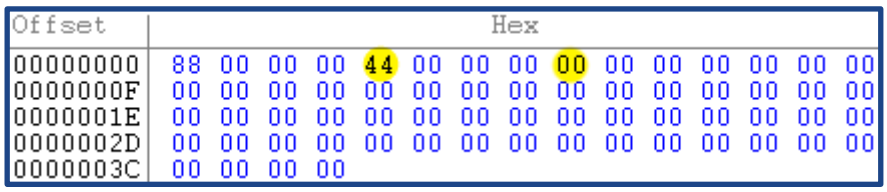


FIGURA 25 - ACTIVACIÓN DE LA SALIDA DIGITAL 1



Offset	Hex														
00000000	88	00	00	00	44	00	00	00	00	00	00	00	00	00	00
0000000F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000001E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000002D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000003C	00	00	00	00											

FIGURA 26 - DESACTIVACIÓN DE LA SALIDA DIGITAL 1

Analizando todas las muestras se ha decidido que debemos estudiar estos dos mensajes en los que parece que el byte 4 representa el código de operación ya que en todas las pruebas de activación de la salida han sido iguales (0x44) y difieren del que se envía continuamente cuando no estamos enviando órdenes al controlador (0x0D) y que en el byte 8, al ser el único que varía al activar/desactivar la salida podría encontrarse el parámetro. Sin embargo no podemos asegurar nada de esto sin haber comprobado con al menos una salida diferente ya que ese 1 podría no ser el parámetro del comando.

Ahora vamos a hacer lo mismo pero activando la salida 5. Vamos a ejecutar otro pequeño programa que active y desactive alternativamente la salida número 5 y una vez aislado el tráfico al igual que antes quedarnos con los que nos interesa, en esta ocasión filtrar los mensajes será más sencillo ya que conocemos el código de operación (0x44).

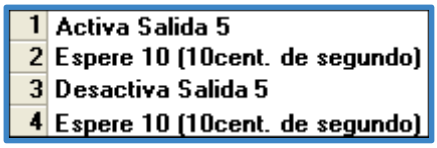
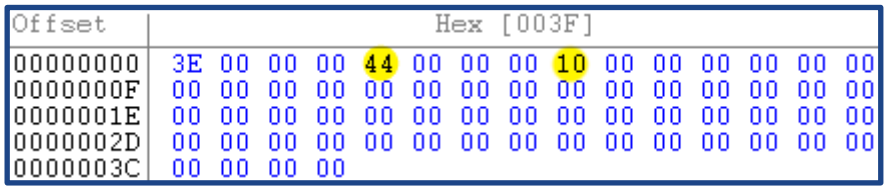


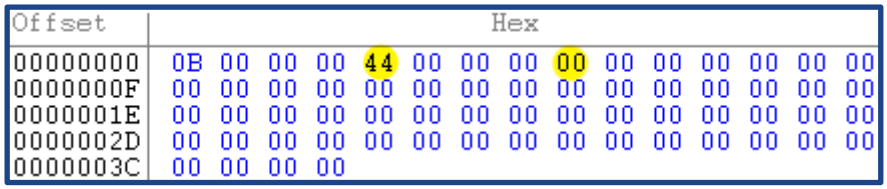
FIGURA 27 - PROGRAMA DE PRUEBA PARA LA SALIDA 2

Observamos que el valor del código de operación efectivamente no cambia pero que ahora el valor del byte donde sospechábamos que estaba el parámetro es otro:



Offset	Hex [003F]														
00000000	3E	00	00	00	44	00	00	00	10	00	00	00	00	00	00
0000000F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000001E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000002D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000003C	00	00	00	00											

FIGURA 28 - ACTIVACIÓN DE LA SALIDA DIGITAL 5



Offset	Hex														
00000000	0B	00	00	00	44	00	00	00	00	00	00	00	00	00	00
0000000F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000001E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000002D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000003C	00	00	00	00											

FIGURA 29 - DESACTIVACIÓN DE LA SALIDA DIGITAL 5

Esto parece indicar que efectivamente el parámetro se encuentra en el byte 8, ya que ha cambiado al cambiar la salida destino para el mismo, pero ¿ por qué si antes se introdujo un 1 y vimos que efectivamente en el byte había un 1 ahora vemos el valor 10 ? Sólo tiene sentido si interpretamos el byte que codifica las salidas digitales como un array de bits:

```
Entrada Digital : 1 2 3 4 5 6 7 8
  Array de bits : 0 0 0 0 0 0 0 0
```

Leyendo el valor hexadecimal 10 en binario tenemos el valor 10000, si le damos la vuelta a los bytes para verlo más fácilmente tenemos que la entrada es 00001, correspondiente a la entrada 5. Por lo tanto podríamos decir que ya conocemos el comportamiento de la función de activación/desactivación de salidas digitales y más adelante podremos implementarla en nuestra librería del controlador.

El método descrito anteriormente es el que se va a utilizar para todas las funciones proporcionadas por el controlador del Scorbot y se seguirá un método como el descrito anteriormente para la función de entradas digitales.

Entrada Digital

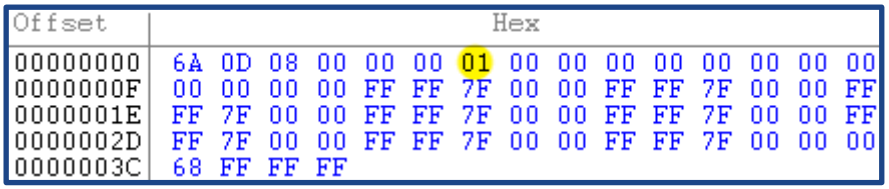
Ya sabemos cómo Scorbase activa y desactiva las salidas digitales así que para completar el estudio de la E/S Digital vamos a realizar una serie de capturas con el fin de analizar el estado del controlador mientras activamos y desactivamos entradas digitales manualmente para luego ver el impacto sobre el estado del controlador.

FIGURA 30 - PRUEBA ACTIVACIÓN LAS ENTRADAS DURANTE UNA CAPTURA DE DATOS

Durante la prueba se han realizado una serie de capturas donde se ha activado y desactivado las entradas digitales estudiando los cambios en los mensajes de entrada que nos envía el controlador.

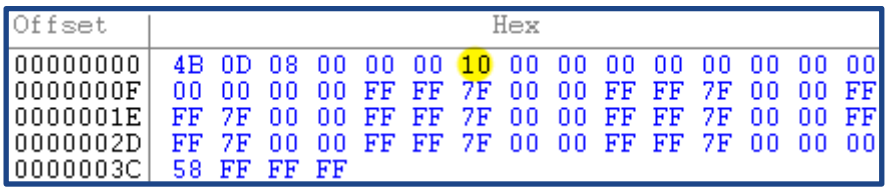
Hemos ido comparando los momentos en los que la salida estaba activa frente cuando estaba desactivada y puede ser que el byte 6 de la respuesta del controlador sea el que codifica el estado actual de la entrada digital. Al no haber entrada alguna el valor para dicha posición en el búfer de entrada permanece a cero y sólo varía al activar alguna de ellas.

Podemos ver las capturas donde activamos la entrada 1, luego la entrada 5 y finalmente ambas a la vez. Esto nos ayudará a comprender cómo se almacenan los valores de entrada en el búfer.



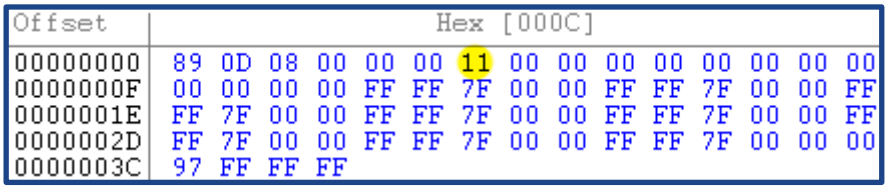
Offset	Hex
00000000	6A 0D 08 00 00 00 01 00 00 00 00 00 00 00 00
0000000F	00 00 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF
0000001E	FF 7F 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF
0000002D	FF 7F 00 00 FF FF 7F 00 00 FF FF 7F 00 00 00
0000003C	68 FF FF FF

FIGURA 31 - CAPTURA CON ENTRADA DIGITAL 1 ACTIVA



Offset	Hex
00000000	4B 0D 08 00 00 00 10 00 00 00 00 00 00 00 00
0000000F	00 00 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF
0000001E	FF 7F 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF
0000002D	FF 7F 00 00 FF FF 7F 00 00 FF FF 7F 00 00 00
0000003C	58 FF FF FF

FIGURA 32 - CAPTURA CON ENTRADA DIGITAL 5 ACTIVA



Offset	Hex [000C]
00000000	89 0D 08 00 00 00 11 00 00 00 00 00 00 00 00
0000000F	00 00 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF
0000001E	FF 7F 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF
0000002D	FF 7F 00 00 FF FF 7F 00 00 FF FF 7F 00 00 00
0000003C	97 FF FF FF

FIGURA 33 - CAPTURA CON ENTRADAS 1 Y 5 ACTIVAS

Esto se ha hecho para todas las entradas digitales y se ha visto que adopta los siguientes valores para cada una de las entradas al ser activadas individualmente:

Entrada	Valor hexadecimal	Valor entero	Valor Binario
1	0x01	1	00000001
2	0x02	2	00000010
3	0x04	4	00000100
4	0x08	8	00001000
5	0x10	16	00010000
6	0x20	32	00100000
7	0x40	64	01000000
8	0x80	128	10000000

Recordemos que hablamos de valores hexadecimales y que lo que nos interesa son los bits concretos que lo forman, de modo que tenemos una serie de bits que comienza a partir del byte 6 de un byte de largo que describe el estado actual de las entradas.

Por lo tanto y tras verificar la consistencia de las capturas podemos dar por hecho que la entrada digital se lee del estado del controlador y su formato es el mismo que el de la salida.

Resumen

La activación de salidas digitales se realiza a través del envío de un código de operación con valor 0x44 y un parámetro de un byte de largo que codifica el estado de la salida en los 8 bits que componen el octavo byte del búfer transferido en el mensaje de salida. Para las entradas digitales se ha descubierto que podemos acceder al estado de las mismas en cualquier momento a través de la lectura y decodificación de los bits que componen el séptimo byte del búfer de un mensaje de entrada.

3.7 Estudio de transferencia de datos en la E/S analógica

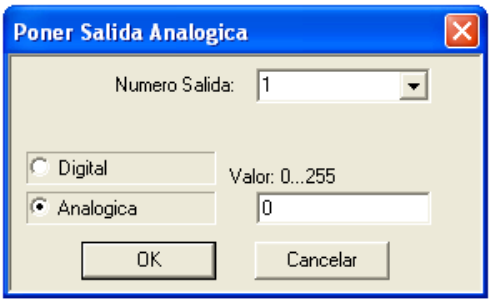
Al igual que hicimos para la entrada/salida digital se va a analizar ahora cómo se comporta la entrada/salida analógica tanto a la hora de realizar lecturas en cualquiera de sus 4 entradas como de asignar un voltaje a sus 2 puertos de salida.

Salida Analógica

El primer análisis lo vamos a hacer para el voltaje de salida ya que este siempre lo podremos asignar nosotros a través de Scorbase lo que nos permitirá realizar dos lecturas en los valores extremos (0-10 v.) y otra en un valor intermedio (5 v.) para intentar identificar el código de operación y el parámetro que envía Scorbase al controlador para que éste establezca la tensión de salida.

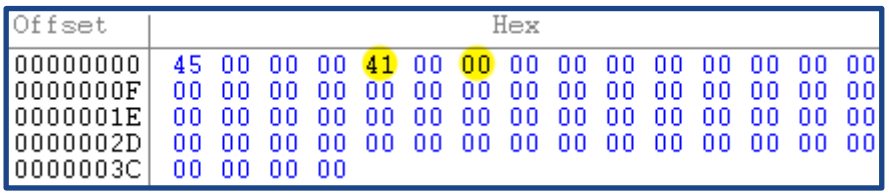
Realizamos una primera lectura tras asignar una salida de 0 a la salida analógica 1, las salidas analógicas ya se encuentran a cero al empezar a trabajar con el robot y no deberían haber cambiado pero nos interesa saber si el código de operación siempre es el mismo y encontrar el byte del parámetro por lo que haremos también esta primera lectura. Posteriormente activaremos la salida con el valor 127 y finalmente tras asignar una salida de 255.

Scorbase nos proporciona el comando y la posibilidad de elegir la salida a activar, por lo que hemos creado un pequeño programa con el fin de analizar los datos en el búfer para los diferentes valores de salida. Se encargará de ir cambiando el valor de la salida desde 0 hasta 255 pasando por el valor intermedio 127 con el fin de encontrar los mismos en el búfer al analizar el tráfico.



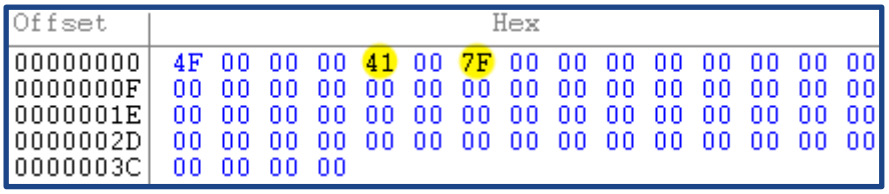
1	Poner Salidas Analogicas 1 a 0
2	Espere 10 (10cent. de segundo)
3	Poner Salidas Analogicas 1 a 127
4	Espere 10 (10cent. de segundo)
5	Poner Salidas Analogicas 1 a 255
6	Espere 10 (10cent. de segundo)

Los datos capturados y filtrados han sido los siguientes:



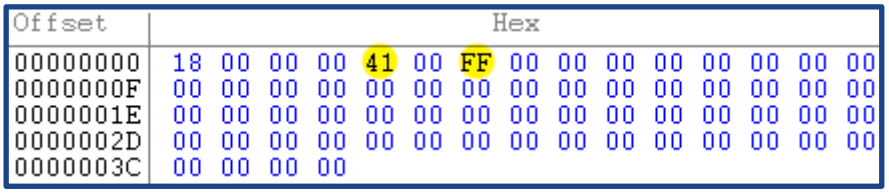
Offset	Hex
00000000	45 00 00 00 41 00 00 00 00 00 00 00 00 00 00 00
0000000F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001E	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000002D	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000003C	00 00 00 00

FIGURA 34 - SALIDA ANALÓGICA 1 A 0.



Offset	Hex
00000000	4F 00 00 00 41 00 7F 00 00 00 00 00 00 00 00 00
0000000F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001E	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000002D	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000003C	00 00 00 00

FIGURA 35 - SALIDA ANALÓGICA 1 A 127.



Offset	Hex
00000000	18 00 00 00 41 00 FF 00 00 00 00 00 00 00 00 00
0000000F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001E	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000002D	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000003C	00 00 00 00

FIGURA 36 - SALIDA ANALÓGICA 1 A 255.

En estos mensajes y tras verificar su repetitividad ejecutando el programa en modo cíclico podemos ver como hicimos antes que para la E/S digital el código de operación se encuentra en el mismo byte, el sexto, y que el código de operación es el 0x41. Es interesante apuntar que de momento parece que todos los códigos de operación se transmiten en el quinto byte.

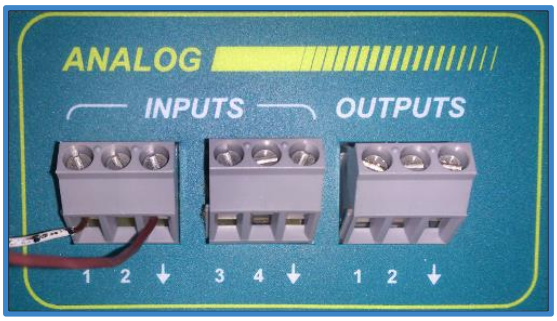
En cuanto a dónde envía Scorbase el parámetro si nos fijamos en las operaciones vemos que el único byte que varía al enviar el código de operación es el séptimo. Repasando los valores para donde creemos se encuentra codificada la salida vemos que para el valor de 0 es efectivamente 0x00, para el de 127 es 0x7F (127 en hexadecimal) y en el último para 255 es 0xFF (255 en hexadecimal) por lo tanto podemos afirmar que es este el byte que identifica el parámetro.

Podemos resumir el comando de salida analógica como un mensaje con el código de operación 0x41 y con un parámetro entre 0 y 255 que se envía en el sexto byte del mensaje.

Entrada Analógica

Para las entradas analógicas vamos a realizar una serie de pruebas en las que conectamos un voltaje a cada una de las mismas con el fin de identificar en qué parte del estado del búfer del controlador aparecen.

La primera prueba consistirá en conectar una fuente a 10 voltios a la entrada 1 con el fin de identificar dónde se encuentra el parámetro en el búfer a través de uno de sus valores límite, ya que el cero aparece muchas veces.



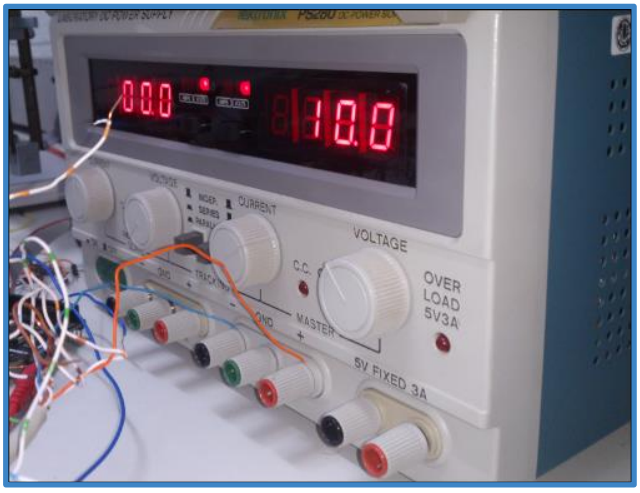


FOTO 1 - CONEXIÓN AL PUERTO 1 DE LA ENTRADA ANALÓGICA Y FUENTE PROBANDO VALOR EN EL EXTREMO DE LA ENTRADA ANALÓGICA.

Vamos a analizar el contenido del búfer y la información que nos proporciona Scorbase:

Offset	Hex [003F]														
00000000	25	0D	08	00	00	01	00	FE	00	00	00	00	00	00	00
0000000F	00	00	00	00	FF	FF	7F	00	00	FF	FF	7F	00	00	FF
0000001E	FF	7F	00	00	FF	FF	7F	00	00	FF	FF	7F	00	00	FF
0000002D	FF	7F	00	00	FF	FF	7F	00	00	FF	FF	7F	00	00	00
0000003C	21	FF	FF	FF											

FIGURA 37- ENTRADA ANALÓGICA 1 A 10 VOLTIOS.

Entradas Analogicas: 1: 254 2: 0 3: 0 4: 0

FIGURA 38 - VALOR DE ENTRADA INDICADO POR SCORBASE.

Una vez realizadas las lecturas vemos que para la entrada 1 parece que varía el byte 7 del búfer. Se ha confirmado realizando la prueba conectando y desconectando la entrada mientras se actualiza el búfer. Ahora vamos a bajar la tensión a 5 voltios y volveremos a realizar una lectura para ver si este valor cambia y lo hace indicando el valor de entrada que podríamos esperar para una entrada de 5 voltios.

Offset	Hex [0006]													
00000000	C6	0D	08	00	00	01	00	82	00	00	00	00	00	00
0000000F	00	00	00	00	FF	FF	7F	00	00	FF	FF	7F	00	00	FF
0000001E	FF	7F	00	00	FF	FF	7F	00	00	FF	FF	7F	00	00	FF
0000002D	FF	7F	00	00	FF	FF	7F	00	00	FF	FF	7F	00	00	00
0000003C	46	FF	FF	FF										

FIGURA 39 - ENTRADA ANALÓGICA 1 A 5 VOLTIOS.

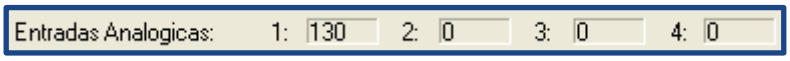


FIGURA 40 - VALOR DE ENTRADA INDICADO POR SCORBASE.

Vemos que de nuevo coincide el valor del búfer con el que nos indica Scorbase por lo que el byte 7 del búfer indica sin lugar a dudas el valor de entrada analógica 1.

Realizando los mismos pasos para las demás entradas vamos a ver si el controlador nos indica el número de la entrada a través de algún otro parámetro o valor o de qué otra manera codifica los valores para las demás entradas analógicas con el fin de distinguirlas. Para ello vamos a realizar una captura con la entrada de 10 voltios en las 3 que faltan por analizar.

Offset	Hex													
00000000	44	0D	08	00	00	01	00	00	FE	00	00	00	00	00
0000000F	00	00	00	00	FF	FF	7F	00	00	FF	FF	7F	00	00	FF
0000001E	FF	7F	00	00	FF	FF	7F	00	00	FF	FF	7F	00	00	FF
0000002D	FF	7F	00	00	FF	FF	7F	00	00	FF	FF	7F	00	00	00
0000003C	40	FF	FF	FF										

FIGURA 41 - ENTRADA 2 CONECTADA A 10 VOLTIOS.

Offset	Hex													
00000000	CA	0D	08	00	00	01	00	00	00	FE	00	00	00	00
0000000F	00	00	00	00	FF	FF	7F	00	00	FF	FF	7F	00	00	FF
0000001E	FF	7F	00	00	FF	FF	7F	00	00	FF	FF	7F	00	00	FF
0000002D	FF	7F	00	00	FF	FF	7F	00	00	FF	FF	7F	00	00	00
0000003C	C6	FF	FF	FF										

FIGURA 42 - ENTRADA 3 CONECTADA A 10 VOLTIOS.

Offset	Hex [003F]													
00000000	A1	0D	08	00	00	01	00	00	00	00	FE	00	00	00
0000000F	00	00	00	00	FF	FF	7F	00	00	FF	FF	7F	00	00	FF
0000001E	FF	7F	00	00	FF	FF	7F	00	00	FF	FF	7F	00	00	FF
0000002D	FF	7F	00	00	FF	FF	7F	00	00	FF	FF	7F	00	00	00
0000003C	9D	FF	FF	FF										

FIGURA 43 - ENTRADA 4 CONECTADA A 10 VOLTIOS.

Viendo como varía el búfer de entrada podemos extraer la conclusión de que las entradas se mapean a los siguientes bytes del búfer de estado:

Posición del byte	Entrada analógica
7	1
8	2
9	3
10	4

Hemos descifrado por lo tanto cómo realizar lecturas de los puertos de entrada analógicos del controlador y obtener el valor hexadecimal que representa la tensión de entrada.

Resumen

Hemos iniciado el estudio del Robot Scorbot-ER 4u centrándonos en el controlador y sus diferentes funcionalidades. Una vez estudiado el búfer y los códigos de operación enviados por Scorbase al controlador sabemos cómo interactuar con el mismo y podremos desarrollar las funciones necesarias para nuestro driver.

Ya podemos dar por finalizado el estudio de los mensajes relacionados con el controlador y empezar a estudiar todo lo relacionado con el brazo robótico.

3.8 Estudio de movimiento de motores

Vamos a estudiar ahora cómo se mueven los motores del brazo robótico y para ello vamos a comenzar por centrarnos en movimientos que hagan uso de un único motor, en este caso el de la cintura del brazo robótico. Se ha elegido la cintura ya que al mover el resto de la estructura los movimientos pequeños son más fácilmente perceptibles a simple vista y además tiene el rango de movimiento más amplio de todos los ejes, sin tener en cuenta claro el giro de la muñeca, aunque ésta se ha descartado a la hora de realizar este estudio debido a que su movimiento viene definido por dos motores lo que complicaría en este momento inicial el análisis de los mensajes.

Parece lógico que todos los motores vayan a tener un comportamiento similar y una vez estudiado el comportamiento de uno se verá si las conclusiones son extrapolables a todos los demás.

Para estudiar el movimiento de este motor se va a utilizar el control directo de los ejes en Scorbase que se nos proporciona a través de la ventana de movimiento manual.

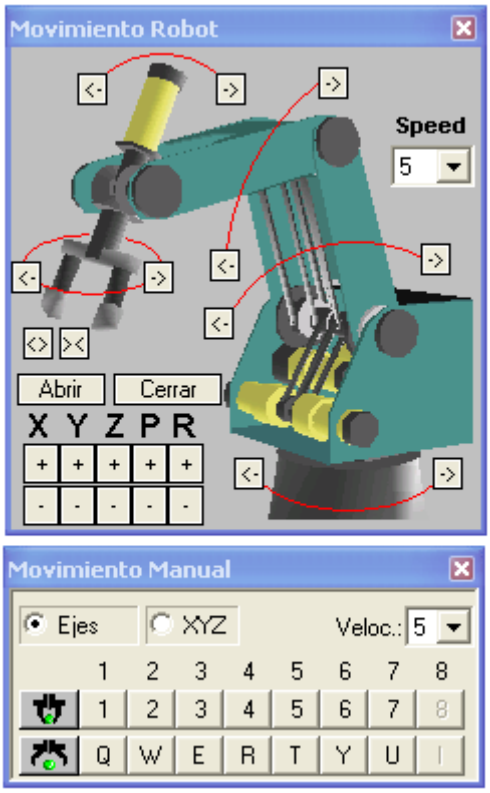


FIGURA 44 - VENTANAS DE MOVIMIENTO MANUAL DEL ROBOT.

Vamos a realizar pequeños movimientos en las dos direcciones y hasta los extremos. Al igual que antes se realizarán múltiples capturas y una vez identificados los mensajes que nos interesan se procederá descifrar que hacen y si son consistentes durante varias pruebas.

Tomaremos una primera captura tras hacer Home y nos moveremos hacia valores positivos del codificador. Se ha decidido realizar el primer movimiento hacia valores positivos ya que estos deberían ser más fáciles de analizar en los bytes que componen el estado del robot. Realizando un movimiento pequeño vemos que el estado del robot cambia y filtramos los mensajes enviados al controlador para descifrarlos y descubrir si se repiten en posteriores movimientos.

Se ha tomado una se las capturas realizadas como ejemplo una vez confirmado que el patrón que siguen todas las realizadas era idéntico.

OUT	ED 00 00 00 47 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	ED 00 00 00 47 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
IN	C0 0D 00 00 00 01 00 00 00 00 00 F2 00 00 00 00 00 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF ..
OUT	EE 00 00 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	EE 00 00 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
IN	ED 47 00 00 00 01 00 00 00 00 00 F2 00 00 00 00 00 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF ..
OUT	EF 00 00 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	EF 00 00 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
IN	ED 47 00 00 00 01 00 00 00 00 00 F2 00 00 00 00 00 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF ..
OUT	F0 00 00 00 0D 00 00 00 00 00 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	F0 00 00 00 0D 00 00 00 00 00 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
IN	ED 47 00 00 00 01 00 00 00 00 00 F2 00 00 00 00 00 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF ..
OUT	F1 00 00 00 0D 00 00 00 00 00 00 00 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	F1 00 00 00 0D 00 00 00 00 00 00 00 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
IN	ED 47 00 00 00 01 00 00 00 00 00 F2 00 00 00 00 00 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF ..
OUT	F2 00 00 00 0D 00 00 00 00 00 00 00 1A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	F2 00 00 00 0D 00 00 00 00 00 00 00 1A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
IN	ED 47 00 00 00 01 00 00 00 00 00 F2 00 00 00 00 00 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF ..
OUT	F3 00 00 00 0D 00 00 00 00 00 00 00 2B 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	F3 00 00 00 0D 00 00 00 00 00 00 00 2B 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
IN	EE 0D 00 00 00 01 00 00 00 00 00 F2 00 00 00 00 00 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF ..
OUT	F4 00 00 00 0D 00 00 00 00 00 00 00 3F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	F4 00 00 00 0D 00 00 00 00 00 00 00 3F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
IN	EE 0D 00 00 00 01 00 00 00 00 00 F2 00 00 00 00 00 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF ..
OUT	F5 00 00 00 0D 00 00 00 00 00 00 00 58 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	F5 00 00 00 0D 00 00 00 00 00 00 00 58 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
IN	F0 0D 00 00 00 01 00 00 00 00 00 F2 00 00 00 00 00 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF ..
OUT	F6 00 00 00 0D 00 00 00 00 00 00 00 74 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	F6 00 00 00 0D 00 00 00 00 00 00 00 74 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
IN	F1 0D 00 00 00 01 00 00 00 00 00 F2 00 00 00 00 00 00 00 00 80 00 00 FF FF 7F 00 00 FF ..
OUT	F7 00 00 00 0D 00 00 00 00 00 00 00 94 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	F7 00 00 00 0D 00 00 00 00 00 00 00 94 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..

CAPTURA 1 - INICIO MOVIMIENTO.

En el momento de iniciar el movimiento vemos que se envía un código diferente a 0x0D que hemos visto que es el enviado en estado ocioso. Se envía el código 0x47 y a partir de ese momento vemos como en la captura se empiezan a producir aumentos en el byte número 13.

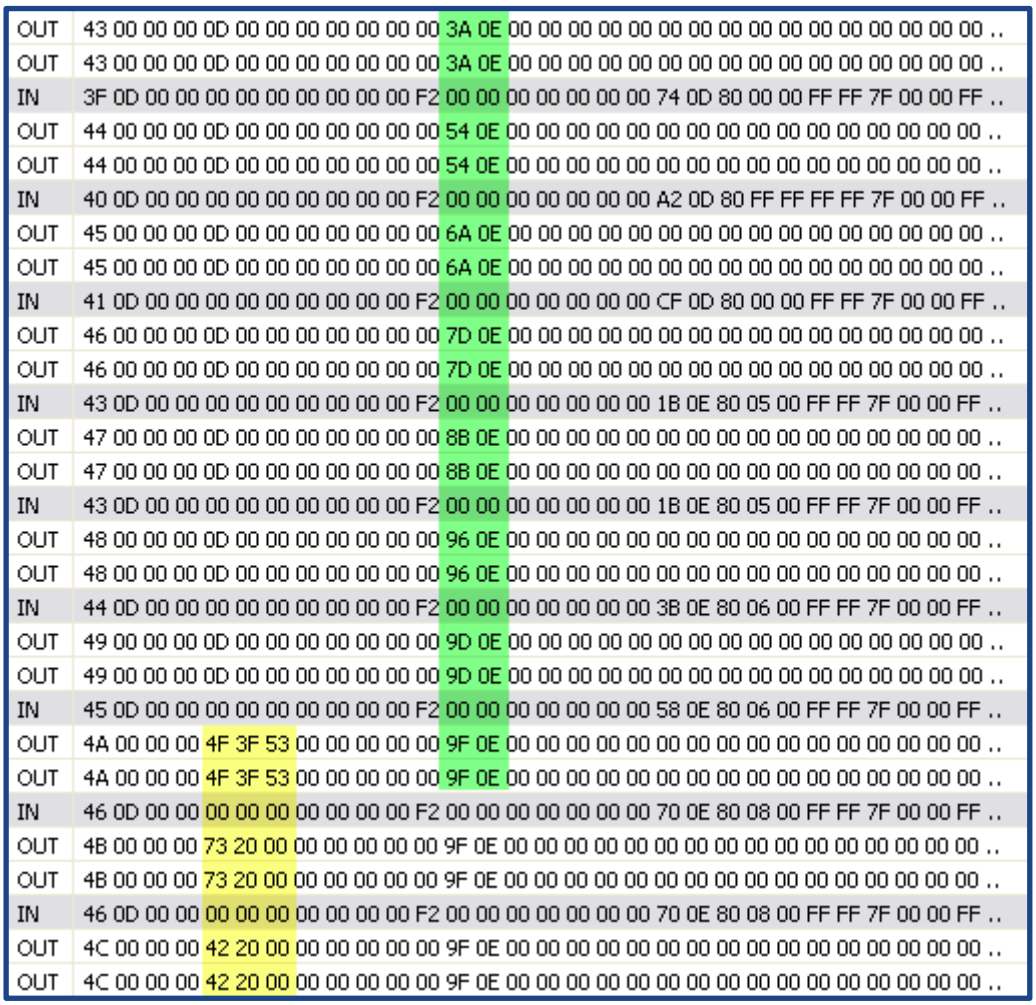
Analicemos ahora qué ocurre durante el transcurso del movimiento:

OUT	F7	00	00	00	0D	00	00	00	00	00	00	94	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..
OUT	F7	00	00	00	0D	00	00	00	00	00	00	94	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..
IN	F2	0D	00	00	00	01	00	00	00	00	00	F2	00	00	00	00	00	00	00	04	00	80	FD	FF	FF	FF	7F	00	00	FF	..
OUT	F8	00	00	00	0D	00	00	00	00	00	00	B8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..
OUT	F8	00	00	00	0D	00	00	00	00	00	00	B8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..
IN	F3	0D	00	00	00	01	00	00	00	00	00	F2	00	00	00	00	00	00	00	0E	00	80	F8	FF	FF	FF	7F	00	00	FF	..
OUT	F9	00	00	00	0D	00	00	00	00	00	00	E1	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..
OUT	F9	00	00	00	0D	00	00	00	00	00	00	E1	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..
IN	F5	0D	00	00	00	01	00	00	00	00	00	F2	00	00	00	00	00	00	00	2E	00	80	F6	FF	FF	FF	7F	00	00	FF	..
OUT	FA	00	00	00	0D	00	00	00	00	00	00	0D	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..
OUT	FA	00	00	00	0D	00	00	00	00	00	00	0D	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..
IN	F6	0D	00	00	00	00	00	00	00	00	00	F2	00	00	00	00	00	00	00	45	00	80	F4	FF	FF	FF	7F	00	00	FF	..
OUT	FB	00	00	00	0D	00	00	00	00	00	00	3B	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..
OUT	FB	00	00	00	0D	00	00	00	00	00	00	3B	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..
IN	F7	0D	00	00	00	00	00	00	00	00	00	F2	00	00	00	00	00	00	00	61	00	80	F1	FF	FF	FF	7F	00	00	FF	..
OUT	FC	00	00	00	0D	00	00	00	00	00	00	6A	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..
OUT	FC	00	00	00	0D	00	00	00	00	00	00	6A	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..
IN	F9	0D	00	00	00	00	00	00	00	00	00	F2	00	00	00	00	00	00	00	A5	00	80	F3	FF	FF	FF	7F	00	00	FF	..
OUT	FD	00	00	00	0D	00	00	00	00	00	00	99	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..
OUT	FD	00	00	00	0D	00	00	00	00	00	00	99	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..
IN	F9	0D	00	00	00	00	00	00	00	00	00	F2	00	00	00	00	00	00	00	A5	00	80	F3	FF	FF	FF	7F	00	00	FF	..
OUT	FE	00	00	00	0D	00	00	00	00	00	00	C8	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..
OUT	FE	00	00	00	0D	00	00	00	00	00	00	C8	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..
IN	FA	0D	00	00	00	00	00	00	00	00	00	F2	00	00	00	00	00	00	00	CC	00	80	F3	FF	FF	FF	7F	00	00	FF	..
OUT	FF	00	00	00	0D	00	00	00	00	00	00	F7	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..
OUT	FF	00	00	00	0D	00	00	00	00	00	00	F7	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..
IN	FB	0D	00	00	00	00	00	00	00	00	00	F2	00	00	00	00	00	00	00	F8	00	80	F1	FF	FF	FF	7F	00	00	FF	..
OUT	00	00	00	00	0D	00	00	00	00	00	00	25	02	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..
OUT	00	00	00	00	0D	00	00	00	00	00	00	25	02	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	..

CAPTURA 2 - MOVIMIENTO MOTOR

Durante el transcurso del movimiento vemos que sigue aumentando el valor que se encuentra en el byte 13, pero vemos que al aumentar pasado el valor 0xFF sigue aumentando pero en este caso usando el byte 14 como el más significativo. Aparte del aumento de ese valor no ocurre nada más que pudiera resultarnos de interés, no se envían más códigos de operación durante el movimiento.

Finalmente veamos que ocurre al finalizar el movimiento:



OUT	43 00 00 00 0D 00 00 00 00 00 00 00	3A 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	43 00 00 00 0D 00 00 00 00 00 00 00	3A 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
IN	3F 0D 00 00 00 00 00 00 00 00 00 00	F2 00 00 00 00 00 00 00 74 0D 80 00 00 FF FF 7F 00 00 FF ..
OUT	44 00 00 00 0D 00 00 00 00 00 00 00	54 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	44 00 00 00 0D 00 00 00 00 00 00 00	54 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
IN	40 0D 00 00 00 00 00 00 00 00 00 00	F2 00 00 00 00 00 00 00 A2 0D 80 FF FF FF FF 7F 00 00 FF ..
OUT	45 00 00 00 0D 00 00 00 00 00 00 00	6A 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	45 00 00 00 0D 00 00 00 00 00 00 00	6A 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
IN	41 0D 00 00 00 00 00 00 00 00 00 00	F2 00 00 00 00 00 00 00 CF 0D 80 00 00 FF FF 7F 00 00 FF ..
OUT	46 00 00 00 0D 00 00 00 00 00 00 00	7D 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	46 00 00 00 0D 00 00 00 00 00 00 00	7D 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
IN	43 0D 00 00 00 00 00 00 00 00 00 00	F2 00 00 00 00 00 00 00 1B 0E 80 05 00 FF FF 7F 00 00 FF ..
OUT	47 00 00 00 0D 00 00 00 00 00 00 00	8B 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	47 00 00 00 0D 00 00 00 00 00 00 00	8B 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
IN	43 0D 00 00 00 00 00 00 00 00 00 00	F2 00 00 00 00 00 00 00 1B 0E 80 05 00 FF FF 7F 00 00 FF ..
OUT	48 00 00 00 0D 00 00 00 00 00 00 00	96 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	48 00 00 00 0D 00 00 00 00 00 00 00	96 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
IN	44 0D 00 00 00 00 00 00 00 00 00 00	F2 00 00 00 00 00 00 00 3B 0E 80 06 00 FF FF 7F 00 00 FF ..
OUT	49 00 00 00 0D 00 00 00 00 00 00 00	9D 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	49 00 00 00 0D 00 00 00 00 00 00 00	9D 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
IN	45 0D 00 00 00 00 00 00 00 00 00 00	F2 00 00 00 00 00 00 00 58 0E 80 06 00 FF FF 7F 00 00 FF ..
OUT	4A 00 00 00 4F 3F 53 00 00 00 00 00	9F 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	4A 00 00 00 4F 3F 53 00 00 00 00 00	9F 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
IN	46 0D 00 00 00 00 00 00 00 00 00 00	F2 00 00 00 00 00 00 00 70 0E 80 08 00 FF FF 7F 00 00 FF ..
OUT	4B 00 00 00 73 20 00 00 00 00 00 00	9F 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	4B 00 00 00 73 20 00 00 00 00 00 00	9F 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
IN	46 0D 00 00 00 00 00 00 00 00 00 00	F2 00 00 00 00 00 00 00 70 0E 80 08 00 FF FF 7F 00 00 FF ..
OUT	4C 00 00 00 42 20 00 00 00 00 00 00	9F 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
OUT	4C 00 00 00 42 20 00 00 00 00 00 00	9F 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..

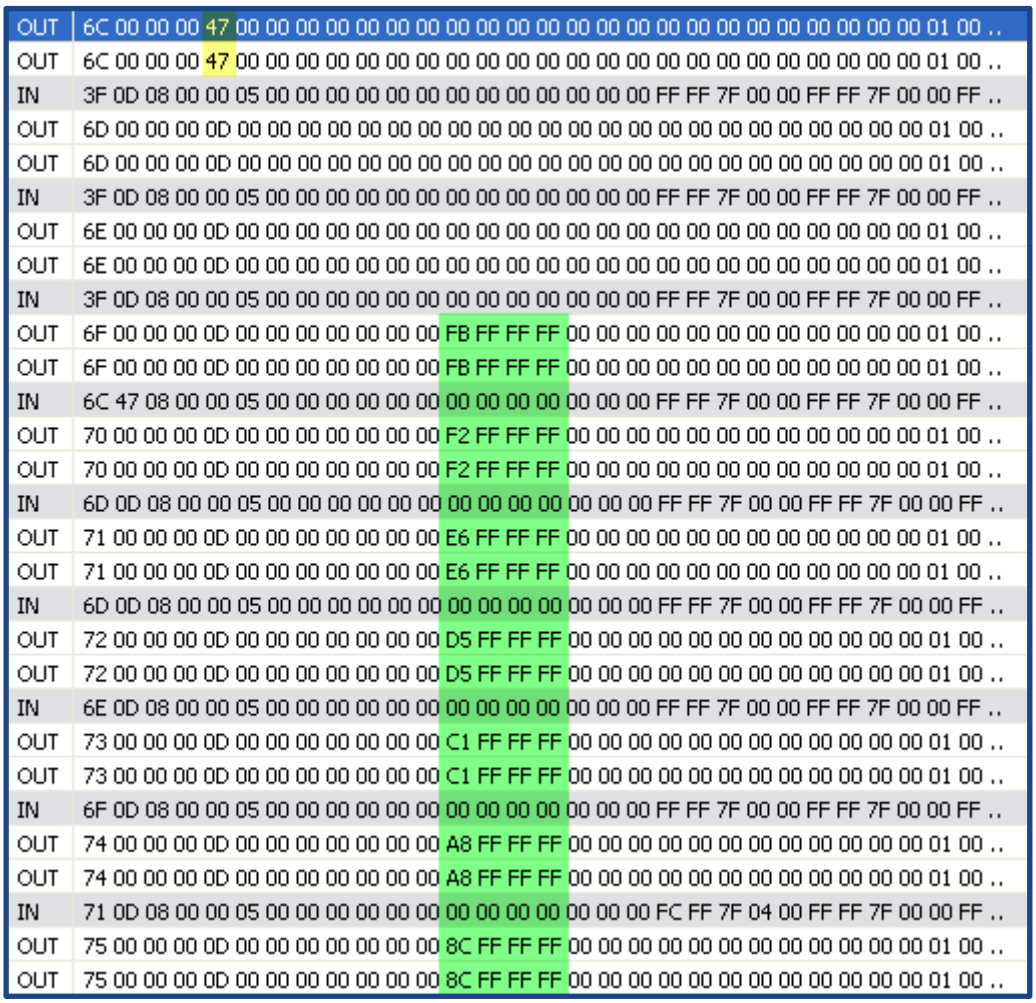
CAPTURA 3 - FIN MOVIMIENTO.

En el momento en que finaliza el movimiento vemos que sí se produce un envío de códigos de operación diferentes al que se envía normalmente (0x0D). Además podemos apreciar como el valor que aumentaba anteriormente ya ha parado.

Se ha verificado realizando pequeños movimientos consecutivos que los comandos que paran el movimiento son los mismos siempre.

Ahora vamos a hacer la misma prueba para valores negativos devolviendo el robot a su posición de Home y partiendo de la misma pero esta vez hacia posiciones negativas. Esto nos servirá para saber si hay grandes diferencias a la hora de realizar movimientos en direcciones opuestas ya sea bien al iniciar, durante, o al finalizar un movimiento.

Como antes partiremos de la posición de Home por lo que hemos visto en los mensajes analizados para movimientos positivos, el movimiento debería empezar con un código de operación 0x47 siempre y cuando los movimientos hacia valores negativos se comporten de igual manera que los realizados a valores positivos y no hagan uso de un código diferente.



OUT	6C 00 00 00 47 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	6C 00 00 00 47 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	3F 0D 08 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF ..
OUT	6D 00 00 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	6D 00 00 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	3F 0D 08 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF ..
OUT	6E 00 00 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	6E 00 00 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	3F 0D 08 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF ..
OUT	6F 00 00 00 0D 00 00 00 00 00 00 00 FB FF FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	6F 00 00 00 0D 00 00 00 00 00 00 00 FB FF FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	6C 47 08 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF ..
OUT	70 00 00 00 0D 00 00 00 00 00 00 00 F2 FF FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	70 00 00 00 0D 00 00 00 00 00 00 00 F2 FF FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	6D 0D 08 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF ..
OUT	71 00 00 00 0D 00 00 00 00 00 00 00 E6 FF FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	71 00 00 00 0D 00 00 00 00 00 00 00 E6 FF FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	6D 0D 08 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF ..
OUT	72 00 00 00 0D 00 00 00 00 00 00 00 D5 FF FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	72 00 00 00 0D 00 00 00 00 00 00 00 D5 FF FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	6E 0D 08 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF ..
OUT	73 00 00 00 0D 00 00 00 00 00 00 00 C1 FF FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	73 00 00 00 0D 00 00 00 00 00 00 00 C1 FF FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	6F 0D 08 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF ..
OUT	74 00 00 00 0D 00 00 00 00 00 00 00 A8 FF FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	74 00 00 00 0D 00 00 00 00 00 00 00 A8 FF FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	71 0D 08 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FC FF 7F 04 00 FF FF 7F 00 00 FF ..
OUT	75 00 00 00 0D 00 00 00 00 00 00 00 8C FF FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	75 00 00 00 0D 00 00 00 00 00 00 00 8C FF FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..

CAPTURA 4 – INICIO MOVIMIENTO NEGATIVO.

Efectivamente, podemos apreciar como los movimientos negativos también utilizan el código de operación 0x47. Sin embargo si ocurre algo más curioso con los valores del codificador, ahora en lugar de 2 bytes vemos que el movimiento parece ocupar ahora 4 bytes siendo los valores de los dos últimos 0xFF.

Una vez realizadas varias capturas y estudiados los valores de las mismas se ha llegado a la conclusión de que los dos últimos bytes indican el signo del encoder, en este caso dos bytes con valor 0x00 para números positivos y 0xFF para negativos.

Durante la parte intermedia del movimiento no debería ocurrir nada más que el cambio en el valor de los bytes 13 y 14, ya que los que indican, al menos de momento por lo que sabemos hasta ahora el signo, cambiaron al pasar de 0 a valores negativos.

OUT	75 00 00 00 0D 00 00 00 00 00 00 00 8C FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	75 00 00 00 0D 00 00 00 00 00 00 00 8C FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	71 0D 08 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 FC FF 7F 04 00 FF FF 7F 00 00 FF ..
OUT	76 00 00 00 0D 00 00 00 00 00 00 00 6C FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	76 00 00 00 0D 00 00 00 00 00 00 00 6C FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	72 0D 08 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 F5 FF 7F 0A 00 FF FF 7F 00 00 FF ..
OUT	77 00 00 00 0D 00 00 00 00 00 00 00 48 FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	77 00 00 00 0D 00 00 00 00 00 00 00 48 FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	73 0D 08 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 E6 FF 7F 0E 00 FF FF 7F 00 00 FF ..
OUT	78 00 00 00 0D 00 00 00 00 00 00 00 1F FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	78 00 00 00 0D 00 00 00 00 00 00 00 1F FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	75 0D 08 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 B8 FF 7F 0D 00 FF FF 7F 00 00 FF ..
OUT	79 00 00 00 0D 00 00 00 00 00 00 00 F3 FE FF FF 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	79 00 00 00 0D 00 00 00 00 00 00 00 F3 FE FF FF 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	75 0D 08 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 B8 FF 7F 0D 00 FF FF 7F 00 00 FF ..
OUT	7A 00 00 00 0D 00 00 00 00 00 00 00 C5 FE FF FF 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	7A 00 00 00 0D 00 00 00 00 00 00 00 C5 FE FF FF 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	76 0D 08 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 9C FF 7F 0E 00 FF FF 7F 00 00 FF ..
OUT	7B 00 00 00 0D 00 00 00 00 00 00 00 96 FE FF FF 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	7B 00 00 00 0D 00 00 00 00 00 00 00 96 FE FF FF 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	77 0D 08 00 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00 7C FF 7F 0D 00 FF FF 7F 00 00 FF ..
OUT	7C 00 00 00 0D 00 00 00 00 00 00 00 67 FE FF FF 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	7C 00 00 00 0D 00 00 00 00 00 00 00 67 FE FF FF 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	78 0D 08 00 00 04 00 00 00 00 00 00 00 00 00 00 00 00 00 59 FF 7F 0D 00 FF FF 7F 00 00 FF ..
OUT	7D 00 00 00 0D 00 00 00 00 00 00 00 38 FE FF FF 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	7D 00 00 00 0D 00 00 00 00 00 00 00 38 FE FF FF 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	7A 0D 08 00 00 04 00 00 00 00 00 00 00 00 00 00 00 00 00 06 FF 7F 0E 00 FF FF 7F 00 00 FF ..
OUT	7E 00 00 00 0D 00 00 00 00 00 00 00 09 FE FF FF 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	7E 00 00 00 0D 00 00 00 00 00 00 00 09 FE FF FF 00 00 00 00 00 00 00 00 00 00 00 00 01 00 ..

CAPTURA 5 - MOVIMIENTO NEGATIVO.

Al igual que ocurrió durante la parte intermedia del movimiento positivo vemos como en este caso simplemente se sigue variando el valor del codificador. Sin embargo también podemos apreciar algo más a la hora de analizar los valores del mismo y es que se cuentan restando del valor 0xFF 0xFF en los bytes 13 y 14 en orden inverso. Teniendo en cuenta que el valor de estos dos bytes sería 65535 se puede conocer el valor negativo del codificador fácilmente restando lo que indican los bytes a esta cifra. Por ejemplo en el primer mensaje de la anterior captura:

0xFF 0x8C - 0xFF 0xFF =65420 - 65535 = -115

Se han realizado una serie de verificaciones con diferentes valores con el fin de confirmar esta interpretación y efectivamente han coincidido con los que nos indica Scorbase.

Finalmente analizaremos si la finalización de un movimiento es igual sea hacia valores positivos como hacia valores negativos.

OUT	BF 00 00 00 0D 00 00 00 00 00 00 00	53 F2 FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	BF 00 00 00 0D 00 00 00 00 00 00 00	53 F2 FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	BB 0D 08 00 00 04 00 00 00 00 00 00	00 00 00 00 00 00 00 16 F3 7F 01 00 FF FF 7F 00 00 FF ..
OUT	C0 00 00 00 0D 00 00 00 00 00 00 00	38 F2 FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	C0 00 00 00 0D 00 00 00 00 00 00 00	38 F2 FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	BC 0D 08 00 00 04 00 00 00 00 00 00	00 00 00 00 00 00 00 E8 F2 7F 00 00 FF FF 7F 00 00 FF ..
OUT	C1 00 00 00 0D 00 00 00 00 00 00 00	22 F2 FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	C1 00 00 00 0D 00 00 00 00 00 00 00	22 F2 FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	BE 0D 08 00 00 04 00 00 00 00 00 00	00 00 00 00 00 00 00 93 F2 7F FE FF FF FF 7F 00 00 FF ..
OUT	C2 00 00 00 0D 00 00 00 00 00 00 00	10 F2 FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	C2 00 00 00 0D 00 00 00 00 00 00 00	10 F2 FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	BE 0D 08 00 00 04 00 00 00 00 00 00	00 00 00 00 00 00 00 93 F2 7F FE FF FF FF 7F 00 00 FF ..
OUT	C3 00 00 00 0D 00 00 00 00 00 00 00	01 F2 FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	C3 00 00 00 0D 00 00 00 00 00 00 00	01 F2 FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	BF 0D 08 00 00 04 00 00 00 00 00 00	00 00 00 00 00 00 00 6E F2 7F FA FF FF FF 7F 00 00 FF ..
OUT	C4 00 00 00 0D 00 00 00 00 00 00 00	F6 F1 FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	C4 00 00 00 0D 00 00 00 00 00 00 00	F6 F1 FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	C0 0D 08 00 00 04 00 00 00 00 00 00	00 00 00 00 00 00 00 4F F2 7F F9 FF FF FF 7F 00 00 FF ..
OUT	C5 00 00 00 0D 00 00 00 00 00 00 00	F0 F1 FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	C5 00 00 00 0D 00 00 00 00 00 00 00	F0 F1 FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	C1 0D 08 00 00 04 00 00 00 00 00 00	00 00 00 00 00 00 00 33 F2 7F F9 FF FF FF 7F 00 00 FF ..
OUT	C6 00 00 00 4F 3F 53 00 00 00 00 00	ED F1 FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	C6 00 00 00 4F 3F 53 00 00 00 00 00	ED F1 FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	C2 0D 08 00 00 04 00 00 00 00 00 00	00 00 00 00 00 00 00 1A F2 7F F8 FF FF FF 7F 00 00 FF ..
OUT	C7 00 00 00 73 20 00 00 00 00 00 00	ED F1 FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	C7 00 00 00 73 20 00 00 00 00 00 00	ED F1 FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
IN	C2 0D 08 00 00 04 00 00 00 00 00 00	00 00 00 00 00 00 00 1A F2 7F F8 FF FF FF 7F 00 00 FF ..
OUT	C8 00 00 00 42 20 00 00 00 00 00 00	ED F1 FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..
OUT	C8 00 00 00 42 20 00 00 00 00 00 00	ED F1 FF FF 00 00 00 00 00 00 00 00 00 00 00 01 00 ..

CAPTURA 6 - FIN MOVIMIENTO NEGATIVO.

Al igual que antes para el movimiento positivo vemos que se envía la misma secuencia de códigos de operación una vez finalizado el movimiento.

Resumiendo lo que sabemos de momento podemos afirmar que antes de cualquier movimiento siempre se envía al controlador el código 0x47 y posteriormente se van incrementando o disminuyendo los valores en el búfer en unos bytes que indican las posiciones del encoder del motor o motores que se van a mover.

Ocurre lo mismo con la secuencia de 3 mensajes que se envían al finalizar el movimiento, son los mismos y esto nos facilitará el trabajo a la hora de desarrollar el driver.

Tras múltiples capturas se ha observado el siguiente patrón de mensajes para todos los movimientos y en cualquier dirección:

1. Envío de código de operación 0x47.

2. Envío de mensajes con código de operación 0x0D en los que se van modificando los valores para el motor que se mueve.

3. Envío de secuencia de 3 mensajes para terminar el movimiento.

Durante las pruebas anteriores se ha visto que el valor leído en los bytes corresponde con la posición del encoder. Esta comprobación se ha hecho al leer los valores a través de la pestaña "Contador de encoders". Haciendo la prueba para un valor aleatorio tras mover un poco la cintura del robot vemos los siguientes valores.

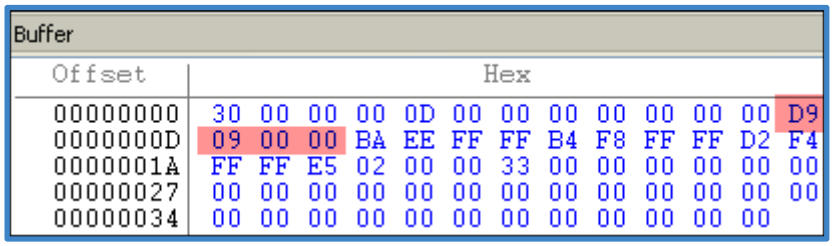


FIGURA 45 - LECTURA DEL BÚFER.

FIGURA 46 - LECTURA SCORBASE

Tenemos el valor hexadecimal 0xD9090000 en el que suponemos es el codificador del primer motor, sin embargo este valor no coincide con el que nos proporciona Scorbase que es 2521.

Esa cifra además de no coincidir es desorbitada por lo que debemos intentar buscar una explicación a cómo Scorbase recibiendo este mismo mensaje llega al valor 2521. Si pasamos 2521 a hexadecimal tenemos el valor 0x9D9 y si nos fijamos en el búfer, para llegar a este valor únicamente es necesario interpretar los bytes en orden inverso y ahora sí vemos como coincide con el valor de Scorbase.

Al estudiar los valores negativos se ha visto que si no tenemos en cuenta los dos bytes con el valor 0xFF pero sí el resto de los valores, de nuevo llegamos al valor que indica Scorbase para números negativos. Por lo tanto es seguro afirmar ahora que estos dos bytes con el valor 0xFF lo que indican es el signo del codificador.

Estas pruebas se han realizado posteriormente para todos los ejes que hacen uso de un solo motor (Cintura, Hombro y Codo) siendo los resultados los mismos.

Ya comprendemos cómo se mueve un eje que se controla a través de un único motor y ahora vamos a ver cómo se comporta al mover ejes que necesitan de dos motores para realizar sus movimientos que en el caso del Scorbot-ER 4u son los correspondientes a la muñeca y el roll de la pinza.

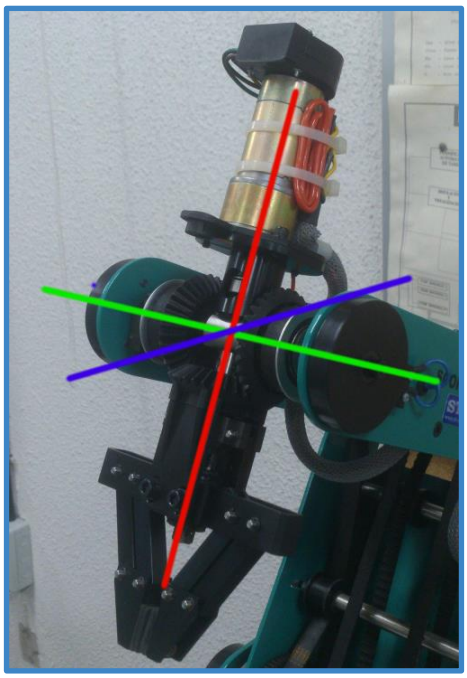


FOTO 2 - EL EJE DE ROTACIÓN EN VERDE CORRESPONDE AL EJE DE INCLINACIÓN DE LA PINZA Y EL ROJO AL EJE DE ROLL.

Vamos a realizar una serie de capturas y observar en Scorbase como varían los valores de los codificadores motores 4 y 5 encargados de los movimientos de estos ejes.

FIGURA 47 - CONTADORES MOTORES AL REALIZAR MOVIMIENTO DE ROLL CON LA PINZA.

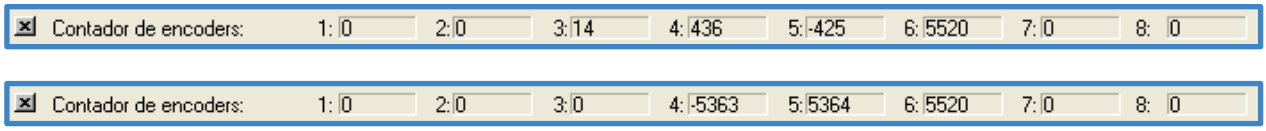


FIGURA 48 - LÍMITES INFERIOR Y SUPERIOR DE INCLINACIÓN DE LA PINZA.

Se puede apreciar como para el roll de la pinza los dos motores se mueven en la misma dirección y exceptuando fluctuaciones mínimas de unos cuantos pasos mantienen un valor casi idéntico.

También vemos como para la inclinación de la pinza los motores se mueven en direcciones opuestas aunque al igual que antes en la misma proporción y manteniendo en muchos casos el mismo valor.

Velocidad de movimiento

Scorbase nos permite escoger una velocidad al realizar movimientos y vamos a intentar averiguar cómo transmite esta indicación al robot cuando se envía una orden. Para ello vamos a realizar una serie de movimientos a diferente velocidad a través de los controles directos para mover los ejes de Scorbase.

El primer movimiento va a ser con velocidad 1, el segundo a velocidad 5 y el tercero a velocidad 10. El movimiento tendrá una duración de 2 segundos tras el cual se volverá a leer el búfer.

Valor codificador inicial	Valor codificador final	Velocidad
0	1500	1
0	10000	5
0	15000	10

Se puede apreciar como el movimiento, durando lo mismo, ha sido mayor cuando hemos incrementado la velocidad por lo que podemos estar seguros que este dato se transmite al robot y éste lo tiene en cuenta al moverse. Ahora debemos averiguar cuándo se indica este parámetro al controlador ya que tras realizar múltiples capturas y estudiar su tráfico se ha apreciado que la secuencia que se envía para realizar los movimientos está compuesta por exactamente los mismos mensajes.

La solución está en interpretar no sólo los mensajes sueltos que se envían al robot, sino toda la secuencia desde que empieza el movimiento hasta que acaba. Si hacemos esto si se aprecian algunos comportamientos interesantes, principalmente dos:

1. El movimiento no se envía al controlador como una cifra final a la que desplazar un codificador motor sino a través de pequeñas variaciones en el estado que enviamos al mismo.

2. El movimiento es más rápido o más lento controlando la variación de estos valores.

No existe ningún parámetro o función que indique al controlador la velocidad a la que realizar los movimientos. Simplemente se realizan variaciones de mayor o menor tamaño al valor del codificador como se ha comprobado al analizar los datos capturados.

3.9 Encontrando la posición Home

El robot Scorbot encuentra su posición de Home a través de una serie de micro interruptores que se encuentran en cada uno de los ejes exceptuando el de apertura/cierre de pinza.

El primer micro interruptor está situado en la base de la cintura del brazo robótico y es accionado por un tope que se encuentra en la base del brazo robótico.

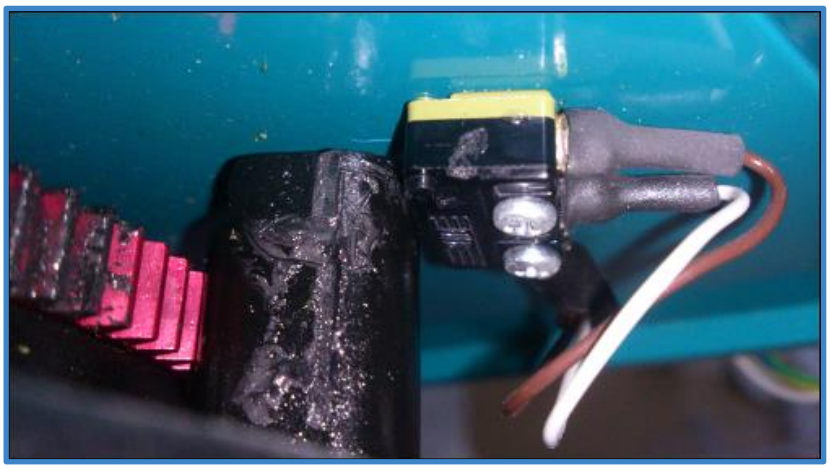


FOTO 3 - MICROINTERRUPTOR CORRESPONDIENTE A LA CINTURA.

Los ejes correspondientes al hombro, codo y muñeca disponen de un micro interruptor accionado por el propio eje al rotar como se puede apreciar en la imagen.

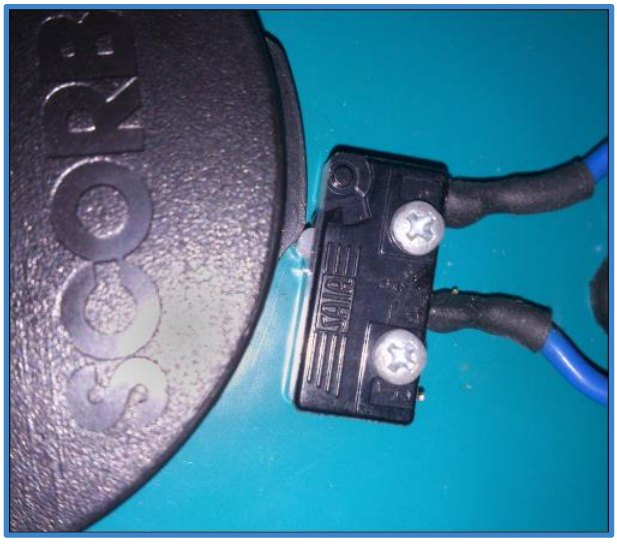


FOTO 4 - MICROINTERRUPTOR CORRESPONDIENTE AL HOMBRO.

El roll de la pinza es accionado a través de una protuberancia en el eje de rotación que pulsa un micro interruptor que se encuentra en la muñeca del brazo robot.

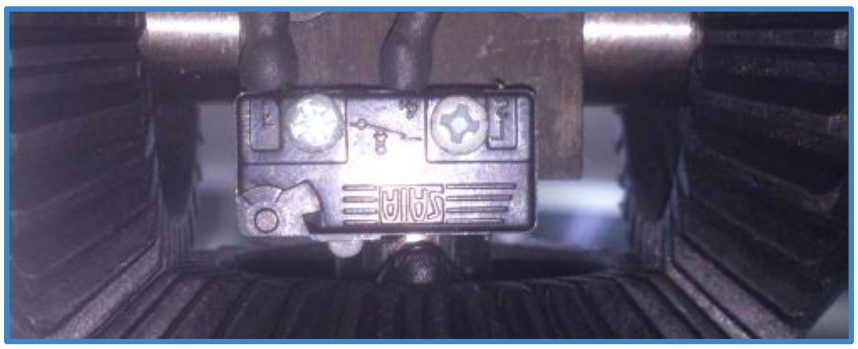


FOTO 5 - MICRINTERRUPTOR CORRESPONDIENTE AL ROLL DE LA PINZA.

Sin embargo encontrar la posición de Home simplemente llegando a los microinterruptores no es una tarea simple ya que en ocasiones como el encendido del robot donde todos los contadores motores están a cero no sabemos a ciencia cierta hacia qué dirección debe rotar el eje para encontrar el interruptor.

Otro problema que ha surgido durante este proceso es el hecho de que cuando intentamos mover un eje más allá de lo que permite el mismo nos encontramos con que se produce una pérdida de conexión con el robot.

Vamos a realizar un estudio de los mensajes transmitidos por el controlador con el fin de entender cómo detectar el estado de los microinterruptores o una colisión.

Los microinterruptores

El primer paso será averiguar cómo comunica el controlador a Scorbase que se ha pulsado un micro interruptor. Para ello se llevará a cabo una captura continua en estado ocioso y se intentará extraer de los mensajes capturados los bytes y el formato utilizados para comunicar el estado de los microinterruptores a Scorbase.

Durante este proceso se va a iniciar una captura para posteriormente ir pulsando uno a uno cada uno de los 5 microinterruptores del robot con la ayuda de un destornillador.

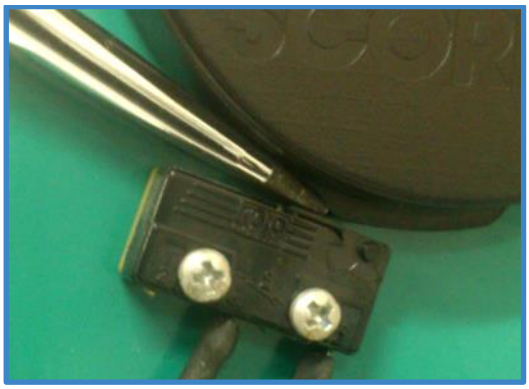


FOTO 6 - MICROINTERRUPTOR.

La primera captura se realiza en estado ocioso y utilizamos los controles de Scorbase para colocar el robot en una posición donde no esté pulsando ninguno de los interruptores.

Offset	Hex
00000000	D3 0D 08 00 00 00 00 00 00 00 00 00 00 00
0000000E	00 00 00 00 00 35 F5 7F 00 00 5E F3 7F 00
0000001C	00 B2 F8 7F 00 00 FF 00 80 00 00 15 0F 80
0000002A	00 00 03 00 80 04 00 FF FF 7F 00 00 FF FF
00000038	7F 00 00 00 2E FF FF FF

Ahora vamos activando los 5 microinterruptores uno a uno durante unos segundos hasta haberlos activado todos y una vez filtrados los mensajes veremos que podemos extraer de los mismos.

Offset	Hex [003F]
00000000	D4 0D 08 00 00 01 00 00 00 00 00 00 00 00
0000000E	00 00 00 00 00 35 F5 7F 00 00 5E F3 7F 00
0000001C	00 B2 F8 7F 00 00 FF 00 80 00 00 15 0F 80
0000002A	00 00 03 00 80 04 00 FF FF 7F 00 00 FF FF
00000038	7F 00 00 00 30 FF FF FF

Offset	Hex
00000000	4D 0D 08 00 00 02 00 00 00 00 00 00 00 00
0000000E	00 00 00 00 00 35 F5 7F 00 00 5E F3 7F 00
0000001C	00 B2 F8 7F 00 00 FF 00 80 00 00 15 0F 80
0000002A	00 00 03 00 80 04 00 FF FF 7F 00 00 FF FF
00000038	7F 00 00 00 AA FF FF FF

Offset	Hex [003F]
00000000	26 0D 08 00 00 04 00 00 00 00 00 00 00 00
0000000D	00 00 00 00 00 00 D8 09 80 00 00 B9 EE
0000001A	7F 00 00 B3 F8 7F 00 00 D1 F4 7F 00 00
00000027	E4 02 80 00 00 32 00 80 00 00 FF FF 7F
00000034	00 00 FF FF 7F 00 00 00 46 FF FF FF

Offset	Hex [003F]
00000000	C9 0D 08 00 00 08 00 00 00 00 00 00 00 00
0000000D	00 00 00 00 00 00 D8 09 80 00 00 B9 EE
0000001A	7F 00 00 B3 F8 7F 00 00 D1 F4 7F 00 00
00000027	E4 02 80 00 00 32 00 80 00 00 FF FF 7F
00000034	00 00 FF FF 7F 00 00 00 ED FF FF FF

Offset	Hex
00000000	72 0D 08 00 00 10 00 00 00 00 00 00 00 00
0000000D	00 00 00 00 00 00 D8 09 80 00 00 B9 EE
0000001A	7F 00 00 B3 F8 7F 00 00 D1 F4 7F 00 00
00000027	E4 02 80 00 00 32 00 80 00 00 FF FF 7F
00000034	00 00 FF FF 7F 00 00 00 9E FF FF FF

FIGURA 49 - ESTADOS DEL CONTROLADOR MIENTRAS SE PULSABAN LOS DIFERENTES MICROINTERRUPTORES.

Una vez estudiados los mensajes capturados apreciamos que el sexto byte del estado es el único que varía si no contamos los de sincronización. Como fuimos tocando los interruptores por el orden indicado podemos saber qué valor en el mensaje corresponde con cada interruptor.

Micro interruptor	Valor byte hexadecimal	Valor byte decimal
1	0x01	1
2	0x02	2
3	0x04	4
4	0x08	8
5	0x10	16

Estos valores se codifican al igual que las entradas digitales donde se interpretaban las 8 entradas como los bits individuales de un byte. En este caso nos encontramos con el mismo tipo de codificación. Aunque no sabemos cómo interpreta Scorbase internamente estos valores no nos afecta para nada a la hora de utilizar nosotros el método que creamos adecuado.

La única duda que nos queda respecto a los microinterruptores es ver si también se comportan como las entradas digitales cuando tenemos múltiples activados a la vez. Para ello vamos a realizar una captura activando dos o más entradas.

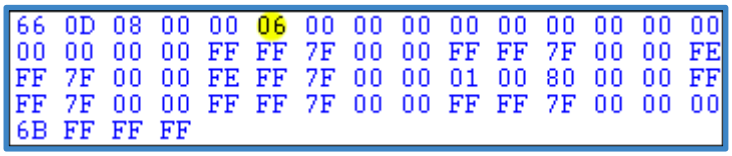


FIGURA 50 - ACTIVACIÓN DE MICROINTERRUPTORES CORRESPONDIENTES A CODO Y HOMBRO DEL ROBOT.

Al activar los microinterruptores del hombro 0x02 y el codo 0x04 del robot podemos ver como recibimos la entrada 0x06, es decir la suma de las dos entradas. Ocurre igual para cualquier otra combinación de las mismas por lo que podemos asegurar que efectivamente se comporta igual que las entradas digitales.

Ya sabemos por lo tanto dónde y cómo nos indica el robot el estado de los microinterruptores.

Desconexión al forzar motor

Ahora que ya sabemos trabajar con los microinterruptores debería ser más sencillo desarrollar una función capaz de guiar el robot a la posición de Hard Home. Sin embargo nos surge una pregunta, Si todos los ejes encuentran su posición de Home a través de un micro interruptor,

¿ Cómo lo hace el eje de apertura y cierre de la pinza ?

Durante las capturas pudimos apreciar que cuando un eje intentaba sobrepasar físicamente su ángulo máximo de giro y chocaba aparecía un valor diferente a 0 en el byte 59 del estado. Al producirse este choque y aparecer este valor se pierde la capacidad de movimiento y el robot se para por lo que Scorbase nos indica el error y en que eje se produjo mientras que también nos informa de que tenemos que volver a hacer un control-on para poder volver a operar el robot.

Offset	Hex
00000000	A3 0D 08 00 00 0E 00 00 00 00 00 00 00 00
0000000E	00 00 00 00 00 1F 3D 80 0C EF 00 00 80 00
0000001C	00 01 00 80 00 00 FF FF 7F 00 00 FF FF 7F
0000002A	00 00 FF FF 7F 00 00 FF FF 7F 00 00 FF FF
00000038	7F 00 00 01 10 FF FF FF

FIGURA 51 - COLISIÓN DEL MOTOR DEL EJE 1.

Offset	Hex [0004]
00000000	CE 0D 0C 00 00 05 00 00 00 00 00 00 00 00
0000000E	00 00 00 00 00 FF FF 7F 00 00 08 00 80 00
0000001C	00 A3 FF 7F D1 FF EB E2 7F D9 04 F0 1B 80
0000002A	2A FB 00 00 80 00 00 FF FF 7F 00 00 FF FF
00000038	7F 00 00 18 4D FF FF FF

FIGURA 52 - COLISIÓN DE LOS MOTORES 4 Y 5 (0X10 + 0X08) DURANTE INCLINACIÓN DE LA PINZA.

La única manera de saber cómo hace Scorbase para retomar el control una vez ocurre esto es engañar al programa original encendiendo el robot en una posición próxima a la de choque, por lo que tendremos todos los codificadores a cero, y realizando un movimiento manual en esa misma dirección.

Se ha observado que al colisionar un eje se pierde el control momentáneamente pero luego Scorbase es capaz de recuperarlo y para ello realiza una re sincronización con el controlador de manera que su mensaje de salida codifique los motores con los valores que el robot tiene internamente.

Así es como llega a la posición de Home la pinza, realizando una apertura indefinida hasta que el controlador informa de la colisión del eje y luego re sincronizando con ella a la posición que nos está devolviendo en su estado.

Este método de re sincronización se ha probado para el resto de los motores y se ha visto que es utilizado por todos los motores del robot en todos los ejes.

Codificadores motores y Home

Una vez llegamos a la posición de Home debemos realizar un reset de los codificadores motores al igual que hace Scorbase ya que en la posición de Home se considera que los codificadores están a cero.

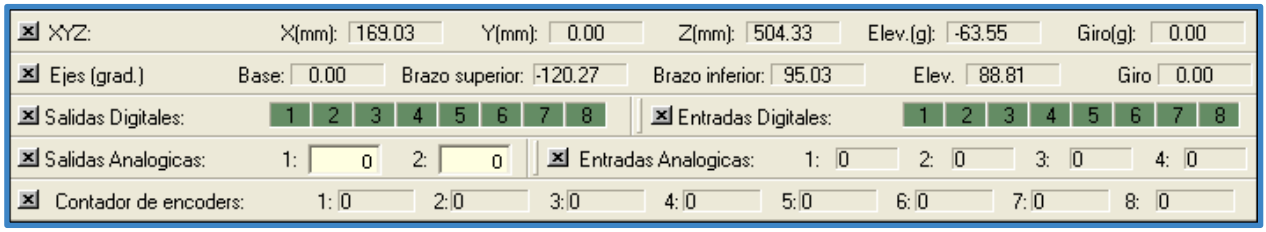


FIGURA 53 - PESTAÑAS DE INFORMACIÓN TRAS REALIZAR HOME.

Normalmente no vamos a tener los motores a cero al llegar a la posición de Home ya que los motores se encuentran a cero al encender el robot y siempre se van a realizar una serie de movimientos para llegar a la posición de Home que no va a dejar todos los codificadores con el valor inicial que deseamos, en ocasiones incluso tras hacer Home con el programa original Scorbase veremos como algún valor puede pasar a ser 1 ó -1 al cabo de un tiempo en estado ocioso.

Lo que nos interesa ahora es averiguar cómo resetea los motores Scorbase para poder luego implementar nosotros esta funcionalidad en nuestra función de Home. Realizar capturas del proceso de Home nos da un exceso de mensajes en la captura ya que se realizan movimientos de motores durante el mismo. Sin embargo existe una función que nos ofrece Scorbase que podría ser interesante.

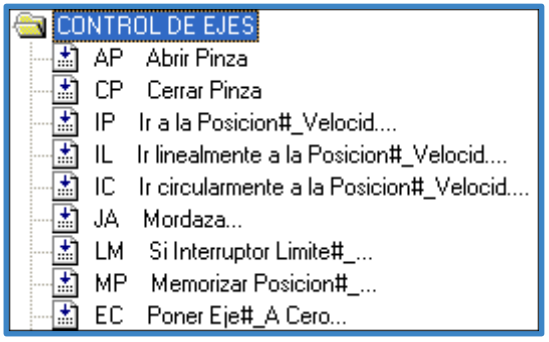


FIGURA 54 - FUNCIONES SCORBASE.

La función Poner Eje A Cero está disponible como vemos en la sección Control de ejes de Scorbase. Vamos a realizar una llamada a la función y el eje 1 ya que Scorbase nos ofrece un menú donde elegir y al parecer está disponible, al menos Scorbase nos la deja elegir.

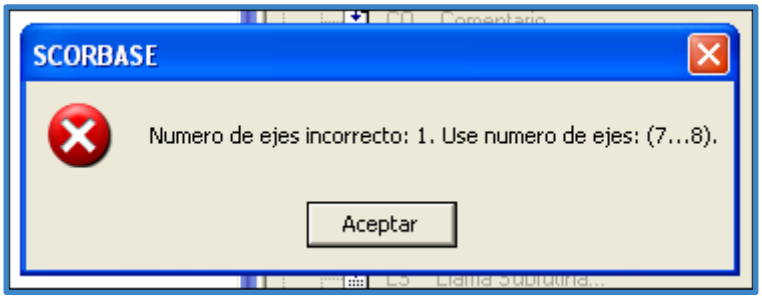


FIGURA 55 - ERROR DE SELECCIÓN DE EJE EN SCORBASE.

Sin embargo podemos ver que existe un problema al intentar su ejecución y es que esta función sólo nos permite resetear los codificadores asociados a los puertos externos 7 y 8. Aun así vamos a realizar una captura y ver qué se envía al controlador durante el reset de los codificadores asociados a los puertos 7 y 8.

Tenemos una secuencia de mensajes de la que hemos extraído todos los que se han enviado al controlador que no fueran los del estado ocioso (0x0D) y han sido los siguientes:

OUT	41 00 00 00 4F 40 54 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
OUT	41 00 00 00 4F 40 54 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
IN	14 0D 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 AD F6 7F 00 00 FF FF 7F 00 00 FE ..
OUT	42 00 00 00 48 40 00 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
OUT	42 00 00 00 48 40 00 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
IN	15 0D 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 AD F6 7F 00 00 FF FF 7F 00 00 FE ..
OUT	43 00 00 00 0D 00 00 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
OUT	43 00 00 00 0D 00 00 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..

CAPTURA 7 - SE ENVÍAN LOS COMANDOS 4F 40 54 Y 48 40 00

IN	42 48 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 AD F6 7F 00 00 FF FF 7F 00 00 FE ..
OUT	70 00 00 00 0D 00 00 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
OUT	70 00 00 00 0D 00 00 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
IN	43 0D 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 AD F6 7F 00 00 FF FF 7F 00 00 FE ..
OUT	71 00 00 00 0D 00 00 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
OUT	71 00 00 00 0D 00 00 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
IN	44 0D 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 AD F6 7F 00 00 FF FF 7F 00 00 FE ..
OUT	72 00 00 00 47 00 00 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
OUT	72 00 00 00 47 00 00 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
IN	45 0D 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 AD F6 7F 00 00 FF FF 7F 00 00 FE ..
OUT	73 00 00 00 4F 40 53 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
OUT	73 00 00 00 4F 40 53 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
IN	45 0D 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 AD F6 7F 00 00 FF FF 7F 00 00 FE ..
OUT	74 00 00 00 0D 00 00 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
OUT	74 00 00 00 0D 00 00 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..

CAPTURA 8 - AL RECIBIR RESPUESTA DE LOS ANTERIORES ENVÍA 47 00 00 Y 4F 40 53

IN	73 4F 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 AD F6 7F 00 00 FF FF 7F 00 00 FE ..
OUT	78 00 00 00 0D 00 00 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
OUT	78 00 00 00 0D 00 00 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
IN	74 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 AD F6 7F 00 00 FF FF 7F 00 00 FE ..
OUT	79 00 00 00 0D 00 00 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
OUT	79 00 00 00 0D 00 00 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
IN	74 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 AD F6 7F 00 00 FF FF 7F 00 00 FE ..
OUT	7A 00 00 00 73 20 00 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
OUT	7A 00 00 00 73 20 00 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
IN	74 0D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 AD F6 7F 00 00 FF FF 7F 00 00 FE ..
OUT	7B 00 00 00 42 20 00 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..
OUT	7B 00 00 00 42 20 00 00 00 00 00 00 AE F6 FF FF 00 00 00 00 FF FF FF FF FF FF FF FF 02 00 ..

CAPTURA 9 - FINALMENTE ENVÍA LOS COMANDOS 73 20 00 Y 42 20 00

Una vez realizadas múltiples capturas y analizados los mensajes tanto para el reset del motor 7 y posteriormente para el 8 se ha apreciado que el único de los mensajes que se envían que varía es el segundo (0x48 0x40 0x00) más concretamente el segundo parámetro y por lo tanto debe ser el que codifica el motor que se va a resetear.

Se han probado diferentes motores y resumido en la siguiente tabla:

Motor	1	2	3	4	5	6	7	8
Valor	12	16	20	24	28	32	36	40

Los bytes del búfer que codifican la posición se resetean en nuestro mensaje, es decir, en nuestro búfer del mensaje de salida vamos a codificar la posición de dicho motor con los cuatro bytes a cero.

Los mensajes finales coinciden con los dos últimos que se envían al controlador al finalizar un movimiento, no se ha realizado ningún movimiento pero al estar reseteando el contador para Scorbase es como si se hubiera producido por lo que no resulta extraño que estos dos mensajes que indican un fin de movimiento se envíen aquí también al finalizar el reset del contador.

Resumen

En este apartado hemos descubierto cómo leer los microinterruptores y cómo salir de un bloqueo generado debido a una colisión en un eje. Además hemos visto como resetear cualquier codificador motor.

Esta información nos será de gran utilidad a la hora de implementar una función que lleve a cabo el Home del robot con éxito.

3.10 Dispositivos Externos

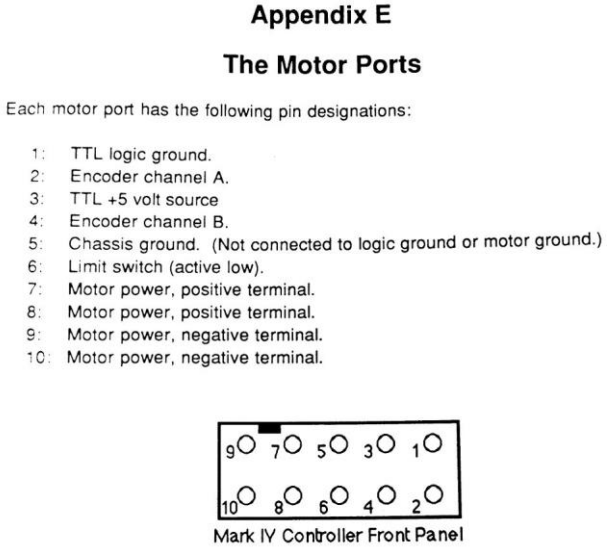
Implementación del uso de una Cinta Transportadora

El robot Scorbot-ER 4u nos proporciona dos salidas adicionales donde conectar ejes motores. Estos pueden ser con contador o sin el mismo como es el caso de la cinta transportadora de la que vamos a hacer uso durante la prueba de la interfaz Web.

Sabemos cómo controlar motores basándonos en los incrementos que debemos enviar en el estado al controlador, sin embargo en este caso no podemos controlar la cinta del mismo modo al no utilizar codificador. Para saber cómo arranca y para la cinta Scorbase vamos a realizar una serie de capturas y analizar los datos obtenidos.

También debemos tener en cuenta que Scorbase nos proporciona la posibilidad de indicar en que eje se encuentra la cinta y la velocidad a la cual queremos que se mueva por lo que se debe intentar averiguar también éstos parámetros con el fin de incluirlos en nuestro driver. Al igual que hicimos para estudiar el resto de funciones, vamos a conectar la cinta y activarla en Scorbase.

Para conectar la cinta es necesario conocer el esquema de conexión que nos proporciona el controlador a la hora de añadir motores externos, sin embargo, estos pines corresponden con el conector de un controlador Mark IV como podemos en la imagen:



Appendix E

The Motor Ports

Each motor port has the following pin designations:

1: TTL logic ground.
2: Encoder channel A.
3: TTL +5 volt source
4: Encoder channel B.
5: Chassis ground. (Not connected to logic ground or motor ground.)
6: Limit switch (active low).
7: Motor power, positive terminal.
8: Motor power, positive terminal.
9: Motor power, negative terminal.
10: Motor power, negative terminal.

Mark IV Controller Front Panel

FIGURA 56 - CONEXIÓN MOTOR CINTA

Conectaremos por lo tanto la salida al motor de la cintra transportadora en los puertos 9 y 7, que a su vez conectaremos a la salida de voltaje del controlador Scorbot.

FOTO 7 - CONEXIÓN MOTOR CINTA

Una vez filtrados los mensajes hemos desarrollado las funciones que emulan los mismos con el fin de nosotros poder controlar con la misma precisión que lo hace Scorbase el funcionamiento de la cinta. Nuestra función aceptará como parámetros el eje y la velocidad, el eje se traduce simplemente en el código de operación que enviamos en el tercer byte del primer mensaje dependiendo del eje y el segundo será la velocidad que se pasa como parámetro a través del quinto byte del segundo mensaje. Estos datos, como en los demás casos los conocemos gracias al estudio del tráfico y se han verificado como correctos los dos parámetros.

La velocidad de salida está originalmente establecida por Scorbase entre 0 y 10, cuando leímos el mensaje en la captura hemos visto como 0x64 (100 en decimal) se correspondía a la velocidad 5, y realizando otra posterior con el fin de ver el límite del parámetro vimos que 0xC8 (200) se corresponde a la velocidad 10. Por lo tanto podemos calcular el parámetro del mensaje en nuestra función antes de enviarlo como VELOCIDAD*20.

Finalmente para terminar la conexión de la cinta hemos conectado a una entrada digital el sensor que se encuentra en la cinta, su diagrama es el siguiente:

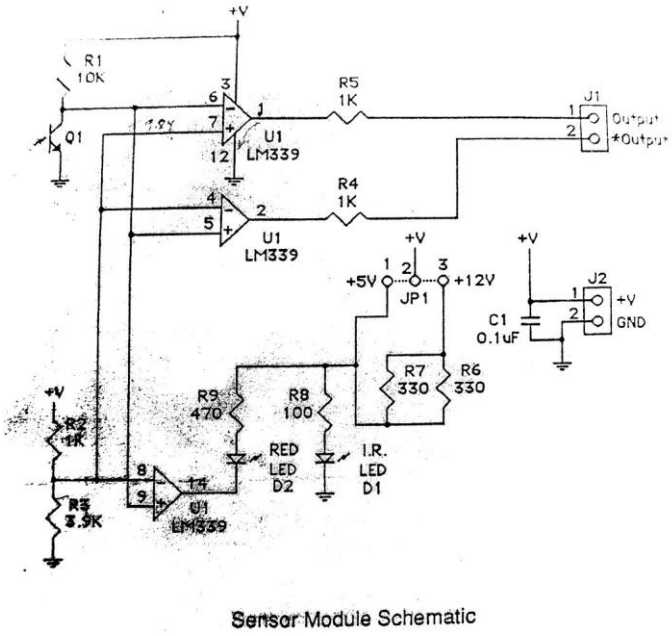


FIGURA 57 - ESQUEMA SENSOR INFRARROJO

Ha sido conectado a un puerto de 12 voltios que nos proporciona el controlador para su alimentación y su salida al puerto de entrada digital 3 del mismo:

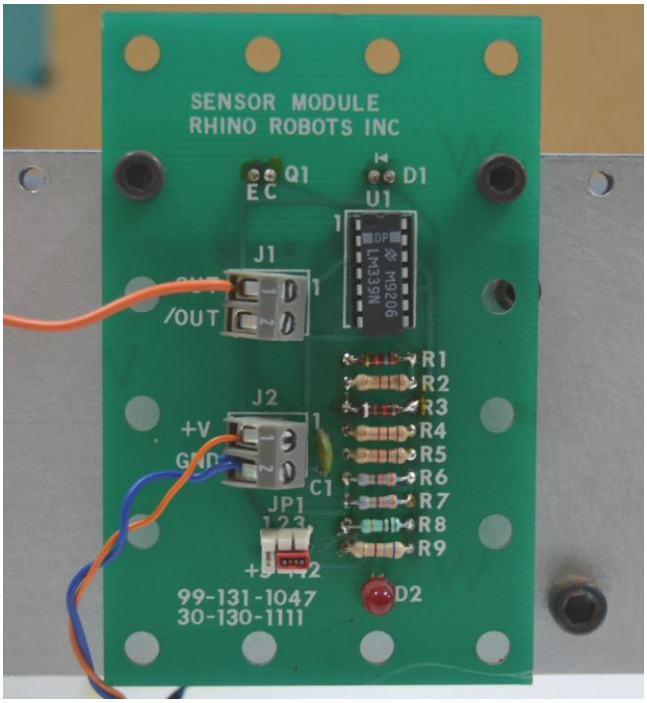


FOTO 8 - SENSOR INFRARROJO CINTA

Por lo tanto las conexiones finales han quedado como podemos apreciar en la siguiente imagen:

FOTO 9 - CONEXIÓN CINTA A CONTROLADOR

Este montaje nos permitirá probar cómo funcionan sobre todo las funciones que interactúan con elementos externos como entradas y salidas a motores adicionales.

4. Diseño

El driver se va a desarrollar como una librería en lenguaje C de manera que pueda ser incluida en cualquier programa que haga uso de este lenguaje y enlace la librería. Esta parte del software es la que luego podrá ser invocada por parte del intérprete o nuestra interfaz web para la comodidad de uso por parte del usuario.

No disponemos de bases de datos u otras estructuras que requieran ser explicadas en detalle aparte de los mensajes que se envían en cada función y sus parámetros, pero estos se analizarán en detalle en la siguiente fase por lo que a continuación vamos a ver la relación entre los diferentes componentes del proyecto:

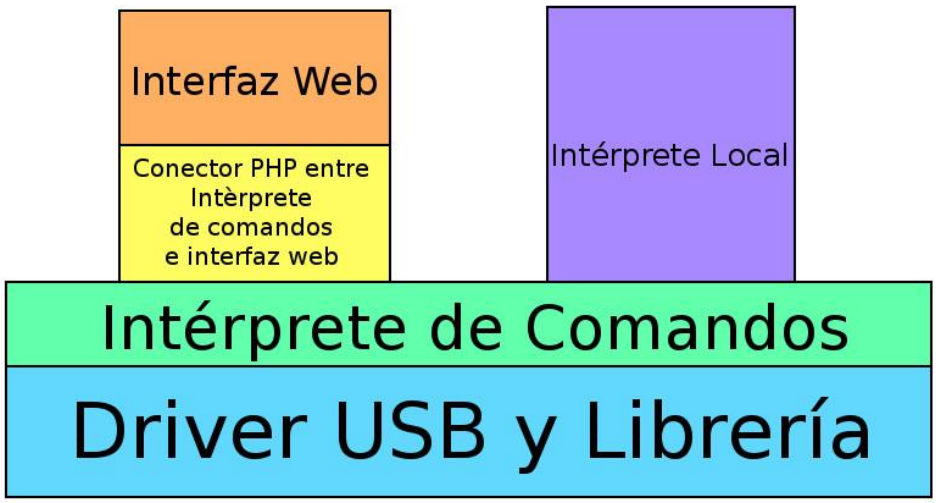


FIGURA 58 - DISEÑO

El driver USB y la librería son imprescindibles para el funcionamiento de cualquiera de los demás componentes. Además se ser utilizada directamente puede servirnos de soporte para un intérprete de comandos que traduce los comandos de un formato legible que hemos diseñado para cara orden en su correspondiente llamada a la función que se requiera junto con sus parámetros. Esto nos permite filtrar en esta nueva capa los mensajes que pueden enviarnos tanto el intérprete local como los mensajes que nos enviará la interfaz web.

Los mensajes procedentes de la interfaz web se comunican a través de un socket con el proceso intérprete de comandos que se ejecuta localmente en el servidor conectado al controlador, todo el proceso y el funcionamiento del conector se explica más detalladamente en el apartado 5.5 de este documento.

Finalmente la interfaz gráfica desarrollada con HTML5 y JQuery proporciona un acceso a las funciones lo más sencillo posible para el usuario además de realizar comprobaciones en el rango de los parámetros asegurándonos así no enviar mensajes no válidos al intérprete.

5. Desarrollo

Nuestra fase de desarrollo se centrará en los parámetros y funcionamiento de las funciones desarrolladas, sus parámetros y su uso. Como vimos durante la fase de análisis hemos obtenido un extenso conocimiento sobre las comunicaciones entre el computador y el controlador del Scorbot-ER 4u, al conocer los parámetros y comportamiento del robot frente a las diferentes acciones que puede realizar ahora debemos emular este comportamiento y conseguir operar de una manera satisfactoria con el mismo.

Por no extender necesariamente este apartado se puede consultar una función ejemplo incluida en el anexo del documento. El resto del código se encuentra disponible en el CD incluido.

5.1 Funciones de conexión

Una de las primeras funciones que hemos desarrollado para probar todas las demás es la de tomar el control del robot como ocurre cuando el programa original Scorbase realiza una llamada a la función control-on.

Sin embargo antes de tomar el control necesitamos conectarnos al dispositivo, esto lo hace Scorbase de manera automática de manera que es transparente para el usuario pero es un paso necesario en todas las aplicaciones que hagan uso del robot.

Por lo tanto para nuestra primera tarea de conectar y controlar el robot necesitaremos:

- Conectar al dispositivo USB.
- Tomar el control del robot.

Las funciones que nos proporcionan la conexión con el controlador son:

- int **conectar**(struct usb_dev_handle *h);
- int **control_on**();

La primera función conectará al dispositivo de manera que la segunda pueda tomar el control. Conectar es una función que simplemente tomará el handle pasado porque es necesario a partir de ahora para todos los mensajes que enviamos al controlador, es uno de los parámetros necesarios como vimos durante la comunicación a través de libUsb

int usb_bulk_write(**usb_dev_handle *dev**, int ep, char *bytes, int size, int timeout);
int usb_bulk_read(**usb_dev_handle *dev**, int ep, char *bytes, int size, int timeout);

5.2 Funciones de Estado

Las funciones de estado son todas aquellas que nos permitirán conocer el estado interno del robot en un momento dado. Esto no incluye todas aquellas funciones que interactúan con el exterior ya que esas se han considerado de entrada/salida.

Por lo tanto las funciones de estado nos proporcionarán:

- Estado de los microinterruptores.
- Estado de los codificadores de los motores.

Tanto en el caso de los microinterruptores como en el de los motores vamos a crear dos tipos de funciones que nos permitan conocer tanto el estado de un único eje así como el estado de todo el conjunto.

Microinterruptores

El estado de los microinterruptores como vimos durante el estudio de los mensajes viene codificado de forma inmediata en sexto byte del búfer.

El acceso al estado de los microinterruptores se realizará a través de las siguientes funciones:

- bool **leeSwitch**(int n);
- void **estadoSwitches**(bool* vector);

leeSwitch(eje) nos permitirá indicar por parámetro cuál de los ejes queremos leer y nos lo devolverá en una variable booleana que indica el estado del switch, siendo 0 inactivo y 1 activo.

Por otro lado la función estadoSwitches(vector) tomará como parámetro un vector booleano de como mínimo 6 elementos y al ser llamada nos lo rellenará con el estado de cada switch para lo que se servirá de la función anterior, leeSwitch(eje).

Codificadores Motores

La codificación de los motores es la información más importante que manejamos a la hora de controlar el movimiento y posición de cualquier robot. El robot Scorbot-ER 4u dispone de

codificadores de tipo incremental y no absolutos por lo que cualquier fallo en el posicionamiento del robot a la hora de realizar Home o de resetear los contadores sería arrastrado a lo largo de toda la operación con el mismo.

Los codificadores motores los podemos leer a través de la función:

- int **leeMotor**(int motor)
- int **valorMotor**(int motor)

La función leeMotor(motor) realizará una lectura del valor para el codificador indicado como parámetro en el estado del robot y nos la devolverá en un entero indicando los pasos actuals que ha dado el codificador desde su encendido o reseteo.

ValorMotor realiza la misma tarea pero devolviendo además el signo.

Visualización de datos

Una vez hemos leído el estado de los microswitches o de los codificadores motores también disponemos de dos funciones simples que se encargan de mostrar por consola el estado del robot desde la última lectura que hayamos hecho.

- int printSwitches();
- Int printEstado();

Simplemente nos mostrará por pantalla el contenido de los contadores motores y estado de los microswitches como podemos ver en la siguiente captura.

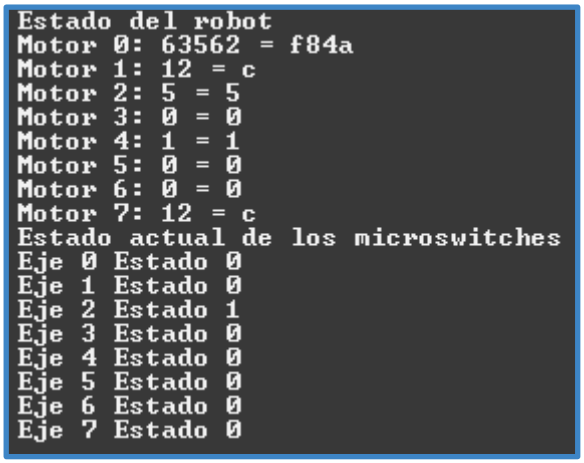


FIGURA 59 - DATOS ALMACENADOS EN LOS VECTORES

5.3 Funciones entrada / salida

Las funciones de entrada salida son las que tratan con todos los puertos del controlador y se dividen en dos categorías: E/S Digitales y E/S Analógicas.

Para el control de las mismas se van a crear una serie de funciones que permitan realizar todas las operaciones que permite el programa Scorbase y mejorar algunos aspectos del mismo.

Entradas/Salidas Digitales

Entrada Digital

El estado de las entradas digitales se encuentra en el byte 6 del estado del controlador que se nos envía con cada mensaje de entrada.

Una vez leído el estado del controlador, en este byte se dispondrá del estado a través de dos funciones:

- bool **leeEntradaDigital**(int numero)
- Char **leerEstadoEntradas**()

La primera de ellas nos permitirá obtener el estado de una de las entradas que indiquemos por parámetro y la segunda nos devolverá el byte completo con el estado de las 8 entradas digitales.

Salida Digital

Las salidas digitales se controlan a través del envío de un código de operación junto con un byte donde los bits que lo componen describen el estado de los 8 puertos de salida digital.

Por lo que la salida digital se puede controlar desarrollando funciones que envíen este código y el byte con el estado de las salidas que desee el usuario.

- void **activaSalidaDigital**(int numero)
- void **desactivaSalidaGidital**(int numero)
- void **cambiaSalidaDigital**(int numero)
- void **estadoSalidaDigital**(char estado)

Las dos primeras funciones tienen como objetivo el activar y desactivar la salida que se indique por parámetro y la tercera cambiar el estado actual de la salida que se indique.

La última función recibirá como parámetro un byte donde se codifica el estado de las 8 salidas y este estado será aplicado sobre las mismas.

E/S Analógicas

En el controlador Scorbot como vimos anteriormente durante el estudio de los mensajes codifica la entrada/salida analógica en un byte.

El controlador por lo tanto permite 255 posibles salidas cuyos valores podemos estimar a como múltiplos de 10 V./255 = 0.039 V. ahora vamos a ver dónde podemos redondear el valor para quedarnos lo más cerca posible de los 255 de los que disponemos, así conseguimos además una mejor resolución en la salida.

Para calcular la salida debemos aplicar la siguiente regla donde vamos a probar una serie de valores para la unidad de salida que es la cantidad más pequeña con la que podemos operar:

$$\frac{Voltaje\ de\ Salida}{unidad\ salida} = Byte\ de\ salida$$

0.04: 0.04*255 = 10.2
0.0393: 0.0393*255 = 10.0215
0.0392: 9.996
Como vemos el valor unitario más cercano que nos permite trabajar con 10 voltios es 0.0392 V.

Esta función enviará una salida al controlador que más se acerque a lo que indica el usuario, por lo tanto trabajará con cualquier valor entre 0 y 10v.

Para trabajar con los valores analógicos que vamos a leer y escribir se van a crear un par de funciones que nos devolverán el valor del byte de salida dado un voltaje y el voltaje cuando lo que leemos es el byte con la entrada.

Las dos funciones son **voltajeAByte** y **byteAVoltaje** que se encuentran en utils.h de nuestra librería. Ambas han sido probadas y se ha verificado su correcto funcionamiento como podemos ver a continuación:

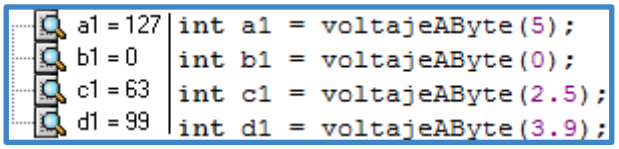


FIGURA 60 - PRUEBA DE LA FUNCIÓN VOLTAJE A BYTE

Entrada Analógica

La entrada analógica se puede leer en cualquier momento en el byte 7 y 9 del estado del controlador. Se dispone de dos entradas analógica por lo que la función tomará como parámetro cuál de las dos entradas deseamos leer.

- float **leeEntradaAnalógica**(int entrada);

Salida Analógica

La salida analógica al igual que la digital se controlar a través del envío de un código de operación y un byte con el valor codificado para el voltaje de salida. Sólo disponemos de una salida analógica por lo que trabajar con la misma resulta sencillo y podemos operar utilizando una única función.

- int **activaSalidaAnalogica**(float voltaje);

El parámetro requerido es un valor entre 0 y 10 voltios, el rango de operación de la salida analógica.

5.4 Funciones de movimiento

Las funciones de movimiento son todas aquellas que nos permitirán mover los ejes del robot.

- int **moverEje** (int eje, int pasos)
- int **moverEjeGrados** (int eje, float grados)
- int **moverEjes** (int [])
- int **moverEjesGrados** (int [])

Las dos primeras funciones se encargan de mover un eje el número de pasos o grados indicado. moverEjeGrados realizará una llamada a moverEje una vez convertidos los grados a pasos para el eje concreto, del mismo modo también es la función llamada por las dos siguientes (moverEjes y moverEjesGrados) que mueven todos los ejes por lo que el correcto funcionamiento de la primera era fundamental para que todas las demás funcionasen correctamente.

Otras funciones de movimiento también importantes son las que hacen uso de los puntos almacenados en memoria y nos permiten movernos entre los mismos.

- int **irPos**(int [])

irPos simplemente recibe como parámetro un vector con los valores objetivo para los 6 ejes y se encarga de que el robot realice el movimiento adecuado para alcanzar ese estado.

La última función de movimiento desarrollada debido a su complejidad y que además necesitaba de una sincronización especial con el controlador ha sido aquella que permite el movimiento directo de los ejes del robot.

- int **moverEjeCont**(int eje, int direccion)

Como hemos visto anteriormente se llama a través de esta función para variar el valor de las variables globales de **movContinuo** y **movContinuoDir** pero el movimiento realmente se realiza a través del *hilo de sincronización*.

5.5 Aplicaciones

A continuación se detallan las aplicaciones desarrolladas aparte del driver y que hacen uso del mismo.

Pequeña aplicación para uso simple del robot

Esta aplicación nos proporciona una manera muy simple de trabajar ya que dispone de menús que nos van informando de lo que hace cada función y proporciona la posibilidad de utilizar el robot de una forma simple.

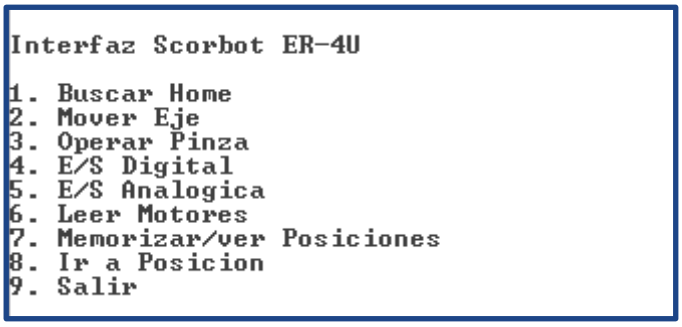


FIGURA 61 - APLICACIÓN DE PRUEBA

Esta aplicación además ha servido de base para ir probando todas las funciones de la librería durante su desarrollo. Esta aplicación es similar a un intérprete de comandos a excepción de que el intérprete no dispone de ningún tipo de menú y sólo se encarga de pasar nuestras órdenes al controlador robot.

Intérprete de comandos

El intérprete de comandos nos da la posibilidad de utilizar el robot a través de una serie de órdenes simples y por lo tanto supone una forma rápida y simple de poder operar con el dispositivo.

Esto no era posible con el controlador original del Scorbot-ER 4U ya que sólo nos ofrecía la operación a través de su interfaz gráfica así como otras limitaciones siendo la más relevante, sobretodo en un robot educativo, el control directo de los ejes motores.

Disponemos de dos versiones del intérprete de manera que podemos utilizarlo directamente a través de comandos que enviamos a través de terminal como en modo servidor donde escuchará los mensajes que le sean retransmitidos desde la interfaz realizada en HTML5 a través del conector PHP.

La mayor parte de las funcionalidades que nos proporcionan las dos versiones del intérprete están incluidas en una librería (interprete.h) que hemos utilizado en ambas versiones por lo que sus diferencias se limitan a la entrada de mensajes y retorno de resultados.

La principal función de la librería es **`ejecutaComando(comando,parametros,resultado)`** donde cómo podemos ver se le pasa el comando a ejecutar, los parámetros del mismo y una variable donde se nos devolverá el resultado si la función en cuestión retorna algún valor.

```
int ejecutaComando(char *comando, char *params,char *resultado){
	retornar = 0;
	if(funcEjes(comando,params))return 1;
	if(funcPinza(comando,params))return 1;
	if(funcMovimiento(comando,params))return 1;
	if(funcES(comando,params))return 1;
	if(funcSalida(comando,params))return 1;
	if(funcEstado(comando,params))return 1;
	return 0;
}
```

Como vemos se van llamando a una serie de funciones que en caso de reconocer el comando retornan 1 indicando que el comando ha sido reconocido. No debemos confundir esto con el valor que retorna el intérprete a la terminal ya que este se devuelve siempre en la variable resultado siempre y cuando la variable global retornar nos indique que existe dicho retorno por parte de la función llamada.

Se ha decidido dividir el reconocimiento de los comandos en diferentes funciones para poder categorizar en cierta manera las diferentes funciones dependiendo de su objetivo y si interactúan únicamente con el controlador o en caso contrario operan con el brazo robot.

Por ejemplo para la entrada/salida digital podemos ver el siguiente fragmento de código:

```
int funcES(char *comando,char *params){
	if(strcmp(comando,"SALIDADIGITAL")==0){
		int salida = atoi(params);
		char *parametro2 = strtok(params," ");
		parametro2 = strtok(NULL,"");
		int estado = atoi(parametro2);
		SetSalidaDigital(salida,bool(estado));
		return 1;
	}
	else if(strcmp(comando,"SALIDAANALOGICA")==0){
		int salida = atoi(params);
		char *parametro2 = strtok(params," ");
		parametro2 = strtok(NULL,"");
		float estado = atof(parametro2);
		activaSalidaAnalogica(salida,estado);
		return 1;
	}
...
```

Intérprete Terminal

En la versión terminal el intérprete operará con el robot enviándole los mensajes que el usuario vaya introduciendo haciendo uso de nuestra librería. Su principal diferencia con el programa simple que hemos diseñado para operar con el robot donde disponemos de menús y guiamos al usuario es que el intérprete se limita a reconocer los comandos y si éstos son válidos enviarlos al controlador directamente.

Intérprete Servidor

En esta versión el intérprete no recibirá la entrada del usuario a través de la terminal sino que se mantendrá escuchando en un puerto a la espera de que el servidor PHP le envíe mensajes.

Podemos apreciar en el siguiente fragmento de código como se abrirá un socket y se esperará que vayan llegando para realizar las llamadas a la función ejecutaComando, estos comandos llegan a través del servidor Apache.

Se hará una explicación extensa sobre esta comunicación en un apartado propio que explica toda la conexión entre el navegador web con la interfaz e intérprete.

A continuación repasamos los comandos del intérprete y desarrollamos con un ejemplo de uso su funcionalidad.

Comandos del Intérprete

Comandos asociados a los ejes

Los comandos de control de ejes son todos aquellos relacionados con el movimiento de los ejes del robot, la memorización de posiciones y su estado.

Inicialización

HardHome: Realiza el Hard Home del robot.

HardHome

SoftHome: Mueve el robot hasta la posición de softHome. Si al llamar a softHome todavía no se ha hecho HardHome se irá a la posición de inicio que tenía el robot al encender el controlador.

SoftHome

Poner eje a cero: este comando resetea el contador interno del robot para el motor indicado.

PonerCero *motor*

Ejemplos:

PonerCero 1

PonerCero 2

FijaSoftHome: La posición actual del robot será la nueva posición de Soft Home para el robot.

FijaSoftHome

HomeEje: Busca la posición de Home únicamente para el eje indicado por parámetro.

HomeEje *eje*

Ejemplos:

HomeEje 2

Control de la pinza

Los comandos descritos a continuación se encargan del control del actuador del robot, en el caso del robot Scorbot-ER 4u se trata de una pinza.

Abrir Pinza: Abre completamente la pinza.

AP | **AbrirPinza**

Cerrar Pinza: Cierra completamente la pinza.

CP | **CerrarPinza**

Estos dos comandos no requieren ningún parámetro.

Mordaza: Permite establecer una apertura de la pinza determinada por milímetros de apertura.

Mordaza *mm*

Movimiento de ejes y motores

Los comandos de movimiento de ejes se encargan del movimiento de los ejes que componen el

brazo robótico mientras que el movimiento de motores realiza movimientos sobre los motores independientemente del eje al que pertenezcan.

MoverEje: Esta orden tiene dos versiones ya que como su nombre indica se encarga de realizar movimientos sobre los ejes del robot y éstos se pueden definir tanto en pasos de motor como en grados de rotación del eje.

MoverEjePasos *eje pasos*

MoverEjeGrados *eje grados*

MoverMotor: El comando moverMotor a diferencia de moverEjes no tendrá en cuenta si un eje debe mover más de un motor para realizar un movimiento ni en qué dirección o si basta con mover un único motor. Este comando puede dar problemas al trabajar con los motores 4-5 ya que como vimos durante el desarrollo de funciones estos dos motores realizar diferentes movimientos en función de su dirección.

En este caso para evitar confusiones se va a limitar el movimiento a pasos de motor y no grados ya que estamos tratando directamente con los motores.

MoverMotor *motor pasos*

Memorización de posiciones

La memorización de posiciones nos permite almacenar en memoria la posición actual del brazo robótico de manera que podamos recuperarla en operaciones posteriores.

Memorizar Posición: nos permitirá almacenar la posición actual del robot en la variable indicada, si no existe, se creará.

MemPos *nombre*

Ir a Posición: llevará al robot a la posición almacenada en la variable indicada siempre y cuando ésta exista, de no existir no se realizará ningún movimiento.

IrPos *nombre*

Comandos de control de flujo

Estos comandos son utilizados en el intérprete servidor ya que éste es el que tendrá capacidad para tener esas etiquetas y saltos a los que vamos a hacer referencia.

Control de flujo

Espere: Pausa la ejecución del programa el número de milisegundos indicado.

Espere *milisegundos*

Salta A: Salta a la subrutina indicada.

SaltaA *nombreRutina*

Si: Ejecuta el comando indicado si se cumple la condición. Este comando se puede combinar con otros para tener el funcionamiento equivalente a una sentencia IF de un lenguaje de programación de alto nivel.

Si *Condición Comando*

Ejemplos:

Ejemplo de utilización del comando Si para funcionamiento equivalente a un if/else

```
entrada = leeEntradaDigital 3

Si  entrada==1 SaltaA rutina1

SaltaA rutina2
```

Comandos Entrada/Salida

Activación y desactivación de salidas digitales

SalidaDigital: El control sobre las salidas digitales se puede realizar a través de este comando donde se indicará el número de la salida y el estado que deseamos que tome.

salidaDigital *[0-7] (0|1)*

SD *[0-7] (0|1)*

Ejemplos:

salidaDigital 2 1

SD 6 0

En el primer caso hemos puesto la salida 2 en estado activo y en el segundo hemos desactivado la salida 6.

Activación de la salida analógica

SalidaAnalogica: Las salidas analógicas se controlan indicando el voltaje de salida y la salida que lo va a proporcionar.

salidaAnalogica *(0|1) voltaje*

SA *(0|1) voltaje*

Ejemplos:

salidaAnalogica 0 5

SA 1 7.5

En este ejemplo estamos poniendo la salida 0 a 5 voltios y la salida 1 a 7.5 voltios. El voltaje puede ser tanto entero como coma flotante siempre y cuando respete los valores dentro del rango de operación del controlador, en este caso 0-10 Voltios.

Lectura de entradas

LeeEntradaDigital: Utilizando este comando junto a una asignación se puede almacenar el valor de una entrada digital en una variable. Si no se realiza asignación se mostrará por pantalla.

LeeEntradaDigital *[0-7]*

ED *[0-7]*

Ejemplos:

variable = LeeEntradaDigital 5

La variable almacenará el valor de la entrada digital 5 en el momento de la llamada.

ED 3

Mostrará por pantalla el valor de la entrada digital 3.

LeeEntradaAnalogica: Devuelve el valor de la entrada analógica. Al igual que con la digital si se utiliza con la asignación almacenamos en una variable, en caso contrario, se muestra por pantalla el valor.

LeeEntradaAnalogica *[0-3]*

EA *[0-3]*

Ejemplos:

entrada2 = EA 2

Se almacena el valor actual de la entrada analógica 2 en la variable entrada2.

LeeEntradaAnalogica 1

Muestra por pantalla el valor de la entrada analógica 1.

Estado del robot

La lectura del estado del robot nos permite acceder a información sobre el estado de los micro interruptores, los ejes y los codificadores motores.

LeeSwitch: El comando leeSwitch permite conocer el estado de cualquiera de los 6 micro interruptores del robot bien mostrándolo por pantalla o asignándolo a una variable.

LeeSwitch *[0-5]*

LS *[0-5]*

Ejemplos:

valorswitch = LeeSwitch 3

valorswitch almacenará el valor del estado del micro interruptor 3.

LeeSwitch 1

Se mostrará por pantalla el valor del micro interruptor 1.

LeeMotor: Nos permitirá conocer el valor del codificador motor indicado.

LeeMotor *[1-6]*

LM *[1-6]*

Ejemplos:

motor1 = LeeMotor 1

Almacena el valor del codificador del motor 1 en la variable motor1.

LeeMotor 4

Muestra por pantalla el valor del codificador del motor 4.

LeeEje: Informa sobre el ángulo de giro actual del eje con respecto a su posición de Home.

LeeEje *eje*

LE *eje*

Ejemplos:

eje3 = LeeEje 3

Almacena en la variable eje3 el valor del giro del eje 3 respecto a Home.

LeeEje 1

Muestra por pantalla el valor del giro del eje 1 respecto a Home.

EstadoRobot: Muestra por pantalla un resumen del estado del robot que incluye el estado de sus microinterruptores, pasos de motor en los codificadores y ángulos de giro de los ejes.

EstadoRobot

ER

Salida a consola y aviso del controlador

Salida a consola

Escribir: Muestra por pantalla ristras definidas en el propio script como valores de variables que hayamos declarado.

Escribir *"ristra"|variable*

Ejemplos:

Escribir "Soy una ristra!"

Simplemente nos mostrará por pantalla la ristra "Soy una ristra!".

motor1=LeeMotor 1

...

Escribir "El valor del codificador del motor 1 era " + motor1

Nos mostrará por pantalla la ristra y luego el valor del motor1 en el momento en que se asignó a la variable. Supongamos que el motor 1 tenía un valor de 500 en el momento de la asignación, la salida sería "El valor del codificador del motor 1 era 500"

En este caso se podría conseguir también el funcionamiento equivalente para el valor actual a través de las siguientes llamadas:

Escribir “El valor del codificador del motor 1 es ahora mismo ”

LeeMotor 1

Esto generaría la salida con el valor actual del codificador ya que las llamadas a las funciones que nos proporcionan datos internos del robot, cuando no tienen asignación, simplemente se muestran por pantalla.

Aviso del controlador

Timbre: Hace sonar el timbre del controlador.

Timbre

Conector PHP / Intérprete

La posibilidad de conectar un servidor web que nos proporcionará la interfaz gráfica para manejar el robot y nuestro intérprete de comandos nos la da el conector PHP. El conector se encuentra asociado a un socket del servidor PHP y se comunican a través de paquetes que son retransmitidos del servidor web al intérprete.

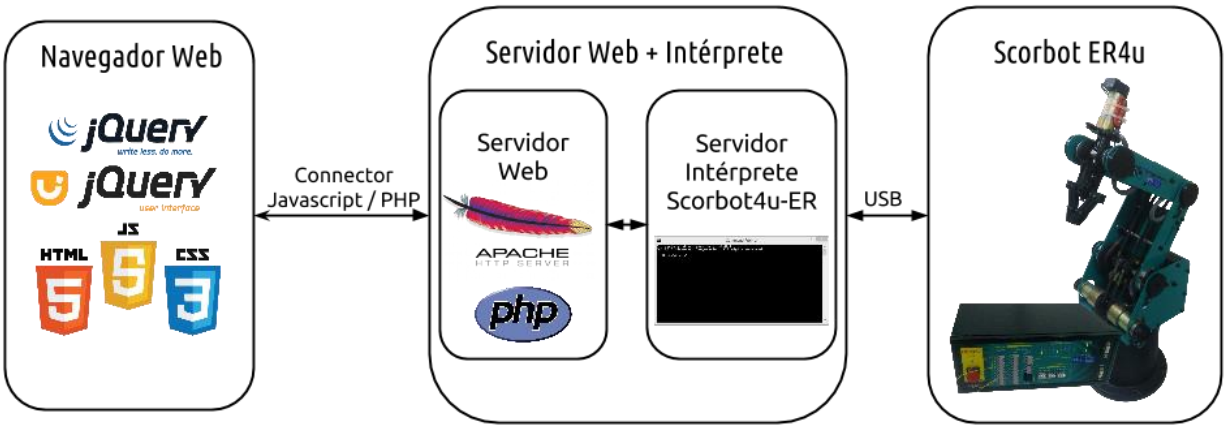


FIGURA 62 - CONEXIÓN CONTROLADOR/SERVIDOR

Como vemos en estos fragmentos de código la conexión se realiza a través de un 'socket' lo que nos permite que el servidor y el ejecutable puedan comunicarse.

```
<?php
    $info = $_GET["command"];
    $socket = socket_create(AF_INET,SOCK_STREAM,SOL_TCP);
    socket_connect($socket,"127.0.0.1",37011);
    socket_send($socket,$info,100,MSG_OOB);
    $answer = socket_read($socket,100);
    echo $answer;
?>
```

Este archivo PHP se encarga de leer el mensaje, comunicarse mediante un socket con el intérprete que se está ejecutando en modo servidor y una vez envíado el mismo esperar y devolvernos una respuesta.

Mediante el uso de Javascript la interfaz web es capaz de comunicarse con el módulo PHP tanto para enviar como para recibir la respuesta como podemos ver en el siguiente fragmento de código de la función **`sendMessage(comando)`** del archivo communications.js.

```
...
xmlhttp=new XMLHttpRequest();
var msg = "scorbotConnector.php?command="+comando.toUpperCase()+".";
xmlhttp.open("GET",msg,false);
xmlhttp.send();
var resp = xmlhttp.responseText;
...
```

Resumiendo la parte relevante vemos cómo se crea la solicitud y pre procesa el mensaje pasándolo a mayúsculas y se envía al conector PHP para posteriormente esperar a la respuesta.

Una ventaja de utilizar la interfaz y el intérprete a través de un conector es que éste puede filtrar e incluso pre procesar los mensajes de manera que se podrían desarrollar nuevas interfaces como acceso a través de dispositivo móvil o una tableta de control física y nos bastaría con enlazarla a nuestro conector para poder trabajar con el intérprete, al que se le retransmitirían una vez procesadas, las órdenes.

Interfaz gráfica HTML5 + Javascript + JQuery UI

Disponemos de una interfaz gráfica que nos proporciona la posibilidad de trabajar con el brazo robótico de manera más cómoda e intuitiva. La interfaz se ha desarrollado utilizando la librería Jquery UI para los elementos gráficos y Javascript para el resto de la programación web.

Las diferentes ventanas y diálogos se han desarrollado como archivos HTML independientes y el aspecto final se ha aplicado a través de un estilo CSS.

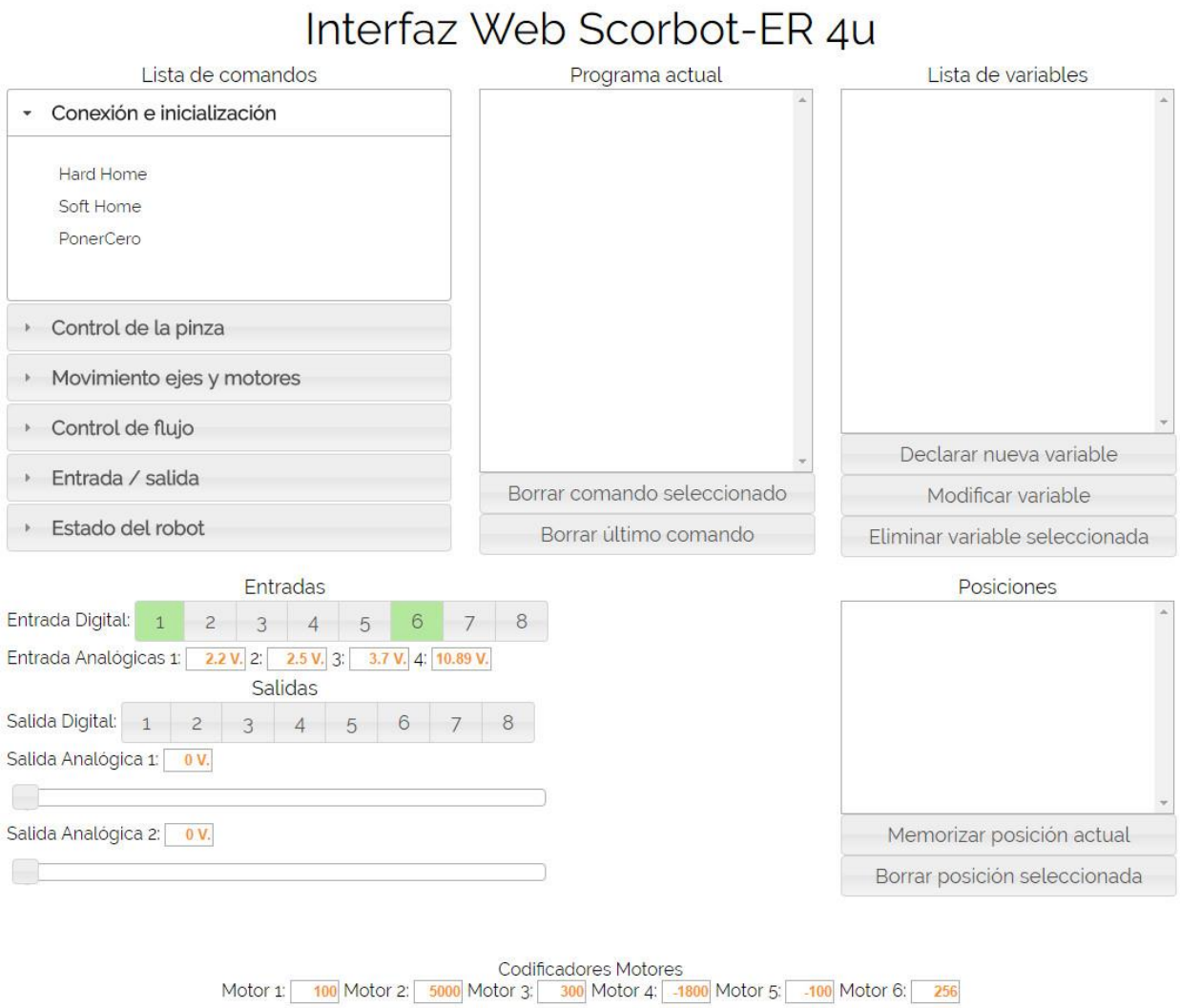


FIGURA 63 - INTERFAZ COMPLETA

Toda la interfaz está compuesta por 21 archivos HTML, 8 archivos .JS, la librería Jquery UI y un archivo de estilo CSS. Se ha hospedado todo el contenido en un servidor Apache local que incluye PHP.

Una parte de la interfaz nos informa del estado actual de las entradas y nos permite modificar las salidas tanto digitales como analógicas:

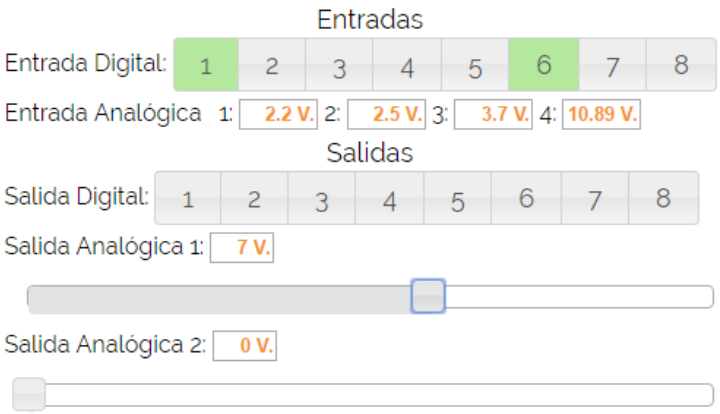


FIGURA 64 - E/S DIGITALES Y ANALÓGICAS.

La interfaz también nos permite conocer en cualquier momento la posición actual del robot:

FIGURA 65 - VALORES CODIFICADORES MOTORES.

También tenemos la posibilidad de almacenar estos valores en las diferentes posiciones que podemos añadir a nuestra lista:

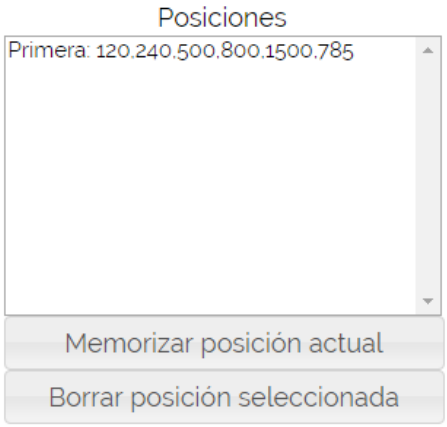


FIGURA 66 - LISTA DE POSICIONES.

La mayoría de comandos disponen de diálogos para hacer el trabajo más sencillo y corregir posibles errores antes de enviarlos a la lista que compone el programa actual. Esto nos da la ventaja de que prácticamente no se van a enviar mensajes erróneos o con valores no válidos al intérprete por lo que supone además una capa extra de protección frente al uso incorrecto del dispositivo.

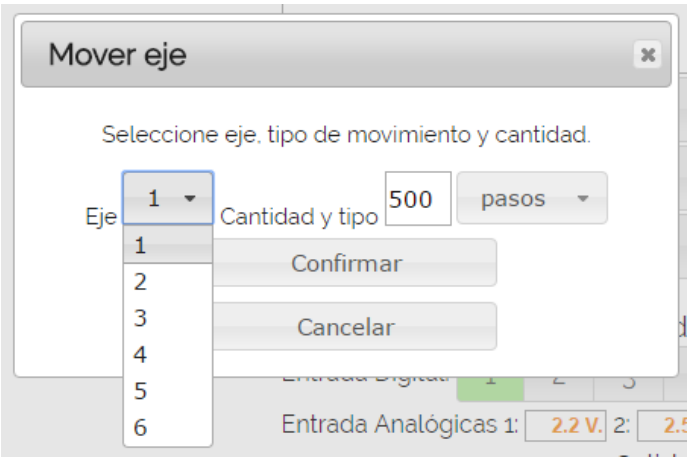


FIGURA 67 - DIÁLOGO PARA MOVIMIENTO DE UN EJE.

El movimiento directo de los ejes que nos proporcionaba Scorbase también se encuentra presente en nuestra interfaz a través de un diálogo:

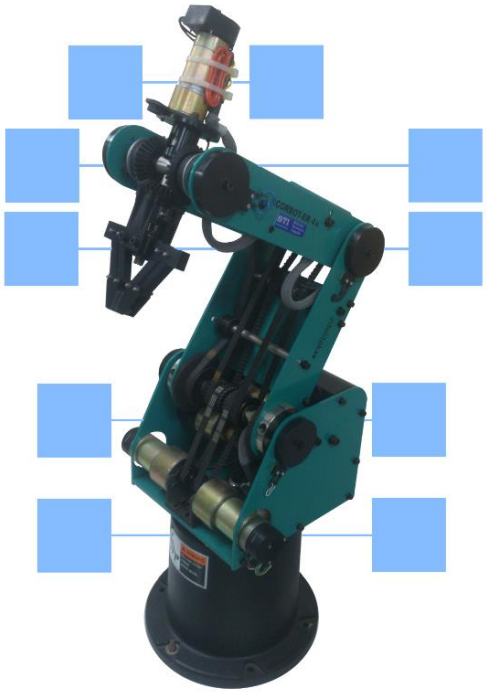


FIGURA 68 - CONTROL DIRECTO DE LOS EJES.

El intérprete servidor

El intérprete que se ejecuta en el servidor para escuchar mensajes del conector PHP es un ejecutable que se comunicará a través de un socket con el servidor PHP Apache que le retransmitirá los mensajes.

Una vez parseado el mensaje se llamará al intérprete de manera que ya sea éste el encargado de utilizar el mensaje recibido como vea conveniente, llamando a una función, desechándolo si no es válido, etc…

Vamos a ver cómo funciona el servidor:

```
int main(void)
{

  …INICIALIZACIÓN Y CONTROL-ON CON EL ROBOT…

  //Variables asociadas al servidor.
  WSADATA wsaData;
  int iResult;
  SOCKET ListenSocket = INVALID_SOCKET;
  SOCKET ClientSocket = INVALID_SOCKET;
  struct addrinfo *result = NULL;
  struct addrinfo hints;
  int iSendResult;

  //Búfer para datos recibidos.
  char recvbuf[DEFAULT_BUFLEN];
  memset(recvbuf,0,sizeof(recvbuf));
  int recvbuflen = DEFAULT_BUFLEN;

  //Inicialización del servidor.
  iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
  if (iResult != 0) {
     printf("WSAStartup failed with error: %d\n", iResult);
     return 1;
  }

  ZeroMemory(&hints, sizeof(hints));
  hints.ai_family = AF_INET;
  hints.ai_socktype = SOCK_STREAM;
  hints.ai_protocol = IPPROTO_TCP;
  hints.ai_flags = AI_PASSIVE;

  //Definimos el puerto que vamos a utilizar.
  iResult = getaddrinfo(NULL, DEFAULT_PORT, &hints, &result);
  if ( iResult != 0 ) {
     printf("getaddrinfo failed with error: %d\n", iResult);
     WSACleanup();
     return 1;
```

```
}

//Escuchamos en un socket.
ListenSocket = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
if (ListenSocket == INVALID_SOCKET) {
   printf("socket failed with error: %ld\n", WSAGetLastError());
   freeaddrinfo(result);
   WSACleanup();
   return 1;
}

//Terminamos de preparar el socket para recibir conexiones.
iResult = bind( ListenSocket, result->ai_addr, (int)result->ai_addrlen);
if (iResult == SOCKET_ERROR) {
   printf("bind failed with error: %d\n", WSAGetLastError());
   freeaddrinfo(result);
   closesocket(ListenSocket);
   WSACleanup();
   return 1;
}

freeaddrinfo(result);

//Escuchamos en el socket.
iResult = listen(ListenSocket, SOMAXCONN);
if (iResult == SOCKET_ERROR) {
   printf("listen failed with error: %d\n", WSAGetLastError());
   closesocket(ListenSocket);
   WSACleanup();
   return 1;
}
…
if(scorbotConnect){
        printf("Conexion al controlador Scorbot OK!\n");
}else{
        //Si falla la conexión al controlador USB cerramos el servidor.
        printf("Error al intentar conectar al controlador!\n");
        closesocket(ListenSocket);
        WSACleanup();
        return 1;
}
//Empezamos a escuchar la conexión en el socket para el comando que se nos envíe.
 do {
                    // Aceptamos el cliente.
                    ClientSocket = accept(ListenSocket, NULL, NULL);
                    if (ClientSocket == INVALID_SOCKET) {
                              printf("accept failed with error: %d\n", WSAGetLastError());
                              closesocket(ListenSocket);
                              WSACleanup();
                              return 1;
                    }
                iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);
                if (iResult > 0) {
                      char *comando = strtok(recvbuf," ");
                      char *parametros = strtok(NULL,"");
                     char *resultado;
```

```
            int res  = ejecutaComando(comando,parametros,resultado);
            memset(recvbuf,0,sizeof(recvbuf));
            printf("Response size: %d",respL);
             //Envíamos respuesta al cliente.
            iSendResult = send( ClientSocket, resp, respL, 0 );
            if (iSendResult == SOCKET_ERROR) {
               printf("Error envío: %d\n", WSAGetLastError());
               closesocket(ClientSocket);
               WSACleanup();
               return 1;
            }
         }
         else if (iResult == 0)
            printf("Cerrando conexión...\n");
         else  {
            printf("Transm. error: %d\n", WSAGetLastError());
            closesocket(ClientSocket);
            WSACleanup();
            return 1;
         }
      } while (iResult > 0);

      // Cierre de conexión y limpieza de recursos.
      iResult = shutdown(ClientSocket, SD_SEND);
      if (iResult == SOCKET_ERROR) {
         printf("Error al cerrar: %d\n", WSAGetLastError());
         closesocket(ClientSocket);
         WSACleanup();
         return 1;
      }
      closesocket(ClientSocket);
      WSACleanup();
      return 0;
   }
```

Como vemos gran parte del programa se encarga de montar el servidor tras conectar con el controlador, una vez abierto el socket nos metemos en un bucle desde el cual empezamos a parsear los mensajes que nos llegan en conjuntos de comandos y parámetros, además de pasar una variable resultado para el mismo.

Con este servidor y el conector PHP se ha solucionado la comunicación a través de la interfaz web.

5.6 Pruebas realizadas

Esta parte del documento está dedicada a las diferentes pruebas que se fueron realizando en forma de pequeños subprogramas con el fin de comprobar el correcto comportamiento de las funciones durante su desarrollo.

Entrada / Salida Digital

Para comprobar la salida digital se han activado y desactivado varios puertos como podemos ver en la siguiente sección de código:

```
…
printf("Salida digital a activar: ");
scanf("%d",&i);
activaSalidaDigital(i);
…
```

En todos los casos se han activado correctamente cada una de las salidas digitales. Como era de esperar, al ordenar que se active una salida ya activa no se produce ninguna variación.

La entrada digital se ha probado activando los puertos mediante el uso de un cable conectado al puerto de tierra que emula lo que sería una entrada digital como pudimos ver durante el estudio de transferencia de la entrada/salida digital.

El código utilizado para leer los puertos de entrada ha sido el siguiente:

```
…
printf("\nEstado actual de la entrada digital:\n");
for(int e=1;e<9;e++){
   printf("Entrada %d = %d\n",e,leeEntradaDigital(e));
}
…
```

Todas las pruebas fueron satisfactorias al leer los puertos de entrada, tanto para pruebas con una única entrada activa como cuando se han activado varias a la vez.

Entrada / Salida Analógica

La entrada analógica se prueba enviando una serie de tensiones a los puertos utilizando una fuente variable de las que disponemos en el laboratorio y verificando que los valores leídos son los indicados por la misma.

El único parámetro necesario es el que indica la entrada que vamos a leer:

```
…
printf(“Seleccione entrada analógica: ”);
scanf("%d",&i);
leeEntradaAnalogica(i);
printf("Tensión en el puerto %d = %f",i,voltaje);
}
…
```

Durante las pruebas la entrada analógica en todos los casos ha coincidido con la indicada por la fuente.

La salida analógica se ha probado indicando el voltaje de salida a aplicar en cada uno de los puertos y realizando mediciones de las mismas con un voltímetro:

```
…
printf(“\nPrueba de salida analógica\n”);
scanf("%d",&i);
float voltaje;
printf("Tensión a aplicar en la salida: ");
scanf("%f",&voltaje);
activaSalidaAnalogica(voltaje);
…
```

Todas las pruebas realizadas nos han activado la salida con el valor correspondiente. Para los voltajes indicados por parámetro que estén fuera de rango se indica al usuario por pantalla y la salida no varía.

Apertura / Cierre de pinza

Se ha decidido probar como primer paso al movimiento de los motores el funcionamiento de la pinza ya que se puede probar fácilmente para valores límites en los extremos de funcionamiento cuando la pinza se encuentra abierta o cerrada.

El siguiente fragmento de código nos ha permitido probar tanto el cierre como la apertura de la pinza:

```
…
int i;
scanf("%d",&i);
if(i==1)abrePinza();
if(i==2)cierraPinza();
}
…
```

Una vez comprobado el funcionamiento del cierre de la pinza hemos probado la apertura de la misma realizando todas las pruebas varias veces de manera satisfactoria. También se ha probado ordenar el cierre de la pinza con la pinza ya cerrada y una apertura con la pinza abierta, en ambos casos no ha habido problemas ya que la función detecta la colisión del eje y retoma el control del robot automáticamente.

En ambos casos la pinza se ha comportado igual que cuando se da esta orden con el programa original Scorbase.

Estado del robot

Para acceder al estado del robot donde se incluye el estado de los microswitches y los encoders tenemos a nuestra disposición una serie de funciones encargadas de leer del búfer de entrada del controlador el estado de los mismos.

Vamos a hacer unas pruebas simples donde realizaremos una lectura del estado tras Home y otras donde una vez colocado el robot en una posición por Scorbase y visto el valor para los codificadores que nos indica realizaremos luego una lectura con nuestro programa con el fin de comprobar si la lectura de los codificadores se realiza de forma correcta.

Si nos fijamos en el estado tras Home vemos que coincide con los valores que indica Scorbase, es decir, todos los codificadores han sido reseteados a cero:

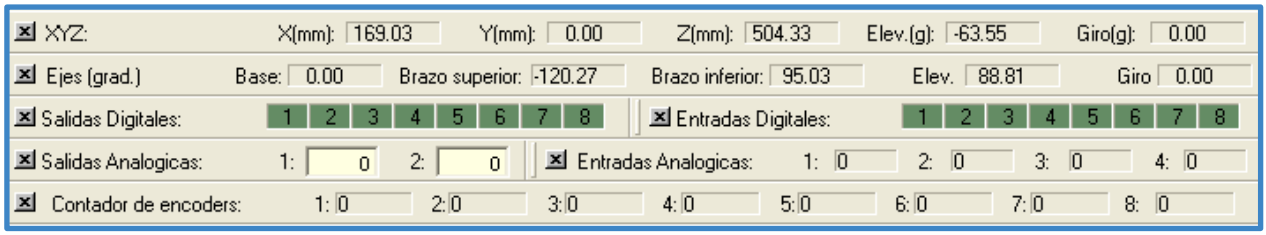


FIGURA 69 - ESTADO DEL ROBOT TRAS HOME.

En este otro caso vamos a irnos a una posición aleatoria donde tengamos valores tanto negativos como positivos para los codificadores.

FIGURA 70 - CAPTURA DE POSICIÓN EN SCORBASE.

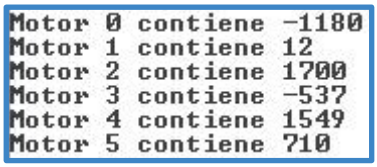


FIGURA 71 - CAPTURA EN NUESTRA PRUEBA.

Vemos que los valores de codificador que muestran Scorbase y nuestra función coinciden por lo que esta función está funcionando correctamente.

Ahora vamos a probar el estado de los microswitches activando primero uno y posteriormente dos para comprobar que la lectura funciona en ambos casos.

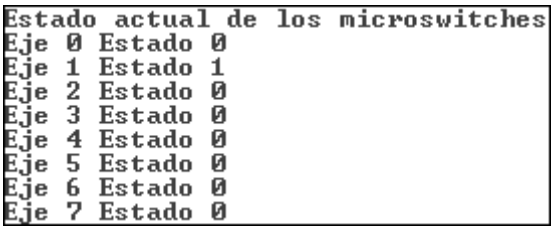


FIGURA 72 - ACTIVACIÓN DEL MICROSWITCH CORRESPONDIENTE AL HOMBRO DEL ROBOT.

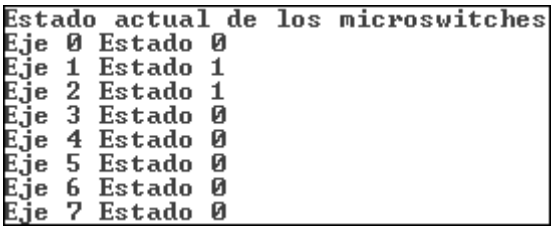


FIGURA 73 - ACTIVACIÓN DE MICROSWITCHES CORRESPONDIENTES A HOMBRO Y CODO DEL ROBOT.

Como vemos tanto para uno como para cualquier otra combinación de microswitches se realiza una lectura correcta por parte de nuestra función.

Movimiento de motores

Una prueba importante a la hora de controlar el robot es evaluar que el movimiento de los motores es análogo al que nos proporcionan el driver y programa originales.

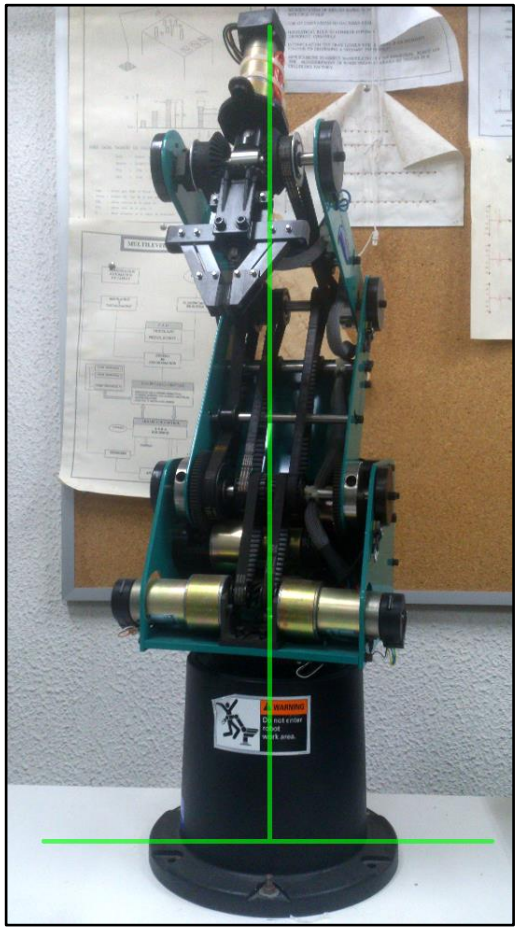
Para ello vamos a realizar varias pruebas, una vez superadas todas podremos decir que el driver funciona correctamente.

Se han utilizado 3 movimientos para poder evaluar el comportamiento de las funciones del driver en 3 casos diferentes, el primero donde solo se mueve un eje, el segundo donde trabajaremos con el eje correspondiente a la inclinación y roll de la pinza y un tercero en el que moveremos todos los ejes a la vez.

Para la primera prueba ni siquiera es necesario realizar un movimiento o lectura previa, ya sabemos que los codificadores los estamos leyendo correctamente como vimos en las pruebas anteriores por lo que ahora nos limitaremos a mover el primer eje, la cintura del brazo robótico, 1500 pasos en dirección positiva.

```
Interfaz Scorbot ER-4U

1. Buscar Home
2. Mover Eje
3. Operar Pinza
4. E/S Digital
5. E/S Analogica
6. Leer Motores
7. Memorizar/ver Posiciones
8. Ir a Posicion
9. Salir

Seleccion: 2

1. Pasos de motor
2. Grados
3. Cancelar

Seleccion: 1

Eje: 0

Pasos de motor: 1500
.........................
Interfaz Scorbot ER-4U

1. Buscar Home
2. Mover Eje
3. Operar Pinza
4. E/S Digital
5. E/S Analogica
6. Leer Motores
7. Memorizar/ver Posiciones
8. Ir a Posicion
9. Salir

Seleccion: 6

1. Indicar Motor
2. Leer Todos
3. Cancelar

Seleccion: 2

Encoder motor 0: 1500
Encoder motor 1: 0
Encoder motor 2: 0
Encoder motor 3: 0
```

Como vemos se ha realizado el movimiento y la lectura es correcta.



Para la segunda serie de pruebas vamos a realizar movimientos para el roll y la inclinación de la pinza que se diferencian de los demás es que necesitan que se muevan dos motores para que el movimiento se realice correctamente.

En este caso vamos a ver el ejemplo de un movimiento de roll de 750 unidades.

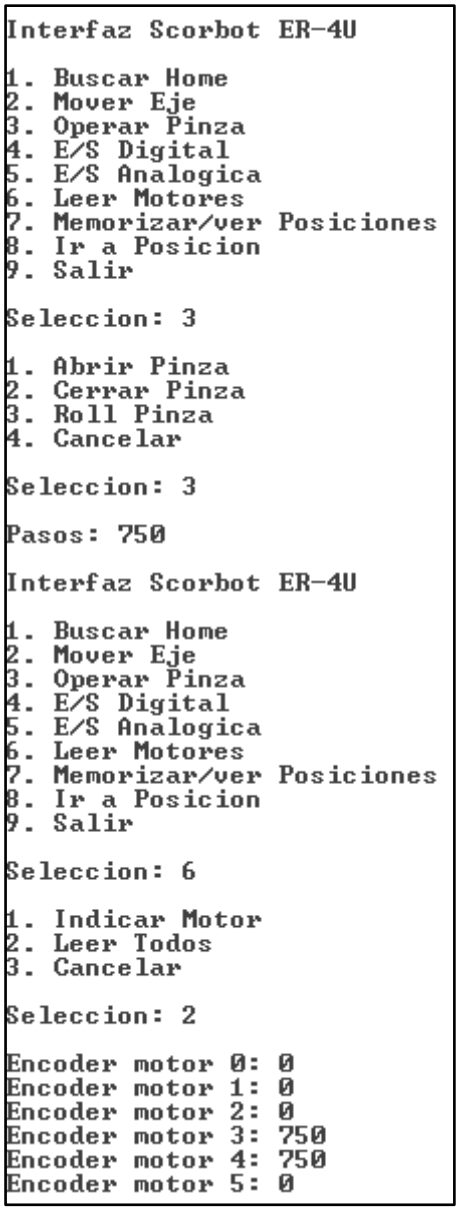


FIGURA 74 - MOVIMIENTO ROLL.

Podemos comprobar que el roll de la pinza funciona correctamente y no se han producido colisiones ya que los motores realizan el movimiento de forma sincronizada.

De la misma forma vamos a realizar una serie de pruebas para la inclinación de la misma, nos interesa especialmente ya que en este caso los motores deben moverse en direcciones opuestas.

Vamos a ver a continuación por lo tanto como el controlador se encarga correctamente del movimiento de inclinación y no se producen colisiones o sobreesfuerzos en los motores aun teniendo en cuenta que rotan en direcciones diferentes. Esto se debe a que los incrementos o decrementos en los contadores se producen de forma simultánea y en la misma cantidad.

Durante la última prueba donde movemos todos los motores para llegar a una posición determinada vamos a arrancar desde la posición de Home del robot y llegar a una posición donde todos los codificadores motores hayan tenido que cambiar.

Como vemos en la demostración partimos de una posición aleatoria, con codificadores tanto positivos como negativos y queremos llegar a una posición concreta que sí conocemos, la función nos moverá al punto indicado y nos informará de cuántos pasos se ha movido cada eje.

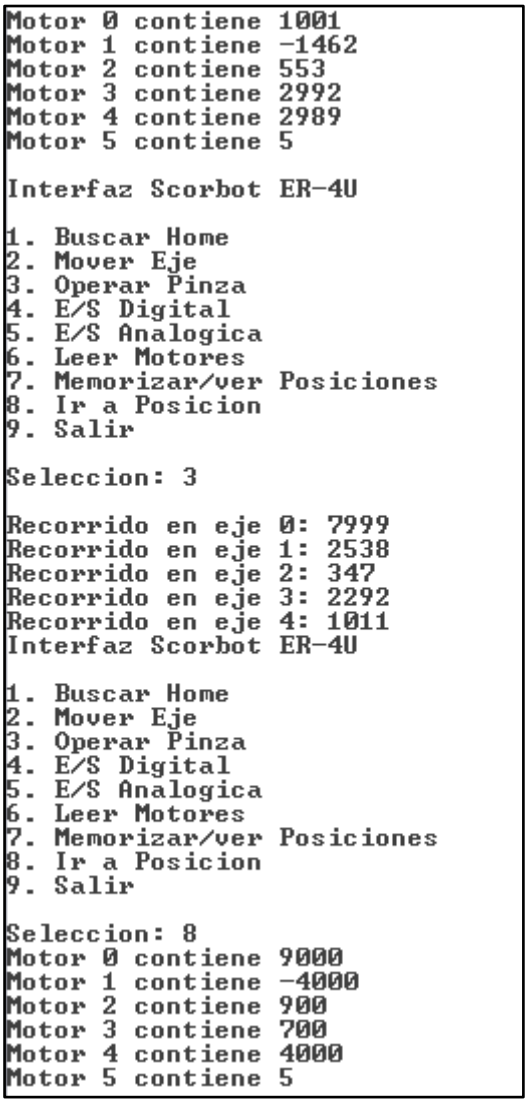


FIGURA 75 - MOVIMIENTO CORRECTO.

Como vemos se ha realizado el movimiento y se nos ha indicado correctamente el número de pasos que se han movido los contadores de cada motor.

Finalmente se debe probar que cuando queremos realizar el movimiento en un único motor o eje éste no cause movimientos indeseados en ninguno de los otros motores, para ello se han realizado una serie de movimientos utilizando un único motor y se ha comprobado que el controlador es capaz de mantener la posición sin variar en los motores que no realizan movimiento.

Como ejemplo podemos ver el siguiente movimiento donde movemos el eje de la cintura, ya que de él derivan todos los demás y por lo tanto es el que más posibilidades tiene de causar inestabilidad en alguno de los ejes. Tenemos una posición inicial, en este caso negativa para el eje de la cintura y los demás codificadores están en posiciones de 0, -1 (Se ha evitado resetear los contadores para así tener dos valores de referencia y no únicamente los valores 0 tras resetear el contador en función HOME). El movimiento será de 10.000 unidades en el eje de la cintura.

```
Motor 0 contiene -1418
Motor 1 contiene -1
Motor 2 contiene 0
Motor 3 contiene -1
Motor 4 contiene 0
Motor 5 contiene 1

Interfaz Scorbot ER-4U

1. Buscar Home
2. Mover Eje
3. Operar Pinza
4. E/S Digital
5. E/S Analogica
6. Leer Motores
7. Memorizar/ver Posiciones
8. Ir a Posicion
9. Salir

Seleccion: 4

Eje 0 termino su movimiento.
Recorrido en eje 0: 10000
Recorrido en eje 1: 0
Recorrido en eje 2: 0
Recorrido en eje 3: 0
Recorrido en eje 4: 0
Interfaz Scorbot ER-4U

1. Buscar Home
2. Mover Eje
3. Operar Pinza
4. E/S Digital
5. E/S Analogica
6. Leer Motores
7. Memorizar/ver Posiciones
8. Ir a Posicion
9. Salir

Seleccion: 8
Motor 0 contiene -11418
Motor 1 contiene -1
Motor 2 contiene 0
Motor 3 contiene -1
Motor 4 contiene 0
Motor 5 contiene 1
```

FIGURA 76 - MOVIMIENTO CINTURA.

Como podemos ver, se realiza el movimiento de manera correcta y llegamos al valor exacto esperado. Además ninguno de los otros contadores se ha visto afectado.

una vez realizadas todas las pruebas que nos confirman que el controlador realiza correctamente todas las funciones básicas se puede afirmar que el driver funciona.

5.7 Trabajos futuros

Algunas mejoras que se podrían realizar en el futuro son las siguientes:

- Incorporación de una cámara web al servidor con el fin de permitir el uso del robot desde entornos remotos.

- Creación de una tableta física(Arduino, etc...) con la que operar con el robot haciendo uso de la misma y el driver creado.

- Crear un driver para otro brazo robótico (respetando los nombres de las funciones) que permita utilizar el intérprete o interfaz web en otro entorno.

6. Conclusiones

El objetivo de lograr desarrollar un controlador para el robot que cubriese todas las funcionalidades básicas que también proporciona Scorbase ha sido satisfecho y además hemos logrado dotar al mismo de un entorno web que permite el uso desde otro programa que no sea el original.

Durante las pruebas se ha verificado el funcionamiento por separado de cada una de las funciones desarrolladas con el fin de ver si cumplían su objetivo como mínimo igual de bien que al ser llamadas por Scorbase. Todas las funciones han cumplido con su cometido y no son distinguibles de sus análogas en Scorbase.

La creación de un entorno web ha permitido demostrar como la comunicación no se ha extendido únicamente a tener una librería que poder utilizar en C/C++ sino de poder interactuar con el robot en un entorno tan remoto como puede ser un navegador web que acceda al servidor donde se encuentra en ejecución un intérprete remoto a la espera de recibir los mensajes.

Finalmente, uno de los puntos más importantes a tener en cuenta es que se ha desarrollado el driver a partir de la observación del tráfico en el bus USB y no accediendo al código original ni desensamblando ningún archivo del Scorbot-ER 4u proporcionado por Intellitek. Aunque este método de ingeniería inversa es arriesgado ya que podríamos habernos encontrado con mensajes codificados de manera más complicada o algún tipo de autentificación por parte del controlador, este no ha sido el caso, y aunque algunas funciones han sido más complicadas que otras, sobre todo las relativas al movimiento y mantenimiento de la conexión, se ha conseguido un driver aceptable y funcional.

7. Anexos

A continuación se incluyen los siguientes anexos a este documento:

- Instalación
  - Configuración como filtro.
  - Instalación Standalone.

- Contenido del CD.
- Ejemplo función: abrePinza().
- Listado de funciones de la librería.
- Listado de comandos para el intérprete.

7.1 Instalación

La librería elegida para comunicarnos utilizando el protocolo USB ha sido 'LibUsb' ya que vamos a desarrollar el driver utilizando sistema operativo Windows XP donde disponemos de la instalación tanto de el driver original como el programa Scorbase necesarios durante el estudio de los comandos que se envían al controlador. La librería libusb proporciona la instalación de drivers filtro que permiten interactuar con el dispositivo de manera transparente.

Esto supone una gran ventaja a la hora de trabajar con el controlador ya que nos da la posibilidad de trabajar con el software original "Scorbase" utilizando su driver propietario mientras a través del filtro tenemos la posibilidad de inyectar y leer mensajes utilizando nuestra aplicación.

El driver está implementado como una biblioteca de enlace dinámico (DLL) y se instalará en la carpeta "win32" a través de su programa de instalación.

Configuración utilizando el filtro

Una vez instalado el driver debemos indicar al mismo qué dispositivo es el que vamos a filtrar. Para ello se nos proporciona una aplicación que nos mostrará un desplegable con los dispositivos conectados actualmente y que nos permite tanto la instalación como la eliminación del driver.

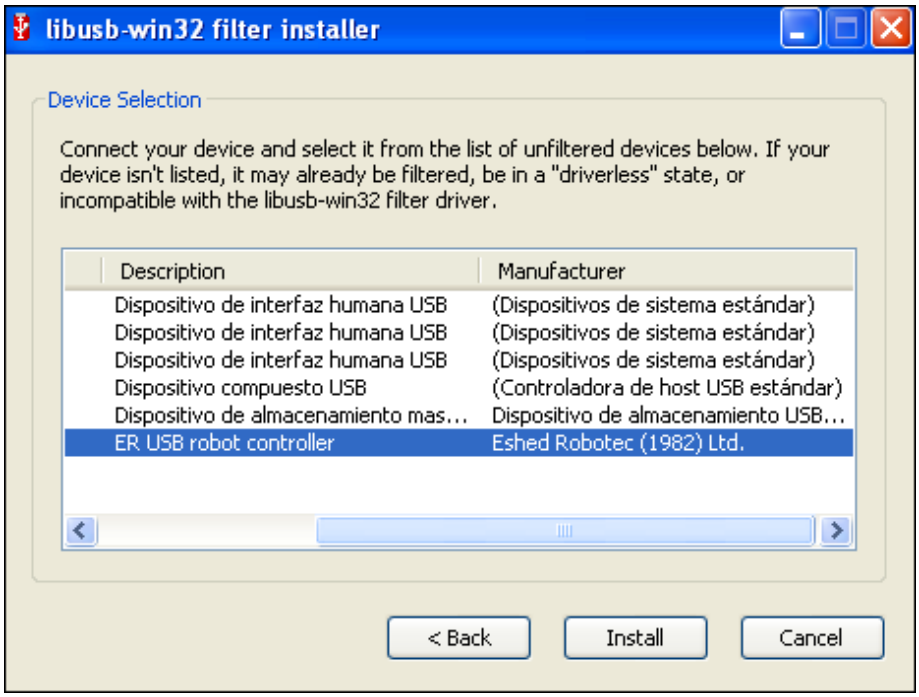


FIGURA 77 - CONFIGURACIÓN FILTRO

Si el filtro ha sido configurado correctamente podremos ver el siguiente mensaje informándonos sobre su correcta instalación. Adicionalmente podemos obtener información detallada sobre el dispositivo filtrado

utilizando una de las aplicaciones proporcionadas por 'LibUsb' llamada 'TestLibUsb' que nos muestra todos los dispositivos que están siendo filtrados actualmente y datos sobre los mismos.

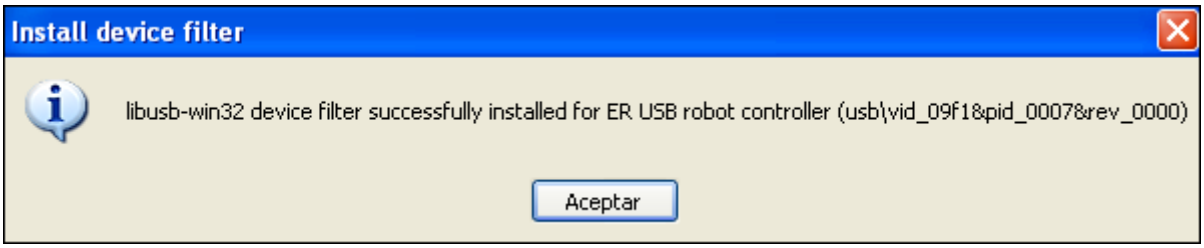


FIGURA 78 - DISPOSITIVO FILTRADO.

Salida del programa TestLibUsb

```
DLL version:   1.2.6.0

Driver version:       1.2.6.0

bus/device  idVendor/idProduct

bus-0/\\.\libusb0-0001--0x09f1-0x0007
09F1/0007

- Unable to fetch manufacturer string

bLength:             18

bDescriptorType:     01h

bcdUSB:              0100h

bDeviceClass:        00h

bDeviceSubClass:     00h

bDeviceProtocol:     00h

bMaxPacketSize0:     08h

idVendor:            09F1h

idProduct:           0007h

bcdDevice:           0000h

iManufacturer:       1

iProduct:            0

iSerialNumber:       0

bNumConfigurations:  1

  wTotalLength:         41

  bNumInterfaces:       1

  bConfigurationValue:  1

  iConfiguration:       140

  bmAttributes:         40h

  MaxPower:             50
```

```
bInterfaceNumber:   0

    bAlternateSetting:  0

    bNumEndpoints:      2

    bInterfaceClass:    255

    bInterfaceSubClass: 0

    bInterfaceProtocol: 0

    iInterface:         186

      bEndpointAddress: 83h

      bmAttributes:     02h

      wMaxPacketSize:   64

      bInterval:        1

      bRefresh:         0

      bSynchAddress:    0

      bEndpointAddress: 02h

      bmAttributes:     02h

      wMaxPacketSize:   64

      bInterval:        1

      bRefresh:         0

      bSynchAddress:    0
```

Vemos que en nuestro caso el dispositivo es detectado y además se nos proporciona una serie de detalles sobre el mismo como los endpoints, su configuración y su número de interfaz.

Instalación Standalone

Para instalar nuestro driver sin tener presente en el equipo el driver original de Scorbase debemos utilizar uno de los dos instaladores generados por libUSB o directamente utilizar el archivo .inf también creado con este fin.

El archivo .inf contiene toda la información relativa al dispositivo y se instala como cualquier otro driver con la única diferencia de que al ser un driver de desarrollo propio no dispone de firma y puede causar algún problema al instalar en sistemas operativos como Windows 8 en adelante donde para instalar drivers sin firmar debemos entrar en el equipo en modo seguro.

A continuación tenemos parte de la información del archivo .inf del driver, que como vemos apunta a que el driver del dispositivo será libusb, por lo que ya no es necesario el filtro el driver original:

```
; Scorbot_ER-4u.inf

; Copyright (c) 2010 libusb-win32 (GNU LGPL)

[Strings]

DeviceName = "Scorbot ER-4u"

VendorName = "Intelitek"

SourceName = "Scorbot ER-4u Install Disk"

DeviceID   = "VID_09F1&PID_0007"

DeviceGUID = "{E8B13CEA-3CB3-436B-A20C-12C55F8962D4}"


[Version]

Signature   = "$Windows NT$"

Class       = "libusb-win32 devices"

ClassGuid   = {EB781AAF-9C70-4523-A5DF-642A87ECA567}

Provider    = "libusb-win32"

CatalogFile = Scorbot_ER-4u.cat

DriverVer   = 01/17/2012, 1.2.6.0
```

```
[ClassInstall32]

Addreg = libusb_class_install_add_reg

[libusb_class_install_add_reg]

HKR,,,0,"libusb-win32 devices"

HKR,,Icon,,-20

[Manufacturer]

%VendorName% = Devices, NT, NTAMD64, NTIA64
```

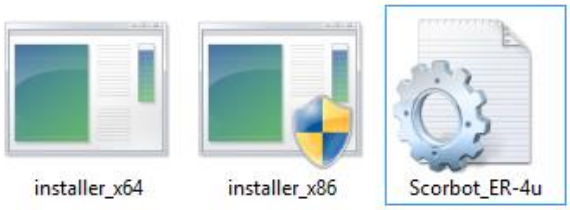


FIGURA 79 - INSTALADORES DEL DRIVER PROPORCIONADOS.

7.2 Contenido del CD

El código fuente de todas las aplicaciones desarrolladas se encuentra en las siguientes carpetas del archivo proporcionado, codigo.7z:

- Driver (driver.c, driver.h)
- InterfazTerminal (main.c)
- Interprete/lib (interprete.h, interprete.c)
- Interprete/InterpreteLocal (intLocal.c)
- Interprete/InterpreteServidor (intServ.c)
- InterfazGrafica (Archivos HTML, JS y CSS)
- ConectorWeb (conectorWeb.php)

Además se han incluido muestras de las capturas utilizadas durante el análisis de los mensajes USB entre Scorbase y el robot así como durante las pruebas de éste driver en la carpeta capturas.

7.3 Función ejemplo: abrePinza( )

A continuación se incluye el código de la función abrePinza(), podemos ver como a su vez hace uso de una función llamada grip(cierra) donde pasamos como parámetro si queremos cerrar la pinza, si es falso se entiende que queremos realizar una apertura.

Esta optimización se hizo con el fin de no tener que escribir dos funciones tan parecidas para la apertura y el cierre cuando podíamos usar un condicional para saber si el codificador debe aumentar o disminuir.

```
//Abre la pinza.
int abrePinza(){
   grip(false);
   codes();
}

//Abre o cierra dependiendo del parámetro.
int grip(bool cierra){
   usb_bulk_read(handleRobot, 0x83, temp, 64, 1000);
   enviaOpCode(0x47,0,0);
   enviaOpCode(0x4f,0x20,0x53);
   enviaOpCode(0x4c,0x20,0x00);
   enviaOpCode(0x42,0x20,0x01);
   int posActual = leeMotor(5);
   int salida = posActual++;
   int v = 1;
   int codMotor = 32;
   int i=0;
   int velocidad=150;
   while(limitSwitch()!=0x20){
      char* hexa;
      hexa = (char*)&salida;
      array[codMotor]=hexa[0];
      array[codMotor+1]=hexa[1];
      printf(".");
      enviaOpCode(0x0d,0,0);
      if(cierra){
            salida -= velocidad;
            if(salida<0){
               salida += 65535;
               array[codMotor+2]=0xff;
               array[codMotor+3]=0xff;
            }
      }else{
            salida += velocidad;
            if(salida>65535){
               salida -= 65535;
               array[codMotor+2]=0;
```

```
            array[codMotor+3]=0;
          }
      }
   }
   enviaOpCode(0x73,0x20,0x00);
   enviaOpCode(0x42,0x20,0x00);
   enviaOpCode(0x0d,0x00,0x00);
   leeEstado();
}
```

En negrita podemos ver los mensajes enviados al controlador. Todas las funciones de la librería hacen uso de esta función con el fin de enviar los códigos, veamosla en detalle:

```
//Envia un mensaje con el código de operación definido por los 3 bytes.
int enviaOpCode(char op1,char op2, char op3){
   array[4]=op1;
   array[5]=op2;
   array[6]=op3;
   usb_bulk_write(handleRobot,0x02,array,64,1000);
   Sleep(Retardo);
   array[0]=array[0]+1;
   usb_bulk_read(handleRobot, 0x83, temp, 64, 1000);
   Sleep(Retardo);
}
```

Se puede ver que su cometido es colocar los parámetros del mensaje en el búfer y enviarlo, una vez esperamos y aumentamos la sincronización en 1 podemos volver a leer el búfer y volcarlo en temp para acceder al estado del controlador.

Funciones de la librería

Las funciones incluidas en la librería desarrollada se encuentran descritas a continuación indicando sus parámetros de entrada y salida.

Comunicación y conexión PC-Controlador

Nombre	conectar
Entrada	usb_dev_handle *h
Salida	int resultado
Descripción	A partir del handle del dispositivo se encarga de iniciar la comunicación creando el hilo de sincronización y enviando los primeros comandos de reconocimiento al controlador.

Nombre	control_on
Entrada	
Salida	int resultado
Descripción	Nos permite tomar el control del controlador de manera que se sincronice el estado interno en software con el del robot. Además envía una serie de comandos de inicialización necesarios para el posterior control.

Nombre	idleThread
Entrada	LPVOID lpParam
Salida	
Descripción	Esta función define el hilo de sincronización y recibe el handle al controlador como parámetro a través de un puntero. Esta función siempre debe ser utilizada al crear el hilo ya que de lo contrario se cortará la conexión al controlador.

Funciones de control

Nombre	buscaHomeRobot
Entrada	
Salida	int resultado
Descripción	Lleva a cabo el Hard Home del robot llamando a buscaHome para cada eje.

Nombre	buscaHome
Entrada	int eje
Salida	int resultado
Descripción	Busca la posición de Hard Home para el eje indicado por parámetro.

Nombre	irSoftHome
Entrada	
Salida	int resultado
Descripción	Lleva al robot a la posición Soft Home definida tras realizar un Hard Home del mismo o por el usuario a través de fijarSoftHome.

Nombre	fijarSoftHome
Entrada	
Salida	int resultado
Descripción	Fija la posición actual del robot como posición Soft Home sobrescribiendo el Soft Home actual.

Nombre	leeSwitches
Entrada	
Salida	int resultado
Descripción	Lee el estado actual de todos los microSwitches y lo coloca en el vector booleano microSwitches.

Nombre	leeSwitch
Entrada	int switch
Salida	bool resultado
Descripción	Lee el estado actual del microSwitch indicado por parámetro.

Nombre	printSwitches
Entrada	
Salida	int resultado
Descripción	Imprime por consola el estado actual de los microSwitch.

Nombre	leeEstado
Entrada	
Salida	int resultado
Descripción	Lee los encoder motores y los almacena en el estado actual del robot.

Nombre	leeMotor
Entrada	int motor
Salida	int resultado
Descripción	Devuelve el valor del encoder para el motor indicado por parámetro.

Nombre	valorMotor
Entrada	int motor
Salida	int resultado
Descripción	Devuelve el valor del encoder del motor teniendo en cuenta el signo y el desbordamiento de los encoders. Mientras que leeMotor es usado internamente por otras funciones, esta es la adecuada para realizar lecturas desde el punto de vista del usuario.

Nombre	limitSwitch
Entrada	
Salida	int resultado
Descripción	Devuelve el eje que ha colisionado o cero en caso de que no exista ninguna colisión.

*Las dos funciones siguientes son utilizadas por la librería y no se recomienda utilizarlas directamente a los usuarios.

Nombre	enviaOpCode
Entrada	char op1, char op2, char op3
Salida	int resultado
Descripción	Envía un código de operación de 3 bytes al controlador. Para ello se encargará de colocar los 3 bytes pasados como parámetro en los bytes 3-6 del búfer de salida.

Nombre	enviaOpCode2
Entrada	char op1, char op2, char op3, char op4, char op5, char op6, char op7, char op8
Salida	int resultado
Descripción	Envía un código de operación de 8 bytes al controlador. Para ello se encargará de colocar los 8 bytes pasados como parámetro en los bytes 3-11 del búfer de salida.

Control de motores

Nombre	abrePinza
Entrada	
Salida	int resultado
Descripción	Abre la pinza del robot.

Nombre	cierraPinza
Entrada	
Salida	int resultado
Descripción	Cierra la pinza del robot.

Nombre	grip
Entrada	bool cierra
Salida	int resultado
Descripción	Cierra la pinza si el parámetro es verdadero y la abre en caso contrario.

Nombre	roll
Entrada	int pasos
Salida	int resultado
Descripción	Realiza un movimiento de roll en el eje de la muñeca del número de pasos indicado por parámetro.

Nombre	rollGrados
Entrada	float grados
Salida	int resultado
Descripción	Realiza un movimiento de roll en el eje de la muñeca del ángulo indicado en el parámetro.

Nombre	moverEje
Entrada	int eje, int pasos
Salida	int resultado
Descripción	Mueve un eje del robot un número de pasos, ambos indicados por parámetro.

Nombre	moverEjeGrados
Entrada	int eje, float grados
Salida	int resultado
Descripción	Mueve un eje del robot el ángulo indicado por parámetro.

Nombre	moverEjes
Entrada	int* vector
Salida	int resultado
Descripción	A partir del vector pasado como parámetro realiza un movimiento igual al número de pasos indicados para cada eje siguiendo el orden de los mismos.

Nombre	moverEjesGrados
Entrada	float* vector
Salida	int resultado
Descripción	Actúa igual que moverEjes pero en este caso el parámetro indicará grados de giro para cada eje.

Funciones de Entrada / Salida

Nombre	activaSalidaDigital
Entrada	int nsalida
Salida	int resultado
Descripción	Activa la salida digital indicada.

Nombre	desactivaSalidaDigital
Entrada	int nsalida
Salida	int resultado
Descripción	Desactiva la salida digital indicada.

Nombre	salidaDigital
Entrada	int salida, bool estado
Salida	int resultado
Descripción	Actualiza el estado de la salida indicada al pasado como segundo parámetro.

Nombre	leeEntradaDigital
Entrada	int nentrada
Salida	bool estadoEntrada
Descripción	Lee la entrada digital indicada y nos devuelve su estado en un booleano.

Nombre	leerEstadoEntradas
Entrada	
Salida	char estadoEntradas
Descripción	El byte devuelto contiene el estado de las 8 entradas digitales y es interpretado a partir del bit menos significativo que se corresponde con la salida digital 1.

Nombre	activaSalidaAnalogica
Entrada	int salida, float voltaje
Salida	int resultado
Descripción	Activa la salida analógica indicada con el voltaje pasado como parámetro.

Nombre	leeEntradaAnalogica
Entrada	int nentrada
Salida	float voltaje
Descripción	Lee el voltaje presente en la entrada indicada por parámetro.

Comandos para el intérprete

Comandos asociados a los ejes

Inicialización

Hard Home	HH, HardHome	Realiza el Hard Home del robot.
Soft Home	SH, SoftHome	Mueve el robot a posición de Soft Home.
Poner eje a cero	PonerCero motor	Resetea a cero el contador del codificador motor.
Fijar Posición Soft Home	FijaSoftHome	La posición actual del robot es la nueva posición de Soft Home.
Busca posición de Home de un eje	HomeEje eje	Busca a través de los microswitches la posición Home de un eje concreto.

Control de la pinza

Abrir Pinza	AP, AbrirPinza	Abre la pinza.
Cerrar Pinza	CP, CerrarPinza	Cierra la pinza.
Mordaza	Mordaza milímetros	Establece apertura pinza.

Movimiento de ejes y motores

Mover Eje	MoverEjePasos eje pasos MoverEjeGrados eje grados	Mueve el eje indicado una cantidad de pasos de motor o grados.
Mover Motor	MoverMotor motor pasos	Mueve el motor indicado.

Memorización de posiciones

Memorizar posición	MemPos nombrevar	Memoriza la posición actual

		del robot en la variable.
Ir a posición	IrPos nombrevar	Mueve el robot a la posición almacenada en la variable.

Comandos de Control de flujo

Control de flujo

Espere	Espere milisegundos	Pausa durante los milisegundos indicados.
Salta A	SaltaA nombreRutina	Salta a una subrutina.
Si Condición Salta A	Si condición Comando	Evalúa una condición y ejecuta el salto si se cumple.

Comandos de Entrada/Salida

Activación y desactivación de salidas digitales

Salida Digital	SalidaDigital [0-7] (0|1) SD [0-7] (0|1)	Cambia el estado de la salida indicada.

Activación de la salida analógica

Salida Analógica	SalidaAnalogica [0-3] voltaje SA [0-3] voltaje	Cambia el voltaje de la salida analógica.

Lectura de entradas

Lee Entrada Digital	LeeEntradaDigital [0-7] ED [0-7]	Lee la entrada digital.
Lee Entrada analógica	LeeEntradaAnalogica [0-3] EA [0-3]	Lee la entrada analógica.

Estado del robot

Lee Switch	LeeSwitch [0-5] LS [0-5]	Devuelve estado de microinterruptor.
Lee Motor	LeeMotor [0-6] LM [0-6]	Devuelve el número de pasos del codificador.
Lee Eje	LeeEje [0-5] LE [0-5]	Devuelve ángulo de giro del eje respecto a Home.
Estado Robot	EstadoRobot ER	Muestra por pantalla resumen del estado del robot.

Salida a consola y aviso del controlador

Salida a consola

Escribir	Escribir “ristra” Escribir var.	Muestra por consola la ristra o variable indicada.

Aviso controlador

Timbre	Timbre	Hace sonar el timbre de aviso del controlador.

8. Bibliografía

Bibliografía Online

- www.libusb.com
- www.jquery.com
- http://www.eeherald.com/section/design-guide/esmod14.html
- http://www.l-com.com/content/USB-Tutorial.html
- http://en.wikipedia.org/wiki/Universal_Serial_Bus
- http://www.beyondlogic.org/usbnutshell/usb3.shtml
- http://www.theoldrobots.com/book45/ER4u_User_Manual.pdf
- http://www.usbmadesimple.co.uk/ums_4.htm
- http://www.freebsd.org/doc/en/books/arch-handbook/usb-dev.html
- http://www.keil.com/pack/doc/mw/USB/html/_u_s_b__endpoints.html
- https://touch-base.com/documentation/USB%20Basics.htm
- http://www.robotics.org/joseph-engelberger/unimate.cfm

Bibliografía Física

- Scorbot-ER 4u, Manual de usuario. Intellitek, STI Soluciones Tecnológicas Integradas.
- Controlador USB, Manual de usuario. Intellitek, STI soluciones tecnológicas Integradas.
- Esquemas Cinta transportadora y sensor infrarrojo. Rhino