UNIVERSIDAD POLITÉCNICA DE PUEBLA

Maestría en Ingeniería

Automatización de Procesos Industriales

"Generación de trayectorias angulares por funciones de interpolación, implementadas en una red de 5 servomotores"

Versión 1: Tesis de Maestría

Luis Enrique Martínez Cruz

Juan C. Bonilla, Puebla. Enero 2013

UNIVERSIDAD POLITÉCNICA DE PUEBLA

Maestría en Ingeniería

Automatización de Procesos Industriales

"Generación de trayectorias angulares por funciones de interpolación, implementadas en una red de 5 servomotores"

Luis Enrique Martínez Cruz

Tesis de Maestría

MC. Azgad Casiano Ramos
Dr. Antonio Benitez Ruíz
Asesores

Comité Evaluador

MC. Antonio Benitez Ruíz
Sinodal

MC. José Pedro Sánchez Santana
Sinodal

Juan C. Bonilla, Puebla. Enero 2013

Índice

Lista de tablas

Lista de figuras

Capítulo 1

Marco teórico

La automatización industrial contempla los avances tecnológicos en mecánica, computación, eléctrica y electrónica. Se ocupa del proceso de fabricación de productos otorgando flexibilidad a los procesos de producción y mejorar la productividad.

Una parte especializada de la automatización industrial es la robótica industrial, ciencia que aplica disciplinas y tecnologías para el desarrollo, diseño, construcción, control, aplicaciones y puesta en marcha de sistemas robóticos en la industria.

Un robot industrial es una máquina integrada por componentes mecánicos, eléctricos, electrónicos y de comunicaciones. Estos robots industriales son en su mayoría de brazos articulados y se les hace llamar de manipulación o robots manipuladores.

Para la Federación Internacional de Robótica (IFR) *Norma ISO/TR 8373*, un robot industrial de manipulación es una máquina automática, reprogramable y multifuncional con tres o más ejes que pueden posicionar y orientar materias, piezas, herramientas o dispositivos especiales para la ejecución de trabajos diversos en las diferentes etapas de la producción industrial, ya sea en una posición fija o en movimiento.

Las sociedad actual exige cada vez mayor calidad en los productos y servicios, esto estimula que muchas empresas se vean en la necesidad de automatizar sus procesos productivos. Entre todas las industrias posiblemente la industria automotriz sea la que

aplica robots manipuladores y máquinas automatizadas en la mayoría de sus procesos de fabricación de sus productos.

Siendo el estado de Puebla poseedor de una gran cantidad de empresas donde la industria se concentra principalmente en el área metropolitana de la Cda. de Puebla y que además comprende el 85 % de la industria del estado, y que entre las industrias que cada vez destacan más; son la industria metálica básica, la de la química ligera y la de artículos eléctricos, además de las industrias relevantes como la textil y la industria automotriz (HYLSA, VolksWagen, etc.) [8], es por esto que se hace necesario que los estudiantes egresados de las áreas de la ingeniería con tendencias al ramo industrial, conozcan la importancia de la automatización y en particular del robot manipulador.

Actualmente en la Universidad Politécnica de Puebla aún no cuenta con algún tipo de robot manipulador para la enseñanza y aprendizaje de muchas áreas de la ingeniería como la robótica y el control. Los altos costos de estos robots reducen toda posibilidad de adquirirlos. Es por esto que investigadores y alumnos de la universidad, se han dado a la tarea de plantearse desarrollar un robot manipulador de arquitectura abierta, aprovechando todos los conocimientos adquiridos, y además con la posibilidad de adquirir los dispositivos necesarios a menores costos.

Muchos proyectos son inspirados en algún modelo, y nuestro proyecto ha sido inspirado en el modelo *LWR* (*Light Weight Robot*) de *KUKA Robotics*. Este robot manipulador fue desarrollado por el Instituto de Robótica y Mecatrónica del Centro Aeroespacial Alemán, en Oberpfaffenhofen-Wessling, Alemania [13], y posteriormente fue comercializado por *The KUKA Robot Group* bajo el modelo *LWR*.

El robot manipulador *KUKA LWR* (ver Figura 1.1) cuenta con 7 grados de libertad y fue presentado en el año 2006. Posee sensores de torque en las articulaciones y es capaz de manipular cargas de 7 a 14 kg de peso. A pesar su apariencia, su peso es aproximadamente de 15 kg. Tiene un alcance horizontal de *868 mm*, con velocidades

de articulación de 110 a 210 grados por segundo [15].

Figura 1.1: Robot manipulador KUKA modelo LWR.

En la Figura 1.2 se puede apreciar al lado izquierdo de la imagen el robot o brazo manipulador *KUKA LWR*, y al lado derecho, un ejemplo de un proyecto de desarrollo de un manipulador móvil tipo humanoide con dos brazos manipuladores como extremidades superiores (Robot TUM-Rosie [14]).

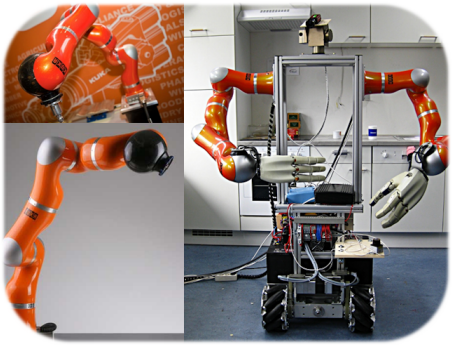


Figura 1.2: Robot TUM-Rosie, es un robot tipo humanoide con brazos superiores constituidos por dos robots *KUKA LWR*

Para este proyecto inicialmente se cuenta con los elementos que constituirán la articulaciones como son los dispositivos mecatrónicos y también con los dispositivos electrónicos que los controlarán. En un futuro este robot manipulador será ocupado para realizar investigaciones y también como equipo didáctico de apoyo en las materias

de ingeniería de la Universidad Politécnica de Puebla. Todo esto será posible pues se cuentan con los conocimientos y experiencia necesaria de docentes y alumnos para conseguirlo.

En todo proyecto de diseño de máquinas, siempre se debe de iniciar con un prototipo de investigación. Este prototipo servirá como base para su investigación y mejoramiento. El desarrollo de este robot manipulador envolverá muchas áreas de la ingeniería, por lo que es imposible terminarlo en un sólo proyecto, es decir, las investigaciones serán divididas en partes, según el área que sea requerida. Una de estas partes es investigada en este documento de tesis, he intenta cubrir los siguientes bloques que se muestran en la Figura 1.3.

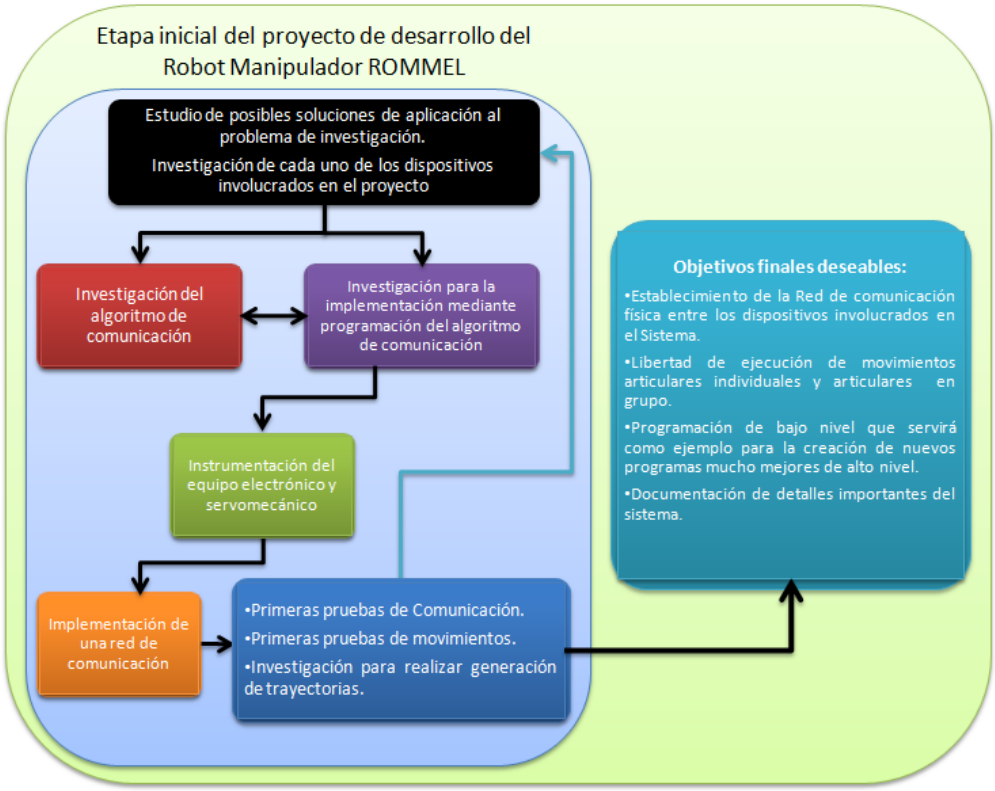


Figura 1.3: Diagrama de bloques que explica detalladamente el procesos que se realizó para lograr cubrir al máximo los objetivos deseables de este proyecto de tesis.

Un robot manipulador está formado por una serie de elementos o eslabones unidos

mediante articulaciones que permiten un movimiento relativo entre cada dos eslabones consecutivos. El movimiento de cada articulación puede ser de desplazamiento, de giro, o una combinación de ambos. Cada movimiento independiente por articulación se denomina *grado de libertad* (GDL) [2]. Los robots manipuladores industriales poseen al menos 6 GDL (ver Figura 1.4), es decir, poseen al menos seis articulaciones. Una articulación puede estar constituida por un actuador eléctrico, neumático o hidráulico.

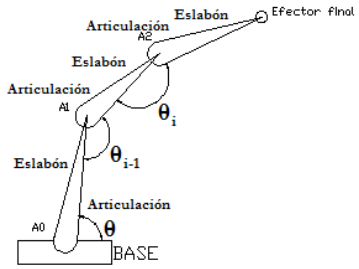


Figura 1.4: Par eslabón y articulación conforman un grado de libertad.

En la Figura 1.5 se pueden observar dos robots industriales de 6 GDL, fabricados por *MITSUBISHI Electric* y *DENSO Corporation*, estos robots manipuladores poseen actuadores eléctricos como articulaciones.

Figura 1.5: (a)Robot industrial de MITSUBISHI Electric y (b) el robot industrial de DENSO Corporation, ambos de 6GDL.

Para el desarrollo de este trabajo de tesis se usarán 5 actuadores eléctricos de CD (corriente directa), por lo que se proyecta un robot manipulador de 5 GDL cuando haya sido terminado. Aunque este robot manipulador no se tiene, lo conoceremos desde sus

inicios con el nombre de ROMMEL (robot manipulador con ejecución de movimiento en línea).

Alumnos que participaron en el verano científico en el año 2011 en la Universidad Politécnica de Puebla, proyectaron en la Figura 1.6, los alcances en el sistema de comunicación y de control de ROMMEL. Propusieron que este robot manipulador o brazo robótico (ROMMEL) de 5 GDL tendrá la capacidad de ser controlado mediante un tablero de control o desde una computadora; y como un aspecto innovador, vislumbran poder controlar a ROMMEL incluso desde *Internet* por medio de cualquier computadora sin importar el sistema operativo que esta posea (Linux o Windows).

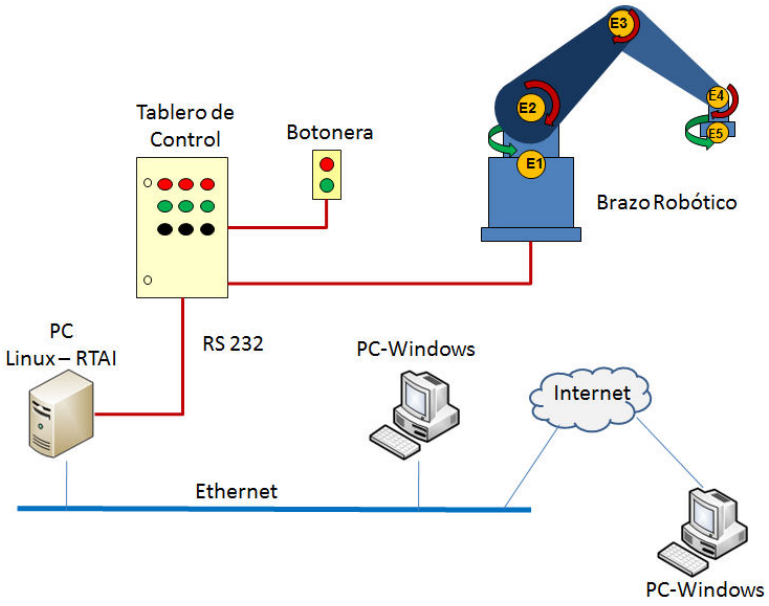


Figura 1.6: Dispositivos involucrados en el proyecto.

Es así como el futuro del desarrollo de ROMMEL es finito. Por lo que este proyecto inicia a partir de todos los aportes técnicos e informativos que existen, que no existe un documento formal publicado, pero que se les reconoce como parte de este proyecto de investigación.

Ahora abordaremos más acerca de algunas de las tantas inquietudes que se quieren

alcanzar al realizar el desarrollo de esta arquitectura abierta para el control de movimiento de 5 servomotores que serán las articulaciones del robot ROMMEL; primero definamos la esencia o propósito del desarrollo de un robot manipulador; es una máquina diseñara y construida como una imitación muy cercana al brazo humano, con el objetivo de mantener la misma calidad y menor tiempo de producción de productos de manufactura industrial. Actualmente, el robot manipulador ha ampliado sus aplicaciones como robots de asistencia en laboratorios de investigación, robots que realizan cirugías, robots manipuladores de asistencia para personas discapacitadas, robots de servicio, etc.

Para poder entender el funcionamiento de un robot manipulador o simplemente manipulador, debemos de analizar su funcionamiento básico en analogía al brazo humano. Para que el robot realice una determinada tarea, éste debe realizar uno o varios movimientos.

Para el movimiento asociado en alguna de sus articulaciones, avanza de una configuración (posición, orientación) inicial a una configuración final; este movimiento resulta ser muy complejo y puede poseer infinitas soluciones. Y porqué infinitas soluciones, pues básicamente cuando nosotros tomamos un objeto con una de nuestras manos, nuestra mano inicia en una determinada configuración y avanza hasta donde se encuentra el objeto que queremos sujetar (configuración final), si tomáramos nota de las configuraciones de nuestra mano cada cierto intervalo de tiempo durante todo el intervalo de tiempo que duro el movimiento hasta llegar al objeto que queremos sujetar, y graficáramos todos los datos obtenidos, observaremos cómo fue el movimiento, como fue su trayectoria.

Ahora supongamos que repetimos la misma actividad un determinado número de veces, llevamos notas de los valores obtenidos de cada prueba y al final los graficamos; seguramente cuando realicemos una comparación de resultados, observaremos que los

movimientos estuvieron muy aproximados, pero jamás iguales; y sin embargo todos alcanzaron el objetivo de tomar el objeto, es decir, nuestra mano puede seguir un número infinito de trayectorias para alcanzar el objeto.

Nosotros podemos decidir cómo realizaremos el movimiento de nuestra cintura, hombro, codo, muñeca y mano, y así podemos realizar un movimiento brusco y rápido, suave y lento, etc.

Sabemos que el robot manipulador es una máquina que imita el movimiento del brazo humano para sujetar y manipular objetos. Utiliza actuadores como articulaciones en sustitución de la cintura, hombro, codo, muñeca y mano del ser humano.

Es de gran interés que un robot pueda realizar movimientos veloces y con un alto grado de exactitud siguiendo una trayectoria deseada en posición y velocidad, y para ello deben de realizar movimientos individuales que tracen una determinada trayectoria deseada. Es por ello que este trabajo de tesis tiene el objetivo de integrar el equipo electrónico necesario para implementar una red de comunicación que permita ejecutar movimientos angulares mediante un programa de control desarrollado en un ambiente de programación libre o IDE (integrated develop environment); capaz de generar los comandos y ordenar los desplazamientos angulares de 5 servomotores, opcionalmente en grupo o individuales. Estos desplazamientos serán calculados mediante la simulación de funciones de interpolación.

1.1. Trabajos relacionados

Existen muchos proyectos relevantes al desarrollo de robots manipuladores, donde los autores resumen sus procedimientos realizados y las ciencias que aplicaron en sus proyectos. A continuación se presentan algunos trabajos que incentivaron el desarrollo de este proyecto de tesis.

1.1.1. Robot articulado SPIDER

El robot manipulador SPIDER consta de 5 GDL (ver Figura 1.7), fue un proyecto desarrollado por estudiantes de posgrado en el Instituto Tecnológico de Chihuahua en Julio de 2008 [12]. La estructura la fabricaron de aluminio y usaron servos resistivos como articulaciones. El diseño de SPIDER está basado en una configuración de un robot articulado de 5 GDL de Williams Karl documentado en [11]. Antes de fabricar cada una de las piezas del manipulador, modelaron el brazo en *SolidWorks*®.

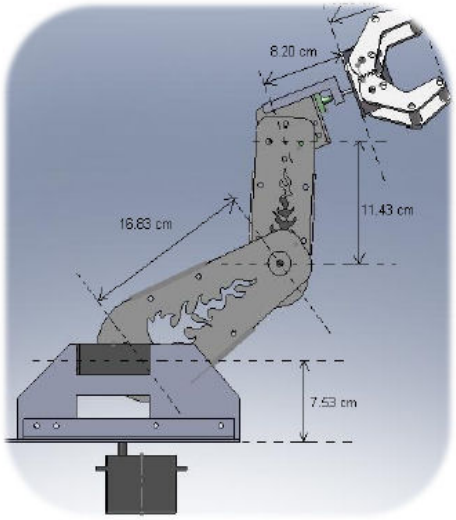


Figura 1.7: Modelo físico del robot manipulador SPIDER [12].

Mediante la realización de una serie de pruebas en simulación digital y aunque no se comentan detalles electrónicos del manipulador, se precisa que mediante simulación encuentran los ángulos permisibles de operación de cada uno de los servomotores de las articulaciones. Utilizaron el software MATLAB® para realizar simulaciones de trayectorias, aplicando teoría de cinemática directa e inversa y utilizando los parámetros físicos del manipulador.

1.1.2. Robot manipulador con 3GDL

El robot manipulador con 3 GDL (ver Figura 1.8), fue desarrollado en la Universidad de los Andes en Mérida, Venezuela [18]. La estructura la fabricaron de aluminio por ser una metal resistente, fácil de conseguir y de bajo costo.

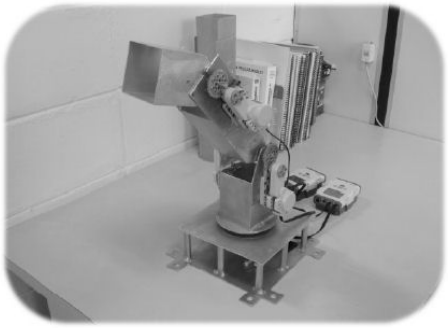


Figura 1.8: Prototipo del robot manipulador de 3GDL [18].

Este robot manipulador fue diseñado y construido para fines educativos y durante la primera etapa de investigación únicamente cumplía las funciones de posicionamiento. Las articulaciones de este manipulador están conformadas por servomotores LEGO® modelo 9842; poseen una velocidad máxima de *170 RPM* y un torque estacionario de 50 N·cm. Mediante el software MATLAB® realizan simulaciones aplicando teoría de cinemática directa e inversa, así como la dinámica del robot. Paralelamente, utilizan una aplicación de MATLAB® de Robótica que les permitió obtener resultados adicionales para comparar resultados de su robot manipulador.

1.1.3. Desarrollo de un robot con plataforma estereoscópica utilizado en la educación de ciencias de la ingeniería y computacionales

En [21] tienen el objetivo de desarrollar un robot de plataforma y con cámaras estereoscópicas, capaz de cooperar en conjunto con otros robots. El diseño está pensado para ser módulo adaptable para otros tipos de proyectos, además de las cámaras también posee una espacio especial para sostener una computadora portátil.

La forma estructural la podemos observar en la Figura 1.9; el sistema está formado por una plataforma móvil por medio de dos ruedas de tracción y una rueda de giro libre centrada al frente. El material que utilizaron fue el aluminio pues es un metal fácil de conseguir, es resistente, es manipulable y de bajo costo.

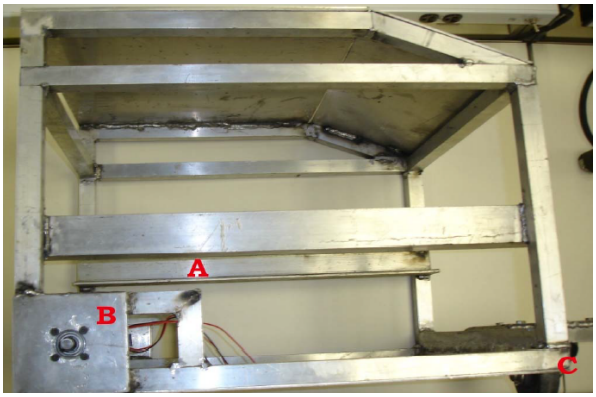


Figura 1.9: Estructura del robot de plataforma [21].

Una computadora con sistema operativo *Gentoo Linux* realizará el procesamiento de la información y envío de comandos al sistema. Los autores utilizan Linux porque ofrece ventajas tales como disponer de librerías públicas para desarrollo, y también por tener la capacidad de adaptar el sistema a las especificaciones requeridas. El protocolo de comunicación utilizado para al comunicación con los servocontroladores es el RS-485.

Las cámaras usadas como sensores de entrada son un par de cámaras de video digital

point grey research dragonfly. Estás cámaras poseen una resolución de 640x480 *pixeles* con 16 *bits* de profundidad de color, procesa a 30 cuadros por segundo y aplica un filtro de Bayer. Las imágenes son transmitidas en imágenes en escalas de grises.

Los desplazamiento de la plataforma son generados por dos servomotores de corriente directa y con escobillas; fabricados por *Dunkermotoren* modelo *GR-63*, con encoders incrementales de 500 *ticks* por revolución y reductores planetarios modelo *PLG52* con una escala de reducción de 20.25 a 1.

El control de movimiento es ejecutado por dos servocontroladores *PIC-SERVO SC 3PH* (ver Figura 1.10), capaces de suministrar hasta 6 amperios de corriente y de 12 a 48 voltios continuos. Para producir movimientos de los servomotores implementaron las librerías propias de los dispositivos servocontroladores suministradas por el fabricante.

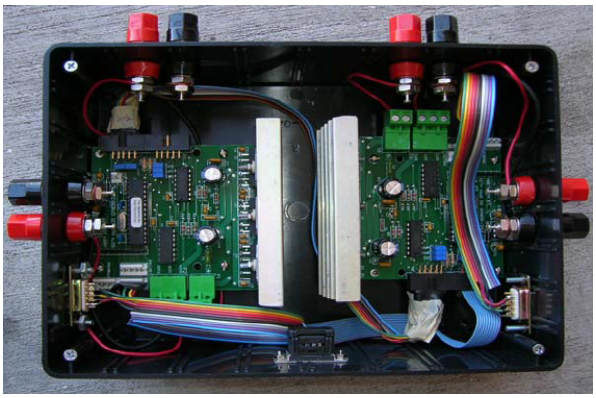


Figura 1.10: Módulo de servocontrol de robot [21].

En la Figura 1.11 observamos la idea general del proyecto. Cada módulo ocupa un papel integral en la solución del problema más grande por la aportación de información útil para el proceso. La interfaz de la cámara toma los datos sin procesar de las cámaras y las traduce en datos que pueden ser fácilmente manipulados por el módulo de procesamiento de imagen. El módulo de procesamiento de imágenes filtra todos los datos superfluos desde el bastidor y proporciona una imagen simplificada para el módulo de procesamiento de la lógica. El módulo de procesamiento lógico entonces calcula la dis-

tancia y la trayectoria a el objetivo y emite un comando de alto nivel para la interfaz de controlador de movimiento.

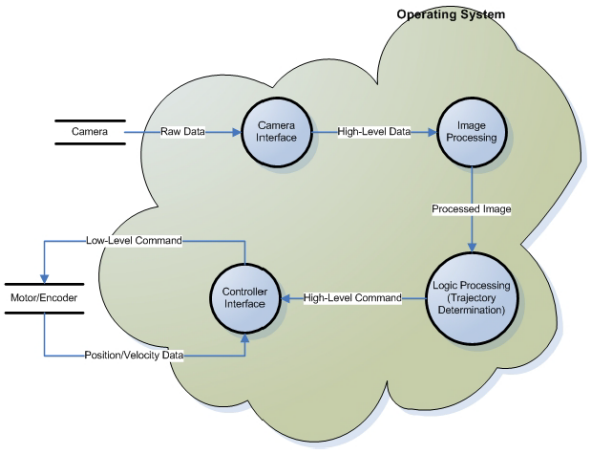


Figura 1.11: Mapa conceptual del proyecto [21].

Este proyecto realizado por los autores aún está en etapas de prueba, por lo que el módulo de control de movimiento aún no ha sido terminado.

1.1.4. Control de un sistema de un brazo robótico de 9 GDL montado sobre una silla de ruedas

En [19] diseñan y construyen un sistema conformado por un brazo robótico de 7 GDL montado sobre una silla de ruedas de 2 GDL. Este sistema está pensado como máquina de asistencia para satisfacer las necesidades de personas discapacitadas, tanto de aquellas que no pueden caminar, como de aquellas que sufren alguna discapacidad en sus brazos o que hayan sufrido algún accidente.

Este sistema robótico combina un brazo robótico de 7 GDL y la movilidad de una silla de ruedas que ha sido modificada con la incorporación de 2 GDL adicionales (ver Figura 1.12). Resuelven redundancias de manera óptima para evitar singularidades y límites de las articulaciones, así como permitir mayor movimiento de la silla de ruedas

o del manipulador en función de la proximidad del objeto a alcanzar.

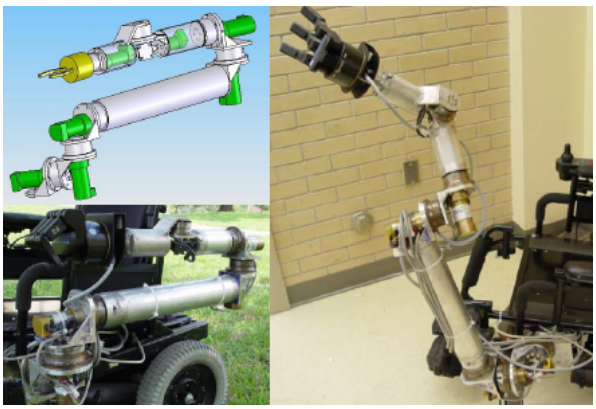


Figura 1.12: Modelo gráfico en computadora y modelo construído [19].

Mediante la combinación de operaciones obtenidas por el cálculo del jacobiano tanto de la silla de ruedas como del manipulador, obtuvieron la posibilidad de usar control convencional y métodos de optimización sin comprometer el control total del sistema coordinado. Para resolver usan el método del gradiente proyectado para obtener una buena representación de las velocidades articulares del brazo y de las ruedas de la silla de ruedas.

Es interesante mencionar que para los dispositivos de control de las articulaciones utilizan servocontroladores *PIC-SERVO SC*. Este brazo manipulador es capaz de levantar cargas de aproximadamente *6 kilogramos* de peso, y al poseer 7 GDL le permite posicionarse en espacios difíciles de su área de trabajo. Cada articulación está integrada por actuadores rotativos. Las longitudes de los eslabones pueden se modificados a modo de adaptarse para otras actividades (ver Figura 1.13).

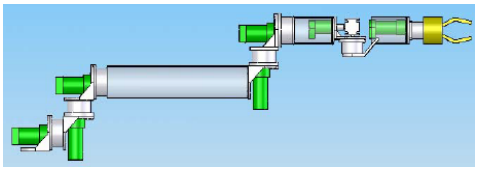


Figura 1.13: Modelo completo del brazo en SolidWorks [19].

Este sistema puede ser operado de dos modos diferentes. El primer modo es el control del brazo en tiempo real por el usuario, ya sea mediante una palanca de mando, un teclado, un *space ball*, un dispositivo háptico, un interruptor por movimiento de cabeza, un interruptor de pedal, una pantalla táctil, un dispositivos de sorbo y soplo, o por sistemas que detectan pulsos electromagnéticos del cerebro. El segundo modo es el autónomo, donde se especifica un objetivo, ya sea directamente por el usuario o por la retroalimentación sensorial. Para este modo les fue necesario generar trayectorias articulares y usaron polinomios de tercer grado, de tal forma que obtienen velocidades suaves. En cuanto a sensores el sistema posee un cámara, un laser para medir distancias, y sensores de proximidad en el efector final y alrededores de la silla de ruedas.

Finalmente, el sistema (ver Figura 1.14) de control fue simulado en Matlab 7.0.4 con la herramienta virtual instalado en un sistema operativo Windows XP, aplicando la solución de control *Weighted Least Norm* [20]. Realizaron pruebas variado los valores de los elementos de la diagonal de la matriz de pesos (W_d) para implementar el sistema de control y verificar su eficacia.

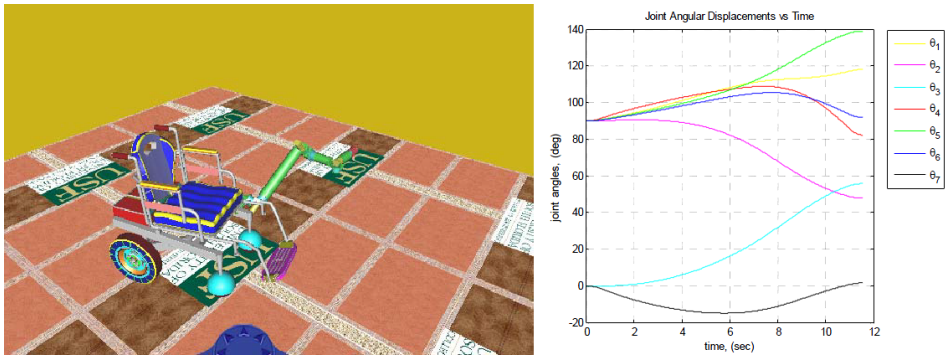


Figura 1.14: Simulación gráfica del efector final y gráfica de las articulaciones del brazo con $W_d = [1, 1, 1, 1, 1, 1, 1, 1]$ [19].

1.1.5. Generación de trayectorias para teleoperación de robots manipuladores por modelo cinemático

En [7] proponen un método más sencillo para realizar el cálculo de las trayectorias articulares de un robot manipulador, de manera teleoperado. Para validar su investigación, diseñaron e implementaron su método sobre un robot de 2 grados de libertad, que representan las articulaciones del hombro y codo.

Utilizan una especie de prótesis como herramienta para capturar los valores continuos de los desplazamientos angulares de ambas articulaciones. Esta prótesis posee dos sensores de desplazamiento angular sobre cada articulación del individuo. Los sensores se encuentran conectados a una computadora que se encarga de guardar los datos generados durante el trabajo que realiza el operador de la herramienta. Con los datos almacenados, pueden reproducir el movimiento sobre el robot de 2 grados de libertad.

En la Figura 1.15 observamos el robot de dos grados de libertad y la herramienta que usa el operador sobre el brazo para capturar los datos de los desplazamientos angulares.

Figura 1.15: Generación de trayectorias mediante una herramienta de teleoperación.

De esta manera, aseguran se reducen los tiempos de procesamiento de datos y se evita la realización de operaciones matemáticas complicadas como es el caso de la cinemática inversa.

En la Figura 1.16 observamos una de las gráficas obtenidas por los autores, donde

se muestran las señales obtenidas por los sensores de desplazamiento angular. Además realizaron una animación por computadora con el modelo cinemático directo, lo que les permitió visualizar en pantalla, los desplazamientos capturados al mover su brazo el operador.

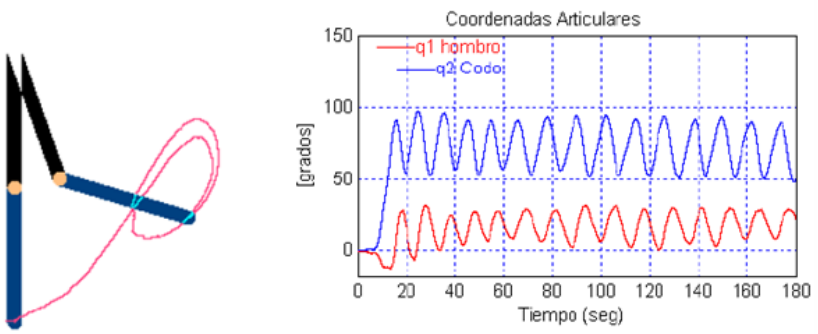


Figura 1.16: Gráfica que muestra las señales generadas por los sensores.

Este método para la generación de trayectorias para teleoperación de robots simplifica el sistema de control del robot manipulador así como el cálculo computacional. De igual manera se pueden reproducir infinitamente trayectorias almacenadas en la memoria de la computadora.

1.2. Generación de trayectorias articulares

En los métodos de generación de trayectorias en el espacio de las articulaciones no existe un control de posiciones y orientaciones cartesianas entre cada dos puntos de paso consecutivos. Se trata de determinar una función suave de interpolación para cada articulación [1].

Una trayectoria es una progresión de puntos en el espacio con restricciones temporales, es una función del tiempo $q(t)$ tal que $q(t_0) = q_{inicial}$ y $q(t_f) = q_{final}$, para un intervalo de tiempo $t = t_f - t_0$.

El autor de [9] define una trayectoria o ruta a través del espacio como un historial

en el tiempo de la posición, de la velocidad y de la aceleración para cada grado de libertad. La generación ocurre en tiempo de ejecución; en el caso más general se calcula la posición, la velocidad y la aceleración.

La meta de la planeación de trayectorias es la de generar entradas de referencia para el sistema de control de movimiento que asegure que el actuador ejecute la trayectoria planeada [15].

1.3. Funciones de interpolación

La articulación debe de moverse y recorrer una determinada distancia entre estos dos puntos ($q_{inicial}, q_{final}$), sin embargo, las distancias entre estos puntos pueden ser muy grandes y el movimiento debe de ser controlado. Para logralo se utilizan las funciones de interpolación, para establecer el tipo de trayectoria que deseamos que nuestra articulación realice y de esta forma controlar el movimiento.

Interpolar no es más que calcular puntos intermedios entre un punto inicial y un punto final, y para lograrlo existen las funciones de interpolación. Se le llama función de interpolación a la función correspondiente que se aproxima a la función real que pasa por esos puntos [2].

Los puntos calculados serán los valores de las entradas de referencia para el sistema de control de movimiento (servocontrolador), que asegurarán que el eje o articulación ejecute la trayectoria deseada. En general, cada articulación ejecutara un movimiento por lo que las funciones se aplicarían a cada una de las articulaciones.

En esta investigación se planea aplicar tres funciones de interpolación para la generación de trayectorias para los actuadores como son la interpolación lineal, la cúbica y la segmentaria o a tramos.

1.3.1. Función de interpolación lineal

La función de interpolación lineal es la operación más simple que puede aplicarse, en la que se asegura la continuidad de la posición. Esta función consiste en evaluar el polinomio obtenido para estimar valores de la función entre los dos puntos disponibles. El resultado de obtener la primera derivada de la función, nos muestra que la velocidad cambiara bruscamente entre punto y punto. Así, la trayectoria entre dos puntos (q^{i-1}, q^i) es:

$$q(t) = (q^i - q^{i-1})\frac{t - t^{i-1}}{T} + q^{i-1} \tag{1.1}$$

Donde $T = t^i - t^{i-1}$ y $q(t)$ en el intervalo $t^{i-1} < t < t^i$ [2].

1.3.2. Función de interpolación cúbica

Si lo que se desea es asegurar la continuidad en la velocidad, esto se logra al implementar la interpolación cúbica donde se hace uso de un polinomio de grado 3 para obtener un desplazamiento suave. Esta función nos permite poder establecer cuatro restricciones del movimiento, dos de posición y dos de velocidad. Las velocidades de paso por cada punto deben ser conocidas a *priori*. Así, la trayectoria entre dos puntos (q^{i-1}, q^i) son [2]:

$$q(t) = a + b(t - t^{i-1}) + c(t - t^{i-1})^2 + d(t - t^{i-1})^3 a = q^{i-1} \tag{1.2}$$

$$a = q^{i-1} \tag{1.3}$$

$$b = \dot{q}^{i-1} \tag{1.4}$$

$$c = \frac{3}{T^2}(q^i - q^{i-1}) - \frac{2}{T^2}\dot{q}^{i-1} - \frac{1}{T^2}\dot{q}^i \tag{1.5}$$

$$d = -\frac{2}{T^3}(q^i - q^{i-1}) + \frac{1}{T^2}(\dot{q}^{i-1} + \dot{q}^i) \tag{1.6}$$

donde $T = t^i - t^{i-1}$ y $q(t)$ en el intervalo $t^{i-1} < t < t^i$.

Este conjunto de polinomios concatenados, escogidos de modo que exista continuidad en la posición y velocidad, se denominan **splines** [2].

1.3.3. Función de interpolación a tramos

La interpolación a tramos consiste en descomponer en tramos consecutivos la trayectoria que une a dos puntos (q^0, q^1), por ejemplo, suponiendo el caso de tener tres tramos, en el tramo central se utiliza un interpolador lineal, por lo que la velocidad se mantendrá constante en ese tramo. En los tramos inicial y final se utilizará un polinomio de grado 2, por lo que la velocidad variará linealmente en ambos tramos y sus aceleraciones se mantendrán constantes distintas de cero. Este tipo de interpolación permite que el paso por los puntos intermedios sea planificado de tal forma que se puedan evitar cambios bruscos, es decir, en vez de pasar por el punto, se pasa tan cerca de él como lo permita la aceleración máxima. Así, la trayectoria entre dos puntos (q^{i-1}, q^i) descompuesta en tres tramos q^0, q^1 y q^2, sería entonces en los tiempos $t = 0, t = T_1$ y $t = T_1 + T_2$ [2].

Para el intervalo $0 \leq t \leq T_1 - \tau$:

$$q(t) = q^0 + \frac{q^1 - q^0}{T_1} t \tag{1.7}$$

Para el intervalo $T_1 - \tau \leq t \leq T_1 + \tau$:

$$q(t) = q^1 + \frac{(q^1 - q^0)}{T_1}(t - T_1) + \frac{a}{2}(t - T_1 + \tau)^2 \tag{1.8}$$

Para el intervalo $T_1 + \tau \leq t \leq T_1 + T_2$:

$$q(t) = q^1 + \frac{q^2 - q^1}{T_2}(t - T_1) \tag{1.9}$$

donde a es la aceleración constante que permite cambiar la velocidad de un tramo al siguiente.

$$a = \frac{T_1(q^2 - q^1) - T_2(q^1 - q^0)}{2T_1T_2\tau} \tag{1.10}$$

Y τ representa la velocidad máxima permitida sobre la aceleración a utiliza, es decir:

$$\tau = \frac{V}{a} \tag{1.11}$$

1.4. Protocolos de comunicación

En [17] se dice que un protocolo de red de comunicación de datos es un conjunto de reglas que gobiernan el intercambio ordenado de datos dentro de la red.

Dependiendo a los requerimientos del sistema físico, la comunicación puede ser de diferentes tipos:

- *Simplex*: con este tipo de comunicación, únicamente se permite que la información se transmita en un sólo sentido y de forma permanente, lo que dificulta la corrección de errores causados por deficiencias de la línea.
- *Duplex, half duplex o semi-duplex*: este tipo de comunicación puede enviar y recibir información en ambos sentidos, pero en tiempos alternos.
- *Full Duplex*: mediante este tipo de comunicación es posible la transmisión y recepción de información de forma simultánea, esto permite corregir errores de manera instantánea y permanente.

Considerando los dispositivos físicos y las características de la señal a transmitir, se determina la forma en la cual se va a realizar el intercambio de información, esto tiene que ver con la sincronización entre los extremos de la línea, detección y corrección de

errores, gestión de enlaces de comunicación con los diferentes dispositivos, entre otros factores.

Las conexiones físicas entre dispositivos se realizan mediante interfases serie, normalizados por la EIA (Asociación de Industrias Electrónicas de los Estados Unidos) que posee estándares recomendados (Recommended Standard, RS), de los cuáles los más conocidos son: RS-232, RS-422, RS-485 y el TTL.

La red de comunicación para nuestro proyecto parte del puerto serial de la computadora que maneja el protocolo de comunicación RS-232, sin embargo debido a la tarjeta interfaz de comunicación o adaptador de señales, convierte el protocolo de comunicación a RS-485. De esta forma el protocolo de nuestra red de comunicaciones sera el RS-485 y de tipo dúplex, por lo que se puede realizar transferencia de datos en ambas direcciones, pero en tiempos alternos.

1.4.1. Protocolo de comunicación RS-232 versus RS-485

El protocolo de comunicación RS-232 fue definido por la EIA en el año 1962, y básicamente, se trata de una norma que gestiona todas las características físicas, eléctricas, mecánicas y funcionales de comunicación entre un equipo terminal de datos (DTE) y un único equipo terminal de circuito de datos (DCE). Este protocolo está limitado por la distancia de separación entre los dos equipos, pues el ruido limita la transmisión de una cantidad elevada de bits por segundo cuando la longitud del cable rebasa los 15 metros. Para compensar las limitaciones de este protocolo, se ha definido el protocolo RS-485 que ofrece mayores velocidades de transmisión, mayor longitud del cable de comunicación, así como, la capacidad de conectar a más de un DCE [16].

En la Tabla 1.1 encontraremos un resúmen comparativo de las principales características que existen entre algunos de los protocolos de comunicación serie.

Características	RS-232C	RS-422A	RS-485
Número máximo de dispositivos	Un tranmisor, un receptor	Un transmisor, diez receptores	treinta y dos transceptores
Tipo de señal	Simple con respecto a masa	Diferencial	Diferencial
Longitud	$15m$	$1,200m$	$1,200m$
Número de hilos	3	4	2
Velocidad	$19,2kBits/s$	$10MBits/s$	$10MBits/s$
Comunicación	Dúplex	Dúplex	Half Dúplex
Rango de la señal de entrada al receptor	$-15V...+15V$	$-7V...+7V$	$-7V...+12V$
Sensibilidad de entrada del receptor	$\pm 3V$	$\pm 200mV$	$\pm 200mV$

Tabla 1.1: Características comparativas de interfaces de comunicación serie [4].

El adaptador *KAE-SSA485-BDV2* será el dispositivo encargado de establecer la red de comunicación con todos los servocontroladores *KAE-T0V10-BDV1* y el *host* o computadora.

1.5. Adaptador *KAE-SSA485-BDV2*

El adaptador serial inteligente (tarjeta interfaz de comunicación) modelo KAE-SSA485-BDV2 (ver Figura 1.17) es un convertidor de señales de comunicación. Realiza las adaptaciones eléctricas necesarias para poder comunicar la computadora con más de un dispositivo externo. El protocolo RS-485 se considera como una interfaz multipunto, por lo que permite la comunicación de hasta 32 dispositivos servocontroladores sobre un mismo bus de datos. Para mayores detalles de esta tarjeta puede consultar su hoja de especificaciones en [6].

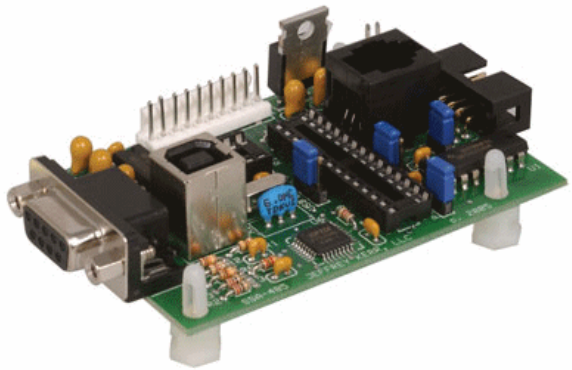


Figura 1.17: Tarjeta interfaz de comunicación modelo KAE-SSA485-BDV2, es una tarjeta que realiza una conversión de protocolos de comunicación.

Esta tarjeta interfaz de comunicación posee las siguientes características:

- Soporta interface USB o interface serial RS-232 con el *host*.
- Este adaptador soporta hasta treinta dos módulos servocontroladores sobre un mismo bus de comunicación.
- Admite velocidades de transmisión desde $19,200 baudios$ hasta $115,200 baudios$.
- Compatibilidad con *USB driver* para *Windows*, *Linux*, *Apple OS* y otros sistemas operativos.
- Puede ser usado como un servidor independiente, ya sea con nuestro chip procesador *Simple Sequencer* o con un microcontrolador PIC18F2620.
- Si se utiliza con un microcontrolador PIC18F2620, la tarjeta ofrece 10 pines de entrada/salida adicionales.
- Su lógica electrónica es energizada mediante un regulador de 9 Voltios y 500mA, que se conecta a través del puerto de dos pines $JP6$.

Este adaptador ofrece otras funciones especial que para este proyecto de investigación no son necesarias y únicamente la usaremos como adaptador de protocolos de comunicación. Un aspecto importante es su capacidad de energizar mediante su socket de comunicación ($JP1$), hasta 4 módulos servocontroladores de la red de comunicación. Los pines de alimentación se muestran en la Figura 1.18.

1.6. Servocontrolador *KAE-T0V10-BDV1*

para este proyecto se decidió usar el servocontrolador PIC-SERVO SC modelo KAE-T0V10-BDV1 para el control de motores de CD, el manual de usuario está disponible

en [5].

La tarjeta servocontroladora posee las siguientes características:

- Incluye un CI controlador de movimiento *PIC-SERVO SC*. Este CI puede controlar servomotores de CD (corriente directa), con encoder incremental. Puede generar perfiles trapezoidales, perfiles de velocidad y además ofrecer soporte para coordinar movimientos multiejes.
- Posee un CI amplificador *LMD18200* capaz de soportar cargas continuas de 3A, picos de 6A máximo y voltajes de hasta de 48VCD.
- Protección térmica interna, protección contra sobrecorriente y bajo voltaje.
- Detección de corriente, limitación de corriente activa, y protección contra sobrevoltaje.
- Señales PWM y DIR disponibles para usarse con amplificadores externos.
- La interface serial *RS485* permite controlar hasta 32 servocontroladores *PIC-SERVO* desde un solo puerto serial. La conexión con el puerto RS232 de la computadora es a través de un adaptador full-dúplex o usando la tarjeta conversora *Z232-485*.

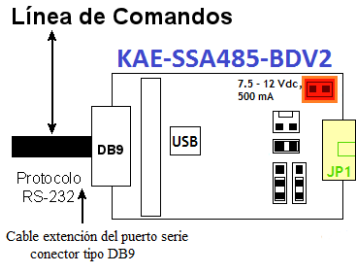


Figura 1.18: Pines de alimentación de la tarjeta interfaz de comunicación o adaptador.

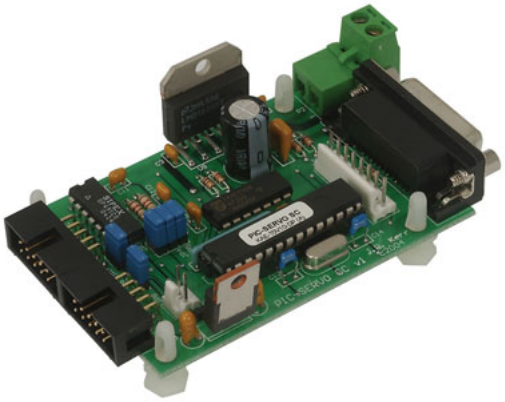


Figura 1.19: Tarjeta servocontroladora para motores de DC modelo KAE-T0V10-BDV1, será el dispositivo controlador del actuador (servomotor).

- Entradas de paso y dirección para el control de sistemas de indexación basados en paso a paso.
- Dos entradas finales de carrera para la protección de sobrecarrera.
- Tamaño reducido (2"x 3") que permite ser instalado cerca de motores, reduciendo el ruido y simplificando el cableado.
- Incluye software de prueba para *SO* (sistema operativo) *Windows 82x* y *Windows DLL* y código fuente en *lenguaje C*. *DOS* basado en código C y código en *Basic* también está disponible.
- Su lógica electrónica es energizada mediante un regulador de 9 Voltios y 500mA.

1.6.1. Especificaciones eléctricas y de sincronización

La tarjeta *PIC-SERVO SC* es un dispositivo de 5V con entradas CMOS (*complementary metal-oxide-semiconductor*) compatible ya sea con niveles lógicos CMOS o TTL (*transistor transistor logic*). Se deben de considerar los siguientes valores de sincronización (ver Tabla 1.2)

1.6.2. Comunicación e inicialización

La velocidad de comunicación es por default de 19,200 baudios, pero puede ser modificado en cualquier momento hasta 230,400 baudios. El protocolo de comunicación usa 8 bits de datos, 1 bit de inicio, 1 bit de parada y no hay paridad.

Los paquetes de instrucciones son transmitidos por el host sobre una línea dedicada de instrucciones. Los paquetes de estatus son recibidos sobre una línea de estatus separada que es compartida por todos los módulos que se interconectan a la red.

Los paquetes de instrucciones estarán formados por la siguiente estructura:

1. Byte cabecera (siempre de 0xAA).
2. Byte de módulo de dirección (1 - 32).
3. Byte de comando.
4. Byte de control (en algunas cadenas de instrucciones puede no necesitarse).
5. Bytes de datos adicionales (0 - 15 bytes).
6. Byte checksum (la suma se debe de realizar desde el byte del módulo de dirección hasta el último byte de datos adicionales).

El checksum es el byte de verificación que valida el paquete de instrucciones que recibe el servocontrolador, si es correcto, entonces los paquetes de instrucciones se han

Frecuencia del servomotor:	1953.125 Hz
Velocidad de comunicación serial:	9,600 - 230,400 baudios
Frecuencia PWM:	19,531.25 Hz (fijos)
Resolución PWM:	10 bit
Tasa de conteo de encoder:	2.5 MHz (máximo)
Paso máximo de la frecuencia de pulso:	100 KHz
Máxima frecuencia de comando:	1000 (máximo aproximado)

Tabla 1.2: Resumen de los parámetros más importantes del C.I.

enviado exitosamente y el servocontrolador retornará inmediatamente un paquete de estatus y ejecutará las instrucciones ordenadas por el paquete.

El paquete de estatus posee la siguiente estructura:

1. Byte de estatus.

2. Bytes adicionales de datos de estatus (programable).

3. Byte *checksum* (la suma de los 8 bits de los bytes mencionados).

Este byte de estatus posee la información básica acerca del estado del módulo, incluso si hubo o no error de *checksum* del comando previo.

El número de bytes de datos de estatus adicionales es programable ya sea mediante el byte de comando *Define Status (0x12)* o *Read Status (0x13)*, se puede obtener información tal como la posición del motor, el error en la posición, la velocidad actual en cuentas de encoder por ciclo, valores A/D, el tipo de módulo y número de versión, o el envío de un byte auxiliar de estatus.

1.6.3. Direccionamiento

Cuando se tienen múltiples tarjetas servocontroladoras o módulos interconectados, le asignaremos una dirección única a cada módulo.

Para asignarle a los módulos una dirección única, realizamos los siguientes pasos:

1. En el encendido, todos los módulos asumirán la dirección por default de 0 (cero). Y únicamente el módulo más alejado al host estará habilitado.

2. Mediante el comando *Set Address (0x21)* modificamos la dirección del módulo a *0x01*. Esto automáticamente habilitará al siguiente módulo de la red que tienen dirección *0x00*.

3. Ahora se envía nuevamente el comando *Set Address*, y modificamos la dirección del segundo módulo con dirección *0x00* por la dirección *0x02*. Esto automáticamente habilitará al tercer módulo.

4. Este proceso se repite hasta haber habilitado todos los módulos interconectados, y con direcciones diferentes y fijas.

Es importante recordar que este procedimiento siempre debe de realizarse en el encendido o reinicio del sistema (de la red de comunicación).

1.6.4. Comandos de grupo

Cada módulo controlador posee dos direcciones; una dirección individual y una dirección de grupo. El objetivo de la dirección de grupo es enviar comandos simples a varias servocontroladoras al mismo tiempo y estas lo ejecuten. Y un comando enviado a una dirección individual es ejecutado únicamente por esa servocontroladora.

Cuando se envía un paquete de comandos a un grupo de controladoras, la controladora líder del grupo, enviará de vuelta el paquete de estatus. Por ello, cuando se programan las direcciones con el comando *Set Address*, se debe especificar también si el módulo será líder o sólo un miembro del grupo de servocontroladoras. Debe de existir solamente un módulo líder por grupo.

1.6.5. Inicialización de la red

Para la inicializar la red, se deben de seguir los siguientes pasos:

1. Establecer la velocidad de comunicación del host a 19,200 Baudios, 1 bit de inicio, 1 bit de parada y que no exista paridad.

2. Enviar una cadena de 20 bytes nulos (*0x00*), esto para reiniciar los buffers. Esperar al menos 1 ms, y entonces extraer los bytes de entrada del *buffer* de recepción del huésped.

3. Usar el comando Set Address para asignar una dirección individual a cada módulo. Para este punto, coloca todas las direcciones de grupo a *0xFF* y no declarar ningún líder de grupo (si no existieran cambios en la velocidad de comunicación o *baudrate*, entonces podemos establecer la dirección individual y grupal al mismo tiempo).

4. Verificar que el número de módulos encontrados coinciden con el número esperado.

5. Envía un comando *Set Baud (0x1A)* a la dirección de grupo *0xFF* para cambiar la velocidad de comunicación al valor deseado. No se retornará aviso de estatus.

6. Si se realizó cambió en la velocidad de comunicación, entonces se debe coincidir ahora con la del host.

7. Averigua en cada uno de los módulos individuales (usando el comando *No Op*) para verificar que todos los módulos estén operando apropiadamente al nuevo *baudrate* establecido.

8. Usar el comando *Set Address* para asignar cualquier dirección de grupo cuando sea necesario.

Ahora, podremos enviar a cualquier módulo específico los comandos de inicialización a los módulos individuales e iniciar la operación.

1.6.6. Control PID

El servocontrolador *PIC-SERVO SC* usa un filtro de control PID (*proportional-incremental-derivative*). La salida del motor es la suma de estos tres componentes. El filtro de control PID, usa la posición del comando y la posición actual del servomotor, produciendo una salida calculada por la siguiente operación:

$$output = K_p \times posError - K_d \times (posError - prevposError) + K_i \times integralError \quad (1.12)$$

Donde:

$$integralError = \frac{1}{256} \sum_{i=1}^{n} posError \quad (1.13)$$

En resumen, las variables Kp, Ki y Kd son las ganancias de 16 bits del servo que deberemos programar si deseamos optimizar el rendimiento del motor.

El valor de la salida actual PWM (*pulse with modulation*), tiene un rango de 0 - 255 y un bit de dirección dado por:

$$PWM = min\left[abs\left(\frac{output}{256}\right) + deadBand, outputliMit\right] - currentliMitAdjustment \quad (1.14)$$

Si $output > 0$, entonces $DIR = 0$.

Si $output < 0$, entonces $DIR = 1$.

El parámetro *dead_band* es un byte de *offset* usado para compensar la fricción estática o una región de banda inactiva en el amplificador.

La señal PWM es una onda cuadrada a una frecuencia de 19,531.25 Hz, y un rango de modulación desde 255 (100 %) a 0 (0 %).

1.6.7. Modos de operación

El servocontrolador *PIC-SERVO SC* tiene varias capas de control, además permite operar en cualquiera de estas capas. Generalmente, cada capa de control envía comandos a una capa de control inferior para generar el movimiento deseado. En la Figura 1.20 observaremos los niveles de control de este servocontrolador.

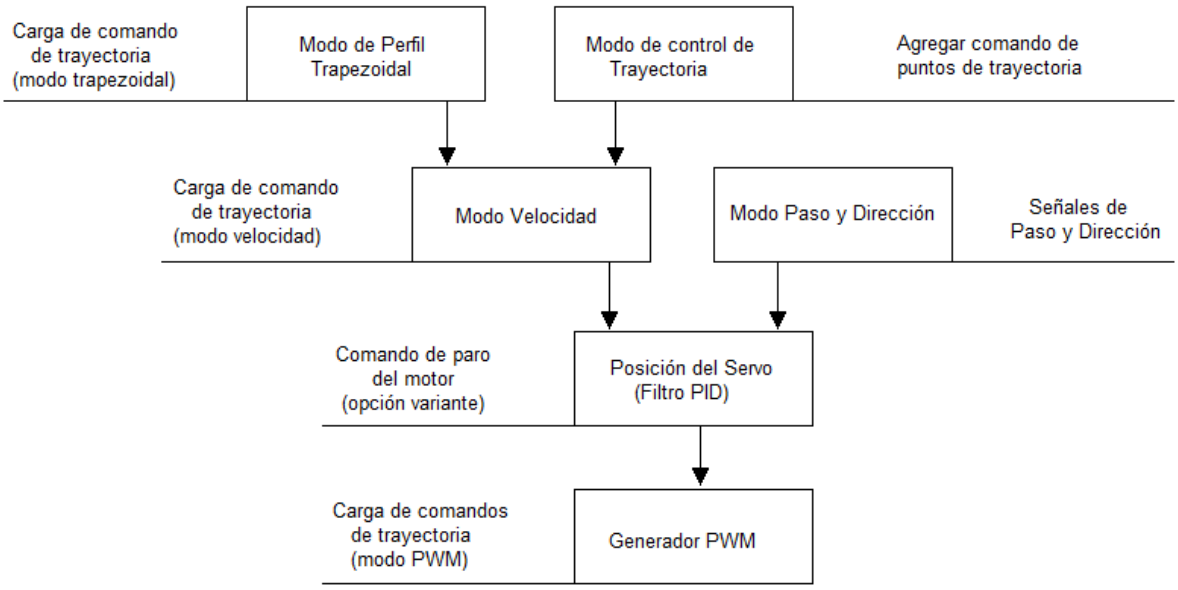


Figura 1.20: Niveles de control del servocontrolador PIC-SERVO SC.

Este trabajo de investigación está dedicado a la generación de trayectorias, por lo que el *modo de control de trayectoria* es el modo de operación que nos interesa.

La característica fundamental de este modo de operación es un *buffer de puntos de trayectoria* que almacena una serie de puntos estrechamente espaciados, más adelante se comentará más acerca de este modo de operación.

1.6.8. Modo de control de movimiento coordinado (CMC)

El modo CMC permite al *host* coordinar el movimiento de varios *PIC-SERVOS's*. La característica fundamental de este modo es la utilización del *buffer* del microcontrola-

dor de la tarjeta servocontroladora *KAE-T0V10-BDV1*. Este *buffer* permite almacenar hasta 128 puntos de trayectoria estrechamente espaciados. El proceso que se realiza para coordinar varios ejes es el siguiente:

1. La computadora o *host* calculará los puntos de trayectoria para uno o más servomotores. La idea es que cada servomotor se moverá entre punto y punto de trayectoria en un intervalo de tiempo fijo. La trayectoria multieje es seguida por todos los motores moviéndose de un punto de trayectoria al siguiente con en tiempo exacto.

2. El *host* es el encargado de descargar los puntos de trayectoria a cada uno de los *PIC-SERVO's SC*.

3. Con un simple byte de comando, el *host* puede iniciar la ejecución de los movimientos al mismo tiempo. Todo esto se logra debido al oscilador de cuarzo de la tarjeta, que permite que cada motor se mueva sincrónicamente de un punto de trayectoria al siguiente.

Tenemos varias características importantes a tener en cuenta sobre este modo de control de trayectoria. En primer lugar, el *buffer* puede ser continuamente rellenado con nuevos puntos de trayectoria mientras el motor se esté moviendo. El *buffer* puede contener hasta aproximadamente 4 segundos de datos de la trayectoria, pero si un movimiento deseado es más que eso, los nuevos puntos se pueden agregar a la memoria intermedia mientras se mueve para crear movimientos continuos de longitud ilimitada. Para algunas aplicaciones, es deseable mantener sólo una pequeña cantidad de datos en el *buffer*, pues esto permite que el host para cambiar la ruta de la marcha con un retraso mínimo.

La segunda característica importante es que cuando el *PIC-SERVO SC* este deslizándose de un punto de trayectoria al siguiente punto de trayectoria con velocidad constante, el servocontrolador aproximará puntos intermedios entre cada dos puntos de trayectoria calculado por el *host*. Los puntos de trayectoria son calculados como las posiciones donde el motor debería estar ya sea en intervalos de $30Hz$, $60Hz$ o en $120Hz$. El objetivo del servocontrolador al aproximar puntos intermedios, es generar movimientos suaves en linea recta entre punto y punto de trayectoria. Esto le permite al *host* generar puntos más espaciados y por lo tanto enviar menor cantidad de datos al *buffer* del servocontrolador. Otro detalle importante es que la tarjeta aproximará los puntos intermedios en intervalos de 1953,125Hz.

El *host* descargará los datos al *buffer* del servocontrolador usando el byte de comando $0xnD$, donde $n = 0, 2, 4, 6, 8, A, C, E$. Y con el comando $0x0D$ podemos iniciar la ejecución de forma grupal o individual. En total se puede descargar hasta 128 puntos de trayectoria al *buffer* del servocontrolador.

Cuando un punto de trayectoria es descargado al *buffer*, el dato de posición también incluye un *bit* de dirección de rotación y un *bit* que especifica si el punto de trayectoria deberá ser alcanzado en $33ms$ ($30Hz$) o en 16,7ms ($60Hz$). Esto permite al *host*, actualizar el estrechamiento entre punto y punto de trayectoria durante la ejecución, para por ejemplo, mejorar movimientos curvos muy cerrados y desplazarse suavemente de tal forma que el error sea el mínimo.

El número de ejes que se pueden controlar está limitado por la velocidad del puerto serial y por la frecuencia de los puntos de trayectoria.

1.7. Servomotor MAXON

El servomotor *MAXON* modelo *397280* se encuentra integrado por los dispositivos electrónicos y dispositivos mecánicos mostrados en la Figura 1.21.

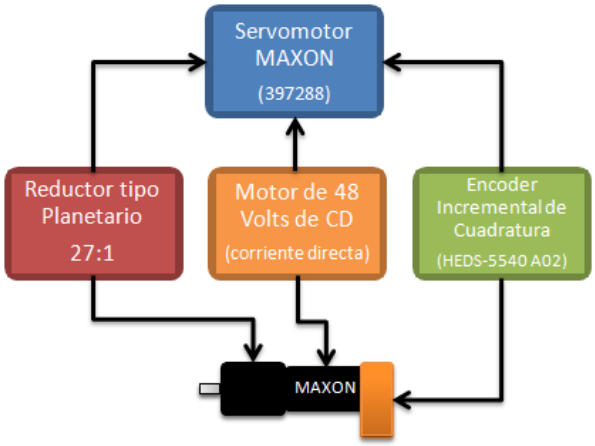


Figura 1.21: El Servomotor de *MAXON Motors* está integrado por un reductor tipo planetario, un motor de 48 Volts de CD y un encoder incremental de cuadratura.

Los servomotores son sistemas electromecánicos encargados de producir movimientos lineales o rotacionales en algunos tipos de robots manipuladores. Existen una gran variedad de robots manipuladores que poseen servomotores como articulaciones, por ejemplo, en la Figura 1.22 observamos algunos tipos de robots industriales de fabricantes tales como *KUKA Robotics*, *KAWASAKI* y *EPSON*.

Figura 1.22: Esta figura muestra 3 robots industriales de diferentes fabricantes como el (a) robot de paletizado para carga pesada de *KUKA Robotics* modelo *KR240270-2PA* de 6 GDL, (b) el robot industrial de *KAWASAKI* modelo *FS06N* de 6 GDL para trabajos de soldadura por arco, manipulación de materiales, ensamblaje, inspección, etc. (c) El robot industrial *E2 SCARA* de *EPSON* con 6 GDL puede realizar tareas de ensamble de discos duros, ensamble de electrónico, ensamble de fibra óptica, carga y manipulación de material.

El servomotor usado como articulación en robots manipuladores genera desplazamientos rotacionales; para controlar estos movimientos tenemos el encoder que es el sensor utilizado para medir su posición angular, y de esta forma poder controlar su posición y velocidad. Para el funcionamiento correcto del servomotor se necesita un amplificador electrónico o *servodrive*; es un dispositivo electrónico encargado de acoplar al motor eléctrico con la parte de control. El *servodrive* entrega y controla el voltaje necesario para el correcto funcionamiento del motor. Finalmente el motor eléctrico es un elemento electromecánico que convierte la energía eléctrica en energía mecánica.

1.7.1. Encoder incremental de cuadratura *HEDS-5540*

En la Figura 1.23 se muestra el encoder incremental de cuadratura. Este encoder se localiza montado de forma axial (parte posterior) al eje del motor de CD. Podemos destacar de este encoder una alta fiabilidad, alta resolución y fácil montaje. El objetivo de este sensor es el cerrar el lazo de control de la articulación, ya que permite la adquisición de información concerniente a la posición del ángulo de rotación del motor.

Figura 1.23: Encoder incremental de cuadratura modelo HEDS-5540.

El encoder está integrado por una fuente de luz *led*, un circuito integrado detector de luz, un disco metálico giratorio y ranurado (*codewheel*), y un circuito de salida. El disco gira entre la fuente de luz y el fotodetector, la luz pasa por las ranuras y son detectadas por A y B, esto produce un cambio de estado la salida del codificador. El circuito de salida entrega pulsos que son proporcionales al ángulo de rotación.

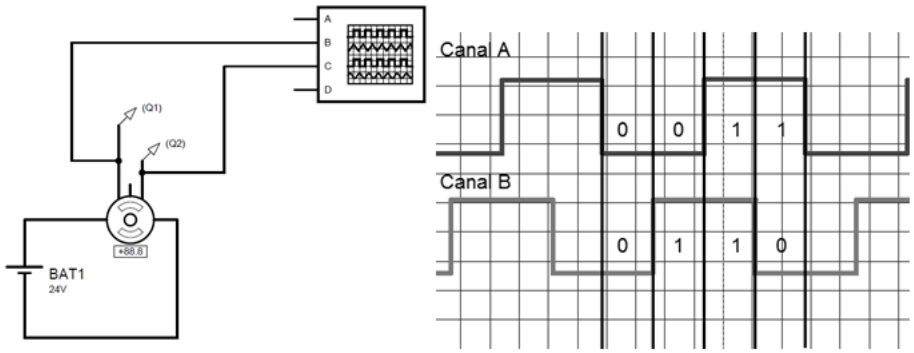


Figura 1.24: Simulación en el *software ISIS Proteus v7.4* de un encoder incremental de cuadratura, donde observamos ambas señales A y B, obtenidas en el osciloscopio digital. Las señales están desfasadas 90°

Un encoder incremental de cuadratura entrega dos señales cuadradas (canal A y Canal B) de igual periodo y frecuencia, pero *desfasadas* en *90°*. Mediante esta combinación de señales cuadradas del encoder podemos conocer la dirección de rotación o sentido de giro del eje del motor. En la Figura 1.25 observamos el interior de este modelo de encoder.

Figura 1.25: Estructura interna de un encoder incremental de cuadratura modelo HEDS-5540.

El encoder incremental de cuadratura HEDS-5540 que utilizaremos, posee las siguientes características importantes:

- Tienen 3 canales de salida (A, B y Z).
- 500 CPR (cuentas por revolución) o 2000 PPR (pulsos por revolución).

- Temperatura de operación de -40 °C a 100 °C.
- Voltaje de operación de $+4{,}5V \leq +5V \leq +5{,}5V$.
- Corriente de operación de $+30mA \leq +57mA \leq +85mA$.
- Carga capacitiva máxima de 100 pF (2.7kΩ pull-up).
- Frecuencia máxima de 100kHz (velocidad [rpm] x N/60).

De acuerdo a los datos del fabricante, este encoder ofrece 500 *CPR* que por cuadratura obtenemos como resultado de multiplicar $500 \times 4 = 2000$ *PPR*. Es decir, el encoder entrega 2000 pulsos por cada 360°.

1.7.2. Reductor planetario modelo *GP26B* con reducción de 27:1

La operación fundamental de un engranaje es transmitir potencia o energía. La potencia de transmisión vía engranes ofrece ventajas útiles y se resumen a continuación:

- Los engranes pueden incrementar o decrementar la velocidad rotacional, con un decremento o incremento correspondiente en torque.
- Los engranes pueden cambiar dirección rotacional o el ángulo de rotación.
- Los engranes pueden ser usados para convertir movimiento rotacional en movimiento lineal.
- Los engranes pueden cambiar la posición de movimiento de rotación.

En [cita] se pueden consultar la aplicación de los sistemas de engranes de muchos dispositivos electrónicos y mecánicos.

Existen varias configuraciones de engranes, y algunas de las más comunes que podrían encontrarse, son de dientes rectos, helicoidales, tornillos sinfín, cónicos, planetarios y cremalleras.

El tipo de engranaje que usa el reductor de velocidad de nuestro servomotor, es de engranaje planetario, y se distingue por el hecho de que uno o más engranes (llamados planetas) se mueven alrededor y a lo largo de un engranaje central (llamado sol).

En la Figura 1.26 observamos a los engranes planeta que están a menudo vinculados dentro de un engranaje anular externo (llamado corona) que engrana con los engranes planetas y el engranaje sol en el centro. Uno de estos tres componentes permanece estacionario, uno más proporciona la entrada y el último la salida.

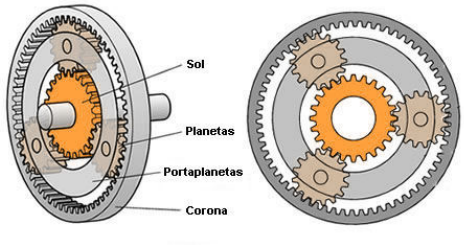


Figura 1.26: Estructura básica interna de un un reductor tipo planetario.

Por efectos de este reductor, obtenemos una reducción de 27 a 1 (27:1), que además de las ventajas útiles resumidas anteriormente, también se observará reflejado en los pulsos por revolución que entregará el encoder, obteniendo lo siguiente:

$500 \times 4 = 2000PPR$

$2000 \times 27 = 54,000CPR$

Finalmente, por efecto del reductor, recibiremos 54,000 cuentas del encoder por cada 360° que se desplace el eje del servomotor.

1.7.3. Unidades

Se realizaron las siguientes operaciones para establecer las unidades correspondientes en revoluciones por minuto y en grados por segundo del servomotor, considerando los parámetros del servocontrolador y del reductor planetario, así como, del encoder incremental de cuadratura.

Para obtener unidades de rpm ver ecuación 1.15:

$$\frac{1 \quad cuentas}{1 \quad tick} \times \frac{1 \quad rev}{54000 \quad cuentas} \times \frac{1 \quad tick}{512 \quad us} \times \frac{60 \quad s}{1 \quad min} = 217 \quad rpm \tag{1.15}$$

Para obtener unidades de $°/s$ ver ecuación 1.16:

$$\frac{360°}{1 \quad rev} \times \frac{1 \quad cuentas}{1 \quad tick} \times \frac{1 \quad tick}{512 \quad us} \times \frac{1 \quad rev}{54000 \quad cuentas} = \frac{13{,}02°}{s} \tag{1.16}$$

Capítulo 2

Construcción de un sistema para el control de 5 servomotores

En la Figura 2.1 observamos el esquema a bloques que resume la estructura general de la red de comunicación diseñada para el control de 5 servomotores (articulaciones).

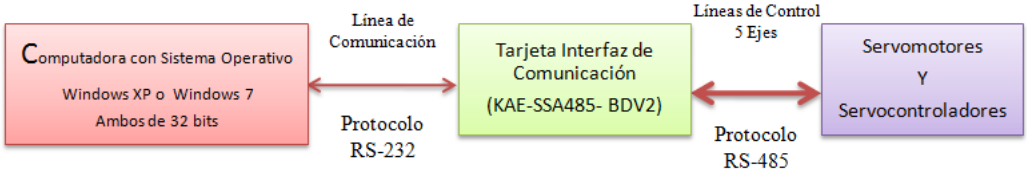


Figura 2.1: Estructura básica de la red de comunicación. Mediante la implementación de esta red, se pretende controlar 5 servomotores MAXON. Estos servomotores funcionarán como articulaciones para un robot articulado de 5 GDL en trabajos de investigación posteriores a este proyecto.

La red de comunicación inicia en la computadora, y *línea de comunicación* se forma al interconectar el puerto serie de la computadora o DB9 a la tarjeta interfaz de comunicación modelo *KAE-SSA485-BDV2*. El puerto serie de la computadora utiliza el protocolo de comunicación RS-232.

Las *líneas de control* son bidireccionales y se forman al interconectar el módulo que incluye a los servocontroladores y servomotores, con la tarjeta interfaz de comunicación o *KAE-SSA485-BDV2*, las conexiones entre módulos son seriales.

La tarjeta interfaz de comunicación utiliza el protocolo de comunicación RS-485

para enviar *paquetes de instrucciones* a las tarjetas servocontroladoras, y a su vez, estas retornan *paquetes de estatus* al *host*.

2.1. Implementación de un sistema de control para un servomotor

Para entender cómo se implementará la red de comunicación de un sistema para el control de movimiento de 5 ejes o articulaciones, comenzaremos por analizar la configuración más sencilla de la red, la implementación de un sistema para el control de movimiento de un único eje.

En la Figura 2.2 observamos el esquema a bloques que muestra el módulo de control para un único eje o articulación de la red de comunicación. Este módulo se conecta a la línea de control. El módulo de control para un eje se integra por la interconexión de una tarjeta servocontroladora modelo *KAE-T0V10-BDV1*, un motor de DC y su encoder incremental de cuadratura.

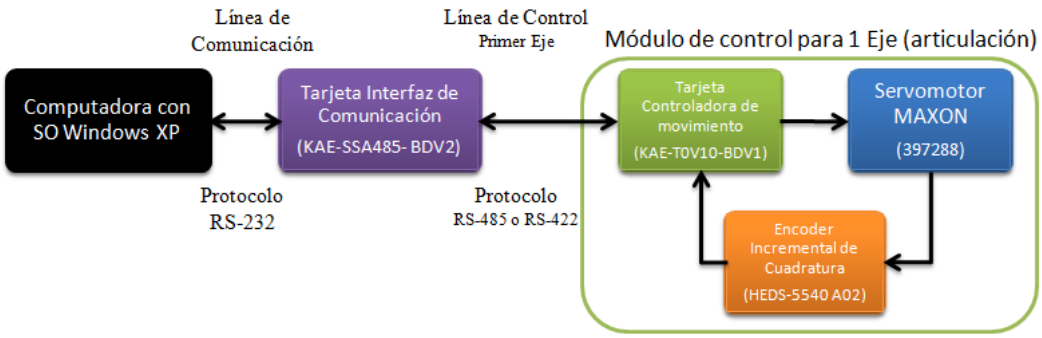


Figura 2.2: Módulo de control para un eje o articulación.

En la Figura 2.3 observamos el diagrama de conexiones que se realizaron para desarrollar la red de comunicación para el control de un único eje. La línea de instrucciones es considerada como el cable que conecta la computadora (host) con el sistema.

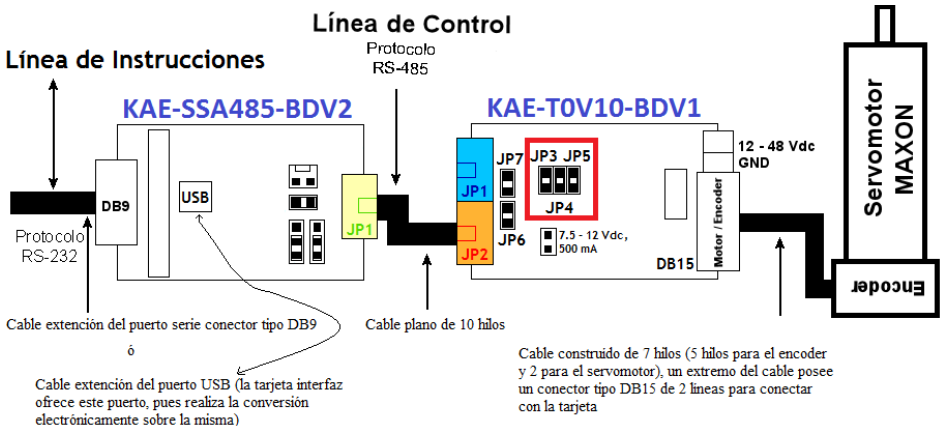


Figura 2.3: Red comunicación para el control de un único eje o articulación.

Consideremos los siguientes puntos importantes:

(a). Utilizamos un cable extensión del puerto serie con conectores DB9.

(b). Si así lo deseamos también podemos conectar la tarjeta interfaz al puerto USB de la computadora, pues la tarjeta interfaz incorpora la circuitería necesaria para realizar la adaptación entre protocolos de comunicación. Se hace necesario un cable extensión USB con un extremo tipo A y el otro tipo B, ambos machos (ver Figura 2.4).

(c). El socket macho JP1 de la tarjeta interfaz (*KAE-SSA485-BDV2*) se conecta al socket macho JP2 de la tarjeta servocontroladora (*KAE-T0V10-BDV1*) por medio de un cable plano de 10 hilos con conectores IDC (*Insulation-Displacement Connector*) en los extremos.

(d). Los *jumpers* JP3, JP4 y JP5 deben estar habilitados, pues únicamente tenemos un servocontrolador presente (ver Figura 2.5).

(e). El servomotor se interconecta con su tarjeta servocontroladora por medio de un cable fabricado que conecta el motor y su encoder.

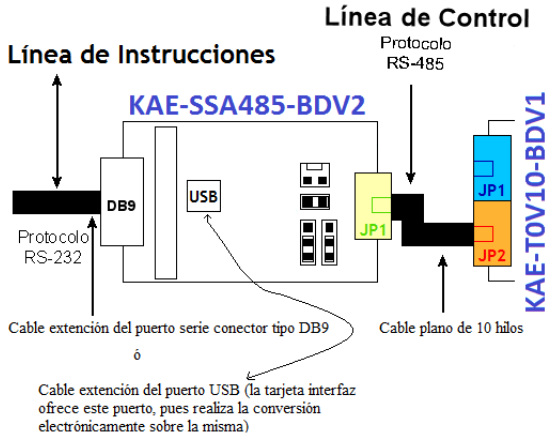


Figura 2.4: La tarjeta interfaz se conecta a la computadora por el puerto serial, y la tarjeta servocontroladora se conecta de su socket macho JP2 al socket macho JP1 de la tarjeta interfaz.

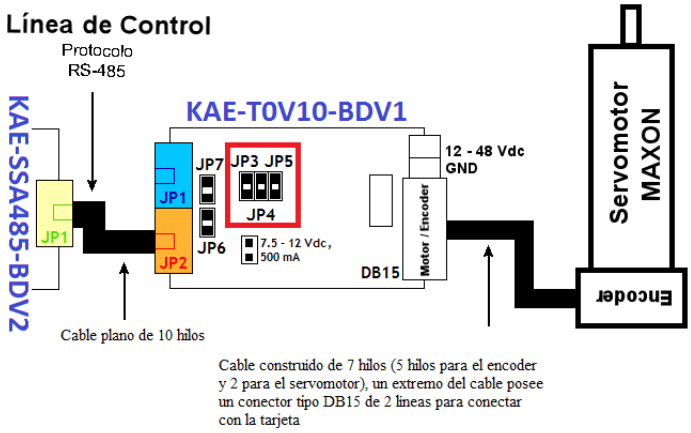


Figura 2.5: La tarjeta servocontroladora se conecta con el servomotor mediante un cable que fabricamos. Debemos de observar que los pares de pines JP3, JP4 Y JP5, deben estar interconectados mediante *jumpers*.

La configuración del cable fabricado, obedece al siguiente diagrama de conexiones mostrado en la Figura 2.6, el conector es un DB15 macho de 2 líneas.

Finalmente, en la Figura 2.7 observamos el sistema integrado físicamente para el control de un único eje.

Hasta este momento se han analizado las conexiones que se realizaron para el control de un único eje, por lo que ahora podremos entender más fácilmente las conexiones que se necesitan realizar para el control de más de un eje.

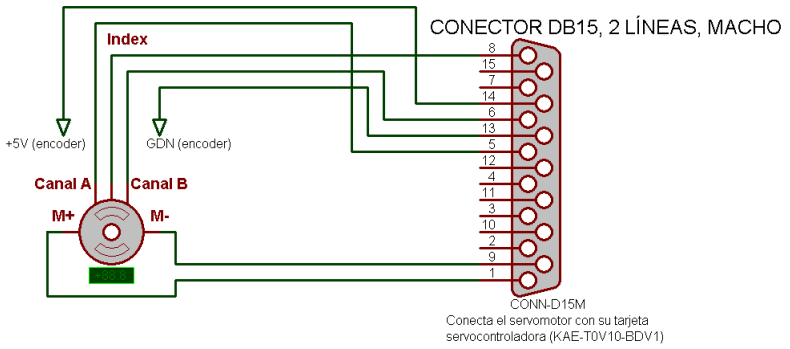


Figura 2.6: Diagrama de conexiones que se realizaron para crear el cable de comunicación del servomotor y el servocontrolador.

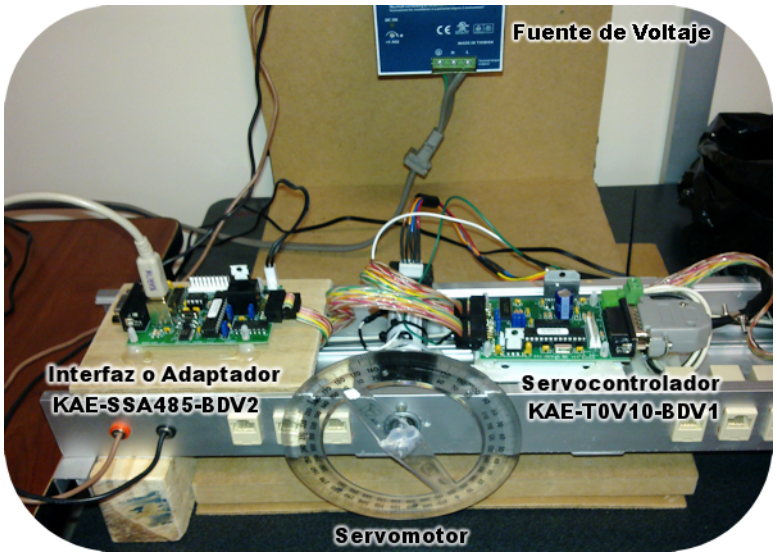


Figura 2.7: Sistema para el control de un eje o articulación.

2.2. Implementación de un sistema de control para 5 servomotores

Analizaremos las conexiones necesarias para desarrollar la red de comunicación para el control 5 ejes. Ya se ha comentado todo lo concerniente al sistema para el control de un solo eje, ahora para el sistema de control de 5 ejes, necesitaremos conectar los 4 módulos restantes.

En la mayoría de los textos refiere a la parte operativa, únicamente a los actuado-

res, sin embargo para este trabajo de investigación se ha considerado incluir también dentro de la parte operativa a los servocontroladores, pues dentro de la parte de mando ubicaremos a la computadora e interfaz de comunicación.

En la Figura 2.8 observamos un diagrama de las conexiones realizadas para formar la red de comunicación. Los sockets machos JP1 y JP2 son los encargados de establecer una red de dispositivos servocontroladores, que de acuerdo al manual técnico de la tarjeta *KAE-T0V10-BDV1* (JAMECO, 2002 - 2012), pueden conectarse hasta 31 tarjetas servocontroladoras.

Es muy importante que los *jumpers* JP3, JP4 y JP5 únicamente estén habilitados en el módulo 1 o en el módulo más alejado del host (ver Figura 2.8), de no hacerlo se corre el riesgo de quemar algún componente de las tarjetas mal configuradas, estos pines resultan ser una referencia que indican cuál es el servocontrolador más alejado del host (último). Para mayor información se sugiere revisar la documentación en [5].

En la Figura 2.9 observamos el sistema implementado físicamente para el control de los 5 ejes o articulaciones de la red de comunicación.

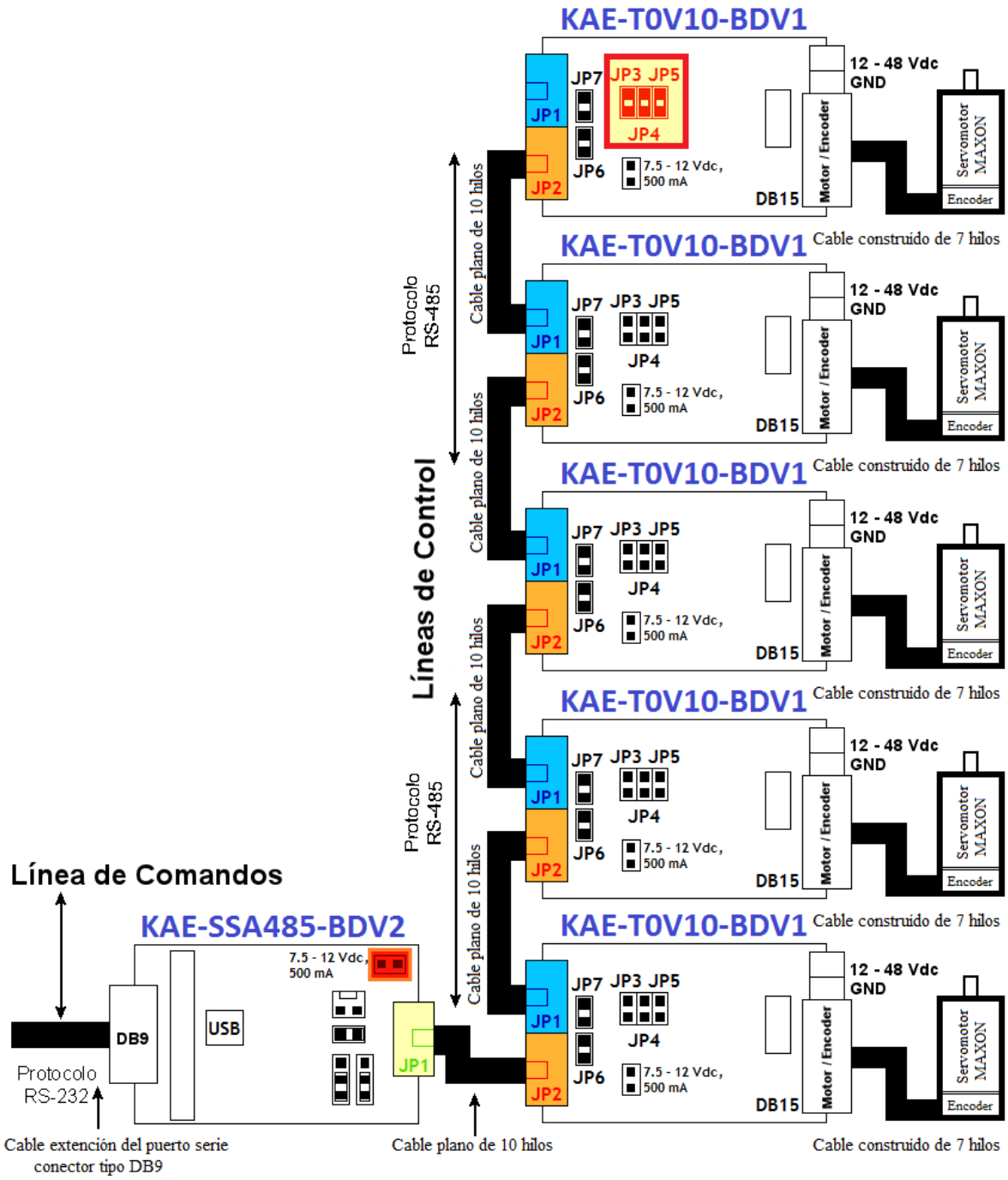


Figura 2.8: Red de dispositivos conectados al puerto serial o DB9 de la computadora.

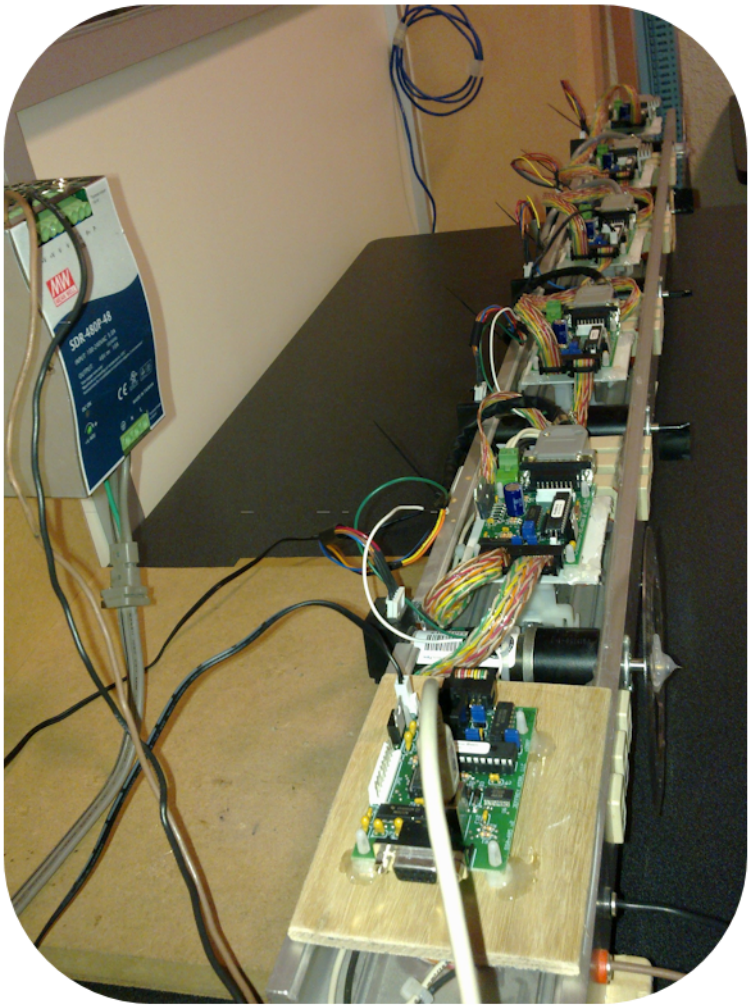


Figura 2.9: Fotografía del sistema implementado físicamente en el laboratorio. Consiste en una red de dispositivos conectados al puerto serial o DB9 de la computadora.

2.3. Implementación de la red de comunicación para el sistema de 5 servomotores

En la Figura 2.10 observamos un esquema a bloques donde se especifica cómo se realizaría la comunicación con los servocontroladores (módulos), así como, su direccionamiento en una conexión múltiple. Se observa un bloque de host que básicamente se refiere a la computadora e interfaz de comunicación, recordemos que la tarjeta interfaz de comunicación hace posible la adaptación de señales RS-232 a RS-485.

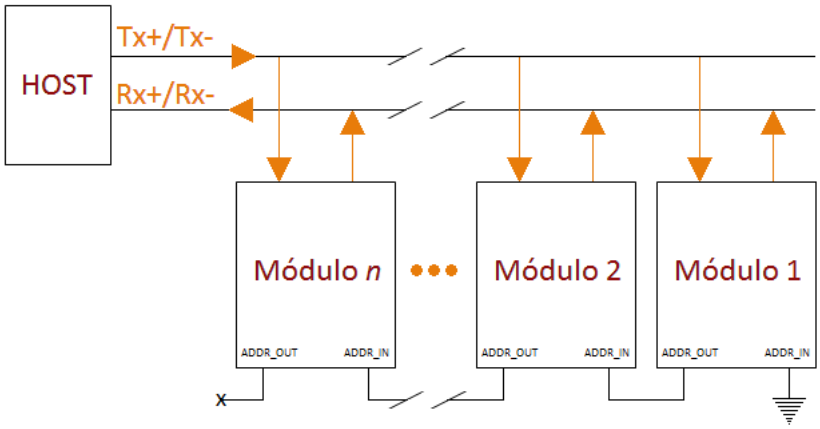


Figura 2.10: Esquema de conexión múltiple de módulos servocontroladores. Cuando se realice el direccionamiento de los módulos, la númeración ira en incremento comenzando desde el módulo más alejado al *host*.

La red de comunicación será declarada como habilitada cuando todos los módulos se encuentren conectados al host. El programa dedicado a establecer la comunicación e interacción entre el host y los 5 módulos que integran la red, será el *software Bloodshed Dev-C++ versión 4.9.9.2* ejecutado en *Windows 7 Ultimate* o en *Windows XP*, ambos sistemas operativos de 32 bits.

Mediante un programa escrito en lenguaje C, se realizaron los siguientes pasos para habilitar la red de comunicación tipo full-dúplex, que permitirá posteriormente implementar comandos de movimiento para los 5 ejes que integran el sistema.

De los programas de prueba desarrollados, resaltamos los siguientes puntos impor-

tantes:

- Se establece comunicación con el puerto serial de la computadora (en este caso fue el puerto COM1), y para lograrlo, fue necesario considerar al puerto como un fichero.

- Mediante la implementación de funciones de *Win API* podremos manipularlo y configurarlo sin que el *firewall* de *Windows* nos bloquee el acceso. La función que se utilizó para crear el fichero fue la función ***HANDLE CreateFile*** de *Win API*. El fichero es configurado con los siguientes parámetros: el protocolo de comunicación usa 8 bits de datos, 1 bit de inicio, 1 bit de parada y no habrá paridad. La velocidad de comunicación es establecida a $19,200 baudios/s$. De igual forma debemos configurar el puerto COM1 en el *administrador de dispositivos* de *Windows* de la computadora.

- Para enviar paquetes de datos (o escribir datos) a través del puerto serial, usaremos la función ***Bool WriteFile*** de *Win API*.

- Para recibir paquetes de datos (o leer datos) a través del puerto serial, usaremos la función ***Bool ReadFile*** de *Win API*. Estas funciones de *Win API* que se han mencionado, se pueden consultar en [3].

Para interactuar con los módulos de control, se hace necesario el envío de *paquetes de instrucciones* por el puerto serial. Estos *paquetes de instrucciones* son básicamente cadenas de números hexadecimales que deberán estar estructurados de acuerdo a la sección 1.6.2.

Por la gran cantidad de opciones posibles de funcionamiento y configuración para generar estos *paquetes de instrucciones*, no se mencionarán todos, y sólo consideraremos los que habremos de utilizar para alcanzar los objetivos del proyecto.

Debemos recordar la importancia del *checksum* (suma de verificación o chequeo), si este valor no es correcto, el *paquete de instrucciones* enviado no será ejecutado por el servocontrolador.

2.3.1. Inicialización de la red

Las estructuras de los *paquetes de instrucción*, así como todos los *bytes de comandos* y combinaciones de *bytes de control* pueden ser consultadas en [5]. Los pasos para inicializar la red de comunicación han sido vistos en la sección 1.6.5, por lo que el procedimiento desarrollado para la inicialización se enumera a continuación:

1. Debemos limpiar los buffers de todos los módulos y ejecutar un reinicio general a la red. Se recomienda enviar un paquete conformado por 20 bytes nulos. En seguida, un paquete más que mediante el comando *0x0F* provocaremos un reinicio general, es decir, de todos los módulos de la red (ver Figura 2.11).

2. Asignamos direcciones a cada módulo, como se indica en la sección 1.6.3. El *byte comando 0x21* permite renombrar la nueva dirección del módulo. Esto es porque cuando necesitemos que un módulo en particular realice alguna operación debera de contar con una dirección específica, además será de importancia conocer el número de dirección de cada módulo de la red. Mediante el *byte de datos adicionales* declaramos que todos los servocontroladores pertenecen al grupo *0x81* (ver Figura 2.12).

3. Establecemos la taza de transferencia para todos los módulos a 19,200 baudios. Mediante el byte comando *0x1A* y el byte de control podremos establecer la taza de transferencia de la red de comunicación (ver Figura 2.13).

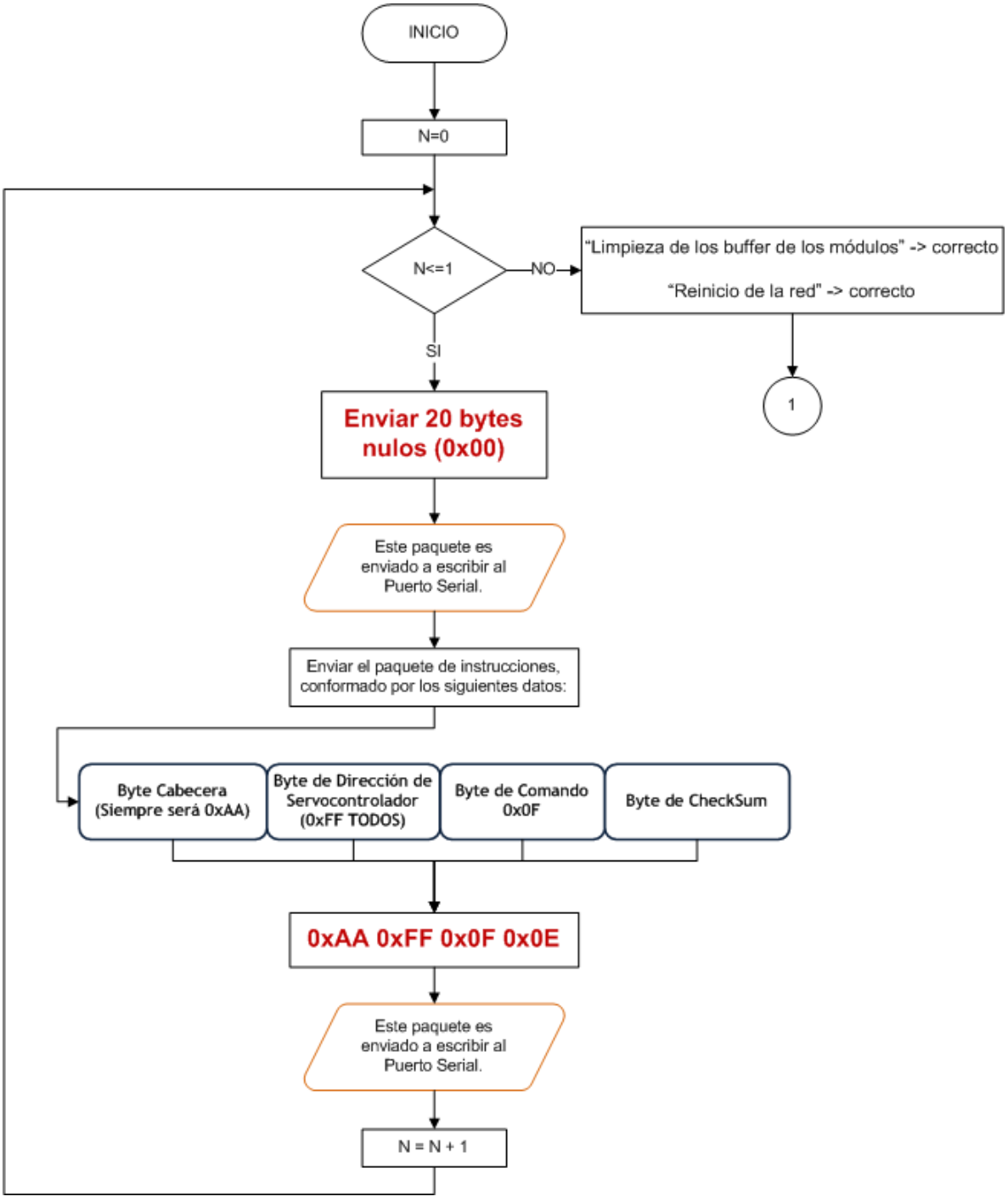


Figura 2.11: Diagrama de flujo que muestra el procedimiento realizado para limpiar los *buffers* de todos los módulos, y además general un *reset* general del sistema.

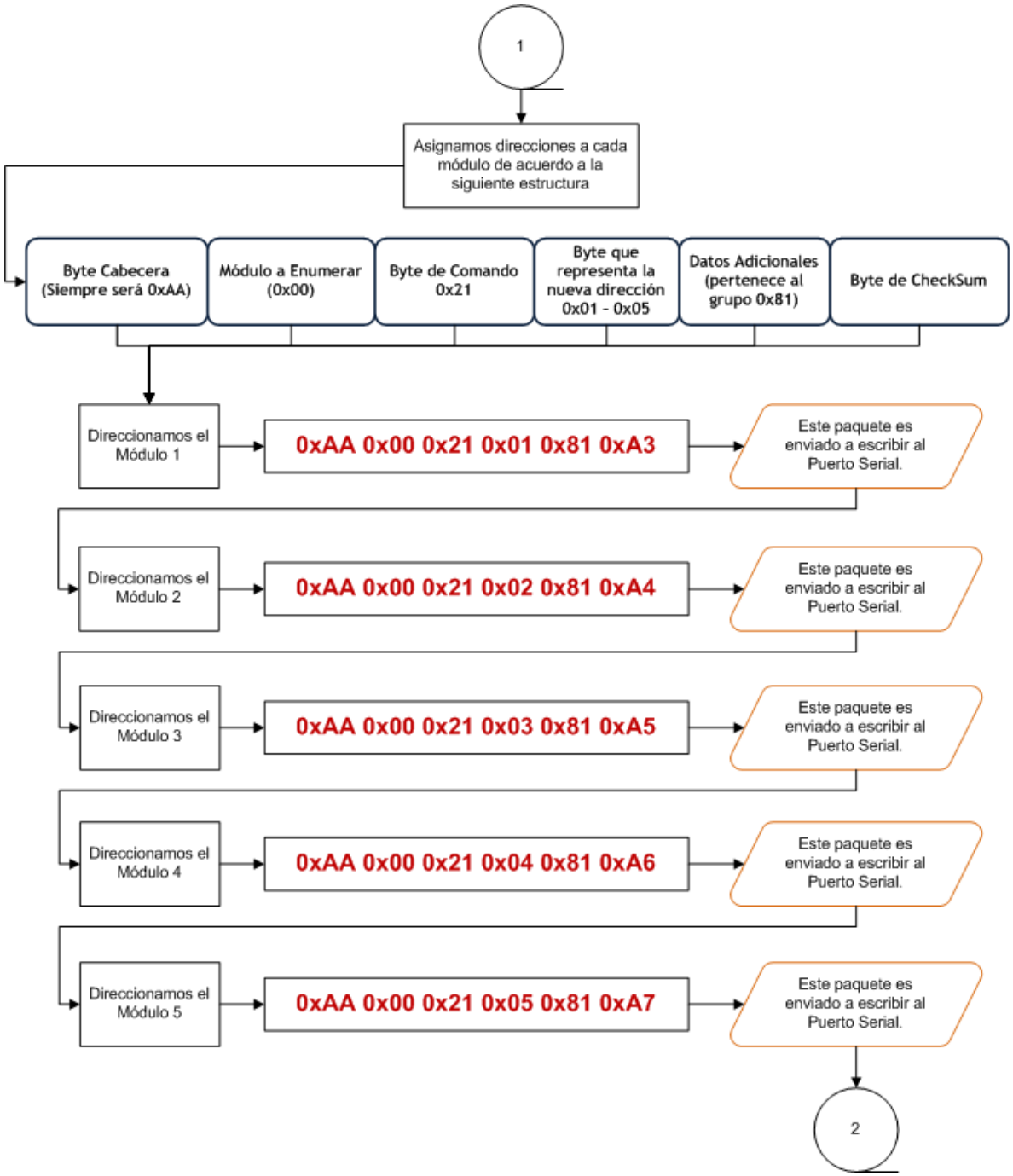


Figura 2.12: Diagrama de flujo que muestra el procedimiento realizado para enumerar cada módulo, de tal forma que cada uno cuente con una dirección diferente.

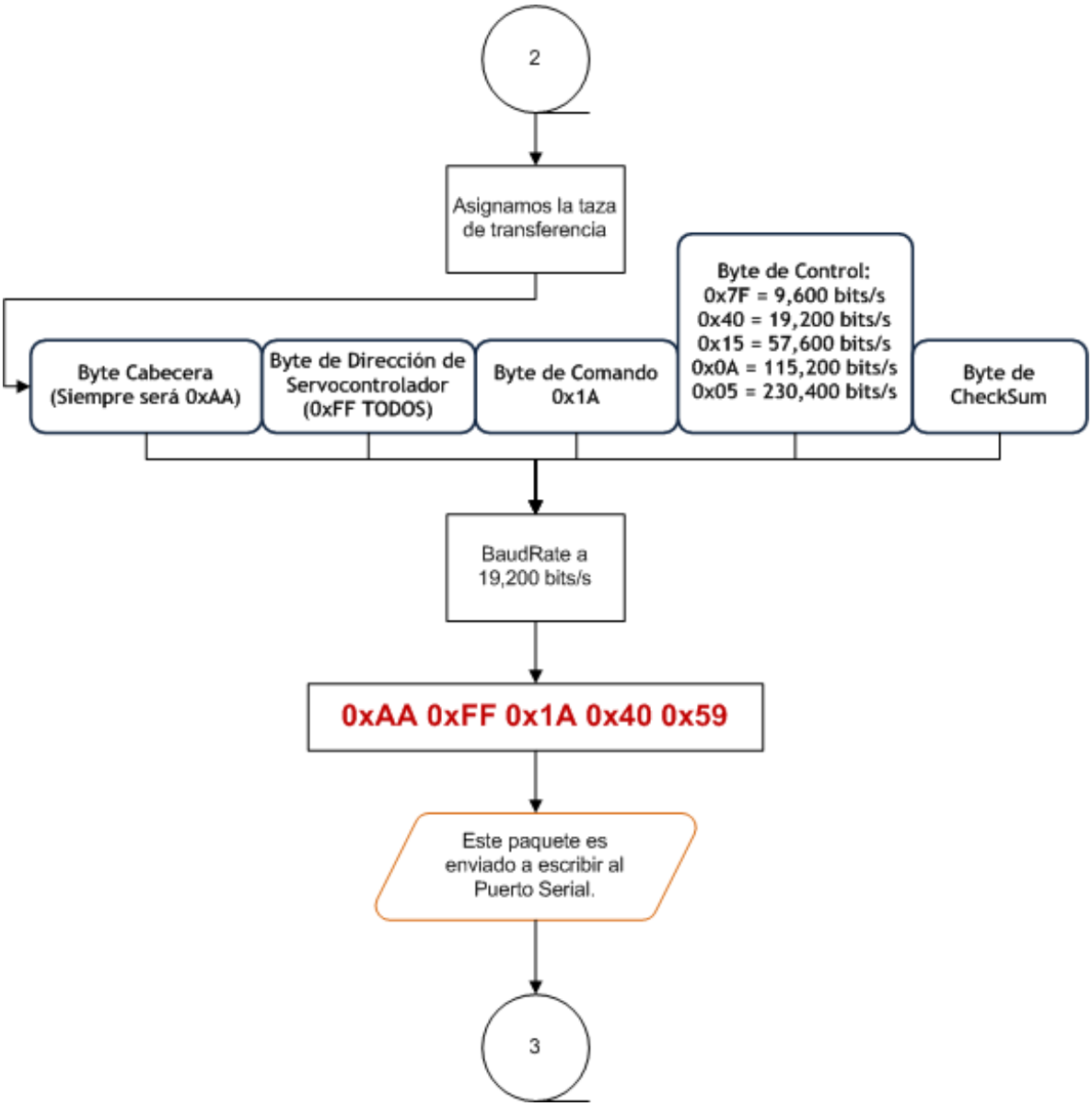


Figura 2.13: Diagrama de flujo que muestra el procedimiento realizado para establecer la velocidad de transmisión de datos de toda la red de comunicación del sistema.

4. Asignamos valores iniciales de operación por cada uno de los módulos. Esto consiste en asignarle parámetros iniciales del filtro de control PID (proporcional-integral-derivativo), necesarios para inicializar correctamente las tarjetas servocontroladoras. Definimos la ganancia proporcional (Kp), la ganancia derivativa (Kd) , la ganancia integral (Ki) , el límite de integración (IL) , el límite de salida (OL) , el límite de corriente (CL) , el límite de posición de error (EL), el divisor de frecuencia del servo (SR), etc.

   El proceso que realizamos es el siguiente:

   a) Detenemos todo posible movimiento de los motores mediante el byte comando *0x17* y el byte de control *0x06* (ver Figura 2.14).

   b) Asignamos los parámetros iniciales del filtro PID, mediante el byte de comando *0xF6* nos permite agregar los parámetros que deseemos agregar al filtro de control. Los datos adicionales son modificables según la planta utilizada por el programador (ver Figura 2.15).

   c) Finalmente, asignamos los parámetros iniciales de posición, velocidad y de aceleración. Mediante el byte de comando *0x04* y el byte de control *0x07*, podremos asignar dichos valores (ver Figura 2.16).

   Todo el procedimiento realizado en el paso 4) debe de repetirse para los cuatro módulos restantes. Los paquetes de instrucciones únicamente variarán en el **byte de dirección** del servocontrolador y en valor del ***checksum***.

   Realizado este procedimiento para todos los módulos, proseguimos con el siguiente paso.

5. Reiniciamos todos los encoders a 0 (cero) mediante el *byte de comando 0x00* y la dirección del servocontrolador que queremos reiniciar (ver Figura 2.17).

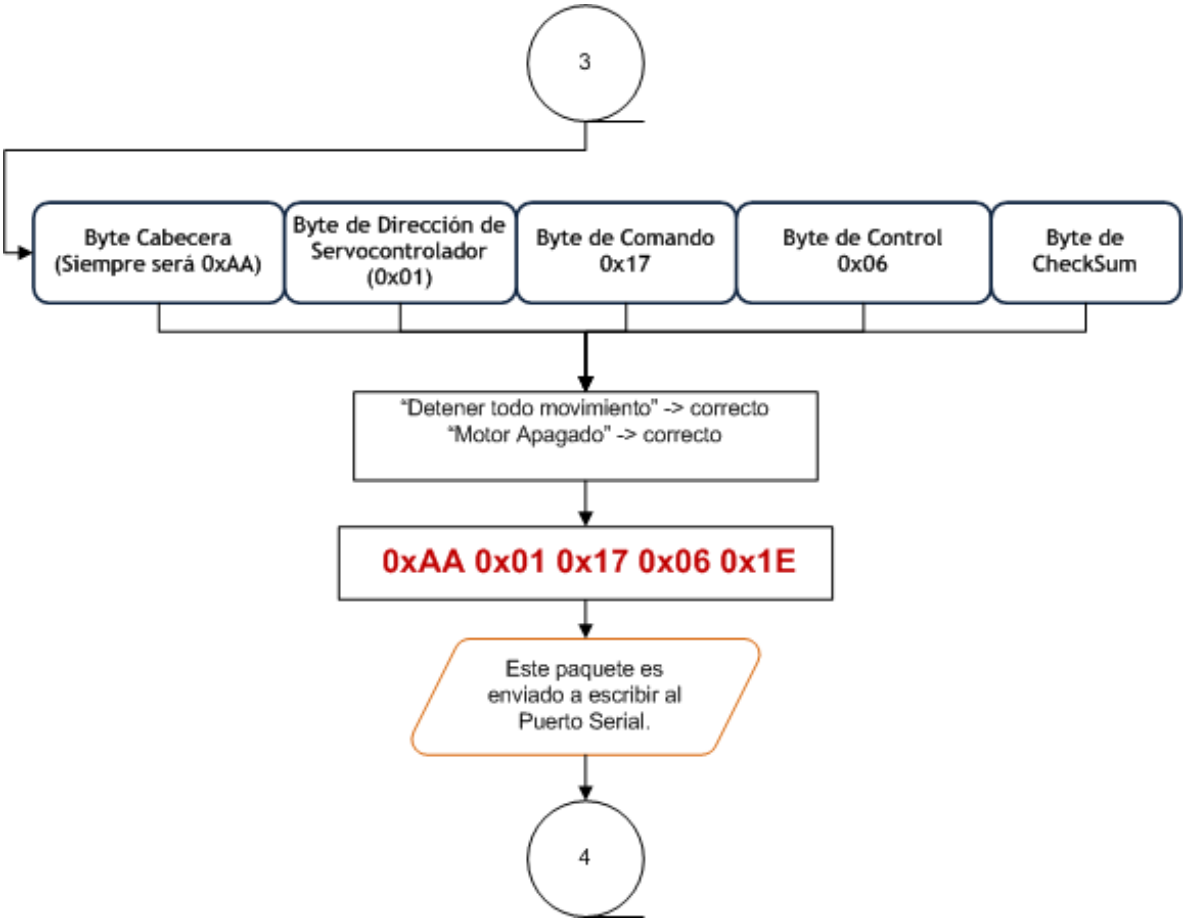


Figura 2.14: Diagrama de flujo que muestra el procedimiento realizado para detener motores y mantenerlos apagados.

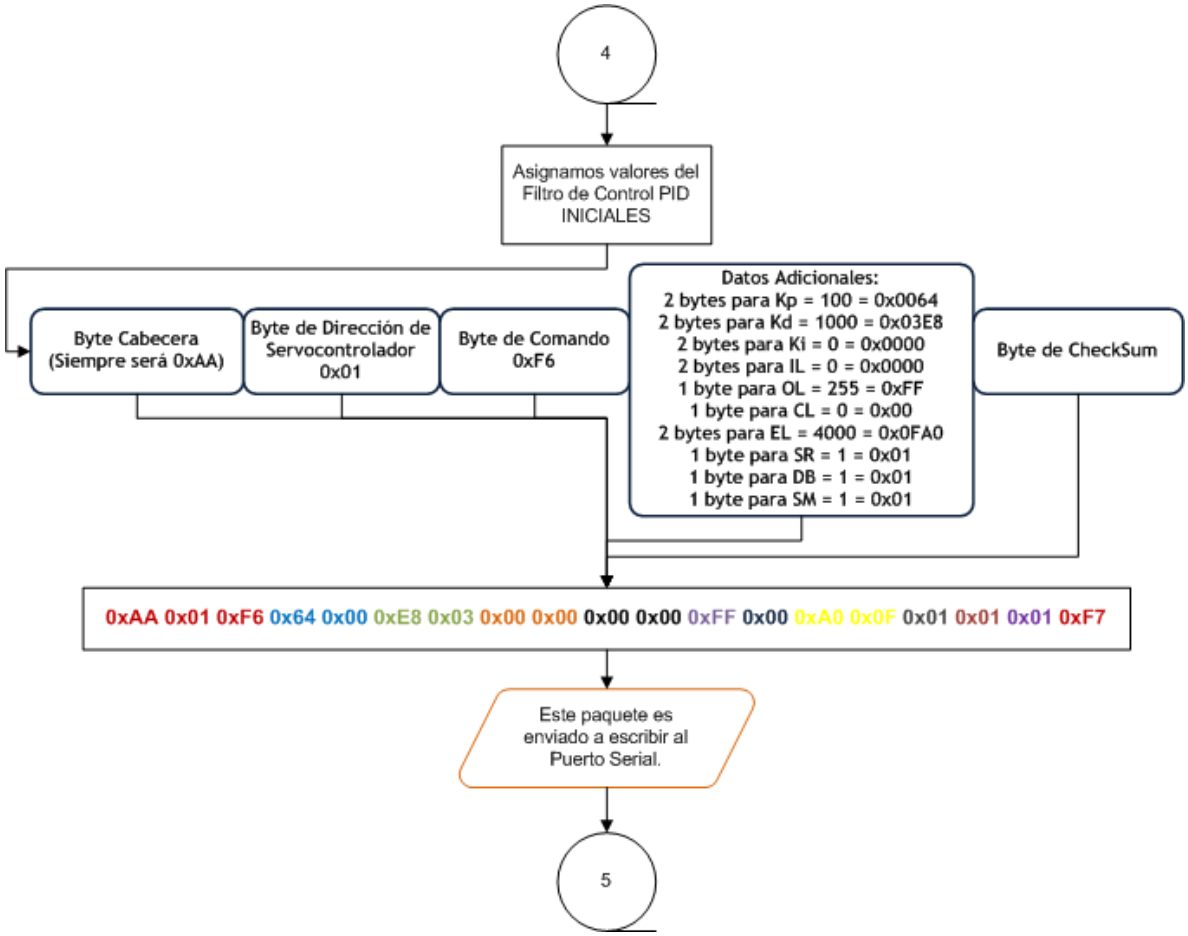


Figura 2.15: Diagrama de flujo que muestra el procedimiento realizado para asignar valores a la lsita de parámetros del filtro de control PID.

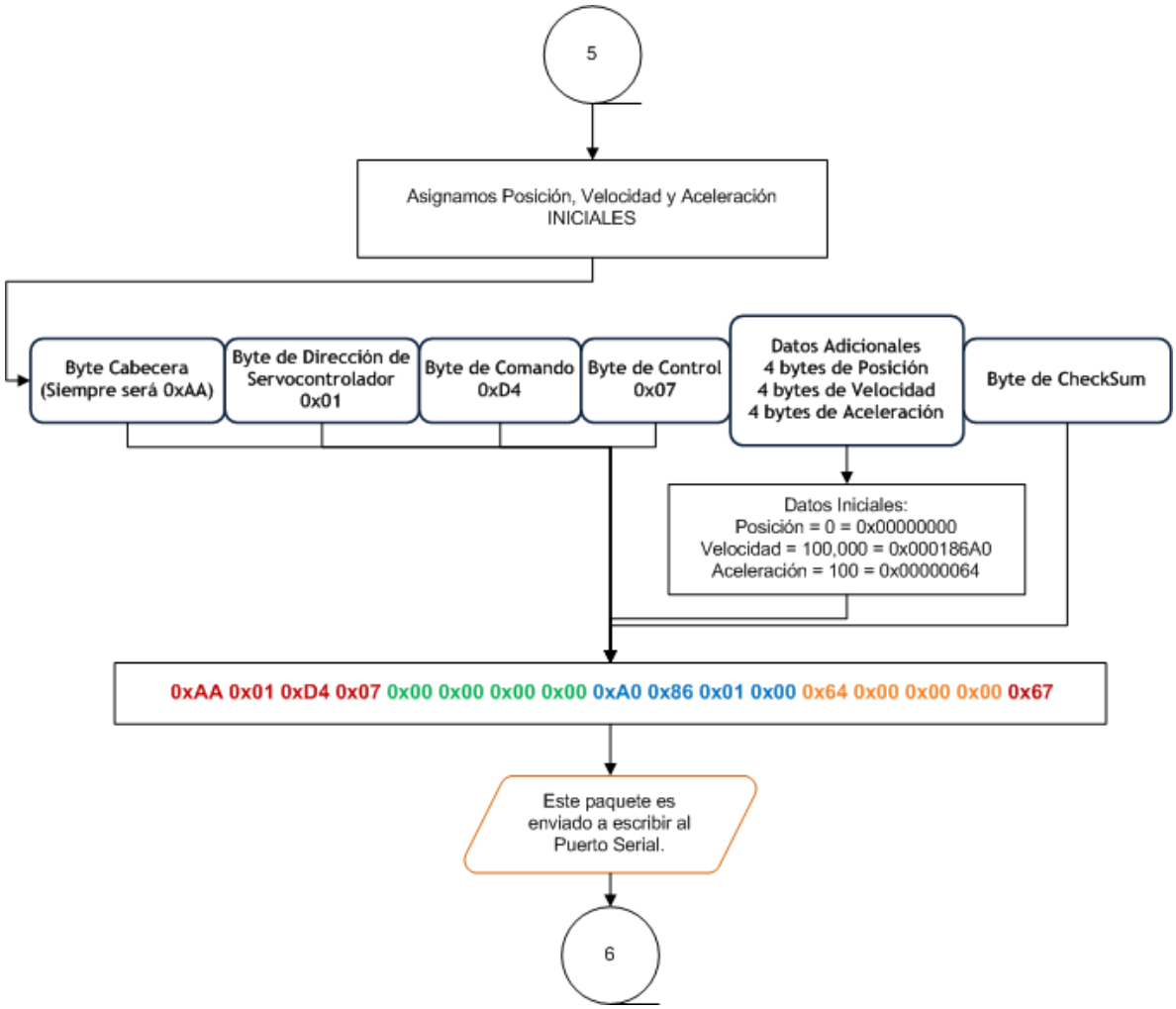


Figura 2.16: Diagrama de flujo que muestra el procedimiento realizado para asignar la posición, velocidad y aceleración iniciales.

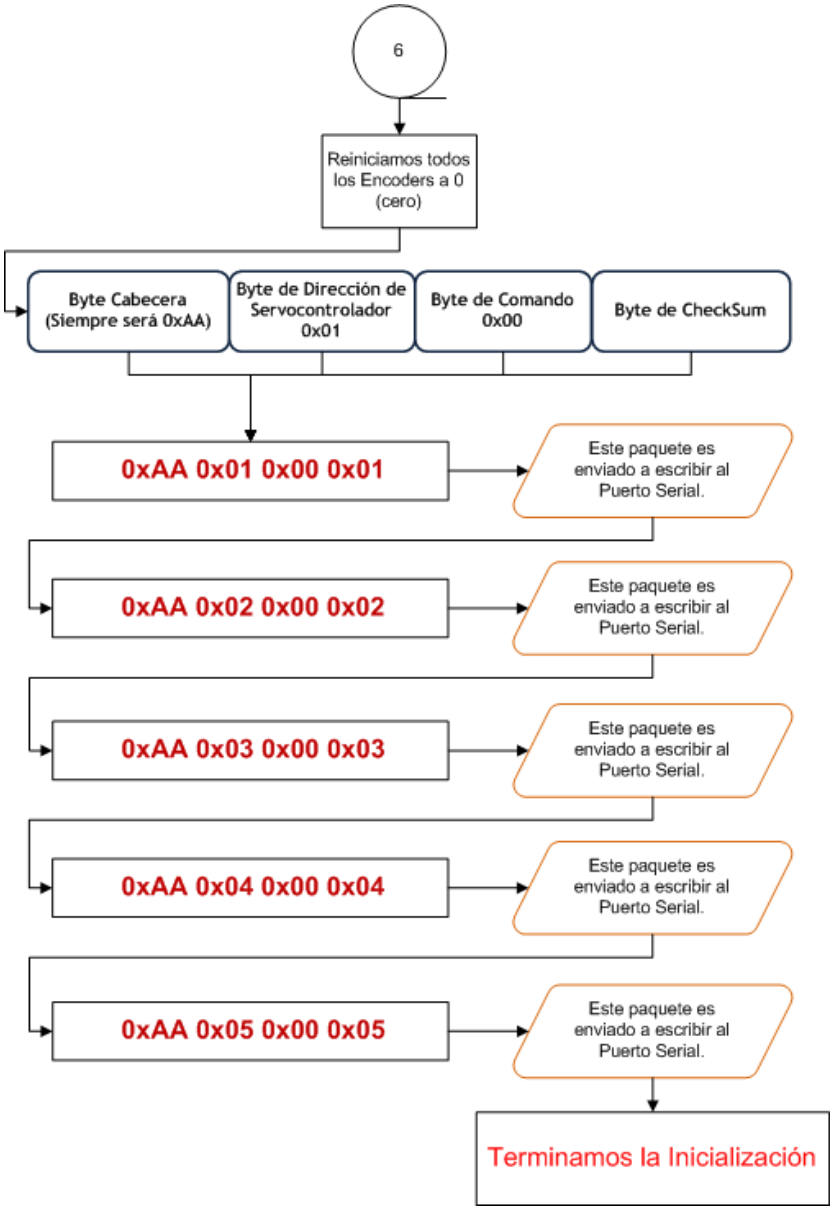


Figura 2.17: Diagrama de flujo que muestra el procedimiento realizado para reiniciar a 0 (cero) cada uno de los encoders del sistema.

Al finalizar estos cinco pasos, podremos anunciar que nuestra red de comunicación ha sido inicializada, por lo que los comandos de movimiento o cualquier otra acción que deseemos que ejecuten los servocontroladores ahora serán posibles. Habremos de haber notado la importancia de los **bytes de comando** y de **bytes de control**. Ambos bytes son configurables de acuerdo a la aplicación que necesitemos, y es por esto que obligadamente debemos revisar la hoja de especifícaciones del fabricante del servocontrolador que se encuentra disponible en [5].

Capítulo 3

Implemetación de trayectorias articulares

La generación de trayectorias articulares consiste en generar desplazamientos rotacionales ya sea en grupo o individuales. Lo que se desea primordialmente es la independencia de cada actuador de la red, de esta forma se poseerá capacidades de movimientos infinitos cuando el robot manipulador (*ROMMEL*) finalmente sea ensamblado.

El *byte de comando* (*0xD4*) y del *byte de control* (*0x17*) de configuración, son *bytes* destinados a la generación de movimientos del actuador, y mediante estos se realizaron pruebas de movimientos individuales para cada eje así como también movimientos en grupo.

Posteriormente haremos uso del *buffer* del servocontrolador para almacenar puntos previamente calculados por el software *MATLAB*©. Estos puntos son aproximados mediante la simulación de funciones de interpolación (ver sección 1.2 y 1.3). De esta forma se espera proseguir con la ejecución de trayectorias, y así los movimientos que ahora se realicen serán al seguimiento de cada punto almacenado en el *buffer* de cada tarjeta servocontroladora.

3.1. Lectura de encoder

Realizar la lectura de encoder es muy importante pues es una señal necesaria para conocer los desplazamientos del servomotor, con esta información podremos analizar posición y velocidad.

Para poder leer cada uno de los encoders se consideró realizar mediante programación un bloque de conmutación (ver Figura 3.1), muy similar a un multiplexor digital.

La lógica de este bloque es la siguiente; una variable de selección es destinada a realizar la conmutación del puerto bidireccional del encoder 1, del encoder 2, del encoder 3, del encoder 4 y del encoder 5 hacia la salida de lectura de encoder, de esta forma cada encoder será leído en momentos diferentes o secuencialmente. Así podremos leer el encoder de cada servomotor de la red de comunciación sin importar el orden de lectura, pues de esta forma que se ha propuesto se considera ordenada y muy adecuada.

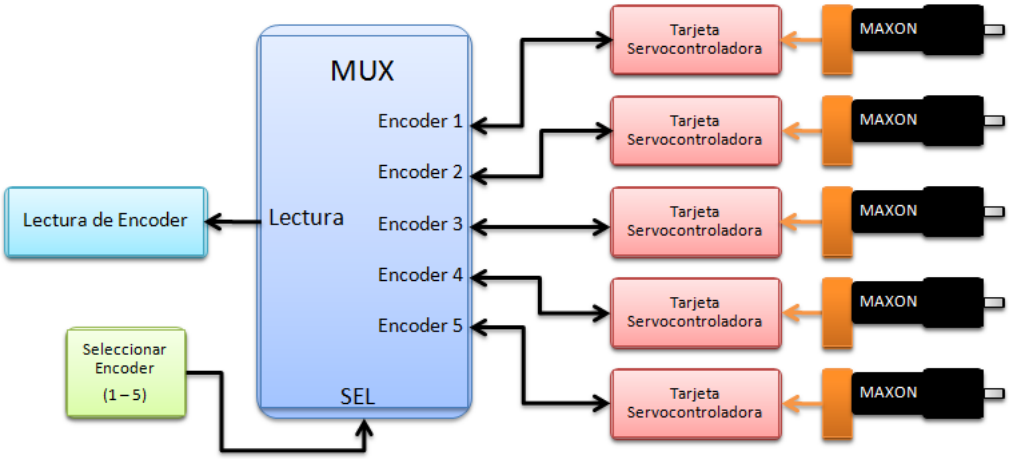


Figura 3.1: Diagrama que muestra el proceso que se realiza para la lectura de cada uno de los 5 encoders presentes en la red de comunicación. La lectura se realiza mediante la programación de un bloque tipo multiplexor, este solicitará información a cada tarjeta servocontroladora de acuerdo a la variable de entrada de selección.

Ya hemos establecido el procedimiento de conmutación de lectura para los encoders, y ahora que hemos seleccionado el encoder a leer, proseguimos con la solicitud de envío de información a la tarjeta servocontroladora mediante el siguiente *paquete de*

instrucción (ver Figura 3.2).

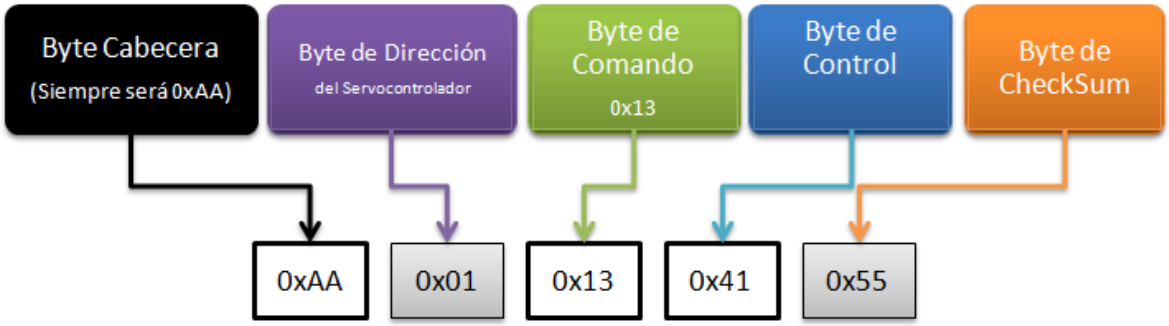


Figura 3.2: Estructura del *paquete de instrucción* para realizar la solicitud de envío de información del encoder al servocontrolador del servomotor 1, al *host*.

En la Figura 3.2 observamos el *paquete de instrucción* que debemos de enviar al servocontrolador para realizar la lectura del encoder del servomotor 1, y para realizar la lectura de los otros cuatro encoders, los *bytes* de dirección y de *checksum* deberán de ser actualizados con los valores que se observan en la Figura 3.3.

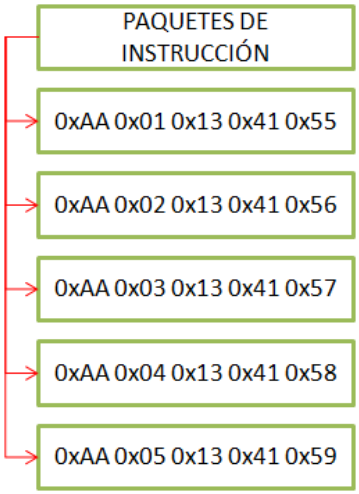


Figura 3.3: Paquetes de instrucción para solicitar el envío de información que registra el encoder.

El procedimiento que se realiza se muestra en la Figura 3.4 donde observamos que dicho procedimiento de lectura se repite de igual manera para los demás. La variable de selección *Sel* espera recibir un valor de 1 a 5, dependiendo del encoder que deseemos leer.

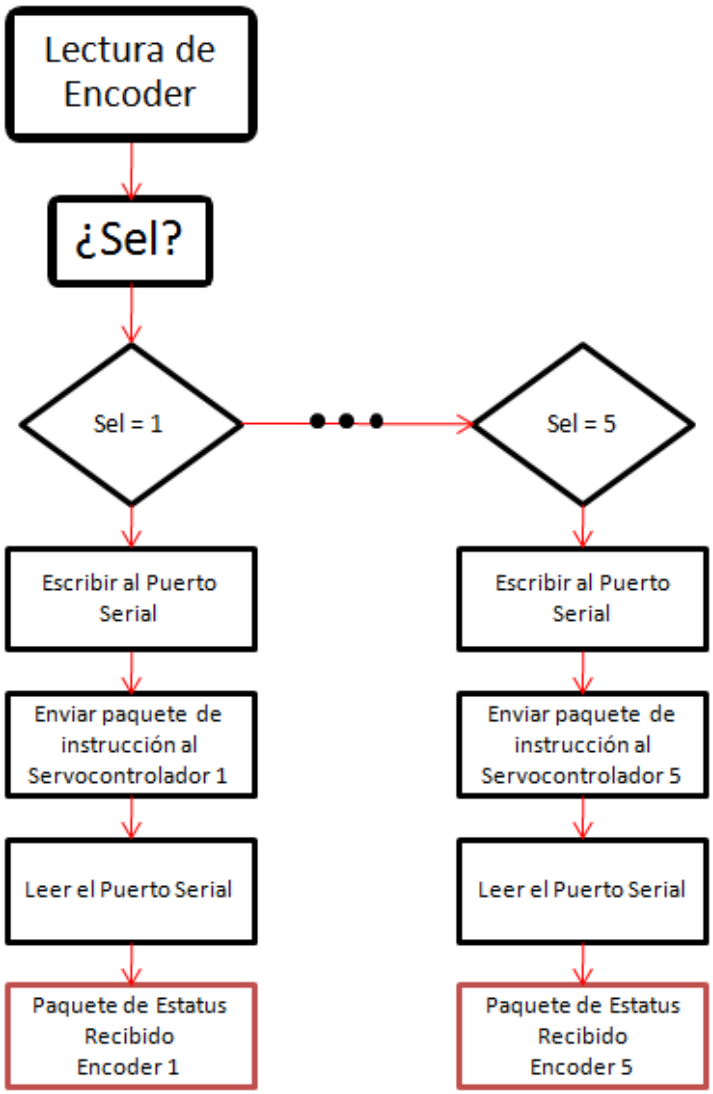


Figura 3.4: Diagrama de flujo que muestra el procedimiento que se realiza para la lectura de los 5 encoder de la red de comunicación.

3.2. Programación de movimientos por comando de instrucción

La programación de movimientos por comando de instrucción se realiza estableciendo los valores de posición, velocidad y aceleración deseadas. De esta forma al enviar el *paquete de instrucción* al servocontrolador, estaremos estableciendo la posición a la que queremos que el servomotor se desplace con cierta velocidad y aceleración previamente establecidas.

3.2.1. Creación del paquete de instrucción por comando de movimiento

El *paquete de instrucción* es similar al de la Figura 2.16, pero con la actualización de los parámetros de datos adicionales.

Revisaremos nuevamente la estructura del *paquete de instrucción* que debe de ser enviado al servocontrolador para ejecutar el movimiento deseado. Analizamos nuevamente en la Figura 3.5 los datos que deben ser modificados para poder producir variaciones en la posición, velocidad y aceleración del actuador.

Mediante el ejemplo mostrado en la Figura 3.5, supongamos que deseamos mover el servomotor a la posición $500,000$ y con una velocidad de $100,000$ y una aceleración de 100, que equivalen a los valores hexadecimales de $4C4B40Hex$, $186A0Hex$ y $64Hex$. Recordemos que el dato siempre serán enviado primero por su *byte menos significativo* hasta su *byte más significativo*.

Finalmente el *paquete de instrucción* que debe ser enviado al servocontrolador para desplazar el servomotor a la posición indicada con los parámetros de velocidiad y aceleración deseados se muestra en la Figura 3.6.

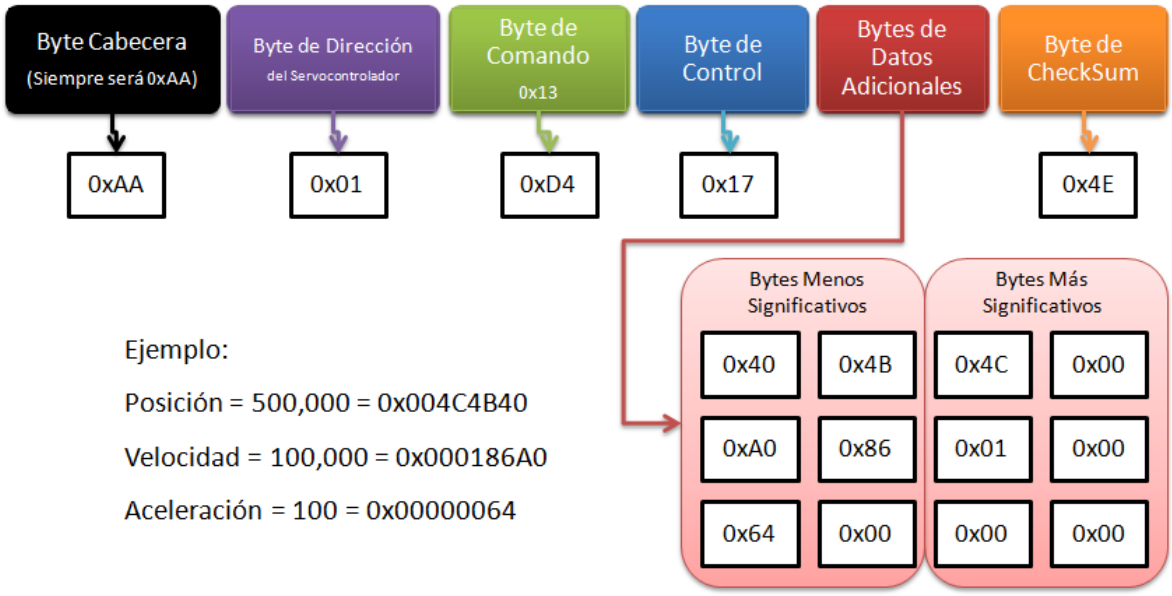


Figura 3.5: Estructura del *paquete de instrucción* que debemos enviar al servocontrolador para desplazar el servomotor de una posición a otra.

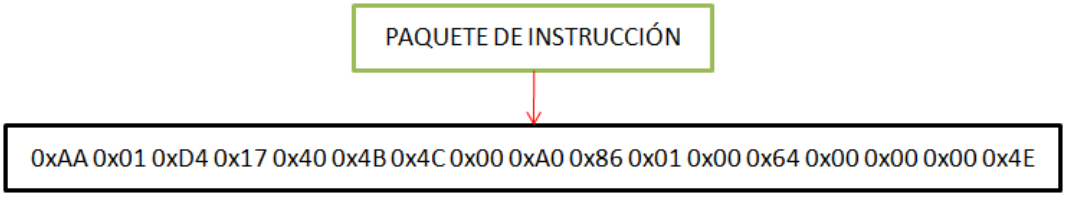


Figura 3.6: *Paquete de instrucción* que debe de recibir el servocontrolador para programar sus parámetros de desplazamiento.

Debemos también de recordar que durante la inicialización del sistema (ver sección 2.3.1), se establecieron condiciones iniciales y parámetros del filtro de control PID (ver Figura 2.15). Cuando nosotros ejecutemos el desplazamiento estos parámetros estarán vigentes y serán aplicados automáticamente en todos los desplazamientos que realicemos durante las pruebas de ejecución de movimientos.

Habiendo programado el movimiento (con valores iguales o diferentes para cada servocontrolador), ahora analizaremos en los siguientes apartados, cómo queremos que sean ejecutados dichos movimientos. Es decir, individualmente o todos en grupo.

3.2.2. Ejecución de movimientos individuales

Para inicial el movimiento en el servomotor 1 de la red e independientemente del estado de los demás servomotores, realizamos los siguientes pasos que observaremos en la Figura 3.7 donde reutilizaremos para este ejemplo, la configuramos de la trayectoria vista en la Figura 3.6.

Mediante el diagrama a bloques de la Figura 3.7 observamos el procedimiento que realizamos para desplazar el servomotor 1, suponiendo que se encuentra inicialmente en la posición 0 (cero), se desplazará hasta la posición $500,000$. El encoder le informa en todo momento a la tarjeta servocontroladora la posición del eje del servomotor.

Cuando el encoder detecta la posición $500,000$, el servocontrolador detiene inmediatamente el giro del rotor, finalizando el movimiento.

Una vez que hemos finalizado el movimiento, podremos enviar un nuevo *paquete de instrucción* para desplazar el servomotor a una nueva posición, ya sea en decremento o incremento al último valor de la posición detectada por el encoder.

Debemos de observar también que la habilitación del amplificador sólo se realizará durante la primera ejecución de trayectoria, en adelante podremos enviar el *paquete*

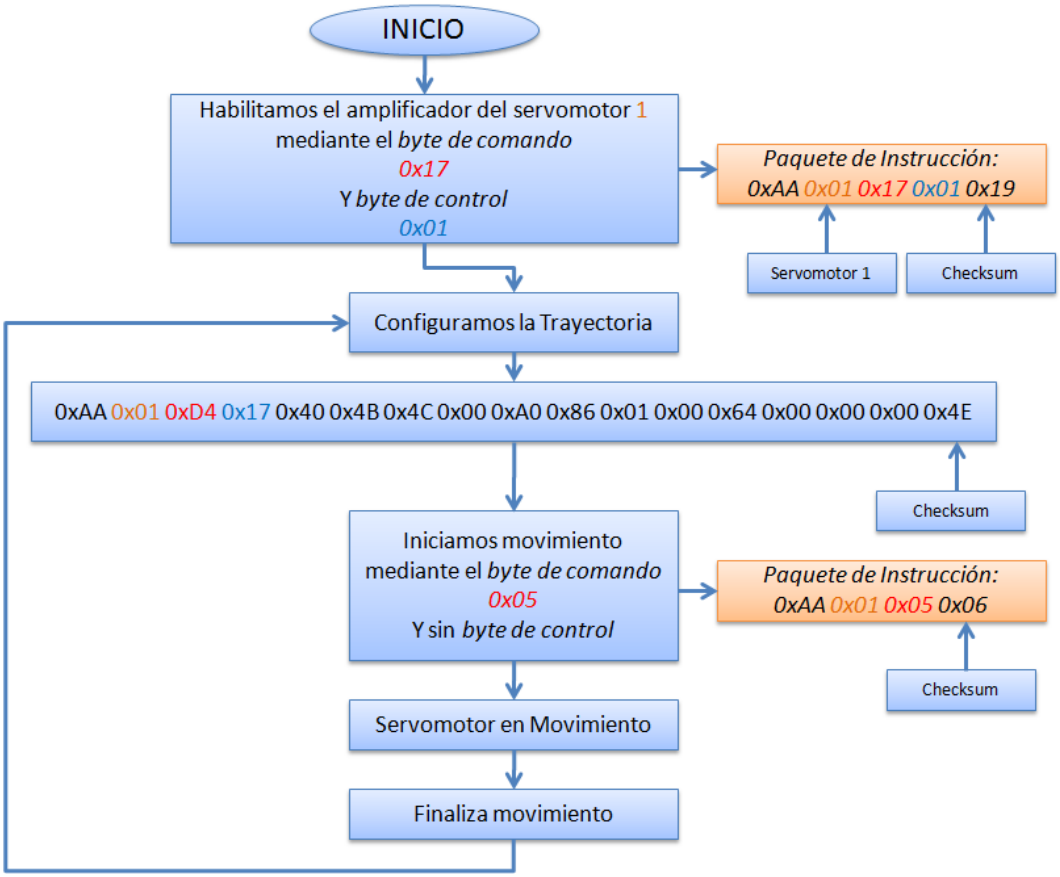


Figura 3.7: Observamos en este diagrama a bloques el *Paquete de instrucción* que habilita el amplificador de la tarjeta servocontroladora del servomotor 1, y además el *paquete de instrucción* que inicia el movimiento de su eje.

de instrucción de trayectoria y el *paquete de instrucción* de inicio de movimiento, siempre uno después del otro e infinidad de configuraciones de trayectoria que necesitemos para cada uno de los servomotores de la red.

3.2.3. Ejecución de movimientos en grupo

Para iniciar el movimiento en grupo, debemos de recordar los datos de inicialización del sistema de la Figura 2.12, donde observamos que además de haber direccionado cada servocontrolador de la red, también los hicimos pertenecer al mismo grupo con dirección *0x81*.

Esta dirección la utilizaremos para enviarle la orden de inicio de movimiento del grupo, es decir, todas aquellos servocontroladores que pertenezcan al grupo con dirección *0x81* deben de iniciar su trayectoria al mismo tiempo. El fabricante asegura la ejecución instantánea del movimiento de todos los miembros del grupo gracias al oscilar o cristal de cuarzo que poseen cada una de las tarjetas servocontroladoras.

Nuevamente debemos de recordar que cada *paquete de instrucción* que generemos, debe de ser enviado o escrito al puerto serial de la computadora de forma secuencial.

Observemos la Figura 3.8, donde observaremos el proceso que realizamos para ejecutar un movimiento en grupo. El procedimiento es totalmente similar a la ejecución de movimientos individuales, pero con la diferencia de que ahora enviaremos los *paquetes de instrucción* necesarios para cada una de las tarjetas servocontroladoras tanto de habilitación de amplificadores como de configuración de trayectorias. Dichas trayectorias pueden ser iguales, como en nuestro ejemplo; pero también pueden ser diferentes.

La gran diferencia aparece con el envío de un *paquete de instrucción* que ahora en vez de incorporar la dirección de dispositivo, incluye ahora la dirección de grupo *0x81*. El envío de este paquete producirá que todos aquellas tarjetas servocontroladoras que

pertenezcan al grupo ejecuten su movimiento al instante.

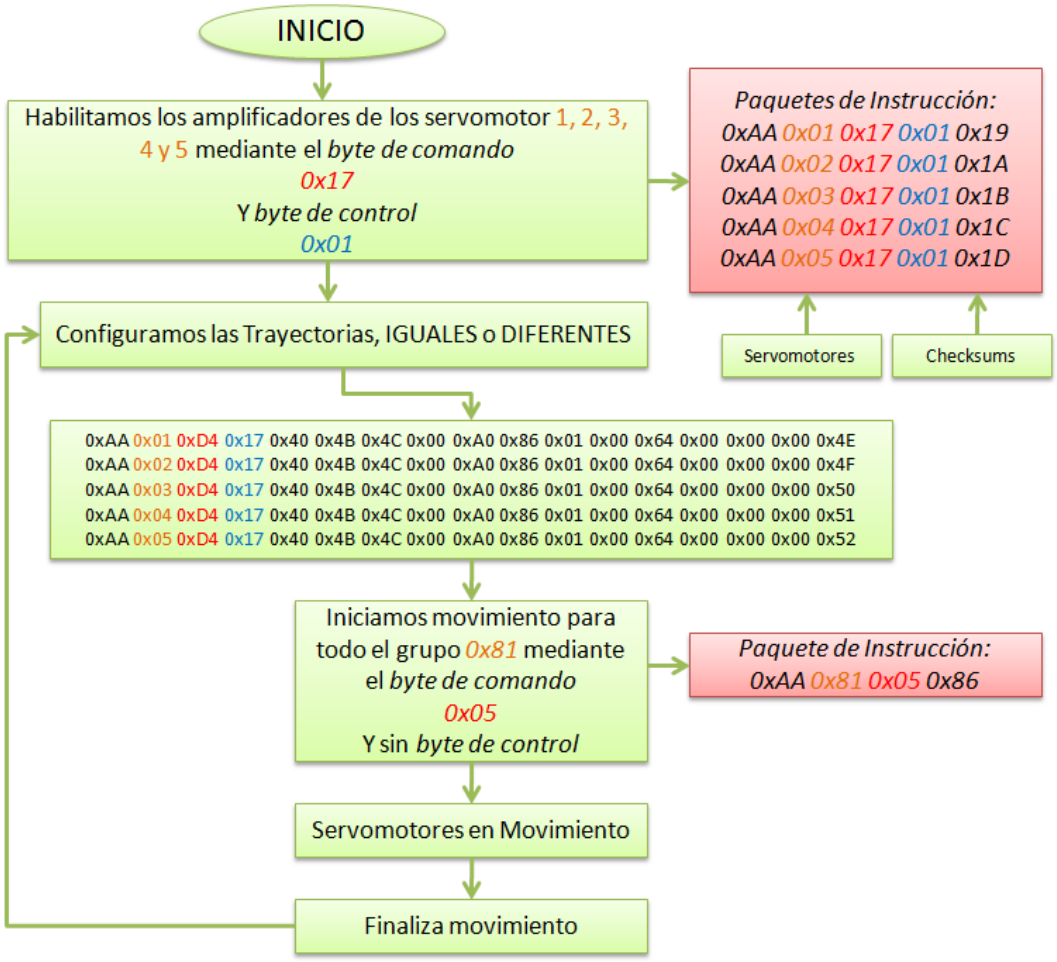


Figura 3.8: Observamos en este diagrama a bloques los *Paquete de instrucción* que habilitan los amplificadores de las tarjetas servocontroladoras de los 5 servomotores de la red, y además el *paquete de instrucción* que inicia el movimiento de todos los ejes al mismo tiempo.

Al igual que en el procedimiento de ejecución de movimientos individuales, también habilitamos los amplificadores una sóla vez. En adelante podremos realizar movimientos en grupo con igual o diferentes configuraciones de trayectorias.

3.3. Control de movimiento coordinado (CMC)

El robot manipulador debe de seguir trayectorias deseadas con el objetivo de realizar alguna tarea o actividad. Y para lograrlo, cada aritculación debe de recorrer una determinada trayectoria de forma individual. La meta primordial de este proyecto de investigación es implementar funciones de interpolación en el software Matlab©para aproximar puntos intermedios entre dos posiciones angulares (inicial y final) del rotor del servomotor. Siendo más precisos, se implementaron 3 funciones de interpolación básicas, tales como, la función de interpolación lineal, la función de interpolación cúbica y por último la función de interpolación a tramos. Cada servocontrolador comandará a su servomotor para realizar el seguimiento de cada punto y generar de esta forma el desplazamiento angular o trayectoria articular.

Nótese que únicamente simulamos las funciones de interpolación en Matlab©, y los puntos aproximados durante la simulación son guardados en un archivo de texto. Posteriormente el programa creado en lenguaje C, necesitará estos datos que han sido guardados en dicho archivo para realizar todas las operaciones necesarias y generar los paquetes de instrucción necesarios para la tarjeta servocontrolador.

Debemos de señalar que según el manual del fabricante, el buffer del servocontrolador puede almacenar temporalmente hasta *128* puntos de trayectoria. Al ser el servocontrolador un dispositivo nuevo y desconocido en funcionamiento y programación, en este proyecto nos interesa conocer cómo debe de ser la programación de la tarjeta para lograr que este opere en el modo CMC; y es por esto que se ha decido usar tan sólo *126* puntos de trayectoria o *126* espacios del *buffer* del *PIC-SERVO SC*. Una vez descifrado el procedimiento necesario para realizar la programación del servocontrolador, se ejecutarán trayectorias de prueba, generadas mediante funciones de interpolación.

Esta información será de importancia en la continuidad de proyectos posteriores,

pues se ofrecerá un programa prototipo que mostrará el procedimiento realizado para programar el servocontrolador, de esta forma se podrán generar nuevos programas que permitirán generar trayectorias más complicadas.

3.3.1. Creación de paquetes de instrucción con puntos de trayectoria

Recordemos que la tarjeta *PIC-SERVO SC* posee varios niveles de operación (ver Figura 1.20), y para la generación de trayectorias, el servocontrolador operará en el modo de control de trayectoria o modo de control de movimiento coordinado (CMC), documentado en la sección 1.8.

Para operar en el modo de control de movimiento coordinado se debe de ensamblar correctamente los *paquetes de instrucción*. Para operar el sistema en el modo CMC, la creación correcta de estos *paquetes de instrucción* es muy importante, pues con el mínimo error binario, el paquete puede programarse erroneamente en la tarjeta.

Cuando simulamos algunas de las tres funciones de interpolación (lineal, cúbica y a tramos) en Matlab©, aproximamos 127 puntos de trayectoria muestreados a $30Hz$, que nos ofrece un tiempo máximo de ejecución de trayectoria de $4{,}2s$. Cada valor númerico de trayectoria representa un valor de *cuenta de encoder*, es decir, representará una posición deseada representada en *cuentas de encoder*. Estos puntos son guardados en un documento de texto o bloc de notas, es decir este documento contendrá 127 valores numéricos que representan *cuentas de encoder* o posiciones deseadas.

Una vez que hemos guardado nuestro archivo con los 127 puntos de trayectoria, ejecutamos nuestro programa prototipo de prueba del sistema de control que se ha creado. En la Figura 3.9 observamos un diagrama a bloques que muestra que la participación de Matlab©en el proyecto es limitada a generar los puntos y posteriormente a graficar

los resultados.

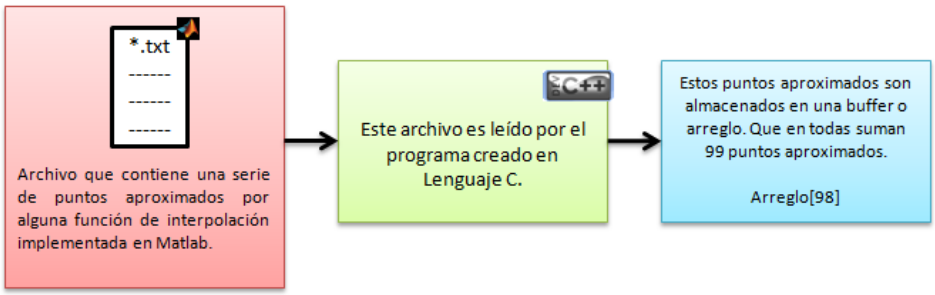


Figura 3.9: Diagrama de bloques que muestra los *paquetes de instrucción* que deben de enviarse al puerto para desconectar el servomotor 1 de la red.

Al ejecutar nuestro programa protipo, el programa inicializa la red de módulos *PIC-SERVO SC* (ver Figura 3.10). Enseguida establece la dirección única de cada módulo (del módulo 1 al módulo 5). Después se establece la velocidad de transferencia del puerto serial y la red de comunicación a $19,200 baudios/s$. Se asignan parámetros operativos por cada módulo servocontrolador mediante el *comando de instrucción* $0xF6$, relevante al establecimiento de ganancias del filtro PID, y finalmente, se inicializan los 5 encoders del sistema a *cero cuentas* [10].

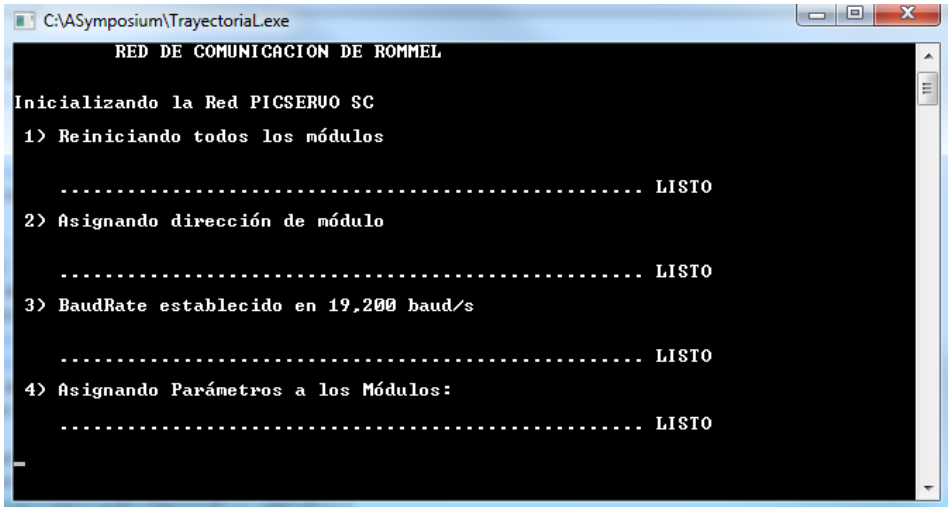


Figura 3.10: Inicialización de la red de comunicación del sistema.

Mediante el *byte de comando 0xnD*, donde $n = 0, 2, 4, 6, 8, A, C, E$; podremos en-

samblar paquetes que contengan desde uno a siete puntos de trayectoria. Este *byte de comando* le indicará a la tarjeta cuantos puntos de trayectoria le estamos enviando a almacenar al *buffer*. Cada punto de trayectoria será representado mediante cantidades de $16bits$ $(2bytes)$, y por esto es que el valor de n representará el número de *bytes* enviados por puntos de trayectoria. Como ejemplo, supongamos que deseamos enviar tres puntos de trayectoria, entonces el valor de $n = 6$, porque cada punto de trayectoria es representando en cantidades de *2 bytes* y *2 bytes x 3 puntos = 6*. En [10] podremos encontrar mayor información de este *byte de comando* para añadir puntos de trayectoria al *buffer*.

En seguida el programa abre el archivo con los 127 puntos de trayectoria guardados destinados para el servomotor 1, es importante observar en la Figura 3.11, que guardamos en total 127 puntos de trayectoria obtenidas mediante la función de interpolación lineal. La razón por la que sólo se envían 126 puntos de trayectoria al *buffer* del servocontrolador, es porque lo que se envía no son los puntos aproximados, sino las diferencias entre los puntos adyancentes, que en total serán 126 valores que representarán en *buffer* 126 posiciones deseadas (en el servocontrolador los llamaremos puntos de trayectoria). Al ejecutarse el movimiento, el servomotor deberá desplazarce desde la posición cero, hasta la posición *108000 cuentas de encoder*, que representan 2 revoluciones del servomotor.

El procedimiento que se enumera a continuación, es para generar los *paquetes de instrucción*, este procedimiento será el mismo para cada uno de los servocontroladores salvo el *byte de dirección* de módulo. En total, se ensamblarán *18 paquetes de instrucción* que contiene 126 puntos de trayectoria, muestreadas a $30Hz$. Recordemos que cada paquete siempre inicializará por un *byte de cabecera 0xAA*, seguida por el *byte de dirección* y el *byte de comando 0xnD*. Como ya hemos establecido que ensamblaremos *18 paquetes de instrucción*, quiere decir que para enviar los 126 puntos de trayectoria

al *buffer*, cada paquete deberá contener *7* puntos de trayectoria, pues $7x18 = 126$. Esto quiere decir que el *byte de comando* estará establecido a *0xED*.

Se ensamblaran *18 paquetes de instrucción* de acuerdo a la estructura que podremos observar en la Figura 3.12.

Entonces, el programa realiza el siguiente procedimiento:

1. Abre el archivo de texto, lee y almacena en una variable tipo arreglo los *127*

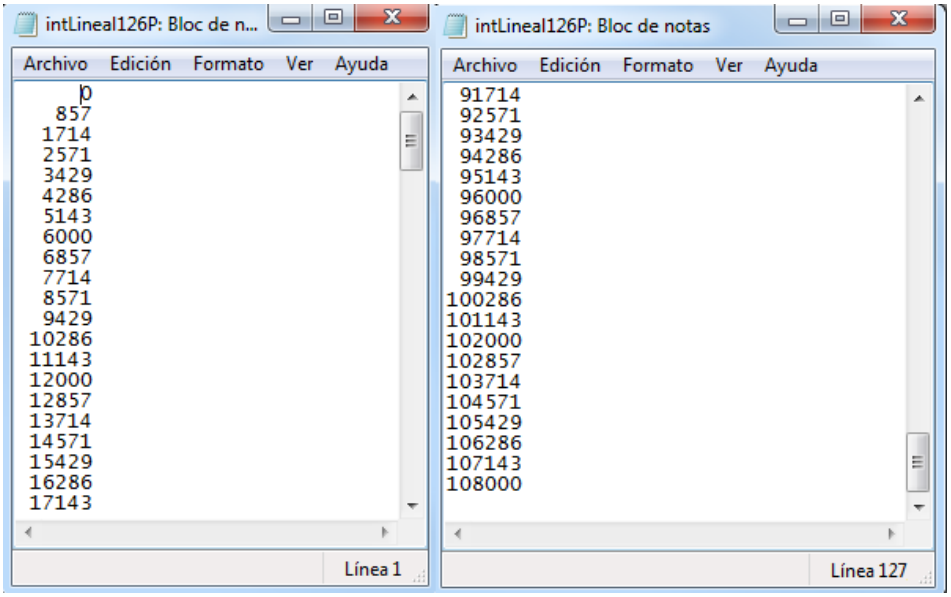


Figura 3.11: Archivo de bloc de notas que contiene 127 puntos de trayectoria.

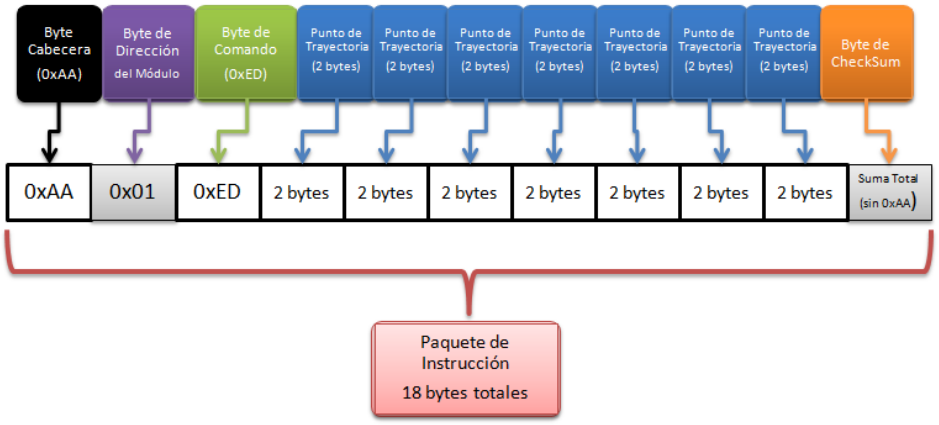


Figura 3.12: Estructura general del paquete de instrucción que contendrá 7 puntos de trayectoria.

puntos de trayectoria aproximados mediante Matlab©.

2. Mediante un algoritmo se obtienen las diferencias entre puntos adyacentes, obteniendose en total *126 diferencias* que representarán *126 puntos de trayectoria*. Estos se guardan en una nueva variable de tipo arreglo, y se cierra el archivo de texto.

3. Ahora, el proceso que realiza el programa a continuación, será el mismo para los 126 puntos de trayectoria. Recordemos que cada punto de trayectoria será un valor de *16 bits*, entonces en la Figura 3.13 observamos lo siguiente:

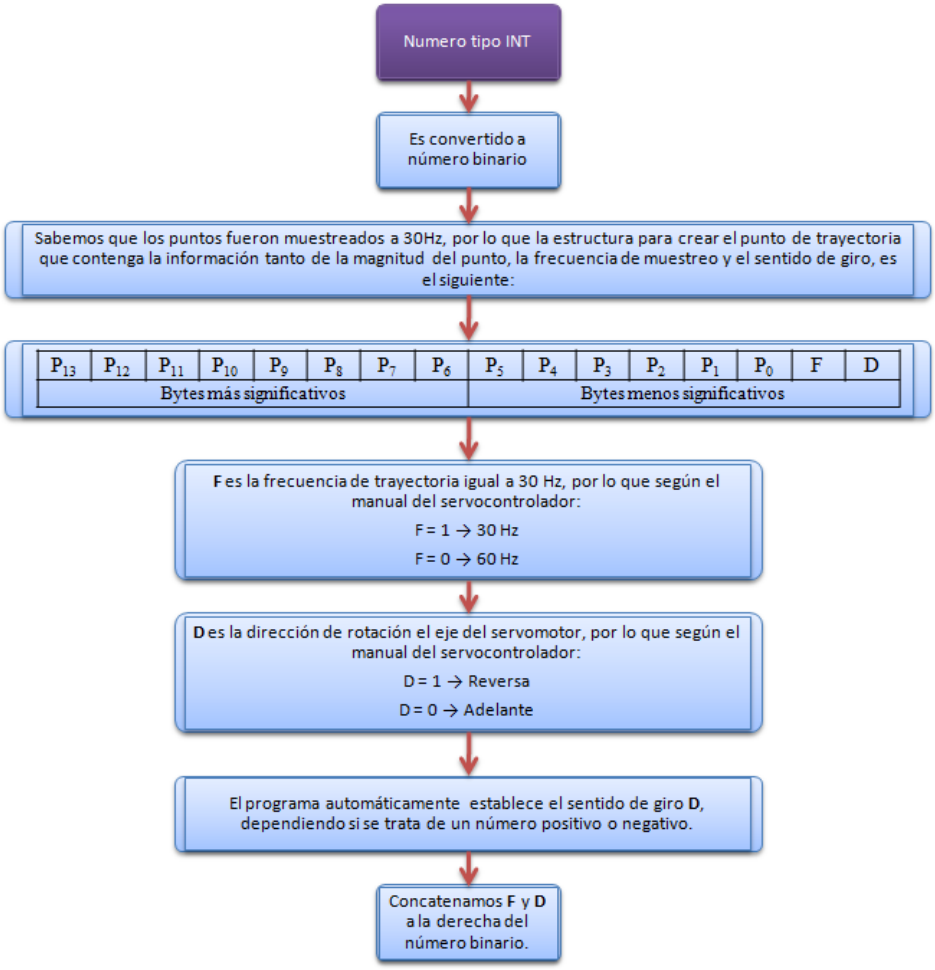


Figura 3.13: Estructura para ensamblar un punto de trayectoria con información de frecuencia de trayectoria y sentido de dirección de giro.

4. Ahora que hemos definido la constante **F** y la variable **D**, y se han concatenado a la derecha del número binario, observamos que el número sufre un cambio de magnitud debido a la concatenación de **F** y **D**.

5. Vamos a generar el primer punto de trayectoria, tal y como lo realizará el programa prototipo. Primero observemos la Figura 3.11, entonces la diferiencia entre los primeros dos puntos aproximados es $857 - 0 = 857$, por lo que al convertirlo a número binario, obtendremos 1101011001. Entonces observemos la Figura 3.14.

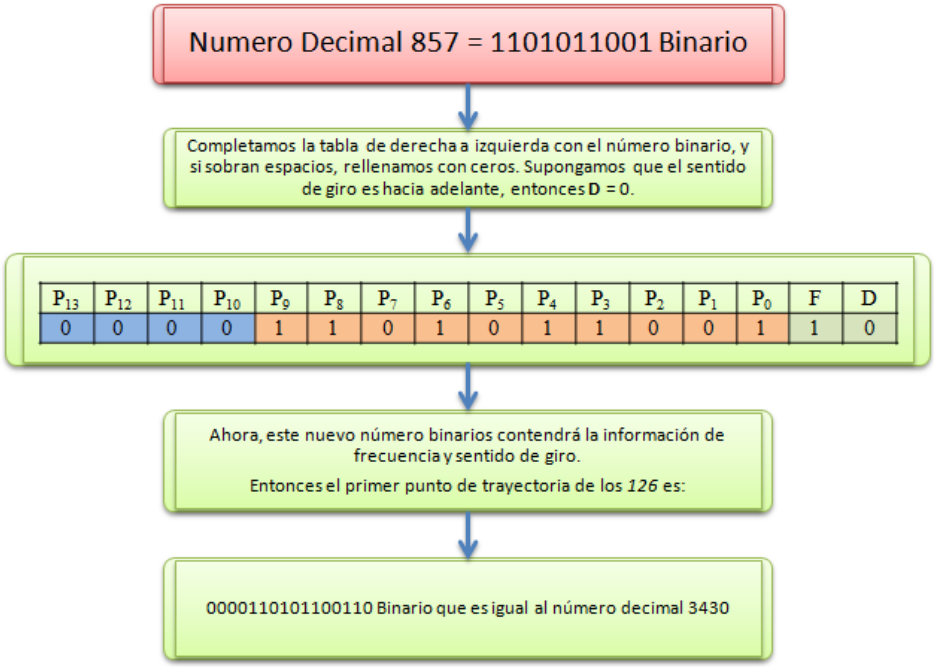


Figura 3.14: Codificación del nuevo punto de trayectoria.

6. Ahora tenemos el número codificado con la frecuencia del punto de trayectoria y el sentido de giro, por lo que el punto de trayectoria codificado tendrá una magnitud de $3430_{decimal}$, muy diferente a 857.

7. Una vez que el programa a obtenido el nuevo número binario, necesita ahora formar los *2 bytes* hexadecimales, por lo que el número binario es convertido a un

numero hexadecimal y entonces separa el *byte más significativo* del *byte menos significativo* e invierte el orden. Observemos la Figura 3.15.

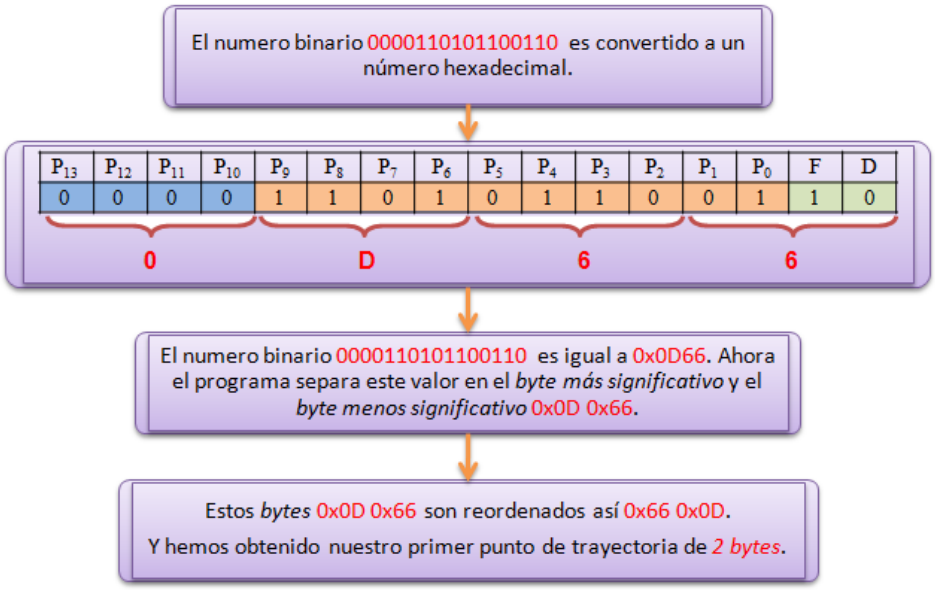


Figura 3.15: Convertimos el número binario a un número hexadecimal y separamos ambos *bytes*.

8. Entonces ya hemos obtenido el primer punto de trayectoria de *2 bytes* (*0x66 0xDD*). Por lo que el proceso se repite para obtener los siguientes 125 puntos de trayectoria restantes.

9. Finalmente, una vez obtenido los 126 puntos de trayectoria, procedemos a ensamblar el primer paquete de los *18 paquetes de instrucción* totales a enviar al servocontrolador. Supongamos que deseamos enviar los paquetes al *buffer* del servocontrolador 5, por lo que en la Figura 3.16 observamos cómo quedarían los *18 paquetes de instrucción*. Debemos notar que como los puntos fueron aproximados por interpolación lineal las diferiencias idealmente son de la misma magnitud entre todos los puntos.

10. En la Figura 3.17 observamos la pantalla donde ahora se muestran cómo quedarían los *18 paquetes de instrucción*, si los puntos de trayectoria son aproximados por

la función de interpolación cúbica. En esta figura sí observaremos la diferencia y también que el programa prototipo funciona correctamente.

11. Finalmente, los paquetes son enviados y almacenados en el *buffer* del servocontrolador 5.

3.3.2. Ejecución de trayectorias individuales

Para ejecutar la trayectoria almaceda y representada mediante 126 puntos de trayectoria almacenados en el *buffer* de determinado servocontrolador, hacemos uso del *byte de comando 0xnD*, donde $n = 0, 2, 4, 6, 8, A, C, E$. Para ejecutar la trayectoria programada, hacemos $n = 0$ y obtenemos el *byte de comando* (*0x0D*). En la Figura 3.18 observaremos los *5 paquetes de instrucción* posibles para ejecutar la trayectoria almacenada en determinado servocontrolador.

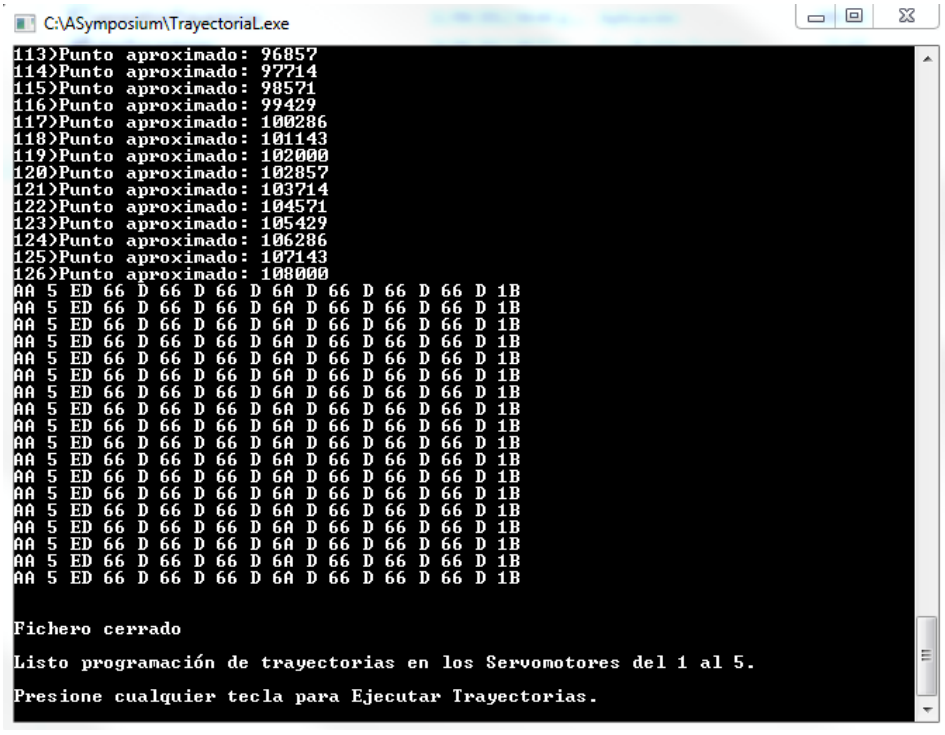


Figura 3.16: El programa ensambla 18 paquetes con la estructura previamente definida, y que contienen los puntos de trayectoria codificados.

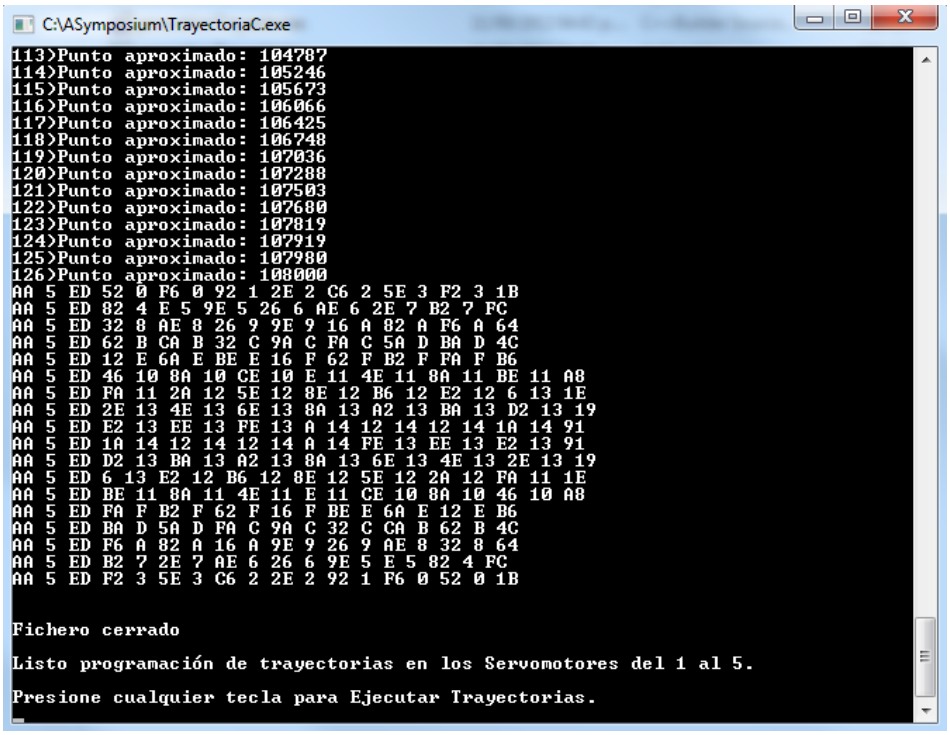


Figura 3.17: El programa ensambla 18 paquetes con la estructura previamente definida, y que contienen los puntos de trayectoria codificados.

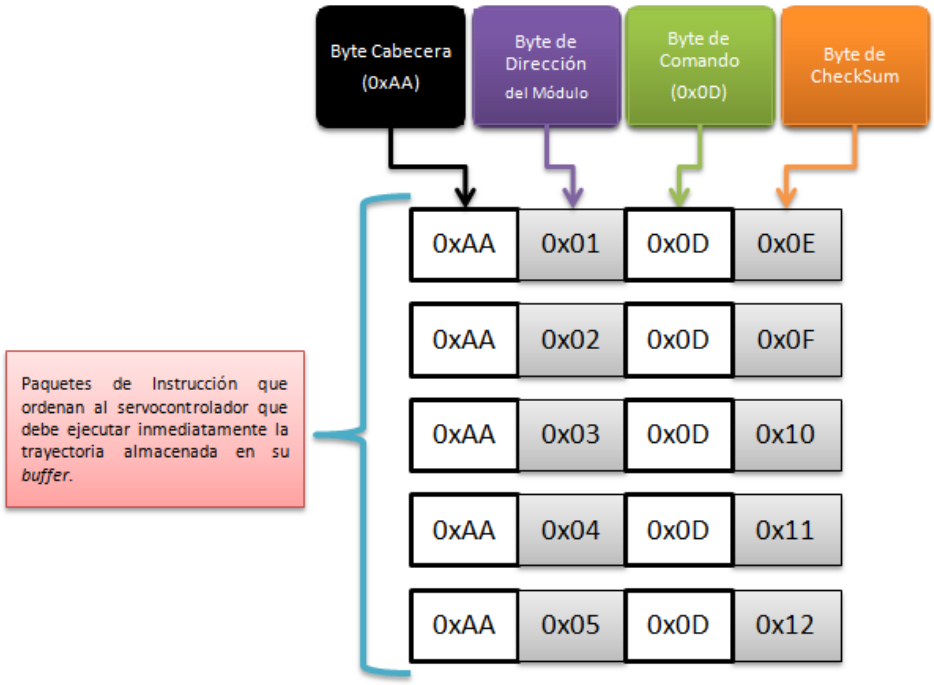


Figura 3.18: Paquetes que le indican a determinado servocontrolador que debe de ejecutar la trayectoria almacenada en su *buffer*.

3.3.3. Ejecución de trayectorias grupales

Ahora recordemos que cuando se incializa el sistema, se direcciona cada módulo con un valor único, pero además el paquete enviado para programar la dirección de módulo mediante el *byte de comando* (*0x21*), contiene además la dirección de grupo establecida de forma general mediante el *byte 0xFF*. Este parámetro programado nos permite ensamblar paquetes que indiquen alguna orden a todas las tarjetas servocontroladoras. En la Figura 3.19 observaremos el *paquete de instrucción* que ordena ejecutar las trayectorias al mismo tiempo en todas las tarjetas servocontroladoras.

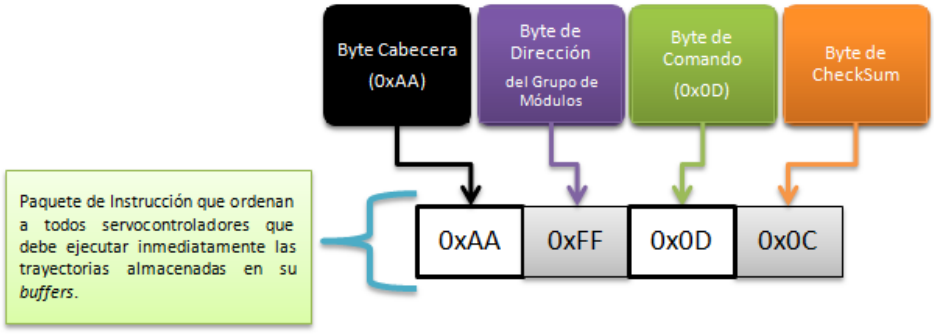


Figura 3.19: Paquete de instrucción que le indican al grupo de servocontroladores que deben de ejecutar la trayectoria almacenada en sus respectivos *buffers*.

3.4. Apagado del servomotor

Ya hemos generado movimientos individuales y movimientos en grupo, ya hemos ejecutado trayectorias mediantela programación de puntos de posición en los *buffers* de los servocontroladores. Ahora hemos de comentar el apagado correcto de cada servomotor. Es muy importante señalar que no debemos desenergizar el sistema sin antes haber deshabilitados todos los amplificadores de las tarjetas servocontroladoras.

El procedimiento que debemos de realizar para detener, apagar y desconectar el servomotor se muestra en la siguiente Figura 3.20, observaremos los pasos y las opciones

de *bytes de control* para detener el servomotor, así como el *byte* de apagado y el *byte* de deshabilitación del amplificador de la tarjeta servocontroladora.

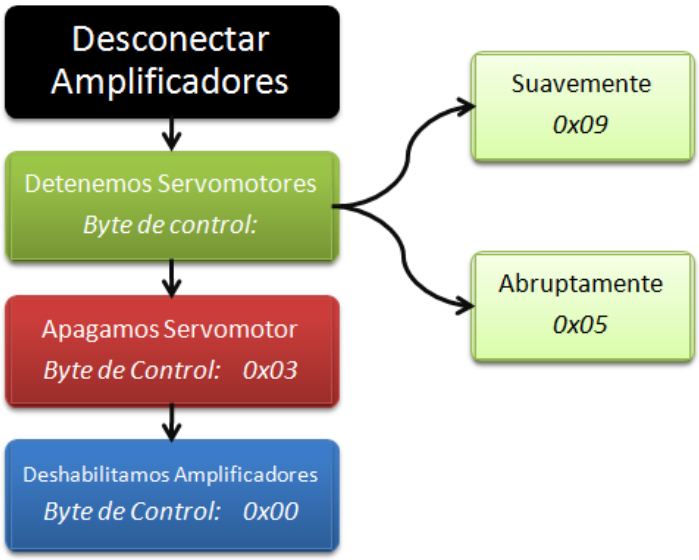


Figura 3.20: Diagrama de bloques que muestra el apagado del servomotor.

Como ejemplo, en la Figura 3.21 observamos los *paquetes de instrucción* que debemos de enviar o escribir al puerto serial de la computadora para apagar el servomotor 1. Observemos que podemos detener el servomotor de forma abrupta o de forma suave, esto depende de la aplicación que estemos realizando.

Por el momento sólo se han probado estos *paquetes de instrucción* para apagar individualmente cada servomotor, por lo que no se han realizados pruebas de apagado en grupo. Sin embargo, es suficiente para seguir avanzando en el proyecto de investigación.

Tal y como se ha venido comentado durante el desarrollo del documento; que para formar el *byte de control* se hace uso de las opciones disponibles del *byte de comando*, estas opciones las pueden consultar en [5] y tambien en la documentación disponible en la página web del fabricante [10].

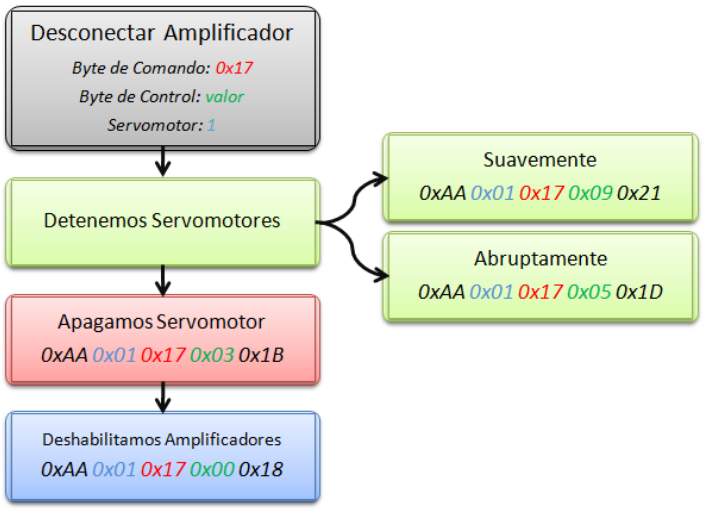


Figura 3.21: Diagrama de bloques que muestra los *paquetes de instrucción* que deben de enviarse al puerto para desconectar el servomotor 1 de la red.

Capítulo 4

Ejecución de trayectorias articulares

En este capitulo hablaremos de las herramientas utilizadas para la generación de trayectorias articulares, así como, de la aplicación de los *softwares Matlab©y Bloodshed Dev-C++* en este proyecto.

También hablaremos de todas aquellas condiciones que fueron consideradas para lograr la ejecución de la trayectoria marcada por los puntos calculados mediante *Matlab*. Debemos de señalar la importancia del programa creado en lenguaje C, que nos permitirá utilizarla como una herramienta para realizar la generación de *paquetes de instrucción* y posteriormente el proceso necesario para realizar el almacenamiento de los puntos de trayectoria sobre cada una de las tarjetas servocontroladoras.

4.1. Cálculo de puntos de trayectoria

Se consideraron 3 tipos de funciones de interpolación para realizar el cálculo de puntos de trayectoria entre dos puntos o posiciones radiales. Estas funciones fueron la de interpolación lineal, de interpolación cúbica y la de interpolación a tramos.

Mediante la simulación de estas tres funciones en el software de Matlab©, se aproximaron de cada una 127 puntos de trayectoria, considerando los siguiente; el *buffer* del

servocontrolador puede almacenar un total de 128 puntos de trayectoria, sin embargo, para facilidad de la programación se consideró fijar y generar *18 paquetes de instrucción* con el mayor número de puntos posibles que puede contener cada paquete (7 puntos máximos por paquete). De esta forma se generó un algoritmo capaz de ensamblar *18 paquetes* que contienen un total de 7 puntos de trayectoria cada uno.

Según lo requerimientos de servocontrolador, y para poder mantener cierto dominio de la duración (tiempo) de la trayectoria, se requiere que cada punto de trayectoria que sea enviado al *buffer* se encuentre muestreado en intervalos ya sea de 30, 60 o $120Hz$.

Para este proyecto el algoritmo fue programado para crear los *paquetes de instrucción* con la información de muestreo a $30Hz$ y con sentido de giro variable según sea un punto de trayectoria positivo o un punto de trayectoria negativo.

En la Figura 4.1 observamos la gráfica de posición y de velocidad obtenidas al aplicar el interpolador lineal.

En la Figura 4.2 observamos la gráfica de posición y de velocidad obtenidas al aplicar el interpolador cúbico.

En la Figura 4.3 observamos la gráfica de posición y de velocidad obtenidas al aplicar el interpolador a tramos.

El desplazamiento siempre iniciará desde la posición *0 cuentas de encoder*. El eje del servomotor al ir girando deberá de aproximarse a dichos puntos calculados, a manera de ir avanzando de punto a punto.

La simulación de estas tres funciones arrojan tres documentos que almacenan cada uno *127 puntos de trayectoria* muestreados a *30 Hz*, y la trayectoria finaliza en la posición *108000 cuentas de encoder* o 720° (ver Figura 4.5).

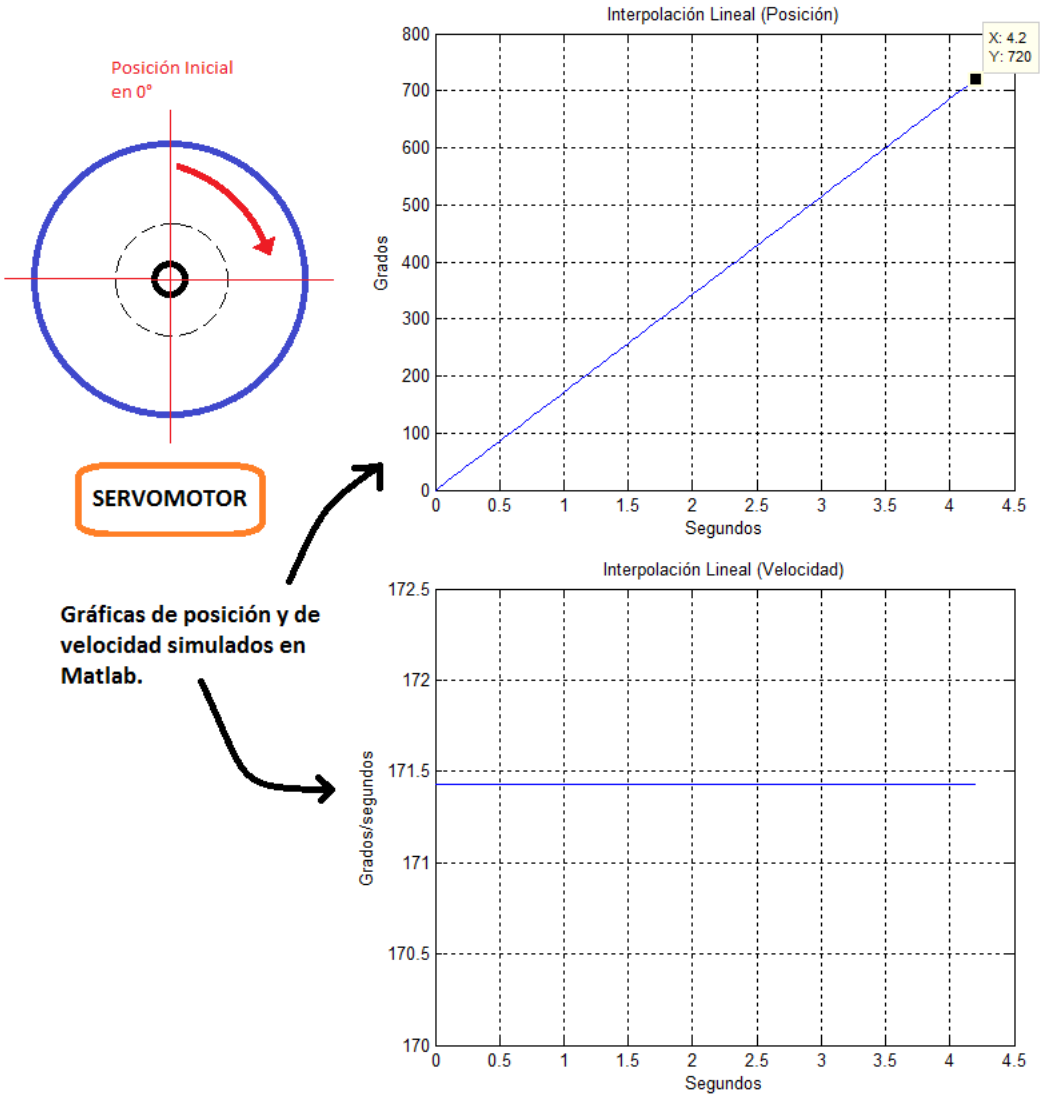


Figura 4.1: Gráfica de posición y de velocidad obtenidas de la simulación de la función de interpolación lineal.

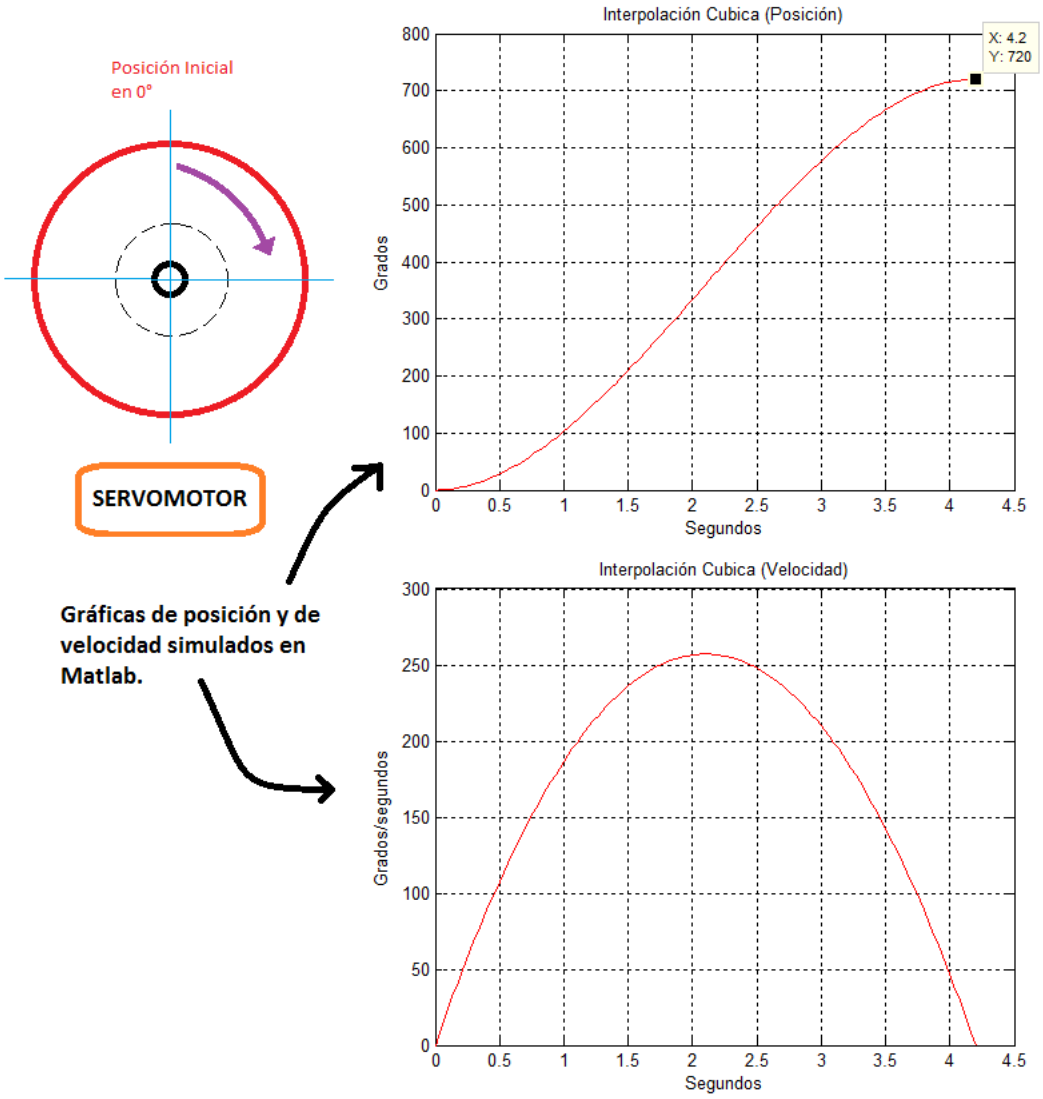


Figura 4.2: Gráfica de posición y de velocidad obtenidas de la simulación de la función de interpolación cúbica.

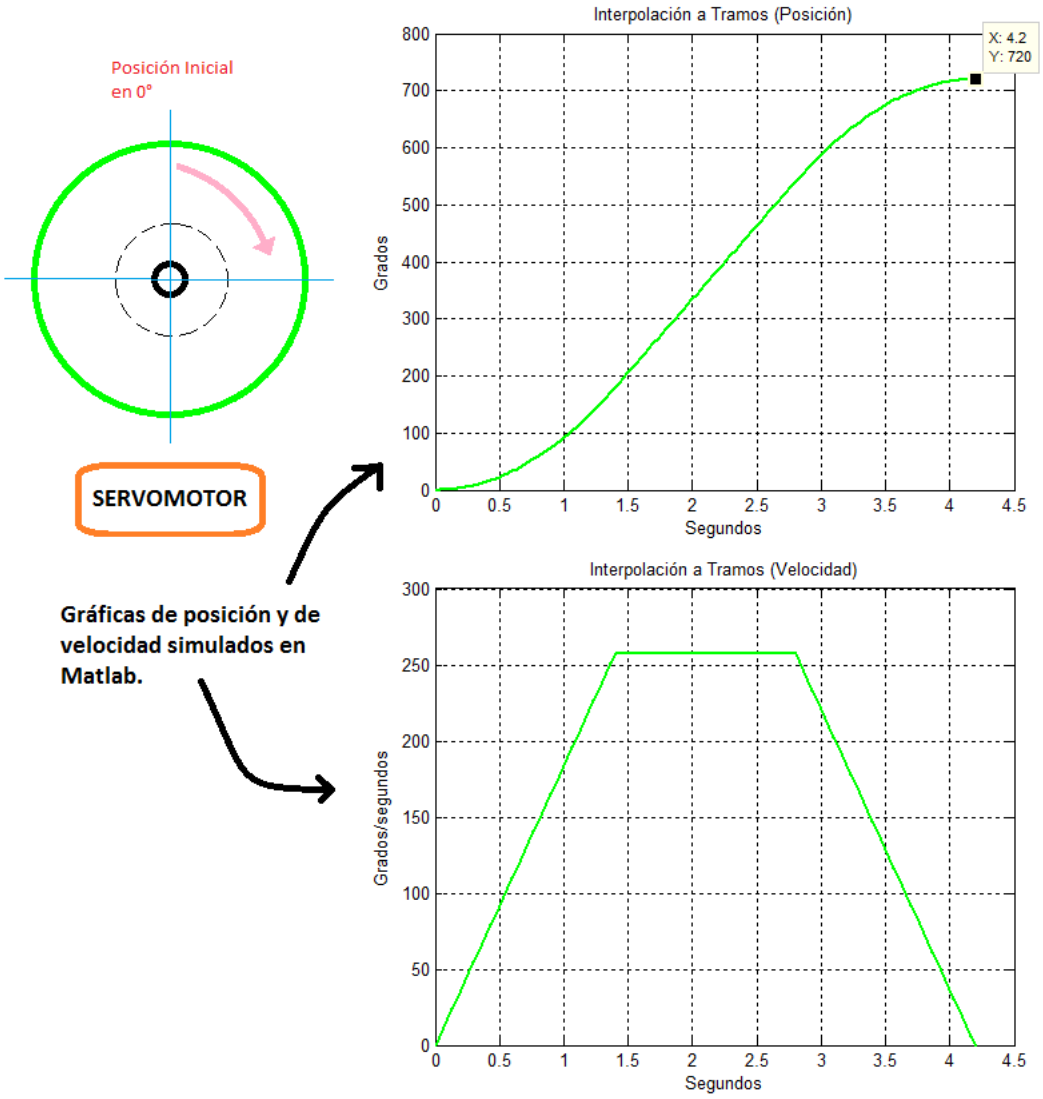


Figura 4.3: Gráfica de posición y de velocidad obtenidas de la simulación de la función de interpolación a tramos.

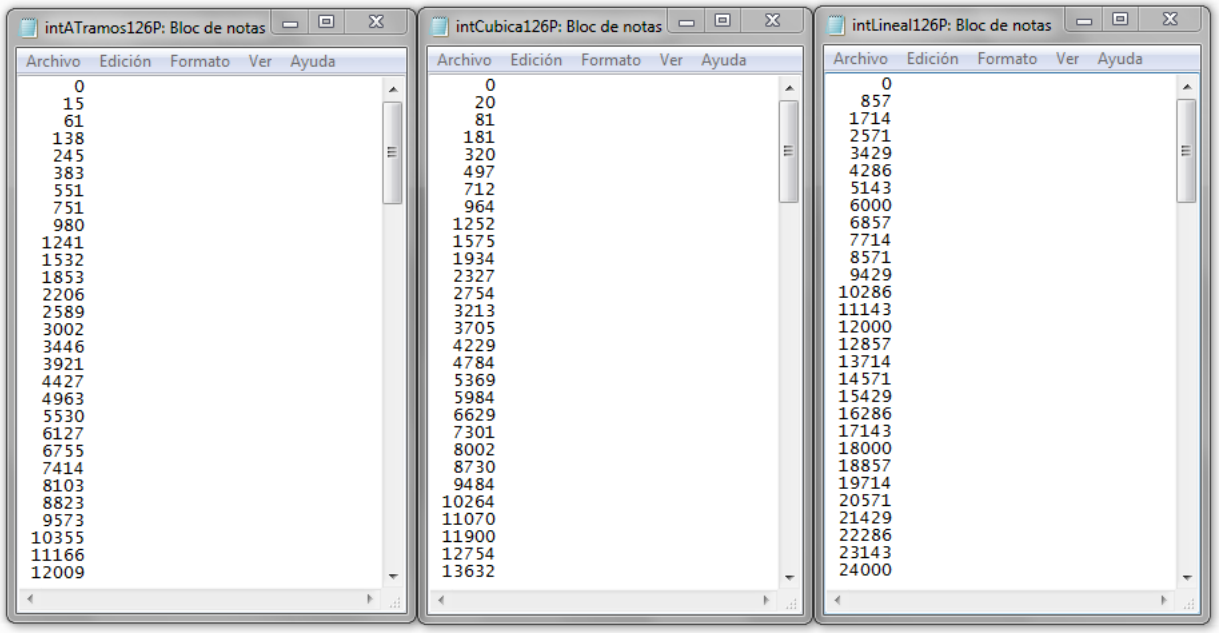


Figura 4.4: El algoritmo en Matlab se encarga de guardar los 127 puntos de posición, expresados en cuentas de encoder. Se genera un archivo por función de interpolación simulada, donde la posición inicial es *0 cuentas de encoder*.

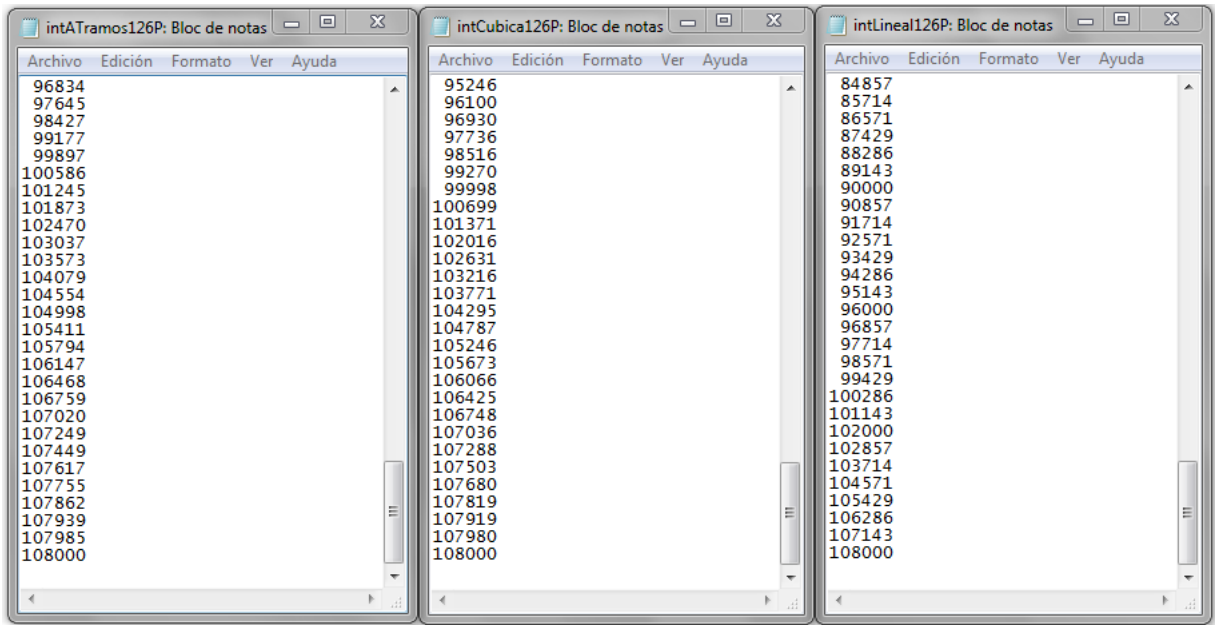


Figura 4.5: Posición final de cada trayectoria a *108000 cuentas de encoder*.

4.2. Procedimiento

Se enumera el procedimiento que se realiza para lograr ejecutar una trayectoria sobre cualquiera de las 5 articulaciones:

1. Cada una de las funciones de interpolación fueron simuladas mediante *Matlab*, la simulación de cualquiera de estas tres funciones generará un archivo de texto que contiene cantidades numéricas que representan los *puntos de trayectoria*. Se han muestreado en intervalos de *30 Hz* y se han obtenido un total de 127 puntos de trayectoria. Estos puntos representan las posiciones en *cuentas de encoder* que el eje del servomotor deberá ir alcanzando conforme vaya girando.

2. Entonces, en la ecuación 4.1 observamos que el tiempo máximo aproximado de duración de la trayectoria es de *4.23 segundos*.

$$\frac{127 \ puntos \ de \ trayectoria}{30 \ Hertz} = 4{,}23 \ segundos \tag{4.1}$$

3. Esta duración máxima representa el tiempo total de trayectoria articular que hasta el momento se puede programar. Por lo tanto, mediante *Matlab* únicamente generamos los puntos de posición y este los guarda en un archivo de texto.

4. Ahora, utilizamos el programa que diseñamos en C y lo ejecutadomos (ver Figura 4.6). El programa realiza un reinicio automático del sistema y realiza las configuraciones necesarias para que el servocontrolador opere en el modo CMC. Posteriormente, para el caso de una articulación, el programa recupera el archivo de texto que contiene los puntos de posición creado por *Matlab*.

5. El programa captura los puntos de posición y calcula todas las diferencias entre puntos adyacentes, de esta forma se obtienen *126 diferencias* que de ahora en

Figura 4.6: Pantalla del programa prototipo para generación de trayectorias.

adelante, las llamaremos de igual forma como *puntos de trayectoria*.

6. El programa realiza las codificaciones necesarias de tal forma que cada nuevo punto codificado de trayectoria contenga la información de la frecuencia a la que fue muestreada y también la dirección del sentido de giro.

7. El programa ensambla un total de *18 paquetes de instrucción* y cada paquete contiene 7 puntos de trayectoria codificados (ver ec. 4.2).

$$(18\ \textit{paquetes})(7\ \textit{puntos codificados}) = 126\ \textit{puntos de trayectoria} \tag{4.2}$$

8. Una vez creados los paquetes, el programa envía por el puerto serial estos 18 paquetes al servocontrolador deseado, y de esta forma se almacenan los puntos en su *buffer*.

9. Finalmente, el programa solicita un *enter* para iniciar la ejecución de la trayectoria almacenada.

10. En cuanto la trayectoria se ejecuta, el programa solicita en todo momento la siguiente información al servocontrolador:

    - Número de muestra.
    - Posición real.
    - Error de posición.
    - Velocidad a la que se desplaza en ese instánte.
    - Puntos vaciados del *buffer*.

11. Estos datos son guardados por el programa en un nuevo archivo de texto que posteriormente serán recuperados por *Matlab* para graficar los resultados. Final-

mente se grafican y comparan la trayectoria deseada (simulada) y la trayectoria real (ejecutada) obtenida. De esta forma validaremos nuestros resultados.

Este fue el procedimiento realizado para la generación de trayectorias, recordando que el programa utilizado fue diseñado , y también será una herramienta que utilizamos en este proceso de ejecución de trayectoria. También debemos de recordar que el servo-controlador internamente aproximará puntos intermedios entre los puntos aproximados que se hayan guardado en el *buffer*, en intervalos de $512us$. El propósito de esto es reducir la cantidad de datos calculados por el *host* y producir desplazamientos suaves y lineales cuando el servomotor se mueva de un punto a otro punto.

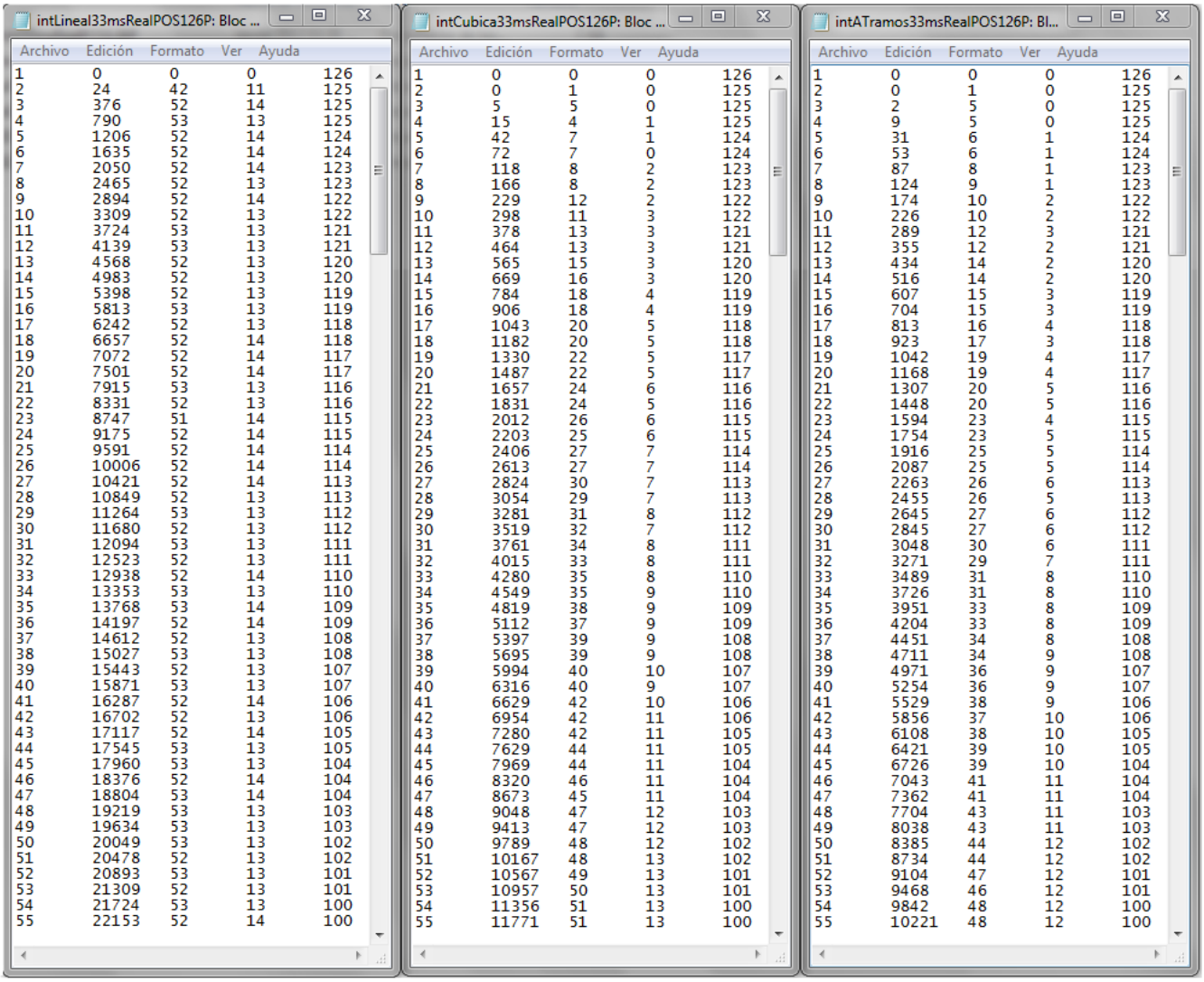
4.3. Resultados

Como resultado de la ejecución de trayectoria, el programa generará un archivo de texto que devuelve la información relevante al movimiento rotacional realizado. En la Figura 4.7 observamos los tres archivos de texto que el programa devuelve con la información solicitada y que ha capturado en determinados instantes de tiempo.

La información que podremos observar se guarda tabulada, y se enumeran a continuación por número de columna.

1. Número de muestra.
2. Posición real.
3. Error de posición.
4. Velocidad a la que se desplaza en ese instánte.
5. Puntos vaciados del *buffer*.

Estos datos posteriormente serán utilizados para validar resultados.



intLineal33msRealPOS126P: Bloc ...

Archivo Edición Formato Ver Ayuda

1	0	0	0	126
2	24	42	11	125
3	376	52	14	125
4	790	53	13	125
5	1206	52	14	124
6	1635	52	14	124
7	2050	52	14	123
8	2465	52	13	123
9	2894	52	14	122
10	3309	52	13	122
11	3724	53	13	121
12	4139	53	13	121
13	4568	52	13	120
14	4983	52	13	120
15	5398	52	13	119
16	5813	53	13	119
17	6242	52	13	118
18	6657	52	14	118
19	7072	52	14	117
20	7501	52	14	117
21	7915	53	13	116
22	8331	52	13	116
23	8747	51	14	115
24	9175	52	14	115
25	9591	52	14	114
26	10006	52	14	114
27	10421	52	14	113
28	10849	52	13	113
29	11264	53	13	112
30	11680	52	13	112
31	12094	53	13	111
32	12523	52	13	111
33	12938	52	14	110
34	13353	53	13	110
35	13768	53	14	109
36	14197	52	14	109
37	14612	52	13	108
38	15027	53	13	108
39	15443	52	13	107
40	15871	53	13	107
41	16287	52	14	106
42	16702	52	13	106
43	17117	52	14	105
44	17545	53	13	105
45	17960	53	13	104
46	18376	52	14	104
47	18804	53	14	104
48	19219	53	13	103
49	19634	53	13	103
50	20049	53	13	102
51	20478	52	13	102
52	20893	53	13	101
53	21309	52	13	101
54	21724	53	13	100
55	22153	52	14	100

intCubica33msRealPOS126P: Bloc ...

Archivo Edición Formato Ver Ayuda

1	0	0	0	126
2	0	1	0	125
3	5	5	0	125
4	15	4	1	125
5	42	7	1	124
6	72	7	0	124
7	118	8	2	123
8	166	8	2	123
9	229	12	2	122
10	298	11	3	122
11	378	13	3	121
12	464	13	3	121
13	565	15	3	120
14	669	16	3	120
15	784	18	4	119
16	906	18	4	119
17	1043	20	5	118
18	1182	20	5	118
19	1330	22	5	117
20	1487	22	5	117
21	1657	24	6	116
22	1831	24	5	116
23	2012	26	6	115
24	2203	25	6	115
25	2406	27	7	114
26	2613	27	7	114
27	2824	30	7	113
28	3054	29	7	113
29	3281	31	8	112
30	3519	32	7	112
31	3761	34	8	111
32	4015	33	8	111
33	4280	35	8	110
34	4549	35	9	110
35	4819	38	9	109
36	5112	37	9	109
37	5397	39	9	108
38	5695	39	9	108
39	5994	40	10	107
40	6316	40	9	107
41	6629	42	10	106
42	6954	42	11	106
43	7280	42	11	105
44	7629	44	11	105
45	7969	44	11	104
46	8320	46	11	104
47	8673	45	11	104
48	9048	47	12	103
49	9413	47	12	103
50	9789	48	12	102
51	10167	48	13	102
52	10567	49	13	101
53	10957	50	13	101
54	11356	51	13	100
55	11771	51	13	100

intATramos33msRealPOS126P: Bl...

Archivo Edición Formato Ver Ayuda

1	0	0	0	126
2	0	1	0	125
3	2	5	0	125
4	9	5	0	125
5	31	6	1	124
6	53	6	1	124
7	87	8	1	123
8	124	9	1	123
9	174	10	2	122
10	226	10	2	122
11	289	12	3	121
12	355	12	2	121
13	434	14	2	120
14	516	14	2	120
15	607	15	3	119
16	704	15	3	119
17	813	16	4	118
18	923	17	3	118
19	1042	19	4	117
20	1168	19	4	117
21	1307	20	5	116
22	1448	20	5	116
23	1594	23	4	115
24	1754	23	5	115
25	1916	25	5	114
26	2087	25	5	114
27	2263	26	6	113
28	2455	26	5	113
29	2645	27	6	112
30	2845	27	6	112
31	3048	30	6	111
32	3271	29	7	111
33	3489	31	8	110
34	3726	31	8	110
35	3951	33	8	109
36	4204	33	8	109
37	4451	34	8	108
38	4711	34	9	108
39	4971	36	9	107
40	5254	36	9	107
41	5529	38	9	106
42	5856	37	10	106
43	6108	38	10	105
44	6421	39	10	105
45	6726	39	10	104
46	7043	41	11	104
47	7362	41	11	104
48	7704	43	11	103
49	8038	43	11	103
50	8385	44	12	102
51	8734	44	12	102
52	9104	47	12	101
53	9468	46	12	101
54	9842	48	12	100
55	10221	48	12	100

Figura 4.7: Documentos generados por el programa que contienen información relevante del movimiento realizado por el servomotor.

Capítulo 5

Resultados

En este capítulo comentaremos los resultados finales obtenidos del proyecto de investigación. Se hablará acerca de los resultados en *hardware* como en *software* para el funcionamiento del sistema. En lo relevante a la generación de trayectorias articulares, se muestran gráficas comparativas de posición y velocidad como resultado de las pruebas experimentales de trayectorias ejecutadas por una de las articulaciones del sistema.

5.1. Construcción de una arquitectura abierta

Como resultado de las conexiones realizadas del *hardware*, se obtuvo el circuito final del sistema. Se analizaron esmeradamente las conexiones eléctricas entre todos los dispositivos electrónicos involucrados.

Las conexiones eléctricas realizadas permitieron consolidar un sistema que permite controlar 5 servomotores o ejes a través del *puerto serie* de la computadora. Este sistema permitirá posteriormente el estudio de cada uno de los ejes que integrarán las 5 articulaciones del robot manipulador *ROMMEL* (ver Figuras 5.1, 5.2, 5.3 y 5.4).

En cuanto al *software* de funcionamiento, se realizó un programa en lenguaje C que permite establecer comunicación con el sistema a través del puerto serial de la

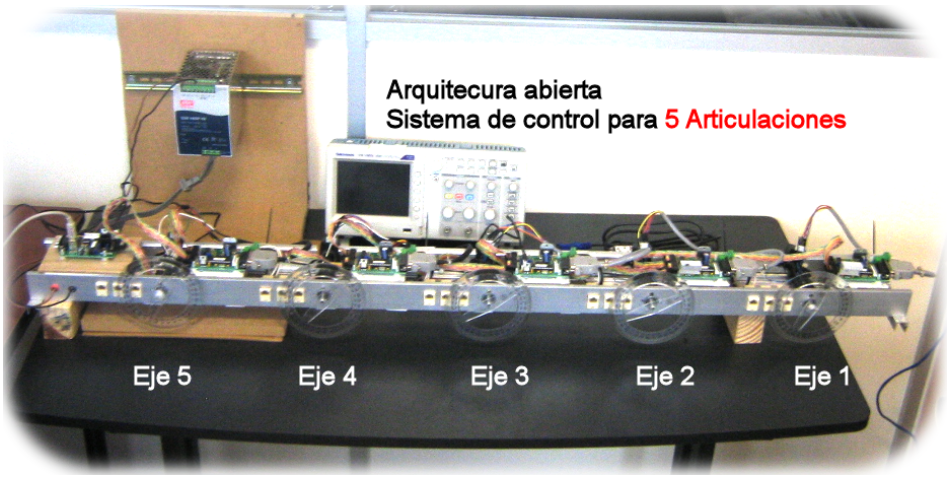


Figura 5.1: Arquitectura abierta para el control de 5 articulaciones.

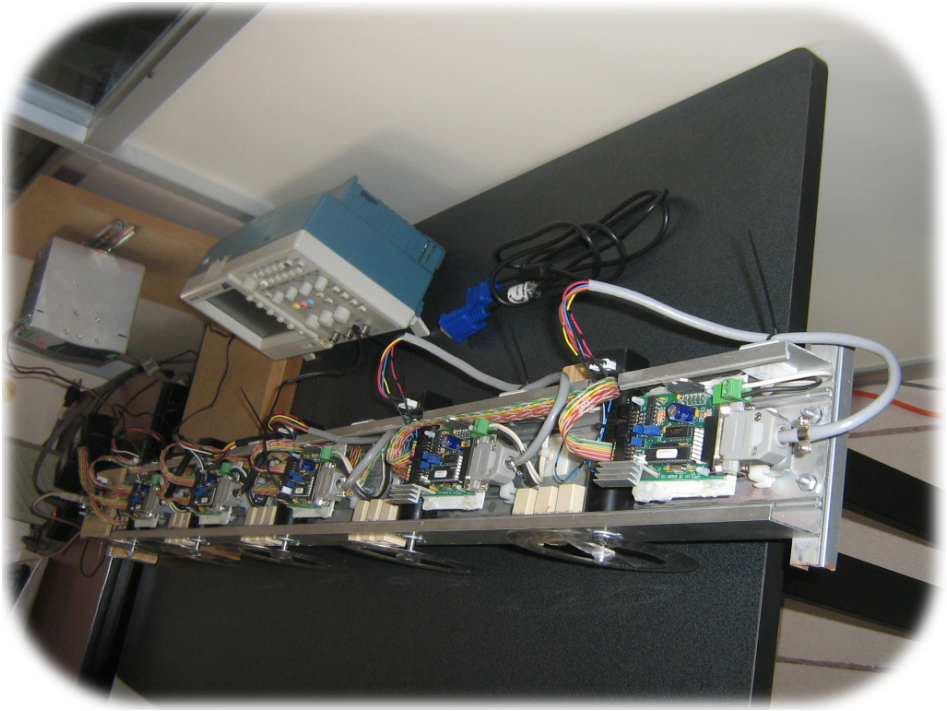


Figura 5.2: Vista superior lateral del sistema para el control de 5 articulaciones.

Figura 5.3: Tarjeta interfaz de comunicación.

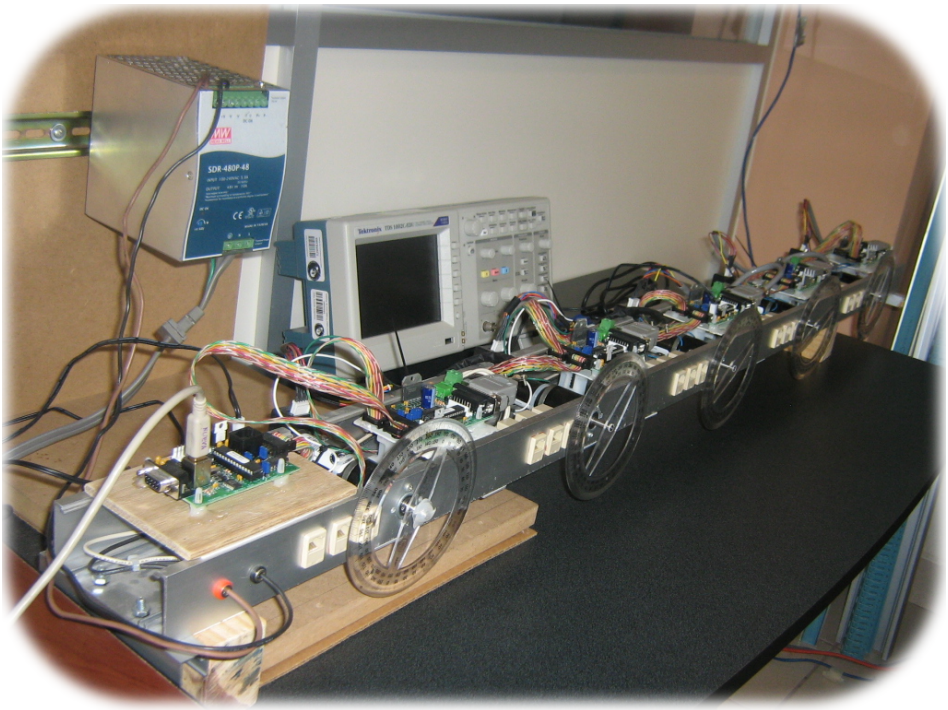


Figura 5.4: Vista lateral del sistema para el control de 5 articulaciones.

computadora (o por el puerto USB).

Enunciamos la característica de poseer arquitectura abierta, al hecho de no hacer uso de ninguna librería de control para el servocontrolador, es decir, no se hizo uso de ninguna librería creada por el fabricante, pues dichas librerías son como cajas negras que no transparentan el procedimiento que realizan en programación a bajo nivel.

El programa diseñado fue totalmente a bajo nivel, pues se realizaron algoritmos que manipulan números binarios. Se generaron algoritmos de conversiones de números enteros a números binarios y a números hexadecimales, se crearon algoritmos de ensamblaje de *paquetes de instrucción.*

Se diseñaron algoritmos de decodificación de *paquetes de estatus* retornados por el servocontrolador, que permitejn conocer datos de operación durante la ejecución de determinada trayectoria. En general, se crearon varios algoritmos importantes.

Podemos mencionar que el resultado final fue favorable pues se logró crear un programa prototipo de control para este sistema, escrito en lenguaje C y en *software* libre, lo que permitirá continuar con la investigación y generar mejores algoritmos y sus propias librerías.

Todo lo anterior nos permitirá poseer un control considerable de cada una de las 5 articulaciones del sistema, lo que permite ampliar enormemente en número de aplicación para la simulación de máquina desde tan sólo una articulación a 5 articulaciones.

5.2. Generación de trayectorias

El programa creado en lenguaje C para la programación de puntos de trayectoria que posteriormente son enviados al *buffer* del servocontrolador fue exitoso, pues se ejecutaron pruebas de trayectorias que aseveraron la funcionalidad del algoritmo diseñado. Estas pruebas consistieron en la generación de trayectorias articulares mediante

la implementación de funciones de interpolación. Anteriormente se había mencionado que el programa solicitaría y guardaría información relevante al comportamiento de la trayectoria.

A continuación se muestran gráficas comparativas entre los perfiles deseados o calculados, con los perfiles de posición reales obtenidos, así mismo, con las velocidades. Las trayectorias son ejecutadas partiendo de la posición 0° hasta la posición 720° o *108,000 cuentas de encoder*. Por cada una de las funciones de interpolación se generaron archivos de texto con datos que posteriormente son graficados mediante *MATLAB*©. En seguida se muestran las gráficas comparativas de los datos recabados por el *software*. Debemos de recordar que no se realizó ningún control de ganancias del filtro PID.

5.2.1. Trayectoria por interpolador lineal

En la Figura 5.5 observamos una gráfica que muestra la comparación entre el perfil de posición obtenido y el perfil de posición deseado, es decir, el calculado por *MATLAB*©. La gráfica muestra que el movimiento real no es completamente lineal, sin embargo, la trayectoria trazada fue aproximada al perfil de deseado.

En la Figura 5.6 observamos la comparación entre perfiles de velocidad. Observamos que el servocontrolador intenta mantener la velocidad constante durante todo el recorrido de la trayectoria. Los picos observados en la gráfica son a causa de la aceleración y desaceleración que realiza el controlador debido posiblemente a falta de sintonización del filtro PID. El perfil de velocidad idealmente debería ser constante, sin embargo, en la realidad esto no sucede.

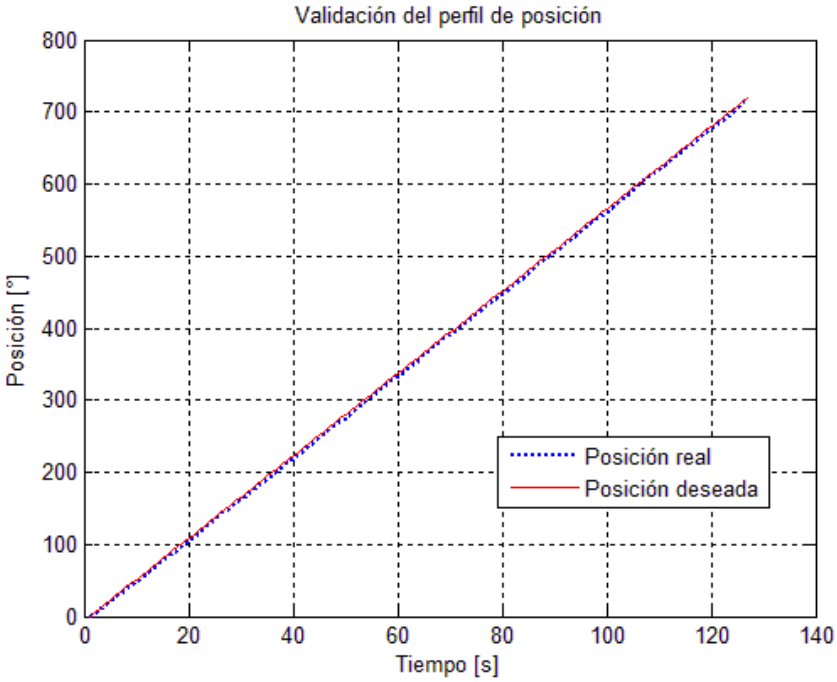


Figura 5.5: Validación del perfil de posición.

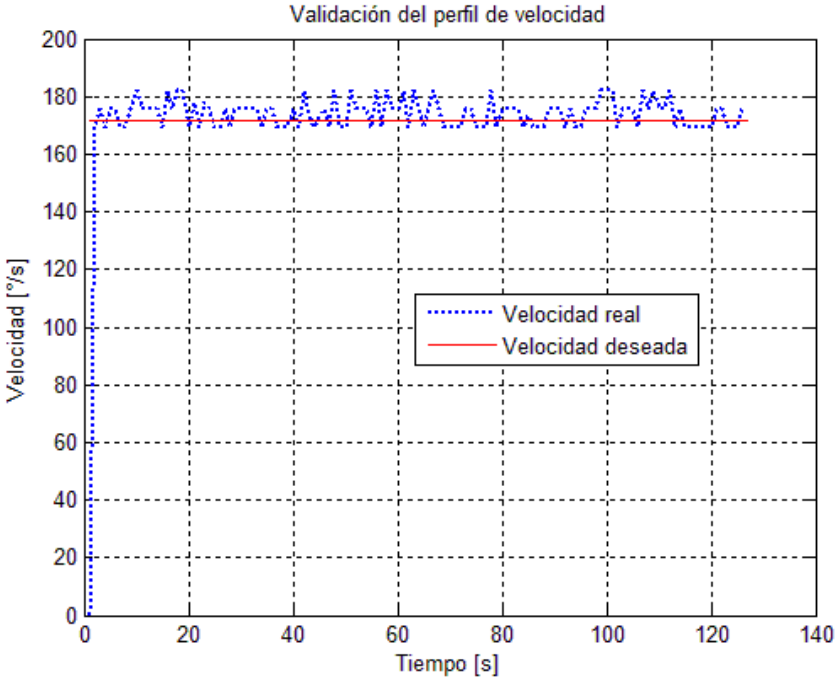


Figura 5.6: Validación del perfil de velocidad.

5.2.2. Trayectoria por interpolador cúbico

En la Figura 5.7 observamos la gráfica que muestra la comparación entre el perfil deseado de posición y el perfil real obtenido. El perfil obtenido fue aproximado al perfil deseado, y se observa un desplazamiento mucho más suave.

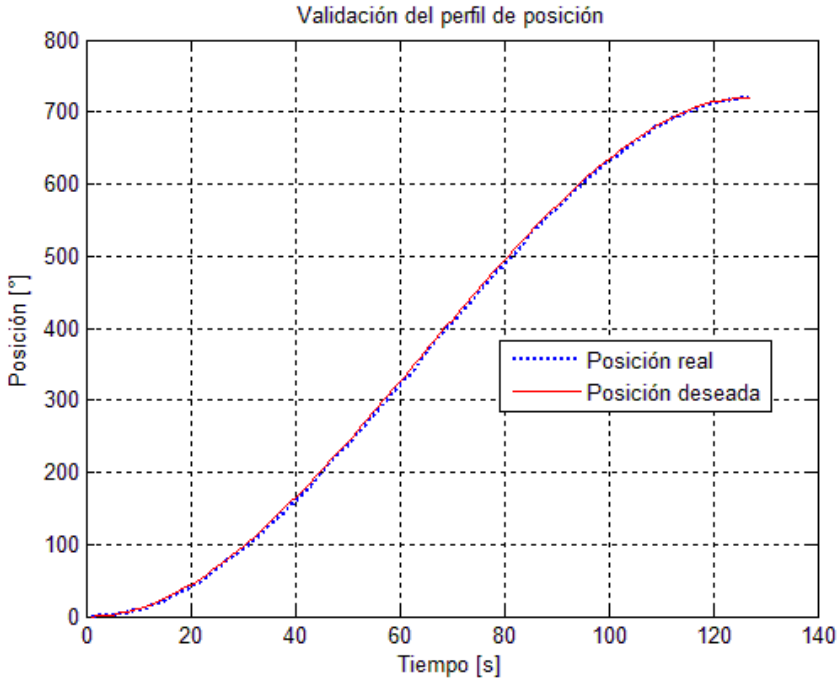


Figura 5.7: Validación del perfil de posición.

En la Figura 5.8 observamos la gráfica comparativa de los perfiles de velocidad obtenido con el perfil de velocidad calculado, tanto la velocidad inicial como la velocidad final es de cero. Esta función no permite establecer la velocidad máxima alcanzada durante la ejecución del movimiento.

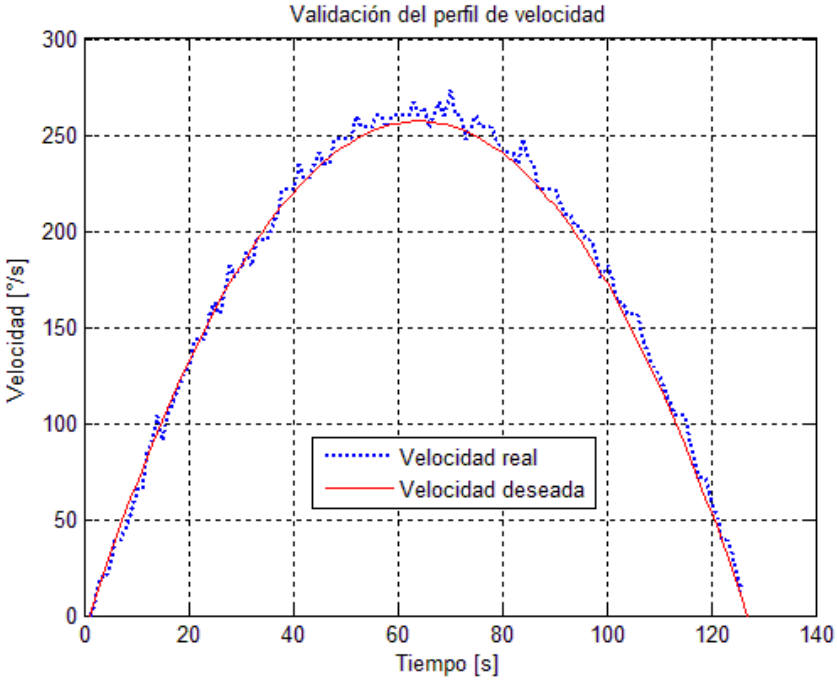


Figura 5.8: Validación del perfil de velocidad.

5.2.3. Trayectoria por interpolador a tramos

En la Figura 5.9 observamos la gráfica comparativa entre el perfil de posición deseado y el perfil de posición real. La ejecución de esta trayectoria resultó muy interesante, pues se observa claramente una gran similitud entre la trayectoria obtenida por función de interpolación cúbica y esta interpolación a tramos.

En la Figura 5.10 observamos la gráfica comparativa entre el el perfil de velocidad deseado y el perfil de velocidad real, observamos aceleraciones y desaceleraciones del perfil de velocidad obtenido que seguramente es debido a la falta de sintonización del filtro PID, pues claramente observamos que aunque existen pequeñas aceleraciones y desaceleraciones durante el movimiento, la ejecución de esta trayectoria por el servomotor fue muy similar al perfil de velocidad deseado. Este tipo de función de interpolación nos permite establecer la velocidad máxima requerida.

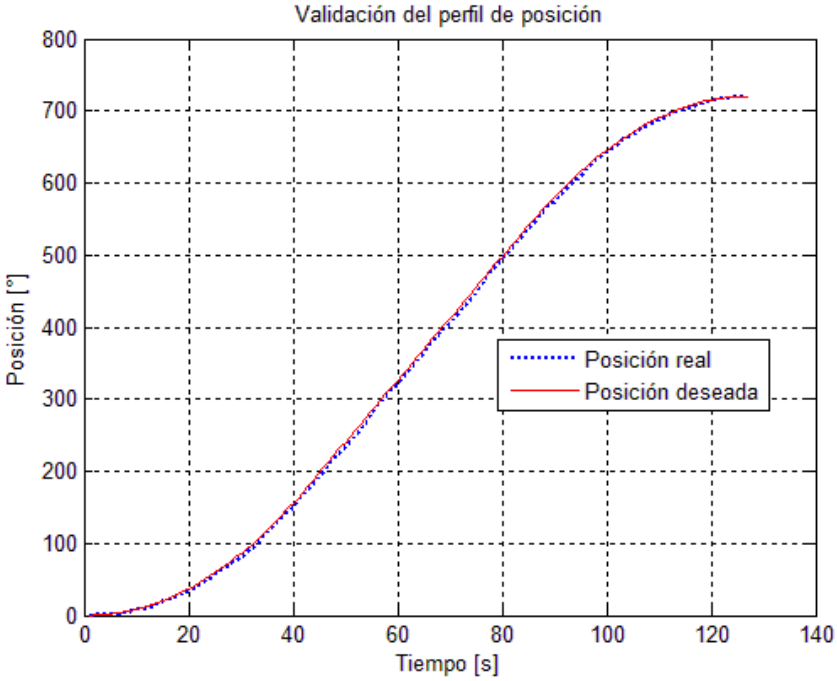


Figura 5.9: Validación del perfil de posición.

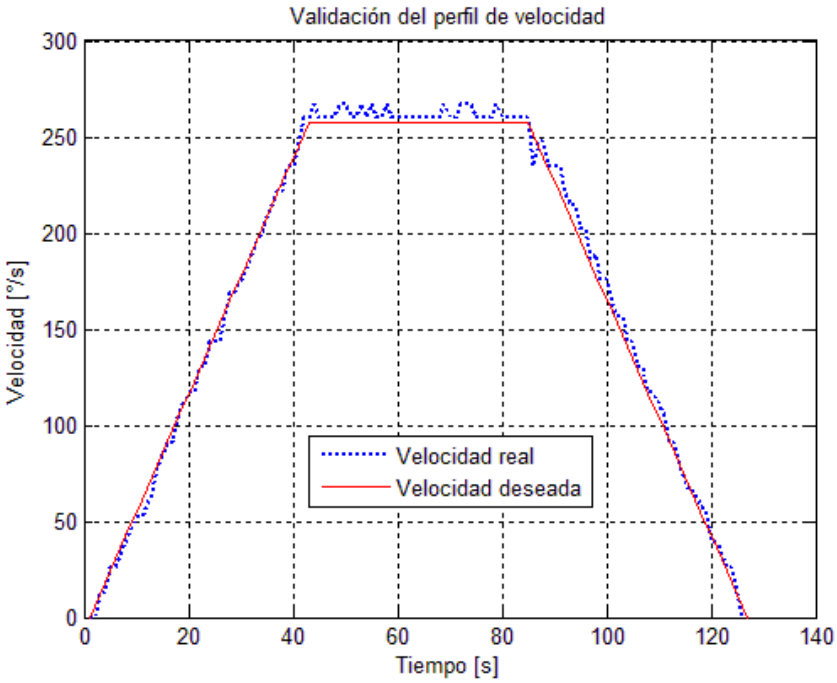


Figura 5.10: Validación del perfil de velocidad.

Debemos de recordar la importancia de la ejecución de trayectorias del sistema, así como su correcto funcionamiento. Las unidades de posición y velocidad fueron calculados y establecidos en las simulaciones, sin embargo, en investigaciones posteriores a esta, deberán de ser analizados con detalle.

Capítulo 6

Conclusiones

Para la integración del *hardware* se realizó un estudio minucioso de las conexiones eléctricas que se deben de realizar para conectar correctamente los ejes del sistema. El sistema está formado por 5 servomotores, sin embargo, el protocolo de comunicación *RS-485* nos permite poder incrementar el número de servomotores a la red hasta un máximo de 32 dispositivos.

Se hace necesario el análisis de las fuentes de alimentación de *48 voltios*, pues para aplicaciones donde el motor necesite drenar mayor corriente, la fuente de voltaje deberá sufragar esto sin problemas.

El uso de reguladores pequeños para alimentar las tarjetas a *9 o 12 voltios*, pueden reducirse al cambiarlos por otra fuente de voltaje que ofrezca la corriente suficiente para alimentar los 5 servocontroladores.

En general, el sistema se ha conectado sin problemas y el funcionamiento ha sido hasta este momento sin problemas. La comunicación del sistema a la computadora puede ser a través del puerto serial (*RS-232*) o del puerto USB, pues el adaptador posee la circuitería necesaria para adaptar ambos medios de comunicación al protocolo *RS-485*.

Los mayores logros se obtuvieron en *software*, pues se realizaron todos los algoritmos

necesarios. Aunque no se utilizaron librerías del fabricante para hacer funcionar el servocontrolador, sí se siguió una lógica similar.

Podemos destacar que disponemos a una arquitectura abierta, al hecho de que el programa es totalmente manipulable y configurable, pues podemos disponer de una o más articulaciones para simular no sólo al futuro robot manipulador ROMMEL, sino a una infinidad de máquinas como por ejemplo los graficadores o routers.

El desarrollo de este primer programa para generar movimiento de los servomotores permitió experimentar con la mayoría de los modos de funcionamiento del servocontrolador, se desarrollaron algoritmos a bajo nivel que realizan operaciones específicas a modo de poder manipular las variables del sistema. Este programa servirá en proyectos posteriores para la generación de nuevos programas de control mejor elaborados.

Bibliografía

[1] Ollero Baturone A. *Robótica, manipuladores y robots móviles*. España, 1st edition, 2001.

[2] Balaguer C. y Aracil R. Barrientos, Peñín L. F. *Fundamentos de robótica*. Mc Graw Hill, Aravaca, Madrid, 2nd edition, 1997.

[3] ConClase.net. *C++ con Clase*. Mayo 2012 [Online]. Available: `http://c.conclase.net/`.

[4] Grau i Saldes A. y Martínez García H. Domingo Peña J., Gámiz Caro J. *Comunicaciones en el entorno industrial*. UOC, First, 2003. ISBN 84-9788-003-X.

[5] Jameco Electronics. *Datasheet PIC-SERVO SC Motion Control Board*. Abril 2012 [Online]. Available: `http://www.jameco.com/Jameco/Products/ProdDS/323942.pdf`.

[6] Jameco Electronics. *Datasheet SSA-485 Smart Serial Adapter Board*. Abril 2012 [Online]. Available: `http://www.jameco.com/Jameco/Products/ProdDS/1586120.pdf`.

[7] Villela Varela R. Reyes Rivas C. y Beltrán Telles A. Gonzáles Elías M. E., Ramírez Esquivel O. E. Generación de trayectorias para teleoperación de robots manipu-

ladores por modelo cinemático. *Revista Digital de la Universidad Autónoma de Zacatecas*, 3(1):410–425, Enero-Abril 2007.

[8] Gobierno de Estado de Puebla Instituto Nacional para el Federalismo y el Desarrollo Municipal. *Enciclopedia de los Municipios de México, Estado de Puebla*. Marzo 2012 [Online]. Available: `http://arcelia.gob.mx/work/templates/enciclo/puebla/econ.htm`.

[9] Craig J. J. *Robótica*. Pearson Educación, México, 3rd edition, 2006.

[10] LLC Jeffrey Kerr. *PIC-SERVO Motion Control*. Abril 2012 [Online]. Available: `http://www.jrkerr.com/software.html`.

[11] Williams K. *Build your own Humanoid Robots*. McGraw-Hill, 2004. ISBN 0-07-142274-9.

[12] Rivera J. y Sandoval R. Martínez G., Jáquez S. Diseño propio y construcción de un brazo robótico de 5gdl. *Revista de ingeniería eléctrica, electrónica y computación*, 4(1):9–15, Julio 2008. ISSN 1870-9532.

[13] The Institute of Robotics and Mechatronics. *DLR Light-Weight Robot (LWR)*. Noviembre 2011 [Online]. Available: `http://www.dlr.de/rm/en/desktopdefault.aspx/tabid-3803/`.

[14] Ros.org. *Robots Using ROS: TUM-Rosie*. Noviembre 2011 [Online]. Available: `http://www.ros.org/`.

[15] Villani L. y Oriolo G. Siciliano B., Sciavicco L. *Robotics Modelling, Planning and control*. Springer-verlag, USA, 2009.

[16] Bolton W. *Mecatrónica. Sistemas de control electrónico en ingeniería mecánica y eléctrica*. Alfaomega Grupo Editor, México D.F, 4 ed edition, Enero 2010. ISBN 978-607-7854-32-6.

[17] Tomasi W. *Sistemas de comunicaciones electrónicas*. Prentice Hall, 4th edition, 2003. ISBN 0-13-022125-2.

[18] Andueza L. y Aguirre I. Diseño de un manipulador robótico con tres grados de libertad para fines educativos. *Revista ciencia e ingeniería*, 30(1):3–14, Diciembre-Marzo 2009. ISSN 1316-7081.

[19] Alqasemi R. y Duvey R. Control of a 9-dof wheelchair-mounted robotic arm system. In *Florida Conference on Recent Advances in Robotics*, May 2007.

[20] Chan T. F. y Duvey R. V. A weighted least-norm solution based scheme for avoiding limits for redundant joint manipulators. *IEEE Robotics and Automation Transactions*, 11(2):286–292, April 1995.

[21] Carper C. y Wu C. H. Development of a stereoscopic robot platform utilizing an education in engineering and computer science. In *ASEE Southeast Section Conference*, 2007.