UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

CONTROL DE UN BRAZO ROBÓTICO TELEOPERADO MEDIANTE UN ACELERÓMETRO

T E S I S

Que para obtener el título de

INGENIERO MECATRÓNICO

P R E S E N T A N

HERBERT HELLEMANN HOLGUÍN

DANIEL SÁNCHEZ CHÁVEZ

DIRECTOR DE TESIS:
M.I. OCTAVIO DÍAZ HERNÁNDEZ

MÉXICO D.F. **MARZO 2011**

1 AGRADECIMIENTOS Y DEDICATORIAS

Esta tesis no hubiese sido posible sin la cooperación y apoyo de cada una de las personas que a continuación se citan:

- A mis padres por su amor, guía y palabras de aliento.
- A mis hermanos Monika y Erich por su compañía y apoyo.
- A Toño por siempre creer en mí.
- A todos mis amigos.
- A Daniel, fue un honor trabajar contigo.
- A la Universidad

Herbert Hellemann Holguín

Esta tesis no hubiese sido posible sin la cooperación y apoyo de cada una de las personas que a continuación se citan:

- A mis padres
- A Cecilia
- A Andrea
- A Lisa
- A Herbert

Daniel Sánchez Chávez

AGRADECIMIENTOS

Se agradece el apoyo brindado por el proyecto PAPIIT clave IN108308, DGAPA, UNAM, con título: "Investigación y desarrollo en robótica móvil, robótica paralela y sistemas mecatrónicos", durante la realización de este trabajo.

2 ÍNDICE

3 RESUMEN

El presente proyecto se enfoca en el desarrollo de la teleoperación de una plataforma robótica prediseñada como dispositivo esclavo. El enfoque propuesto involucra un sistema de medición que utiliza un acelerómetro embebido en una tarjeta inalámbrica como dispositivo maestro.

Se empleó un robot antropomórfico comercial de cinco grados de libertad con una tenaza como órgano terminal al cual se le envían las coordenadas (xyz) a alcanzar dentro de un marco de referencia, mediante un microcontrolador. Estos puntos son determinados con base en la integración de los datos de aceleración del dispositivo maestro y mediante ellos son calculados los parámetros de cada articulación a través de la cinemática inversa del robot, fijando al último eslabón en una posición vertical en todo momento. Ambos cálculos resultan complejos y son llevados a cabo en una computadora. Sobre el órgano terminal fue colocado un marcador para dibujar la trayectoria generada por el dispositivo maestro. Para el experimento se utilizó una trayectoria recta sobre un plano horizontal.

Los retrasos en el movimiento del esclavo, durante la teleoperación, redefinieron la forma de uso de la plataforma desarrollada, por lo que el operador debe esperar a que se complete un movimiento para poder realizar otro movimiento.

Las limitantes tecnológicas llevaron a la falta de precisión y distorsión de las instrucciones deseadas. El dispositivo esclavo elegido para el desarrollo de ésta tecnología provoca directamente la lentitud en los movimientos del dispositivo esclavo, y la falta de un control más robusto y complejo provoca que los movimientos sean imprecisos y bruscos.

Es, por lo tanto, necesario completar las soluciones iniciales con sistemas mecánicos, electrónicos y de control más complejos que puedan soportar las necesidades de una teleoperación en tiempo real precisa y satisfactoria.

Se pretendió investigar algunos de los errores comunes en teleoperación, como los retrasos y la falta de precisión. La solución que se ha desarrollado muestra claramente que es posible mover al robot esclavo Scorbot con el envío de posiciones obtenidas mediante el uso de un acelerómetro sin que éste provoque los errores antes mencionados. Asimismo, demuestra que es posible desarrollar tecnología propia; dentro de ésta Universidad.

4 INTRODUCCIÓN

4.1 Antecedentes

4.1.1 Mecatrónica

La Mecatrónica debe combinar la mecánica, la electrónica y la informática unos con otros, y hacer una descripción integradora en un solo modelo, en vez de varios modelos independientes. De ésta manera la mecatrónica es la siergia de la mecánica de precisón, la electrónica de control y los sistemas informáticos (ver Imagen 1). Los sistemas mecatrónicos tienen la tarea de, junto con la sensórica, el procesamiento, la actórica y elementos de la mecánica, electrónica e informática (así como otras tecnologías que sean necesarias), de transformar, transportar o almacenar la energía, la materia y/o la información.

Un sistema mecatrónico típico recoge señales, las procesa y, como salida, genera fuerzas y movimientos. Los sistemas mecánicos son entonces extendidos e integrados con sensores, microprocesadores y controladores. Los robots, las máquinas controladas digitalmente, los vehículos guiados automáticamente, las cámaras electrónicas, las máquinas de telefax y las fotocopiadoras pueden considerarse como productos mecatrónicos. Al aplicar una filosofía de integración en el diseño de productos y sistemas se obtienen ventajas importantes como son mayor flexibilidad, versatilidad, nivel de "inteligencia" de los productos, seguridad y confiabilidad así como un bajo consumo de energía. Estas ventajas se traducen en un producto con más orientación hacia el usuario y que puede producirse rápidamente a un costo reducido. (1)

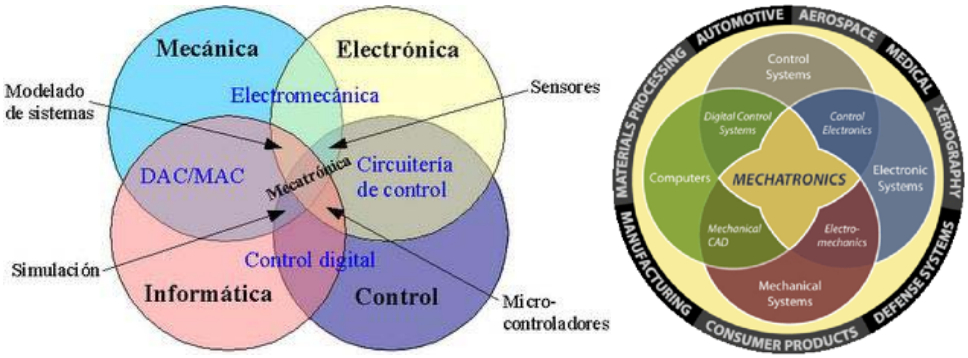


Imagen 1 Campo de la mecatrónica, la mecatrónica es la convergencia de varias tecnologías

4.1.2 Robótica

La Robótica es una ciencia o rama de la tecnología y de la Mecatrónica, que estudia el diseño y construcción de máquinas capaces de desempeñar tareas realizadas por el ser humano o que requieren del uso de inteligencia. De forma general, la Robótica se define como: El conjunto de conocimientos teóricos y prácticos que permiten concebir, realizar y automatizar sistemas basados en estructuras mecánicas poli-articuladas, dotados de un determinado grado de "inteligencia" y destinados a la producción industrial o la sustitución del hombre en muy diversas tareas.

Un sistema Robótico puede describirse, como "aquel que es capaz de recibir información, de comprender su entorno a través del empleo de modelos, de formular y de ejecutar planes, y de controlar o supervisar su operación". (2)

En la Imagen 2 pueden verse ejemplos de aplicaciones de robótica utilizada en la actualidad.

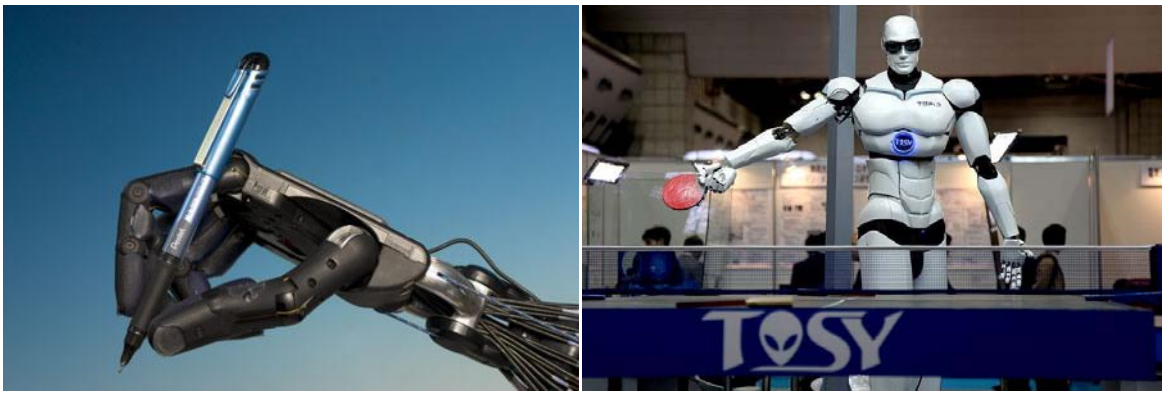


Imagen 2 Mano robótica (izquierda) y Robot humanoide (derecha)

4.1.3 Clasificación de los tipos de robots

La Asociación japonesa de Robots industriales (*Japanese Industrial Robot Association,* JIRA) clasifica a los robots de la siguiente manera (ver Imagen 3):

- Clase 1: Dispositivo de manejo manual: es actuado por un manipulador.
- Clase 2: Robot de secuencia fija: Un dispositivo que realiza etapas sucesivas en una tarea de acuerdo a un método determinado y fijo.
- Clase 3: Robot de secuencia variable: Un dispositivo que realiza etapas sucesivas en una tarea de acuerdo a un método determinado pero que puede ser modificado con facilidad.
- Clase 4: Robot *Playback*: Un operador humano realiza la tarea manualmente llevando al robot, el cual registra y graba el movimiento para repetirlo más tarde.
- Clase 5: Robot de control numérico: El operador dota al robot con un programa para movimientos en vez de enseñarle la tarea manualmente

- Clase 6: Robot Inteligente: Es un robot con los medios para entender su ambiente y lo que lo rodea y tiene la habilidad de realizar una tarea con éxito a pesar de los cambios en el ambiente en el que debería trabajar. (3)

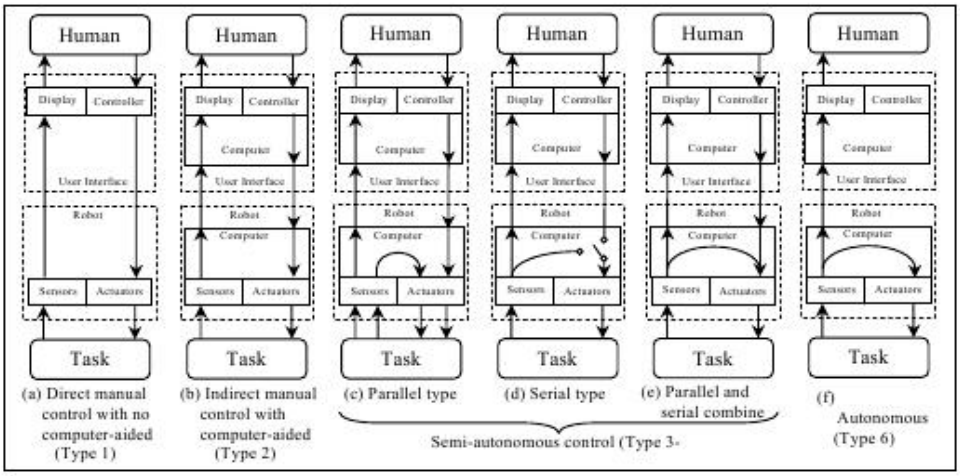


Imagen 3 Clasificación de los diferentes tipos de robots

4.1.4 Componentes de un Robot

Un Robot, consiste de los siguientes elementos (algunos ejemplos son mostrados en la Imagen 4):

- Manipulador: Es el cuerpo principal del robot
- Efector: Es la parte que generalmente realiza la tarea, hace conexiones con otros robots o manipula los objetos
- Actuadores: Son los "músculos" de los manipuladores. Algunos actuadores comunes son:
    - Servomotores
    - Motores a pasos
    - Cilindros neumáticos o hidráulicos
    - Actuadores magnéticos
    - Metales con memoria de forma
- Sensores: Los sensores son usados para recolectar información del estado interno del robot o para comunicarse con el medio ambiente. Algunos sensores comunes son:
    - Sensores de posición
        - Potenciómetros
        - Encoders o resolvers
        - Transformador Lineal diferencial variable
        - Sensor de desplazamiento
    - Sensores de velocidad
        - Encoders
        - Tacómetros

        - Señal de diferenciación de posición
    - Sensores de aceleración
        - Acelerómetros
    - Sensores de fuerza y presión
        - Sensores piezoeléctricos
        - Resistor de fuerza
        - Galgas extensiométricas *(Strain Gauges)*
    - Sensores de Torque
    - Micro switches
    - Sensores de luz e infrarrojos
    - Sensores de tacto
    - Sensores de proximidad
        - Magnéticos
        - Ópticos
        - Ultrasónicos
        - Inductivos
        - Capacitivos
        - De corriente de Eddy
    - Sensores de rango
    - Sistemas de visión
    - Sensores de gases
    - Reconocimiento de voz
- Controlador: Recibe la información, controla el movimiento de los actuadores y coordina el movimiento con la información que se recibe de los sensores
- Procesador: Calcula los movimientos en las articulaciones y determina la velocidad
- Software: Hay generalmente tres clases de software usado:
    - Sistema operativo: controla la computadora
    - Software robótico: calcula los movimientos necesarios en cada articulación
    - Programas de rutinas y aplicaciones: para el uso de dispositivos periféricos (3)

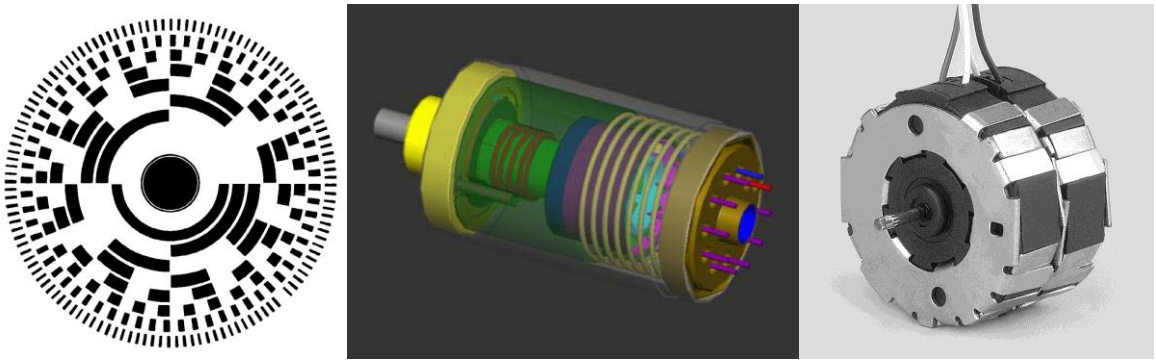


Imagen 4 Encoder de 8 bits, acelerómetro y motor a pasos

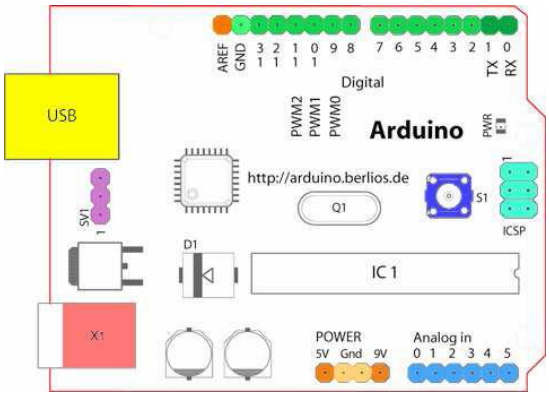


Imagen 5 Diseño de la tarjeta de desarrollo Arduino

4.1.5 Grados de libertad de un Robot

Para localizar un cuerpo rígido en el espacio, se necesita especificar la ubicación de un punto sobre él. Hay una infinidad de formas posibles para orientar el objeto hacia el punto deseado. Es necesario entonces definir la orientación hacia el objeto. Tomando en cuenta estos dos puntos, se requiere de un total de 6 datos para especificar la orientación y posición de un cuerpo rígido. Por lo tanto se necesita de 6 grados de libertad para colocar el objeto en el punto en el espacio deseado con la orientación deseada. Si un robot tiene menos de 6 grados de libertad (como en la Imagen 6), sus capacidades están limitadas. Un robot con 7 grados de libertad no tiene una solución única. (El efector no es un grado de libertad) (3)

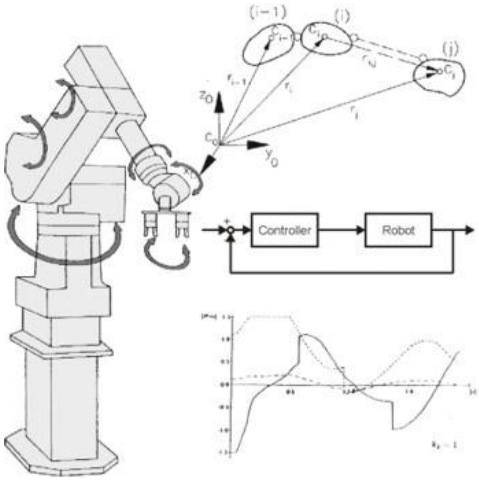


Imagen 6 Brazo Robótico con 5 grados de libertad

4.1.6 Articulaciones

Los robots usan distintos tipos de articulaciones: lineales, rotacionales, de deslizamiento o esféricos. Las articulaciones esféricas son usadas sobre todo en investigación. (3)

4.1.7 *Configuraciones físicas típicas*

1. Configuración en coordenadas cartesianas: Las tres direcciones “x”, “y” y “z”, están especificadas. Sus direcciones son ortogonales entre sí. Además, los ejes están descritos de manera tal que una rotación en sentido horario del eje “x” hacia el eje “y” provoque con la regla de la mano derecha el pulgar sobre el eje positivo de “z”
2. Configuración en coordinadas cilíndricas: El movimiento de un brazo robótico describe una superficie cilíndrica, cuando el radio del brazo esta fijo. Las coordenadas son “R”, “θ” y “z”, donde R es el radio del brazo, θ es la posición angular y “z” es la posición vertical.
3. Configuración en coordenadas polares: La superficie descrita es una semiesfera de radio R. Las coordenadas son “R”, “θ” y “Φ”. Θ es la rotación alrededor de una vertical a través de la base de soporte del brazo. Φ es el ángulo desde la horizontal sobre los 180 grados verticales.
4. Configuración en coordenadas de revolución: Todos los movimientos son en forma angular. Las coordenadas son “θ”, “Φ” y “α”. Θ es la rotación alrededor de la vertical que pasa sobre la base, Φ es el ángulo entre la horizontal a través de la base y el primer miembro del brazo. α es el ángulo del segundo brazo en referencia al primero. (ver Imagen 7) (4)

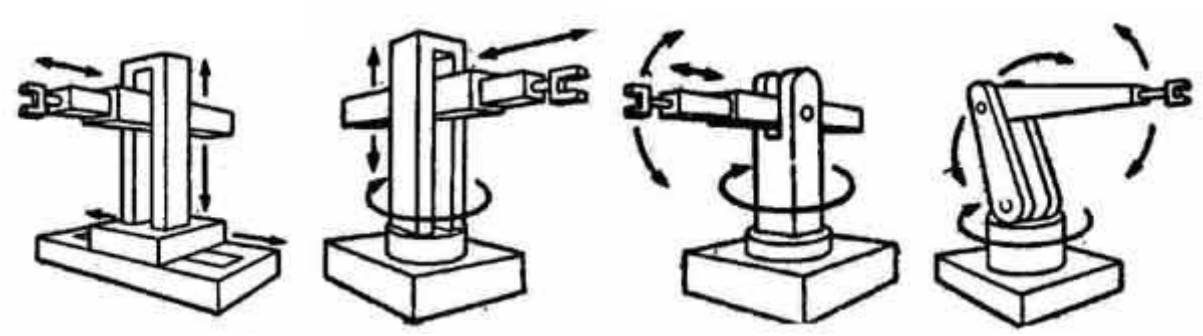


Imagen 7 Configuraciones típicas de los brazos robóticos (1), (2), (3), (4)

4.1.8 *Rango de movimientos (Work envelopes)*

Incluye todos los puntos dentro del espacio que pueden ser alcanzados por el brazo robótico, sin considerar el área al que llega el efector. A continuación se describe este espacio para las diferentes configuraciones:

1. Robots en coordenadas rectangulares: un paralelepípedo
2. Robots en coordenadas cilíndricas: Dos cilindros parciales, uno dentro de otro. Gracias a las limitaciones mecánicas, el brazo no puede alcanzar todos los puntos del cilindro, éste espacio es llamado espacio muerto
3. Robots en coordenadas polares: Dos esferas parciales con espacio muerto dentro de la primera esfera.
4. Robots en coordenadas de revolución: Todo el espacio dentro del volumen de trabajo. (4)

4.1.9 *Ventajas y desventajas de las diferentes configuraciones:*

Configuración	Ventajas	Desventajas
Coordenadas cartesianas	Se mueve en direcciones lineales y es por lo tanto fácil de visualizar	Requiere volúmenes grandes para trabajar
	Estructura muy rígida	Las superficies expuestas necesitan protección
	Fácil para controlar	La mayor demanda de superficie
Coordenadas cilíndricas	Sencillo de visualizar y controlar	Volumen restringido de acceso
	Adecuada para dispositivos hidráulicos, por lo tanto grande potencia	Difícil de sellar contra polvo y líquidos
	Buen acceso a cavidades y huecos	Interferencia con el rango de movimiento
Coordenadas polares	Cubre un gran volumen desde el soporte central	Difícil de visualizar y controlar
	Las partes rotacionales pueden ser selladas fácilmente	
Coordenadas de revolución	Flexibilidad máxima, gracias a que cada punto puede ser alcanzado	Difícil visualización y control
	Puede estar completamente sellado	Costoso para usos con dispositivos hidráulicos
	Útil en aplicaciones bajo el agua	(4)

Tabla 1 Ventajas y desventajas de las configuraciones en robótica

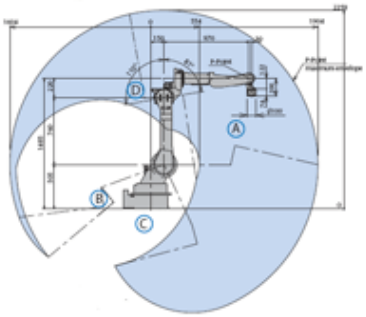


Imagen 8 Rango de movimientos de un brazo robótico (zona azul)

4.1.10 *Efectores finales*

Los efectores finales son los dispositivos que toman los objetos y que están al final del brazo robótico. Son los dispositivos que realizan la tarea. Son muy buenos tratando con objetos pesados, sustancias corrosivas o muy calientes y objetos filosos o peligrosos, sin embargo no son tan ideales para tratar formas complejas y objetos frágiles. Algunas técnicas comunes para agarrar o sostener objetos son:

- Por vacio
- Dispositivos electromagnéticos
- Pinzas y ganchos como en la Imagen 9
- Manos con tres o más dedos
- Adhesivos
- Garras u otros dispositivos mecánicos (4)

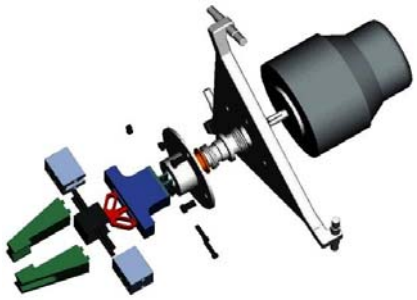


Imagen 9 Ejemplo de efector final para un brazo robótico

4.1.11 Teleoperación

La teleoperación es un tema explorado desde algunas décadas con el fin de extender la capacidad humana a una localización remota, muchas veces ésta localización es de difícil acceso, peligrosa o inadecuada para el trabajo in situ del hombre. Estas habilidades humanas que se extienden más allá del entorno inmediato, pueden incluir además de una manipulación directa, cierta inteligencia del robot que se encuentra en el sitio para darle mayor autonomía y capacidad de toma de decisiones.

En 1947 comenzaron las primeras investigaciones, lideradas por Raymond Goertz del Argonne National Laboratory en Estados Unidos, encaminadas al desarrollo de algún tipo de manipulador de fácil manejo a distancia mediante el uso por parte del operador de otro manipulador equivalente. (ver Imagen 10)

Imagen 10 Raymond Goertz manipulando elementos radioactivos por medio de teleoperacion

En la regla, existe un manipulador maestro y otro esclavo que reproduce los movimientos generados en el maestro. Los sistemas simples utilizan sólo las instrucciones del maestro y las trasladan, a través del medio de comunicación, al esclavo. Sin embargo, con el objeto de conseguir un acoplamiento entre ambos manipuladores se emplea la realimentación a las articulaciones del maestro. (5)

4.1.12 Arquitecturas de control en la teleoperación

Las diversas arquitecturas de control en teleoperación se diferencian básicamente por la información que se intercambia entre el maestro y el esclavo y por el tipo de instrumentación que se requiere. En función de la información intercambiada entre el maestro y el esclavo pueden clasificarse en las siguientes categorías:

- *Esquema posición-posición*: la posición del esclavo se determina a partir del maestro y viceversa. No hay necesidad de sensores de fuerza.
- *Esquema fuerza-posición*: la posición del esclavo se determina por el seguimiento del robot maestro y las fuerzas que aparecen sobre el esclavo se miden y se generan en el maestro mediante sus motores. Sólo se requiere medida de fuerzas en el esclavo.
- *Esquema fuerza-fuerza*: Las trayectorias del esclavo y del maestro se determinan a partir de las lecturas de fuerza de ambos. También existe un control local de posición en ambos robots.
- *Esquema de cuatro canales*: En este caso hay intercambio de información tanto en posición como en fuerza. El análisis teórico de esta solución refleja que es capaz de proporcionar transparencia infinita. (5)

4.1.13 Telerobótica

Es una forma evolucionada de teleoperación, caracterizada por un aumento de autonomía (capacidad de decisión y actuación) en el sistema remoto manteniendo una intervención significativa del operador para supervisión o teleoperación directa. Existen, de igual manera, vehículos robóticos móviles, capaces de extender la capacidad a localizaciones lejanas, como se muestra en la Imagen 11.

Imagen 11 Robótica móvil en la exploración de Marte y su uso militar

4.1.14 Trabajos previos

La teleopetación ha tenido un desarrollo importante durante los últimos 50 años. Se muestra a continuación breves descripciones de trabajos ya realizados en el área de la teleoperación y la telerobótica que sean pertinentes y que ubiquen al presente proyecto dentro del marco global, muchos de ellos muestran áreas interesantes donde la teleoperación tiene mucho potencial, así como las teorías de control que éstos sitemas utilizan.

1. ***Bilateral teleoperation: An historical survey, Peter F. Hokayem, Mark W. Spong, Universidad de Illinois, 2006***

Desde la mitad de los años 40, cuando el primer teleoperador maestro-esclavo fue construido por Goertz, el campo de la teleoperación tuvo un incremento en su interés. Los esfuerzos realizados desde el experimento de Goertz introducen fuerzas de retroalimentación y el control de supervisión (*Supervisory Control*), el cual intenta solucionar el problema de los retrasos, responsables de la baja de rendimiento del sistema y de la adopción de diversas estrategias como respuesta del usuario, tal como la estrategia de mover y esperar (*move-and-wait*). A partir de los años 80 y 90, surgieron teorías de control más avanzadas, como el análisis de Lyapunov y el control basado en pasividad, y, recientemente, la teoría de redes y la aparición de la teleoperación sobre Internet. Desde el punto de vista de todas estas teorías de control, las metas principales de la teleoperación son dos: encontrar la estabilidad de un sistema de lazo cerrado y proveer al operador humano una experiencia de telepresencia. Éste trabajo muestra una línea histórica de los trabajos realizados por centenares de investigadores en ésta área y menciona que muchos de los métodos desarrollados han encontrado su espacio en aplicaciones como el tratamiento de materiales radiactivos, operación de vehículos autónomos bajo el agua, la robótica espacial, telecirujía y recientemente en la teleoperación de robots móviles.

2. ***Sistemas Roboticos Teleoperados*, Alexander Ceron Correa, Bogota, Colombia 2005**

La elección de locomoción específica depende de características del proyecto a realizar tales como el tipo de terreno y la velocidad requerida. Los sensores utilizados en un robot móvil dependen del tipo de obstáculos que se encuentren y las características del entorno donde se desplace el mismo. En éste caso, el desarrollo de esta clase de tecnología es muy importante, ya que existen muchas circunstancias en las cuales se emplean operarios para realizar tareas de alto riesgo, como la explotación minera, la industria química, la detección, manipulación, eliminación de cargas explosivas y desminado. Por tal razón es de gran interés la investigación y el desarrollo tecnológico en la ingeniería actual.

3. ***A triaxial accelerometer for measuring arm movements,*** **Eva Bernmark, Christina Wiktorin, Elsevier 2002**

Se determina la viabilidad de utilizar un acelerómetro como instrumento de medición de los movimientos de un brazo y poder determinar su cambio de posición, dentro de un marco de referencia. El acelerómetro se ha colocado en el brazo del sujeto, como se muestra en la Imagen 12

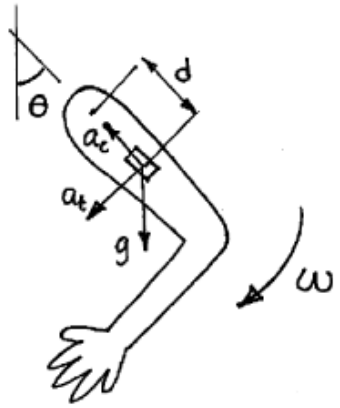


Imagen 12 Uso de un acelerómetro para medir movimientos de un brazo

4. ***Accelerometer-Based Control of an Industrial Robotic Arm,*** **Pedro Neto, Norberto Pires, Paulo Moreira, IEEE, Toyama Japon 2009**

En este trabajo se observa la evolución de la programación, intentando superar las técnicas previas, como lo es el "*teach pendant*" o posicionamiento por puntos entre otros. Esta solución demuestra ser una económicamente viable al encontrar en el mercado acelerómetros a muy bajo costo y los cuales pueden ser integrados en interfaces muy simples de utilizar por el usuario. (ver Imagen 13)

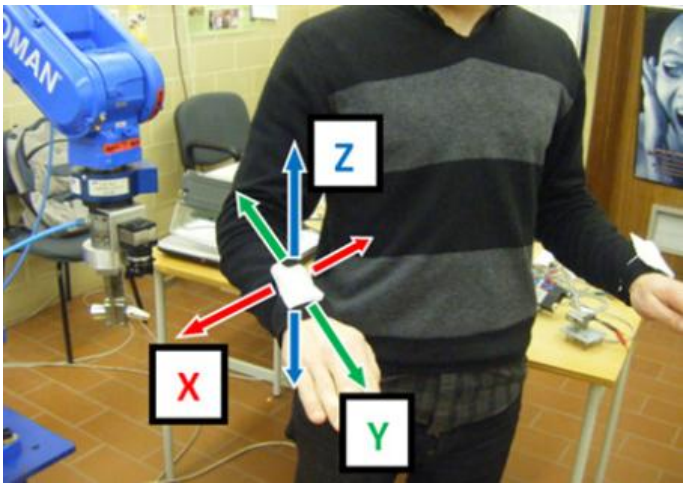


Imagen 13 Sistema capaz de sensar aceleraciones en tres ejes

5. ***MODELADO DEL ROBOT SCORBOT ER-V+,*** **M en C. Josué Román Martínez Mireles, Dr. Gerardo Vicente Guerrero Ramírez, Universidad Politecnica de Pachuca, 2007**

En este trabajo se utilizó como punto de referencia para la utilización de la cinemática inversa del brazo robótico Scorbot ER V +. Se describe el modelado cinemático directo (en la Imagen 14 se muestra el diagrama de eicho brazo) e inverso y después se presenta el procedimiento de modelado dinámico del robot el cual no fue tomado en cuenta dentro de este trabajo de tesis pero como se comenta en la conclusión es de gran relevancia para un correcto control del brazo en el seguimiento de trayectorias.

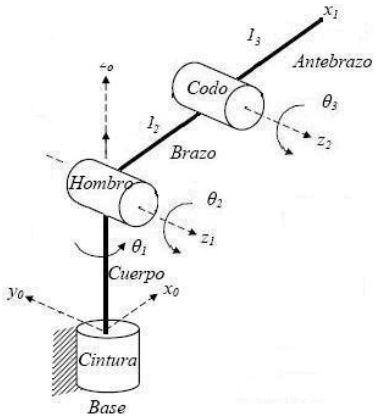


Imagen 14 Definición de la solución de la cinemática inversa del Scorbot

6. ***Design of a web-enabled anthropomorphic robotic arm for teleoperation,*** **G. Sen Gupta, S. C. Mukhopadhyay, Matthew Finnie, School of Engineering and Advanced Technology, New Zeland 2008.**

Con este trabajo se puede definir un tipo general de arquitectura en la teleoperación de brazos robóticos (ver Imagen 15). Se trabaja con un controlador de la potencia de los motores, el cual se comunica de manera inalámbrica con un microcontrolador encargado de la comunicación con una computadora y con un elemento maestro que sirve como referencia para el esclavo.

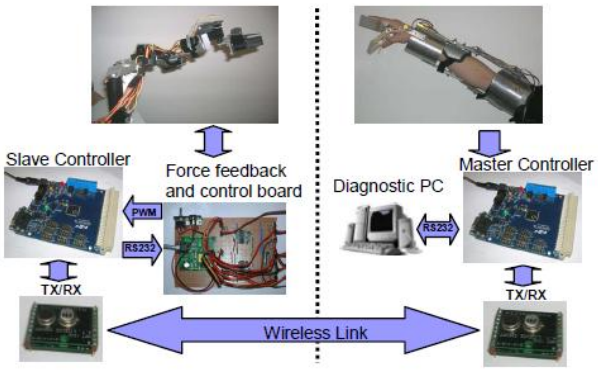


Imagen 15 Arquitectura para teleoperación

7. ***Development of a 6-Axis Robotic Arm Controller Implemented on a Low-Cost Microcontroller*, Agus Bejo, Hiroaki Kunieda, University of Thailand, 2009.**

El control de la etapa de potencia de un brazo robótico se puede obtener utilizando un microcontrolador como elemento principal. En la Imagen 16 se muestran las relaciones entre los diferentes dispositivos, del microcontrolador al brazo robótico.

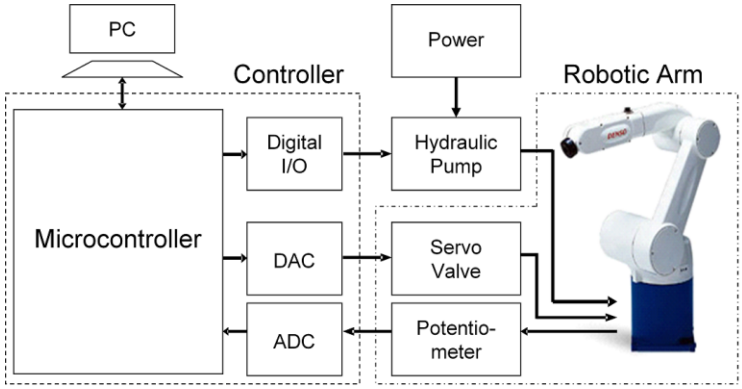


Imagen 16 Microcontrolador como elemento principal en el control de un brazo robótico

4.2 Objetivo

Tele-operar un brazo robótico antropomórfico utilizando un acelerómetro para calcular las posiciones del mismo. Los objetivos particulares son:

- Obtener la cinemática inversa de un robot antropomórfico a partir de los datos de posición.
- Mover el robot dentro de un espacio de trabajo con los datos obtenidos en la cinemática inversa.
- Demostrar la posibilidad de utilizar éste sistema en labores de corte, manipulación de objetos y dibujo.

4.3 Hipótesis

Implementar un sistema de control de un brazo robótico que pueda ser usado para labores de pintura (o escritura) y corte láser. El sistema propuesto debe minimizar los problemas típicos de retraso en las señales, de uso de señales de un maestro, que en éste caso es un sensor de aceleraciones que transmite las posiciones deseadas, y con el problema de reproducir trayectorias.

5 PLANTEAMIENTO DEL PROBLEMA

5.1 Conceptualización general

Para poder comprender el problema, éste debe ser separado en sus partes, a continuación se muestran las partes que están involucradas en la solución del presente problema; se pueden observar todos los componentes que participan en el sistema, la forma en que están comunicados con otros y el flujo de información del dispositivo maestro al esclavo en el Diagrama 1:

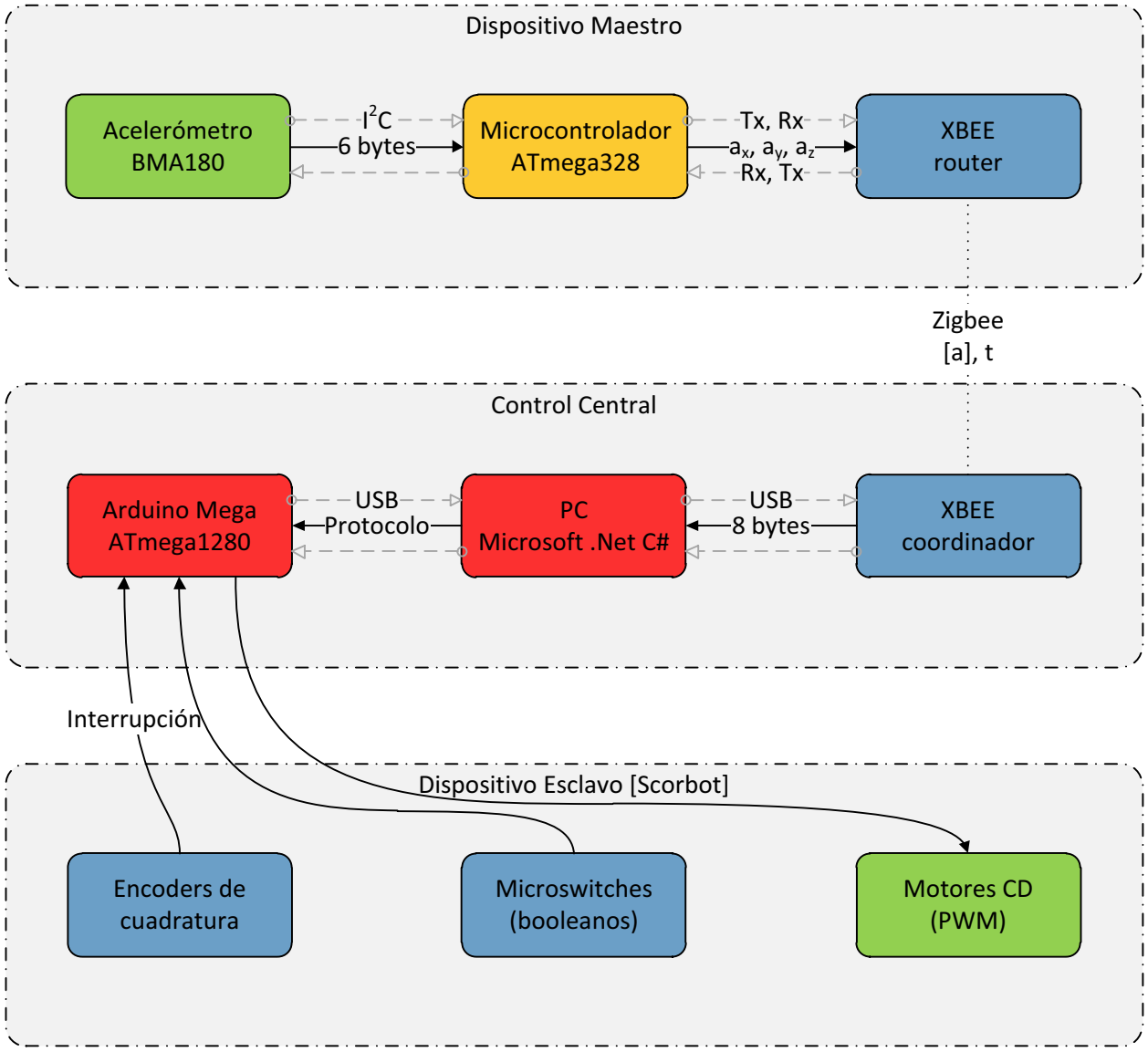


Diagrama 1 Flujo general de componentes del sistema

5.1.1 División entre Dispositivo Maestro, Control Central y Dispositivo Esclavo

Como se puede observar en el diagrama, existe una definición para cada una de las partes que corresponden a una teleoperación, donde cada una de las tres grandes divisiones (maestro, esclavo y control) está a su vez dividida en varios componentes que permiten su funcionamiento. Se pueden definir en general que actividades realiza cada uno de estos componentes a continuación:

5.1.2 Acelerómetro

El acelerómetro funciona como el dispositivo de entrada de datos, es donde el usuario puede manipular el sistema y desde el cual los parámetros iniciales se obtienen para más tarde usarlos en las diversas partes del proyecto. Es en realidad el sensor más importante ya que la información de éste manipula a todo el sistema.

5.1.3 Microcontrolador ATmega328

Éste microcontrolador primario se encarga de leer los datos del acelerómetro, manipular la información que obtiene para traducirla en útil y que tenga significado. Asimismo realiza la configuración del acelerómetro y envía la información obtenida al XBEE para transmitirse inalámbricamente al control central en la PC.

5.1.4 Módulos XBEE

Los módulos XBEE utilizan radiofrecuencia para transmitir la información en bytes del dispositivo maestro a la computadora, que va a procesar los datos. De ésta manera existen dos módulos: el coordinador y el router o esclavo. Éstos permiten que el dispositivo maestro tenga mayor libertad y que tenga menores restricciones espaciales.

5.1.5 PC (Personal Computer)

La computadora PC con Windows 7 en la plataforma .Net C# es la encargada de realizar las operaciones y cálculos más complejos (debido a su mayor velocidad), como lo son el cálculo de los ángulos de la cinemática inversa y la integración de las aceleraciones. Además es la encargada de realizar las tareas de visualización, simulación y corrección. En de suponerse que la PC funciona como el cerebro del sistema y es el que coordina todas las comunicaciones y realiza las decisiones principales. Además comunica el sistema de sensado con el sistema de actuado.

5.1.6 *Microcontrolador Arduino MEGA*

El microcontrolador seleccionado es el Arduino MEGA (cuenta con el microcontrolador ATmega1280), y con éste es posible realizar la comunicación con los distintos sensores que participan en el sistema (acelerómetro, encoders y micro-switches). En él se realizan algunas operaciones como el control PID y la comunicación serial con la computadora principal. Se encarga de toda la manipulación de las variables, lectura y escritura del manipulador Scorbot.

5.1.7 *Encoders de cuadratura (posición) y micro-switches*

Los encoders de cuadratura indican la posición actual de los motores del brazo robótico. Mediante ellos se puede efectuar el control de las posiciones deseadas.

Los micro-switches indican una posición determinada de cada una de las articulaciones del brazo robótico y es gracias a ellos que se puede tener un sistema de referencia; con ellos es posible llegar a una posición inicial '*home*' y a partir de ella puede realizarse un movimiento en espacio.

5.1.8 *Motores*

Los motores son los actuadores del sistema y gracias a ellos se pueden mover las articulaciones y realizar el trabajo deseado. Son parte indispensable porque son los que realizan la tarea deseada.

6 METODOLOGÍA

6.1 Procedimiento

El procedimiento de desarrollo del proyecto sigue cronológicamente al diagrama de flujo de datos entre el dispositivo maestro y el esclavo presentado en el planteamiento del problema, de forma que se solucionan uno a uno los subsistemas y se enlazan para crear la solución completa.

Sin embargo, se ha dividido en 2 el trabajo, siendo la primera parte la que inicia con el acelerómetro como sensor y termina en la PC, después de haber realizado todos los procesos propios dentro de ésta, como son la visualización, la cinemática inversa y otros. La segunda parte es la que comienza con los datos que se envían por comunicación serial de la PC al microcontrolador Arduino Mega y termina moviendo todos los motores del manipulador Scorbot y con la lectura de los sensores del mismo. Al dividir el trabajo en dos, se ha podido realizar una división de labores entre los dos integrantes del equipo.

De igual manera se han trabajado muchas de las funciones del control central, que descansa en la PC, en paralelo. Algunas de estas funciones incluyen la visualización de aceleraciones y posiciones acumuladas en una pantalla, un mapa de trayectorias y otros; la integración de los datos de aceleración, la realización de la cinemática inversa del Scorbot, entre otras.

Una bitácora de trabajo ha sido utilizada durante la duración de todo el proyecto, y en base a esa misma se ha construido el Diagrama 2.

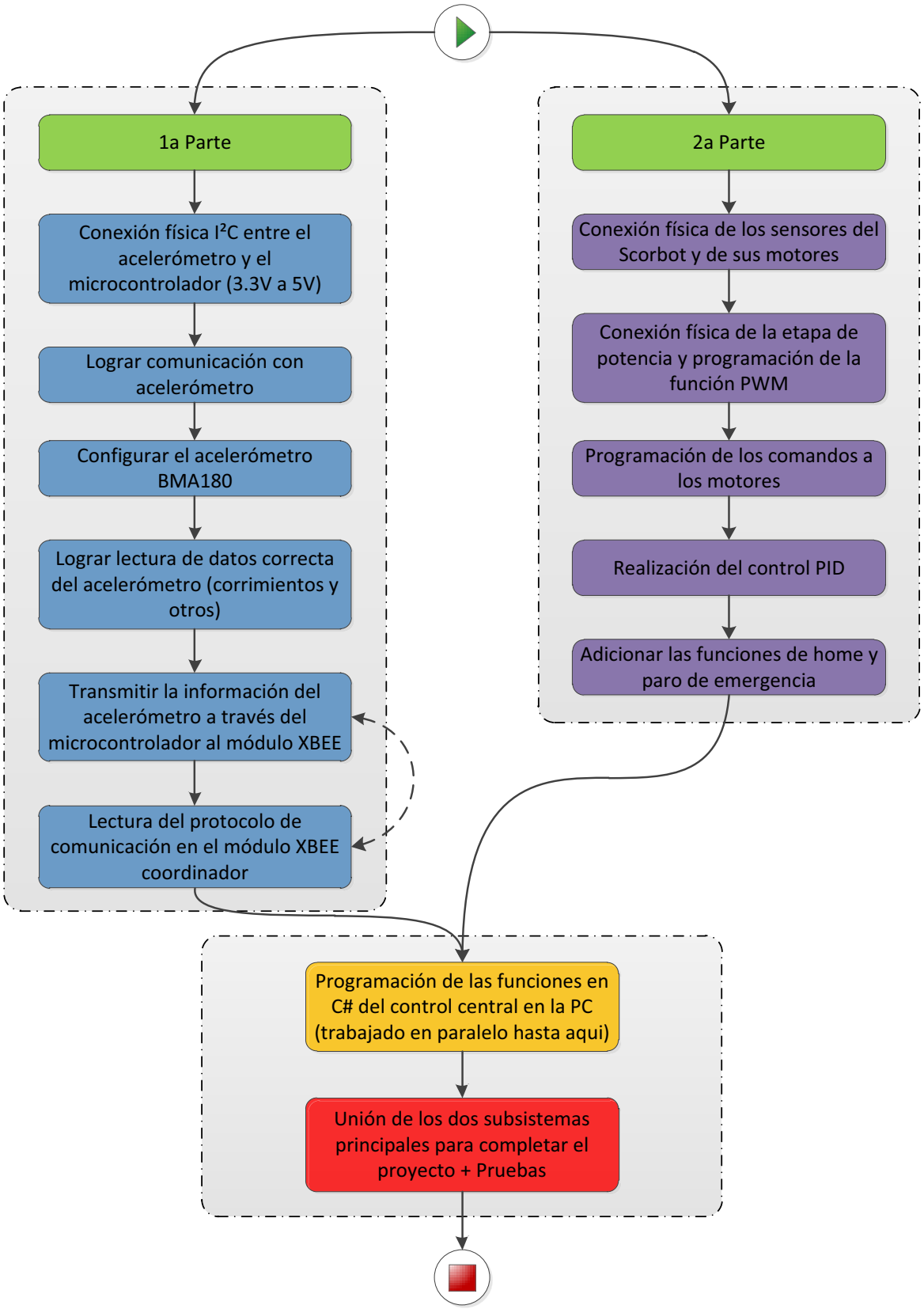


Diagrama 2 Flujo del desarrollo en paralelo del proyecto

7 DISPOSITIVO MAESTRO

7.1 Diseño del dispositivo maestro

7.1.1 Definición del problema

En la tele-robótica, existe siempre un dispositivo maestro y otro que funge como esclavo, el dispositivo maestro es el que dirige al dispositivo esclavo, y gracias a él y la información que envía al esclavo es posible realizar las tareas que se requieren. De ésta manera, en el presente proyecto, que es el control de un brazo robótico a través de un 'control inalámbrico', puede dividirse entre la parte del maestro y del esclavo, éste capítulo trata acerca del diseño del dispositivo maestro.

7.1.2 Árbol de objetivos

En el Diagrama 3 se muestran los objetivos primarios y secundarios:

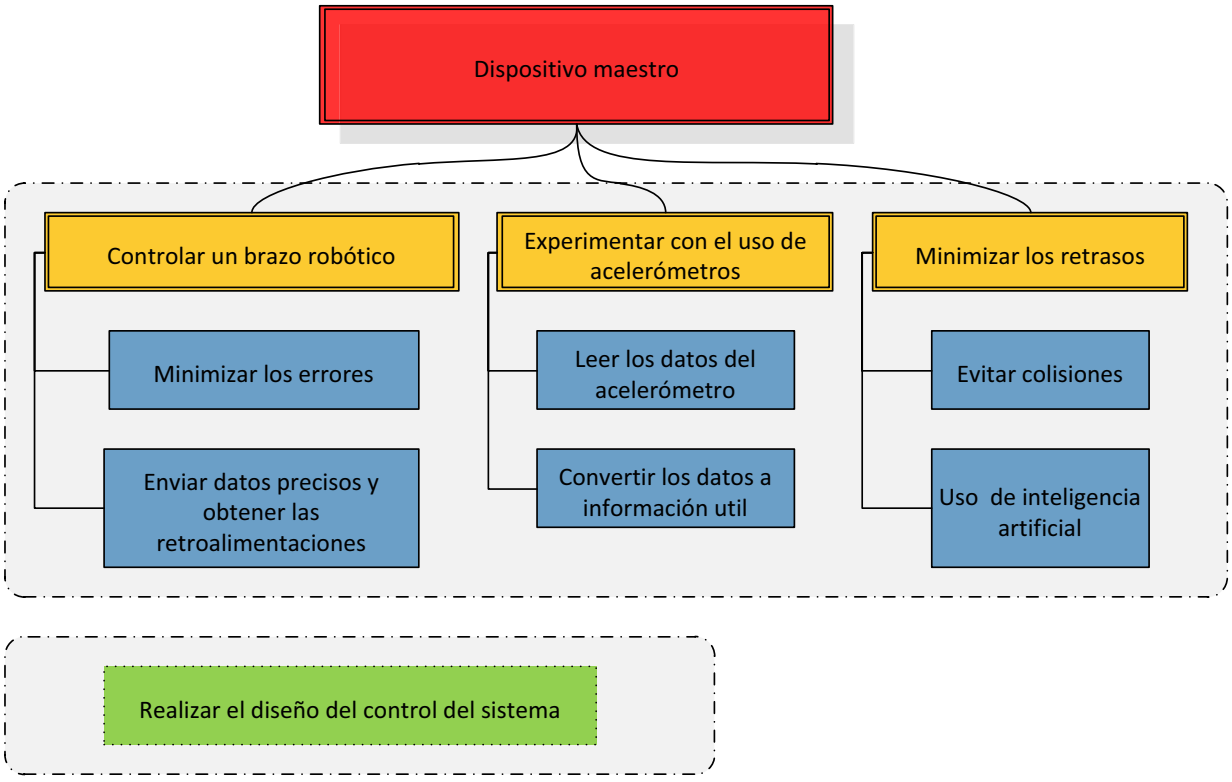


Diagrama 3 Árbol de objetivos del dispositivo maestro

7.1.3 Diagrama de funciones

Para establecer las funciones del sistema, se divide en funciones esenciales y funciones anexas. Las funciones esenciales son las que se deben realizar de forma obligada. Las otras funciones pueden depender de la configuración y pueden variar de acuerdo a las necesidades y al desarrollo.

En el caso del dispositivo maestro es importante mencionar que las funciones deben soportar el único objetivo que tiene el dispositivo, que es ser el dispositivo de entrada de datos del sistema general.

A continuación se muestran las funciones principales encontradas para dicho dispositivo:

- Funciones esenciales:
  - Alimentar
  - Sensar
  - Controlar (manipular la información)
  - Enviar
  - Soportar (mecánicamente) (ver Diagrama 4)
- Funciones anexas:
  - Acondicionar señales de entrada
  - Amplificar señales de salida
  - Conectar componentes

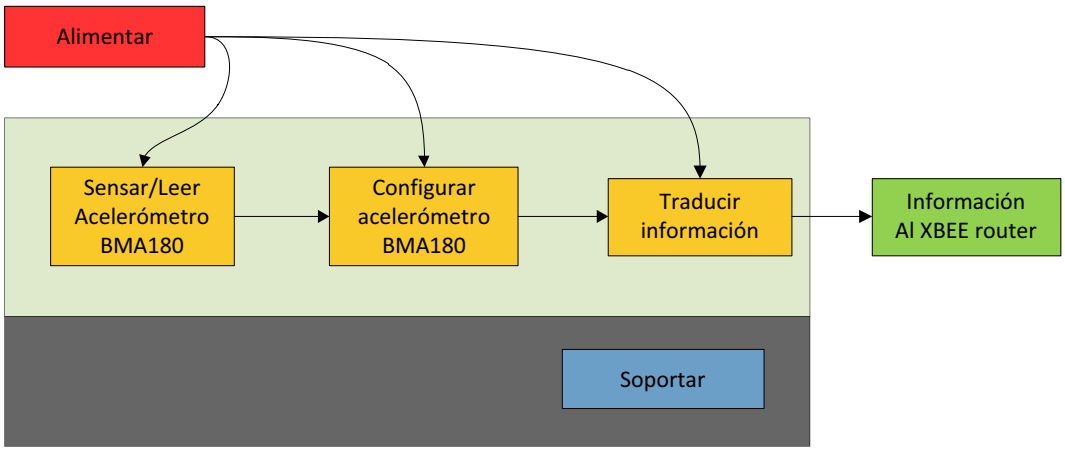


Diagrama 4 Funciones del dispositivo maestro

Los límites de las funciones (ver Diagrama 5) son importantes de definir, la mayoría de los errores se da en éstas zonas. Se muestra a continuación un diagrama que explica éstas interacciones:

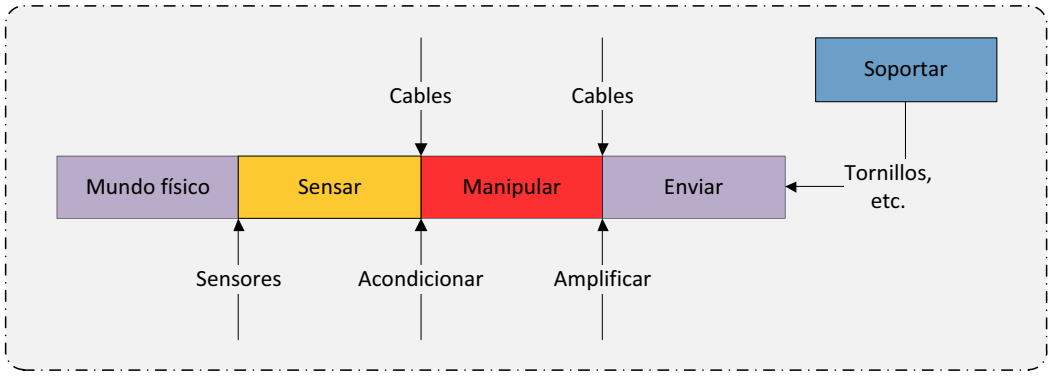


Diagrama 5 Límites de las funciones del dispositivo maestro

7.1.4 Requerimientos del sistema

Después de un análisis se obtuvieron los siguientes requisitos:

- El sistema debe leer con precisión las señales del sensor (acelerómetros)
- El sistema debe poder convertir eficientemente la información de los sensores, manipularla y finalmente enviarla al esclavo de manera inalámbrica
- Minimizar la posibilidad de que existan colisiones y otros errores
- Minimizar el espacio ocupado por el dispositivo
- Se debe diseñar un maestro con un diseño con cierta ergonomía

7.2 Componentes del dispositivo maestro y diseño de tarjeta

Los dispositivos que componen al dispositivo maestro son:

- Sensado (Acelerómetro BMA180)
- Arduino Duemilanove (microcontrolador ATmega328)
- Comunicación inalámbrica por Xbee (módulo esclavo)

Se ha realizado un diseño (no implementado) de todos los componentes del dispositivo maestro en una tarjeta, se pueden ver a continuación una imágen del diseño realizado:

7.2.1 Vistas superior e inferior de la tarjeta diseñada

Se muestran en la Imagen 17 la tarjeta que se ha diseñado, donde se puede apreciar que cuenta con todos los componentes antes mencionados, así como resistencias, capacitores, reguladores y un cristal. El acelerómetro y el módulo XBEE se montan sobre la tarjeta.

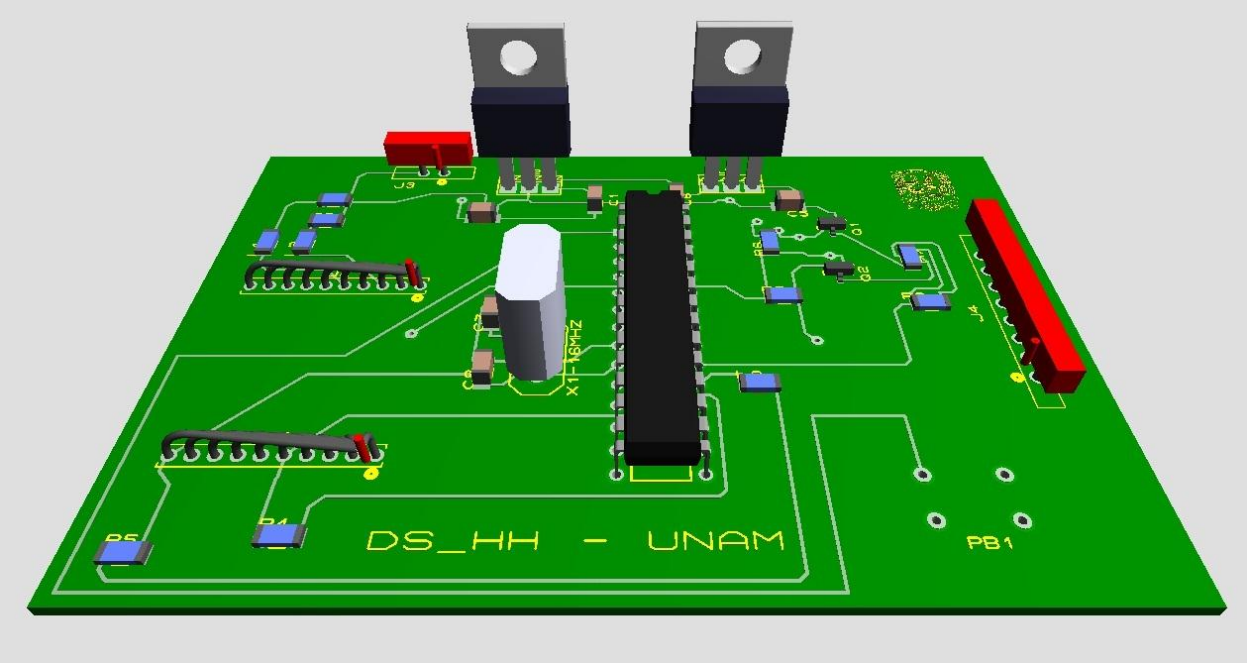


Imagen 17 Vista superior de la tarjeta del dispositivo maestro

8 ACELERÓMETRO

8.1 El acelerómetro como sensor

Los acelerómetros se han vuelto muy populares en la industria del consumo electrónico. En realidad un acelerómetro es solamente un sensor que mide la aceleración, es decir que tan rápido algo aumenta de velocidad o frena. Como es de esperarse, la aceleración es medida en m/s^2 o en fuerzas G. Una fuerza G equivale a 9.8 m/s^2 (más o menos dependiendo de la altitud y otras condiciones).

Los acelerómetros son usados tanto para medir aceleración estática (como la gravedad) y aceleración dinámica (cambios bruscos e inesperados). Una de las aplicaciones más comunes para los acelerómetros es medir inclinación. Debido a que son afectados por la fuerza de la gravedad, un acelerómetro puede indicarnos cuál es la orientación respecto a la superficie de la tierra. Otras aplicaciones comunes de los acelerómetros es medir movimiento y medir caída libre.

En el presente proyecto el acelerómetro es el sensor utilizado dentro del dispositivo maestro, por lo que juega un papel fundamental en el funcionamiento del sistema.

8.2 Características importantes para seleccionar un acelerómetro

- *Rango*: el rango nos indica los límites superiores e inferiores de la aceleración que se puede medir. Es importante elegir bien el rango necesario para el proyecto donde se usará porque también influye en la precisión de sensado.
- *Interface*: existen tres tipos
    - Analógicos: producen un voltaje directamente proporcional a la aceleración a la que son sometidos.
    - PWM: producen una señal cuadrada de una frecuencia fija, pero su ciclo de trabajo varía de acuerdo a la aceleración.
    - Digitales. Generalmente incluyen una interfaz serial (SPI o I^2C). Éstos son los que poseen más características y son afectados en menor medida por el ruido.
- Número de ejes medidos: los hay de 1, 2, 3 y 6 ejes (incluyen aceleración angular).
- Ancho de banda: nos indica que tan frecuentemente puede ser medido el acelerómetro y ser confiable en los datos que entrega.
- Consumo de energía: generalmente alrededor de las centenas de µA. Algunos acelerómetros tienen la función de 'dormirse' cuando no son ocupados y así ahorrar energía.
- Características adicionales: algunos acelerómetros incluyen funcionalidades extra.

8.3 Selección del acelerómetro

Con el fin de realizar una manipulación precisa y en la medida de lo posible, con las menores fluctuaciones en los valores de la aceleración, se ha encontrado que los acelerómetros digitales son una buena opción. Existen muchos tipos de acelerómetros digitales, sin embargo, gracias a que su costo no es tan elevado y a que tienen una resolución de 14 bits (suficientes para poder medir cambios de 0.25 mg) se ha elegido trabajar con el acelerómetro BMA180 de la marca Bosch.

Para ver una tabla comparativa entre algunos diferentes comerciales ver el Apéndice A.

8.4 Estudio del acelerómetro elegido BMA180

8.4.1 Características y propiedades clave del acelerómetro

- Acelerómetro de 3 ejes con sensor de temperatura integrado
- Operación de alto rendimiento (colocar el rango)
    - Operación del ADC de 14 bit
- Interfaces digitales
    - I^2C
    - SPI de 4 cables
    - Pin de interrupción
- Rangos programables de medición (1g, 1.5g, 2g, 3g, 4g, 8g, 16g)
- Filtros digitales integrados programables
    - 8 filtros pasa-bajas de 10, 20, 40, 75, 150, 300, 600, 1200 Hz
    - 1 filtro pasa-altas de 1Hz
    - 1 filtro pasa-bandas de 0.2 – 300Hz
- Memoria EEPROM de 256 bits para calibración
- Poco consumo de energía, típicamente 650µA, en varios modos de consumo de energía
    - 2 modos estándar
        - Bajo ruido
        - Bajo consumo
    - 2 modos intermedios
    - Modo de dormido (*sleep-mode*)
    - Modo de despertarse (*wake-up mode*)
- Bajo voltaje de operación
    - +1.62V … +3.6V para VDD
    - +1.2V … +3.6V para VDDIO
- Rangos de temperatura -40°C a 85°C
- Interrupciones programables

- Despertado (*wake-up*)
- Detección de bajo g (*low-g detection*)
- Detección de alto g (*high-g detection*)
- Detección del '*tap*' (golpe suave)
- Detección de la inclinación (*tilt*)

- Calibración de varios parámetros
  - Offset
  - Sensibilidad
  - Coeficiente de temperatura para el offset (*TCO*)
  - Coeficiente de temperatura para la sensibilidad (*TCS*)

8.4.2 Aplicaciones típicas

El acelerómetro es utilizado de forma regular en funciones de sensado de inclinación, movimiento y vibración en campos como la navegación, la robótica, la realidad virtual, los celulares, etc.

8.4.3 No linealidad

En la Tabla 2 se muestran los valores de no linealidad en cada rangos de aceleración: (no corregida)

Rango de medición	Porcentaje de no linealidad
1g	±0.15%
1.5g	±0.15%
2g	±0.15%
3g	±0.25%
4g	±0.25%
8g	±0.75%
16g	±0.75%

Tabla 2 No linealidad del acelerómetro BMA180

8.4.4 Modo de dormir(Sleep mode) y función wake-up

Éstos útiles modos para el ahorro de energía (hasta un 97%) y reducción de corriente a menos de 1 µA funcionan modificando un bit del registro. El tiempo de arranque del *wake-up* es de 3.5 ms. Sin embargo, por el momento estas funciones no están implementadas.

8.4.5 Diagrama de bloque

Los componentes más importantes como son los elementos micro mecánicos de sensado en las direcciones 'x', 'y' y 'z' y el sensor de temperatura (medición de variables físicas). Los elementos analógicos funcionan como pre-amplificadores y pre-filtros. El convertidor analógico-digital es necesario

para toda la etapa digital (regulación de voltaje, offset, calibración, filtrado). Las interrupciones y las interfaces de comunicación son los elementos de salida del sensor, como se puede ver a continuación:

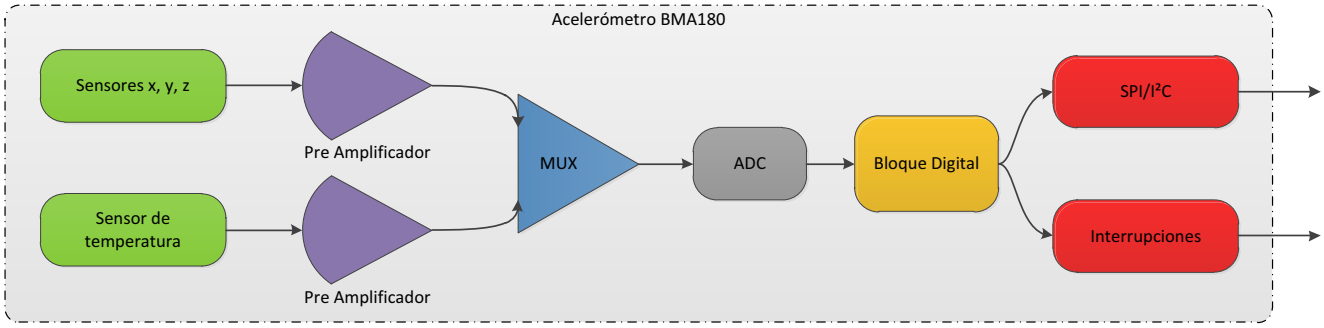


Diagrama 6 Flujo de información en los componentes del acelerómetro

8.4.6 *Ajustes principales*

Para poder configurar el acelerómetro con las características elegidas, es necesario acceder a su memoria interna y modificar los registros. Mediante alguna de las interfaces digitales es necesario enviar el byte que se desea escribir, y el valor que desea escribirse. Es, por lo tanto, importante conocer claramente los registros que se modificarán y sus valores posibles.

Se muestra en la Tabla 3 se muestran los ajustes principales:

- Ajuste del rango (*range*)

Registro binario/Modo	Escala de la aceleración [±g]	Resolución del ADC [mg]
000	1	0.13
001	1.5	0.19
010	2 (implementada)	0.25
011	3	0.38
100	4	0.50
101	8	0.99
110	16	1.98
111	Código no autorizado	Código no autorizado

Tabla 3 Tabla de rangos del acelerómetro BMA180)

- Ajuste del ancho de banda (*bw*) (ver Tabla 4)

Registro binario	Ancho de banda seleccionado
0000	10
0001	20
0010	40
0011	75

0100	150
0101	300
0110	600
0111	1200
1000	Pasa-altas de 1Hz
1001	Pasa-bandas de 0.2 – 300Hz
1010 a 1111	Código no autorizado

Tabla 4 Tabla de anchos de banda del acelerómetro BMA180

- Configuración del modo (*mode_config*) (ver Tabla 5)

Registro binario	Descripción	
00	Bajo ruido	Corriente alta Ancho de banda completo (1kHz)
01	Ultra bajo ruido	Corriente alta Ancho de banda reducido (300Hz)
10	Bajo ruido con bajo poder	Ancho de banda reducido (150Hz) 1200 muestras/segundo
11	Bajo poder	Ancho de banda reducido por factor de 2 Mayor ruido

Tabla 5 Tabla de modos del acelerómetro BMA180

8.4.7 Aceleración en x, y, z

Los valores de la aceleración están guardados en los registros (*acc_x, acc_y, acc_z*) para ser leídos por la interface serial. Los valores en 14 bits están escritos en complementos de 2, como se ve en la Tabla 6:

	1g	1.5g	2g	3g	4g	8g	16g
10 0000 0000 0000	-1.00000 g	-1.50000 g	-2.00000 g	-3.00000 g	-4.00000 g	-8.00000 g	-16.00000 g
10 0000 0000 0001	-0.99988 g	-1.49982 g	-1.99976 g	-2.99963 g	-3.99951 g	-7.99902 g	-15.99805 g
11 1111 1111 1111	-0.00012 g	-0.00018 g	-0.00024 g	-0.00037 g	-0.00049 g	-0.00098 g	-0.00195 g
00 0000 0000 0000	0.00000 g	0.00000 g	0.00000 g	0.00000 g	0.00000 g	0.00000 g	0.00000 g
00 0000 0000 0001	+0.00012 g	+0.00018 g	+0.00024 g	+0.00037 g	+0.00049 g	+0.00098 g	+0.00195 g
01 1111 1111 1111	+0.99988 g	+1.49982 g	+1.99976 g	+2.99963 g	+3.99951 g	+7.99902 g	+15.99805 g

Tabla 6 Formato de los datos en el acelerómetro BMA180

8.4.8 Número de muestras

El número de muestras depende del ancho de banda como se muestra en la Tabla 7:

Ancho de banda	1200	600	300	150	75	40	20	10
Número de muestras	0	6	9	18	35	64	127	253

Tabla 7 Número de muestras en el acelerómetro BMA180

8.4.9 Elección de la interface serial

Existen dos posibles protocolos posibles:

- I²C es activado cuando el registro CSB = 1
- SPI es activado cuando el registro CSB = 0

Debido a la facilidad de implementación de la interface I²C, que únicamente necesita de la línea de reloj (*SCL: Serial Clock*) y la línea de datos (*SDA: Serial Data Line*), y al hecho de que en la plataforma Arduino existe una librería para el uso de dicha interfaz, se ha elegido trabajar con ésta y no con la SPI (*Serial Peripheral Interface Bus*) que utiliza 4 líneas de comunicación.

8.4.10 Diagrama de conexión para la interface I²C

El diagrama de conexión del productor del acelerómetro nos muestra cómo debe conectarse la interfaz I²C (ver Imagen 18), sin embargo, en la práctica es importante considerar que el acelerómetro trabaja a 3.3V y que el microcontrolador trabaja a 5V, por lo que se debe poner una etapa de cambio de voltaje para cada una de las líneas. Esto se puede lograr sin grandes dificultades utilizando transistores MOSFET, como los transistores BSS138, que son transistores tipo n.

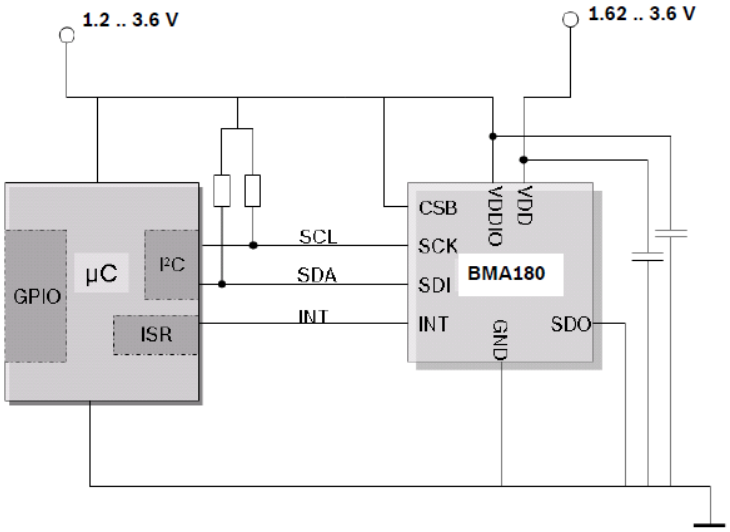


Imagen 18 Conexión del protocolo I2C en el acelerómetro BMA180

8.4.11 Algunos de registros de memoria importantes en la memoria EEPROM

En la memoria EEPROM del acelerómetro existen muchos registros (700 bits de datos de lectura o para valores de calibración), algunos de ellos utilizados en funciones más complejas dentro del acelerómetro. En la Tabla 8 se muestran los registros más importantes utilizados en el presente proyecto:

Nombre del registro	Valor default	Hexadecimal en la EEPROM	Lectura/escritura	Volátil/No volátil
bw	0100	40 (bit7, bit6, bit5, bit4)	Lectura/escritura	Volátil
range	010	55 (bit3, bit2, bit1)	Lectura/escritura	Volátil
mode_config	00	50 (bit1, bit0)	Lectura/escritura	Volátil
acc_x_lsb	-	02	Lectura	Volátil
acc_x_msb	-	03	Lectura	Volátil
acc_y_lsb	-	04	Lectura	Volátil
acc_y_msb	-	05	Lectura	Volátil
acc_z_lsb	-	06	Lectura	Volátil

Tabla 8 Registros importantes del acelerómetro BMA180

9 MICROCONTROLADOR ATmega328

9.1 *Microcontroladores*

Un microcontrolador, a veces abreviado µC, uC o MCU, es una pequeña computadora en un circuito integrado que contiene un procesador, una memoria y periféricos de entrada y/o salida programables. Los microcontroladores están diseñados para aplicaciones embebidas, a diferencia de los microprocesadores en computadoras personales y otras aplicaciones de uso general.

Los microcontroladores son usados en productos y dispositivos controlados automáticamente como en sistemas de control de motores de automóviles, dispositivos médicos, controles remotos, máquinas de oficina, herramientas, etc. Al reducir el tamaño y costo comparado con diseños que usen microprocesadores separados, memoria y periféricos de entrada y/o salida, los microcontroladores hacen más económico el control digital de varios dispositivos y procesos.

Los programas en los microcontroladores deben residir en la memoria que es para dicho propósito. Compiladores y ensambladores son usados para convertir el lenguaje de alto nivel y lenguaje ensamblador a lenguaje máquina para que pueda ser guardado en la memoria. Componentes y características principales de los microcontroladores (ver Imagen 19):

- CPU (*Central Processing Unit*)
- Interrupciones externas y de tiempo
- Entradas y salidas digitales y analógicas
- PWM (*Pulse Width Modulation*) para control de dispositivos como motores
- UART (*Universal Asynchronous Receiver/Transmitter*) para lineas serials
- Interfaces de comunicación, como SPI, CAN, I^2C
- Convertidores analógicos-digitales y convertidores digitales-analógicos
- Memoria ROM, EPROM, EEPROM o Flash
- Un generador de señal de reloj

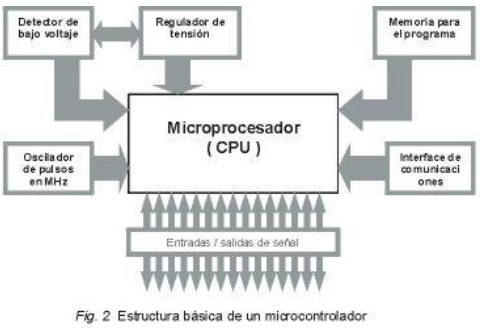


Fig. 2 Estructura básica de un microcontrolador

Imagen 19 Elementos que componen a un microcontrolador

9.2 *Microcontrolador Arduino Duemilanove*

Arduino, es una plataforma basada en un circuito impreso y un entorno de desarrollo basado en el lenguaje *Wiring*/*Processing*. Arduino puede ser usado para desarrollar instalaciones interactivas autónomas, o bien, interactuar con software instalado en un ordenador.

El Arduino Duemilanove es un microcontrolador basado en el ATmega380 (ver Imagen 20). Tiene 14 entradas o salidas digitales, de las cuales 6 pueden ser usadas como PWM (*Pulse Width Modulation*), 6 entradas analógicas, puertos seriales, un cristal de 16 MHz y una conexión USB. Características:

- Voltaje de operación de 5V
- Voltajes de entrada recomendados de 7 a 12V
- Voltajes máximos de entrada 6 a 20V
- Corriente por los pines de entrada o salida de 40 mA
- Memoria flash de 32 kB
- Memoria SRAM de 2 kB
- Memoria EEPROM de 1 kB

Además de ello posee muchas cualidades similares en su lenguaje de programación a la programación en C, y por ello facilita mucho su implementación. Por ello ha sido utilizado para responder a las labores de lectura del acelerómetro, y de envío de datos al módulo inalámbrico XBEE.

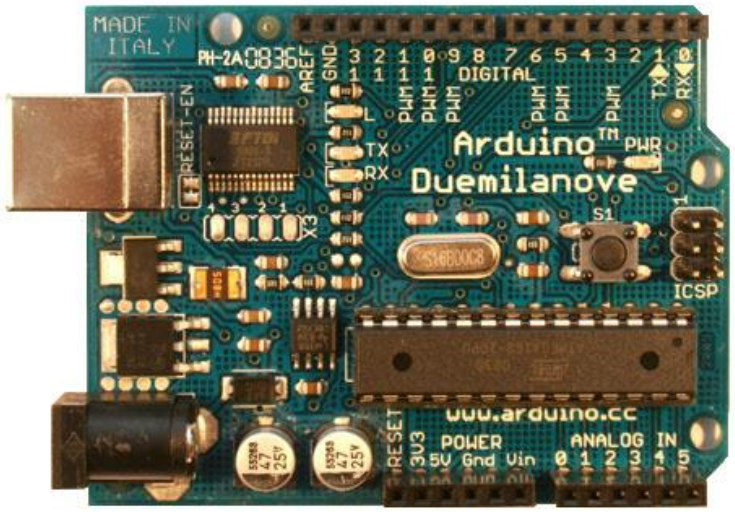


Imagen 20 Microcontrolador Arduino Duemilanove

9.2.1 Interfaz de comunicación I²C

Como se ha mencionado en el pasado capítulo, el protocolo elegido, I²C, posee una librería que facilita su implementación en el microcontrolador ATmega328. Es necesario, sin embargo, comprender su funcionamiento y la manera como interactúa con dicho microcontrolador.

I²C (*Inter-Integrated Circuit*) es un bus de comunicación usado para comunicar circuitos integrados (periféricos) de baja velocidad a una motherboard, microcontrolador o algún sistema embebido. Usa únicamente dos líneas bidireccionales de comunicación, SDA y SCL con resistencias de '*pull up*' y con voltajes típicos de 3.3 V o 5 V.

En su diseño el protocolo tiene direcciones de 7 bits comúnmente con 16 direcciones reservadas. La velocidad del bus es de 100 kbit/s en modo estándar o de 400kbit/s, 1Mbit/s o 3.4Mbit/s en otras configuraciones. Tiene dos tipos de nodos, que son el nodo maestro y el esclavo, el nodo maestro escribe mensajes en el esclavo o lee mensajes de él de la siguiente manera:

- La transferencia de datos inicia con un bit de inicio '*Start*' (S) mientras la señal del reloj es alta y la de datos baja
- Entonces la señal SDA envía el dato cuando la señal SCL es baja y cuando la señal de reloj sube, el dato es leído y registrado. Éste proceso se repite hasta que ha sido enviado todo el mensaje completo, como se aprecia en la Imagen 21.
- Cuando todos los bits de información han sido enviados, el bit de parada '*Stop*' (P) es enviado mientras nuevamente liberando la señal SDA es baja y la señal SCL se mantiene alta.

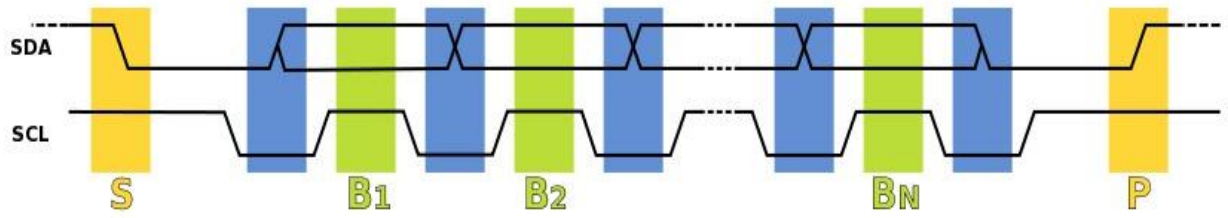


Imagen 21 Transferencia de información del protocolo I2C

El diagrama de conexiones entre el microcontrolador y el acelerómetro está descrito en la Imagen 22, que posibilitan la implementación de la comunicación entre el acelerómetro BMA180 y el microcontrolador ATmega328:

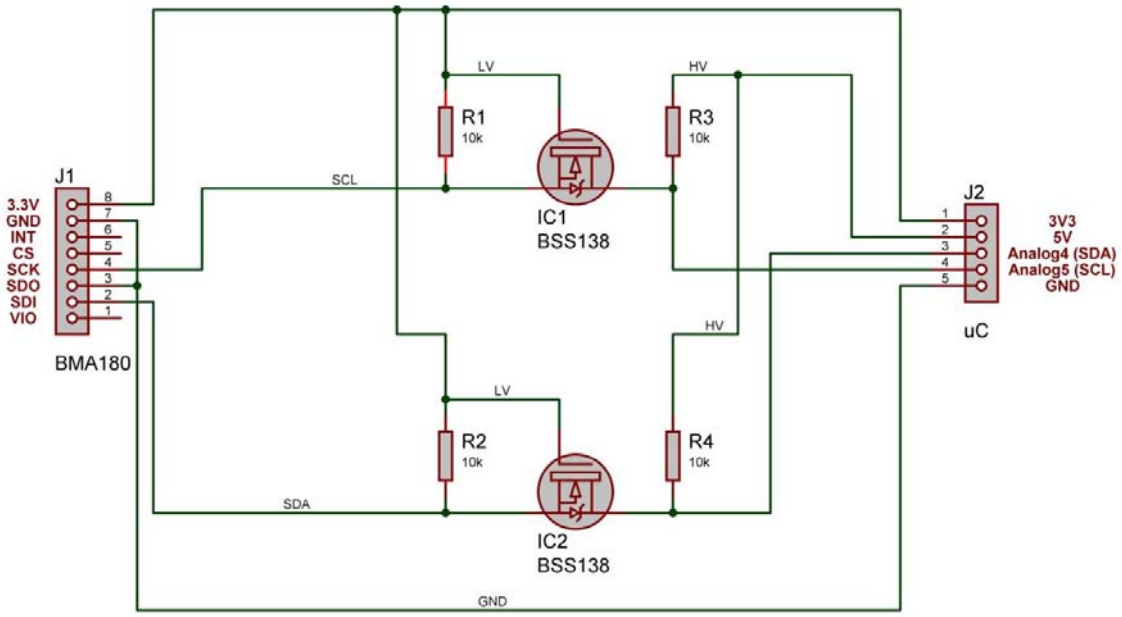


Imagen 22 Conexión del microcontrolador con el acelerómetro implementada

9.3 Lógica de programación en el Arduino Duemilanove

La programación en compilador Arduino se realiza para descargar código en el microcontrolador ATmega328, que se encuentra dentro de la tarjeta del Arduino Duemilanove. La única función de dicho microcontrolador es leer la información del acelerómetro BMA180 utilizando el protocolo I^2C, traducirla a información útil y enviarla al módulo XBEE esclavo. Como se ha visto con anterioridad, se leen 6 bytes de información de aceleración, donde 2 bytes corresponden a cada uno de los ejes coordenados. Lo que se hace es leer cada uno de los registros de datos de la aceleración (registros 02 a 07 de la memoria interna del acelerómetro) y transformarlos en información útil, ya que como solamente se utilizan 14 bits de los 16 que se leen, debemos realizar un corrimiento en ambos bytes, y unirlos en un entero. Ésta misma función debe tomar en cuenta los formatos de los números negativos, que es '*two's complement*' (complementos de dos)

Más adelante se realiza un promedio de datos en cada uno de los ejes para compensar un poco los cambios continuos de lectura del sensor y finalmente se vuelve a descomponer la información en 6 bytes de aceleración y un 7º byte que corresponde a la variable de tiempo. Ésta información se envía al módulo XBEE, pero para poder transmitir la información al XBEE debemos cumplir plenamente con el protocolo designado, y por lo tanto imprimimos un total de 26 bytes a través de las terminales Tx y Rx (pines digitales del puerto serial conocidos como UART o USART), donde el último de los bytes es el

checksum, que es un byte de verificación para saber si no se han presentado errores en la transmisión de la información.

Existe además una función de '*setup*' donde configuramos al acelerómetro sus parámetros principales: ID del dispositivo, filtro, rango de aceleración y un bit que permite la escritura sobre la memoria EEPROM. En éste caso se ha elegido utilizar el rango de ±2g, un filtro pasa bajas de 10Hz. Es importante que ésta función no esté corriendo continuamente y que no se llame a ella desde el *loop* ni cada vez que se inicie el programa, dado que la memoria EEPROM se desgasta y no puede escribirse sobre ella muchas veces. Por éste motivo hemos realizado la configuración una sola vez y desde ahí no se ha utilizado nuevamente.

El Diagrama 7 muestra la forma general en que fluye el programa:

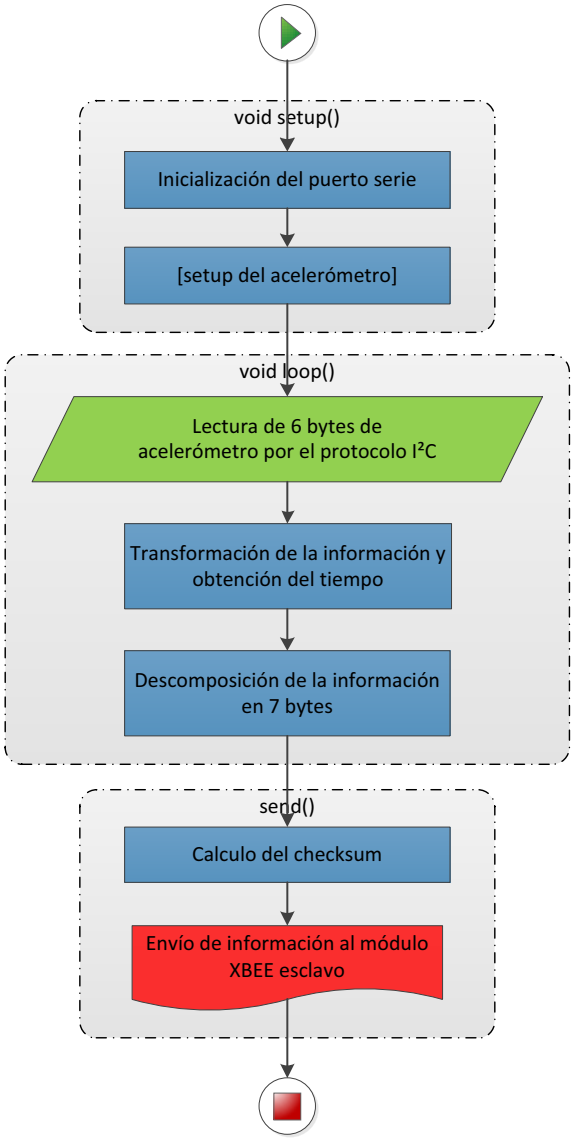


Diagrama 7 Flujo de información en el ATmega328

10 CONTROL CENTRAL EN LA PC

El control central dentro de la PC es en realidad el sistema donde se encuentran los procesamientos más complejos y es la unión entre el sistema de sensado y el sistema de actuadores.

10.1 Lenguaje de programación C#

La programación se realizó en Microsoft Visual C# 2010, que forma parte de la plataforma .Net de Microsoft. C# es un lenguaje de programación de multi-paradigma orientado a objetos similar a C++. La plataforma .Net (.NET Framework) es una plataforma de software para los sistemas operativos de Windows. Incluye una librería muy amplia que soporta varios lenguajes de programación, lo que permite que exista interoperabilidad entre lenguajes, lo que significa que cada lenguaje puede utilizar código escrito en otro lenguaje. Ésta plataforma utiliza un entorno de ejecución llamado CLR (*Common Language Runtime*).

En la Imagen 23 se muestra como pueden varios lenguajes de programación, como C#, J# y Visual Basic tener interoperabilidad:

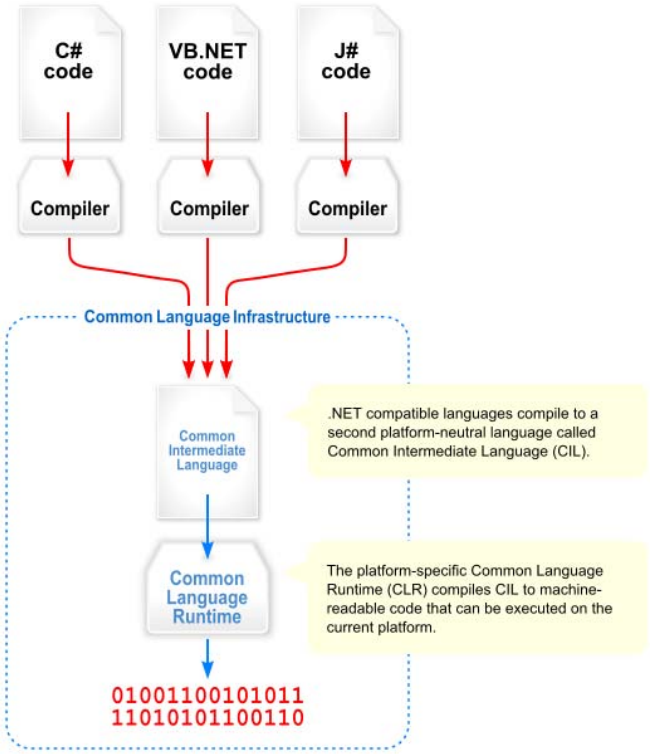


Imagen 23 Plataforma .Net de Windows

10.2 Tareas que desempeña la PC en el control

La PC, al ser el dispositivo con mayor capacidad de procesamiento y visualización de los sistemas con los que se cuentan, es la encargada de realizar las tareas más complejas que requieren de poco tiempo de procesamiento. Las tareas que se desempeñan en la PC son las siguientes:

- Lectura de datos seriales del módulo XBEE coordinador (o mediante USB en el caso alámbrico)
- Obtención de datos reales de aceleración
- Calibración de los valores de aceleración
- Visualización de diversos tipos para el usuario
    - Gráficas de barras de las aceleraciones en todos sentidos
    - Gráficas generales de las posiciones en todos los ejes
    - Gráfica polar mostrando tendencias y ángulos
    - Gráfica de trayectorias seguidas
    - Indicadores diferentes para asistencia al usuario
- Realización de la Integración de dos tipos:
    - Integración por regla de Boole
    - Integración trapezoidal (Newton de 1er orden)
- Realización de la cinemática inversa
- Guardado de trayectorias útiles
- Control Manual/Automático
    - Control Manual de Articulaciones
    - Control Manual de Posiciones
    - Control Automático
- Transmisión de instrucciones al microcontrolador Arduino Mega
- Paro de Emergencia

La aplicación realizada, que implementa todas las anteriores tareas, tiene la capacidad de hacer que el usuario tenga mucho control sobre el sistema, y que también pueda visualizar claramente lo que está ocurriendo. En general, es formado por una exclusiva pantalla donde se muestran los valores de las principales variables (aceleraciones, posiciones y ángulos), las gráficas de estado (gráficas de barras, polares, trayectorias, posición). De igual manera cuenta con mucho botones que permiten manipular al sistema de diferentes maneras y en sus diferentes modos, como lo son el control manual de articulaciones, el control manual de posiciones y el control automático.

Es importante siempre abrir los puertos seriales de los dispositivos maestros y esclavos para que funcione el sistema, de otra manera no existen datos de entrada ni de salida y el sistema no funciona. Existe una excepción, y es cuando el sistema no se encuentra en el modo de control automático, en éste caso no es necesario abrir el puerto serial del dispositivo maestro para poder mover al manipulador.

Las funcionalidades generales del software se muestran en el Diagrama 8:

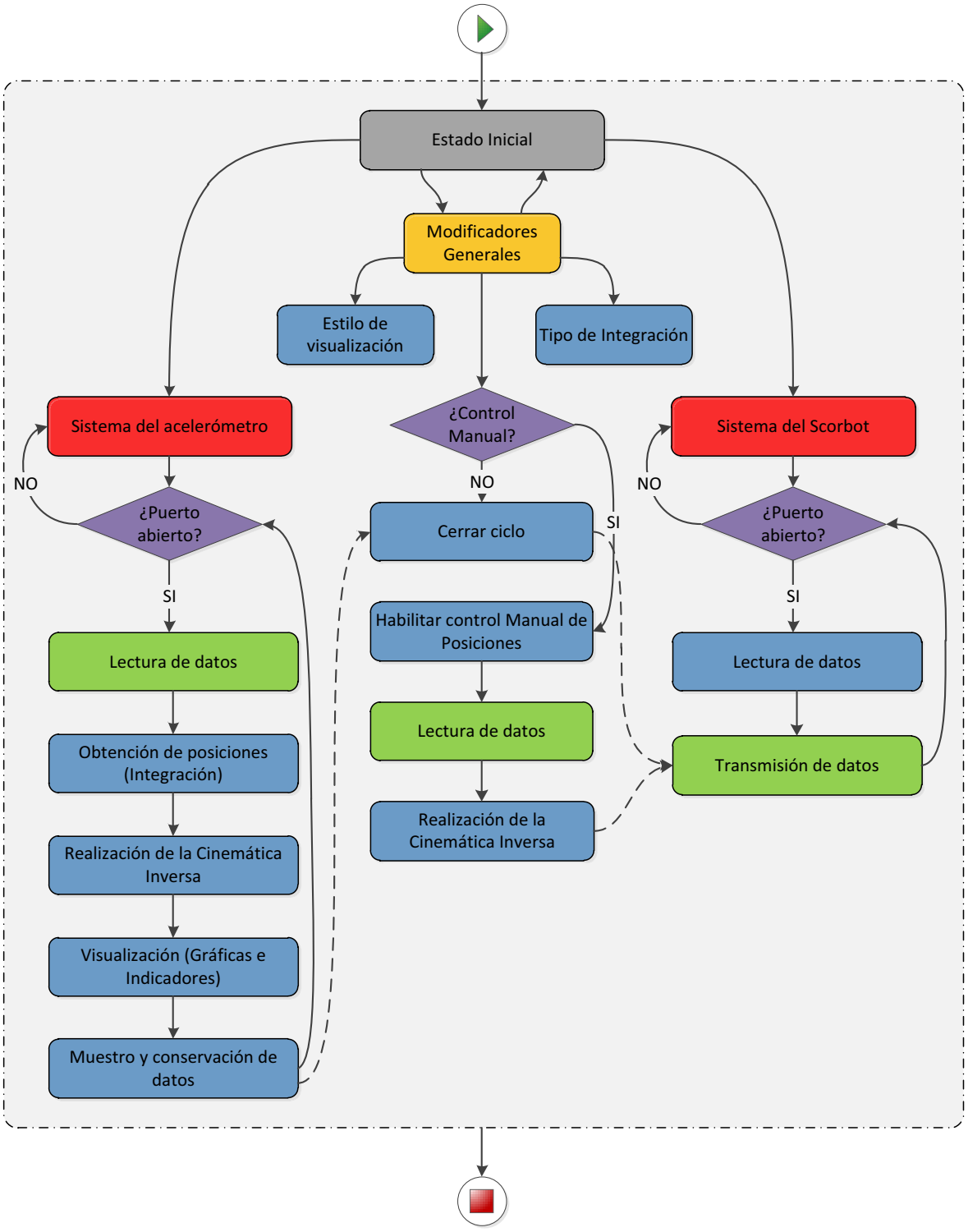


Diagrama 8 Funciones generales de la aplicación en C#

10.3 Función inicial

La función inicial en el programa es '*public Telerobtica()*' y se encarga de las siguientes tareas:

- Inicializar los diferentes componentes del sistema
- Iniciar los mensajes que despliega el puntero del mouse al ser recargado sobre objetos importantes (elemento *tooltip*).
- Otorgar las posiciones iniciales de los ángulos del Scorbot
- Crear las diferentes interrupciones que utiliza el programa
    - Interrupciones de los puertos serie
    - Interrupciones de algunas casillas (como los de la Cinemática Manual)
- Iniciar algunos objetos gráficos que se usan más adelante
- Definir los colores iniciales de algunos elementos que cambian dependiendo de su estado

10.4 Lectura de datos

La lectura de datos seriales del módulo XBEE coordinador debe respetar ciertas reglas, y en realidad debemos permanecer dentro de lo que el protocolo Zigbee dicta. El paquete de información recibido tiene 26 bytes organizados de forma que se muestra en la Imagen 24:

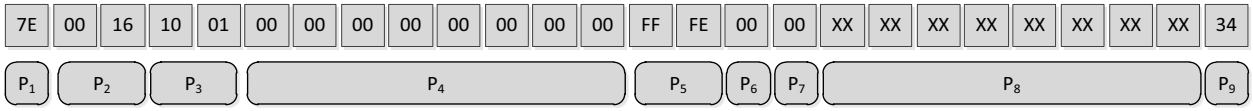


Imagen 24 Paquete de información recibido en el módulo maestro del XBEE

- P_1 = Start Delimiter
- P_2 = Tamaño del dato
- P_3 = Api Frame Type: Node Identification Indicator
- P_4 = Dirección de 64 o 00 si enviado al coordinador
- P_5 = Dirección de red de 16 bit FFFE
- P_8 = Mensaje de 8 bytes
- P_9 = Checksum

El funcionamiento general sigo un algoritmo sencillo, donde se espera a que se llene el buffer cierta cantidad de bytes, luego se leen todos los bytes, y se calcula el *checksum*, para verificar que la información haya sido recibido de manera correcta. Si la información es correcta se procede a obtener datos reales de aceleración (transformar el número entero a una aceleración en términos de g). El Diagrama 9 muestra el proceso descrito anteriormente:

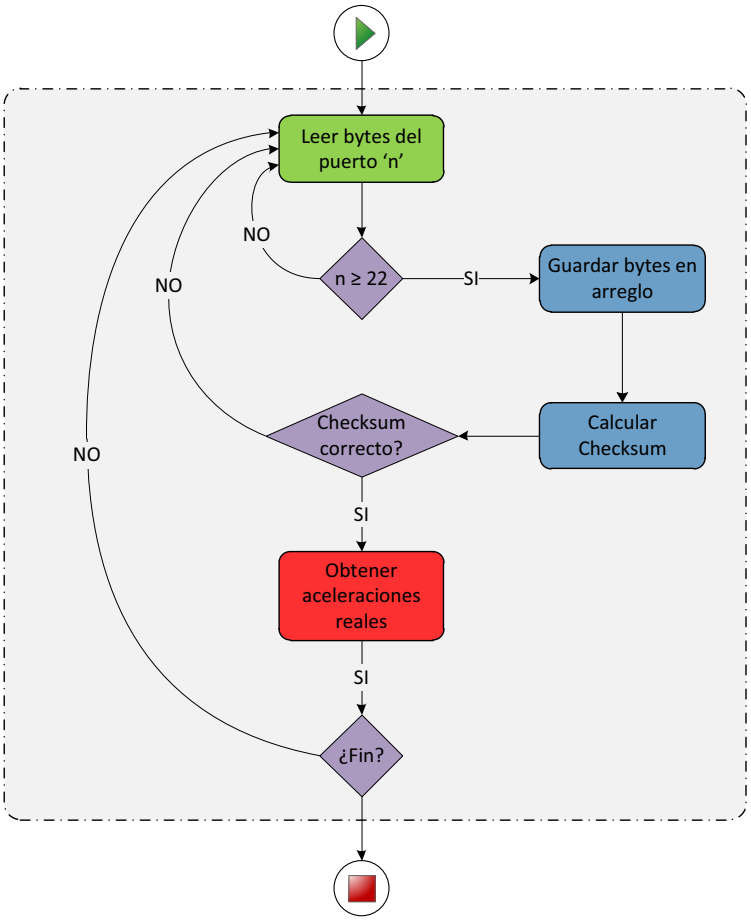


Diagrama 9 Verificación de la información recibida en el módulo XBEE maestro

10.5 Obtención de datos reales de aceleración

Para obtener los datos reales de aceleración a partir de los bytes que se obtuvieron en la comunicación serial, es indispensable tomar la información adecuada de los bits de aceleración en los bytes, para más tarde transformarlos a aceleración real tomando en cuenta el rango de aceleración al cual pertenecen y las reglas de transformación (se debe conocer el formato de salida de los datos del acelerómetro BMA180). El Diagrama 10 muestra como se obtienen los datos reales de aceleración:

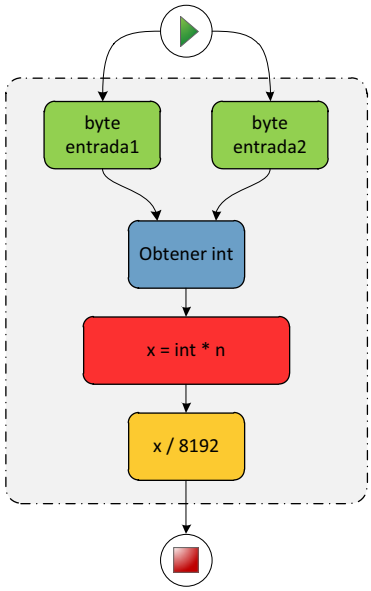


Diagrama 10 Obtención de datos reales de aceleración

Lo que se hace es teniendo los dos bytes se obtiene un número entero uniendo la parte más significativa y menos significativa de éste con los bytes. Teniendo éste número, se sabe además que el valor máximo que se puede tener en el entero es el del valor máximo en el rango de aceleración. Como se tienen 14 bits en los valores de aceleración que el acelerómetro BMA180 nos entrega, se tiene un valor máximo de 8192 (2^13) más el signo positivo o negativo de esa aceleración. Entonces es fácil ver que es necesario que dividir nuestro entero entre éste valor y multiplicarlo por el rango de aceleración que se esté leyendo. Con esto se obtiene una aceleración real en g's [m/s^2]

10.6 Calibración de valores de aceleración

Ésta calibración además tiene la función de asegurar que el sistema se mantenga estable mientras el acelerómetro no esté en uso, es decir que se sobrescriben los valores de la aceleración a cero en caso de que las tres aceleraciones se encuentren cerca del punto de reposo. Esto permite mantener las mismas aceleraciones y posiciones, así como otros parámetros sin cambio mientras el acelerómetro se mueva poco; además permite mover al acelerómetro con finura en caso de que se requiera mover sin enviar información al Scorbot.

Es importante siempre mantener la orientación del dispositivo maestro de manera que la aceleración de la gravedad recaiga exclusivamente en el eje vertical, de otra manera existe una componente de la aceleración de la gravedad en los ejes 'x', y 'y'.

10.7 Integración numérica

La integración numérica representa un paso de vital importancia para que el sistema se mueva de forma continua. Es en realidad ésta parte la que transforma la información generada por el dispositivo maestro y las posiciones utilizadas de aquí en adelante en varios procesos. Debido a ésta importancia, es conveniente realizarla de la manera más eficiente y precisa posible. Lo que requiere realizar es una doble integración de los datos de aceleración para obtener posiciones. Naturalmente la aceleración $(\ddot{x})$ es la segunda derivada de la posición (x). Como no tenemos ninguna función analítica y se tienen únicamente datos de aceleración en cada uno de los ejes de forma discreta, conociendo también el tiempo en que dichos datos son generados, se puede utilizar una integración siguiendo métodos numéricos.

En análisis numérico, la integración numérica constituye una amplia familia de algoritmos para calcular el valor numérico de una integral definida y, por extensión, para obtener soluciones numéricas de ecuaciones diferenciales. El problema básico es calcular la integral definida que tiene la siguiente forma:

$$\int_a^b f(x)dx$$

Si f(x) es una función suave y si los límites de integración son finitos, existen muchos métodos para aproximar la solución con diferentes tipos de precisiones, algunos lo ven como un área bajo la curva, como en la Imagen 25. (7)

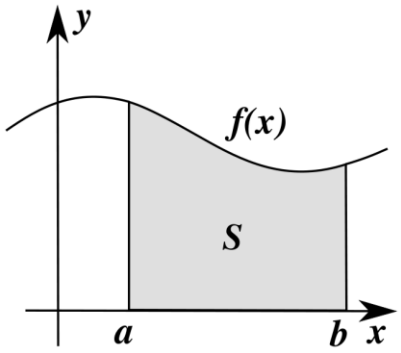


Imagen 25 Integral vista como un área bajo la curva

Las fórmulas de integración por interpolación, donde los nodos deben estar igualmente distanciados (en el caso presente están distanciados casi de forma casi uniforme, existiendo siempre un tiempo entre 50 y 51 milisegundos entre datos de aceleración), son llamadas generalmente fórmulas de Newton-Cotes. Estas fórmulas son particularmente eficientes para funciones tabuladas, como las que se utilizaban antes de la era de la computación. Algunos casos particulares de estas fórmulas son, dependiendo del grado de la fórmula de Newton-Cotes, son los siguientes:

10.7.1 Regla del Trapecio (grado 2)

Es una función sencilla de integración que usa solamente dos puntos, como se muestra a continuación:

$$\int_a^b f(x)dx = \frac{b-a}{2}\big(f(a)+f(b)\big)$$

10.7.2 Regla de Boole (grado 4)

Ésta regla es llamada así en honor a George Boole, su creador. Tiene la siguiente forma:

$$\int_a^b f(x)dx = \int_{x_0}^{x_4} f(x)dx = \frac{b-a}{90}(7f(x_0)+32f(x_1)+12f(x_2)+32f(x_3)+7f(x_4))$$

Donde $x_0 \ldots x_4$ son los puntos igualmente distanciados en el rango de integración de 'a' a 'b'.

10.7.3 Lógica de programación de la integración

Se ha realizado el estudio de dos diferentes formas de integrar numéricamente funciones. La integración trapezoidal requiere menos puntos, dos únicamente, y en esto está su mayor ventaja. Puede obtenerse un dato de posición por cada dato de aceleración que se tiene como entrada, debido a que el último punto, es en realidad el primer punto del par requerido. Esto funciona de igual manera para las dos integraciones que se deben realizar, y, por lo tanto el tiempo necesario para generar puntos de aceleración es de aproximadamente 50.5 milisegundos.

En el caso de la integración por la regla de Boole, se requieren cinco puntos, y debido a que se tiene que integrar dos veces consecutivas, se necesitan cuatro puntos de velocidad (el quinto punto necesario es el primero, y corresponde al último punto de aceleración del cálculo anterior). Para obtener cada uno de éstos puntos de velocidad es necesario tener cinco puntos de aceleración, donde de igual manera el punto anterior puede reducirnos en uno la cantidad de puntos necesarios. En el mejor de los casos se requiere de 17 puntos que se traducen en aproximadamente 858.8 milisegundos. Puede verse que se requiere de mucho más tiempo para obtener los datos de posición en ésta integración.

La integración de Boole tiene mayor precisión, sobre todo para funciones que cambian bruscamente de valores, como es el caso de los datos generados en el acelerómetro. La integración trapezoidal funciona como si estuviera obteniendo un promedio de los valores máximo y mínimo dentro del rango de datos.

Las consideraciones de tiempo son muy importantes si se desea realizar una teleoperación que el operador pueda caracterizar como intuitiva. Es por ello que se ha elegido utilizar la integración trapezoidal, aunque más imprecisa, como la integración programada por default. Pero, para tareas donde se requiera un poco mayor de finura, se ha realizado de igual manera la programación de la integración de Boole. El usuario tiene la posibilidad de cambiar a ésta configuración si es que así lo desea.

En el Diagrama 11 se muestra ésta funcinoalidad del sistema:

Diagrama 11 Flujo de información en la integración de las aceleraciones

10.8 Visualización

La parte de visualización en Tele-robótica adquiere importancia debido a que el usuario debe tener una manera de observar lo que está realizando en el mundo real, y nace de la necesidad de poder ver los valores de las variables principales y saber si están dentro de un rango de operación normal. La tarea de visualización es complicada porque es importante que se muestre la información de forma sencilla (sin datos irrelevantes que cansen a la vista), probablemente de forma gráfica, para que se pueda obtener la mayor información del estado del sistema en poco tiempo. De ésta manera se pueden evitar muchos accidentes y pueden detectarse posibles complicaciones de forma muy veloz.

A continuación se muestra en imágenes como ha quedado éste trabajo de visualización y se explica brevemente la información que de ahí podemos deducir:

- Gráfica de barras de la aceleración: muestreo de la aceleración en cada uno de los ejes ('x', 'y', y 'z') en barras. La información es refrescada cada 51 milisegundos aproximadamente. Sirve para conocer de manera visual, sencilla y rápida el sentido y magnitud de las aceleraciones. La Imagen 26 muestra como se ve esta información:

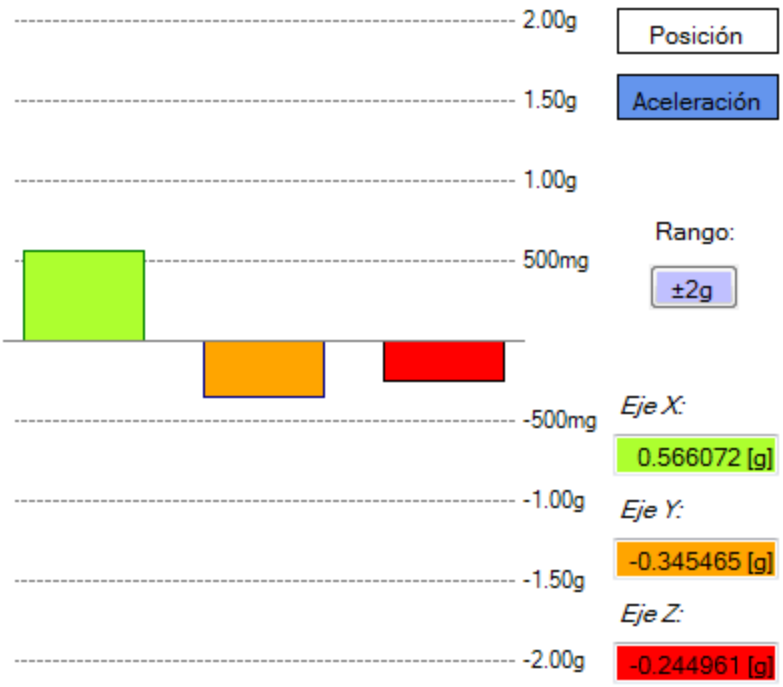


Imagen 26 Gráfica de barras de la aceleración en la aplicación de C#

- Gráfica de posiciones; muestra las posiciones y su cambio histórico en cada uno de los ejes. Se obtiene cada 102 o 867 milisegundos (dependiendo del tipo de integración utilizada).

- Gráfica polar de aceleración: su función es mostrar la magnitud total (R) de la aceleración en un mapa polar, así como los ángulos de elevación (ϕ) y azimuth (θ). Para considera a la coordenada z de la posición, se ha realizado una función exponencial, donde el tamaño del punto aumenta cuando ésta coordenada aumenta de valor. Todo esto es útil, entre otras cosas, para saber la inclinación que tiene el aceleración respecto a la gravedad terrestre (ver Imagen 27):

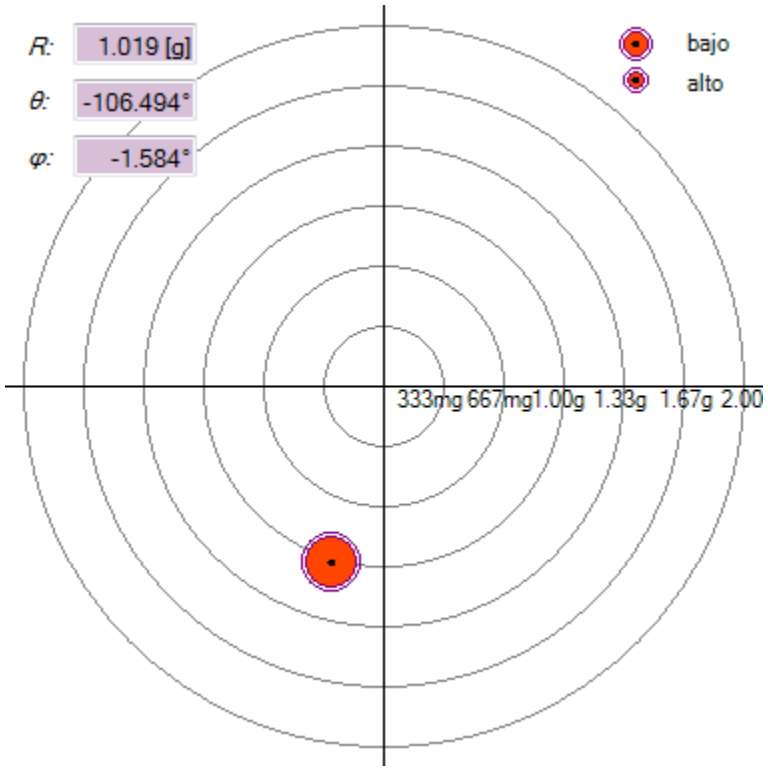


Imagen 27 Gráfica polar de la aceleración en la aplicación de C#

- Gráfica de trayectorias en el plano z = constante. Muestra una perspectiva superior de los posiciones (trayectoria generada) del efector del Scorbot. Igualmente la verificación del espacio de trabajo (anillo formado por los círculos) y el muestreo de las coordenadas XY de cualquier punto (indicadores). La escala aumenta toda posición para cambiar el tamaño de la trayectoria:

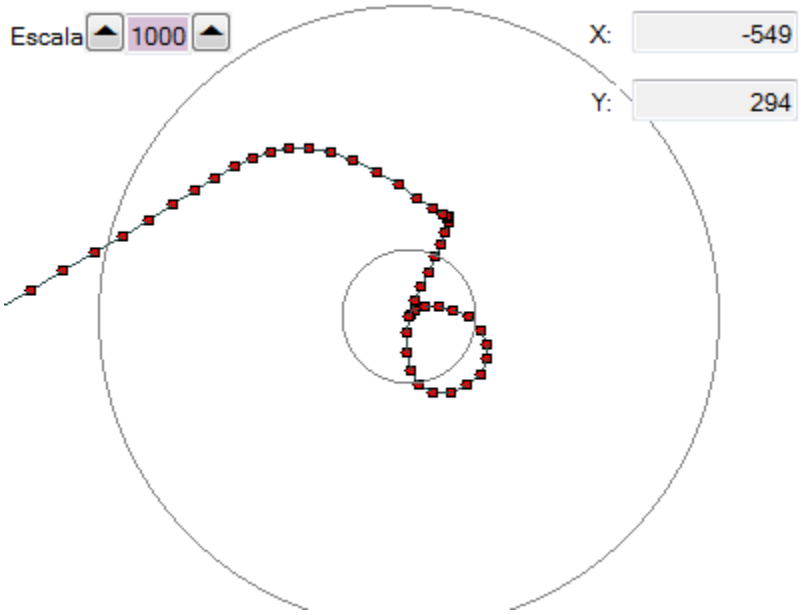


Imagen 28 Gráfica de trayectorias en el plano z = cte. en la aplicación de C#

10.9 Cinemática Inversa

Se han desarrollado dos maneras de resolver la cinemática inversa, cuyos procedimientos de obtención se muestran en una sección más adelante. Estas dos soluciones diferentes otorgan resultados similares, sin embargo la primera, y más completa utiliza las matrices homogéneas para obtener sus resultados, mientras que la segunda, más sencilla, es una solución geométrica. La solución programada por default es la solución geométrica (llamada aquí Cinemática Inversa Simplificada).

10.10 Guardado de trayectorias útiles

Con el fin de poder realizar en caso de necesidad la misma teleoperación, se ha desarrollado un sistema que permite guardar la trayectoria recorrida en un archivo de texto y recuperarla más adelante. Esto permite repetitividad en las trayectorias si es que son útiles para cierto propósito.

De igual manera, con el fin de mantener la integridad y seguridad y de todas las partes y dispositivos, se ha creado un tipo de archivo solamente para este fin (.trx). Con esto nos aseguramos que solamente se importen archivos del tipo .trx, y que no se envíe información por el puerto serial que pueda dañar algo.

10.11 Control Manual/Automático

Existen dos principales formas de funcionamiento, El modo manual y el modo automático.

El modo manual puede ser manipulado con 6 pares de botones, que mueven cada articulación en sentido horario y antihorario. De igual manera puede utilizarse la cinemática inversa para mandar al manipulador manualmente a una posición deseada, o seguir una trayectoria generada a mano (usando incrementos variables en los valores de las coordenadas ‘x’, ‘y’ y ‘z’).

El modo automático utiliza la información del acelerómetro para obtener las posiciones a las que debe llegar el Scorbot, y éste sigue la trayectoria punto a punto sin necesidad de realizar alguna otra acción.

Para ambos modos se tienen botones de paro de emergencia y reset. El paro de emergencia detiene el funcionamiento del brazo controlando la posición en el momento que se presiona el paro de emergencia. Para salir del modo de paro de emergencia es necesario presionar ‘*reset*’.

10.12 Transmisión al microcontrolador Arduino Mega

Esta transmisión define qué tipo de funciones debe de realizar el microcontrolador. Se definió un protocolo de comunicación serial para poder transmitir adecuadamente información. Estas son las características principales de esta comunicación:

- BaudRate:9600
- Parity:None
- DataBits:8
- StopBits:one

Las cadenas enviadas al microcontrolador Arduino Mega pueden ser de varios tipos, dependiendo del modo que se esté utilizando, sin embargo, en el caso en que se quieran enviar trayectorias completas y no solamente un punto, o en su defecto mover manualmente un solo motor, existe además una sincronización entre el control central en la PC y el control sobre el microcontrolador Arduino Mega, donde solamente en caso de que el brazo Scorbot haya llegado a una posición el microcontrolador envía una confirmación por el puerto serie avisando que se encuentra en espera de la siguiente posición de la trayectoria. Con esto se asegura que el manipulador pase por todos los puntos dentro de la trayectoria.

10.13 Interfaz humano – máquina

La interfaz generada en C# permite al usuario controlar el sistema y conocer algunas de sus variables y parámetros. Ésta interfaz fue diseñada de manera que hubiera la menor cantidad de datos presentes en pantalla, para evitar confusiones y una densidad de información elevada. Las funciones gráficas de visualización ocupan la parte central, mientas que los parámetros importantes ocupan un papel secundario en los extremos de la aplicación, provocando una vista limpia de toda la pantalla. A continuación se muestran las partes con las que cuenta, a excepción de los elementos visuales de información, debido a que éstos han sido explicados ya con anterioridad.

10.13.1 Vista general

Se muestra en la Imagen 29 una vista general de toda la aplicación y más adelante se explican parte a parte:

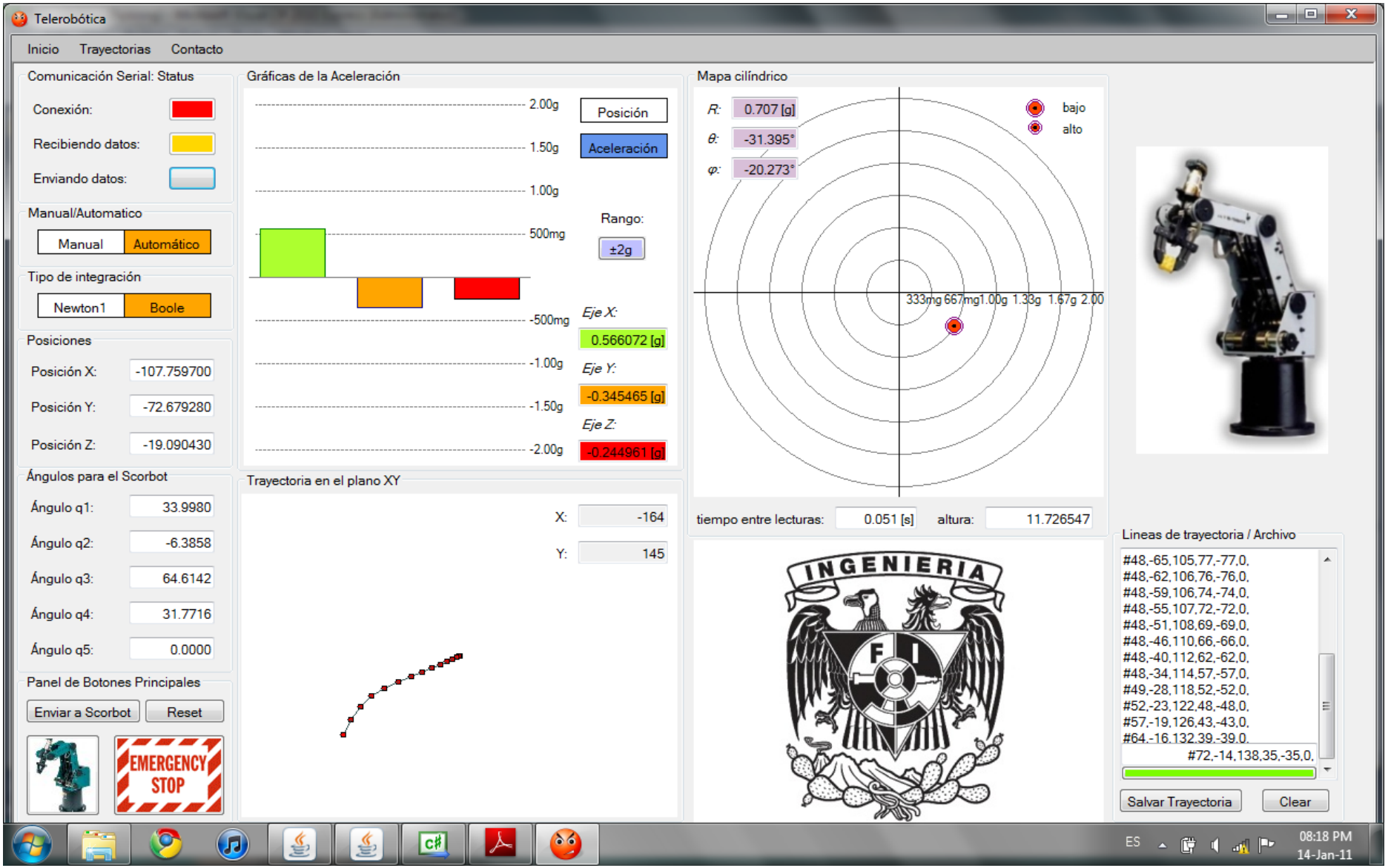


Imagen 29 Vista general de la aplicación en C#

10.13.2 Menú inicial

El menú inicial nos permite abrir y cerrar los puertos serie, recuperar trayectorias de archivos y ver la información de contacto de los desarrolladores del proyecto, como se ve en la Imagen 30:

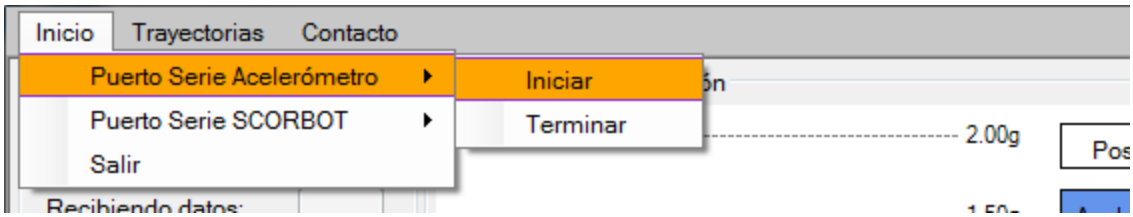


Imagen 30 Menú general de la aplicación en C#

10.13.3 Cuadro de comunicación serial

Un cuadro que muestra el status de la comunicación serial está disponible en la parte superior izquierda de la aplicación, cuenta con tres indicadores. (ver Imagen 31)

- El primero nos muestra si existe una conexión inalámbrica con el XBEE entre la PC y el acelerómetro, un botón cambia su color de fondo a verde en cuanto se detecta una conexión.
- Además de ello existen dos indicadores que muestran la recepción y el envío de información en los respectivos puertos seriales. Éstos indicadores cambian de color cada vez que se envía o recibe información por el puerto serie, viéndose claramente en el funcionamiento normal.

Imagen 31 Indicadores del estado de la comunicación serial en la aplicación de C#

10.13.4 Modo de funcionamiento y tipo de integración

Existen, como se ha mencionado antes, un modo manual y un modo automático; de igual manera hay dos formas de integración, una de las cuales (Newton1) es más veloz que la otra. Para controlar esto sencillamente se presionan los botones para la función que se desee, como se ve en la Imagen 32:

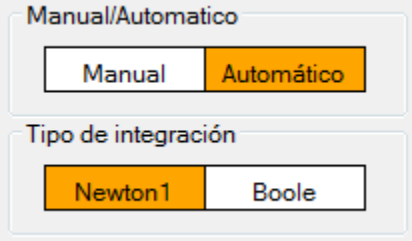


Imagen 32 Modos de funcionamiento de la aplicación de C#

10.13.5 Muestreo de posiciones/aceleraciones y ángulos

Pueden verse éstas variables en la parte izquierda de la aplicación. De ésta manera, como en la Imagen 33, conocemos el estado actual del sistema:

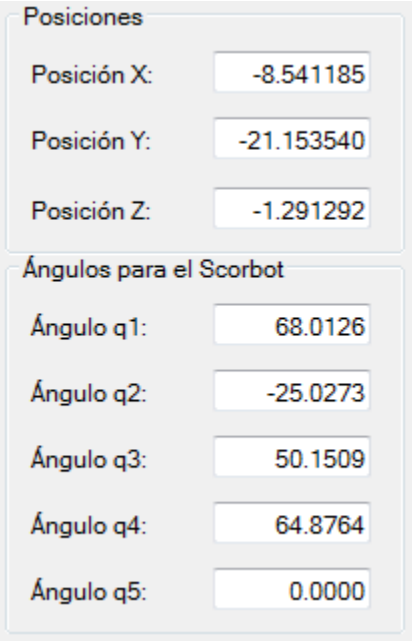


Imagen 33 Muestro de posiciones/aceleraciones y ángulos en la aplicación de C#

10.13.6 Paro de emergencia y otros

El botón de paro de emergencia detiene al robot en la posición en la que se encuentra y realiza el control de posición ahí mismo, para asegurarse que no pueda caerse. El botón de GoHome (inferior izquierda) manda al Scorbot a la posición inicial y los botones de 'Reset' y 'Enviar a Scorbot' resetean la función en el Arduino Mega y envían cadenas individuales al mismo (ver Imagen 34):

Imagen 34 Botones de funcionamiento general de la aplicación en C#

10.13.7 Movimiento manual de articulaciones

Los 12 botones mueven las articulaciones del brazo en sentido horario y anti horario, además se puede ver que se despliegan los datos del número de pulsos de encoder actual en cada articulación en unos cuadros. En la parte superior se muestran cuatro indicadores que se prenden color verde cuando están presionados los microswitches del Scorbot (ver Imagen 35):

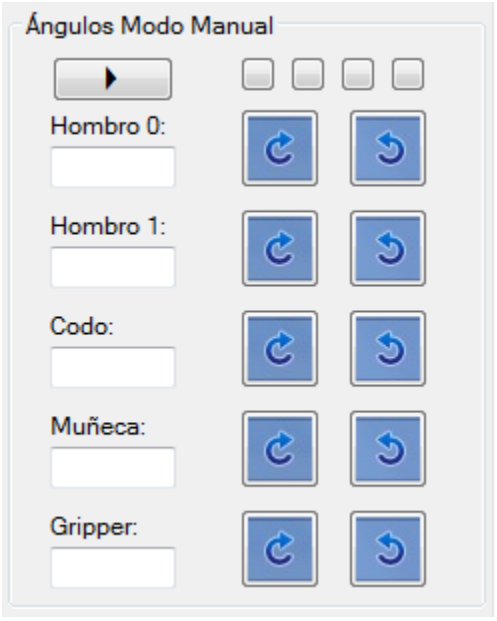


Imagen 35 Movimiento manual de articulaciones en la aplicación en C#

10.13.8 Cinemática Manual

Podemos aumentar o disminuir la posición en cada uno de los ejes y la cinemática inversa se calcula de manera automática al realizar éstos cambios, que además se despliegan en un cuadro de texto contiguo. El control de abajo nos indica los incrementos en cada eje, que puede ser cambiado. (ver Imagen 36)

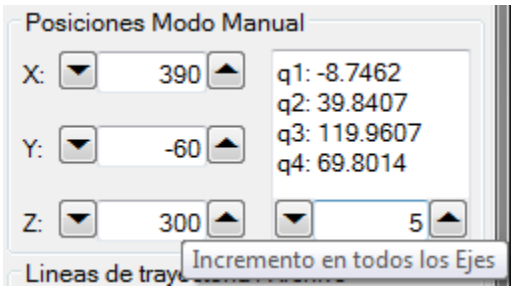


Imagen 36 Cinemática Manual en la aplicación de C#

10.13.9 Cuadro de comandos

El lugar donde se muestran las nuevas líneas o instrucciones para el Scorbot se encuentra en la parte inferior izquierda de la aplicación. Además puede observarse que existe un indicador verde (que muestra si estamos dentro del espacio de trabajo al estar en verde o fuera de él al estar rojo) y tres botones. El primero nos permite guardar la trayectoria en un archivo .trx, el segundo limpia todas las gráficas, valores y comandos del cuadro de texto, mientras que el tercero inicia el envío de todas las líneas de comandos, de la forma explicada anteriormente, al Scorbot. Éste último botón se pone verde mientras esté enviando puntos, y regresa a su estado inicial gris cuando ha terminado toda la trayectoria. Esto se puede constatar en la Imagen 37:

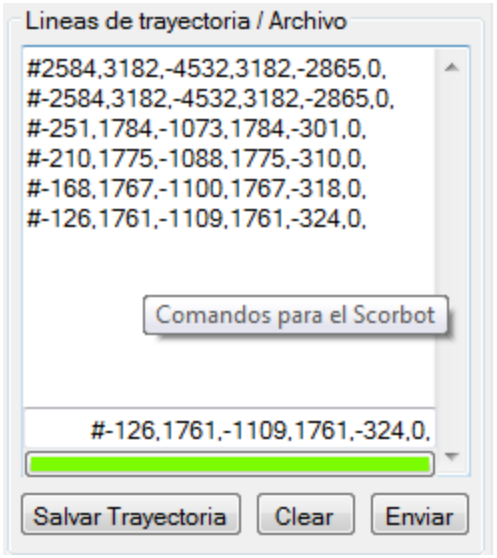


Imagen 37 Cuadro de comandos y trayectorias en la aplicación de C#

10.1 Confort y manejo del sistema

Una tarea importante en teleoperación es que el usuario se sienta cómodo con el sistema y que éste responda de manera intuitiva a las instrucciones que el usuario genera sobre el dispositivo esclavo en el dispositivo maestro. La medición de éste tipo de parámetros es complicada, sin embargo, aunque sea de manera cualitativa, la evaluación de éste confort es un resultado de gran valor. Nos indica si el sistema podrá ser utilizado con facilidad, cuáles son las limitantes, y que competencias debe tener el usuario para manejar el sistema de forma satisfactoria.

Es mediante el uso del sistema de manera rutinaria que se han elegido muchos de los valores estándar que utiliza el sistema, tal es el caso del factor de escalamiento de las trayectorias. El uso de factores pequeños provoca que un movimiento amplio en el dispositivo maestro provoque una trayectoria más reducida de lo que se quisiera, por lo que se le ha dado un valor por default de 1000, teniendo el usuario la capacidad de modificar éste factor en un rango de 1 hasta 2000.

También se ha evaluado la distribución de los diferentes elementos dentro de la pantalla, como botones, indicadores numéricos, indicadores visuales. Éstos son muy cómodos y facilitan el trabajo sobre el sistema. De forma global puede decirse que el gran problema que tiene el sistema es la lentitud con la que se mueve el manipulador, provocando siempre atrasos con lo que el usuario desearía realizar.

11 DISPOSITIVO ESCLAVO

11.1 Manipulador Scorbot

11.1.1 Características principales

Se desea realizar el control óptimo de un brazo robótico de cinco grados de libertad. El brazo robot Scorbot-ER V Plus está construido como brazo vertical articulado, de seis grados de libertad y cuenta con un efector no intercambiable que en este caso es una pinza. Las articulaciones son todas de revolución excepto la pinza cuyo movimiento es prismático (apertura y cierre). A esta configuración se le conoce como configuración antropomórfica, por su semejanza a un brazo humano, como se ve en la Imagen 38.

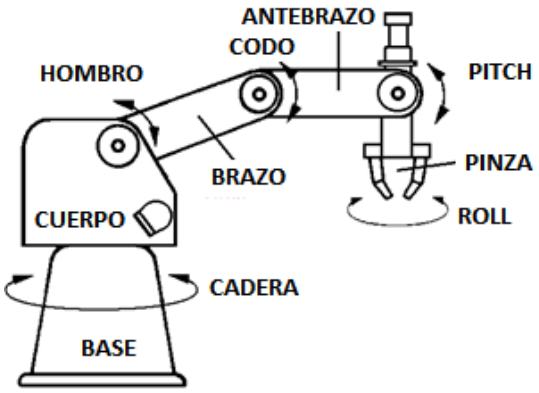


Imagen 38 Movimientos principales del Scorbot

Las articulaciones están accionadas mediante motores, los cuales están acoplados indirectamente; esto es, el motor está montado lejos de las articulaciones y el movimiento del motor se transmite a través de bandas o engranes, lo que ayuda a que el peso de los motores quede sostenido por la base y no por cada una de las articulaciones, de igual forma permite variar la velocidad angular de cada articulación proporcionalmente a la velocidad del motor. Las articulaciones se muestran en la Tabla 9.

# de Eje	Articulación	Movimiento	# de Motor
1	Cintura o Base	Rota el cuerpo	1
2	Hombro	Sube y baja brazo	2
3	Codo	Sube y baja antebrazo	3
4	Pitch	Sube y baja pinza	4+5
5	Roll	Rota pinza	4+5
6	Pinza	Apertura y Cierre	6

Tabla 9 Grados de libertad del Scorbot

11.1.2 Espacio de trabajo

La longitud de cada eslabón y la rotación determina el espacio de trabajo del robot. (Ver Imagen 39)

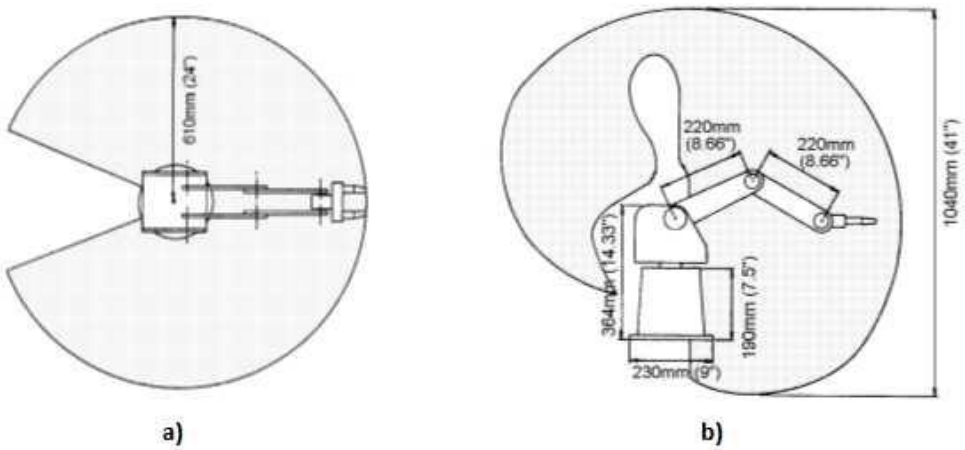


Imagen 39 Espacio de trabajo del Scorbot. Vistas: a) superior b) lateral

11.1.3 Motores del Scorbot

El Motor a controlar es un motor de corriente directa. El cual trabaja a 15 V, y tiene un encoder de cuadratura en su parte posterior con el cual se retroalimenta su funcionamiento y un reductor en la parte frontal, como se puede apreciar en la Imagen 40.

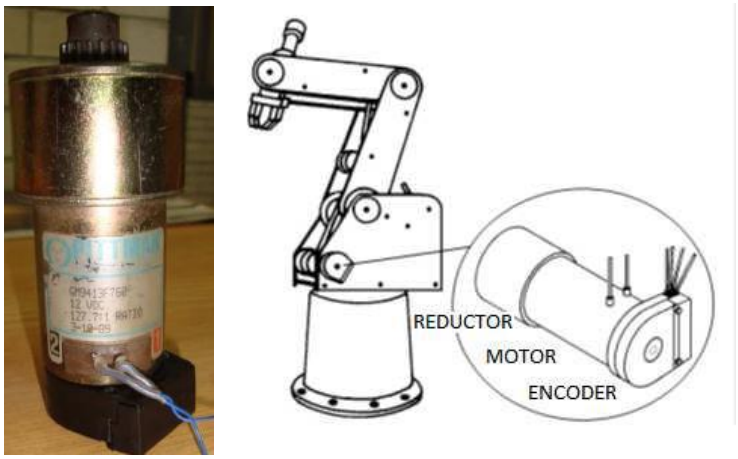


Imagen 40 Motor del Scorbot (izquierda) y ubicación dentro del brazo (derecha)

Cada motor cuenta con diferentes tipos de transmisión, mientras que para la base y el hombro se usa una transmisión de engranajes dentados, para el codo se usan también engranajes dentados y además correas de regulación. La muñeca hace uso de correas de regulación y una unidad diferencial de

engranajes dentados en el extremo del brazo (ver Imagen 41). En la pinza se transmite por medio de un tornillo de avance directamente acoplado al motor. Todo esto se puede ver en la Tabla 10.

Motor	Relación de transmisión (en vueltas)
1,2,3	127.1:1
4,5	65.5:1
6 (Pinza)	19.5:1

Tabla 10 Relaciones de transmisión de las articulaciones del Scorbot

Cada articulación tiene un ángulo de giro limitado, como se ve en la Tabla 11.

Eslabón	Limite [°]
Cadera	310
Hombro	+130/-35
Codo	±130
Pitch	±130
Roll	Sin limite

Tabla 11 Rango de los ángulos de giro de las articulaciones del Scorbot

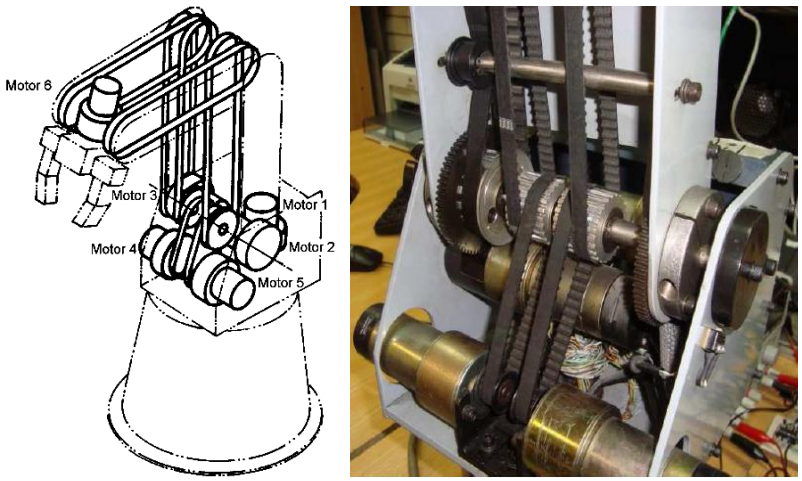


Imagen 41 Vistas de las poleas del Scorbot, conceptual (izquierda) y real (derecha)

Por otro lado la pinza tiene una apertura máxima de 65 [mm].

11.1.4 Encoders del manipulador Scorbot

Los encoders son mecanismos utilizados para entregar la posición, velocidad y aceleración del rotor de un motor. Sus principales aplicaciones incluyen aplicaciones en robótica, lentes fotográficas, aplicaciones industriales que requieren medición angular, militares, etc. Es en realidad un dispositivo electromecánico que convierte la posición angular de un eje, directamente a un código digital. Los tipos más comunes de encoders se clasifican en: absolutos y relativos (conocidos también como incrementales).

Los encoders de cuadratura corresponden a un tipo de encoder incremental que utiliza dos sensores ópticos posicionados con un desplazamiento de ¼ de ranura el uno del otro, generando dos señales de pulsos digitales desfasada en 90°. A estas señales de salida, se les llama comúnmente A y B. Mediante ellas es posible suministrar los datos de posición, velocidad y dirección de rotación del eje. (Ver Imagen 42)

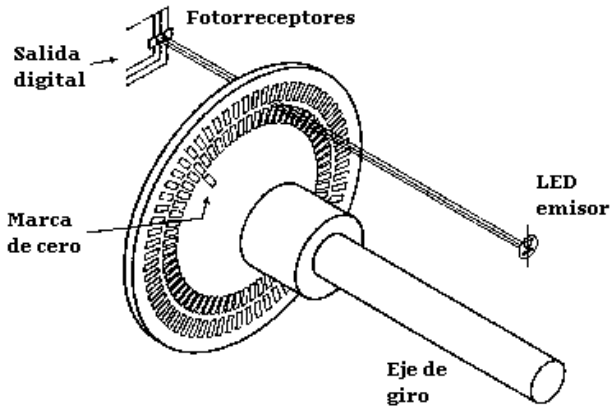


Imagen 42 Encoder incremental

En la figura se muestra un ejemplo de las señales generadas por un encoder girando en sentido horario (vista superior de la Imagen 43) y en sentido antihorario (vista inferior de la Imagen 43).

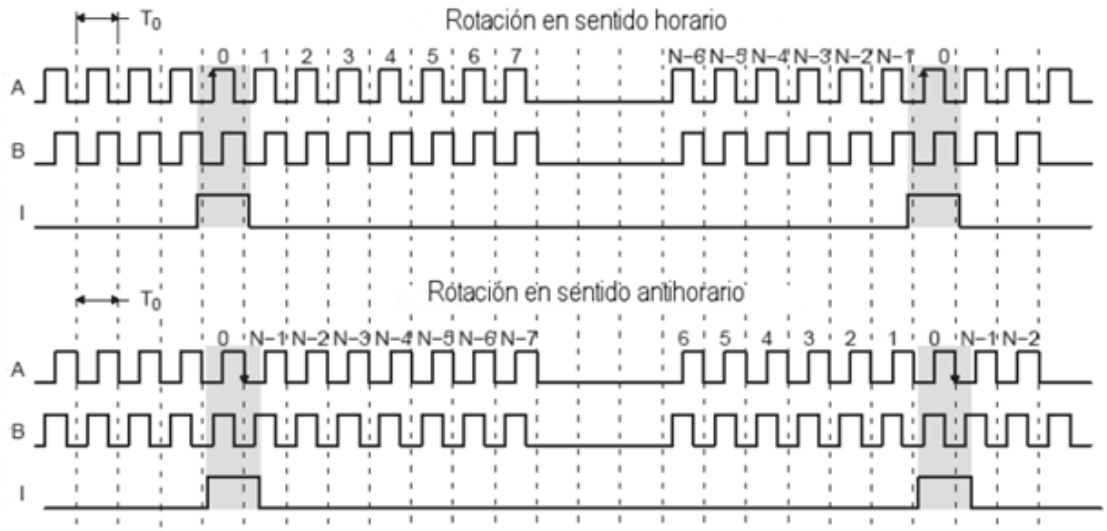


Imagen 43 Señal de un encoder de cuadratura

Los motores del manipulador cuentan con encoders de cuadratura, como se ve en la Imagen 44:

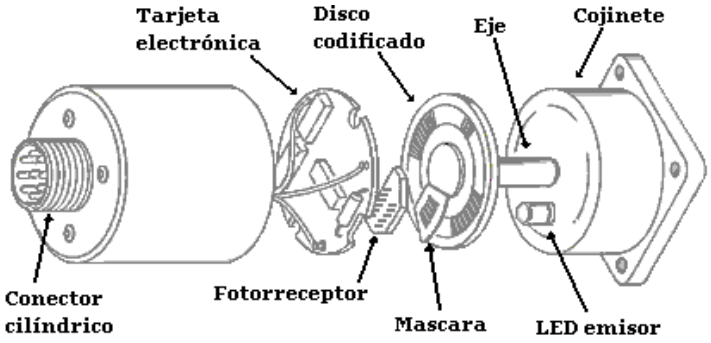


Imagen 44 Encoder dentro de los motores del Scorbot

La lógica implementada para la correcta obtención de las señales de los encoders de cuadratura es resuelta por el microcontrolador en la plataforma Arduino Mega, explicada a detalle más adelante.

11.1.5 Microswitches del manipulador

Hay 5 microswitches encargados del sensado de la posición de carrera de cada una de las articulaciones. Por medio de estos microswitches se determina la posición de Inicio o HOME.

11.2 Cinemática Inversa del manipulador SCORBOT

La cinemática inversa es importante porque transforma las posiciones que se desean encontrar a ángulos. Estos ángulos son los ángulos a las que deben ajustarse las articulaciones del manipulador para conseguir en su efector final la posición deseada, que es la obtenida a partir de la integración. Cabe mencionar que se obtuvieron dos posible soluciones y que ambas se encuentran implementadas en la programación, y que existe la posibilidad de conmutar entre ellas. La primera de las soluciones es la solución formal, utilizando las matrices homogéneas entre sistemas coordenados. La segunda solución (presentada un poco más adelante) es una solución geométrica en base a la trigonometría.

11.2.1 Diagrama del Scorbot

En la Imagen 45 se muestra un diagrama con los grados de libertad, distancias, ángulos y otros datos importantes en el manipulador utilizado:

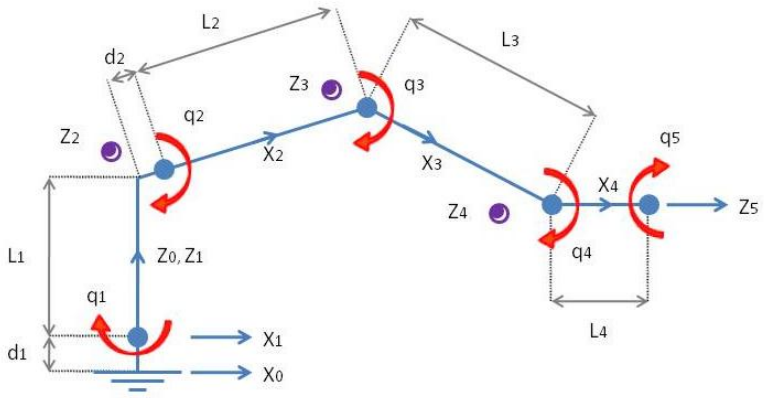


Imagen 45 Diagrama de fuerzas del manipulador

11.2.2 Parámetros de Denavit-Hartenberg

Para poder obtener los ángulos de cada una de las articulaciones a partir de algún punto en el espacio, es necesario obtener las matrices homogéneas de transformación, éstas matrices transforman un sistema de coordenadas en otro. Sin embargo, para construir éstas matrices en cada uno de los sistemas coordenados involucrados en el manipulador, se requiere conocer los parámetros de Denavit-Hartenberg, que se muestran a continuación en la Tabla 12:

i	ψ_i	θ_i	u_i	v_i
1	q_1	0	d_1	0
2	q_2	$-\pi/2$	L_1	d_2
3	q_3	0	0	L_2
4	q_4	0	0	L_3
5	q_5	0	0	L_4

Tabla 12 Parámetros de Denavit-Hartenberg

11.2.3 Matrices homogéneas de transformación de sistemas coordenados

Entonces se usa la siguiente matriz de transformación para cada cambio de sistema coordenado:

$$D_{i,i+1} = \begin{bmatrix} \cos\psi_i & -\sin\psi_i & 0 & v_i \\ \sin\psi_i \cos\theta_i & \cos\psi_i \cos\theta_i & -\sin\theta_i & -u_i \sin\theta_i \\ \sin\psi_i \sin\theta_i & \cos\psi_i \sin\theta_i & \cos\theta_i & u_i \cos\theta_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

Se ha simplificado haciendo el 5º ángulo irrelevante, pues solamente gira al efector final y su movimiento no modifica de ninguna manera los puntos finales alcanzados, además el 4º ángulo se

pretende mantener siempre en la vertical, para realizar las pruebas de corte láser o dibujo. De ahí se obtienen las siguientes matrices de transformación:

$$D_{1,0} = \begin{bmatrix} \cos q_1 & -\sin q_1 & 0 & 0 \\ \sin q_1 & \cos q_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

$$D_{2,1} = \begin{bmatrix} \cos q_2 & -\sin q_2 & 0 & d_2 \\ 0 & 0 & 1 & L_1 \\ -\sin q_2 & -\cos q_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

$$D_{3,2} = \begin{bmatrix} \cos q_3 & -\sin q_3 & 0 & L_2 \\ \sin q_3 & \cos q_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4)$$

$$D_{4,3} = \begin{bmatrix} 1 & 0 & 0 & L_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5)$$

Para obtener una matriz que transforma del sistema coordenado original hasta el del efector final se debe de multiplicar cada una de estas matrices en la siguiente forma:

$$D_{4,0} = D_{1,0} * D_{4,1} \quad (6)$$

$$D_{4,1} = D_{2,1} * D_{4,2} \quad (7)$$

$$D_{4,2} = D_{3,2} * D_{4,3} \quad (8)$$

Y por lo tanto:

$$D_{4,0} = D_{1,0} * D_{2,1} * D_{3,2} * D_{4,3} \quad (9)$$

Se indica a continuación todas las matrices que serán de utilidad para más adelante:

$$D_{4,2} = \begin{bmatrix} \cos q_3 & -\sin q_3 & 0 & L_2 + L_3 \cos q_3 \\ \sin q_3 & \cos q_3 & 0 & L_3 \sin q_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (10)$$

$$D_{4,1} = \begin{bmatrix} \cos(q_2 + q_3) & -\sin(q_2 + q_3) & 0 & d_2 + L_2\cos q_2 + L_3\cos(q_2 + q_3) \\ 0 & 0 & 1 & L_1 \\ -\sin(q_2 + q_3) & -\cos(q_2 + q_3) & 0 & -L_2\sin q_2 - L_3\sin(q_2 + q_3) \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (11)$$

$$D_{4,0} = \begin{bmatrix} \cos q_1\cos(q_2 + q_3) & -\cos q_1\sin(q_2 + q_3) & -\sin q_1 & \cos q_1\,[d_2 + L_2\cos q_2 + L_3\cos(q_2 + q_3)] - L_1\sin q_1 \\ \sin q_1\cos(q_2 + q_3) & -\sin q_1\sin(q_2 + q_3) & \cos q_1 & \sin q_1\,[d_2 + L_2\cos q_2 + L_3\cos(q_2 + q_3)] + L_1\cos q_1 \\ -\sin(q_2 + q_3) & -\cos(q_2 + q_3) & 0 & d_1 - L_2\sin q_2 - L_3\sin(q_2 + q_3) \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (12)$$

11.2.4 Solución de los ángulos

Entonces se obtiene la siguiente fórmula que relaciona los puntos y los ángulos:

$$[P] = D_{4,0} \quad (13)$$

Donde la matriz P contiene las coordenadas P_x, P_y y P_z del Punto P en el espacio, así como todas las rotaciones posibles en ese punto:

$$[P] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & P_x \\ r_{21} & r_{22} & r_{23} & P_y \\ r_{31} & r_{32} & r_{33} & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (14)$$

Para poder resolver algunos de los ángulos se puede retroceder un poco y realizar la siguiente operación:

$$D_{0,1} * [P] = D_{4,1} \quad (15)$$

Para obtener $D_{0,1}$ se usa la siguiente igualdad:

$$D_{0,1} = D_{1,0}^{-1} \quad (16)$$

Se sabe que para cada matriz homogénea de éste tipo, su inversa es:

$$D^{-1} = \begin{bmatrix} R^T & -R^T P_{xyz} \\ 0 & 1 \end{bmatrix} \quad (17)$$

De ésta forma se obtiene:

$$D_{0,1} = \begin{bmatrix} \cos q_1 & \sin q_1 & 0 & 0 \\ -\sin q_1 & \cos q_1 & 0 & 0 \\ 0 & 0 & 1 & -d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{18}$$

Y entonces:

$$\begin{bmatrix} \cos q_1 & \sin q_1 & 0 & 0 \\ -\sin q_1 & \cos q_1 & 0 & 0 \\ 0 & 0 & 1 & -d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} [P] = \begin{bmatrix} \cos(q_2 + q_3) & -\sin(q_2 + q_3) & 0 & d_2 + L_2 \cos q_2 + L_3 \cos(q_2 + q_3) \\ 0 & 0 & 1 & L_1 \\ -\sin(q_2 + q_3) & -\cos(q_2 + q_3) & 0 & -L_2 \sin q_2 - L_3 \sin(q_2 + q_3) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{19}$$

Se toma el elemento (2, 4), que es:

$$L_1 = -P_x \sin q_1 + P_y \cos q_1 \tag{20}$$

A partir de aquí se pueden definir las siguientes variables:

$$P_x = \rho * \cos(\varphi) \tag{21}$$

$$P_y = \rho * \sin(\varphi) \tag{22}$$

$$\rho = \sqrt{P_x^2 + P_y^2} \tag{23}$$

$$\varphi = \tan^{-1} \frac{P_y}{P_x} = Atan2(P_y, P_x) \tag{24}$$

Por lo tanto sustituyendo este valor se obtiene:

$$\frac{L_1}{\varphi} = -\sin q_1 * \cos(\varphi) + \cos q_1 * \sin(\varphi) \tag{25}$$

$$\frac{L_1}{\varphi} = \sin(\varphi - q_1) \tag{26}$$

Se sabe además que:

$$sin^2\alpha + cos^2\alpha = 1 \tag{27}$$

Por lo tanto:

$$\sin(\varphi - q_1) = \pm\sqrt{1 - \frac{L_1^2}{\rho^2}} \tag{28}$$

Entonces finalmente:

$$\tan(\varphi - q_1) = \frac{\sin(\varphi - q_1)}{\cos(\varphi - q_1)} = \frac{L_1}{\pm\sqrt{\rho^2 - L_1^2}} \tag{29}$$

De ahí se obtiene fácilmente el ángulo q_1:

$$q_1 = Atan2(P_y, P_x) - Atan2\left(L_1, \pm\sqrt{P_x^2 + P_y^2 - L_1^2}\right) \tag{30}$$

Ahora e toman los elementos (1, 4) y (3, 4):

$$P_x \cos q_1 + P_y \sin q_1 = d_2 + L_2 \cos q_2 + L_3 \cos(q_2 + q_3) \tag{31}$$

$$P_z - d_1 = -L_2 \sin q_2 - L_3 \sin(q_2 + q_3) \tag{32}$$

Se deja entonces del lado izquierdo de la igualdad las constantes y el ahora conocido ángulo q_1, y se suman los cuadrados de cada ecuación de la siguiente forma:

$$\begin{aligned}\left(P_x \cos q_1 + P_y \sin q_1 - d_2\right)^2 + (d_1 - P_z)^2 \\ = (L_2 \cos q_2 + L_3 \cos(q_2 + q_3))^2 + (L_2 \sin q_2 + L_3 \sin(q_2 + q_3))^2\end{aligned} \tag{33}$$

Y después de reducir los términos, desaparece q_2 y se puede obtener finalmente el ángulo q_3:

$$q_3 = \pm\cos^{-1}\left(\frac{\left(P_x \cos q_1 + P_y \sin q_1 - d_2\right)^2 + (d_1 - P_z)^2 - L_2^2 - L_3^2}{2L_2L_3}\right) \tag{34}$$

Ahora, para obtener el ángulo q_2 se tiene que obtener la matriz $D_{1,2}$ de la misma forma que se había realizado anteriormente, y:

$$D_{1,2}D_{0,1} = \begin{bmatrix} \cos q_1 \cos q_2 & \sin q_1 \cos q_2 & -\sin q_2 & \mathrm{d}_1 \sin q_2 - \mathrm{d}_2 \cos q_2 \\ -\cos q_1 \sin q_2 & -\sin q_1 \sin q_2 & -\cos q_2 & \mathrm{d}_1 \cos q_2 + \mathrm{d}_2 \sin q_2 \\ -\sin q_1 & 0 & 1 & -L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{35}$$

Y entonces la siguiente ecuación:

$$D_{1,2}D_{0,1}[P] = D_{4,2} \tag{36}$$

Queda de la siguiente forma:

$$\begin{bmatrix} \cos q_1 \cos q_2 & \sin q_1 \cos q_2 & -\sin q_2 & \mathrm{d}_1 \sin q_2 - \mathrm{d}_2 \cos q_2 \\ -\cos q_1 \sin q_2 & -\sin q_1 \sin q_2 & -\cos q_2 & \mathrm{d}_1 \cos q_2 + \mathrm{d}_2 \sin q_2 \\ -\sin q_1 & 0 & 1 & -L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} [P] = \begin{bmatrix} \cos q_3 & -\sin q_3 & 0 & L_2 + L_3 \cos q_3 \\ \sin q_3 & \cos q_3 & 0 & L_3 \sin q_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{37}$$

Y se toma el elemento (1, 4):

$$P_x \cos q_1 \cos q_2 + P_y \sin q_1 \cos q_2 - P_z \sin q_2 + \mathrm{d}_1 \sin q_2 - \mathrm{d}_2 \cos q_2 = L_2 + L_3 \cos q_3 \tag{38}$$

Entonces se factoriza:

$$\cos q_2 \left(P_x \cos q_1 + P_y \sin q_1 - \mathrm{d}_2\right) + \sin q_2 \left(d_1 - P_z\right) = L_2 + L_3 \cos q_3 \tag{39}$$

Se puede definir para simplificar:

$$A = P_x \cos q_1 + P_y \sin q_1 - \mathrm{d}_2 \tag{40}$$

$$B = d_1 - P_z \tag{41}$$

$$C = L_2 + L_3 \cos q_3 \tag{42}$$

$$\cos q_2 * A + \sin q_2 * B = C \tag{43}$$

Se realiza una separación de variables de la siguiente forma:

$$\cos q_2 = \frac{1-U^2}{1+U^2}, \sin q_2 = \frac{2U}{1+U^2} \tag{44}$$

Por lo tanto se obtiene la siguiente ecuación:

$$\frac{1-U^2}{1+U^2} * A + \frac{2U}{1+U^2} B = C \tag{45}$$

Si se realiza la factorización de U:

$$(C+A)U^2 - 2B * U + C - A = 0 \tag{46}$$

Si se resuelve la ecuación cuadrática obtenemos:

$$U = \frac{B \pm \sqrt{B^2 + A^2 - C^2}}{C + A} \tag{47}$$

Utilizando la siguiente identidad trigonométrica:

$$\tan\frac{\alpha}{2} = \frac{1-\cos\alpha}{\sin\alpha} \tag{48}$$

Se obtiene finalmente el valor del ángulo q_2:

$$q_2 = 2\tan^{-1}\left(\frac{d_1 - P_z \pm \sqrt{\left(P_x\cos q_1 + P_y \sin q_1 - \mathrm{d}_2\right)^2 + (d_1 - P_z)^2 - (L_2 + L_3\cos q_3)^2}}{P_x\cos q_1 + P_y \sin q_1 - \mathrm{d}_2 + L_2 + L_3\cos q_3}\right) \tag{49}$$

Se quiere que el efector final se encuentre en un ángulo específico con la superficie plana, es necesario asignarle un valor de la siguiente manera:

$$q_4 = \frac{\pi}{2} - q_2 - q_3 \tag{50}$$

De ésta forma se han calculado todos los ángulos relevantes. Existe entonces únicamente la restricción de que el último de los eslabones, que corresponde al efector final se encuentra siempre en una posición vertical. Además es importante considerar que éstos ángulos deben ser transformados a número de pulsos de encoder, que es el valor enviado al dispositivo esclavo, con el que se efectúa el movimiento a tal punto, y su control.

11.2.5 Ángulos finales respecto a los parámetros

Ahora, para finalizar, se debe considerar que la longitud 4 (L_4) no ha sido considerada pero puede ser sumada de manera sencilla al final en los términos en los que participa. De ésta manera se muestra a continuación una tabla con los valores finales de los cuatro ángulos obtenidos a través de éste procedimiento (Se puede ver con claridad que los valores de los ángulos dependen solamente de la posición final a la que se desea llegar y de los parámetros del manipulador u otro ángulo):

q_i	Valor
q_1	$q_1 = Atan2(P_y, P_x) - Atan2\left(L_1, \pm\sqrt{P_x^2 + P_y^2 - L_1^2}\right)$
q_3	$q_3 = \pm\cos^{-1}\left(\frac{\left(P_x \cos q_1 + P_y \sin q_1 - d_2\right)^2 + (d_1 - P_z + L_4)^2 - L_2^2 - L_3^2}{2L_2L_3}\right)$
q_2	$q_2 = 2\tan^{-1}\left(\frac{d_1 - P_z + L_4 \pm \sqrt{\left(P_x \cos q_1 + P_y \sin q_1 - \mathrm{d}_2\right)^2 + (d_1 - P_z + L_4)^2 - (L_2 + L_3 \cos q_3)^2}}{P_x \cos q_1 + P_y \sin q_1 - \mathrm{d}_2 + L_2 + L_3 \cos q_3}\right)$
q_4	$q_4 = \frac{\pi}{2} - q_2 - q_3$

Tabla 13 Ángulos obtenidos por la Cinemática Inversa

11.3 Cinemática Inversa Simplificada

Otra solución, la trigonométrica, se muestra en la Imagen 46:

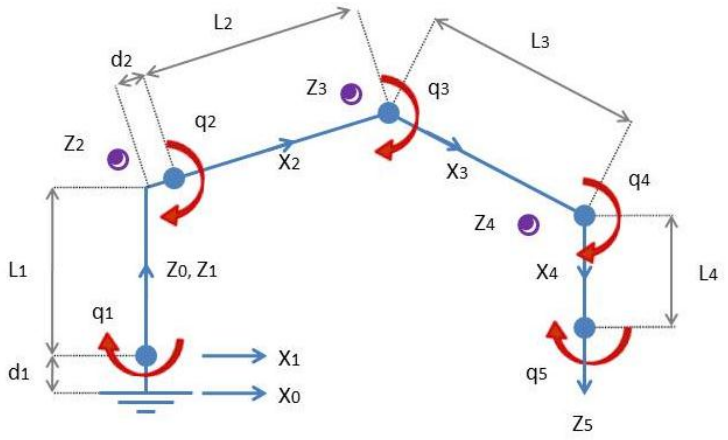


Imagen 46 Diagrama del Scorbot con le articulación final fija verticalmente

11.3.1 Vista Superior

Partimos mostrando una vista superior del Scorbot, Imagen 47:

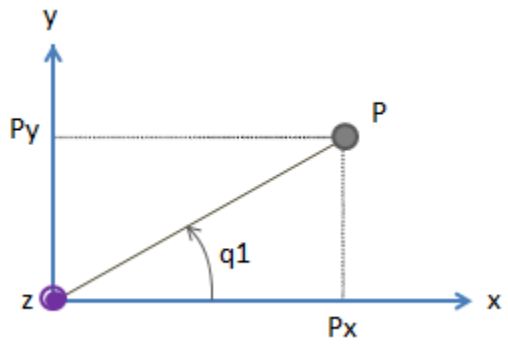


Imagen 47 Vista superior de las articulaciones del Scorbot

Por lo tanto puede deducirse simplemente el ángulo q_1:

$$q_1 = angtan\left(\frac{P_y}{P_x}\right) \tag{1}$$

11.3.2 Vista Lateral

Mostramos un esquema general de la vista en la Imagen 48, con todos los ángulos que participan:

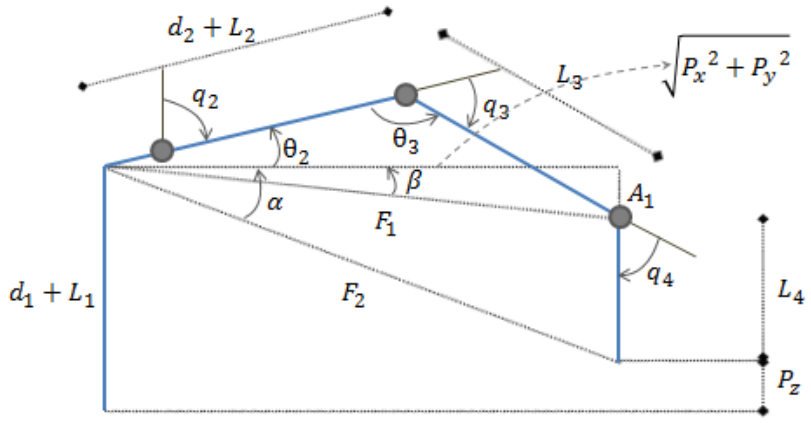


Imagen 48 Vista lateral de las articulaciones del Scorbot

Se puede obtener rápidamente la siguiente igualdad:

$$A_1 = d_1 + L_1 - L_4 - P_z \tag{2}$$

De igual manera se obtiene F_1 de una forma sencilla.

$$F_1 = \sqrt{{A_1}^2 + {P_x}^2 + {P_y}^2} = \sqrt{(d_1 + L_1 - L_4 - P_z)^2 + {P_x}^2 + {P_y}^2} \tag{3}$$

También se puede deducir $\tan(\beta)$:

$$\tan(\beta) = \frac{A_1}{\sqrt{{P_x}^2 + {P_y}^2}} = \frac{d_1 + L_1 - L_4 - P_z}{\sqrt{{P_x}^2 + {P_y}^2}} \tag{4}$$

Y por lo tanto e obtiene β:

$$\beta = angtan\left(\frac{d_1 + L_1 - L_4 - P_z}{\sqrt{{P_x}^2 + {P_y}^2}}\right) \tag{5}$$

Se puede obtener el ángulo $\theta_2$utilizando la ley de cosenos en el triángulo que tiene el ángulo $(\theta_2 + \beta)$:

$${L_3}^2 = (d_2 + L_2)^2 + {F_1}^2 - 2F_1(d_2 + L_2)\cos(\theta_2 + \beta) \tag{6}$$

Y despejando se obtiene θ_2:

$$\theta_2 = angcos\left(\frac{(d_2 + L_2)^2 + {F_1}^2 - {L_3}^2}{2F_1(d_2 + L_2)}\right) - \beta \tag{7}$$

Para ahora obtener el ángulo θ_3 se utiliza de nuevo la ley de cosenos en el mismo triángulo para el ángulo de interés:

$${F_1}^2 = {L_3}^2 + (d_2 + L_2)^2 - 2L_3(d_2 + L_2)\cos(\theta_3) \tag{8}$$

Y se obtiene finalmente el ángulo θ_3:

$$\theta_3 = angcos\left(\frac{{L_3}^2 + (d_2 + L_2)^2 - {F_1}^2}{2L_3(d_2 + L_2)}\right) \tag{9}$$

Obtener el ángulo q_4 es trivial si se reconocen las siguientes igualdades:

$$q_2 + q_3 + q_4 = \pi \tag{10}$$

$$\theta_2 - q_2 = {}^{\pi}/_{2} \tag{11}$$

$$\theta_3 - q_3 = \pi \tag{12}$$

$$q_4 = {}^{\pi}/_{2} - \theta_2 - \theta_3 \tag{13}$$

11.3.3 Ángulos finales obtenidos a partir de la Cinemática Inversa Simplificada

Y de ésta manera se obtienen todos los parámetros necesarios para resolver la cinemática inversa, que se muestran en la Tabla 14:

q_i	Valor
q_1	$q_1 = angtan\left(\frac{P_y}{P_x}\right)$
θ_2	$\theta_2 = angcos\left(\frac{(d_2 + L_2)^2 + {F_1}^2 - {L_3}^2}{2F_1(d_2 + L_2)}\right) - \beta$
θ_3	$\theta_3 = angcos\left(\frac{{L_3}^2 + (d_2 + L_2)^2 - {F_1}^2}{2L_3(d_2 + L_2)}\right)$
q_4	$q_4 = {}^{\pi}/_{2} - \theta_2 - \theta_3$

Tabla 14 Tabla de ángulos obtenidos por la Cinemática Inversa Simplificada

11.3.4 Transformación a pulsos de encoder

Los ángulos obtenidos en cualquiera de los casos de cinemática inversa deben ser convertidos a valores de pulsos de encoder. Éstos valores estará referenciados a la posición de Home, donde el valor de los pulsos de encoder es conocido, y donde también se conocen los ángulos de las articulaciones. De ésta forma, puede encontrarse una sencilla fórmula que transforme los ángulos a pulsos de encoder tomando en cuenta esta posición inicial (cabe mencionar que el sentido donde el número de pulsos de encoder aumenta corresponde al sentido donde los ángulos aumentan):

$$n_i = r_i\,(\theta_i - \theta_0)$$

Donde el subíndice 'i' es el número de la articulación, 'o' es la posición inicial y r es la razón de pulsos de encoder a ángulos de la articulación i. Entonces, para cada una de las articulaciones se ha utilizado el ángulo inicial que se muestra en la Tabla 15.

Articulación	Ángulo inicial θ_0 [°]
Base	0.0
Hombro	105.5
Codo	100.0
Muñeca	53.0

Tabla 15 Ángulo inicial de las articulaciones del manipulador

12 IMPLEMENTACIÓN SOBRE EL DISPOSITIVO ESCLAVO

Como antes descrito en la sección anterior, el brazo robótico cuenta con motores de CD, Encoders y Microswitches. Este conjunto de elementos nos permiten manipular correctamente a todas las articulaciones del brazo y conocer su respectiva posición en el espacio. La arquitectura diseñada para el control del dispositivo esclavo fue definida en base a estos elementos.

Se puede diferenciar a estos elementos de la siguiente forma:

- Actuadores: Motores
- Sensores: Encoders de Cuadratura, Microswitches
- Lógicos: Microcontrolador Arduino Mega
- Intermediarios: Etapa de Potencia, Resistencias, cableado, etc.

Se utiliza un Microcontrolador ATmega1280, que se encuentra integrado en la plataforma Arduino, para generar la lógica del funcionamiento del brazo. Generando así las señales de control de los actuadores y procesando la información proveniente de los sensores.Para conjuntar al microcontrolador con los motores es necesaria la existencia de una etapa de potencia, la cual se encarga de alimentar correctamente los motores y aislar la alimentación de los mismos del microcontrolador.

El brazo Scorbot cuenta con un cableado de todos sus elementos (Motores, Encoders, Microswitches) integrado en un conector de tipo DB50 con terminación hembra. Por lo que todas las conexiones provenientes del microcontrolador hacia la etapa de potencia y señales de sensado fueron integradas en un conector de tipo DB50 con terminación macho.

12.1 Etapa de potencia

Como se ha mencionado, para poder utilizar adecuadamente el microcontrolador en conjunto con los motores es necesario tener una etapa de potencia. Esta etapa aísla la corriente y el voltaje, que consume el motor, de la corriente y voltaje que consume el microcontrolador. Si no existiera esta etapa, el microcontrolador podría ser severamente dañado por los efectos del funcionamiento normal de los motores

La etapa de potencia diseñada fue realizada utilizando tres circuitos integrados que desempeñan la función de puentes 'H'. Estos circuitos son el L293D, el cual es controlado por señales de 5V y permite voltajes de salida de hasta 36V con una corriente máxima de 1A, por salida. En este tipo de circuito integrado se cuenta con la posibilidad de controlar 2 motores por CI. El brazo Scorbot cuenta con 6 motores, por lo que se utilizan tres CI. Estos circuitos serán controlados por medio de señales de tipo PWM para poder regular el Voltaje en de alimentación de cada motor.

Son necesarias 3 señales de control por cada motor:

- Una señal de salida analógica, para poder generar señales de tipo PWM, para controlar el pin *Eneable* del puente H.
- Dos señales de control para generar los sentidos de giro de cada motor.

12.1.1 Prototipo de la etapa de potencia

Se muestra una imagen real de los componentes que forman parte de la etapa de potencia, rodeado de los demás subsistemas de control del dispositivo esclavo (ver Imagen 49):

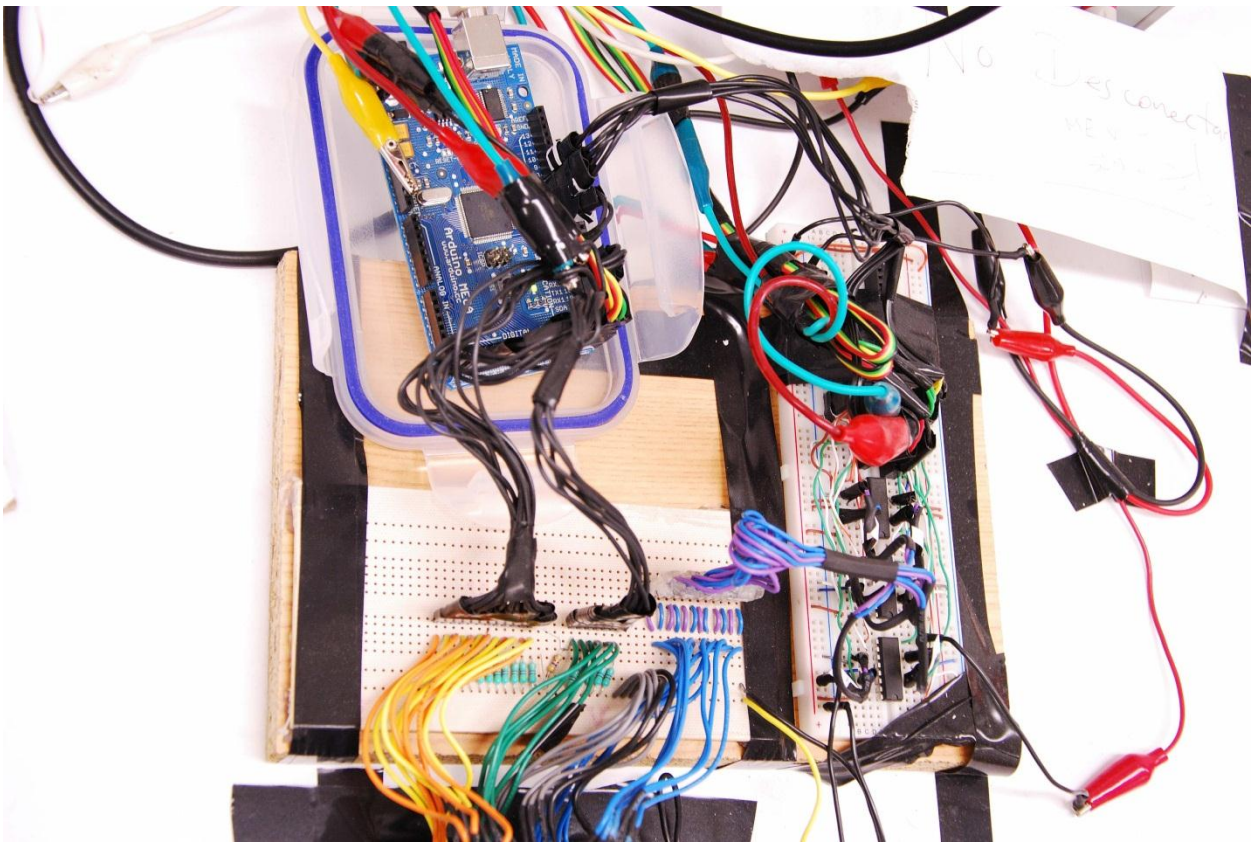


Imagen 49 Vista real del control del dispositivo esclavo

12.2 Arduino Mega

El Arduino Mega es un microcontrolador basado en el ATmega1280. Tiene 54 entradas o salidas digitales, de las cuales 14 pueden ser usadas como PWM, 16 entradas analógicas, 4 puertos seriales, un cristal de 16 MHz y una conexión USB básicamente. Características:

- Voltaje de operación de 5V
- Voltajes de entrada recomendados de 7 a 12V
- Voltajes máximos de entrada 6 a 20V
- Corriente por los pines de entrada o salida de 40mA
- Memoria flash de 128kB
- Memoria SRAM de 8kB
- Memoria EEPROM de 4kB

Además de ello posee muchas cualidades similares en su lenguaje de programación a la programación en C, y por ello facilita mucho su implementación. (Ver Imagen 50)

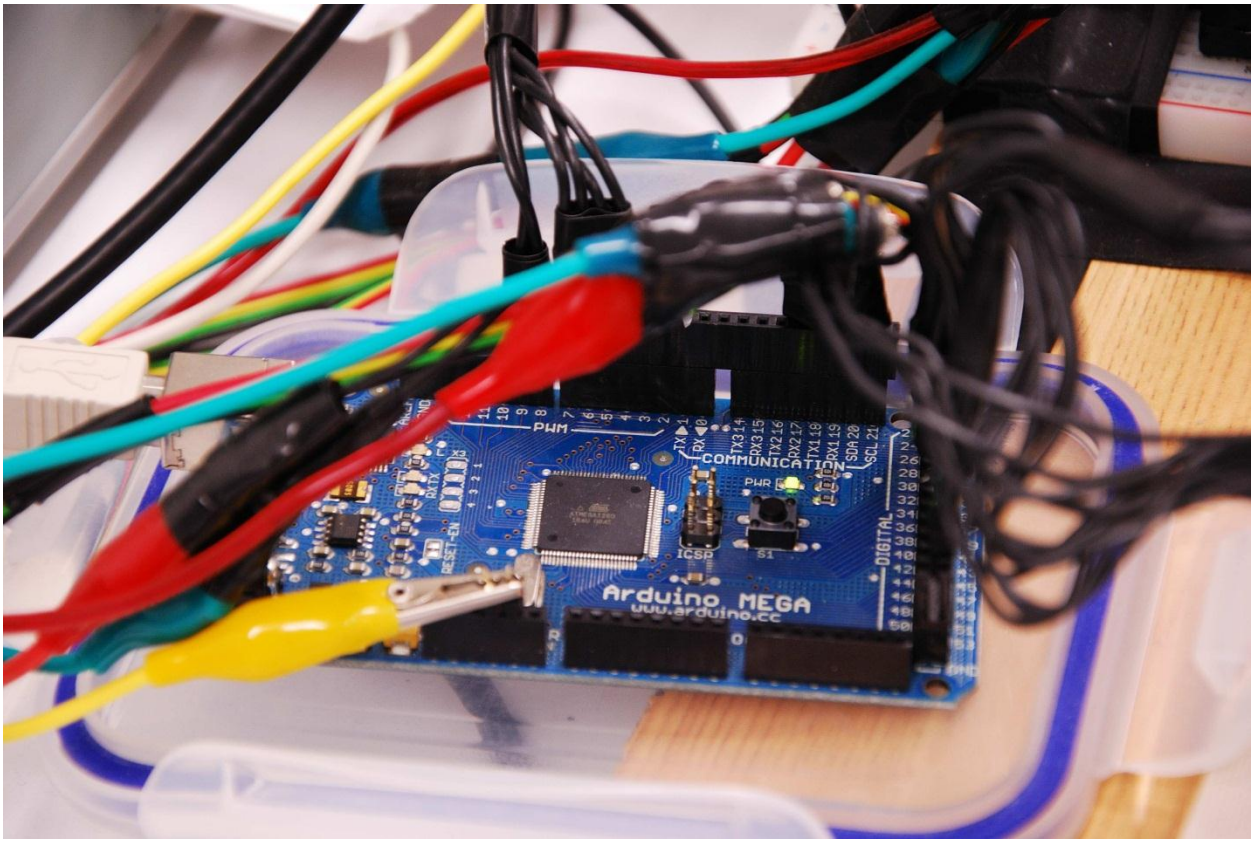


Imagen 50 Microcontrolador Arduino Mega en el entorno real

El microcontrolador se encarga de procesa todas las señales de entrada y salida. Este procesamiento se puede definir como las tareas del microcontrolador, estas de dividen de la siguiente forma:

- Procesar las señales de entrada de los microswitches
- Procesar las señales de entrada de los encoders de cuadratura
- Procesar la información serial recibida de la CPU
- Generar señales variables de control para los motores
- Enviado de información serial a la CPU
- Implementación de otras funciones

12.3 Procesamiento de las señales de entrada de los microswitches

El procesamiento de las señales de entrada de los microswitches se realiza a través de un comparador digital en donde se sensa el estado de la señal del microswitch.

La principal función de los microswitches es poder conocer la configuración inicial de las articulaciones del brazo. Y poder utilizar esta configuración como punto de partida para el movimiento de cada articulación controlada por medio de los encoders de cuadratura. El brazo cuenta con 5 microswitches para poder detectar la posición inicial de cada articulación la posición dada cuando todas las articulaciones se encuentran presionando a todos los microswitches se le llama posición de "HOME".

La lógica de las señales de los microswitches es la siguiente:

- Si el microswitch se encuentra liberado se deberán ver 5V o un lógico HIGH, en su terminal correspondiente.
- Si el microswitch se encuentra presionado se deberán ver 0V o un lógico LOW, en su terminal correspondiente.

Esta lógica evita que, por falta de alimentación a los microswitches se tenga una incorrecta medición ya que normalmente estos ms se encuentran abiertos cuando el brazo esta en operación.

12.4 Procesamiento de las señales de entrada de los encoders de cuadratura

El procesamiento de las señales de entrada de los encoders de cuadratura se realiza a través de interrupciones externas en el microcontrolador. La lógica de los encoders de cuadratura es la siguiente:

- Si el sensor IR se encuentra libre se deberán ver 5V o un lógico HIGH, en su terminal.
- Si el sensor IR se encuentra bloqueado se deberán ver 0V o un lógico LOW, en su terminal.

El procesamiento es realizado de la siguiente forma:

1. Un encoder de cuadratura parte de 2 señales que se encuentran desfasadas por 90° en su frecuencia, como explicado en la sección 10.1.1. En base a este principio, las señales son procesadas en pares. Una de las señales del encoder es conectada directamente a un pin con interrupciones externas y el otro a una entrada digital normal.
2. Cuando existe un cambio en el estado del pin, de la interrupción externa, se llama a un método en específico dentro de la programación del microcontrolador.
3. Esta función lee el estado en el encoder pareja al que se encuentra en la interrupción y determina en qué sentido gira el motor y va sumando o restando un pulso a un contador que relaciona a un motor.

12.5 Control PID de posición (Regulación de voltaje a los motores)

12.5.1 Control

La teoría de control ha permitido que la tecnología pueda desarrollarse, pues es la que trata el comportamiento de los sistemas dinámicos y la que permite que las máquinas funcionen como se espera. Muchos sistemas de control están basados en el principio de retroalimentación, donde la señal que se desea controlar es comparada con una señal de referencia y la discrepancia entre estas señales es la que provoca una acción correctiva. Cuando una o varias variables de un sistema necesitan seguir cierta referencia en base al tiempo, un "controlador" manipula las entradas al sistema de manera que se obtenga la salida deseada del sistema (ver Imagen 51 e Imagen 52):

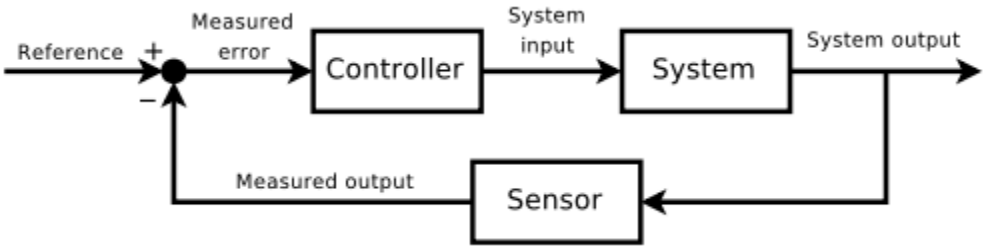


Imagen 51 Concepto de control retroalimentado

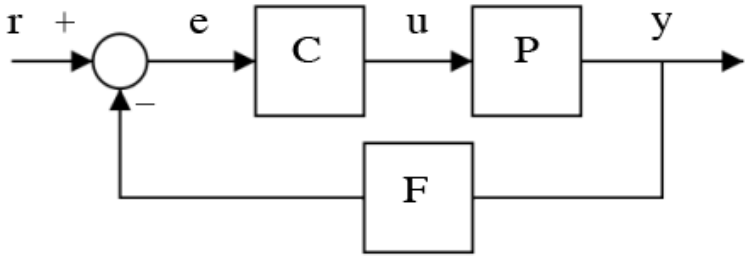


Imagen 52 Funciones de transferencia del control retroalimentado

Algunos parámetros importantes a la hora de diseñar un control:

- Sistemas abiertos o sistemas cerrados
- Función de transferencia
- Control PID, control difuso (*Fuzzy*)
- Variables de estado
- Estabilidad
- Sistemas no lineales
- Controlabilidad y observabilidad
- Robustez
- Control adaptativo, jerárquico, inteligente, óptimo, robusto, estocástico
- Redes neuronales y algoritmos genéticos (6)

El control PID es probablemente el control con retroalimentación más utilizado. Es un acrónimo para control *Proporcional-integral-derivativo*, refiriéndose a los tres términos con los que cuenta. Tiene la forma:

$$u(t) = K_p e(t) + K_I \int e(t)dt + K_D \frac{d}{dt} e(t) \quad \text{(A)}$$

La dinámica particular del sistema se obtiene manipulando los parámetros constantes K_x. Con este tipo de controlador se conjuntan tres diferentes acciones dependiendo de tres diferentes parámetros. Estos parámetros son, en la ecuación antes vista, la parte proporcional, la parte integral y la parte derivativa.

Estos parámetros dependen de un factor común. Este es el error. El error se define como la diferencia que existe entre un valor deseado menos el valor real. En este trabajo el error es la diferencia entre la posición real que tiene una articulación, medida a través de los pulsos de encoder, menos una posición deseada, mediada también a través de pulsos de encoder.

Los efectos que cada parte del PID se describen como sigue:

- Parte Proporcional: Esta tendrá una respuesta proporcional al error. Esto es, si el error es pequeño, o sea la articulación en cuestión se encuentra cerca de la posición final deseada, la respuesta del sistema generará una pequeña acción para corregir este error. En cambio si el error es grande, se tomara una acción de mayor tamaño, para poder alcanzar al punto rápidamente.
- Parte Integral: Esta tendrá un efecto en base a la acumulación de los errores pasados. Esto es, en un intervalo de tiempo determinado se consideran los resultados de los errores pasados y en base a estos se tomará una acción. Mientras la acumulación del error sea más grande cada vez, el controlador tendrá un efecto más grande para así poder corregir este error pasado. Este tipo de controlador se dice que ve el pasado para corregir su error,
- Parte Derivativa: Esta tendrá un efecto en base a la tendencia del error. Esto es, en un intervalo de tiempo determinado se analiza cual es la tendencia de crecimiento o decrecimiento del error, en base a esta tendencia se tomara una determinada acción. Si esta tendencia es grande, el controlador intentara atenuarla conforme se acerque a la posición deseada. Este tipo de controlador se dice que ve el futuro para corregir el error.

Como se ve en la ecuación (A) para poder implementar un controlador PID se deben de calcular integrales y diferenciales. Para poder realizar estas funciones en el microcontrolador se debe de restringir un intervalo de tiempo conocido, para poder resolver el problema a través de una discretización. Por lo que la parte integral y diferencial se pueden definir como una sumatoria finita con intervalos de tiempo constantes.

El algoritmo de programación utilizado para este controlador toma la lógica del siguiente controlador:

```
previous_error = 0
integral = 0
start:
  error = setpoint - actual_position
  integral = integral + (error*dt)
  derivative = (error - previous_error)/dt
  output = (Kp*error) + (Ki*integral) + (Kd*derivative)
  previous_error = error
  wait(dt)
  goto start
```

Por lo que la salida del sistema (output) se convierte en el voltaje que se debe aplicar a un determinado motor para poder alcanzar la posición deseada en esa articulación. Esta señal de salida se maneja como una señal variable, PWM que regula el pin eneable de un l293D.

12.6 Procesamiento de la información recibida del CPU

El procesamiento de la información recibida de la CPU se realiza en base al siguiente protocolo de comunicación serial desarrollado para este trabajo de tesis. Este protocolo de comunicación se definió con los siguientes parámetros: 9600 Baudios sin paridad, 8 bits con un bit de parada. Con ésta configuración se definió que con el envío de 1byte se podrán accesar a diferentes métodos los cuales pueden recibir más bytes dependiendo del método que se accede.

- Recepción por Arduino
- Paro de emergencia
- Reset
- Modo Manual
- Modo Automático
- Recepción de datos de posición deseada
- Manipulación Manual de Cada Articulación
- Envío de datos Arduino
- Posición Deseada alcanzada
- Información de Sensores (Microswitches y Pulsos de Encoders)
- Otra Información, Se pueden imprimir otras variables, parámetros o mensajes. Si no son específicamente manejados por la aplicación en la CPU, estos no se mostrarán en la misma.

12.7 Descripción de las funciones programadas al Microcontrolador

- Main: Esta función se encarga de la recepción de comunicación serial. (Ver Diagrama 12)

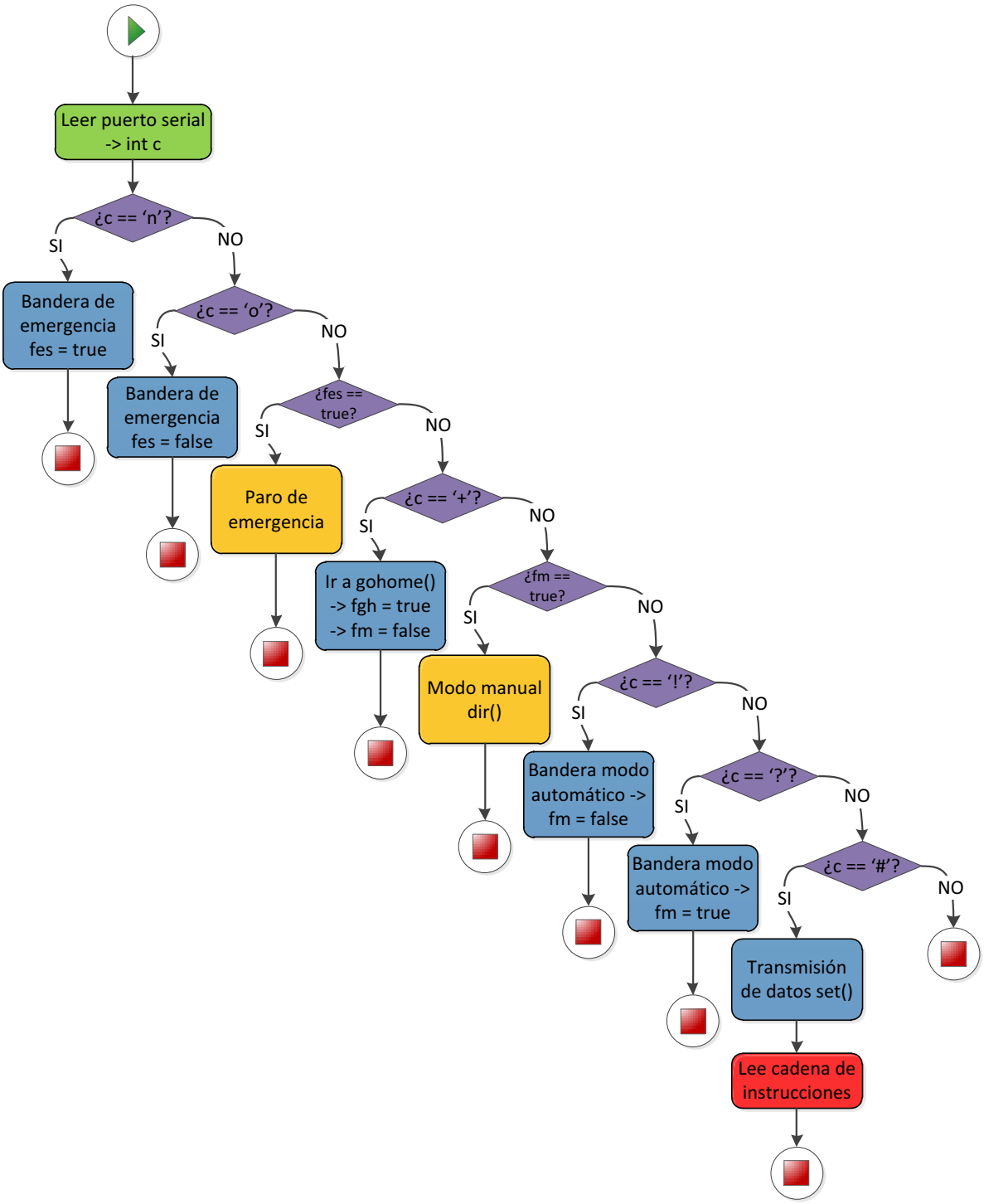


Diagrama 12 Recepción de datos de la comunicación serial en el ATmega1280

- Enc: Este método es utilizado para cada motor. En este método se determina en qué sentido está girando cada motor y dependiendo del sentido de giro incrementa o decrementa en 1 una variable determinada. Los motores cuentan con encoders de cuadratura, en base al funcionamiento de los encoders de cuadratura explicado anteriormente, la lógica que se utiliza es la siguiente. Una de las dos señales del encoder de cuadratura se mide desde un pin con interrupciones externas. La interrupción externa utilizada es cuando existe un cambio en el estado, de dicha entrada, de una medición lógica HIGH a una medición lógica LOW Cuando ocurre este cambio de estado se prosigue a medir el estado de la señal "pareja" de este motor. Dependiendo de cada motor se determina en qué sentido gira, en el sentido de las manecillas del reloj o en contra de las manecillas del reloj. Al determinar en qué sentido gira cada articulación se incrementa o decrementa un contador. Con este contador podemos relacionar el incremento de pulsos de encoders con el incremento en ángulos de cada articulación. La lógica general se puede ver en el Diagrama 13.

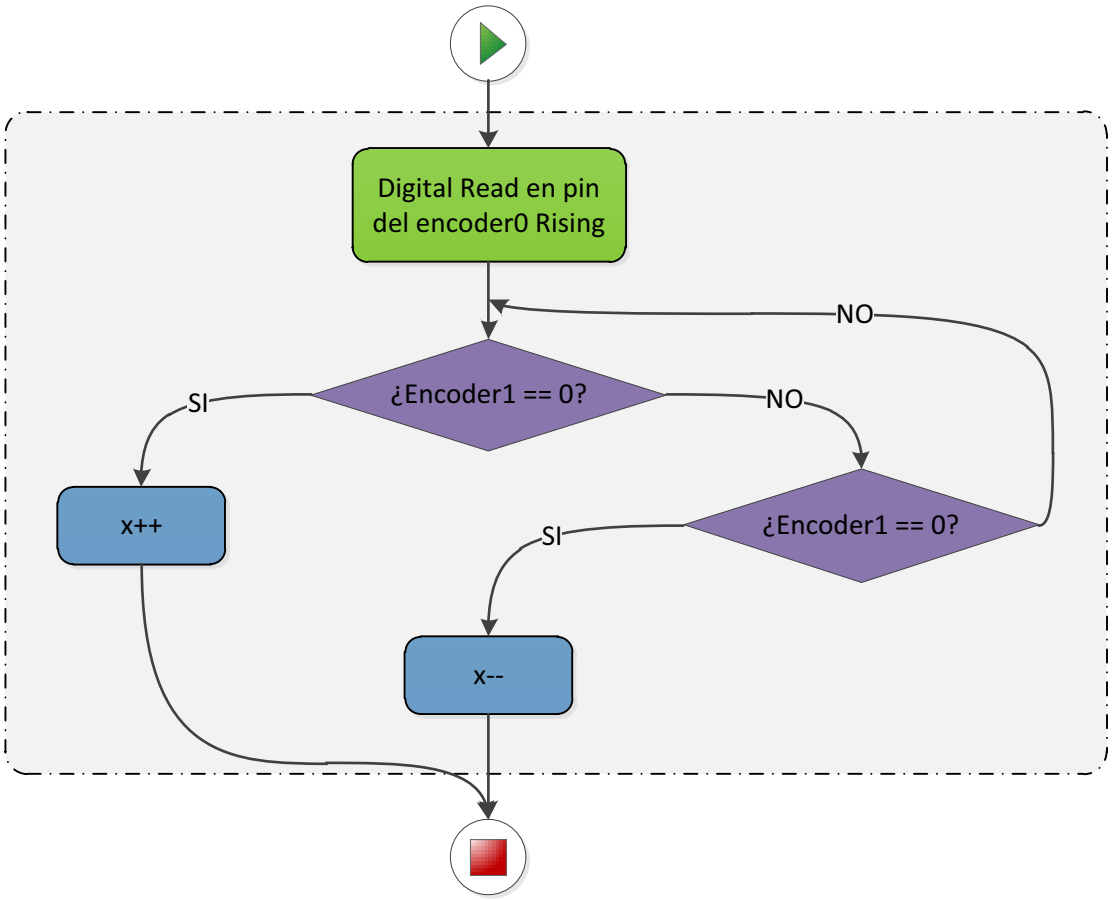


Diagrama 13 Lectura de encoders del Scorbot en el microcontrolador ATmega1280

- Gohome: En esta función se realiza el posicionamiento de cada articulación en su posición inicial. Cuando cada articulación se encuentra en la posición inicial se dice que el robot se encuentra en HOME. Esta posición es crucial para el funcionamiento autónomo del robot pues establece el marco de referencia del movimiento del robot. Esta posición, establece un lugar físico conocido con el cual se calculan todas las posiciones siguientes. La determinación de la posición de Home se realiza con la ayuda de cinco microswitches, los cuales son presionados en el momento que una articulación dada pase por el área de sensado del mismo. La posición de Home es alcanzada por cada articulación, una a la vez. Empezando por el hombro y continuando por cada articulación hasta llegar al efector final o la apertura y cierre del gripper. Para cada articulación se enciende el motor correspondiente en un sentido de giro. El motor se detendrá hasta que encuentre el microswitch. Si el movimiento de la articulación es tal, que no pasa por el microswitch y llega a un tope físico. Este se detecta y esa articulación se mueve en el sentido opuesto hasta encontrar el microswitch. Esta lógica se aplica a cada una de las articulaciones hasta que se alcance la posición de HOME.

- Isr: En esta función se determina el control de la posición deseada del brazo. En base a un controlador PID se determina con que velocidad y en qué sentido de giro una articulación dada debe de moverse para poder alcanzar una posición deseada. Como se explicó anterior mente un Controlador PID utiliza una parte Proporcional, una parte Integral y una parte Diferencial para determinar la velocidad que se requiere utilizar en una determinada articulación para alcanzar una posición deseada. La parte proporcional se define como el error multiplicado por una constate. Este error es igual a la diferencia que existe entre la posición actual menos la posición deseada. La parte integral es la integral de los errores en un intervalo conocido multiplicado por una constante. La parte integral es la diferencial de los errores en un intervalo conocido multiplicado por una constante. Al integrar cada uno de estos parámetros se obtiene un valor, con el cual se determina el sentido de giro que debe tener una articulación y la velocidad con la cual debe de moverse. Una vez que todas las articulaciones se encuentren en un rango cercano al punto deseado se envía al CPU un mensaje indicando que ha llegado al punto deseado y que un nuevo punto puede ser determinado. Por lo que el CPU intentara enviar una nueva posición deseada a alcanzar. (Ver Diagrama 14)

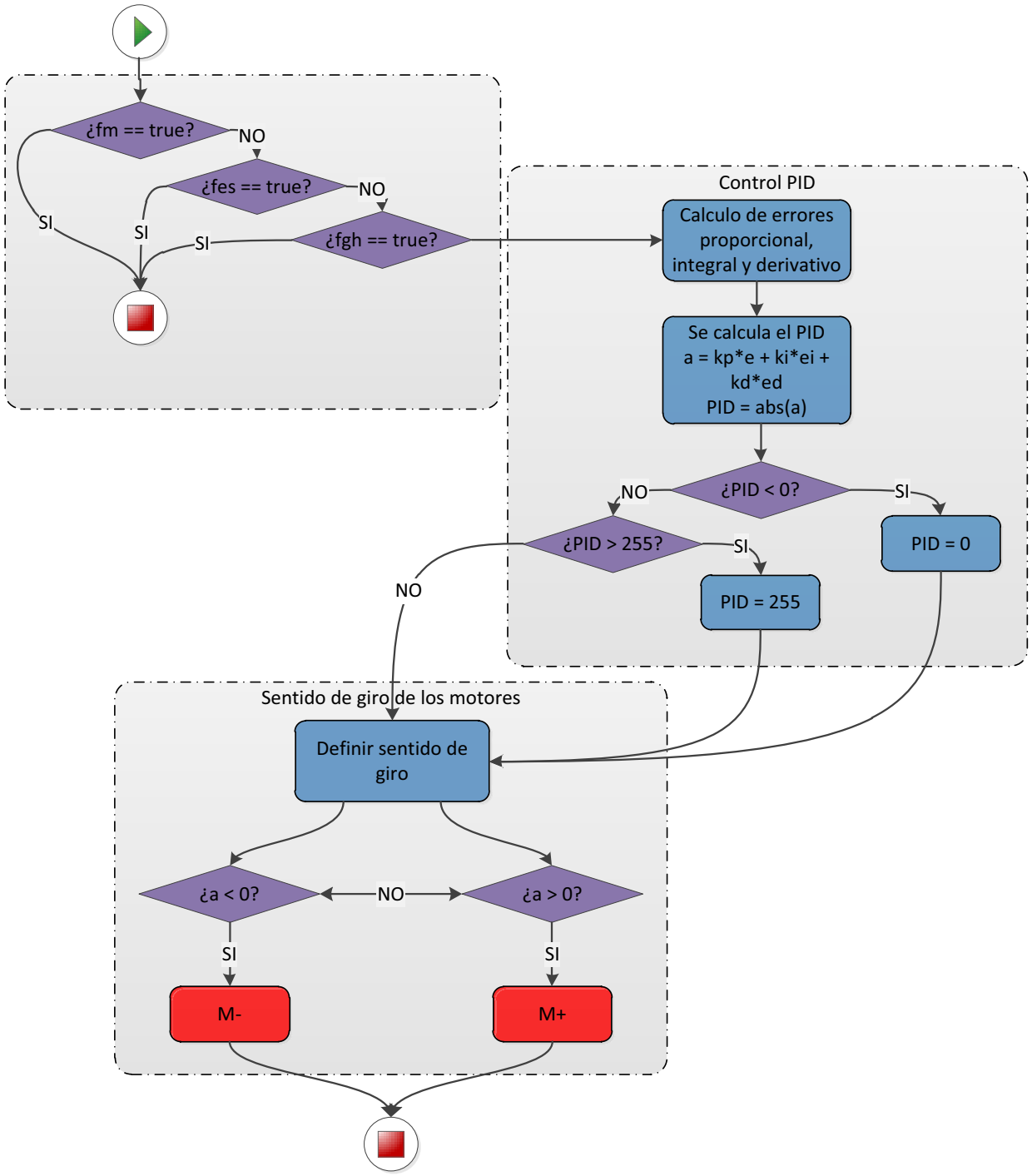


Diagrama 14 Control PID y determinación de los parámetros de los motores del Scorbot

13 RESULTADOS

13.1 Funcionamiento automático y control

El diseño del funcionamiento automático se creó para poder tener control de la posición del brazo. Esto es que el brazo pueda responder a la acción de fuerzas externas.

El control de posición se resuelve por medio de un controlador PID. Después de su implementación en el microcontrolador se realizó la sintonización del control. Se partió con el control proporcional. Al existir fuerzas de fricción, inerciales, entre otras, cuando los errores son pequeños la acción señalada por este control no genera una suficiente repuesta para superar estas fuerzas por lo que nunca se alcanza la posición deseada. El error que resulta de este principio no supera el 0.2%. Para eliminar este error se implementó la parte integral del controlador. Esta genera una respuesta proporcional a la acumulación de errores pasados. Al haber un error por pequeño que sea, el controlador detectará este error e intentara generar una acción para corregirlo, al ir acumulando un error la acción para contraponerse será mayor, por lo que el controlador intentará alcanzar un error igual a cero. Al acumular estos errores se puede generar un sobre paso en la respuesta el controlador, este sobre paso será después considerado y la respuesta tendera a amortiguarse para este sistema. Sin embargo, al ver el robot en funcionamiento se decidió que el error que existe en el controlador proporcional es aceptable, mientras que el pequeño sobrepaso del controlador proporcional-Integral no es aceptable en una cuestión visual.

Al alcanzar una posición deseada por el robot, el microcontrolador manda una señal al CPU. Esta señal indica al CPU que una nueva posición puede ser mandada por el mismo, para que el microcontrolador pueda alcanzar esta nueva posición. Esto genera un ciclo en el cual el microcontrolador resuelve las posiciones deseadas y señalándole al CPU que una nueva posición puede ser enviada para resolverse.

13.2 Obtención de las posiciones a partir de la Integración

El cálculo de las posiciones es una operación que inicia con la lectura de los datos de aceleración en la PC, y con éstos datos se realiza una doble integración de dos formas posibles: la integración rectangular y la integración de Boole. La primera de las formas de integración numérica es rápida y ágil, ya que se necesitan solo de 2 puntos para realizarse, y como siempre tenemos el punto anterior guardado, cada vez que llega un dato de aceleración se calcula una posición al instante, tardando entre 50 y 51 milisegundos. La segunda integración, más precisa, requiere de 17 datos de aceleración, debido a que para obtener mayor precisión se requieren más puntos intermedios. Se ha propuesto un experimento para verificar que las funciones de integración realicen su tarea de forma correcta. Los datos de

aceleración que se reciben en la PC, así como el tiempo que existe entre lecturas consecutivas se han guardado en un arreglo de 7 elementos, cuyos finales 3 elementos son las posiciones obtenidas luego de los métodos de integración. Con el fin de mejorar la validez de dicho experimento se han realizado dos pruebas. A continuación se muestran dos tablas (Tabla 16 para la prueba 1) donde se presentan los datos de aceleración leídos en la PC y las posiciones que resultan luego de la doble integración:

Prueba 1

AceX [m/s2]	AceY [m/s2]	AceZ [m/s2]	tiempo entre lecturas [s]	PosX [m]	PosY [m]	PosZ [m]	Tiempo acumulado [s]
0.133	0.615	-0.097	0.050	0.08	0.38	-0.06	0.050
-0.729	1.403	0.100	0.050	-0.12	2.41	-0.18	0.100
-0.554	1.436	0.040	0.050	-1.50	7.48	-0.21	0.150
-0.587	2.148	0.049	0.050	-4.40	16.56	-0.10	0.200
-0.903	0.962	-0.027	0.050	-8.94	29.83	0.09	0.250
-1.114	1.796	-0.528	0.051	-15.83	47.13	-0.07	0.301
-2.931	0.551	-0.252	0.051	-26.67	67.76	-1.10	0.352
-0.820	0.244	-0.147	0.051	-42.57	90.43	-2.90	0.403
-0.700	-1.104	-1.021	0.051	-61.90	113.05	-5.71	0.454
-1.732	-1.484	-0.305	0.050	-83.35	133.07	-10.04	0.504
0.112	-3.939	-0.042	0.050	-107.32	148.08	-15.42	0.554
1.177	-3.105	-0.183	0.050	-131.50	155.29	-21.15	0.604
-0.279	-3.120	0.186	0.050	-154.32	154.22	-27.02	0.654
1.627	-3.237	0.042	0.050	-175.73	145.28	-32.75	0.704
0.504	-3.792	-0.247	0.051	-195.32	127.54	-38.58	0.755
0.504	-3.792	-0.247	0.051	-212.87	100.29	-44.86	0.806
3.102	-1.992	-0.604	0.051	-227.43	64.35	-52.02	0.857
3.171	1.286	-0.547	0.051	-235.55	24.20	-60.48	0.908
3.368	0.462	-0.774	0.051	-235.35	-15.29	-70.55	0.959

Tabla 16 Prueba 1 de la obtención de posiciones por integración

Como puede constatarse en laTabla 16, se ha tenido que agregar la columna de tiempo acumulado, necesaria para saber el tiempo en que ocurren las aceleraciones, y no solamente el tiempo que transcurre entre lecturas. De la columna de tiempo entre lecturas puede obtenerse con facilidad la columna de tiempo acumulado.

Es importante hacer notar que en las gráficas puede constatarse que al cambiar de signo la aceleración se obtiene un punto de inflexión (cambio de concavidad) en la posición, lo cual es como debe ser, debido a que la aceleración $(\ddot{x})$ es la segunda derivada de la posición (x). Como sabemos, para hallar el punto de inflexión basta con la segunda derivada de la función a cero $(f''(x) = \ddot{x} = 0)$, que corresponde lógicamente con el punto donde la aceleración cambia de signo. (Ver Imagen 53)

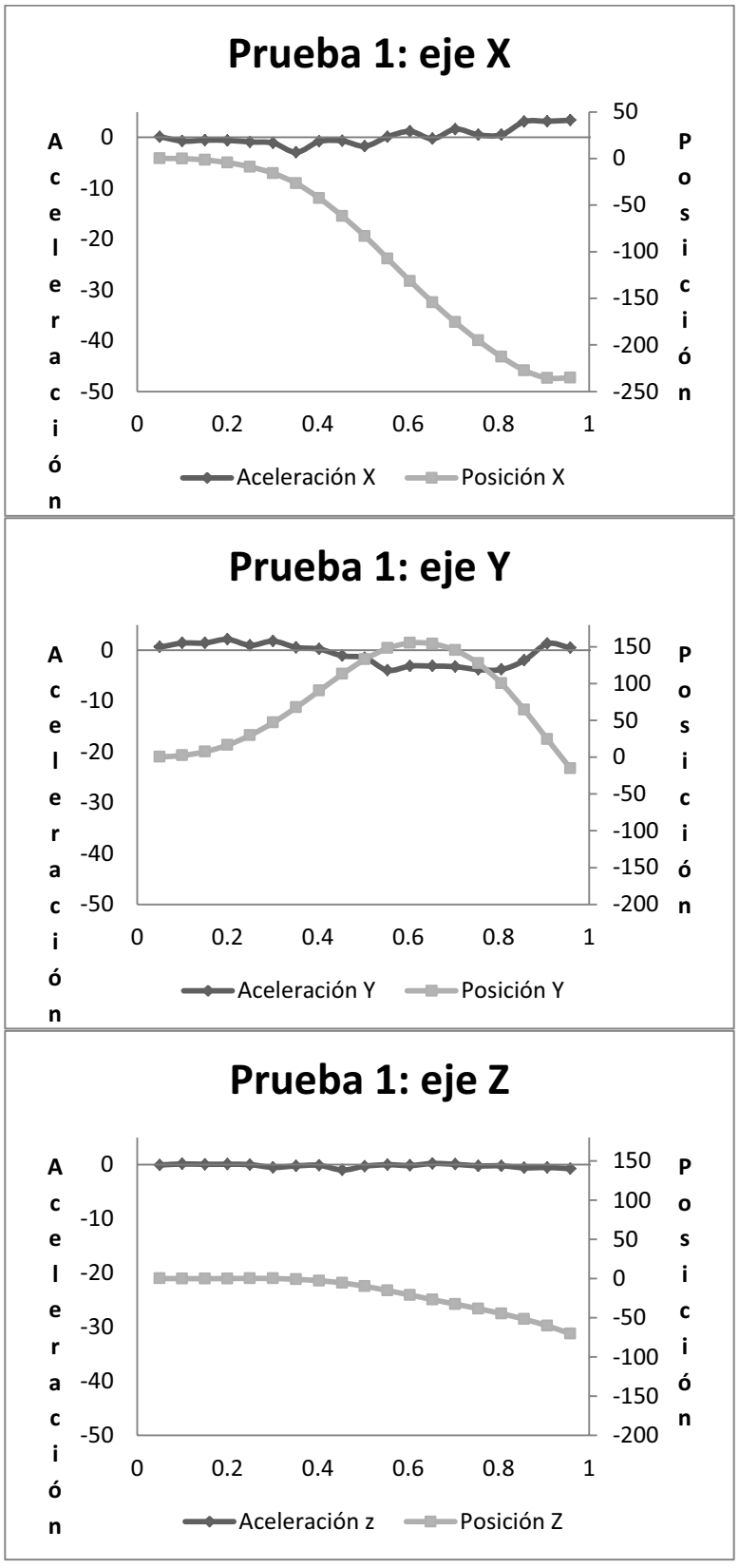


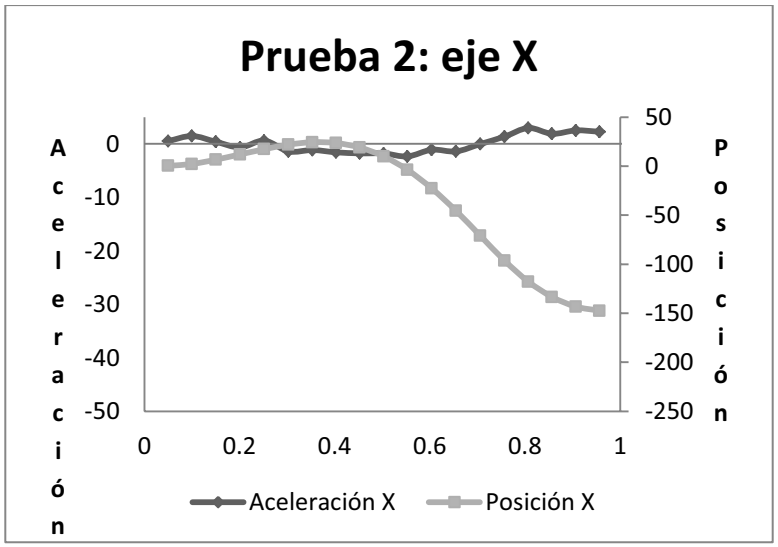
Imagen 53 Gráfica de aceleraciones [m/s^2] y posiciones [mm] de la prueba 1

Con el fin de verificar los resultados obtenidos por la prueba 1 se ha realizado una segunda, cuya tabla y gráficas, realizadas con la misma metodología que la anterior, se presentan la Tabla 17:

Prueba 2

AceX [m/s2]	AceY [m/s2]	AceZ [m/s2]	tiempo entre lecturas [s]	PosX [m]	PosY [m]	PosZ [m]	Tiempo acumulado [s]
0.533	-0.137	-0.042	0.050	0.33	-0.09	-0.03	0.050
1.469	-0.414	0.035	0.050	2.25	-0.60	-0.08	0.100
0.397	-0.599	-0.022	0.050	6.59	-2.09	-0.13	0.150
-0.654	-0.419	-0.044	0.051	12.03	-4.92	-0.22	0.201
0.607	-0.943	0.047	0.051	17.27	-9.30	-0.35	0.252
-1.418	-0.307	-0.037	0.051	21.96	-15.37	-0.47	0.303
-1.152	-0.472	-0.113	0.050	24.43	-22.61	-0.68	0.353
-1.583	-0.280	0.016	0.050	23.59	-30.81	-1.04	0.403
-1.770	-0.354	0.042	0.050	18.94	-39.87	-1.42	0.453
-1.792	2.143	-0.094	0.050	9.96	-48.21	-1.81	0.503
-2.347	3.019	0.004	0.050	-3.82	-52.21	-2.28	0.553
-1.095	2.040	-0.140	0.051	-22.76	-49.71	-2.90	0.604
-1.409	1.499	-0.793	0.051	-45.57	-41.62	-4.22	0.655
-0.030	0.919	-0.585	0.051	-70.94	-29.65	-7.05	0.706
1.354	1.997	-0.496	0.051	-96.38	-14.22	-11.47	0.757
2.975	1.295	-0.152	0.050	-117.77	4.83	-16.90	0.807
1.881	0.879	-0.085	0.050	-133.43	27.30	-22.88	0.857
2.518	0.857	0.057	0.050	-143.30	52.20	-29.03	0.907
2.271	-0.081	-0.281	0.050	-147.42	78.68	-35.34	0.957

Tabla 17 Prueba 2 de la obtención de posiciones por integración



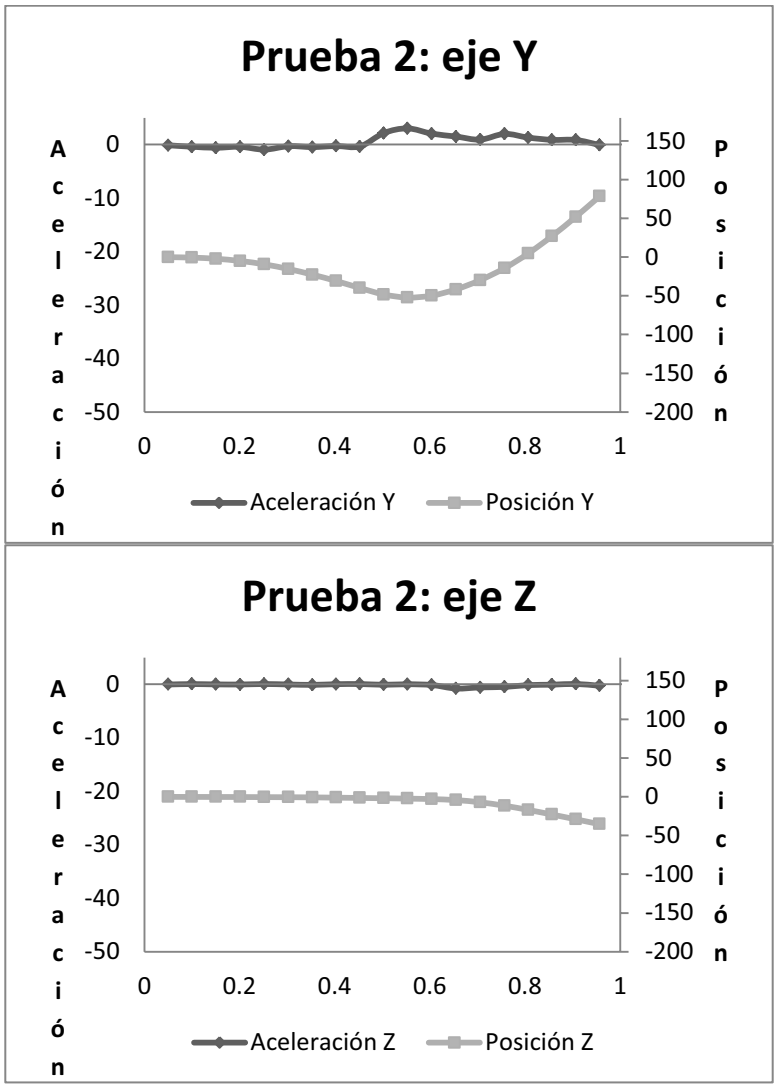


Imagen 54 Gráfica de aceleraciones [m/s^2] y posiciones [mm] de la prueba 2

13.3 Precisión de la teleoperación

Los resultados más tangibles y medibles se obtienen al comparar las trayectorias deseadas (las generadas por el dispositivo maestro) con las trayectorias realizadas por el manipulador Scorbot. Con el fin de presentar resultados de éste orden, se ha propuesto un simple experimento, en el cual el dispositivo maestro ha generado una trayectoria casi lineal. Podemos más adelante medir la trayectoria realizada y comparar algunos de sus parámetros con la trayectoria deseada. Los resultados de dicho experimento (punto por punto y trayectoria) son presentados a continuación.

13.3.1 Instalación de los componentes para la prueba

Para realizar el experimento propuesto se ha montado sobre el Scorbot un dispositivo que dibuja utilizando un plumón. Más adelante, una prueba de éste estilo sustituyendo el plumón con un cortador podría mostrar por primera vez una aplicación útil en la industria para un brazo de éste tipo. El gripper del Scorbot sostiene éste dispositivo de forma cilíndrica, de manera que siempre esté apuntando hacia abajo. El dispositivo cuenta con un resorte en su interior, de manera que pequeños cambios e imprecisiones dentro del movimiento del efector final no impidan al plumón marcar la trayectoria por la que pasa. Como se puede ver en la Imagen 55, se ha tenido que sumar la longitud del dispositivo a la cinemática inversa para trabajar en el plano de la mesa:

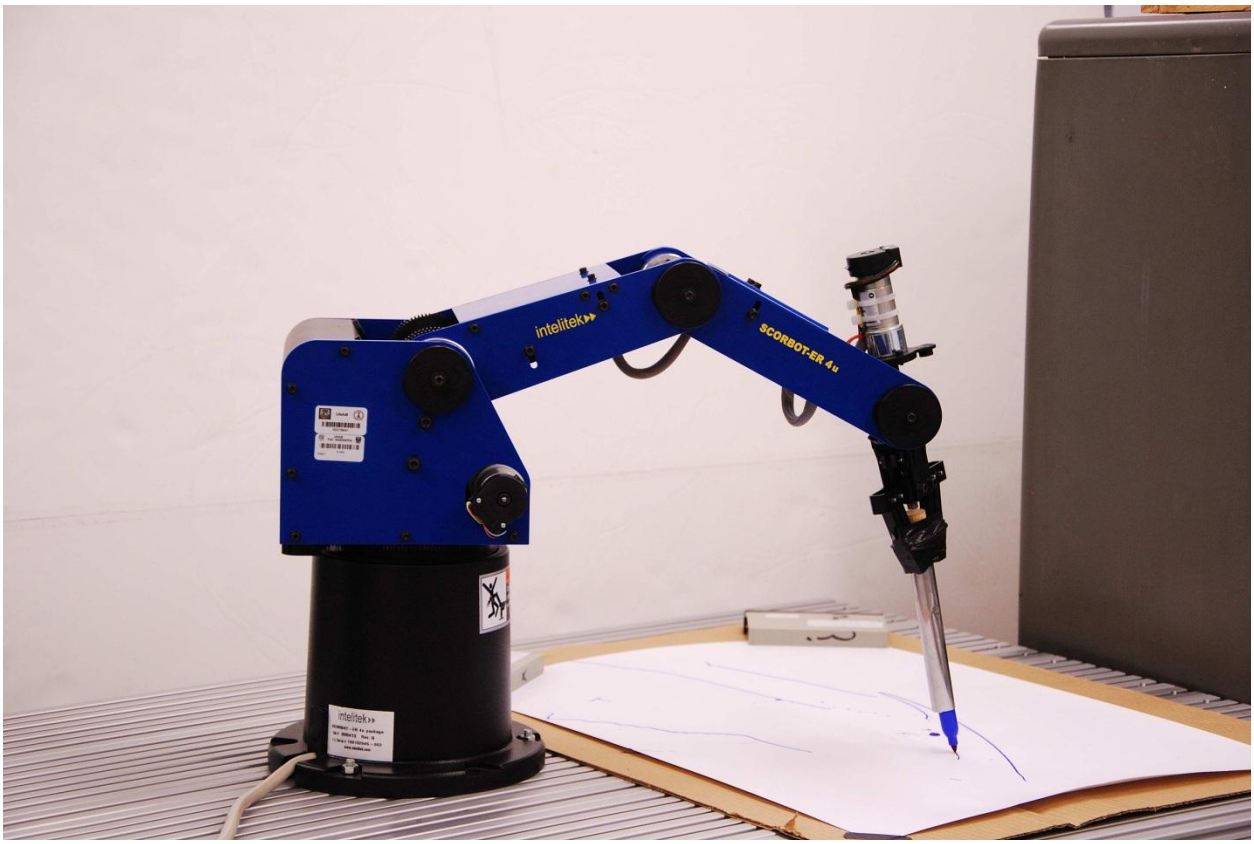


Imagen 55 Instalación del equipo para el experimento de dibujado

13.3.2 Generación de la trayectoria de prueba

Se ha generado una trayectoria, con el acelerómetro, en línea recta sobre el plano z = constante = 152 (donde 152 es la altura del plumón utilizado, de forma que se dibujan trayectorias en la mesa de trabajo, donde, para efectos del sistema coordenado z = 0). Cabe mencionar que los ejes han sido dibujados sobre la imagen para mostrar claramente lo que sucede, y que en la pantalla de la aplicación final no existen dichos ejes. Además, los círculos delimitan el área de trabajo, solo mientras los puntos de la

trayectoria dentro del anillo formado por dichos círculos puede ese punto ser tomado en cuenta para ser enviado al manipulador Scorbot.

En la Imagen 56 se muestra la trayectoria que se despliega en al PC y sobre la que se realizaron las pruebas:

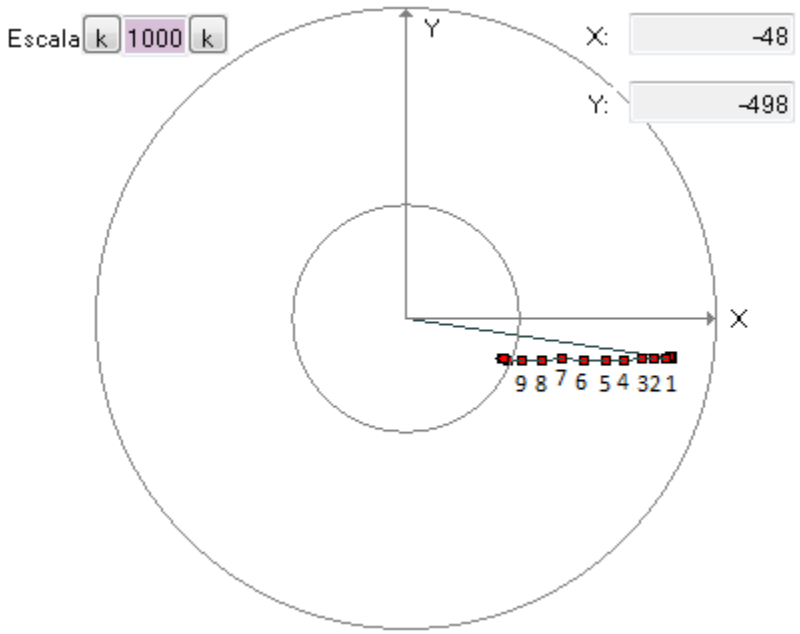


Imagen 56 Generación de la trayectoria lineal de prueba.

13.3.3 Obtención de los puntos deseados para la prueba Punto por Punto

Se desea comparar cada uno de los puntos que se debieron alcanzar con los que se alcanzaron, para dicho propósito, se han obtenido los puntos deseados a partir del muestreo en pantalla de los puntos desplegados. Si colocamos el puntero del mouse sobre el centro del punto de la trayectoria deseada, la programación nos indica las coordenadas 'x' y 'y' de dicho punto. Al realizar éste proceso, obtenemos la Tabla 18, donde el primer punto es el punto más alejado del centro del eje coordenado:

Punto	Coordenadas en [mm]
1	(393, -60,152)
2	(357, -67,152)
3	(327, -66,152)
4	(303, -66,152)
5	(267, -66,152)
6	(234, -60,152)
7	(204, -63,152)
8	(177, -63,152)
9	(153, -63,152)

Tabla 18 Coordenadas en [mm] de los puntos de la trayectoria deseada

13.3.4 Obtención de los puntos reales para la prueba Punto por Punto

Gracias a la capacidad de repetitibilidad del sistema, con el guardado de trayectorias se ha realizado la trayectoria física con el Scorbot en repetidas ocasiones, en total 7 veces. Deseamos ver con estas repeticiones que tan similares son las trayectorias entre sí, además de medir y comparar con la trayectoria deseada. Por lo tanto, se muestran en la Tabla 19 los datos recabados de ésta prueba, para 3 de las trayectorias:

Punto	Coordenadas en [mm]		
	Prueba 1	Prueba 2	Prueba 3
1	(443, -57)	(440, -58)	(445, -58)
2	(402, -60)	(407, -64)	(418, -65)
3	(368, -63)	(379, -65)	(382, -64)
4	(340, -64)	(339, -66)	(345, -57)
5	(289, -63)	(296, -65)	(309, -63)
6	(258, -58)	(259, -57)	(275, -60)
7	(216, -60)	(221, -65)	(225, -58)
8	(181, -63)	(189, -62)	(184, -61)
9	(134, -62)	(142, -63)	(138, -63)

Tabla 19 Puntos (x, y, z = 0) de las pruebas

13.3.5 Repetetibilidad de las trayectorias generadas

Podemos observar tanto gráficamente como analíticamente que las tres trayectorias estudiadas son muy similares. Sin embargo, se quiere conocer su factor de repetibilidad, por lo que se realiza un estudio estadístico. Para realizar éste estudio y ver que variables queremos saber si son repetibles, se debe definir un parámetro de estudio, en éste caso se eligió la distancia de cada punto de cada prueba hasta el eje z. Ésta distancia r se puede conocer de forma sencilla (de donde se obtiene la Tabla 20):

$$r = \sqrt{Px^2 + Py^2}$$

Punto	Distancia r en [mm]		
	Prueba 1	Prueba 2	Prueba 3
1	447	444	449
2	406	412	423
3	373	385	387
4	346	345	350
5	296	303	315
6	264	265	281
7	224	230	232
8	191	199	194
9	148	155	152

Tabla 20 Distancias r de los puntos al eje z

La repetibilidad es una medida estadística de la consistencia entre medidas repetidas de un mismo carácter en un mismo individuo. Generalmente se la denomina como r_i y su valor se expresa como una proporción. Un valor de repetibilidad de uno indica que la medida es perfectamente consistente y repetible, y que el investigador no comete ningún error en la medición de ese carácter. Un valor de cero indica que las medidas repetidas obtenidas de ese carácter son tan distintas como si se hubieran tomado a partir de individuos distintos tomados al azar. Para calcularla:

$$r_i = \frac{E}{E + D} \tag{1}$$

Donde:

$$E = \frac{MS_{Effect} - MS_{Error}}{2} \tag{2}$$

$$D = MS_{Error} \tag{3}$$

Éstos valores (MS_{Effect} y MS_{Error}) son componentes del análisis de varianzas (*one-way ANOVA*), y para obtenerlos es necesario realizar un cálculo complejo. Debido a su complejidad dicho cálculo no se muestra a continuación, más se muestra el resultado final del cálculo de la repetibilidad:

$$r_i = 0.9973$$

Podemos ver que la repetibilidad es excelente, pues se acerca mucho a uno.

13.3.6 Comparación de la trayectoria deseada con la real

Si deseamos saber que tan fielmente sigue el Scorbot una sucesión de puntos, se debe realizar una comparación entre la trayectoria deseada y la trayectoria real. Se puede realizar una comparación directa de los puntos de éstas dos trayectorias y obtener un porcentaje de error. Para realizar esto podemos obtener la distancia con respecto al eje z y comparar de ésta manera las dos trayectorias. A continuación se muestra la Tabla 21, donde estos valores son comparados, y donde se muestra el porcentaje de error:

Punto	Distancia r en [mm]				Porcentaje de error [%]		
	Deseada	Prueba 1	Prueba 2	Prueba 3	Prueba 1	Prueba 2	Prueba 3
1	398	447	444	449	12.3	11.6	12.8
2	363	406	412	423	11.8	13.5	16.5
3	334	373	385	387	11.7	15.3	15.9
4	310	346	345	350	11.6	11.3	12.9
5	275	296	303	315	7.6	10.2	14.5
6	242	264	265	281	9.1	9.5	16.1
7	214	224	230	232	4.7	7.5	8.4
8	188	191	199	194	1.6	5.9	3.2
9	165	148	155	152	10.3	6.1	7.9

Tabla 21 Comparación entre la trayectoria deseada y la obtenida

Estos porcentajes de error surgen a partir de la incapacidad del gripper de mantener una posición totalmente vertical. Éste error provoca un alargamiento de la trayectoria debido a que al estar el gripper mas alejado del eje z, se provoca un ángulo que provoca que el gripper se aleje un poco más de su destino. De manera similar, al estar el gripper cerca del eje z, se provoca un ángulo en sentido contrario al anterior que provoca que el gripper se acerque de más al punto destino. Éste desfasamiento del gripper es provocado por la falta de un perfecto control sobre el mismo, ya que ésta articulación utiliza dos motores girando en sentidos contrarios que tienen características físicas diferentes.

13.4 Trayectorias adicionales

Con el fin de conocer el alcance del sistema, se han realizado numerosas pruebas con diferentes trayectorias generadas. Esto nos permite ver el espectro posible de trayectorias que el sistema es capaz de manejar correctamente. Algunas de estas trayectorias, las que se han considerado de mayor relevancia, se muestran a continuación:

13.4.1 Trayectoria rectangular

Se ha realizado una trayectoria rectangular en el plano y = constante, utilizando el control manual de posiciones. Ésta trayectoria, y su funcionamiento se muestra en la Imagen 57.

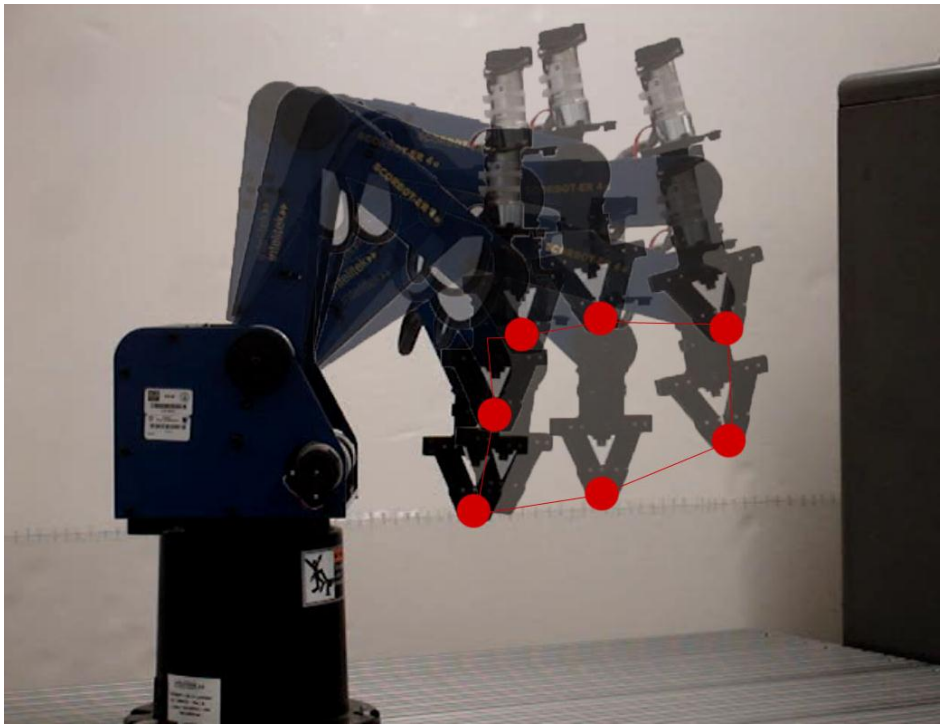


Imagen 57 Trayectoria rectangular en el plano y = constante

Puede verse claramente en la imagen anterior, donde se han colocado 7 de los puntos por donde pasa el brazo, que éste realiza la trayectoria deseada con algunos errores. Existe una tendencia a aumentar la coordenada de z siempre que los puntos estén más alejados del origen. Esto es por una acumulación de diferentes errores: errores de sensado de los pulsos de encoder y errores en las mediciones de las articulaciones ocupados en la cinemática inversa.

13.4.2 Trayectoria circular

De igual manera se ha realizado una trayectoria circular en el plano z = constate, donde se deseaba estudiar la capacidad del brazo de realizar movimientos que no fueran solamente líneas rectas. A En la Imagen 58 se muestra la trayectoria generada (la trayectoria deseada) por el dispositivo maestro:

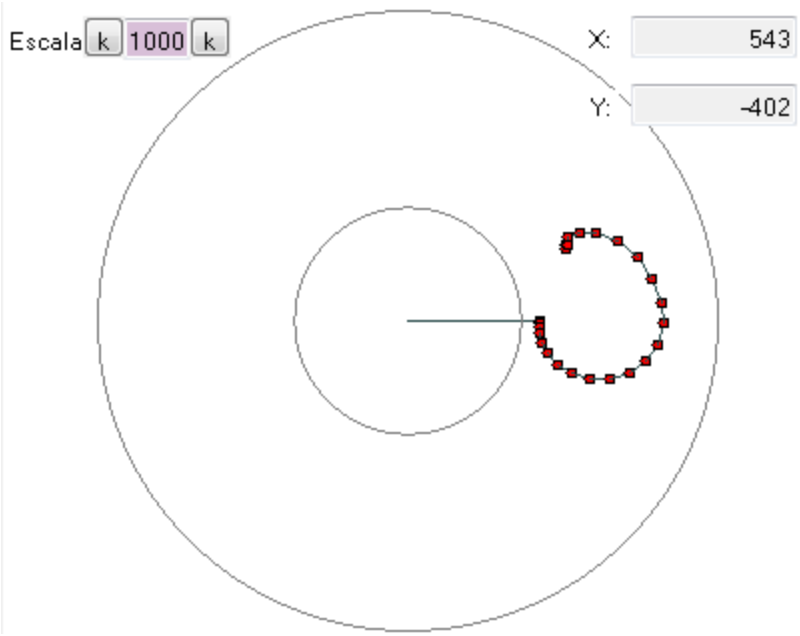


Imagen 58 Trayectoria circular deseada en el plano z = constante

Puede verse en la imagen 61 la trayectoria es realizada, que es muy similar a la deseada. De igual manera, existe la misma tendencia a elevar la coordenada de z mientras los puntos se encuentren más alejados del origen. Esto no se puede constatar en la imagen, pues ésta es una vista superior.

Por lo tanto, el sistema es capaz de realizar diversas trayectorias en el espacio de trabajo con efectividad, siempre y cuando una alta precisión no sea un requisito indispensable. (Ver Imagen 59)

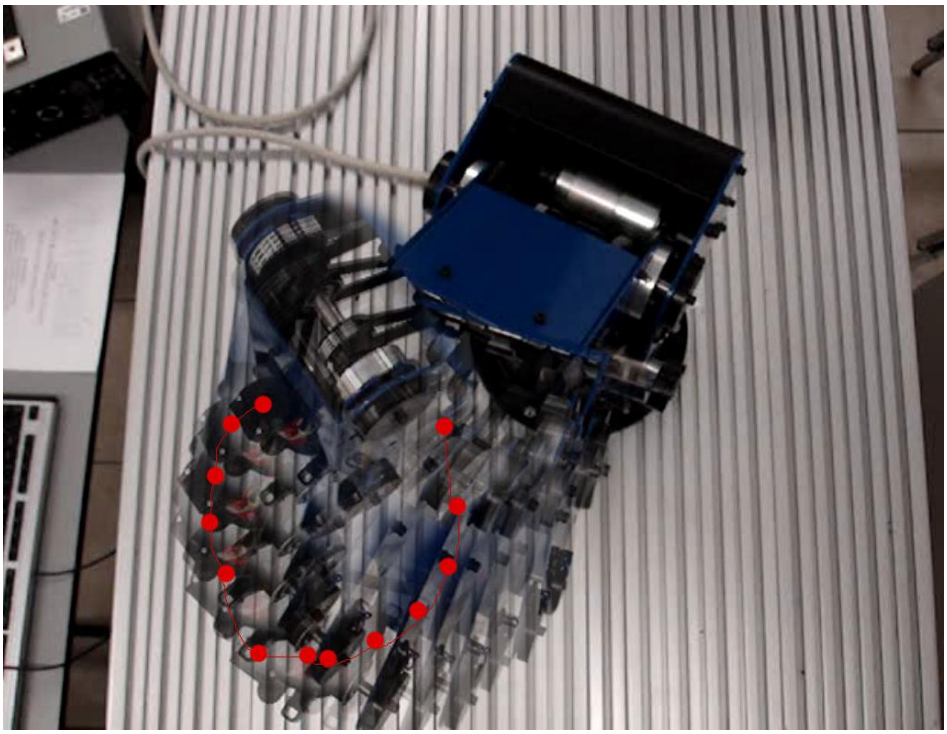


Imagen 59 Trayectoria circular en el plano z = constante

14 CONCLUSIONES Y TRABAJO FUTURO

Ésta tesis presenta una teleoperación de un brazo robótico tipo Scorbot mediante un dispositivo maestro que funciona con un sensor de aceleración. A lo largo del desarrollo, pero sobre todo en el capítulo de resultados se puede ver que dicho sistema funciona de manera correcta, pero tiene muchas limitaciones. Las soluciones individuales generadas en cada uno de los subsistemas han funcionado de manera satisfactoria.

Si se realiza un análisis del proceso general, puede verse que la adquisición de los datos de inicio, que en éste caso son aceleraciones en los tres ejes, es muy eficiente, veloz y rara vez presenta errores. De igual manera, la recepción de los datos en el control central de la PC es adecuada, incluso a pesar de las limitaciones que los módulos de comunicación inalámbrica podrían presentar. En éste caso, los módulos XBEE han realizado su tarea de forma correcta. Las operaciones correspondientes al control en la PC se realizan sin grandes sobresaltos y es muy eficiente. Todo el sistema hasta éste punto funciona de forma óptima. Sin embargo, se ha notado que la manera en que el usuario desearía manipular el dispositivo maestro podría mejorarse. El manejo de dicho dispositivo es intuitivo y en realidad muy sencillo, pero el hecho de tener que mantener la orientación correcta en todos momentos para evitar problemas donde la aceleración de la gravedad genere componentes en un eje cualquiera que no sea el eje vertical z provoca que el usuario deba manejar el dispositivo en una superficie plana, limitando los movimientos a dos ejes. Para solucionar éste problema debe diseñarse un soporte al dispositivo de manera que pueda trabajarse en el espacio sin estropear la orientación del artefacto. Éste dispositivo podría ser uno donde dos pesos compensadores estuvieran montados en los extremos de un eje vertical, entre los cuales estaría el dispositivo de sensado.

Se agradece muchas veces el contar con algunas de las funciones del control en la PC, como todas las de visualización, que facilitan mucho el trabajo, la operación y la detección de errores en el sistema. De igual manera es de gran ayuda un sistema que guarde todos los puntos de la trayectoria, y que ésta pueda ser grabada para realizarse en repetidas ocasiones. Además, el hecho de que sean grabadas en un formato propio permite mucha mayor seguridad en el sistema en tu totalidad, evitando mandar información que podría dañar a algún componente.

Se sospecha que el sistema de sensado dentro para los motores no es el más adecuado y que existe probablemente pérdida de información. La implementación recurre al uso de varias interrupciones en el microcontrolador para la lectura de los encoders de cuadratura. Es posible que el microcontrolador se sature (si la frecuencia de lectura es muy alta comparada con la frecuencia a la que trabaja el microcontrolador) que no pueda procesar tantas interrupciones además de leer el puerto serial y realizar las tareas de control de los motores. Para reducir o evitar estas pérdidas se deberá diseñar una plataforma donde un solo microcontrolador se encargue específicamente en leer estas interrupciones. Esta solución intenta mantener la posibilidad de mover todas las articulaciones al mismo tiempo, al igual que la velocidad con la que se podría mover cada articulación.

Mantener la integridad del equipo es una necesidad primordial, y para ello la detección de colisiones en la búsqueda de la posición de HOME cumple una función crucial. Esta localiza a cada articulación en una posición inicial determinada, sin importar en qué sentido comienza a girar una articulación.

El brazo Scorbot representa una buena solución para la implementación de un robot antropomórfico con 6 grados de libertad. Aunque es un diseño de los años 80, sigue estando en buenas condiciones para su funcionamiento e integración con sistemas más modernos en su utilización. La gran mayoría de sus articulaciones son controladas por un solo motor, a excepción del gripper. El cual consta de 2 motores que generan diferentes movimientos en combinación con las posibilidades de encendido de cada motor. Esto representó varias dificultades en el control preciso de la posición de la articulación, cuando ambos motores se movían al mismo tiempo.

Una de las grandes dificultades a la hora de realizar una teleoperación es acercarse a una operación en tiempo real. En el presente proyecto se estuvo muy lejos de tal objetivo. Existe poco retraso de la información en todo el proceso electrónico, que va desde la toma de datos hasta las instrucciones generadas por el microcontrolador que gobierna al brazo robótico. Justamente gran parte de la pérdida de tiempo se encontraba en el movimiento mecánico del mismo manipulador Scorbot. La velocidad de sus motores es muy baja para los fines deseados, y es imposible realizar trayectorias con la suficiente agilidad.

Otro de los grandes problemas con los que se enfrento es el control de las trayectorias. Se realizó un control PID de los puntos dentro de las trayectorias, y éste se implementó solamente con el uso de un PWM, las señales de pulso modulado pueden ser eficientes, al simular señales analógicas. Sin embargo, en el futuro sería de gran ventaja diseñar un control que tome en cuenta todas las partes del sistema, desde la generación de trayectorias per se, hasta el control preciso de la dinámica de los motores. Dicho círculo de control es mucho más complicado que el realizado hasta el momento, pero aseguraría un movimiento más fino siguiendo una trayectoria bien realizada, debido a que hasta ahora solo se ha tomado en cuenta la cinemática inversa, pero no la dinámica. Tomar en cuenta las inercias y otros factores que en el presente trabajo fueron omitidos mejoraría sustancialmente la calidad de los movimientos del brazo. Un control que es muy usado por su eficiencia en robótica es aquel donde la planta considera tres partes (las articulaciones del robot y todas sus partes, los motores de DC y la electrónica de potencia) y un control P/PI que controla tanto posición como velocidad.

Cabe mencionar, de igual manera, que las plataformas elegidas para la realización del proyecto fueron muy útiles y que ayudan a formar una solución de una manera muy conveniente. La plataforma de Arduino, con su interfaz de programación similar a la de cualquiera con programación en C, facilita la sencillez en la implementación de soluciones. De igual manera, la plataforma .Net de Microsoft con su entorno en C# promueve un ambiente de desarrollo benéfico.

15 ANEXOS

15.1 Tabla de figuras y tablas

16 BIBLIOGRAFÍA

1. **Mecatronica - Portal.** Sistema mecatrónico. *Mecatrónica portal. Automatización, Robótica, Control y Diseño.* [En línea] 06 de Abril de 2009. http://www.mecatronica-portal.com/2009/04/185-sistema-mecatronico/.

2. **Macchiavello, Tatiana.** Robótica. *Monografías.* [En línea] 2008. http://www.monografias.com/trabajos31/robotica/robotica.shtml#concept.

3. **Niku, Saeed B.** *Introduction to Robotics: Analysis, Systems, Applications.* s.l. : Prentice Hall, 2001. p. 2-16, 222-242.

4. **Critchlow, Arthur J.** *Introduction to Robotics.* s.l. : Macmillan Publishing Company, 1985. p.58-62.

5. **M., Padilla.** *Teleoperación.* s.l. : UDLAP.

6. Numerical Integration. *wikipedia.org.* [En línea] 20 de 8 de 2010. http://en.wikipedia.org/wiki/Numerical_integration.

7. Control theory. *wikipedia.org.* [En línea] 20 de 9 de 2010. http://en.wikipedia.org/wiki/Control_theory.

8. *Real-Time Inverse Kinematics Techniques for Anthropomorphic Limbs.* **Deepak Tolani, Ambarish Goswami and Norman I. Badler.** s.l. : Computer and Information Science Department, University of Pennsylvania, 2000.

9. *A Two-Arm Robot System based on Trajectory Optimization and Hybrid Control including Experimental Evaluation.* **Wolfgang Meier, Joachim Graf.** s.l. : Institute for Real-Time Computer Control Systems and Robotics, Karlsruhe University, Germany, 1991.

10. *Accelerometer-Based Control of an Industrial Robotic Arm.* **Pedro Neto, J. Norberto Pires.** s.l. : The 18th IEEE International Symposium on Robot and Human Interactive Communication, 2009.

11. *Adaptive Simultaneous Position and Stiffness Control for a Soft Robot Arm.* **Bicchi, Giovanni Tonietti Antonio.** s.l. : Centro “E. Piaggio”, University of Pisa, Italy, 2002.

12. *Bilateral teleoperation:An historical survey.* **Peter F. Hokayem, MarkW. Spong.** s.l. : Coordinated Science Laboratory, University of Illinois, USA, 2006.

13. *Inverse Kinematics of a Human Arm.* **Kondo, Koichi.** s.l. : Robotics Laboratory Department of Computer Science, Stanford University, CA, USA.

14. *Design and Control of Dual Arm Robot Manipulator for Precision Assembly.* **ChanHun Park YoungDong Son, DooHyung Kim, KyoungTaik Park YoonSung Shin and HeeSeoK Ahn.** s.l. : International Conference on Control, Automation and Systems, 2007.

15. *Design of a web-enabled anthropomorphic robotic arm for teleoperation.* **G. Sen Gupta, S. C. Mukhopadhyay, Matthew Finnie.** s.l. : School of Engineering and Advanced Technology Massey University, Palmerston North, New Zealand, 2008.

16. *Development of a 6-Axis Robotic Arm Controller Implemented on a Low-Cost Microcontroller.* **Agus Bejo, Wanchalerm Pora and Hiroaki Kunieda.** s.l. : Tokyo Institute of Technology, Japan, 2009.

17. *Development of Industrial Dual Arm Robot for Precision Assembly of Mechanical Parts for Automobiles.* **Chan-Hun Park, Kyoung-Taik Park and Dae-Gab Gweon.** s.l. : Department of Mechanical Engineering, KAIST, Taejeon, Korea, 2006.

18. *Using fuzzy logic in low cost microcontroller to increase accelerometer performances.* **Gaysse, Jérôme.** s.l. : ATMEL – ARM Application Group, 2005.

19. *Microcontroller-Based Architecture for Control of a Six Joints Robot Arm.* **Kabuka, Mansur R.** s.l. : IEEE Transactions on Industrial Electronics, 1988, Vol. 35. 2.

20. *Low-cost teleoperable robotic arm.* **Rogers, John R.** s.l. : Department of Civil and Mechanical Engineering, United States Military Academy, West Point, USA, 2009.

21. *Internet-based and Visual Feedback Networked Robot Arm Teleoperation System.* **Cong, Shuang.** s.l. : IEEE, 2010.

22. *The Assistive Robotic Arm.* **Asma S. Ali, Megan G. Madariaga.** s.l. : University of Connecticut, Department of Biomedical Engineering, USA, 2007.

23. *Teleoperación: técnicas, aplicaciones, entorno sensorial y teleoperación inteligente.* **Emmanuel Nuño Ortega, Luis Basañez Villaluenga.** s.l. : Universidad Politecnica de Cataluña, España, 2004.

24. *Use of path planning techniques based on harmonic functions for the haptic guidance of teleoperated assembly tasks.* **Carlos Vázquez, Jan Rosell.** s.l. : Universidad Politecnica de Cataluña, España, 2007.

25. *Human-like Motion of a Humanoid Robot Arm Based on a Closed-Form Solution of the Inverse Kinematics Problem.* **Dillmann, T. Asfour and R.** s.l. : Computer Science Department, University of Karlsruhe, Alemania, 2003.

26. *A triaxial accelerometer for measuring arm movements.* **Wiktorin, Eva Bernmark Christina.** s.l. : Karolinska Institute, Stockholm, Sweden, 2002.

27. *Stable Tracking in Variable Time-Delay Teleoperation.* **Benedetti, Carlo.** s.l. : California Institute of Technology, CA, USA, 2001.

28. **Björck, Germund Dalquist and Åke.** *Numerical Methods in Scientific Computing, Volume 1.* 2008 : Society for Industrial and Applied Mathematics.

17 APÉNDICES

17.1 Apéndice A: Tabla comparativa entre diferentes tipos de acelerómetros comerciales

	Acelerómetro		
Característica	ADXL203CE	ADXL213AE	ADXL320
Marca	Analog Devices	Analog Devices	Analog Devices
Rango	±1.7g	±1.2g	±5g
Interface	Analógico	PWM	Analógico
Ejes	2	2	2
Ancho de banda	500Hz	500Hz	500Hz
Consumo de energía	3-6V, 700-1100µA	3-6V, 700-1100µA	2.4-6V, 350-480µA
Características adicionales	-	-	-
Ventajas	Buena resolución y un amplio rango de voltaje de operación	Buena resolución y un amplio rango de voltaje de operación	Bajo consumo de energía, amplio rango de voltaje, sencillo de utilizar
Desventajas	Costoso, requiere mucha energía y antiguo	Costoso, requiere mucha energía y antiguo	Solamente tiene 2 ejes

	Acelerómetro		
Característica	ADXL321	ADXL322	ADXL193
Marca	Analog Devices	Analog Devices	Analog Devices
Rango	±18g	±2g	±250g
Interface	Analógico	Analógico	Analógico
Ejes	2	2	1
Ancho de banda	500Hz	500Hz	400Hz
Consumo de energía	2.4-6V, 350-480µA	2.4-6V, 350-480µA	3.5-6V, 1.5-2mA
Características adicionales	-	-	-
Ventajas	Bajo consumo de energía, amplio rango de voltaje, sencillo de utilizar	Bajo consumo de energía, amplio rango de voltaje, sencillo de utilizar	Rango muy elevado, amplio voltaje de operación y pequeño
Desventajas	Solamente tiene 2 ejes y poca resolución	Solamente tiene 2 ejes	Muy baja resolución, alto consumo de energía, únicamente tiene 1 eje

	Acelerómetro		
Característica	ADXL335	MMA7260Q	MMA7361
Marca	Analog Devices	Freescale	Freescale
Rango	±3g	±1.5, 2, 4, 6g	±1.5, 6g
Interface	Analógico	Analógico	Analógico
Ejes	3	3	3
Ancho de banda	1600 (x/y), 550 (z) Hz	350 (x/y), 150 (z) Hz	350 (x/y), 150 (z) Hz
Consumo de energía	1.8-3.6V, 350µA	2.2-3.6V, 500-800µA	2.2-3.6V, 400-600µA
Características adicionales		Selección de Gs, Modo de 'sleep'	Selección de Gs, Modo de 'sleep', detección de 0g
Ventajas	Sencillo de usar, 3 ejes, bajo costo, bajo consumo de energía	Económico, ahorro de energía, flexibilidad gracias a la elección de los Gs, interfaz sencilla	Económico, ahorro de energía, flexibilidad gracias a la elección de los Gs, interfaz sencilla, detección de caída libre
Desventajas	El rango podría ser limitado, alimentación de 3.6V	Más complicado para configurar	Más complicado para configurar

	Acelerómetro		
Característica	ADXL345	BMA180	SCA3000
Marca	Analog Devices	Bosch	VTI
Rango	±2, 4, 8, 16g	±1, 1.5, 2, 3, 4, 8, 16g	±2g
Interface	SPI and I^2C	SPI and I^2C	SPI
Ejes	3	3	3
Ancho de banda	3200Hz	2400Hz	300Hz
Consumo de energía	2.0-3.6V, 40-145µA	2-3.6V, 650-975µA	2.35-3.6V, 300-650µA
Características adicionales	Rango variable, detección de caída, detección de tap y doble tap	Rango variable, filtros digitales programables, detección de caída libre, detección de tap y doble tap, detección de pendiente	Frecuencia de respuesta seleccionable, salida de temperatura, interrupciones
Ventajas	Mucha flexibilidad, poco susceptible al ruido, miniatura, interfaces seriales	Mucha flexibilidad, poco susceptible al ruido, miniatura, interfaces seriales	Ofrece la mayor precisión, tiene regulador de voltaje
Desventajas	Difícil de configurar, requiere mayor cableado	Difícil de configurar, requiere mayor cableado	Puede estar limitado en el rango, es bastante costoso

17.2 Apéndice B: Código de programación del Arduino Duemilanove

```
//Incluimos las librerías que incluye la comunicación I2C
#include <Wire.h>
#include <MsTimer2.h>

//Definimos la dirección 0x40 que es la dirección predeterminada del acelerómetro BMA180
#define address 0x40

//Definimos variables
int lectura[6] = {0, 0, 0, 0, 0, 0};
int samples = 17;
int aceleracionX[17];
int aceleracionY[17];
int aceleracionZ[17];
int aceX = 0;
int aceY = 0;
int aceZ = 0;
long sumaX = 0;
long sumaY = 0;
long sumaZ = 0;
int i = 0;
boolean falla = false;
int prueba = 0;
float modulo = 0;
boolean bandera = false;
int tiempo = 0;
byte byteXYZ[6] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

//variables para transmisión en el XBEE
int checkSumInt = 0;
int sw=26;
byte bdato[26];
byte checkSum = 0;

void setup()
{
	//Iniciamos ls comunicación I2C y la comunicación serial con la computadora
	Wire.begin();
	Serial.begin(19200);
	//Llamamos a la función que configura al acelerómetro y esperamos un tiempo de 2s
	para iniciar el ciclo
	//Esto debe realizarse solo una vez o cada vez que cambiemos configuración
	//initializeBMA180();
	//Timer interrupt
	TCCR1A = B01000000; //bin
	TCCR1B = B00001011; //prescaled 1/64 [Hz] ciclos de .025s= 1/(16Mhz/64)*6250
	.001s= 1/(16Mhz/64)*250
```

```
        TIMSK1 = B00000010;   /* Enables the Timer1 Compare A interrupt */
        OCR1AH = 0x00; //0h.186A= dec6250          0h.1388=dec 250
        OCR1AL = 0xFA;
        //Imprimimos inicializacion
        //Serial.println("");
        //Serial.println("Inicio correcto, enviando datos!");
        //Serial.println("");
}

void loop()
{
        //Las direcciones en la memoria del BMA180 son:
        //0x03: acc_x_msb
        //0x02: acc_x_lsb
        //0x05: acc_y_msb
        //0x04: acc_y_lsb
        //0x07: acc_z_msb
        //0x06: acc_z_lsb
        //Inicia la comunicación y envía la dirección que se desea leer
        //Más adelante se recibe un byte de éste dispositivo y se almacena en una variable
        temporal
        Wire.beginTransmission(address);
        Wire.send(0x03);
        Wire.requestFrom(address, 1);
        while(Wire.available())
        {
                //Leemos los bits menos significativos de ace_X
                lectura[0] = Wire.receive();
        }
        prueba = Wire.endTransmission();
        if (prueba >= 1)
        {
                falla = true;
        }
        Wire.beginTransmission(address);
        Wire.send(0x02);
        Wire.requestFrom(address, 1);
        while(Wire.available())
        {
                //Leemos los bits mas significativos de ace_X
                lectura[1] = Wire.receive();
        }
        prueba = Wire.endTransmission();
        if (prueba >= 1)
        {
                falla = true;
        }
        Wire.beginTransmission(address);
        Wire.send(0x05);
        Wire.requestFrom(address, 1);
```

```
while(Wire.available())
{
	//Leemos los bits menos significativos de ace_Y
	lectura[2] = Wire.receive();
}
prueba = Wire.endTransmission();
if (prueba >= 1)
{
	falla = true;
}
Wire.beginTransmission(address);
Wire.send(0x04);
Wire.requestFrom(address, 1);
while(Wire.available())
{
	//Leemos los bits mas significativos de ace_Y
	lectura[3] = Wire.receive();
}
prueba = Wire.endTransmission();
if (prueba >= 1)
{
	falla = true;
}
Wire.beginTransmission(address);
Wire.send(0x07);
Wire.requestFrom(address, 1);
while(Wire.available())
{
	//Leemos los bits menos significativos de ace_Z
	lectura[4] = Wire.receive();
}
prueba = Wire.endTransmission();
if (prueba >= 1)
{
	falla = true;
}
Wire.beginTransmission(address);
Wire.send(0x06);
Wire.requestFrom(address, 1);
while(Wire.available())
{
	//Leemos los bits mas significativos de ace_Z
	lectura[5] = Wire.receive();
}
prueba = Wire.endTransmission();
if (prueba >= 1)
{
	falla = true;
}
//Obtenemos los valores reales de la aceleración, para poderlos mandar
```

```
aceleracionX[i] = transformarbits(lectura[0], lectura[1]);
aceleracionY[i] = transformarbits(lectura[2], lectura[3]);
aceleracionZ[i] = transformarbits(lectura[4], lectura[5]);
i++;
if (i == samples)
{
	for (int k = 0; k < samples; k++)
	{
		sumaX += aceleracionX[k];
		sumaY += aceleracionY[k];
		sumaZ += aceleracionZ[k];
	}
	i = 0;
	aceX = int(sumaX/samples);
	aceY = int(sumaY/samples);
	aceZ = int(sumaZ/samples);
	modulo = sqrt((sumaX/samples)*(sumaX/samples) +
	(sumaY/samples)*(sumaY/samples) + (sumaZ/samples)*(sumaZ/samples))/8192*2;
	sumaX = 0;
	sumaY = 0;
	sumaZ = 0;
	if (falla == false)
	{
		//Mandamos la información a través del puerto serie: aceX, aceY,
	aceZ y tiempo
		/*
		Serial.print("X:");
		Serial.print(aceX);
		Serial.print("\tY:");
		Serial.print(aceY);
		Serial.print("\tZ:");
		Serial.print(aceZ);
		Serial.print("\ttiempo: ");
		Serial.print(tiempo);
		Serial.print("\tg: ");
		Serial.print(modulo);
		Serial.println("");
		*/

		byteXYZ[0] = (byte)(aceX & 0xFF);
		byteXYZ[1] = (byte)((aceX >> 8) & 0xFF);
		byteXYZ[2] = (byte)(aceY & 0xFF);
		byteXYZ[3] = (byte)((aceY >> 8) & 0xFF);
		byteXYZ[4] = (byte)(aceZ & 0xFF);
		byteXYZ[5] = (byte)((aceZ >> 8) & 0xFF);

		send();
	}
	else
	{
```

```
                        Serial.println("FALLA");
                        falla = false;
                }
                tiempo = 0;
                //delay(40);
        }
}

void initializeBMA180()
{
        //Definimos variables
        int temp, result;
        boolean error = false;

        //Enviamos la primera dirección, que corresponde al ID del dispositivo, para fines
        de verificación
        Wire.beginTransmission(address);
        Wire.send(0x00);
        Wire.requestFrom(address, 1);
        while(Wire.available())
        {
                temp = Wire.receive();
        }
        Serial.print("Id = ");
        Serial.println(temp);
        result = Wire.endTransmission();
        if(result > 0)
        {
                error = true;
        }
        delay(10);

        // Enviamos la dirección ctrl_reg1 del registro y establecemos ee_w bit a 1 para
        habilitar la escritura de la memoria EEPROM
        Wire.beginTransmission(address);
        Wire.send(0x0D);
        Wire.send(B0001);
        result = Wire.endTransmission();
        if(result > 0)
        {
                error = true;
        }
        delay(10);

        // Enviamos la dirección bw_tcs del registro y establecemos el bw como un pasabaja
        de 10 Hz (0000 b)
        // De igual manera aqui estamos afectando la compensación de la temperatura, tcs,
        por ello elegimos 0% de compensación: 8 (1000)
        Wire.beginTransmission(address);
        Wire.send(0x20);
```

```
        Wire.send(B00001000);
        result = Wire.endTransmission();
        if(result > 0)
{
                error = true;
        }
        delay(10);

        // Enviamos la dirección offset_lsb1 del registro y establecemos el rango en +-2g
        (010 b)
        // de igual manera afectamos a offset_x y a smp_skip
        Wire.beginTransmission(address);
        Wire.send(0x35);
        Wire.send(B0100);
        result = Wire.endTransmission();
        if(result > 0)
        {
                error = true;
        }
        delay(10);

        //Verificamos que no hayan existido errores en la inicializacion
        if(error == false)
        {
                Serial.print("BMA180 configurado con éxito");
        }
        else
        {
                Serial.print("Error en la configuración");
        }
}

//Interrupción del timer
ISR (TIMER1_COMPA_vect)
{
        tiempo ++;
}

int transformarbits(int entrada1, int entrada2)
{
        int dato = 0;
        entrada1 >>= 2;
        entrada2 = entrada2 << 8;
        entrada2 = entrada2 >> 2;
        dato = entrada1 | entrada2;
        return dato;
}

void send()
{
```

```
//Inicio de comunicación
bdato[0] = (byte)0x7e;

//Numero de bytes a enviar
bdato[1] = (byte)0x00;
bdato[2] = (byte)0x16; //22 bytes

//Identificador
bdato[3] = (byte)0x10;

//Ack
bdato[4] = (byte)0x01;

//Dirección del dispositivo a enviar (Router)
bdato[5] = (byte)0x00;
bdato[6] = (byte)0x00;
bdato[7] = (byte)0x00;
bdato[8] = (byte)0x00;
bdato[9] = (byte)0x00;
bdato[10] = (byte)0x00;
bdato[11] = (byte)0x00;
bdato[12] = (byte)0x00; //b1router        a9 controller

//dirección de 16 bits
bdato[13] = (byte)0xFF;
bdato[14] = (byte)0xFE;

//Broadcast al máximo
bdato[15] = (byte)0x00;   //13

//Opciones
bdato[16] = (byte)0x00;   //14
//Mensaje
bdato[17] = (byte)0x24;
bdato[18] = byteXYZ[0];
bdato[19] = byteXYZ[1];
bdato[20] = byteXYZ[2];
bdato[21] = byteXYZ[3];
bdato[22] = byteXYZ[4];
bdato[23] = byteXYZ[5];
bdato[24] = (byte)tiempo;

for (int i = 3; i < (sw-1); i++)
{
       checkSumInt = checkSumInt + bdato[i];
}
while (checkSumInt > 255)
{
       checkSumInt = checkSumInt - 256;
}
```

```
        checkSumInt = 255 - checkSumInt;
        checkSum = (byte)checkSumInt;

        //Checksum
        bdato[25] = checkSum;
        //bdato[19] =(byte)0x5D;

        for(int n=0;n<sw;n++)
        {
               Serial.print(byte(bdato[n]));
        }

        checkSumInt=0;
        checkSum = 0;
}
```

17.3 Apéndice C: Código de programación de C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;
using System.Drawing.Imaging;
using System.Threading;
using System.IO;

namespace Gráfico_para_potenciómetro
{
    public partial class Telerobótica : Form
    {

        /*******************DECLARACIÓN DE VARIABLES Y CONSTANTES*******************/

        //Objeto y variables de Excel
        object[,] arregloExcel = new object[20, 7];
        Excel.Application DatosAExcel;
        int contadorExcel = 0;

        //Iniciamos objetos de otras clases
        Contacto contactos = new Contacto();

        //Definimos objetos gráficos
        static Graphics dibujo;
        static Image imagen;
        static Graphics dibujo2;
        static Image imagen2;
        static Graphics dibujo3;
```

```
static Image imagen3;

//Definimos las variables principales en la lectura del sensor y funcionamiento
general e interno del programa
int contadora17 = 0;
int numerografica = 1;
int contadorrango = 2;
int indicador = 0;
float velocidadinicialX = 0;
float velocidadinicialY = 0;
float velocidadinicialZ = 0;
string iniciocadena = "########";
string cadenaleidax = "00000000";
string cadenaleiday = "00000000";
string cadenaleidaz = "00000000";
string cadenaleidatiempo = "000";
float aceleracionX;
float aceleracionY;
float aceleracionZ;
float posicionX;
float posicionY;
float posicionZ;
float tiempoensegundos = 0F;
float alturaX;
float alturaY;
float alturaZ;
float n;
float salida;
bool banderadibujar = true;
float ua = 0;
float ub = 0;
float uc = 0;
float ud = 0;
float ue = 0;
int TemporalXUp = 0;
int TemporalXDown = 0;
int TemporalYUp = 0;
int TemporalYDown = 0;

//Definimos las variables de la cinemática inversa del Scorbot
//Los parámetros de entrada son:    d1: distancia de la base al motor1 en [mm]
//                                  d2: distancia del eje vertical al eje del
motor2 en [mm]
//                                  Li: longitud del eslabon i en [mm]
int IncReal = 10;
double d1 = 254;
double d2 = 25;
double L1 = 110;
double L2 = 220;
double L3 = 220;
double L4 = 145;
double[] q = new double[4];
double[] angulosScorbot = new double[4];
double EspacioDeTrabajo = 0;
bool DentroDelEspacio = false;
double FdeCosaTan = 0;
int XManual = 0;
int YManual = 0;
```

```
int ZManual = 0;
double F1 = 0;
double beta = 0;
float radioinutil = 100;

//Definimos las variables para integración
int escalaRealPosicion = 1000;
int tipodeintegracion = 0;
float[] velocidad = new float[5];
float[] vectorAceleracionX = new float[17];
float[] vectorAceleracionY = new float[17];
float[] vectorAceleracionZ = new float[17];
float AceleracionRectangularX = 0;
float AceleracionRectangularY = 0;
float AceleracionRectangularZ = 0;
float posicionRectangularX = 0;
float posicionRectangularY = 0;
float posicionRectangularZ = 300;
float aceleracionRectanteriorX = 0;
float aceleracionRectanteriorY = 0;
float aceleracionRectanteriorZ = 0;
float velocidadRectanteriorX = 0;
float velocidadRectanteriorY = 0;
float velocidadRectanteriorZ = 0;
float[] vectorTiempos = new float[17];
float[] kx = new float[5];
float[] ky = new float[5];
float[] kz = new float[5];
float posicionXX = 0;
float posicionYY = 0;
float posicionZZ = 0;

//Iniciamos constantes gráficas
float escala = 3F;
Point origen = new Point(0, 0);
Point ejeI = new Point(5, 175);
Point ejeD = new Point(265, 175);
Point ejeA = new Point(20, 5);
Point ejeB = new Point(20, 345);
Point ejeayuda1I = new Point(10, 215);
Point ejeayuda1D = new Point(260, 215);
Point ejeayuda2I = new Point(10, 255);
Point ejeayuda2D = new Point(260, 255);
Point ejeayuda3I = new Point(10, 295);
Point ejeayuda3D = new Point(260, 295);
Point ejeayuda4I = new Point(10, 135);
Point ejeayuda4D = new Point(260, 135);
Point ejeayuda5I = new Point(10, 95);
Point ejeayuda5D = new Point(260, 95);
Point ejeayuda6I = new Point(10, 55);
Point ejeayuda6D = new Point(260, 55);
Point ejeayuda7I = new Point(10, 335);
Point ejeayuda7D = new Point(260, 335);
Point ejeayuda8I = new Point(10, 15);
Point ejeayuda8D = new Point(260, 15);
Point texto1 = new Point(263, 8);
Point texto2 = new Point(263, 48);
Point texto3 = new Point(263, 88);
```

```
Point texto4 = new Point(263, 128);
Point texto5 = new Point(263, 208);
Point texto6 = new Point(263, 248);
Point texto7 = new Point(263, 288);
Point texto8 = new Point(263, 328);
Point Xmas = new Point(0, 190);
Point Xmenos = new Point(380, 190);
Point Ymas = new Point(190, 380);
Point Ymenos = new Point(190, 0);
Point punto1 = new Point(195, 190);
Point punto2 = new Point(230, 190);
Point punto3 = new Point(262, 190);
Point punto4 = new Point(293, 190);
Point punto5 = new Point(326, 190);
Point punto6 = new Point(357, 190);
Point leyenda1 = new Point(340, 12);
Point leyenda2 = new Point(340, 32);
Point marca0X = new Point(20, 175);
Point marca1X = new Point(50, 175);
Point marca2X = new Point(80, 175);
Point marca3X = new Point(110, 175);
Point marca4X = new Point(140, 175);
Point marca5X = new Point(170, 175);
Point marca6X = new Point(200, 175);
Point marca7X = new Point(230, 175);
Point marca8X = new Point(260, 175);
Point marca9X = new Point(290, 175);
Point marca0Y = new Point(20, 175);
Point marca1Y = new Point(50, 175);
Point marca2Y = new Point(80, 175);
Point marca3Y = new Point(110, 175);
Point marca4Y = new Point(140, 175);
Point marca5Y = new Point(170, 175);
Point marca6Y = new Point(200, 175);
Point marca7Y = new Point(230, 175);
Point marca8Y = new Point(260, 175);
Point marca9Y = new Point(290, 175);
Point marca0Z = new Point(20, 175);
Point marca1Z = new Point(50, 175);
Point marca2Z = new Point(80, 175);
Point marca3Z = new Point(110, 175);
Point marca4Z = new Point(140, 175);
Point marca5Z = new Point(170, 175);
Point marca6Z = new Point(200, 175);
Point marca7Z = new Point(230, 175);
Point marca8Z = new Point(260, 175);
Point marca9Z = new Point(290, 175);
Point presente = new Point();
Point anterior = new Point(202, 160);
Font fuente = new Font("Microsoft Sans Serif", 8, FontStyle.Regular);
Font fuente2 = new Font("Microsoft Sans Serif", 8, FontStyle.Regular);
Pen pluma = new Pen(Brushes.Gray, 1);

//Definimos constantes de dibujaraceleraciones y dibujarposiciones
double anguloinclinacion;
double anguloazimutal;
double radioesf;
double altura;
```

```
double Xsinescala = 0D;
double Ysinescala = 0D;
double Zsinescala = 0D;
double X = 0D;
double Y = 0D;
double Z = 0D;

//Definimos constantes de deibujarpunto
int Xell1 = 0;
int Yell1 = 0;
int Xell2 = 0;
int Yell2 = 0;
int Xell3 = 0;
int Yell3 = 0;
int radio = 0;
int radioAmp = 0;

//Constantes de enviada por el puerto serial, con las constantes de
transformación del puerto serie
string cadenaenviada = "#+0000,+0000,+0000,+0000,+0000";
string cadenaleidaSCORBOT = "";
string[] lineadeScorbot = new string[10];
int leerlinea = 0;
float[] gearratio = new float[5];
float[] angulosinicial = new float[4];
int bytesatport = 0;
byte[] XBEE = new byte[24];
int[] PulsosEncoders = new int[5];

/************************FUNCIÓN PRINCIPAL DE ENTRADA************************/

public Telerobótica()
{
    //Iniciamos componentes
    InitializeComponent();

    //Iniciamos los mensajes a desplegar del ToolTip
    toolTip1.SetToolTip(MapaPolar, "Mapa polar de aceleraciones");
    toolTip1.SetToolTip(Trayectorias, "Mapa de trayectorias -> Z = cte");
    toolTip1.SetToolTip(Aceleraciones, "Gráfica de baras de aceleraciones");
    toolTip1.SetToolTip(groupBox5, "Control Manual");
    toolTip1.SetToolTip(txtLineasTrayectoria, "Comandos para el Scorbot");
    toolTip1.SetToolTip(groupBox6, "Switch Control Manual/Automático");
    toolTip1.SetToolTip(groupTipoIntegración, "Switch Tipo de Integración");
    toolTip1.SetToolTip(txtIncrementoManual, "Incremento en todos los Ejes");
    toolTip1.SetToolTip(Iniciar, "Enviar comandos al Scorbot");
    toolTip1.SetToolTip(ClearTrayectorias, "Limpiar información de comandos y
    posiciones");

    //Hacemos override del color de fondo del menu
    Menu.Renderer = new MyRenderer();

    //Iniciamos posiciones iniciales de ángulos del Scorbot (28.71, 27.176,
    35.596, 9.76, 0)
    gearratio[0] = 28.71F;
    gearratio[1] = 27.176F;
    gearratio[2] = 35.596F;
    gearratio[3] = 9.76F;
```

```
        gearratio[4] = 0F;
        angulosinicial[0] = 0.0F;
        angulosinicial[1] = 105.5F;
        angulosinicial[2] = 100.0F;
        angulosinicial[3] = 53.0F;

        //Creamos la interrupción del puerto serial como evento
        PuertoSerial.DataReceived += new
        System.IO.Ports.SerialDataReceivedEventHandler(leerdatosseriales);
        SerialPort.DataReceived += new
        System.IO.Ports.SerialDataReceivedEventHandler(leerSCORBOT);

        //Creamos el evento del picture box para obtener la posición XY
        this.Trayectorias.MouseMove += new
        System.Windows.Forms.MouseEventHandler(this.Trayectorias_MouseMove);

        //Evento para el control manual de la Cinemática
        this.txtCinematicaX.TextChanged += new
        System.EventHandler(this.Cinematica_Manual);
        this.txtCinematicaY.TextChanged += new
        System.EventHandler(this.Cinematica_Manual);
        this.txtCinematicaZ.TextChanged += new
        System.EventHandler(this.Cinematica_Manual);

        //Inicialimos los objetos gráficos
        imagen = new Bitmap(Aceleraciones.Width, Aceleraciones.Height);
        imagen2 = new Bitmap(Mapa.Width, Mapa.Height);
        imagen3 = new Bitmap(Trayectorias.Width, Trayectorias.Height);
        dibujo = Graphics.FromImage(imagen);
        dibujo2 = Graphics.FromImage(imagen2);
        dibujo3 = Graphics.FromImage(imagen3);

        //Se desabilitan los botones de la comunicación serial, servirán solo de
        indicadores
        ConexionSerie.Enabled = false;
        Recibiendo.Enabled = false;

        //Se inicializan las variables gráficas
        ConexionSerie.BackColor = Color.Red;
        txtEjeX.BackColor = Color.GreenYellow;
        txtEjeY.BackColor = Color.Orange;
        txtEjeZ.BackColor = Color.Red;
        btnCambiarAceleracion.BackColor = Color.CornflowerBlue;
        btnAutomatico.BackColor = Color.Orange;
        btnIntBoole.BackColor = Color.White;
        btnIntRectangular.BackColor = Color.Orange;
        btnCambiarAceleracion.BackColor = Color.CornflowerBlue;
        btnCambiarPosicion.BackColor = Color.White;
        pictureScorbot.Visible = true;
        pictureBox1.Visible = true;
        groupBox5.Visible = false;
        grpModoManual2.Visible = false;
        dibujaraceleracion(0, 0, 0);
    }

    /*************************CAMBIO DE COLOR DEL MENU*************************/

    private class MyRenderer : ToolStripProfessionalRenderer
```

```
{
    //Se cambia el color de fondo del rectángulo del MenuStrip principal
    protected override void
    OnRenderMenuItemBackground(ToolStripItemRenderEventArgs e)
    {
        if (!e.Item.Selected) base.OnRenderMenuItemBackground(e);
        else
        {
            Rectangle rc = new Rectangle(Point.Empty, e.Item.Size);
            e.Graphics.FillRectangle(Brushes.Orange, rc);
            e.Graphics.DrawRectangle(Pens.BlueViolet, -1, 0, rc.Width + 2,
            rc.Height - 1);
        }
    }
}

/**************************CIERRE DEL PUERTO SERIE**************************/

private void apagarToolStripMenuItem_Click(object sender, EventArgs e)
{
    //Cerramos los puertos series
    if (PuertoSerial.IsOpen == true)
    {
        PuertoSerial.DiscardInBuffer();
        PuertoSerial.Close();
    }
    if (SerialPort.IsOpen == true)
    {
        SerialPort.DiscardInBuffer();
        SerialPort.Close();
    }
    this.Close();
}

/**********************MOSTRAR INFORMACIÓN DE CONTACTOS**********************/

private void contactoToolStripMenuItem_Click(object sender, EventArgs e)
{
    //Se muestra la ventana de los créditos
    contactos.Show();
}

/**************APERTURA Y CIERRE DE PUERTO SERIE DE ACELERÓMETRO**************/

private void iniciarToolStripMenuItem_Click(object sender, EventArgs e)
{
    //Configurar la comunicación serial con el Acelerómetro
    if (PuertoSerial.IsOpen == false)
    {
        PuertoSerial.Open();
    }
    if (PuertoSerial.IsOpen == true && SerialPort.IsOpen == true)
    {
        ConexionSerie.BackColor = Color.LawnGreen;
    }
}

private void terminarToolStripMenuItem_Click(object sender, EventArgs e)
```

```
{
    if (PuertoSerial.IsOpen == true)
    {
        PuertoSerial.DiscardInBuffer();
        PuertoSerial.Close();
        ConexionSerie.BackColor = Color.Red;
    }
}

/**********************GRÁFICAS DE ACELERACIÓN EN BARRAS**********************/

private void dibujaraceleracion(float A, float B, float C)
{
    //Dibujo de las gráficas en barra de las aceleraciones
    //Definimos las variables necesarias: gráficas, de texto, puntos, flotantes,
    etc.
    //Limpiamos la pantalla y dibujamos los ejes auxiliares y las escalas
    try
    {
        Aceleraciones.Image = imagen;
        pluma.DashStyle = System.Drawing.Drawing2D.DashStyle.Dash;
        dibujo.Clear(Color.White);
        dibujo.DrawLine(pluma, ejeayuda1D, ejeayuda1I);
        dibujo.DrawLine(pluma, ejeayuda2D, ejeayuda2I);
        dibujo.DrawLine(pluma, ejeayuda3D, ejeayuda3I);
        dibujo.DrawLine(pluma, ejeayuda4D, ejeayuda4I);
        dibujo.DrawLine(pluma, ejeayuda5D, ejeayuda5I);
        dibujo.DrawLine(pluma, ejeayuda6D, ejeayuda6I);
        dibujo.DrawLine(pluma, ejeayuda7D, ejeayuda7I);
        dibujo.DrawLine(pluma, ejeayuda8D, ejeayuda8I);
        if (contadorrango == 0)
        {
            //Aceleracion de 1g
            n = 1;
            dibujo.DrawString("1.00g", fuente2, Brushes.Black, texto1);
            dibujo.DrawString("750mg", fuente2, Brushes.Black, texto2);
            dibujo.DrawString("500mg", fuente2, Brushes.Black, texto3);
            dibujo.DrawString("250mg", fuente2, Brushes.Black, texto4);
            dibujo.DrawString("-250mg", fuente2, Brushes.Black, texto5);
            dibujo.DrawString("-500mg", fuente2, Brushes.Black, texto6);
            dibujo.DrawString("-750mg", fuente2, Brushes.Black, texto7);
            dibujo.DrawString("-1.00g", fuente2, Brushes.Black, texto8);
        }
        else if (contadorrango == 1)
        {
            //Aceleracion de 1.5g
            n = 1.5F;
            dibujo.DrawString("1.50g", fuente2, Brushes.Black, texto1);
            dibujo.DrawString("1.13g", fuente2, Brushes.Black, texto2);
            dibujo.DrawString("750mg", fuente2, Brushes.Black, texto3);
            dibujo.DrawString("375mg", fuente2, Brushes.Black, texto4);
            dibujo.DrawString("-375mg", fuente2, Brushes.Black, texto5);
            dibujo.DrawString("-750mg", fuente2, Brushes.Black, texto6);
            dibujo.DrawString("-1.13g", fuente2, Brushes.Black, texto7);
            dibujo.DrawString("-1.50g", fuente2, Brushes.Black, texto8);
        }
        else if (contadorrango == 2)
        {
```

```
    //Aceleracion de 2g
    n = 2;
    dibujo.DrawString("2.00g", fuente2, Brushes.Black, texto1);
    dibujo.DrawString("1.50g", fuente2, Brushes.Black, texto2);
    dibujo.DrawString("1.00g", fuente2, Brushes.Black, texto3);
    dibujo.DrawString("500mg", fuente2, Brushes.Black, texto4);
    dibujo.DrawString("-500mg", fuente2, Brushes.Black, texto5);
    dibujo.DrawString("-1.00g", fuente2, Brushes.Black, texto6);
    dibujo.DrawString("-1.50g", fuente2, Brushes.Black, texto7);
    dibujo.DrawString("-2.00g", fuente2, Brushes.Black, texto8);
}
else if (contadorrango == 3)
{
    //Aceleracion de 3g
    n = 3;
    dibujo.DrawString("3.00g", fuente2, Brushes.Black, texto1);
    dibujo.DrawString("2.25g", fuente2, Brushes.Black, texto2);
    dibujo.DrawString("1.50g", fuente2, Brushes.Black, texto3);
    dibujo.DrawString("750mg", fuente2, Brushes.Black, texto4);
    dibujo.DrawString("-750mg", fuente2, Brushes.Black, texto5);
    dibujo.DrawString("-1.50g", fuente2, Brushes.Black, texto6);
    dibujo.DrawString("-2.25g", fuente2, Brushes.Black, texto7);
    dibujo.DrawString("-3.00g", fuente2, Brushes.Black, texto8);
}
else if (contadorrango == 4)
{
    //Aceleracion de 4g
    n = 4;
    dibujo.DrawString("4.00g", fuente2, Brushes.Black, texto1);
    dibujo.DrawString("3.00g", fuente2, Brushes.Black, texto2);
    dibujo.DrawString("2.00g", fuente2, Brushes.Black, texto3);
    dibujo.DrawString("1.00g", fuente2, Brushes.Black, texto4);
    dibujo.DrawString("-1.00g", fuente2, Brushes.Black, texto5);
    dibujo.DrawString("-2.00g", fuente2, Brushes.Black, texto6);
    dibujo.DrawString("-3.00g", fuente2, Brushes.Black, texto7);
    dibujo.DrawString("-4.00g", fuente2, Brushes.Black, texto8);
}
else if (contadorrango == 5)
{
    //Aceleracion de 8g
    n = 8;
    dibujo.DrawString("8.00g", fuente2, Brushes.Black, texto1);
    dibujo.DrawString("6.00g", fuente2, Brushes.Black, texto2);
    dibujo.DrawString("4.00g", fuente2, Brushes.Black, texto3);
    dibujo.DrawString("2.00g", fuente2, Brushes.Black, texto4);
    dibujo.DrawString("-2.00g", fuente2, Brushes.Black, texto5);
    dibujo.DrawString("-4.00g", fuente2, Brushes.Black, texto6);
    dibujo.DrawString("-6.00g", fuente2, Brushes.Black, texto7);
    dibujo.DrawString("-8.00g", fuente2, Brushes.Black, texto8);
}
else if (contadorrango == 6)
{
    //Aceleracion de 16g
    n = 16;
    dibujo.DrawString("16.0g", fuente2, Brushes.Black, texto1);
    dibujo.DrawString("12.0g", fuente2, Brushes.Black, texto2);
    dibujo.DrawString("8.00g", fuente2, Brushes.Black, texto3);
    dibujo.DrawString("4.00g", fuente2, Brushes.Black, texto4);
```

```
                dibujo.DrawString("-4.00g", fuente2, Brushes.Black, texto5);
                dibujo.DrawString("-8.00g", fuente2, Brushes.Black, texto6);
                dibujo.DrawString("-12.0g", fuente2, Brushes.Black, texto7);
                dibujo.DrawString("-16.0g", fuente2, Brushes.Black, texto8);
            }
            else
            {
                n = 0;
            }
            //Obtenemos la altura en pixeles de las barras
            alturaX = 160 * A / n;
            alturaY = 160 * B / n;
            alturaZ = 160 * C / n;
            //Dibujamos las barras y el eje principal
            if (alturaX >= 0)
            {
                dibujo.FillRectangle(Brushes.GreenYellow, 15, 175 - alturaX, 60,
                alturaX);
                dibujo.DrawRectangle(Pens.Green, 15, 175 - alturaX, 60, alturaX);
            }
            else
            {
                dibujo.FillRectangle(Brushes.GreenYellow, 15, 175, 60,
                Math.Abs(alturaX));
                dibujo.DrawRectangle(Pens.Green, 15, 175, 60, Math.Abs(alturaX));
            }
            if (alturaY >= 0)
            {
                dibujo.FillRectangle(Brushes.Orange, 105, 175 - alturaY, 60,
                alturaY);
                dibujo.DrawRectangle(Pens.DarkBlue, 105, 175 - alturaY, 60, alturaY);
            }
            else
            {
                dibujo.FillRectangle(Brushes.Orange, 105, 175, 60,
                Math.Abs(alturaY));
                dibujo.DrawRectangle(Pens.DarkBlue, 105, 175, 60, Math.Abs(alturaY));
            }
            if (alturaZ >= 0)
            {
                dibujo.FillRectangle(Brushes.Red, 195, 175 - alturaZ, 60, alturaZ);
                dibujo.DrawRectangle(Pens.Black, 195, 175 - alturaZ, 60, alturaZ);
            }
            else
            {
                dibujo.FillRectangle(Brushes.Red, 195, 175, 60, Math.Abs(alturaZ));
                dibujo.DrawRectangle(Pens.Black, 195, 175, 60, Math.Abs(alturaZ));
            }
            dibujo.DrawLine(Pens.Gray, ejeI, ejeD);
        }
        catch
        {

        }
        banderadibujar = true;
        dibujarpunto(A, B, C);
    }
```

```
/**************************GRÁFICAS DE POSICIONES**************************/

private void dibujarposiciones(float A, float B, float C)
{
    //Dibujo de los puntos de las posiciones de X, Y y Z
    //Definimos las variables necesarias: gráficas, de texto, puntos, flotantes,
    etc.
    //Limpiamos la pantalla y dibujamos los ejes auxiliares y las escalas
    if (contadora17 == 16)
    {
        marca9X.Y = 175 - Convert.ToInt32(A);
        marca9Y.Y = 175 - Convert.ToInt32(B);
        marca9Z.Y = 175 - Convert.ToInt32(C);
        PointF[] marcaposicionesX = new PointF[] { marca0X, marca1X, marca2X,
        marca3X, marca4X, marca5X, marca6X, marca7X, marca8X, marca9X };
        PointF[] marcaposicionesY = new PointF[] { marca0Y, marca1Y, marca2Y,
        marca3Y, marca4Y, marca5Y, marca6Y, marca7Y, marca8Y, marca9Y };
        PointF[] marcaposicionesZ = new PointF[] { marca0Z, marca1Z, marca2Z,
        marca3Z, marca4Z, marca5Z, marca6Z, marca7Z, marca8Z, marca9Z };
        try
        {
            Aceleraciones.Image = imagen;
            pluma.DashStyle = System.Drawing.Drawing2D.DashStyle.Dot;
            dibujo.Clear(Color.White);
            dibujo.DrawLine(Pens.Gray, ejeA, ejeB);
            dibujo.DrawLine(Pens.Gray, ejeI, ejeD);
            dibujo.DrawLine(pluma, ejeayuda1D, ejeayuda1I);
            dibujo.DrawLine(pluma, ejeayuda2D, ejeayuda2I);
            dibujo.DrawLine(pluma, ejeayuda3D, ejeayuda3I);
            dibujo.DrawLine(pluma, ejeayuda4D, ejeayuda4I);
            dibujo.DrawLine(pluma, ejeayuda5D, ejeayuda5I);
            dibujo.DrawLine(pluma, ejeayuda6D, ejeayuda6I);
            dibujo.DrawLine(pluma, ejeayuda7D, ejeayuda7I);
            dibujo.DrawLine(pluma, ejeayuda8D, ejeayuda8I);
            dibujo.DrawString("400mm", fuente2, Brushes.Black, texto1);
            dibujo.DrawString("30mm", fuente2, Brushes.Black, texto2);
            dibujo.DrawString("200mm", fuente2, Brushes.Black, texto3);
            dibujo.DrawString("100mm", fuente2, Brushes.Black, texto4);
            dibujo.DrawString("-100mm", fuente2, Brushes.Black, texto5);
            dibujo.DrawString("-200mm", fuente2, Brushes.Black, texto6);
            dibujo.DrawString("-300mm", fuente2, Brushes.Black, texto7);
            dibujo.DrawString("-400m", fuente2, Brushes.Black, texto8);
            dibujo.DrawLines(Pens.GreenYellow, marcaposicionesX);
            dibujo.DrawLines(Pens.Orange, marcaposicionesY);
            dibujo.DrawLines(Pens.Red, marcaposicionesZ);
            marca0X.Y = marca1X.Y;
            marca1X.Y = marca2X.Y;
            marca2X.Y = marca3X.Y;
            marca3X.Y = marca4X.Y;
            marca4X.Y = marca5X.Y;
            marca5X.Y = marca6X.Y;
            marca6X.Y = marca7X.Y;
            marca7X.Y = marca8X.Y;
            marca8X.Y = marca9X.Y;
            marca0X.Y = marca1Y.Y;
            marca1Y.Y = marca2Y.Y;
            marca2Y.Y = marca3Y.Y;
            marca3Y.Y = marca4Y.Y;
```

```
                marca4Y.Y = marca5Y.Y;
                marca5Y.Y = marca6Y.Y;
                marca6Y.Y = marca7Y.Y;
                marca7Y.Y = marca8Y.Y;
                marca8Y.Y = marca9Y.Y;
                marca0Z.Y = marca1Z.Y;
                marca1Z.Y = marca2Z.Y;
                marca2Z.Y = marca3Z.Y;
                marca3Z.Y = marca4Z.Y;
                marca4Z.Y = marca5Z.Y;
                marca5Z.Y = marca6Z.Y;
                marca6Z.Y = marca7Z.Y;
                marca7Z.Y = marca8Z.Y;
                marca8Z.Y = marca9Z.Y;
            }
            catch
            {

            }
        }
        banderadibujar = true;
        dibujarpunto(A, B, C);
    }

    /*******************GRÁFICAS DE TRAYECTORIAS EN EL PLANO*******************/

    private void dibujartrayectoriaXY(float A, float B)
    {
        //Hacemos una escala en las trayectorias para ver todo el espacio de trabajo
        presente.X = 202 + (Convert.ToInt32(A) / Convert.ToInt32(escala));
        presente.Y = 160 - (Convert.ToInt32(B) / Convert.ToInt32(escala));
        try
        {
            Trayectorias.Image = imagen3;
            dibujo3.DrawEllipse(Pens.Gray, 202 - radioinutil / escala, 160 –
            radioinutil / escala, 2 * radioinutil / escala, 2 * radioinutil /
            escala);
            dibujo3.DrawEllipse(Pens.Gray, 202 - Convert.ToInt32(d2 + L2 + L3) /
            escala, 160 - Convert.ToInt32(d2 + L2 + L3) / escala, 2 *
            Convert.ToInt32(d2 + L2 + L3) / escala, 2 * Convert.ToInt32(d2 + L2 + L3)
            / escala);
            dibujo3.DrawLine(Pens.DarkSlateGray, anterior, presente);
            dibujo3.FillEllipse(Brushes.Black, presente.X - 3, presente.Y - 3, 6, 6);
            dibujo3.FillEllipse(Brushes.Red, presente.X - 2, presente.Y - 2, 4, 4);
            anterior = presente;
        }
        catch
        {

        }
        banderadibujar = true;
    }

    /*****************************CIERRE DE APLICACIÓN*****************************/

    private void Telerobótica_FormClosed(object sender, FormClosedEventArgs e)
    {
        //Se cierra el puerto serie al cerrar el programa
```

```
    if (PuertoSerial.IsOpen == true)
    {
        PuertoSerial.DiscardInBuffer();
        PuertoSerial.Close();
    }
    if (SerialPort.IsOpen == true)
    {
        SerialPort.DiscardInBuffer();
        SerialPort.Close();
    }
}

/*****************************CONTROL DEL RANGO*****************************/

private void btnRango_Click(object sender, EventArgs e)
{
    //Se actualiza el valor del rango en el botón y en el mapa
    contadorrango++;
    if (contadorrango == 7)
    {
        contadorrango = 0;
        btnRango.Text = "±1g";
        n = 1;
    }
    else if (contadorrango == 1)
    {
        btnRango.Text = "±1.5g";
        n = 1.5F;
    }
    else if (contadorrango == 2)
    {
        btnRango.Text = "±2g";
        n = 2;
    }
    else if (contadorrango == 3)
    {
        btnRango.Text = "±3g";
        n = 3;
    }
    else if (contadorrango == 4)
    {
        btnRango.Text = "±4g";
        n = 4;
    }
    else if (contadorrango == 5)
    {
        btnRango.Text = "±8g";
        n = 8;
    }
    else if (contadorrango == 6)
    {
        btnRango.Text = "±16g";
        n = 16;
    }
    dibujaraceleracion(0, 0, 0);
}

/***********************GRÁFICA POLAR DE ACELERACIONES***********************/
```

```
private void dibujarpunto(float D, float E, float F)
{
    //Dibujo del mapa polar y su punto de localización
    //Se usara un area utilizable del picturebox de 600*600
    //Definimos las variables, puntos, demás...
    try
    {
        MapaPolar.Image = imagen2;
        //Limpiamos la pantalla y colocamos la leyenda del eje z
        dibujo2.Clear(Color.White);
        dibujo2.DrawEllipse(Pens.Gray, 10, 10, 360, 360);
        dibujo2.DrawEllipse(Pens.Gray, 40, 40, 300, 300);
        dibujo2.DrawEllipse(Pens.Gray, 70, 70, 240, 240);
        dibujo2.DrawEllipse(Pens.Gray, 100, 100, 180, 180);
        dibujo2.DrawEllipse(Pens.Gray, 130, 130, 120, 120);
        dibujo2.DrawEllipse(Pens.Gray, 160, 160, 60, 60);
        dibujo2.FillEllipse(Brushes.OrangeRed, 310, 13, 12, 12);
        dibujo2.FillEllipse(Brushes.OrangeRed, 312, 33, 8, 8);
        dibujo2.DrawEllipse(Pens.Purple, 308, 11, 16, 16);
        dibujo2.DrawEllipse(Pens.Purple, 310, 31, 12, 12);
        dibujo2.DrawEllipse(Pens.Purple, 310, 13, 12, 12);
        dibujo2.DrawEllipse(Pens.Purple, 312, 33, 8, 8);
        dibujo2.FillEllipse(Brushes.Black, 314, 17, 4, 4);
        dibujo2.FillEllipse(Brushes.Black, 314, 35, 4, 4);
        dibujo2.DrawString("bajo", fuente, Brushes.Black, leyenda1);
        dibujo2.DrawString("alto", fuente, Brushes.Black, leyenda2);
        //Dibujamos los ejes 'x' y 'x'
        dibujo2.DrawLine(Pens.Black, Xmas, Xmenos);
        dibujo2.DrawLine(Pens.Black, Ymas, Ymenos);
        //Colocamos las escalas
        if (contadorrango == 0)
        {
            dibujo2.DrawString("167mg", fuente, Brushes.Black, punto1);
            dibujo2.DrawString("333mg", fuente, Brushes.Black, punto2);
            dibujo2.DrawString("500mg", fuente, Brushes.Black, punto3);
            dibujo2.DrawString("667mg", fuente, Brushes.Black, punto4);
            dibujo2.DrawString("833mg", fuente, Brushes.Black, punto5);
            dibujo2.DrawString("1.00g", fuente, Brushes.Black, punto6);
        }
        else if (contadorrango == 1)
        {
            dibujo2.DrawString("250mg", fuente, Brushes.Black, punto1);
            dibujo2.DrawString("500mg", fuente, Brushes.Black, punto2);
            dibujo2.DrawString("750mg", fuente, Brushes.Black, punto3);
            dibujo2.DrawString("1.00g", fuente, Brushes.Black, punto4);
            dibujo2.DrawString("1.25g", fuente, Brushes.Black, punto5);
            dibujo2.DrawString("1.50g", fuente, Brushes.Black, punto6);
        }
        else if (contadorrango == 2)
        {
            dibujo2.DrawString("333mg", fuente, Brushes.Black, punto1);
            dibujo2.DrawString("667mg", fuente, Brushes.Black, punto2);
            dibujo2.DrawString("1.00g", fuente, Brushes.Black, punto3);
            dibujo2.DrawString("1.33g", fuente, Brushes.Black, punto4);
            dibujo2.DrawString("1.67g", fuente, Brushes.Black, punto5);
            dibujo2.DrawString("2.00g", fuente, Brushes.Black, punto6);
        }
```

```
else if (contadorrango == 3)
{
    dibujo2.DrawString("500mg", fuente, Brushes.Black, punto1);
    dibujo2.DrawString("1.00g", fuente, Brushes.Black, punto2);
    dibujo2.DrawString("1.50g", fuente, Brushes.Black, punto3);
    dibujo2.DrawString("2.00g", fuente, Brushes.Black, punto4);
    dibujo2.DrawString("2.50g", fuente, Brushes.Black, punto5);
    dibujo2.DrawString("3.00g", fuente, Brushes.Black, punto6);
}
else if (contadorrango == 4)
{
    dibujo2.DrawString("667mg", fuente, Brushes.Black, punto1);
    dibujo2.DrawString("1.33g", fuente, Brushes.Black, punto2);
    dibujo2.DrawString("2.00g", fuente, Brushes.Black, punto3);
    dibujo2.DrawString("2.67g", fuente, Brushes.Black, punto4);
    dibujo2.DrawString("3.33g", fuente, Brushes.Black, punto5);
    dibujo2.DrawString("4.00g", fuente, Brushes.Black, punto6);
}
else if (contadorrango == 5)
{
    dibujo2.DrawString("1.33g", fuente, Brushes.Black, punto1);
    dibujo2.DrawString("2.66g", fuente, Brushes.Black, punto2);
    dibujo2.DrawString("4.00g", fuente, Brushes.Black, punto3);
    dibujo2.DrawString("5.33g", fuente, Brushes.Black, punto4);
    dibujo2.DrawString("6.67g", fuente, Brushes.Black, punto5);
    dibujo2.DrawString("8.00g", fuente, Brushes.Black, punto6);
}
else if (contadorrango == 6)
{
    dibujo2.DrawString("2.66g", fuente, Brushes.Black, punto1);
    dibujo2.DrawString("5.33g", fuente, Brushes.Black, punto2);
    dibujo2.DrawString("8.00g", fuente, Brushes.Black, punto3);
    dibujo2.DrawString("10.7g", fuente, Brushes.Black, punto4);
    dibujo2.DrawString("13.3g", fuente, Brushes.Black, punto5);
    dibujo2.DrawString("16.0g", fuente, Brushes.Black, punto6);
}
//Transformamos los valores a aceleración real
X = 180 * D / n;
Y = 180 * E / n;
Z = 180 * F / n;
//Obtenemos los valores de las coordenadas cilindricas, donde para z se
usa una función exponencial paara dibujarla
if (Zsinescala >= 0)
{
    altura = 10 + ((20) / (Math.Exp(Zsinescala * 10)));
}
else
{
    altura = 10 + ((20) / (Math.Exp(-Zsinescala * 10)));
}
//Obtenemos las coordenadas en pixeles
Xell1 = 190 + Convert.ToInt32(X) - Convert.ToInt32(altura / 2D);
Yell1 = 190 - Convert.ToInt32(Y) - Convert.ToInt32(altura / 2D);
Xell2 = 188 + Convert.ToInt32(X) - Convert.ToInt32(altura / 2D);
Yell2 = 188 - Convert.ToInt32(Y) - Convert.ToInt32(altura / 2D);
Xell3 = 188 + Convert.ToInt32(X);
Yell3 = 188 - Convert.ToInt32(Y);
radio = Convert.ToInt32(altura);
```

```
            radioAmp = Convert.ToInt32(altura) + 4;
            if (X != 0 || Y != 0 || Z != 0)
            {
                //Dibujamos las figuras de la localización
                dibujo2.FillEllipse(Brushes.OrangeRed, Xell1, Yell1, radio, radio);
                dibujo2.DrawEllipse(Pens.Purple, Xell1, Yell1, radio, radio);
                dibujo2.DrawEllipse(Pens.Purple, Xell2, Yell2, radioAmp, radioAmp);
                dibujo2.FillEllipse(Brushes.Black, Xell3, Yell3, 4, 4);
            }
        }
        catch
        {

        }
    }

    private void Trayectorias_MouseMove(object sender, MouseEventArgs e)
    {
        //Obtenemos las coordenadas en pixeles del mapa polar, como una interrupción
        o un evento
        txtGridX.Text = Convert.ToString((e.X - 202) * escala);
        txtGridY.Text = Convert.ToString((150 - e.Y) * escala);
    }

    /**********************TRANSFORMACIÓN A ACELERACIONES**********************/

    private float datosrealesaceleraciones(int valor)
    {
        //Convertimos los datos binarios en aceleraciones reales en unidades de g
        if (valor >= -8192 && valor <= 8191)
        {
            salida = valor * n / 8192;
        }
        else
        {
            MessageBox.Show("Valor no dentro de los rangos posibles");
            salida = 0;
        }
        return salida;
    }

    /********************LECTURA DEL PUERTO SERIE VIA USB/XBEE********************/

    public void leerdatosseriales(object sender, SerialDataReceivedEventArgs e)
    {
        //Crear un thread standard
        Thread th1 = new Thread(new ThreadStart(threadtexto1));
        Thread th2 = new Thread(new ThreadStart(threadtexto2));
        if (PuertoSerial.IsOpen == true)
        {
            //Comunicación alámbrica vía USB
            /*
            //El primer dato es: $data$
            iniciocadena = PuertoSerial.ReadLine();
            if (String.Equals(iniciocadena, "$data$\r") == true)
            {
                try
                {
```

```
cadenaleidax = PuertoSerial.ReadLine();
cadenaleiday = PuertoSerial.ReadLine();
cadenaleidaz = PuertoSerial.ReadLine();
cadenaleidatiempo = PuertoSerial.ReadLine();
//Restamos la gravedad en 'z' y calibramos cada uno de los ejes
aceleracionX =
datosrealesaceleraciones(Convert.ToInt32(cadenaleidax)) + 0.017F;
aceleracionY =
datosrealesaceleraciones(Convert.ToInt32(cadenaleiday)) + 0.031F;
aceleracionZ =
datosrealesaceleraciones(Convert.ToInt32(cadenaleidaz)) - 1.035F;
if (aceleracionX <= 0.050F && aceleracionX >= -0.050F &&
aceleracionY <= 0.050F && aceleracionY >= -0.050F && aceleracionZ
<= 0.100F && aceleracionZ >= -0.100F)
{
    aceleracionX = 0.0F;
    aceleracionY = 0.0F;
    aceleracionZ = 0.0F;
    velocidadinicialX = 0;
    velocidadinicialY = 0;
    velocidadinicialZ = 0;
    velocidadRectanteriorX = 0;
    velocidadRectanteriorY = 0;
    velocidadRectanteriorZ = 0;
}
tiempoensegundos = Convert.ToSingle(cadenaleidatiempo) / 1000F;
if (tipodeintegracion == 0)
{
    AceleracionRectangularX = aceleracionX * 9.80665F;
    AceleracionRectangularY = aceleracionY * 9.80665F;
    AceleracionRectangularZ = aceleracionZ * 9.80665F;
    posicionX = IntegracionTrapezoidalX(AceleracionRectangularX,
    tiempoensegundos);
    posicionY = IntegracionTrapezoidalY(AceleracionRectangularY,
    tiempoensegundos);
    posicionZ = IntegracionTrapezoidalZ(AceleracionRectangularZ,
    tiempoensegundos);
    posicionX *= escalaRealPosicion;
    posicionY *= escalaRealPosicion;
    posicionZ *= escalaRealPosicion;
    //angulosScorbot = CinematicaInversa(posicionX, posicionY,
    posicionZ);
    angulosScorbot = CinematicaInversaSimplificada(posicionX,
    posicionY, posicionZ);
    th2.Start();
}
if (tipodeintegracion == 1)
{
    vectorAceleracionX[contadora17] = aceleracionX * 9.80665F;
    vectorAceleracionY[contadora17] = aceleracionY * 9.80665F;
    vectorAceleracionZ[contadora17] = aceleracionZ * 9.80665F;
    vectorTiempos[contadora17] = tiempoensegundos;
    if (contadora17 == 16)
    {
        ua = vectorTiempos[0] + vectorTiempos[1] +
        vectorTiempos[2] + vectorTiempos[3];
        ub = vectorTiempos[4] + vectorTiempos[5] +
        vectorTiempos[6] + vectorTiempos[7];
```

```
                    uc = vectorTiempos[8] + vectorTiempos[9] +
                    vectorTiempos[10] + vectorTiempos[11];
                    ud = vectorTiempos[12] + vectorTiempos[13] +
                    vectorTiempos[14] + vectorTiempos[15];
                    ue = ua + ub + uc + ud;
                    posicionX = IntegracionBoolesRuleX(vectorAceleracionX,
                    ua, ub, uc, ud, ue);
                    posicionY = IntegracionBoolesRuleY(vectorAceleracionY,
                    ua, ub, uc, ud, ue);
                    posicionZ = IntegracionBoolesRuleZ(vectorAceleracionZ,
                    ua, ub, uc, ud, ue);
                    posicionX *= escalaRealPosicion;
                    posicionY *= escalaRealPosicion;
                    posicionZ *= escalaRealPosicion;
                    EspacioDeTrabajo = Math.Pow(posicionX, 2D) +
                    Math.Pow(posicionY, 2D) + Math.Pow((posicionZ- d1 - L1 +
                    L4), 2D);
                    if (EspacioDeTrabajo <= Math.Pow((d2 + L2 + L3), 2D))
                    {
                        DentroDelEspacio = true;
                    }
                    else
                    {
                        DentroDelEspacio = false;
                    }
                    //angulosScorbot = CinematicaInversa(posicionX,
                    posicionY, posicionZ);
                    angulosScorbot = CinematicaInversaSimplificada(posicionX,
                    posicionY, posicionZ);
                    th2.Start();
                    contadora17 = -1;
                }
                contadora17++;
            }
        }
        catch
        {
            aceleracionX = 0;
            aceleracionY = 0;
            aceleracionZ = 0;
        }
    }
    */
    //Comunicación inalámbrica vía XBEE
    //Cadena: 7E 00 16 10 01 00 00 00 00 00 00 00 00 FF FE 00 00 XX XX XX XX
    XX XX XX XX 34
    byte dat;
    byte checksum = 0x00; ;
    //Start delimiter = 0x07
    do { dat = (byte)PuertoSerial.ReadByte(); }
    while (dat != (byte)0x7E);
    XBEE[0] = dat;
    bytesatport = PuertoSerial.BytesToRead;
    //Verificamos que haya la cantidad mínima de bytes en el puerto para leer
    datos completos de aceleración
    if (bytesatport >= 22)
    {
        PuertoSerial.Read(XBEE, 1, 14);
```

```
if (XBEE[0] == (byte)0x7E && XBEE[1] == (byte)0x00 && XBEE[2] ==
    (byte)0x14 && XBEE[3] == (byte)0x90 &&
    XBEE[4] == (byte)0x00 && XBEE[5] == (byte)0x13 && XBEE[6] ==
    (byte)0xA2 && XBEE[7] == (byte)0x00 &&
    XBEE[8] == (byte)0x40 && XBEE[9] == (byte)0x3C && XBEE[10] ==
    (byte)0x8C && XBEE[11] == (byte)0xB1 &&
    XBEE[12] == (byte)0x21 && XBEE[13] == (byte)0xAE && XBEE[14] ==
    (byte)0x01)
{
    //Add all bytes (include checksum, but not the delimiter and
    length). If the checksum is correct, the sum will equal 0xFF
    PuertoSerial.Read(XBEE, 15, 9);
    //Verificamos la integridad de los datos obtenidos
    for (int icheck = 3; icheck < 24; icheck++)
    {
        checksum += XBEE[icheck];
    }
    if (checksum == 0xFF)
    {
        aceleracionX = datosrealesaceleraciones
        (transformarbits(XBEE[16], XBEE[17])) + 0.017F;
        aceleracionY = datosrealesaceleraciones
        (transformarbits(XBEE[18], XBEE[19])) + 0.031F;
        aceleracionZ = datosrealesaceleraciones
        (transformarbits(XBEE[20], XBEE[21])) - 1.035F;
        cadenaleidatiempo = Convert.ToString(XBEE[22]);
        tiempoensegundos = Convert.ToSingle(cadenaleidatiempo) /
        1000F;
        if (aceleracionX <= 0.050F && aceleracionX >= -0.050F &&
        aceleracionY <= 0.050F && aceleracionY >= -0.050F &&
        aceleracionZ <= 0.100F && aceleracionZ >= -0.100F)
        {
            aceleracionX = 0.0F;
            aceleracionY = 0.0F;
            aceleracionZ = 0.0F;
            velocidadinicialX = 0;
            velocidadinicialY = 0;
            velocidadinicialZ = 0;
            velocidadRectanteriorX = 0;
            velocidadRectanteriorY = 0;
            velocidadRectanteriorZ = 0;
        }
        try
        {
            if (tipodeintegracion == 0)
            {
                AceleracionRectangularX = aceleracionX * 9.80665F;
                AceleracionRectangularY = aceleracionY * 9.80665F;
                AceleracionRectangularZ = aceleracionZ * 9.80665F;
                posicionX = IntegracionTrapezoidalX
                (AceleracionRectangularX, tiempoensegundos);
                posicionY = IntegracionTrapezoidalY
                (AceleracionRectangularY, tiempoensegundos);
                posicionZ = IntegracionTrapezoidalZ
                (AceleracionRectangularZ, tiempoensegundos);
                posicionX *= escalaRealPosicion;
                posicionY *= escalaRealPosicion;
                posicionZ *= escalaRealPosicion;
```

```
            //Inicio de Guardar en Excel
            arregloExcel[contadorExcel, 0] =
            AceleracionRectangularX;
            arregloExcel[contadorExcel, 1] =
            AceleracionRectangularY;
            arregloExcel[contadorExcel, 2] =
            AceleracionRectangularZ;
            arregloExcel[contadorExcel, 3] = tiempoensegundos;
            arregloExcel[contadorExcel, 4] = posicionX;
            arregloExcel[contadorExcel, 5] = posicionY;
            arregloExcel[contadorExcel, 6] = posicionZ;
            contadorExcel++;
            if (contadorExcel == 19)
            {
                //escribirAExcel(arregloExcel);
                contadorExcel = 0;
            }
            //Fin de Guardar en Excel

            EspacioDeTrabajo = Math.Pow(posicionX, 2D) +
            Math.Pow(posicionY, 2D) + Math.Pow((posicionZ - d1 –
            L1 + L4), 2D);
            if (EspacioDeTrabajo <= Math.Pow((d2 + L2 + L3), 2D))
            {
                DentroDelEspacio = true;
                btnEspacioDeTrabajo.BackColor = Color.LawnGreen;
            }
            else
            {
                DentroDelEspacio = false;
                btnEspacioDeTrabajo.BackColor = Color.Red;
            }
            //angulosScorbot = CinematicaInversa(posicionX,
            posicionY, posicionZ);
            angulosScorbot = CinematicaInversaSimplificada
            (posicionX, posicionY, posicionZ);
            th2.Start();
        }
        if (tipodeintegracion == 1)
        {
            vectorAceleracionX[contadora17] = aceleracionX *
            9.80665F;
            vectorAceleracionY[contadora17] = aceleracionY *
            9.80665F;
            vectorAceleracionZ[contadora17] = aceleracionZ *
            9.80665F;
            vectorTiempos[contadora17] = tiempoensegundos;
            if (contadora17 == 16)
            {
                ua = vectorTiempos[0] + vectorTiempos[1] +
                vectorTiempos[2] + vectorTiempos[3];
                ub = vectorTiempos[4] + vectorTiempos[5] +
                vectorTiempos[6] + vectorTiempos[7];
                uc = vectorTiempos[8] + vectorTiempos[9] +
                vectorTiempos[10] + vectorTiempos[11];
                ud = vectorTiempos[12] + vectorTiempos[13] +
                vectorTiempos[14] + vectorTiempos[15];
                ue = ua + ub + uc + ud;
```

```
                                posicionX = IntegracionBoolesRuleX
                                (vectorAceleracionX, ua, ub, uc, ud, ue);
                                posicionY = IntegracionBoolesRuleY
                                (vectorAceleracionY, ua, ub, uc, ud, ue);
                                posicionZ = IntegracionBoolesRuleZ
                                (vectorAceleracionZ, ua, ub, uc, ud, ue);
                                posicionX *= escalaRealPosicion;
                                posicionY *= escalaRealPosicion;
                                posicionZ *= escalaRealPosicion;
                                EspacioDeTrabajo = Math.Pow(posicionX, 2D) +
                                Math.Pow(posicionY, 2D) + Math.Pow((posicionZ –
                                d1 - L1 + L4), 2D);
                                if (EspacioDeTrabajo <= Math.Pow((d2 + L2 + L3),
                                    2D))
                                {
                                    DentroDelEspacio = true;
                                    btnEspacioDeTrabajo.BackColor =
                                    Color.LawnGreen;
                                }
                                else
                                {
                                    DentroDelEspacio = false;
                                    btnEspacioDeTrabajo.BackColor = Color.Red;
                                }
                                //angulosScorbot = CinematicaInversa(posicionX,
                                posicionY, posicionZ);
                                angulosScorbot = CinematicaInversaSimplificada
                                (posicionX, posicionY, posicionZ);
                                th2.Start();
                                contadora17 = -1;
                            }
                            contadora17++;
                        }
                    }
                    catch
                    {

                    }

                }
                else
                {
                    //MessageBox.Show("XBEE\r\nDatos no recibidos
                    correctamente");
                }
            }
        }
    }
}
Xsinescala = Convert.ToDouble(aceleracionX);
Ysinescala = Convert.ToDouble(aceleracionY);
Zsinescala = Convert.ToDouble(aceleracionZ);
radioesf = Math.Sqrt(Xsinescala * Xsinescala + Ysinescala * Ysinescala +
Zsinescala * Zsinescala);
anguloinclinacion = Math.Atan2(Ysinescala, Xsinescala) * 180 / Math.PI;
anguloazimutal = Math.Asin(Zsinescala / radioesf) * 180 / Math.PI;
th1.Start();
//Ilumina el indicador de que se están transfiriendo datos
if (indicador == 1)
```

```
    {
        Recibiendo.BackColor = Color.Gold;
    }
    else if (indicador == 2)
    {
        Recibiendo.BackColor = Color.Turquoise;
        indicador = 0;
    }
    indicador++;
    if (banderadibujar == true)
    {
        banderadibujar = false;
        if (numerografica == 1)
        {
            dibujaraceleracion(aceleracionX, aceleracionY, aceleracionZ);
            dibujartrayectoriaXY(posicionX, posicionY);
        }
        else if (numerografica == 0)
        {
            dibujarposiciones(posicionX, posicionY, posicionZ);
        }
    }
}

/*****************************REACOMODO DE BITS*****************************/

private int transformarbits(int entrada1, int entrada2)
{
    int dato = 0;
    entrada2 <<= 24;
    entrada2 >>= 16;
    dato = entrada1 | entrada2;
    return dato;
}

/*********************LECTURA DEL PUERTO SERIE DEL SCORBOT*********************/

private void leerSCORBOT(object sender, SerialDataReceivedEventArgs e)
{
    //Creamos un Thread para enviar informaciónn visual de los encoders
    Thread th5 = new Thread(new ThreadStart(threadtexto3));
    //Leer el puerto serial
    if (SerialPort.IsOpen == true)
    {
        cadenaleidaSCORBOT = SerialPort.ReadLine();
        //Enviar cuando llegue del Arduino Mega: "%%"
        if (String.Equals(cadenaleidaSCORBOT, "%%\r") == true)
        {
            try
            {

                if (leerlinea <= txtLineasTrayectoria.Lines.Length - 1)
                {

                    SerialPort.Write
                    (txtLineasTrayectoria.Lines.ElementAt(leerlinea));
                    leerlinea++;
                    if (txtLineasTrayectoria.Lines.Length == leerlinea)
```

```
                {
                    SerialPort.Write
                    (txtLineasTrayectoria.Lines.ElementAt(leerlinea - 1));
                }
                else
                {
                    SerialPort.Write
                    (txtLineasTrayectoria.Lines.ElementAt(leerlinea));
                }

            }
            else
            {
                Iniciar.BackColor = Color.Gray;
            }
        }
        catch
        {

        }
        finally
        {
            cadenaleidaSCORBOT = "";
        }
    }
    //Leer encoders y microswitches cuando llegue del Arduino Mega: "@@"
    if (String.Equals(cadenaleidaSCORBOT, "@@\r") == true)
    {
        //Microswitch presioando: 1, no presionado: 0. [6] ... [9]
        try
        {
            lineadeScorbot[0] = SerialPort.ReadLine();
            lineadeScorbot[1] = SerialPort.ReadLine();
            lineadeScorbot[2] = SerialPort.ReadLine();
            lineadeScorbot[3] = SerialPort.ReadLine();
            lineadeScorbot[4] = SerialPort.ReadLine();
            lineadeScorbot[5] = SerialPort.ReadLine();
            lineadeScorbot[6] = SerialPort.ReadLine();
            lineadeScorbot[7] = SerialPort.ReadLine();
            lineadeScorbot[8] = SerialPort.ReadLine();
            lineadeScorbot[9] = SerialPort.ReadLine();
        }
        catch
        {

        }
        finally
        {
            cadenaleidaSCORBOT = "";
        }
        //Llamamos al Thread, imprime valores de encoder
        th5.Start();
        if (String.Equals(lineadeScorbot[6], "0\r") == true)
        {
            btnMicroswitch1.BackColor = Color.LawnGreen;
        }
        else
        {
```

```
                btnMicroswitch1.BackColor = Color.Gray;
            }
            if (String.Equals(lineadeScorbot[7], "0\r") == true)
            {
                btnMicroswitch2.BackColor = Color.LawnGreen;
            }
            else
            {
                btnMicroswitch2.BackColor = Color.Gray;
            }
            if (String.Equals(lineadeScorbot[8], "0\r") == true)
            {
                btnMicroswitch3.BackColor = Color.LawnGreen;
            }
            else
            {
                btnMicroswitch3.BackColor = Color.Gray;
            }
            if (String.Equals(lineadeScorbot[9], "0\r") == true)
            {
                btnMicroswitch4.BackColor = Color.LawnGreen;
            }
            else
            {
                btnMicroswitch4.BackColor = Color.Gray;
            }
            cadenaleidaSCORBOT = "";
        }
        else
        {

        }
    }
}

/************************MUESTREO DE DATOS EN PANTALLA************************/

private void threadtexto1()
{
    //Verificar si estamos en el thread correcto, y si no invocarlo
    if (InvokeRequired)
    {
        MethodInvoker metodo1 = new MethodInvoker(threadtexto1);
        try
        {
            Invoke(metodo1);
        }
        catch
        {

        }
        return;
    }
    //Colocamos en la leyenda los valores de las aceleraciones
    txtEjeX.Text = aceleracionX.ToString("N6") + " [g]";
    txtEjeY.Text = aceleracionY.ToString("N6") + " [g]";
    txtEjeZ.Text = aceleracionZ.ToString("N6") + " [g]";
    txtGridAltura.Text = altura.ToString("N6");
```

```
    txtdistancia.Text = radioesf.ToString("N3") + " [g]";
    txtangulo.Text = anguloinclinacion.ToString("N3") + "°";
    txtangulo2.Text = anguloazimutal.ToString("N3") + "°";
    txtOtro.Text = tiempoensegundos.ToString("N3") + " [s]";
}

private void threadtexto2()
{
    if (InvokeRequired)
    {
        MethodInvoker metodo2 = new MethodInvoker(threadtexto2);
        try
        {
            Invoke(metodo2);
        }
        catch
        {

        }
        return;
    }
    //Colocamos en la leyenda los valores de las posiciones
    txtAceX.Text = posicionX.ToString("N6");
    txtAceY.Text = posicionY.ToString("N6");
    txtAceZ.Text = posicionZ.ToString("N6");
    txtAngq1.Text = angulosScorbot[0].ToString("N4");
    txtAngq2.Text = angulosScorbot[1].ToString("N4");
    txtAngq3.Text = angulosScorbot[2].ToString("N4");
    txtAngq4.Text = angulosScorbot[3].ToString("N4");
    txtAngq5.Text = "0.0000";
    // pulsos encoder = gearratio * (angulo - angulo inicial)
    try
    {
        PulsosEncoders[0] = Convert.ToInt32((gearratio[0] * (angulosScorbot[0] -
        angulosinicial[0])));
        PulsosEncoders[1] = Convert.ToInt32((gearratio[1] * (angulosScorbot[1] -
        angulosinicial[1])) * -1);
        PulsosEncoders[2] = Convert.ToInt32(gearratio[2] * (angulosScorbot[2] -
        angulosinicial[2]) - PulsosEncoders[1]);
        PulsosEncoders[3] = Convert.ToInt32(gearratio[3] * (angulosScorbot[3] -
        angulosinicial[3]) + PulsosEncoders[2] * 0.433F);
        PulsosEncoders[4] = Convert.ToInt32((gearratio[4] * (angulosScorbot[3] -
        angulosinicial[3])) * -1 + PulsosEncoders[1]);
    }
    catch
    {

    }
    //cadenaenviada = "#" + PulsosEncoders[0].ToString("N0") + "," +
    PulsosEncoders[1].ToString("N0") + "," + PulsosEncoders[2].ToString("N0") +
    "," + PulsosEncoders[3].ToString("N0") + "," +
    PulsosEncoders[4].ToString("N0") + "," + "0" + ",";
    cadenaenviada = "#" + String.Format("{0:}", PulsosEncoders[0]) + "," +
    String.Format("{0:}", PulsosEncoders[1]) + "," + String.Format("{0:}",
    PulsosEncoders[2]) + "," + String.Format("{0:}", PulsosEncoders[4]) + "," +
    String.Format("{0:}", PulsosEncoders[3]) + "," + "0" + ",";
    if (Math.Pow(posicionX, 2D) + Math.Pow(posicionY, 2D) >=
        Math.Pow(radioinutil, 2D) && DentroDelEspacio == true)
```

```
    {
        try
        {
            if (txtLineasTrayectoria.Lines.ElementAt
                (txtLineasTrayectoria.Lines.Length - 2) != cadenaenviada)
            {
                txtLineasTrayectoria.Text += cadenaenviada + "\r\n";
            }
        }
        catch
        {
            txtLineasTrayectoria.Text += cadenaenviada + "\r\n";
        }
    }
    else if (Math.Pow(XManual, 2D) + Math.Pow(YManual, 2D) >=
             Math.Pow(radioinutil, 2D) && DentroDelEspacio == true)
    {
        txtLineasTrayectoria.Text += cadenaenviada + "\r\n";
    }
    txtLineasTrayectoria.SelectionStart = txtLineasTrayectoria.Text.Length;
    txtLineasTrayectoria.ScrollToCaret();
    txtCadenaEnviada.Text = cadenaenviada;
}

private void threadtexto3()
{
    //Verificar si estamos en el thread correcto, y si no invocarlo
    if (InvokeRequired)
    {
        MethodInvoker metodo1 = new MethodInvoker(threadtexto3);
        try
        {
            Invoke(metodo1);
        }
        catch
        {

        }
        return;
    }
    //Colocamos los valores numéricos de los pulsos de encoder
    txtEncoder1.Text = lineadeScorbot[0];
    txtEncoder2.Text = lineadeScorbot[1];
    txtEncoder3.Text = lineadeScorbot[2];
    txtEncoder4.Text = lineadeScorbot[3] +  ", " + lineadeScorbot[4];
    txtEncoder5.Text = lineadeScorbot[5];
}

/*************************FUNCIONES DE INTEGRACIÓN*************************/

private float IntegracionBoolesRuleX(float[] aceleracion, float a, float b, float
c, float d, float e)
{
    //Realizamos los cálculos de la integración doble
    //Debemos mandar 17 datos de aceleración, así como los límites en el tiempo
    'a' y 'b'
    kx[0] = a / 90F;
    kx[1] = b / 90F;
```

```
    kx[2] = c / 90F;
    kx[3] = d / 90F;
    kx[4] = e / 90F;
    velocidad[0] = velocidadinicialX;
    velocidad[1] = (kx[0] * (7 * aceleracion[0] + 32 * aceleracion[1] + 12 *
    aceleracion[2] + 32 * aceleracion[3] + 7 * aceleracion[4])) + velocidad[0];
    velocidad[2] = (kx[1] * (7 * aceleracion[4] + 32 * aceleracion[5] + 12 *
    aceleracion[6] + 32 * aceleracion[7] + 7 * aceleracion[8])) + velocidad[1];
    velocidad[3] = (kx[2] * (7 * aceleracion[8] + 32 * aceleracion[9] + 12 *
    aceleracion[10] + 32 * aceleracion[11] + 7 * aceleracion[12])) +
    velocidad[2];
    velocidad[4] = (kx[3] * (7 * aceleracion[12] + 32 * aceleracion[13] + 12 *
    aceleracion[14] + 32 * aceleracion[15] + 7 * aceleracion[16])) +
    velocidad[3];
    posicionXX += (kx[4] * (7 * velocidadinicialX + 32 * velocidad[1] + 12 *
    velocidad[2] + 32 * velocidad[3] + 7 * velocidad[4]));
    velocidadinicialX = velocidad[4];
    return posicionXX;
}

private float IntegracionBoolesRuleY(float[] aceleracion, float a, float b, float
c, float d, float e)
{
    //Realizamos los cálculos de la integración doble
    //Debemos mandar 17 datos de aceleración, así como los límites en el tiempo
    'a' y 'b'
    ky[0] = a / 90F;
    ky[1] = b / 90F;
    ky[2] = c / 90F;
    ky[3] = d / 90F;
    ky[4] = e / 90F;
    velocidad[0] = velocidadinicialY;
    velocidad[1] = (ky[0] * (7 * aceleracion[0] + 32 * aceleracion[1] + 12 *
    aceleracion[2] + 32 * aceleracion[3] + 7 * aceleracion[4])) + velocidad[0];
    velocidad[2] = (ky[1] * (7 * aceleracion[4] + 32 * aceleracion[5] + 12 *
    aceleracion[6] + 32 * aceleracion[7] + 7 * aceleracion[8])) + velocidad[1];
    velocidad[3] = (ky[2] * (7 * aceleracion[8] + 32 * aceleracion[9] + 12 *
    aceleracion[10] + 32 * aceleracion[11] + 7 * aceleracion[12])) +
    velocidad[2];
    velocidad[4] = (ky[3] * (7 * aceleracion[12] + 32 * aceleracion[13] + 12 *
    aceleracion[14] + 32 * aceleracion[15] + 7 * aceleracion[16])) +
    velocidad[3];
    posicionYY += (ky[4] * (7 * velocidadinicialY + 32 * velocidad[1] + 12 *
    velocidad[2] + 32 * velocidad[3] + 7 * velocidad[4]));
    velocidadinicialY = velocidad[4];
    return posicionYY;
}

private float IntegracionBoolesRuleZ(float[] aceleracion, float a, float b, float
c, float d, float e)
{
    //Realizamos los cálculos de la integración doble
    //Debemos mandar 17 datos de aceleración, así como los límites en el tiempo
    'a' y 'b'
    kz[0] = a / 90F;
    kz[1] = b / 90F;
    kz[2] = c / 90F;
    kz[3] = d / 90F;
```

```
    kz[4] = e / 90F;
    velocidad[0] = velocidadinicialZ;
    velocidad[1] = (kz[0] * (7 * aceleracion[0] + 32 * aceleracion[1] + 12 *
    aceleracion[2] + 32 * aceleracion[3] + 7 * aceleracion[4])) + velocidad[0];
    velocidad[2] = (kz[1] * (7 * aceleracion[4] + 32 * aceleracion[5] + 12 *
    aceleracion[6] + 32 * aceleracion[7] + 7 * aceleracion[8])) + velocidad[1];
    velocidad[3] = (kz[2] * (7 * aceleracion[8] + 32 * aceleracion[9] + 12 *
    aceleracion[10] + 32 * aceleracion[11] + 7 * aceleracion[12])) +
    velocidad[2];
    velocidad[4] = (kz[3] * (7 * aceleracion[12] + 32 * aceleracion[13] + 12 *
    aceleracion[14] + 32 * aceleracion[15] + 7 * aceleracion[16])) +
    velocidad[3];
    posicionZZ += (kz[4] * (7 * velocidadinicialZ + 32 * velocidad[1] + 12 *
    velocidad[2] + 32 * velocidad[3] + 7 * velocidad[4]));
    velocidadinicialZ = velocidad[4];
    return posicionZZ;
}

private float IntegracionTrapezoidalX(float aceleracion, float tiempo)
{
    posicionRectangularX += velocidadRectanteriorX * tiempo + 0.5F * tiempo *
    tiempo * ((aceleracion + aceleracionRectanteriorX) / 2F);
    velocidadRectanteriorX += tiempo * ((aceleracion + aceleracionRectanteriorX)
    / 2F);
    aceleracionRectanteriorX = aceleracion;
    return posicionRectangularX;
}

private float IntegracionTrapezoidalY(float aceleracion, float tiempo)
{
    posicionRectangularY += velocidadRectanteriorY * tiempo + 0.5F * tiempo *
    tiempo * ((aceleracion + aceleracionRectanteriorY) / 2F);
    velocidadRectanteriorY += tiempo * ((aceleracion + aceleracionRectanteriorY)
    / 2F);
    aceleracionRectanteriorY = aceleracion;
    return posicionRectangularY;
}

private float IntegracionTrapezoidalZ(float aceleracion, float tiempo)
{
    posicionRectangularZ += velocidadRectanteriorZ * tiempo + 0.5F * tiempo *
    tiempo * ((aceleracion + aceleracionRectanteriorZ) / 2F);
    velocidadRectanteriorZ += tiempo * ((aceleracion + aceleracionRectanteriorZ)
    / 2F);
    aceleracionRectanteriorZ = aceleracion;
    return posicionRectangularZ;
}

/**************************CINEMÁTICA INVERSA**************************/

private double[] CinematicaInversa(double Px, double Py, double Pz)
{
    //Se reciben los datos de entrada Px, Py y Pz que con las posiciones deseadas
    //La salida son los ángulos q1, q2, q3 y q4 en radianes
    //Falta agregar el +-
    /*
    q[0] = Math.Atan2(Py, Px) - Math.Atan2(L1, Math.Sqrt(Math.Pow(Px, 2D) +
    Math.Pow(Py, 2D) + Math.Pow(Pz, 2D)));
```

```
        q[2] = Math.Acos((Math.Pow((Px * Math.Cos(q[0]) + Py * Math.Sin(q[0]) - d2),
        2D) + Math.Pow((d1 - Pz + L4), 2D) - Math.Pow(L2, 2D) - Math.Pow(L3, 2D)) /
        (2D * L2 * L3));
        q[1] = 2 * Math.Atan((d1 - Pz + L4 + Math.Sqrt(Math.Pow((Px * Math.Cos(q[0])
        + Py * Math.Sin(q[0]) - d2), 2D) + Math.Pow((d1 - Pz + L4), 2D) +
        Math.Pow((L2 + L3 * Math.Cos(q[2])), 2D))) / (Px * Math.Cos(q[0]) + Py *
        Math.Sin(q[0]) - d2 + L2 + L3 * Math.Cos(q[2])));
        q[3] = (Math.PI / 2D) - q[1] - q[2];
        q[0] = q[0] * 180 / Math.PI;
        q[1] = q[1] * 180 / Math.PI;
        q[2] = q[2] * 180 / Math.PI;
        q[3] = q[3] * 180 / Math.PI;
        */
        q[0] = Math.Atan2(Py, Px) - Math.Atan2(L1, Math.Sqrt(Math.Pow(Px, 2D) +
        Math.Pow(Py, 2D) + Math.Pow(Pz, 2D)));
        FdeCosaTan = (Math.Pow((Px * Math.Cos(q[0]) + Py * Math.Sin(q[0]) - d2), 2D)
        + Math.Pow((d1 - Pz + L4), 2D) - Math.Pow(L2, 2D) - Math.Pow(L3, 2D)) / (2D *
        L2 * L3);
        q[2] = Math.Atan(Math.Sqrt(Math.Pow((1/FdeCosaTan), 2D) - 1D));
        q[1] = 2 * Math.Atan((d1 - Pz + L4 + Math.Sqrt(Math.Pow((Px * Math.Cos(q[0])
        + Py * Math.Sin(q[0]) - d2), 2D) + Math.Pow((d1 - Pz + L4), 2D) +
        Math.Pow((L2 + L3 * Math.Cos(q[2])), 2D))) / (Px * Math.Cos(q[0]) + Py *
        Math.Sin(q[0]) - d2 + L2 + L3 * Math.Cos(q[2])));
        q[3] = (Math.PI / 2D) - q[1] - q[2];
        q[0] = q[0] * 180 / Math.PI;
        q[1] = q[1] * 180 / Math.PI;
        q[2] = q[2] * 180 / Math.PI;
        q[3] = q[3] * 180 / Math.PI;
        return q;
    }

    private double[] CinematicaInversaSimplificada(double Px, double Py, double Pz)
    {
        //Aqui q[0] es q1, q[1] = Theta2, q[2] es Theta3 y q[3] es q4
        q[0] = Math.Atan(Py / Px);
        F1 = Math.Sqrt(Math.Pow(Px, 2D) + Math.Pow(Py, 2D) + Math.Pow((d1 + L1 - L4 -
        Pz), 2D));
        beta = Math.Atan((d1 + L1 - L4 - Pz) / (Math.Sqrt(Math.Pow(Px, 2D) +
        Math.Pow(Py, 2D))));
        q[1] = Math.Acos((Math.Pow(d2 + L2, 2D) + Math.Pow(F1, 2D) - Math.Pow(L3,
        2D)) / (2 * F1 * (d2 + L2))) - beta;
        q[2] = Math.Acos((Math.Pow(L3, 2D) + Math.Pow(d2 + L2, 2D) - Math.Pow(F1,
        2D)) / (2 * L3 * (d2 + L2)));
        q[3] = (Math.PI / 2D) * -1 + q[1] + q[2];
        q[0] = q[0] * 180 / Math.PI;
        q[1] = q[1] * 180 / Math.PI;
        q[2] = q[2] * 180 / Math.PI;
        q[3] = q[3] * 180 / Math.PI;
        return q;
    }

    /**************************CAMBIO DE INDICADORES*****************************/

    private void btnCambiarPosicion_Click(object sender, EventArgs e)
    {
        btnCambiarPosicion.BackColor = Color.CornflowerBlue;
        btnCambiarAceleracion.BackColor = Color.White;
        GraficasAceleracion.Text = "Gráficas de la Posición";
```

```
    groupBox1.Text = "Aceleraciones";
    label7.Text = "Aceleracón X:";
    label8.Text = "Aceleracón Y:";
    label9.Text = "Aceleracón Z:";
    numerografica = 0;
}

private void btnCambiarAceleracion_Click(object sender, EventArgs e)
{
    btnCambiarAceleracion.BackColor = Color.CornflowerBlue;
    btnCambiarPosicion.BackColor = Color.White;
    GraficasAceleracion.Text = "Gráficas de la Aceleración";
    groupBox1.Text = "Posiciones";
    label7.Text = "Posición X:";
    label8.Text = "Posición Y:";
    label9.Text = "Posición Z:";
    numerografica = 1;
}

/**********************COMUNICACIÓN SERIAL CON EL SCORBOT**********************/

private void iniciarToolStripMenuItem1_Click(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == false)
    {
        //Configurar la comunicación serial con el SCORBOT
        SerialPort.Open();
    }
    if (PuertoSerial.IsOpen == true && SerialPort.IsOpen == true)
    {
        ConexionSerie.BackColor = Color.LawnGreen;
    }
}

private void terminarToolStripMenuItem1_Click(object sender, EventArgs e)
{

    if (SerialPort.IsOpen == true)
    {
        SerialPort.DiscardInBuffer();
        SerialPort.Close();
        ConexionSerie.BackColor = Color.Red;
    }
}

/************************CONTROL MANUAL DE ACELERACIONES************************/

private void Hombro_0_Hor_MouseDown(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("1");
    }
}

private void Hombro_0_Hor_MouseUp(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
```

```
    {
        SerialPort.Write("0");
    }
}

private void Hombro_0_Anti_MouseDown(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("3");
    }
}

private void Hombro_0_Anti_MouseUp(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("2");
    }
}

private void Hombro_1_Hor_MouseDown(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("5");
    }
}

private void Hombro_1_Hor_MouseUp(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("4");
    }
}

private void Hombro_1_Anti_MouseDown(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("7");
    }
}

private void Hombro_1_Anti_MouseUp(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("6");
    }
}

private void Codo_Hor_MouseDown(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("9");
```

```
    }
}

private void Codo_Hor_MouseUp(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("8");
    }
}

private void Codo_Anti_MouseDown(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("b");
    }
}

private void Codo_Anti_MouseUp(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("a");
    }
}

private void Muneca_0_Hor_MouseDown(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("h");
    }
}

private void Muneca_0_Hor_MouseUp(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("g");
    }
}

private void Muneca_0_Anti_MouseDown(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("j");
    }
}

private void Muneca_0_Anti_MouseUp(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("i");
    }
}
```

```
private void Gripper_Hor_MouseDown(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("l");
    }
}

private void Gripper_Hor_MouseUp(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("k");
    }
}

private void Gripper_Anti_MouseDown(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("n");
    }
}

private void Gripper_Anti_MouseUp(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("m");
    }
}

private void btnEmergencyStop_Click(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("y");
    }
}

private void btnReset_Click(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("o");
    }
}

private void btnEnviarDatosS2_Click(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write(txtCadenaEnviada.Text);
    }
}

private void btnGoHome_Click(object sender, EventArgs e)
```

```
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("+");
    }
}

/*************************CONTROL MANUAL/AUTOMÁTICO*************************/

private void btnAutomatico_Click(object sender, EventArgs e)
{
    groupBox5.Visible = false;
    grpModoManual2.Visible = false;
    pictureScorbot.Visible = true;
    btnAutomatico.BackColor = Color.Orange;
    btnManuel.BackColor = Color.White;
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("!");
    }
}

private void btnManuel_Click(object sender, EventArgs e)
{
    groupBox5.Visible = true;
    grpModoManual2.Visible = true;
    pictureScorbot.Visible = false;
    btnAutomatico.BackColor = Color.White;
    btnManuel.BackColor = Color.Orange;
}

private void btnIniciarManual_Click(object sender, EventArgs e)
{
    if (SerialPort.IsOpen == true)
    {
        SerialPort.Write("?");
        btnIniciarManual.BackColor = Color.FromArgb(((int)(((byte)(192)))),
        ((int)(((byte)(192)))), ((int)(((byte)(255)))));
    }
}

/*************************LIMPIEZA DE TRAYECTORIAS*************************/

private void ClearTrayectorias_Click(object sender, EventArgs e)
{
    txtCinematicaX.Text = "0";
    txtCinematicaY.Text = "0";
    txtCinematicaZ.Text = "0";
    XManual = 0;
    YManual = 0;
    ZManual = 0;
    txtAngulosManuales.Clear();
    txtLineasTrayectoria.Clear();
    txtCadenaEnviada.Clear();
    dibujo3.Clear(Color.White);
}

/************************CAMBIO DE TIPO DE INTEGRACIÓN************************/
```

```
private void btnIntRectangular_Click(object sender, EventArgs e)
{
    tipodeintegracion = 0;
    btnIntBoole.BackColor = Color.White;
    btnIntRectangular.BackColor = Color.Orange;
    posicionRectangularX = posicionXX;
    posicionRectangularY = posicionYY;
    posicionRectangularZ = posicionZZ;
}

private void btnIntBoole_Click(object sender, EventArgs e)
{
    tipodeintegracion = 1;
    btnIntBoole.BackColor = Color.Orange;
    btnIntRectangular.BackColor = Color.White;
    posicionXX = posicionRectangularX;
    posicionYY = posicionRectangularY;
    posicionZZ = posicionRectangularZ;
}

/**************************CINEMÁTICA MANUAL**************************/

private void Cinematica_Manual(object sender, EventArgs e)
{
    //Llamamos a la Cinemática Inversa
    XManual = Convert.ToInt32(txtCinematicaX.Text);
    YManual = Convert.ToInt32(txtCinematicaY.Text);
    ZManual = Convert.ToInt32(txtCinematicaZ.Text);
    double[] AnguloManual = new double[4];
    AnguloManual = CinematicaInversaSimplificada(XManual, YManual, ZManual);
    EspacioDeTrabajo = Math.Pow(XManual, 2D) + Math.Pow(YManual, 2D) +
    Math.Pow((ZManual - d1 - L1 + L4), 2D);
    if (EspacioDeTrabajo <= Math.Pow((d2 + L2 + L3), 2D))
    {
        DentroDelEspacio = true;
        btnEspacioDeTrabajo.BackColor = Color.LawnGreen;
    }
    else
    {
        DentroDelEspacio = false;
        btnEspacioDeTrabajo.BackColor = Color.Red;
    }
    txtAngulosManuales.Clear();
    txtAngulosManuales.Text = "q1: " + AnguloManual[0].ToString("N4") + "\r\n" +
    "q2: " + AnguloManual[1].ToString("N4") + "\r\n" + "q3: " +
    AnguloManual[2].ToString("N4") + "\r\n" + "q4: " +
    AnguloManual[3].ToString("N4") + "\r\n";
    angulosScorbot = AnguloManual;
    dibujartrayectoriaXY(XManual, YManual);
    Thread th3 = new Thread(new ThreadStart(threadtexto1));
    Thread th4 = new Thread(new ThreadStart(threadtexto2));
    th3.Start();
    th4.Start();
}

private void CinematicaXUp_Click_1(object sender, EventArgs e)
{
```

```
    if (Math.Pow(XManual, 2D) + Math.Pow(YManual, 2D) < Math.Pow(radioinutil +
        IncReal, 2D))
    {
        XManual = Convert.ToInt32((Math.Ceiling((Math.Sqrt(Math.Pow(radioinutil,
        2D) - Math.Pow(YManual, 2D))) / IncReal)) * IncReal);
        if (XManual == TemporalXUp)
        {
            XManual += IncReal;
        }
        TemporalXUp = XManual;
    }
    else
    {
        XManual += IncReal;
    }
    txtCinematicaX.Text = Convert.ToString(XManual);
}

private void CinematicaXDown_Click_1(object sender, EventArgs e)
{
    if (Math.Pow(XManual, 2D) + Math.Pow(YManual, 2D) < Math.Pow(radioinutil +
        IncReal, 2D))
    {
        XManual = Convert.ToInt32((Math.Floor((Math.Sqrt(Math.Pow(radioinutil,
        2D) - Math.Pow(YManual, 2D))) / -IncReal)) * IncReal);
        if (XManual == TemporalXDown)
        {
            XManual -= IncReal;
        }
        TemporalXDown = XManual;
    }
    else
    {
        XManual -= IncReal;
    }
    txtCinematicaX.Text = Convert.ToString(XManual);
}

private void CinematicaYUp_Click_1(object sender, EventArgs e)
{
    if (Math.Pow(XManual, 2D) + Math.Pow(YManual, 2D) < Math.Pow(radioinutil +
        IncReal, 2D))
    {
        YManual = Convert.ToInt32((Math.Ceiling((Math.Sqrt(Math.Pow(radioinutil,
        2D) - Math.Pow(XManual, 2D))) / IncReal)) * IncReal);
        if (YManual == TemporalYUp)
        {
            YManual += IncReal;
        }
        TemporalYUp = YManual;
    }
    else
    {
        YManual += IncReal;
    }
    txtCinematicaY.Text = Convert.ToString(YManual);
}
```

```
private void CinematicaYDown_Click_1(object sender, EventArgs e)
{
    if (Math.Pow(XManual, 2D) + Math.Pow(YManual, 2D) < Math.Pow(radioinutil +
        IncReal, 2D))
    {
        YManual = Convert.ToInt32((Math.Floor((Math.Sqrt(Math.Pow(radioinutil,
        2D) - Math.Pow(XManual, 2D))) / -IncReal)) * IncReal);
        if (YManual == TemporalYDown)
        {
            YManual -= IncReal;
        }
        TemporalYDown = YManual;
    }
    else
    {
        YManual -= IncReal;
    }
    txtCinematicaY.Text = Convert.ToString(YManual);
}

private void CinematicaZUp_Click_1(object sender, EventArgs e)
{
    ZManual = Convert.ToInt32(txtCinematicaZ.Text);
    ZManual += IncReal;
    txtCinematicaZ.Text = Convert.ToString(ZManual);
}

private void CinematicaZDown_Click_1(object sender, EventArgs e)
{
    ZManual = Convert.ToInt32(txtCinematicaZ.Text);
    ZManual -= IncReal;
    txtCinematicaZ.Text = Convert.ToString(ZManual);
}

private void IncrementoManualUp_Click(object sender, EventArgs e)
{
    if (IncReal < 40)
    {
        IncReal = Convert.ToInt32(txtIncrementoManual.Text);
        IncReal++;
        txtIncrementoManual.Text = Convert.ToString(IncReal);
    }
}

private void IncrementoManualDown_Click(object sender, EventArgs e)
{
    if (IncReal > 5)
    {
        IncReal = Convert.ToInt32(txtIncrementoManual.Text);
        IncReal--;
        txtIncrementoManual.Text = Convert.ToString(IncReal);
    }
}

/********************GUARDADO Y RECUPERADO DE TRAYECTORIAS********************/

private void SalvarTrayectoriaArchivo_Click(object sender, EventArgs e)
{
```

```
    //Iniciamos las variables para la escritura de datos
    Stream myStream;
    SaveFile.InitialDirectory = System.Environment.GetFolderPath
    (Environment.SpecialFolder.MyDocuments);
    SaveFile.Filter = "All files (*.trx)|*.trx";
    SaveFile.RestoreDirectory = true;
    if (SaveFile.ShowDialog() == DialogResult.OK)
    {
        if ((myStream = SaveFile.OpenFile()) != null)
        {
            myStream.Close();
            StreamWriter EscritorArchivo = new StreamWriter(SaveFile.FileName);
            EscritorArchivo.Write(txtLineasTrayectoria.Text);
            EscritorArchivo.Flush();
            EscritorArchivo.Close();
        }
    }
}

private void cuadradoToolStripMenuItem_Click(object sender, EventArgs e)
{
    Stream myStream = null;
    OpenFile.InitialDirectory = System.Environment.GetFolderPath
    (Environment.SpecialFolder.MyDocuments);
    OpenFile.Filter = "All files (*.trx)|*.trx";
    OpenFile.RestoreDirectory = true;
    if ( OpenFile.ShowDialog() == DialogResult.OK )
    {
        try
        {
            if ((myStream = OpenFile.OpenFile()) != null)
            {
                using (myStream)
                {
                    //Instrucciones a realizar sobre el archivo
                    StreamReader lectorArchivo = new StreamReader
                    (OpenFile.FileName);
                    string Comandos = lectorArchivo.ReadToEnd();
                    txtLineasTrayectoria.Text = Comandos;
                    txtLineasTrayectoria.ScrollToCaret();
                }
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error: Archivo no leido. Error original: " +
            ex.Message);
        }
    }
}

/******************INICIO DE ENVIO DE INFORMACIÓN AL SCORBOT******************/

private void Iniciar_Click(object sender, EventArgs e)
{
    leerlinea = 0;
    if (SerialPort.IsOpen == true)
    {
```

```
            try
            {
                Iniciar.BackColor = Color.LawnGreen;
                //SerialPort.Write("[[");
                SerialPort.Write(txtLineasTrayectoria.Lines.ElementAt(leerlinea));
                leerlinea++;
                SerialPort.Write(txtLineasTrayectoria.Lines.ElementAt(leerlinea));
                cadenaleidaSCORBOT = "";
            }
            catch
            {

            }
        }
    }

    private void btnEscalaUp_Click(object sender, EventArgs e)
    {

        if (escalaRealPosicion == 1)
        {
            escalaRealPosicion = 2;
            txtEscala.Text = Convert.ToString(escalaRealPosicion);
        }
        else if (escalaRealPosicion < 40 && escalaRealPosicion != 1)
        {
            escalaRealPosicion += 2;
            txtEscala.Text = Convert.ToString(escalaRealPosicion);
        }
        else if (escalaRealPosicion <= 98 && escalaRealPosicion >= 40)
        {
            escalaRealPosicion += 10;
            txtEscala.Text = Convert.ToString(escalaRealPosicion);
        }
        else if (escalaRealPosicion <= 375 && escalaRealPosicion >= 100)
        {
            escalaRealPosicion += 25;
            txtEscala.Text = Convert.ToString(escalaRealPosicion);
        }
        else if (escalaRealPosicion <= 900 && escalaRealPosicion >= 375)
        {
            escalaRealPosicion += 100;
            txtEscala.Text = Convert.ToString(escalaRealPosicion);
        }
        else if (escalaRealPosicion <= 1900 && escalaRealPosicion >= 900)
        {
            escalaRealPosicion += 200;
            txtEscala.Text = Convert.ToString(escalaRealPosicion);
        }
    }

    private void btnEscalaDown_Click(object sender, EventArgs e)
    {
        if (escalaRealPosicion == 2)
        {
            escalaRealPosicion = 1;
            txtEscala.Text = Convert.ToString(escalaRealPosicion);
        }
```

```
            else if (escalaRealPosicion > 2 && escalaRealPosicion <= 40)
            {
                escalaRealPosicion -= 2;
                txtEscala.Text = Convert.ToString(escalaRealPosicion);
            }
            else if (escalaRealPosicion > 40 && escalaRealPosicion <= 100)
            {
                escalaRealPosicion -= 10;
                txtEscala.Text = Convert.ToString(escalaRealPosicion);
            }
            else if (escalaRealPosicion > 100 && escalaRealPosicion <= 400)
            {
                escalaRealPosicion -= 25;
                txtEscala.Text = Convert.ToString(escalaRealPosicion);
            }
            else if (escalaRealPosicion > 400 && escalaRealPosicion <= 1000)
            {
                escalaRealPosicion -= 100;
                txtEscala.Text = Convert.ToString(escalaRealPosicion);
            }
            else if (escalaRealPosicion > 1000)
            {
                escalaRealPosicion -= 200;
                txtEscala.Text = Convert.ToString(escalaRealPosicion);
            }
        }

        public void escribirAExcel(object[,] Sent)
        {
            DatosAExcel = new Excel.ApplicationClass();
            DatosAExcel.Visible = true;
            string workbookPath = "C:/Users/zermanik/Documents/UNAM/Servicio Social
            Tesis/Escrito Tesis/TrayectoriasAceleracionPosicion.xlsx";
            Excel.Workbook Librodetrabajo = DatosAExcel.Workbooks.Open(workbookPath, 0,
            false, 5, "", "", false, Microsoft.Office.Interop.Excel.XlPlatform.xlWindows,
            "", true, false, 0, true, false, false);
            Excel.Sheets Hojaexcel = Librodetrabajo.Worksheets;
            string hojaactual = "Sheet1";
            Excel.Worksheet Hojadetrabajo =
            (Excel.Worksheet)Hojaexcel.get_Item(hojaactual);
            Hojadetrabajo.get_Range("A2", "G21").Value2 = Sent;
        }
    }
}
```

17.4 Apéndice D: Código de programación del Arduino Mega

```
int o_di[12];  //Output_Direccion for +-MN pin
int o_ene[6];  //Output_Eneable ICn (PWM)
int i_ms[6];   //Input_Microswitch pin
int i_enc[12]; //Input_EN/P0-P1 pin

int v_ms[6];  //Variable to use internally Microswitch value
int v_ms_l[6]; //Variable to use internally Microswitch value on loop
```

```
int v_enc[12]; //Variable to use internally Encoder value
int v_ms_p[6];

int fmsr[6]; //Flag varibale microswitch right spin
int fmsl[6]; //Flag varibale virtual microswitch left spin
int fpr[6];  //Flag position reached
int fmcgh[6]; //Flag Motor Controll on go home
int fhh[6]; //Flag Hard home
int fdr;     //Flag Data Received
int fsp;     //Flag Serial Print %%
int fes;     //Flag Emergency Stop
int fgh;     //Flag went to home
int fm;      //Flag manual Mode

int count[6];  //Count Variable for fpr
int x[6];  //Present Encoder measurement
int xp[6]; //Past Encoder Measurement
int xp_isr[6]; //Past Encoder Measurement used in Interrupt Service Rutine
int xp_l[6]; //Past encoder measurment used in loop
int dx[6]; //Desired Encoder Measurement
int pw[6]; //Value of PWM

//PID parameters
float e [6];  //Error
float ep[6];  //Past Error
float ei[6]; //Integral Error
float ed[6]; //Differential Error
float r[6]; //Partial result for pwm

float kp[6]; //Propotional Constant
float ki[6]; //Integral Consatant
float kd[6]; //Differential Constant

//Timer parameters
byte IFMask = B00000010;
byte IFClrMask = B11111101;

//Set() variables
int mt=6;
double a;
float ba=10;
double tx[12];
int stx[12];
int arr[10];
int ni;
int si;
int yi;

int txp=0;
int led=13;
int state=0;
ISR (TIMER1_COMPA_vect)
{
       if(fm==false)
       {
              extisr();
       } //end iffm
} //end ISR
```

```
void extisr()
{
	//dt is set to 50 ms this is set on the setup code
	int dt=.05;
	kp[0]=5; //Hombro 0
	kp[1]=6; //Hombro 1
	kp[2]=6; //Codo
	kp[3]=0; // 2.5 Muneca
	kp[4]=4; //Muneca
	kp[5]=6; //Gripper
	ki[0]=0;
	ki[1]=0;
	ki[2]=0;
	ki[3]=0;
	//ki[4]=0;
	ki[5]=0;
	kd[0]=0;
	kd[1]=0;
	kd[2]=0;
	kd[3]=0;
	//kd[4]=0;
	kd[5]=0;

	for(int qw=0;qw<6;qw++)
	{
		if(fmcgh[qw]== true)
		{
			e[qw]=dx[qw]-x[qw];
			ei[qw]=ei[qw]+(e[qw]*20);
			ed[qw]=(e[qw]-ep[qw])*.025;
			r[qw]=e[qw]*kp[qw] + ei[qw]*ki[qw] + ed[qw]*kd[qw];
			pw[qw] = abs(r[qw]) ;
			if(pw[qw]>255)
				pw[qw]=255;
			if(pw[qw]<50)
				pw[qw]=0;

			//Set if qw = 3  eneable qw=3 & qw=4 digital outpus
			//Eneable dir +M/-M
			if(qw!=4){
				if(r[qw]>0){
					digitalWrite(o_di[qw*2+1],LOW);
					digitalWrite(o_di[qw*2],HIGH);

				}
				if(r[qw]<0){
					digitalWrite(o_di[qw*2],LOW);
					digitalWrite(o_di[qw*2+1],HIGH);
				}
```

```
                        } //End if qw!=4
                        //Set PWM
                        analogWrite(o_ene[qw],pw[qw]);

                        if(qw==4){
                                if(r[qw]<0){
                                        digitalWrite(o_di[qw*2+1],LOW);
                                        digitalWrite(o_di[qw*2],HIGH);

                                }
                                if(r[qw]>0){
                                        digitalWrite(o_di[qw*2],LOW);
                                        digitalWrite(o_di[qw*2+1],HIGH);

                                }
                        } //End if qw==4

                        if(r[qw]==0){
                                digitalWrite(o_di[qw*2],LOW);
                                digitalWrite(o_di[qw*2+1],LOW);
                                //If e[qw]=0 -> pw[qw]=0
                        } //end ifr[qw]

                        if(fdr==true){
                                if(((xp_isr[qw])-8) <= x[qw] && x[qw] <= ((xp_isr[qw])+8)){
                                        count[qw]++;
                                }
                                if(count[qw]==2){
                                        fpr[qw]=true;
                                        count[qw]=0;
                                }

                                if(fpr[0] == true && fpr[1]==true && fpr[2]==true &&
                                fpr[3]==true && fpr[4]==true && fpr[5]==true)
                                {
                                        fsp=true;
                                } //end if fpr
                        } //endif fdr
                        //ep[qw]=e[qw];//for diferential controller
                        xp_isr[qw]=x[qw];
                } //end if fmcgh
        } //end forqw
} //end extisr


void setup()
{
        pinMode(led,OUTPUT);//Debug led

        //Timer interrupt
        TCCR1A = B01000000; //bin Configure Timer
```

```
TCCR1B = B00001011; //prescaled 1/64 [Hz]   ciclos de .025s= 1/(16Mhz/64)*6250
.050s= 1/(16Mhz/64)*12500    .100 s= 1/(16Mhz/64)*25000
TIMSK1 = B00000010;   /* Enables the Timer1 Compare A interrupt */
OCR1AH = 0x61; //0h.186A= dec6250           0h.30d4=dec 12500        0h.61AB=
dec25000
OCR1AL = 0xab;
//Pin Numbers
o_di[0] = 22; //+M1
o_di[1] = 23; //-M1
o_di[2] = 24; //+M2
o_di[3] = 25; //-M2
o_di[4] = 26; //+M3
o_di[5] = 27; //-M3
o_di[6] = 28; //+M4
o_di[7] = 29; //-M4
o_di[8] = 30; //+M5
o_di[9] = 31; //-M5
o_di[10] = 32; //+M6
o_di[11] = 33; //-M6

o_ene[0] = 9; //Eneable M1 PWM
o_ene[1] = 8; //Eneable M2 PWM
o_ene[2] = 7; //Eneable M3 PWM
o_ene[3] = 4; //Eneable M4 PWM
o_ene[4] = 6; //Eneable M5 PWM
o_ene[5] = 5; //Eneable M6 PWM

i_ms[0]=35; //MS1 Hombro 0
i_ms[1]=38; //MS2 Hombro 1
i_ms[2]=34; //MS1 Codo
i_ms[3]=37; //MS4 Muñeca
i_ms[4]=36; //MS3 ROLL
i_ms[5]=39; //MS6 not wired yet Gripper

i_enc[0]= 2;  //EN1P0 used as external intrrupt, not as input
i_enc[1]= 43; //EN1P1
i_enc[2]= 3;  //EN2P0 used as external intrrupt, not as input
i_enc[3]= 45; //EN2P1
i_enc[4]= 21; //EN3P0 used as external intrrupt, not as input
i_enc[5]= 47; //EN3P1
i_enc[6]= 20; //EN4P0 used as external intrrupt, not as input
i_enc[7]= 49; //EN4P1
i_enc[8]= 19; //EN5P0 used as external intrrupt, not as input
i_enc[9]= 51; //EN5P1
i_enc[10]=18; //EN6P0 Not Wired used as external intrrupt, not as input
i_enc[11]=53; //EN6P1 Not Wired

//Set flags to false Microswitches
for(int k=0;k<6;k++){
       fmsr[k]=false; //right ms
       fmsl[k]=false; //virtual ms
} //end for

fes=false;  //Set Emergency stop flag to false
fgh=false;  //flag passed through home set to false
fm=true;
//Seting pin modes to output
//Spin direction
```

```
pinMode(o_di[0], OUTPUT);
pinMode(o_di[1], OUTPUT);
pinMode(o_di[2], OUTPUT);
pinMode(o_di[3], OUTPUT);
pinMode(o_di[4], OUTPUT);
pinMode(o_di[5], OUTPUT);
pinMode(o_di[6], OUTPUT);
pinMode(o_di[7], OUTPUT);
pinMode(o_di[8], OUTPUT);
pinMode(o_di[9], OUTPUT);
pinMode(o_di[10], OUTPUT);
pinMode(o_di[11], OUTPUT);

pinMode(o_ene[0], OUTPUT);
pinMode(o_ene[1], OUTPUT);
pinMode(o_ene[2], OUTPUT);
pinMode(o_ene[3], OUTPUT);
pinMode(o_ene[4], OUTPUT);
pinMode(o_ene[5], OUTPUT);

//Seting pin modes to input
//Microswitch
pinMode(i_ms[0], INPUT);
pinMode(i_ms[1], INPUT);
pinMode(i_ms[2], INPUT);
pinMode(i_ms[3], INPUT);
pinMode(i_ms[4], INPUT);
pinMode(i_ms[5], INPUT);

//Encoders
pinMode(i_enc[1], INPUT);
pinMode(i_enc[3], INPUT);
pinMode(i_enc[5], INPUT);
pinMode(i_enc[7], INPUT);
pinMode(i_enc[9], INPUT);
pinMode(i_enc[11],INPUT);

//Define External Interrupts
attachInterrupt(0, enc0, RISING);  //PIN 2
attachInterrupt(1, enc2, RISING);  //PIN 3
attachInterrupt(2, enc4, RISING);  //PIN 21
attachInterrupt(3, enc6, RISING);  //PIN 20
attachInterrupt(4, enc8, RISING);  //PIN 19
attachInterrupt(5, enc10, RISING);  //PIN 18

/*  //As input, no external interruption
 pinMode(i_enc[0], INPUT);
 pinMode(i_enc[2], INPUT);
 pinMode(i_enc[4], INPUT);
 pinMode(i_enc[6], INPUT);
 pinMode(i_enc[8], INPUT);
 pinMode(i_enc[10],INPUT);
*/

//Set LOW in all OUTPUTS
for(int r=22; r<33;r++){
       digitalWrite(r,LOW);
}
```

```
        Serial.begin(9600);//19200
        Serial.println("Ready");

} //end setup


//------------------------------          Loop       ----------------------------------
void loop()
{
        //Methods for this class: es(); go home(); ms();
        //ISR
        //External interrupts enc0(); enc2(); ... enc10();

        if(Serial.available())
        {
                int c = Serial.read();

                //Set or reset Emergency Stop
                if(c=='y')
                        fes=true;
                if(c=='o')
                {
                        fes=false;
                        reset();
                }
                if(fes==true) //if emergency stop flag
                        es(); //es method called turn off motors
                if(c=='+')
                        gohome(); //Start position of robot arm

                /*    if(c=='-')
                 gohome-(); //Start position of robot arm
                 */

                if(c=='!')
                        fm=false;
                if(c=='?')
                {
                        Serial.println("Manual");
                        fm=true;
                }
                if(c=='#')
                        set();
                if(fm==true)
                        dir(c);

        } //endifserialread
        ms(); //Check for MS states

        if(fes!=true)
        {
                if(fsp==true)
                {
                        Serial.println("%%");
                        fpr[0]=false;
                        fpr[1]=false;
                        fpr[2]=false;
```

```
                fpr[3]=false;
                fpr[4]=false;
                fpr[5]=false;
                fsp=false;
                fdr=false;
        } //end iffsp
}

//Debuging

Serial.println("@@");
Serial.println(x[0]); //H0
Serial.println(x[1]); //H1
Serial.println(x[2]); //Codo
Serial.println(x[3]); //M1
Serial.println(x[4]); //M2
Serial.println(x[5]); //Gripper
Serial.println(v_ms_l[0]);
Serial.println(v_ms_l[1]);
Serial.println(v_ms_l[2]);
Serial.println(v_ms_l[3]);

/*
if(txp != tx[0])
{
        Serial.println("Data on tx[n]");
        Serial.print("tx[0]:");
        Serial.println(tx[0]);
        Serial.print("tx[1]:");
        Serial.println(tx[1]);
        Serial.print("tx[2]:");
        Serial.println(tx[2]);
        Serial.print("tx[3]:");
        Serial.println(tx[3]);
        Serial.print("tx[4]:");
        Serial.println(tx[4]);
        Serial.print("tx[5]:");
        Serial.println(tx[5]);
        Serial.print("tx[6]:");
        Serial.println(tx[6]);
        Serial.print("tx[7]:");
        Serial.println(tx[7]);
        Serial.print("tx[8]:");
        Serial.println(tx[8]);
        Serial.print("tx[9]:");
        Serial.println(tx[9]);
        Serial.print("tx[10]:");
        Serial.println(tx[10]);
        Serial.print("tx[11]:");
        Serial.println(tx[11]);
        txp = tx[0];
}//end if txp

//Serial.println();

/*
for(int ki=0;ki<6;ki++)
 {
```

```
                if(x[ki] != xp_l[ki])
                {
                        Serial.print("x");
                        Serial.print(ki);
                        Serial.print(":");
                        Serial.println(x[ki]);
                        Serial.print("dx");
                        Serial.print(ki);
                        Serial.print(":");
                        Serial.println(dx[ki]);
                        Serial.print("e:");
                        Serial.println(e[ki]);
                        //Serial.print("ei:");
                        //Serial.println(ei[ki]);
                        Serial.print("pwm:");
                        Serial.println(pw[ki]);
                        Serial.println("------");
                        // coment slash asterix
                        Serial.print("xp_isr");
                        Serial.print(ki);
                        Serial.print(":");
                        Serial.print(xp_isr[ki]);
                        Serial.print("count");
                        Serial.print(ki);
                        Serial.print(":");
                        Serial.print(count[ki]);
                        Serial.print("fpr");
                        Serial.print(":");
                        Serial.print(fpr[ki]);
                        Serial.print("fsp");
                        Serial.print(":");
                        Serial.print(fsp);
                        Serial.println("");

                }//ENd ifx[]
                xp_l[ki]=x[ki];
        }//End forki
        */
} //end loop



//------------------------------          ms          ----------------------------------
void ms()
{
        for(int kaw=0;kaw<6;kaw++)
        {
                v_ms_l[kaw]=digitalRead(i_ms[kaw]);
        } //end for kaw
}
//------------------------------         Set          ----------------------------------

void set()
{
        do{
                ;
        }
        while(Serial.available()==false);
```

```
	if(Serial.available()>0)
	{
		delay(5);
		yi=0; //counter for position in array reset
		si=0; //counter for position in array reset
		for(int mn=0;mn<12;mn++)
		{
			stx[mn]=1; //temp sign of data reset
			tx[mn]=0; //temp array for data reset
		} //end formn

		do{
			ni=Serial.read();

			if(ni != 44 && ni != 45 && ni !=35) //44 = ,     45 = -    35 = #
			{
				arr[yi]=ni;
				delay(5);
				yi++;
			}
			if(ni == 44) //44 = ,
			{
				yi--;
				int gk=yi;
				for(int i=0;i<=yi;i++)
				{
					a=pow(ba,(float)i);
					tx[si]+=(cnvrt(arr[gk]))*a;
					gk--;
				}
				yi=0;
				si++; //If si==5 Serial.Flush(); break();

			} //IF 44
			if(ni==45) //45 = -
				stx[si]= -1;
		} //Do
		while(Serial.available()>0);
	} //end if serialavailable

	for(int ks=0;ks<mt;ks++)
	{
		tx[ks]=tx[ks]*stx[ks];
		dx[ks]=tx[ks];
	}

	fdr=true;

} //end set
//--------------------------------       Convert       --------------------------------

int cnvrt(int w)
{
	if(w==48)
		w=0;
	if(w==49)
		w=1;
```

```
        if(w==50)
                w=2;
        if(w==51)
                w=3;
        if(w==52)
                w=4;
        if(w==53)
                w=5;
        if(w==54)
                w=6;
        if(w==55)
                w=7;
        if(w==56)
                w=8;
        if(w==57)
                w=9;
        return w;
} //end cnvrt


//--------------------------------   Emergency Stop    --------------------------------

void es()
{
        //Stop all eneables
        //Stop all dir motors
        fm=false;
        for(int p=0;p<6;p++)
        {
                fmcgh[p]=true;
                dx[p]=x[p];
        }
} //end es


//--------------------------------        Go Home      --------------------------------

void gohome()
{
        //Move every motor in correct sens until Microswitches are activated
        //MS1 Hombro 0
        //MS2 Hombro 1
        //MS3 Codo
        //MS4 MuÃ±eca
        //MS5 MuÃ±eca
        //MS6 Gripper
        reset();
        fm=false;
        for(int m=0;m<6;m++) //each motor
        {
                if(m!=4)
                {
                        do{
                                v_ms[m]=digitalRead(i_ms[m]);  //Read ms

                                if(m==5)
                                        v_ms[m]=true;
```

```
        if(v_ms[m]==false) //if ms is pressed
               break; //MS pressed

        //EMERGENCY STOP FLAG
        if(Serial.available())
        {
               if(Serial.read()=='y');
               {
                      fes=true; //Flag emergency stop
                      es(); //stop
                      m=6; //For break the "for"
                      break;
               }
        } //If serial.available

        if(m!=2 && m!=5)
        {
               digitalWrite(o_ene[m],HIGH); //turn on motor eneable
               digitalWrite(o_di[m*2],HIGH); //turn on motor in
               correct sense from correct joint
        }
        if(m==2)
        {
               digitalWrite(o_ene[m],HIGH); //Eneable off
               digitalWrite(o_di[m*2+1],HIGH); //When Microswitch is
               reached motors are off
        }
        if(m==3)
        {
               //For the gripper motor
               digitalWrite(o_ene[m+1],HIGH); //turn on motor eneable
               digitalWrite(o_di[(m+1)*2+1],HIGH); //turn on motor in
               correct sense from correct joint +1 to get in other
               sense the M5
        }
        if(m==4)
        {
               //For the gripper motor
               digitalWrite(o_ene[m-1],HIGH); //turn on motor eneable
               digitalWrite(o_di[(m-1)*2],HIGH); //turn on motor in
               correct sense from correct joint +1 to get in other
               sense the M5
        }
        if(m==5)
        {
               digitalWrite(o_ene[m],HIGH); //Eneable off
               digitalWrite(o_di[m*2+1],HIGH); //When Microswitch is
               reached motors are off
        }
        xp[m] = x[m];
        delay(13);
        if(xp[m] == x[m])
        fhh[m]=true;

}
while(fhh[m]==false);

if(m==5)
```

```
        v_ms[m]=false;
//Digital Write LOW
if(m!=2)
        digitalWrite(o_di[m*2],LOW); //turn on motor in correct sense
        from correct joint
if(m==2)
        digitalWrite(o_di[m*2+1],LOW);//When Microswitch is reached
        motors are off
if(m==3)
        digitalWrite(o_di[(m+1)*2+1],LOW); //turn on motor in correct
        sense from correct joint +1 to get in other sense the M5
if(m==4)
        digitalWrite(o_di[(m-1)*2],LOW); //turn on motor in correct
        sense from correct joint +1 to get in other sense the M5

if(v_ms[m]==true)
{
        do{
                v_ms[m]=digitalRead(i_ms[m]);  //Read ms
                if(v_ms[m]==false) //if ms is pressed
                        break; //MS pressed

                //EMERGENCY STOP FLAG
                if(Serial.available())
                {
                        if(Serial.read()=='y');
                        {
                                fes=true; //Flag emergency stop
                                es(); //stop
                                m=6; //For break the "for"
                                break;
                        }
                }//If serial.available

                if(m!=2)
                {
                        digitalWrite(o_ene[m],HIGH); //turn on motor
                        eneable
                        digitalWrite(o_di[m*2+1],HIGH); //turn on motor
                        in correct sense from correct joint
                }
                if(m==2)
                {
                        digitalWrite(o_ene[m],HIGH); //Eneable off
                        digitalWrite(o_di[m*2],HIGH);//When Microswitch
                        is reached motors are off
                }
                if(m==3)
                {
                        //For the gripper motor
                        digitalWrite(o_ene[m+1],HIGH); //turn on motor
                        eneable
                        digitalWrite(o_di[(m+1)*2],HIGH); //turn on
                        motor in correct sense from correct joint +1 to
                        get in other sense the M5
                }
                if(m==4)
                {
```

```
                                        //For the gripper motor
                                        digitalWrite(o_ene[m-1],HIGH); //turn on motor
                                        eneable
                                        digitalWrite(o_di[(m-1)*2+1],HIGH); //turn on
                                        motor in correct sense from correct joint +1 to
                                        get in other sense the M5
                                }
                                delay(5);
                                xp[m]=x[m];
                        }
                        while(v_ms[m]==true);
                }//end Ifvms
                digitalWrite(o_di[m*2],LOW); //When Microswitch is reached motors
                are off
                digitalWrite(o_di[m*2+1],LOW); //When Microswitch is reached motors
                are off
                digitalWrite(o_ene[m],LOW); //Eneable off

                if(m==2)
                {
                        digitalWrite(o_ene[m],LOW); //Eneable off
                        digitalWrite(o_di[m*2+1],LOW); //When Microswitch is reached
                        motors are off
                        digitalWrite(o_di[m*2],LOW); //When Microswitch is reached
                        motors are off
                }
                if(m==3)
                {
                        digitalWrite(o_ene[m+1],LOW); //Eneable off
                        digitalWrite(o_di[(m+1)*2+1],LOW); //When Microswitch is
                        reached motors are off
                        digitalWrite(o_di[(m+1)*2],LOW); //When Microswitch is reached
                        motors are off
                }
                if(m==4)
                {
                        digitalWrite(o_ene[m-1],LOW); //Eneable off
                        digitalWrite(o_di[(m-1)*2],LOW); //When Microswitch is reached
                        motors are off
                        digitalWrite(o_di[(m-1)*2+1],LOW); //When Microswitch is
                        reached motors are off
                }
        } //If m!=4
        //Set x[]=0
        x[m]=0; //resets the encoder counters
        dx[m]=0;
        delay(5);
        fmcgh[m]=true;
    } //end for
} //end gohome
//--------------------------------         Reset       --------------------------------

void reset()
{
    for(int pu=0;pu<12;pu++)
    {
        digitalWrite(o_ene[pu],LOW);
        digitalWrite(o_di[pu*2],LOW);
```

```
		digitalWrite(o_di[pu*2+1],LOW);
	}

	for(int rtz=0;rtz<12;rtz++)
	{
		v_enc[rtz]=0; //Variable to use internally Encoder value
	}

	for(int rtu=0;rtu<6;rtu++)
	{
		v_ms[rtu]=0;  //Variable to use internally Microswitch value
		v_ms_l[rtu]=0; //Variable to use internally Microswitch value on loop
		v_ms_p[rtu]=0;
		fmsr[rtu]=0; //Flag varibale microswitch right spin
		fmsl[rtu]=0; //Flag varibale virtual microswitch left spin
		fpr[rtu]=0;  //Flag position reached
		fmcgh[rtu]=0; //Flag Motor Controll on go home
		fhh[rtu]=0; //Flag Hard home
		count[rtu]=0;  //Count Variable for fpr
		x[rtu]=0;  //Present Encoder measurement
		xp[rtu]=0; //Past Encoder Measurement
		xp_isr[rtu]=0; //Past Encoder Measurement used in Interrupt Service Rutine
		xp_l[rtu]=0; //Past encoder measurment used in loop
		dx[rtu]=0; //Desired Encoder Measurement
		pw[rtu]=0; //Value of PWM
		e [rtu]=0;  //Error
		ep[rtu]=0;  //Past Error
		ei[rtu]=0; //Integral Error
		ed[rtu]=0; //Differential Error
		r[rtu]=0; //Partial result for pwm
		kp[rtu]=0; //Propotional Constant
		ki[rtu]=0; //Integral Consatant
		kd[rtu]=0; //Differential Constant
		tx[rtu]=0;
		stx[rtu]=0;
	}

	fdr=0;     //Flag Data Received
	fsp=0;     //Flag Serial Print %%
	fes=0;     //Flag Emergency Stop
	fgh=0;     //Flag went to home
	fm=true;      //Flag manual Mode

	/*
	fmcgh[0]=false;fmcgh[1]=false;fmcgh[2]=false;fmcgh[3]=false;fmcgh[4]=false;fmcgh[5
	]=false;
	x[0]=0;x[1]=0;x[2]=0;x[3]=0;x[4]=0;x[5]=0;
	xp[0];xp[1];xp[2];xp[3];xp[4];xp[5];
	*/
}
//--------------------------------Encoder Interruptions---------------------------------



//Digital Pin 2 interrupt
void enc0()
{
	v_enc[1]=digitalRead(i_enc[1]); //Read on digital pin: i_enc[1] = 43
```

```
        if(v_enc[1]==LOW)
                x[0]++;
        if(v_enc[1]==HIGH)
                x[0]--;
}//end enc0

//Digital Pin 3 interrupt
void enc2()
{
        v_enc[3]=digitalRead(i_enc[3]); //Read on digital pin: i_enc[3] = 45
        if(v_enc[3]==HIGH)
                x[1]++;
        if(v_enc[3]==LOW)
                x[1]--;
} //end enc1

//Digital Pin 21 interrupt
void enc4()
{
        v_enc[5]=digitalRead(i_enc[5]); //Read on digital pin: i_enc[5] = 47
        if(v_enc[5]==HIGH)
                x[2]++;
        if(v_enc[5]==LOW)
                x[2]--;
} //end enc2

//Digital Pin 20 interrupt
void enc6()
{
        v_enc[7]=digitalRead(i_enc[7]); //Read on digital pin: i_enc[7] = 49
        if(v_enc[7]==HIGH)
                x[3]++;
        if(v_enc[7]==LOW)
                x[3]--;
} //end enc3

//Digital Pin 19 interrupt
void enc8()
{
        v_enc[9]=digitalRead(i_enc[9]); //Read on digital pin: i_enc[9] = 51
        if(v_enc[9]==HIGH)
                x[4]++;
        if(v_enc[9]==LOW)
                x[4]--;
} //end enc4

//Digital Pin 18 interrupt
void enc10()
{
        v_enc[11]=digitalRead(i_enc[11]); //Read on digital pin: i_enc[11] = 53
        if(v_enc[11]==HIGH)
                x[5]--;
        if(v_enc[11]==LOW)
                x[5]++;
} //end enc4
```

```
//--------------------------------      Rotate +M/-M   --------------------------------

void dir(int c)
{
	int pwr=255;
	//    MOTOR 1
	//+M1
	if(fmsr[0]==false)
	{
		if (c == '1')
		{
			digitalWrite(o_di[0],HIGH);
			analogWrite(o_ene[0],pwr); //PWM of the motor
		} //end if motor on
	} //end if flag
	if (c == '0')
	{
		digitalWrite(o_di[0],LOW);
		analogWrite(o_ene[0],0);
	} //end if motor off
	//-M1
	if(fmsl[0]==false)
	{
		if (c == '3')
		{
			digitalWrite(o_di[1],HIGH);
			analogWrite(o_ene[0],pwr); //PWM of the motor
		}//end if motor on
	}//end if flag
	if (c == '2')
	{
		digitalWrite(o_di[1],LOW);
		analogWrite(o_ene[0],0);
	}//end if motor off

	//    MOTOR 2
	//+M2
	if(fmsr[1]==false)
	{
		if (c == '5')
		{
			digitalWrite(o_di[2],HIGH);
			analogWrite(o_ene[1],pwr); //PWM of the motor
		} //end if motor on
	} //end if flag
	if (c == '4')
	{
		digitalWrite(o_di[2],LOW);
		analogWrite(o_ene[1],0);
	} //end if motor off
	//-M2
	if(fmsl[1]==false)
	{
		if (c == '7')
		{
			digitalWrite(o_di[3],HIGH);
			analogWrite(o_ene[1],pwr); //PWM of the motor
		} //end if motor on
```

```
} //end if flag
if (c == '6')
{
	digitalWrite(o_di[3],LOW);
	analogWrite(o_ene[1],0);
} //end if motor off

//    MOTOR 3
//+M3
if(fmsr[2]==false)
{
	if (c == '9')
	{
		digitalWrite(o_di[4],HIGH);
		analogWrite(o_ene[2],pwr); //PWM of the motor
	} //end if motor on
} //end if flag
if (c == '8')
{
	digitalWrite(o_di[4],LOW);
	analogWrite(o_ene[2],0);
} //end if motor off
//-M3
if(fmsl[2]==false)
{
if (c == 'b')
	{
		digitalWrite(o_di[5],HIGH);
		analogWrite(o_ene[2],pwr); //PWM of the motor
	} //end if motor on
} //end if flag
if (c == 'a')
{
	digitalWrite(o_di[5],LOW);
	analogWrite(o_ene[2],0);
}//end if motor off

//    MOTOR 4
//+M4
if(fmsr[3]==false)
{
	if (c == 'd')
	{
		digitalWrite(o_di[6],HIGH);
		analogWrite(o_ene[3],pwr); //PWM of the motor
		digitalWrite(o_di[8],HIGH);
		analogWrite(o_ene[4],pwr); //PWM of the motor
	}//end if motor on
}//end if flag
if (c == 'c')
{
	digitalWrite(o_di[6],LOW);
	analogWrite(o_ene[3],0);
	digitalWrite(o_di[8],LOW);
	analogWrite(o_ene[4],0);
}//end if motor off
//-M4
if(fmsl[3]==false)
```

```
{
	if (c == 'f')
	{
		digitalWrite(o_di[7],HIGH);
		analogWrite(o_ene[3],pwr); //PWM of the motor
		digitalWrite(o_di[9],HIGH);
		analogWrite(o_ene[4],pwr); //PWM of the motor
	}//end if motor on
}//end if flag
if (c == 'e')
{
	digitalWrite(o_di[7],LOW);
	analogWrite(o_ene[3],0);
	digitalWrite(o_di[9],LOW);
	analogWrite(o_ene[4],0);
}//end if motor off

//    MOTOR 5
//+M5
if(fmsr[4]==false)
{
	if (c == 'h')
	{
		digitalWrite(o_di[8],HIGH);
		analogWrite(o_ene[4],pwr); //PWM of the motor
	}//end if motor on
}//end if flag
if (c == 'g')
{
	digitalWrite(o_di[8],LOW);
	analogWrite(o_ene[4],0);
}//end if motor off
//-M5
if(fmsl[4]==false)
{
	if (c == 'j')
	{
		digitalWrite(o_di[9],HIGH);
		analogWrite(o_ene[4],pwr); //PWM of the motor
	}//end if motor on
}//end if flag
if (c == 'i')
{
	digitalWrite(o_di[9],LOW);
	analogWrite(o_ene[4],0);
}//end if motor off



//    MOTOR 6 lk nm
//+M6
if(fmsr[5]==false)
{
	if (c == 'l')
	{
		digitalWrite(o_di[10],HIGH);
		analogWrite(o_ene[5],pwr); //PWM of the motor
```

```
                }//end if motor on
        }//end if flag
        if (c == 'k')
        {
                digitalWrite(o_di[10],LOW);
                analogWrite(o_ene[5],0);
        }//end if motor off
        //-M6
        if(fmsl[5]==false)
        {
                if (c == 'n')
                {
                        digitalWrite(o_di[11],HIGH);
                        analogWrite(o_ene[5],pwr); //PWM of the motor
                }//end if motor on
        }//end if flag
        if (c == 'm')
        {
                digitalWrite(o_di[11],LOW);
                analogWrite(o_ene[4],0);
        } //end if motor off
} //end dir
```