Informe Técnico

Hardware y *software* para el posicionamiento milimétrico

Benjamín Valera Orozco
Itzel Olimpia Ortiz Avila
Salvador Enrique Sánchez Minero
Rigoberto Nava Sandoval

Abril 2023

Proyecto de desarrollo tecnológico

Financiamiento Interno

Hardware y *software* para el posicionamiento milimétrico

Benjamín Valera Orozco, Itzel Olimpia Ortiz Avila, Salvador Enrique Sánchez Minero, Rigoberto Nava Sandoval

Instituto de Ciencias Aplicadas y Tecnología
Universidad Nacional Autónoma de México, Circuito Exterior S/N, Ciudad Universitaria
AP 70-186, C.P. 04510, México D.F. México

Resumen

El posicionamiento micrométrico de diversos elementos que componen un arreglo experimental en el estudio de la física desempeña una labor importante mediante el cual se aseguran la estabilidad estructural y la repetibilidad de los resultados. Tal es el caso del arreglo experimental para la velocimetría de imagen de partículas, conocido por sus siglas en inglés como PIV (*Photo Image Velocimetry*) en donde se buscan posicionar diversos elementos ópticos como lo son cámaras, láseres de estado sólido y lentes ópticos sobre una mesa óptica con el objetivo de estudiar partículas en movimiento dentro de un fluido. En este informe se describe el desarrollo de *hardware* y *software* para acoplar tres motores a pasos a las perillas giratorias de tres platinas milimétricas que en conjunto forman un sistema de posicionamiento en tres dimensiones. Mediante este enfoque se busca sustituir el movimiento de la mano humana aplicado sobre las perillas con el objetivo de aislar esfuerzos que puedan influir en la incertidumbre del posicionamiento. Se describen el circuito electrónico que controla el movimiento de los motores a paso desarrollado a partir de componentes electrónicos básicos, el *software* de operación elaborado en Visual Studio 2019 y los acoplamientos mecánicos que permiten conectar los motores a paso con las perillas de las platinas milimétricas. Se presentan resultados cualitativos en el posicionamiento tridimensional.

Índice

Nomenclatura

Símbolo, término o abreviatura	Significado
IDE	*Integrated Development Environment*. Ambiente Integrado de Desarrollo. Son plataformas que permiten agrupar herramientas de desarrollo.
USB	*Universal Serial Bus*. Conector Universal Serie. Es un medio para conectar computadoras y periféricos.
PIV	*Particle Image Velocimeter*, Velocimetría de Imágenes de Partículas, técnica empleada para analizar el movimiento en fluidos.
TBJ	Transistor Bipolar de Juntura, dispositivo electrónico de estado sólido.
PWM	*Pulse Width Modulation*, Modulación por ancho del pulso. Técnica utilizada para variar la componente de directa de señales de pulsos.
Joystick	Palanca de mandos utilizada comúnmente para videojuegos.
PCB	*Printed Circuit Board*. Tarjeta de circuito impreso.
C	Lenguaje de programación para procesadores electrónicos.
MFC	*Microsoft Foundation Classes*. Clases de programación orientada a objetos de Microsoft.

Introducción

Definición del problema

En la elaboración de experimentos científicos en los ámbitos académico e industrial, generalmente se buscan posicionar diversos dispositivos de manera que el arreglo experimental en conjunto tenga estabilidad mecánica con el objetivo de incrementar la confiabilidad de los resultados. En particular, un arreglo experimental para implementar la técnica de PIV requiere de resoluciones milimétricas en el posicionamiento de sus diversos componentes como lo son láseres, cámaras y lentes sobre una mesa óptica de laboratorio, como se muestra en la figura 1.

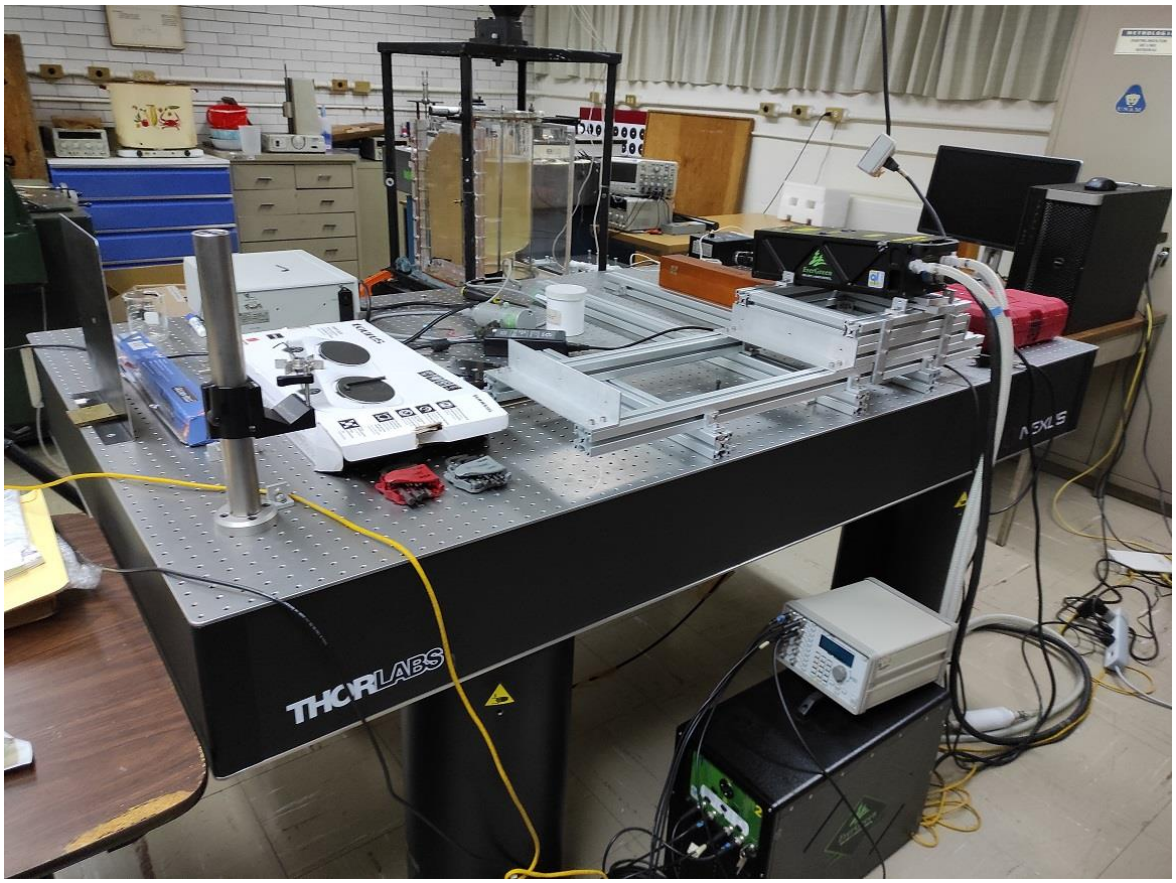

Figura 1. Arreglo experimental sobre una mesa óptica de laboratorio.

Para el posicionamiento de los dispositivos se utilizan platinas linealmente desplazables en los ejes coordenados y diversos soportes para la sujeción firme y robusta.

Entorno actual

En el Laboratorio del Grupo de Ingeniería de Proceso del ICAT UNAM actualmente se experimenta con diversos arreglos para la instrumentación de un PIV. En particular, el grupo de académicos del Laboratorio identificó la necesidad de contar con mecanismos

de desplazamiento micrométrico que permitan la correcta alineación de los diversos componentes.

Descripción del problema a resolver y motivación

En el posicionamiento de los diversos componentes se detectó que una fuente de incertidumbre importante es la operación manual sobre ellos, al introducir esfuerzos impredecibles que afectan la repetibilidad de las condiciones geométricas del arreglo experimental. Por esta razón, se plantea como una solución, el posicionamiento automático de los componentes por medio de platinas desplazables mediante motores a paso que eviten la intervención humana. A la par, se debe contar con un software de operación que permita el movimiento automático de los componentes.

La principal motivación para desarrollar la automatización en el movimiento es la creación de infraestructura exprofeso para las necesidades del laboratorio, en donde pueden participar académicos, alumnos y trabajadores del taller del instituto.

Relevancia y limitaciones

La relevancia del presente trabajo radica en la sustitución de las costosas soluciones que se comercializan en el mercado (Mach3, 2023) por desarrollos propios, que resultan ser soluciones accesibles. Además, estos desarrollos tienen una alta posibilidad de integrarse a sistemas de instrumentación más complejos que seguramente se elaborarán en el Laboratorio, debido a que se tiene el conocimiento y la documentación detallada del desarrollo.

La principal limitación en el desarrollo del presente proyecto está relacionada con el alcance y resolución de las platinas milimétricas propiedad del Laboratorio. Debe ser claro que, con el trabajo reportado en este informe, no es posible superar las características inherentes de las platinas desplazables utilizadas.

Relación con otras áreas

Existe gran relación con las áreas de sistemas digitales, microcontroladores y programación de computadoras.

Método

El método por emplear consiste en la implementación de un sistema electrónico que permita el control de movimiento en tres motores a paso y que cuente con interconexión a una computadora personal en donde se ejecute el software de operación, como se muestra en la figura 2.

En el método de la figura 2, en una computadora con *software* desarrollado a partir de herramientas básicas, se implementa la interfase de usuario que permitirá posicionar las platinas mediante el uso de una palanca de mandos comercial o mediante una posición establecida con el teclado. El sistema electrónico se conecta a la computadora

mediante el puerto USB en espera de recibir comandos para girar los tres motores a pasos conectados a sus respectivas platinas milimétricas.

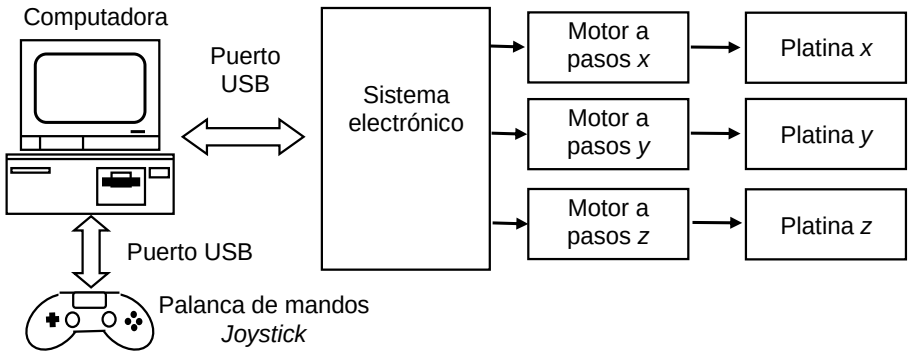

Figura 2. Método por emplear.

Objetivos y resultados esperados

Se espera contar con un prototipo funcional y *software* para posicionar dispositivos ópticos en un arreglo experimental para PIV. Con la infraestructura desarrollada se apoyará tanto a alumnos como académicos del Laboratorio de Ingeniería de Proceso en sus trabajos de investigación. Las principales especificaciones para el sistema de posicionamiento son las siguientes:

- Alcance de 60 mm en cada eje.
- Resolución de fracciones de milímetro. Se estima que la división mínima del tornillo milimétrico se puede dividir en aproximadamente 5 fracciones.
- Consumo mínimo de potencia para evitar sobrecalentamientos ambientales. Se espera que cuando los motores no estén desarrollando algún movimiento, el consumo de potencia se reduzca al mínimo para mantener al control del sistema electrónico.
- *Software* de operación integrado y amigable.

Organización del informe

En este apartado de introducción se describe el problema a resolver y el método por emplear para abordarlo. El capitulo uno describe el sistema electrónico desarrollado en sus componentes *hardware* y *software*. El capítulo dos describe el *software* de operación para PC específicamente elaborado para la presente aplicación en los ámbitos del programador y del operador, así como una guía de instalación. El capitulo tres contiene los resultados cualitativos obtenidos con el desarrollo del presente trabajo. Los anexos incluyen material detallado que permiten la reproducibilidad del sistema electrónico y de programación para el sistema de posicionamiento.

1 Sistema electrónico

El sistema electrónico que se describe en este informe está construido sobre la base del microcontrolador Arduino Nano y puentes H discretos.

1.1 Descripción general

La figura 3 muestra el esquema general del sistema electrónico desarrollado y sus conexiones hacia una computadora y los motores a pasos.

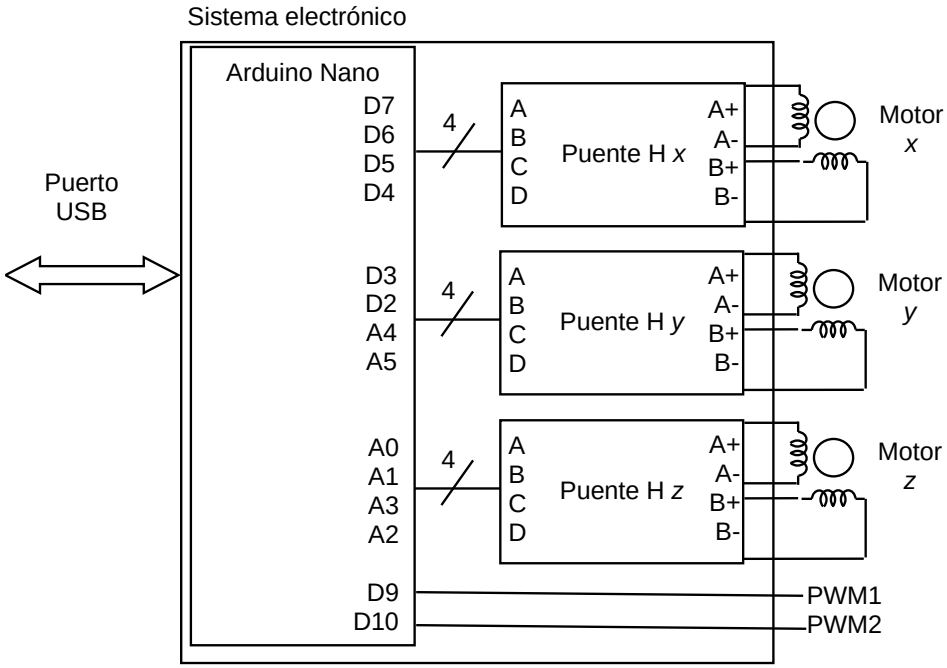

Figura 3. Esquema general.

En la figura 3, el sistema electrónico está compuesto por cuatro placas electrónicas, que se describen a continuación:

- Puentes H. Tres placas que implementan cada una el par de puentes H necesarios para impulsar los dos embobinados de los motores a pasos bipolares.
- Arduino Nano. Una placa que contiene al microcontrolador Arduino Nano encargado de generar las secuencias para impulsar los motores a pasos en sus 12 salidas digitales de los puertos A y D. Además, se encarga de mantener la comunicación con la computadora PC mediante el puerto USB en donde el algoritmo espera a recibir instrucciones de la interfase de usuario para ejecutar las ordenes en los motores a paso. También se disponen de dos salidas PWM con el propósito de variar la potencia en dos láseres de estado sólido.

Las cuatro placas descritas, están construidas en circuitos impresos del tipo PCB.

1.2 Descripción de *hardware*

El *hardware* de la figura 3 está construido sobre la base de componentes discretos de fácil adquisición en el mercado nacional.

1.2.1 Puentes H

Las tres placas, puente H, de la figura 3 tienen como base la implementación de un par de puentes H con transistores TBJ complementarios como se muestra en la figura 4.

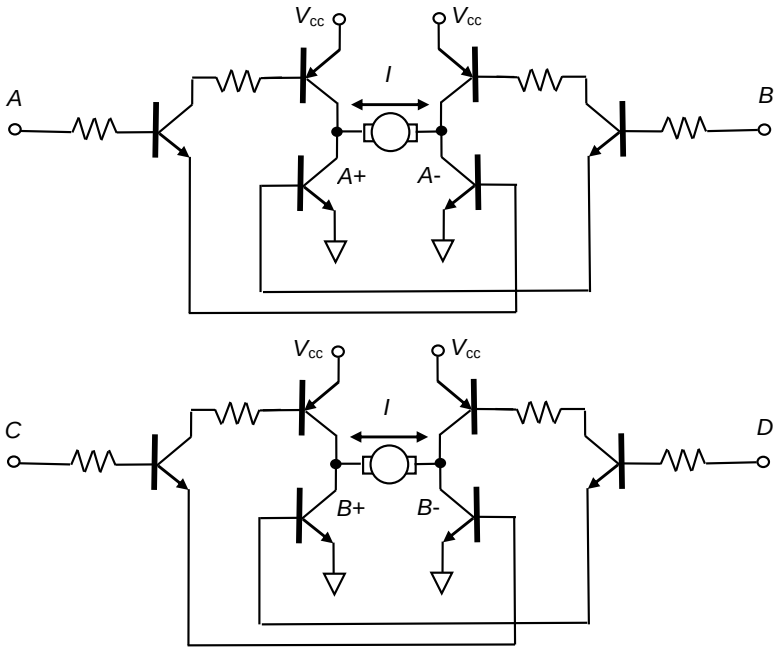

Figura 4. Puente H.

El par de puentes H son utilizados para energizar los dos embobinados de cada motor a pasos, *A*± y *B*±, posibilitando de esta manera el giro horario y antihorario mediante la alternancia de la corriente *I* y la secuencia de la tabla 1 en las señales de control digital.

A	B	C	D
0	0	0	1
0	1	0	0
0	0	1	0
1	0	0	0

Tabla 1. Secuencia de las señales de control para un motor a pasos.

Las señales de control digital son generadas en el algoritmo que se ejecuta en el microcontrolador Arduino Nano por medio de las salidas digitales en los puertos A y D de este.

El anexo A contiene el diagrama electrónico detallado de la placa de puentes H, el anexo B contiene el dibujo de su respectivo circuito impreso fabricado y el anexo C contiene la lista de partes electrónicas utilizadas.

1.2.2 Microcontrolador Arduino Nano

El microcontrolador Arduino Nano es una plataforma completa de desarrollo de 8 bits con base en el procesador ATmega328 (Arduino, 2021).

En la presente aplicación, el Arduino Nano se utiliza como interfase entre el *software* de usuario en la PC y los motores a pasos que mueven la platina en tres ejes. Adicionalmente, en el Arduino Nano se disponen de dos señales PWM que eventualmente pueden controlar la potencia en dos láseres de estado sólido y de dos entradas para interruptores límite que nos son utilizados en la presente aplicación. El esquema completo de conexiones se muestra en la figura 5.

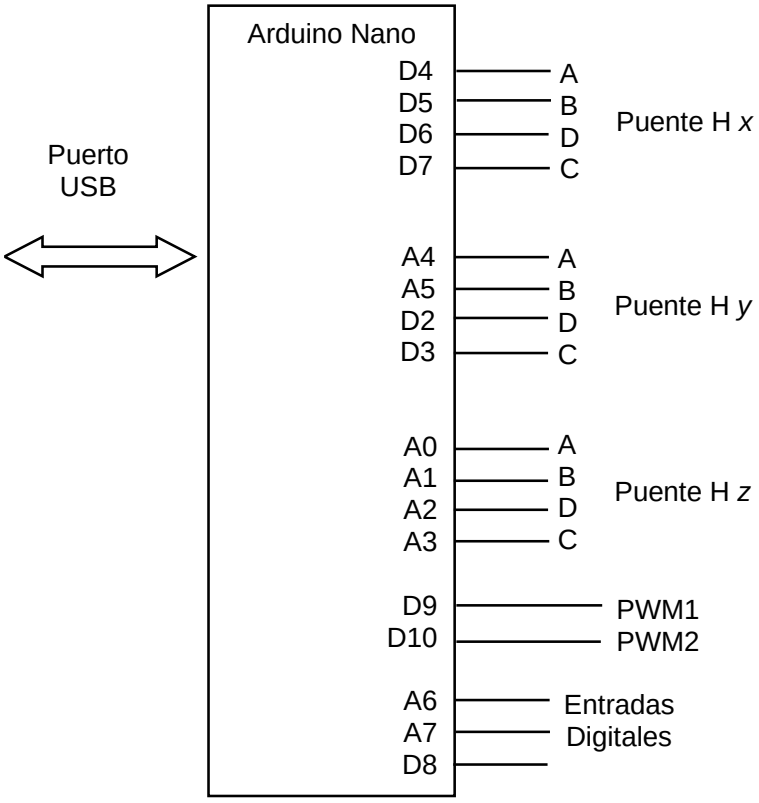

Figura 5. Arduino Nano.

El anexo A contiene el diagrama electrónico detallado de la placa para el Arduino Nano, el anexo B contiene el dibujo de su respectivo circuito impreso fabricado y el anexo C contiene la lista de partes electrónicas utilizadas.

1.3 Descripción del programa en el Arduino Nano

El programa en el Arduino Nano ejecuta los algoritmos de control y comunicación para mover los motores que posicionan las platinas milimétricas. El programa se desarrolló utilizando el IDE de Arduino programando en lenguaje C (Arduino, 2022). El programa es un ciclo cerrado en espera de recibir un comando proveniente del *software* de operación en la PC para ejecutar una actividad predeterminada. Cuando el algoritmo esta en estado de reposo en espera de ejecutar una tarea, el consumo de potencia en los motores se interrumpe con el objetivo de evitar sobrecalentamientos en el medio ambiente. Adicionalmente, se configura la interrupción del temporizador Timer2 para realimentar al *software* de operación con la posición actual de los motores. La figura 6 muestra el diagrama de flujo del programa en el Arduino Nano.

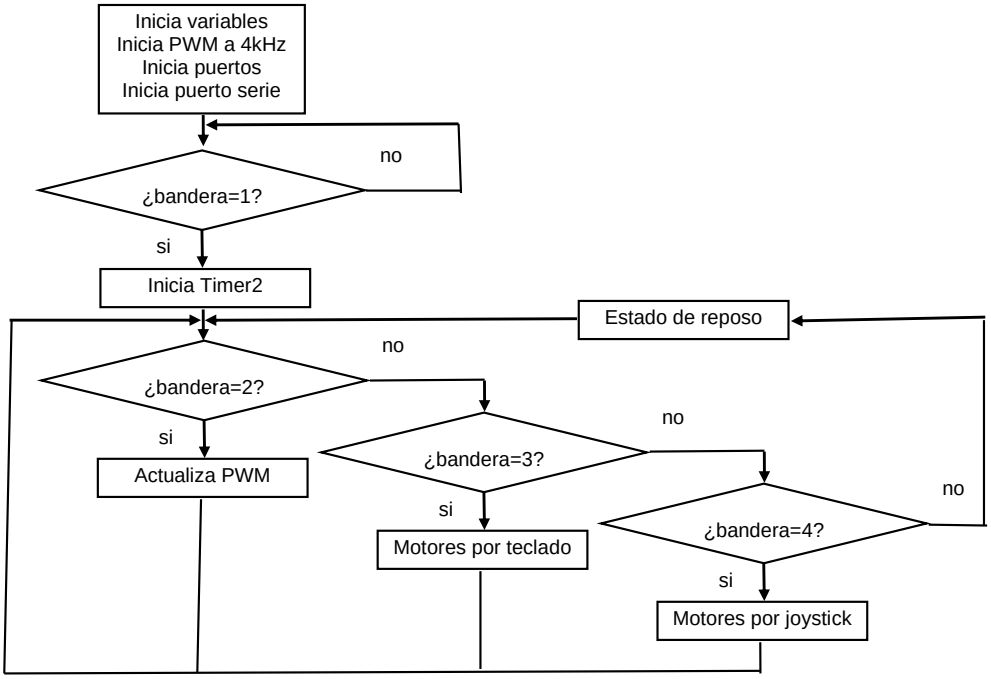

Figura 6. Programa en el Arduino Nano.

El programa que ejecuta el algoritmo de la figura 6 se organiza en las siguientes funciones del código ejecutable:

- setup(). Se ejecuta al inicio por única ocasión. Aquí se establece la inicialización del programa con la primera sentencia condicional en donde se compara la bandera recibida por puerto serie con “1” y se inicia el Timer2.
- loop(). Como su nombre lo indica, es el lazo principal del programa en donde se compara la bandera recibida por puerto serie para ejecutar la tarea indicada, como se muestra en la figura 6.

- pasox(int índice). Es la función que genera la secuencia de la tabla 1 para el movimiento del motor *x*.
- pasoy(int índice). Es la función que genera la secuencia de la tabla 1 para el movimiento del motor *y*.
- pasoz(int índice). Es la función que genera la secuencia de la tabla 1 para el movimiento del motor *z*.
- incrementa(int índice). Es la función que incrementa el índice entre 0 y 3 para generar la secuencia de la tabla 1.
- decrementa(int índice). Es la función que decrementa el índice entre 0 y 3 para generar la secuencia de la tabla 1.
- ISR_Serie(). Es la función de atención a la interrupción por desbordamiento del Timer2 en donde se transmiten las posiciones de los motores al *software* de operación por puerto serie.

El código completo del programa en el Arduino Nano se puede consultar en el anexo F del presente informe.

2 Descripción del *software* de operación

El *software* de operación permite al usuario manipular el sistema electrónico para posicionar las tres platinas desde una computadora con sistema operativo Windows 8 y 10.

2.1 Descripción en el ámbito del operador

La figura 7 muestra la pantalla principal del *software* de operación, denominado “Comm”, desarrollado específicamente para el control de posición de tres platinas milimétricas y la variación de potencia en dos láseres de estado sólido.

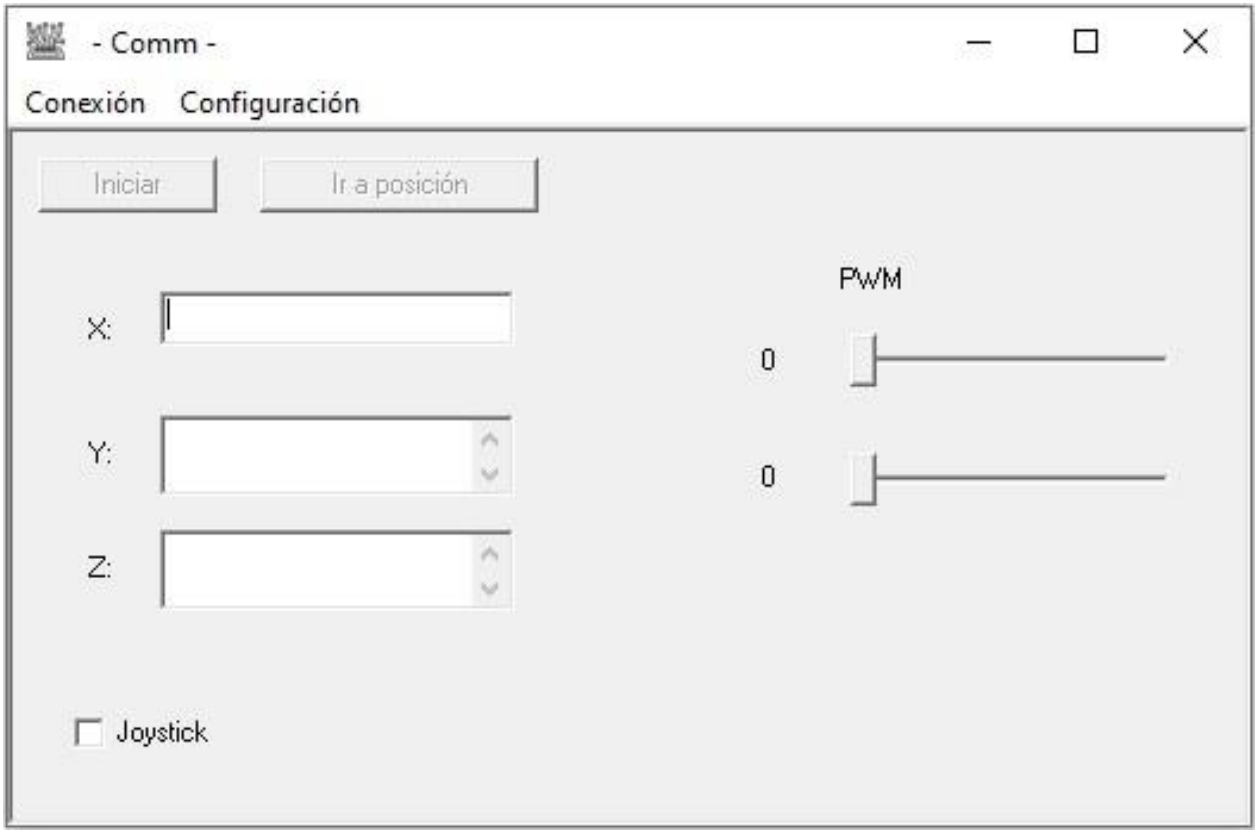

Figura 7. Pantalla principal del software de operación.

En esta pantalla, el usuario primero deberá establecer la conexión con el puerto detectado por el sistema operativo para el sistema electrónico al presionar sobre el menú “Configuración”, como se muestra en la figura 8.

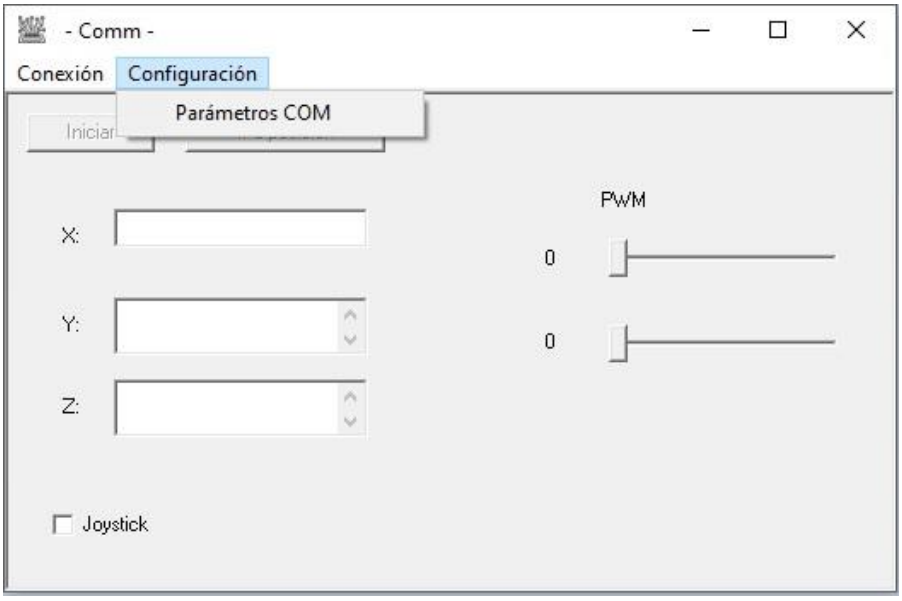

Figura 8. Opciones del menú "Configuración".

Al seleccionar la opción "Parámetros COM" se desplegará el diálogo "Configuración" de la figura 9 en donde se seleccionarán los parámetros mostrados para el puerto en donde se detectó el sistema electrónico.

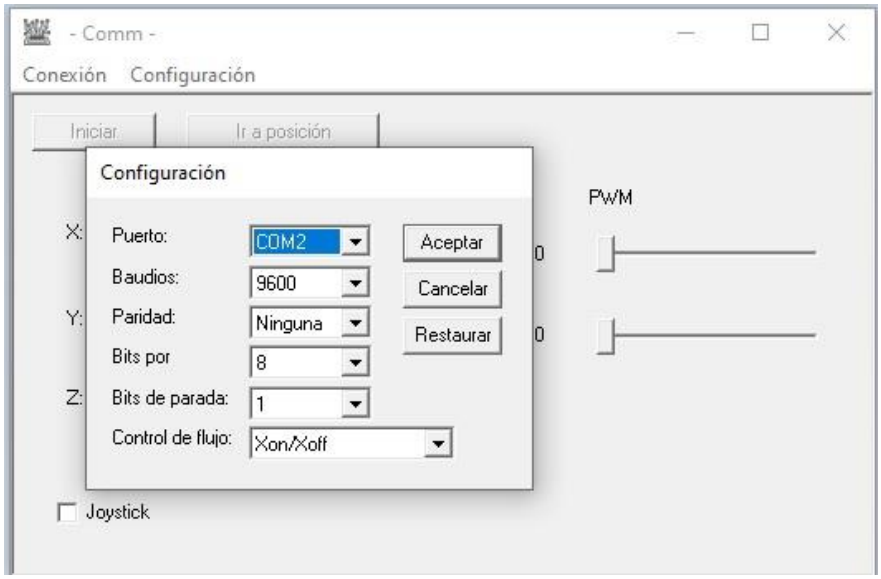

Figura 9. Caja de diálogo "Configuración".

Al presionar el botón "Aceptar", si la conexión fue exitosa, se desplegará el diálogo de la figura 10 indicando que el puerto de comunicaciones fue abierto exitosamente y el programa puede continuar. En caso de no desplegarse el diálogo de la figura 10, se deberá repetir el proceso procurando ingresar los valores apropiados en el diálogo de la figura 9.

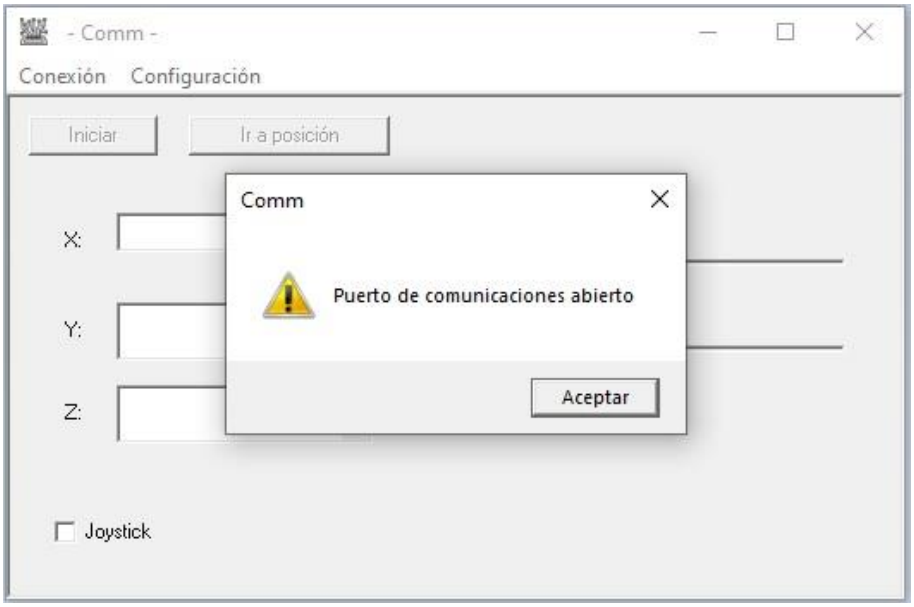

Figura 10. Puerto de comunicaciones abierto exitosamente.

Al aceptar el diálogo de la figura 10, el botón “Iniciar” de la pantalla principal deberá aparecer habilitado como se muestra en la figura 11.

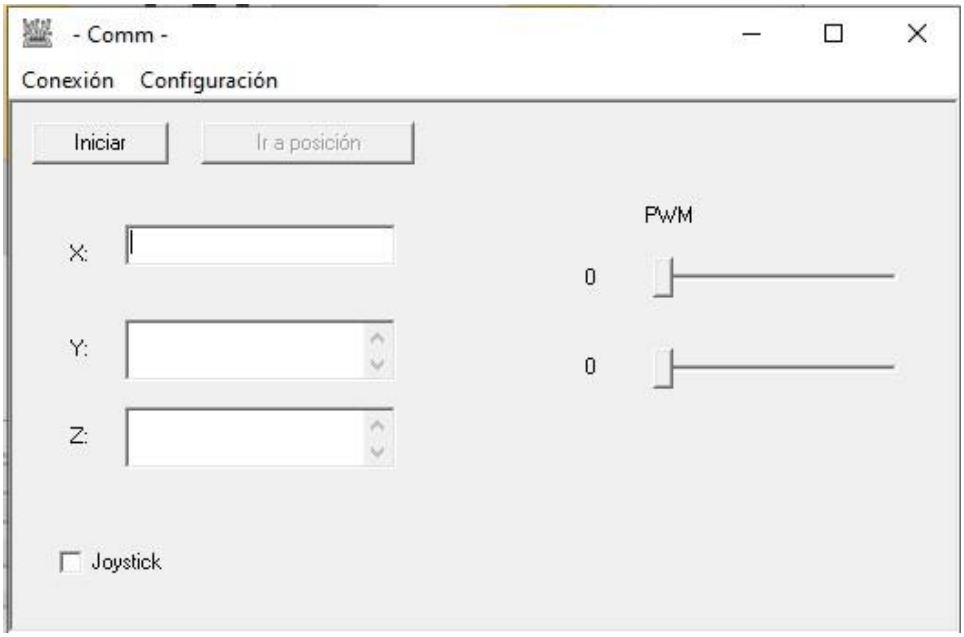

Figura 11. Botón “Iniciar” habilitado.

En este momento se podrá establecer comunicación con el sistema electrónico al presionar el botón “Iniciar”. Después de presionar este último botón, se desplegará la

pantalla de la figura 12 que indica la posición cero absoluta del sistema y la posibilidad de desplazar hacia una nueva posición mediante teclado o *joystick*.

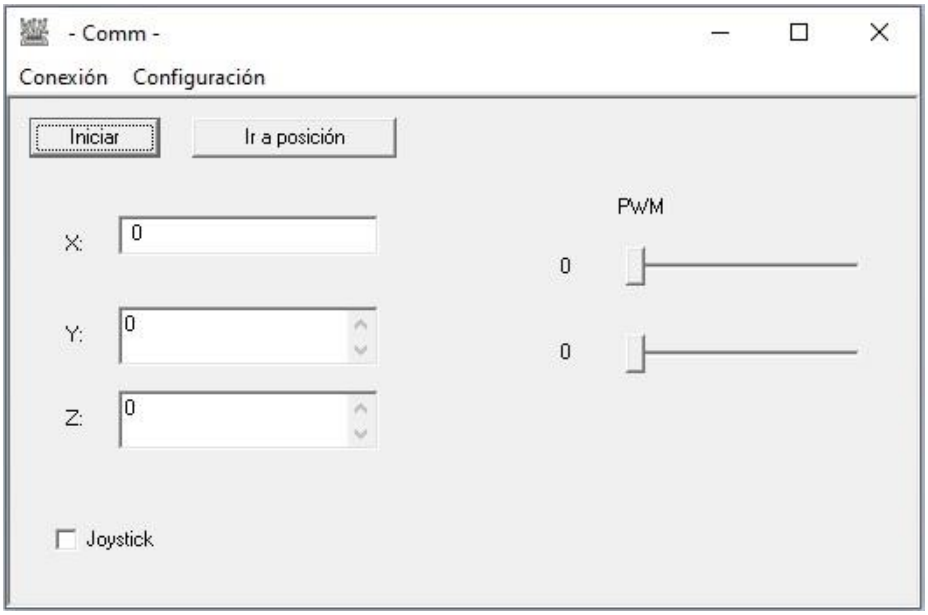

Figura 12. Pantalla indicando que el sistema de posicionamiento puede iniciar operaciones.

Un ejemplo para posicionar el sistema bajo las condiciones indicadas consiste en presionar el botón “Ir a posición” e ingresar por teclado la nueva posición que se desea establecer en el diálogo “Ir a posición”, como se muestra en la figura 13.

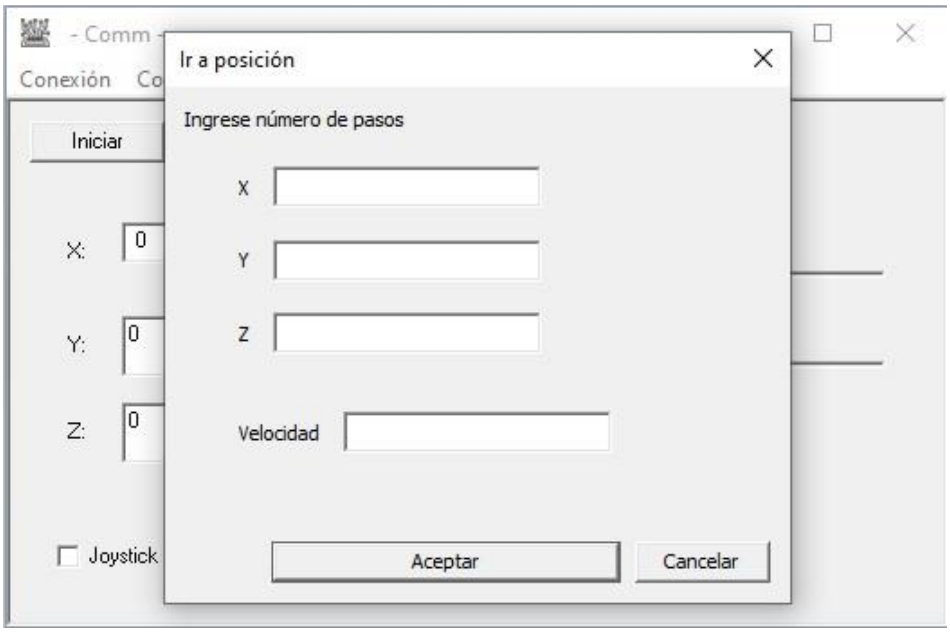

Figura 13. Diálogo “Ir a posición”.

En el diálogo de la figura 13 se ingresan el número de pasos que se desplazará el sistema a partir de la posición actual y el tiempo en milisegundos por paso asignado a la variable “Velocidad”, como se muestra en la figura 14.

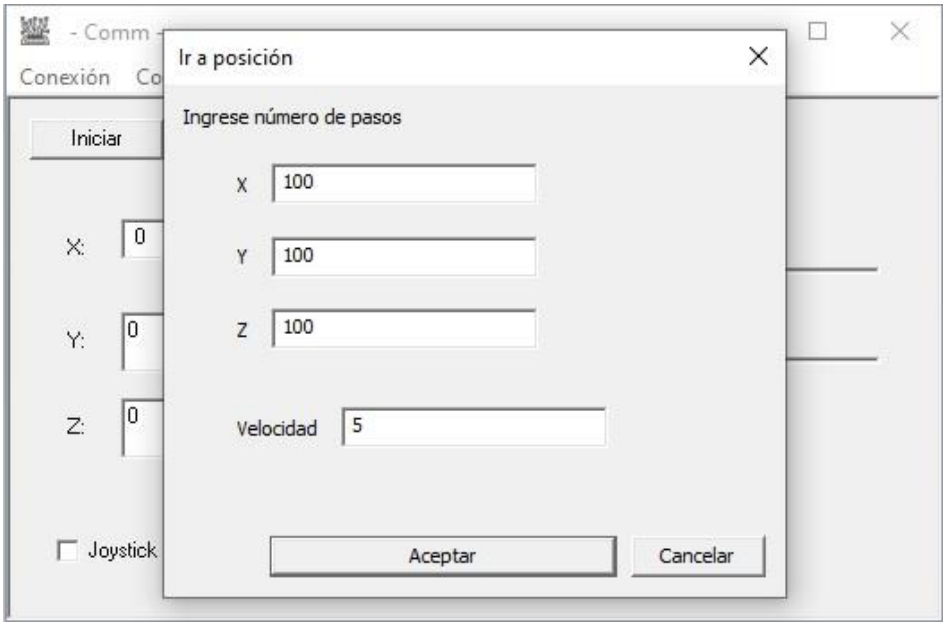

Figura 14. Nueva posición en el diálogo “Ir a posición”.

El resultado de establecer una nueva posición por teclado se muestra en la figura 15.

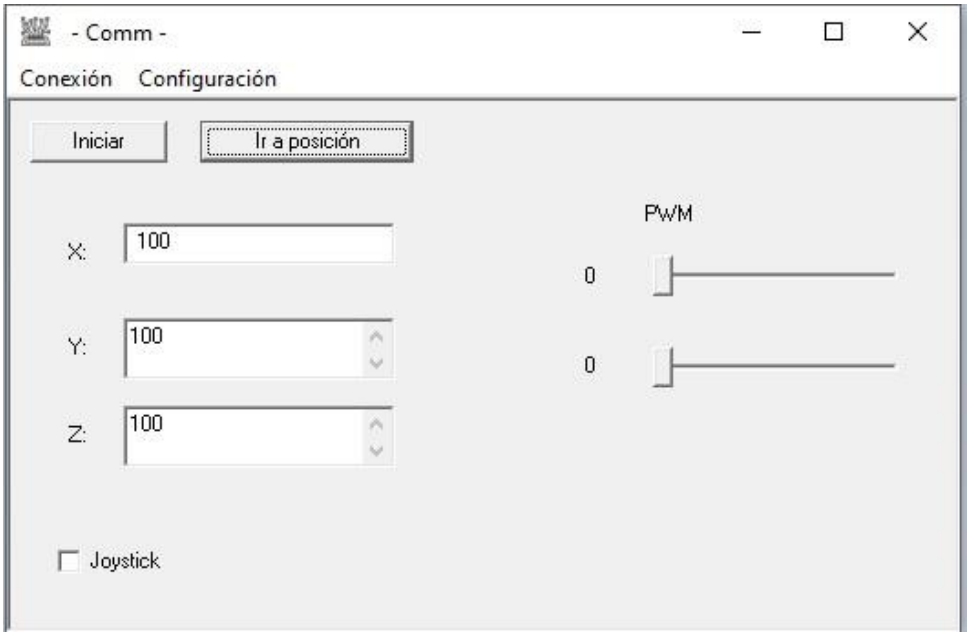

Figura 15. Nueva posición en el sistema.

Otro ejemplo de posicionamiento para el sistema se realiza mediante la ayuda de la palanca de mandos o *joystick* al habilitar la selección “Joystick” de la pantalla principal y mover los mandos de control manualmente, como se muestra en la figura 16.

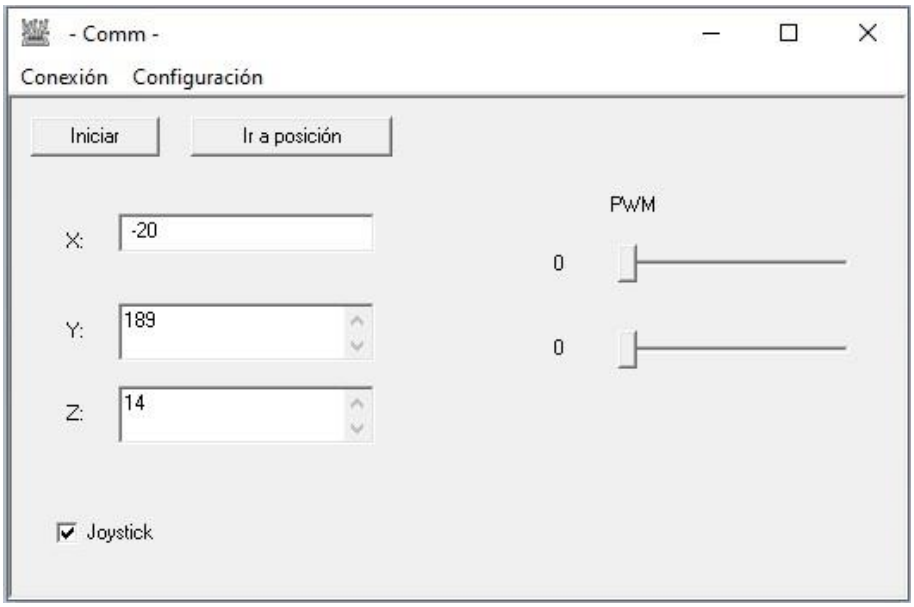

Figura 16. Nueva posición mediante Joystick.

Los dos controles deslizables, localizados en la parte derecha de la pantalla, se utilizan para variar las señales de PWM en las salidas analógicas D9 y D10 del microcontrolador Arduino Nano. Un ejemplo para la señal de PWM en D9 se muestra en las figuras 17 y 18, las cuales corresponden a un PWM de 11%.

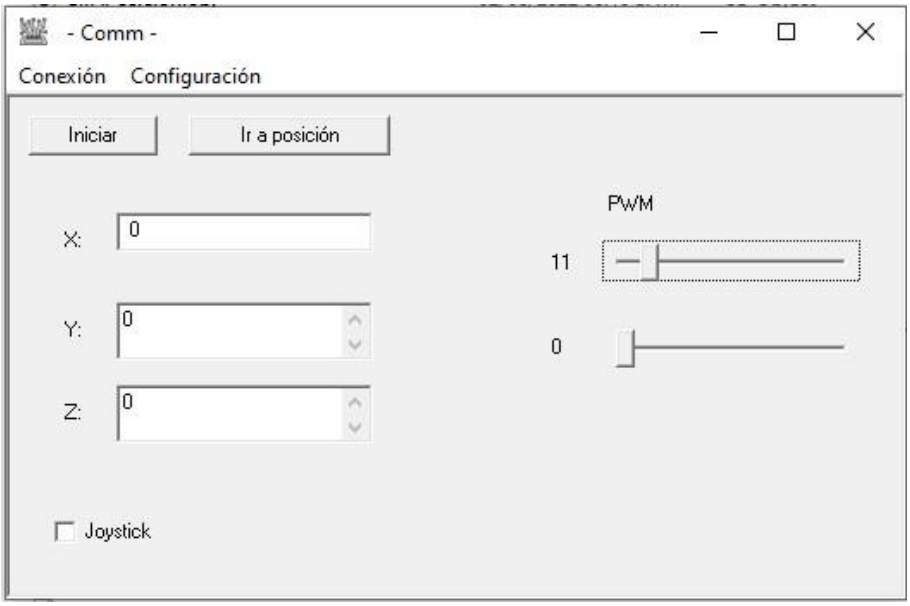

Figura 17. PWM 11% en la pantalla principal.

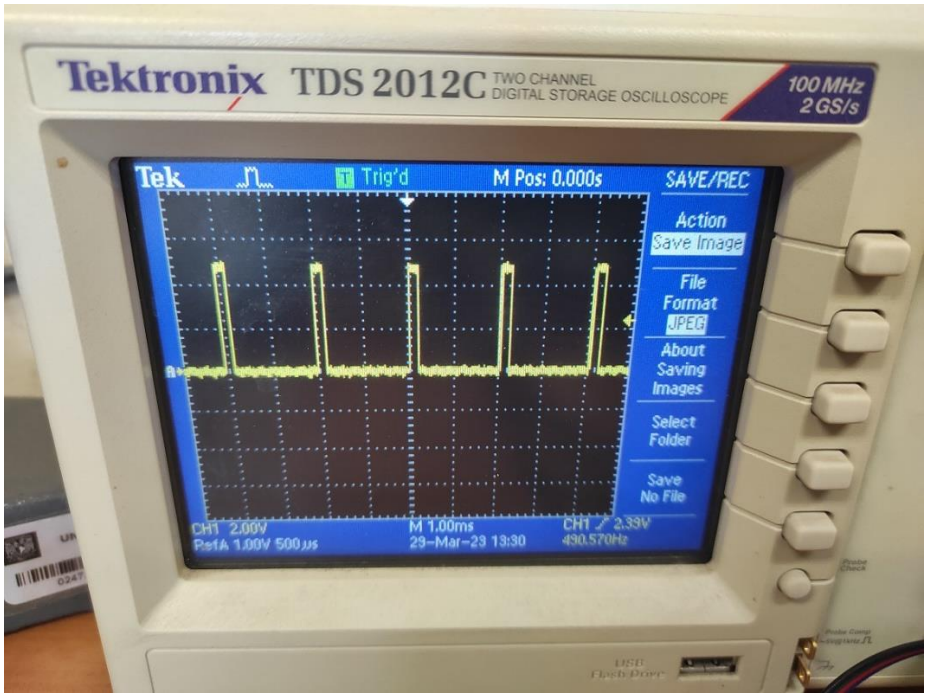

Figura 18. PWM 11% en la salida D9.

Otro ejemplo consiste en generar un PWM de 84% utilizando la misma estrategia. El resultado se observa en las figuras 19 y 20.

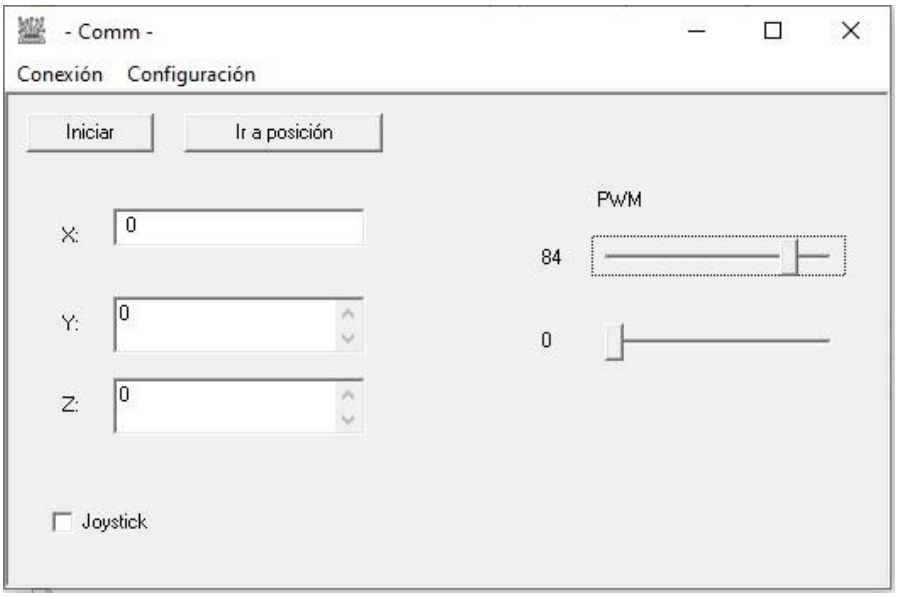

Figura 19. PWM 84% en la pantalla principal.

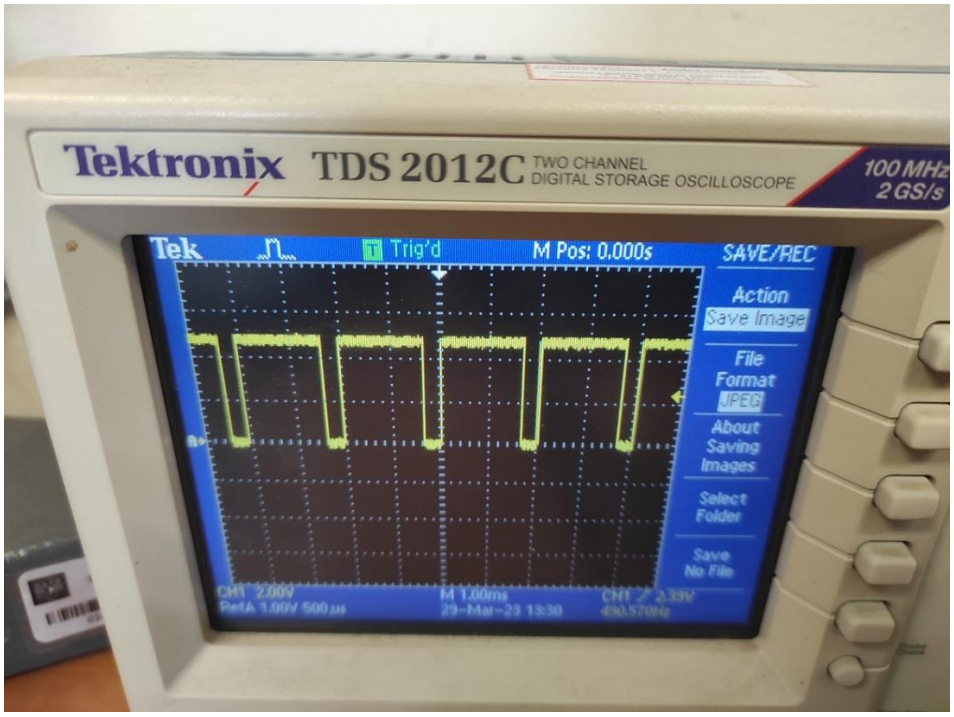

Figura 20. PWM 84% en la salida D9.

2.2 Descripción en el ámbito del programador

El programa ejacutable “Comm” es el resultado de un proyecto de programación llamado “Solución” de Visual Studio 2020. Fue desarrollado en lenguaje C utilizando la MFC que soporta el tipo de aplicaciones desarrolladas en ambiente Windows. La figura 21 muestra la solución “Comm” en el ambiente integrado de desarrollo Visual Studio 2020 (Kruglinski, 1988).

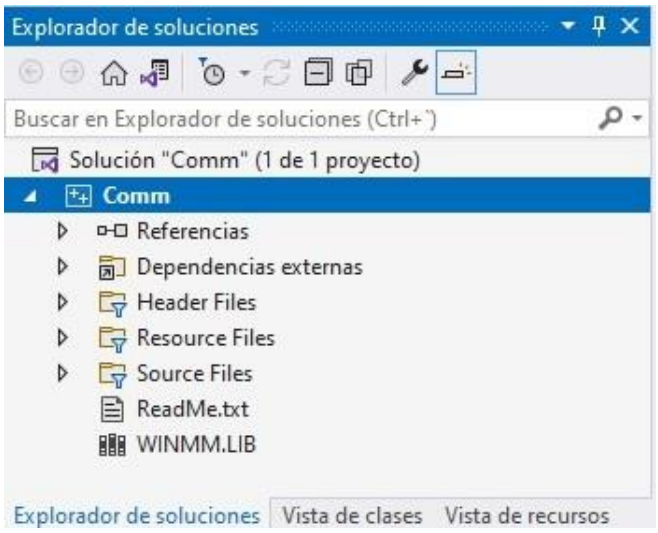

Figura 21. Solución “Comm”.

Se observa que la solución está integrada por diversos archivos clasificados como cabecera, código, recursos y el elemento existente WINMM.LIB, agregado a la solución, que es la librería que proporciona el soporte para el manejo del *joystick*. Los archivos de la solución son generados y administrados por el IDE de Visual Studio 2020 en el momento de crear un nuevo proyecto sobre la base de “Aplicación de documento sencillo”. El código completo se puede consultar en el anexo E del presente informe.

El IDE crea el esqueleto de la aplicación conteniendo las clases que se muestran en la figura 22. Estas clases son insertadas por omisión por el IDE de Visual Studio con excepción de la clase CParamComm que es exclusiva de esta solución. En el presente reporte organizamos de la siguiente manera la descripción de las clases que componen la solución:

- Clases insertadas por omisión. Son las clases generadas automáticamente por el IDE pero que no fueron modificadas de acuerdo con la funcionalidad de la aplicación.
- Clase principal para el soporte de la solución. Es la clase que, aunque fue generada automáticamente por el IDE, merece una atención especial ya que es la clase que contiene el algoritmo de comunicaciones y control de la presente aplicación.
- Clase exclusiva de la solución. Es la clase creada particularmente para la presente aplicación.

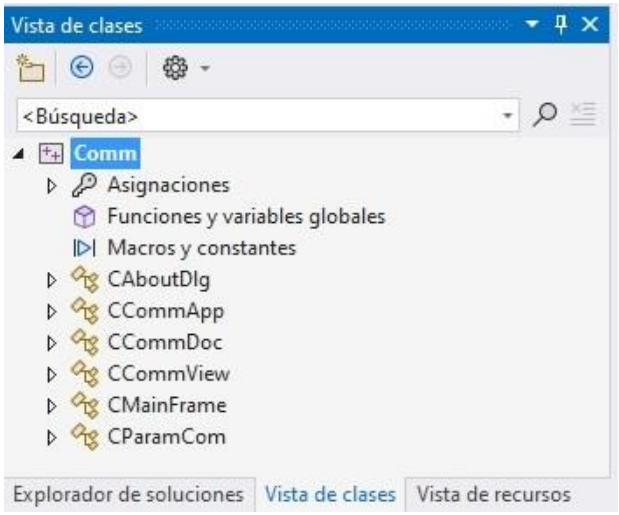

Figura 22. Clases de la solución “Comm”.

2.2.1 Clases insertadas por omisión

CAboutDlg

Los métodos de la clase se muestran en la figura 23.

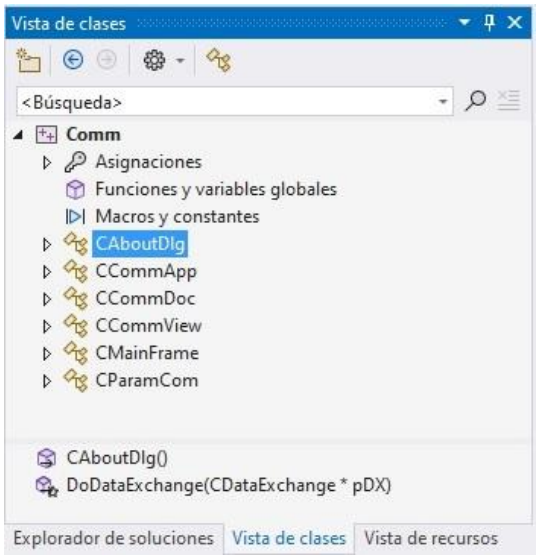

Figura 23. Clase CAboutDlg.

Los métodos de esta clase son:

CAboutDlg(). Es el constructor de la clase. No fue modificado de acuerdo con su origen por omisión.

DoDataExchange(CDataExchange* pDX). Es el método que permite el intercambio dinámico de datos. No fue modificado de acuerdo con su origen por omisión.

CCommApp

Los métodos de la clase se muestran en la figura 24.

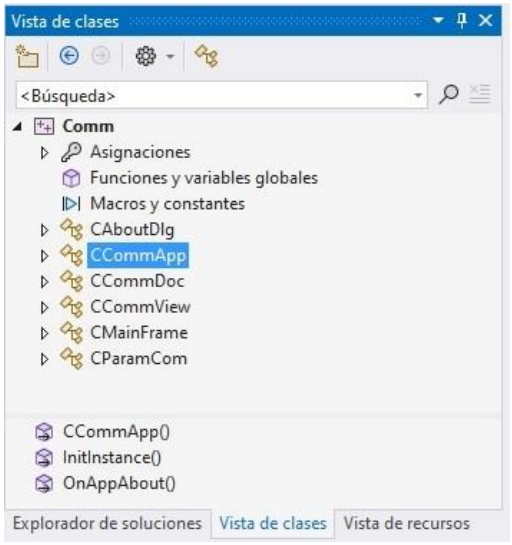

Figura 24. Clase CCommApp.

Los métodos de esta clase son:

CCommApp(). Es el constructor de la clase. No fue modificado de acuerdo con su origen por omisión.

InitInstance(). Es el método que realiza la inicialización estándar para una aplicación de documento sencillo. No fue modificado de acuerdo con su origen por omisión.

OnAppAbout(). Es el método que atiende el evento por generar la caja de diálogo “Acerca de…”. No fue modificado de acuerdo con su origen por omisión.

CComDoc

Los métodos de la clase se muestran en la figura 25.

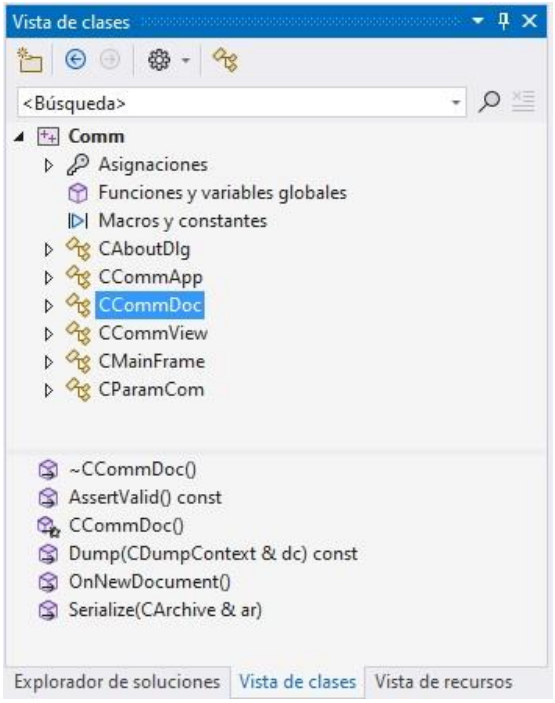

Figura 25. Clase CCommDoc.

Los métodos de esta clase son:

~CCommDoc(). Es el destructor de la clase. No fue modificado de acuerdo con su origen por omisión.

AssertValid(). Verifica la integridad del objeto. No fue modificado de acuerdo con su origen por omisión.

Dump(CDumpContext& dc). Proporciona servicios de diagnóstico. No fue modificado de acuerdo con su origen por omisión.

OnNewDocument(). Es el método que atiende el comando por generar un documento nuevo en una aplicación de documento sencillo. No fue modificado de acuerdo con su origen por omisión.

Serialize(CArchive& ar). Es el método que proporciona soporte para la serialización de documentos, es decir, lectura y escritura. No fue modificado de acuerdo con su origen por omisión.

CMainFrame

Los métodos de la clase se muestran en la figura 26.

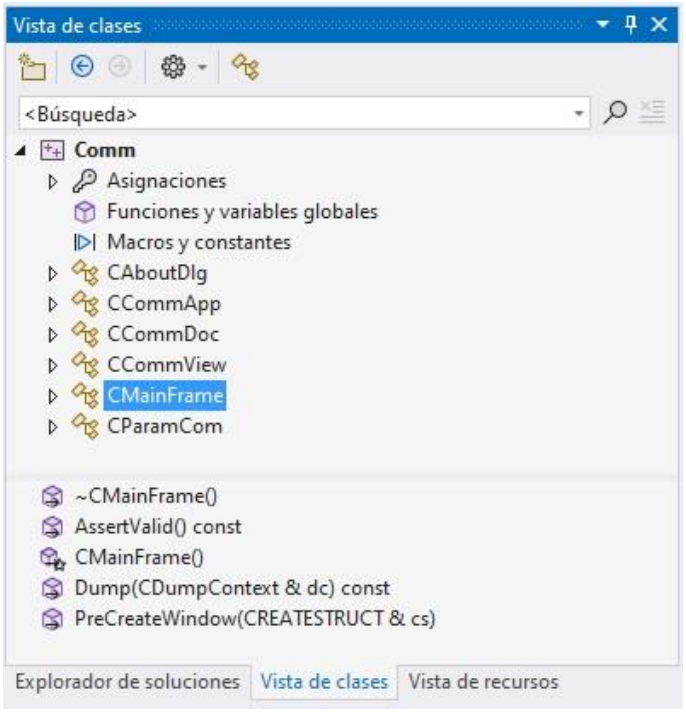

Figura 26. Clase CMainFrame.

Los métodos de esta clase son:

~CMainFrame(). Es el destructor de la clase. No fue modificado de acuerdo con su origen por omisión.

AssertValid(). Verifica la integridad del objeto. No fue modificado de acuerdo con su origen por omisión.

CMainFrame(). Es el constructor de la clase. No fue modificado de acuerdo con su origen por omisión.

Dump(CDumpContext& dc). Proporciona servicios de diagnóstico. No fue modificado de acuerdo con su origen por omisión.

PreCreateWindow(CREATESTRUCT& cs). Proporciona acceso a la creación de la aplicación. No fue modificado de acuerdo con su origen por omisión.

2.2.2 Clase principal para el soporte de la solución

Los métodos de la clase CCommView se muestran en la figura 27.

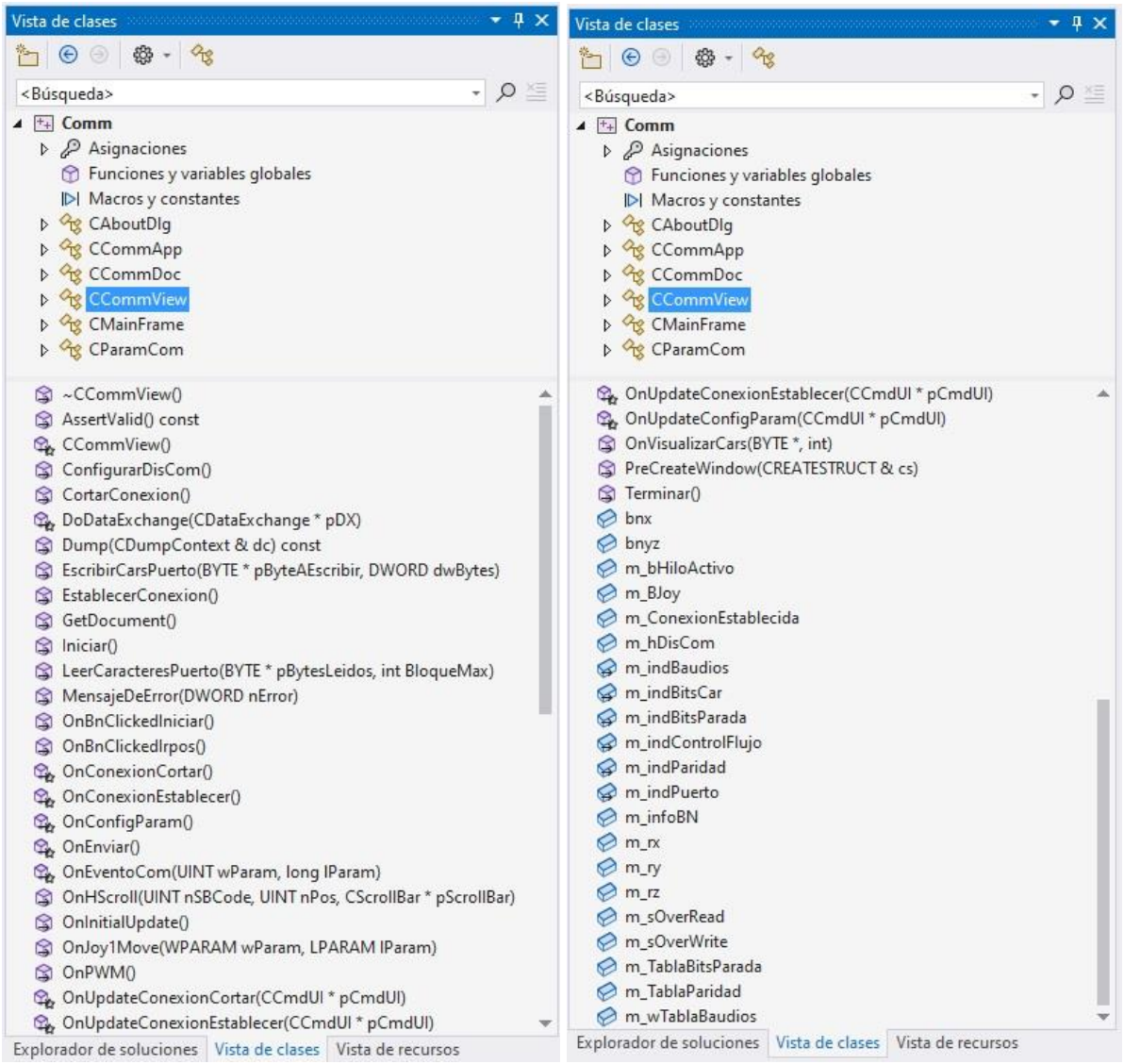

Figura 27. Clase CCommView.

Los métodos de esta clase son:

~CCommView(). Es el destructor de la clase. Corta la conexión establecida en el puerto serie.

AssertValid(). Verifica la integridad del objeto. No fue modificado de acuerdo con su origen por omisión.

CCommView(). Es el constructor de la clase. Inicializa variables globales de la aplicación.

ConfigurarDisCom(). Configura los parámetros de operación del puerto serie.

CortarConexion(). Corta la conexión con el puerto serie.

DoDataExchange(CDataExchange* pDX). Establece el intercambio de datos DDX/DDV.

Dump(CDumpContext& dc). Proporciona servicios de diagnóstico. No fue modificado de acuerdo con su origen por omisión.

EscribirCarsPuerto(BYTE *pByteAEscribir, DWORD dwBytes). Escribe la cadena de bytes apuntados por pByteAEscribir en el puerto serie.

EstablecerConexion(). Abre el puerto de comunicaciones y establece el hilo secundario ControlarEventos.

GetDocument(). Proporciona el soporte para el manejo de documentos. No se utiliza en la presente aplicación.

Iniciar(). Restablece los parámetros del puerto serie de acuerdo con la última vez que se configuró.

LeerCaracteresPuerto(BYTE *pBytesLeidos, int BloqueMax). Regresa el apuntador pBytesLeidos a la cadena de caracteres leídos por el puerto serie.

MensajeDeError(DWORD nError). Formatea y despliega cadenas que manipulan situaciones de error.

OnBnClickedIniciar(). Es el método que atiende el evento por pulsar sobre el botón “Iniciar”. Envía el comando “1” por puerto serie para inicializar el sistema electrónico. Verifica la instalación del *joystick*.

OnBnClickedIrpos(). Es el método que atiende el evento por pulsar sobre el botón “Ir a posición”. Despliega la caja de diálogo auxiliar “Ir a posición” para obtener del teclado los nuevos valores de posición para el sistema electrónico.

OnConexionCortar(). Es el método que atiende el evento por pulsar sobre la opción del menú “Cortar”. Corta la conexión con el puerto serie.

OnConexionEstablecer(). Es el método que atiende el evento por pulsar sobre la opción del menú “Establecer”. Manipula el establecimiento de la conexión con el puerto serie.

OnConfigParam(). Es el método que atiende el evento por pulsar sobre la opción del menú “Parámetros COM”. Despliega la caja de diálogo auxiliar “Configuración” para obtener los nuevos parámetros de comunicación para el puerto serie.

OnEnviar(). No tiene funcionalidad. Usada con fines de depuración.

OnEventoCom(UINT wParam, long lParam). Manipula los mensajes recibidos desde el hilo, principalmente los caracteres recibidos por puerto serie.

OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar). Manipula los eventos por desplazar los controles deslizables PWM.

OnInitialUpdate(). Establece las condiciones iniciales para la apariencia de la pantalla principal de la aplicación. Inicia el puerto serie.

OnJoy1Move(WPARAM wParam, LPARAM lParam). Realiza la lectura de los valores establecidos por el *joystick*. Formatea las cadenas a enviar por puerto serie al sistema electrónico.

OnPWM(). Formatea las cadenas de los controles deslizables para enviarlas por puerto serie al sistema electrónico.

OnUpdateConexionCortar(CCmdUI* pCmdUI). Renueva el estado de la opción del menú “Cortar”.

OnUpdateConexionEstablecer(CCmdUI* pCmdUI). Renueva el estado de la opción del menú “Establecer”.

OnUpdateConfigParam(CCmdUI* pCmdUI). Renueva el estado de la opción del menú “Parámetros COM”.

OnVisualizarCars(BYTE *, int). Renueva el valor de las cajas de texto “X”, “Y” y “Z” de la pantalla principal de la aplicación con los valores recibidos por puerto serie del sistema electrónico.

PreCreateWindow(CREATESTRUCT& cs). Proporciona acceso a la creación de la aplicación. No fue modificado de acuerdo con su origen por omisión.

Terminar(). Respalda los parámetros establecidos para el puerto serie por última vez.

Adicionalmente, el método ControlarEventos(LPVOID p) es la función establecida para atender el hilo secundario que responde a los eventos asíncronos que ocurren en el puerto serie de comunicaciones de la PC (Ceballos, 1999).

2.2.3 Clase exclusiva de la solución

La clase de la figura 28 soporta el intercambio de los parámetros de comunicaciones de la caja de diálogo “Configuración” para que sean establecidos en la clase principal que soporta la aplicación, CCommView. Sus métodos son los siguientes:

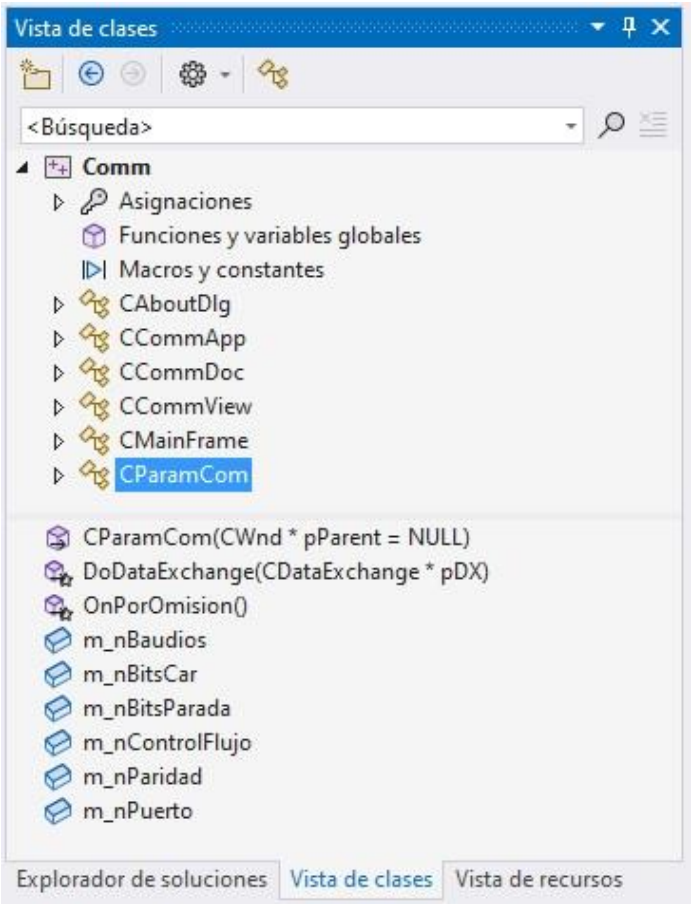

Figura 28. Clase CParamCom.

CParamCom(CWnd* pParent = NULL); Es el constructor estándar en donde se inicializan los parámetros de comunicaciones.

DoDataExchange(CDataExchange* pDX); Es el método que soporta el intercambio de datos DDX/DDV para transferir los valores asignados a los parámetros de comunicaciones con otras clases de la solución.

OnPorOmision(); Es el método que asifna a los parámetros de comunicaciones valores por omisión.

2.3 Procedimiento de instalación

A continuación, se presenta una guía de conexiones detalladas para utilizar el sistema mecatrónico reportado en este informe.

2.3.1 Conexiones

La figura 29 muestra las conexiones necesarias para operar el sistema electrónico para el posicionamiento milimétrico de tres platinas milimétricas mediante una PC.

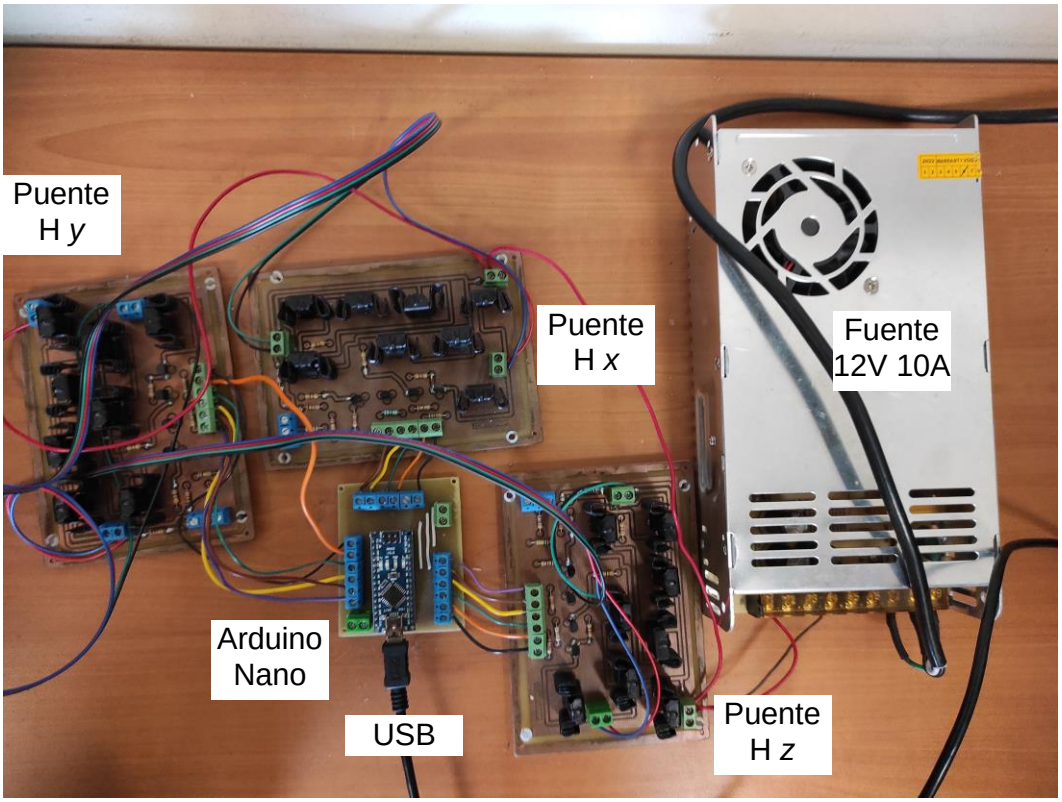

Figura 29. Conexiones del sistema electrónico.

Un diagrama esquemático para las conexiones de la figura 29 se muestra en la figura 30.

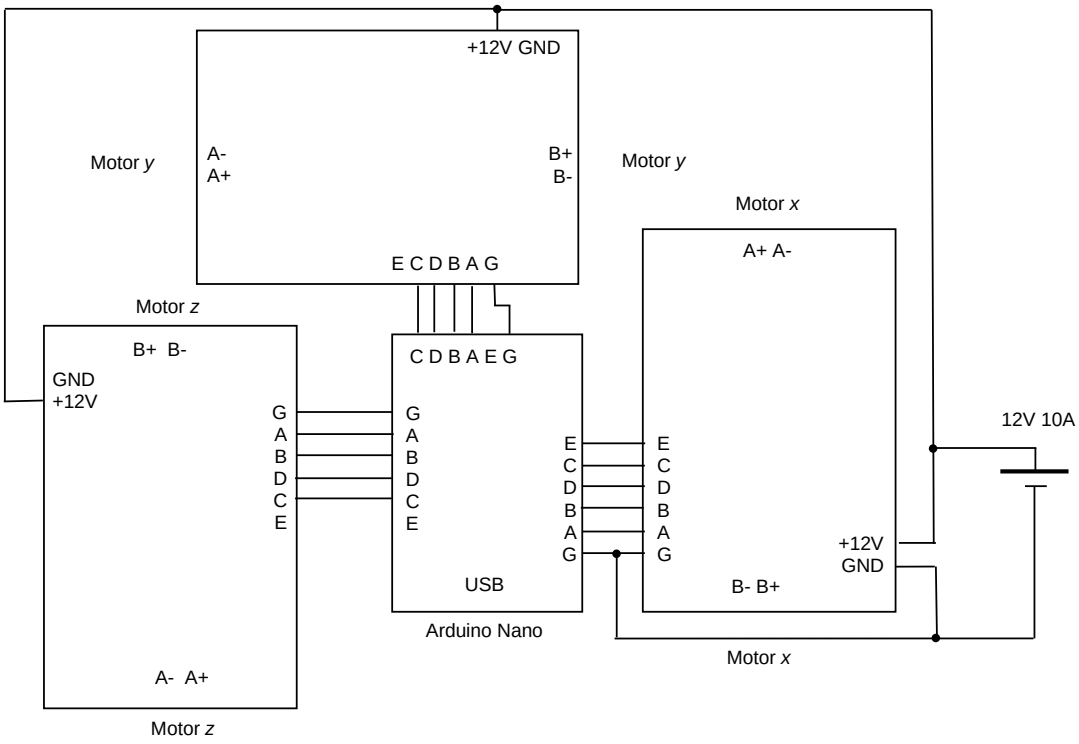

Figura 30. Diagrama esquemático para las conexiones del sistema electrónico.

2.3.2 Instalación del microcontrolador Arduino Nano

La tarjeta Arduino Nano es la parte principal del sistema electrónico para el posicionamiento milimétrico y es en donde residen los algoritmos de control y comunicaciones. El procedimiento de instalación en una computadora es muy sencillo y se explica detalladamente a continuación.

Paso 1. Descargar e instalar el IDE de Arduino (Arduino, 2022).

Paso 2. Conectar el Arduino Nano a la computadora mediante el cable mini USB.

Paso 3. Verificar el puerto que le asigno el sistema operativo Windows al Arduino Nano mediante la utilería “Administrador de dispositivos”, como se muestra en la figura 31.

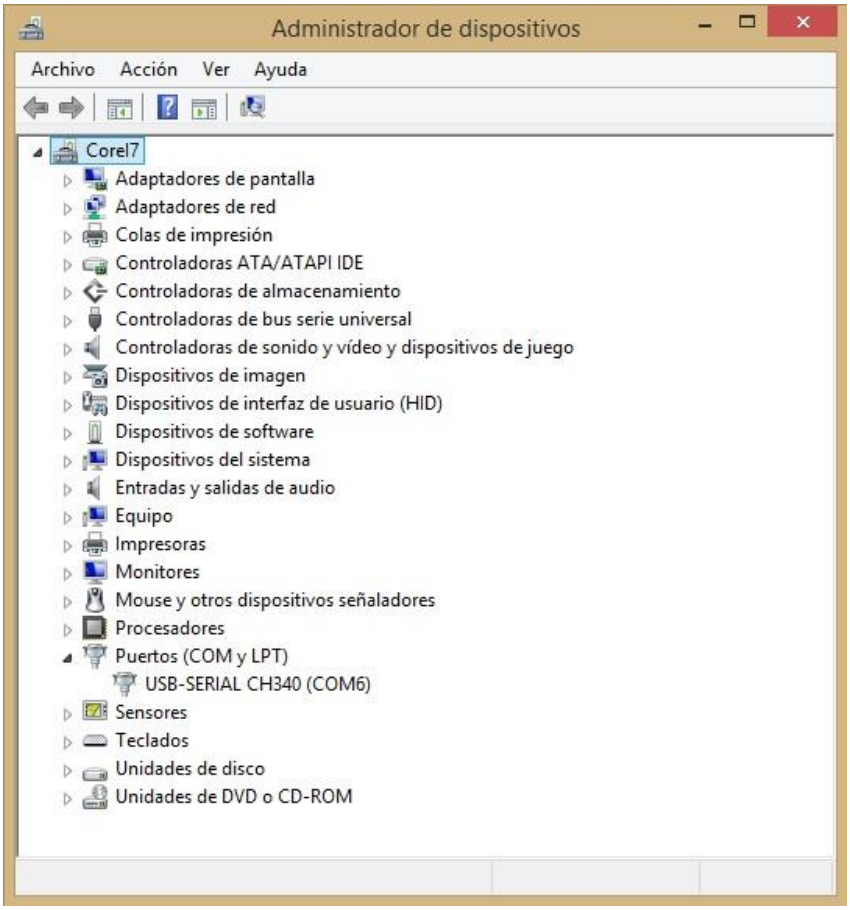

Figura 31. Administrador de dispositivos detectando el Arduino Nano.

En el caso de la figura 31, los parámetros que se deberán ingresar en la opción “Parámetros COM” del menú “Configuración” deberán ser los siguientes:

Puerto: COM6.
Baudios: 9600.
Paridad: Ninguna.
Bits por: 8.
Bits de parada: 1.
Control de flujo: Xon/Xoff.

3 Resultados y discusión

En este capítulo se exponen los resultados cualitativos y discusión para el sistema de posicionamiento milimétrico.

3.1 Resultados

Los principales resultados del presente proyecto son el prototipo construido y el *software* desarrollado. Ambos se construyeron a partir de herramientas básicas de desarrollo como lo son el microcontrolador Arduino Nano y Visual Studio como plataforma de programación.

3.1.1 Prototipo desarrollado

La figura 32 muestra el sistema de posicionamiento milimétrico en tres ejes.

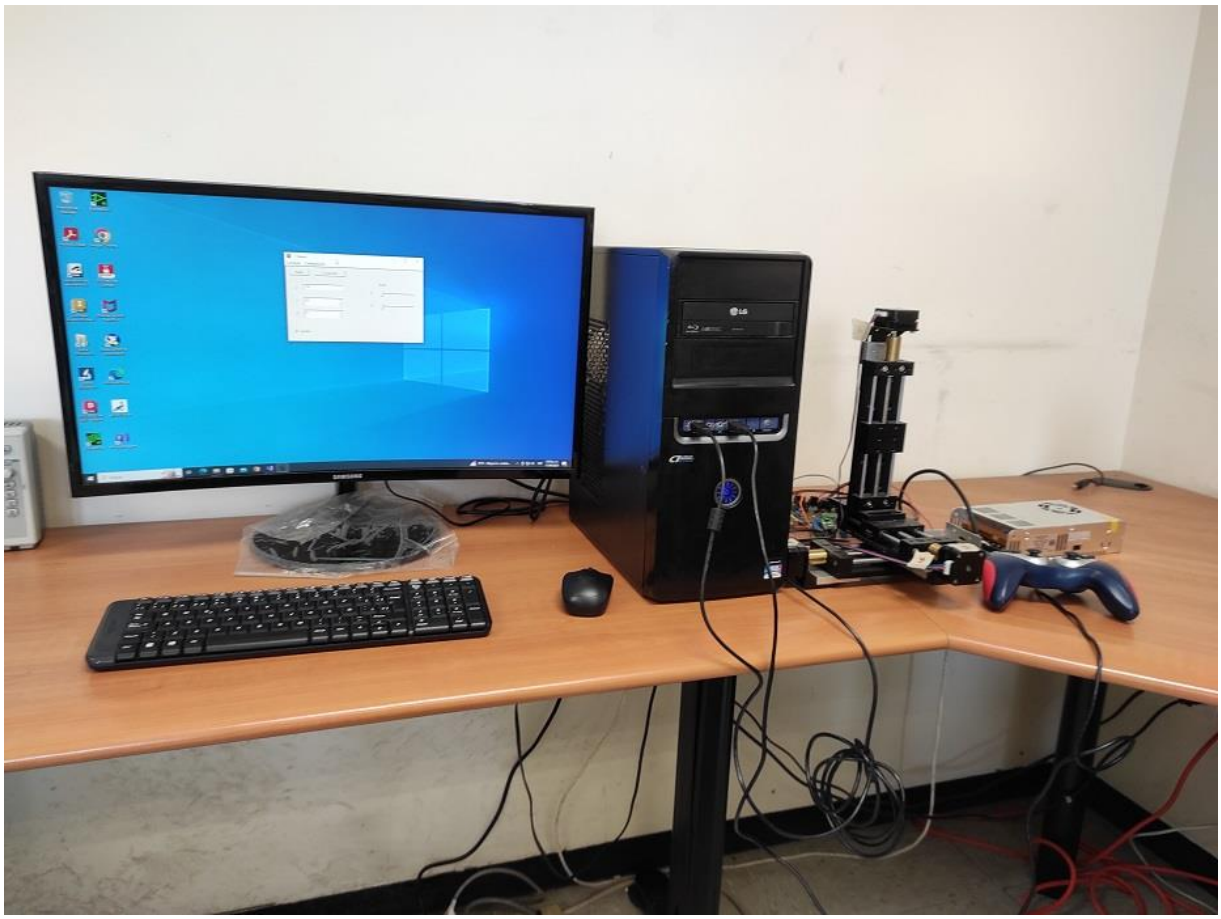

Figura 32. Prototipo hardware y software.

La figura 33 muestra con detalle las tres platinas milimétricas acopladas a los motores a paso. El anexo F muestra los planos de los acoplamientos mecánicos utilizados para conectar los motores a las platinas.

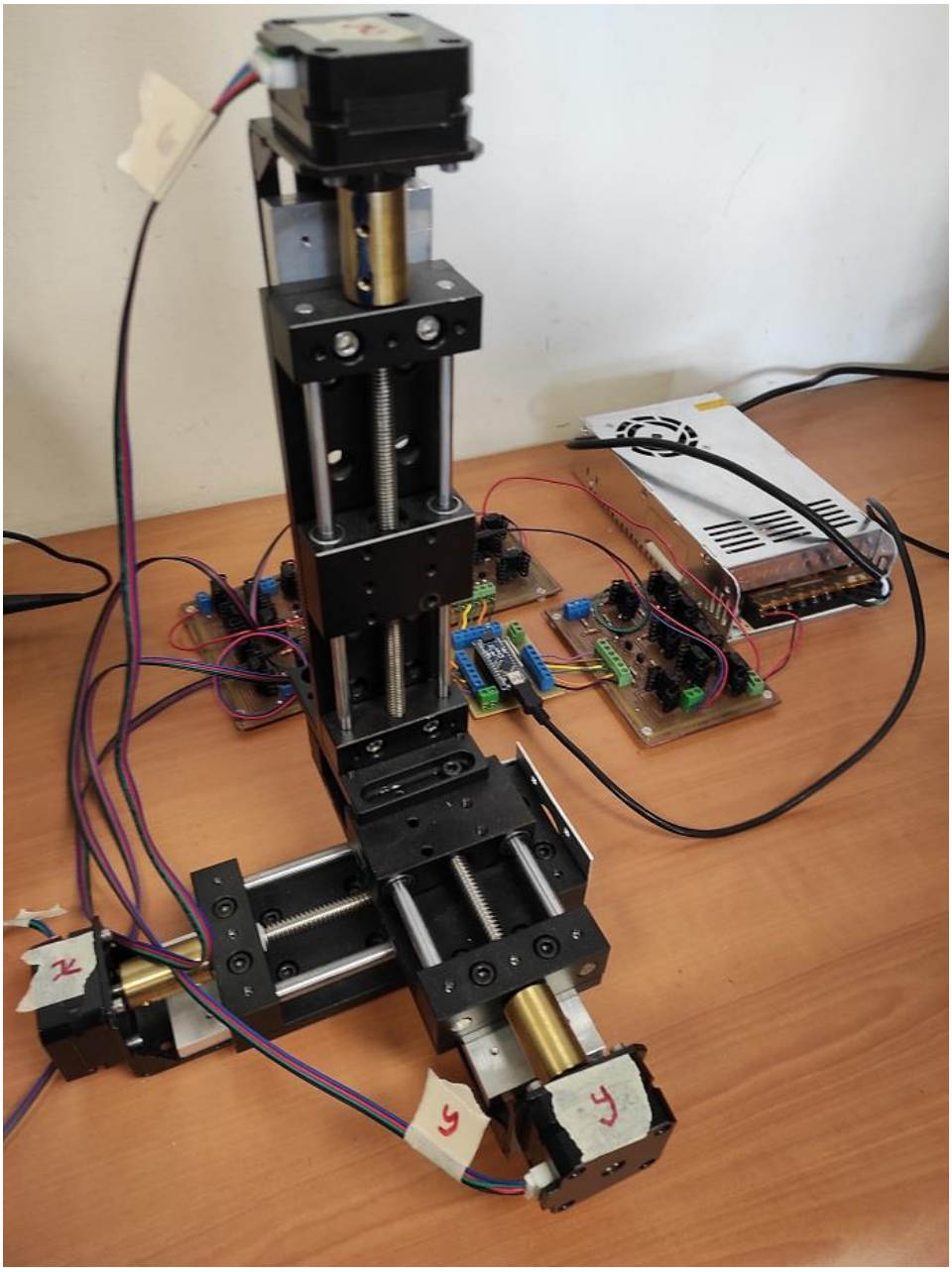

Figura 33. Platinas milimétricas y motores a paso.

Con este sistema se tiene contemplado el posicionamiento y control de los diversos dispositivos que componen un PIV.

3.1.2 *Software* de operación

La figura 34 muestra la pantalla principal del software de operación ejecutándose en tiempo real.

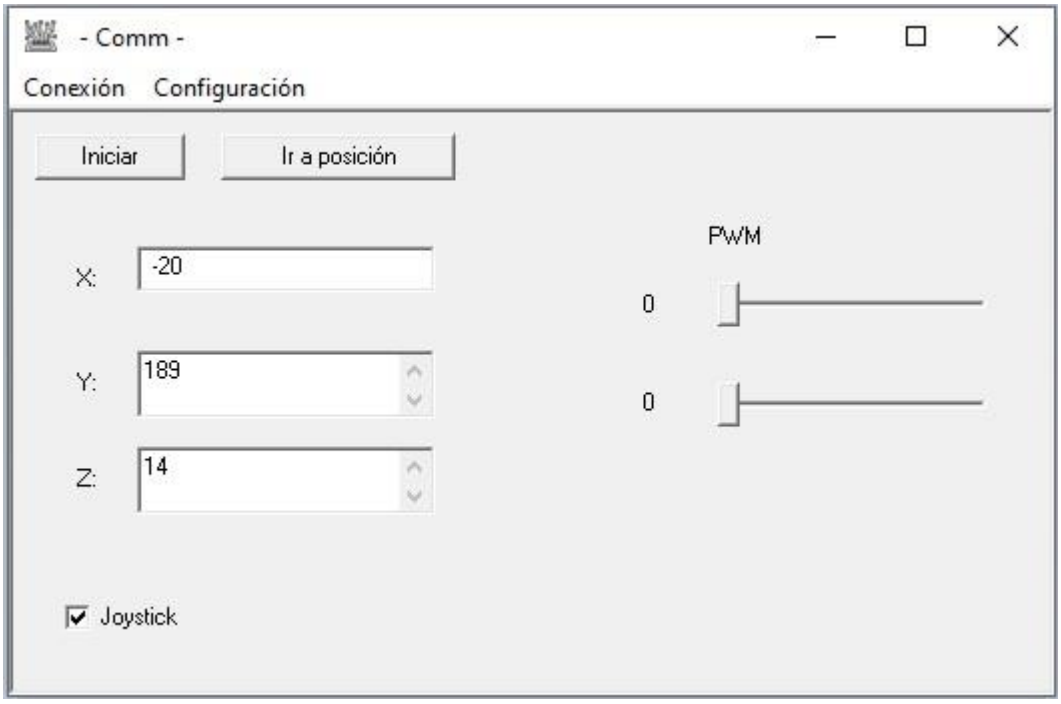

Figura 34. Software de operación.

3.2 Discusión y trabajo a futuro

El sistema electrónico desarrollado y *software* de operación permiten el control de movimiento de tres motores a paso, además de la variación de dos señales PWM que se pueden utilizar para dosificar la potencia de dos láseres de estado sólido. La capacidad de carga inercial a mover debe ser ligera, ya que, en principio, las platinas fueron de diseño exprofeso para componentes ópticos con poco tamaño y peso, como los son: lentes, espejos, cámaras, láseres, etc.

Con el sistema de posicionamiento reportado en el presente informe, es posible disminuir la intervención humana en el posicionamiento de diversos componentes de un arreglo experimental, lo cual disminuye la incertidumbre en la colocación de los dispositivos y asegura la estabilidad mecánica del arreglo por largos periodos de tiempo.

El trabajo a futuro esta dirigido a automatizar el posicionamiento de diversos componentes ópticos utilizando arreglos mecánicos de diferentes tipos, como lo pueden ser los brazos robóticos o arreglos tipo delta.

3.3 Especificaciones del sistema desarrollado

- Alcance del posicionamiento: Eje *x* ±35 mm, eje *y* ±35 mm y eje *z* ±60 mm.
- Resolución: fracciones de milímetro. Aproximadamente 5 fracciones de división mínima.
- Voltaje de operación: +12V_{DC} 10A.

- Sistema operativo: Windows 8 y 10.
- Tipo de conexiones utilizadas: dos puertos USB 2.0.
- Consumo de mínimo de potencia en el estado de reposo. Algunos miliamperes dependiendo de la actividad del puerto USB.

3.4 Guía para la detección de fallas

En primer lugar, para verificar la correcta instalación de los componentes que se muestran en la figura 32, se debe ejecutar el “Administrador de dispositivos” del sistema operativo Windows 10 y observar un despliegue como el mostrado en la figura 35.

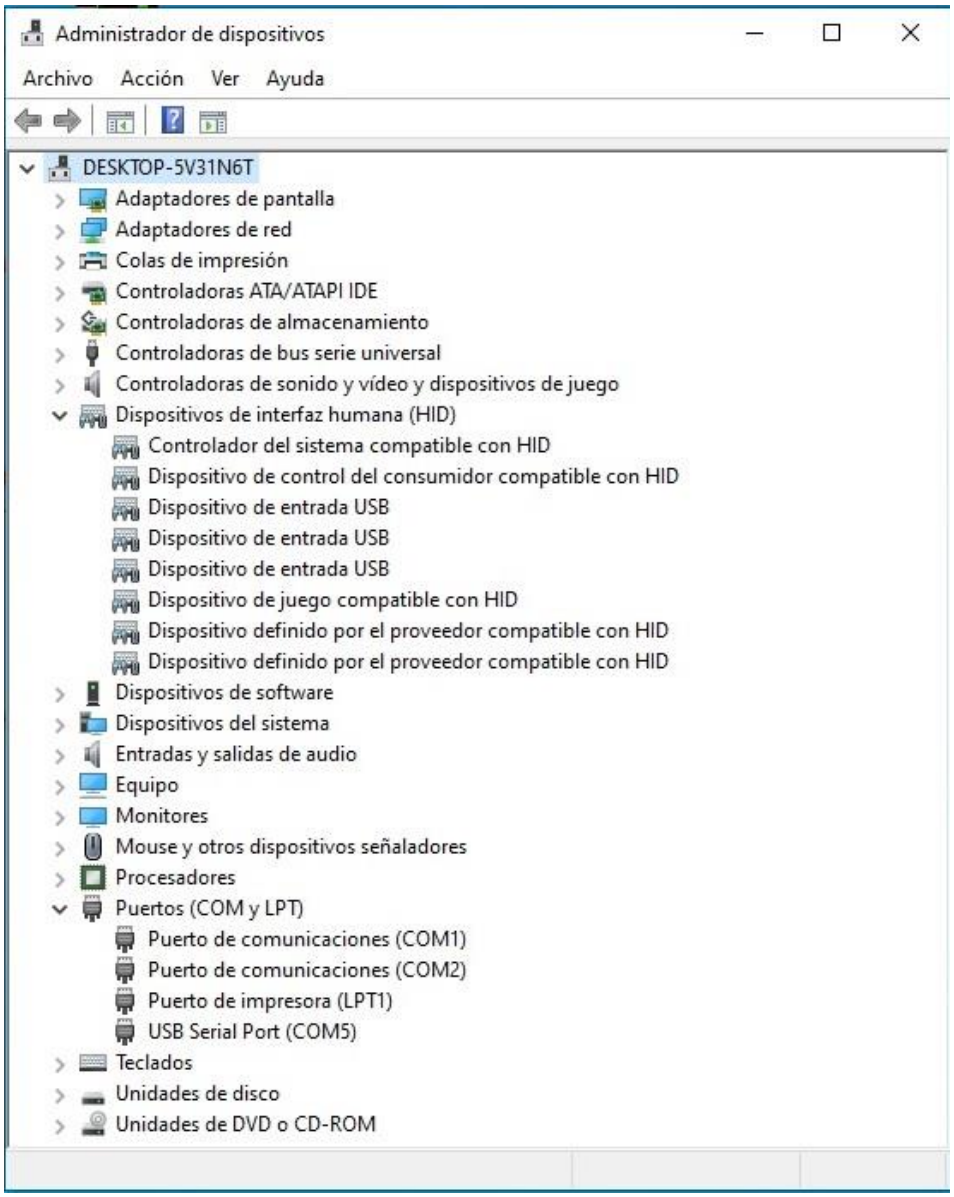

Figura 35. Administrador de dispositivos de Windows 10 mostrando la correcta instalación.

En donde se observa que el *joystick* es detectado como “Dispositivo de juego compatible con HID” y el sistema electrónico es detectado como “USB Serial Port (COM5)”.

Por otra parte, el programa para el posicionamiento milimétrico, “Comm”, puede detectar las dos siguientes anomalías:

1. El *joystick* no ha sido conectado como se muestra en la figura 36.

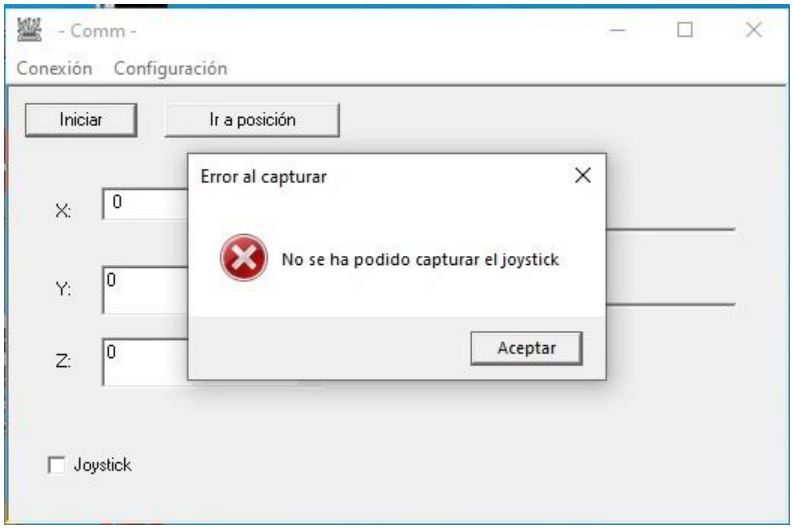

Figura 36. Error en la conexión con el joystick.

2. El sistema electrónico no ha sido conectado como se muestra en la figura 37.

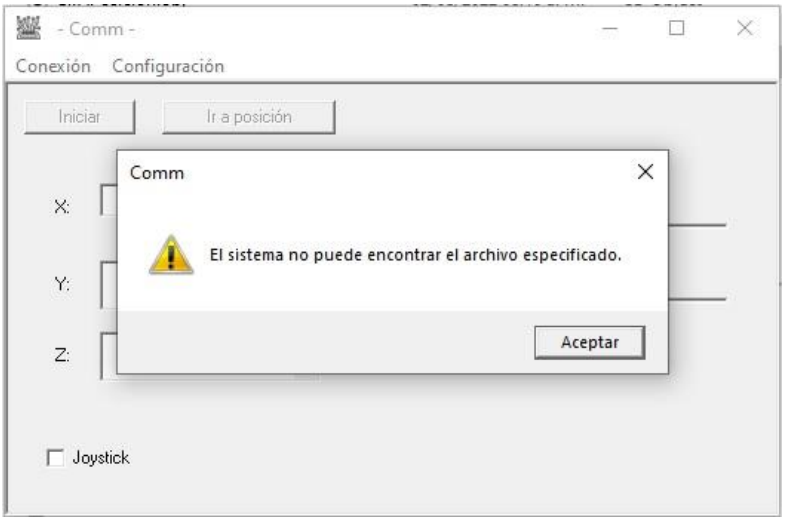

Figura 37. Error en la conexión con el sistema electrónico.

Bajo cualquiera de estas dos situaciones de error, se debe proceder a la correcta instalación de los dispositivos o drivers para que el administrador de dispositivos de Windows muestre la pantalla de la figura 35.

Conclusiones

Se desarrolló y construyó un sistema electrónico *hardware* y *software* para el posicionamiento milimétrico de dispositivos ópticos ligeros sobre una mesa de laboratorio. El sistema elimina la intervención humana en el posicionamiento, con lo cual se incrementa la confiabilidad y robustes del posicionamiento de los dispositivos. El sistema desarrollado permite configurar diversos experimentos científicos en donde se requieren alineaciones precisas que aseguren la calidad y repetibilidad de los resultados. Adicionalmente, el sistema electrónico desarrollado puede variar la potencia de hasta dos láseres de estado sólido mediante sus salidas PWM.

El software de operación desarrollado funciona bajo ambiente Windows para computadoras personales de escritorio o portátiles que cuenten con, por lo menos, dos puertos USB 2.0. El software fue desarrollado en lenguaje de programación C utilizando Visual Studio 2020.

El sistema electrónico tiene como base el microcontrolador Arduino Nano programado en lenguaje C. Los circuitos electrónicos fueron construidos a partir de elementos discretos de fácil adquisición en el mercado nacional.

Tanto el *hardware* como el *software* elaborados, fueron desarrollados a partir de herramientas básicas, con lo cual se puede afirmar que se cuenta con la base tecnológica para prescindir de soluciones comercializadas por terceras partes.

Agradecimientos

A los talleres de construcción de prototipos y serigrafia del ICAT UNAM.

Anexo A Diagrama electrónico esquemático

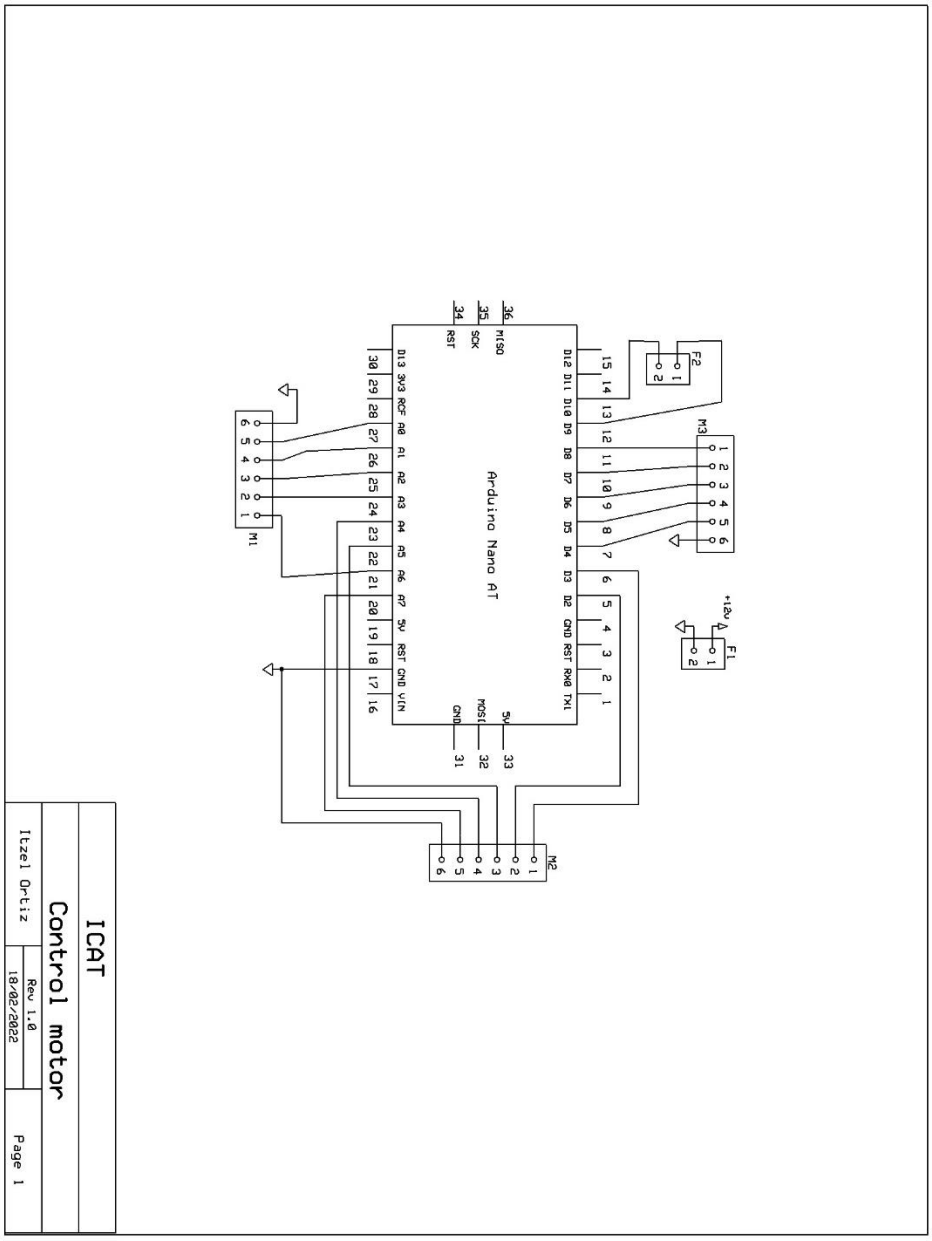

Figura 38. Dibujo esquemático, placa Arduino Nano.

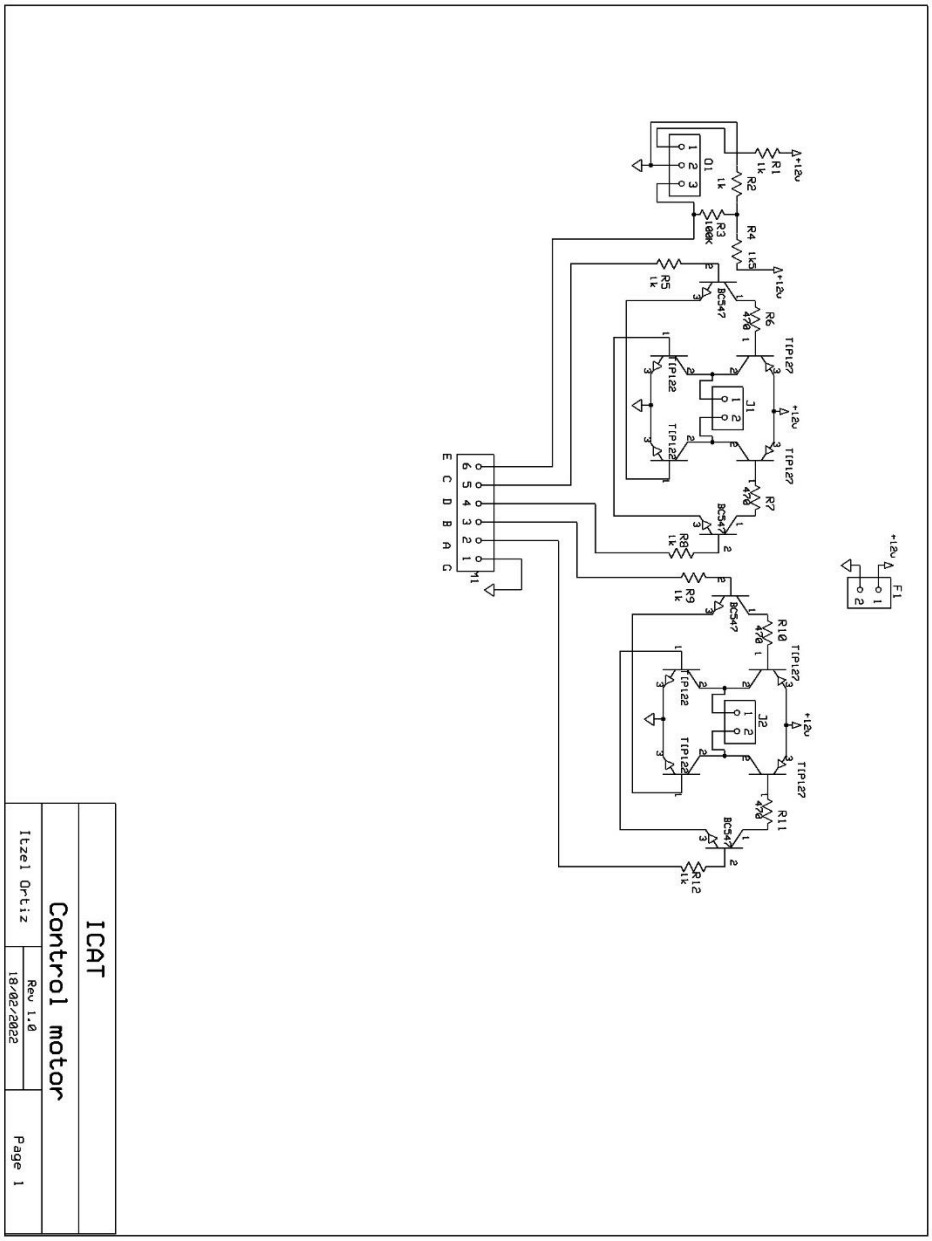

Figura 39. Dibujo esquemático, placa puente H.

Anexo B Circuito electrónico impreso

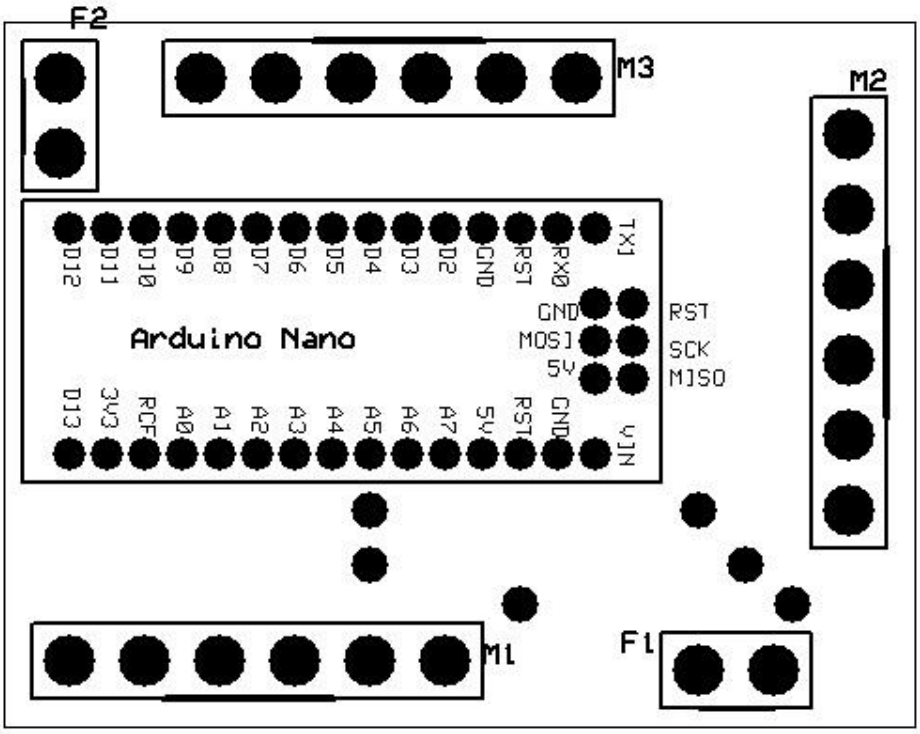

Figura 40. Circuito impreso, placa Arduino Nano mascarilla.

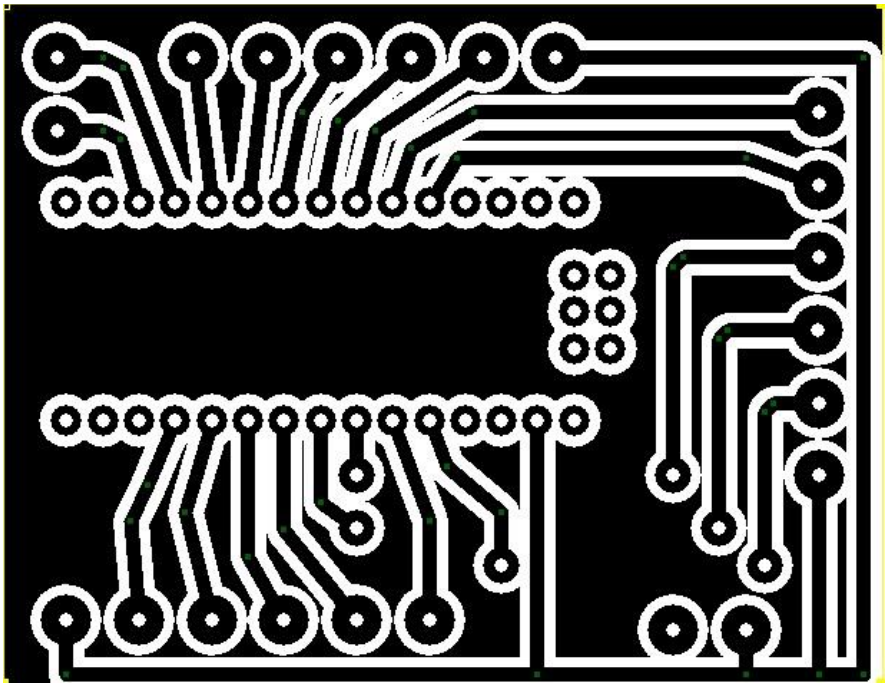

Figura 41. Circuito impreso, Arduino Nano soldadura.

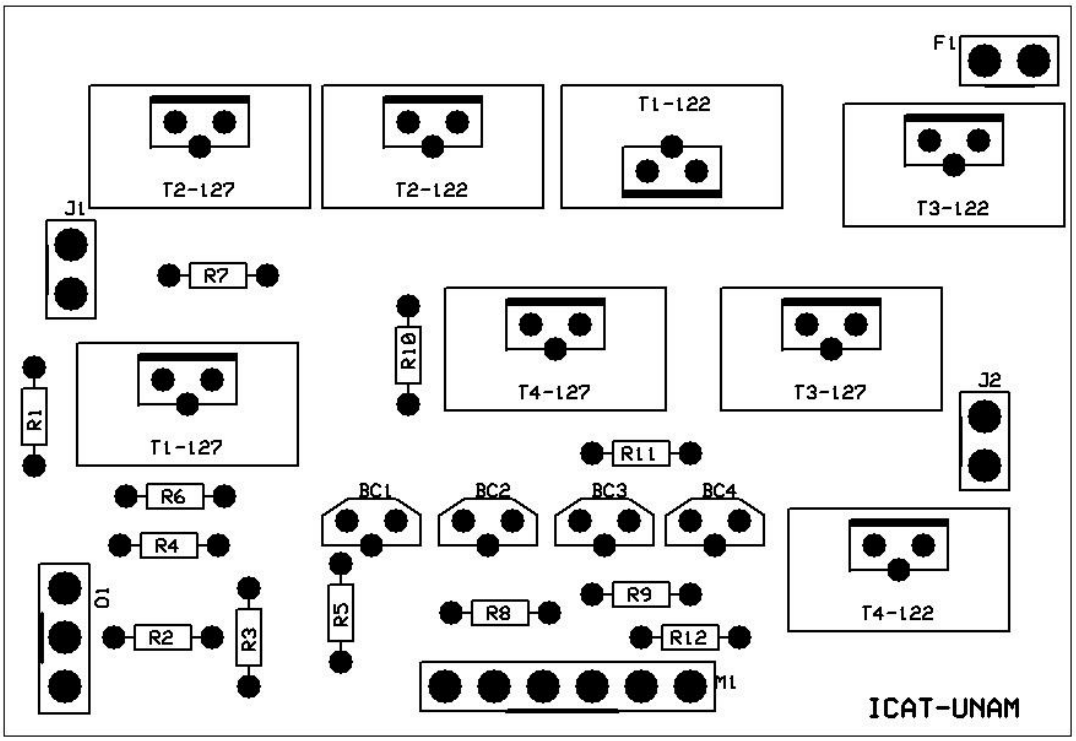

Figura 42. Circuito impreso, puente H mascarilla.

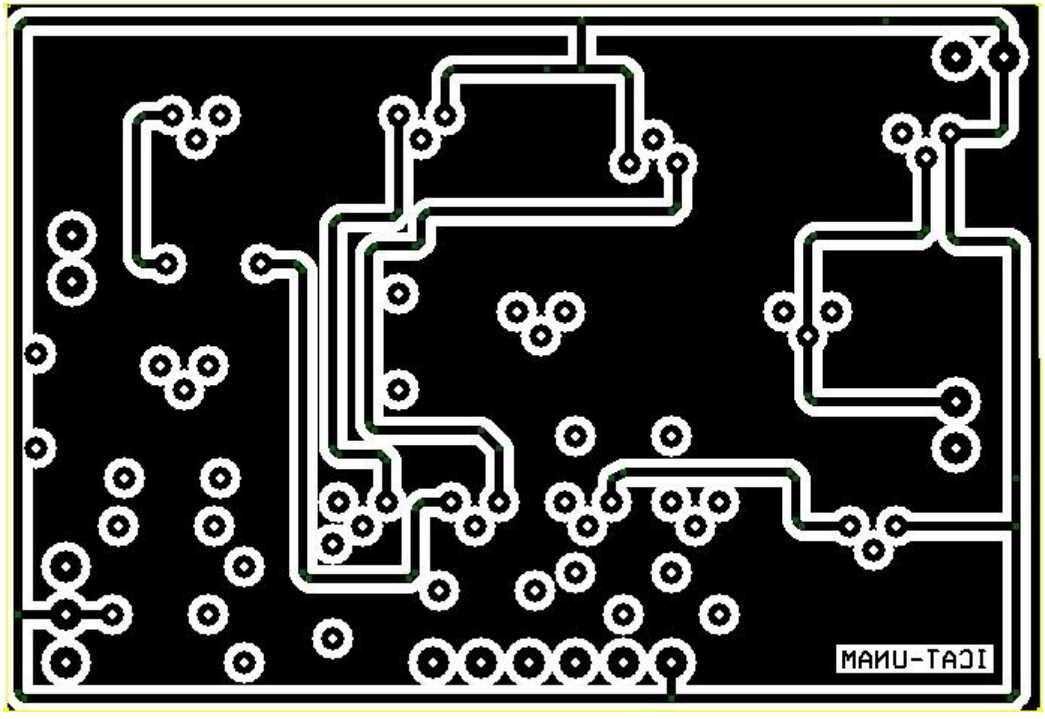

Figura 43. Circuito impreso, puente H soldadura.

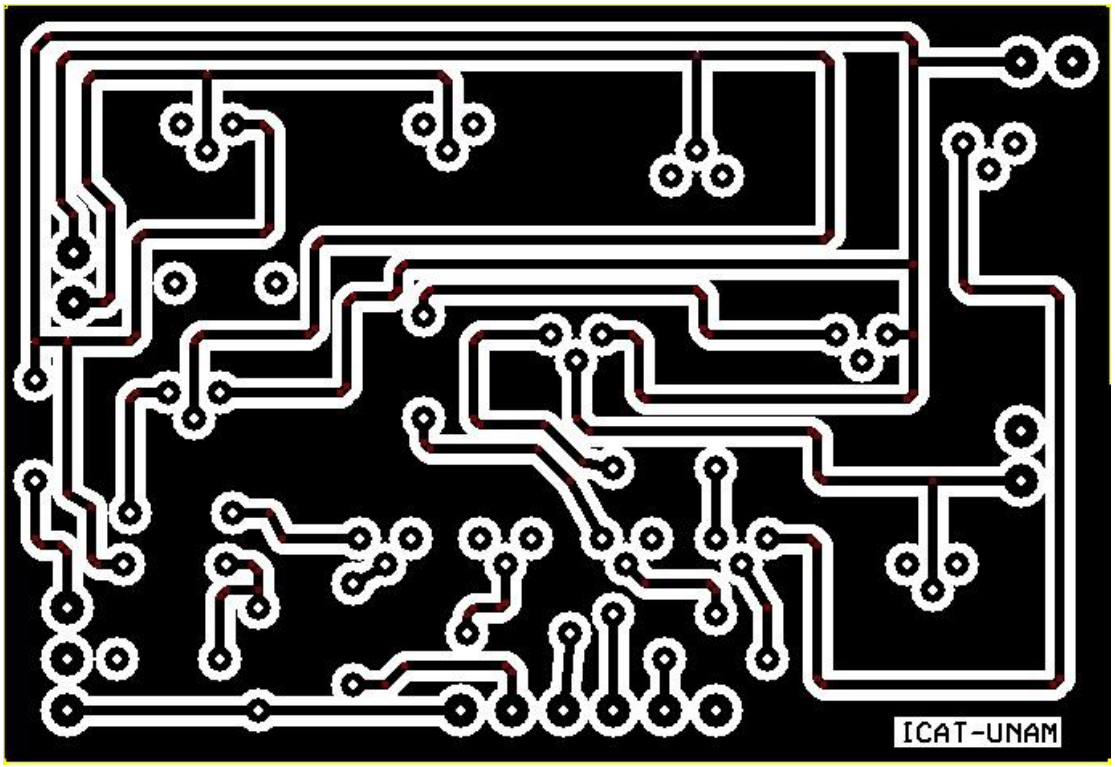

Figura 44. Circuito impreso, puente H componentes.

Anexo C Lista de partes electrónicas

Id de parte	Descripción	Proveedor/Numero de parte
Arduino Nano	Circuito integrado	AG Electrónica/A000005
Arduino Nano	Base para circuito	AG Electrónica/40PF
F1	Conector de tornillo	Steren/TRT-02
F2	Conector de tornillo	Steren/TRT-02
M1	Conector de tornillo	Steren/TRT-02
M2	Conector de tornillo	Steren/TRT-02
M3	Conector de tornillo	Steren/TRT-02

Tabla 2. Lista de partes Arduino Nano.

Id de parte	Descripción	Proveedor/Numero de parte
BC1	Transistor BC547	AG Electrónica/BC547A
BC2	Transistor BC547	AG Electrónica/BC547A
BC3	Transistor BC547	AG Electrónica/BC547A
BC4	Transistor BC547	AG Electrónica/BC547A
F1	Conector de tornillo	Steren/TRT-02
J1	Conector de tornillo	Steren/TRT-02
J2	Conector de tornillo	Steren/TRT-02
M1	Conector de tornillo	Steren/TRT-02
O1	Conector de tornillo	Steren/TRT-03
R1	Resistencia 1k	AG Electrónica/RC-1K/1/2
R2	Resistencia 1k	AG Electrónica/RC-1K/1/2
R3	Resistencia 100K	AG Electrónica/RC-100K/1/2
R4	Resistencia 1k5	AG Electrónica/RC-1K5/1/2
R5	Resistencia 1k	AG Electrónica/RC-1K/1/2
R6	Resistencia 470	AG Electrónica/RC-470E/1/2
R7	Resistencia 470	AG Electrónica/RC-470E/1/2
R8	Resistencia 1k	AG Electrónica/RC-1K/1/2
R9	Resistencia 1k	AG Electrónica/RC-1K/1/2
R10	Resistencia 470	AG Electrónica/RC-470E/1/2
R11	Resistencia 470	AG Electrónica/RC-470E/1/2
R12	Resistencia 1k	AG Electrónica/RC-1K/1/2
T1-122	Transistor TIP122	AG Electronica/T122
T1-122	Disipador de calor	Steren/TO-220 C/C
T1-127	Transistor TIP127	AG Electronica/T127
T1-127	Disipador de calor	Steren/TO-220 C/C
T2-122	Transistor TIP122	AG Electronica/T122
T1-122	Disipador de calor	Steren/TO-220 C/C
T2-127	Transistor TIP127	AG Electronica/T127
T1-127	Disipador de calor	Steren/TO-220 C/C
T3-122	Transistor TIP122	AG Electronica/T122

T1-122	Disipador de calor	Steren/TO-220 C/C
T3-127	Transistor TIP127	AG Electronica/T127
T1-127	Disipador de calor	Steren/TO-220 C/C
T4-122	Transistor TIP122	AG Electronica/T122
T1-122	Disipador de calor	Steren/TO-220 C/C
T4-127	Transistor TIP127	AG Electronica/T127
T1-127	Disipador de calor	Steren/TO-220 C/C

Tabla 3. Lista de partes puente H.

Anexo D Acoplamientos mecánicos

Se utilizaron tres motores Nema 17 17hs4023 de 0.7A que proporcionan un torque de 1.4kg/cm^2 para mover respectivamente cada una de las tres platinas del sistema de posicionamiento. Para acoplar el eje del motor al eje de la platina se utilizó el esquema mostrado en la figura 45.

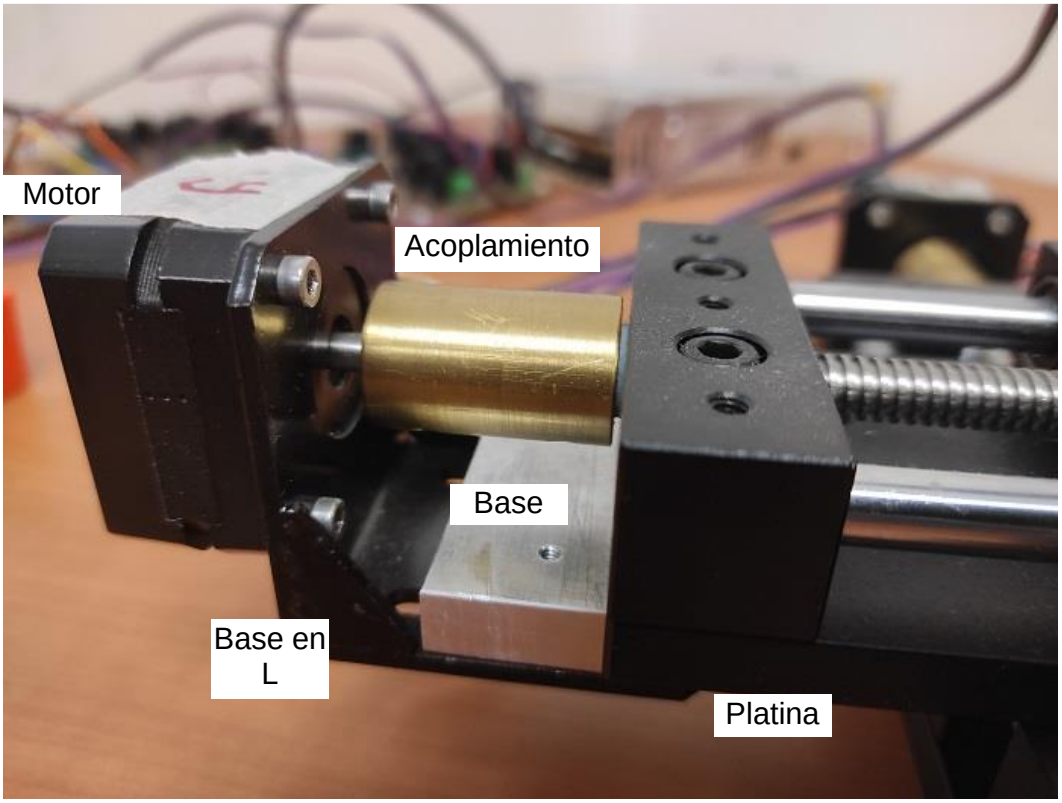

Figura 45. Conexión mecánica entre motores y platinas.

Con respecto a la figura 45, el eje del motor se conecta al eje de la platina mediante el dispositivo “Acoplamiento” que es una pieza maquinada en latón con prisioneros de sujeción a los respectivos ejes. El soporte del motor con respecto a la platina se consigue con el dispositivo comercial denominado “Base en L”, que, por una parte, sostiene al motor con cuatro tornillos, y por otra parte se sujeta al cuerpo de la platina mediante otra pieza denominada “Base” maquinada en aluminio.

La figura 46 muestra la base en L que se comercializa en el mercado de componentes robóticos.

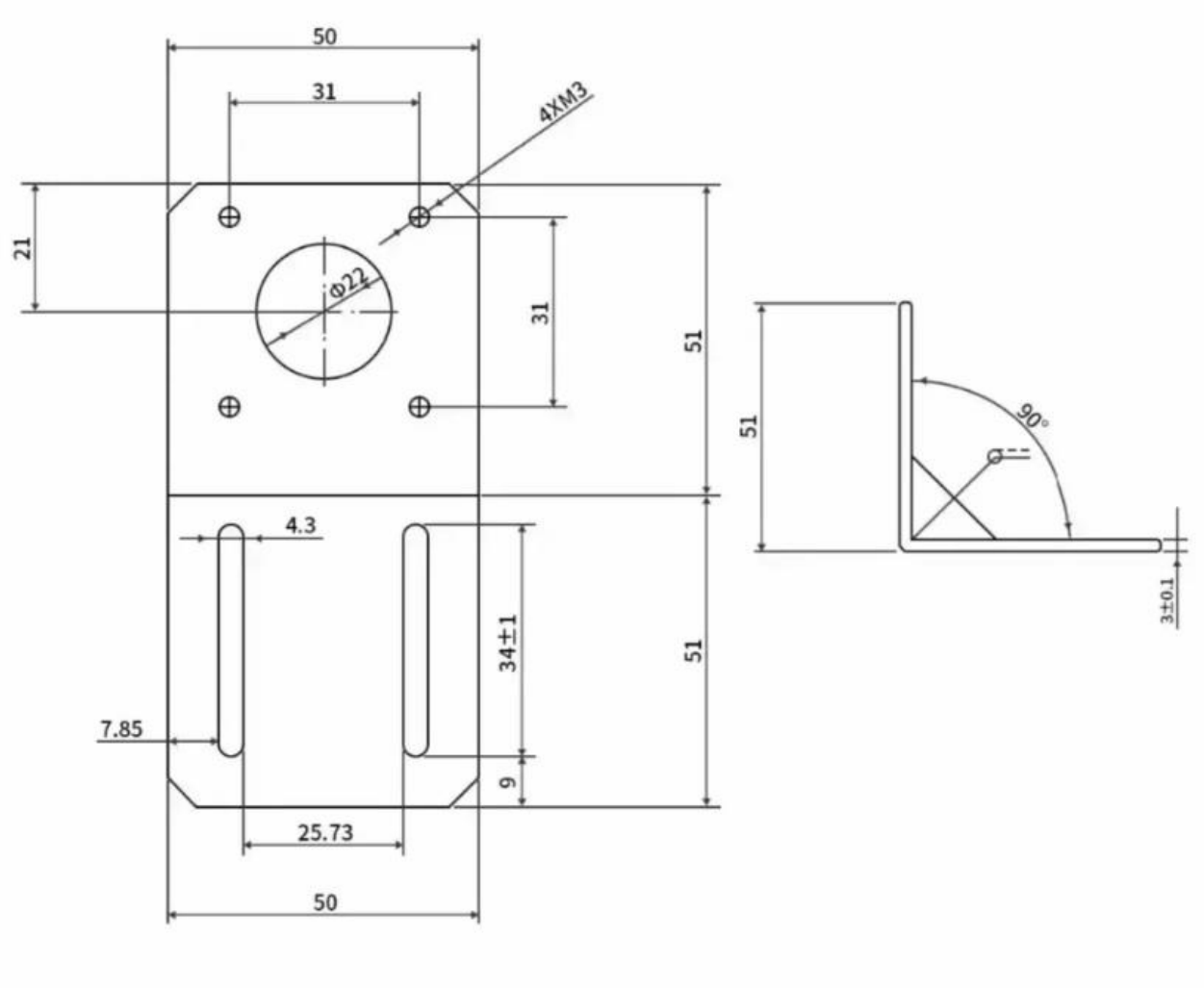

Figura 46. Base en L.

La figura 47 muestra los planos para el maquinado del acoplamiento.

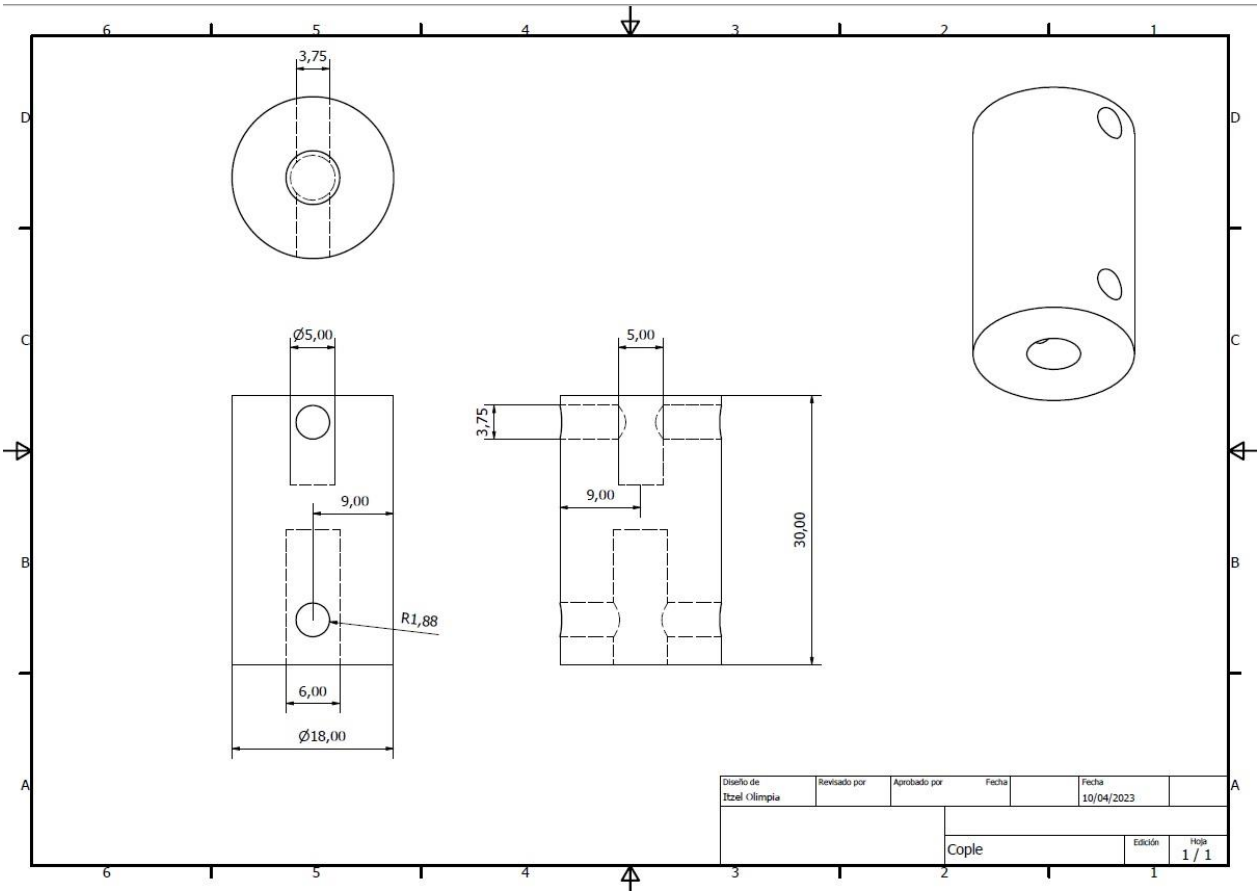

Figura 47. Acoplamiento.

La figura 48 muestra los planos para el maquinado de la base.

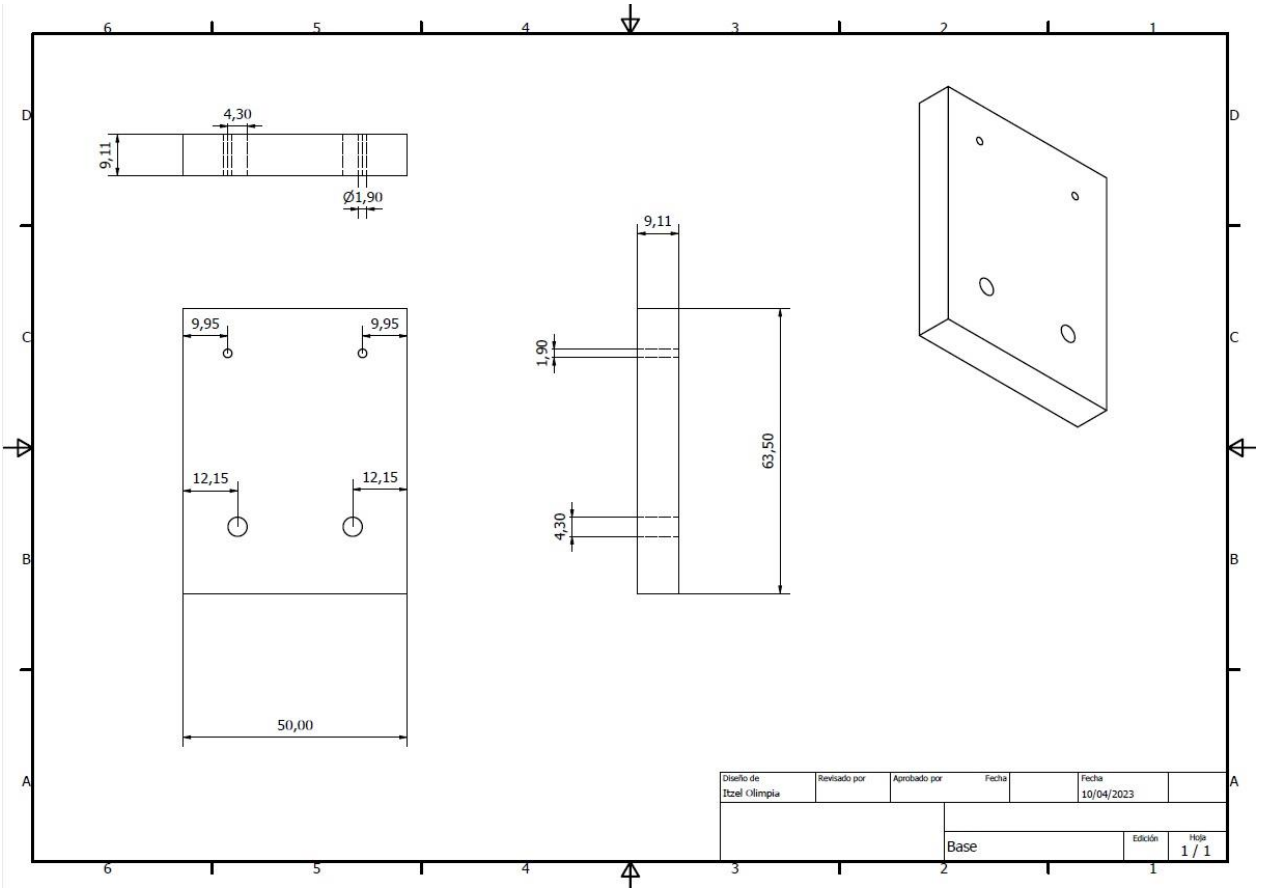

Figura 48. Base.

Anexo E Código fuente Visual C

Comm.h

```
// Comm.h : main header file for the COMM application
//

#if !defined(AFX_COMM_H__645F7287_F003_11D1_8197_CA29344DCF3E__INCLUDED_)
#define AFX_COMM_H__645F7287_F003_11D1_8197_CA29344DCF3E__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

/////////////////////////////////////////////////////////////////////////////
// CCommApp:
// See Comm.cpp for the implementation of this class
//

class CCommApp : public CWinApp
{
public:
	CCommApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CCommApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CCommApp)
	afx_msg void OnAppAbout();
		// NOTE - the ClassWizard will add and remove member functions here.
		//    DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_COMM_H__645F7287_F003_11D1_8197_CA29344DCF3E__INCLUDED_)
```

CommDoc.h

```
// CommDoc.h : interface of the CCommDoc class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_COMMDOC_H__645F728D_F003_11D1_8197_CA29344DCF3E__INCLUDED_)
#define AFX_COMMDOC_H__645F728D_F003_11D1_8197_CA29344DCF3E__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000


class CCommDoc : public CDocument
{
protected: // create from serialization only
	CCommDoc();
	DECLARE_DYNCREATE(CCommDoc)

// Attributes
public:

// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CCommDoc)
	public:
	virtual BOOL OnNewDocument();
	virtual void Serialize(CArchive& ar);
	//}}AFX_VIRTUAL

// Implementation
public:
	virtual ~CCommDoc();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
	//{{AFX_MSG(CCommDoc)
		// NOTE - the ClassWizard will add and remove member functions here.
		//    DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the
previous line.
```

```
#endif // !defined(AFX_COMMDOC_H__645F728D_F003_11D1_8197_CA29344DCF3E__INCLUDED_)
```

ComView.h

```
// CommView.h : interface of the CCommView class
//
/////////////////////////////////////////////////////////////////////////////

#include "CommDoc.h"
#if !defined(AFX_COMMVIEW_H__645F728F_F003_11D1_8197_CA29344DCF3E__INCLUDED_)
#define AFX_COMMVIEW_H__645F728F_F003_11D1_8197_CA29344DCF3E__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define WM_EVENTO_COM WM_USER + 100 // mensaje de notificación
UINT ControlarEventos(LPVOID p); // hilo

class CCommView : public CFormView
{
protected: // create from serialization only
	CCommView();
	DECLARE_DYNCREATE(CCommView)

public:
	//{{AFX_DATA(CCommView)
	enum { IDD = IDD_COMM_FORM };
	//CButton	m_botonEnviar;
	CString	m_rx;
	//CString	m_tx;
	//}}AFX_DATA

// Attributes
public:
	CCommDoc* GetDocument();

// Operations
public:
  ////////////////////////////////////////////////
  // Interfaz para comunicaciones
  static int m_indPuerto;
  static int m_indBaudios;
  static int m_indParidad;
  static int m_indBitsCar;
  static int m_indBitsParada;
  static int m_indControlFlujo;

  HANDLE m_hDisCom; // handle al dispositivo de comunicaciones
	OVERLAPPED m_sOverRead;	// utilizada en una entrada asíncrona
	OVERLAPPED m_sOverWrite;	// utilizada en una salida asíncrona
  UINT m_wTablaBaudios[13];   // tabla de velocidades
  BYTE m_TablaParidad[5];     // tabla de paridades
  BYTE m_TablaBitsParada[3];  // tabla bits de parada
  BOOL m_ConexionEstablecida; // TRUE si el puerto fue abierto
  BOOL m_bHiloActivo;         // TRUE si el hilo está activo
```

```
  static void Iniciar();
  static void Terminar();
  BOOL EstablecerConexion();
  BOOL ConfigurarDisCom();
  BOOL CortarConexion();
  int LeerCaracteresPuerto(BYTE *pBytesLeidos, int BloqueMax);
  BOOL EscribirCarsPuerto(BYTE *pByteAEscribir, DWORD dwBytes);
  void MensajeDeError(DWORD nError);
  ////////////////////////////////////////////////

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CCommView)
	public:
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	virtual void OnInitialUpdate();
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
public:
	void OnPWM();
	void OnVisualizarCars(BYTE *, int);
	virtual ~CCommView();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
	afx_msg long OnEventoCom(UINT wParam, long lParam);
	//{{AFX_MSG(CCommView)
	afx_msg void OnConfigParam();
	afx_msg void OnUpdateConfigParam(CCmdUI* pCmdUI);
	afx_msg void OnConexionEstablecer();
	afx_msg void OnUpdateConexionEstablecer(CCmdUI* pCmdUI);
	afx_msg void OnConexionCortar();
	afx_msg void OnUpdateConexionCortar(CCmdUI* pCmdUI);
	afx_msg void OnEnviar();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
public:
	afx_msg void OnBnClickedIniciar();
	afx_msg void OnBnClickedIrpos();
	afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
	CString m_infoBN;

	CString m_ry;
	CString m_rz;
	BOOL m_BJoy;

	//Banderas control
	BOOL bnx;
	BOOL bnyz;
```

```
	LRESULT OnJoy1Move(WPARAM wParam, LPARAM lParam);
};

#ifndef _DEBUG  // debug version in CommView.cpp
inline CCommDoc* CCommView::GetDocument()
   { return (CCommDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_COMMVIEW_H__645F728F_F003_11D1_8197_CA29344DCF3E__INCLUDED_)
```

MainFrm.h

```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__645F728B_F003_11D1_8197_CA29344DCF3E__INCLUDED_)
#define AFX_MAINFRM_H__645F728B_F003_11D1_8197_CA29344DCF3E__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
	CMainFrame();
	DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CMainFrame)
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
	//}}AFX_VIRTUAL

// Implementation
public:
	virtual ~CMainFrame();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

// Generated message map functions
```

```
protected:
	//{{AFX_MSG(CMainFrame)
		// NOTE - the ClassWizard will add and remove member functions here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_MAINFRM_H__645F728B_F003_11D1_8197_CA29344DCF3E__INCLUDED_)
```

ParamCom.h

```
#if !defined(AFX_PARAMCOM_H__76C07801_F15E_11D1_8197_D9FCD13ABA3E__INCLUDED_)
#define AFX_PARAMCOM_H__76C07801_F15E_11D1_8197_D9FCD13ABA3E__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// ParamCom.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CParamCom dialog

class CParamCom : public CDialog
{
// Construction
public:
	CParamCom(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(CParamCom)
	enum { IDD = IDD_PARAMETROSCOM };
	int		m_nBitsCar;
	int		m_nBaudios;
	int		m_nBitsParada;
	int		m_nControlFlujo;
	int		m_nPuerto;
	int		m_nParidad;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CParamCom)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
```

```
	// Generated message map functions
	//{{AFX_MSG(CParamCom)
	afx_msg void OnPorOmision();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
public:

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_PARAMCOM_H__76C07801_F15E_11D1_8197_D9FCD13ABA3E__INCLUDED_)
```

CIrAPosicion.cpp

```
// CIrAPosicion.cpp: archivo de implementación
//

#include "stdafx.h"
#include "Comm.h"
#include "afxdialogex.h"
#include "resource.h"
#include "CIrAPosicion.h"
#include "CommView.h"


// Cuadro de diálogo de CIrAPosicion

IMPLEMENT_DYNAMIC(CIrAPosicion, CDialogEx)

CIrAPosicion::CIrAPosicion(CWnd* pParent /*= nullptr*/)
	: CDialogEx(IDD_IRAPOSICION, pParent)
	, m_Avancex(_T(""))
	, m_Avancey(_T(""))
	, m_Avancez(_T(""))
	, m_velocidades(_T(""))
{
#ifndef _WIN32_WCE
	EnableActiveAccessibility();
#endif

}

CIrAPosicion::~CIrAPosicion()
{
}

void CIrAPosicion::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
	DDX_Text(pDX, IDC_AVX, m_Avancex);
	DDX_Text(pDX, IDC_AVY, m_Avancey);
	DDX_Text(pDX, IDC_AVZ, m_Avancez);
	DDX_Text(pDX, IDC_VELOCIDADES, m_velocidades);
```

```
}


BEGIN_MESSAGE_MAP(CIrAPosicion, CDialogEx)

END_MESSAGE_MAP()


// Controladores de mensajes de CIrAPosicion
```

Comm.cpp

```
// Comm.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "Comm.h"

#include "MainFrm.h"
#include "CommDoc.h"
#include "CommView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CCommApp

BEGIN_MESSAGE_MAP(CCommApp, CWinApp)
	//{{AFX_MSG_MAP(CCommApp)
	ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG_MAP
	// Standard file based document commands
	ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
	ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCommApp construction

CCommApp::CCommApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CCommApp object

CCommApp theApp;
```

```
/////////////////////////////////////////////////////////////////////////////
// CCommApp initialization

BOOL CCommApp::InitInstance()
{
	AfxOleInit();
	AfxEnableControlContainer();

	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
//	Enable3dControls();			// Call this when using MFC in a shared DLL
#else
	Enable3dControlsStatic();	// Call this when linking to MFC statically
#endif

	// Change the registry key under which our settings are stored.
	// You should modify this string to be something appropriate
	// such as the name of your company or organization.
	SetRegistryKey(_T("App Comm"));

	LoadStdProfileSettings();  // Load standard INI file options (including MRU)

	// Register the application's document templates.  Document templates
	//  serve as the connection between documents, frame windows and views.

	CSingleDocTemplate* pDocTemplate;
	pDocTemplate = new CSingleDocTemplate(
		IDR_MAINFRAME,
		RUNTIME_CLASS(CCommDoc),
		RUNTIME_CLASS(CMainFrame),       // main SDI frame window
		RUNTIME_CLASS(CCommView));
	AddDocTemplate(pDocTemplate);

	// Parse command line for standard shell commands, DDE, file open
	CCommandLineInfo cmdInfo;
	ParseCommandLine(cmdInfo);

	// Dispatch commands specified on the command line
	if (!ProcessShellCommand(cmdInfo))
		return FALSE;

	// The one and only window has been initialized, so show and update it.
	m_pMainWnd->ShowWindow(SW_SHOW);
	m_pMainWnd->UpdateWindow();

	return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
```

```
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
		// No message handlers
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CCommApp::OnAppAbout()
{
	CAboutDlg aboutDlg;
	aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CCommApp commands
```

CommDoc.cpp

```
// CommDoc.cpp : implementation of the CCommDoc class
//

#include "stdafx.h"
```

```
#include "Comm.h"

#include "CommDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CCommDoc

IMPLEMENT_DYNCREATE(CCommDoc, CDocument)

BEGIN_MESSAGE_MAP(CCommDoc, CDocument)
	//{{AFX_MSG_MAP(CCommDoc)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCommDoc construction/destruction

CCommDoc::CCommDoc()
{
	// TODO: add one-time construction code here

}

CCommDoc::~CCommDoc()
{
}

BOOL CCommDoc::OnNewDocument()
{
	if (!CDocument::OnNewDocument())
		return FALSE;

	// TODO: add reinitialization code here
	// (SDI documents will reuse this document)

	return TRUE;
}



/////////////////////////////////////////////////////////////////////////////
// CCommDoc serialization

void CCommDoc::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
	{
		// TODO: add storing code here
	}
	else
```

```
	{
		// TODO: add loading code here
	}
}

/////////////////////////////////////////////////////////////////////////////
// CCommDoc diagnostics

#ifdef _DEBUG
void CCommDoc::AssertValid() const
{
	CDocument::AssertValid();
}

void CCommDoc::Dump(CDumpContext& dc) const
{
	CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CCommDoc commands
```

ComView.cpp

```
// CommView.cpp : implementation of the CCommView class
//

#include "stdafx.h"
#include "Comm.h"

#include "CommDoc.h"
#include "CommView.h"

#include "ParamCom.h"

#include "CIrAPosicion.h"
#include <iostream>
#include "mmsystem.h" //Incluir biblioteca
#include <cstring>
#include <vector>

using namespace std;

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define COLARX 4096
#define COLATX 4096
#define BLOQUEMAX 80

/////////////////////////////////////////////////////////////////////////////
// CCommView
```

```
IMPLEMENT_DYNCREATE(CCommView, CFormView)

BEGIN_MESSAGE_MAP(CCommView, CFormView)
  //{{AFX_MSG_MAP(CCommView)
  ON_COMMAND(ID_CONFIG_PARAM, OnConfigParam)
  ON_UPDATE_COMMAND_UI(ID_CONFIG_PARAM, OnUpdateConfigParam)
  ON_COMMAND(ID_CONEXION_ESTABLECER, OnConexionEstablecer)
  ON_UPDATE_COMMAND_UI(ID_CONEXION_ESTABLECER, OnUpdateConexionEstablecer)
  ON_COMMAND(ID_CONEXION_CORTAR, OnConexionCortar)
  ON_UPDATE_COMMAND_UI(ID_CONEXION_CORTAR, OnUpdateConexionCortar)
  //ON_BN_CLICKED(IDC_ENVIAR, OnEnviar)
  ON_MESSAGE(MM_JOY1MOVE, OnJoy1Move)
      //}}AFX_MSG_MAP
  ON_MESSAGE(WM_EVENTO_COM, OnEventoCom)
    ON_BN_CLICKED(IDC_INICIAR, &CCommView::OnBnClickedIniciar)
    ON_BN_CLICKED(IDC_IRPOS, &CCommView::OnBnClickedIrpos)
    ON_WM_HSCROLL()
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CCommView construction/destruction

CCommView::CCommView()
  : CFormView(CCommView::IDD)
    //, m_infoBN(_T(""))
    , m_ry(_T(""))
    , m_rz(_T(""))
    , m_BJoy(FALSE)
{
  m_wTablaBaudios[0]   = CBR_110;
  m_wTablaBaudios[1]   = CBR_300;
  m_wTablaBaudios[2]   = CBR_600;
  m_wTablaBaudios[3]   = CBR_1200;
  m_wTablaBaudios[4]   = CBR_2400;
  m_wTablaBaudios[5]   = CBR_4800;
  m_wTablaBaudios[6]   = CBR_9600;
  m_wTablaBaudios[7]   = CBR_14400;
  m_wTablaBaudios[8]   = CBR_19200;
  m_wTablaBaudios[9]   = CBR_38400;
  m_wTablaBaudios[10]  = CBR_56000;
  m_wTablaBaudios[11]  = CBR_128000;
  m_wTablaBaudios[12]  = CBR_256000;

  m_TablaParidad[0]    = NOPARITY;
  m_TablaParidad[1]    = EVENPARITY;
  m_TablaParidad[2]    = ODDPARITY;
  m_TablaParidad[3]    = MARKPARITY;
  m_TablaParidad[4]    = SPACEPARITY;

  m_TablaBitsParada[0] = ONESTOPBIT;
  m_TablaBitsParada[1] = ONE5STOPBITS;
  m_TablaBitsParada[2] = TWOSTOPBITS;

  //{{AFX_DATA_INIT(CCommView)
  m_rx = _T("");
  //m_tx = _T("");
  //}}AFX_DATA_INIT
  // TODO: add construction code here
```

```
  m_hDisCom = NULL;
  m_bHiloActivo = FALSE;
  m_ConexionEstablecida = FALSE;
}

CCommView::~CCommView()
{
  if ( m_ConexionEstablecida )
    CortarConexion();
}

void CCommView::DoDataExchange(CDataExchange* pDX)
{
    CFormView::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CCommView)
    //DDX_Control(pDX, IDC_ENVIAR, m_botonEnviar);
    DDX_Text(pDX, IDC_RX, m_rx);
    //DDX_Text(pDX, IDC_TX, m_tx);
    //}}AFX_DATA_MAP
    //DDX_Text(pDX, IDC_INFO, m_infoBN);
    DDX_Text(pDX, IDC_RY, m_ry);
    DDX_Text(pDX, IDC_RZ, m_rz);
    DDX_Check(pDX, IDC_CHECKJOY, m_BJoy);
}

BOOL CCommView::PreCreateWindow(CREATESTRUCT& cs)
{
  // TODO: Modify the Window class or styles here by modifying
  //  the CREATESTRUCT cs

  return CFormView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CCommView diagnostics

#ifdef _DEBUG
void CCommView::AssertValid() const
{
  CFormView::AssertValid();
}

void CCommView::Dump(CDumpContext& dc) const
{
  CFormView::Dump(dc);
}

CCommDoc* CCommView::GetDocument() // non-debug version is inline
{
  ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CCommDoc)));
  return (CCommDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// Interfaz para comunicaciones

int CCommView::m_indPuerto       = 1; // COM2
```

```
int CCommView::m_indBaudios      = 6; // 9600
int CCommView::m_indParidad      = 0; // ninguna
int CCommView::m_indBitsCar      = 4; // 8
int CCommView::m_indBitsParada   = 0; // 1
int CCommView::m_indControlFlujo = 1; // Xon/Xoff

static char szComu[]         = "Comunicaciones";
static char szPuerto[]       = "Puerto";
static char szBaudios[]      = "Baudios";
static char szParidad[]      = "Paridad";
static char szBitsCar[]      = "BitsCar";
static char szBitsParada[]   = "BitsParada";
static char szControlFlujo[] = "ControlFlujo";

void CCommView::Iniciar()
{
  CWinApp *pApp= AfxGetApp();
  // Recuperar configuración del registro de Windows
  m_indPuerto     = pApp->GetProfileInt(szComu, szPuerto, 1);
  m_indBaudios    = pApp->GetProfileInt(szComu, szBaudios, 6);
  m_indParidad    = pApp->GetProfileInt(szComu, szParidad, 0);
  m_indBitsCar      = pApp->GetProfileInt(szComu, szBitsCar, 4);
  m_indBitsParada   = pApp->GetProfileInt(szComu, szBitsParada, 0);
  m_indControlFlujo = pApp->GetProfileInt(szComu, szControlFlujo, 1);
}

void CCommView::Terminar()
{
  CWinApp *pApp= AfxGetApp();
  // Guardar configuración en el registro de Windows
  pApp->WriteProfileInt(szComu, szPuerto, m_indPuerto);
  pApp->WriteProfileInt(szComu, szBaudios, m_indBaudios);
  pApp->WriteProfileInt(szComu, szParidad, m_indParidad);
  pApp->WriteProfileInt(szComu, szBitsCar, m_indBitsCar);
  pApp->WriteProfileInt(szComu, szBitsParada, m_indBitsParada);
  pApp->WriteProfileInt(szComu, szControlFlujo, m_indControlFlujo);
}

BOOL CCommView::EstablecerConexion()
{
  char szPuerto[10];
  BOOL bExito = FALSE;

  // Formar la cadena "COM" más el número de dispositivo
  //Aqui le cambie
  wsprintf(szPuerto, "COM5");

  // Cerrar el puerto si estuviera abierto
  if (m_hDisCom) CloseHandle(m_hDisCom);

  // Abrir el puerto de comunicaciones
  m_hDisCom = CreateFile(szPuerto,
                         GENERIC_READ | GENERIC_WRITE,
                         0,    // acceso exclusivo
                         NULL, // sin atributos de seguridad
                         OPEN_EXISTING,
                         FILE_FLAG_OVERLAPPED,
                         NULL);
```

```
  if( m_hDisCom == INVALID_HANDLE_VALUE )
  {
    // Visualizar el error ocurrido.
    MensajeDeError( GetLastError() );
    MessageBeep(0xFFFFFFFF);
    return FALSE;
  }

  // Especificar los eventos que serán atendidos
  SetCommMask( m_hDisCom, EV_RXCHAR | EV_TXEMPTY | EV_RX80FULL | EV_ERR);

  // Establecer el tamaño de las colas de recepción y de transmisión
  SetupComm(m_hDisCom, COLARX, COLATX);

  // Terminar las operaciones de lectura y escritura pendientes
  // y limpiar las colas Rx y Tx
  PurgeComm( m_hDisCom, PURGE_TXABORT | PURGE_RXABORT |
                        PURGE_TXCLEAR | PURGE_RXCLEAR );

  // Establecer los parámetros de la comunicación
  bExito = ConfigurarDisCom();

  if ( bExito )
  {
    m_ConexionEstablecida = TRUE;
    // Crear un hilo secundario para ver qué evento ocurre
    if ( AfxBeginThread(ControlarEventos, this) == NULL )
    {
      AfxMessageBox( "Error: No se puede iniciar el hilo",
                     MB_OK | MB_ICONEXCLAMATION );
      m_ConexionEstablecida = FALSE;
      CloseHandle(m_hDisCom);
      return FALSE;
    }
    m_bHiloActivo = TRUE;
    // Enviar la señal DTR (data-terminal-ready).
    EscapeCommFunction(m_hDisCom, SETDTR);
  }
  else
  {
    AfxMessageBox( "Error: No se puede configurar el dispositivo",
                   MB_OK | MB_ICONEXCLAMATION );
    m_ConexionEstablecida = FALSE;
    CloseHandle(m_hDisCom);
  }
  return bExito;
}

BOOL CCommView::ConfigurarDisCom()
{
  BYTE bEstablecer;
  DCB  dcb;

  dcb.DCBlength   = sizeof(DCB);

  GetCommState(m_hDisCom, &dcb);
```

```
  dcb.BaudRate    = m_wTablaBaudios[m_indBaudios];
  dcb.Parity      = m_TablaParidad[m_indParidad];
  dcb.ByteSize    = (BYTE)(m_indBitsCar + 4);
  dcb.StopBits    = m_TablaBitsParada[m_indBitsParada];

  // Establecer el control de flujo software
  bEstablecer     = (BYTE)(m_indControlFlujo == 1); // Xon/Xoff
  dcb.fInX        = dcb.fOutX = bEstablecer;
  dcb.XonChar     = 0x11; // ASCII_XON
  dcb.XoffChar    = 0x13; // ASCII_XOFF
  dcb.XonLim      = 100;
  dcb.XoffLim     = 100;

  // Establecer el control de flujo hardware
  bEstablecer     = (BYTE)(m_indControlFlujo == 2); // DTR/DSR
  dcb.fOutxDsrFlow = bEstablecer;
  if (bEstablecer)
    dcb.fDtrControl = DTR_CONTROL_HANDSHAKE;
  else
    dcb.fDtrControl = DTR_CONTROL_ENABLE;

  bEstablecer     = (BYTE)(m_indControlFlujo == 3); // RTS/CTS
  dcb.fOutxCtsFlow = bEstablecer;
  if (bEstablecer)
    dcb.fRtsControl = RTS_CONTROL_HANDSHAKE;
  else
    dcb.fRtsControl = RTS_CONTROL_ENABLE;

  // Otras especificaciones
  dcb.fBinary = TRUE;
  dcb.fParity = TRUE;

  if(SetCommState(m_hDisCom, &dcb) == 0)
  {
    // Visualizar el error ocurrido.
    MensajeDeError( GetLastError() );
    MessageBeep(0xFFFFFFFF);
    return FALSE;
  }

  // Establecer los tiempos límites para las operaciones de E/S
  COMMTIMEOUTS CommTimeOuts;

  CommTimeOuts.ReadIntervalTimeout = MAXDWORD;
  CommTimeOuts.ReadTotalTimeoutMultiplier = 0;
  CommTimeOuts.ReadTotalTimeoutConstant = 1000;
  // CBR_9600 es aproximadamente 1 byte/ms. Para nuestros
  // propósitos permitiremos un tiempo de espera por carácter
  // doble al necesario
  CommTimeOuts.WriteTotalTimeoutMultiplier = 2*CBR_9600/dcb.BaudRate;
  CommTimeOuts.WriteTotalTimeoutConstant = 0;

  SetCommTimeouts( m_hDisCom, &CommTimeOuts);

  return TRUE;
}

UINT ControlarEventos(LPVOID p)
```

```
{
  DWORD      dwMascEvt;
  CCommView  *const pView = (CCommView *)p;
  OVERLAPPED sOver = {0, 0, 0, 0, 0};

  // Crear un evento de E/S utilizado para lecturas solapadas
  sOver.hEvent = CreateEvent( NULL,    // sin seguridad
                              TRUE,    // iniciación manual
                              FALSE,   // inicialmente ocupado
                              NULL );  // sin nombre
  if (sOver.hEvent == NULL)
  {
    AfxMessageBox( "Fallo al crear el evento para el hilo",
                   MB_OK | MB_ICONEXCLAMATION );
    return 0;
  }

  // Restablecer los eventos por si hubieran cambiado
  if (!SetCommMask( pView->m_hDisCom, EV_RXCHAR | EV_TXEMPTY |
                                      EV_RX80FULL | EV_ERR ))
    return 0;

  while ( pView->m_ConexionEstablecida )
  {
    dwMascEvt = 0;
    WaitCommEvent( pView->m_hDisCom, &dwMascEvt, &sOver );
    if ((dwMascEvt & EV_RXCHAR) == EV_RXCHAR)
      ::PostMessage(pView->m_hWnd, WM_EVENTO_COM, EV_RXCHAR, 0);
    else if ((dwMascEvt & EV_TXEMPTY) == EV_TXEMPTY)
      ::PostMessage(pView->m_hWnd, WM_EVENTO_COM, EV_TXEMPTY, 0);
    else if ((dwMascEvt & EV_RX80FULL) == EV_RX80FULL)
      ::PostMessage(pView->m_hWnd, WM_EVENTO_COM, EV_RX80FULL, 0);
    else if ((dwMascEvt & EV_ERR) == EV_ERR)
      ::PostMessage(pView->m_hWnd, WM_EVENTO_COM, EV_ERR, 0);
  }
  pView->m_bHiloActivo = FALSE;

  // Liberar el manipulador del evento
  CloseHandle( sOver.hEvent );

  return 1;
}

void CCommView::MensajeDeError( DWORD nError )
{
  LPVOID lpMsg;

  ::FormatMessage(
      FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM,
      NULL,
      nError,
      MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
      (LPTSTR) &lpMsg,
      0,
      NULL
    );

  // Mostrar el error
```

```
  AfxMessageBox( (LPCTSTR)lpMsg, MB_OK | MB_ICONEXCLAMATION );

  // Liberar el buffer
  ::LocalFree( lpMsg );
}

long CCommView::OnEventoCom(UINT wParam, long lParam)
{
  BYTE BytesLeidos[BLOQUEMAX + 1];
  int  nBytes;

  // Mensajes recibidos desde el hilo
  switch (wParam)
  {
    case EV_RXCHAR:
//        AfxMessageBox(_T("Hola"), MB_OK | MB_ICONEXCLAMATION );
        //OnPWM();

             if (nBytes = LeerCaracteresPuerto( BytesLeidos, BLOQUEMAX ))
        OnVisualizarCars( BytesLeidos, nBytes );

      break;

    case EV_TXEMPTY:
      // ...
      break;

    case EV_RX80FULL:
      // ...
      break;

    case EV_ERR:
      // ...
      break;
  }
  return 0;
}

int CCommView::LeerCaracteresPuerto(BYTE *pBytesLeidos, int BloqueMax)
{
  BOOL       bLeer;
  COMSTAT    EstadoCom;
  DWORD      dwCodsError;
  DWORD      dwNumBytes;
  DWORD      dwError;
  char       szError[10];
  OVERLAPPED osReader = {0};

  // Leer cada vez como máximo BloqueMax caracteres
  /*  ClearCommError( m_hDisCom, &dwCodsError, &EstadoCom );*/
    dwNumBytes = min( (DWORD)BloqueMax, EstadoCom.cbInQue );
/*
       CString cadena;
       cadena.Format(_T("dwNumBytes=%d"),dwNumBytes);
    AfxMessageBox(cadena, MB_OK | MB_ICONEXCLAMATION );
*/
       if (dwNumBytes > 0)
    {
```

```
    bLeer = ReadFile( m_hDisCom, pBytesLeidos, dwNumBytes,
                      &dwNumBytes, &osReader );
    pBytesLeidos[dwNumBytes] = 0; // finalizar con el carácter nulo
/*
       CString cadena;
       cadena.Format(_T("dwNumBytes=%d, bLeer=%d"), dwNumBytes, bLeer);
    AfxMessageBox(cadena, MB_OK | MB_ICONEXCLAMATION );
*/
    if (!bLeer)
    {
         DWORD error;
         error=GetLastError();
/*
      CString cadena;
         cadena.Format(_T("error=%d"), error);
      AfxMessageBox(cadena, MB_OK | MB_ICONEXCLAMATION );
*/
      if (error == ERROR_IO_PENDING)
      {
        // Tenemos que esperar a que la lectura se complete. Esta
        // función será interrumpida cuando transcurra un tiempo
        // CommTimeOuts.ReadTotalTimeoutConstant.

        // Chequear errores en el puerto
        while(!GetOverlappedResult( m_hDisCom,
              &osReader, &dwNumBytes, FALSE ))
        {
          dwError = GetLastError();
          // Si no terminó, dwError vale ERROR_IO_INCOMPLETE
          if(dwError == ERROR_IO_INCOMPLETE)
            continue;
          else
          {
            // Ocurrió un error, intentar recuperarlo
            MensajeDeError( dwError );
            ClearCommError( m_hDisCom, &dwCodsError, &EstadoCom );
            if ( dwCodsError > 0 )
            {
              wsprintf( szError, "Com: <CE-%u>", dwCodsError );
              // Los errores IE son < 0 (winbase.h)
              AfxMessageBox( szError, MB_OK | MB_ICONEXCLAMATION );
            }
            break;
          }
        } // fin while
      }
      else // error distinto de ERROR_IO_PENDING
      {
        dwNumBytes = 0;
        ClearCommError( m_hDisCom, &dwCodsError, &EstadoCom );
/*
        CString cadena;
              cadena.Format(_T("error=%d"), dwCodsError);
        AfxMessageBox(cadena, MB_OK | MB_ICONEXCLAMATION );
*/
        if ( dwCodsError > 0 )
        {
          wsprintf( szError, "Com: <CE-%u>", dwCodsError );
```

```
          AfxMessageBox( szError, MB_OK | MB_ICONEXCLAMATION );
        }
      }
    }
  }
  return (dwNumBytes);
}

BOOL CCommView::EscribirCarsPuerto(BYTE *pBytesAEscribir, DWORD dwBytes)
{
  COMSTAT    EstadoCom;
  BOOL       bEscribir;
  DWORD      dwNumBytes;
  DWORD      dwCodsError;
  DWORD      dwError;
  DWORD      dwBytesEnviados=0;
  char       szError[128];

  OVERLAPPED osWrite = { 0 };

  bEscribir = WriteFile( m_hDisCom, pBytesAEscribir, dwBytes,
                         &dwNumBytes, &osWrite);

  if (!bEscribir)
  {
    if( GetLastError() == ERROR_IO_PENDING )
    {
      while(!GetOverlappedResult( m_hDisCom,
            &osWrite, &dwNumBytes, FALSE ))
      {
        dwError = GetLastError();
        // Si no terminó, dwError vale ERROR_IO_INCOMPLETE
        if(dwError == ERROR_IO_INCOMPLETE)
        {
          dwBytesEnviados += dwNumBytes;
          continue;
        }
        else
        {
          // ocurrió un error, intentar recuperarlo
          MensajeDeError( dwError );
          ClearCommError( m_hDisCom, &dwCodsError, &EstadoCom );
          if ( dwCodsError > 0 )
          {
            wsprintf( szError, "Com: <CE-%u>", dwCodsError );
            // Los errores IE son < 0 (winbase.h)
            AfxMessageBox( szError, MB_OK | MB_ICONEXCLAMATION );
          }
          break;
        }
      }
      dwBytesEnviados += dwNumBytes;
      if( dwBytesEnviados != dwBytes )
        wsprintf(szError,"\nProbablemente se ha sobrepasado el "
            "tiempo límite.\nBytes enviados %ld", dwBytesEnviados);
      else
        wsprintf(szError,"\n%ld bytes escritos", dwBytesEnviados);
    }
```

```
    else // error distinto de ERROR_IO_PENDING
    {
      ClearCommError( m_hDisCom, &dwCodsError, &EstadoCom );
      if ( dwCodsError > 0 )
      {
        wsprintf( szError, "Com: <CE-%u>", dwCodsError );
        AfxMessageBox( szError, MB_OK | MB_ICONEXCLAMATION );
      }
      return FALSE;
    }
  }
  return TRUE;
}

BOOL CCommView::CortarConexion()
{
  m_ConexionEstablecida = FALSE; // finaliza el hilo

  // Inhabilitar todos los eventos
  SetCommMask( m_hDisCom, 0 );

  // Esperar hasta que el hilo finalice
  while( m_bHiloActivo );

  // Desactivar DTR
  EscapeCommFunction( m_hDisCom, CLRDTR );

  // Terminar las operaciones de lectura y escritura pendientes
  // y limpiar las colas Rx y Tx
  PurgeComm( m_hDisCom, PURGE_TXABORT | PURGE_RXABORT |
                        PURGE_TXCLEAR | PURGE_RXCLEAR );

  CloseHandle( m_hDisCom );
  m_hDisCom = NULL;

  return TRUE;
}
/////////////////////////////////////////////////////////////////////////////
// CCommView message handlers

void CCommView::OnInitialUpdate()
{
  CFormView::OnInitialUpdate();

  // Ajustar el tamaño de la ventana marco a la vista
  ResizeParentToFit( FALSE );

  // Iniciar el puerto y los controles de la IU
  UpdateData(FALSE);
  //m_botonEnviar.EnableWindow(FALSE);
  GetDlgItem(IDC_INICIAR)->EnableWindow(FALSE);
  GetDlgItem(IDC_IRPOS)->EnableWindow(FALSE);
  Iniciar(); // configuración inicial del puerto COM
}

void CCommView::OnConfigParam()
{
  // Crea el objeto de diálogo
```

```
  CParamCom dlg;

  // Actualizar los datos del diálogo
  dlg.m_nPuerto       = m_indPuerto;
  dlg.m_nBaudios      = m_indBaudios;
  dlg.m_nParidad      = m_indParidad;
  dlg.m_nBitsCar      = m_indBitsCar;
  dlg.m_nBitsParada   = m_indBitsParada;
  dlg.m_nControlFlujo = m_indControlFlujo;

  // Mostrar el cuadro de diálogo y verificar el botón pulsado
  if(dlg.DoModal() != IDOK) return;

  // Actualizar la configuración
  m_indPuerto       = dlg.m_nPuerto;
  m_indBaudios      = dlg.m_nBaudios;
  m_indParidad      = dlg.m_nParidad;
  m_indBitsCar      = dlg.m_nBitsCar;
  m_indBitsParada   = dlg.m_nBitsParada;
  m_indControlFlujo = dlg.m_nControlFlujo;

  if ( EstablecerConexion() )
  {
    m_ConexionEstablecida = TRUE;
    AfxMessageBox( "Puerto de comunicaciones abierto",
                   MB_OK | MB_ICONEXCLAMATION );
    //m_botonEnviar.EnableWindow(TRUE);

    GetDlgItem(IDC_INICIAR)->EnableWindow(TRUE);
  }
}

void CCommView::OnUpdateConfigParam(CCmdUI* pCmdUI)
{
  pCmdUI->Enable(!m_ConexionEstablecida);
}

void CCommView::OnConexionEstablecer()
{
  if ( EstablecerConexion() )
  {
    UpdateData(TRUE);
    //m_botonEnviar.EnableWindow(TRUE);
    UpdateData(FALSE);
  }
}

void CCommView::OnUpdateConexionEstablecer(CCmdUI* pCmdUI)
{
  pCmdUI->Enable(!m_ConexionEstablecida);
}

void CCommView::OnConexionCortar()
{
  Terminar();   // guardar la configuración
  CortarConexion();
  //m_botonEnviar.EnableWindow(FALSE);
}
```

```
void CCommView::OnUpdateConexionCortar(CCmdUI* pCmdUI)
{
  pCmdUI->Enable(m_ConexionEstablecida);
}

void CCommView::OnEnviar()
{
  //int vr, n;
  //BYTE *pszBytes;

  //UpdateData(TRUE);
  //// Enviar los datos que hay en la caja transmisión
  //if ( n = m_tx.GetLength() )
  //{
  //  pszBytes = (BYTE *)m_tx.GetBuffer(n+1);
  //  vr = EscribirCarsPuerto(pszBytes, n );
  //  m_tx.ReleaseBuffer();
  //  // Eliminar los caracteres transmitidos
  //  if ( vr )
  //  {
  //    m_tx = "";
  //    UpdateData( FALSE );
  //  }
  //}
}

void CCommView::OnVisualizarCars(BYTE *pszBytes, int nBytes)
{
    UpdateData(TRUE);
    CString str((char*)pszBytes);
    CString cadena= str;

    int valorASCII1 = (int)cadena[0];
    int valorASCII2 = (int)cadena[cadena.GetLength() - 1];


   // Probando protocolo de comunicaión
    //m_rx = str;
    //UpdateData(FALSE);

    if (valorASCII1==2)
    {
        if (valorASCII2 ==10) //Probar con su valor en ASCII ya funciona pero el error no
esta aqui
        {

            CString field;

            CArray<CString, CString> v;

            int index = 0;
            // last argument is the delimitter
            while (AfxExtractSubString(field, cadena, index, _T(',')))
            {
                v.Add(field);
                ++index;
            }
```

```
            m_rx = v[0];
            m_ry = v[1];
            m_rz = v[2];

            //AfxMessageBox(m_rz, 0, 0);

            UpdateData(FALSE);
        }
    }
    m_rx = "";
}


void CCommView::OnBnClickedIniciar()
{
    GetDlgItem(IDC_IRPOS)->EnableWindow(TRUE);

    int vr, n;
    BYTE* pszBytes;
    CString comando = "1";
    UpdateData(TRUE);
   /* m_rx = "0";
    m_ry = "0";
    m_rz = "0";*/

    bnx = true;
    bnyz = true;

    // Enviar los datos que hay en la caja transmisión
    if (n = comando.GetLength())
    {
        pszBytes = (BYTE*)comando.GetBuffer(n + 1);
        vr = EscribirCarsPuerto(pszBytes, n);
        comando.ReleaseBuffer();
        // Eliminar los caracteres transmitidos
        if (vr)
        {
            comando = "";
            UpdateData(FALSE);
        }
    }

    // TODO: agregar aquí inicialización adicional

    if (!joyGetNumDevs())
    {
        MessageBox(_T("No hay joystick instalado"), _T("Sin joystick"), MB_ICONERROR);
        //EndDialog(IDCANCEL);
    }
    else
        if (joySetCapture(this->m_hWnd, JOYSTICKID1, NULL, FALSE) != JOYERR_NOERROR)
        {
            MessageBox(_T("No se ha podido capturar el joystick"), _T("Error al
capturar"), MB_ICONERROR);
            //EndDialog(IDCANCEL);
        }
```

```
    //CString cadena = "0,0,0";

    //int nTokenPos = 0;
    //CString strToken = cadena.Tokenize(_T(","), nTokenPos);
    //AfxMessageBox(strToken, 0, 0);

    //while (!strToken.IsEmpty())
    //{
    //    // do something with strToken
    //    // ....
    //    strToken = cadena.Tokenize(_T(""), nTokenPos);
    //    AfxMessageBox(strToken, 0, 0);
    //}



   //Prueba 2
    /*std::vector<std::string> substrings;
    string str = "Hola,Como,Estas";

    for (size_t i = 0; i < str.size(); i += n) {
        substrings.push_back(str.substr(i, n));
    }

    std::vector<std::string> split(const std::string & str, int n)
    {
        std::vector<std::string> substrings;

        for (size_t i = 0; i < str.size(); i += n) {
            substrings.push_back(str.substr(i, n));
        }

        return substrings;
    }

    int main()
    {
        std::string str = "abcdefg";
        int n = 2;

        std::vector<std::string> tokens = split(str, n);
        for (auto& token : tokens) {
            std::cout << token << std::endl;
        }

        return 0;
    }*/

}




void CCommView::OnBnClickedIrpos()
{
    CIrAPosicion dlg;
```

```
    if (dlg.DoModal() == IDOK)
    {
        int vr, n;
        BYTE* pszBytes;
        CString comando;

        comando = "3," + dlg.m_Avancex + "," + dlg.m_Avancey + "," + dlg.m_Avancez + ","+
dlg.m_velocidades;

        //Puder que aqui este el error
        UpdateData(TRUE);
        // Enviar los datos que hay en la caja transmisión
        if (n = comando.GetLength())
        {
            //m_infoBN = comando;
            pszBytes = (BYTE*)comando.GetBuffer(n + 1);
            vr = EscribirCarsPuerto(pszBytes, n);
            comando.ReleaseBuffer();
            // Eliminar los caracteres transmitidos
            if (vr)
            {
                comando = "";
                UpdateData(FALSE);
            }
        }
    }
    else
    {
        //Por si se utiliza más adelante
    }

}


void CCommView::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    // TODO: Agregue aquí su código de controlador de mensajes o llame al valor
predeterminado
    CString infoPWM;
    CSliderCtrl* pPWM = (CSliderCtrl*)pScrollBar;

    switch (pScrollBar->GetDlgCtrlID())
    {
    case IDC_PWM1:
        UpdateData(TRUE);
        infoPWM.Format("%d", pPWM->GetPos());
        SetDlgItemText(IDC_txPWM1, infoPWM);
        break;

    case IDC_PWM2:
        infoPWM.Format("%d", pPWM->GetPos());
        SetDlgItemText(IDC_txPWM2, infoPWM);
        break;
    }

    OnPWM();
```

```
    CFormView::OnHScroll(nSBCode, nPos, pScrollBar);

}


void CCommView::OnPWM()
{
    //Error agregar al hilo solo aparece una vez y vuelve a 0 debe mantenerlo
    UpdateData(TRUE);

    CString comando;
    CString infoPWM1;
    CString infoPWM2;

    CSliderCtrl* pPWM1 = (CSliderCtrl*)GetDlgItem(IDC_PWM1);
    infoPWM1.Format("%d", pPWM1->GetPos());
    CSliderCtrl* pPWM2 = (CSliderCtrl*)GetDlgItem(IDC_PWM2);
    infoPWM2.Format("%d", pPWM2->GetPos());

    comando = "2," + infoPWM1 + "," + infoPWM2;
    //m_infoBN = comando;

    int vr, n;
    BYTE* pszByt;

    // Enviar el comando correcto (Explicación en el arduino)
    if (n = comando.GetLength())
    {
        pszByt = (BYTE*)comando.GetBuffer(n + 1);
        vr = EscribirCarsPuerto(pszByt, n);
        comando.ReleaseBuffer();
        // Eliminar los caracteres transmitidos

    }
    UpdateData(FALSE);
}


LRESULT CCommView::OnJoy1Move(WPARAM wParam, LPARAM lParam)
{
    UpdateData(true);

    if (m_BJoy)
    {
        //Variable a usar
        int x;
        int yz;



        int vr, n;
        BYTE* pszByt;

        //Cadenas con los comandos
        CString comando;
        CString comandoyz;
```

```
//Asingnando valores
x = LOWORD(lParam) >> 6; //No modificar el 6 control de 1 eje
yz = LOWORD(wParam); //Control de 2 o 3 ejes //Usa Case
x = x - 507;//Iniciarlo en 0
//Todo en digital, analogico solo funciona x

//Eje X
switch (x) //Listo bandera para x
{
case -507:
    comando.Empty();
    comando.Format(_T("4, %i"), x);
    if (!bnx)
    {
        bnx = true;
        UpdateData(true);
        if (n = comando.GetLength())
        {
            pszByt = (BYTE*)comando.GetBuffer(n + 1);
            vr = EscribirCarsPuerto(pszByt, n);
            comando.ReleaseBuffer();
            // Eliminar los caracteres transmitidos
        }
        UpdateData(false);
        //m_infoBN = comando;
        //AfxMessageBox(comando, 0, 0);
    }
    break;
case 0:
    comando.Empty();
    comando.Format(_T("4, %i"), x);
    if (bnx)
    {
        bnx = false;
        UpdateData(true);
        if (n = comando.GetLength())
        {
            pszByt = (BYTE*)comando.GetBuffer(n + 1);
            vr = EscribirCarsPuerto(pszByt, n);
            comando.ReleaseBuffer();
            // Eliminar los caracteres transmitidos

        }
        UpdateData(false);
        //AfxMessageBox(comando, 0, 0);
    }
    break;

case 516:
    comando.Empty();
    comando.Format(_T("4, %i"), x);
    if (!bnx)
    {
        bnx = true;

        UpdateData(true);
        if (n = comando.GetLength())
        {
```

```
            pszByt = (BYTE*)comando.GetBuffer(n + 1);
            vr = EscribirCarsPuerto(pszByt, n);
            comando.ReleaseBuffer();
            // Eliminar los caracteres transmitidos

        }
        UpdateData(false);
        //AfxMessageBox(comando, 0, 0);
    }
    break;
}


//Los dos ejes solo palanca sin diagonales
switch (yz)
{

case 0: //Los dos ejes
    comandoyz.Empty();
    comandoyz.Format(_T("4, %i"), yz);
    if (bnyz)
    {
        bnyz = false;
        UpdateData(true);
        if (n = comandoyz.GetLength())
        {
            pszByt = (BYTE*)comandoyz.GetBuffer(n + 1);
            vr = EscribirCarsPuerto(pszByt, n);
            comandoyz.ReleaseBuffer();
            // Eliminar los caracteres transmitidos
        }
        UpdateData(false);
        //AfxMessageBox(comando, 0, 0);
    }
    break;

    //Eje y
case 2: //Derecha palanca
    comandoyz.Empty();
    comandoyz.Format(_T("4, %i"), yz);
    if (!bnyz)
    {
        bnyz = true;
        UpdateData(true);
        if (n = comandoyz.GetLength())
        {
            pszByt = (BYTE*)comandoyz.GetBuffer(n + 1);
            vr = EscribirCarsPuerto(pszByt, n);
            comandoyz.ReleaseBuffer();
            // Eliminar los caracteres transmitidos
        }
        UpdateData(false);
        //AfxMessageBox(comando, 0, 0);
    }
    break;

case 8: //Izquierda palanca
    comandoyz.Empty();
```

```
            comandoyz.Format(_T("4, %i"), yz);
            if (!bnyz)
            {
                bnyz = true;
                UpdateData(true);
                if (n = comandoyz.GetLength())
                {
                    pszByt = (BYTE*)comandoyz.GetBuffer(n + 1);
                    vr = EscribirCarsPuerto(pszByt, n);
                    comandoyz.ReleaseBuffer();
                    // Eliminar los caracteres transmitidos
                }
                UpdateData(false);
                //AfxMessageBox(comando, 0, 0);
            }
            break;

            //Eje z
        case 1: //Arriba palanca
            comandoyz.Empty();
            comandoyz.Format(_T("4, %i"), yz);
            if (!bnyz)
            {
                bnyz = true;
                UpdateData(true);
                if (n = comandoyz.GetLength())
                {
                    pszByt = (BYTE*)comandoyz.GetBuffer(n + 1);
                    vr = EscribirCarsPuerto(pszByt, n);
                    comandoyz.ReleaseBuffer();
                    // Eliminar los caracteres transmitidos
                }
                UpdateData(false);
                //AfxMessageBox(comando, 0, 0);
            }
            break;

        case 4: //Abajo palanca
            comandoyz.Empty();
            comandoyz.Format(_T("4, %i"), yz);
            if (!bnyz)
            {
                bnyz = true;
                UpdateData(true);
                if (n = comandoyz.GetLength())
                {
                    pszByt = (BYTE*)comandoyz.GetBuffer(n + 1);
                    vr = EscribirCarsPuerto(pszByt, n);
                    comandoyz.ReleaseBuffer();
                    // Eliminar los caracteres transmitidos
                }
                UpdateData(false);
                //AfxMessageBox(comando, 0, 0);
            }
            break;
        }
    }
    else
```

```
    {
        /*m_ejex = "";
        m_ejey = "";
        m_ejez = "";*/
        UpdateData(false);
    }
    return 0;
}
```

MainFrm.cpp

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "Comm.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
	//{{AFX_MSG_MAP(CMainFrame)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
	// TODO: add member initialization code here

}

CMainFrame::~CMainFrame()
{
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
	// TODO: Modify the Window class or styles here by modifying
	//  the CREATESTRUCT cs

	return CFrameWnd::PreCreateWindow(cs);
}
```

```
/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
	CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
	CFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
```

ParamCom.cpp

```
// ParamCom.cpp : implementation file
//

#include "stdafx.h"
#include "Comm.h"
#include "ParamCom.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CParamCom dialog


CParamCom::CParamCom(CWnd* pParent /*=NULL*/)
  : CDialog(CParamCom::IDD, pParent)
{
  //{{AFX_DATA_INIT(CParamCom)
  m_nBitsCar = -1;
  m_nBaudios = -1;
  m_nBitsParada = -1;
  m_nControlFlujo = -1;
  m_nPuerto = -1;
  m_nParidad = -1;
  //}}AFX_DATA_INIT
}


void CParamCom::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CParamCom)
```

```
  DDX_CBIndex(pDX, IDC_BITSCAR, m_nBitsCar);
  DDX_CBIndex(pDX, IDC_BAUDIOS, m_nBaudios);
  DDX_CBIndex(pDX, IDC_BITSPARADA, m_nBitsParada);
  DDX_CBIndex(pDX, IDC_CONTROLFLUJO, m_nControlFlujo);
  DDX_CBIndex(pDX, IDC_PUERTO, m_nPuerto);
  DDX_CBIndex(pDX, IDC_PARIDAD, m_nParidad);
  //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CParamCom, CDialog)
  //{{AFX_MSG_MAP(CParamCom)
  ON_BN_CLICKED(IDC_DEFAULT, OnPorOmision)
  //}}AFX_MSG_MAP

END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CParamCom message handlers

void CParamCom::OnPorOmision()
{
  // Parámetros por defecto de la comunicación
  m_nPuerto       = 1;    // COM2
  m_nBaudios      = 6;    // 9600
  m_nParidad      = 0;    // Ninguna
  m_nBitsCar      = 4;    // 8
  m_nBitsParada   = 0;    // 1
  m_nControlFlujo = 1;    // Xon/Xoff

  UpdateData(FALSE); // Refrescar las listas desplegables (combobox)
}
```

Anexo F Código fuente Arduino Nano

```
//Timers MsTimer2-> Inidca Posición
//No usar timer1
#include <TimerOne.h>
#include <MsTimer2.h>

//PWM
#define PWMA 9                // El PWM está conectado al pin D9
#define PWMB 10               //PWMA=PWM1    PWMB=PWM2

//Motores
  //X
  #define xA 7
  #define xB 6
  #define xC 5
  #define xD 4
  #define xE 8
  //Y
  #define yA 3
  #define yB 2
  #define yC A4
  #define yD A5
  #define yE A7
  //Z
  #define zA A0
  #define zB A1
  #define zC A3
  #define zD A2
  #define zE A6

//Variable Motores
  //Variable Motor x
  int indice_x = 0;
  int cnt_x = 0;

  //Variable Motor y
  int indice_y ;
  int cnt_y = 0;

  //Variable Motor z
  int indice_z = 0;
  int cnt_z = 0;


//Cadena
String data = "";

//Datos a procesar en la cadena
byte bandera=0;
 byte bn_mov=0;

 //PWM'S
    //PWM1
    byte PWM1=0;
    int r_PWM1=150;
```

```
    //PWM2
    byte PWM2=0;
    int r_PWM2=50;

  //Motores  (Posiciones relativas)
  int pos_x = 0;
  int pos_y = 0;
  int pos_z = 0;

  //Velocidades (Velociadad default)
  byte vel=5;
  //Velocidad por separado
//  byte vel_x=5;
//  byte vel_y=5;
//  byte vel_z=5;


//  //Pruebas
//  int contador;

  //Joystick
int Joy_xyz;

void setup() {
  TCCR2B = TCCR2B & B11111000 | B00000010; // for PWM frequency of 3.92116 Hz
= 4KHz
  pinMode(PWMA, OUTPUT);       // Configura el pin 9 como salida PWM1
  analogWrite(PWMA, 0);

  pinMode(PWMB, OUTPUT);       // Configura el pin 10 como salida PWM2
  analogWrite(PWMB, 0);

  //Inicializar
    //Para X
      //Motores
      int indice_x = 0;
      pinMode(xA, OUTPUT); // Configuración de los PINes como salida digital
Motor x
      pinMode(xB, OUTPUT);
      pinMode(xC, OUTPUT);
      pinMode(xD, OUTPUT);
      pasox(indice_x);
      //Opto
      pinMode(xE, INPUT);

    //Para Y
      //Motores
      int indice_y = 0;
      pinMode(yA, OUTPUT); // Configuración de los PINes como salida digital
Motor y
      pinMode(yB, OUTPUT);
      pinMode(yC, OUTPUT);
      pinMode(yD, OUTPUT);
      pasoy(indice_y);
      //Opto
      pinMode(yE, INPUT);
```

```
    //Para Z
      //Motores
      int indice_z = 0;
      pinMode(zA, OUTPUT); // Configuración de los PINes como salida digital
Motor y
      pinMode(zB, OUTPUT);
      pinMode(zC, OUTPUT);
      pinMode(zD, OUTPUT);
      pasoz(indice_z);
      //Opto
      pinMode(zE, INPUT);


    //Inicio comunicación serial
    Serial.begin(9600);

  //Detener el inicio de setup()
  do {
    if (Serial.available() > 0)
    {
      data = "";
      do {
        char caracter_leido;
        delay(5);
        caracter_leido = Serial.read();
        data += caracter_leido;
      } while (Serial.available() > 0);
      //Serial.println("Buscando 0");
    }
  } while (data.toInt() != 1);

    //Ir a posición 0
      //Para X
//   bool estadoOpto_x = digitalRead(xE);
//      do {
//      estadoOpto_x = digitalRead(xE);
//      indice_x = decrementa(indice_x);
//      pasox(indice_x);
//  delay (vel);
//  } while (!estadoOpto_x);

  //inicializar posiciones motores
    //Motor x
    cnt_x = 0;
    pos_x = 0;
    //Motor y
    cnt_y = 0;
    pos_y = 0;
    //Motor z
    cnt_z = 0;
    pos_z = 0;

//timer2
  MsTimer2::set(180, ISR_Serie); // 180ms period
  MsTimer2::start();

}
```

```
void loop() {
  //Recibir cadena
  if (Serial.available() > 0) {
    data = "";
    do {
      char caracter_leido;
      delay(5);
      caracter_leido = Serial.read();
      data += caracter_leido;
    } while (Serial.available() > 0);

    //Determinar el tipo de comando
    data.trim();
    bandera = (data.toInt());
    //Serial.println(bandera);
    //Serial.println(data);

    //Actuar segun comando
    switch (bandera) {

      case 2: //PWM's
      //Procesamiento cadena
          data.remove(0, ((data.indexOf(",")) + 1));
          PWM1 = (data.toInt());
           //Serial.println(PWM1);//No indispensable
          data.remove(0, ((data.indexOf(",")) + 1));
          PWM2 = (data.toInt());
           //Serial.println(PWM2);//No indispensable

           //Conversión Enceder laser
           r_PWM1 = round(PWM1 * 2.55);
           r_PWM2 = round(PWM2 * 2.55);
           analogWrite(PWMA, r_PWM1);
           analogWrite(PWMB, r_PWM2);
        break;

      case 3: //Movimiento motores teclado
      //Procesamiento cadena
        //Posición
          data.remove(0, ((data.indexOf(",")) + 1));
          pos_x = (data.toInt());
          //Serial.println(pos_x);//No indispensable
          data.remove(0, ((data.indexOf(",")) + 1));
          pos_y = (data.toInt());
          //Serial.println(pos_y);//No indispensable
          data.remove(0, ((data.indexOf(",")) + 1));
          pos_z = (data.toInt());
          //Serial.println(pos_z);//No indispensable
        //Velocidad
          data.remove(0, ((data.indexOf(",")) + 1));
          vel = (data.toInt());
              //Serial.println(vel);//No indispensable

        //Comienza Movimiento Motores
          //Para X
            //Creciente
```

```
            if(pos_x>=0){
              for (int i=0; i<pos_x; i++){
                cnt_x=cnt_x+1;
                  indice_x=incrementa(indice_x);
                  pasox(indice_x);
                  delay (vel);
//                if(cnt_x<6450){//Maximo # de pasos
//
//                }
              }
            }
            //Decreciente
            else if(pos_x<0){
              for (int i=0; i<pos_x*-1; i++){
                cnt_x=cnt_x-1;
                  indice_x=decrementa(indice_x);
                  pasox(indice_x);
                  delay (vel);
//                if(cnt_x>0){//Maximo # de pasos
//
//                }
              }
            }

          //Para y
            //Creciente
            if(pos_y>=0){
              for (int i=0; i<pos_y; i++){
                cnt_y=cnt_y+1;
                  indice_y=incrementa(indice_y);
                  pasoy(indice_y);
                  delay (vel);
//                if(cnt_y<6450){//Maximo # de pasos
//
//                }
              }
            }
            //Decreciente
            else if(pos_y<0){
              for (int i=0; i<pos_y*-1; i++){
                cnt_y=cnt_y-1;
                  indice_y=decrementa(indice_y);
                  pasoy(indice_y);
                  delay (vel);
//                if(cnt_y>0){//Maximo # de pasos
//
//                }
              }
            }

          //Para z
            //Creciente
            if(pos_z>=0){
              for (int i=0; i<pos_z; i++){
                  cnt_z=cnt_z+1;
                  indice_z=incrementa(indice_z);
                  pasoz(indice_z);
```

```
                    delay (vel);

//                  if(cnt_z<6450){//Maximo # de pasos
//
//                  }
                  }
                }
                //Decreciente
                else if(pos_z<0){
                  for (int i=0; i<pos_z*-1; i++){
                      cnt_z=cnt_z-1;
                      indice_z=decrementa(indice_z);
                      pasoz(indice_z);
                      delay (vel);

//                  if(cnt_z>0){//Maximo # de pasos
//
//                  }
                  }
                }
            break;

    break;
    }// Ejecuaciòn de los comandos 1,2,3

    //Serial.println(cnt_x);//prueba no indispensable

   //Estado de reposo
    //Motor x
    digitalWrite(xA, 0);
    digitalWrite(xB, 0);
    digitalWrite(xC, 0);
    digitalWrite(xD, 0);
    //Motor y
    digitalWrite(yA, 0);
    digitalWrite(yB, 0);
    digitalWrite(yC, 0);
    digitalWrite(yD, 0);
    //Motor x
    digitalWrite(zA, 0);
    digitalWrite(zB, 0);
    digitalWrite(zC, 0);
    digitalWrite(zD, 0);
  }

  //Continuaciòn nde los posibles casos de comando 4
  switch (bandera) {
    case 4://Joystick
      data.remove(0, ((data.indexOf(",")) + 1));
        Joy_xyz = (data.toInt());

        //Comienza el avance dependiendo de la dirección en el Joystick
        switch (Joy_xyz) {
          //Para x
          case -507://Giro antihorario avanza a la izquierda
            do {
              cnt_x=cnt_x-1;
```

```
              indice_x=decrementa(indice_x);
              pasox(indice_x);
              delay (4);
              Joy_xyz =Serial.read();
//            if(cnt_x>0){//Maximo # de pasos
//
//            }
            } while (Joy_xyz==0);//Duda de si esto esta bien
          break;
          case 516://Giro horario avanza a la derecha
            do {
              cnt_x=cnt_x+1;
                indice_x=incrementa(indice_x);
                pasox(indice_x);
                delay (4);
                Joy_xyz =Serial.read();
//              if(cnt_x<6450){//Maximo # de pasos
//
//              }
            } while (Joy_xyz==0);
          break;

          //Para y
          case 8://Giro antihorario avanza a
            do {
              cnt_y=cnt_y-1;
                indice_y=decrementa(indice_y);
                pasoy(indice_y);
                delay (4);
                Joy_xyz =Serial.read();
//              if(cnt_y>0){//Maximo # de pasos
//
//              }
            } while (Joy_xyz==0);//Duda de si esto esta bien
          break;
          case 2://Giro horario avanza a
            do {
              cnt_y=cnt_y+1;
                indice_y=incrementa(indice_y);
                pasoy(indice_y);
                delay (4);
                Joy_xyz =Serial.read();
//              if(cnt_y<6450){//Maximo # de pasos
//
//              }
            } while (Joy_xyz==0);
          break;

          //Para z
          case 4://Giro antohorario avanza a
            do {
              cnt_z=cnt_z-1;
                indice_z=decrementa(indice_z);
                pasoz(indice_z);
                delay (4);
                Joy_xyz =Serial.read();
//              if(cnt_z>0){
```

```
//
//                }
            } while (Joy_xyz==0);
          break;
          case 1://Giro antohorario avanza a
            do {
              cnt_z=cnt_z+1;
                indice_z=incrementa(indice_z);
                pasoz(indice_z);
                delay (4);
                Joy_xyz =Serial.read();
//              if(cnt_z<6450){
//
//              }
            } while (Joy_xyz==0);
          break;

        }
  break;
  }//Fin procesamiento cadena cadena/comando

  //Estado de reposo
    //Motor x
    digitalWrite(xA, 0);
    digitalWrite(xB, 0);
    digitalWrite(xC, 0);
    digitalWrite(xD, 0);
    //Motor y
    digitalWrite(yA, 0);
    digitalWrite(yB, 0);
    digitalWrite(yC, 0);
    digitalWrite(yD, 0);
    //Motor x
    digitalWrite(zA, 0);
    digitalWrite(zB, 0);
    digitalWrite(zC, 0);
    digitalWrite(zD, 0);
}

//Avance Motor x
void pasox(int indice) {
  if (indice == 0) {
    digitalWrite(xA, 0);
    digitalWrite(xB, 0);
    digitalWrite(xC, 0);
    digitalWrite(xD, 1);
  }
  if (indice == 1) {
    digitalWrite(xA, 0);
    digitalWrite(xB, 1);
    digitalWrite(xC, 0);
    digitalWrite(xD, 0);
  }
  if (indice == 2) {
    digitalWrite(xA, 0);
    digitalWrite(xB, 0);
    digitalWrite(xC, 1);
```

```
    digitalWrite(xD, 0);
  }
  if (indice == 3) {
    digitalWrite(xA, 1);
    digitalWrite(xB, 0);
    digitalWrite(xC, 0);
    digitalWrite(xD, 0);
  }
}


//Avance Motor y
void pasoy(int indice) {
  if (indice == 0) {
    digitalWrite(yA, 0);
    digitalWrite(yB, 0);
    digitalWrite(yC, 0);
    digitalWrite(yD, 1);
  }
  if (indice == 1) {
    digitalWrite(yA, 0);
    digitalWrite(yB, 1);
    digitalWrite(yC, 0);
    digitalWrite(yD, 0);
  }
  if (indice == 2) {
    digitalWrite(yA, 0);
    digitalWrite(yB, 0);
    digitalWrite(yC, 1);
    digitalWrite(yD, 0);
  }
  if (indice == 3) {
    digitalWrite(yA, 1);
    digitalWrite(yB, 0);
    digitalWrite(yC, 0);
    digitalWrite(yD, 0);
  }
}

//Avance Motor z
void pasoz(int indice) {
  if (indice == 0) {
    digitalWrite(zA, 0);
    digitalWrite(zB, 0);
    digitalWrite(zC, 0);
    digitalWrite(zD, 1);
  }
  if (indice == 1) {
    digitalWrite(zA, 0);
    digitalWrite(zB, 1);
    digitalWrite(zC, 0);
    digitalWrite(zD, 0);
  }
  if (indice == 2) {
    digitalWrite(zA, 0);
    digitalWrite(zB, 0);
    digitalWrite(zC, 1);
```

```
    digitalWrite(zD, 0);
  }
  if (indice == 3) {
    digitalWrite(zA, 1);
    digitalWrite(zB, 0);
    digitalWrite(zC, 0);
    digitalWrite(zD, 0);
  }
}


//Cambio indice Mores x,y,z (Depende del argumento)
int incrementa (int indice) {
  if (indice == 0)
    indice = 1;
  else if (indice == 1)
    indice = 2;
  else if (indice == 2)
    indice = 3;
  else if (indice == 3)
    indice = 0;
  return indice;
}
int decrementa(int indice) {
  if (indice == 0)
    indice = 3;
  else if (indice == 1)
    indice = 0;
  else if (indice == 2)
    indice = 1;
  else if (indice == 3)
    indice = 2;
  return indice;
}

void ISR_Serie()//indicar posición cuando se cambie la función
{
  //Para X
  Serial.print("\x02");
  Serial.print(cnt_x);Serial.print(",");
Serial.print(cnt_y);Serial.print(",");Serial.println(cnt_z);

}
```

Referencias bibliográficas

Arduino (2021) Arduino [Internet], Disponible desde < https://store-usa.arduino.cc/products/arduino-nano?selectedStore=us> [Acceso 14 de abril de 2023].

Arduino (2022) Arduino [Internet], Disponible desde < https://www.arduino.cc/en/software> [Acceso 14 de abril de 2023].

Ceballos F J. (1999) *Microsoft Visual C++6 Programación Avanzada en Win32*. Alfaomega, México, p. 852.

Kruglinski DJ, Shepherd G y Wingo S. (1988) *Programming Microsoft Visual C++*. Microsoft Press, USA, p. 1153.

Mach3 (2023) New Fangled Solutions [Internet], Disponible desde <https://www.machsupport.com/software/mach3/> [Acceso 14 de abril de 2023]