UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Mantenimiento e Implementación de la Planta Cooperativa A255-A465

TESIS

Que para obtener el título de

Ingeniero Eléctrico Electrónico

P R E S E N T A

Hiroshi Lestrade Hernández

DIRECTOR DE TESIS

Dr. Marco Antonio Arteaga Pérez

Ciudad Universitaria, Cd. Mx., 2024

Para Pam, sin ti no habría llegado hasta aquí.

Agradecimientos

- A mis padres, Verónica y Héctor, y mi hermana Dana, por creer en mí.
- A mis abuelos, Yolanda y Crispín, por todo su amor.
- A Pamela, por acompañarme en el proceso y nunca dejarme solo.
- Al Doctor Marco Antonio Arteaga Pérez y al Maestro Evert Josué Guajardo Benavides por darme el voto de confianza y abrirme las puertas del laboratorio.
- A la Universidad Nacional Autónoma de México, por darme los medios para crecer.

Índice

Ilustraciones

Tablas

Códigos

Ecuaciones

1 Introducción

El laboratorio de Robótica del posgrado de la Facultad de Ingeniería de la Universidad Nacional Autónoma de México cuenta con una planta formada por dos manipuladores industriales capaces de trabajar de forma cooperativa. Este sistema es producto de un esfuerzo de integración continua y ha jugado un papel central en diversos proyectos de investigación realizados en el laboratorio que han buscado llevarlo desde la adquisición de los robots hasta la configuración coordinada que presenta actualmente.

Desafortunadamente, durante el periodo de confinamiento impuesto por la pandemia de COVID-19, hubo la necesidad de suspender temporalmente las actividades en el laboratorio, sin embargo, al reanudar se descubrió que la planta simplemente ya no funcionaba. El abandono, la inoperatividad y la falta de mantenimiento siempre tienen un impacto negativo en cualquier sistema. Además, se debe añadir que se cambió equipo de lugar, el aseo fue suspendido y se realizaron obras estructurales dentro del laboratorio como la instalación de un sistema de flujo de aire, del cual, un tubo de ventilación pasa sobre la planta. Todo esto en conjunto pudo, si no ser el causante, al menos ser detonante del estado inoperable en que se encontró.

Por otro lado, la planta cooperativa ha evolucionado a través de una serie de proyectos que la han transformando a lo largo del tiempo y que persiguen un objetivo común: una arquitectura abierta. Inicialmente se contaba sólo con dos robots de la extinta compañía canadiense CRS Robotics, un A255 y un A465 gestionados independientemente y cada uno con su respectivo controlador C500-C que, sólo ofrece un *teach pendant* - dispositivo de control y programación manual - y un software sencillo como métodos de control. Posteriormente, se construyó una tarjeta que inhibe dichos controladores exponiendo las señales de los robots para leer los *enconders* y enviar los voltajes que controlan los motores, permitiendo mayor libertad en el uso de los robots. Adicional a ello, se diseñó una segunda placa para adquisición de datos que acondiciona las señales para incorporarlas en un controlador de National Instruments cRIO-9014, además de agregar un paro de emergencia externo como medida de seguridad.

Los robots, junto con todas las modificaciones citadas, constituyen en su conjunto la planta cooperativa A255-A465. Su integración permite una mayor libertad experimental, superando las limitaciones de los controladores de fábrica. A pesar de esto, el cRIO-9014 expone la comunicación con la planta para ser programada desde un computador común, pero, el hecho de que sea un controlador con arquitectura de fabricante aún nos condiciona a usar los medios que National Instruments provee para trabajar y a la continuidad que decida darles. Distintos proyectos sobre la planta han sorteado este problema de implementación y uso de múltiples formas, sin embargo, la documentación no es suficiente ni homogénea ya que siempre ha sido un problema a sortear y no el objetivo central de estudio.

Lo expuesto en esta introducción habla del contexto de la planta al inicio del proyecto y permite establecer el problema a resolver. En esencia, se tiene que devolver la planta a un estado operativo y se necesita un marco de trabajo que permita homogeneizar los proyectos que se generen con ella. Adicionalmente, se pretende generar documentación que facilite el mantenimiento de la planta en su conjunto y la implementación de proyectos, para así centrar los nuevos desarrollos directamente en su implementación y no tener que lidiar con detalles en un bajo nivel de abstracción.

También es importante mencionar que este proyecto inició como una actividad de servicio social que escaló hasta tomar la densidad para convertirse en un trabajo de tesis; el potencial que tiene para seguir siendo investigando es muy alto y hay muchas ideas que pudiesen haberse incluido, pero en algún punto

se tiene que acotar el tema y a consideración personal, la base para generar proyectos es una buena delimitación para el tema de estudio. Se espera que este trabajo pueda servir como punto de partida para el trabajo de futuros estudiantes e investigadores en el laboratorio y así, potenciar ideas y proyectos reduciendo el trabajo para llegar a sus objetivos.

1.1 Objetivos y alcances

Una vez presentado el contexto y entrando en materia específica, el objetivo de esta tesis es doble. En primer lugar, la planta cooperativa A255-A465 será devuelta a su estado operativo, documentando una memoria de fallos y su respectiva solución para facilitar su mantenimiento futuro. En segundo lugar, será creado un marco de trabajo que permita abstraer la operación de bajo nivel - esencialmente de software - para centrarse en el uso directo de los robots de forma individual y cooperativa, apegándonos al ideal de la arquitectura abierta que se ha desarrollado alrededor de ella.

Como productos, se perseguirá:

- La documentación de la memoria de fallos a través de este escrito
- Una biblioteca de software que incluya las funciones principales de la planta a nivel de usuario y que permita implementar proyectos en diferentes plataformas
- Una interfaz humano-máquina que utilice la biblioteca desarrollada y permita comprobar las funciones básicas en los robots
- Documentación de implementación y uso, y un experimento cooperativo que compruebe el correcto funcionamiento del sistema

El método a seguir para abordar esta tarea consistirá en recopilar toda la información disponible sobre la planta, las modificaciones que se la han hecho y sus implementaciones anteriores para comprender en detalle su proceso de funcionamiento y detectar los posibles puntos de error. Posteriormente, se abordarán sistemáticamente las fallas, desde el encendido hasta lograr el movimiento conjunto de los robots, documentando cada paso del proceso. Teniendo un estado operativo comprobable, se comenzará a diseñar la biblioteca de software, introduciendo capas de abstracción que separen las configuraciones de bajo nivel del controlador de las funciones con las que el usuario interactuaría. Dicha biblioteca debe implementarse en el lenguaje de programación seleccionado para desarrollar el proyecto y eso daría nota para definir una ley de control que permita operar los robots, en la cual - dado que este punto es justo la puerta a futuras investigaciones - se usará un sencillo controlador PID. La interfaz humano-máquina debe implementar funciones básicas de trabajo:

- Ir a un punto en coordenadas articulares
- Ir a un punto en coordenadas cartesianas
- Volver al punto de referencia y ser capaz de establecerlo
- Recibir conjuntos de instrucciones a ejecutar
- Un control manual de cada articulación

todo esto para cada robot de forma individual y cooperativa. Finalmente, utilizando la función de programación por instrucciones de la interfaz, se realizará un experimento de posicionamiento de forma conjunta.

Los pasos mencionados constituyen también la estructura de esta tesis, comenzando por esta introducción y la Sección 1.2 que hablan del contexto de la planta, los objetivos de este trabajo y el marco teórico

alrededor del enfoque aplicado, pasando a la Sección 1.3 donde se dará una descripción del sistema sintetizando la documentación de los robots y la información recabada de la planta. En el Capítulo 2 se abordará el estado inicial de la planta y la memoria de mantenimiento, culminando con una tabla de fallos y soluciones a modo de resumen. El Capítulo 3 define los requerimientos funcionales de un sistema robótico desde la perspectiva del software y con eso se da forma a la arquitectura utilizada en el sistema, también se definen las abstracciones de los robots respecto a propiedades y comportamiento, además de su cinemática directa e inversa, ley de control y el seguimiento de trayectoria, junto con el intérprete que gestionará las instrucciones con que el usuario se comunicará con la planta. Para el cuarto capítulo se construirán los componentes descritos durante el Capítulo 3 y se implementarán a través de una interfaz humano-máquina que cumpla con los requerimientos previamente establecidos, ejecutando el experimento definido y mostrando los resultados de la operación. Al final, el quinto capítulo resume todo el abordaje, comenta los resultados y profundiza en las implicaciones del marco de trabajo para proyectos futuros, además de dar un panorama de ideas para nuevos proyectos.

1.2 Arquitectura y Control de Robots

En esta sección se mencionan distintos conceptos que serán útiles a lo largo del proyecto, comenzando con las definiciones más básicas. El término *robot* fue mencionado por primera vez en el libro Rossum's Universal Robots derivado de la palabra checa *robota* que significa *trabajo forzado* [1]. En general, podemos describir a un robot como un sistema electromecánico, autónomo, controlado por una computadora y que realiza una tarea. El problema con esta definición es su amplitud, donde múltiples dispositivos podrían entrar sin ser necesariamente robots. Esto último nos lleva a definir tipos específicos de robots que apoyen a aclarar el término. Múltiples categorías pueden encontrarse y las líneas que las dividen son difusas, es por ello que en este texto nos centraremos en el único tipo de robot que nos atañe: el **manipulador industrial.** En una definición más concreta del Robot Institute of America:

> *Un robot es un manipulador multifuncional y reprogramable diseñado para mover material, partes, herramientas o dispositivos especializados a través de movimientos variables programados para la ejecución de diversas tareas* [1].

Definición que, en un sentido más técnico, podemos explicar a través de una **cadena cinemática** formada por eslabones y articulaciones que dotan de estructura y movilidad al robot, dándole en algunos casos una forma similar a la de un brazo humano. Para mayor simplicidad, cada vez que se diga la palabra robot en este texto, nos estaremos refiriendo a un manipulador industrial.

Unimate fue el primer robot industrial desarrollado por George Devol y Joseph Engelberger quienes fundaron la primera empresa fabricante de robots, Unimation, en 1953. Setenta años después, la teoría detrás de estos robots, bajo el concepto de cadena cinemática y todas las diferentes configuraciones que se pueden obtener según el tipo y número de articulaciones, es bastante sólida y probada. Pareciera que los manipuladores que vemos a través de distintos fabricantes sólo son cambios de diseño. Entonces ¿qué busca la investigación en robótica? El hecho de que la apariencia y configuración de múltiples robots sea similar, típicamente configuraciones de tres grados de libertad con una muñeca esférica que añade otros tres grados para la orientación, no significa que el problema esté resuelto, sino que nos da un sistema dinámico típico donde el *cómo* puede variar sustancialmente. Por ejemplo, aunque tengamos una sencilla tarea de posicionamiento – moverse del punto A al punto B – podemos definir múltiples técnicas de control, distintos algoritmos para ejecutarlas, diferentes trayectorias entre los puntos y, todo esto tendría que lidiar con parametrizaciones, mediciones y el entorno del robot. Por tanto, la **investigación robótica**

se consolida como un campo bajo la idea integral de la configuración del robot, la tarea que va a realizar y el cómo la va ejecutar, dando lugar a múltiples soluciones para un mismo experimento.

Retomando el último punto del ejemplo anterior, el espacio del robot es una consideración importante, pero se puede complicar significativamente la tarea si agregásemos un segundo robot al entorno y esto nos lleva a definir la **robótica colaborativa**. En un sistema cooperativo, los robots deben trabajar conjuntamente para realizar tareas compartidas – también puede incluirse la colaboración con humanos, pero ello implica consideraciones de seguridad fuera de las cotas de este escrito – comunicándose entre ellos para tener una interoperabilidad segura. Esto amplía aún más el campo de investigación porque la ejecución de una solución de forma aislada puede no necesariamente funcionar en un entorno cooperativo.

De forma general, podemos abstraer todo lo mencionado bajo el concepto de **sistema robótico**. Este debe estar formado – en su configuración más simple – por un sistema de potencia que aporte la energía necesaria, un controlador que gestione las instrucciones para realizar la tarea y un robot que la ejecute. Según el número de instancias de esta forma básica y el acoplamiento entre ellas, podemos tener sistemas aislados o colaborativos. En esencia, un sistema robótico terminará abstrayéndose como unidad, pero hay decisiones de desarrollo que pueden implementarse de forma individual o conjunta a cada uno de los robots y el efecto de dicha selección puede ser mayor o menor según sea el caso. La elección de un control centralizado o distribuido puede ser uno de los componentes que modifique la implementación del sistema de forma más drástica. En el caso del **control distribuido**, cada robot tendría su controlador y realizaría sus propios cálculos de forma individual, ejecutando la operación conjunta a través de la comunicación entre ellos; si uno fallara, el resto puede seguir operando debido a la flexibilidad que otorga la modularidad y el poco acoplamiento de la colaboración. Por otro lado, en el **control centralizado** sólo tenemos un controlador que efectúa la operación de todos los robots al mismo tiempo, lo que otorga robustez y velocidad, al eliminar la necesidad de un sistema de comunicación.

La estructura de un sistema robótico y las decisiones de implementación en su configuración pueden ser agrupadas bajo la idea de **arquitectura** [2] y esto abre la puerta a distintos arreglos que persiguen diferentes propósitos. Desde la vista más general, podemos encontrar dos configuraciones globales: **arquitectura cerrada** o de fabricante y **arquitectura abierta**.

Por arquitectura cerrada entenderemos la configuración de un sistema definida por su fabricante y de la cual, podemos tener información de cómo está construida – hasta cierto punto – pero no tenemos acceso para realizar modificaciones. Como ventajas encontramos la robustez de un producto terminado y la seguridad para realizar una tarea específica. Sin embargo, en la investigación estas ventajas se convierten en limitantes ya que no podemos acceder a modificar el interior del sistema y tenemos que ceñirnos a los métodos de uso que el fabricante provea. Por otro lado, una arquitectura abierta permite que los desarrolladores adapten y expandan el sistema con mayor facilidad al no imponer restricciones en las herramientas a utilizar o el cómo realizar las cosas, agregando características como la flexibilidad, la personalización y la interoperabilidad.

La arquitectura de un sistema robótico puede ser segmentada en dos partes: **arquitectura de software** y **arquitectura de hardware** [2]. Respecto al hardware, la arquitectura de la planta será definida en la siguiente sección, pero podemos adelantar que el objetivo de una arquitectura abierta – aunque se pueda refinar – se ha alcanzado ya. Por otro lado, en el software no se cuenta como tal con una arquitectura y es precisamente lo que buscaremos definir en posteriores capítulos.

Por último, una de las características más importantes de la arquitectura abierta y que será buscada en el desarrollo de esta tesis es la **flexibilidad**. El *IEEE Standard Glossary of Software Engineering Terminology* define la flexibilidad como *la facilidad con que un sistema o componente puede ser modificado para su uso en aplicaciones o entornos diferentes a con el que fue específicamente diseñado* [3] [4]; esta característica es la piedra angular de las decisiones de diseño tomadas, ya que la arquitectura de software que se diseñará busca dotar a la planta cooperativa de la capacidad para adaptarse a distintos proyectos a través de componentes modulares que pueden ser reutilizados, expandidos, reescritos o desacoplados con facilidad.

Todos estos conceptos forman las columnas que sostienen esta tesis, siendo las ideas centrales para el planteamiento que se le dio al problema de la planta, el abordaje para solucionarlo y el enfoque para los objetivos buscados. La planta cooperativa A255-A465 se conforma por dos robots, siendo un sistema robótico, acompañado de un par de componentes que lo llevan un poco más allá de la forma básica; como su nombre lo indica, es un sistema colaborativo y, como veremos en la siguiente sección, cuenta con un solo controlador para ambos robots, lo que nos habla de un control centralizado, sin embargo a nivel de software, buscaremos abstraer ambos robots por separado para emular un control distribuido y así modularizar el sistema a nivel tarea; la cadena de modificaciones alrededor de la planta ha buscado llevar a los robots de la arquitectura cerrada inicial a una arquitectura abierta en su configuración cooperativa y, finalmente, buscaremos subir un escalón en la abstracción donde la implementación a bajo nivel de la planta no sea una carga para proyectos que no se enfoquen en ello.

1.3 Descripción del sistema

Es importante, antes de cualquier etapa de mantenimiento, conocer el sistema a trabajar; si bien no en cada detalle, al menos en su estructura general. Anteriormente se dijo que esta planta es un desarrollo continuo y distintas modificaciones han sido agregadas con el tiempo; por ello, el abordaje para su descripción será elemento a elemento y luego considerando las integraciones.

En primer lugar, tenemos al controlador CRS-C500C (abreviado C500), el cual forma una tercia con cada robot y el *teach pendant* en la configuración original del sistema [4], en las figuras 1, 2 y 3 se observan las parejas formadas con cada robot y el flujo de comunicación original. Esta estructura inicial permitía la operación a nivel tarea sobre los robots, pero no su programación, sin embargo, nos permite entender las funciones generales que realiza el controlador: la adquisición de datos de los codificadores digitales o *encoders* en los robots, una etapa de amplificación que se encarga de proveer el voltaje de salida necesario y un controlador PID de arquitectura cerrada encargado de computar dicha salida. Cada controlador cuenta con botón E-stop que inhibe el suministro de potencia al robot y bloquea sus articulaciones, este, a su vez, tiene un sistema detrás que permite la expansión de la señal de paro de emergencia.

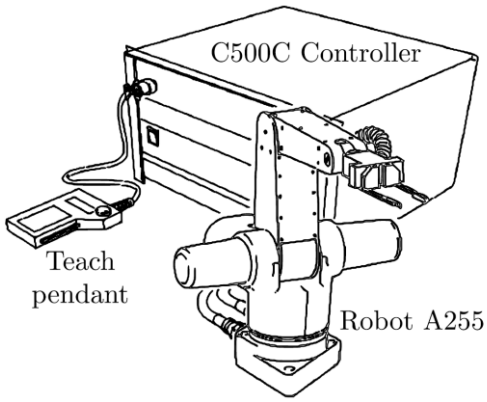

Ilustración 1: Componentes originales del manipulador A255 (García Gutiérrez, 2016)

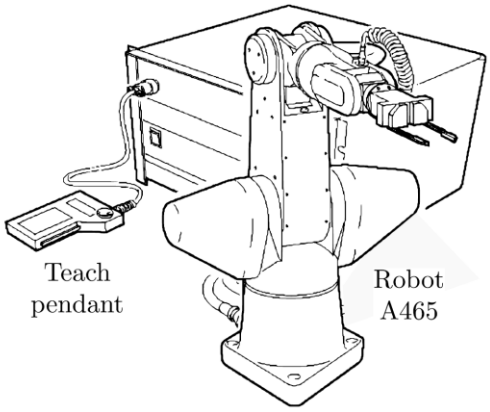

Ilustración 2: Componentes originales del manipulador A465 (García Gutiérrez, 2016)

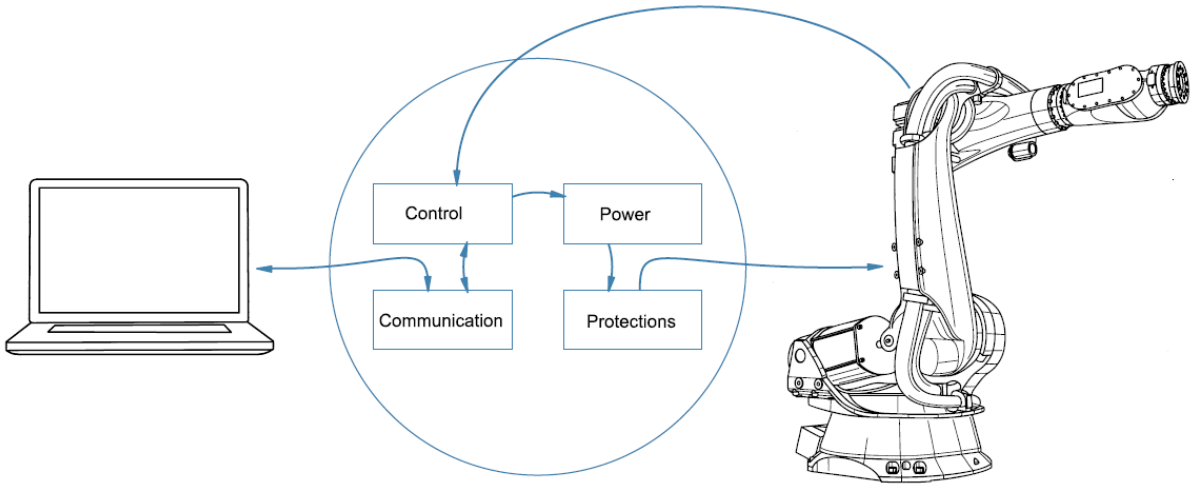

Ilustración 3: Comunicación original de cada robot con su módulo de control C500 (García Gutiérrez, 2016)

Sobre los robots, se cuenta con dos manipuladores modelos A255 y A465 de la compañía CRS Robotics [4]. El A465 posee 6 grados de libertad en una configuración antropomórfica donde las últimas tres articulaciones forman una muñeca esférica, dotándolo de un espacio de trabajo esférico, mientras que su hermano, el A255 posee 5 grados de libertad en una configuración antropomórfica que resulta en un espacio de trabajo esférico limitado en orientación. Las cargas máximas son de 2kg para el A465 y 1kg para el A255, el peso de cada robot es de 31kg y 17kg respectivamente y ambos cuentan con *encoders* incrementales de alta resolución para medir la posición angular. Una consideración importante es que, en el A255 las articulaciones 2, 3 y 4 se encuentran relacionadas en su movimiento por correas dentadas, teniendo el efecto de no alterar la orientación del efector final en la cuarta articulación.

La primera modificación a la planta consiste en una tarjeta de inhibición del control original que expone las señales de los *encoders* en los robots y las señales de voltaje que van a la etapa de amplificación [5]. La instalación se hizo en el interior del gabinete y esto dejó tres nuevos conectores: AB y Z que llevan las señales digitales de los *encoders* y VCOM, con las señales de voltaje.

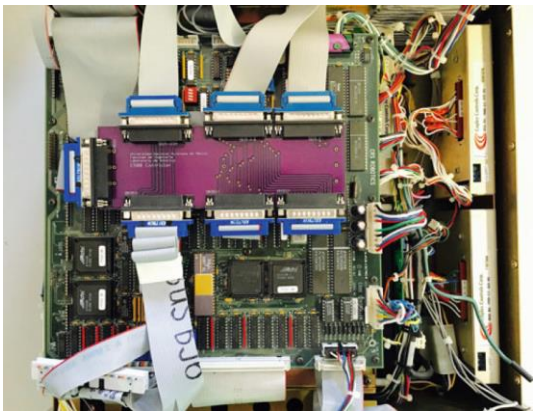

Ilustración 4: Vista interior del C500 con la tarjeta de inhibición del controlador (Arteaga, 2021)

Ilustración 5: Vista trasera del C500 con las conexiones de las señales del robot

Tras exponer las señales del módulo C500 y utilizarlas por primera vez en un controlador, se notó que había pérdidas en la cuenta de los *encoders* por la calidad de las señales digitales, esto derivó en el desarrollo de una tarjeta de adquisición de datos que se encarga de la adecuación de las señales de entrada [5]. En general, la tarjeta de adquisición de datos también sirve como etapa final para unificar la planta en un solo gabinete con el cuál conectarse, ya que se agregaron conectores para las señales de ambos robots, además de conectar una botonera que aprovecha la conexión externa del E-Stop como un paro de emergencia manual más accesible al usuario.

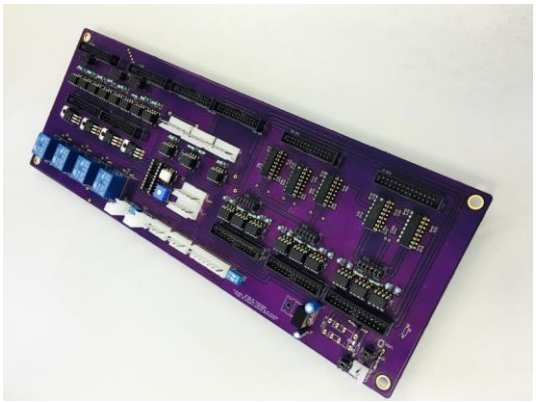

Ilustración 6: Tarjeta de adquisición de datos (García Gutiérrez, 2016)

Al final de la etapa de adquisición de datos se encuentra un módulo CompactRIO de National Instruments, el cual es una FPGA (*Field Programmable Gate Array*) para procesamiento de tareas de alta demanda [4], siendo el encargado de procesar las señales de entrada y salida; su comunicación con la computadora es a través del puerto Ethernet. Para el manejo de las señales, en el gabinete del cRIO se colocaron dos tipos de módulos: NI-9401 para gestionar las entradas y salidas digitales y del cual se agregaron 5 unidades y NI-9263 para las señales analógicas de salida, utilizándose tres módulos. Este arreglo es alimentado por una fuente de poder Quint de Phoenix Contact que suministra 24V y un máximo de 5A.

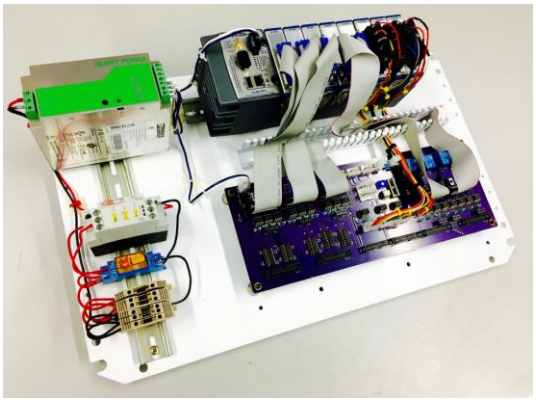

Ilustración 7: Interior del gabinete de la planta cooperativa (García Gutiérrez, 2016)

Finalmente, los elementos que suelen ser dejados fuera de la planta, pero que forman parte del flujo para el contexto del mantenimiento, son la computadora y las unidades UPS. En el caso del computador, dado que está dedicado exclusivamente al uso de la planta, nos referiremos a él como *computadora host*, ya que más adelante modularizaremos este componente. La computadora host cuenta con las siguientes características:

- Marca: Apple
- Modelo: Mac Pro 5.1
- Procesador: 2.8 GHz Quad-Core Intel Xeon 5300
- Ram: 16 GB of 800 MHz DDR2 ECC
- Sistema operativo: Windows XP

Las unidades UPS (Uninterruptable Power Supply), una por cada robot, son especialmente relevantes porque se encargan de la protección eléctrica de los componentes en la planta. Sus características son:

- Marca: APC
- Modelo: Back-UPS XS 1500

2 Memoria de mantenimiento

Este capítulo se centra en la puesta en marcha de la planta cooperativa. Tras haber descrito el sistema en la sección anterior, ahora disponemos de un panorama general de cómo debería operar correctamente. Esto nos permite abordar las fallas de manera secuencial, preguntándonos *¿Qué no está haciendo?* Y ¿en qué etapa se está deteniendo?

En primer lugar, es crucial definir el estado inicial de la planta, estableciéndolo como punto de partida y como contexto para el abordaje del mantenimiento. Las fallas deben atenderse siguiendo el flujo de la información a través del sistema. Este método no sólo permite verificar de forma inmediata que el problema se ha resuelto a medida que avanzamos, sino que también ayuda a identificar posibles fallas ocultas a primera vista.

Paralelamente a la resolución de los errores y con el objetivo de documentar exhaustivamente todos los detalles para futuras implementaciones, se elaboró una memoria de mantenimiento con todas las fallas que se presentaron a partir del estado inicial de la planta. La segunda parte de este capítulo recoge una narración cronológica de cómo se fueron solucionando los problemas, complementada con una tabla resumen que especifica el fallo y la solución implementada.

Los retos que se presentaron en esta sección fueron diversos. Las fallas en la planta no eran localizadas y necesitaban ser rastreadas, por lo que el entendimiento inicial del sistema y la aproximación secuencial fueron cruciales para llegar a la solución. Sin embargo, la documentación de proyectos anteriores no es integral, ya que se enfocan específicamente en el módulo que desarrollaron. Esto llevó a que muchas de las pruebas requirieran un proceso de ingeniería inversa combinado con prueba y error para entender el flujo que precisamente tenía que ser la guía para el mantenimiento.

Hablando específicamente de las fallas, se emplearon dos herramientas clave: el Diagrama de Ishikawa y el método de Los 5 Porqués de Toyota. El Diagrama de Ishikawa, o diagrama de causa y efecto, visualiza las causas potenciales de un problema como espinas que se extienden desde una columna vertebral, con la falla en la cabeza [6]. En complemento, Los 5 Porqués profundizan en una causa potencial preguntando *¿por qué?* con relación al problema, hasta llegar a una causa raíz [7]. Juntos, ambos métodos se complementan de forma robusta: mientras el diagrama esquematiza las posibles causas, los 5 Porqués filtran y profundizan cada una de ellas. Dado que este es un proceso repetitivo y las causas potenciales descartadas o restantes se vuelven irrelevantes cuando se encuentra la causa real, se incluirá sólo un ejemplo para demostrar el análisis. En la Ilustración 8 se muestra el diagrama de Ishikawa para el problema número 5 de la tabla resumen al final de la Sección 2.2 y en la Ilustración 9 se encuentra el análisis de porqués.

Finalmente, el estado operativo de la planta se define mínimamente como la capacidad de comunicarse con el cRIO-9014 para poder enviar voltajes a las articulaciones y leer la posición de los *encoders*. Sin embargo, se espera poder ejecutar movimientos o, en su defecto, enviar voltajes directamente que hagan girar las articulaciones - teniendo cuidado de no golpear en sus límites - utilizando algún programa preexistente en la computadora host. Si este movimiento se logra, las articulaciones de ambos brazos no deben bloquearse o disparar los limitadores de corriente. La sección final de este capítulo describe el estado de la planta posterior al mantenimiento y evalúa la operatividad con estos criterios.

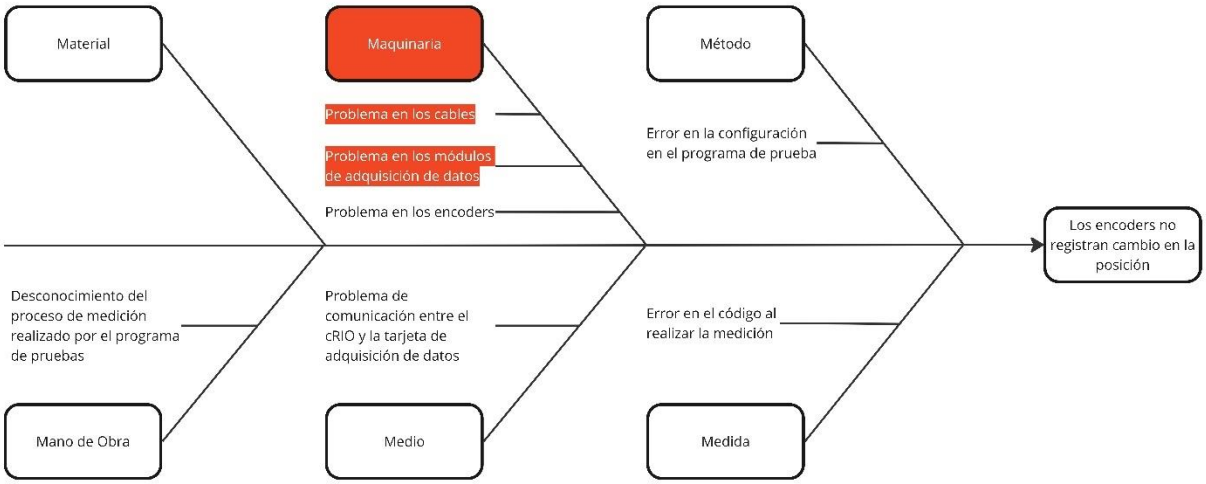

Ilustración 8: Diagrama de causa-efecto del problema número 5, en naranja se resalta la causa del problema encontrada al final del análisis

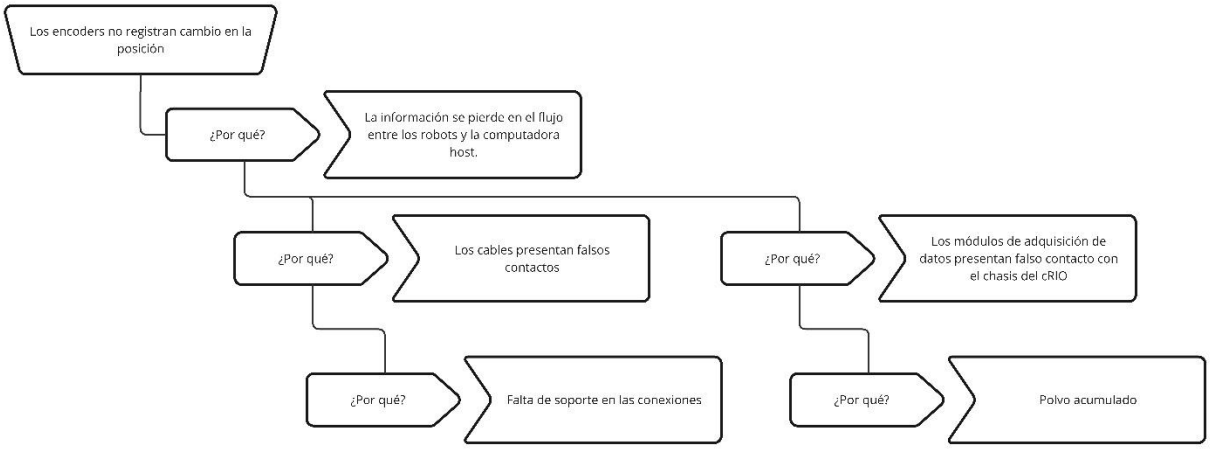

Ilustración 9: Método de los 5 porqués para el diagrama en la ilustración 8

2.1 Estado inicial de la planta

Debido a la suspensión del aseo, el laboratorio se encontraba cubierto de polvo, esto incluye a la planta, su gabinete y el escritorio con la computadora y los controladores C500. En el gabinete que contiene la placa de adquisición de datos y el cRIO, todos los cables que comunican ésta con la computadora y los C500 se encontraban desordenados pero conectados. La estructura que soporta a los robots, tenía una superficie en diagonal sujeta a perfiles que posiblemente sirvió para un experimento previo y en la parte debajo de los brazos, se colocaron diversos objetos del laboratorio. Adicionalmente, se cuenta con dos UPS, sin embargo, los controles C500 se encontraban conectados directamente a la alimentación del laboratorio.

Tras una limpieza general del área de trabajo, tratando de manipular en lo mínimo las conexiones, se descubrió un problema con la computadora host: ésta se reiniciaba unos segundos después de entrar al sistema operativo. La fuente que alimenta la placa de adquisición de datos y el cRIO, al igual que los controladores C500, encendía con normalidad, sin embargo, aun reemplazando la computadora, en este

punto era imposible comprobar el estado de los robots por seguridad, ya que no se tenía certeza sobre la programación de la planta.

Ilustración 10: Estado inicial de la planta posterior a la limpieza y con el gabinete desmontado

Por otro lado, en la parte del frente, mostrada en la Ilustración 11, los controladores C500 cuentan con un panel de protección que contiene fusibles para la alimentación de forma global e interruptores limitadores de corriente para cada articulación. Si uno de estos últimos se llegase a disparar, el robot entero quedará bloqueado como si se hubiera utilizado el paro de emergencia, hasta que dicho interruptor vuelva a ser presionado.

Ilustración 11: Panel de fusibles e interruptores de los controladores C500, el conjunto de seis cuadrados de color negro son los interruptores de corriente, mientras que a su derecha se observan tres tapas circulares donde se insertan los fusibles.

Generalizando, podemos afirmar que el estado inicial de la planta era inoperable. En un sentido eléctrico, no parecía haber fallas, ya que todo el equipo encendía con normalidad, pero al quitar el paro de emergencia en los robots, la acción gravitatoria comenzaba a tirar de ellos hacia abajo, por lo que fue fácil deducir que, aunque el control cRIO se encargara de la gestión de entradas y salidas, las instrucciones

provenían de la computadora. Esto dio una idea del programa cargado en el módulo de control, pero limitaba la posibilidad de una prueba inicial al mero encendido de los componentes. Hasta este punto se podía decir, de manera superficial, que todo servía con normalidad.

A partir de aquí, comienzan una serie de acciones de mantenimiento y pruebas de funcionamiento con las que se abordó una situación a la vez. En primer lugar, se necesitaba acceder a la computadora de la planta, luego conseguir un programa para probar el movimiento de los robots; si dicho movimiento fuese correcto, el abordaje sería el estudio del código utilizado para la implementación del marco de trabajo, en caso contrario – como sucedió – el proceder sería verificar la comunicación con el controlador y luego verificar el paso de las señales a través de los diferentes componentes del sistema, hasta su llegada a los robots.

2.2 Memoria de fallas y correcciones

Ya se mencionó, al describir el estado inicial de la planta, que se encontraba totalmente cubierta de polvo, esto dificultaba trabajar y podía ser potencialmente dañino para la tarjeta de adquisición de datos y el controlador – que se encuentran sin cobertura en el gabinete – al igual que para los robots. Limpiar el área de trabajo de la computadora y la parte externa de los robots fue un buen inicio. Con la intención de comprobar el estado inicial de la planta, era importante no interferir en las conexiones al limpiar, por lo que para los elementos dentro del gabinete se realizó una limpieza superficial con aire comprimido. También se verificó que las conexiones fuesen correctas de acuerdo a los diagramas de conexiones [5]: de ambos robots a los controladores C500, de los controladores a la tarjeta de adquisición de datos, de la tarjeta de adquisición al cRIO y los módulos de entrada/salida y, por último, del cRIO a la computadora.

Después de verificar que cada conexión estaba correctamente instalada, se encendió la computadora host y se presentó la primera falla descrita en el estado inicial de la planta: cerca de treinta segundos después de entrar al sistema operativo, la computadora se apagaba sin previo aviso, como si la alimentación hubiera sido retirada y vuelta a conectar de inmediato, entrando en un bucle de reinicio. Este comportamiento sugiere – considerando el que la computadora no se había utilizado en casi tres años - una falla por sobrecalentamiento del procesador. La solución fue relativamente simple: dar mantenimiento a la computadora para remover el polvo acumulado en su interior y el cambio de la pasta térmica entre el disipador de calor y el procesador. Se utilizó pasta de la marca ARTIC MX-4 de alta conductividad térmica y tras el reensamblaje del equipo, el problema quedó resuelto satisfactoriamente.

La computadora, no presentaba contraseña de acceso y no contaba con conexión a internet habilitada, condiciones que no afectaban el uso previsto en la planta. Se inició una exploración en los archivos de desarrollos anteriores almacenados en busca de un software que permitiera hacer una prueba operativa. Esta tarea se complicó por la ausencia de un sistema general de organización y control de versiones, lo cual acumuló desorden a lo largo de los años, además de que era necesario inferir, a partir de los nombres de archivos y carpetas, el propósito del software y las intenciones de su autor. La búsqueda no debía limitarse a ubicar un programa y entender qué hace, sino que se debía rastrear a nivel de código el cómo hacía su propósito, ya que era riesgoso ejecutar a ciegas cualquier cosa debido al desconocimiento del estado del código y de las fallas en los robots. Por ejemplo, una ley de control no ajustada o parcialmente implementada, podría derivar en una señal demasiado alta que hiciera soltar un golpe al aire, hecho potencialmente riesgoso para el robot en sí mismo. De igual manera, algunos programas requerían configuración inicial - como el caso de los que utilizan el sensor de fuerza - para ejecutarse, lo que los descartaba al agregar complejidad innecesaria a la prueba inicial.

Después de una búsqueda exhaustiva, se encontró el código *Cooperativo* entre los archivos de la computadora host. Este programa fue crucial en el desarrollo de esta tesis, permitiendo a través de ingeniería inversa, deducir el mecanismo de operación de la planta que se detallará en el próximo capítulo. A partir de aquí, denominaremos a este programa como *software de referencia.* El programa consiste en una interfaz gráfica que, al ejecutarse establece la comunicación con uno o ambos robots. El problema número 3 se presentó aquí ya que la planta debe estar encendida previamente a la ejecución del software o el sistema arrojará un error y no permitirá realizar una conexión posterior, siendo la solución cerrar y volver a ejecutar. Una vez establecida la comunicación con los robots, se mostrará una ventana con cuatro paneles:

- Panel del robot A255
- Panel del robot A465
- Panel cooperativo
- Panel para el sensor de fuerza

En la Ilustración 12 puede observarse la interfaz descrita. Los paneles de los robots son idénticos, tienen un botón para leer los *encoders* con los campos de texto para mostrarlos; un botón para enviar voltajes con los campos de texto para introducir los valores; un botón para establecer la posición actual y otro para ir a la posición de home. Se incluyen un par de elementos más, pero de momento se centrará la atención en los controles mencionados ya que incluyen la funcionalidad básica para el fin actual. El movimiento de los robots se realiza a través de un control PID ejecutado por el *callback* de un *timer* de la biblioteca Mmsystem.

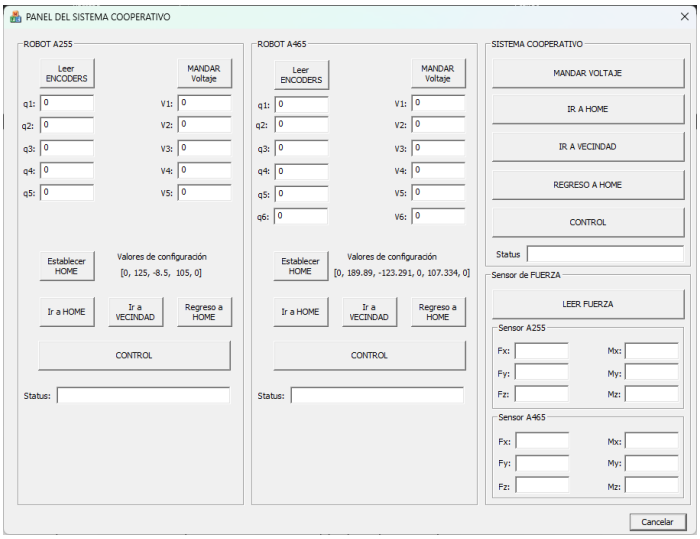

Ilustración 12: Interfaz gráfica del software de referencia

Al ejecutar el software se presentó la cuarta falla, ya que a pesar de que las conexiones se revisaron en un inicio y la planta se encontraba encendida, el programa mostró el error de conexión a ambos robots. En esencia, lo que distingue la conexión de un robot al otro son los pines asignados a cada conector de la tarjeta de adquisición, por lo que un error de este tipo a ambos robots se trata en realidad de un error de conexión al propio controlador. El cRIO salió a la venta a principios de los años dos mil, debido a esta antigüedad, National Instruments lo descontinuó y su software sólo tiene soporte para él hasta la versión de 2018; este detalle es relevante ya que esa es la razón por la que encontrar información del fallo - o

incluso de la propia configuración del controlador - fue complicado. Después de realizar las pruebas citadas en la entrada *NI CompactRIO o el dispositivo de red no aparece o falta en Max* del blog *knowledge.ni.com* de National Instruments [8] y que no funcionara, una segunda entrada *Cómo arreglar o restablecer el archivo de la base de datos de NI MAX* [9] sugirió la corrupción del archivo que almacena la configuración del dispositivo cRIO, aunque sus instrucciones tampoco surtieron efecto; esto llevó, por sugerencia de la misma publicación, a tratar de restaurar manualmente dicho archivo de base de datos con la entrada *Manually Resetting the NI Measurement & Automation Explorer Database* [10] que indicaba remover manualmente los archivos pertenecientes a la configuración de controladores para que el sistema; al no encontrarlo, permita agregar de nuevo el cRIO al gestor de dispositivos MAX y volver a crearlo de forma limpia para que la conexión funcione. Esta última solución surtió efecto y, entrando a terreno especulativo, la causa del fallo podría deberse a que los sistemas operativos antiguos, como Windows XP, no tenían las protecciones actuales para el sistema de archivos, por lo que, si estos se estaban utilizando y no se cerraron correctamente, podían corromperse fácilmente. El monitor de dispositivos de NI MAX, se ejecuta con el arranque de la computadora y ésta se estaba apagando por el problema de sobrecalentamiento, puede que justo ahí se produjera el daño a los archivos al cerrarse abruptamente el sistema operativo tras haber arrancado. Evidentemente es una idea especulativa, pero conocer la causa del error a bajo nivel no es relevante una vez sabiendo cómo corregirla y prevenirla, además de que se eliminaría su posibilidad de ocurrencia al desacoplar la computadora host para trabajar con sistemas operativos más modernos.

Teniendo solucionada la comunicación, se comenzó con las primeras pruebas con la interfaz del software de referencia. Por seguridad, el primer paso sería comprobar la correcta lectura de los *encoders*, ya que se puede realizar con el paro de emergencia activado y así garantizar que la ejecución del control PID durante los movimientos se encontrará en lazo cerrado. Fue una buena elección porque se suscitó justo el quinto error: las lecturas eran valores aleatorios y si los brazos se movían manualmente, no registraban cambios en la posición. Rastrear esta falla fue complicado porque el flujo de información de los *encoders* va desde los robots hasta la computadora host y se debía revisar todo el trayecto. La primera prueba consistió en verificar la continuidad de los cables desde un extremo del sistema hasta el otro, comenzando por el cable que va de cada robot a los C500, luego los cables que salen del C500 a la tarjeta de adquisición de datos y por último de la tarjeta a los módulos NI. No se detectaron fallas – aunque más adelante se encontrarían problemas de falso contacto - pero se llegó a la conclusión de que el desorden en los cables sería un potencial problema si se dejaban igual de desordenados, por lo que se utilizó la impresora 3D del laboratorio para crear un soporte que los mantuviera en una posición fija.

Este problema pudo ser considerablemente profundo, ya que podía implicar un fallo en las tarjetas adicionadas al sistema, lo que implicaba un análisis exhaustivo de ambos circuitos. Afortunadamente la solución no llegó a tales implicaciones: los módulos NI-9401, por su antigüedad, presentan un fallo de conexión con el gabinete, por lo que basta con retirarlos – con el sistema apagado, por seguridad – y volverlos a conectar. Esta situación se presentó dos veces más en el mantenimiento y se solucionó definitivamente al limpiar los conectores del gabinete y los de los módulos con alcohol isopropílico.

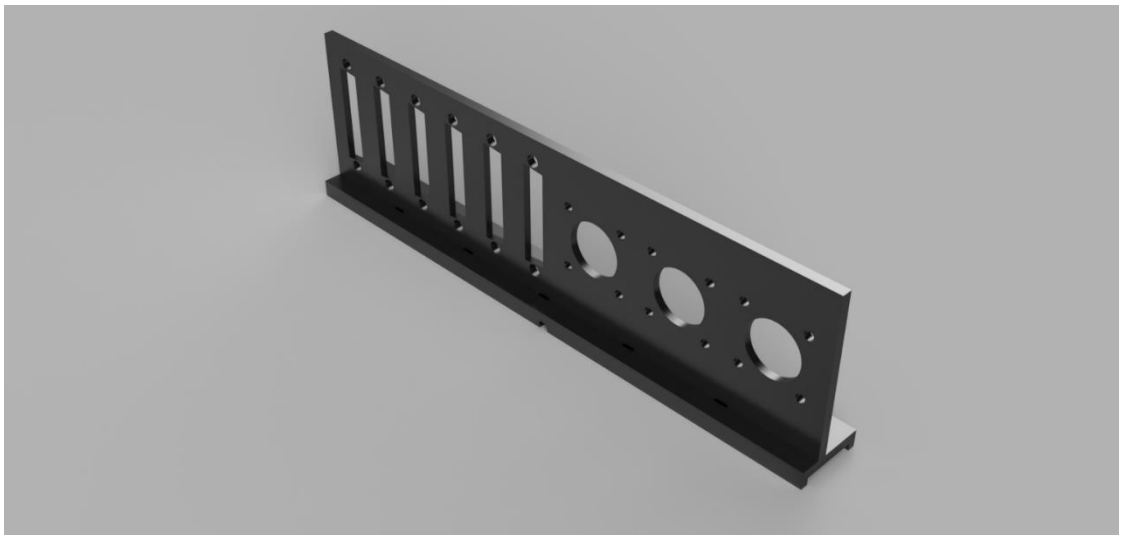

Ilustración 13: Soporte impreso en 3D para los cables en el gabinete de la planta

Las primeras pruebas de movimiento resultaron parcialmente bien. Se envió un *set-point* desde la interfaz y algunas articulaciones lo alcanzaron y otras se quedaron quietas. Sin embargo, después de un rato, las articulaciones que no se habían movido comenzaban a hacerlo de forma repentina – seguramente acumulando error integral en el controlador PID – y hubo que activar rápidamente el paro de emergencia para evitar una situación de riesgo. La solución estaba clara, la señal de control era muy pequeña para vencer la fricción estática hasta que se acumulara suficiente energía potencial que rompiese la inercia, por lo que modificar las cotas de voltaje a un valor que facilitara el movimiento desde un inicio, además de ajustar las constantes del controlador, podría compensar la señal de control para esta situación. El problema fue que no había forma de modificar dichos valores desde la interfaz, por lo que fue necesario ingresarlos directamente en código; esto se convirtió en dos requerimientos funcionales para la nueva interfaz de usuario que se desarrollaría. El ajuste surtió efecto, el movimiento en las articulaciones no era el ideal, pero sí completo y esto define el error 6 y los requerimientos 7 y 8 en la tabla resumen.

Haciendo diferentes pruebas de movimiento, éste llegaba a ser incorrecto en posición, pero no en alcance, por lo que se dedujo que el punto de referencia estaba mal. El problema venía de la interfaz que reiniciaba dicho punto entre un experimento y otro; esto hacía que, si los robots se quedaban en una posición distinta de home al apagar la planta, la siguiente vez que se encendiera esa fuese su nueva referencia. La solución se encontró, momentáneamente, en devolver siempre los robots a su posición original y calibrar la posición de home al iniciar cada prueba. Esto llevó al requerimiento funcional de controles manuales que permitan ajustar la referencia cada vez que sea necesario.

Con el paso de los experimentos moviendo los robots a diferentes posiciones, se entendió el mecanismo detrás del cuál el control PID realizaba el movimiento: al establecer un punto de referencia, la interfaz lo convertía en el punto actual e intentaba volver a home a partir de ello, por lo que la posición final de las articulaciones siempre sería cero. Este comportamiento no es deseable porque no se podría tener un registro correcto de la ubicación espacial del robot, por ello se corregiría en los requerimientos funcionales. Aunado a esto, se encontró una falla en la articulación 3 del A255, este manipulador nunca alcanzaba completamente la posición indicada y, si se llegaba a quedar en una posición que le requiriera esfuerzo considerable por mucho tiempo, disparaba el interruptor limitador de corriente. Esto llevó al

desmontaje del robot para limpiar su mecanismo. El desarme es bastante intuitivo, sin embargo, es importante mencionar la complejidad de hacerlo en solitario debido a que hay que sujetar la estructura una vez se retiren los motores.

Después de la limpieza interior, los robots funcionaban perfectamente acorde al software de referencia. El siguiente paso natural era tratar de desacoplar la planta de la computadora host. El entorno de National Instruments es vasto, con diversos componentes y se necesita una licencia para usar la mayoría. Por suerte, el único componente obligatorio para utilizar el cRIO – debido a que ya estaba programado – es el driver NI CompactRIO versión 18.0 que puede encontrarse en la página oficial de National Instruments [11] y es de libre descarga. Esta instalación incluye el software NI MAX que es el gestor desde el cual se tiene que dar de alta el controlador cRIO-9014.

La instalación del driver es el único requisito obligatorio porque el cRIO ya cuenta con un programa cargado y sólo necesitamos comunicarnos con él, de otro modo, necesitaríamos una instalación de LabVIEW con su licencia correspondiente para realizar la programación en el controlador. El programa base del cRIO es uno de los componentes añadidos de investigaciones anteriores, la explicación completa y diagramas de su contenido puede encontrarse en la tesis *Desarrollo y Construcción de una Tarjeta para la Adquisición de Datos y Control de Robots Cooperativos* [5] pero, a modo de resumen, crea un lazo de repetición que ejecuta rutinas para la identificación del sentido de giro de los motores, para actualizar la cuenta de la posición en memoria y para escribir valores en las salidas analógicas, todo esto cuando sea requerido por el usuario desde la computadora y con una frecuencia de actualización de $5[\mu s]$. Estas funciones quedan expuestas gracias a los archivos:

```
APICRIO.h
NiFpga.cpp
NiFpga.dll
NiFpga.h
NiFpga_final.h
NiFpga_final.lvbtix
stdafx.h
targetver.h
```

los cuales, a partir del análisis de ingeniería inversa sobre el software de referencia, se infirió que deben ser incluidos en cualquier nuevo proyecto. Se destaca entre los elementos el archivo con extensión `lvbtix` ya que su ubicación absoluta debe pasarse a la función `NiFpga_MergeStatus` en el desarrollo para poder establecer comunicación con el cRIO [5].

El último detalle a mencionar apareció de forma arbitraria. Tras una fluctuación en la red eléctrica, el resto del equipo y componentes de la planta trabajaban con normalidad, pero los controladores C500 simplemente ya no encendían. Algunas fuentes de poder cuentan con una protección ante dichas fluctuaciones bloqueándose para evitar que nuevos picos vayan a dañar el equipo, por lo que basta con desconectar los controladores de la alimentación entre 1 y 2 minutos, para que vuelvan a estar en operación. Esto remarca la necesidad de que todos los equipos de la planta permanezcan conectados a las unidades UPS y no directamente a la alimentación del laboratorio.

A modo de resumen, la Tabla 1 incluye el listado de fallas que se encontraron, las causas y la solución utilizada, como referencia en caso de que problemas similares pudieran presentarse en el futuro. Cabe mencionar que dicha tabla es una guía y un punto de partida, ya que síntomas similares no son

determinantes para que se trate del mismo problema. La implementación de métodos heurísticos tendrá la última palabra, pero la referencia aquí presentada puede ser un punto de partida.

Id	Falla/Requerimiento	Causa	Solución
1	La computadora se reinicia después de unos segundos de entrar al sistema.	Sobrecalentamiento del procesador.	- Limpieza y cambio de pasta térmica
2	Falta de un software para probar la planta en un primer acercamiento.	El controlador sólo ejecuta las instrucciones que le son dadas por la computadora.	- Utilizar y documentar un desarrollo prexistente. - Crear software de pruebas con las adecuaciones necesarias.
3	Error al iniciar comunicación con el controlador cRIO.	La planta debe estar encendida antes de ejecutar el software de referencia.	- Encender la planta antes de ejecutar el software de referencia. - Implementar la conexión en un botón para el nuevo software.
4	Error al iniciar la comunicación con el controlador cRIO a pesar de estar encendida la planta.	Daño en el archivo de configuración del dispositivo por un incorrecto apagado de la computadora.	- Borrar el controlador en NI MAX y volverlo a agregar, cuidando no modificar la IP.
5	Los *encoders* no registran cambio en la posición.	Las lecturas no están llegando a la computadora Host.	- Verificar continuidad en todos los cables. - Crear un soporte que evite desconexiones involuntarias en los cables. - Verificar la conexión de los módulos de adquisición de datos en el chasis.
6	Algunas articulaciones no responden.	La señal de control es muy pequeña al generar el torque necesario para para romper la inercia de la articulación.	- Aumentar la cota máxima y mínima de voltaje en la configuración del software. - Modificar las constantes del controlador PID.
7	Modificar las cotas de voltaje para las articulaciones.	El software de referencia no cuenta con un campo en la interfaz gráfica para introducir el máximo y mínimo voltaje que se podrá enviar a cada articulación.	- Introducir los voltajes directamente en los métodos ActivarArticulacionesA465 y ActivarArticulacionesA255 del software de referencia. - Crear una vista en la interfaz gráfica del nuevo software que permita configurar estos valores.

8	Modificar las constantes de la ley de control.	El software de referencia no cuenta con un campo en la interfaz gráfica para introducir valores para las constantes de la ley de control.	- Introducir los voltajes directamente en los métodos IR_HOMEA465 e IR_HOMEA255 del software de referencia. - Crear una vista en la interfaz gráfica del nuevo software que permita configurar estos valores.
9	El robot se mueve de forma brusca o hacia un punto distinto al deseado.	El punto de home no está calibrado o se modificó al no registrar ciertos movimientos.	- Establecer home antes de iniciar cualquier movimiento. - Implementar controles manuales para calibrar la posición en el nuevo software.
10	El movimiento de los robots siempre va hacia el origen.	El software de referencia trabaja con el error de posición en lugar de establecer un *set-point* en el espacio articular.	- Modificar la forma de alcanzar el *set-point* en el nuevo software. - Ajustar la sintonización de las constantes de la ley de control.
11	Disparo de los interruptores limitadores de corriente tras finalizar un movimiento.	La articulación se encuentra atascada.	Lubricar las articulaciones y los mecanismos de transmisión de movimiento.
12	Cambiar la computadora host.	La computadora host funciona con un sistema operativo considerablemente antiguo.	- Instalar el driver NI CompactRIO versión 18.0 de la página oficial [11]. - Incluir en el nuevo desarrollo los archivos necesarios para la comunicación con el cRIO.
13	Los controladores C500 no encienden.	La protección eléctrica interna bloquea el suministro por protección cuando existen fluctuaciones en la red.	Desconectar los controladores de la alimentación por un minuto.

Tabla 1: Resumen de fallas/requerimientos y su solución

2.2 Estado operativo de la planta

Al llegar a este punto la planta cooperativa ha sido restaurada y puede ser utilizada de nuevo. La computadora host ahora puede comunicarse con el cRIO sin problemas, permitiendo el uso del software de referencia para enviar movimientos a cada robot de forma individual y simultánea. También podemos configurar una segunda computadora para ejecutar el proyecto desde ahí y crear un entorno de trabajo más específico, por lo que modularizamos esa parte de la planta.

Se llevó a cabo la limpieza general del sistema y para cada robot, se limpiaron y lubricaron las articulaciones, además de los mecanismos de transferencia de movimiento. También se volvió a montar el gabinete, la tarjeta de adquisición de datos y el cRIO con los cables organizados para no interferir en las conexiones y el área de trabajo alrededor de la planta fue reorganizada. La Ilustración 14 muestra el estado final de la planta mientras que la 15 muestra de cerca el gabinete montado nuevamente.

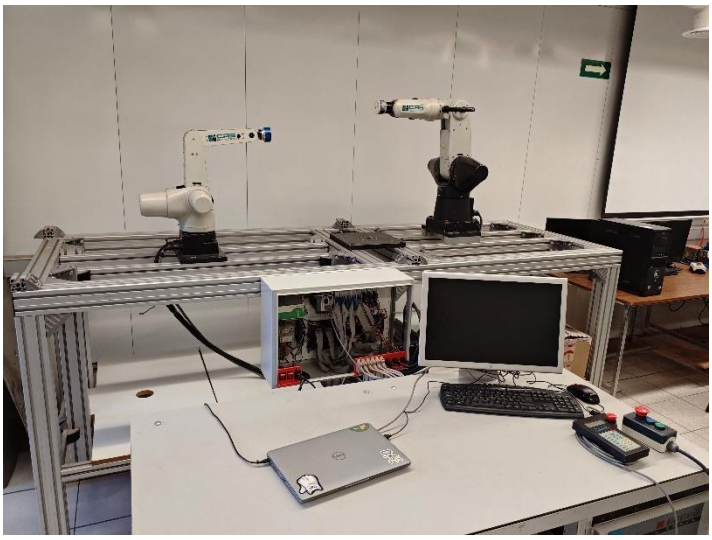

Ilustración 14: Estado final de la planta cooperativa

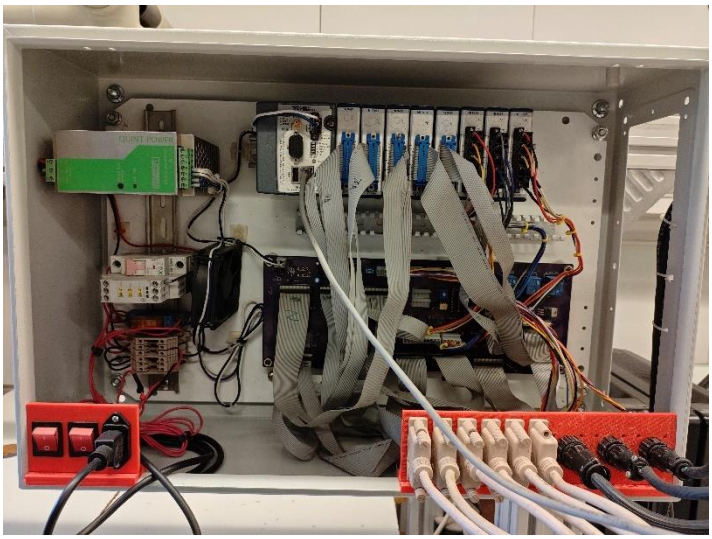

Ilustración 15: Montaje final del gabinete con los soportes para los cables de conexión

Curiosamente, al revisar la memoria de mantenimiento, las fallas parecen ahora menores y fácilmente solucionables, sin embargo, el esfuerzo está detrás del análisis para concluir que esos puntos son los que están deteniendo la operación. La intención de esta tesis es justo que no se tenga que volver a hacer la misma cantidad de esfuerzo para solventarlos.

Por último, respecto al software de referencia, éste nos dio la pauta para probar el funcionamiento general del sistema y los archivos necesarios para comunicarnos directamente con el cRIO sin necesidad de reprogramarlo, sin embargo, hay algunas funcionalidades que no tiene y son necesarias para validar una implementación completa de la planta. Estas características pasarán a convertirse en implementaciones en la interfaz humano-máquina del Capítulo 4, pero primero serán definidos como requerimientos en el siguiente capítulo.

3 Requerimientos funcionales

Una vez descrito *el qué* se hizo a la planta al definir su estado operativo actual, explicaré *el cómo*. Ya se han puesto sobre la mesa las bases teóricas alrededor del segundo objetivo: se busca mantenernos sobre el camino de la arquitectura abierta y crear un marco de trabajo que estandarice los desarrollos. Ambos puntos, para esta etapa del proyecto, recaen en el software. Claro que aún podríamos seguir iterando sobre el hardware de la planta, pero las características necesarias para un mínimo producto viable ya las tenemos: inhibición del control del fabricante, adquisición de señales y un controlador reprogramable. Sin embargo, este último sigue siendo un componente de arquitectura cerrada que, aunque nos expone cierta versatilidad al programar, nos limita a trabajar bajo las reglas impuestas por National Instruments. Tal es el caso que el controlador cRIO-9014 fue descontinuado y su soporte se retiró de LabVIEW en versiones posteriores al año 2018 lo que determina que, si queremos programarlo, tenemos que usar una versión antigua del software. Esto conlleva el no poder utilizar los nuevos componentes que NI desarrolle, dificultando así el interconectar la planta con tecnologías modernas.

Este capítulo se centrará en definir una arquitectura de software para la planta y, a la par, establecer el marco de trabajo que buscamos. El primer punto es definir las necesidades de un sistema robótico que son comunes para cualquier implementación de la planta, estos requerimientos funcionales delimitarán la operación y nos permitirán trazar un camino a seguir.

Posteriormente debemos definir una arquitectura que nos permita cumplir con los requerimientos antes mencionados, para ello nos centraremos en los conceptos de modularidad – dividiendo la operación en componentes desacoplables – y flexibilidad – seleccionando tecnologías que nos permitan adaptarnos a diferentes contextos de implementación.

Por último, se describirán en detalle los componentes presentados en la arquitectura a implementar.

3.1 Requerimientos de un sistema robótico

En el primer capítulo se dijo que la investigación robótica es un campo muy amplio y los proyectos que se pueden proponer ya son inconmensurables simplemente bajo iteraciones de una misma idea. Si nos ceñimos a la planta cooperativa, aún tenemos un grupo muy grande y diverso de posibilidades, pero, aunque estás pueden parecer dispares, existe un conjunto de requerimientos que serán afines entre ellas.

Evidentemente, los elementos que siempre van a existir sin importar el proyecto son la planta, quien es el objeto de estudio, y el usuario, que designa las tareas a realizar y los escenarios que brindan información para la investigación. Ambos elementos necesitarán comunicarse entre sí y esto exige funcionalidades según hacia dónde se esté moviendo la información. Siguiendo la idea de capas de abstracción, podemos clasificar dichos requerimientos en dos grupos según su cercanía al usuario o a la planta: requerimientos de bajo y alto nivel [2].

Los **requerimientos de alto** nivel son más fáciles de definir porque podemos ponernos en el papel del usuario y determinar qué necesitamos para trabajar con la planta. En esencia, precisamos una interfaz donde podamos dar tareas al robot definidas en nuestro lenguaje, ver información de su movimiento y herramientas de configuración para establecer parámetros de operación.

Por otro lado, los **requerimientos de bajo nivel** se centran en cómo la planta va a realizar las tareas que le solicitemos. Para ello, es necesario comunicarnos con el controlador cRIO, distinguir entre ambos robots para poder controlarlos de forma individual, medios para controlar al robot y para obtener información

qué se pueda devolver al usuario. Estas condiciones son muy generales entre implementaciones y, en caso de diferir, se puede seguir la idea como guía de desarrollo.

3.2 Descripción de la planta a nivel de software

El flujo de la información entre el usuario y la planta debe ser bidireccional, cambiando progresivamente de nivel en las capas de abstracción y relegando cada componente a realizar una sola función. Los extremos del flujo serán la interfaz de comunicación, que permitirá enviar acciones directas al cRIO, y la interfaz humano-máquina, donde el usuario podrá interactuar con la planta en su conjunto.

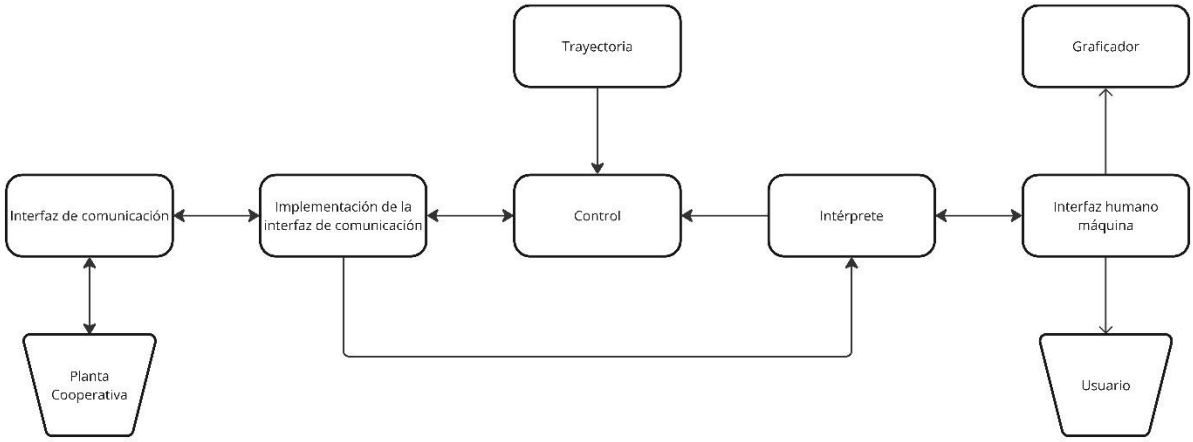

Ilustración 16: Diagrama de arquitectura de software para la planta cooperativa

En la Ilustración 16 se puede observar la arquitectura propuesta y las interacciones entre cada componente. Este diagrama no sólo sirve como esqueleto para este trabajo, sino que se convierte en un marco de trabajo que se puede seguir en implementaciones distintas o se puede modificar a conveniencia debido a la modularidad con la que se va a construir. La descripción de sus componentes es la siguiente:

- Interfaz de comunicación: se encargará de abstraer las operaciones básicas que el cRIO puede ejecutar sobre cada robot. Este elemento es la parte más importante de la integración porque encapsula el desarrollo a bajo nivel que se construiría en cualquier proyecto, no importa qué tan distinto de este fuese, por lo que es el módulo obligado a utilizar en cualquier implementación.
- Implementación de la interfaz de comunicación: parte del objetivo es hacer independiente el desarrollo a la tecnología implementada, por lo que este componente sería la representación de la interfaz de comunicación en el lenguaje de programación seleccionado.
- Control: una vez se tenga comunicación directa con los robots, podrá implementarse un método para controlarlos. El artefacto de control se encargaría de encapsular la lógica común para cualquier método de control que se utilice, permitiendo expandirle con nuevas leyes de control a definir.
- Trayectoria: este elemento es utilizado por el control para seguir un movimiento específico, también está diseñado para poder extenderse con diferentes formas de movimiento.
- Intérprete: es el punto de inflexión entre la abstracción de bajo nivel y la de alto nivel, ya que se encargará de traducir la información entre un lenguaje definido para el usuario y las instrucciones de la planta para realizar tareas determinadas.

- Interfaz humano-máquina: expondrá una serie de controles e información gráfica para que el usuario pueda enviar instrucciones a la planta y consumir de forma cómoda la información que venga de ella.
- Graficador: es utilizado por la interfaz gráfica para desplegar información de los movimientos de los robots. Se define como un componente a parte debido a que puede ser redefinido cambiando los métodos de graficación o las gráficas en sí mismas.

3.3 Descripción de los robots A255 y A465

A nivel de software, los robots A255 y A465 son muy similares, por lo que una definición individual nos llevaría a escribir dos veces el mismo código. La solución más simple es distinguir entre ambos robots a través de una constante numérica, escribiendo el código general separando solamente para ejecutar código particular a uno de los robots. Sin embargo, la mayor distinción que encontraremos es que el A465 cuenta con una articulación más que el A255, por lo que sólo en caso de que se trate del primero, ejecutaremos las instrucciones referentes a la sexta articulación. Los atributos necesarios para abstraer ambos robots bajo un mismo modelo son:

- Una variable entera que distinguirá entre un robot u otro, 0 para el A255 y 1 para el A465
- El factor de conversión digital-analógico para el módulo NI-9263: 999.95327214209
- Un arreglo para almacenar las lecturas de los *encoders* en radianes
- Factores de conversión de cuentas del módulo de control a grados
- Factores de conversión de cuentas del módulo de control a radianes
- Límites de posición de las articulaciones

Respecto a las cotas de voltaje, los valores utilizados pueden encontrarse en la Tabla 2 y con ellos se configurará la ley de control de forma segura, restringiendo la velocidad máxima a la que podrá operar el robot. Además, en Tabla 3 se encuentran los factores de conversión de las cuentas por cada pulso del encoder a radianes y grados, mientras que en la Tabla 4 se encuentran los límites para cada articulación.

Articulación	**A255**		**A465**	
	$\boldsymbol{V_{min}}[V]$	$\boldsymbol{V_{máx}}[V]$	$\boldsymbol{V_{min}}[V]$	$\boldsymbol{V_{máx}}[V]$
q_1	-3	3	-1	1
q_2	-3	3	-3	3
q_3	-5	5	-5	5
q_4	-5	5	-5	5
q_5	-5	5	-5	5
q_6	-	-	-5	5

Tabla 2: Configuración de cotas de voltaje para la operación

Articulación	A255		A465	
	[grados/ticks]	[radianes/ticks]	[grados/ticks]	[radianes/ticks]
q_1	1.250×10^{-3}	2.182×10^{-5}	9×10^{-4}	1.571×10^{-5}
q_2	1.250×10^{-3}	2.182×10^{-5}	9×10^{-4}	1.571×10^{-5}
q_3	1.250×10^{-3}	2.182×10^{-5}	9×10^{-4}	1.571×10^{-5}
q_4	-5.625×10^{-3}	-9.817×10^{-4}	1.782×10^{-3}	3.110×10^{-5}
q_5	1.430×10^{-2}	2.496×10^{-4}	1.800×10^{-3}	3.142×10^{-5}
q_6	-	-	1.800×10^{-3}	3.142×10^{-5}

Tabla 3: Factores de conversión entre las lecturas del controlador a radianes y grados

Articulación	A255		A465	
	$\boldsymbol{\theta}_{\boldsymbol{min}}[°]$	$\boldsymbol{\theta}_{\boldsymbol{máx}}[°]$	$\boldsymbol{\theta}_{\boldsymbol{min}}[°]$	$\boldsymbol{\theta}_{\boldsymbol{máx}}[°]$
q_1	-175	175	-175	175
q_2	0	110	-90	90
q_3	-125	0	-110	110
q_4	-110	110	-180	180
q_5	-180	180	-105	105
q_6	-	-	-180	180

Tabla 4: Límites de posición para cada articulación (Arteaga, 2021)

El comportamiento esperado debe satisfacer las operaciones básicas para realizar con los robots, sin contemplar el control de los mismos:

- Conectarse al controlador cRIO
- Activar las articulaciones a utilizar
- Configurar las cotas mínimas de voltaje de salida
- Configurar las cotas máximas de voltaje de salida
- Leer la posición de los *encoders*
- Escribir voltajes de salida para los motores
- Establecer el punto de referencia para el conteo de los *encoders*
- Ejecutar la cinemática directa
- Ejecutar la cinemática inversa

En lo referente a las cinemáticas, la funcionalidad esperada es sólo de cálculo y se incluye aquí ya que es parte de la definición de los robots. Simplemente se espera que reciban un conjunto de coordenadas en un espacio – coordenado o articular - y devuelvan su correspondiente valor en el otro, sin generar ningún tipo de movimiento.

El objetivo de la cinemática directa es obtener las coordenadas cartesianas del efector final de un robot a partir de la posición de sus articulaciones [12]. El método más común para obtener este análisis es el de Denavit-Hartenberg [13], del cual definimos los conjuntos de parámetros para ambos robots en las tablas 5 y 6 y la asignación de ejes coordenados en las figuras 17 y 18.

Articulación	$\boldsymbol{\theta}[°]$	$\boldsymbol{\alpha}[°]$	$\boldsymbol{d}[\boldsymbol{m}]$	$\boldsymbol{a}[\boldsymbol{m}]$
1	q_1	90	0.254	0
2	q_2	-	0	0.254
3	q_3	-	0	0.254
4	$q_4 + 90$	90	0	0
5	q_5	0	0.0508	0

Tabla 5: Parámetros de Denavit-Hartenberg para el robot A255

Articulación	$\boldsymbol{\theta}[°]$	$\boldsymbol{\alpha}[°]$	$\boldsymbol{d}[\boldsymbol{m}]$	$\boldsymbol{a}[\boldsymbol{m}]$
1	q_1	90	0.33	0
2	q_2	0	0	0.305
3	q_3	90	0	0
4	q_4	-90	0.33	0
5	q_5	90	0	0
6	q_6	0	0.076	0

Tabla 6: Parámetros de Denavit-Hartenberg para el robot A465

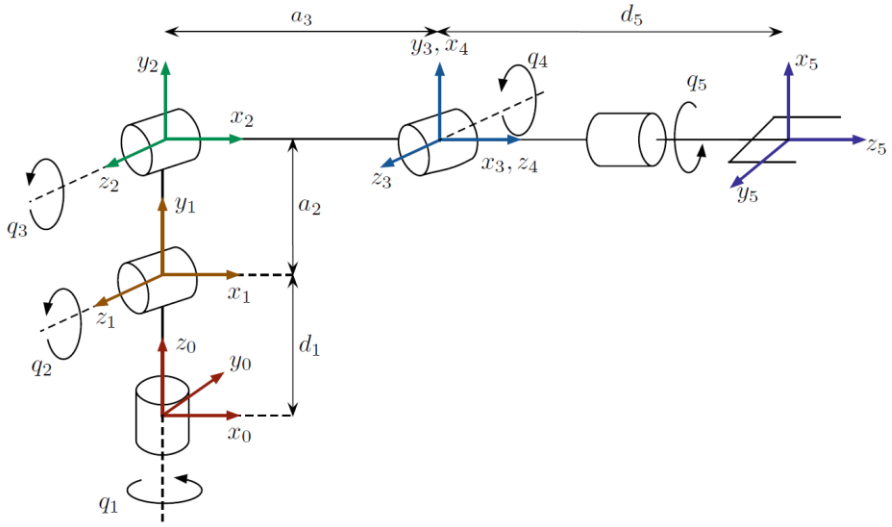

Ilustración 17: Asignación de ejes coordenados según el método de Denavit-Hartenberg para el robot A255 (Arteaga, 2021)

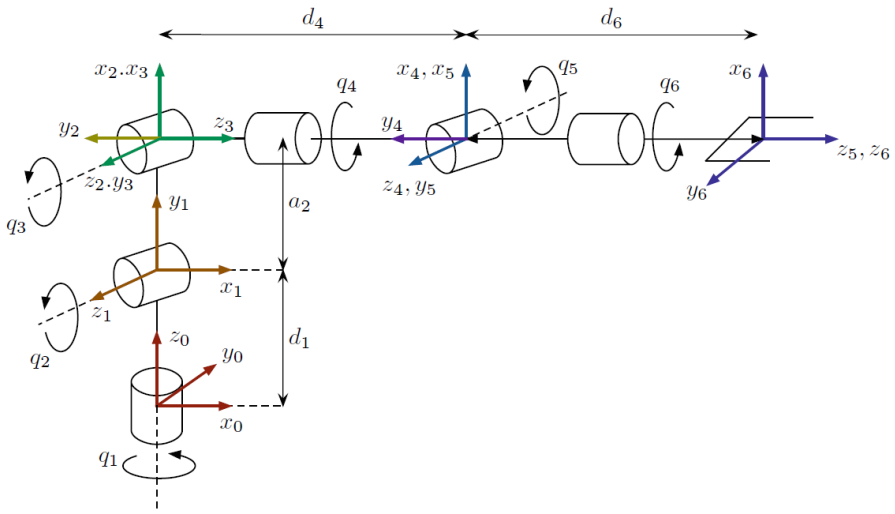

Ilustración 18: Asignación de ejes coordenados según el método de Denavit-Hartenberg para el robot A465 (Arteaga, 2021)

Las expresión del vector de posición para el robot A255 [4] es:

$$p_5^0 = \begin{bmatrix} c_1(a_2c_2 + a_3c_3 + d_5c_4) \\ s_1(a_2c_2 + a_3c_3 + d_5c_4) \\ d_1 + a_2s_2 + a_3s_3 + d_5s_4 \end{bmatrix}$$

Ec. 1

Donde:

- $s_i = \sin q_i$
- $c_i = \cos q_i$

Y las expresión para el robot A465 [4] es:

$$p_6^0 = \begin{bmatrix} a_2c_1c_2 + d_4c_1s_{12} + d_6(c_1(c_{23}c_4s_5 + s_{23}c_5) + s_1s_4s_5) \\ a_2s_1c_2 + d_4s_1s_{23} + d_6(s_1(c_{23}c_4s_5 + s_{23}c_5) - c_1s_4s_5) \\ d_1 + a_2s_2 - d_4c_{23} + d_6(s_{23}c_4s_5 - c_{23}c_5) \end{bmatrix}$$

Ec. 2

Donde:

- $s_i = \sin q_i$
- $c_i = \cos q_i$
- $s_{ij} = \sin(q_i + q_j)$
- $c_{ij} = \cos(q_i + q_j)$

La descripción completa del efector final se logra con el vector de posición y una matriz de orientación. El método de Denavit-Hartenberg nos devuelve ambas, sin embargo, debido a la diferencia en las últimas articulaciones para los robots, omitiremos el problema de la orientación del efector final.

El problema de cinemática inversa busca obtener los valores articulares a partir de un punto en el espacio cartesiano [12]. Utilizando desacople cinemático podemos separar el problema de cinemática inversa en

dos problemas más sencillos de posicionamiento y orientación, pero al igual que en la cinemática directa omitiremos la parte de la solución que resuelve la orientación del efector ya que, dicho problema debe ser resuelto de forma individual para cada robot.

En las ilustraciones 19 y 20 se puede observar un mismo análisis de cinemática inversa para ambos robots, ya que la configuración de sus primeras tres articulaciones es idéntica, derivando en el mismo conjunto de ecuaciones que sólo diferirá por el valor de los parámetros de cada uno, que se pueden obtener en las tablas 5 y 6 de Denavit-Hartenberg. Para el caso del robot A255, simplemente usaremos d_5 en lugar de d_6.

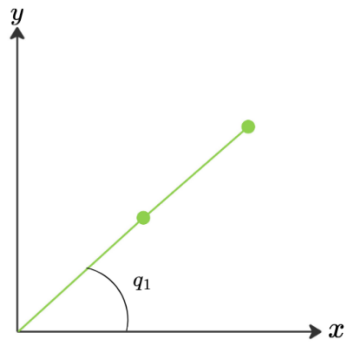

Ilustración 19: Análisis de cinemática inversa, vista superior

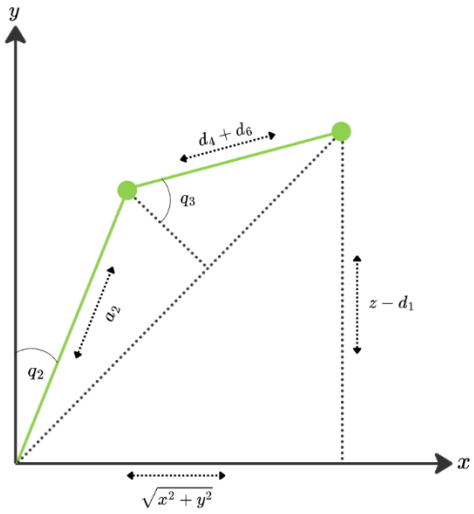

Ilustración 20: Análisis de cinemática inversa, vista lateral

$$q_1 = \tan^{-1}\left(\frac{y}{x}\right)$$

Ec. 3

$$q_2 = 90° - \tan^{-1}\left(\frac{(d_4 + d_6)\sin q_3}{a_2 + d_4 \cos q_3}\right) - \tan^{-1}\left(\frac{z - d_1}{\sqrt{x^2 + y^2}}\right)$$

Ec. 4

$$q_3 = 90° - \cos^{-1}\left(\frac{x^2 + y^2 + (z - d_1)^2 - a_2^2 - (d_4 + d_6)^2}{2a_2(d_4 + d_6)}\right)$$

Ec. 5

3.4 Ley de control y seguimiento de trayectoria

La principal ventaja de utilizar un controlador PID es que no se necesita conocer el modelo dinámico de la planta, lo cual para nuestros fines es útil ya que sólo necesitamos hacer la definición de un componente de control que puede ser expandido en desarrollos posteriores.

En concepto, un controlador PID busca realizar una compensación a la salida de la planta a partir del error entre la señal de referencia buscada y la medición del estado actual [14]. Dicha compensación es la suma de tres acciones de control reguladas por constantes que determinan su peso en el valor de salida:

- Acción proporcional: su nombre proviene de que simplemente es la introducción del error actual como compensación regulada por una constante de proporcionalidad. El valor de su constante determinará qué tan rápido llegará el sistema a la posición deseada, con el inconveniente de que a mayor velocidad se introducirán oscilaciones en el sistema debido a que nunca alcanzará un error en estado estable igual a cero por sí misma.
- Acción integral: es la integración de la señal de error a través del tiempo, relacionada con los valores pasados del controlador. Incrementando su constante estaríamos ralentizando el sistema, pero obtendríamos un comportamiento más estable en estado estacionario.
- Acción derivativa: es la tasa de variación del error con respecto al tiempo y se relaciona con los valores futuros predichos por el controlador. El objetivo de esta acción es mejorar la respuesta del sistema en estado transitorio o, en otras palabras, responder a los cambios en el sistema.

Este controlador tiene distintas formas de expresarse, pero por simplicidad y para segmentar su aplicación, se utilizará la forma en paralelo [14]:

$$e(t) = q_d(t) - q(t)$$

Ec. 6

$$u(t) = -k_p e(t) - k_i \int_0^t e(\tau)d\tau - k_d \frac{d}{dt} e(t)$$

Ec. 7

La expresión 6 define la señal de error, que es la diferencia entre la señal de referencia o *set-point* $q_d(t)$ y la medición de la señal de salida $q(t)$. La ecuación 7 es para un sistema continuo, sin embargo, la implementación en código que haremos será a través de su expresión para sistemas discretos [14] debido a la naturaleza digital de las computadoras:

$$u(t_k) = -k_p e(t_k) - k_i \frac{1}{T} \sum_{i=0}^{k} e(t_{k_i})\mathrm{T} - k_d \frac{e(t_{k_i}) - e(t_{k_{i-1}})}{T}$$

Ec. 8

Donde:

- k_p: constante de acción proporcional
- k_i: constante de acción integral
- k_d: constante de acción derivativa
- T: tiempo de muestreo
- u: salida del controlador
- t_k: tiempo discreto

El algoritmo con el que se debe realizar la ley de control se compone de los siguientes pasos:

1. Realizar una medición del tiempo actual con relación al tiempo en que se inició el movimiento.
2. Obtener el valor de referencia a partir del seguimiento de trayectoria.
3. Calcular la acción proporcional.
4. Calcular la acción derivativa.
5. Calcular la acción integral.
6. Calcular y devolver la salida del controlador con la expresión 8
7. Actualizar la posición del robot después del envío de la señal de salida.

Finalmente, la principal desventaja de este método es encontrar las ganancias para las acciones de control. Existen múltiples procedimientos sistemáticos para encontrarlas, pero la naturaleza no lineal de un sistema robótico complica la utilización de ellos, por lo que la aproximación más simple es el método heurístico, que consiste en encontrar dichos valores a través de prueba y error. En las tablas 7 y 8 se pueden encontrar los valores finales obtenidos para el desempeño aceptable del movimiento en cada robot, resaltando que necesitaremos un juego de constantes por cada articulación.

Articulación	$\boldsymbol{k_p}$	$\boldsymbol{k_i}$	$\boldsymbol{k_d}$
q_1	80	8	0
q_2	105	8.09	0
q_3	100	8.5	0
q_4	10	0	0
q_5	10	0	0

Tabla 7: Constantes PID para el robot A255

Articulación	$\boldsymbol{k_p}$	$\boldsymbol{k_i}$	$\boldsymbol{k_d}$
q_1	20	1	0.05
q_2	100	3	0.05
q_3	100	3	0.05
q_4	80	1	0.2
q_5	100	3	0.1
q_6	40	1	0.05

Tabla 8: Constantes PID para el robot A465

Para la trayectoria que seguirá el control al realizar un movimiento, se utilizará un polinomio de quinto orden, cuyas expresiones de posición y velocidad para cada articulación son [1]:

$$\boldsymbol{q} = a_0 + a_3 t^3 + a_4 t^4 + a_5 t^5$$

Ec. 9

$$\dot{\boldsymbol{q}} = 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^5$$

Ec. 10

Donde las constantes del polinomio serán:

$$\boldsymbol{a}_0 = \boldsymbol{q}_0$$

Ec. 11

$$\boldsymbol{a}_3 = 10\left(\frac{\boldsymbol{q}_f - \boldsymbol{q}_0}{t_f^3}\right)$$

Ec. 12

$$\boldsymbol{a}_4 = -15\left(\frac{\boldsymbol{q}_f - \boldsymbol{q}_0}{t_f^4}\right)$$

Ec. 13

$$\boldsymbol{a}_5 = 6\left(\frac{\boldsymbol{q}_f - \boldsymbol{q}_0}{t_f^5}\right)$$

Ec. 14

Donde:

- $\boldsymbol{q}_0$: posición inicial al comienzo de la trayectoria
- $\boldsymbol{q}_f$: posición final de la trayectoria
- t_f: tiempo en que debe completarse la trayectoria

3.5 Descripción del sistema cooperativo

La distinción entre ambos robots, a pesar de ser controlados por el mismo cRIO, se considerará en la creación de la biblioteca de software por el requerimiento en la Sección 3.3, por lo que los brazos serán controlados de forma individual. Esto eleva a un nivel más alto en la abstracción el trabajo cooperativo y podemos definirlo como tareas.

Para poder comunicarnos con los robots a través de comandos, se definirá un componente interprete que se encargue de *traducir* dichas instrucciones en operaciones del robot a bajo nivel. En general, podremos dividir los comandos en dos tipos: individuales y cooperativos. Los individuales serán duplicados para que cada robot cuente con su orden propia para la misma tarea y la lista debe comprender:

- Ir a la posición de referencia

- Ir a un punto en coordenadas articulares
- Ir a un punto en coordenadas cartesianas
- Establecer la posición de referencia
- Realizar una lectura de la posición de los *encoders*

Para los comandos cooperativos se repetirán los de movimiento individual para que sean ejecutados por ambos robots al mismo tiempo, pero en movimiento coordinado – que consiste en desplazarse exactamente de la misma forma – y movimiento invertido – como si ambos robots fueran un reflejo en un espejo. La lista de comandos sería la siguiente:

- Ir a la posición de referencia
- Ir a un punto en coordenadas articulares de forma coordinada
- Ir a un punto en coordenadas articulares de forma invertida
- Ir a un punto en coordenadas cartesianas de forma coordinada
- Ir a un punto en coordenadas cartesianas de forma invertida

En cuanto a la interfaz gráfica, esta condensará la implementación presentando medios para ejecutar todas las tareas del intérprete, la configuración para la planta (las cotas de voltaje y la activación de las articulaciones) y la configuración del controlador (constantes de la ley de control, tiempo de muestreo y tiempo de trayectoria).

4 Implementación y uso

En el mundo del software, siempre es importante contar con una especificación funcional antes de comenzar a escribir código. En el Capítulo tres creamos precisamente esa especificación que marcará la ruta de la implementación hacia los objetivos que buscamos.

En este capítulo se presentará la implementación de la interfaz para comunicarnos con la planta y cada uno de los componentes que forman la arquitectura de software definida; la intención es ser un material de referencia técnico desde el cual se pueda entender cómo utilizar cada módulo en caso de que se quiera crear otro proyecto bajo las mismas tecnologías o cómo replicar la arquitectura en una tecnología distinta, definiendo así el marco de trabajo buscado en los objetivos de esta tesis. También se mostrará la construcción de la interfaz gráfica y cada una de las funcionalidades que brinda al implementar los módulos, además de dedicar una apartado específico al componente graficador. En la última sección se incluye una memoria de los experimentos que se realizaron para validar el funcionamiento de la planta.

Respecto a la selección de las tecnologías a usar, en el caso de la interfaz de comunicación, el lenguaje seleccionado fue **C++**, en primer lugar porque la API del cRIO fue exportada en él y en segundo porque la compilaremos en forma de `dll` - que es una biblioteca que contiene código y datos que pueden usar más de un programa al mismo tiempo [15] en entornos de Windows – ya que la mayoría de lenguajes modernos permiten importar funciones a partir de este tipo de archivos, además de que existen métodos para ser usados desde Linux, con lo que también abrimos la oportunidad a utilizar otros sistemas operativos. Para la construcción de la interfaz gráfica se utilizará **C#** por la facilidad que otorga para la manipulación de arreglos y cadenas de texto, su naturaleza orientada a objetos y por su componente de Windows Forms – marco de interfaz de usuario para compilar aplicaciones de escritorio de Windows [16] – que nos permite crear ventanas arrastrando y soltando elementos, abstrayendo toda la programación trivial en la construcción de una interfaz gráfica.

Windows Forms también puede utilizarse desde C++, entonces, superficialmente pareciera que es una complicación innecesaria el utilizar un segundo lenguaje para usar una herramienta que ya está disponible en el primero. Sin embargo, el punto es justo ese, probar que esta arquitectura funciona a través de dos tecnologías diferentes. Utilizar otro lenguaje como Python o JavaScript sólo se difiere de esta implementación en el conocimiento sobre la herramienta para generar interfaces gráficas – por ejemplo, Tkinter de Python, Electron de JavaScript o las GUI de MATLAB – volviendo independiente el desarrollo a la tecnología usada y permitiendo que el investigador utilice las herramientas que ya conozca, en lugar de verse forzado a aprender una nueva.

Por último, siguiendo la Sección 4.1 del capítulo, el desarrollo puede llevarse a otra tecnología. La interfaz de comunicación `Robot.dll` y los archivos heredados del proyecto anterior deben incluirse, `Robot.cs` es la implementación de la `dll` en C#, por lo que, según el lenguaje, debe tener un homólogo – Robot.py en Python, Robot.js en JavaScript o Robot.m en MATLAB – y este será el corazón de la arquitectura de la planta. El resto de los elementos son opcionales, pero seguir la estructura es una buena manera de ahorrarse el trabajo aquí expuesto. En caso de que la implementación sea distinta, la modularidad en la arquitectura permite la adecuación al nuevo proyecto:

- Agregando una nueva ley de control en la clase Control y agregando una nueva tarea que la utilice el intérprete.

- Agregando una nueva trayectoria como método a la clase Trajectory y su correspondiente comando en el intérprete.
- Construyendo una nueva interfaz de usuario que utilice la arquitectura.
- Agregando un nuevo método al graficador que utilice los datos de otra forma.

En resumen, el objetivo es que este marco de trabajo cubra las necesidades básicas comunes de la planta, siendo lo suficientemente flexible para adaptarse a la diversidad de proyectos que pudiesen surgir. En la Ilustración 21 podemos observar el diagrama de arquitectura presentado en el capítulo anterior, pero con los nombres de los artefactos de software específicos para este proyecto.

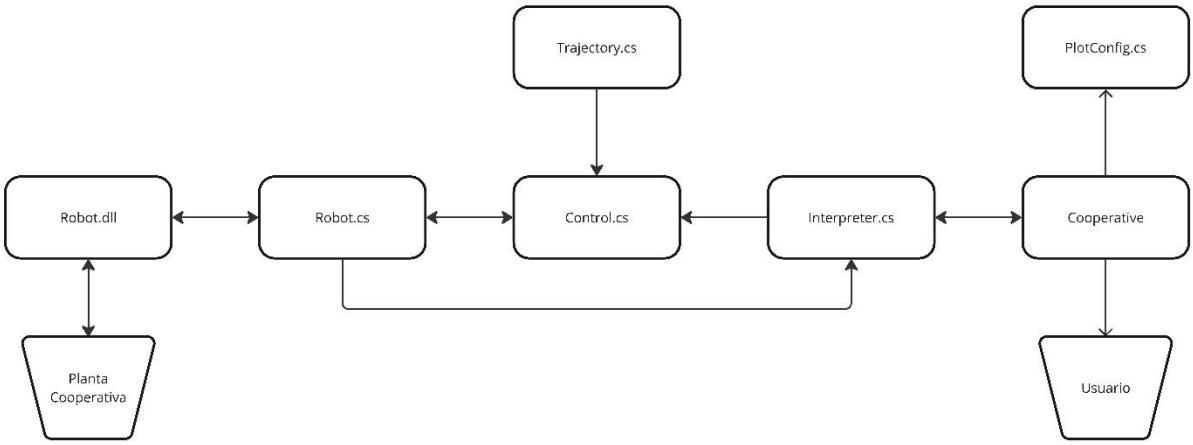

Ilustración 21: Diagrama de arquitectura para la planta cooperativa con los nombres de los artefactos de software

4.1 Componentes

Interfaz de comunicación

El primer y más importante componente dentro de la arquitectura es la interfaz de comunicación con el cRIO, que se definirá en la clase `Robot.h` en C++ y que permitirá abstraer el modelo de información de cada robot. Esta clase debe ser capaz de distinguir si se trata del A255 o del A465 para evitar la duplicidad de código, para ello se identificará a cada manipulador con una constante numérica y en cada método se distinguirá por medio de una condicional si se trata de uno u otro, además de esto, debe recibir la dirección IP configurada en el cRIO y la ruta del archivo `lvbitx`; también se almacenarán los voltajes mínimos y máximos configurados, la constante de conversión digital/analógica para el voltaje de salida, los valores de radianes por cuenta del encoder en cada articulación y su respectiva conversión a grados, además del vector de posición. Salvo el factor de conversión digital/analógica para el voltaje de salida, los demás datos son conjuntos numéricos que representan valores para cada articulación, la longitud seleccionada para definirlos es de seis elementos dados los grados de libertad del A465. En el uso, aunque el robot A255 posea sólo cinco grados de libertad, el último valor para cualquiera de las propiedades será gestionado por la clase y sólo se enviará si se trata del A465; no tendría efecto colocarle un valor, aunque para evitar confusiones bastaría con configurarlo como cero o falso según el dato del que se trate.

En el caso de la función para conectarse, si se definen dos objetos de la clase `Robot.h`, es necesario que cada uno establezca su conexión. Los métodos de configuración tienen como valor de retorno un entero que facilitaría el manejo de errores en caso de que se produjeran, estos se pueden consultar en el archivo

`NiFpga.h` en las definiciones de los valores `NiFpga_status`. Finalmente, cada método debe ser expuesto para que, al compilar la clase como un archivo `dll`, pueda ser utilizado por la plataforma de desarrollo del proyecto: se define un método externo con el mismo nombre del método a exponer, pero sobrescribiéndolo al recibir una instancia de la clase `Robot` como primer parámetro.

```
extern "C" ROBOT_API int connect(Robot* r) {
	return r->connect();
}
```

Código 1: Ejemplo de exposición del método connect

La intención de la clase es no tener que replicar este código cada vez que se implemente un proyecto, ya que su definición siempre será la misma, bastaría con incluir el archivo `Robot.dll` para poder utilizar los métodos expuestos, además de los archivos mencionados en la definición del requerimiento.

El modo de uso, en caso de ejecutarse directamente desde C++ es:

1. Crear una instancia de la clase Robot, dándole como argumentos el robot del que se trata: 0 para el A255 y 1 para el A465.
2. Ejecutar el método `connect` pasándole en forma de `string` la dirección IP del cRIO y la ruta absoluta del archivo `lvbitx`.
3. Configurar voltajes mínimos, máximos y activar las articulaciones que se van a utilizar.
4. Hacer una lectura de seguridad de los *encoders* para verificar que la conexión se estableció correctamente.
5. Comenzar a enviar voltajes a las articulaciones.

La Tabla 9 muestra los métodos que conforman la interfaz de comunicación.

Método	**Retorno**	**Parámetros**	**Descripción**
`RobotDLL`	`void*`	`int robot`	Constructor de la clase Robot. Recibe el identificador numérico del robot a utilizar. Devuelve un puntero al objeto creado.

`connect`	`int`	`Robot* robot`
		`char* ip`
		`char* route`
			Método de conexión con el controlador cRIO. Recibe una instancia de la clase Robot, la cadena de caracteres que forma la dirección IP y la ruta del archivo `lvbitx`. Devuelve un entero con el estatus de la conexión, 0 significa que fue exitosa.

`activeJoints`	`int`	`Robot* robot` `bool* q`	Método para activar las articulaciones del robot. Recibe una instancia de la clase Robot y un arreglo de 6 elementos de tipo booleano con la configuración deseada. Devuelve un entero con el estatus de la operación, 0 significa que fue exitosa.
`setMinVoltages`	`int`	`Robot* robot` `double* v`	Método para configurar el límite de voltaje mínimo del robot. Recibe una instancia de la clase Robot y un arreglo de 6 elementos de tipo `double` con la configuración deseada. Devuelve un entero con el estatus de la operación, 0 significa que fue exitosa.

`setMaxVoltages`	`int`	`Robot* robot` `double* v`	Método para configurar el límite de voltaje máximo del robot. Recibe una instancia de la clase Robot y un arreglo de 6 elementos de tipo `double` con la configuración deseada. Devuelve un entero con el estatus de la operación, 0 significa que fue exitosa.
`readJoints`	`void`	`Robot* robot` `double* result`	Método para leer las articulaciones del robot. Recibe una instancia de la clase Robot y un arreglo de 6 elementos de tipo `double` donde escribirá los valores leídos. No devuelve datos.
`writeJoints`	`void`	`Robot* robot` `double* v`	Método para escribir voltajes en las articulaciones del robot. Recibe una instancia de la clase Robot y un arreglo de seis elementos de tipo `double` con los valores a enviar. No devuelve datos.
`setHome`	`void`	`Robot* robot` `double* q`	Método para establecer la posición de referencia con respecto a la cual se realizan las mediciones. Recibe una instancia de la clase Robot y un arreglo con los valores articulares que se desean establecer como la posición actual. No devuelve datos.

`directKinematics`	`void`	`Robot* robot`
		`double* q`
		`double* x`
			Método para calcular la cinemática directa del robot. Recibe un arreglo de tipo `double` de seis elementos con los valores de las coordenadas articulares y un arreglo de tres elementos de tipo `double` donde escribirá las coordenadas cartesianas.

`inverseKinematics`	`Void`	`Robot* robot`
		`double* x`
		`double* q`
			Método para calcular la cinemática inversa del robot. Recibe un arreglo de tipo `double` de tres elementos con los valores de las coordenadas cartesianas y un arreglo de seis elementos de tipo `double` donde escribirá las coordenadas articulares.

Tabla 9: Funciones expuestas en la biblioteca Robot.dll

Instancia de la interfaz de comunicación

Dependiendo del entorno de desarrollo seleccionado para desarrollar el proyecto, se deberá crear una clase espejo que implemente todas las funciones de Robot.dll en el nuevo lenguaje de programación. La importación de las funciones variará según el lenguaje, para este proyecto se utilizará C# y en el código 2 se puede observar la forma de hacerlo, mientras que en el código 3 se observa la nueva implementación del método para un uso más cómodo. De nuevo, este proceso será ligeramente diferente en cada lenguaje, pero la idea es la misma, creando en este caso la clase `Robot.cs`. Si se quisiera desarrollar un proyecto en C# completamente distinto al propuesto en la arquitectura, la clase `Robot.cs` seguiría siendo necesaria; de igual forma, si el nuevo proyecto se escribiera en otro lenguaje, tendríamos que crear un análogo Robot.py, Robot.js, Robot.m, según sea el caso.

```
[DllImport("Robot.dll", CharSet = CharSet.Ansi, CallingConvention = CallingConvention.Cdecl)]
```

```
static extern int connect(IntPtr r, string ip, string route);
```
Código 2: Ejemplo de sintaxis en C# para importar una función, en este caso implementando el método de conexión de Robot.dll

```
public void Connect(string ip, string route){
    this.connected = connect(robot, ip, route);
                                            }
```
Código 3: Ejemplo de implementación en C# del método importado

Este proceso debe repetirse por cada función de la biblioteca Robot.dll, por lo que sería redundante repetir la tabla de métodos y el uso para Robot.cs. La única mención relevante que podría hacer falta es que se agregó un atributo llamado `connected` que, en el método `connect` guarda el resultado de la conexión, esto con la intención de utilizar dicho dato en la interfaz gráfica.

Control

El componente de control será el encargado de codificar la ley de control y todo el comportamiento relacionado a ella para el movimiento de los robots. La información de sus propiedades puede ser segmentada en cuatro partes:

- Tiempo: el control necesita conocer el tiempo actual, los tiempos inicial y final de un movimiento, el periodo de muestreo y, para gestionar todo esto, utilizará un timer encargado de ejecutar el movimiento o mantener la posición actual en todo momento y un reloj que permitirá completar la trayectoria deseada en el tiempo final establecido.
- Estado: para realizar mediciones en la posición será necesaria una instancia de la clase Robot, además de ello, es necesario llevar un seguimiento de la posición actual del robot, la velocidad (calculada a partir de las mediciones de posición), la posición inicial y final, la posición de referencia y un vector de ceros por utilidad.
- Trayectoria: la información de la trayectoria se encapsulará en un objeto de la clase `Trajectory`, definida más adelante.
- Control: la naturaleza del control PID nos marca la necesidad de tener un juego de constantes `kp`, `ki` y `kd` para cada componente de la ley de control, además de conocer el error de posición, el error de velocidad y la integral del error de posición.

El comportamiento de la clase también se compone de cuatro métodos principales:

- `timerSetup`: se encarga de la configuración inicial del timer, estableciendo el tiempo de muestreo y el método `move` como callback a llamar cada que el periodo se complete.
- `move`: es el método que regirá el control de posición en todo momento. Ejecutándose a través del timer en cada periodo de muestreo, primero realizará una lectura de los *encoders*, luego ejecutará la ley de control para calcular los voltajes necesarios en ese punto del tiempo, para finalmente enviar dichos voltajes a las articulaciones del robot.
- `goTo`: este método se utilizará cada que queramos mover el robot a un punto distinto. Realizará una lectura del tiempo actual y de los *encoders* para establecer la posición del robot en el momento de la ejecución como la posición inicial del movimiento y la posición actual, recibirá también un punto de referencia que establecerá como posición final y enviará estas condiciones iniciales al componente de trayectoria.

- `pidControl`: en primer lugar calculará el tiempo actual a partir del tiempo inicial del movimiento; después obtendrá el valor de referencia para el controlador: si nos encontramos en un tiempo menor al tiempo de trayectoria, obtendrá el punto de referencia del objeto `trayectory`, si no, mantendrá la posición final del movimiento; de forma posterior calculará los errores proporcional, integral y derivativo para componer los valores de la ley de control y, finalmente, establecer la posición alcanzada como la posición actual del robot.

El uso de este componente está ligado a las instancias de las clases `Robot` y `Trajectory`, ya que son utilizados en su ejecución. Considerando esto, el uso de la clase `Control` será el siguiente:

1. Crear una instancia de la clase `Robot`.
2. Crear una instancia de la clase `Control` pasándole el robot como argumento en su constructor.
3. Configurar el tiempo de muestreo y las constantes proporcional, integral y derivativa.
4. Ejecutar `timerSetup` para realizar la configuración del timer.
5. Realizar un `goTo` al vector de ceros definido para buscar la posición de home como inicio.
6. Ejecutar el método `Start` del timer y del reloj para que el control comience a computar.
7. Ejecutar el método `goTo` cada vez que se quiera cambiar de posición.

En caso de que se quiera experimentar con diferentes leyes de control, sólo sería necesario implementar un nuevo método con la lógica del controlador y sustituir la ejecución de `pidControl` en el `método` move.

Trajectory

Ya se mencionó en el componente de control que este elemento almacenará la lógica para el seguimiento de trayectoria en la clase `Trajectory.cs`. En sus atributos definiremos las constantes del polinomio de quito grado y su comportamiento es muy simple:

- `poly`: es el método que se utilizará para calcular las constantes del polinomio de trayectoria a partir de la posición inicial y final del robot cumpliendo con alcanzarlo en un determinado tiempo final.
- `polyPos`: teniendo configuradas las constantes del polinomio, este método recibirá un instante de tiempo y devolverá el valor de posición en dicho instante para una articulación determinada.
- `polySpeed`: la idea es similar a `polyPos` pero calculando el valor de velocidad.

La utilización del componente de trayectoria consiste en:

1. Crear una instancia de la clase `Trayectory`.
2. Cada vez que se requiera iniciar un nuevo movimiento, ejecutar el método `poly` para establecer sus condiciones iniciales.
3. Ejecutar `polyPos` y `polySpeed` en cada instante de tiempo para obtener los valores articulares de posición y velocidad.

Interpreter

El intérprete será la máxima capa de abstracción del desarrollo, tomando todos los componentes anteriores y gestionándolos para exponernos una interacción con la planta a nivel de tarea. Considerando esto, es aquí donde formamos la abstracción del concepto de la planta, por lo que en sus atributos necesitaremos dos parejas de instancias de la clase `Robot` y la clase `Control`, una pareja para el A255 y la otra para el A465. En su comportamiento sólo encontraremos el método `processLine` que recibe un comando en forma de `string` como parámetro y se encargará de realizar las acciones establecidas para

éste. En la Tabla 10 se encuentra el listado y la descripción de los comandos, cabe destacar que las instrucciones con el prefijo A son para el robot A255, las B para el A465 y las C para trabajar de forma cooperativa.

Comando	Parámetros	Descripción
A00	-	Movimiento a home para el robot A255.
A01	Q[double] W[double] E[double] R[double] T[double]	Movimiento a un punto en el espacio articular para el robot A255. Cada parámetro, en orden, representa el valor de una variable articular.
A02	X[double] Y[double] Z[double]	Movimiento a un punto en el espacio cartesiano para el robot A255. Cada parámetro representa el valor de una coordenada.
A03	-	Establecer el punto actual como home para el robot A255.
A04	-	Realizar una lectura de los *encoders* del robot A255, devolviéndola en un formato de `string`.
B00	-	Movimiento a home para el robot A255.
B01	Q[double] W[double] E[double] R[double] T[double] Y[double]	Movimiento a un punto en el espacio articular para el robot A465. Cada parámetro, en orden, representa el valor de una variable articular.
B02	X[double] Y[double] Z[double]	Movimiento a un punto en el espacio cartesiano para el robot A465. Cada parámetro representa el valor de una coordenada.
B03	-	Establecer el punto actual como home para el robot A465.
B04	-	Realizar una lectura de los *encoders* del robot A465, devolviéndola en un formato de `string`.
C00	-	Movimiento a home de forma cooperativa.
C01	Q[double] W[double] E[double] R[double] T[double] Y[double]	Movimiento a un punto en el espacio articular de forma cooperativa coordinada. Cada parámetro, en orden, representa el valor de una variable articular.
C02	X[double] Y[double] Z[double]	Movimiento a un punto en el espacio cartesiano de forma cooperativa coordinada. Cada parámetro representa el valor de una coordenada.
C11	Q[double] W[double] E[double] R[double] T[double] Y[double]	Movimiento a un punto en el espacio articular de forma cooperativa invertida, el movimiento del parámetro W será contrario entre los robots. Cada parámetro, en orden, representa el valor de una variable articular.
C12	X[double] Y[double] Z[double]	Movimiento a un punto en el espacio cartesiano de forma cooperativa invertida, el movimiento de la coordenada Y será contrario entre los robots. Cada parámetro representa el valor de una coordenada.
C21	Q[double] W[double] E[double] R[double] T[double] Y[double] F[double] G[double] H[double] J[double] K[double]	Movimiento simultáneo a dos puntos en el espacio articular, uno para cada robot. Los primeros seis parámetros representan los valores de las coordenadas para el A465 y los restantes para el A255.

C22	X[double] Y[double] Z[double] U[double] V[double] W[double]	Movimiento simultáneo a dos puntos en el espacio cartesiano, uno para cada robot. Los primeros seis parámetros representan los valores de las coordenadas para el A465 y los restantes para el A255

Tabla 10: Lista de comandos para el intérprete

Si se deseara agregar una nueva tarea al módulo, bastaría con agregar un caso más al método `processLine` con el comando deseado, agregar la lógica para procesar sus parámetros y ejecutar las instrucciones deseadas.

4.2 Interfaz humano-máquina

Teniendo ya los componentes construidos, el usuario necesita una forma correcta de interactuar con la planta e interpretar la información que viene de ella. A veces se subestima la cantidad de tiempo acumulado que se pierde al ejecutar una prueba directamente desde código, por lo que es menester de optimización utilizar la herramienta adecuada para cada trabajo. El mundo actual está lleno de interfaces gráficas que nos permiten interactuar con diferentes dispositivos y, aunque no son el único medio por el que se puede usar un producto, son el más común ya que la vista es por lejos el sentido dominante con el que interpretamos el mundo.

Siguiendo los requerimientos definidos en el capítulo anterior, la interfaz gráfica estará dividida en cuatro vistas principales:

- El panel de robots
- La configuración de la planta
- La configuración del control
- Graficador

El panel de robots es la vista más importante de la interfaz, ya que en ella se encuentran todas las funcionalidades de la planta. Podemos dividirla en cuatro grupos:

- Grupo de conexión
- Grupo de datos
- Grupo de control automático
- Grupo de control manual

En el grupo de conexión, podemos encontrar dos cajas de selección, una para el A255 y otra para el A465, además de un botón para establecer la comunicación con el cRIO. Para este último, es necesario que se escoja al menos un robot primero, de otro modo, al presionar el botón no se establecerá la comunicación con el controlador y se enviará un mensaje indicando esto. Si la conexión es exitosa, se enviará una alerta mostrándolo y el estado de la barra en la parte más baja de la ventana cambiará a “Connected” en color verde. Una vez establecida la conexión, podemos seleccionar y deseleccionar los robots a voluntad sin necesidad de conectarnos de nuevo, ya que estos cuadros de selección nos permiten decidir quién va a responder a las instrucciones de cada botón que tengamos en toda la interfaz gráfica.

El grupo de datos contiene un área de texto que mostrará el valor numérico de las variables articulares en tiempo real mientras el control se está ejecutando. A pesar de esta visualización, una buena práctica de seguridad es realizar una lectura a los *encoders* para verificar el correcto funcionamiento de estos y que los robots se encuentran adecuadamente en lazo cerrado; esta función se ejecuta con el botón “Read

encoders” y el resultado se mostrará en una alerta por cada robot seleccionado. Por último, aquí también se aloja el botón “Set home” para establecer el punto de referencia del movimiento, el cual configurará en cero el valor de todas las variables articulares.

En el grupo de control manual podemos encontrar seis barras de deslizamiento, una por cada articulación, desde las que podemos enviar un movimiento directamente a cada articulación para calibrar la posición. Estas también son afectadas por el botón “Set home”, el cual las regresa a su punto medio para que siempre nos movamos en torno a la posición actual de los robots.

Finalmente, el grupo de control automático cuenta con dos pestañas, en una podemos establecer valores en el espacio articular y en la otra en el espacio cartesiano, haciendo que el robot se mueva a dicho punto con el botón “Go to setpoint” o regrese con el botón “Go home”; el botón “clear” es sólo una utilidad para limpiar los campos para enviar valores. En la parte baja fuera de las pestañas, tenemos un cuadro de texto desde el que podemos introducir instrucciones en el lenguaje del intérprete para ser ejecutadas con el botón “Send command”

Como nota, se debe mencionar que para el caso del robot A255 que cuenta con sólo cinco grados de libertad a diferencia de su compañero, los componentes que hacen referencia a q6 no tendrán efecto alguno. En la Ilustración 22 podemos observar la vista del panel de robots.

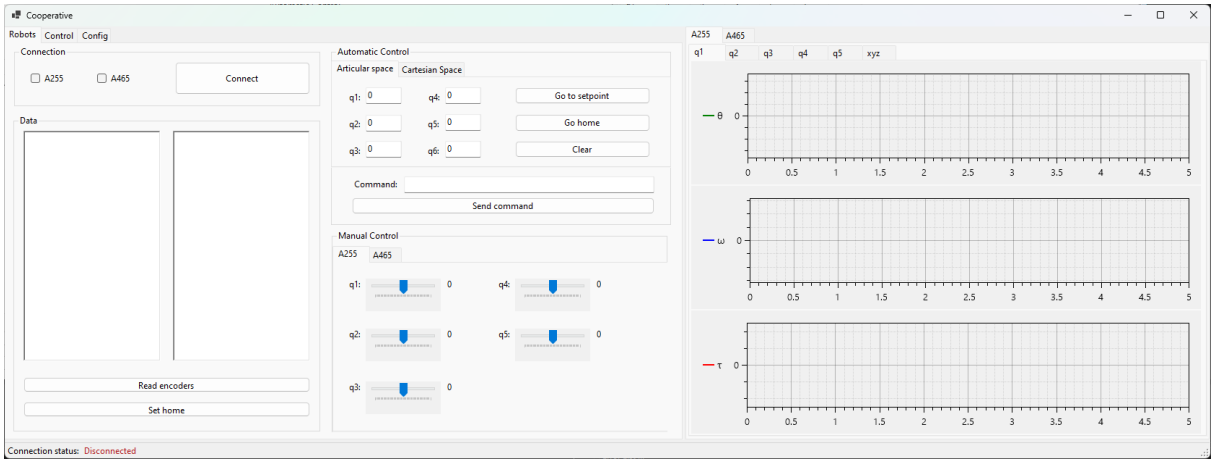

Ilustración 22: Panel de robots de la interfaz humano-máquina

En la configuración de la planta – Ilustración 23 - se tienen dos secciones idénticas, una para el A255 y otra para el A465, desde las que podemos configurar para cada articulación el valor máximo y mínimo de voltaje que otorgarán las salidas, además de si esta estará activa o no. Esto permite que podamos mantener seccionado el funcionamiento de cada articulación para estudiarlas individualmente si es necesario. También se incluyen dos campos de texto para establecer la dirección IP del cRIO y la ruta absoluta del archivo `lvbitx`, quitando la necesidad de establecerlas directamente en el código.

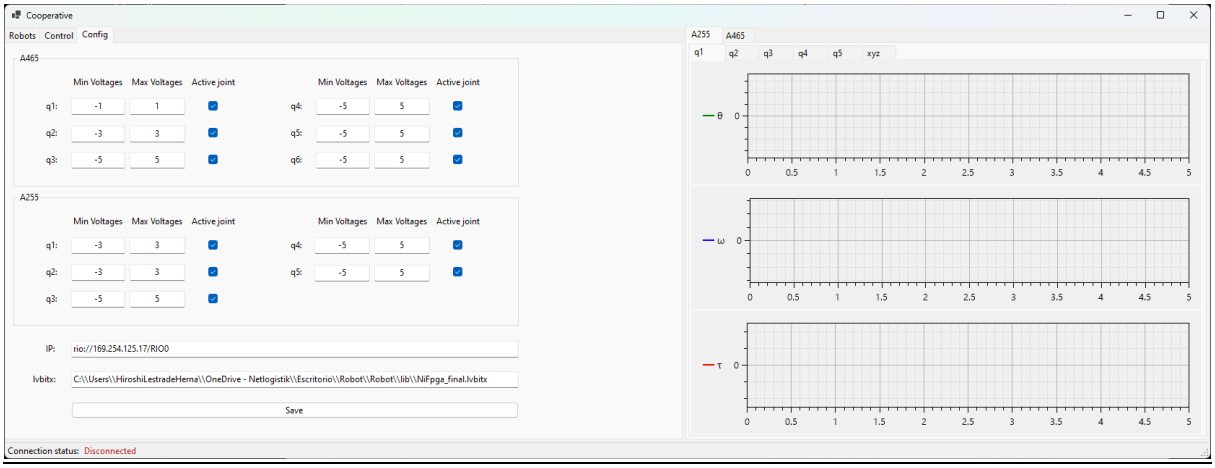

Ilustración 23:Vista de configuración de la planta

La idea detrás de la vista de configuración para el control es muy similar a la de la configuración de la planta. De igual forma, se puede observar en la Ilustración 24 que tenemos dos secciones, una para cada robot, desde las que podemos configurar el valor de las constantes proporcional, integral y derivativa de la ley de control, además también podemos definir el tiempo final en que queremos que se complete la trayectoria y el tiempo de muestreo para la ley de control y el graficador. El botón “Save" configurará los valores en el código, por lo que podremos observar los efectos en la ley de control de forma inmediata. Cabe mencionar que la función de guardado para ambas vistas de configuración, no sólo modifica los valores de las variables en tiempo de ejecución, sino que guarda la información en un archivo de formato `json`, por lo que la siguiente vez que se ejecute la interfaz, recordará la última configuración guardada.

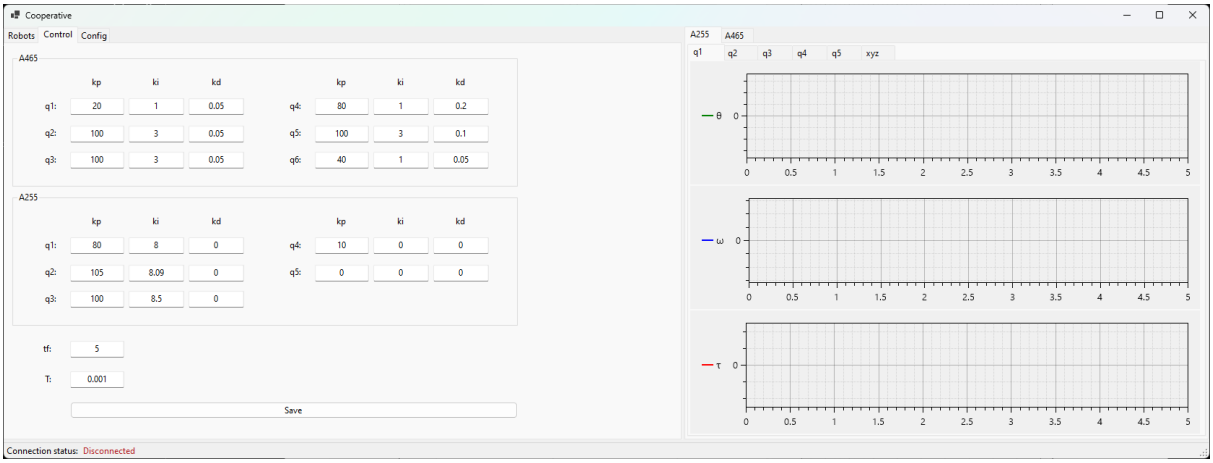

Ilustración 24: Vista de configuración del control

El último componente es el graficador. Omitida en el capítulo anterior, la clase `Plotter` contiene dos métodos, `configArt` y `configCart`, que se encargan de añadir y estilizar las gráficas en la sección derecha de la vista principal. Debemos crear tres arreglos de objetos `Plotter`, `plots_qA255`, `plots_qA465` y `plots_xyz` en el formulario; el primer y segundo arreglo constarán de cinco y seis elementos respectivamente, que serán para las gráficas de las coordenadas articulares del A255 y del A465; `plots_xyz` tendrá dos elementos que serán para las coordenadas espaciales de cada robot. Cada `Plotter`, según la configuración que le demos, contendrá tres gráficas: posición angular, velocidad angular

y torque para el caso de `configArt` y las coordenadas *x, y, z,* para el caso de `configCart`. La adquisición de datos requiere del uso de un timer para ejecutar las lecturas periódicamente, sin embargo, a diferencia del resto de los componentes, la operación del timer no se abstraerá a `Plotter` ya que tendríamos que gestionar un total de 13 temporizadores. Es más práctico definir un solo timer en la interfaz gráfica que ejecute una función `dataAq` donde se realicen las lecturas para cada graficador, mencionando que la graficación sólo se hará dentro del tiempo final en que el brazo debe alcanzar el setpoint. También aquí se incluirá la muestra de valores en el área de texto del grupo de datos, aprovechando esta implementación periódica dentro del formulario principal. En la Ilustración 25 se observan los componentes de adquisición de datos funcionando.

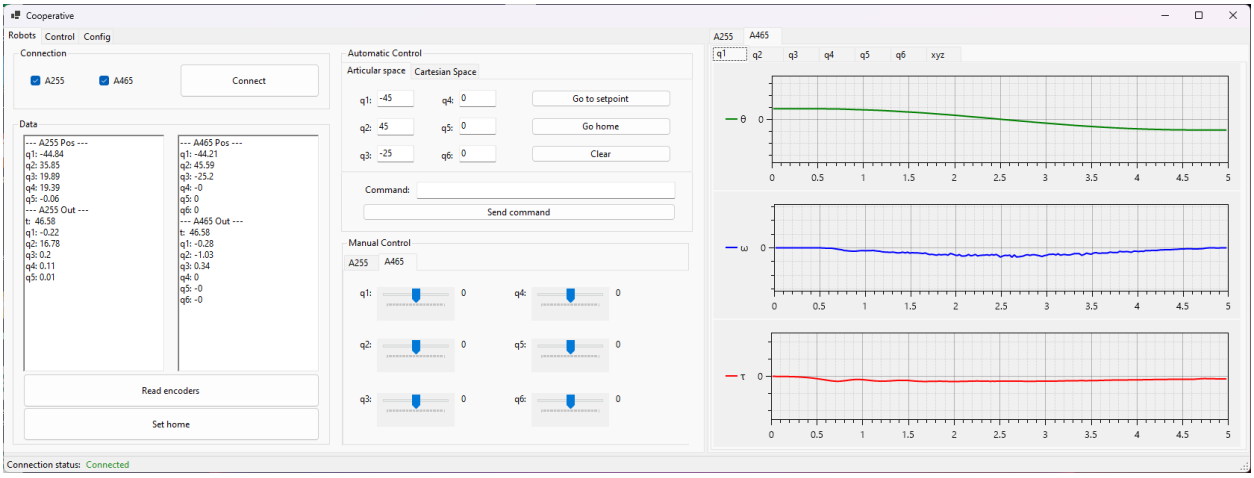

Ilustración 25: Interfaz gráfica operando.

4.3 Memoria de experimentos

El culmen de este trabajo debe reflejar los dos objetivos iniciales: la planta debe estar operativa y el marco de trabajo debe permitir una implementación sencilla donde los robots puedan trabajar de forma cooperativa. Hubo considerables pruebas durante el desarrollo que buscaban reflejar distintos puntos, pero con el avance fueron dando forma a tres ejercicios fundamentales como prueba de la operación del sistema cooperativo:

- Movimiento coordinado
- Movimiento en espejo
- Intercalación de instrucciones

En el movimiento coordinado, el objetivo es enviar la misma posición en el espacio de trabajo a ambos robots y que estos la alcancen de forma síncrona. Obviamente se debe considerar un punto que sea alcanzable por ambos brazos considerando que las dimensiones del A255 son más pequeñas que el A465. La secuencia que se utilizará partirá del origen, enviando una a una las coordenadas de cada punto; pasaremos por alto la orientación debido a que el A255 sólo cuenta con dos grados de libertad para ello y usaremos los controles manuales para colocar el punto de referencia de frente al usuario. Los comandos se listan en la Tabla 11 y la secuencia de cada punto puede observarse en la Ilustración 26 con un dibujo del camino seguido.

No.	Comando
1	C01 Q45 W45 E25
2	C01 Q-45 W45 E-25
3	C01 Q-45 W-45 E25
4	C00

Tabla 11: Comandos del intérprete para el experimento de movimiento coordinado

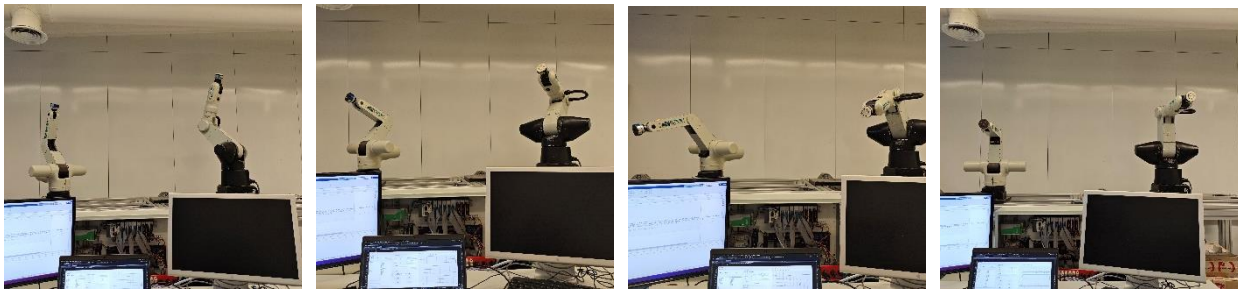

Ilustración 26: Secuencia de puntos del experimento de movimiento coordinado

El experimento de movimiento espejo es similar al anterior con la diferencia de que los efectores finales deben estar uno frente al otro en todo momento, esto obliga a que tomemos la perspectiva de uno de los robots y al otro le enviemos las coordenadas horizontales invertidas. Tendremos las mismas consideraciones que en el experimento anterior, tomando la perspectiva del A465 con el A255 como su espejo.

No.	Comando
1	C11 Q45 W45 E25
2	C11 Q-45 W45 E-25
3	C11 Q-45 W-45 E25
4	C00

Tabla 12: Comandos del intérprete para el experimento de movimiento espejo

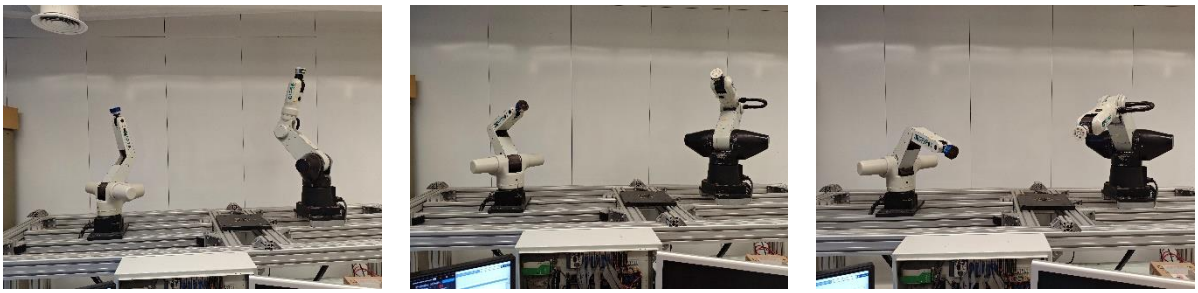

Ilustración 27: Secuencia de puntos del experimento de movimiento espejo

Finalmente, el último experimento busca enviar a ambos robots a puntos distintos al mismo tiempo, por lo que se creó una instrucción que condensa las variables articulares de ambos robots. Las condiciones del experimento serán las mismas que en el movimiento coordinado sólo que a cada robot se le enviarán en orden invertido.

No.	Comando
1	C12 Q45 W45 E25 F-45 G-45 H25
2	C22 Q-45 W45 E-25 F-45 G45 H25
3	C22 Q-45 W-45 E25 F45 G45 H25
4	C00

Tabla 13: Comandos del intérprete para el experimento de movimiento intercalado

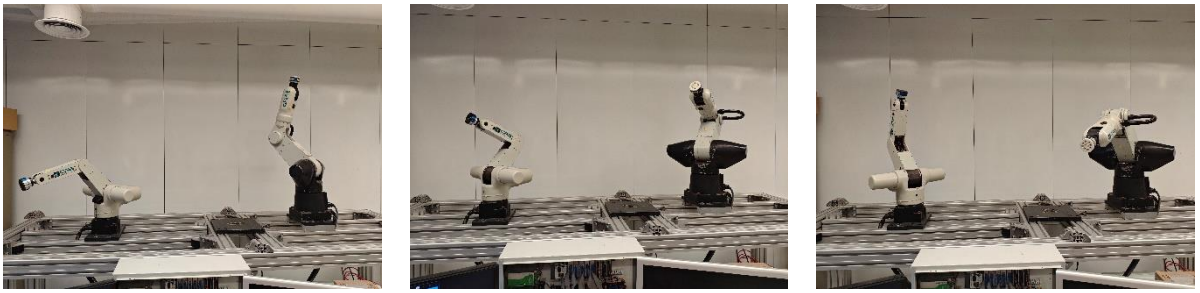

Ilustración 28: Secuencia de puntos del experimento de movimiento intercalado

Es un tanto difícil visualizar el movimiento a través de imágenes, sin embargo, en las figuras 26, 27 y 28 se muestran ejemplos de gráficas para diferentes articulaciones, tomadas de cada uno de los experimentos y en ellas podemos ver que las coordenadas deseadas en cada tarea son alcanzadas y que los tres tipos de movimiento cooperativo se cumplen exitosamente, comprobando así que se cumplieron los objetivos centrales de este trabajo.

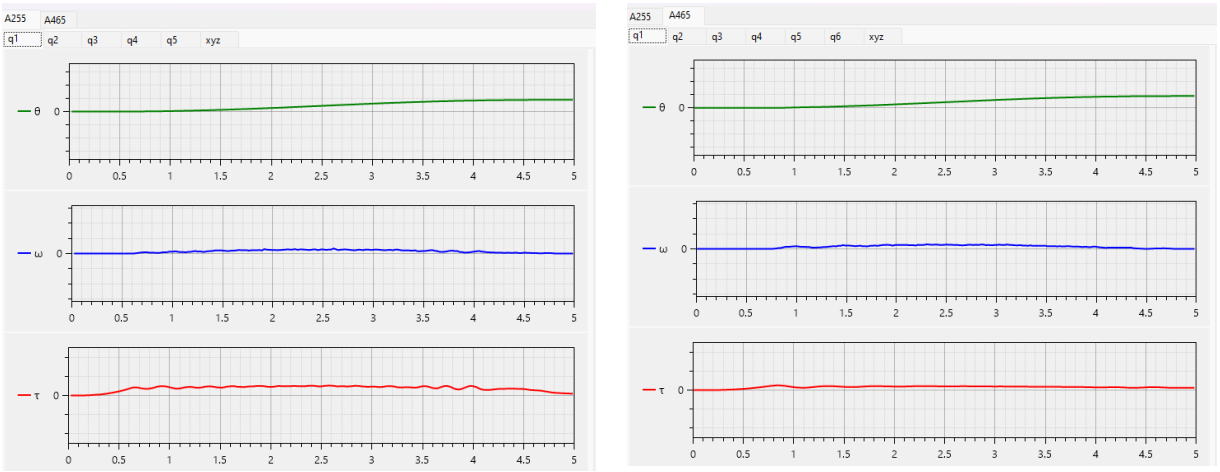

Ilustración 29: Gráficas de la articulación 1 para el experimento coordinado al utilizar el primer comando

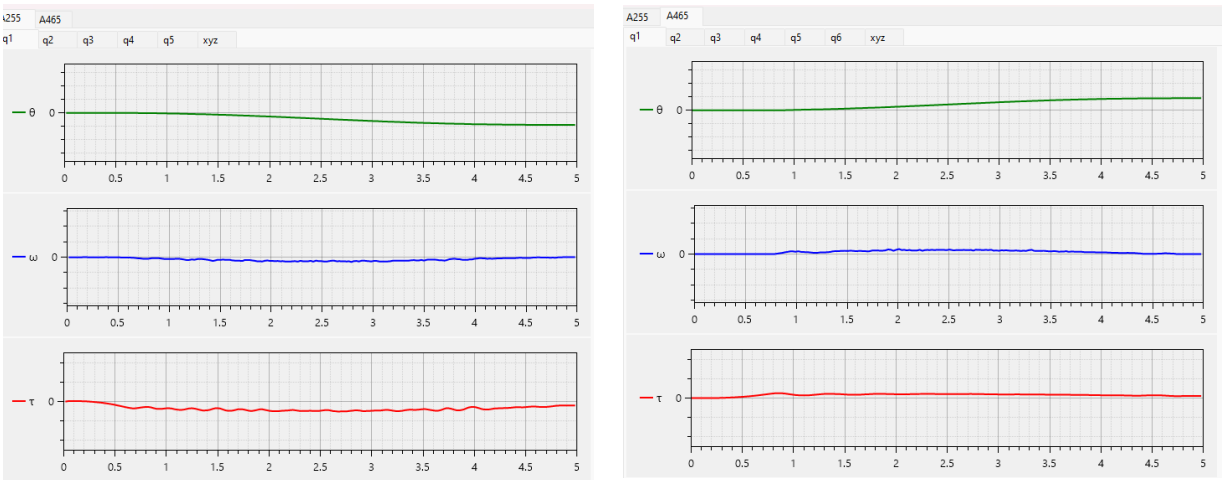

Ilustración 30: Gráficas de la articulación 1 para el experimento invertido al utilizar el primer comando

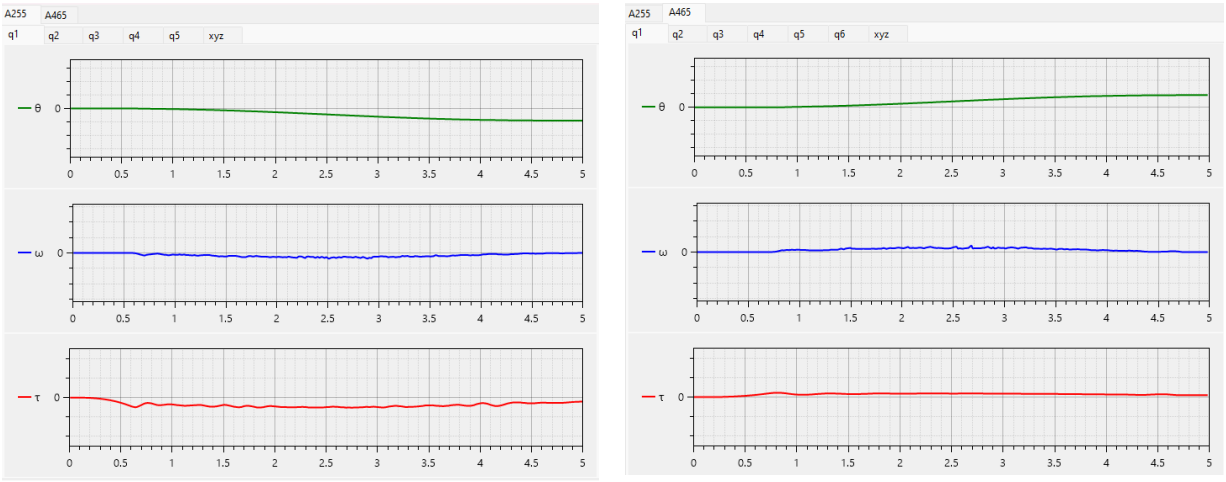

Ilustración 31: Gráficas de la articulación 1 para el experimento intercalado al utilizar el primer comando

El código de todo el desarrollo se puede encontrar en los siguientes enlaces a sus repositorios públicos:

https://github.com/HiroLestrade/cooperativo-system Interfaz gráfica e implementación del sistema.

https://github.com/HiroLestrade/cooperativo-drivers Biblioteca de software y su implementación.

5 Conclusión

Los resultados de este trabajo son tangibles y eso facilita considerablemente validar el cumplimiento de los objetivos planteados en un principio. En la parte de mantenimiento, basta con preguntarse ¿la planta funciona? Más de la mitad de este escrito comprueba que sí, ya que no se podría haber continuado con el desarrollo en caso contrario. La memoria de mantenimiento del Capítulo 2 sintetiza el camino recorrido desde cómo se encontró la planta en un inicio hasta que se pudieron hacer pruebas de movimiento y la tabla al final de la Sección 2.2 constituye un material de consulta en caso de que alguno de los fallos ahí mencionados volviese a producirse.

Los requerimientos encontrados durante el mantenimiento dieron forma a la arquitectura definida en los capítulos posteriores, lo que permitió construir un desarrollo implementado en la interfaz humano-máquina; esto también constituye un resultado tangible ya que el código puede ser revisado en el repositorio citado al final del Capítulo cuatro y el diagrama de arquitectura del Capítulo tres provee una estructura que puede seguirse, modificarse o extenderse. Esto cubre el objetivo del marco de trabajo y establece esta tesis como un punto de partida para nuevos proyectos alrededor de la planta cooperativa.

Por último, este escrito también funciona como un nodo que concentra información principal de la planta que es relevante para los desarrollos, pero también conjunta referencias de dónde extraer esta información. Si bien aquí no se aborda el modelo dinámico de los robots, puede ser consultado en *Local Stability and Ultimate Boundedness in the Control of Robot Manipulators* [4]; si se quiere acceder los esquemas electrónicos de las placas en la planta, este puede consultarse la ya citada tesis [5]; la intención es que no se tenga que pasar de nuevo por la recopilación de información que fue necesaria aquí.

5.1 Trabajo futuro sobre la implementación

Este desarrollo, en las condiciones que finaliza, aún tiene mucho potencial para seguirse experimentando. Como se mencionó al inicio, varias ideas que se deseaban incluir se quedaron fuera de esta tesis, sin embargo, cada una proveería suficiente información para realizar su propio escrito, considerando a todas como secuelas directas de este trabajo. A continuación, explico algunas de esas ideas:

Ley de control con modelo matemático y estimación de parámetros

Dado que el fin de este proyecto sólo era definir una estructura de trabajo, la ley de control, aunque necesaria, no era el eje de la investigación. El controlador PID utilizado permitió saltarse la parte de definir un modelo matemático para el robot y, con ello, tener que lidiar con sus parámetros dinámicos. Al ser robots de arquitectura cerrada en su origen, los parámetros de inercia no suelen ser un dato disponible, pero se pueden estimar geométricamente como en [4] y construir un experimento para validarlo o estimarlos directamente a través de observadores. Determinar el impacto que estos parámetros pueden tener en diferentes leyes de control, además de considerar la dinámica de los motores en el modelo del robot es un tema sólido para continuar.

Extensión del intérprete

La lista de códigos generados para el intérprete es también un tema superficial en esta tesis, ya que se necesita el componente y su definición, pero su contenido se restringe a necesidades básicas. Extender la lista de instrucciones puede ir desde agregar tareas compuestas como la auto calibración o extender las ya existentes como moverse de un punto a otro, pero utilizando distintas trayectorias o distintas leyes de

control. Además, se puede construir un procesador de comandos que permita interpretar un archivo de texto con instrucciones y ejecutarlas una a una para definir tareas aún más complejas.

Prueba de sensores de fuerza y aceleración

El efector final fue aquí el gran ausente. Ambos robots actualmente cuentan con un sensor de fuerza que, desde un principio se pudo validar que funcionaba correctamente. Sin embargo, la inclusión de dicho sensor hubiera obligado a utilizar una implementación más robusta, añadiendo dificultad innecesaria a la definición que se buscaba hacer. Además, el efector de los robots está diseñado de una forma modular, donde distintos sensores pueden acoplarse, por lo que haberlo incluido habría agregado particularidad a una arquitectura generalista. Se puede estudiar la construcción de distintos soportes que permitan agregar nuevos sensores, como cámaras o acelerómetros, y así experimentar con nuevos tipos de control.

Comunicación con otros dispositivos

Las instrucciones para el control de los robots no necesitan venir específicamente de una entidad de software, como lo hemos trabajado hasta ahora. Los controladores C500 tenían de fábrica un teach pendant, que no es más que otro tipo de interfaz para que el usuario se comunique con los robots. Un controlador de hardware similar podría construirse y que éste, a su vez, envíe instrucciones a la planta. También, otros dispositivos podrían tomar el lugar de dicho controlador, como otro robot. El laboratorio cuenta con varios dispositivos hápticos *Geomatic Touch* [4] que podrían utilizarse para enviar información de movimiento.

5.2 Trabajo futuro más allá de la implementación

La intención de este trabajo ha sido abrir la puerta a nuevos desarrollos con la planta cooperativa A255-A465, haciendo accesible, fácil y cómoda su utilización a través del marco de trabajo para nuevos usuarios. La modularidad también ha sido con la intención de que los componentes definidos en la arquitectura puedan ser reimplementados o sustituidos con facilidad, además de ser independientes a la tecnología utilizada. Algunas ideas en esta frontera son las siguientes:

Visión por computadora

La biblioteca OpenCV provee herramientas para el procesamiento de imágenes y modelos pre entrenados para la interpretación de su contenido, como seguimiento o detección de elementos [17]. La implementación más básica sería la medición de la posición de los robot a través de cámaras, de forma cooperativa se podría lograr fácilmente que ambos pudieran ver dónde está el otro y dónde se encuentran diferentes elementos del entorno que podrían cambiar dinámicamente de posición. También, considerando una idea de la sección pasada, si se colocara una cámara en el efector final, podríamos implementar las mediciones de la visión directamente en las tareas del robot. Cabe mencionar que OpenCV es una biblioteca escrita en C++ con implementación en Python, sin embargo, es más fácil y cómodo trabajar en este último, considerando que podemos armar fácilmente una interfaz gráfica que acompañe a las mediciones de la cámara.

Internet of Things

Una implementación en JavaScript a través de Node.js permitiría exponer a internet la comunicación con los robots, además de permitirnos consumir información. Desde comunicarnos con otros robots formando una red de trabajo que no necesite estar localizada físicamente en un mismo punto pero que pudiera ser

controlada desde una misma interfaz, teleoperación, consumo de modelos de inteligencia artificial, hasta cómputo descentralizado, conectar la planta a internet expande sus fronteras más allá de las paredes del laboratorio.

Gemelo digital

En tecnologías como Unity - motor para desarrollo de videojuegos – aunque no sea el objetivo con el que fue diseñada, se pueden simular sistemas físicos de forma visual e interactuar con ellos. Estos motores permiten definir colisiones y físicas que doten a un modelo 3D del realismo suficiente para representar un sistema real. Un modelo 3D de la planta, alimentado con un modelo matemático preciso y con leyes de control que gobiernen nuestra interacción con él, sería una herramienta de estudio muy valiosa.

En estas ideas se permite dejar volar la imaginación, aún faltaría analizar la viabilidad o utilidad final de ellas, sin embargo, como objetivo personal implícito en esta tesis, se espera contribuir como base a las futuras investigaciones que logren llegar más allá de estos bordes.

Referencias

[1] M. W. Spong, Robot Modeling and Control, Hoboken, Nueva Jersey, EE.UU.: John Wiley & Sons, Inc., 2005.

[2] B. Siciliano, Robotics: Modelling, Planning and Control, Londres, Reino Unido: Springer, 2009.

[3] A. Cavalcanti, Software Engineering for Robotics, Cham, Suiza: Springer, 2021.

[4] M. A. Arteaga, Local Stability and Ultimate Boundedness in the Control of Robot Manipulators, Springer, 2021.

[5] F. J. García Gutiérrez, Desarrollo y construcción de una tarjeta para la adquisición de datos y control de robots cooperativos, Ciudad de México, 2016.

[6] J. K. Liker, The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer, Nueva York, Estados Unidos: McGraw-Hill, 2004.

[7] B. Andersen y T. Fagerhaug, Root Cause Analysis: Simplified Tools and Techniques, Milwaukee, Estados Unidos: Quality Press, 2006.

[8] National Instruments Corp, «NI CompactRIO o el dispositivo de red no aparece o falta en MAX,» 5 Abril 2024. [En línea]. Available: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z000000P70YSAS&l=es-MX.

[9] National Instruments Corp, «Cómo arreglar o restablecer el archivo de la base de datos de NI MAX,» 31 Julio 2023. [En línea]. Available: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z000000P8awSAC&l=es-MX.

[10] National Instruments Corp, «Manually Resetting the NI Measurement & Automation Explorer Database,» 10 Marzo 2023. [En línea]. Available: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA03q000000YHcVCAW&l=es-MX.

[11] National Instruments Corp, «NI CompactRIO,» 5 Mayo 2018. [En línea]. Available: https://www.ni.com/es/support/downloads/drivers/download.ni-compactrio.html#306668.

[12] J. J. Craig, Robótica, Madrid, España: Pearson, 2006.

[13] M. Mihelj, Robotics, Cham, Suiza: Springer, 2019.

[14] A. Visioli, Practical PID Control, Londres, Reino Unido: Springer, 2006.

[15] H. Liang, «Qué es una DLL,» 19 Febrero 2024. [En línea]. Available: https://learn.microsoft.com/es-es/troubleshoot/windows-client/setup-upgrade-and-drivers/dynamic-link-library.

[16] Microsoft, «Documentación de Windows Forms,» [En línea]. Available: https://learn.microsoft.com/es-es/dotnet/desktop/winforms/?view=netdesktop-8.0.

[17] OpenCV Team, «Introduction,» [En línea]. Available: https://docs.opencv.org/4.10.0/d1/dfb/intro.html.

[18] CRS Robotics Corporation, C500C Controller User Guide, Ontario, Canada, Ontario, 2000.