Desarrollo de una interfaz compatible con el sistema de un robot paralelo tipo doble SCARA

Proyecto final de estudios

Ingeniería en Mecatrónica

Carole Huet

Director : Ing. Eric Sanchez

Facultad de Ingeniería,

Universidad Nacional de Cuyo,

Mendoza, Argentina

2023

Agradecimientos

En primer lugar, quiero agradecer a los profesores Eric Sanchez y Hernán Garrido por su grande apoyo durante el desarrollo de este proyecto.

Al director de la carrera mecatrónica por darme buenos consejos y ponerme en contacto con las personas adecuadas para que esta experiencia transcurra lo mejor posible.

A mi familia francesa y argentina, a mis amigos de toda parte por animarme siempre.

A mis compañeros de mecatrónica sin quien no hubiera tenido éxito.

A los estudiantes Gino Avanzini, Gonzalo Fernández y Jeremías Pino Demichelis por el desarrollo de este robot y su disponibilidad para proporcionarme detalles.

A la ENIB y a la UNCUYO por darme esta oportunidad de acceder a un segundo título.

Resumen

En este documento, se presenta el desarrollo de un interfaz usuario creado para completar el proyecto de robot paralelo de tipo doble SCARA desarrollado por estudiantes de la facultad de ingeniería de la Universidad nacional de CUYO. Este proyecto se desarrolla en el marco del proyecto final de estudios de la carrera Ingeniería en mecatrónica.

Este escrito trata del desarrollo de una simulación del modelo cinemático del mecanismo de cinco barras para entender su funcionamiento. En seguido, apoyado sobre este sistema paralelo, se desarrolla un sistema de control de posición con un cambio de motores por motores de corriente continua. En segundo lugar, se expone el desarrollo de la interfaz usuario, su análisis, su modelación y sus adaptaciones futuras. Para terminar, un análisis de simulaciones de transmisión de datos por puerto virtual está expuesto.

Como resultado, este proyecto permite de agregar una interfaz usuario al robot paralelo tipo doble SCARA y facilitar su uso por parte del usuario. Diferentes modos de funcionamiento pueden ser utilizados, respetando el formato de la trama configurada en el microcontrolador del nodo orquestador.

Palabras claves : Robot paralelo, simulación, interfaz usuario, UART, PyQt

Índice general

Agradecimientos 2

Resumen 3

Índice general 4

Índice de figuras 6

Vocabulario 8

Introducción 9

Capítulo 1 10

1. Estado del arte 10

2. Problemática 10

3. Objetivos 11

5. Organización del informe 12

Capítulo 2 13

1. Modelo geométrico 13

1. Modelo geométrico inverso 14

2. Modelo geométrico directo 15

3. Simulación del modelo geométrico 16

2. Proyecto de control y sistema 19

1. Sistema linealizado 20

2. Modelo dinámico 21

3. Agregado del sistema de control 22

Capítulo 3 23

1. Estado del arte 23

1. Arquitectura del sistema 24
2. El planificador de trayectorias 25
3. UART (universal asynchronous receiver-transmitter) 26
4. Trama UART 27
5. Control de flujo 28
6. Modos implementados 29
2 – Entorno y descargas 30
3 – Desarrollo de la interfaz 32
1. Widgets 33
2. Ventana principal 33
3. Barra de herramientas 34
4. Conexión al puerto serie 35
5. Control de flujo 37
6. Selección del tipo de trayectoria 38
Capítulo 4 44
1. Conexión virtual 44
2. Prueba de conexión UART 46
Conclusión 51
Trabajos futuros 52
Bibliografía 53

Índice de figuras

Figura 1 : Robot Doble SCARA paralelo o mecanismo de cinco barras construido 10
Figura 2 : División de las dos cadenas cinemáticas. Extraído de Khalil y Briot [2] 14
Figura 3 : Función del modelo geometrico inverso 17
Figura 4 : Función del modelo geométrico directo 17
Figura 5 : Simulación del robot tipo doble SCARA 18
Figura 6 : Trayectorias de la simulación de desplazamiento del robot en modelo inverso y directo 19
Figura 7 : Modelo abierto de la plata linealizada, en simulink 21
Figura 8 : Modelo cerrado de la plata linealizada, en simulink 22
Figura 9 : Simulación de la planta cerrada diseñando un circuló con perturbaciones y sistema de control 22
Figura 10 : Capas y jerarquía de la arquitectura del sistema 24
*Figura 11 : Diagrama de clases del planificador de trayectorias** 26
Figura 12 : Diagrama temporal indicando la secuencia de envıo de consignas por UART y el envıo de consignas sincronizadas a los extremos 30
Figura 13 : Diagrama que enumera las clases que heredan de la clase QObject, especialmente en el módulo Widgets 32
Figura 14 : Disposición definida de una ventana QMainWindow 33
Figura 15 : Definicion del widget Toolbar 34
Figura 16 : Extracto de la fonción Toolbar 35
Figura 17 : Herramienta «addWidget» para agregar un widget a la ventana 35
Figura 18 : Lanzado de conneción del puerto serie 36
Figura 19 : Conexión del puerto serie si está disponible 36
Figura 20 : lanzado de conexión de lectura y escritura 37
Figura 21 : Función de lectura con el control de flujo y función de escritura 37
Figura 22 : Ventana principal QMainWindow 38
Figura 23 : Configuración de un QPushButton para llamar el diálogo para diseñar la trayectoria en 2 puntos 39
Figura 24 Función de llamada del diálogo de creación de la trayectoria 39
Figura 25 : Creación de un widget de tipo spinbox 39
Figura 26 : Configuración de un widget botón de tipo QDialogButton y acceso a sus funciones asociadas 40
Figura 27 : Funciones asociadas al envió de datos a la ventana principal 40
Figura 28 : Ventana diálogo para diseñar una trayectoria con 2 puntos 40

Figura 29 : Foncion llamada a la recuperacion de los datos entrados en la ventana de dialogo y implementacion de una ventana QMessageBox para mostrar los datos recibidos en la lista ____ *41*
Figura 30 : Ventana creída al recibir la lista de datos ingresados en el diálogo ____ *41*
Figura 31 : Lectura y uso de los datos ingresados para calcular la trayectoria ____ *42*
Figura 32 : Función de conversión de los datos en hexadecimal y creación y envió de la trama ____ *42*
Figura 33 : Ejemplo de trama enviada por puerto serie ____ *42*
Figura 34 : Ventana diálogo del modo PP con menú desplegable para la selección del comando ____ *43*
Figura 35 : Administrador de dispositivos, puerto COM ____ *44*
Figura 36 : Interfaz del software Virtual Serial Port Driver, par de los COM 1 y 2 ____ *45*
Figura 37 : Interfaz del software PuTTY, configurado para conectarse en serial a 921600 baudios ____ *45*
Figura 38 : Prueba de envío de datos entre el puerto COM1 y el puerto COM2 ____ *46*
Figura 39 : Conexión y lectura del puerto serie ____ *47*
Figura 40 : Recuperación de los datos transmitidos en las tramas enviado sobre la conexión UART ____ *48*
Figura 41 : Simulaciones de la interfaz y de Matlab sobre una misma trayectoria ____ *50*

Vocabulario

(1) Setpoint : el punto de ajuste o configuración que se utiliza como referencia

(2) Framework : palabra inglesa que significa marco o entorno de trabajo

(3) Widget : palabra inglesa que significa elementos de la interfaz de usuario

(4) Parenting system : palabras inglesas que significan el sistema de jerarquía

(5) True : palabra inglesa que significa verdadero

(6) Double, integer, string, array : palabras inglesas usadas en la programación para determinar los tipos de los datos.

Introducción

La robótica reúne diferentes campos de la ingeniería, la mecánica, la electrónica y la informática. Basado en el funcionamiento de un robot paralelo, este proyecto de interfaz usuario es totalmente en adecuación con la carrera mecatrónica y permite de aplicar los varios dominios estudiados durante la carrera.

El lanzamiento de este proyecto fue iniciado por estudiantes de la facultad de ingeniería de la Universidad Nacional de CUYO, en el contexto de su proyecto final de estudio. Animados por la importancia que toman los robots paralelos en las investigaciones científicas tanto teóricas como prácticas, eligieron diseñar un robot con un mecanismo de cinco barras [Fig. 1]. Este tipo de robot son diseñados en cinemática cerrada, tienen muy buena precisión y velocidad en comparación con un robot serie de tamaño y peso similar. Su estructura paralela permite diseños más rígidos con un desempeño dinámico superior, una disminución de la masa de los eslabones y un uso de actuadores de menor capacidad. Además de su excelente relación carga/peso, estos robots tienen una tolerancia sobre los errores de posición del efector final mayor que los de los robots serie. Por otro lado, los robots paralelos tienen un sistema de control más complejo por su cinemática cerrada. El espacio de trabajo que puede cubrir es más reducido y complejo. De hecho, su sistema tiene singularidades de tipo 2 además de las singulares de tipo 1, encontrados también en los sistemas de robot serie. Es decir que además de perder la habilidad a moverse a lo largo de una o más direcciones, el sistema paralelo tiene también movimientos incontrolables y se vuele inestable.

A lo largo de este trabajo, se estudió el desarrollo de una simulación de la cinemática directa e inversa del robot de cinco barras y 2 grados de libertad. Se agregó a este sistema motores de corriente continua para aplicar un control de posición angular sobre estos motores. Además, en una dinámica de seguir el proyecto desarrollado, se implementa una interfaz usuario codificado en Python para simplificar la transmisión de datos al robot.

Capítulo 1

1. Estado del arte

Estudiando las investigaciones del repositorio del proyecto*, el informe escrito por los estudiantes [1] y de las obras científicas, en particular la de Khalil y Briot [2] y la de Bourbonnais, Bigras y Bonev [3], se desarrolló una simulación del modelo geométrico del robot y una interfaz usuario adaptada al medio de transmisión UART usada para comunicar la trayectoria al robot. [Fig. 1]

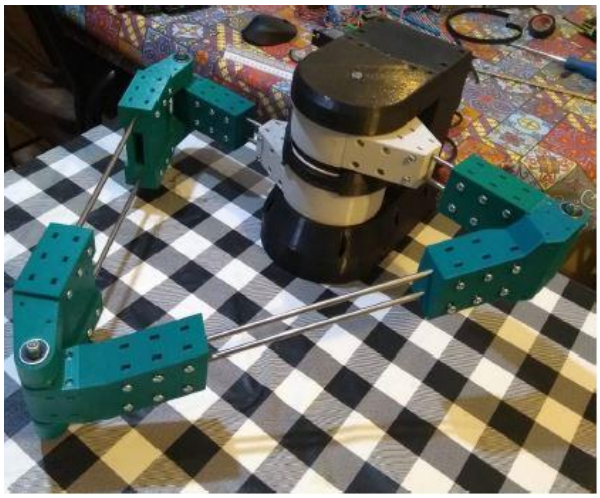

Figura 1 : Robot Doble SCARA paralelo o mecanismo de cinco barras construido

2. Problemática

Como estudiante en intercambio a la facultad de ingeniería de la UNCUYO, pude acceder a un aspecto de la robótica diferente de el que conocí de mis estudios en Francia. Con este proyecto de robot paralelo, descubrí un funciono mecánico y un diseño completo, con la necesidad de entender cada parte del desarrollo para seguir con mejoras.

* Repositorio del desarrollo del Robot : https://github.com/gonzafernan/five-bar-robot

Uno de los desarrollos que no pudieron llevar a cabo en su proyecto es el desarrollo de una interfaz usuario para comunicarse con el robot. De hecho, los estudiantes desarrollaron un programa para realizar los cálculos necesarios para pasar de las coordenadas del efector final a la posición de los motores paso a paso. Estos datos son puestos en el formato de la trama elegida y guardados en un archivo .csv para enviarlos. Este sistema, aunque sea funcional, no es el más adecuado porque requiere cierta manipulación por parte del usuario.

Por otra parte, habiendo ingresado a la facultad de ingeniería de UNCUYO en un contexto de doble titulación, no pude realizar el trabajo de fin de estudios allí por cuestiones de tiempo. Por tanto, este proyecto se lleva a cabo en un contexto adecuado para poder realizarse de forma remota. Este contexto implica, por tanto, que el acceso físico al robot no será posible y que todas las validaciones se realizarán virtualmente mediante simulaciones.

3. Objetivos

En primer lugar, se estudió el funcionamiento global del sistema. Después, era necesario trabajar sobre la parte cinemática y dinámica del sistema para entender los datos que actúan sobre los motores y los eslabones. Luego, se establecieron ecuaciones linealizadas para determinar los torques necesarios para los motores. Además, se presenta el sistema a lazo abierto y a lazo cerrado de la planta.

Por otra parte, se desarrolla una interfaz con pyQt, un módulo libre que le permite vincular el lenguaje Python con la biblioteca Qt, una descripción más detallada de este módulo podrá ser encontrada en la sección 3.3. Esta interfaz permite al usuario introducir los puntos de la trayectoria que quiere hacer recorrer por el efector final, en la zona de trabajo. Para transmitir las coordenadas elegidas por el usuario, se envía una trama, detallada en la sección 3.1.4, al microcontrolador del nodo orquestador por puerto serie. Esta trama manda a los dos nodos secundarios la posición angular que deben tomar los motores. Estos ángulos son calculados en el software en el modo de posición interpolada, en cuanto al modo PP, solo se envía una posición objetivo. Para permitir la comunicación con el microcontrolador, una configuración

de puerto serie debe ser desarrollada en la interfaz usuario, con los diferentes parámetros necesarios en una trama UART (baudios, cantidad de bit, bit stop, bit de paridad, control de flujo,...).

4. Herramientas

El desarrollo de este proyecto se divide en dos grandes partes. La primera esta desarrollada con el firmware MATLAB con su entorno simulink. Para la segunda parte, es necesario descargar Python y PyQt. Las versiones descargadas son la tercera por Python (python3) y la quinta por PyQt (PyQt5).

El proyecto fue desarrollado en el sistema operativo Windows, pero es compatible con Linux o MacOS entre otros. Además, la biblioteca PyQt es mundialmente usada y más en América del sur y del norte y en Europa.

Además, realizando el desarrollo de la interfaz sin acceso al robot físico, los softwares Virtual Serial Port Driver y PuTTY me permitieron respectivamente crear una conexión entre dos puertos COM en la computadora y tener una interfaz para ver los mensajes recibidos.

Por otra parte, se creó un GitHub* para dejar accesible los archivos de esta porción del proyecto de robot paralelo tipo doble SCARA. La elección del repositorio está en continuidad del primero, también accesible en GitHub**.

5. Organización del informe

Por cuestiones de organización el desarrollo del trabajo se dividió en 3 partes

- El desarrollo de una simulación del modelo geométrico del robot paralelo tipo doble SCARA.
- El diseño de una interfaz usuario para facilitar el uso del robot paralelo.
- Las pruebas de conexión entre la interfaz usuario y la simulación del robot.

* Repositorio del desarrollo de la interfaz : https://github.com/CaroleHuet/dual-scara-interfaz

** Repositorio del desarrollo del Robot : https://github.com/gonzafernan/five-bar-robot

Capítulo 2

En el inicio de un proyecto se definen objetivos y se recolecta la información y recursos para lograrlos. En este proyecto, teniendo el objetivo de agregar una interfaz usuario al robot tipo doble SCARA, sin acceso físico al mismo, el desarrollo de una simulación de su funcionamiento se vuelve necesario.

Para realizar esta simulación, en primer lugar, se estudiaron las ecuaciones cinemáticas directas e inversas del robot. Con estas ecuaciones, se desarrolló un programa y una simulación para verificar el buen funcionamiento. Esta simulación permite luego tener una aplicación concreta para verificar si las tramas enviadas por UART transmiten las coordenadas a los motores. Además, este programa también permitió tener una base para poder iniciar el Proyecto final de Control y Sistemas. De hecho, siguiendo los consejos de mi profesor, decidí realizar el proyecto sobre un mecanismo real y no sobre una planta ficticia, por lo que dicho proyecto terminó siendo más complejo debido a la mecánica del robot en cuestión. En parte, la complejidad se debe a que estos mecanismos no se tratan en la materia Control y Sistemas, y a que no realicé el cursado de Robótica de la facultad.

1. Modelo geométrico

El modelo geométrico permite establecer la relación entre la configuración del robot y las coordenadas cartesianas de su extremo.

El desarrollo siguiente se basa en el trabajo previo de los estudiantes que fabricaron el prototipo, y se puede consultar en [1].

El sistema geométrico de un robot paralelo con un mecanismo de 5 barras tiene una cadena cinemática cerrada. Para realizar el estudio cinemático de este sistema, se divide la cadena cinemática en dos partes, de la base hasta el efector final. El efector final corresponde al nodo A13 de la cadena cinemática 1 [Fig. 2]. Esta articulación, con las articulaciones A12 y A22 son

las articulaciones pasivas. De hecho, estos nodos se mueven dependiendo de las articulaciones activas A11 y A21, que son las articulaciones motorizadas. Además, los nodos están conectados por eslabones denominados : d_{i1}, d_{i2}, d_{i3}.

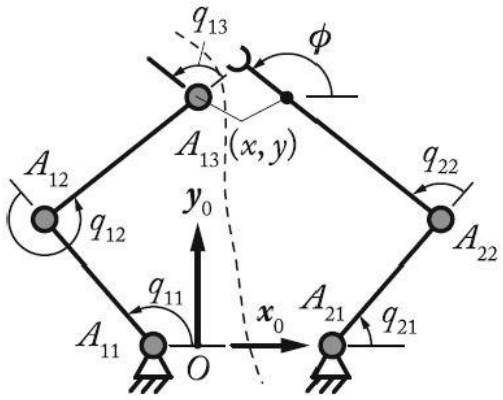

Figura 2 : División de las dos cadenas cinemáticas. Extraído de Khalil y Briot [2]

1. Modelo geométrico inverso

Primero, se pone el interés sobre el modelo geométrico inverso del sistema. Este modelo inverso permite de encontrar las coordenadas angulares de los nodos activos para una posición específica del efector final en el espacio de trabajo.

Para determinar estas coordenadas articulares se calcula el vector de coordenadas del efector final desde la base como si fuera una cadena abierta. Obteniendo una relación implícita entre las coordenadas del efector final y las coordenadas angulares, se define una ecuación matricial (1.1).

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} d_{i1} + d_{i3}\cos(q_{i1} + q_{i2}) + d_{i2}\cos(q_{i1}) \\ d_{i3}\sin(q_{i1} + q_{i2}) + d_{i2}\sin(q_{i1}) \end{bmatrix} \tag{1.1}$$

De esta ecuación, se eliminan las articulaciones pasivas para reducir la complejidad del sistema. De esta forma, las coordenadas angulares de los nodos activos quedan directamente vinculadas con las del efector final (1.2).

$$h_p(\mathbf{x}, q_a) = \begin{bmatrix} (x - d_{11} - d_{12}\cos(q_{11}))^2 + (y - d_{12}\sin(q_{11}))^2 - d_{13}^2 \\ (x - d_{21} - d_{22}\cos(q_{21}))^2 + (y - d_{22}\sin(q_{21}))^2 - d_{23}^2 \end{bmatrix} = 0 \tag{1.2}$$

Resolviendo el sistema de ecuaciones, se obtiene el modelo geométrico inverso que permite de determinar los valores de las articulaciones activas (1.3).

$$q_{i1} = 2\arctan\frac{-b_i \pm \sqrt{b_i^2 - c_i^2 + a_i^2}}{c_i - a_i} \tag{1.3}$$

Para $i = 1, 2$. Donde: $a_i = -2d_{i2}(x - d_{i1})$, $b_i = -2d_{i2}y$, $c_i = (x - d_{i1})^2 + y_2 + d_{i2}^2 - d_{i3}^2$.

Una vez obtenidos los valores de las articulaciones activas, se pueden obtener las articulaciones pasivas (1.4)(1.5) mediante la ecuación matricial (1.1).

$$q_{i2} = \text{atan2}\frac{y - d_{i2}\sin(q_{i1})}{x - d_{i1} - d_{i2}\cos(q_{i1})} - q_{i1} \tag{1.4}$$

$$q_{13} = q_{21} + q_{22} - q_{11} - q_{12} \tag{1.5}$$

2. Modelo geométrico directo

Por su parte, el modelo geométrico directo permite de determinar las coordenadas del efector final a partir de las coordenadas angulares de las articulaciones activas.

Para el caso en estudio, utilizando la ecuación 1.2, se puede operar algebraicamente para obtener la siguiente expresión:

$$h_p(\bar{x}, \bar{q_a}) = \begin{bmatrix} x^2 + y^2 + a_1x + b_1y + c_1 \\ x^2 + y^2 + a_2x + b_2y + c_1 \end{bmatrix} = 0 \tag{2.1}$$

Donde, para i = 1, 2 :

$$\begin{aligned} \boldsymbol{a}_i &= -2\left(d_{i1} + d_{i2}cos\left(q_{i1}\right)\right) \\ \boldsymbol{b}_i &= -2d_{i2}sin\left(q_{i1}\right) \\ \boldsymbol{c}_i &= \left(d_{i1} + d_{i2}cos\left(q_{i1}\right)\right)^2 + d_{i2}^2sin\left(q_{i1}\right) - d_{i3}^2 \end{aligned}$$

Resolviendo la ecuación 2.1, se obtiene el modelo geométrico directo del sistema. Este modelo se divide en varias ecuaciones dependiendo de b_1 y b_2, siendo $b_i = -2*d_{i2}*\sin(q_{i1})$.

Para el caso en que b1 ≠ b2 :

$$x = -\frac{-f_2 \pm \sqrt{f_2^2 - 4f_1 f_3}}{2f_1} \quad ; \quad y = e_1 x + e_2 \tag{2.1}$$

Donde :

$$\begin{aligned} f_1 &= 1 + e_1^2 \\ f_2 &= 2e_1 e_2 + a_1 + b_1 e_1 \\ f_3 &= e_2^2 + b_1 e_2 + c_1 \\ e_1 &= -\frac{d_1}{d_2} \\ e_2 &= -\frac{d_3}{d_2} \\ d_1 &= a_2 - a_1 \\ d_2 &= b_2 - b_1 \\ d_3 &= c_2 - c_1 \end{aligned}$$

Para el caso en que b1 = b2 :

$$x = -\frac{c_2 - c_1}{a_2 - a_1} \quad ; \quad y = \frac{-b_1 \pm \sqrt{b_1^2 - 4(x^2 + a_1 x + c_1)}}{2} \tag{2.2}$$

Donde a_i, b_i y c_i se definen en la simplificación de la ecuación 2.1.

3. Simulación del modelo geométrico

El modelado del modelo geométrico del robot se realizó en MATLAB. Primero se agregó una función que define el modelo geométrico inverso del sistema. La función recibe las coordenadas x e y del efector final y el tamaño de cada eslabón, siendo d_{i1} = 0m, d_{i2} = 0.25m y d_{i3} = 0.38m. Con estos datos, se calculan los valores de cada nodo para cada posición del efector final [Fig. 3].

```
function [q11, q21, q12, q22] = q_from_xy(x, y, di1, di2, di3)

    ai = -2*di2*(x - di1)
    bi = -2*di2*y
    ci = (x - di1)^2 + y^2 + di2^2 - di3^2

    q11 = 2*atan((-bi + sqrt(bi^2 - ci^2 + ai^2))/(ci-ai));
    q21 = 2*atan((-bi - sqrt(bi^2 - ci^2 + ai^2))/(ci-ai));

    q12 = atan2(((y - di2*sin(q11))),(x - di1 - di2*cos(q11))) - q11;
    q22 = atan2(((y - di2*sin(q21))),(x - di1 - di2*cos(q21))) - q21;

    q13 = q21 + q22 - q11 - q12;

end
```

Figura 3 : Función del modelo geometrico inverso

De la misma manera se desarrolló el modelo geométrico directo. La función toma los valores de las coordenadas angulares de los nodos activos y el tamaño de los eslabones para calcular las coordenadas del efector final [Fig. 4].

```
function [x, y] = xy_from_q(q11, q21, di1, di2, di3, xa, ya)

    a1 = -2*(di1 + di2 * cos(q11));
    b1 = -2*di2*sin(q11)
    c1 = (di1 + di2*cos(q11))^2 + (di2*sin(q11))^2 - di3^2;

    a2 = -2*(di1 + di2 * cos(q21));
    b2 = -2*di2*sin(q21)
    c2 = (di1 + di2*cos(q21))^2 + (di2*sin(q21))^2 - di3^2;

    d1 = a2 - a1;
    d2 = b2 - b1;
    d3 = c2 - c1;
    e1 = -d1/d2;
    e2 = -d3/d2;
    f1 = 1 + e1^2;
    f2 = 2*e1*e2 + a1 + b1*e1;
    f3 = e2^2 + b1*e2 + c1;

    if b1 == b2
        if ya < 0
            x = -((c2 - c1)/(a2 - a1));
            y = (-b1 - sqrt(b1^2 - 4*(x^2 + a1*x + c1)))/ 2 ;
        else
            x = -((c2 - c1)/(a2 - a1));
            y = (-b1 + sqrt(b1^2 - 4*(x^2 + a1*x + c1)))/ 2 ;
        end
    else
        if xa < 0
            x = (-f2 - sqrt(f2^2 - 4*f1*f3))/ (2*f1);   % val de x_t negativas
            y = e1*x + e2;
        else
            x = (-f2 + sqrt(f2^2 - 4*f1*f3))/ (2*f1);   % val de x_t positivas
            y = e1*x + e2;
        end
    end
end
```

Figura 4 : Función del modelo geométrico directo

Para comprobar la validez de estas funciones se graficó en dos ejes, abscisa y ordenada, y una trayectoria en el espacio de trabajo. Con un bucle, se determinaron los valores de cada articulación para cada punto de la trayectoria y se registró cada valor en matrices. Una matriz con los dos nodos activos, y una con los dos nodos pasivos. Con estos valores, las medidas de los eslabones y las coordenadas del efector final, se realiza una simulación para verificar si las coordenadas angulares encontradas corresponden a la posición del efector final. En la figura 5, se verificó que la posición del efector final, indicada con un pequeño círculo azul, corresponde con la posición de los eslabones, es decir con los valores angulares.

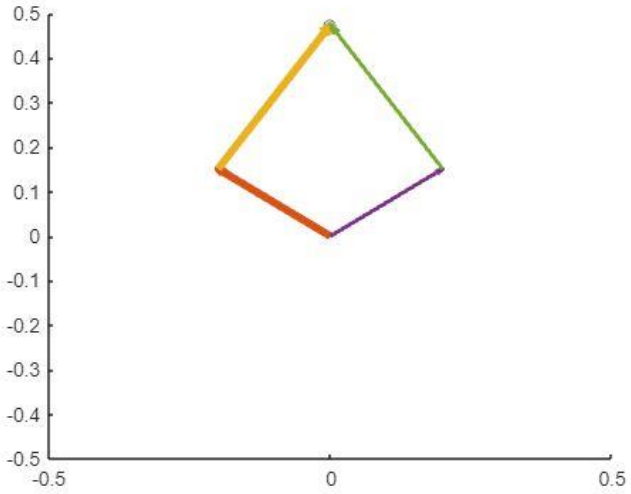

Figura 5 : Simulación del robot tipo doble SCARA

En segundo lugar, se verifica el modelo directo utilizando los valores de las posiciones angulares encontradas con el modelo inverso. Con estos valores se puede verificar que utilizando el modelo directo o inverso el sistema realiza las mismas trayectorias.

Por ejemplo, se desea diseñar una trayectoria rectangular en el espacio de trabajo. Por eso, es posible utilizar las coordenadas de los puntos definiendo la trayectoria en el modelo inverso para determinar la posición angular de los dos nodos activos, y así recorrer la trayectoria rectangular. O de la misma manera, se puede usar las coordenadas angulares para conocer las coordenadas de los puntos definiendo la trayectoria, y asimismo, recorrer la trayectoria

rectangular [Fig. 6]. Con este ejemplo, se puede confirmar que utilizando las coordenadas articulares o las coordenadas cartesianas, es decir utilizando el modelo directo o inverso, las trayectorias recogidas son las mismas. Así que la implementación de los modelos en la simulación de trayectoria en Matlab es verificada.

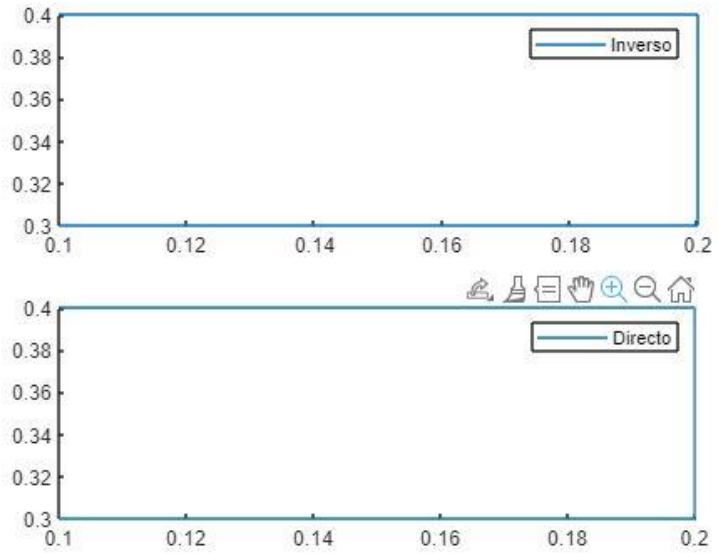

Figura 6 : Trayectorias de la simulación de desplazamiento del robot en modelo inverso y directo

Por último, es importante notar que utilizando el modelo directo, existen diferentes ecuaciones permiten encontrar las coordenadas del efector final. Además de depender de b_i, la elección de las ecuaciones depende de la zona por cual se encuentra el efector final, positivos o negativos en abscisa u ordenada.

2. Proyecto de control y sistema

El robot paralelo de tipo doble SCARA es un modelo muy interesante para aplicar un proyecto sobre el control de una planta. De hecho, un seguimiento de la posición en tiempo real de los motores sería adecuado y permitiría asegurar la posición del efector final. Para la realización de este estudio, se eligió usar motores con corriente continua y no los motores de paso a paso para tener la noción de continuidad en el modelo.

A continuación se resumen los pasos más importantes de la investigación realizada hasta el momento. Primero se linealizó la planta en un punto de equilibrio, después se implementó el modelo dinámico de esta planta. Por último, se agregó un control de posición y se comprobó su eficacia con una trayectoria.

1. Sistema linealizado

En un proceso de simplificación del problema para realizar el proyecto final de la materia de Control y sistemas, se linealizó la planta y se ignoraron las perturbaciones menos impactantes. De esta manera, se tomó el punto de equilibrio [0.00 , 0.38], punto en el medio del espacio de trabajo, y se linealizaron los desplazamientos de los nodos activos dependiendo de x e y alrededor de este punto.

Además, utilizando la linealización de una función multivariable definida, como se puede ver en [4]:

The equation for the linearization of a function $f(x, y)$ at a point $p(a, b)$ is:

$$f(x,y) \approx f(a,b) + \left.\frac{\partial f(x,y)}{\partial x}\right|_{a,b} (x-a) + \left.\frac{\partial f(x,y)}{\partial y}\right|_{a,b} (y-b)$$

The general equation for the linearization of a multivariable function $f(\mathbf{x})$ at a point $\mathbf{p}$ is:

$$f(\mathbf{x}) \approx f(\mathbf{p}) + \nabla f|_{\mathbf{p}} \cdot (\mathbf{x} - \mathbf{p})$$

where $\mathbf{x}$ is the vector of variables, ∇f is the gradient, and $\mathbf{p}$ is the linearization point of interest .[2]

Se encuentra un sistema linealizado de forma :

$$\begin{bmatrix} q_{11} - 2{,}8064 \\ q_{21} - 0{,}3352 \end{bmatrix} = \begin{bmatrix} \frac{-5}{13} & \frac{-10}{33} \\ \frac{-5}{13} & \frac{-10}{33} \end{bmatrix} \begin{bmatrix} x \\ y - 0{,}38 \end{bmatrix} \tag{1.1}$$

2. Modelo dinámico

El paso siguiente es de determinar el modelo dinámico lineal del sistema. Desde el modelo linealizado de la planta, teniendo en cuento solo la masa del efector final y los torques de fricción viscosa, se obtiene un modelo de esta forma :

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \ddot{x}\\ \ddot{y}\end{bmatrix} = \begin{bmatrix}0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & \frac{-c}{m} & 0\\ 0 & 0 & 0 & \frac{-c}{m}\end{bmatrix}\begin{bmatrix}x\\ y\\ \dot{x}\\ \dot{y}\end{bmatrix} + \begin{bmatrix}0 & 0\\ 0 & 0\\ \frac{1}{m} & 0\\ 0 & \frac{1}{m}\end{bmatrix}\begin{bmatrix}F_{dx}\\ F_{dy}\end{bmatrix} + \begin{bmatrix}0 & 0\\ 0 & 0\\ \frac{-5}{13m} & \frac{-5}{13m}\\ \frac{-10}{33m} & \frac{10}{33m}\end{bmatrix}\begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix} \tag{1.2}$$

Con este modelo, se desarrolló un diagrama a lazo abierto en simulink, siendo m = 0.272g y c = 6.75 N m/s. Este modelo permitió confirmar el funcionamiento esperado de la planta con pruebas en contextos conocidos.

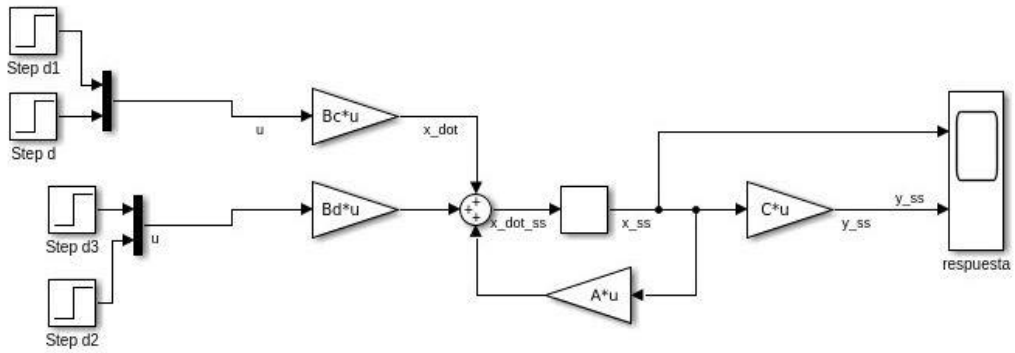

Figura 7 : Modelo abierto de la plata linealizada, en simulink

Donde :

```
%% State space system
A = [0 0 1 0
     0 0 0 1
     0 0 -c/m 0
     0 0 0 -c/m];

Bd = [0 0
     0 0
     1/m 0
     0 1/m];

Bc = [0 0
     0 0
     -5/(13*m) -5/(13*m)
     -10/(33*m) 10/(33*m)];

C = [1 0 0 0
     0 1 0 0];
```

3. Agregado del sistema de control

Una vez que el sistema a lazo abierto fue probado, se cierra el lazo para implementar un control a la planta.

Se eligió agregar una matriz de ganancia K y un coeficiente kr para realizar un control sobre las posiciones angulares de los motores. El sistema con el lazo cerrado aparece de la forma siguiente :

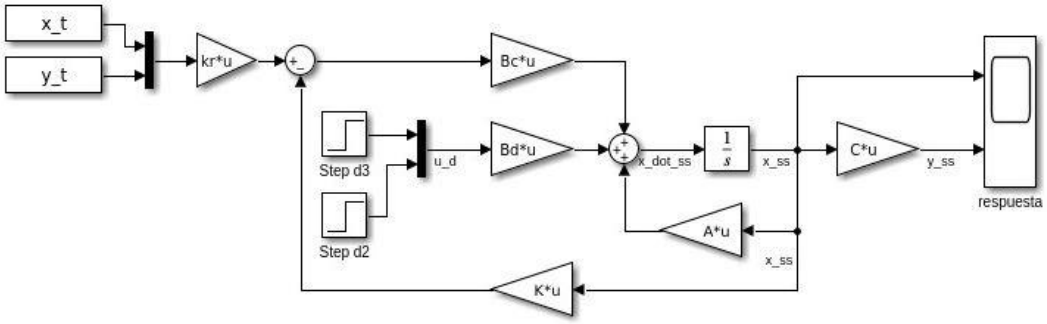

Figura 8 : Modelo cerrado de la plata linealizada, en simulink

En Matlab, con la función « lqr » se determina la matriz de ganancia K. Además, se implementó una trayectoria circular como referencia, siendo x = r*cos(omega*t)-0.05 e y=r*sin(omega*t). Asimismo, agregando un escalón sobre las perturbaciones, se observa que el sistema sigue la trayectoria esperada [Fig. 9].

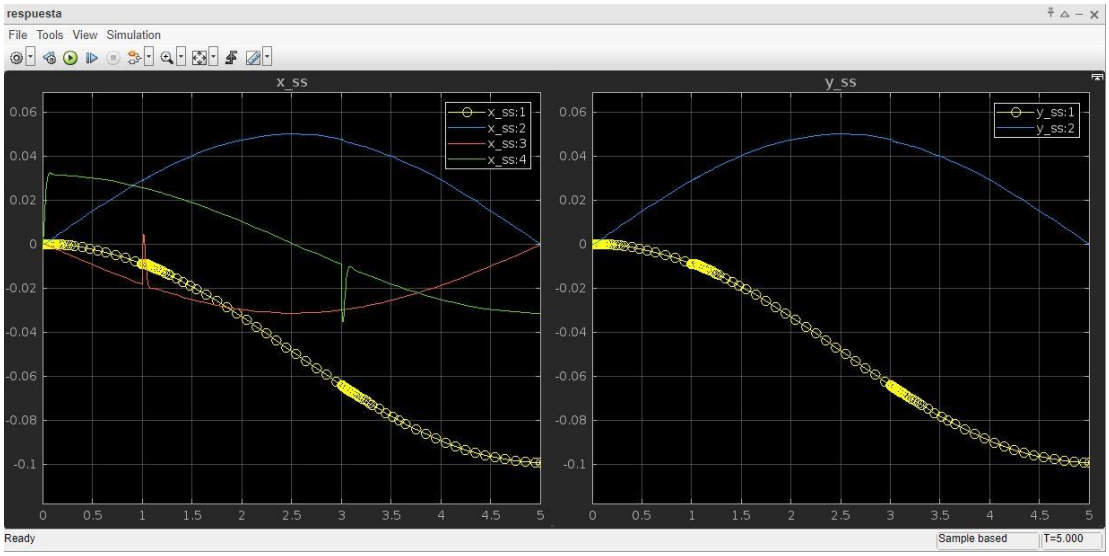

Figura 9 : Simulación de la planta cerrada diseñando un circuló con perturbaciones y sistema de control

Capítulo 3

Hoy el uso de las interfaces es muy variado. Las pantallas son parte de nuestra vida diaria ya sea en el ámbito privado o en el trabajo y responden a una demanda creciente.

En el ámbito de la industria, las pantallas han estado presentes en el mundo industrial durante décadas, pero su forma sigue cambiando. De hecho, las interfaces numéricas y las computadoras aparecieron durante los años 70 en entornos industriales. Y durante los 30 años siguientes, sus evoluciones fueron deslumbrantes, las pantallas permiten a los usuarios ver los procesos de manera más intuitivas. Ahora, con una mayor conectividad, con los sensores inteligentes y la recopilación de datos en tiempo real, un seguimiento más preciso y decisiones más rápidas son posibles.

Por su parte, los robots industriales aparecieron en el medio del siglo pasado y de la misma manera, conocieron grandes progresos durante los 30 años que siguieron los 70. Después, el siglo XXI se dedicó más a la colaboración con los humanos y la integración de inteligencia artificial para permitir la toma de decisiones.

De hecho, la colaboración con los robots es posible en parte a través del enlace de interfaces usuario y son un eslabón clave de la industria en un ámbito de productividad y eficiencia.

1. Estado del arte

Antes de iniciar el desarrollo de la interfaz usuario, se realizó una lista de los recursos del proyecto, los modos de funcionamientos y las características ya implementadas. [1]

1. Arquitectura del sistema

La parte de la arquitectura, desarrollada entre el usuario y el robot es una interfaz PC-microcontrolador. El protocolo de comunicación elegido entre los dos fue una UART. Esta elección proviene del hecho de que es posible utilizar como nivel superior de control cualquier dispositivo que pueda ejecutar Python y tenga un puerto serie o USB. Del lado de la capa de aplicación, desarrollaron un planificador de trayectorias. Es un programa escrito en Python que genera la secuencia de puntos que siguen los actuadores, en función de la trayectoria solicitada por el usuario, se encuentra más detalles en la sección 3.3.6.1. Por último, la comunicación se realizó a través de una interfaz utilizando un archivo .csv a donde se guardan los datos a enviar a cada actuador. El nodo orquestador, o primero, es el encargado de comunicarse con el software de planificación de trayectoria, solicitar el setpoint (1) o la secuencia de puntos que deben cumplir los nodos secundarios, o extremo. El nodo primero se asegura también de la sincronización y el cumplimiento en términos de tiempo [Fig 10].

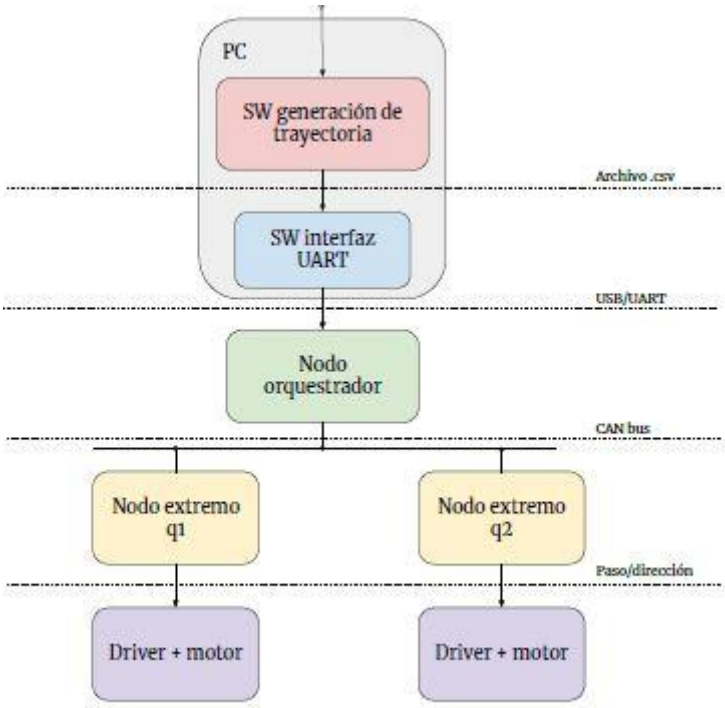

Figura 10 : Capas y jerarquía de la arquitectura del sistema

2. El planificador de trayectorias

Esta parte del software se encarga de generar las trayectorias y resolver la cinemática del robot. Para su implementación, se decidió que el programa no genere una trayectoria en tiempo real. Este solo ofrece una interacción mediante su API y se le facilita al usuario en forma de librería.

Por otra parte, se decidió usar el paradigma de la programación orientada a objectos, lo cual llevó a organizar la librería en las siguientes clases [Fig. 11] :

- FiveBar : en esta clase se encuentra el modelo del robot. Es decir, la solución de la cinemática directa (CD) e inversa (CI)*, los parámetros geométricos, los límites articulares del robot, la interferencia mecánica y métodos para graficar el robot y su espacio de trabajo.

* El método elegido para resolver CD y CI fue el propuesto en Bourbonnais, Bigras y Bonev [3] debido a que no hay necesidad de obtener la posicion, velocidad ni aceleracion de las articulaciones pasivas.

- TimeLaw : esta clase se implementan las leyes de tiempo trapezoidal y polinómica de grado 5*

- Path : en esta clase se implementan los caminos geométricos, estos pueden ser líneas, círculos, arcos de circunferencia, movimientos relativos al efector final y movimientos en el espacio articular*. También provee una función para graficar las trayectorias.

* Como se explicó en la sección 1.5.2 del informe [1]

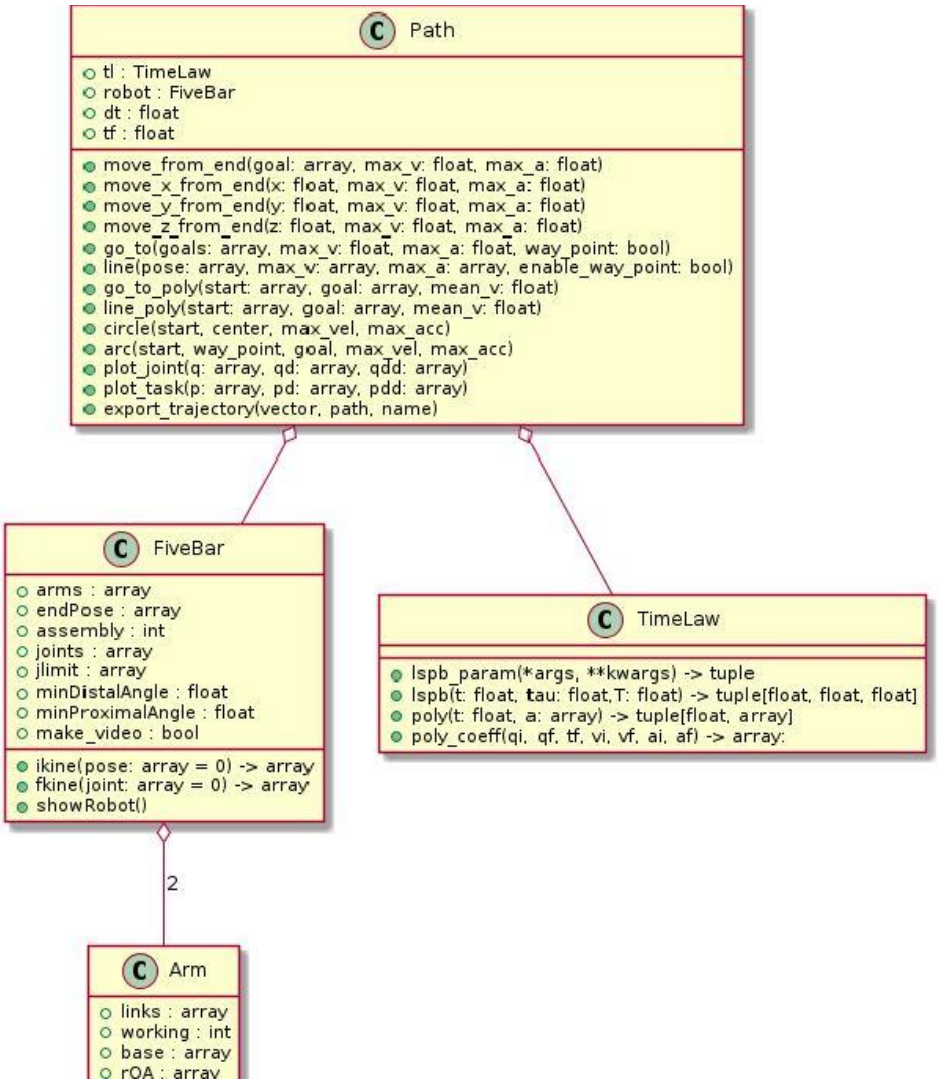

*Figura 11 : Diagrama de clases del planificador de trayectorias**

3. UART (universal asynchronous receiver-transmitter)

La configuración de la UART utilizada es de 921600 baudios, 8 bits por palabra, sin bit de paridad, 1 bit de stop y control de flujo por software. La conexión entre la computadora y el nodo orquestador se hace por USB, utilizando un puente USB-UART basado en el circuito integrado CP2102.

* El software se encuentra disponible en Github: https://github.com/gonzafernan/five-bar-robot

4. Trama UART

Los datos guardados en el archivo .csv tienen una forma de trama bien definida para ser enviado por el puerto UART y permitir la lectura por el nodo orquestador. Esta trama contiene la información necesaria para ejecutar la trayectoria pedida por el usuario, es decir identificador de comando y las coordenadas a enviar a los nodos secundarios. Cuatro comandos son definidos, de cambio de modo de trabajo, de envió de consignas en modo PP, de consignas en modo IP o de cambio de velocidad en modo PP [Tab. 1]. La trama estándar es ASCII y está compuesta por 2+(GDL-1) campos, el primero es el identificador de comando y los siguientes son los campos de datos. Los campos de datos son definidos para cada grado de libertad que se desea controlar, acá tenemos dos. Además, cada trama empieza con un carácter « : » , se termina con un retorno de carro y una nueva línea « \r\n » y entre cada datos se agrega una coma « , ». Así que cada trama tiene este formato :

: X, YYYYYYYY, ZZZZZZZZ\r\n

Donde « X » es el identificador de comando e « YYYYYYYY » y « ZZZZZZZZ » los campos de datos , uno para cada nodo extremo. La trama definida tiene dos campos de datos para los dos grados de control aplicados sobre el sistema, pero la trama es extensible. En consecuencia, la longitud de la trama está predeterminada en función de la cantidad de grados de libertad, de forma que l_{trama} = 4+9*GDL. Cada miembro del campo de datos tiene 8 caracteres ASCII que corresponde a la codificación hexadecimal de números enteros con signo de 32 bits. Dos razones llevaron a esta elección. Primera, un método de control de flujo por software es utilizado, este sistema emplea símbolos determinados (0x11 y 0x13) y pueden aparecer en cualquier momento del envió de la trama, se encuentra más detalles en la sección 3.1.5. Así que una trama binaria no correspondía al contexto. En segundo, el uso de trama de longitud fija facilita el procesamiento en el nodo orquestador. Además, se sabe que la mayoría de las entradas del diccionario de objectos son enteros de 32 bits o pueden ser convertidos a ese tipo fácilmente y también que esta combinación de características es un buen balance entre longitud de la trama, esfuerzo de implementación y flexibilidad para futuras características.

Identífi-cador	Significado	Campo de datos	Ejemplo
M	Cambio de modo de trabajo	Modo de trabajo para cada nodo, siguiendo la convención de la entrada ModesOfOperation (6060h) del OD int32_t	:M,00000007, 00000007\r\n
P	Nueva consigna en modo de posicionamiento por perfil (PP)	Consigna de posición absoluta a cada nodo, en incrementos del motor. Implementa la entrada NewTargetPosition (607Ah) del OD. int32_t	:P,00000AA8,FFFFF060\r\n
S	Nuevo punto de interpolación en el modo de posición interpolada (IP)	Consigna de posición interpolada a cada nodo, en incrementos del motor. Implementa la entrada InterpolationDataRecord del OD int32_t	:S,FFFFFFBA0,00000D4A\r\n
V	Seteo de velocidad máxima para el modo de posicionamiento por perfil	Valor de velocidad máxima en el perfil de velocidad del modo PP. Se expresa en pulsos por segundo. int32_t	:V,000005DC,00000321\r\n

Tabla 1 : Tipos de identificadores en la trama UART

5. Control de flujo

Como se mencionó antes, un control de flujo es necesario en este sistema. De hecho, la comunicación se hace desde un emisor « rápido » (la computadora) a un receptor « lento » (el microcontrolador). Como la computadora puede enviar los datos mucho más rápidamente que lo que puede procesar el microcontrolador o que lo que puede enviar por CAN [Fig.10], y como este receptor tiene un buffer limitado, se utiliza la técnica llamada control de flujo. De esta manera, el microcontrolador puede controlar la tasa de transmisión y realizar una

comunicación sin sobrecargar. Entonces, se eligió un control de flujo por software, también conocido como control XON/XOFF. A la diferencia del control de flujo por hardware, a donde se usa pines dedicadas, el control por software utiliza caracteres determinados para indicar si la transmisión debe pausarse o reanudarse. La comunicación es iniciada por la computadora que envía periódicamente una trama de datos que se almacena en un buffer de entrada del nodo orquestador. Cuando este buffer está casi lleno, no espera que se llene para asegurar la recepción de todos los paquetes, se envía un carácter XOFF (0x13), la computadora lo recibe y deja de transmitir. Una vez que el buffer se vacío hasta su límite inferior, por ejemplo 20% de su capacidad, se manda el carácter XON (0x11), la computadora lo recibe y sigue emitiendo sus datos.

6. Modos implementados

Entre los diferentes comandos posibles, dos modos de operación fueron implementados.

Primero, tenemos el modo de posicionamiento por perfil (PP) identificado con la letra « P» en la trama. Este primer modo de operación envía una posición objetivo a los dos nodos extremos, que la alcanzarán siguiendo un perfil de velocidad trapezoidal a partir de la posición inicial. Este modo no requiere ninguno cálculo ya que el procesamiento es efectuado en los nodos extremos.

Segundo, el modo de posición interpolada (IP) identificado con la letra « S » requiere que los puntos de trayectoria sean calculados en el software de la PC con la planificación de trayectoria mencionada arriba. Una vez que la trayectoria es discretizada en puntos cada 10ms, se convierten las consignas de posición angular de las articulaciones en posición a incrementos del motor. Las consignas están en pasos del motor e indican posiciones absolutas. A su vez, el nodo orquestador envía periódicamente las consignas que deben cumplir los nodos secundarios en el siguiente instante de tiempo junto con un mensaje de sincronización. Por último, es interesante notar que es responsabilidad del software de generar trayectorias consistentes, válidas y libre de singularidades [Fig. 12].

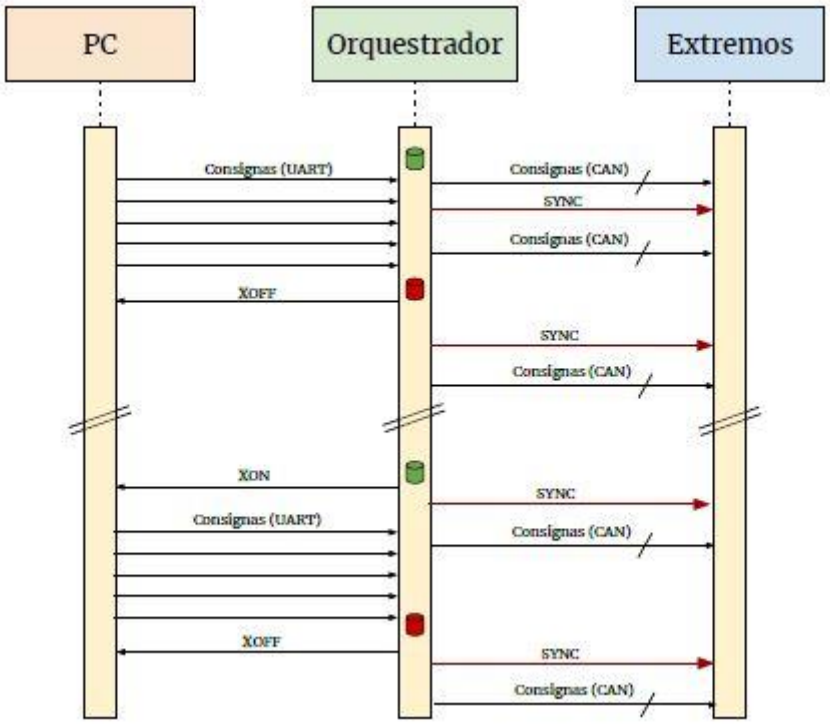

Figura 12 : Diagrama temporal indicando la secuencia de envıo de consignas por UART y el envıo de consignas sincronizadas a los extremos

2 – Entorno y descargas

La parte de comunicación entre el software y el nodo orquestador del robot fue desarrollada con una UART. Como se mencionó arriba, esta elección proviene del hecho que se puede utilizar cualquier nivel superior de control que puede ejecutar Python. Además, la parte de software programada para calcular las trayectorias fue redactada en Python.

Así que, para seguir en esta dinámica, y simplificar el agregado de una parte interfaz usuario a la parte software, todo fue desarrollo en Python.

Primero, la elección del framework (2) debía tomar en cuenta que queremos desarrollar un software en la computadora, en local. Así que los frameworks como django o flask que requieren un navegador web no cumplen con esta primera característica.

Segundo, usar un frameworks que no tiene muchos requerimientos para instalar es importante. Además, si es conocido, nos da un acceso a una cantidad de información más importante sobre las herramientas o los posibles errores encontrados en los navegadores web. Asimismo,

desarrollando este proyecto como estudiante de intercambio era necesario que sea conocido para tener un uso autorizado en América del sur y Europa. Con estos requerimientos, resaltaron dos nombres principales, PyQt y Thinker. Por el consejo de mi tutor sobre el framework PyQt y los comentarios de usuarios que lo describen más intuitivo, este último fue seleccionado para desarrollar la interfaz usuario del proyecto.

PyQt es un módulo de interfaz gráfica de usuario (GUI). Le permite conectar el marco multiplataforma C++ Qt con el lenguaje Python por el enlace « PySide ». Para crear una GUI en PyQt hay dos opciones principales, o usar QT Designer o codificar manualmente en código Python. Aunque la primera puede permitir más productividad, la segunda da un control sobre el código de la aplicación y aumenta la flexibilidad del proyecto. Asimismo, QT Designer requiere descargas adicionales y tampoco permite de agregar características interesantes para este proyecto. Y un desarrollo manual es preferible para agregar con más sencillez características futuras también.

La utilización de PyQt requiere la instalación de :

- Python 3.11.6
- PyQt5

Además, se generó un paquete Python que se puede instalar con la herramienta « pip », se cuenta con :

- Matplotlib 3.4: utilizada para graficar.
- Numpy 1.21: utilizada para el uso de matrices y vectores.
- Pandas 1.4: lectura y escritura de archivos .csv.
- Pyserial 3.5: comunicación con el nodo orquestador.

Para la creación de puertos serie virtuales se uso el software Virtual Serial Port Driver. Además, para facilitar la lectura de estos puertos el software PuTTY fue también descargado.

Por fin, un repositorio GitHub es accesible para acceder al software desarrollado :

- https://github.com/CaroleHuet/dual-scara-interfaz

3 – Desarrollo de la interfaz

Para empezar, PyQt es un framework, como se mencionó arriba, que utiliza una cantidad de módulos significativa. Entre otros, « QtCore » una biblioteca base que proporciona contenedores, gestión de subprocesos, gestión de eventos y mucho más o « QtGui » y « QtWidgets » un conjunto de herramientas GUI para escritorio, que proporciona muchos componentes gráficos para diseñar aplicaciones [5].

La jerarquía de PyQt por la parte de los objectos es la siguiente : « QObject » está en el módulo « QtCore » y los « Widgets » (3) heredan « QObject ». QObject es la clase más básica de Qt, la mayoría de las clases en Qt heredan de esta clase [Fig 13].

Los widgets pueden responder a eventos, usar el « parenting system » (4) o los mecanismos de señal. La clase « QWidget » contiene la mayoría de las propriedades que se utilizan para describir una ventana o un widget, como la posición, el tamaño, la etiqueta…

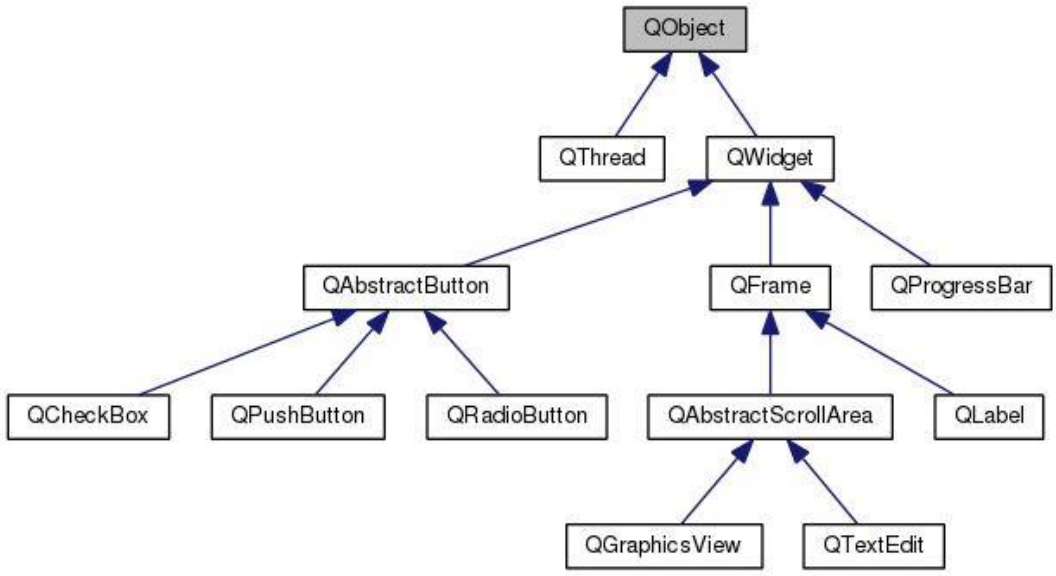

Figura 13 : Diagrama que enumera las clases que heredan de la clase QObject, especialmente en el módulo Widgets

1. Widgets

Los widgets que se usan en el siguiente desarrollo son muchos, por la parte de la entrada de datos están principalmente el « QComboBox » y el « QDoubleSpinBox ». Por la parte de dialogo entre las ventanas se usó el « QPushButton, » o el « QDialogButtonBox » asociado con un « QMessageBox ». Para organizar los widgets en las ventanas se usa « QGridLayout » y « QVBoxLayout ». Por último, una parte más de sistema definida con el « QApplication » que especializa « QGuiApplication » con algunas funciones necesarias para aplicaciones basadas en « QWidget » como la inicialización y finalización específicas del widget.

2. Ventana principal

Para empezar el proyecto se define una primera ventana con un clase « Window » del widget « QMainWindow ». Un proyecto puede contener varias ventanas de este tipo, no significa que sea la ventana principal. Pero este tipo de ventana tiene su propia disposición, el cual puede contenir un « QToolBar », un « QMenuBar » o un « QStatusBar »[Fig 14]. Este diseño es muy interesante por la parte de conexión por UART para proponer los diferentes parámetros que compone la trama pero también para informar del estado de la conexión sobre el puerto.

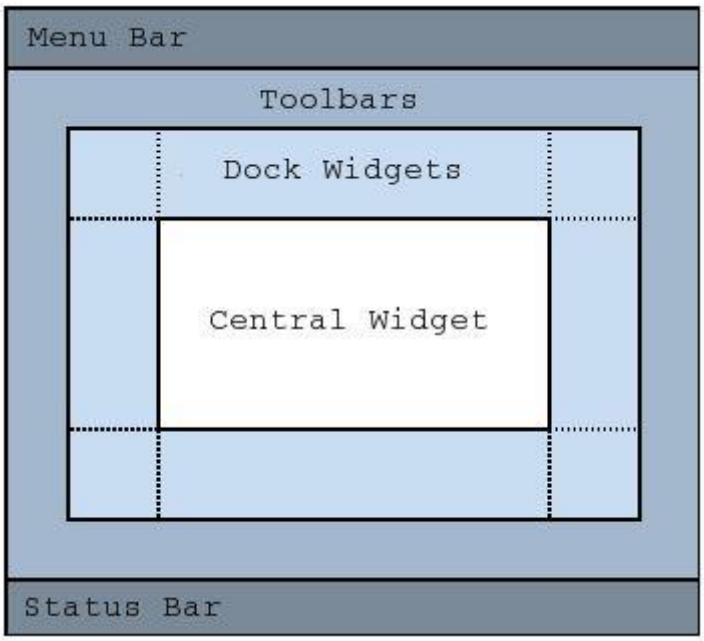

Figura 14 : Disposición definida de una ventana QMainWindow

3. Barra de herramientas

El desarrollo de la barra de herramienta o en ingles toolbar se hace en una clase dedicada. Desde la clase « Window » se llama la clase « ToolBar » para crearla y agregarla a la ventana principal. También se define la parte « status bar » de la « QMainWindow » [Fig 15].

```
### Tool Bar ###
self.toolBar = ToolBar(self)
self.addToolBar(self.toolBar)

### Status Bar ###
self.setStatusBar(QtWidgets.QStatusBar(self))
self.statusText = QtWidgets.QLabel(self)
self.statusBar().addWidget(self.statusText)
self.statusText.setText('Conectar el puerto')
```

Figura 15 : Definicion del widget Toolbar

En la clase « ToolBar » se define los diferentes menús desplegables de cada característica de la conexión UART. Como recordatorio, esta conexión necesita una definición de baudios, de bit de palabra, de bit de paridad, de bit de stop y de control de flujo. Por ejemplo, la definición del menú para los baudios se hace con la creación de un « QtWidgets.QComboBox » que se llamó « baudRates ». Se asocia características a este nuevo widget, empezando con una lista de valores que puede tomar el widget, eso con la función « addItems ». Además, se elijió el valor por defecto que tomara el widget, acá se define el valor « 921600 » elegido en el proyecto gracias a la función « SetCurrentText ». Por fin, se define una altura predeterminada con la función « setMinimunHeigh ».

En esta barra se implementa también un botón para conectarse al puerto elegido. En primer lugar, se detectan los puertos conectados gracias a la función « availablePorts » perteneciente a « QserialPortInfo » de « QserialPort ». Los nombres de los puertos conectados son agregados

al menú desplegable « portNames ». Una vez está el puerto elegido en la lista, se presiona el botón « portOpenButton », un widget de tipo « QPushButton », y con la función asociada « setCheckable » a « true » (5) se mantiene pulsado [Fig 16]. Todos los widgets son agregados a su ventana con la función « addWidget » a veces asociada con un widget de posición, como se mencionó arriba. [Fig 17]

```
class ToolBar(QtWidgets.QToolBar):
    def __init__(self, parent):
        super(ToolBar, self).__init__(parent)

        self.portOpenButton = QtWidgets.QPushButton('Conectar') #Connect
        self.portOpenButton.setCheckable(True)
        self.portOpenButton.setMinimumHeight(32)

        self.portNames = QtWidgets.QComboBox(self)
        self.portNames.addItems([port.portName() for port in QSerialPortInfo().availablePorts()])
        self.portNames.setMinimumHeight(30)

        self.baudRates = QtWidgets.QComboBox(self)
        self.baudRates.addItems(['300', '600', '1200', '2400', '4800', '9600', '14400', '19200', '28800',
                                 '38400', '57600', '115200', '230400', '460800', '576000', '921600'
        ])
        self.baudRates.setCurrentText('921600')
        self.baudRates.setMinimumHeight(30)

        self.dataBits = QtWidgets.QComboBox(self)
        self.dataBits.addItems(['5 bit', '6 bit', '7 bit', '8 bit'])
        self.dataBits.setCurrentIndex(3)
        self.dataBits.setMinimumHeight(30)
```

Figura 16 : Extracto de la fonción Toolbar

```
self.addWidget(self.portOpenButton)
self.addWidget(self.portNames)
```

Figura 17 : Herramienta «addWidget» para agregar un widget a la ventana

4. Conexión al puerto serie

Una vez la barra de herramientas creada con todos sus widgets, la ventana principal recupera la información y el puerto puede conectarse. [Fig 22] Para conectar el puerto serie, se programa una función « portOpen » que conecta el puerto al hacer clic sobre el botón « portOpenButton », de la barra de herramienta mencionado en el párrafo precedente [Fig 18]. Esta función,

recupera toda la información seleccionada en la barra de herramientas para configurar el puerto «port » configurado como « QSerialPort ». En segundo lugar, se verifica si el puerto puede conectarse con el módulo « QtCore » y dependiendo de la respuesta se conecta o no el puerto. En cada caso, el estado de la conexión se actualiza en la parte « status bar » de la ventana principal. Cuando el puerto está abierto, las características de la barra de herramientas se fijan con la función « serialControlEnable » [Fig 19].

```
### Signal Connect ###
self.toolBar.portOpenButton.clicked.connect(self.portOpen)
```

Figura 18 : Lanzado de conneción del puerto serie

```
def portOpen(self, flag):
    if flag:
        self.port.setBaudRate(self.toolBar.baudRate())
        self.port.setPortName(self.toolBar.portName())
        self.port.setDataBits(self.toolBar.dataBit())
        self.port.setParity(self.toolBar.parity())
        self.port.setStopBits(self.toolBar.stopBit())
        self.port.setFlowControl(self.toolBar.flowControl())
        r = self.port.open(QtCore.QIODevice.ReadWrite)

        if not r:
            self.statusText.setText('Puerto abierto error') #'Port open error'
            self.toolBar.portOpenButton.setChecked(False)
            self.toolBar.serialControlEnable(True)
        else:
            self.statusText.setText('Puerto abierto') #'Port opened'
            self.toolBar.serialControlEnable(False)
    else:
        self.port.close()
        self.statusText.setText('Puerto cerrado')   #'Port closed'
        self.toolBar.serialControlEnable(True)
```

Figura 19 : Conexión del puerto serie si está disponible

El puerto serie está configurado y conectado, ahora se activó la lectura y la escritura con las funciones « readFromPort » y « sendFromPort » respectivamente [Fig 20].

```
self.port.readyRead.connect(self.readFromPort)
self.serialSendstr.connect(self.sendFromPort)
```

Figura 20 : lanzado de conexión de lectura y escritura

5. Control de flujo

En la parte de la lectura del puerto, se agregó el control de flujo de la conexión. Se realizó una estructura « if » con la condición de que si llego el XOFF (0x13) sobre el puerto, la escritura se desconecta y al contrario, cuando se recibe el XON (0x11) se vuelve a conectar. Una prueba de funcionamiento fue realizada con otros tipos de datos que las trayectorias porque su generación es muy rápida y no deja el tiempo de enviar un mensaje de parada antes que sea enviada toda la trama. Esta parte del programa necesitará una prueba en contexto real para confirmar totalmente su eficacia, comprobar el seguimiento del envío de datos [Fig 21].

```
def readFromPort(self):

    data = self.port.readAll()

    if(data == "0x13"):
        action = "wait"
        self.serialSendstr.disconnect(self.sendFromPort)

    elif (data == "0x11"):
        action = "write"
        self.serialSendstr.connect(self.sendFromPort)


def sendFromPort(self, text):

    self.port.write(text.encode())
```

Figura 21 : Función de lectura con el control de flujo y función de escritura

6. Selección del tipo de trayectoria

En la ventana principal se encuentra también diferentes botones que permiten la elección del tipo de trayectoria deseada. Se dividió en dos partes, primero se colocó el modo de posicionamiento por perfil (PP) y segundo el modo de posición interpolada (IP). El modo PP tiene solo un botón en contrario del modo IP porque el modo PP requiere solo las coordenadas objetivo del robot [Fig 22].

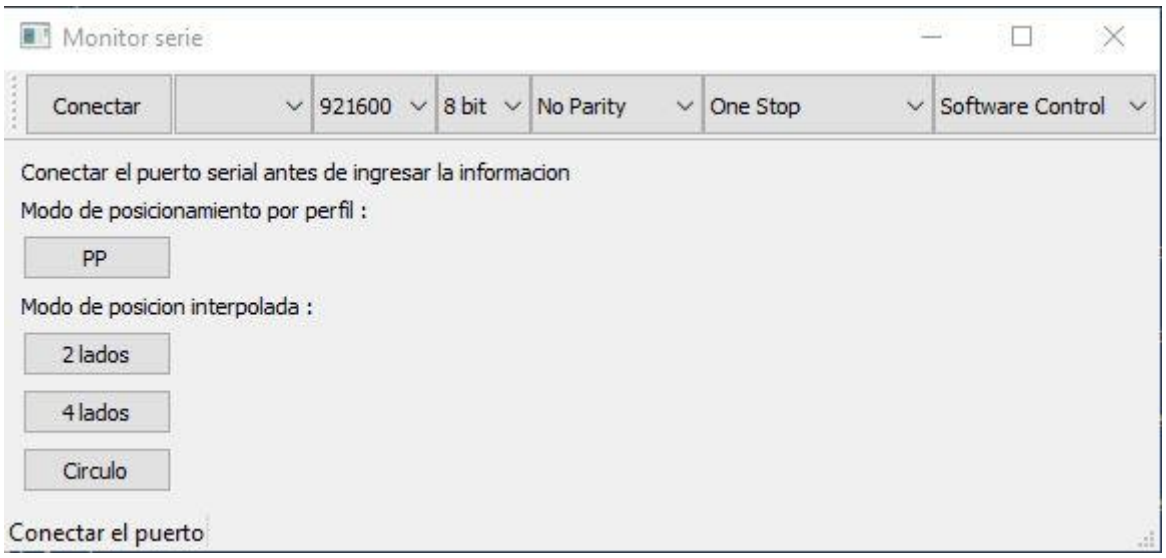

Figura 22 : Ventana principal QMainWindow

1. *Ventana de dialogo – modo IP*

Tres trayectorias fueron configuradas para probar el modo de posición interpolada, una circular, una con 2 lados y una con 4. Para mostrar este modo, se tomó por ejemplo la trayectoria con 2 lados.

El inició de una trayectoria se hace al clickear sobre el botón « 2 lados », un widget de tipo « QPushButton » [Fig 23]. De hecho, la acción llama la función « enter2 » y activa la ventana de diálogo correspondiendo al hacer una trayectoria de 2 lados, nombradas « Dialog2 » [Fig 24].

```
self.button2 = QtWidgets.QPushButton("2 lados", self)
self.button2.clicked.connect(self.enter2)
self.button2.setSizePolicy(QtWidgets.QSizePolicy.Maximum, QtWidgets.QSizePolicy.Preferred)
```

Figura 23 : Configuración de un QPushButton para llamar al diálogo para diseñar la trayectoria en 2 puntos

```
def enter2(self):
    """launch Dialog Window for data entry"""

    self.dialog = Dialog2(self)
    self.dialog.data.connect(self.dataEntered2)
    self.dialog.exec_()
```

Figura 24 Función de llamada del diálogo de creación de la trayectoria

La ventana dedicada a la entrada de los datos para diseñar la trayectoria está únicamente compuesta de un widget. Los valores de las coordenadas absolutas son ingresados en inputs numéricos, cada uno ha sido configurada para permitir la selección de números negativos con « setMinimum ». Además, los inputs fueron configurados como « double » y no « integer » (6) por el espacio de trabajo del robot reducido [Fig 25].

```
self.spinbox_wp_1x = QtWidgets.QDoubleSpinBox(self)
self.spinbox_wp_1x.setMinimum(-9.00)
```

Figura 25 : Creación de un widget de tipo spinbox

Una vez que los datos han sido ingresados para cada punto, se confirma la selección con el botón « Ok ». Este botón no es de tipo « QPushButton » como los anteriores porque el dialogo necesita una configuración con dos botones. Uno para confirmar los datos y el otro para cancelar y cerrar la ventana sin cerrar el programa. De hecho, los botones son de tipo « QDialogButtonBox » y asocia un botón « Ok » y un botón « Cancel » [Fig 26]. El envió de los datos se realiza a través de una lista de tipo « QtCore.pyqtSignal » inicialmente configurada en el diálogo. Esta señal permite al envió de los datos a la clase principal que llamó a la clase de

diálogo. En este contexto, dos retornos son posibles, o la información sobre los puntos fueron confirmadas y el acceso al hacer clic ejecuta la función « accept » que emite la lista de coordenadas y cierra el diálogo. O el diálogo es cancelado y la función « rejet » devuelve una lista vacía a la clase « Window » y igualmente, cierra el diálogo [Fig 27] [Fig 28].

```
# create buttons to end the dialog
qbb = QtWidgets.QDialogButtonBox.Ok | QtWidgets.QDialogButtonBox.Cancel
self.buttonbox = QtWidgets.QDialogButtonBox(qbb)
self.buttonbox.accepted.connect(self.accept) # => button "Ok"
self.buttonbox.rejected.connect(self.rejet) # => button "Annuler"
```

Figura 26 : Configuración de un widget botón de tipo QDialogButton y acceso a sus funciones asociadas

```
#=====================================================================
def accept(self):
    """normal closing of the dialog window ("Ok" button)"""

    self.data.emit([self.spinbox_wp_1x.value(), self.spinbox_wp_1y.value(), self.spinbox_wp_1z.value(),
                    self.spinbox_gl_x.value(), self.spinbox_gl_y.value(), self.spinbox_gl_z.value()])
    self.close()

#=====================================================================
def rejet(self):
    """canceling the data search ("Cancel" button)
       NB: same for closing using the cross or the system menu"""

    self.data.emit([None,None,None,None,None,None])
    self.close()
```

Figura 27 : Funciones asociadas al envió de datos a la ventana principal

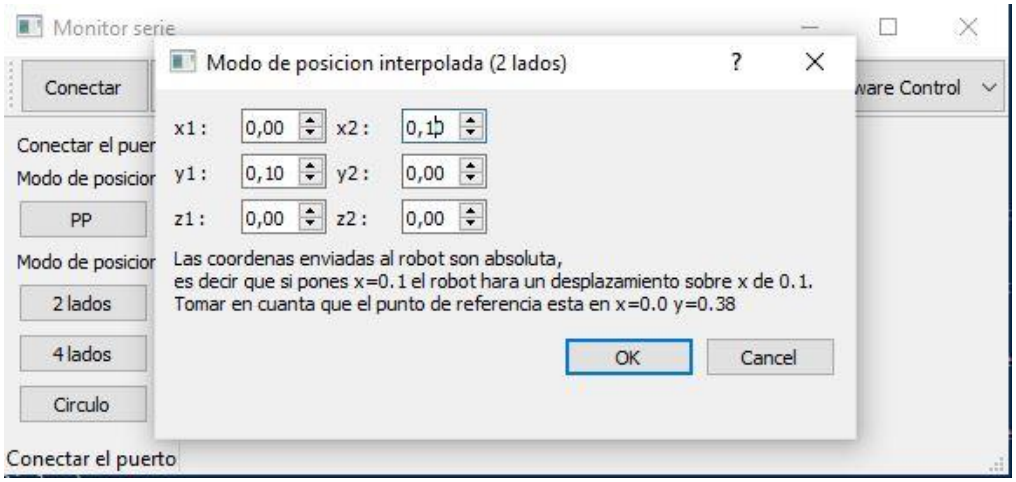

Figura 28 : Ventana diálogo para diseñar una trayectoria con 2 puntos

A la recepción de los datos por parte de la clase « Windows », la función « enter2 » transfiere los datos a la función « dataEntered2 », que si recibe una lista no nula abre un diálogo con los datos ingresados [Fig 29] [Fig 30].

```
def dataEntered2(self, list):
    """Retrieving information"""
    self.dialog.hide() # hides the dialog window still displayed (it will be closed immediately afterwards)
    if list!=[None, None, None, None, None, None]:
        QtWidgets.QMessageBox.information(self,
            "Puntos entrados:",
            "wp_1x: {}\nwp_1y : {}\nwp_1z : z : {}\ngl_x: {}\ngl_y : {}\ngl_z : {}\n".format(list[0],list[1],list[2],
```

Figura 29 : Foncion llamada a la recuperacion de los datos entrados en la ventana de dialogo y implementacion de una ventana QMessageBox para mostrar los datos recibidos en la lista

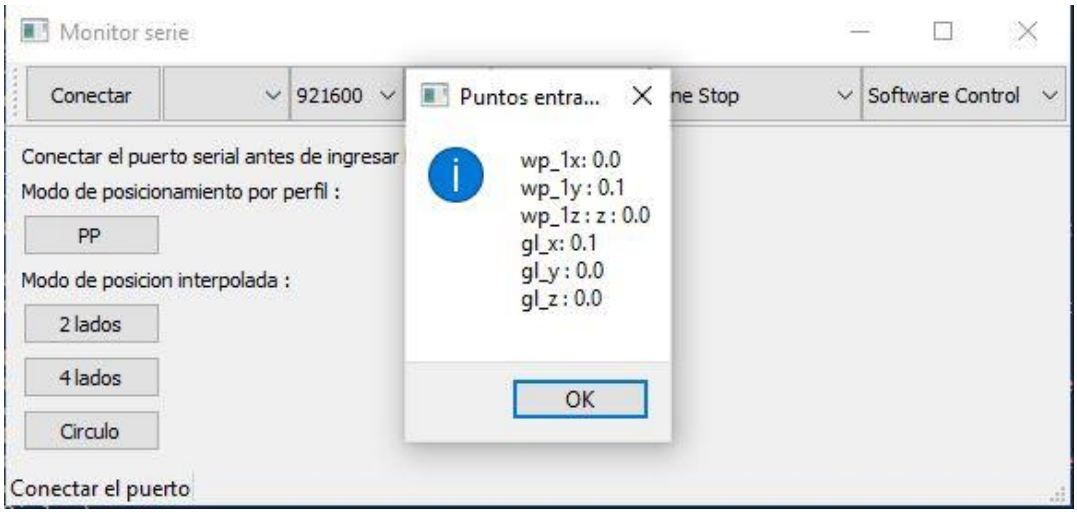

Figura 30 : Ventana creada al recibir la lista de datos ingresados en el diálogo

A continuación, una vez que se cierra el diálogo, todos los cálculos de los puntos de la trayectoria se hacen y son enviados sobre el puerto serie. La parte de programación de los cálculos es extraída del software desarrollado para guardar los datos en un archivo .csv. De hecho, se crea un objecto « Path » y se asocian los datos ingresados por el usuario a los vectores « wp_1 » y « gl» respectivamente el segundo punto de la trayectoria y el ultimó. Después, se determinan los ángulos « q_lin » que cada motor debe tener para cada punto de la trayectoria. Luego, estos ángulos tienen que ser normalizados para corresponder a los pasos de los motores, así que se calcula el factor y se convierten los valores con la función

« normalize_trajectory ». Por último, antes de emitir la trama por el puerto UART, se convierten los datos de ángulos en hexadecimal y se forma la trama con todos los datos. Asimismo, la función « encode_send_trayectory » fue modificada para enviar los datos por puerto UART y no en .csv [Fig 31] [Fig 32] [Fig 33].

```
# Create a Path object
path = Path()

# Define some interesting points
wp_1x = list[0]
wp_1y = list[1]
wp_1z = list[2]
gl_x = list[3]
gl_y = list[4]
gl_z = list[5]

st = path.robot.fkine(np.deg2rad([180, 0, 0]))
wp_1 = st + np.array([wp_1x, wp_1y, wp_1z])
gl = wp_1 + np.array([gl_x, gl_y, gl_z])
pose = np.block([[st], [wp_1], [gl]])
max_v = [0.3, 0.3, 0.3, 0.3]
max_a = [1, 1, 1, 1]

# LINE
q_lin, qd_lin, qdd_lin, p_lin, pd_lin, pdd_lin = path.line(
    pose=pose, max_v=max_v, max_a=max_a, enable_way_point=False)

# Normalize desire trajectory
factor = 4 * 300 / np.pi
q = normalize_trajectory(q_lin, factor)
self.encode_send_trajectory(q)
```

Figura 31 : Lectura y uso de los datos ingresados para calcular la trayectoria

```
def encode_send_trajectory(self,points):
    for point in points:
        q1 = int(point[0]).to_bytes(length=4, byteorder='big',
                                    signed=True).hex()
        q2 = int(point[1]).to_bytes(length=4, byteorder='big',
                                    signed=True).hex()
        frame = 'S:,' + q1 + ',' + q2 + '\r\n'
        print(frame)
        self.serialSendstr.emit(frame)
```

Figura 32 : Función de conversión de los datos en hexadecimal y creación y envió de la trama

```
S:,000003cb,0000004c
```

Figura 33 : Ejemplo de trama enviada por puerto serie

2. Ventana de dialogo – modo PP

El envió de datos en modo de posicionamiento por perfil se desarrolla sobre el mismo funcionamiento. Se puede mencionar algunos puntos de divergencias interesante como en la ventana de diálogo de entrada de los datos. De hecho, el modo PP tiene dos comandos asociados, el modo normal « P » y el modo con una velocidad máxima « V ». Para seleccionar esta duplicación de comando, un menú desplegable es agregado en el diálogo [Fig 34]. En la parte de cálculos, no se requiere tantos cálculos como en el modo IP, así que se implementó las ecuaciones del modelo inverso mencionado en el proyecto para calcular los ángulos de los motores. Después, se convierte de igual manera los datos angulares con el factor determinado, y se convierte en hexadecimal. Por otro lado, el comando a enviar en la trama se determina mediante una estructura « if » que compara el dato enviado en primer valor de la lista. El comando « P » corresponde a el valor « 0 » y el « V » a el valor « 1 ».

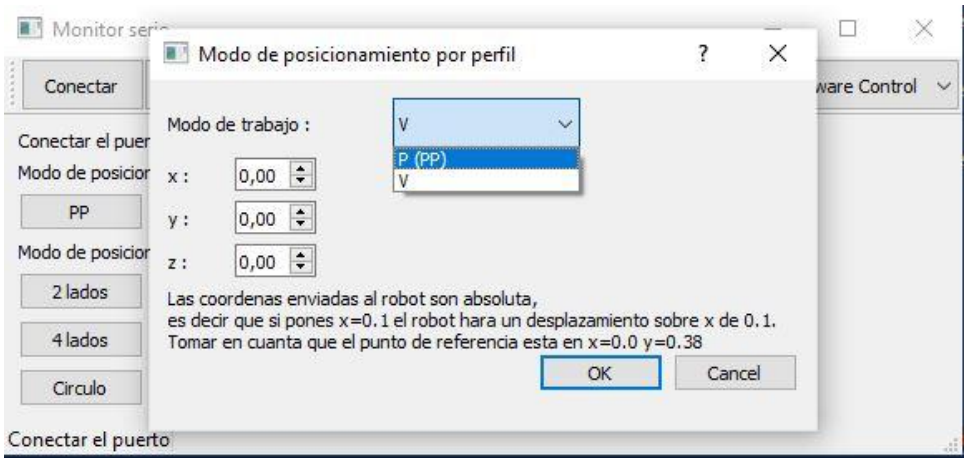

Figura 34 : Ventana diálogo del modo PP con menú desplegable para la selección del comando

Capítulo 4

El desarrollo de esta interfaz estuvo acompañado de numerosas pruebas de funcionamiento y finalmente de una prueba de funcionamiento global de la conexión con la simulación en MATLAB.

1. Conexión virtual

En primer lugar, algunos preparativos son necesarios para realizar una conexión virtual en una computadora. De hecho, la conexión UART puede utilizar un puerto USB para acoplarse con la computadora, asimismo en el « Administrador de dispositivos » se encuentran como puerto COM [Fig. 35]. Para crear un par de puertos COM virtuales, dos softwares fueron utilizados pero con un límite impuesto por los creadores. En primero, el « Virtual Null Modem » permite establecer una conexión virtual entre dos puertos pero con un limite de 65k de datos intercambiados. Este límite es alcanzado rápidamente y no es el más adecuado para este proyecto, aunque es interesante señalarlo para compararlo con otro software. Por otro lado, el software « Virtual Serial Port Driver » propone una versión de prueba para 14 días sin límites sobre él envió de datos [Fig. 36]. Es con este software que la mayoría de las pruebas fueron hechas, pero en un proyecto de varios meses, 14 días pueden volverse cortos. De hecho, desinstalar e instalar de nuevo el software no permite de acceder de nuevo a una versión de prueba. Solo con el uso de otra computadora pude completar validaciones del proyecto.

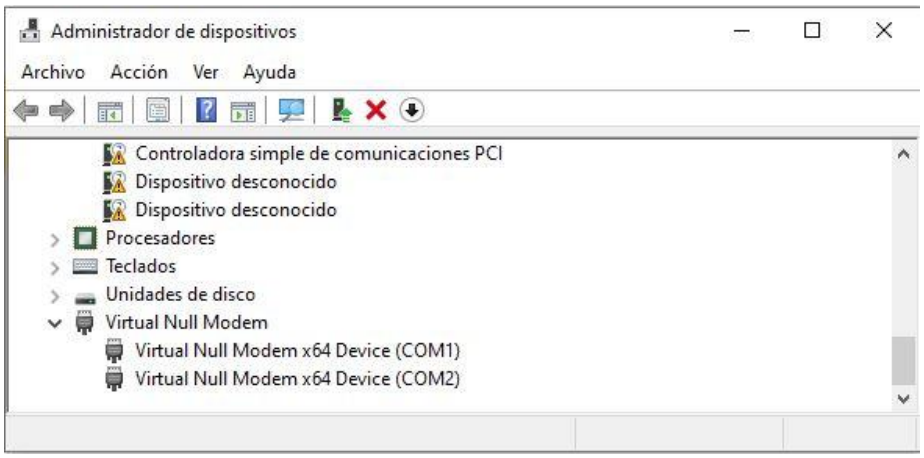

Figura 35 : Administrador de dispositivos, puerto COM

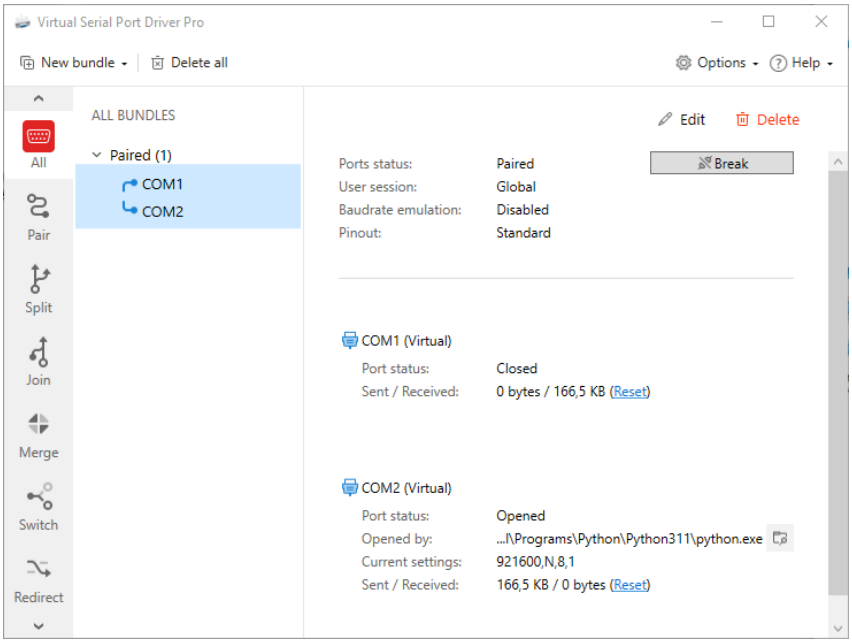

Figura 36 : Interfaz del software Virtual Serial Port Driver, par de los COM 1 y 2

Una vez que la conexión virtual fue realizada, realicé prueba rápida con el envío de una trayectoria a un emulador de terminal como PuTTY y confirmé el funcionamiento del lazo entre los dos puertos [Fig. 37]. Una prueba puede también hacerse comunicando entre dos terminales « Windows PowerShell ». Por ejemplo, haciendo andar en cado una una interfaz pero teniendo cuidado de conectar cada una a un puerto serie diferente del par.

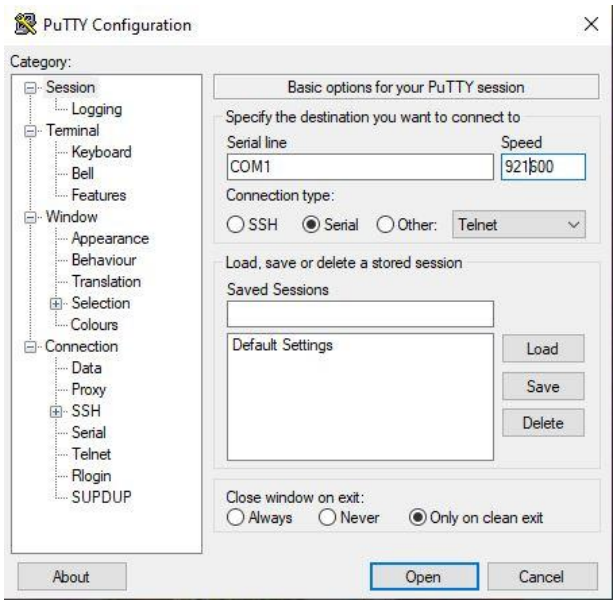

Figura 37 : Interfaz del software PuTTY, configurado para conectarse en serial a 921600 baudios

Con una prueba de transmisión de una trayectoria con dos puntos, el emulador conectado al puerto COM1 y la interfaz conectada al puerto COM2, se verificó el funcionamiento del enlace virtual desarrollado [Fig. 38].

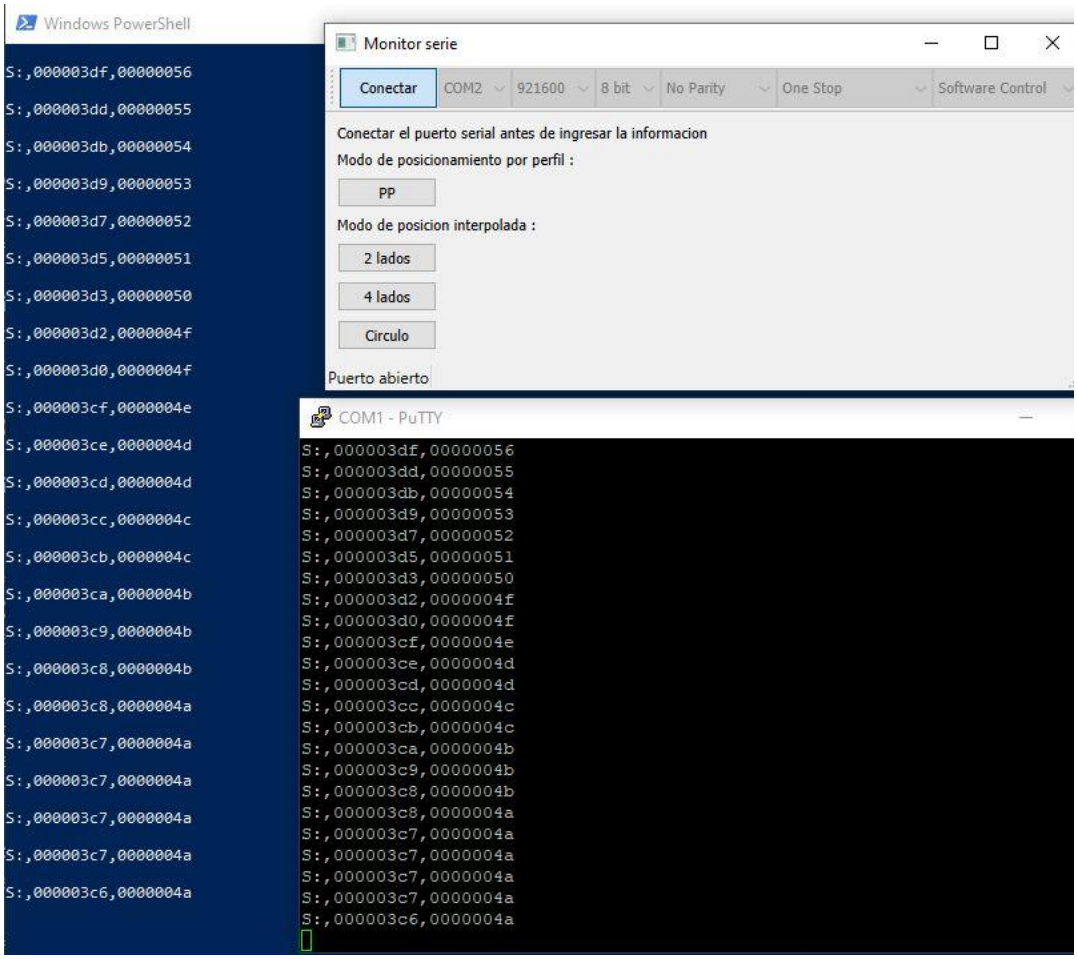

Figura 38 : Prueba de envío de datos entre el puerto COM1 y el puerto COM2

2. Prueba de conexión UART

El envió de los datos entre la interfaz y el robot se realiza por puerto UART con una trama definida en la sección 3.1.4. Con el objetivo de confirmar la viabilidad de los datos enviados desde la interfaz, se desarrolla en Matlab un programa para leer las tramas enviadas desde la interfaz. Una vez leídas, se trata de usar los datos de la trayectoria enviada por el usuario para verificar que la trayectoria inicial en la interfaz es la realizada por el robot.

En Matlab la lectura de un puerto serie se hace con la función « serialport » que requiere el puerto COM con cual conectarse y la velocidad en baudios , los demás parámetros se configuran por defecto pero se pueden cambiar. Después de haber conectado Matlab, se lee los datos recibidos con la función « read ». En este caso, se elegió leer los datos por paquete de 22 valores y guardarlos en « r » en formato « string » (6). Se eligió un paquete de 22 valores porque una trama contiene 22 valores entre el comando, las coordenadas y la puntuación. Una vez que una trama es leía, la función « isempty » permite saber si quedan valores por leer en el puerto. De hecho, esta función queda verdadera hasta que no quedan valores a leer en « r ». Cada vez que se lee una trama se agrega a una lista y este sistema se repite hasta que «isempty » vuelve a ser falso [Fig. 39].

```
%% Lectura puerto serial

n_step = 1;
s = serialport("COM1",921600);
r = read(s,22,"string");
e = isempty(r);

while e == 0
    tramaList(:,n_step)=[r];
    n_step = n_step + 1;
    r = read(s,22,"string");
    e=isempty(r);
end
```

Figura 39 : Conexión y lectura del puerto serie

Una vez que las tramas son guardadas en una lista, empieza la extracción de la trayectoria punto a punto. Para realizar eso, se descifra trama por trama y se guarda cada punto en un vector. Para iniciar este proceso se localiza ciertos caracteres conocidos de la trama : la puntuación. Después, se puede descifrar el modo de comando desde la posición de los dos puntos con la función « extract». Además, es necesario de cambiar el tipo para guardar el dato porque los tipos « array » (6) no permiten el uso de muchas funciones, a diferencia del tipo « string ».

De la misma manera se extraen los datos de los nodos activos q_{11} y q_{12}. La diferencia es que estos datos son enviados sobre varios bytes, así que se aplica un bucle para recuperar los valores en un « string » único. La función « append » permite agrupar a la cadena de caracteres

descifrada el último carácter leído. Para continuar, se convierte el valor angular descifrado de hexadecimal a decimal. Esta conversión es necesaria porque la simulación se basa en un sistema dirigido con motores a corriente continua y no de paso a paso. Para pasar de los valores en el sistema continuo a los valores en paso, se usó un factor correspondiendo a las características de los motores paso a paso. Así que una división entre el valor angular en pasos recibido y el factor mencionado devuelve el valor angular en espacio continuo. Una vez realizado este cálculo, se guardan los valores angulares en una matriz [Fig. 40].

```
28      factor = 4*300/pi
29
30      for m_step = 1:n_step-1
31          str = '';
32          i_step = 1;
33          t = tramaList(m_step)
34          t_char = char(t)
35          [m,n]=size(t_char)
36          pos_puntos = strfind(t,":");
37          pos_coma = strfind(t,",");
38          coma1 = pos_coma(1,1);
39          coma2 = pos_coma(1,2);
40
41          % Extract comando
42          pos_com = pos_puntos + 1;
43          val = extract(t,pos_com);
44          A = cell2mat(val); %Comando para pasar en str y no en cell array
45          str_commando = append(str,A); % comando en str
46
47          % Extract q11
48          str = '';
49          while i_step+3 < coma2
50              pos = coma1 + i_step;
51              val = extract(t,pos);
52              A = cell2mat(val);
53              str = append(str,A);
54              i_step = i_step + 1;
55          end
56
57          str_dec11 = hex2dec(str);
58          q11 = str_dec11/factor
59
60          % Extract q12
61          str = ''
62          while i_step+5 < n
63              pos = coma2 + i_step-8;
64              val = extract(t,pos);
65              A = cell2mat(val)
66              str = append(str,A)
67              i_step = i_step + 1;
68          end
69
70          str_dec12 = hex2dec(str)
71          q12 = str_dec12/factor
72          qi1_t(:,m_step) = [q11,q12]';
73      end
```

Figura 40 : Recuperación de los datos transmitidos en las tramas enviado sobre la conexión UART

Ahora que se recuperó la trayectoria en espacio articular, se verifica con la simulación desarrollada que la trayectoria recibida es igual a la trayectoria pedida por el usuario. Para realizar esta prueba, el cálculo de los valores de las articulaciones pasivas es necesario, y se realizan con las ecuaciones del modelo geométrico inverso [Fig. 3]. Una vez que estas coordenadas son guardadas en una matriz se puede realizar la simulación de la posición de los eslabones.

Además, para verificar que las coordenadas del efector final corresponden a los valores angulares recibidos, se utiliza el modelo geométrico directo para recuperar los valores de x e y del efector en una matriz. Estos valores permiten comprobar en vivo que la simulación de los eslabones y del efector final siguen la misma trayectoria. Además, se traza una curva XY para comparar la trayectoria de la simulación de la interfaz con la trayectoria de Matlab. Se muestran también las curvas de x e y dependiendo del tiempo y de q_{11} y q_{21} dependiendo del tiempo para ver si la ejecución de la trayectoria se realiza en el tiempo requerido. Con todas estas verificaciones se puede confirmar que las trayectorias pedidas por el usuario se realizan correctamente [Fig. 41].

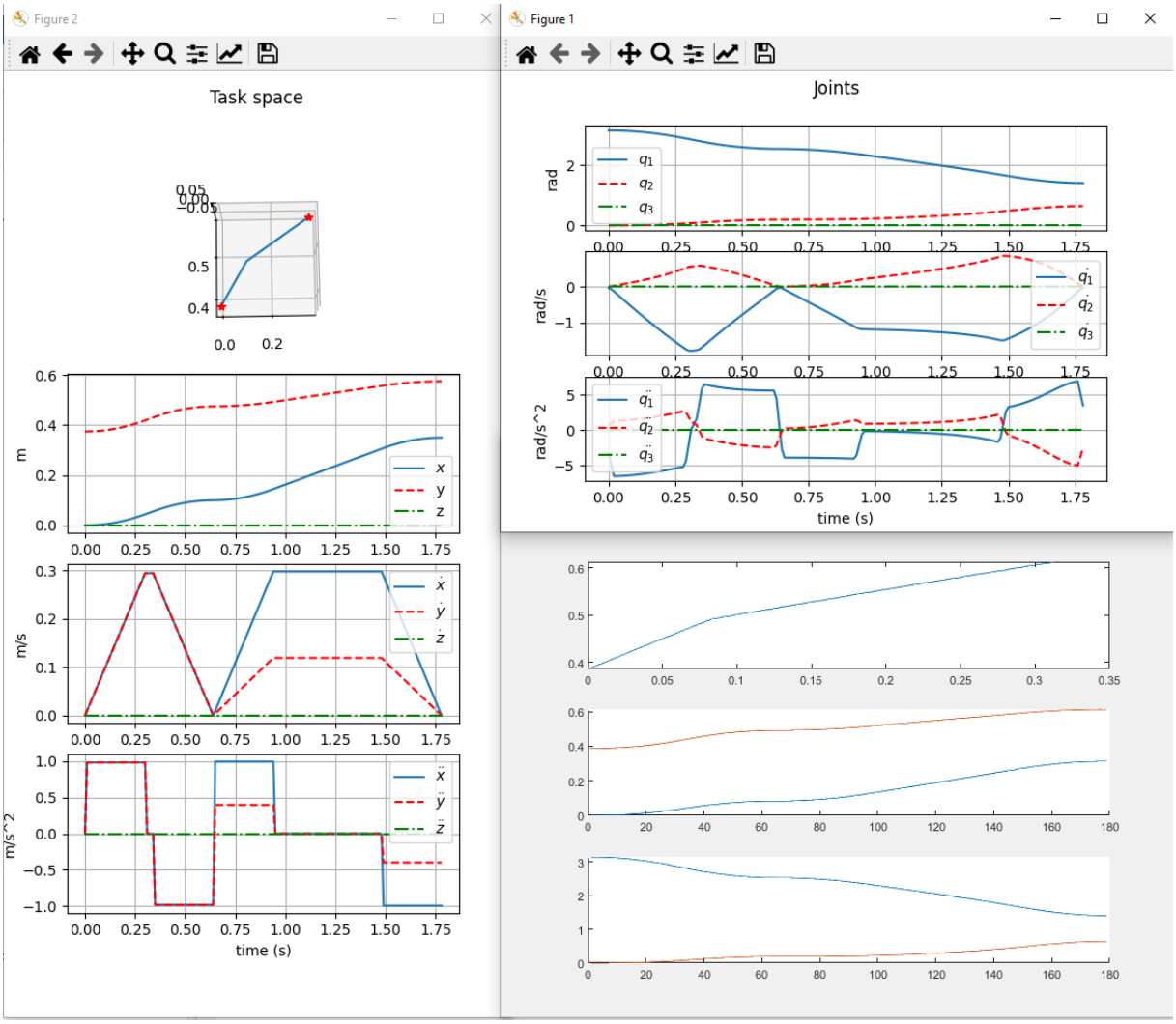

Figura 41 : Simulaciones de la interfaz y de Matlab sobre una misma trayectoria

Cuadro izquierdo: trayectoria y coordenadas, velocidad y aceleración del efector final en el tiempo. Cuadro arriba a la derecha : coordenadas, velocidad y aceleración de los nodos activos en el tiempo. Cuadro abajo a la derecha trayectoria, coordenadas del efector final y coordenadas de los nodos activos en el tiempo en Matlab.

En resumen, comparando las trayectorias trazadas en Matlab después del descifrado de la trayectoria recibida, se puede confirmar que el robot realiza las trayectorias pedidas desde la interfaz. La trayectoria recorre los mismos puntos al mismo tiempo como se puede ver en las gráficas de las coordenadas del efector final y de los nodos activos.

Conclusión

El desarrollo de una interfaz para seguir ampliando las capacidades del proyecto de robot paralelo tipo doble SCARA es funcional. Esta interfaz permite la elección de diferentes modos de trabajos por un lado, pero también el tipo de trayectoria que quiere realizar el usuario.

El envió y la recepción de tramas pudo ser probado a través de comunicaciones virtuales. La trama recibida en Matlab hizo posible probar realmente la trayectoria definida por las coordenadas transmitidas. Estas coordenadas desarrollan las mismas trayectorias, en espacio o en tiempo, que las que fueron pedidas por el usuario.

Debido a que fue desarrollado y probado solo en simulaciones, es posible que no funcione perfectamente con el robot físico. Pero es un proyecto bastante flexible que permitiría una adaptación sencilla y rápida sobre los puntos que sean necesarios.

Trabajos futuros

La interfaz desarrollada a lo largo de este proyecto estará destinada a mejorar al mismo tiempo que se realicen modificaciones sobre el robot.

Pero, en el contexto actual, el primer paso a dar sería comprobar el funcionamiento de la interfaz con el robot. Además de verificar si las trayectorias siguen el objetivo, es necesario verificar que el control de flujo asegure un envío de datos continuo, sin pérdidas.

Por otro lado, se podría diseñar una interfaz más interactiva con una evolución de la cantidad de datos a entrar dependiendo del número de puntos o de la forma deseada. Así el primer diálogo permitiría la elección del modo de trabajo y el segundo diálogo la geometría de la trayectoria.

Además, otros modos de trabajo podrían ser implementado, como los modos mencionados en el proyecto [1].

Por último, la conexión UART y la interfaz fueron desarrolladas para facilitar los cambios sobre el proyecto. Así que el agregado del eje z en la programación de la interfaz podría ser también posible y deseable.

Bibliografía

[1] Gino Avanzini, Gonzalo Fernández, Jeremías Pino Demichelis. Desarrollo de robot paralelo tipo doble SCARA: diseño, fabricación, simulación, control y experimentación.

[2] Wisama Khalil y Sebastien Briot. Dynamics of Parallel Robots. Springer International, 2015. Isbn : 9783319197883.

[3] Bourbonnais, Bigras y Bonev. Minimum-Time trajectory Planning and Control of a Pick-and-Place Five-Bar parallel robot. 2015.

[4] Wikipedia, *Linealization*, 09-06-2023, https://en.wikipedia.org/wiki/Linearization

[5] Qt, *Qt for beginners,* 16-04-2023, https://wiki.qt.io/Qt_for_Beginners