Universidad Zaragoza
1474

Trabajo Fin de Grado

Guiado remoto 2D de un robot ABB desde una interfaz interactiva diseña en Matlab

Autora

Karen Nicole Mena Rodríguez

Director

Antonio Romeo Tello

Grado en Ingeniería Electrónica y Automática

Escuela de Ingeniería y Arquitectura

Universidad de Zaragoza

Índice

Lista de figuras

Lista de abreviaturas y acrónimos

Acrónimo/Abreviatura	Descripción
RAPID	Lenguaje de programación utilizado para robots ABB.
MATLAB	Software para cálculos matemáticos, simulaciones y programación técnica.
IRC5	Controlador de robots ABB que gestiona su movimiento y operaciones.
TCP/IP	Conjunto de protocolos usados para la comunicación entre dispositivos en redes
IPv4	Protocolo que utiliza direcciones de 32 bits para identificar dispositivos en una red.
IPv6	Protocolo mejorado que utiliza direcciones de 128 bits para redes más grandes y eficientes.
tcpclient	Función de MATLAB que establece una conexión de cliente TCP/IP con un servidor, permitiendo el intercambio de datos a través de la red.
strfind	Función de MATLAB que busca una subcadena dentro de una cadena de texto y devuelve las posiciones de inicio donde se encuentra.
gcbf	Función de MATLAB que devuelve la figura asociada al callback en ejecución.
strcmp	Función de MATLAB que compara dos cadenas de texto y devuelve un valor lógico indicando si son iguales o no.

1 Introducción

1.1 Objetivos del proyecto

El objetivo principal de este proyecto es desarrollar un sistema de comunicación entre un robot físico y una aplicación en *MATLAB* a través del uso de sockets para la transmisión de datos en tiempo real. Este sistema se utilizará para diseñar una herramienta de guiado que permita controlar tanto un robot real como un robot virtual, brindando la posibilidad de ejecutar dos tipos de guiado: trayectoria continua y punto a punto.

Además, el proyecto contempla la evaluación de los retardos inherentes a la comunicación, analizando cómo afectan los tiempos de respuesta y la precisión en la sincronización de los movimientos del robot. Para este fin, se emplearán las opciones de sincronización proporcionadas por el lenguaje Rapid, propio de la programación de robots ABB.

La comunicación se basa en un modelo cliente-servidor, en la que *Robot Studio* actúa como el servidor y la aplicación *MATLAB* como el cliente. Se permite que ambos operen en el mismo ordenador, utilizando la misma dirección IP, aunque también se podrá establecer comunicación entre dos equipos distintos, siempre que se adapten las configuraciones de IP.

El funcionamiento del sistema se basa en la creación de sockets, donde el servidor, después de inicializar, se pone en modo escucha (*SocketListen*) para aceptar solicitudes de conexión del cliente. El cliente, por su parte, establece la conexión con el servidor. Es importante destacar que el servidor solo ejecuta la función *SocketAccept* para aceptar y verificar que el cliente está conectado.

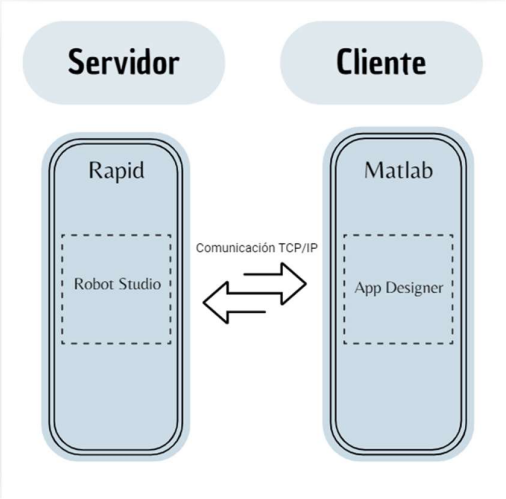


Figura 1. 1 *Esquema básico de los bloques*

2 Herramientas utilizadas

2.1 Robot Studio

RobotStudio, desarrollado por ABB Robotics, es un software avanzado de simulación y programación offline que controla específicamente robots industriales de ABB. Las herramientas de programación de robots virtuales resultan fundamentales en el campo de la automatización industrial; permite a los ingenieros, programadores y operadores diseñar, simular y optimizar de manera eficiente los movimientos y operaciones de los robots.

A través del uso de *RobotStudio*, los usuarios pueden crear entornos de trabajo virtuales que replican con precisión las condiciones reales de la planta. Esto ayuda al usuario a ver las operaciones del robot antes de ejecutarlo realmente, en un entorno seguro y controlado. Además, minimiza los errores de programación, ya que las trayectorias y los movimientos del robot serán perfectamente precisos.

Una de las funcionalidades más atractivas de *RobotStudio* es su capacidad de ejecutarse sin conexión. Con esta característica, los usuarios pueden programar y probar trayectorias y movimientos del robot en un modelo virtual sin afectar la producción real. Esta es una funcionalidad muy útil para la industria, ya que se pueden realizar modificaciones y ajustes más importantes en los sistemas de automatización sin poner en riesgo el equipo o el flujo de trabajo. Además, el software incluye un editor de programas con simulador, lo que permite visualizar fácilmente el comportamiento del robot en un entorno virtual antes de la implementación física del código y de su modificación. Esta visualización en una etapa temprana puede ser muy importante, ya que permite al programador detectar y corregir posibles errores antes de que afecten al entorno real.

La integración con otras soluciones de software, como *MATLAB*, permite fusionar las capacidades de análisis de datos y algoritmos de *MATLAB* con lo que se puede hacer en *RobotStudio* en relación con la programación de robots. Además, la comunicación TCP/IP admite el control remoto del robot en tiempo real, que es un factor muy esencial en los entornos de automatización avanzados.

La aplicación se diseñará inicialmente en un entorno virtual sobre *RobotStudio*, y luego se trasladará a la ejecución sobre un robot real tras la depuración y las pruebas necesarias.

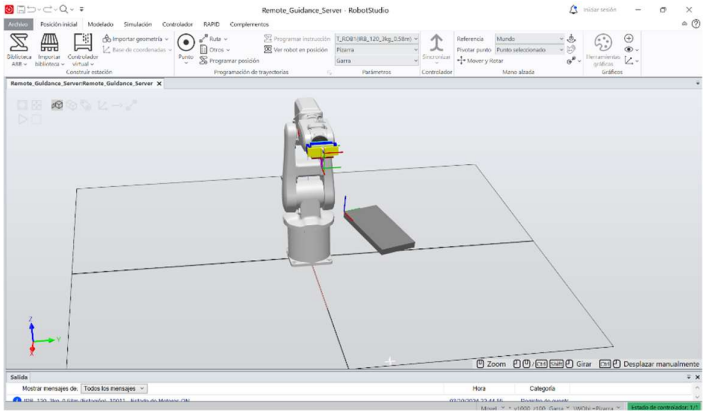


Figura 1. 2 *Interfaz de Robot*

2.2 Matlab

MATLAB es un entorno de programación integral y una plataforma de desarrollo ampliamente utilizado en ingeniería, ciencia e investigación. Su versatilidad y potentes capacidades de análisis de datos lo convierten en una herramienta esencial en muchos campos, incluido el procesamiento de señales, el análisis estadístico y la simulación dinámica de sistemas. En este contexto, *App Designer* se considera una de las herramientas más innovadoras y efectivas para crear aplicaciones interactivas en *MATLAB*, permitiendo a los usuarios diseñar interfaces gráficas de usuario (GUI) de una manera intuitiva y eficiente. Además, en lo relativo a las funciones de comunicación, es importante destacar que se utilizan las funciones y datos proporcionados por la *Instrument Control Toolbox* de *MATLAB*, que permite la comunicación.

Descripción del diseñador de aplicaciones.

App Designer es un entorno de desarrollo integrado (IDE) que facilita la creación de aplicaciones interactivas mediante un método de arrastrar y soltar componentes de interfaz gráfica. Este entorno no sólo permite a los usuarios crear interfaces de usuario atractivas y con todas las funciones, sino que también simplifica el proceso de programación al proporcionar un entorno vinculante en el que tanto la interfaz de usuario como la lógica de programación se pueden integrar en un solo archivo.

Los usuarios pueden crear una variedad de componentes de interfaz, incluidos botones, cuadros de texto, gráficos, tablas, etc., que son esenciales para una interacción eficaz del usuario. Una de las características más notables de *App Designer* es su capacidad para generar código automáticamente. Al arrastrar y soltar componentes en la interfaz, *MATLAB* crea automáticamente el código correspondiente en el lenguaje de programación *MATLAB*. Esta característica no sólo acelera el proceso de desarrollo sino que también permite a los desarrolladores centrarse en la lógica de la aplicación en lugar de en los detalles de la interfaz.

Funciones de App Designer

Diseño Visual de Interfaz: *App Designer* proporciona una interfaz de diseño donde los usuarios pueden colocar y organizar elementos de la interfaz. Capacidad para ver cómo se ve su aplicación en tiempo real para que pueda realizar ajustes de inmediato y mejorar la experiencia del usuario.

Componentes Interactivos: *App Designer* incluye una variedad de componentes interactivos para crear aplicaciones ricas y dinámicas. Estos componentes incluyen botones, menús desplegables, controles deslizantes y gráficos interactivos. Estos elementos permiten a los usuarios interactuar con la aplicación de diversas formas, mejorando la usabilidad y la experiencia general.

El uso de *MATLAB* con *App Designer* representa una poderosa combinación para el desarrollo de aplicaciones interactivas en diversos campos de la ciencia y la ingeniería. Su capacidad para integrar cálculos complejos, visualización de datos y una interfaz de usuario intuitiva la convierte en una herramienta valiosa para profesionales y académicos. A medida que la tecnología avanza y la demanda de soluciones interactivas sigue creciendo, la relevancia de *MATLAB* y *App Designer* en el desarrollo de aplicaciones seguirá en aumento.

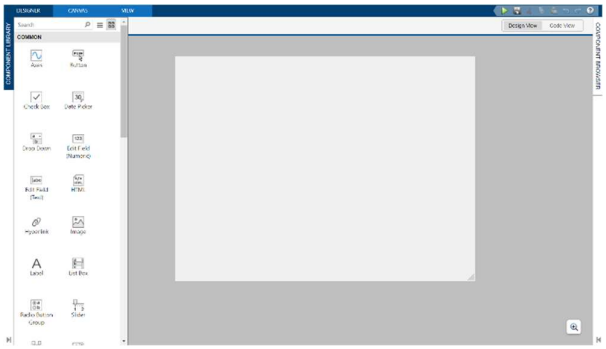


Figura 1. 3 *Interfaz App Designer*

2.3 Controlador IRC5

El controlador IRC5 fue desarrollado por ABB Robotics, es un controlador ampliamente utilizado en aplicaciones de automatización industrial. Desde su lanzamiento, ha sido reconocido por su capacidad para adaptarse a diversas necesidades de producción, lo que lo convierte en una herramienta eficaz para la automatización de procesos. En un entorno de creciente automatización y robótica, el IRC5 se presenta como una solución que mejora la productividad, la flexibilidad y la calidad en las operaciones industriales.

Uno de los aspectos clave del IRC5 es su lenguaje de programación nativo, RAPID, diseñado para programar robots industriales. RAPID es un lenguaje de alto nivel que permite a los operadores desarrollar programas complejos de manera eficiente. Admite programación de movimiento, lógica de control y manipulación de datos, lo que lo convierte en una poderosa herramienta para la automatización de procesos. Los movimientos programados en RAPID se realizan a través de comandos específicos que permiten definir trayectoria, velocidad y aceleración, entre otros parámetros. Esto asegura que el robot opere de manera precisa y eficiente, adaptándose a las necesidades de cada tarea.

Además de la programación de movimiento, RAPID también admite estructuras de control como iterativas y condicionales, lo que permite a los usuarios implementar lógica compleja en sus aplicaciones. Esto es particularmente útil en entornos donde las decisiones deben tomarse en tiempo real.

El entorno de desarrollo de software del IRC5 proporciona una interfaz fácil de usar que permite a los usuarios crear, modificar y simular programas para el robot de manera eficiente. Esta

interfaz cuenta con herramientas de simulación que permiten visualizar el movimiento del robot en el entorno virtual antes de que se ejecute realmente. Esta función no solo ahorra tiempo y recursos, sino que también reduce la probabilidad de errores durante la programación e incluso en la fase de implementación.

La simulación, como elemento del proceso de programación del robot, permite a los operadores visualizar cómo funcionará el robot en diferentes situaciones. Esto es notable en cualquier entorno industrial, especialmente donde el espacio disponible es limitado o donde las interacciones con otro equipo pueden ser cruciales. La posibilidad de ver el movimiento del robot antes de implementarlo en el entorno real brinda a los usuarios la oportunidad de perfeccionar las trayectorias y hacer preparativos antes de las producciones.

Figura 1. 4 *ABB IRC5*

3 Protocolo de comunicación

3.1 Descripción del protocolo TCP/IP

El protocolo TCP/IP (Transmission Control Protocol / Internet Protocol) establece una serie de reglas y estándares que posibilitan una adecuada comunicación en una red que comprende desde redes locales (LAN) hasta una red global como Internet. Este modelo es la espina dorsal de Internet, así como de otras redes más modernas, y se encuentra constituido por varias capas que cooperan para una transmisión de los datos adecuada y eficiente.

Capa red (IP). Se encarga de la dirección, el enrutamiento y la entrega de los paquetes de datos entre un emisor y un receptor en la red. Cualquier dispositivo que se conecta a la red presenta una dirección IP que le permite identificarse dentro de ella y facilita a los dispositivos comunicarse entre sí.

La misión IP consiste fundamentalmente en descomponer los datos en pequeños bloques o paquetes y atribuirles una dirección de origen y una dirección de destino. Posteriormente, estos paquetes serán enrutados a través de la red pasando por varios nodos (routers, etc.) hasta alcanzar la dirección final. En esta fase es importante destacar que, mediante el protocolo IP, se garantiza que los paquetes sigan el camino correcto independientemente de la necesidad de atravesar varios puntos intermedios.

Existen dos versiones principales del Protocolo de Internet:

- **IPv4:** Utiliza direcciones de 32 bits, lo que permite aproximadamente 4.3 mil millones de direcciones únicas (ejemplo: 192.168.1.1).
- **IPv6:** Emplea direcciones de 128 bits, lo que proporciona una cantidad significativamente mayor de direcciones, diseñada para la expansión futura de Internet

Capa de transporte (TCP): Los datos lleguen de forma íntegra, en el correcto orden y sin errores. TCP realiza la segmentación de los datos en pequeños trozos o segmentos de tamaño no inferior al máximo definido y se ocupa de su envío. O sea, a medida que el receptor los recibe TCP comprobará que estos segmentos sean recibidos de manera correcta y en el correcto orden. En caso de detectar cualquier error o pérdida de segmentos durante la transmisión este los retransmitirá automáticamente. A la vez TCP incluye mecanismos de control de flujo mediante los cuales ajusta la velocidad de transmisión a la capacidad de la red, a fin de evitar sobrecargarla y, en consecuencia, minimizar el riesgo de congestión.

El protocolo TCP/IP es esencial para asegurar la comunicación efectiva entre los diferentes sistemas, independientemente de sus características o de la plataforma. Además, la capacidad de escalar desde una red local y hasta la Internet global lo han convertido en el protocolo más flexible y robusto de la actualidad, pues TCP/IP fue diseñado para gestionar redes de cualquier tamaño, desde, por ejemplo, intranets privadas, hasta redes corporativas y, por supuesto, la Internet.

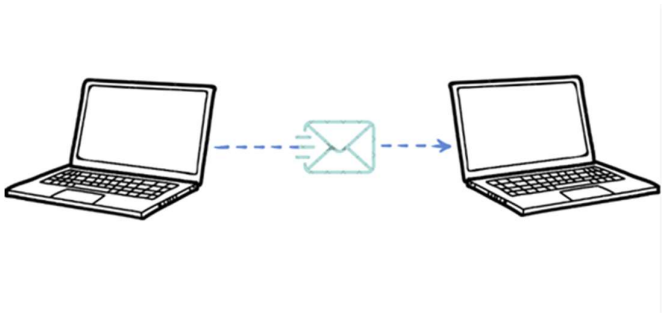


Figura 1. 5 *TCP/ IP*

3.2 Establecimiento de la comunicación

Para establecer la comunicación entre *MATLAB* y *RobotStudio* mediante el protocolo TCP/IP, es fundamental configurar adecuadamente una dirección IP y un puerto que sean accesibles para ambos sistemas. Este proceso se lleva a cabo siguiendo una serie de pasos que permiten la creación y gestión de un socket, así como la aceptación de conexiones entrantes.

En primer lugar, se definen las variables necesarias para la comunicación. En este contexto, se utilizan las siguientes declaraciones:

```
VAR socketdev servidor;
VAR socketdev cliente;
VAR string IP_servidor:="10.3.17.176";
VAR num puerto:=4000;
```

Figura 1. 6 *Definición de las variables en RobotStudio*

Aquí, servidor y cliente son objetos de tipo socket, mientras que IP_servidor representa la dirección IP del servidor que está escuchando las solicitudes, y puerto es el número de puerto utilizado para la conexión.

A continuación, se inicia el proceso de conexión creando un socket para el servidor. Esto se logra utilizando la función *SocketCreate*, que establece un nuevo socket en el entorno de *RobotStudio*. Una vez creado el socket, se asocia a una dirección IP y un número de puerto específicos mediante la función *SocketBind*. Esto garantiza que el socket esté vinculado a la configuración de red deseada, permitiendo la recepción de conexiones en esa dirección y puerto.

Posteriormente, el servidor entra en un estado de escucha utilizando *SocketListen*. En este estado, el servidor se prepara para recibir conexiones entrantes de clientes, lo que le permite estar disponible para cualquier solicitud que pueda surgir.

El siguiente paso implica la aceptación de una conexión entrante por parte del servidor. Esto se logra con la función *SocketAccept*, que permite al servidor esperar hasta un tiempo límite (en este caso, 60 segundos) para aceptar la conexión de un cliente. Esta función también crea un socket específico para la comunicación con el cliente.

Una vez que se establece la conexión, el servidor envía un mensaje de confirmación al cliente utilizando *SocketSend*. Este mensaje puede ser un simple texto que indica que la comunicación ha sido aceptada, proporcionando al cliente una señal de que está listo para continuar.

Para mantener un registro visual de la comunicación en el entorno de *RobotStudio*, se utiliza la función TPWrite. Este paso es útil para depurar y verificar el estado del sistema, mostrando en la consola del robot un mensaje que confirma la aceptación de la comunicación.

```
SocketCreate servidor;
SocketBind servidor, IP_servidor, puerto;
SocketListen servidor;
SocketAccept servidor, cliente, \Time:=60;
SocketSend cliente,\str:="COMUNICACION ACEPTADA";
TPWrite "COMUNICACION ACEPTADA";
```

Figura 1. 7 *Establecimiento de la conexión en RobotStudio*

3.3 Comunicación cliente-servidor

3.3.1. Transmisión de datos desde MATLAB.

La vinculación que se realiza entre la aplicación de *MATLAB* y el entorno de *RobotStudio*, se llevará a cabo haciendo uso del protocolo TCP/IP. Para ello, la propia interfaz debe disponer de un botón "Conectar" que una vez pulsado procederá a realizar la conexión con *RobotStudio* con la finalidad de comenzar a recibir datos de las trayectorias y los movimientos del robot.

La forma de transmitir datos desde *MATLAB* se debe realizar haciendo uso de un cliente TCP, que se configura para conectarse a un servidor situado en *RobotStudio*. Para ello, el código utiliza las variables Puerto y Dirección para establecer los parámetros de dicha conexión, creando, por ende, un objeto tcpclient que facilita la comunicación entre la aplicación de *MATLAB* y el entorno de *RobotStudio*.

Debemos señalar que el valor del puerto, el cual en principio se encuentra definido como un string, debe ser convertido en número utilizando la función str2double puesto que la función tcpclient espera que el puerto definido sea una entrada numérica como argumento. Se necesita, de igual forma, establecer un timeout de 50 segundos para la conexión. Esto quiere decir que de no poder establecer la conexión en el tiempo que se encuentra especificado se generará un error.

Se ha incorporado un elemento visual de tipo app.lamp que indica el estado de la conexión. Al establecerse la conexión, el indicador se ilumina en verde, proporcionando una referencia clara y visual del estado de la comunicación.

Una vez configurado el cliente de esta manera, se implementa un pause con el propósito de garantizar que el cliente esté completamente preparado para recibir datos. Después de este breve intervalo, el cliente permanece a la espera de un mensaje enviado por el servidor con un tamaño mínimo de 21 bytes. Una vez que el cliente ha recibido el mensaje es capaz de leerlo y transformarlo a caracteres con la función char(read(app.cliente)), almacenando en el propio script el resultado para su posterior tratamiento.

```
% Value changed function: Conectar
function ConectarValueChanged(app, event)
    %Variables
    Puerto = app.Puerto.Value;%string
    Direccion = app.Direccionpuerto.Value;%string
    %Creaccion del cliente
    app.cliente = tcpclient(Direccion,str2double(Puerto),"Timeout",50);
    pause(1);
    app.Lamp.Color=[0.59,0.94,0.43];%esta modo on
    waitfor(app.cliente,'NumBytesAvailable',21);
    mensaje = char(read(app.cliente))
    app.mensaje_1 = mensaje;
```

Figura 1. 8 *Establecimiento de la conexión en Matlab*

3.3.2. Obtención de información desde RobotStudio

El proceso de recepción de mensajes enviados al servidor comienza esperando activamente un mensaje del cliente, el mensaje se almacena en la variable mensaje_recibido, desde aquí comienza un bucle que se ejecuta de forma cíclica mientras el mensaje recibido sea distinto de "FIN COMUNICACION". De este modo se permite al servidor seguir recibiendo y procesando mensajes indefinidamente hasta recibir una indicación explícita que le indique que debe dejar de procesar mensajes.

Por cada mensaje que se recibe se incrementa un contador que tiene la función de contar el número total de mensajes hasta el momento. Posteriormente, se procede a analizar el mensaje para conseguir la información que se requiere, utilizando para ello la función StrFind, que permite determinar las posiciones de las comas que determinan el origen de los distintos datos contenidos en el mensaje. La porción del mensaje anterior a la primera coma se interpreta como la coordenada X, denotada como pos_X, mientras que la sección entre la primera y la segunda coma se identifica como la coordenada Y, representada como pos_Y.

Como resultado de este proceso, y de cara a las operaciones posteriores, se determina la longitud total del mensaje recibido. Con toda esta información, conseguimos extraer el valor de la coordenada X, transformando así la cadena anterior a la primera coma en un número. De igual modo, conseguimos extraer la coordenada Y. La coordenada Y se encuentra situada entre la primera y la segunda coma.

Finalmente, se determina el modo de operación a partir del contenido que queda a continuación de la segunda coma del mensaje y que indica el modo de operación como "continuo" o bien "no continuo", siendo su identificación necesaria a la hora de establecer la correcta interpretación y respuesta del sistema según las acciones que se quieren desencadenar.

3.3.3. Desconexión desde Matlab

Para llevar a cabo este proceso, la interfaz debe contar con un botón de "Desconectar". Al presionarlo, se procederá a finalizar la conexión con *RobotStudio* este enviará un mensaje de notificación mediante la función write(app.cliente, "FIN COMUNICACION"), asegurando que el cliente reciba una confirmación explícita de que la comunicación ha finalizado.

A continuación, se actualizará la propiedad de app.Lamp para cambiar el indicador visual a un color rojo, lo cual simboliza el estado de desconexión del sistema.

Con el fin de asegurar un cierre adecuado y controlado de la conexión, se empleará una breve pausa. Finalmente, se limpiará la conexión mediante clear app.cliente, concluyendo así de manera eficaz la comunicación con el cliente.

3.4 Escenario Virtual y Real

Cuando trabajamos en un escenario de simulación virtual en *RobotSudio*, asignamos un WorkObject denominado Pizarra_virtual, el cual tiene un sistema de coordenadas específico. Estas coordenadas no coinciden exactamente con las del WorkObject asociado a la posición de la caja en el entorno físico real. Es decir, aunque el programa simulado para el control del robot es prácticamente el mismo que en el entorno real, existen diferencias importantes, ya que las coordenadas de la caja en el entorno virtual no son exactamente las mismas que las de la caja real.

Para el entorno real, hemos definido un WorkObject llamado Pizarra_real, lo cual asegura que la ubicación y orientación de la caja sean coherentes tanto en el entorno simulado como en el real.

4 Aplicación Gráfica en MATLAB

El presente capítulo aborda la generación e implementación de una aplicación gráfica interactiva mediante *App Designer* de *MATLAB*, con el objetivo de capturar y manejar la información de las coordenadas en tiempo real para su acoplamiento a sistemas robóticos. La creación de la aplicación se persigue bajo un único objetivo: mejorar la interacción entre el usuario y el entorno automatizado, así como generar una interfaz intuitiva y con eficiencia en el control y la supervisión de trayectorias y coordenadas del robot en tiempo real. La solución gráfica debería contribuir a la maximización de la usabilidad, facilitando la monitorización y ajuste de los parámetros de operación del sistema robótico.

La finalidad de esta aplicación es ofrecer una plataforma de tipo visual con un uso amigable que persiga la captura, el ajuste y la visualización de las coordenadas de un robot industrial en tiempo real, buscando mejoras en la operatividad del control y en la precisión de los procesos automatizados. Además, la aplicación obtiene una descripción dinámica del comportamiento del sistema robótico, ajustando en tiempo real y permitiendo al usuario obtener la visión del movimiento y los parámetros de operación del robot; la facilidad de uso de esta herramienta tiene el efecto de maximizar la supervisión y ajuste de trayectorias, eliminando errores y mejorando la eficiencia operativa en ambientes automatizados.

4.1 Desarrollo de una interfaz gráfica con App Designer

Este entorno visual facilita la creación de interfaces gráficas mediante la capacidad de arrastrar y soltar componentes y definir su comportamiento mediante devoluciones de llamada. Esta característica es particularmente valiosa para proyectos que requieren una interacción continua entre el usuario y el sistema, como en el caso de esta aplicación, diseñada para la manipulación de coordenadas del robot.

La interacción en tiempo real con el sistema robótico se organiza mediante callbacks y eventos asociados con diferentes componentes de la interfaz. Estos elementos responden a acciones del usuario, como modificar el valor en un campo de entrada o activar un botón.

Gracias a estos eventos, *MATLAB* actualiza y gestiona el estado de la interfaz, permitiendo la ejecución de simulaciones y la visualización de resultados en tiempo real, lo que garantiza una interacción fluida y efectiva.

A continuación, se describen algunos de los aspectos clave en el diseño de la interfaz:

- **Callbacks:** Los callbacks se utilizan para gestionar la interacción del usuario con los componentes de la interfaz gráfica. Esta configuración garantiza que cada acción realizada por el usuario se procese en tiempo real, permitiendo actualizaciones dinámicas y precisas de la visualización de la trayectoria del robot.

- **Gráficos dinámicos:** La representación visual de las trayectorias y comportamientos de los robots se logra mediante el uso de componentes gráficos (por ejemplo, uiaxes) integrados en *MATLAB*. Estos gráficos permiten visualizar las coordenadas del robot y su trayectoria en un espacio tridimensional, proporcionando al usuario una representación clara y precisa del funcionamiento del sistema. El componente uiaxes se actualiza dinámicamente con cada interacción del usuario, ofreciendo un entorno de simulación robusto y visualmente informativo.

4.2 Captura y manejo de coordenadas en tiempo real

Uno de los desafíos clave en el desarrollo de esta aplicación gráfica es capturar y administrar las coordenadas del robot en tiempo real, lo cual es esencial para monitorear adecuadamente la trayectoria del robot y ajustar su movimiento en función de las condiciones del entorno o las necesidades del proceso.

En esta aplicación desarrollada con App Designer, el comando app.UIAxes.CurrentPoint desempeña un papel central porque hace referencia a la propiedad CurrentPoint del eje de gráficos UIAxes. Esta propiedad devuelve las coordenadas tridimensionales (x, y, z) del punto en el espacio de datos donde se encuentra el cursor del mouse en un momento dado. Esta característica es particularmente útil para determinar la ubicación exacta del cursor en un gráfico. Es importante señalar que el tiempo real dependerá de los retrasos inherentes a la comunicación y el procesamiento computacional. Estos retrasos pueden ser suficientes para afectar el rendimiento en tiempo real, especialmente si el control está "en línea". En estos casos, pueden surgir retrasos en los procesos de comunicación y ejecución de instrucciones, incluidas, por supuesto, las relacionadas con el movimiento del robot.

4.2.1. Función gcbf y su aplicación

La función gcbf (Get Current Button Figure) se utiliza para obtener el identificador de la figura (ventana gráfica) que contiene el objeto cuya devolución de llamada o función de devolución de llamada se está ejecutando actualmente. En otras palabras, gcbf permite identificar qué figura o ventana de *MATLAB* está activa en el momento en que se llama a esta función.

El siguiente fragmento de código muestra el uso de gcbf junto con la función get, que se emplea para acceder a las propiedades de objetos gráficos en *MATLAB*:

get(gcbf, 'SelectionType')

La función get recupera el valor de una propiedad específica de un objeto gráfico. En este caso, get(gcbf, 'SelectionType') obtiene el tipo de acción realizada con el ratón en la figura activa, el cual puede ser uno de los siguientes valores:

- **'normal':** Un clic con el botón izquierdo del ratón.
- **'alt':** Un clic con el botón derecho del ratón.
- **'extend':** Un clic con el botón central del ratón o la rueda.
- **'open':** Un doble clic.

En la siguiente línea de código, se utiliza la función strcmp para comparar el valor devuelto por SelectionType con la cadena 'normal':

strcmp(get(gcbf, 'SelectionType'), 'normal')

La función strcmp devuelve verdadero si las dos cadenas son iguales, falso en caso contrario. Por lo tanto, este código evalúa si la operación del mouse corresponde a un clic izquierdo. De esta forma, la comparación sirve para verificar que el usuario mantiene presionado el botón izquierdo del mouse sobre la figura activa, activándose así un bucle que continúa ejecutándose mientras se mantenga esta condición.

4.2.2. Implementación de Callbacks para la Actualización Continua de Coordenadas

La aplicación también implementa callback para actualizar continuamente las coordenadas del robot en tiempo real, lo que permite una interacción fluida con el sistema. En este caso, el evento del botón del mouse presionado en los UIAxes se utiliza para capturar las coordenadas actuales del cursor.

El siguiente fragmento de código muestra cómo se implementa el callback que reacciona al evento de movimiento del ratón sobre el área de los UIAxes:

```
function UIFigureWindowButtonMotion(app, event)

    % Código para manejar el movimiento del ratón

end
```

En cuanto al manejo de la interacción con el botón del ratón sobre el gráfico, comienza verificando si el mensaje recibido desde el servidor contiene el texto "COMUNICACION ACEPTADA". Esto garantiza que la conexión entre la aplicación y el servidor está activa antes de proceder con la captura de coordenadas. La instrucción:

```
if contains(app.mensaje_1, "COMUNICACION ACEPTADA")
```

asegura que solo si se ha establecido la comunicación con éxito se procederá con las siguientes operaciones. Se inicializan las coordenadas x e y con valores de 0, representando el punto inicial en el gráfico.

El punto inicial en el gráfico, correspondiente a las coordenadas (0,0), se dibuja utilizando la función plot. El marcador es un círculo ('o') con relleno azul:

```
plot(app.UIAxes, x, y, 'o', 'MarkerFaceColor', 'b');

hold(app.UIAxes, 'on');
```

La instrucción hold(app.UIAxes, 'on') permite mantener los puntos anteriores en el gráfico a medida que se añaden nuevos puntos sin que el gráfico se borre.

A continuación, el código entra en un bucle while que sigue ejecutándose mientras el usuario mantenga el botón izquierdo del ratón presionado sobre el gráfico. Este bucle utiliza la función get(gcbf, 'SelectionType') para comprobar que la acción del ratón es un clic izquierdo. Dentro del bucle, el código captura las coordenadas actuales del cursor utilizando la propiedad CurrentPoint del objeto UIAxes:

```
point = app.UIAxes.CurrentPoint;

x = point(1, 1);  % Coordenada X

y = point(1, 2);  % Coordenada Y
```

Las coordenadas obtenidas se almacenan en las variables app.LastX y app.LastY para mantener un registro del último punto capturado:

```
app.LastX = x;

app.LastY = y;
```

Para facilitar la transmisión de las coordenadas al servidor, las coordenadas x e y se convierten a cadenas de texto formateadas con un decimal:

```
x_pos = num2str(x, '%.1f');

y_pos = num2str(y, '%.1f');
```

A continuación, se construye una línea de texto que incluye las coordenadas x e y, y se agrega el valor 'SI', lo que indica que el sistema está en modo continuo. Esta cadena de texto se envía al servidor utilizando la función write:

```
line = [x_pos, ',', y_pos, ',', 'SI'];

write(app.cliente, line);
```

Finalmente, se lee una respuesta del servidor mediante la función read, que recibe dos caracteres en formato de cadena:

```
mensaje = read(app.cliente, 2, "char");
```

A través del uso de callbacks y la propiedad CurrentPoint, se capturan y actualizan las coordenadas del cursor del ratón mientras el usuario interactúa con el gráfico. Además, las coordenadas se envían de manera continua al servidor para su procesamiento, lo que permite una monitorización y ajuste dinámico de los movimientos del robot en respuesta a las interacciones del usuario.

4.2.3. Advertencia visual y textual al exceder los márgenes en la Interfaz Gráfica

Si el punto (x, y) se encuentra fuera de los márgenes predefinidos, la interfaz gráfica presenta una advertencia tanto visual como textual. En primer lugar, se muestra un mensaje en el campo de texto denominado Advertencia, informando al usuario que el punto ha excedido los límites establecidos.

Además, para aumentar la visibilidad de la advertencia, se activa un efecto visual en la interfaz: la imagen de advertencia Image_advertencia parpadea alternativamente, alternando entre su estado visible e invisible a intervalos de 0.25 segundos. Este parpadeo tiene como objetivo atraer la atención del usuario y asegurarse de que la advertencia se perciba de manera clara y destacada.

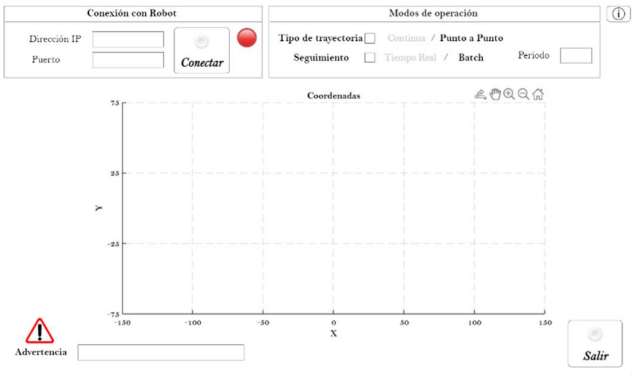


Figura 1. 9 *Interfaz del cliente*

4.2.4. Función de Ayuda para la Guía del Usuario

Se ha desarrollado una función destinada a ofrecer instrucciones completas y claras sobre el uso de la aplicación. Esta función se activa cuando el usuario hace clic en un botón de ayuda dentro de la interfaz. Al presionar dicho botón, se muestra un cuadro de alerta que contiene un mensaje estructurado con explicaciones detalladas sobre las diversas funcionalidades de la aplicación.

El mensaje de ayuda está diseñado para guiar al usuario de manera eficiente en la interacción con la interfaz, describiendo los pasos a seguir y las opciones disponibles.

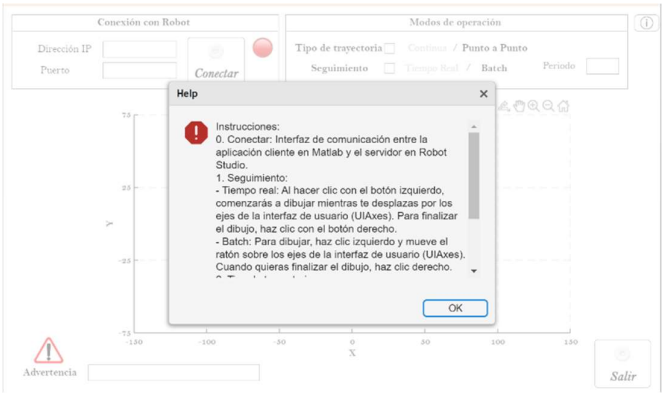


Figura 1. 10 *Instrucción de ayuda*

5 Modos de Operación

Se describen los distintos modos de operación implementados en el sistema de guiado del robot ABB120. Se detallan tanto el Modo Continuo como el Modo Punto a Punto, abordando sus definiciones, funcionamiento e implementación en Tiempo Real y Modo Batch.

Para habilitar los cuatro modos de operación (Continua y Punto a Punto en Tiempo Real y en Modo Batch), se implementa una codificación binaria que utiliza dos variables booleanas. Cada combinación de estas variables representa un modo específico:

- **1,1:** Modo Continuo en Tiempo Real
- **0,1:** Modo Continuo en Modo Batch
- **1,0:** Modo Punto a Punto en Tiempo Real
- **0,0:** Modo Punto a Punto en Modo Batch

Este enfoque asegura que el usuario pueda seleccionar fácilmente el modo deseado a través de una interfaz gráfica, permitiendo una codificación clara y estructurada de los diferentes modos de operación.

5.1 Modo Continuo

El Modo Continuo se caracteriza por la creación de una trayectoria fluida y sin interrupciones, permitiendo que el robot siga una línea continua en el espacio de trabajo. Este modo es ideal para dibujar curvas suaves y trayectorias complejas que requieren un movimiento constante y continuo del robot. A diferencia del Modo Punto a Punto, donde la trayectoria se define por segmentos lineales entre puntos específicos, el Modo Continuo genera una secuencia de pequeños segmentos que, al unirse, forman una curva suave.

En este modo, la comunicación entre la aplicación *MATLAB* y el robot se establece de dos formas: en Tiempo Real o en Modo Batch. La elección entre estos dos submodos determina cómo se envían y ejecutan las trayectorias definidas por el usuario.

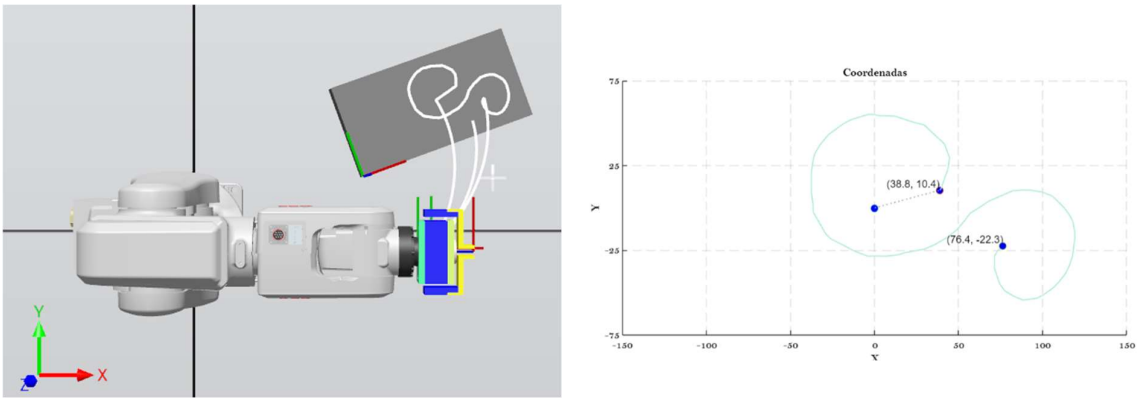


Figura 1. 11 *Trayectoria Continua*

5.1.1 Implementación en Tiempo Real

La implementación en Tiempo Real del Modo Continuo permite que el robot siga la trayectoria mientras el usuario la está trazando en la interfaz gráfica. Cada vez que el usuario dibuja una pequeña parte de la trayectoria, la aplicación *MATLAB* envía estos datos al robot de manera inmediata. El robot, a su vez, ejecuta estos movimientos en tiempo real, respondiendo dinámicamente a las actualizaciones de la trayectoria.

Este enfoque es ideal para aplicaciones donde se requiere una respuesta inmediata del robot, permitiendo una interacción fluida y adaptativa. Sin embargo, es importante considerar que la comunicación en tiempo real puede estar sujeta a retardos inherentes que podrían afectar la precisión y suavidad del movimiento del robot. Por ese motivo, se hace necesario un modo de operación que permita el registro mas preciso de la trayectoria, sin que se vea afectado por los retardos en la comunicación: el modo Batch.

5.1.2 Implementación en Modo Batch

En contraste con el Tiempo Real, la implementación en Modo Batch del Modo Continuo implica que el usuario complete toda la trayectoria antes de enviarla al robot para su ejecución. Es decir, el usuario dibuja toda la línea continua primero y, una vez finalizada, la trayectoria completa se envía al robot mediante un comando (por ejemplo, un clic derecho).

Este enfoque permite al usuario revisar y ajustar la trayectoria antes de que el robot la siga, asegurando una mayor precisión en la ejecución final. Durante el modo Batch, el robot permanece en espera hasta que se le proporciona la trayectoria completa, lo que elimina la necesidad de manejar actualizaciones continuas en tiempo real y mejora la precisión en el registro de la trayectoria.

5.2 Modo Punto a Punto

La definición del Modo Punto a Punto permite la definición de puntos específicos que el robot ha de seguir secuencialmente, generando una trayectoria por segmentos. Un punto definido corresponde a una posición del espacio de trabajo del robot y el movimiento entre dos puntos es por una línea recta. Este modo es adecuado para trayectorias en las que la continuidad no es exigida, por ejemplo, movimientos entre posiciones discretas, tareas que implican movimientos por segmentos, etc.

Frente al modo continuo en el que la trayectoria se define continuamente, en Modo Punto a Punto se ha de definir cada punto de forma explícita, permitiendo de este modo mayor control sobre cada segmento de la trayectoria.

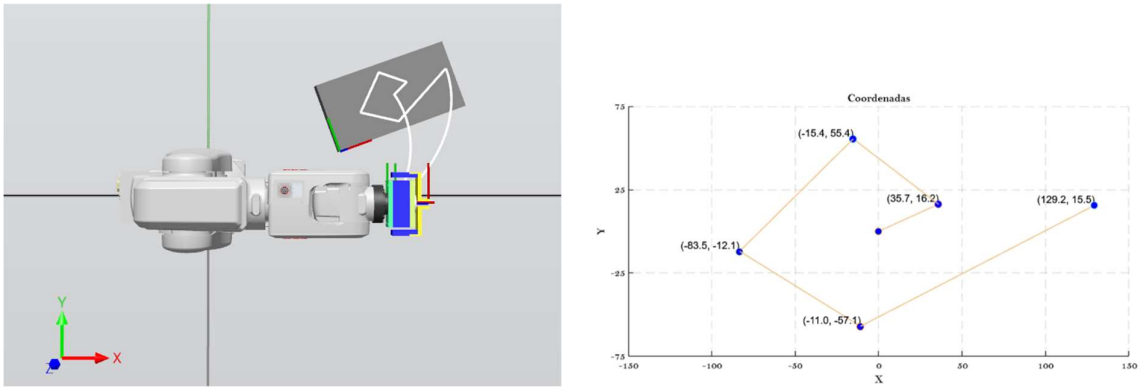


Figura 1. 12 *Trayectoria Punto a punto*

5.2.1 Implementación en Tiempo Real

La implementación en Tiempo Real del Modo Punto a Punto hace que el robot se desplace a cada nuevo punto definido por el usuario de forma inmediata. Cada vez que el usuario selecciona un nuevo punto en la interfaz gráfica de *MATLAB*, este se envía al robot, que ejecuta el movimiento hacia esa posición en tiempo real. Este procedimiento facilita una interacción interactiva donde el usuario puede observar y ajustar el movimiento del robot conforme define cada punto.

5.2.2 Implementación en Modo Batch

En la implementación en Modo Batch del Modo Punto a Punto, el usuario define previamente todos los puntos de la trayectoria y a continuación, envía la lista de puntos al robot para su ejecución. Esto hace que el robot espere hasta recibir todos los puntos para comenzar a moverse, ejecutando la trayectoria por segmentos de una sola vez.

Este enfoque es útil cuando se requiere una planificación previa de la trayectoria completa, permitiendo al usuario optimizar y ajustar todos los puntos antes de la ejecución. Además, al evitar la comunicación continua durante la definición de puntos, se minimizan los posibles retardos y se mejora la eficiencia de la ejecución.

5.3 Configuración de la velocidad en los diferentes modos

En este apartado, analizamos cómo estas configuraciones impactan en la ejecución de trayectorias, especialmente en los modos continuo y punto a punto.

Velocidades y Zonas de Precisión

- **Velocidades:** Se refiere a la velocidad con la que el robot se desplaza entre los puntos de la trayectoria.
  Velocidades fijas:
  - **Vbaja:** El robot se desplaza lentamente, lo que le permite realizar movimientos más suaves y con mayor precisión. Sin embargo, si la velocidad es excesivamente baja, como en el caso de **v50**, puede hacer que el robot se pierda detalles de la trayectoria y no siga correctamente las curvas. Esto es debido a que el robot tarda mucho en llegar al punto, y el tiempo para dar por finalizada la instrucción se vuelve problemático, lo que provoca retrasos.

- **VAlta:** En este modo, el robot se mueve rápidamente entre los puntos, lo que permite completar trayectorias más largas en menos tiempo. Sin embargo, si la velocidad es demasiado alta (como **v500**), el robot tiene un comportamiento nervioso, y el muestreo de la trayectoria se hace más impreciso, generando movimientos bruscos que dificultan el seguimiento de trayectorias complejas.
- **Vmedia:** Este es un punto intermedio entre la velocidad baja y la alta. Una **v300** puede ser ideal, ya que permite que el robot siga la trayectoria sin ser excesivamente nervioso, manteniendo la suavidad del movimiento, especialmente cuando se trata de trayectorias curvas.

Velocidad en continuo:

La velocidad no debe ser fija, ya que debe adaptarse a la velocidad con la que el operador guía al robot. Si el operador mueve el controlador rápidamente, el robot debe aumentar su velocidad, y si el operador guía lentamente, el robot debe reducir su velocidad. Esto asegura que el movimiento del robot siga de cerca las acciones del operador.

Para lograr esto, la velocidad se calcula dinámicamente utilizando la fórmula:

$$Velocidad = \frac{Distancia}{Tiempo}$$

Donde:

- **Distancia:** Es la distancia entre el punto anterior y el punto actual.
- **Tiempo:** Es el tiempo transcurrido desde el último cálculo de velocidad.

Sin embargo, al calcular la velocidad de esta manera, existe el riesgo de que la velocidad calculada sea excesivamente alta o baja debido a fluctuaciones en el tiempo o en la distancia. Para evitar esto, se imponen límites en la velocidad:

- **Velocidad mínima:** 10
- **Velocidad máxima:** 5000
- **Duración mínima del movimiento:** 0.01 segundos

Esto asegura a que la velocidad del robot se ajuste a las condiciones del entorno y las acciones del operador, manteniendo el control del movimiento dentro de un rango seguro y eficiente.

- **Zonas de precisión (fine, z20):** Definen el radio de tolerancia en torno al punto objetivo que el robot debe alcanzar. Cada zona específica un área dentro de la cual el robot puede considerar que ha llegado al punto objetivo, permitiendo cierta flexibilidad en su trayectoria. Las zonas más grandes, como z20, permiten que el robot avance hacia el siguiente punto antes de alcanzar con precisión el actual. La configuración fine, por otro lado, exige que el robot llegue exactamente al punto antes de proceder al siguiente movimiento, lo que introduce un pequeño tiempo de espera, dedicado a la comprobación precisa de la localización.

Uso de "z20" y "fine" en Modos Continuo y No Continuo

Basándonos en los experimentos realizados, es recomendable utilizar z20 cuando se trabaja en el modo continuo, ya que permite una ejecución más fluida y precisa de trayectorias curvas. Esta configuración permite al robot avanzar por la trayectoria sin tener que detenerse completamente en cada punto, lo que resulta en un trazado más suave y se ajustará a la trayectoria definida por el usuario. Sin embargo, los inevitables retardos en la comunicación ocasionarán que la aproximación poligonal a la trayectoria curva especificada sea más o menos aproximada, dependiendo de la magnitud de dichos retardos.

Sin embargo, cuando se trabaja en el modo no continuo, es preferible utilizar la configuración fine. Esto asegura que el robot llegue exactamente a cada punto antes de continuar, lo cual es crucial para trayectorias segmentadas donde la precisión es primordial y la anticipación de puntos no es necesaria ni beneficiosa.

5.4 Aceleración del Robot

Para configurar la aceleración del robot, se ha utilizado la función AccSet, que ajusta la aceleración en dos direcciones: la aceleración, que regula la velocidad del robot a lo largo de su trayectoria, y la duración de la rampa de aceleración. En este caso, los valores utilizados son 20 para el porcentaje aceleración y 10 para el porcentaje de la rampa de aceleración.

Al optar por una aceleración más baja, se facilita la capacidad del robot para realizar ajustes finos en su trayectoria, lo cual es fundamental para tareas que requieren alta precisión.

6 Evaluación del sistema

6.1 Análisis de Tiempos de Comunicación y ejecución en ordenador único

Los datos utilizados en este estudio han sido recopilados desde un mismo ordenador en distintas pruebas, lo que nos permite calcular la media y la desviación estándar de los tiempos de ejecución en diversas situaciones. El objetivo principal es estimar el tiempo que toma cada ciclo completo de comunicación y movimiento, y analizar cómo varía este tiempo entre diferentes iteraciones.

6.1.1 Caso Portátil-Portátil

Hemos realizado un análisis detallado utilizando varios conjuntos de datos, observando las tendencias y calculando tanto la media como la desviación estándar de los tiempos registrados.

Tiempos de Ejecución Registrados:

1. **Tiempo 1** (83 valores):

   Media: 0.0553 segundos

   Desviación Estándar: 0.0655 segundos

2. **Tiempo 2** (51 valores):

   Media: 0.0971 segundos

   Desviación Estándar: 0.1450 segundos

3. **Tiempo 3** (90 valores):

   Media: 0.0576 segundos

   Desviación Estándar: 0.0759 segundos

4. **Tiempo 4** (79 valores):

   Media: 0.0702 segundos

   Desviación Estándar: 0.0955 segundos

Al analizar la relación entre la media y la desviación estándar de los distintos conjuntos de datos, se observa lo siguiente:

El Tiempo 2 presentan unas desviaciones estándar alta (0.1450 segundos), lo que indica una mayor variabilidad en los tiempos de ejecución, posiblemente debido a valores atípicos (ciclos largos).

Los Tiempos 1 y 3 tienen promedios más bajos y presentan una variabilidad moderada, lo que sugiere un comportamiento más consistente y estable en los ciclos de ejecución, con tiempos más cortos. El Tiempo 4 tiene un promedio más alto que los Tiempos 1 y 3, pero aún presenta una desviación estándar más moderada en comparación con el Tiempo 2.

Para facilitar la comprensión de los datos y observar la distribución de los tiempos de ejecución, se han creado histogramas para cada conjunto de tiempos. Estos histogramas, que se encuentran en el Anexo 9.3.

6.2 Análisis de Tiempos de Comunicación y ejecución en ordenadores distintos

Se han recopilado datos sobre los tiempos de ejecución de un sistema que involucra la comunicación y el movimiento de un robot, controlado remotamente a través de *MATLAB* y *RobotStudio* desde dos ordenadores diferentes.

6.2.1 Caso Pc-Portátil

A continuación, se presentan los resultados de los tiempos medidos para cada conjunto de datos.

Tiempos de Ejecución Registrados:

1. **Tiempo 1** (90 valores):

   Media: 0.0574 segundos

   Desviación Estándar: 0.0491 segundos

2. **Tiempo** 2 (48 valores):

   Media: 0.0881 segundos

   Desviación Estándar: 0.1261 segundos

3. **Tiempo 3** (58 valores):

   Media: 0.1059 segundos

   Desviación Estándar: 0.1610 segundos

4. **Tiempo 4** (68 valores):

   Media: 0.0726 segundos

   Desviación Estándar: 0.1051 segundos

Los tiempos con mayores desviaciones estándar, como los Tiempos 2 y Tiempos 3, presentan una mayor variabilidad en el rendimiento del sistema. Esto podría deberse a las características de la comunicación entre *MATLAB* y *RobotStudio*, o a fluctuaciones en la ejecución del movimiento del robot.

Los Tiempos 1 y Tiempos 4, presentan un comportamiento más estable y menor variabilidad. Un menor grado de variabilidad es deseable en aplicaciones donde la precisión y la consistencia del tiempo de ejecución son cruciales.

Con el propósito de facilitar la interpretación de los datos y analizar la distribución de los tiempos de ejecución, se han elaborado histogramas específicos para cada conjunto de tiempos. Estos gráficos pueden consultarse en el Anexo 9.4.

6.2.2 Caso Pc-Robot

Se ha realizado un análisis general de los cuatro conjuntos de datos correspondientes a los tiempos de ejecución registrados en un sistema que implica la comunicación entre un ordenador (PC) y un robot.

Tiempos de Ejecución Registrados:

1. **Tiempo 1** (61 valores):

   Media: 0.0560 segundos

   Desviación Estándar: 0.0678 segundos

2. **Tiempo** 2 (41 valores):

   Media: 0.0866 segundos

   Desviación Estándar: 0.1412 segundos

3. **Tiempo 3** (61 valores):

   Media: 0.0743 segundos

   Desviación Estándar: 0.1308 segundos

4. **Tiempo 4** (28 valores):

   Media: 0.0951 segundos

   Desviación Estándar: 0.1137 segundos

El Tiempo 1 tiene la media más baja y una desviación estándar más baja, lo que indica que los ciclos en este conjunto son más consistentes y rápidos.

Por otro lado, los Tiempos 2, 3 y 4 presentan medias más altas y desviaciones estándar mayores, lo que evidencia una mayor variabilidad y un desempeño menos uniforme en estos ciclos.

Para mejorar la interpretación de los datos y analizar la distribución de los tiempos de ejecución, se han diseñado histogramas para cada conjunto de tiempos. Los gráficos correspondientes están en el Anexo 9.5.

6.2.3 Caso Pc-Pc

Este análisis presenta los tiempos de ejecución registrados en un sistema que involucra la comunicación y el control de un robot entre dos ordenadores (Pc-Pc).

Tiempos de Ejecución Registrados:

1. **Tiempo 1** (53 valores):

   Media: 0.0884 segundos

   Desviación Estándar: 0.1219 segundos

2. **Tiempo** 2 (36 valores):

   Media: 0.1159 segundos

   Desviación Estándar: 0.1519 segundos

3. **Tiempo 3** (83 valores):

   Media: 0.0705 segundos

   Desviación Estándar: 0.0524 segundos

4. **Tiempo 4** (37 valores):

   Media: 0.1065 segundos

   Desviación Estándar: 0.1361 segundos

El Tiempo 3 muestra la media más baja y la desviación estándar más reducida, lo que indica que los ciclos de ejecución en este caso son más consistentes y rápidos.

Por otro lado, los Tiempos 2 y 4 presentan valores más altos tanto en media como en desviación estándar, lo que refleja una mayor variabilidad y menos predictibilidad en la duración de estos ciclos.

A fin de facilitar la visualización de los datos y examinar la distribución de los tiempos de ejecución, se han generado histogramas para cada conjunto, los cuales pueden consultarse en el Anexo 9.6.

6.2.4 Caso Portátil-Robot

A continuación, se exponen los tiempos de ejecución obtenidos en un sistema que gestiona la comunicación y el control de un robot entre dos ordenadores (Portátil-Robot).

Tiempos de Ejecución Registrados:

1. **Tiempo 1** (46 valores):

   Media: 0.0805 segundos

   Desviación Estándar: 0.1133 segundos

2. **Tiempo** 2 (73 valores):

   Media: 0.0704 segundos

   Desviación Estándar: 0.1268 segundos

3. **Tiempo 3** (116 valores):

   Media: 0.0668 segundos

   Desviación Estándar: 0.0988 segundos

La desviación estándar es relativamente alta en todos los Tiempos, lo que refleja una gran variabilidad en los tiempos de ciclo. Esto sugiere que, aunque la mayoría de los ciclos son rápidos, hay una cantidad significativa de ciclos que toman mucho más tiempo, lo que genera una dispersión y media de tiempo por ciclo baja, alrededor de 0.07 a 0.08 segundos, lo que indica que la mayoría de los ciclos se completan rápidamente.

Con el objetivo de facilitar la visualización de los datos y analizar la distribución de los tiempos de ejecución, se han creado histogramas para cada conjunto, los cuales están disponibles en el Anexo 9.7.

7 Conclusiones

7.1 Problemas identificados

A continuación, se detallan los problemas principales identificados durante el diseño e implementación del sistema:

- **Dificultades en la Ejecución en Diferentes Equipos:** En pruebas con distintos ordenadores (cliente y servidor ejecutándose en equipos separados), surgen problemas lo que afecta la comunicación estable y requiere cambios en los parámetros de red. La ejecución en dos ordenadores distintos ha mostrado problemas de sincronización adicionales al requerir configuraciones específicas para asegurar la conexión adecuada.
- **Detección y Seguimiento del Movimiento del Mouse:** La aplicación debe detectar cuando el usuario realiza un clic izquierdo dentro del área de UIAxes para iniciar el proceso de guiado. Esto significa que se requieren algunos eventos que reconozcan el primer clic y sus coordenadas. Para ello, la propiedad CurrentPoint de los UIAxes en *MATLAB* proporciona las coordenadas actuales del cursor. Sin embargo, es esencial controlar la actualización de las coordenadas de estos puntos para mantener un dibujo consistente de la imagen sin perder su precisión.
- **Retardos en la comunicación:** Se ha implementado un mecanismo de registro del tiempo en cada iteración del bucle de comunicación mediante la función tic-toc de *MATLAB*. Esta función puede ayudar a cronometrar cada ciclo de comunicación y estos datos pueden anotarse en un array con el propósito de evaluar el rendimiento de la comunicación. Se calcula la media de los últimos diez períodos de comunicación para obtener una estimación. Hay que resaltar que el tiempo obtenido corresponde al tiempo total del ciclo de comunicación. Este tiempo está compuesto por varios tipos de retardos:
    - **Retardo por tiempo de cómputo**: Es el tiempo necesario para llevar a cabo las operaciones de cálculo en cada ciclo de comunicación.
    - **Retardo por comunicación**: Indica el tiempo que necesita la información para llegar desde el lugar de comienzo de la comunicación hasta los diferentes elementos del sistema.
    - **Retardo por la gestión de tareas que lleva a cabo el sistema operativo**: El retardo más significativo, ya que involucra los tiempos de espera y la asignación de recursos del sistema operativo para gestionar las tareas y procesos que se ejecutan en los computadores intervinientes.

7.2 Resumen de conclusiones

En conclusión, la implementación de un sistema de guiado para un robot a través de *MATLAB* empleando sockets de comunicación plantea una serie de desafíos técnicos en la sincronización y administración de la conexión en tiempo real. La sincronización precisa es fundamental para que el sistema funcione correctamente en los dos modos de guiado propuestos, punto a punto y continua, y para que el robot responda a las instrucciones generadas en *MATLAB* de manera adecuada.

El uso de sockets permitió un modelo cliente-servidor con *RobotStudio* actuando como servidor y *MATLAB* como cliente. Las funciones RAPID ejecutadas en el robot, tales como *SocketCreate*,

SocketBind, *SocketListen*, y *SocketAccept*, permiten al servidor estar en constante escucha, preparado para aceptar conexiones de clientes. *MATLAB*, como cliente, solicita conexión al servidor y envía datos en tiempo real al robot, asegurando así una comunicación bidireccional sincrónica.

Al aplicar estos mecanismos de comunicación y sincronización, el sistema logra tiempos de respuesta y ejecución adecuados para aplicaciones en tiempo real. Esto facilita el trazado de trayectorias en modalidades continua o punto a punto, tanto en modo online (tiempo real) como offline (modo Batch). La capacidad de controlar y analizar los tiempos de respuesta y de movimiento, a través de tic-toc en *MATLAB*.

Es importante señalar que, en ninguno de los casos analizados, se ha alcanzado un verdadero "tiempo real", debido a los retrasos en la comunicación. El sistema PC-robot es el que más se aproxima a una ejecución ideal, pero aún presenta ciertos retardos. Estos retrasos pueden ser atribuidos a dos factores principales:

- **El proceso de comunicación**: Contribuye al tiempo de espera necesario para que los datos sean transmitidos y procesados.
- **El sistema operativo de los computadores de propósito general:** A diferencia de los controladores dedicados como el IRC5, los sistemas operativos convencionales no están optimizados para operar en tiempo real, lo que provoca retrasos adicionales.

Además, se ha observado que los tiempos de ejecución varían dependiendo de la longitud de los segmentos de las trayectorias. Al aumentar la velocidad de ejecución, los segmentos se alargan y se reducen los puntos intermedios, lo que puede dar lugar a tiempos de ejecución más largos por cada segmento. Aunque la velocidad de movimiento es mayor, la mayor longitud de los segmentos implica una mayor espera para completar la trayectoria en cada instante, lo que puede provocar un aumento en los tiempos de espera entre cada ciclo. Es importante diferenciar entre las distintas series de datos y no sorprenderse por estas variaciones, ya que los tiempos más largos son el resultado natural de trayectorias más extensas con menos puntos intermedios.

Cabe destacar que, para obtener un seguimiento preciso de trayectorias continuas, es recomendable utilizar el modo offline. Esto permite un registro más preciso de las trayectorias antes de su ejecución, lo que mejora la exactitud en el proceso.

8 Bibliografía

[1] ABB. (año). Technical reference manual: RAPID Instructions, Functions and Data Types. ABB Group.

[2] ABB. (año). Application Manual: Controller Software IRC5. ABB Group.

[3] MathWorks. (2023). MATLAB Documentation.

[4] MathWorks. (2023). MATLAB and Simulink for Engineering Applications.

[5] MathWorks. (n.d.). App Designer - Create Professional Apps in MATLAB.

[6] ABB Robotics. (2023). RobotStudio: Offline Programming & Simulation

[7] ABB Robotics. (2023). *IRC5*

[8] Hagen, S. (2006). TCP/IP Network Administration (3ª ed.). O'Reilly Media.

9 Anexos

9.1 Código RAPID

```
MODULE Module1
    PERS wobjdata Pizarra_virtual:=[FALSE,TRUE,"",[[304.079397779,101.484779365,324.620193825],
[0.981060262,-0.015134436,0.085831651,0.172987394]],[[0,0,0],[1,0,0,0]]];
    !PERS wobjdata Pizarra_real:=[FALSE,TRUE,"",[[304.079397779,101.484779365,324.620193825],
[0.981060262,-0.015134436,0.085831651,0.172987394]],[[0,0,0],[1,0,0,0]]]; ! A establecer por método
3 puntos
    PERS tooldata Punta:=[TRUE,[[80,0,23.5],[0.707106781,0,-0.707106781,0]],[0.68,[5,0,25],
[1,0,0,0],0,0,0]];
    !PERS tooldata Punta:=[TRUE,[[115.5,0,23.5],[0.707106781,0,-0.707106781,0]],[0.68,[5,0,25],
[1,0,0,0],0,0,0]]; ! Puntero real en el L0.06
    PERS tooldata Garra:=[TRUE,[[0,0,76],[0,0.707106781,0.707106781,0]],[0.68,[5,0,25],
[1,0,0,0],0,0,0]];
    CONST confdata config_fun:=[0,0,0,0];
    CONST confdata config_fuf:=[0,0,0,1];
    CONST extjoint ejes_externos:=[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09];
    CONST jointtarget casa:=[[0,0,0,0,0,0],ejes_externos];
    CONST speeddata vbaja:= v100;
    CONST speeddata vmedia:= v200;
    CONST speeddata valta:= v500;
    VAR speeddata velocidad:= vmedia;
    VAR robtarget loc_inicial;
    VAR stoppointdata my_inpos := [ inpos, TRUE, [ 0, 0, 0.1, 2], 0, 0, "", 0, 0];

    !Variables for socket communication
    VAR socketdev servidor;
    VAR socketdev cliente;
    VAR string IP_servidor:="192.168.1.130"; ! Equipo casa
    !VAR string IP_servidor:="10.3.17.176"; ! Equipo 176 L0.06
    !VAR string IP_servidor:="10.3.17.206"; ! robot ABB viejo
    !VAR string IP_servidor:="10.3.17.204"; ! robot ABB nuevo
    VAR num puerto:=4000;
    VAR string mensaje_recibido;
    VAR string modo_continuo;
    VAR num pos_coma;
    VAR num pos_segunda_coma;
    VAR num longitud_mensaje;
    VAR bool basurilla;
    VAR num pos_X;
    VAR num pos_Y;
    VAR num pos_X_anterior:=0;
    VAR num pos_Y_anterior:=0;
    VAR num contador_mensajes:=0;
    VAR num duracion;
    VAR num distancia;
    VAR num vel;

    PROC main()
        AccSet 30, 50;
        loc_inicial:=[[150,75,20],[1,0,0,0],config_fuf,ejes_externos];
        MoveAbsJ casa, valta, fine, Punta;
        MoveJ loc inicial. valta. fine. Punta\WObj:=Pizarra virtual:
        ! Sockets Communication section
        SocketCreate servidor;
        SocketBind servidor, IP_servidor, puerto;
        SocketListen servidor;
        SocketAccept servidor, cliente, \Time:=60;
        SocketSend cliente,\str:="COMUNICACION ACEPTADA";
        TPWrite "COMUNICACION ACEPTADA";
        SocketReceive cliente \Str:=mensaje_recibido;
```

```
        WHILE mensaje_recibido <> "FIN COMUNICACION" DO
            ClkStop clock1; ! Parar reloj
            duracion:= Clkread(clock1); ! leer reloj

            IF duracion<0.01 THEN
               duracion:=0.01;
            ENDIF
            contador_mensajes:=contador_mensajes+1;
            TPWrite "MENSAJE Nº" + NumToStr(contador_mensajes, 0) + " RECIBIDO: " +
mensaje_recibido;
            pos_coma:=StrFind(mensaje_recibido, 1, ","); ! Encuentra la posición de la primera coma
            pos_segunda_coma := StrFind(mensaje_recibido, pos_coma + 1, ","); ! Encuentra la
posición de la segunda coma
            longitud_mensaje:=StrLen(mensaje_recibido); ! Longitud total del mensaje
            basurilla:=StrToVal(StrPart(mensaje_recibido, 1, pos_coma-1), pos_X); ! Extraer x
            !basurilla:=StrToVal(StrPart(mensaje_recibido, pos_coma+1, longitud_mensaje-pos_coma),
pos_Y);
            basurilla := StrToVal(StrPart(mensaje_recibido, pos_coma + 1, pos_segunda_coma -
pos_coma - 1), pos_Y); ! Extraer y
            modo_continuo := StrPart(mensaje_recibido, pos_segunda_coma + 1, longitud_mensaje -
pos_segunda_coma);
           ! Verificar el indicador de tipo de trayectoria y realizar el movimiento
            IF modo_continuo="NO"  THEN
                MoveL RelTool(loc_inicial, pos_X, pos_Y, 0), vmedia, fine \inpos:=my_inpos, Punta
\WObj:=Pizarra_virtual;
            ELSEIF modo_continuo="SI" THEN
                MoveL RelTool(loc_inicial, pos_X, pos_Y, 0), vmedia, z20 , Punta
\WObj:=Pizarra_virtual;
                distancia:=Sqrt((pos_X-pos_X_anterior)*(pos_X-pos_X_anterior)+(pos_Y-pos_Y_anterior)
*(pos_Y-pos_Y_anterior));
                vel:=distancia/duracion;
                IF vel>5000 THEN
                   vel:=5000;
                ELSEIF vel<10 THEN
                   vel:=10;
                ENDIF
                velocidad.v_tcp:=vel;
                MoveL RelTool(loc_inicial, pos_X, pos_Y, 0), velocidad, z20 , Punta
\WObj:=Pizarra_virtual;
            ENDIF
            ClkReset clock1;
            ClkStart clock1; ! Comenzar a contar tiempo para cálculo de la velocidad en modo
continuo
            SocketSend cliente,\str:="OK";
            pos_X_anterior:=pos_X;
            pos_Y_anterior:=pos_Y;
            !WaitTime 0.1;
            SocketReceive cliente \Str:=mensaje_recibido;
        ENDWHILE
        TPWrite "FIN COMUNICACION";
        MoveAbsJ casa, valta, fine, Punta;
        SocketClose cliente;
        WaitTime 1;
        SocketClose servidor;
    ENDPROC

ENDMODULE
```

9.2 Código Matlab

```
classdef Remote_Guidance_Client < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public) ...


    properties (Access = private)
        cliente,mensaje_1,LastX=0,LastY=0,lista_coordenadas; % Description
        exit=false;
        %estructura para almacenar las coordenadas
        coordenadas=[];
        %variable global
        esContinua=false;
        esTiempoReal=false;
        %IP_servidor:="192.168.1.130"; ! Equipo casa
        %IP_servidor:="10.3.17.176"; ! Equipo 176 L0.06
        %IP_servidor:="10.3.17.206"; ! robot ABB viejo
        %IP_servidor:="10.3.17.204"; ! robot ABB nuevo

    end


    % Callbacks that handle component events
    methods (Access = private)

        % Value changed function: Conectar
        function ConectarValueChanged(app, event)
            %Variables
            Puerto = app.Puerto.Value;%string
            Direccion = app.Direccionpuerto.Value;%string
            %Creaccion del cliente
            app.cliente = tcpclient(Direccion,str2double(Puerto),"Timeout",50);
            pause(1);
            app.Lamp.Color=[0.59,0.94,0.43];%esta modo on
            waitfor(app.cliente,'NumBytesAvailable',21);
            mensaje = char(read(app.cliente))
            app.mensaje_1 = mensaje;
        end

        % Button pushed function: Salir
        function SalirButtonPushed(app, event)
            app.exit=false;
             if app.exit==false
                 write(app.cliente,"FIN COMUNICACION")
                 app.Lamp.Color=[0.93,0.20,0.20];%esta modo off
                 disp('fin de comunicacion con cliente')
                 pause(1);
                 clear app.cliente;
             end
        end

        % Window button motion function: UIFigure
        function UIFigureWindowButtonMotion(app, event)

        end

        % Button down function: UIAxes
        function UIAxesButtonDown(app, event)
            if contains(app.mensaje_1,"COMUNICACION ACEPTADA")

                %inicializar coordenadas (0,0)
                x=0;
                y=0;

                plot(app.UIAxes,x,y,'o','MarkerFaceColor','b');
                hold(app.UIAxes,'on')
                %definir margenes
                x_min = -150;  % Límite inferior en X
                x_max = 150;   % Límite superior en X
                y_min = -75;  % Límite inferior en Y
                y_max = 75;   % Límite superior en Y

                if app.esContinua==true && app.esTiempoReal==true
                    disp('Continua y tiempo real ');

                    tiempos=[];% Almacena los tiempos de cada iteración
                    contador =0;%Contador de iteraciones
                    point = app.UIAxes.CurrentPoint;
                    x_c = point(1,1);
                    y_c= point(1,2);
                    plot(app.UIAxes,x_c,y_c,'o','MarkerFaceColor','b');
                    text(app.UIAxes, x_c,y_c, sprintf('(%0.1f, %0.1f)',x_c,y_c), 'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right');

                    inicio_continua = false;
                    while strcmp(get(gcbf,'SelectionType'),'normal')
                        point = app.UIAxes.CurrentPoint;
                        x = point(1,1);
                        y = point(1,2);
```

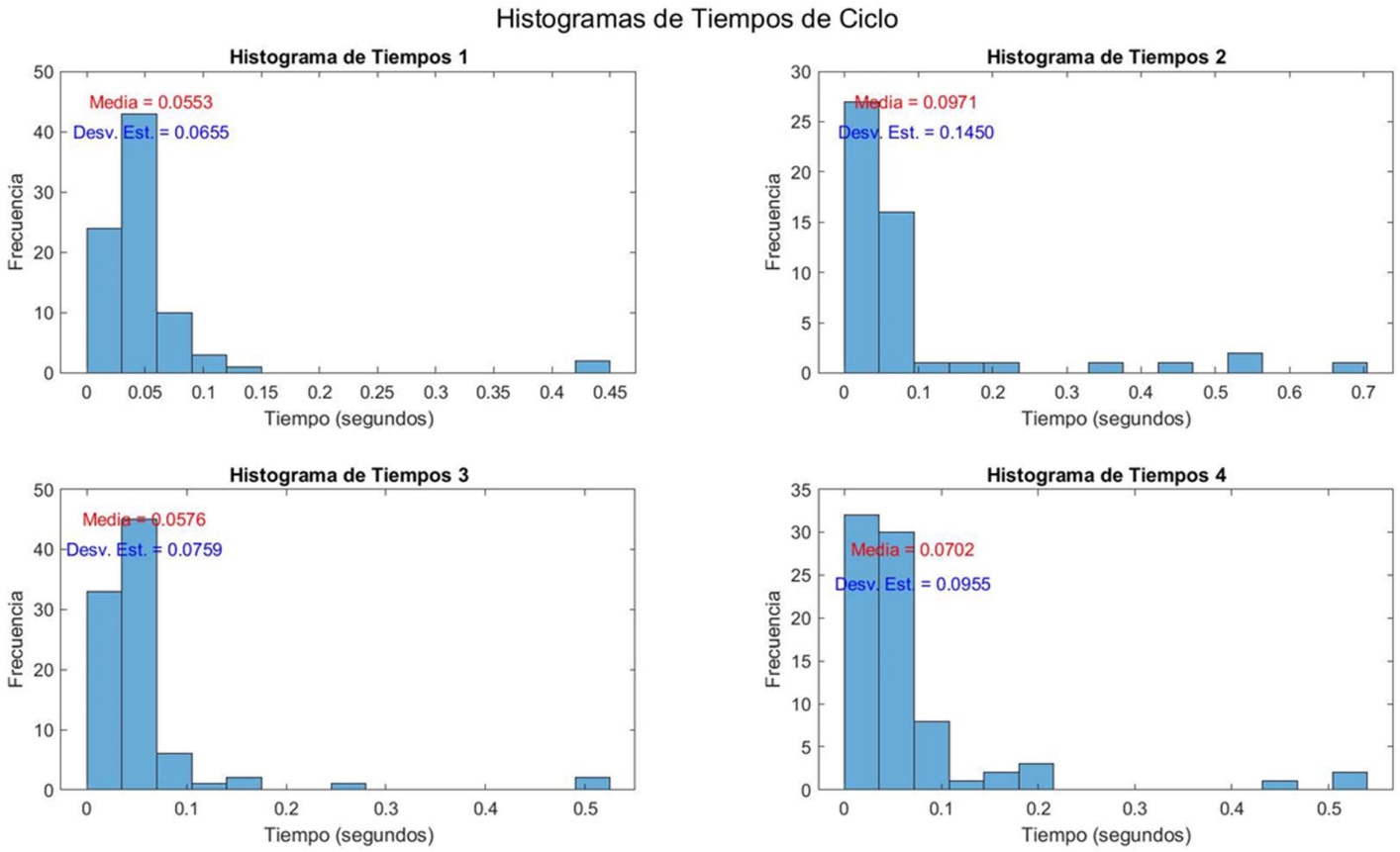
```
        % Verificar si las coordenadas están fuera de los márgenes
        if (x < x_min || x > x_max || y < y_min || y > y_max)
            % Mostrar un mensaje emergente
            app.Advertencia.Value = 'Punto fuera de los márgenes permitidos';
            pause(0.1);  % Pausa breve para permitir que el mensaje se procese
            % Hacer parpadear la imagen de advertencia
            app.Image_advertencia.Visible = 'on';   % Mostrar la imagen
            pause(0.25);                            % Pausa breve de 0.25 segundos
            app.Image_advertencia.Visible = 'off';  % Ocultar la imagen
            pause(0.25);                            % Pausa breve de 0.25 segundos
        else
            app.Advertencia.Value = '';  % Limpiar el mensaje de advertencia si está dentro de los márgenes
            app.Image_advertencia.Visible = 'on';   % Mostrar la imagen
        end

        if  inicio_continua

            plot(app.UIAxes, [app.LastX, x], [app.LastY, y], '-', 'LineWidth', 1,'Color', [0.70,0.92,0.87]);

        end

        if abs(x - x_c) < 0.1 && abs(y - y_c) < 0.1
            plot(app.UIAxes, [app.LastX, x], [app.LastY, y], ':', 'LineWidth', 1, 'Color', [0.7, 0.7, 0.7]);
            inicio_continua=true;
        end


        %Actualiza las coordenadas "x" e "y" del ultimo punto registrado con la coordenada del punto actual
        app.LastX=x;
        app.LastY=y;
        %Convierte los valores 'x' e 'y' en  una cadena de texto
        x_pos=num2str(x,'%.1f')
        y_pos=num2str(y,'%.1f')

        line = [x_pos,',',y_pos,',','SI'];%Añadimos si para estar modo si CONTINUO
        tic;%Inicia el temporizador
        write(app.cliente,line);%Envio de datos al servidor
        mensaje = read(app.cliente,2,"char")
        tiempos=[tiempos;toc];% Guarda el tiempo transcurrido en la iteración
        disp('Tiempos:');
        disp(tiempos);
        % Incrementa el contador de iteraciones
        contador = contador + 1;
        % Cada 10 iteraciones, calcula la media de los últimos 10 tiempos
        if mod(contador, 10) == 0
            % Si el número de tiempos es mayor o igual a 10, calculamos la media
            ultimos_10_tiempos = tiempos(end-9:end);  % Últimos 10 tiempos
            media_ultimos_10 = mean(ultimos_10_tiempos);  % Media de los últimos 10
            app.Periodo.Value= num2str(media_ultimos_10);
        end

    end

    % Cálculo de la media y la desviación estándar de los tiempos
    media_tiempo = mean(tiempos);
    desviacion_tiempo = std(tiempos);
    % Mostrar los resultados
    disp(['Media de tiempo por ciclo: ', num2str(media_tiempo)]);
    disp(['Desviación estándar de los tiempos: ', num2str(desviacion_tiempo)]);

elseif app.esContinua==false && app.esTiempoReal==true
    disp('punto a punto y tiempo real ');

    if  strcmp(app.UIFigure.SelectionType, 'normal')

        estaDibujandoPuntoaPunto=true;


        % Obtenemos las coordenada "x" e "y" del punto donde se hizo clic
        % en los ejes de la interfaz de usuario (UIAxes)
        point=app.UIAxes.CurrentPoint;
        %Asignamos a las variables "x" e "y"
        x=point(1,1)
        y=point(1,2)

        % Verificar si las coordenadas están fuera de los márgenes
        if (x < x_min || x > x_max || y < y_min || y > y_max)
            % Mostrar un mensaje emergente
            app.Advertencia.Value = 'Punto fuera de los márgenes permitidos';
            pause(0.1);  % Pausa breve para permitir que el mensaje se procese
            % Hacer parpadear la imagen de advertencia
            app.Image_advertencia.Visible = 'on';   % Mostrar la imagen
            pause(0.25);                            % Pausa breve de 0.25 segundos
            app.Image_advertencia.Visible = 'off';  % Ocultar la imagen
            pause(0.25);                            % Pausa breve de 0.25 segundos
        else
            app.Advertencia.Value = '';  % Limpiar el mensaje de advertencia si está dentro de los márgenes
            app.Image_advertencia.Visible = 'on';   % Mostrar la imagen
        end

        % Mantiene el grafico actual y agrega un punto  y un texto en
        % las coordenadas clicadas
        hold(app.UIAxes,'on')
        plot(app.UIAxes,x,y,'o','MarkerFaceColor','b');
```

```
        plot(app.UIAxes, [app.LastX,x], [app.LastY,y], '-', 'LineWidth', 1,'Color', [0.94,0.78,0.56]);

        app.LastX=x;
        app.LastY=y;
        x_pos=num2str(x,'%.1f')
        y_pos=num2str(y,'%.1f')

        line = [x_pos,',',y_pos,',','NO'];%Añadimos NO para estar modo continuo
        write(app.cliente,line);
        mensaje = read(app.cliente,2,"char")

        text(app.UIAxes, x,y, sprintf('(%0.1f, %0.1f)',x,y), 'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right');
        app.lista_coordenadas=0;

    else strcmp(app.UIFigure.SelectionType, 'alt')
        estaDibujandoPuntoaPunto=false;
        x=app.LastX;
        y=app.LastY;

        % Restaurar el último valor de las coordenadas
        app.LastX = x;
        app.LastY = y;
    end


    % Si hemos marcado la opcion de checkbox de Seguimiento
elseif app.esContinua==true && app.esTiempoReal==false
    disp('Continua y Batch');

    i=1;
    suma=0;

    point = app.UIAxes.CurrentPoint;
    x1 = point(1,1);
    y1 = point(1,2);
    plot(app.UIAxes,x1,y1,'o','MarkerFaceColor','b');
    text(app.UIAxes, x1,y1, sprintf('(%0.1f, %0.1f)',x1,y1), 'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right');

    inicio_dibujo = false;


    while strcmp(get(gcbf,'SelectionType'),'normal') %Verifica si el tipo de seleccion del raton  es un clic izq. ,si son iguale
        %Ponemos un tiempo de espera
        pause (0.1)
        point = app.UIAxes.CurrentPoint;
        x = point(1,1);
        y = point(1,2);

        % Verificar si las coordenadas están fuera de los márgenes
        if (x < x_min || x > x_max || y < y_min || y > y_max)
            % Mostrar un mensaje emergente
            app.Advertencia.Value = 'Punto fuera de los márgenes permitidos';
            pause(0.1);  % Pausa breve para permitir que el mensaje se procese
            % Hacer parpadear la imagen de advertencia
            app.Image_advertencia.Visible = 'on';   % Mostrar la imagen
            pause(0.25);                      % Pausa breve de 0.25 segundos
            app.Image_advertencia.Visible = 'off';  % Ocultar la imagen
            pause(0.25);                      % Pausa breve de 0.25 segundos
        else
            app.Advertencia.Value = '';  % Limpiar el mensaje de advertencia si está dentro de los márgenes
            app.Image_advertencia.Visible = 'on';   % Mostrar la imagen
        end

        if  inicio_dibujo
            plot(app.UIAxes, [app.LastX,x], [app.LastY,y], '-', 'LineWidth', 1,'Color', [0.95,0.77,0.73]);
        end

        if abs(x - x1) < 0.1 && abs(y - y1) < 0.1
            plot(app.UIAxes, [app.LastX, x], [app.LastY, y], ':', 'LineWidth', 1, 'Color', [0.7, 0.7, 0.7]);
            inicio_dibujo = true;
        end


        app.LastX=x;
        app.LastY=y;

        %Almacenamos las coordenadas en la lista
        app.coordenadas=[app.coordenadas;x,y];

        disp( app.coordenadas);



    end


    %Recorrer la lista de coordenadas
    for i=1:size(app.coordenadas,1)
        x= app.coordenadas(i,1);
        y=app.coordenadas(i,2);
```

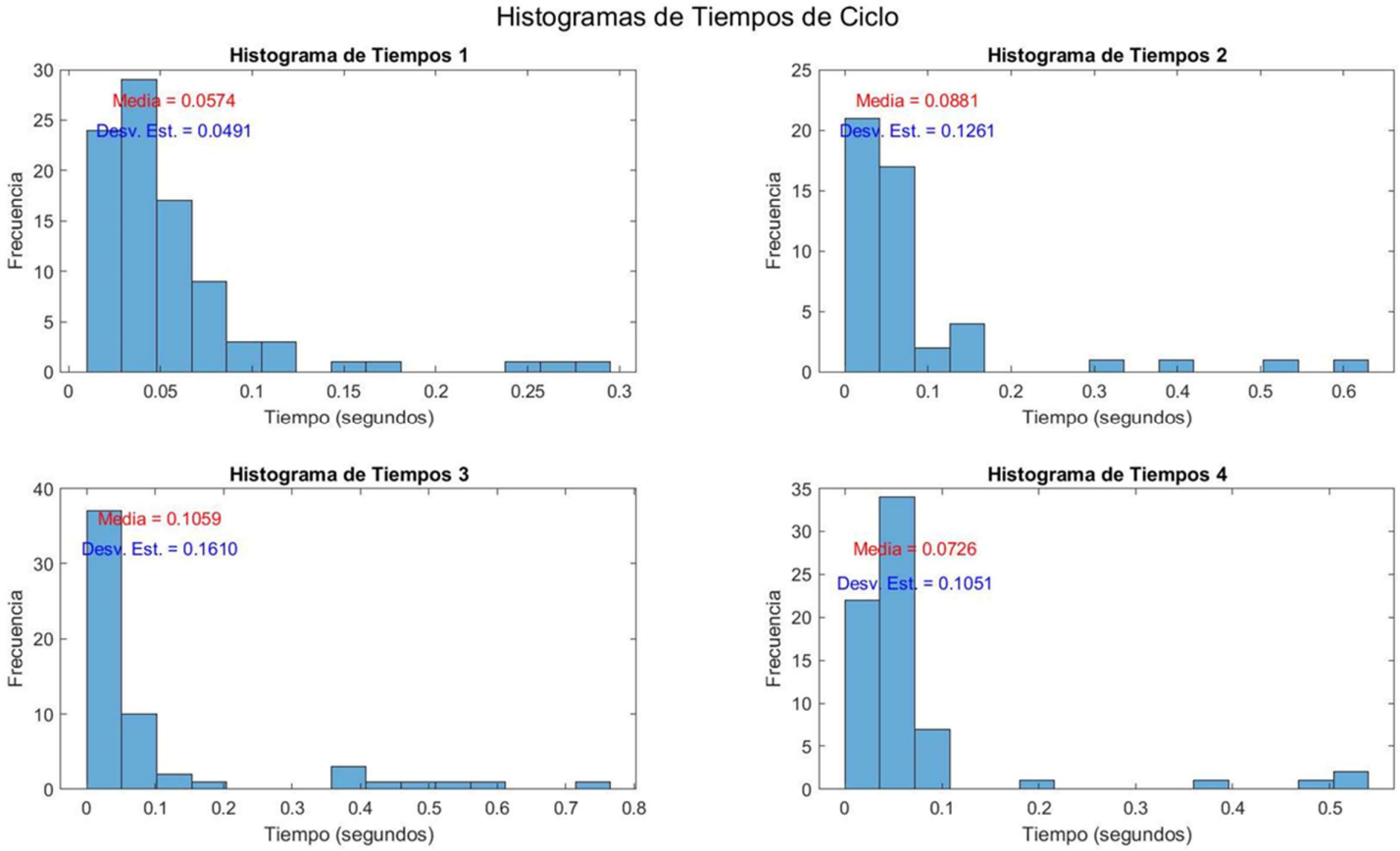
```
                x_pos=num2str(x,'%.1f')
                y_pos=num2str(y,'%.1f')
                line = [x_pos,',',y_pos,',','SI'];%Añadimos SI para estar modo Continuo
                write(app.cliente,line);
                mensaje = read(app.cliente,2,"char")
            end
            app.LastX=x;
            app.LastY=y;
            app.coordenadas=[app.LastX,app.LastY];
        elseif app.esContinua==false && app.esTiempoReal==false
            disp('Punto a punto y Bach ');

            if  strcmp(app.UIFigure.SelectionType, 'normal')

                % Obtenemos las coordenada "x" e "y" del punto donde se hizo clic
                % en los ejes de la interfaz de usuario (UIAxes)
                point=app.UIAxes.CurrentPoint;
                %Asignamos a las variables "x" e "y"
                x=point(1,1)
                y=point(1,2)


                % Verificar si las coordenadas están fuera de los márgenes
                if (x < x_min || x > x_max || y < y_min || y > y_max)
                    % Mostrar un mensaje emergente
                    app.Advertencia.Value = 'Punto fuera de los márgenes permitidos';
                    pause(0.1);  % Pausa breve para permitir que el mensaje se procese
                    % Hacer parpadear la imagen de advertencia
                    app.Image_advertencia.Visible = 'on';   % Mostrar la imagen
                    pause(0.25);                        % Pausa breve de 0.25 segundos
                    app.Image_advertencia.Visible = 'off';  % Ocultar la imagen
                    pause(0.25);                        % Pausa breve de 0.25 segundos
                else
                    app.Advertencia.Value = '';  % Limpiar el mensaje de advertencia si está dentro de los márgenes
                    app.Image_advertencia.Visible = 'on';   % Mostrar la imagen
                end

                % Mantiene el grafico actual y agrega un punto  y un texto en
                % las coordenadas clicadas
                hold(app.UIAxes,'on')
                plot(app.UIAxes,x,y,'o','MarkerFaceColor','b');

                plot(app.UIAxes, [app.LastX,x], [app.LastY,y], '-', 'LineWidth', 1,'Color', [0.96,0.90,0.98]);

                app.LastX=x;
                app.LastY=y;
                %Almacenamos las coordenadas en la lista
                app.coordenadas=[app.coordenadas;x,y];

                disp(app.coordenadas);


            else strcmp(app.UIFigure.SelectionType, 'alt')

                for i=1:size(app.coordenadas,1)
                    x=app.coordenadas(i,1);
                    y=app.coordenadas(i,2);

                    x_pos=num2str(x,'%.1f')
                    y_pos=num2str(y,'%.1f')
                    line = [x_pos,',',y_pos,',','NO'];%Añadimos NO para estar modo NO CONTINUO
                    write(app.cliente,line);
                    mensaje = read(app.cliente,2,"char")
                    % Opcional: Mostrar la respuesta del robot para depuración
                    disp(['Respuesta del robot para punto (', x_pos, ',', y_pos, '): ', mensaje]);

                end
                app.LastX=x;
                app.LastY=y;
                app.coordenadas=[app.LastX,app.LastY];
            end


        end

        %Agregar texto con las coordenadas del punto
        text(app.UIAxes, app.LastX,app.LastY, sprintf('(%0.1f, %0.1f)',app.LastX,app.LastY), 'VerticalAlignment', 'bottom', 'HorizontalA
        plot(app.UIAxes,app.LastX,app.LastY,'o','MarkerFaceColor','b');


    end
end
```

```
        % Button pushed function: help
        function helpButtonPushed(app, event)
            helpMessage = sprintf(['Instrucciones:\n' ...
                '0. Conectar: Interfaz de comunicación entre la aplicación cliente en Matlab y el servidor en Robot Studio.\n' ...
                '1. Seguimiento:\n' ...
                '- Tiempo real: Al hacer clic con el botón izquierdo, comenzarás a dibujar mientras te desplazas por los ejes ' ...
                'de la interfaz de usuario (UIAxes). Para finalizar el dibujo, haz clic con el botón derecho.\n' ...
                '- Batch: Para dibujar, haz clic izquierdo y mueve el ratón sobre los ejes de la interfaz de usuario (UIAxes). ' ...
                'Cuando quieras finalizar el dibujo, haz clic derecho.\n' ...
                '2. Tipo de trayectoria:\n' ...
                '- Continua:Puedes hacer un trazo fluido.Para finalizar, haz clic derecho...\n' ...
                '- Punto a punto:Puedes hacer clic izquierdo tantas veces como desees. Para finalizar, haz clic derecho.\n' ...
                '3. Desconectar: Fin de comunicación entre Cliente y Servidor.\n']);

            uialert(app.UIFigure, helpMessage, 'Help');
        end

        % Value changed function: CheckBox_Seguimiento
        function CheckBox_SeguimientoValueChanged(app, event)
            if app.CheckBox_Seguimiento.Value == 1

                disp('Tiempo real');

                app.esTiempoReal = true;

                app.TiempoRealLabel.Enable= "on";
                app.TiempoRealLabel.FontColor=[0.00,0.00,0.00];
                app.TiempoRealLabel.FontWeight='bold'; % Cambiar a negrita si la casilla de verificación está marcada
                app.BatchLabel.Enable="off";%desactivamos offline(Batch)
                app.BatchLabel.FontWeight='normal';


            elseif app.CheckBox_Seguimiento.Value == 0

                disp('Batch');

                app.esTiempoReal = false;

                app.BatchLabel.Enable="on";
                app.BatchLabel.FontColor=[0.00,0.00,0.00];
                app.BatchLabel.FontWeight='bold'; % Si la casilla de verificación está marcada
                app.TiempoRealLabel.Enable= "off";%desactivamos online(Tiempo real)
                app.TiempoRealLabel.FontWeight='normal';

            end

        end

        % Value changed function: CheckBox_Trayectoria
        function CheckBox_TrayectoriaValueChanged(app, event)
            if app.CheckBox_Trayectoria.Value == 1
                disp('Continua');

                app.esContinua = true;

                app.ContinuaLabel.Enable= "on";
                app.ContinuaLabel.FontColor=[0.00,0.00,0.00];
                app.ContinuaLabel.FontWeight='bold'; % Cambiar a negrita si la casilla de verificación está marcada
                app.PuntoaPuntoLabel.Enable= "off";
                app.PuntoaPuntoLabel.FontWeight='normal';

            elseif app.CheckBox_Trayectoria.Value == 0
                disp('Punto a Punto');

                app.esContinua = false;

                app.PuntoaPuntoLabel.Enable= "on";
                app.PuntoaPuntoLabel.FontColor=[0.00,0.00,0.00];
                app.PuntoaPuntoLabel.FontWeight='bold'; % Cambiar a negrita si la casilla de verificación está marcada
                app.ContinuaLabel.Enable= "off";
                app.ContinuaLabel.FontWeight='normal';

            end
        end
    end
```

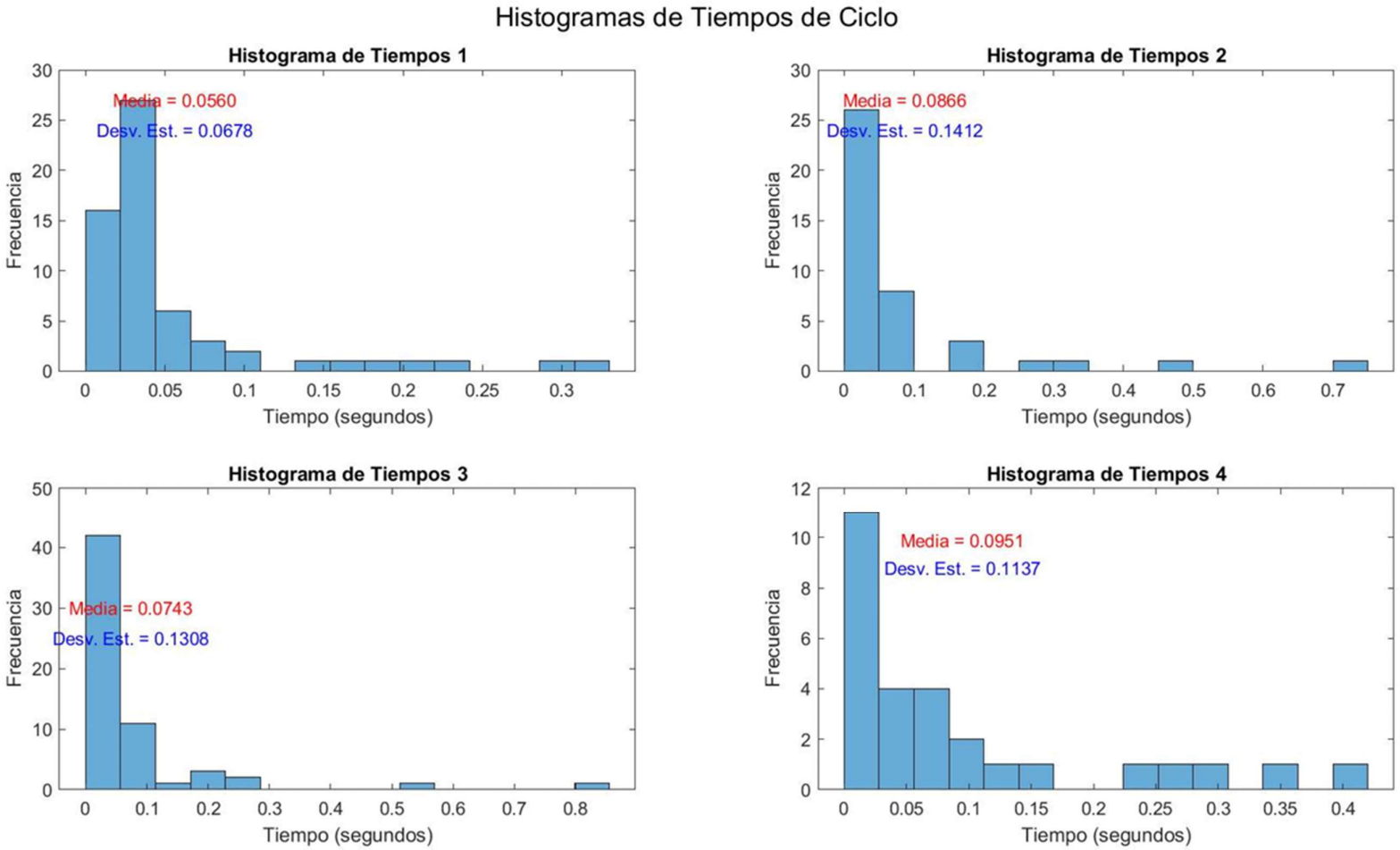
9.3 Histograma en un ordenador único (Portátil- Portátil)



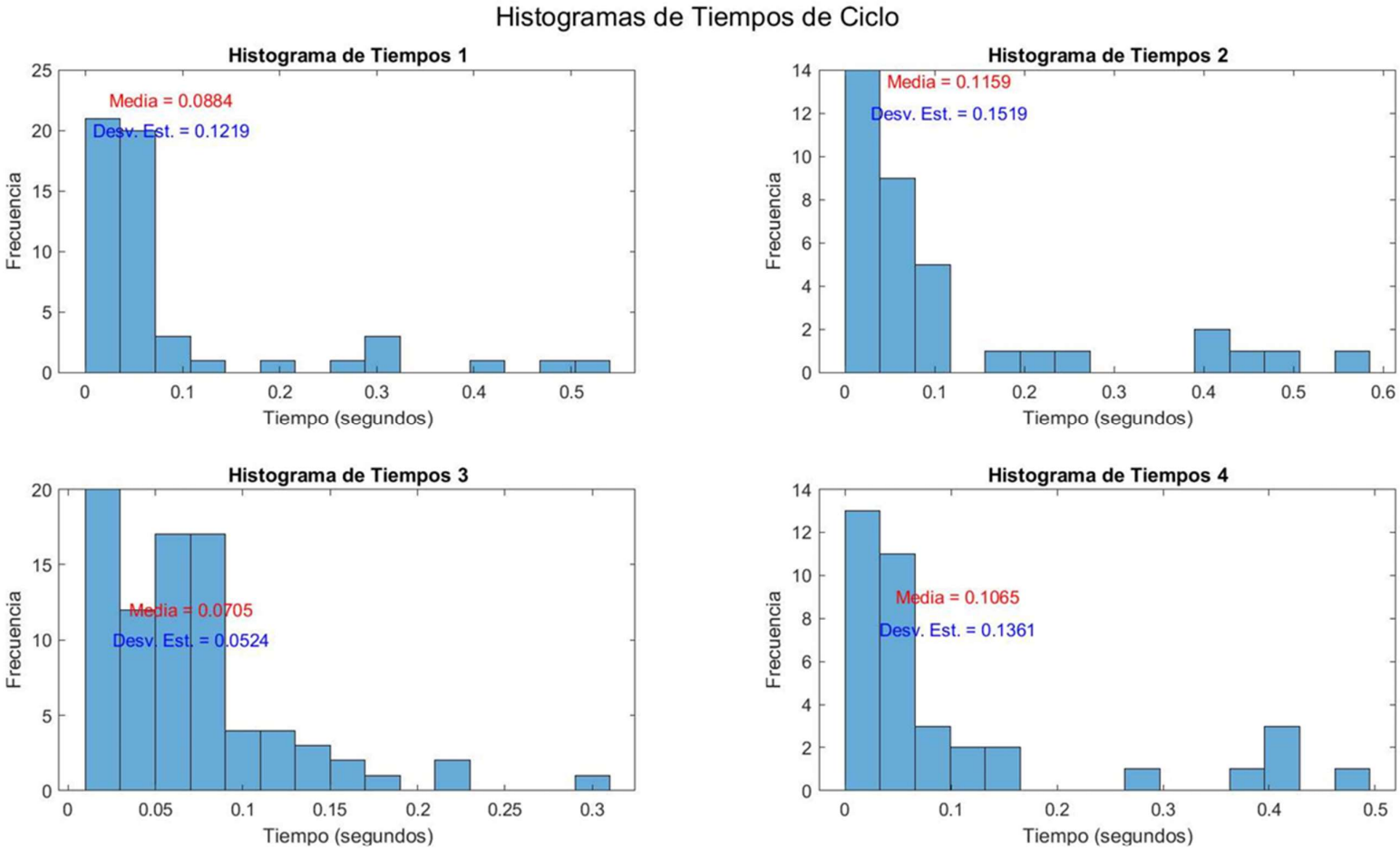
9.4 Histograma en ordenadores distintos (Pc- Portátil)



9.5 Histograma en ordenadores distintos (Pc-Robot)



9.6 Histograma en ordenadores distintos (Pc-Pc)



9.7 Histograma en ordenadores distintos (Portátil-Robot)