VOLUMEN 48
XXXI Verano De la Ciencia
ISSN 2395-9797
www.jovenesenlaciencia.ugto.mx

Control Remoto para un Robot Planar Serial de 2 GDL

Jeshua Lestat Barrón Martínez, Juan Armando Gutierréz Robledo, Adrián Sebastián Rocha Ramírez, Naomi Mayte Vázquez García
División de Ingenierías Campus Irapuato-Salamanca, Universidad de Guanajuato
jl.barronmartinez@ugto.mx, ja.gutierrezrobledo@ugto.mx, as.rocharamirez@ugto.mx, nm.vazquezgarcia@ugto.mx

Resumen

Este artículo presenta el desarrollo de un sistema de control remoto para un robot planar serial de dos grados de libertad (2 GDL), diseñado para realizar movimientos precisos del efector final mediante una interfaz de control inalámbrica. La metodología empleada integra el modelado cinemático del manipulador, la implementación de algoritmos de cinemática directa e inversa y la programación del sistema de control para el accionamiento coordinado de los motores. El prototipo desarrollado permite ejecutar desplazamientos controlados en el plano cartesiano, mejorando la precisión y la respuesta del sistema ante diferentes comandos de operación. Los resultados obtenidos demuestran la viabilidad del control remoto en manipuladores de baja complejidad, constituyendo una plataforma didáctica y experimental para aplicaciones de automatización, robótica educativa e investigación en sistemas de control.

Palabras clave: robot planar, DART, control remoto, servidor MQTT, Wifi

Introducción

El desarrollo de sistemas robóticos con capacidades de control remoto ha cobrado una relevancia significativa en los ámbitos de la automatización industrial, la investigación y la educación, debido a que permiten supervisar y operar manipuladores de manera segura, eficiente y precisa sin requerir la presencia física del operador. En este contexto, los robots planares seriales de dos grados de libertad (2 GDL) constituyen una plataforma ampliamente utilizada para el estudio de la cinemática, el control de movimiento y la planificación de trayectorias, ya que su arquitectura mecánica simplificada facilita el análisis de modelos matemáticos y la implementación de estrategias de control.

El presente trabajo desarrolla un sistema de control remoto para un robot planar serial de 2 GDL, cuyo objetivo es permitir el posicionamiento de su efector final mediante comandos enviados desde una aplicación móvil, empleando un módulo ESP32 como puente de comunicación entre la interfaz de usuario y el controlador físico del robot basado en Arduino. La arquitectura propuesta integra tecnologías de comunicación inalámbrica y protocolos de mensajería para proporcionar un sistema flexible, escalable y de fácil implementación.

El sistema se implementó utilizando una plataforma ESP32 para la gestión de las comunicaciones, un microcontrolador Arduino encargado del control del manipulador y una aplicación móvil desarrollada en Flutter para la interacción con el usuario. La comunicación remota se realiza mediante el protocolo MQTT (Message Queuing Telemetry Transport), el cual permite el intercambio eficiente de mensajes entre los diferentes componentes del sistema. Entre las principales funciones implementadas destacan la conexión a una red Wi-Fi, el establecimiento de comunicación con un servidor MQTT, la recepción y procesamiento de comandos, la comunicación serial mediante UART entre la ESP32 y el Arduino, la publicación del estado de ejecución del robot y el control remoto a través de la aplicación móvil.

La arquitectura de comunicación se fundamenta en un esquema de intercambio bidireccional de información. La aplicación móvil publica comandos en el servidor MQTT, los cuales son recibidos por la ESP32 para su posterior procesamiento. Una vez interpretados, dichos comandos son transmitidos al microcontrolador Arduino mediante comunicación UART, donde se ejecutan las rutinas correspondientes para controlar los actuadores del robot y generar el movimiento solicitado. Paralelamente, tanto la ESP32 como el Arduino

generan mensajes de respuesta que son enviados nuevamente al servidor MQTT, permitiendo que la aplicación móvil supervise en tiempo real el estado de ejecución de las instrucciones.

La integración de tecnologías como ESP32, Arduino, MQTT y Flutter demuestra la viabilidad de implementar sistemas de control remoto de bajo costo con capacidades de monitoreo y operación en tiempo real. Además, la arquitectura desarrollada constituye una base sólida para futuras aplicaciones orientadas al control de manipuladores robóticos de mayor complejidad, la automatización de procesos y el desarrollo de plataformas educativas para el estudio de la robótica y las comunicaciones industriales.

Materiales y métodos

Hardware

El mecanismo robótico utilizado en este trabajo se presenta en la **Figura 1**. El sistema físico está conformado por diversos componentes electrónicos y mecánicos que permiten el accionamiento, control y comunicación del manipulador. El movimiento de las dos articulaciones se realiza mediante dos motores paso a paso **NEMA 17 HS4401**, uno por cada grado de libertad, proporcionando la precisión requerida para el posicionamiento del efector final.

Para la calibración de la posición de referencia (posición cero) de cada articulación, se integran dos potenciómetros de **10 kΩ**, en los ejes de las articulaciones principales, los cuales funcionan como sensores de posición. El control del sistema se implementa mediante un **Arduino Mega 2560**, encargado de procesar las señales de entrada, ejecutar los algoritmos de control y generar las señales necesarias para el accionamiento de los motores.

El control de potencia de los motores paso a paso se realiza mediante dos controladores **A4988**, los cuales permiten regular la corriente suministrada a los motores e implementar diferentes resoluciones de micropasos, mejorando la precisión y la suavidad del movimiento del manipulador (Negrete Martínez). La alimentación del sistema se proporciona mediante una fuente de corriente continua de **12 V y 4 A**, garantizando el suministro energético requerido por los actuadores y los dispositivos electrónicos.

Como medida de protección, se incorporan dos interruptores de límite (*limit switches*), cuya función es restringir el rango de movimiento de las articulaciones, evitando desplazamientos que puedan ocasionar daños mecánicos al robot (Negrete Martínez). Finalmente, la comunicación remota entre el sistema de control y el manipulador se establece mediante un módulo **ESP32** con conectividad Wifi, permitiendo la transmisión de comandos de operación y el monitoreo del estado del robot de manera inalámbrica

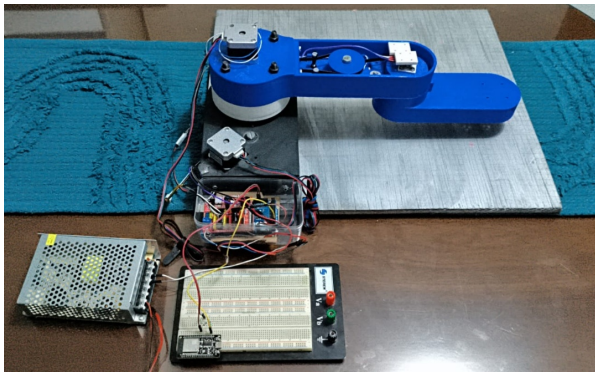


Figura 1. ***Componentes físicos del robot planar serial de 2 GDL.***

Software

El software desarrollado para el sistema de control del brazo robótico se divide en tres módulos principales: comunicación inalámbrica, control del manipulador y aplicación de usuario. Esta arquitectura modular facilita la integración de los diferentes componentes del sistema, así como su mantenimiento y escalabilidad.

El primer módulo corresponde a la programación del **ESP32**, encargado de establecer la conexión a la red Wi-Fi y gestionar la comunicación mediante el protocolo **MQTT**. A través de este protocolo, los comandos generados por el usuario son publicados en un bróker MQTT y posteriormente enviados al microcontrolador responsable del control del robot.

El segundo módulo se implementó en el entorno de desarrollo **Arduino IDE**, donde se programó el algoritmo de control del manipulador. El **Arduino Mega 2560** recibe los comandos enviados por el ESP32 mediante comunicación serial y, con base en ellos, genera las señales necesarias para accionar los controladores **A4988**, permitiendo el movimiento coordinado de los motores paso a paso que accionan las articulaciones del robot.

El tercer módulo consiste en el desarrollo de una aplicación móvil que proporciona una interfaz gráfica intuitiva para el usuario. Esta aplicación permite enviar comandos al sistema mediante el protocolo MQTT, facilitando el control remoto del manipulador desde cualquier dispositivo con acceso a Internet. La interfaz fue diseñada para simplificar la operación del robot, permitiendo que usuarios con conocimientos básicos puedan manipular el sistema de forma sencilla y segura.

Debido a la extensión del software desarrollado y a las limitaciones de espacio propias de un artículo científico, únicamente se describen los aspectos fundamentales de la implementación. El código fuente completo, junto con la documentación y los archivos del proyecto, se encuentra disponible en un repositorio público de GitHub para su consulta y reproducción.

https://github.com/Jagtz-R/Verano-UG

Comunicación inalámbrica

La programación del módulo **ESP32** se estructuró de forma modular mediante la implementación de múltiples archivos fuente, con el propósito de separar las responsabilidades de cada componente del sistema, facilitar el mantenimiento del código y favorecer su escalabilidad para futuras actualizaciones. Esta organización permite mejorar la legibilidad del software y simplificar la incorporación de nuevas funcionalidades sin afectar el funcionamiento de los demás módulos.

La arquitectura del software se basa en un conjunto de archivos denominados *Manager*, cada uno de los cuales implementa una función específica dentro del sistema. Entre sus responsabilidades se encuentran la gestión de la conexión a la red Wifi, el establecimiento de la comunicación mediante el protocolo MQTT, el intercambio de información a través del puerto serial y la interpretación de los comandos recibidos.

Cada módulo fue diseñado siguiendo el principio de responsabilidad única, de manera que cada archivo se encarga exclusivamente de una función determinada. Por ejemplo, el **WiFiManager** administra únicamente la conexión inalámbrica del dispositivo, mientras que las tareas relacionadas con el procesamiento de comandos, la comunicación serial y la publicación de mensajes MQTT son ejecutadas por sus respectivos módulos. Esta separación funcional incrementa modularidad del software, reduce el acoplamiento entre componentes y facilita tanto la depuración como la reutilización del código. La descripción de los principales archivos que conforman esta arquitectura se presenta en la **Tabla 1**

Tabla 1. Descripción de los archivos main.

Archivo	Descripción

wifi_manager	Gestiona la conexión del ESP32 a la red WiFi y espera indefinidamente hasta conectarse (hasta que se le asigne un IP).
mgtt_manager	Gestiona la conexión con el bróker MQTT, la recepción de mensajes y la publicación de respuestas.
command_manager	Interpreta los comandos recibidos por MQTT y actualiza el estado del sistema.
serial_manager	Envía comandos al controlador del robot mediante Serial2 y recibe sus respuestas.

El segundo grupo de archivos corresponde a los módulos denominados **Robot**, cuya función es definir las estructuras de datos, variables de estado y modelos utilizados para representar el comportamiento del manipulador robótico. Estos archivos constituyen la base lógica del sistema, proporcionando una representación consistente del estado del robot que puede ser utilizada por los distintos módulos de software.

A diferencia de los archivos *Manager*, los módulos **Robot** no realizan operaciones relacionadas con el control directo del hardware ni con la gestión de las comunicaciones. Su propósito es almacenar y organizar la información necesaria para el funcionamiento del sistema, facilitando el intercambio de datos entre los diferentes componentes de la arquitectura de software. De esta manera, los módulos encargados de la comunicación, el control y la interpretación de comandos pueden acceder a un modelo común del estado del robot, promoviendo una arquitectura modular y reduciendo el acoplamiento entre componentes.

La descripción de los archivos que conforman este grupo se presenta en la **Tabla 2.**

***Tabla 2**. Descripción de los archivos Robot.*

Archivo	Descripción	Comando/Respuesta
robot_command.h	Define los comandos que puede recibir el robot. Los comandos llegan desde MQTT en forma JSON y son convertidos a estructuras para ser enviados al robot por medio de Serial2.	MOVE HOME STOP STATUS CALIBRATE UNKNOWN
robot_response.h	Define las respuestas publicadas por MQTT. La respuesta es dada según la interpretación de los acuses de recibo (ACK) recibidos desde el Serial2 o al detectar errores locales.	OK ERROR BUSY READY UNKNOWN
robot_context.h	Contiene el estado global del robot y permite coordinar el flujo entre MQTT, comandos y comunicación serial.	IDLE COMMAND_READY SENDING_SERIAL WAITING_SERIAL_RESPONSE RESPONSE_READY

La arquitectura modular propuesta favorece la escalabilidad y el mantenimiento del sistema, ya que permite incorporar nuevas funcionalidades sin afectar significativamente el funcionamiento de los demás componentes. La adición de nuevos comandos o respuestas requiere únicamente la definición de las estructuras correspondientes dentro de los módulos **Robot**, así como la implementación de su procesamiento

en los módulos encargados de la interpretación y ejecución de instrucciones. Este enfoque modular reduce el acoplamiento entre componentes, facilita la reutilización del código y simplifica futuras tareas de actualización y mantenimiento.

Un componente fundamental de la arquitectura es el módulo **Config**, el cual centraliza todos los parámetros de configuración del sistema. En este archivo se definen las credenciales de acceso a la red inalámbrica (SSID y contraseña), la dirección IP y el puerto del bróker MQTT utilizado para la comunicación, los tópicos destinados al envío de comandos y la recepción de respuestas, la configuración de la interfaz de comunicación serial UART mediante **Serial2** (pines RX y TX) entre la ESP32 y el controlador del robot, así como el tiempo máximo de espera para la recepción de una respuesta después del envío de un comando. La centralización de estos parámetros permite realizar modificaciones desde un único punto del software, facilitando la configuración inicial del sistema, la portabilidad de la aplicación y las labores de mantenimiento, además de minimizar la posibilidad de errores derivados de configuraciones distribuidas en múltiples archivos.

Finalmente, los archivos **Main** constituyen el punto de entrada de la aplicación y coordinan la ejecución del sistema. Su funcionamiento se fundamenta en las funciones `setup()` y `loop()`, propias del entorno de desarrollo Arduino. La función `setup()` se ejecuta una única vez durante el inicio del sistema e inicializa la comunicación serial, la interfaz de comunicación con el robot, la conexión a la red Wi-Fi y el establecimiento de la comunicación con el bróker MQTT. Posteriormente, la función `loop()` se ejecuta de forma continua, encargándose de mantener activa la conexión MQTT, procesar los comandos recibidos, traducirlos en acciones de control, transmitirlos al microcontrolador mediante la interfaz **Serial2** y publicar las respuestas generadas en los tópicos MQTT correspondientes. Este esquema de funcionamiento garantiza la operación continua del sistema y una comunicación confiable entre la aplicación de usuario y el brazo robótico.

Control del manipulador

El archivo **Robot2gdl_Control.ino** constituye el programa principal ejecutado en el microcontrolador Arduino Mega 2560, el cual actúa como controlador del manipulador robótico. Su función principal es recibir, interpretar y procesar los comandos transmitidos por el módulo ESP32 mediante comunicación serial UART, transformándolos en señales de control para el accionamiento de los motores paso a paso responsables del movimiento de las articulaciones del robot.

El algoritmo desarrollado genera las señales de pulso y dirección requeridas por los controladores A4988, permitiendo el control independiente y asíncrono de cada motor paso a paso. Esta estrategia posibilita la ejecución simultánea de los movimientos articulares, lo que contribuye a mejorar la sincronización, la suavidad del desplazamiento y el desempeño dinámico del manipulador durante la ejecución de trayectorias.

El sistema implementa un controlador de posición angular, en el cual las instrucciones de entrada contienen los valores objetivo de cada articulación. A partir de dichos valores, el programa calcula el número de pasos requeridos para cada motor y genera la secuencia de pulsos correspondiente hasta alcanzar la posición deseada, garantizando un posicionamiento preciso del efector final dentro del espacio de trabajo del robot.

Asimismo, el software incorpora rutinas para la recepción y validación de comandos, la ejecución de procedimientos de calibración y retorno a la posición inicial (Home), el control del movimiento de las articulaciones y el envío de mensajes de respuesta al módulo ESP32, permitiendo informar el estado de ejecución de cada instrucción. Los comandos admitidos por el controlador y las respuestas generadas por el sistema se presentan en la **Tabla 3**.

***Tabla 3**. Comandos de entrada y salida.*

Entrada	Salida
CMD:CALIBRATE	ACK:OK
CMD:HOME	ACK:BUSY
CMD:MOVE,th1,th2	

CMD:STOP	ACK:READY
CMD:STATUS	ACK:ERROR

El sistema implementa cinco comandos principales para el control del manipulador. El comando **CALIBRATE** realiza la calibración de las articulaciones utilizando la información proporcionada por los potenciómetros. El comando **HOME** retorna el robot a su posición de referencia o posición inicial. El comando **MOVE** ejecuta el desplazamiento de las articulaciones hacia los ángulos especificados por los parámetros *th1* y *th2*. Por su parte, **STOP** detiene el movimiento de los motores en cualquier instante, mientras que **STATUS** permite consultar el estado actual del manipulador, indicando si el sistema se encuentra disponible o ejecutando una operación.

Al inicio del programa se incorpora la biblioteca **AccelStepper**, la cual permite implementar perfiles de aceleración y desaceleración para los motores paso a paso, logrando un arranque gradual y un movimiento más preciso (Librería La Tijera, 2025). Asimismo, se definen las constantes **PASOS_GRADO_1** y **PASOS_GRADO_2**, encargadas de convertir los desplazamientos angulares en el número de pasos requerido por cada motor.

Con el propósito de mejorar la confiabilidad de la calibración, la lectura de los potenciómetros se procesa mediante un filtro de media, implementado en la función **readPot()**, la cual obtiene el promedio de 20 muestras adquiridas con un intervalo de 2 ms entre cada una. Este procedimiento reduce el efecto del ruido presente en la señal y proporciona una medición más estable de la posición de las articulaciones. Posteriormente, el programa define las rutinas correspondientes a cada uno de los comandos implementados para el control del manipulador.

La rutina **CALIBRATE** determina la posición inicial de cada articulación a partir de la lectura de los potenciómetros y calcula el error respecto a la referencia establecida. Cuando dicho error excede la tolerancia definida, el controlador acciona los motores en la dirección correspondiente hasta alcanzar la posición objetivo. Una vez finalizado el proceso de calibración, los contadores internos de posición se reinician y el sistema informa que el manipulador se encuentra listo para recibir nuevas instrucciones.

La función **HOME** desplaza el robot hacia su posición de referencia, mientras que la función **moveRobot** recibe los ángulos objetivo de las articulaciones y los convierte en el número equivalente de pasos mediante los factores de conversión **PASOS_GRADO**. Posteriormente, el controlador genera las señales necesarias para posicionar los motores en los valores requeridos. Por su parte, la función **stopRobot** detiene de manera inmediata el movimiento de las articulaciones.

La recepción y el procesamiento de los comandos se realizan mediante la función **readCommands()**, la cual supervisa continuamente el estado del sistema. Cuando el manipulador se encuentra ejecutando una operación de movimiento o calibración, el controlador mantiene la ejecución de la tarea hasta su finalización. Una vez concluido el proceso, se transmite el mensaje **Ready**, indicando que el robot se encuentra disponible para recibir una nueva instrucción.

Aplicación móvil

Antes de describir la implementación de la aplicación móvil, es conveniente presentar los conceptos fundamentales relacionados con las tecnologías empleadas en su desarrollo. La aplicación fue desarrollada utilizando el lenguaje de programación **Dart** y el framework **Flutter**, una plataforma de desarrollo multiplataforma creada por Google que permite construir aplicaciones nativas para dispositivos móviles a partir de una única base de código. Dart es un lenguaje de programación orientado a objetos, diseñado para ofrecer un alto rendimiento y una ejecución eficiente tanto durante el desarrollo como en la compilación final de las aplicaciones.

Flutter basa su arquitectura en el uso de **widgets**, los cuales representan los componentes fundamentales de la interfaz de usuario. Un widget puede corresponder a un elemento visual, como un botón, un campo de texto o una imagen, o bien a un componente encargado de administrar el comportamiento y la disposición de otros elementos. Los widgets se organizan mediante una estructura jerárquica en forma de árbol (*Widget*

Tree), donde cada widget puede contener otros widgets como hijos. Esta organización favorece el desarrollo de interfaces modulares, reutilizables y de fácil mantenimiento, además de facilitar la actualización dinámica de la interfaz conforme cambia el estado de la aplicación.

Adicionalmente, Flutter proporciona un amplio conjunto de bibliotecas y herramientas que simplifican tareas comunes durante el desarrollo, tales como el renderizado de la interfaz gráfica, la administración del estado de la aplicación, el acceso a recursos del dispositivo y la comunicación mediante protocolos de red. Estas características permiten concentrar el desarrollo en la lógica funcional de la aplicación, reduciendo la complejidad asociada a la implementación de interfaces móviles.

La estructura del proyecto se organizó siguiendo la arquitectura recomendada por el kit de desarrollo de software (SDK) de Flutter, con el propósito de mantener una adecuada separación entre la interfaz de usuario y la lógica de la aplicación. El código fuente se encuentra principalmente en la carpeta **lib**, la cual contiene las subcarpetas **screens** y **services**. La carpeta **screens** almacena las pantallas que conforman la interfaz gráfica de usuario, implementadas mediante los archivos **login_screen.dart** y **control_screen.dart**, mientras que la carpeta **services** agrupa los módulos responsables de la lógica de negocio y de la comunicación con el servidor MQTT, destacando el archivo **mqtt_service.dart**. Finalmente, el archivo **main.dart** constituye el punto de entrada de la aplicación, donde se inicializa Flutter y se define la pantalla principal desde la cual comienza la ejecución del sistema.

El archivo *main* es el encargado de realizar el llamado a la aplicación. El ciclo *void main* contiene a *runApp*, esta función lanza la aplicación hasta que esta se detiene por alguna instrucción o hasta que el usuario cierra la aplicación. En la clase *MyClass* se tiene a la aplicación heredada con widgets estáticos, estos mantienen el bróker y el tópico fijos para asegurar que el mensaje llegue, también se le indica que la primera ventana será la contenida en el archivo *login_screen.*

El archivo **mqtt_service.dart** implementa la comunicación entre la aplicación móvil y el manipulador robótico mediante el protocolo **MQTT (Message Queuing Telemetry Transport)**. Este módulo concentra toda la lógica relacionada con la conexión al bróker MQTT, la publicación de comandos, la suscripción a los tópicos de respuesta y la recepción de mensajes provenientes del sistema de control, proporcionando una interfaz de comunicación independiente del resto de la aplicación. La implementación se basa en el uso de funciones de tipo **Future**, las cuales permiten ejecutar operaciones asíncronas sin bloquear el hilo principal de ejecución de la aplicación. Este mecanismo es esencial para preservar la capacidad de respuesta de la interfaz gráfica durante el establecimiento de conexiones, la transmisión de comandos y la recepción de respuestas del manipulador, evitando interrupciones en la experiencia del usuario (Flutter, s.f.).

De manera complementaria, el módulo emplea **callbacks** para gestionar los eventos asociados con la comunicación MQTT, incluyendo el establecimiento y cierre de la conexión, la recepción de mensajes publicados por el servidor, la detección de errores y la actualización del estado de la comunicación. Esta arquitectura orientada a objetos permite que la aplicación responda de forma inmediata a los cambios en la conexión y procese las respuestas del robot en tiempo real. Entre las funciones más relevantes del módulo se encuentra **publishMessage()**, encargada de publicar el contenido del mensaje (*payload*) en el tópico (*topic*) correspondiente utilizando el nivel de **Calidad de Servicio (Quality of Service, QoS)** previamente configurado. Esta función encapsula el proceso de construcción y envío de los mensajes MQTT, garantizando una transmisión confiable de los comandos hacia el manipulador robótico y simplificando la interacción entre la interfaz de usuario y la infraestructura de comunicación.

El archivo **login_screen.dart** implementa la pantalla de autenticación y configuración inicial de la aplicación móvil. En este módulo se importa el servicio de comunicación definido en **mqtt_service.dart** y se implementa una clase con manejo de estado (*StatefulWidget*), lo que permite actualizar dinámicamente los elementos de la interfaz en función de las acciones realizadas por el usuario, sin necesidad de reconstruir completamente la pantalla. La interfaz incorpora dos campos de entrada destinados a la configuración de la dirección IP y de la dirección del bróker MQTT. Ambos parámetros se inicializan con valores predeterminados correspondientes al entorno de desarrollo, aunque pueden ser modificados por el usuario antes de establecer la comunicación con el sistema. Al seleccionar la opción **Conectar**, la aplicación invoca los métodos implementados en **mqtt_service.dart** para establecer la conexión con el servidor MQTT. Posteriormente, el resultado del proceso de conexión es verificado y notificado al usuario mediante mensajes mostrados en la interfaz gráfica. El diseño de la pantalla se desarrolla utilizando la estructura jerárquica de widgets propia de

Flutter. En particular, la clase **Scaffold** proporciona la estructura principal de la interfaz, organizando los diferentes componentes visuales, como la barra de navegación, los campos de entrada y los botones de interacción. La pantalla de configuración inicial desarrollada se presenta en la **Figura 2-a)**.

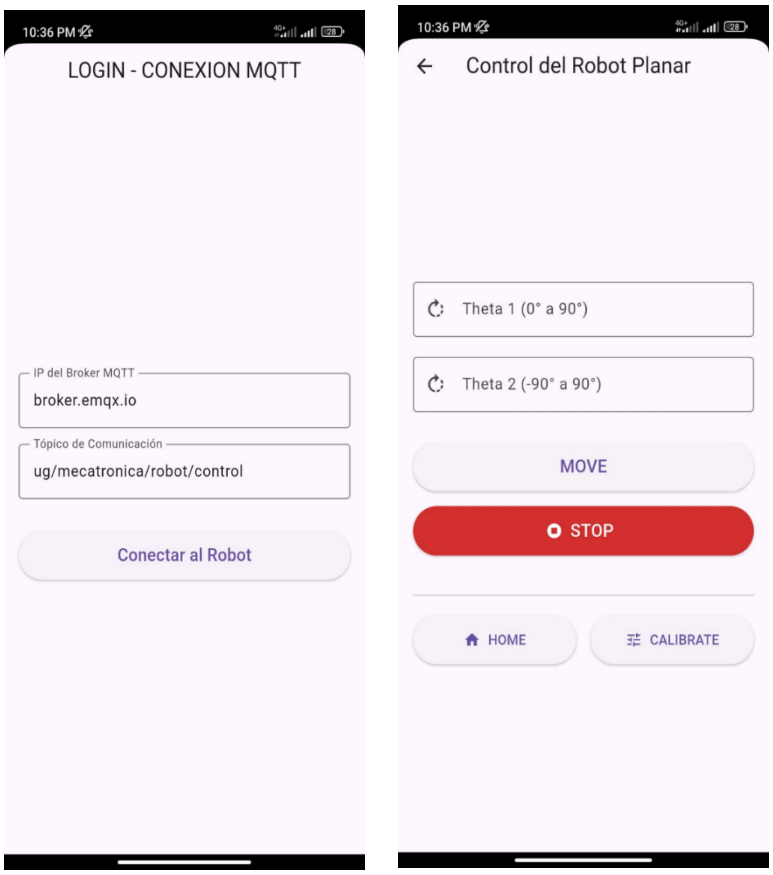
El archivo **control_screen.dart** implementa la interfaz encargada del control remoto del manipulador robótico. Al igual que **login_screen.dart**, este módulo utiliza el servicio de comunicación definido en **mqtt_service.dart** para establecer la conexión con el bróker MQTT y transmitir los mensajes correspondientes. Asimismo, emplea la clase **Scaffold** como estructura base para la organización de los componentes gráficos de la aplicación. A diferencia de la pantalla de autenticación, esta interfaz está orientada al envío de instrucciones de control hacia el robot. Para ello, incorpora diferentes elementos de interacción que permiten al usuario introducir los parámetros requeridos por cada comando. Tómese como ejemplo el comando **MOVE**, la interfaz dispone de dos campos de entrada correspondientes a los ángulos articulares $\boldsymbol{\theta_1}$ y $\boldsymbol{\theta_2}$, expresados en grados. El rango permitido para $\boldsymbol{\theta_1}$ es de 0° a 90°, mientras que $\boldsymbol{\theta_2}$ admite valores entre −90° y 90°, límites que corresponden al espacio de trabajo definido para el manipulador, los valores ingresados por el usuario son convertidos de cadenas de caracteres a valores numéricos tipo **double** mediante la función **tryParse()**, la cual permite realizar la conversión de manera segura sin generar excepciones cuando la entrada es inválida. Dado que el lenguaje **Dart** incorpora seguridad frente a valores nulos (*null safety*), el sistema verifica que ambos campos contengan información válida antes de generar el comando. En caso de que alguno de los parámetros no sea numérico, permanezca vacío o se encuentre fuera de los límites establecidos, la aplicación informa la situación al usuario mediante mensajes de validación, evitando el envío de instrucciones incorrectas al sistema robótico. Una vez validados los parámetros, la aplicación construye el comando en formato **JSON** mediante la creación de un diccionarios udsando **Map**, en el cual cada clave representa un atributo del comando y cada valor corresponde a la información asociada, como el nombre de la instrucción y los ángulos de movimiento. La estructura utilizada para el empaquetado de los datos se presenta en la **Tabla 4**. Posteriormente, este objeto es serializado utilizando la función **jsonEncode()**, obteniendo una cadena de texto que constituye el *payload* del mensaje MQTT. Finalmente, dicha cadena es enviada al servidor mediante la función **publishMessage()** implementada en **mqtt_service.dart**, permitiendo que el módulo ESP32 reciba, interprete y ejecute la instrucción correspondiente sobre el manipulador robótico.

Tabla 4. *Mapeo para el comando move.*

String (carácter)	Dynamic (Dinamico)
"command"	"move",
"joint1	theta1,
"joint2":	theta2

Las demás funciones implementadas en la pantalla de control siguen la misma metodología descrita para el comando **MOVE**. En todos los casos se realiza la validación de los datos de entrada, la construcción del mensaje en formato **JSON**, la serialización del contenido y su posterior envío mediante el servicio de comunicación MQTT. Debido a que el procedimiento de implementación es análogo para el resto de los comandos, se omite su descripción detallada con el fin de evitar redundancias. La interfaz correspondiente a la pantalla de control desarrollada se muestra en la **Figura 2-b)**.

Por otra parte, la navegación entre pantallas se gestiona mediante el sistema de rutas proporcionado por Flutter, el cual administra una pila (*stack*) de vistas siguiendo el principio **LIFO (Last In, First Out)**. Bajo este esquema, cada nueva pantalla se coloca sobre la anterior sin eliminarla de la memoria; cuando el usuario regresa a la pantalla previa, la última vista agregada es retirada de la pila, restaurándose automáticamente la interfaz anterior junto con su estado. Este mecanismo simplifica la navegación entre las diferentes interfaces de la aplicación y permite conservar la información asociada a cada pantalla durante la interacción del usuario.



a) *b)*

Figura 2. Vista móvil de las pantallas, a) Pantalla de carga, b) Pantalla de control

Resultados y discusión

Durante el desarrollo del sistema se presentaron diversos retos técnicos relacionados con la selección de las tecnologías de desarrollo y la implementación de la comunicación remota. Uno de los primeros aspectos analizados fue la elección de la plataforma para el desarrollo de la aplicación móvil. Entre las alternativas consideradas se encontraba **MIT App Inventor**, herramienta que ofrece un entorno de programación visual orientado a bloques y facilita el desarrollo de aplicaciones móviles de manera intuitiva. Sin embargo, tras evaluar las características de ambas plataformas, se optó por utilizar **Dart** y **Flutter**, debido a las ventajas que ofrecen para el desarrollo de aplicaciones con arquitecturas de mayor complejidad.

La primera ventaja corresponde a la **escalabilidad y mantenibilidad del software**. Mientras que MIT App Inventor almacena los proyectos principalmente en la nube y utiliza un entorno gráfico de programación, Flutter emplea archivos de texto plano, lo que facilita la integración con sistemas de control de versiones como Git y plataformas colaborativas como GitHub. Esta característica resulta especialmente útil para proyectos de investigación, donde la reproducibilidad, el trabajo colaborativo y el mantenimiento del código representan aspectos fundamentales.

La segunda ventaja está relacionada con el **manejo de operaciones asíncronas**. El lenguaje Dart incorpora mecanismos como *Future* y *async/await*, que permiten ejecutar tareas de comunicación en segundo plano sin afectar la capacidad de respuesta de la interfaz gráfica. Gracias a ello, la aplicación mantiene una interacción fluida con el usuario mientras se establece la conexión con el servidor MQTT o se realiza el intercambio de mensajes con el manipulador robótico.

Finalmente, Flutter ofrece un mayor **nivel de personalización y acceso a los servicios del sistema**, permitiendo implementar arquitecturas de comunicación más robustas y protocolos personalizados. Estas características proporcionan una mayor flexibilidad para integrar servicios de red, administrar conexiones y desarrollar interfaces gráficas adaptadas a los requerimientos específicos del sistema de control remoto.

Otro desafío importante se presentó durante la implementación de la comunicación mediante **WebSockets**, mecanismo que permite establecer una comunicación bidireccional en tiempo real entre clientes y servidores sobre una única conexión TCP (ZetCode, 2025). Durante las pruebas con el bróker EMQX se observó que la conexión no podía realizarse utilizando únicamente la dirección del servidor, ya que era necesario emplear el punto de acceso correspondiente al servicio MQTT sobre WebSocket. En consecuencia, la dirección de conexión debió configurarse con la ruta **ws://broker.emqx.io/mqtt**, correspondiente al puerto destinado al servicio WebSocket del bróker, permitiendo establecer correctamente la comunicación entre la aplicación móvil y el servidor MQTT.

A pesar de las dificultades encontradas durante el desarrollo, el sistema fue implementado satisfactoriamente y cumplió con los objetivos planteados. Las pruebas experimentales se realizaron utilizando un robot planar serial de dos grados de libertad desarrollado por la alumna **Ana Cristina Negrete Martínez**, de la Universidad de Guanajuato, como parte de su trabajo de tesis. El prototipo contaba con la estructura mecánica, los motores paso a paso y la electrónica de potencia necesaria para el accionamiento del manipulador. Como parte de la implementación propuesta, se integró el módulo ESP32 con el controlador basado en Arduino Mega 2560 mediante comunicación serial UART, además de incorporar la fuente de alimentación requerida para energizar el sistema de comunicación inalámbrica. La configuración final del prototipo experimental se presenta en la **Figura 1**.

Con el propósito de validar el funcionamiento del sistema desarrollado, se realizaron pruebas de operación remota enviando diferentes comandos desde la aplicación móvil, incluyendo instrucciones de movimiento con distintos valores de ángulos articulares y comandos de control del manipulador. Los resultados obtenidos demostraron el correcto funcionamiento de la arquitectura propuesta, verificando la comunicación bidireccional entre la aplicación móvil, el servidor MQTT, el módulo ESP32 y el controlador del robot. Asimismo, el funcionamiento completo del sistema puede observarse en el video disponible en el enlace proporcionado, donde se muestran las diferentes pruebas de operación realizadas sobre el manipulador robótico.

https://ugtomx-my.sharepoint.com/:v:/g/personal/as_rocharamirez_ugto_mx/IQAox9asa0IZT4VYO0xqIA9yAYoYdvxcs-hS-OSoFryotP0?e=i4Ts6S

Bibliografía/Referencias

Flutter. (s.f.). Obtenido de Future<T> class: https://api.flutter.dev/flutter/dart-async/Future-class.html

Librería la tijera. (10 de Abril de 2025). Obtenido de Guía completa de AccelStepper con Arduino y L298N: https://librerialatijera.com.ar/aprender-a-usar-libreria-accelstepper/#que_es_la_libreria_accelstepper_

Negrete Martínez, A. C. (s.f.). *Diseño, instrumentación y control de robot planar de dos grados de libertad de bajo costo.* Salamanca: Universidad de Guanajuato.

ZetCode. (4 de Abril de 2025). Obtenido de Dart WebSocket: https://www.zetcode.com/dart/websocket/