CEA Comité Español de Automática

XLV JORNADAS AUTOMÁTICA MÁLAGA 2024

UNIVERSIDADE DA CORUÑA

Jornadas de Automática

Una interfaz gráfica para monitorización, control y teleoperación de robots móviles

Vilella-Cantos, J.[a], Ballesta, M.[a], Valiente, D.[a], Gil, A.[a], Máximo, M.[a], Reinoso, O.[a]

[a] *Instituto de Investigación en Ingeniería de Elche (I3E), Universidad Miguel Hernández de Elche, Avda. de la Universidad s/n, Edificio Innova, 03202, Elche, Alicante, España.*

To cite this article: Vilella-Cantos, J., Ballesta, M., Valiente, D., Gil, A., Máximo, M., Reinoso, O. 2024. A graphic interface for mobile robots monitorization, control and teleoperation.
Jornadas de Automática, 45. https://doi.org/10.17979/ja-cea.2024.45.10798

Resumen

El presente artículo tiene como objetivo mostrar el diseño y desarrollo de una interfaz gráfica para monitorización, teleoperación y control de un robot móvil, la cual permite ver los datos de los sensores del robot además de comandarlo manualmente y realizar una planificación y seguimiento de trayectorias. La interfaz ha sido implementada sobre Foxglove Studio, una herramienta de visualización de datos de robots que trabaja sobre ROS (*Robot Operating System*) en cualquiera de sus versiones. Para la comunicación entre la interfaz y el robot se ha hecho uso de *Rosbridge*, un paquete que facilita la interacción con ROS a través de *Websocket*. De este modo, el robot (el cual usa la versión de ROS1 Noetic) publica sus datos en el servidor *Rosbridge* mediante código Python. La interfaz es cliente de este servidor, teniendo acceso a todos los datos publicados por el robot y pudiendo publicar a su vez información. Dicha interfaz se ha desarrollado creando componentes personalizados mediante el *framework* React y el lenguaje de programación Typescript.

Palabras clave: Robots móviles, Planificación de trayectorias, Seguimiento de trayectorias, Integración de sensores y percepción, Teleoperación, Control remoto y distribuido, Adquisición remota de datos de sensores

A graphic interface for mobile robots monitorization, control and teleoperation

Abstract

This paper's goal is to portray the design and development of a graphic interface for monitoring, teleoperation and control of a mobile robot, which provides sensor data visualization, manual command and path planning and following. The interface has been developed over Foxglove Studio, a tool for visualizing robot data that works with ROS (Robot Operating System) in any of its versions. For the communication between the robot and the interface Rosbridge, a package that provides tools for the interaction with ROS via Websocket, was used. This way, the robot (which uses the ROS1 version Noetic) publishes its data on the Rosbridge server using Python code. The interface retrieves the data on the client side, having access to the robot's published information and being able to publish its own data. The interface was developed creating custom components using the React framework with the programming language Typescript.

Keywords: Mobile robots, Trajectory and path planning, Trajectory tracking and path following, Sensor integration and perception, Teleoperation, Remote and distributed control, Remote sensor data acquisition

1. Introducción

En sistemas robóticos complejos que integran sensores de distinto tipo, es conveniente disponer de una manera de tener acceso a los datos de estos en tiempo real desde cualquier parte, además de mostrar estos datos de una manera visualmente atractiva e intuitiva. Del mismo modo, es conveniente el poder enviar nuevas instrucciones al robot de manera remota, y que éste procese esta nueva información y actúe en consecuencia.

Herramientas como (Foxglove Studio, 2021) ofrecen una

*Autor para correspondencia: jvilella@umh.es

respuesta a esta necesidad, implementando componentes que permiten visualizar los datos de los sensores de un robot a través de una conexión local o remota, esta última mediante el protocolo *Websocket* (Fette and Melkinov, 2011), o bien visualizando datos ya almacenados en un fichero con extensión *.bag* tras haber grabado los datos de los sensores en dicho fichero mediante la herramienta *Rosbag*, implementada en un paquete de ROS (*Robot Operating System*).

Si bien el mencionado software Foxglove Studio ofrece multitud de paneles para visualizar y distruibuir la información en base a las preferencias del usuario, no alcanza a suplir otras necesidades que frecuentemente aparecen a la hora de trabajar remotamente con robots. Por ejemplo, el hecho de enviar comandos al robot a través de tópicos de ROS y que el robot pueda procesar la información y actuar en consecuencia. Tener en cuenta estas posibilidades enriquecería fuertemente la experiencia de usuario con el software.

Con el objetivo de suplir las mencionadas carencias, el presente artículo propone una solución mediante paneles personalizados implementados sobre esta herramienta existente. Para ello, se ha hecho uso del lenguaje de programación Typescript junto con el *framework* (React, 2013), ya que son las tecnologías sobre las que están construidos los componentes por defecto de Foxglove Studio.

La comunicación entre el robot y la interfaz se hace a través de un servidor *Rosbridge*, el cual permite que se publiquen tópicos y se lean a través de conexiones *Websocket*. La Figura 1 muestra un esquema que ilustra las conexiones entre los distintos componentes que entran en juego en la aplicación. En el presente artículo, se repasa el estado del arte para la visualización de datos y control de robots y la comunicación entre máquinas con ROS. En el apartado *Implementación de la interfaz* se muestra cómo se ha partido de estos conceptos a la hora de construir el producto propuesto, haciendo un repaso de todos los componentes que forman la aplicación web. Por último, se realiza un balance del estado actual de la interfaz gráfica y se comentan mejoras a futuro.

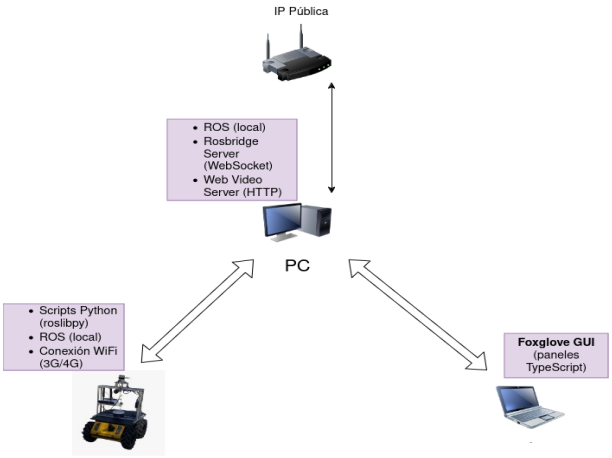

Figura 1: Esquema de conexiones.

En cuanto al robot móvil utilizado en el contexto del presente artículo, se ha utilizado el modelo Husky UGV de la marca Clearpath Robotics, como el que se observa en la Figura 2. Éste lleva integrados una serie de sensores, entre los que se encuentran: un sensor LiDAR de la marca Ouster, modelo OS1-128-U; dos cámaras Fujinon Fish Eye y un sensor GPS+RTK tractorDrive. Para la conexión con el servidor *Rosbridge* en todo momento mediante peticiones HTTP (*Hypertext Transfer Protocol*) y comunicación *Websocket*, el robot cuenta con un router MikroTik Chateau 5G, el cual proporciona conexión 3G, 4G, 5G y WiFi.

Figura 2: Robot Husky UGV navegando en exteriores.

2. Estado del arte

El uso de herramientas software que permitan visualizar e interactuar con sistemas robóticos, ya sea local o remotamente, son esenciales para poder trabajar de manera intuitiva, comprensible y dinámica en estos entornos. Una de las herramientas de visualización más conocidas actualmente es RViz, la cual forma parte de ROS (Quigley et al., 2009) y ofrece la posibilidad de suscribirse a cualquiera de los tópicos activos en el sistema y mostrar la información que se publica en éstos en tiempo real, haciendo uso de elementos visuales adaptados al tipo de mensaje del tópico en cuestión. No obstante, RViz presenta una serie de limitaciones, entre ellas, el hecho de no ser flexible hace que no se adapte a las necesidades de una aplicación de usuario final. Por otro lado, existen otras de herramientas de visualización de datos anteriores a la salida de ROS, como RoboSim (Speck and Klaeren, 1999), la cual permite a su vez el control de los brazos robóticos que muestra la interfaz, sean simulados o reales; o la plataforma distribuida planteada por (Payá et al., 2006) para el robot WifiBot. En el caso de (Cho and Jeon, 2008), esta herramienta de visualización y control se ofrece en forma de aplicación móvil.

Algunos autores han planteado el uso de aplicaciones multiplataforma para que el usuario pueda tener acceso a la información del sistema robótico desde cualquier dispositivo de manera cómoda, presentando la información de manera visual y teniendo en cuenta las necesidades del usuario (Ilijoski et al., 2022). En este caso, es atractivo el abordar esta problemática desde la solución de una aplicación web, ya que ésta ofrece la posibilidad de acceder a los datos desde cualquier dispositivo, pudiendo usarse para un gran abanico de tareas. Por ejemplo, en (Ishibashi et al., 2013), se utiliza una aplicación de estas características para monitorizar remotamente procesos de agricultura que involucran a un robot.

Para hacer posible la transferencia de información entre cualquier aplicación que se esté utilizando desde el lado del usuario y el propio robot, la solución típica es la de utilizar un servidor web público. De este modo, los distintos agentes

de la comunicación (robots e interfaz) están conectados entre sí a través de un mismo intermediario. Concretamente, las comunicaciones mediante el protocolo HTTP construido sobre TCP (*Transmission Control Protocol*) se usan mucho en el contexto de aplicaciones web (Gourley and Totty, 2002), pero también se ha extendido el uso del protocolo *Websocket* a lo largo de los últimos años, debido a que, al evaluar su rendimiento, se ha comprobado que consume muy poco tráfico de red, concretamente consume el mismo tráfico de red que una comunicación TCP básica (Skvorc et al., 2014). La principal diferencia entre HTTP y *Websocket* es que HTTP se trata de un protocolo síncrono (solicitud/respuesta), mientras que *Websocket* es asíncrono. Esta es la razón por la que *Websocket* es notablemente más rápido y con una latencia mucho más reducida, lo que hace que sea un protocolo idóneo a la hora de implementar una arquitectura que involucre transferencia de datos en tiempo real (Pimentel and Nickerson, 2012).

En resumen, se puede observar que el hecho de tener acceso a los datos de un sistema robótico y poder interactuar con ellos en tiempo real es una solución muy demandada desde los comienzos de la implementación de estos sistemas y lo seguirá siendo a lo largo de los años. Por ello, en el presente artículo se presenta una implementación aplicada a robots móviles en un entorno web, utilizando el protocolo *Websocket* para obtener la información y enviar comandos en tiempo real evitando las subidas de latencia en la medida de lo posible.

3. Implementación de la interfaz

Para el correcto funcionamiento de la interfaz, se ha requerido el desarrollo de código desde los dos extremos de la comunicación: desde el robot móvil y desde la interfaz gráfica de usuario, los cuales se comunican entre sí a través de un PC que actúa como *middleware* teniendo en una IP pública los puertos correspondientes a la comunicación a través de *Rosbridge* y de un servidor de imágenes de ROS. La Figura 1 ilustra el esquema de conexiones necesario para la comunicación entre robots e interfaz a través de un PC que actúa como *middleware*, con un servidor *Rosbridge* y *Web video server* abiertos en distintos puertos.

3.1. Diseño

El diseño de la aplicación se ha planteado mediante pestañas, poniendo en cada una de estas pestañas uno de los componentes personalizados implementados, alcanzando un número de cinco pestañas. El diagrama de flujo de la Figura 3 ilustra las distintas pestañas presentes en la aplicación junto con las opciones que ofrece cada una de ellas. En la subsección *Componentes* se explica en detalle el funcionamiento y el propósito de cada una de estas pestañas.

3.2. Componentes de la interfaz

Una vez visto el diseño, se pasa a comentar en detalle cada una de las pestañas que ofrece la aplicación.

3.2.1. Mapa

En esta primera pestaña, se utiliza uno de los componentes proporcionados por la propia herramienta Foxglove Studio. Este componente se llama *Map* y permite, como su propio nombre indica, visualizar sobre un mapa de *OpenStreetMap* (Bennett, 2010) la información de todos los mensajes de tipo *LocationFix* o *GeoJSON*. Esto permite ver en tiempo real, como muestra la Figura 4, todas las posiciones GPS de todos los robots que se estén publicando activamente en el servidor, además de la información de los diversos caminos que hayan sido calculados para cada uno de ellos.

Figura 4: Aspecto visual de la pestaña ”Rutas”.

3.2.2. Localización

En este primer componente personalizado se ofrece la posibilidad de, para cada robot activo presente en el sistema, ver su posición en tiempo real y, haciendo click en el mapa de *OpenStreetMap* (Bennett, 2010) que se observa en la Figura 5, establecer uno o varios puntos que se podrán enviar al servidor publicándolos en un tópico */id_robot/clicked_point*. Este tópico contiene mensajes de tipo *foxglove_msgs/TextPrimitive* con la latitud y longitud del punto o los puntos marcados. Esta información se lee desde el lado del robot y se procesa para publicar en el servidor la información visual del camino, como muestran los componentes visuales aquí revisados, además de para iniciar el control del robot. El funcionamiento del planificador de trayectorias y control del robot en base a esta interacción se desarrolla en el apartado *Planificador y seguidor de trayectorias*.

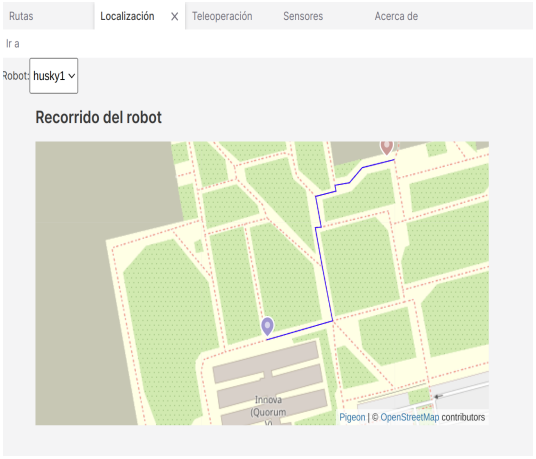

Figura 5: Aspecto visual de la pestaña ”Localización”.

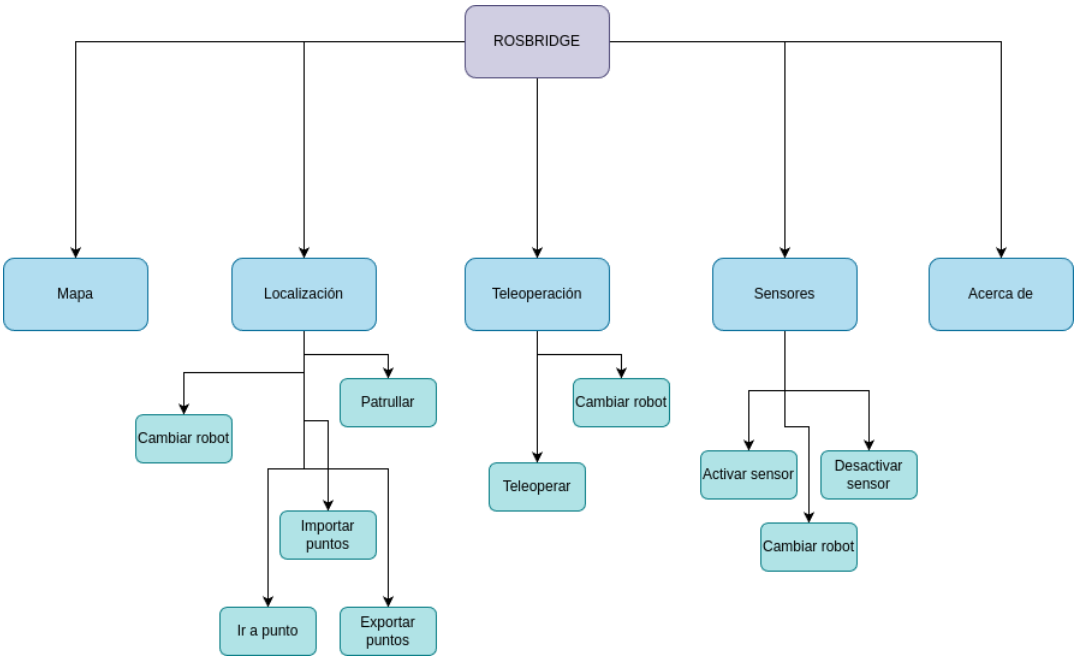

Figura 3: Diagrama de flujo de la interfaz gráfica.

3.2.3. *Teleoperación*

Este componente muestra, a modo de cuadrícula, cuatro imágenes correspondientes a sensores del robot seleccionado en el desplegable (siguiendo el mismo esquema de funcionamiento que el componente presentado anteriormente). Las imágenes pueden cambiarse seleccionando un tópico distinto en los cuatro desplegables contiguos al desplegable para seleccionar el robot activo, y ofrecerán como opciones los tópicos de tipo *sensor_msgs/Image* que están activos en el servidor de *Rosbridge*. Además, el componente contiene una barra que muestra la cantidad de carga que tiene la batería del robot (si esta está siendo publicada en el servidor *Rosbridge*), además de unos botones con flechas que permiten la teleoperación del robot a velocidad normal (0.2 m/s) o turbo (0.5 m/s), valor seleccionado en función al estado de un elemento de tipo *checkbox*. La Figura 6 muestra todos estos elementos en conjunto dentro del panel personalizado.

3.2.4. *Sensores*

En este panel, se muestran todos los sensores que está publicando en el servidor *Rosbridge* el robot seleccionado en el desplegable (desplegable presente en los anteriores dos componentes personalizados). Se muestra en una tabla, como se aprecia en la Figura 7, el nombre del tópico del sensor en cuestión, su estado (*ACTIVO* o *INACTIVO*) y, por último, una columna con un botón para activar o desactivar el sensor. La lógica de este botón consiste en publicar en el servidor un mensaje de tipo *foxglove_msgs/TextPrimitive* en el tópico */id_robot/signal_topic*, el cual contiene en una línea de texto el nombre del tópico pulsado y la palabra *ACTIVAR* o *DESACTIVAR*. Luego, desde el robot, se captura este mensaje y, en función de su valor, se actualiza una variable de tipo booleano para seguir publicando o no mensajes en el servidor *Rosbridge*. De este modo, es posible adaptarse a condiciones de baja transferencia, limitando la cantidad de datos que se envían del robot al servidor.

Figura 7: Aspecto visual de la pestaña ”Sensores”.

3.2.5. *Acerca de*

Esta pestaña muestra en forma de texto plano toda la información relacionada con la financiación que ha hecho posible el desarrollo de la presente aplicación web y del grupo de investigación en el cual ha sido desarrollada (ARVC, 2024).

3.3. *Scripts en el robot*

Desde el lado del robot, para hacer la información de los sensores visible para los usuarios de la aplicación web, se han desarrollado una serie de scripts utilizando el lenguaje de programación Python y la librería *roslibpy*. Esta librería ofrece clases y métodos que facilitan la conexión a un servidor *Rosbridge* y la recepción y envío de mensajes a través de este. Haciendo uso de dicha librería, se publican tópicos y mensajes con información de los sensores, además de mensajes con información procesada tras leer información del servidor.

Entre los scripts que se lanzan desde el robot para proporcionar los datos de los sensores a la interfaz, algunos de

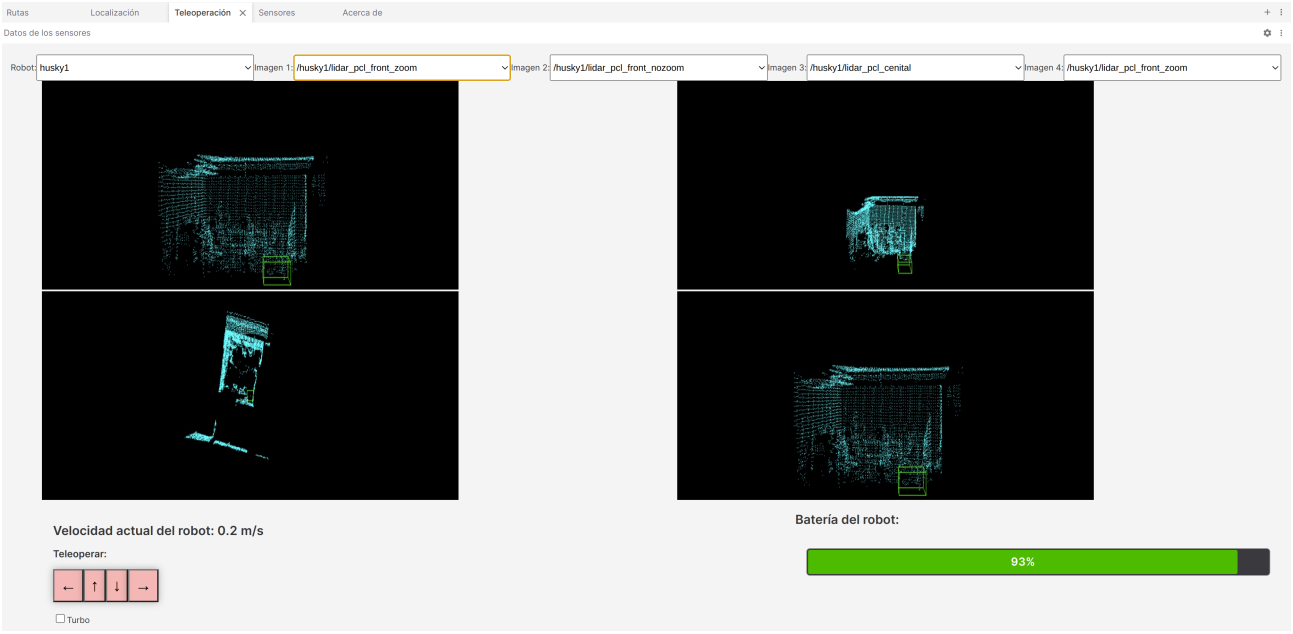

Figura 6: Aspecto visual de la pestaña "Teleoperación".

ellos procesan los datos para facilitar la comunicación, evitando problemas de latencia en la transferencia. Concretamente, en el caso de las nubes de puntos captadas por el sensor Ouster, se ha aplicado código para convertirlas en imágenes proyectadas en distintas vistas, publicando cada vista (frontal, frontal con zoom y cenital) en un tópico distinto de tipo *sensor_msgs/Image*. Para realizar esta conversión se hace uso de la biblioteca de visión por computador *OpenCV*. Hay que tener en cuenta que la comunicación a través de *Rosbridge* se realiza enviando los mensajes y recibiéndolos en formato JSON, lo que haría muy costoso el enviar todos los puntos de la nube (sobre todo teniendo en cuenta que nuestro sensor es de 128 canales) en un diccionario.

A continuación, se muestra un ejemplo de un código genérico para republicar los datos de un sensor (en este caso, del sensor del GPS) del robot en el servidor de *Rosbridge*, haciendo uso de la librería *roslibpy* previamente mencionada para la conexión remota y de la librería *rospy* para interactuar con ROS localmente.

```
#!/usr/bin/env python

import math
import roslibpy
import rospy
from foxglove_msgs.msg import LocationFix
from sensor_msgs.msg import NavSatFix
from rospy_message_converter import message_converter

class sendMap(object):
    def __init__(self):
        self.publish_map = True
        self.client = roslibpy.Ros(host=rosbridge_ip, port=rosbridge_port)
        self.topic_to_publish = robot_name + map_topic
        self.topic = roslibpy.Topic(self.client, self.topic_to_publish,
        "foxglove_msgs/LocationFix")
        self.map_sub = rospy.Subscriber(robot_map_topic,NavSatFix,self.callback)
        signal_topic = robot_name + signal_topic
        self.signal = roslibpy.Topic(self.client, signal_topic,
        "foxglove_msgs/TextPrimitive")
        self.topic.advertise()
        self.signal.subscribe(self.callback_signal)
        self.client.run_forever()

    def callback_signal(self, msg):
        topic = msg['text']
        if self.topic_to_publish in topic:
            if "DESACTIVAR" in topic:
                self.publish_map = False
            else:
                self.publish_map = True


    def callback(self, data):
        if self.publish_map:
            if(not math.isnan(data.latitude)):
                loc = LocationFix()
                loc.latitude = data.latitude
                loc.longitude = data.longitude
                loc.altitude = data.altitude
                self.topic.publish(roslibpy.Message(message_converter.
                convert_ros_message_to_dictionary(loc)))

if __name__ == '__main__':
    rospy.init_node('send_map', anonymous=True)
    sm = sendMap()
    print("Shutting down")
```

3.3.1. *Servidor de imágenes*

También se emplea código Python para publicar los datos de los sensores de tipo imagen en un puerto extra, que ya no estaría relacionado con *Rosbridge*, sino con el paquete de ROS *Web video server*, ejecutándose en un puerto público en la misma máquina que el servidor. De este modo, se pueden ver las imágenes directamente en la interfaz a través de la URL que proporciona dicho paquete, obteniendo los datos a través del protocolo HTTP, lo que facilita la inclusión de las imágenes en el componente personalizado de manera mucho más sencilla que teniendo que estar suscrito a todos los tópicos de tipo *sensor_msgs/Image* y esperar mensajes nuevos de cada uno de ellos, lo que consumiría muchos recursos desde el lado de la interfaz. Para ver el resultado visual, véase el apartado del componente *Teleoperación* expuesto anteriormente, en el que se ve cómo se han incrustado los datos de las imágenes mediante las URL en el código HTML (*HyperText Markup Language*) del componente personalizado.

3.3.2. *Planificador y seguidor de trayectorias*

En el mismo script en el que el robot recibe del servidor las posiciones GPS de los puntos marcados en la interfaz (como se ha comentado previamente en la definición del componente *Localización*) se realiza una planificación de trayectorias en base a estos como puntos de destino, tomando como punto inicial la posición GPS en la que se encuentra el robot móvil. Para ello, se hace uso de la librería de Python *osmnx*, la cual utiliza Overpass Turbo para calcular los caminos a partir de

un grafo aportado por fichero y las coordenadas de origen y destino. De este modo, se obtienen las coordenadas que forman el camino entre el origen y el destino y estas se pintan en la interfaz tras publicarlas en un mensaje de tipo *foxglove_msgs/GeoJSON*. También se ofrece la posibilidad de exportar las coordenadas en un archivo *.csv*.

En el caso del seguimiento de trayectorias, actualmente se sigue una lógica utilizando un controlador PID (proporcional, integral y derivativo) que integra funciones de Lyapunov (Guldner and Utkin, 1994), la cual tiene en cuenta la posición actual del robot y la siguiente posición en el camino planificado y va publicando mensajes en el tópico del controlador de velocidad del robot móvil cada vez que se actualiza su posición. Una vez alcanzada la posición objetivo, se pasa a la siguiente, hasta que se haya completado el recorrido.

No obstante, la planificación y seguimiento de trayectorias en el estado actual de la aplicación web no contempla la posibilidad de que haya obstáculos en el camino. A futuro se busca implementar un planificador de trayectorias utilizando el algoritmo RRT (*Rapidly exploring random tree*) (Kuffner and Valle, 2000), lo que permitiría tener en cuenta estos obstáculos en tiempo real para recalcular la trayectoria y que el seguidor la tenga en cuenta.

4. Conclusiones y trabajo futuro

En este artículo se ha presentado una implementación sobre Foxglove Studio para visualizar, teleoperar y comandar un robot móvil integrando técnicas de planificación y seguimiento de trayectorias, además de procesamiento de nubes de puntos. Al tratarse de componentes desarrollados por separado, si bien se ofrece el diseño explicado en este artículo, esto da total libertad al usuario para reestructurar los paneles de la manera que crea conveniente. En cuanto a las comunicaciones entre robot e interfaz, se ha proporcionado un servidor *Rosbridge* en una IP pública. Además, en el PC que actúa como intermediario ejecutando el servidor *Rosbridge*, también se ha abierto un puerto para el servicio de ROS *Web video server* que admite conexiones HTTP, para hacer posible el obtener los datos de imágenes de los robots mediante una URL.

Con esta implementación se ha conseguido desarrollar una solución orientada a una aplicación de usuario final que suple las necesidades planteadas en nuestro caso concreto, gracias a las herramientras proporcionadas por Foxglove Studio y la integración de componentes personalizados. No obstante, para casos en los que sea necesario implementar funcionalidades más complejas o más extensas, es recomendable desarrollar la aplicación de manera nativa haciendo uso de otras tecnologías (por ejemplo, el lenguaje de programación JavaScript junto con el *framework* React, o usando Python junto con la librería *tkinter*, la cual proporciona herramientas para desarrollar aplicaciones de escritorio), a modo de evitar las limitaciones impuestas por el hecho de trabajar sobre Foxglove Studio.

A futuro, sería conveniente el realizar pruebas de funcionamiento de larga duración para pulir el comportamiento de la aplicación web ante situaciones inesperadas, como pérdida y recuperación de la conexión o subidas bruscas de latencia.

Agradecimientos

Este trabajo es parte del proyecto TED2021-130901BI00, financiado por MCIN/AEI/10.13039501100011033 y la Unión Europea "NextGenerationEU"/PRTR. También es parte del proyecto PROMETEO/2021/075 subvencionado por la Generalitat Valenciana.

Referencias

ARVC, 2024. Laboratorio de automatización, robótica y visión por computador (ARVC). https://arvc.umh.es/, accedido: 2024-05-22.

Bennett, J., 2010. OpenStreetMap. Packt Pub.

Cho, Y. C., Jeon, J. W., 2008. Remote robot control system based on DTMF of mobile phone. 2008 6th IEEE International Conference on Industrial Informatics, 1441–1446.
DOI: 10.1109/INDIN.2008.4618331

Fette, I., Melkinov, A., 2011. The websocket protocol. Tech. rep., No. rfc6455.

Foxglove Studio, 2021. Foxglove: visualizing and debugging your robotics data. https://foxglove.dev/, accedido: 2024-05-21.

Gourley, D., Totty, B., 2002. HTTP: The Definitive Guide. O'Reilly Media, Inc.

Guldner, J., Utkin, V., 1994. Stabilization of non-holonomic mobile robots using Lyapunov functions for navigation and sliding mode control. Proceedings of the IEEE Conference on Decision and Control 3, 2967–2972.
DOI: 10.1109/CDC.1994.411340

Ilijoski, B., Ackovska, N., Zorcec, T., Popeska, Z., 2022. Extending robot therapy for children with autism using mobile and web application. Sensors 22, 5965.
DOI: 10.3390/S22165965

Ishibashi, M., Iida, M., Suguri, M., Masuda, R., 2013. Remote monitoring of agricultural robot using web application. IFAC Proceedings Volumes 46, 138–142.
DOI: 10.3182/20130828-2-SF-3019.00047

Kuffner, J., Valle, S. L., 2000. RRT-connect: an efficient approach to single-query path planning. Proceedings - IEEE International Conference on Robotics and Automation 2, 995–1001.
DOI: 10.1109/ROBOT.2000.844730

Payá, L., Gil, A., Reinoso, O., Juliá, M., Riera, L., Jiménez, L., 2006. Distributed platform for the control of the WifiBot robot through internet. IFAC Proceedings Volumes 39, 59–64.
DOI: 10.3182/20060621-3-ES-2905.00012

Pimentel, V., Nickerson, B., 2012. Communicating and displaying real-time data with websocket. IEEE Internet Computing 16, 45–53.
DOI: 10.1109/MIC.2012.64

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A., 2009. ROS: an open-source robot operating system. ICRA workshop on open source software 3.

React, 2013. React. https://es.react.dev/, accedido: 2024-05-21.

Skvorc, D., Horvat, M., Srbljic, S., 2014. Performance evaluation of websocket protocol for implementation of full-duplex web streams. 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 1003–1008.
DOI: 10.1109/MIPRO.2014.6859715

Speck, A., Klaeren, H., 1999. RoboSim: Java 3D robot visualization. IECON Proceedings (Industrial Electronics Conference) 2, 821–826.
DOI: 10.1109/IECON.1999.816506