CEA Comité Español de Automática

XLV JORNADAS AUTOMÁTICA MÁLAGA 2024

UNIVERSIDADE DA CORUÑA

Jornadas de Automática

UR2A: comunicación bidireccional Android-ROS 2 para arquitecturas edge-cloud en sistemas robóticos conectados

Córdoba-Ramos, M.[a,*], Bravo-Arrabal, J.[1], Fernández-Lozano, J.J.[1], Mandow, A.[1], García-Cerezo, A.[1]

[1] *Departamento de Ingeniería de Sistemas y Automática. Instituto de Ingeniería Mecatrónica y Sistemas Ciberfísicos. Universidad de Málaga, Arquitecto Francisco Peñalosa, nº 6, 29071, Málaga, España.*

To cite this article: Córdoba-Ramos, M., Bravo-Arrabal, J., Fernández-Lozano, J.J., Mandow, A., García-Cerezo, A. 2024. UR2A: bidirectional Android-ROS 2 communication for edge-cloud architectures in networked robotic systems. Jornadas de Automática, 45. https://doi.org/10.17979/ja-cea.2024.45.10896

Resumen

Con el Internet de las Cosas Robóticas (IoRT) se hace referencia a cualquier parte de la carga útil de un robot que esté conectada a Internet. En el contexto de la robótica de campo, es cada vez más habitual utilizar *smartphones* como elementos IoRT aprovechando sus ventajas: ligereza y reducido tamaño, calidad de imagen, alta capacidad de procesamiento, diversidad de sensores, y excelente conectividad (5G/6G). Un *smartphone* moderno puede embarcarse en cualquier tipo de robot para obtener más información de su estado y de su entorno, ya que dispone de sensores internos y puede conectarse a otros externos. Se ha desarrollado una aplicación o *app open-source*, para Android, denominada *UMA-ROS2-Android* (UR2A), capaz de alojar nodos de ROS 2 con el fin de transmitir la información sensorial del *smartphone*, como su posición y orientación en el espacio, o su porcentaje de batería disponible. Además, se ha habilitado la transmisión de imágenes a distintas resoluciones, así como la capacidad de que el dispositivo reciba comandos remotamente, integrándose así en una arquitectura *edge-cloud*. La *app* UR2A ha sido validada en un caso de posicionamiento, remoto y en tiempo real, de vehículos aéreos no tripulados (UAV). La *app* está disponible en un repositorio público: `https://github.com/Robotics-Mechatronics-UMA/UMA-ROS2-Android/`.

Palabras clave:
Adquisición remota de datos sensoriales, Robots aéreos, Localización, Sistemas robóticos conectados, Robótica de campo

UR2A: bidirectional Android-ROS 2 communication for edge-cloud architectures in networked robotic systems

Abstract

Internet of Robotic Things (IoRT) refers to any part of a robot's payload connected to the Internet. In the context of field robotics, using *smartphones* as IoRT elements is increasingly common, taking advantage of their benefits: lightness and small size, image quality, high processing power, sensor diversity, and excellent connectivity (5G/6G). A modern *smartphone* can be carried by any kind of robot to obtain more information about its status and environment since it has internal sensors and can connect to external ones. A *open-source* application has been developed for Android, called *UMA-ROS2-Android* (UR2A), capable of hosting ROS 2 nodes to transmit sensory information from the *smartphone*, such as its position and orientation in space, or its percentage of available battery. In addition, the transmission of images at different resolutions has been enabled, as well as the ability for the device to receive commands remotely, thus integrating into a *edge-cloud* architecture. The UR2A *app* has been validated in a case of remote and real-time positioning of uncrewed aerial vehicles (UAV). The *app* is available at a public repository: `https://github.com/Robotics-Mechatronics-UMA/UMA-ROS2-Android/`.

Keywords: Networked robotic systems, Field robotics, Flying robots, Localization, Remote sensor data acquisition

*Autor para correspondencia: jfl@uma.es
Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)

1. Introducción

1.1. Enfoque para creación de apps en robótica

El uso de un *smartphone* moderno como elemento IoRT (Internet of Robotic Things) se justifica por sus características: reducido tamaño y ligereza, cámaras de alta resolución, sensores internos que habilitan la navegación de robots, como un módulo GNSS (*Global Navigation Satellite System*) y una unidad de medida inercial o IMU (*Inertial Measurement Unit*), gran capacidad de procesamiento, muy buena conectividad (5G/6G), alta estandarización de protocolos y conexiones con dispositivos externos, y todo a un coste relativamente bajo.

En robótica, no se entiende que la generación y distribución de servicios e información no se haga a través de ROS (*Robot Operating System)* y, en particular, en su segunda versión: ROS 2, que actualmente está soportada por las distribuciones *Iron* y *Humble*, esta última con soporte hasta 2027. Es bien sabido que los sistemas operativos Android y Ubuntu se basan en Linux, cobrando sentido el desarrollo de apps *open-source* capaces de integrar ROS.

En la bibliografía destaca el escaso desarrollo y divulgación de *apps* que hagan uso de la biblioteca *ROS2-Java* [1], que permite una comunicación directa del *smartphone* con cualquier red de ROS 2. Entre los posibles motivos:

- Existen diversas *apps open-source* que han sido desarrolladas para ROS 1 (Rottmann et al., 2020) pero que aún no han sido migradas a ROS 2, por incompatibilidad de algunos *drivers*. La última distribución de ROS 1 (Noetic) quedará obsoleta en mayo de 2025, por lo que muchos desarrolladores aún no han dado el salto a ROS 2, prefiriendo una transición suave puenteando mensajes entre ROS 1 y ROS 2 (Nguyen et al., 2022).
- La biblioteca *ROS2-Java* no está correctamente actualizada y no permite el uso de mensajes de tipo *sensor_msgs* sin que la *app* se detenga.
- No es trivial modificar el valor de la variable de entorno (*ROS_DOMAIN_ID*) que regula el acceso a los nodos en el *smartphone* y, aunque es viable hacerlo estableciendo modo de superusuario (que no sólo de desarrollador), suele evitarse este cambio por pérdida de garantía del dispositivo y/o por razones de seguridad (Elsersy et al., 2023). Por ello, es necesario establecer en los demás dispositivos o *hosts*, que alojen nodos que deban comunicarse con el *smartphone*, el mismo valor para esta variable de entorno de ROS 2 (por defecto, cero).
- En ROS 2, los mensajes pueden tener dos tipos de calidad de servicio (QoS): fiable o *reliable* (análoga al protocolo TCP) y forzado o *best effort* (análoga al protocolo UDP). Es decir, *reliable* prioriza la garantía de la entrega, sin importar una mayor lentitud, mientras que *best effort* no usa acuse de recibo (proceso más rápido), permitiendo perder mensajes para priorizar mensajes recientes (enfoque a tiempo real), lo cual es útil cuando la red no es robusta (por ejemplo, con alta latencia).

1.2. Inclusión de smartphones en robótica de campo

Es usual distribuir el procesamiento y monitorización de datos entre el escenario de aplicación (*edge*) y servidores remotos (*cloud*) con el fin de compartir los recursos disponibles entre agentes robóticos y humanos, especialmente, en el contexto de las emergencias. Los *smartphones* modernos habilitan nuevas aplicaciones en robótica. (Bravo-Arrabal et al., 2021) incluye, por primera vez, *smartphones* en una arquitectura *edge-cloud* para transmitir vídeo e incluso monitorizar la biométrica de intervinientes o agentes SAR (*Search and Rescue*), lo cual comienza a ser un recurso interesante para los equipos de bomberos, sometidos a gran estrés durante sus intervenciones (Vera-Ortega et al., 2022). En este sentido, (Toscano-Moreno et al., 2022) presenta una aplicación en ROS 1, que permite llamar a un robot móvil hasta la posición determinada por el módulo GNSS interno del *smartphone* utilizado por un agente SAR. De nuevo, en (Müller and Koltun, 2021) se presenta un vehículo eléctrico económico, de reducidas dimensiones, utilizado para portar *smartphones* Android, sacando provecho de sus sensores internos para hacer tareas de navegación.

1.3. Arquitecturas edge-cloud para ROS

En (Zhang et al., 2022) se comparan distintos tipos de compresión de vídeo en una arquitectura *cloud* para ROS 2, para analizar cómo afectan, en términos de latencia, a los *frames* transmitidos. Se destaca que el estándar H.264 incrementa la velocidad de subida de una determinada resolución, con respecto a compresiones normales. No obstante, no se han encontrado referencias sobre su sucesor, el estándar H.265 (en arquitecturas basadas en ROS), que puede ofrecer una calidad de vídeo similar a H.264 con un tamaño de archivo más pequeño o una mejor calidad de vídeo con el mismo tamaño de archivo, haciéndolo ideal para transmisiones de vídeo de ultra alta definición (UHD) y 4K. Por otro lado, en (Groshev et al., 2023) se indica que la aplicación de arquitecturas *edge* favorece la comunicación y coordinación entre robots móviles, especialmente en entornos donde la cobertura queda limitada a las redes inalámbricas locales generadas por cada robot (WLAN). Sin embargo, dependiendo del tipo de información, especialmente cuando conviven redes híbridas inalámbricas de distinto tipo con robots que deben cooperar entre sí (Bravo-Arrabal et al., 2021), pueden integrarse arquitecturas *edge* y *cloud* (Hassan et al., 2019). Por último, es importante poner el foco en la necesidad de llevar la computación desde el *cloud* hacia el *edge* y, en particular, en aplicaciones de búsqueda y rescate con robots (Militano et al., 2023).

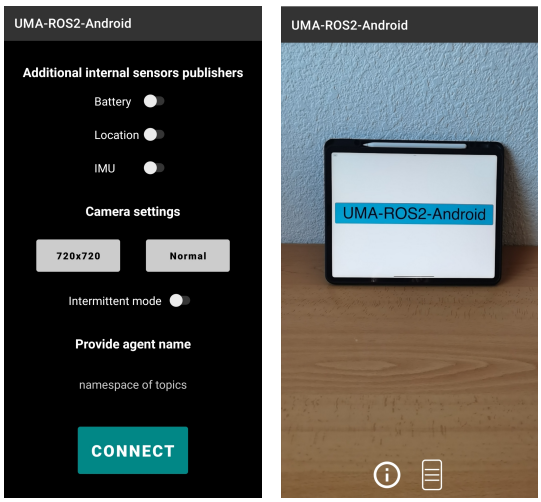
2. Desarrollo e implementación de U2RA

La *app* desarrollada, UMA-ROS-Android (UR2A)[2], es compatible con las versiones más recientes del sistema operativo Android, específicamente desde la versión 7, lanzada en 2016, hasta la versión 14, siendo esta la última, lanzada en octubre de 2023, permitiendo así ser utilizada en la mayoría de *smartphones* modernos.

[1] Repositorio de la biblioteca: `https://github.com/ros2-java/ros2_java?tab=readme-ov-file`
[2] Repositorio de la app: `https://github.com/Robotics-Mechatronics-UMA/UMA-ROS2-Android/`

La interfaz de usuario contiene los elementos interactivos imprescindibles que permiten: configurar el nodo de ROS 2 que se va a lanzar, mostrar la imagen transmitida y visualizar información relacionada con su funcionamiento.

La *app* utiliza una única *activity*, a partir de la cual se crean dos interfaces diferentes, una de configuración y otra de ejecución. La interfaz de configuración (Figura 1(a)) permite configurar el nodo que va a ser lanzado y seleccionar parámetros como el nombre del agente, la resolución de la imagen publicada, la cámara utilizada para obtener la imagen, la activación del modo intermitente, y la publicación opcional del porcentaje de batería, localización y valores de la IMU, teniendo la posibilidad de elegir la frecuencia de publicación de cada uno de ellos de forma individual en el rango $0,01 - 5$ Hz.



(a) Interfaz de configuración (b) Interfaz de ejecución

Figura 1: Interfaces de UR2A.

La interfaz de ejecución (Figura 1(b)) queda visible en pantalla tras lanzar (pulsando el botón *CONNECT*) el único nodo de ROS 2 que aloja UR2A. Este nodo tiene doble rol publicador/suscriptor, y transmite la imagen de la cámara elegida en un *topic*, permitiendo su visibilidad en tiempo real. Además, en esta misma interfaz se muestra información relacionada con los *topics* activados en la primera interfaz o con los relacionados con los comandos procedentes de nodos externos al *smartphone*, cuya visibilidad puede ser habilitada o deshabilitada por el usuario si pulsa los iconos visibles en la parte inferior de la pantalla. En la figura 2 se muestra la funcionalidad de la *app* desarrollada.

Se denomina configuración incorrecta a aquella en la que la frecuencia elegida para alguno de los publicadores opcionales no se encuentra dentro del rango especificado, o bien, el nombre del agente introducido no cumple con el siguiente formato: comenzar por una letra y contener únicamente letras y números, es decir, caracteres alfanuméricos.

Para obtener la imagen de la cámara del *smartphone* se ha utilizado la API *CameraX*, diseñada para simplificar el desarrollo de aplicaciones que utilizan la cámara. Debido a que la mayoría de los *smartphones* actuales cuentan con más de una cámara trasera (principal, gran angular y teleobjetivo), se ha añadido la opción de elegir cuál de ellas será utilizada para obtener la imagen. Además, la resolución de dicha imagen se puede elegir de entre seis posibles: 720x720, 1280x720, 1440x720, 1920x1080, 2048x1536 y 3840x2160 píxeles, conocida la última y mayor como resolución 4K. Estas pueden ser modificadas fácilmente desde el código de la *app*, pudiendo particularizarlas para cualquier *smartphone*.

La elección de ambos parámetros se realiza mediante dos botones situados en la interfaz de configuración (Figura 1(a)) y pueden ser modificados en tiempo de ejecución (tras elegir la configuración del nodo y ser lanzado) de forma remota.

La herramienta *ImageAnalisys* (o análisis de imágenes) de la mencionada API permite obtener el *frame* actual de la cámara seleccionada a la resolución elegida en formato YUV_420, a partir del cual se realiza una compresión en formato JPEG y se empaqueta en un mensaje del tipo *UInt8MultiArray* perteneciente al paquete *std_msgs*, que se publica en su respectivo *topic*. El procedimiento descrito es utilizado como alternativa al uso de paquetes habituales en la transmisión de imágenes en ROS, como es el caso de *image_transport*, por la falta de soporte en su uso con Java.

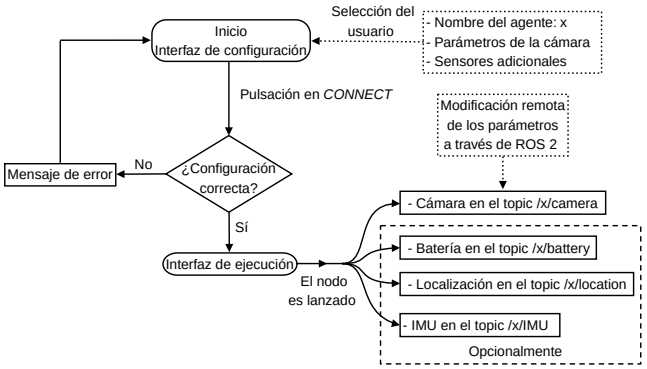


Figura 2: Diagrama de flujo de UR2A.

La elección de la cámara se realiza a través de unos índices (cada índice designa una cámara física), los cuales deben ser ajustados en el código de la *app* en el momento de su instalación. Estos índices dependen de cada terminal, siendo el único que no cambia el de la cámara trasera (por defecto 0). Así, se ha desarrollado una versión Lite de UR2A, en la que se elimina la funcionalidad de elegir la cámara empleada, utilizando por defecto la cámara trasera principal. Esta versión permite usar *smartphones* que dispongan de una única cámara trasera, o bien, que dispongan de varias cámaras pero solo una de ellas sea accesible a través de la API *CameraX*, sin tener que realizar modificaciones en el código de la *app*. Este es el caso del Pixel 7 Pro (Google, California, EEUU), empleado durante el desarrollo de la *app*.

Por otro lado, se ha probado la *app* en el P40 Pro (Huawei, Shenzhen, China), del que se han utilizado sus tres cámaras traseras (principal, gran angular y teleobjetivo), pudiendo elegir entre ellas haciendo uso de la versión estándar de UR2A.

Para obtener la localización del *smartphone* se utiliza la librería nativa de Android *android.location*. El mensaje publicado, de tipo *Float64MultiArray*, contiene la latitud, longitud y altitud (sobre elipsoide de referencia WGS84 y no sobre el nivel del mar) actuales del *smartphone*. Tanto la latitud como la longitud están expresadas en grados sexagesimales, mientras que la altitud se expresa en metros. La elección del tipo

de dato *float64* para almacenar los valores de la localización radica en su alta precisión. Aunque el módulo GNSS interno del *smartphone* no es altamente preciso y no aprovecha todos los decimales de *float64*, esta precisión es necesaria cuando se utiliza un módulo GNSS RTK (*Real Time Kinematics*) externo conectado al *smartphone*, que puede alcanzar precisión RTK-Fix (1-3 cm). Se aborda este aspecto en la sección 4.

A través de la librería nativa de Android *android.hardware* se accede a los datos registrados por la IMU, y se publican los siguientes valores empaquetados en un único mensaje del tipo *Float32MultiArray*:

- Cuaternión unitario que define la orientación relativa del sistema de referencia local al *smartphone* respecto de un sistema de referencia global en el que el eje Y apunta al norte magnético, el eje Z es perpendicular al suelo (apuntando al cielo) y el eje X se define como el producto vectorial de $Y \times Z$.
- Velocidad angular (rad/s) que experimenta el *smartphone* respecto a su sistema local de referencia.
- Aceleración lineal (m/s^2) aplicada al *smartphone*, teniendo en cuenta la gravedad terrestre. De nuevo, estos valores son expresados respecto a su sistema de referencia local.

El porcentaje de batería del *smartphone*, obtenido a través de la librería *android.os*, se publica utilizando un mensaje del tipo *UInt8*, idóneo para almacenar este valor.

La *app* presenta un modo de funcionamiento llamado *modo intermitente*. Cuando se activa, la transmisión de imágenes no se realiza de forma continua, sino que se transmite un *frame* cada dos segundos (0.5 Hz), habilitando la transmisión de imágenes a la más alta resolución. Además, reduce la carga computacional del *smartphone* y la del resto de dispositivos de la arquitectura que reciben las imágenes, al mismo tiempo que disminuye el ancho de banda de la red consumido.

A partir del código de la *app* desarrollada, es sencillo ampliar su funcionalidad, pudiendo añadir nodos que realicen tareas distintas, como por ejemplo, publicar la posición actual del *smartphone* y sólo esa en un *topic* al pulsar en un botón, o enviar un *bool* para solicitar un servicio en un *host* remoto.

3. Arquitectura edge-cloud

La *app* UR2A permite crear nodos de ROS 2 que convierten al *smartphone* en una fuente de información sensorial para otros nodos remotos en la red que precisen suscribirse. Para que puedan coexistir varios *smartphones* ejecutando la *app* en la misma red de ROS 2, se ha priorizado que cada usuario, al introducir el nombre del agente (Figura 1(a)) genere un *namespace* único para los *topics* de cada *smartphone*, así como un nombre único para el nodo lanzado desde cada *smartphone*. De esta forma, los nodos remotos pueden acceder a la información de cada agente a través de su nombre, impidiendo así sobrescribir datos indebidamente.

Los mensajes comandados remotamente a la *app* se realizan a través del *topic* /x/setcamera, mediante mensajes de tipo *std_msgs/UInt8MultiArray* que siguen el siguiente formato: vector de cuatro elementos, representado como $[M_1, M_2, M_3, M_4]$, en el que M_1 identifica la resolución objetivo mediante un número entero en el rango $0-5$, M_2 designa la cámara a emplear $(0-2)$, M_3 identifica el modo de funcionamiento $(0-1)$, y M_4 es una contraseña (actualmente, configurable en el código en el momento de la instalación) que debe ser introducida junto al mensaje para que este sea recibido correctamente por la *app*.

Como solución a limitaciones del paquete *sensor_msgs* respecto al uso de mensajes, se han creado nodos republicadores en la arquitectura. Estos nodos son lanzados desde un dispositivo remoto, y se encargan de convertir los mensajes *std_msgs* publicados desde el *smartphone* a mensajes de tipo *sensor_msgs*. La creación de estos nodos republicadores es inmediata, gracias a que los mensajes publicados por la *app* contienen los mismos campos de información que sus mensajes análogos del paquete *sensor_msgs*.

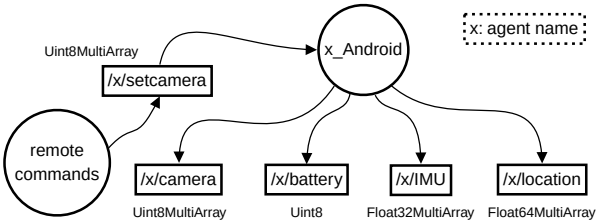


Figura 3: Comunicación de UR2A mediante ROS 2.

El uso de un *smartphone* permite crear fácilmente arquitecturas de distinto tipo, gracias a la excelente conectividad que presenta (por Wi-Fi, datos o cable). Una opción es crear una red de área local (LAN) mediante un router Wi-Fi, al que se conecten tanto el *smartphone* como el ordenador encargado de procesar su información, pero suelen existir limitaciones de alcance y ancho de banda.

Otra opción pasa por utilizar los datos móviles del *smartphone*, y utilizar una VPN (*Virtual Private Network*), que permita su conexión con un ordenador conectado a una red diferente (por cable o Wi-Fi). Esta solución permite disponer en un puesto de control de un ordenador encargado de procesar la información que está siendo obtenida a través de un *smartphone* ejecutando la *app* UR2A, sin importar donde este se encuentre, a la vez que se puede modificar el comportamiento de la aplicación (cambio remoto de parámetros) desde el puesto de control. Esta solución se basa en el concepto *Cloud Computing*, o computación en la nube.

El *smartphone* también puede ser incluido en una arquitectura *Edge Computing*. Este tipo de arquitectura se ha implementado haciendo uso de un mini-PC, que gracias a sus reducidas dimensiones puede ser utilizado como carga útil de distintas plataformas robóticas como un UAV. El mini-PC, encargado de procesar la información, se conecta a la red de datos móviles del *smartphone*, encontrándose ambos en la misma red local y obteniendo como principal beneficio la disminución de la latencia de los datos, favoreciendo así el procesamiento en tiempo real. Además, para favorecer un alto *throughput*, la compartición de datos se ha hecho mediante cableado (USB-C a USB-3.1). A su vez, se dispone de otro PC en el puesto de control conectado tanto al *smartphone* como al mini-PC a través de una VPN, encargado de modificar remotamente los parámetros de la *app*, y que además monitoriza dis-

tintos resultados publicados por el mini-PC a partir del procesamiento realizado, formando así una arquitectura *edge-cloud* que aprovecha las ventajas de ambos tipos de computación.

4. Caso de uso en robótica de campo

Para validar el correcto funcionamiento de UR2A se ha realizado una prueba en los terrenos del Laboratorio y Área de Experimentación en Nuevas Tecnologías para la Intervención en Emergencias (LAENTIEC), situados junto a la Escuela de Ingenierías Industriales de la Universidad de Málaga (UMA).

En este ejercicio, se han embarcado dos *smartphones*, Google Pixel 7 Pro y Huawei P40 Pro en un UAV (Figura 4(a)), modelo Matrice 600 (DJI Shenzhen, Guangdong, China) junto a un módulo GNSS externo, modelo ZED-F9P (U-blox, Thalwil, Suiza), conectado al primero de ellos (ver Figura 4(b)).

(a) DJI Matrice 600 (b) Dispositivos embarcados

Figura 4: Imágenes de la prueba realizada.

El módulo GNSS externo, conectado mediante USB-C al Google Pixel 7 Pro, puede recibir correcciones diferenciales de la estación *MLGA 13460M001* ubicada en Málaga, adquiriendo así precisión RTK-Fix. Para que la *app* UR2A reciba la localización del módulo GNSS externo en lugar de la del interno del *smartphone*, es necesario usar una *app* de terceros (en este caso, *SW Maps*), y configurar el dispositivo a través de los ajustes del modo desarrollador.

La realización de esta prueba permite:

- Comprobar la escalabilidad de UR2A, mediante el procesado simultáneo de la información registrada por dos *smartphones* distintos en los que se ejecuta la *app*.
- Disponer en el puesto de control de la posición del UAV con precisión RTK-Fix en tiempo real, haciendo uso del módulo GNSS externo, y comparar el error que comete el módulo interno del *smartphone* respecto a este.
- Visualizar las imágenes obtenidas por ambos *smartphones* desde el puesto de control e identificar la máxima resolución que puede ser transmitida a través de Internet (WAN) sin sufrir un aumento excesivo de la latencia, comprobando el comportamiento de la comunicación utilizando los dos tipos de calidad de servicio (QoS) de ROS 2.

Para realizar simultáneamente el procesamiento de la información de ambos *smartphones* se han empleado dos PCs conectados a un router distinto cada uno, ver Figura 5.

Se ha conseguido recibir en un puesto de control remoto al escenario de aplicación toda la información procedente de ambos *smartphones*, durante los 20 minutos que duró el vuelo del UAV: imágenes de la cámara, localización (5 Hz), valores de la IMU (5 Hz) y porcentaje de batería (0.1 Hz). La Figura 6 muestra una ortofoto generada del terreno experimental, con la trayectoria registrada por el UAV superpuesta, constituida por los puntos medidos por el módulo GNSS externo con correcciones RTK, que fueron transmitidos por el Google Pixel 7 Pro a través de su nodo *Pixel_Android*, ejecutándose en UR2A.

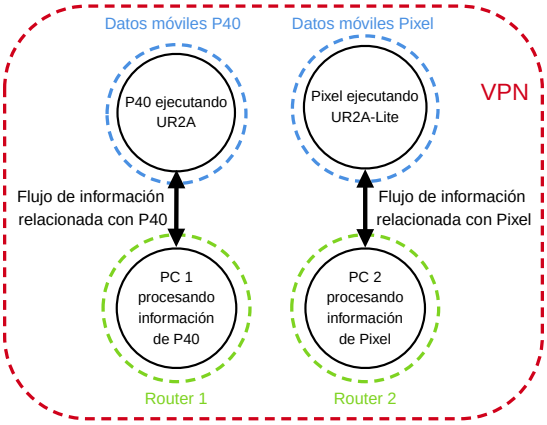


Figura 5: Esquema de comunicación.

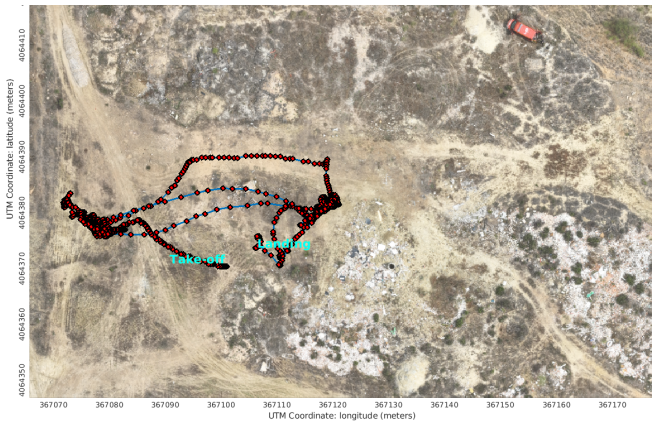


Figura 6: Trayectoria del UAV transmitida por el *smartphone* con RTK.

La evolución temporal del error cometido en la posición obtenida con el módulo GNSS interno del Huawei P40 Pro respecto de la obtenida con el módulo externo aparece representada en la Figura 7. De este se deduce que la localización del módulo GNSS interno no es muy precisa, sin embargo, esta puede ser valiosa en distintas aplicaciones como fusión de sensores junto a la IMU, gracias a la cual podría mejorar su precisión.

El modo de funcionamiento elegido para la cámara fue el modo intermitente, con el que se recibió un flujo controlado de *frames* cada dos segundos (0.5 Hz) para todas las resoluciones, a excepción de las dos mayores. Con la resolución 2048x1536 píxeles, el flujo de imágenes es controlado, pero la frecuencia disminuye a 0.4 Hz. Sin embargo, cuando se emplea la máxima resolución (3840x2160 píxeles o 4K), las imágenes dejan

de recibirse, siendo este el inconveniente encontrado al transmitir las imágenes haciendo uso de Internet (WAN).

Durante un período de tiempo se probó el modo continuo a la resolución más baja de ambos dispositivos, 800x600 para el Pixel y 720x720 para el Huawei, obteniendo en ambos una frecuencia de transmisión en el entorno de los 15 Hz.

El QoS se ha establecido como *best effort* en el publicador de imágenes, ya que debe priorizarse la llegada más actualizada de este tipo de mensajes, sin importar la pérdida de mensajes antiguos. Para el resto de información se emplea *reliable*.

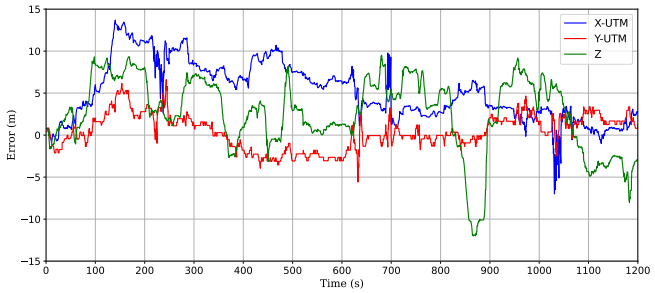


Figura 7: Error de posición del módulo GNSS interno con respecto al externo con precisión RTK-Fix, tomado como *ground truth*.

Posteriormente se han realizado varias pruebas, tanto haciendo uso de un mini-PC abordo del UAV y una arquitectura *edge*, como conectando el *smartphone* a la misma red LAN que un PC, es decir, transmitiendo las imágenes sin utilizar una red WAN. Así, se han conseguido recibir imágenes en 4K haciendo uso del modo intermitente, a 0.5 Hz. Con el modo fluido se han conseguido recibir imágenes a 30 fps, utilizando la resolución más baja.

5. Conclusiones

Se ha desarrollado una *app* para Android, denominada *UMA-ROS2-Android* (UR2A), compatible con *smartphones* modernos que sirvan como carga útil de plataformas robóticas. UR2A implementa ROS 2 y es capaz de transmitir información sobre sus sensores internos, como orientación (IMU) y localización (GNSS), incluso a través de módulos externos. Además, UR2A puede comunicarse bidireccionalmente con otros nodos de una arquitectura *Edge-Cloud*, pudiendo recibir remotamente parámetros para seleccionar una de las cámaras disponibles en el *smartphone*, así como la resolución con la que se desea monitorizar el entorno del robot. Además, la *app* muestra información en pantalla al usuario.

Como líneas futuras, se plantea las siguientes:

- Actualizar la librería *ROS2-Java* para incluir los mensajes *sensor_msgs*.
- Incluir servicios en la *app* en vez de usar una (*activity*) de Android, para poder ejecutarla en segundo plano. Esto ahorrará batería, al poder bloquear el dispositivo.
- Posibilitar la elección de otros valores para el ROS_DOMAIN_ID, permitiendo la división de nodos por subredes de ROS 2.
- Integrar UR2A en aplicaciones de visión y localización en robótica de campo. (Park et al., 2024) propone el uso de marcadores de referencia para el aterrizaje autónomo de un UAV, empleando ROS 1 y una cámara conectada al PC mediante conexión USB. En (Springer et al., 2024) utilizan conjuntamente tres cámaras para conseguir un aterrizaje preciso del UAV: zoom, gran angular e infrarrojos. El uso de UR2A permitiría disponer de tres cámaras móviles de resolución variable, que se pueden manipular remotamente, al mismo tiempo que proporciona información sensorial sobre el UAV y todo ello en el reducido espacio que ocupa un *smartphone*.

Agradecimientos

Este trabajo ha recibido financiación del Ministerio de Ciencia, Innovación y Universidades, Gobierno de España (proyecto PID2021-122944OB-I00).

Referencias

Bravo-Arrabal, J., Toscano-Moreno, M., Fernandez-Lozano, J. J., Mandow, A., Gomez-Ruiz, J. A., García-Cerezo, A., 2021. The Internet of Cooperative Agents architecture (X-IoCA) for robots, hybrid sensor networks, and MEC centers in complex environments: A Search and Rescue case study. Sensors 21 (23), 7843.
DOI: 10.3390/s21237843

Elsersy, W. F., Anuar, N. B., Razak, M. F. A., 2023. Rootector: robust android rooting detection framework using machine learning algorithms. Arabian Journal for Science and Engineering 48 (2), 1771–1791.

Groshev, M., Baldoni, G., Cominardi, L., de la Oliva, A., Gazda, R., 2023. Edge robotics: Are we ready? an experimental evaluation of current vision and future directions. Digital Communications and Networks 9 (1), 166–174.

Hassan, N., Yau, K. L. A., Wu, C., 2019. Edge computing in 5G: A review. IEEE Access 7, 127276–127289.

Militano, L., Arteaga, A., Toffetti, G., Mitton, N., 1 2023. The Cloud-to-Edge-to-IoT Continuum as an Enabler for Search and Rescue Operations. Future Internet 2023, Vol. 15, Page 55 15, 55.
DOI: 10.3390/FI15020055

Müller, M., Koltun, V., 2021. OpenBot: Turning Smartphones into Robots. Proceedings - IEEE International Conference on Robotics and Automation 2021-May, 7205–7211.

Nguyen, Q.-D., Dhouib, S., Huang, Y., Bellot, P., 2022. An Approach to Bridge ROS 1 and ROS 2 Devices into an OPC UA-based Testbed for Industry 4.0. 2022 IEEE 1st Industrial Electronics Society Annual On-Line Conference (ONCON), 1–6.

Park, Y., Park, C., Song, W., Lee, C., Kwon, J., Park, J., Noh, G., Lee, D., 1 2024. Fiducial Marker-Based Autonomous Landing Using Image Filter and Kalman Filter. International Journal of Aeronautical and Space Sciences 25, 190–199.

Rottmann, N., Studt, N., Ernst, F., Rueckert, E., 2020. Ros-mobile: An android application for the robot operating system. arXiv preprint arXiv:2011.02781.

Springer, J., pór Guomundsson, G., Kyas, M., 3 2024. A Precision Drone Landing System using Visual and IR Fiducial Markers and a Multi-Payload Camera. arXiv preprint arXiv:2403.03806.

Toscano-Moreno, M., Bravo-Arrabal, J., Sánchez-Montero, M., Barba, J. S., Vázquez-Martín, R., Fernández-Lozano, J. J., Mandow, A., Garcia-Cerezo, A., 2022. Integrating ros and android for rescuers in a cloud robotics architecture: application to a casualty evacuation exercise. In: 2022 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR). IEEE, pp. 270–276.
DOI: 10.1109/SSRR56537.2022.10018629

Vera-Ortega, P., Vázquez-Martín, R., Fernández-Lozano, J. J., García-Cerezo, A., Mandow, A., 2022. Enabling remote responder bio-signal monitoring in a cooperative human–robot architecture for search and rescue. Sensors 23 (1), 49.

Zhang, J., Keramat, F., Yu, X., Hernández, D. M., Queralta, J. P., Westerlund, T., 2022. Distributed robotic systems in the edge-cloud continuum with ros 2: A review on novel architectures and technology readiness. In: 2022 Seventh International Conference on Fog and Mobile Edge Computing (FMEC). IEEE, pp. 1–8.