Máster en Robótica y Automatización

Ingeniería de Sistemas y Automática

Sistemas Operativos de Robots

Curso 2022/2023

Trabajo final

Interacción

Autores: Daniel Gil Morales y Jaime Godoy Calvo

Profesor: Fernando Alonso Martín

19 de diciembre de 2022

Índice

1. Introducción

Este trabajo[1] se ha desarrollado con el objetivo de crear un sistema ROS completo que sea capaz de recopilar información sobre un usuario y mostrarla en pantalla. Se ha diseñado de manera modular, siguiendo la norma habitual en los sistemas ROS, por lo que consta de varios nodos que se comunican entre sí a través de temas y servicios.

El sistema está dividido en tres niveles jerárquicos. El primero se encarga de recoger los datos, donde el usuario debe introducir la información necesaria a través del teclado. Para ello, se han implementado tres nodos distintos, cada uno de los cuales solicita un tipo de información específica. El segundo nivel es un nodo único que agrega toda la información obtenida en el primer nivel. De esta manera, se consigue modularidad: se pueden añadir o eliminar tipos de información y cambiar la forma de introducir los datos (por ejemplo, leyendo un archivo) sin afectar al resto de nodos del sistema. Por último, el tercer nivel es el encargado de recopilar la información agregada y mostrarla en pantalla.

Además de esta estructura principal, se pueden añadir procesos adicionales al sistema según sea necesario. Por ejemplo, se puede utilizar un servicio para llamar a un nodo que procese los datos de alguna manera y devuelva la información. También se puede añadir un nodo que muestre periódicamente algún tipo de información o, por ejemplo, intentar ejecutar nodos de forma remota utilizando una estructura *master-slave*. Además, la información mostrada en pantalla se puede leer en voz alta mediante un sintetizador de texto a voz.

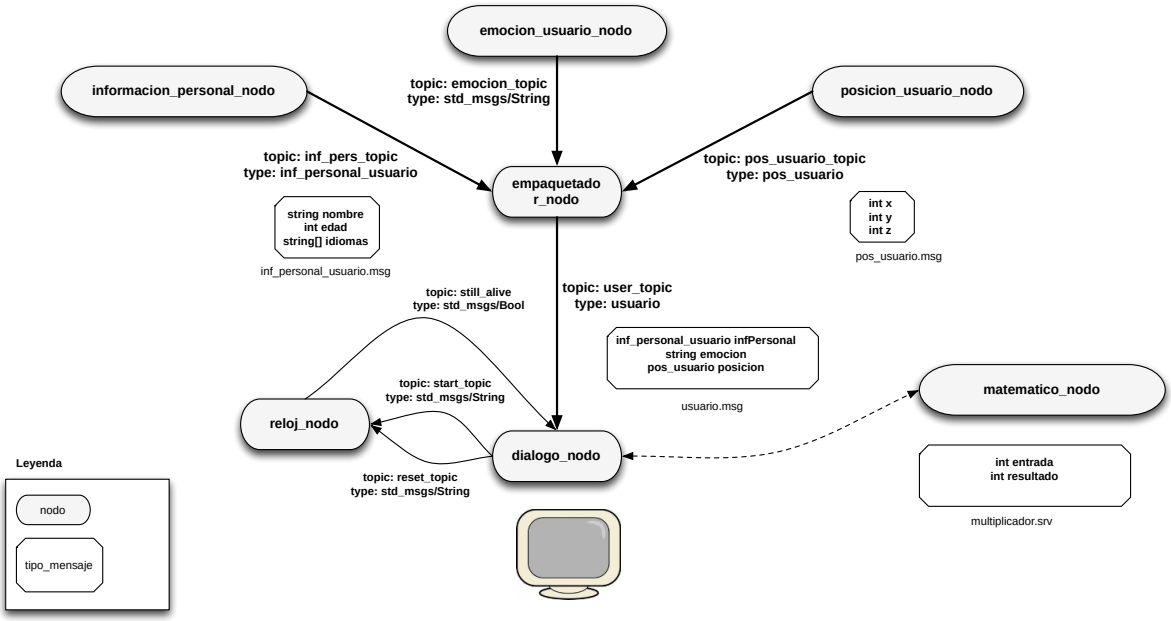

Figura 1: Esquema de comunicaciones.

[1] https://github.com/Jagoca98/ros_ws

2. Nodos

En este sección se describe en detalle cada uno de los nodos que forman parte de la estructura en tres niveles mencionada anteriormente. Se explica el papel que desempeñan en la estructura, el tipo de nodo al que pertenecen, los *topics* de comunicación que utilizan y los mensajes que intercambian.

2.1. Información personal

Este nodo tiene como función recopilar información personal del usuario a través del teclado. Solicitará el nombre y la edad del usuario, el número de idiomas que desea introducir y cuáles son dichos idiomas. Es un nodo *publisher*, ya que una vez recopilada la información solicitada por teclado, la publicará en el *topic /inf_pers_topic*.

Para ello, primero es necesario crear el tipo de mensaje que se publicará. En el paquete *interacción*, se crea una carpeta llamada *msg*, que contendrá la definición de los mensajes. Para este nodo, se crea un tipo de mensaje llamado *inf_personal_usuario.msg*, que incluye tres variables: una cadena para almacenar el nombre, un entero para almacenar la edad y un array de cadenas para almacenar todos los idiomas del usuario.

Una vez definida la estructura del mensaje, se puede publicar la información recopilada por teclado mediante el *topic /inf_pers_topic*, que publicará un mensaje de tipo *inf_personal_usuario.msg*. Tanto la solicitud de información como la publicación del mensaje se llevan a cabo en un bucle, lo que permite introducir información de varios usuarios.

2.2. Posición

Este nodo se encarga de solicitar por teclado las coordenadas del usuario en el espacio (X, Y, Z). Es un nodo *publisher*, ya que una vez recopilada la posición del usuario, la publicará en el *topic /pos_usuario_topic*.

Para ello, primero se debe crear el tipo de mensaje que se publicará. En la carpeta *msg*, se crea un nuevo tipo de mensaje llamado *pos_usuario.msg*, que incluye tres variables, que son tres enteros, uno para cada una de las coordenadas del usuario. Una vez definida la estructura del mensaje, se puede publicar la información recopilada por teclado mediante el *topic pos_usuario_topic*, que publicará un mensaje de tipo *pos_usuario.msg*. Tanto la solicitud de las coordenadas del usuario como la publicación del mensaje se llevan a cabo en un bucle, lo que permite introducir la posición espacial de varios usuarios.

2.3. Emoción

Este nodo solicita al usuario que ingrese una emoción a través del teclado. Es un nodo *publisher*, ya que una vez que recibe la emoción del usuario, la publica en el *topic /emocion_topic*. No es necesario crear un nuevo tipo de mensaje, ya que podemos utilizar un tipo de mensaje predefinido por ROS. Este mensaje será de tipo *std_msgs::String*, que permite publicar tópicos que contengan una única variable de cadena.

Dado que este nodo solo necesita publicar una variable de cadena, este tipo de mensaje es suficiente para publicar la emoción del usuario. Por lo tanto, podemos publicar la información

recibida a través del teclado a través del *topic /emocion_topic*, que publicará un mensaje de tipo *std_msgs::String*. Tanto la solicitud de la emoción como la publicación del mensaje se encuentran en un bucle, lo que permite que se puedan ingresar las emociones de tantos usuarios como se desee.

2.4. Empaquetador

El propósito de este nodo es recibir los mensajes enviados en los *topics* relacionados con los nodos anteriores y reunir la información recibida para luego enviar un único mensaje con todos los datos. Por lo tanto, este nodo es un *subscriber*, ya que debe recibir los datos de los demás nodos y, al mismo tiempo, es un *publisher*, ya que, al recibir los datos de los tres nodos, los publica en el *topic /user_topic*.

Para comenzar, debemos crear el tipo de mensaje utilizado en la publicación. En concreto, creamos un tipo de mensaje llamado *usuario.msg* que contendrá tres variables referidas a los tres tipos de mensajes que recibe de los nodos del primer nivel: una variable de tipo *inf_personal_usuario.msg* para guardar el mensaje que contiene la información del usuario, una cadena para guardar la emoción del usuario y, finalmente, una variable de tipo *pos_usuario.msg* para guardar la posición enviada por su nodo correspondiente.

El funcionamiento de este nodo se basa en tres funciones de respuesta distintas asignadas a cada tópico, que permiten guardar la información de los mensajes en los campos de una variable de tipo usuario, además de utilizar una variable booleana para confirmar que se ha recibido el mensaje. Cuando las tres variables sean verdaderas, el nodo publicará el mensaje creado y esperará a que los tres campos reciban nueva información.

La comprobación de las variables booleanas y la publicación se realizan dentro de un bucle para que el nodo pueda comprobar la condición en cada iteración y pueda repetir el proceso.

2.5. Diálogo

Finalmente, este nodo completa el tercer nivel de la aplicación y recibe el mensaje del nodo empaquetador, que contiene toda la información del usuario, con el fin de mostrar todos los datos por pantalla. Por lo tanto, este nodo es un suscriptor y muestra la información por pantalla.

Además, este nodo implementa comunicaciones con dos nodos opcionales: el del servicio matemático y el del temporizador. Por lo tanto, actúa como cliente del servicio y se suscribe al tópico del reloj. En ambos casos, el objetivo del nodo sigue siendo recibir información y mostrarla por pantalla.

La principal función de este nodo es la respuesta a los mensajes recibidos a través del *topic /usuario_topic*, es decir, debemos esperar a que el nodo empaquetador termine su proceso. Una vez recibido, la función de respuesta mostrará por pantalla cada uno de los campos del mensaje, al mismo tiempo que el sintetizador de voz realiza su función, como se explicará más adelante.

Además de esta función, es necesario establecer un bucle de trabajo en la función principal que nos permita manejar las funcionalidades con los dos nodos opcionales. En primer lugar, para controlar las llamadas al servicio, debemos comprobar que hemos recibido previamente un mensaje del usuario. En caso afirmativo, se realizará la llamada al servicio y se mostrará el resultado obtenido por pantalla.

Por otra parte, en lo que respecta al temporizador, nuevamente debemos esperar a que se envíe un mensaje a este nodo. Sin embargo, debemos diferenciar entre el primer mensaje recibido y los siguientes, ya que en cada caso debemos publicar en uno de los tópicos asociados al temporizador. En concreto, si es el primer mensaje, publicaremos un mensaje de inicio del tipo *std_msgs::String* en el *topic /start_topic* y en los demás casos se publicará un mensaje de reinicio del mismo tipo en el topic */reset_topic*.

Por lo tanto, el bucle de la función principal debe comprobar principalmente si se ha recibido un mensaje a través de una variable booleana y, en caso afirmativo, se procederá a las llamadas mencionadas anteriormente.

3. Servicio

Una vez comprobado que la estructura central del sistema funciona correctamente, se implementa un nuevo canal de comunicación basado en un servicio en lugar de en tópicos. La principal diferencia entre un servicio y un tópico es que el servicio es un canal de comunicación síncrono mientras que el tópico es asíncrono. Por lo tanto, hay un nodo que actúa como cliente, que cuando necesita la información llama al nodo que actúa como servidor, que es el que envía la información solicitada por el cliente.

En este caso, el servicio consiste en obtener el doble de la edad del usuario. Para ello, se implementa un nuevo nodo llamado matemático que actúa como servidor. En este nodo se implementa la función que recibe como parámetro de entrada la edad del usuario y devuelve el doble de esta. Al igual que con los tópicos, es necesario especificar el tipo de mensaje que se está transfiriendo a través del servicio. En este caso, se define un archivo llamado *multiplicador.srv* dentro del paquete, cuya entrada es un dato de tipo *int16* y devuelve otro dato de tipo *int16*.

Finalmente, una vez implementado el nodo que actúa como servidor del servicio, se implementa el cliente del mismo dentro del nodo de diálogo mencionado anteriormente. En este nodo, se debe invocar al servicio con el parámetro de entrada necesario y recoger el valor de retorno producido. En ROS, al llamar a un servicio, la ejecución del programa del cliente se queda en espera hasta que se recibe el valor de retorno. Si esto fuera incompatible con el flujo del programa del cliente, sería necesario implementar una comunicación basada en acciones, pero en este caso no es necesario.

4. Timer

El segundo de los nodos auxiliares implementados será el encargado de establecer un timer independiente al nodo de diálogo. Para llevar a cabo su tarea, este nodo reloj debe suscribirse a los tópicos */start_topic* y */reset_topic*, que se comunicarán con el nodo de diálogo. A su vez, este nodo publicará su estado de activación en el topic */still_alive*, de manera que el nodo de diálogo se asegure de su correcto funcionamiento.

Su modo de proceder dependerá de las señales enviadas por el nodo de diálogo, ya que este nodo iniciará el temporizador cuando el nodo de diálogo se lo indique, es decir, después de que este reciba información de un nuevo usuario por primera vez. Por otro lado, cuando se recibe otro usuario en la misma ejecución, la señal enviada será para reiniciar el temporizador. Además, este nodo debe publicar cada minuto que sigue en funcionamiento, de manera que el nodo de diálogo pueda indicarlo por pantalla.

En cuanto a su estructura, el nodo debe tener dos funciones de respuesta asociadas a cada tópico para iniciar el temporizador o reiniciarlo. Además, dentro de la función principal, se debe establecer, en primer lugar, la configuración de los parámetros relacionados con el temporizador. Luego, se deben llevar a cabo las siguientes acciones dentro de un bucle de funcionamiento: comprobar si el temporizador está activo y, por otro lado, si ha transcurrido un minuto desde que se recibió el mensaje de inicio del temporizador.

5. Síntesis de voz

La integración de esta funcionalidad extra se realizará dentro del nodo diálogo, dotándolo de la capacidad de comentar el mensaje final obtenido por todo el proceso explicado. Para ello se utiliza el paquete espeak, que incluye diferentes herramientas.

Se crea el comando de voz de tipo *string* con los valores que se publican por la terminal del nodo diálogo, posteriormente, se mandará al sistema el comando creado.

Por lo tanto, el funcionamiento de esta aplicación se basará en distintas llamadas a la función explicada. Estas deben hacerse utilizando como parámetro un tipo *string* que contenga lo que se quiera expresar, por consiguiente, es posible declarar la llamada de las siguientes opciones:

- El mensaje es declarado directamente en la llamada.
- Se manda uno de los campos de los mensajes recibidos que compartan el tipo *string*.
- Se transforma un dato de tipo *int* a *string*, utilizando la función *std::to_string()*.

Con estas tres posibilidades podemos hacer que el sistema de voz pueda ir comentando los resultados que salgan por pantalla progresivamente y de manera clara y natural. Por lo tanto, este tipo de aplicación permite aumentar las utilidades del programa creado, puesto que es capaz de transmitir la información introducida de manera hablada o escrita.

6. Archivo *launch*

Un *launch file* es un archivo de texto que nos permite iniciar varios nodos de un paquete de manera simultánea. Especifica qué nodos deben iniciarse, con qué parámetros deben iniciarse y bajo qué condiciones.

Para este proyecto, se ha creado un lanzador que inicia los nodos de información personal, posición del usuario, emoción del usuario, empaquetador y diálogo. Además, iniciará el nodo matemático, utilizado para el servicio y el nodo del reloj. En este archivo, debemos indicar el paquete que los contiene, que en este caso es interacción, los nombres de cada uno de los nodos y cómo queremos mostrarlos, que en este caso utilizamos *gnome-terminal –command*.

```
<launch>
    <node pkg="interaccion" type="informacion_personal_node" name="informacion_personal_nodo"
        output="screen" launch-prefix="gnome-terminal --command"/>
    <node pkg="interaccion" type="emocion_usuario_node" name="emocion_usuario_nodo" output="
        screen" launch-prefix="gnome-terminal --command"/>
    <node pkg="interaccion" type="posicion_usuario_node" name="posicion_usuario_nodo" output="
        screen" launch-prefix="gnome-terminal --command"/>
    <node pkg="interaccion" type="empaquetador_node" name="empaquetador_nodo"/>
    <node pkg="interaccion" type="dialogo_node" name="dialogo_nodo" output="screen" launch-
        prefix="gnome-terminal --command"/>
    <node pkg="interaccion" type="multiplica_server" name="multiplica_server"/>
    <node pkg="interaccion" type="reloj_node" name="reloj_nodo" output="screen" launch-prefix=
        "gnome-terminal --command"/>
</launch>
```

Launch 1: Archivo *interaccion.lauch*

7. Bag

Para grabar y reproducir información de los *topics* en ROS, se utiliza *rosbag*. Primero, se crea una carpeta llamada *bagfiles* en el paquete interacción, donde se almacenarán los archivos *bag*. Para grabar la información, se deben lanzar los nodos y publicar la información a través de los *topics* deseados. A continuación, se inicia la grabación con el comando *rosbag record -a* y se detiene una vez se ha publicado toda la información deseada. Para reproducir la información almacenada, se lanzan los nodos que utilizan dichos *topics* y se ejecuta el comando *rosbag play <nombre_bag>* en un nuevo terminal.

8. Ejecución distribuida

Una prueba más que se realizará a la aplicación desarrollada, se basará en la utilización de ROS de forma distribuida, es decir, comunicando diversos ordenadores a través de la estructura de nodos creada. Para ello, un requisito indispensable es que los equipos a utilizar deben compartir la misma red local para poder conectarse entre ellos. Asimismo, uno de los ordenadores debe establecerse como el *master*, en el que se ejecute el *roscore*, mientras, los demás equipos deben establecer una variable de entorno que indique la dirección *IP* del *master*:

```
export ROS_MASTER_URI=<direccion master>:11311
```

Por lo tanto, una forma de comprobar el correcto funcionamiento de esta idea es que el *master* lance los nodos principales para la recepción de datos y comprobación de los mismos, es decir, el nodo empaquetador y el nodo diálogo. Mientras tanto, los ordenadores conectados deben lanzar los nodos del primer nivel asociados a la recogida de datos del usuario, de manera que estos datos sean recibidos por el ordenador principal.

9. Conclusión

Se ha logrado completar la implementación de una aplicación ROS mediante la separación de los nodos en tres niveles y la conexión entre ellos a través de *topics*. Se han creado mensajes personalizados para enviar la información necesaria y se han establecido nodos como publicadores y suscriptores en un *topic* específico para lograr la comunicación.

Además se ha implementado un servicio ROS a través de un nodo servidor y cliente, y se han agregado elementos adicionales como un Timer y un sintetizador de voz para mejorar las funcionalidades de la aplicación.