UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA ELECTRÓNICA DE COMUNICACIONES

TRABAJO FIN DE GRADO

DISEÑO DE MÓDULOS SOFTWARE PARA UN SISTEMA AVANZADO ROBÓTICO DE ASISTENCIA A LA MOVILIDAD BASADO EN ENTORNO DE DESARROLLO ROBÓTICO ROS

AUTOR: JAVIER LEÓN ALMAZÁN

TUTORA: MARTA MARRÓN ROMERA

TRIBUNAL:

PRESIDENTE: JUAN CARLOS GARCÍA GARCÍA

VOCAL 1º: ANA JIMÉNEZ MARTÍN

VOCAL 2º: MARTA MARRÓN ROMERA

FECHA: .....

ÍNDICE

Índice de figuras

Índice de tablas

Resumen

Los grandes avances en robótica móvil han permitido el desarrollo de soluciones complejas para personas con discapacidad. Las sillas de ruedas inteligentes son un ejemplo de este tipo de soluciones en las que aún queda por investigar.

Continuando los trabajos del Departamento de Electrónica sobre una silla de ruedas y aprovechando la tecnología ROS ("Robotic Operating System"), en este Trabajo Fin de Grado se plantea incorporar una interfaz gráfica con la biblioteca GTK para *Python* que permite tener control de los nodos de bajo nivel comunicados a través del bus CAN con el alto nivel.

Palabras claves: ROS, *topic*, *Python*, GTK.

Abstract

The great advances achieved on mobile robotics allowed the development of complex solutions for disabled people. Intelligent wheelchairs are an example of this kind of solutions, which must still be investigated in depth though.

By continuing previous researches of the Electronics Department regarding wheelchairs, and by taking advantage of ROS technology (Robotic Operating System), the inclusion of a graphic interface with the GTK library ("GIMP Tool Kit") for Python has been raised on this end-of-degree project. This interface allows taking control of low-level control nodes, communicated through the CAN bus to the high-level laptop.

Resumen extendido

En este Trabajo Fin de Grado (TFG), se ha trabajado en el alto nivel del Sistema Avanzado Robótico de Asistencia a la movilidad (SARA), diseñado en el Departamento de Electrónica durante los últimos años.

En concreto se ha desarrollado un joystick digital dentro de una interfaz gráfica que consigue mover una silla de ruedas y observar al mismo tiempo la distancia de los obstáculos que se encuentran a su paso. Esta interfaz gráfica se ha creado con la librería GTK (GIMP Tool Kit) del lenguaje de programación *Python* dentro del *Framework de Desarrollo de Robótica* (RFS) llamado ROS (*Robotic Operating System*) incorporado anteriormente por [1].

La Figura 1 muestra el diagrama general de SARA, incluido en el modelo de comunicación de ROS, que hace posible usar las múltiples funciones de navegación autónoma sobre SARA.

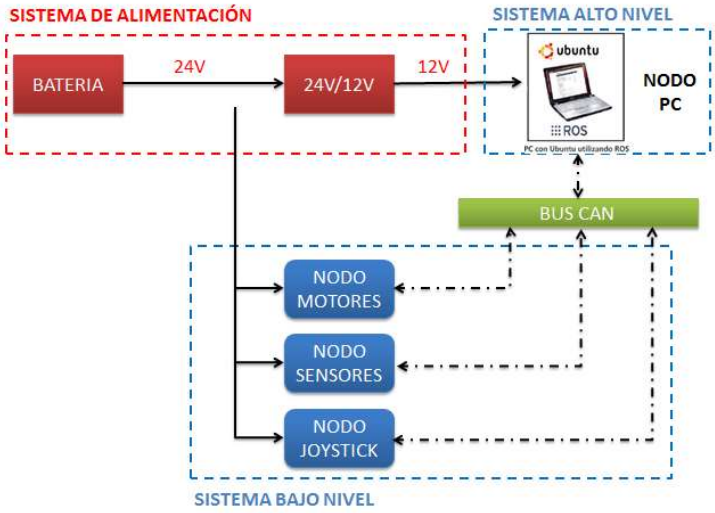


Figura 1 - Esquema general de SARA

Hardware

SARA dispone de varios módulos (motores, joystick y sensores) que se comunican entre sí mediante un bus CAN (ver figura 1). Los distintos nodos envían mensajes por este bus, cada uno con su propio identificador. La silla dispone de un adaptador USB-CAN que permite comunicar a nivel de aplicación los módulos con un nodo PC gracias a un nodo de comunicaciones implementado en ROS. Este nodo implementado en ROS no está completo, ya que no existe una comunicación o control sobre todos los módulos de la silla de ruedas. Por ejemplo, el nodo de los motores aún no se ha implementado completamente en ROS y sería necesario para poder desplazar a SARA por un entorno con el alto nivel (PC) sin necesidad de usar el joystick analógico.

Como se ha mencionado antes la silla de ruedas dispone de varios nodos, comunicados entre sí mediante el bus CAN. Los nodos que actualmente tiene la silla son:

- Nodo joystick: Formado por un joystick analógico, un display, un potenciómetro, varios botones y un teclado.
- Nodo motores: Formado por los motores, sus encoders y la tarjeta de potencia.
- Nodo sensores: Formado por sensores de ultrasonidos colocados alrededor de la silla y un acelerómetro para detectar colisiones.
- Nodo PC: Formado por un ordenador con pantalla táctil con el que podemos ver toda la información del resto controlar una parte de los nodos. Este TFG se centra en este nodo desarrollando e implantando aplicaciones (sobre ROS) para disponer de un mayor control sobre el resto de nodos.

A nivel hardware, para el desarrollo de este TFG se usará todo el hardware de bajo nivel con que cuenta actualmente SARA, procedente de anteriores trabajos.

El punto de partida hardware es, por tanto, una silla de ruedas eléctrica a la que le han sido añadidos distintos elementos y circuitería electrónica. Entre otros elementos cabe destacar dos encoders incrementales acoplados a los ejes de los motores, una tarjeta de

potencia para controlar cada uno de estos dos últimos, sensores de proximidad, acelerómetros y un ordenador con pantalla táctil en el que se ejecuta el nodo PC.

Software de integración física

La aplicación desarrollada para la comunicación del PC con los nodos de la silla de ruedas está implementada en ROS por [1] con el lenguaje de programación *Python*. El objetivo de este TFG será seguir utilizando ROS y mejorar la aplicación para tener control sobre todos los nodos de SARA (ya que actualmente no se tiene) y conseguir que esta se desplace por un entorno real mediante la implementación de un joystick digital que maneja el usuario y que se comunica con el nodo motores del bajo nivel.

Para entender y realizar esta parte del trabajo, es necesario conocer los conceptos fundamentales de ROS, así como las herramientas que incluye y que se describirán más adelante.

Interfaz gráfica

Aprovechando que se está trabajando con *Python* se desarrollará una GUI (G*raphical User Interface*) con la biblioteca gráfica GTK de este lenguaje. En esta interfaz se implementará un joystick digital con el que poder mover los motores de la silla de ruedas. Además, se añadirá también a la interfaz todos los sensores de ultrasonidos de tal forma que el usuario cuando se esté desplazando por un entorno con el joystick digital pueda tener información de la distancia a la que se encuentran los obstáculos que se encuentran en su camino.

Para comprender todo esto, se describirá en el TFG someramente el lenguaje de programación *Python* y la biblioteca grafica GTK para *Python* mencionando cuales son los conceptos y funciones básicas.

Resultados

El paquete que se ha implementado en ROS para la comunicación de los nodos de la silla de ruedas con el PC se llama “canusb” y contiene dos aplicaciones:

La primera aplicación se llama “canusb.py” y se encarga de leer todas las tramas CAN que se reciben de SARA

La segunda aplicación se llama “test.py” y sirve para enviar tramas CAN en este caso al nodo motores.

En la Figura 2 se muestra el paquete ejecutado con los dos scripts en funcionamiento.

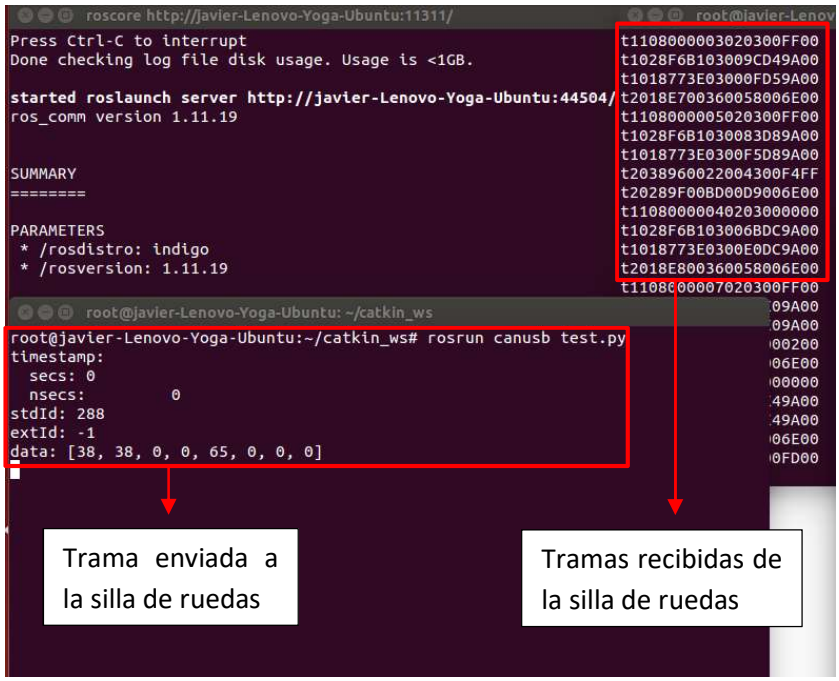


Figura 2 - Resultados del software de integración física

Cuando se le indica a la silla de ruedas que se quiere pasar al modo PC, esta hace una comprobación para verificar que existe una comunicación entre el PC y el bajo nivel de SARA. Si existe la comunicación aparece un mensaje de “Sincronizado” en la pantalla del joystick analógico como se muestra en la Figura 3.

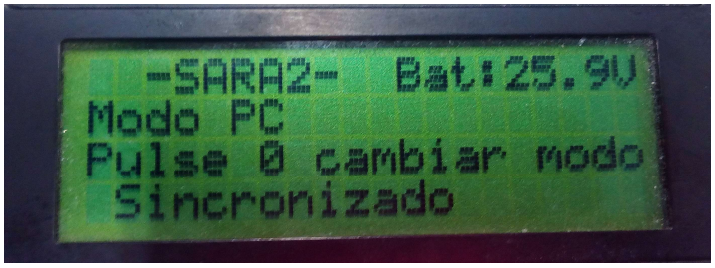


Figura 3 - Estado que muestra la silla de ruedas cuando entramos en modo PC

Tras verificar que la integración física funciona, se añade al archivo “test.py” del paquete “canusb” el código de la interfaz gráfica que contiene el joystick digital y se comprueba su funcionamiento. En la Figura 4 se puede ver la GUI funcionando con los valores que se están recibiendo en tiempo real de los sensores de ultrasonidos.

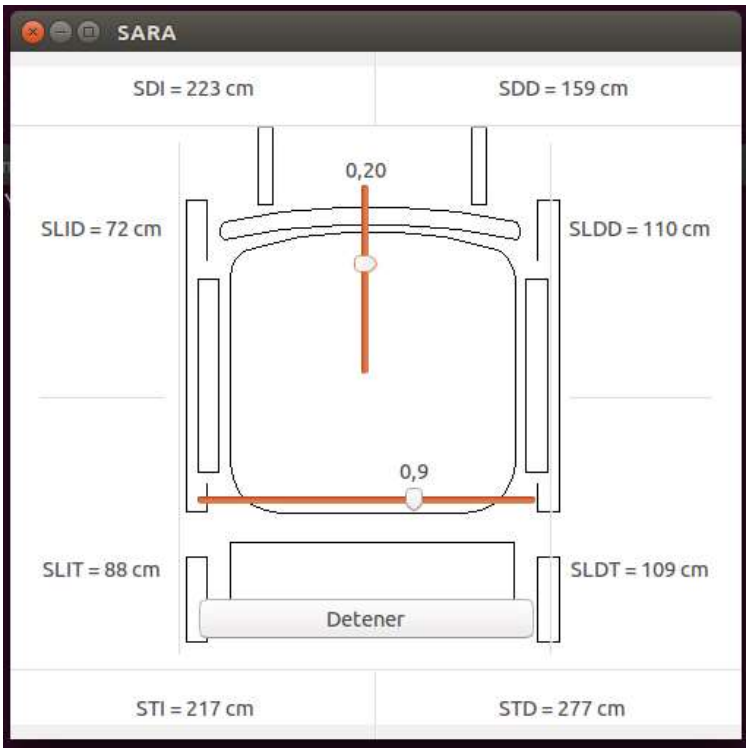


Figura 4 - Ejecución de la interfaz grafica

Capítulo 1

Introducción

Este Trabajo Fin de Grado (TFG) permite avanzar en el proyecto "Diseño y desarrollo de módulos hardware / software para un sistema avanzado robótico de asistencia a la movilidad (SARA)" de David Pinedo Ávila, con el objetivo global de diseñar una silla de ruedas autónoma modulable. El proyecto de David fue a su vez la continuación del proyecto "Guiado de una silla de ruedas mediante joystick y soplido por bus CAN I y II" de Juan Bellón Álvarez y Jose Basterrechea García. Con todos estos proyectos se ha conseguido crear una silla de ruedas avanzada (*Advanced Wheelchairs*, AW) capaz de ser controlada mediante un joystick o un sensor de soplo y comunicarse con un PC mediante un bus CAN.

Los grandes avances en la tecnología y más concretamente en la robótica móvil han permitido la creación de herramientas, aplicaciones y dispositivos que facilitan el desarrollo de soluciones complejas. Este TFG se enfoca en seguir desarrollando el software de alto nivel sobre las herramientas ya utilizadas anteriormente en SARA implementando la comunicación de todos los nodos de la silla de ruedas con el PC y desarrollando un joystick digital dentro de una interfaz gráfica que permita al usuario dirigir la silla de ruedas. En cuanto a las posibles soluciones hardware no se realiza ningún avance, dejando todos los dispositivos de los que ya dispone SARA como están.

La plataforma software (framework) de robótica ROS (Robotic Operating System), parece que es una buena solución para el objetivo que atañe a este trabajo, ya que, entre sus múltiples ventajas, incorpora algoritmos de código abierto para la navegación reactiva de robots, con multitud de ejemplos y aportaciones de la comunidad científica.

1.1 Objetivos

Desde el departamento de Electrónica de la Universidad de Alcalá, de busca hacer de la silla de ruedas SARA una silla de ruedas inteligente (*Smart Wheelchairs*, SW), capaz de navegar de manera autónoma con ayuda de múltiples sensores por un espacio cerrado.

El principal objetivo de este trabajo es avanzar en el desarrollo del nodo PC de la silla de ruedas, continuando con la plataforma ROS, implementando la comunicación bidireccional entre el Bus CAN y ROS y realizando una interfaz gráfica (GUI) que mediante el diseño de un joystick digital permita dirigir la silla de ruedas al mismo tiempo que se visualizan datos de los diferentes nodos como pueden ser los sensores de ultrasonidos, el nivel de batería etc.

1.2 Estructura de la memoria

La memoria está distribuida en seis capítulos. En el primero de los capítulos, el capítulo actual, se introduce el trabajo desarrollado, explicando las motivaciones que han llevado a realizarlo y los objetivos planteados.

En el Capítulo 2, se hace una introducción sobre las sillas de ruedas inteligentes (*IW-Intelligent Wheelchairs*). También se describirá ROS, *Python* y GTK, explicando sus principales características, herramientas y los conceptos fundamentales que hay que tener claros para trabajar con esta plataforma y este lenguaje de programación.

En el Capítulo 3, se expondrá cual es la estructura general del sistema de bajo nivel, mencionando muy brevemente todos los elementos de los que dispone la silla y profundizando en el funcionamiento de la comunicación mediante el uso del protocolo CAN. También se comentará el sistema de alto nivel desarrollado en ROS en el último trabajo de [1], explicando la integración visual y física del sistema.

En el Capítulo 4, se describe uno de los bloques principales de este TFG: las modificaciones software de la integración física. En este bloque se describen los cambios realizados en el software en lo referente a la lectura Bus CAN-ROS y como se ha implementado la comunicación para enviar datos desde el nodo ROS instalado en el PC al resto de nodos de la silla de ruedas. Además, se muestran los resultados obtenidos de todos estos cambios.

En el Capítulo 5, se describe otro de los bloques principales de este TFG: la creación de la interfaz gráfica (GUI). Se explica cómo se ha creado y diseñado la interfaz gráfica con *Python* y GTK, implementando un joystick digital para poder dirigir la silla de ruedas y una visualización de los datos recibidos de los sensores de ultrasonidos de SARA. Se muestran los resultados obtenidos con la interfaz creada.

En el Capítulo 6, se exponen las conclusiones obtenidas y los posibles trabajos futuros a desarrollar.

Capítulo 2

Marco teórico

Este capítulo está dividido en tres partes. En la primera de ellas, se describen las sillas de ruedas inteligentes para tener una visión global del funcionamiento y de cómo se ha llegado hasta el punto actual en la investigación de la mismas. Se menciona el principal problema que existe en el mercado que hace que las investigaciones realizadas no lleguen a los usuarios domésticos y cuales podrían ser las soluciones a dichos problemas.

En la segunda parte, se habla de la plataforma ROS la cual es la base de este TFG. Se comentarán las principales características de esta plataforma y de los conceptos (*topics*, nodos, mensajes, paquetes, etc.) básicos que hay que tener en cuenta para poder empezar a trabajar con ROS.

Por último, en la tercera parte de este capítulo, se incluye una breve introducción de *Python* ya que es el lenguaje de programación que se ha usado para el desarrollo, así como GTK para la implementación de la interfaz gráfica de usuario.

2.1 Sillas de ruedas inteligentes

En las últimas décadas ha habido continuos y espectaculares avances en las áreas tecnológicas relacionadas con los sistemas de robótica móvil. Sensores, procesadores y actuadores cada vez mejores y más baratos, unidos a avanzadas estrategias de control, son los actores de estas mejoras. Gracias a ello se han venido proponiendo aplicaciones innovadoras a estos sistemas robóticos, algunas de ellas en el campo de las Tecnologías de Apoyo o de la Asistencia a las personas con dependencia.

Ya a comienzos de los años 80, ciertos trabajos de la Universidad de Stanford [2] proponen utilizar sensores ultrasónicos sobre una silla de ruedas adaptada. En los años 90, otros grupos de investigación alrededor del mundo comenzaron a desarrollar vehículos de asistencia a la movilidad innovadores [3], [4] y [5]. En su concepto inicial, estos vehículos eran similares a cualquier robot móvil estándar: ruedas, motores, baterías y controladores.

El propósito de estos trabajos era lograr un vehículo autónomo al que podemos denominar "Silla Inteligente" (IW - *Intelligent* Wheelchair, Figura 5).

Figura 5 - Modelo modular de la silla de ruedas inteligente (IW)

Los términos más usados para describir a una Silla de Ruedas (*Wheelchair*, W) son: *Smart* (SW), *Intelligent* (IW), *Advanced* (AW), *Autonomous* (AuW), *Robotic* (RW). Siguiendo la línea del trabajo descrito en [6], en el que se propone un modelo modular de una silla de ruedas, se entiende por silla básica (BW) una construida únicamente por motores y un joystick. Si a BW se le añaden módulos tanto software como hardware que proporcionen a la silla capacidad de comunicación, integración de sensores y conducción asistida entre otras características, se puede hablar de sillas avanzadas (AW). Añadiendo módulos de alto nivel que proporcionen autonomía a AW, como sensores de visión, sensores 3D, conducción autónoma y navegación entre otras, se habla de silla de ruedas inteligente (IW).

Como se ha expuesto la investigación en IW ha sido un tema de investigación activa en las Tecnologías de Apoyo (ATs) desde los años 80. Pero tras más de 30 años de investigaciones casi ninguno de estos prototipos ha llegado a los potenciales usuarios. Como principales motivos destacan la falta de sensores adecuados (en términos de coste y fiabilidad) y la falta de una plataforma estándar en el mercado sobre la que construir una IW. Por ejemplo, para detectar obstáculos y reconocer el entorno se requieren sensores muy fiables. La faltan de sensores con la suficiente fiabilidad para todas las posibles situaciones de peligro, mantiene a las IW lejos del mercado por las implicaciones legales resultantes. Actualmente, existen sensores modernos con una relación coste/precio que pueden reducir este problema.

SARA la IW sobre la que se trabaja en este TFG se creó usando elementos electrónicos y de comunicaciones estándar (USB, CAN, Wifi, etc.) con el objetivo de garantizar su compatibilidad con otros sistemas y servicios presentes en el entorno y facilitar su adaptación a diversos perfiles de usuarios, minimizando este problema.

2.2 ROS *framework*

ROS es una plataforma software que incluye una serie de librerías y herramientas de código abierto que ayuda a los desarrolladores de software a crear aplicaciones para robots. Fue desarrollado originalmente en 2007 bajo el nombre de *switchyard* por el laboratorio de inteligencia artificial de Stanford para dar soporte al proyecto del robot con inteligencia artificial de Stanford (STAIR) [7]. Desde 2008 continua su desarrollo primordialmente en

Willow Garaje y en febrero de 2013 se transfiere a la *Open Source Robotics Foundation* (OSRF).

Se desarrolló con la intención de crear un *framework* que permita la integración de la robótica en una gran variedad de situaciones. Además, el hecho de que ROS haya sido creado totalmente bajo la licencia de código abierto BSD (*Berkeley Software Distribution*) ha ayudado a que actualmente ROS sea uno de los *frameworks* más usados en la comunidad robótica.

2.2.1 Características principales

Las principales características de ROS son las que se listan a continuación:

- **"Peer to Peer".** ROS está basado en un sistema distribuido en el que los diferentes procesos (que pueden estar localizados en distintas máquinas o hosts) se comunican entre sí utilizando una topología "peer to peer". Esta topología se basa en utilizar un canal nuevo para la comunicación entre dos procesos distintos, evitando tener que usar un servidor central para comunicar todos los procesos.
- **Multilenguaje**. ROS permite trabajar en diferentes lenguajes de programación, como C++, *Python* y Lisp. También dispone de bibliotecas en Java y Lua, en fase experimental.
- **Basado en herramientas**. ROS proporciona una gran cantidad de herramientas, las cuales permiten implementar diferentes tareas, desde navegar por ficheros, obtener y modificar los parámetros de configuración de un robot, proceso o driver a visualizar la topología de los procesos en ejecución o realizar la comunicación entre procesos.
- **Código libre y abierto**. ROS es código libre bajo términos de licencia BSD que incluye libertad para uso comercial e investigador. El código completo de ROS está disponible al público [8].

2.2.2 Conceptos fundamentales

Para poder comprender el funcionamiento de ROS, en este apartado se describen los elementos fundamentales de esta plataforma, a saber: nodos, ROS Master, mensajes, *topics*, servidor de parámetros, de servicios y de paquetes.

2.2.2.1. Nodos

Son ejecutables que se comunican con otros procesos usando *topics* o servicios. El uso de nodos en ROS proporciona tolerancia a fallos y separa el código del sistema, haciéndolo más simple. Por su parte un paquete (ver en el aptado posterior 2.2.2.7) puede contener varios nodos (cada nodo dispone de un nombre único), cada uno de los cuales lleva a cabo una determinada acción.

2.2.2.2. "ROS Master"

Es una aplicación que proporciona un registro de los nodos que se están ejecutando en cada momento y permite la comunicación entre nodos pues sin el maestro los nodos no serían capaces de encontrar al resto de nodos, intercambiar mensajes o invocar servicios.

2.2.2.3. Mensajes

Los nodos se comunican entre sí mediante el paso de mensajes. Los tipos primitivos de mensajes *(integer, floating point, boolean, etc.*) están soportados y además se pueden crear tipos personalizados.

En la Tabla 1 se puede ver un ejemplo de tipos de mensajes que se usan en ROS:

TIPO PRIMITIVO	DECLARACIÓN TIPO	*C++*	*Python*
uint16	Entero 16 bits sin signo	uint16_t	Int
float32	Flotante de 32 bits	float	float
String	Cadena de caracteres	Std::string	String

Tabla 1 - Tipos de mensaje

2.2.2.4. *topics*

Son canales de comunicación para transmitir datos (mensajes) sin una comunicación directa entre nodos, lo que significa que la producción y el consumo de mensajes esta desacoplada. Así, los nodos pueden actuar como publicadores (*publisher*) y como suscriptores (*subscriber*) de los *topics*, del siguiente modo:

- **Publicador**. El nodo que actúa como publicador es el encargado de crear el *topic* por el cual van a difundir ciertos mensajes. Estos mensajes serán visibles por los nodos que estén suscritos a este *topic*.
- **Suscriptor**. Este nodo se deberá suscribir a los *topics* correspondientes para poder acceder a los mensajes que hay publicados en ellos.

En la Figura 6 se puede ver un ejemplo de comunicación entre dos nodos por medio de un *topic*. En este ejemplo, el nodo "publisher odometry" publica en el *topic* "Odom" un mensaje del tipo "nav_msgs/odometry" y el nodo "Base_movil" accede a este mensaje suscribiéndose a este *topic* "Odom".

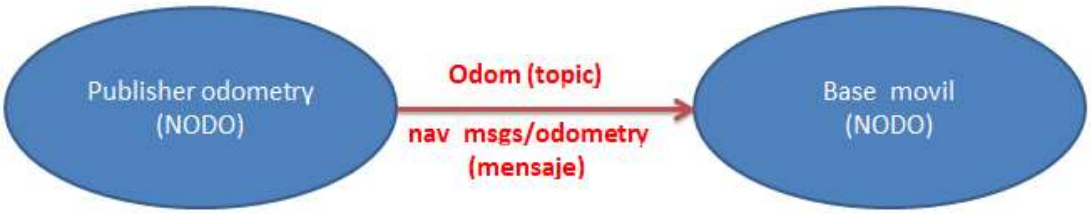


Figura 6 - Comunicación por *topics*

2.2.2.5. Servidor de parámetros

Es un diccionario nombre-valor compartido que puede ser accedido desde cualquier nodo de toda la red de ROS. Los nodos pueden utilizar este servidor para guardar u obtener parámetros en tiempo de ejecución, si bien está pensado para guardar datos estáticos y no demasiado complejos, como por ejemplo parámetros de configuración.

2.2.2.6. Servicios

Cuando se necesita recibir una respuesta en una comunicación con un nodo, no se pueden realizar con *topics*, sino que se necesita hacerlo mediante servicios. En este sistema de comunicación, se dispone de un nodo que actúa como servidor y otro como cliente, los cuales se comunicaran entre sí por medio de mensajes de solicitud/respuesta (ver Figura 7).

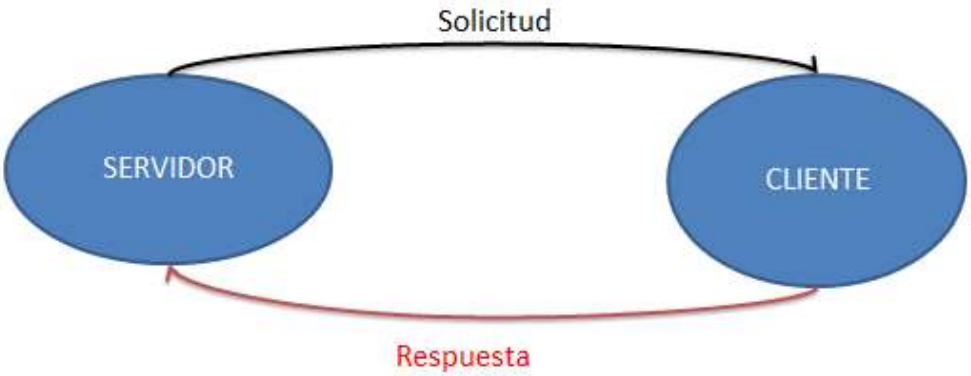


Figura 7 – Comunicación por servicios

2.2.2.7. Paquetes

Los paquetes son la unidad principal de organización en ROS. Un paquete permite agrupar una serie de aplicaciones (nodos), servicios o bibliotecas, facilitando su portabilidad e instalación en diferentes sistemas. La estructura de los paquetes se puede ver en la Figura 8.

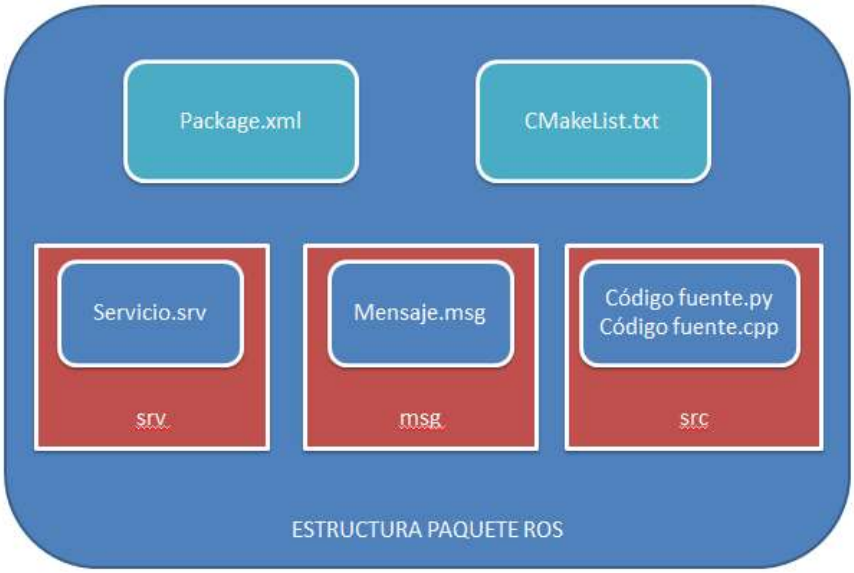


Figura 8 - Estructura de paquetes ROS

A continuación, se explica con más detalle todos los elementos que forman un paquete de ROS:

- **Package.xml.** Fichero utilizado para añadir dependencias e información acerca del paquete.
- **CMakeList.txt.** Fichero que describe cómo se construye el paquete y donde se instala.
- **Directorio src**. Dentro de este directorio se almacenan los códigos fuente del paquete. Estos definen las aplicaciones (nodos) que forman el paquete y pueden estar escritos en diversos lenguajes de programación, p.ej. en C++ (extensión .cpp) o en *Python* (extensión .py).
- **Directorio srv**. En este directorio se almacenan los servicios que, como se ha comentado, son arquitecturas de solicitud o de respuesta encargadas de realizar la comunicación entre nodos.
- **Directorio msg**. En este directorio se almacenan los ficheros que definen la estructura de los mensajes que los nodos utilizaran para comunicarse.

2.3 *Python*

Se trata de un lenguaje de programación interpretado, lo que quiere decir que las instrucciones del mismo no son compiladas, sino que son interpretadas en tiempo de ejecución. Una de las ventajas de estos lenguajes es que son multiplataforma, lo cual permite que los programas en los que se usen puedan ser ejecutados en cualquier sistema operativo. [9]

Una de las características de este lenguaje, es su sencillez a la hora de leerlo, ya que en él se sustituyen muchos de los símbolos que se usan en otros lenguajes de programación por palabras que son más fácilmente relacionables con su significado.

Este lenguaje es multiparadigma. Esto significa que más que forzar a los programadores a adoptar un estilo particular de programación, permite varios estilos: programación orientada a objetos, programación imperativa y programación funcional.

Una de las grandes ventajas de este lenguaje de programación es la gran comunidad de desarrolladores software que lo usan y aportan su documentación, lo cual facilita mucho la tarea del aprendizaje, así como la búsqueda de documentación.

Para el desarrollo del proyecto actual se ha utilizado la versión 2.7 de este lenguaje, ya incluida en el propio sistema operativo (Linux, distribución Ubuntu) usado en este TFG. De la misma forma, se ha utilizado el editor de texto de Ubuntu para crear los diferentes ficheros de *Python*, sin necesidad de descargar ningún IDE específico.

Para el desarrollo del TFG, ha sido necesario introducirse en el mundo de la programación en *Python*, debido que ROS está basado en este lenguaje.

2.4 GTK

GTK son las siglas de GIMP Tool Kit [10], que es una biblioteca que contiene los objetos y funciones que han sido usadas en este TFG para crear la interfaz de usuario.

El uso de esta biblioteca para el desarrollo de interfaces está extendido entre los programadores, lo que hace disponer de multitud de ejemplos y documentación. Esta es una de las razones por las que se ha querido usar GTK en el desarrollo de este TFG.

Para diseñar una interfaz con GTK es necesario crear una ventana donde añadiremos contenedores en los cuales se alojan los objetos como pueden ser etiquetas, botones, escalas etc. En la Figura 9 – Uso de objetos en GTKFigura 9 se muestra un ejemplo del uso de estos objetos. En [11] y [12] podemos encontrar más información sobre cómo se diseña con GTK.

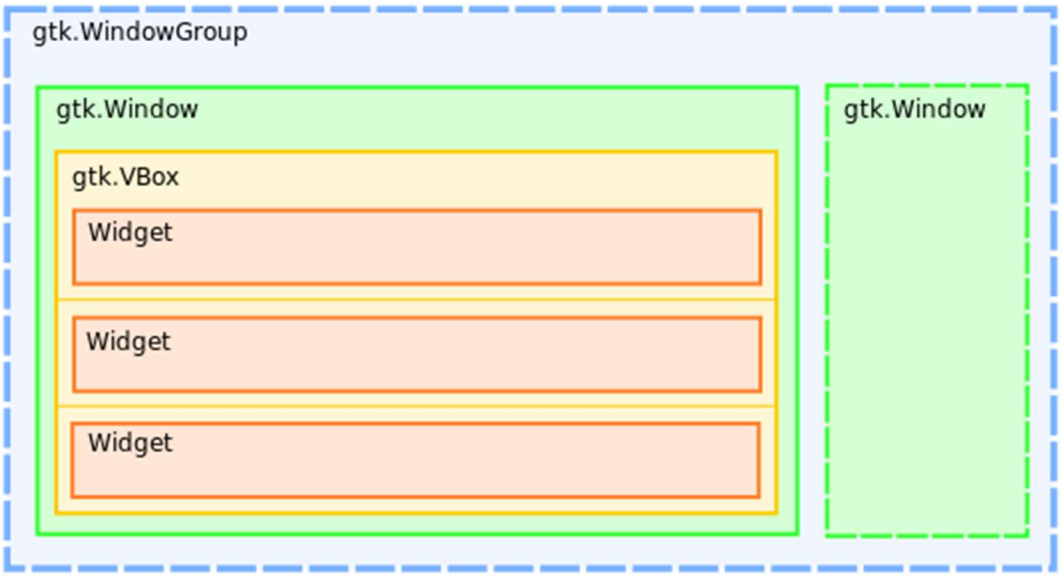


Figura 9 – Uso de objetos en GTK

Actualmente existen gran variedad de lenguajes en los que se puede usar GTK como son: C*++, C#, Java, Javascript, Python y Vala*.

Para el desarrollo de este proyecto, ha sido necesario crear una interfaz de usuario, que se ha definido con GTK para *Python*¸ aprovechando el uso de este lenguaje en ROS. Para ello se ha importado el módulo GTK en la aplicación desarrollada. Una vez importado, en esta aplicación se han ido añadiendo ventanas con diferentes objetos (botones, cuadros de texto, etc.) de la biblioteca GTK.

Capítulo 3

Arquitectura general del sistema

En este capítulo se describe la arquitectura de la silla de ruedas sobre la que se trabaja en el TFG, organizándola en dos partes: en la primera de ellas se incluye la descripción del bajo nivel, necesaria para la comprensión de la comunicación entre ROS y los distintos sensores y actuadores de la silla de ruedas; y en la segunda parte se describe el alto nivel, donde se ha realizado la mayor aportación de desarrollo en este TFG.

3.1 Bajo nivel

Como ya se ha comentado, la silla de ruedas SARA dispone de diversos sensores y dispositivos que se comunican entre sí a través de un bus serie con protocolo CAN. Estos elementos se han agrupado físicamente en 3 módulos hardware diferentes (ver Figura 10), también llamados nodos: motores, sensores y joystick.

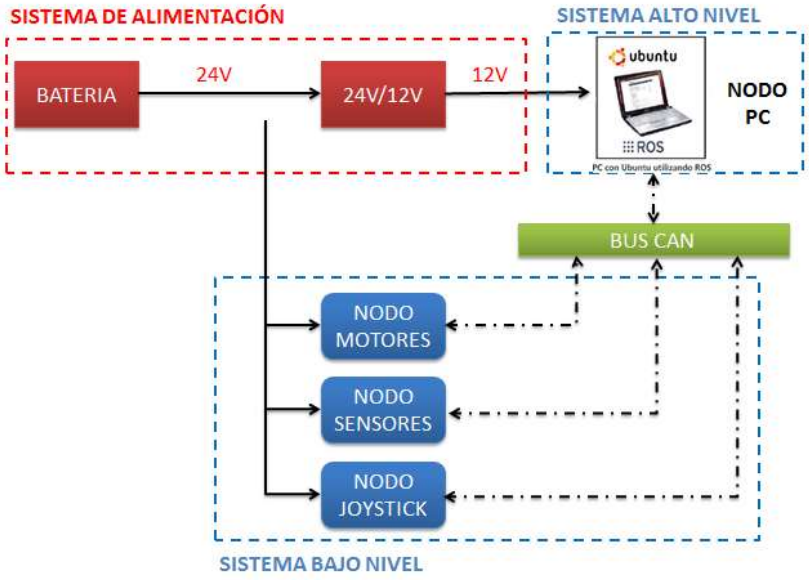


Figura 10 – Esquema general de SARA

Todos estos nodos tienen una tarjeta LPC2129 que dispone de interfaz con bus CAN y realiza, entre otras, la función de gestión de la comunicación con protocolo CAN entre nodos.

En los siguientes apartados se describen en mayor detalle la funcionalidad y el funcionamiento implementado en el bajo nivel de la silla de ruedas de todos estos elementos: bus CAN y nodos.

3.2 Nodos de bajo nivel de la silla de ruedas

El nodo motores está constituido por 2 motores DC con sus correspondientes drivers AX3500 y enconder ópticos HEDS-5500A11, uno para la rueda derecha y otro para la rueda izquierda motrices de la silla de ruedas.

El nodo sensores lo forman ocho sensores de ultrasonidos SRF02 (emisor, receptor y sus correspondientes etapas adaptadoras) repartidos alrededor de la silla, que se comunican, a su vez, con la tarjeta LPC2129 mediante un bus serie con protocolo I2C. Además, también está incorporado en este nodo (en el mismo bus I2C) un acelerómetro para la detección de choques en la silla, aunque su gestión no está implementada.

El nodo joystick incluía, hasta el desarrollo de este TFG, la única interfaz hombre-máquina de SARA, muy básica y compuesta de los siguientes elementos: display, teclado, bocina, luz de encendido, joystick analógico de dos ejes y potenciómetro (ver el documento de [13] para más características sobre estos elementos).

Con la interfaz hombre-máquina básica mencionada, al poner en funcionamiento la silla se pueden elegir entre 4 modos de funcionamiento distintos:

- **Modo joystick**. En este modo la silla de ruedas se mueve manualmente a través del joystick analógico. Corresponde al modo 1 en la silla de ruedas.
- **Modo soplido**. En este modo la silla se mueve usando un detector de soplo a modo de joystick digital (ver [14]). Corresponde al modo 2 en la silla de ruedas.
- **Modo PC**. Este modo es el usado en este TFG para conectar el bajo nivel con la plataforma de ROS. Corresponde al modo 3 en la silla de ruedas.
- **Modo joystick digital**. En este modo la silla de ruedas se controla también manualmente a través del joystick, pero de manera digital, es decir, mediante toques interpretados por un software ad-hoc (ver [13]). Corresponde al modo 4 en la silla de ruedas.

3.3 Funcionamiento del bus CAN

El protocolo CAN está diseñado como una comunicación pensada hacía el mensaje (trama) y no hacia el destinatario. Debido a ello, la información en el bus es trasmitida en forma de tramas estructurados en los que una parte es un identificador que indica la clase de dato que contienen. Así, cuando el bus está libre cualquier nodo conectado puede empezar a trasmitir tramas, y cuando se envía una por el bus todos los nodos lo reciben y verifican el identificador para determinar si la trama va a ser utilizado por ellos, de modo

que si necesitan los datos de la trama lo procesan, y si no los necesitan, la trama es ignorada.

El protocolo dispone de mecanismos para detectar errores en la trasmisión de tramas. Esto hace que las probabilidades de error en la emisión y recepción de tramas sean muy bajas, por lo que es un sistema extraordinariamente seguro, y por tanto perfecto para la aplicación que requiere SARA.

3.4 Estructura de las tramas CAN en la silla de ruedas

Las tramas de los mensajes CAN tienen la estructura mostrada en la Figura 11 cuyos campos se describen a continuación:

Figura 11 - Estructura de una trama en un bus CAN

- **SOF**. Marca el inicio de la trama.
- ***Arbitration Field***. Identificador de la trama. En la trama estándar tiene un tamaño de 12 bits y en la trama extendida 32 bits.
- ***Control field***. Campo de control. Los dos primeros bits están reservados y los 4 restantes indican el número de bytes de datos incluidos en la trama.
- ***Data Field***. Campo de datos de la trama.
- **CRC**. Código de redundancia cíclica. Con este código se comprueba si hay o no errores en la trama.
- **ACK**. Celda de reconocimiento. Indica si la trama ha sido recibida correctamente.
- **EOF**. Marca el fin de la trama.

En el bus CAN implementado en SARA, el empaquetamiento de datos en la trama es manual, es decir, se guardan en ella varias variables concatenadas.

Como se observa en la Tabla 2, por el bus CAN implementado en la silla de ruedas se envían los datos sensados en cada momento del bajo nivel que se listan a continuación:

- **Joyx, Joyy.** Valores de los ejes de ordenadas y abscisas (eje x y eje y), respectivamente, del joystick.
- **velI, velD.** Valores de velocidad de las ruedas izquierda y derecha respectivamente.
- **modo.** Modo de funcionamiento de la silla (ver al principio de este apartado).
- **modo PC.** Este mensaje a diferencia del resto, solo se envía una única vez al acceder al modo PC de funcionamiento o al salir de este. Este mensaje es importante en el desarrollo del TFG ya que lo utiliza el bajo nivel para comprobar si existe comunicación a través del bus CAN con el PC. Funciona de la siguiente manera:
    - Si se accede al modo PC (pulsando el 3 en el teclado de la silla), se envía un solo mensaje con el valor 3 por la trama.

  - Si se sale del modo PC (pulsando el 0 en el teclado de la silla), se envía un solo mensaje con el valor 0 por la trama.

- **encoderAABS, encoderBABS**. Valores (en cuentas) de los encoders A y B, respectivamente (ver en [14] y [15]).
- **tAabs, tBabs**. Marca temporal desde la puesta en marcha hasta la lectura actual del encoder A y B, respectivamente (ver en [14] y [15]).
- **nivel de batería**.
- **SDI, STD SLDD, SLIT, SLID, SDD, SLDT, STI.** Datos de los ocho sensores de ultrasonidos que lleva la silla, siendo S Sensor, D Delantero o Derecho, I Izquierdo y T Trasero.
- **EjeX, EjeY y EjeZ**. Lectura de los 3 ejes del acelerómetro en partes por mil del valor de la gravedad (ver en [14]).
- **DatoD, DatoI y lazo**. Comandos de velocidad para excitar los motores y valor del control que se quiere sobre los mismos (carácter 'A' será lazo abierto y el 'C' lazo cerrado).

EMISOR	IDENTIFICADOR	MENSAJE	TIPO	TAMAÑO
JOYSTICK	0x110 (272)	Joyx	short	2 bytes
		velI	char	1 byte
		velD	char	1 byte
		Joyy	short	2 bytes
		modo	short	2 bytes
	0x111 (273)	modo(para PC)	short	2 bytes
		----	VACIO	6 bytes
MOTORES	0X101 (257)	encoderAABS	int	4 bytes
		tAabs	int	4 bytes
	0x102 (258)	encoderBABS	int	4 bytes
		tBabs	int	4 bytes
	0x210 (528)	nivel batería	int	4 bytes
		----	VACIO	4 bytes
SENSORES	0x201 (513)	SDI	short	2 bytes
		STD	short	2 bytes
		SLDD	short	2 bytes
		SLIT	short	2 bytes
	0x202 (514)	SLID	short	2 bytes
		SDD	short	2 bytes
		SLDT	short	2 bytes
		STI	short	2 bytes
	0x203 (515)	EjeZ	short	2 bytes
		SJO	short	2 bytes
		EjeX	short	2 bytes
		EjeY	short	2 bytes
PC	0x120 (288)	DatoD	char	1 bytes
		DatoI	char	1 bytes

		----	VACIO	2 bytes
		Lazo	short	1 byte
		----	VACIO	3 bytes

Tabla 2 - Formato de las tramas del bus CAN en la silla de ruedas

3.5 Alto nivel

En este apartado se explican todos los paquetes que hasta el momento se han desarrollado sobre la plataforma ROS para la silla de ruedas, en el TFG previo [1].

Concretamente, se han desarrollado 2 paquetes de integración visual, y uno de integración física del robot SARA en ROS.

- El paquete de integración física implementa la comunicación entre el bus CAN de SARA y ROS. En el capítulo 4 se profundiza en la implementación, modificación y ampliación de esta parte, ya que este TFG parte de este punto y continua el desarrollo de este paquete.
- Los paquetes de integración visual permiten trabajar con SARA en sendos simuladores de ROS y se describen en detalle en los apartados siguientes.

3.6 Modelo URDF de la silla de ruedas

En el modelo URDF de la silla se describe su morfología en lenguaje XML para poder representarla y visualizarla mediante el paquete URDF de ROS. Se trata de un modelo morfológico 3D detallado que permite describir tanto el tamaño como el acoplamiento de los distintos elementos del robot (SARA en este caso) para facilitar su tratamiento topológico posterior como nodos dentro de ROS.

En la Figura 12 se puede apreciar cómo quedó finalmente el modelo "SARA_URDF" con todos sus archivos, tras el desarrollo descrito en [1], cuya funcionalidad se indica a continuación:

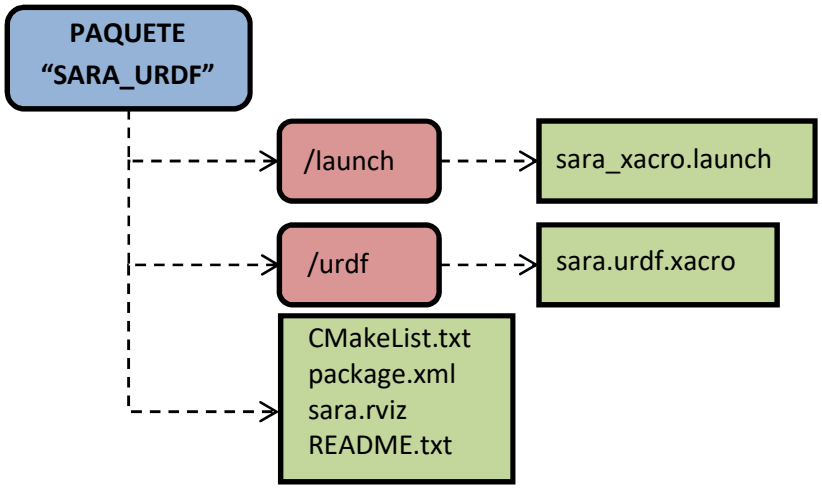


Figura 12 - Descripción topológica del paquete URDF para SARA.

- El archivo *.launch* ejecuta todos los nodos que forman el paquete.
- El archivo *.xacro* contiene la descripción URDF de la silla.
- Los archivos *CMakeList.txt y package.xml* se usan para crear dependencias y construir el paquete.
- El archivo *.rviz* contiene la configuración del modelo en RVIZ. RVIZ es un paquete de ROS que permite visualizar en un entorno 3D la información de *topics* (para más detalles ver [1]).
- El archivo *.txt* contiene pasos a seguir para lanzar este paquete.

En la Figura 13 se muestra la apariencia visual del modelo URDF de la silla de ruedas tal y como queda finalmente.

3.7 Modelo STDR de la silla de ruedas e integración conjunta

El segundo paquete que se utilizó para la integración visual de SARA en ROS fue STDR. El modelado en STDR consiste en una nueva visualización morfológica muy básica de la silla de ruedas, pero integrada en un entorno visual en este caso, para su simulación 2D.

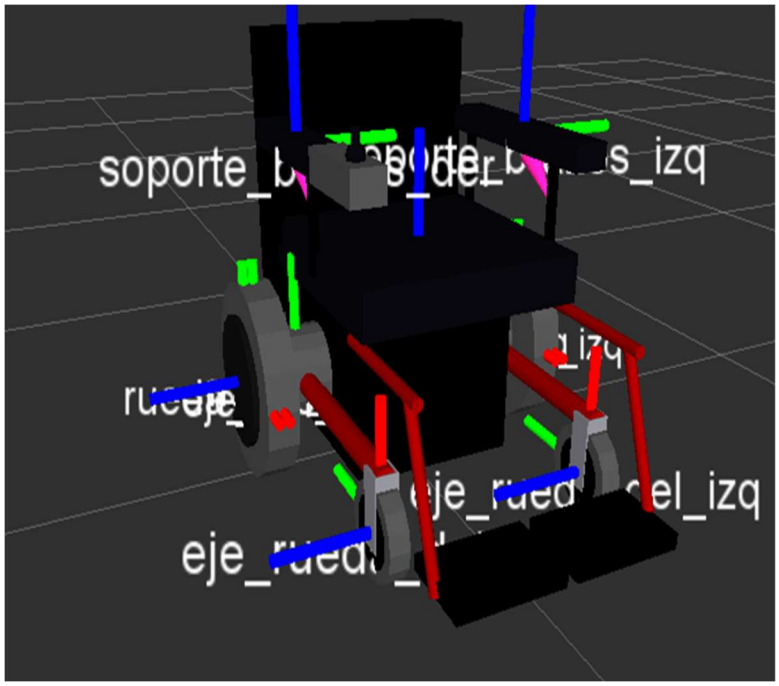


Figura 13 - Apariencia visual del modelo URDF de SARA.

Este simulador permite desplazar el robot por el entorno virtual, con los sensores incluidos (fundamentalmente los de ultrasonidos para el caso de SARA), visualizando los valores que se van obteniendo en estos sensores según se avanza en el entorno.

En la Figura 14 se muestra un ejemplo de visualización del modelo STDR de SARA en un entorno virtual de ejemplo.

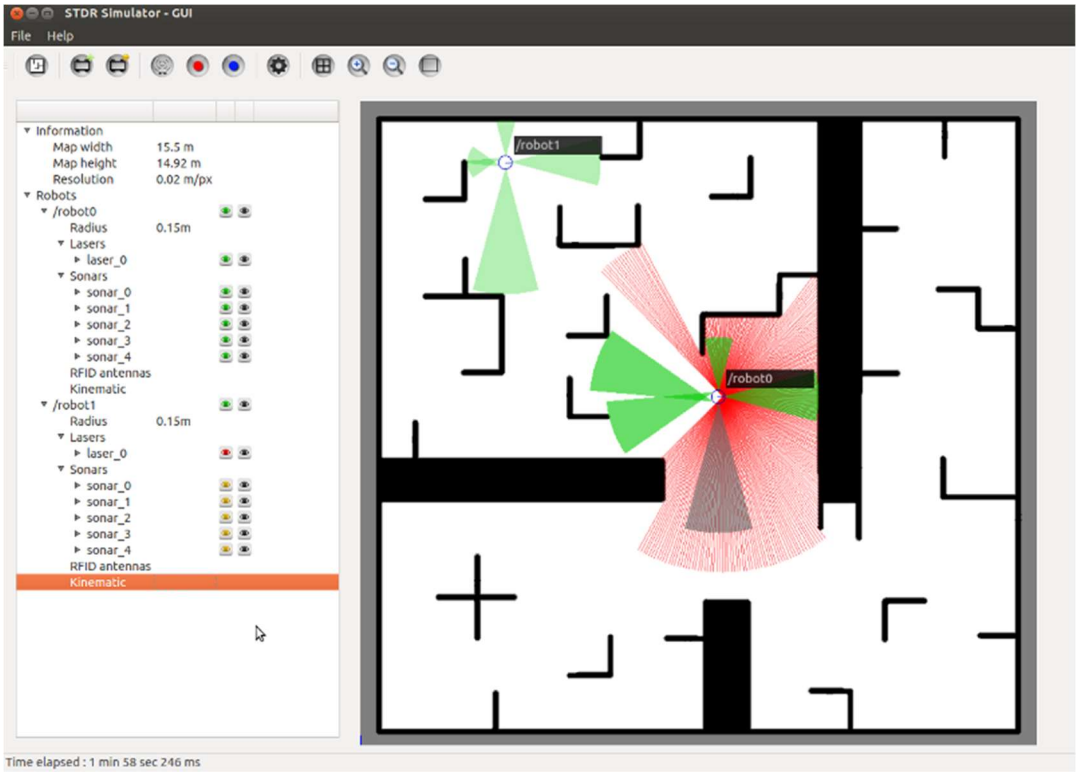


Figura 14 - Modelo STDR de SARA.

Por otra parte, en la Figura 15 se puede apreciar cómo quedó finalmente el modelo "SARA_STDR" con todos sus archivos, tras el desarrollo descrito en [1], cuya funcionalidad se indica a continuación:

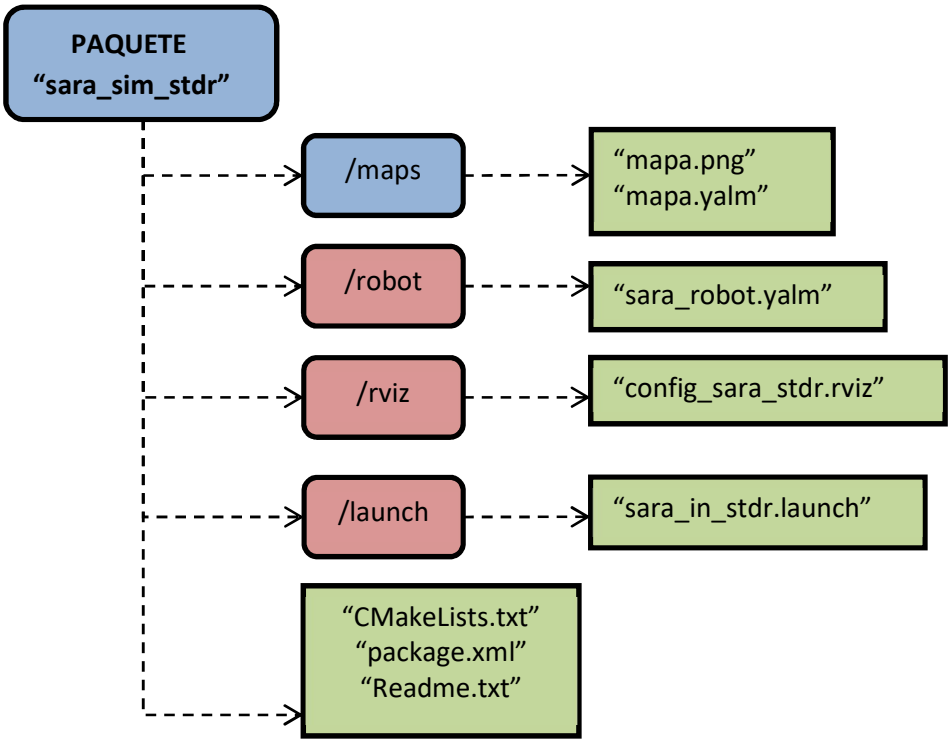


Figura 15 - Descripción topológica del paquete STDR de SARA.

Como último paso, en la integración visual de SARA en los paquetes ROS mencionados, se ha hecho también en el TFG [1] la integración de los dos modelos visuales, con objeto de facilitar la incorporación en trabajos futuros, de otros paquetes de navegación ROS en los que es necesario un conocimiento más completo y complejo de la cinemática y la dinámica del robot (SARA en este caso).

En la Figura 16 se muestra un ejemplo de la visualización de SARA en la integración conjunta de los dos modelos de integración visual ROS desarrollados para ella.

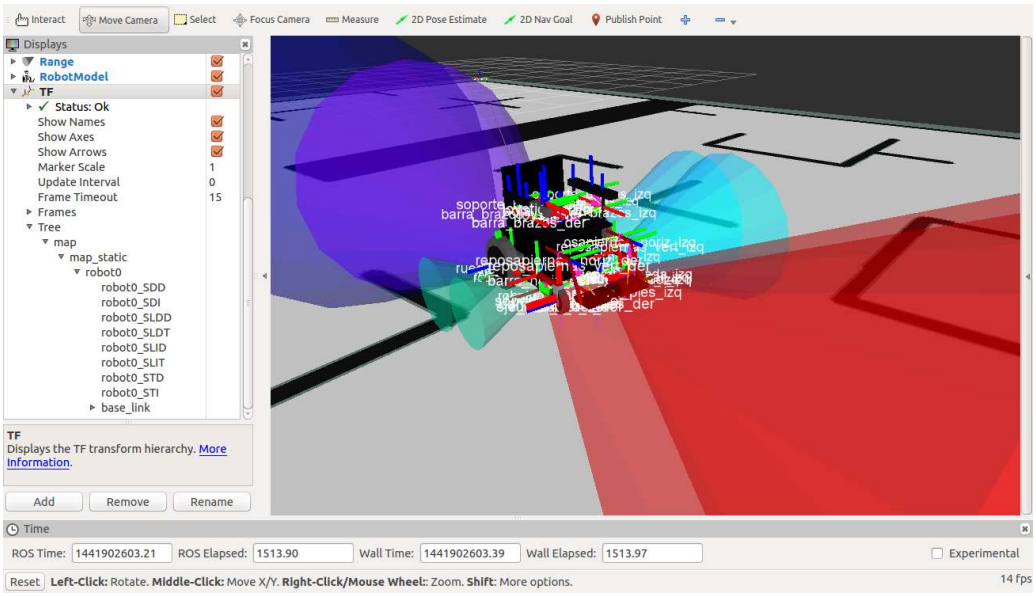


Figura 16 - Visualización de SARA en la integración conjunta del paquete STDR y el modelo URDF.

Capítulo 4

Software de integración física

En este capítulo se explicar el proceso seguido para conseguir que la silla de ruedas se comunique con ROS a través del bus CAN para enviar y recibir los datos de los sensores y actuadores y usarlos en la interfaz gráfica descrita en el siguiente capítulo.

Como se ha explicado antes, se toma como punto de partida el modelo creado en el anterior TFG, con el que solo se podían recibir los datos que enviaban los distintos nodos del bajo nivel, y se describen las modificaciones que se hacen de este paquete para mejorar su funcionamiento, en el primer apartado de este capítulo.

En el siguiente apartado se describen las incorporaciones y se muestra el código añadido al modelo de partida modificado, para lograr que se puedan también enviar datos desde el alto hasta el bajo nivel.

Por último, en este capítulo, se describen también los resultados obtenidos de esta parte del desarrollo del TFG.

4.1 Modificaciones sobre la versión anterior

En el TFG de partida [1] se usó como referencia para el desarrollo del modelo de integración física de la silla de ruedas, un paquete ROS del instituto tecnológico de Kyoto [16]. Este paquete contiene dos scripts y funciona de la siguiente manera (ver Figura 17 y Figura 18):

- El primer script (“canusb.py”) lanza una aplicación que recibe tramas del bus CAN, las desempaqueta y procesa y las publica en un *topic* llamado “canrx” al que puedan suscribirse otras aplicaciones ROS. Al mismo tiempo, también se encuentra suscrito a un *topic* llamado “cantx” que debe recibir las tramas que serán enviadas al bus CAN.
- Por su parte, el segundo script (“test.py”) se encarga de empaquetar los datos que se mandarán al bajo nivel a través del bus Can y de publicar el *topic* “cantx”, mencionado anteriormente.

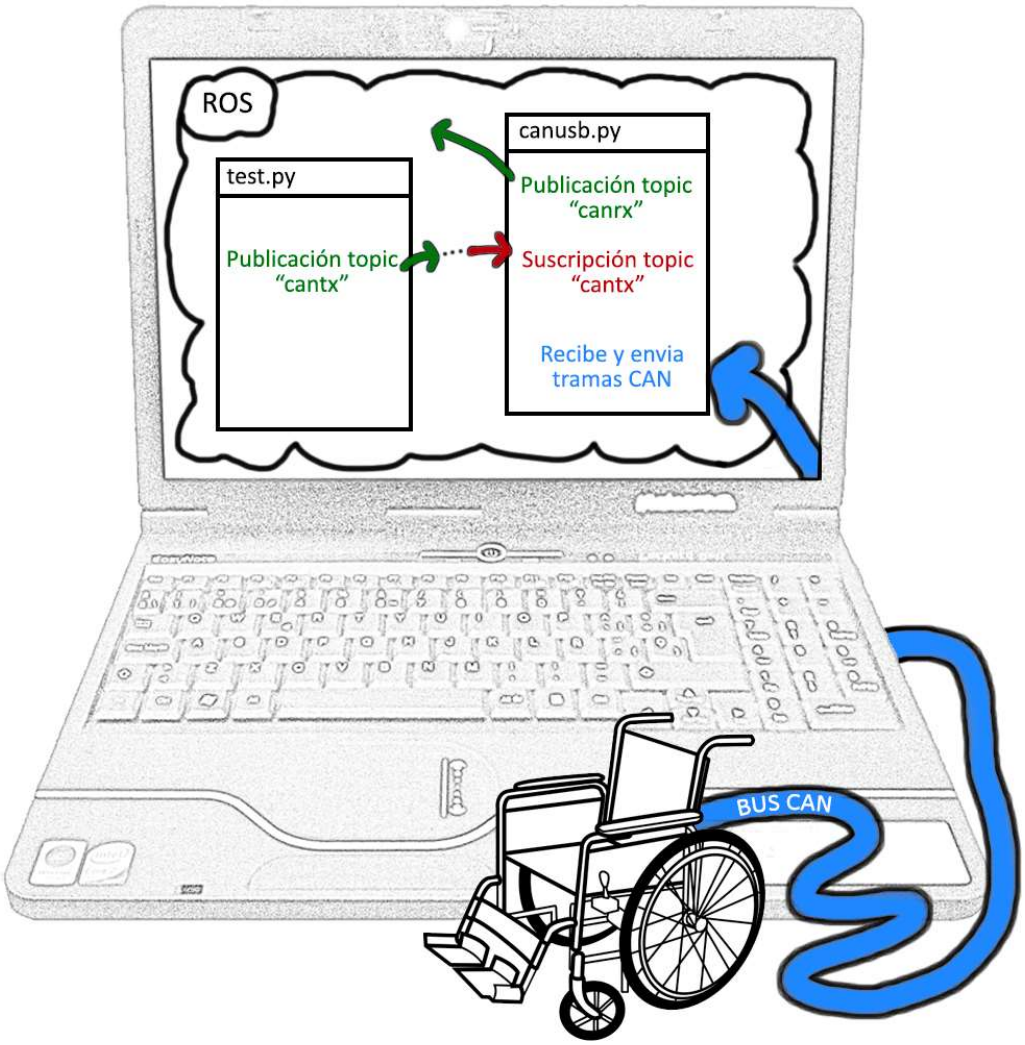


Figura 17 - Descripción gráfica de la integración física del paquete ROS del instituto tecnológico de Kyoto [16].

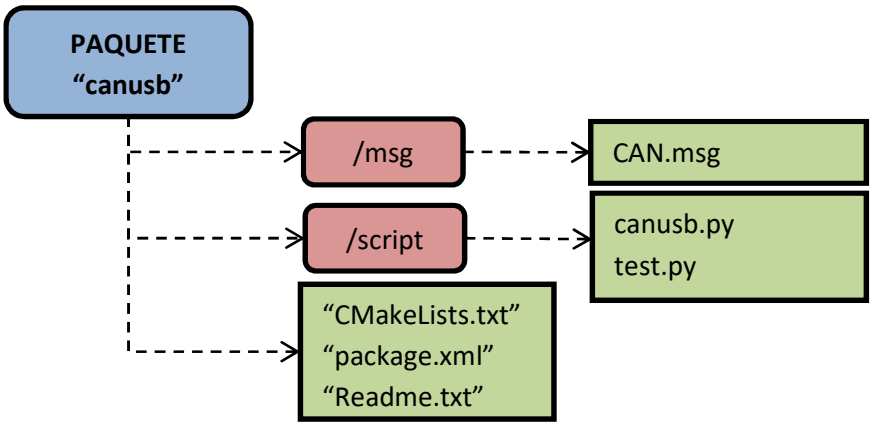


Figura 18 – Descripción topológica de la integración física del paquete ROS del instituto tecnológico de Kyoto [16].

Dicho de otro modo, el paquete original del instituto tecnológico de Kyoto genera dos *topics* correspondientes a las tramas CAN ("canrx" y "cantx"), que están circulando por el sistema ROS para ser utilizados cuando se necesiten, y al mismo tiempo mantiene la comunicación con el bajo nivel, empaquetando y desempaquetando tramas del bus CAN para permitir el constante envío de comandos y la recepción de los datos de los sensores de la silla.

Como en el trabajo que se realizó en el TFG [1] no se necesitaba la parte de envío de comandos al bajo nivel de SARA, ésta se eliminó de la integración física diseñada en ese TFG para SARA.

Fue eliminado, por tanto, el archivo "test.py" y modificado el archivo "canusb.py", de modo que se publicase un *topic* por cada uno de los datos enviados (desde cada uno de los sensores) por la silla de ruedas. Es decir, en vez de publicar sólo el *topic* "canrx" con toda la trama CAN recibida desde el bajo nivel de SARA, se demultiplexan todos los datos incluidos en la trama recibida y se publican 22 *topics* correspondientes a los 22 datos sensados desde los nodos de la silla. Para agrupar estos *topics* en un solo mensaje se creó, además, el archivo "tramaCAN.msg" con la definición de la estructura de cada uno ellos.

En la Figura 20 se puede ver el esquema de integración física que resultó tener SARA después de dichas modificaciones.

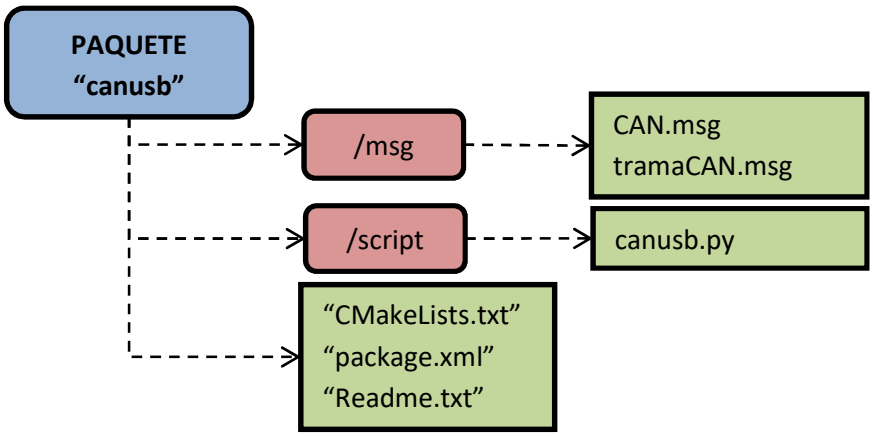


Figura 19 - Descripción topológica de la integración física para SARA del TFG [1].

En el presente TFG se opta por volver a partir del paquete original del instituto tecnológico de Kyoto, para implementar la comunicación bidireccional completa en el modelo de integración física entre el bajo y el alto nivel de SARA. Para ello se toma como referencia parte de dicho código original y se siguen los pasos que se detallan a continuación.

Lo primero que se hace es eliminar el archivo "tramaCAN.msg" del trabajo desarrollado en [1], para volver a utilizar en su lugar el archivo original "CAN.msg".

Después se crea un fichero de aplicación ROS nuevo llamado "lectura.py" donde se añade una suscripción al *topic* "canrx" y parte del código que se encontraba en el script "canusb.py" del TFG de [1], concretamente la parte de demultiplexación de las tramas de

bus CAN y posterior publicación de los 22 *topics* correspondientes a los datos sensados del bajo nivel de SARA.

Las 2 partes (demultiplexacion y publicación) se incluyen en el nuevo código "lectura.py" en un único método llamado "callback", que es llamado cada vez que se recibe un mensaje del topic "canrx", y que pasa a incluir el método llamado "publicar" el llamado "tramas" del código del TFG de [1], con objeto de simplificarlo y hacerlo más intuitivo.

Para hacer la demultiplexación se ha utilizado en este caso el método unpack de la clase struct que está incluido en las librerías de *Python*. Con este método se obtiene una tupla a partir de la cadena que se le pasa como parámetro. Esto se hace así ya que en este caso se debe desagrupar el *tópic* "canrx" que viene como una cadena de caracteres. Para más información sobre este método en [17].

En la Figura 20 y la Figura 21 se muestran, respectivamente, los cambios que se han realizado y como ha quedado el código del fichero "lectura.py" finalmente.

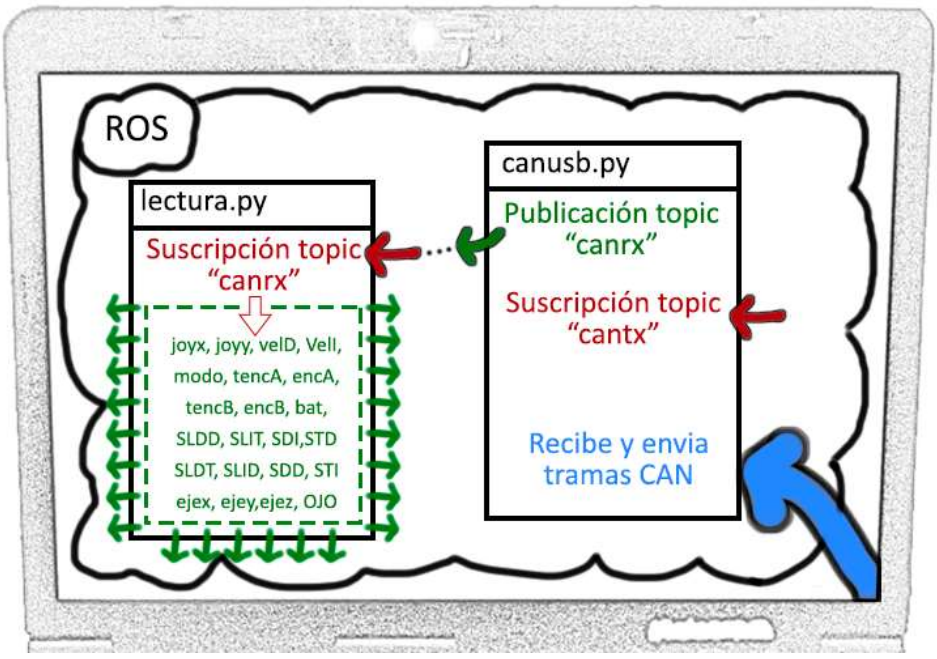


Figura 20 - Descripción gráfica de la integración física para SARA después de las modificaciones del TFG [1].

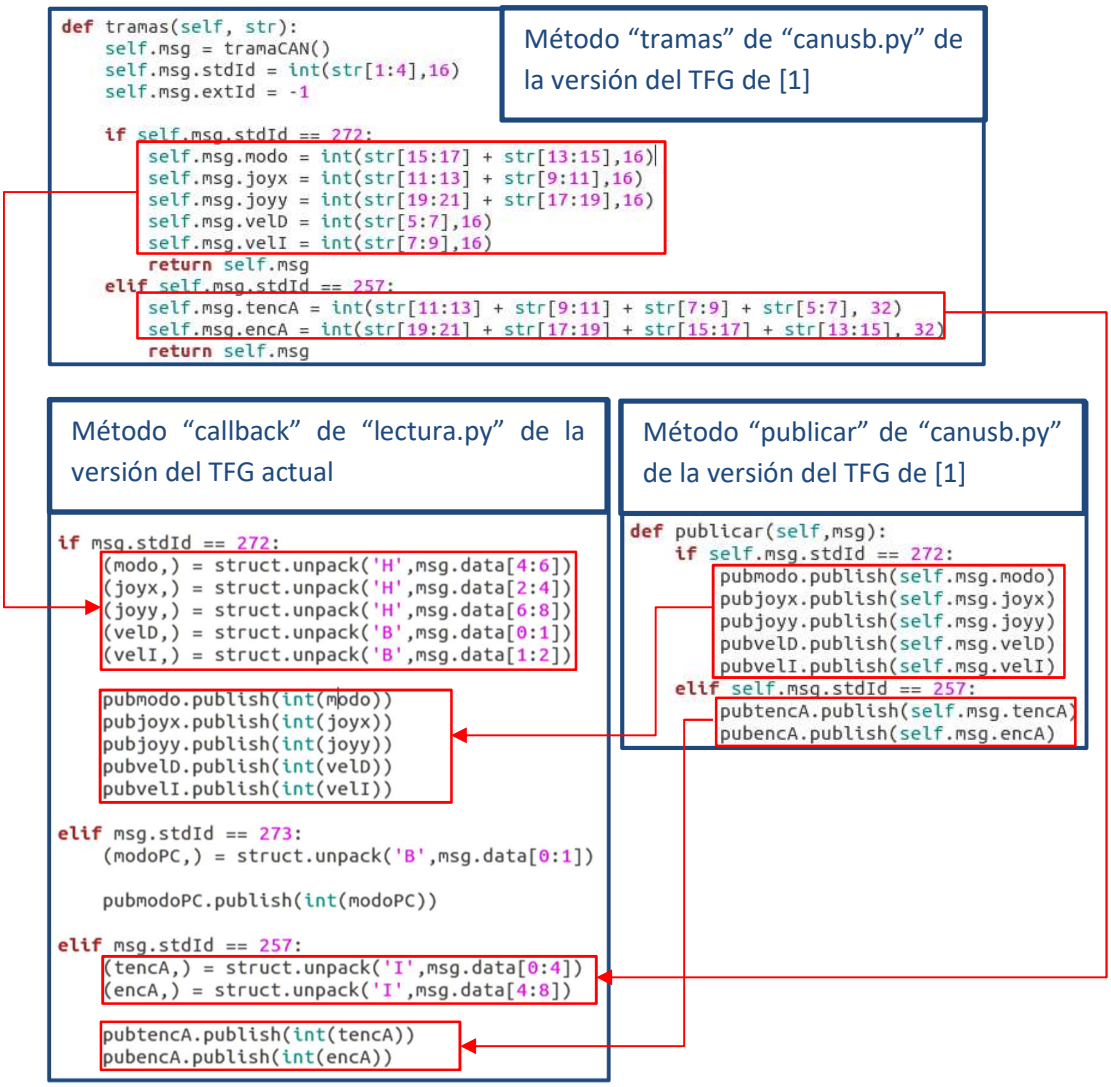


Figura 21 – Cambios realizados en el código del TFG de [1]

Además, se corrigen algunos errores del código "canusb.py" que hacían que algún mensaje no se leyera correctamente: por ejemplo, en el mensaje que contiene la lectura del sensor de tensión de la batería siempre se recibía un 0, ya que estaban mal asignados los bits de la trama CAN que le correspondían.

En la Figura 22 se muestra la evolución del código aplicando las correcciones anteriores

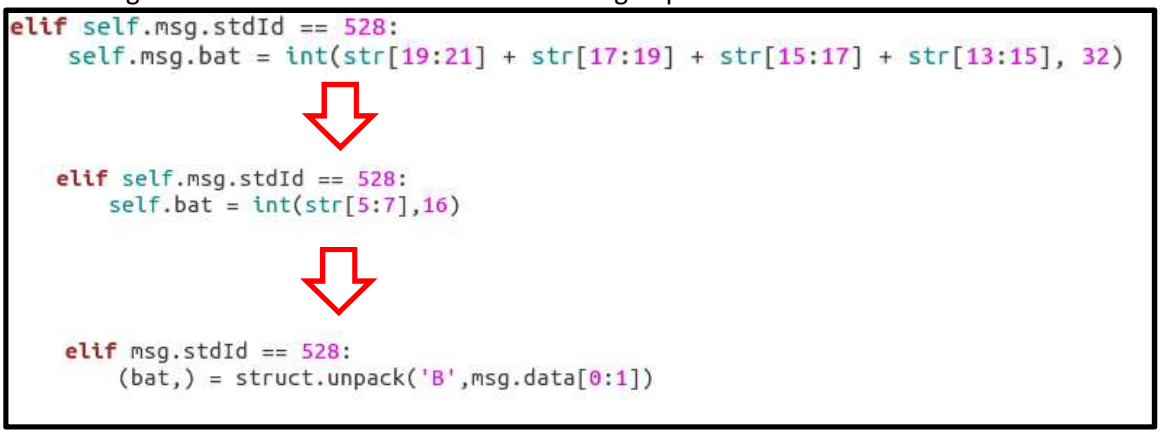


Figura 22 – Corrección de errores del código del TFG de [1]

4.2 Desarrollo sobre la versión anterior

En este apartado se describen todas las funcionalidades desarrolladas a partir del modelo explicado anteriormente.

4.3 Comunicaciones

Para poder enviar tramas al bajo nivel y que este las reciba, la silla de ruedas debe estar en el modo PC. Cuando se pasa al modo PC, el bajo nivel comprueba si existe comunicación entre él y el PC a través del bus CAN y si dicha comunicación concurre el sistema se considera sincronizado.

Como se ha comentado, en la aplicación generada en este TFG se vuelve a incluir el script "test.py" del paquete original desarrollado por del instituto tecnológico de Kyoto [16], que se encargará de enviar los mensajes al bajo nivel y de realizar la sincronización con la silla de ruedas.

En la Figura 23 se puede ver el esquema de integración física del paquete ROS que resultara de este TFG.

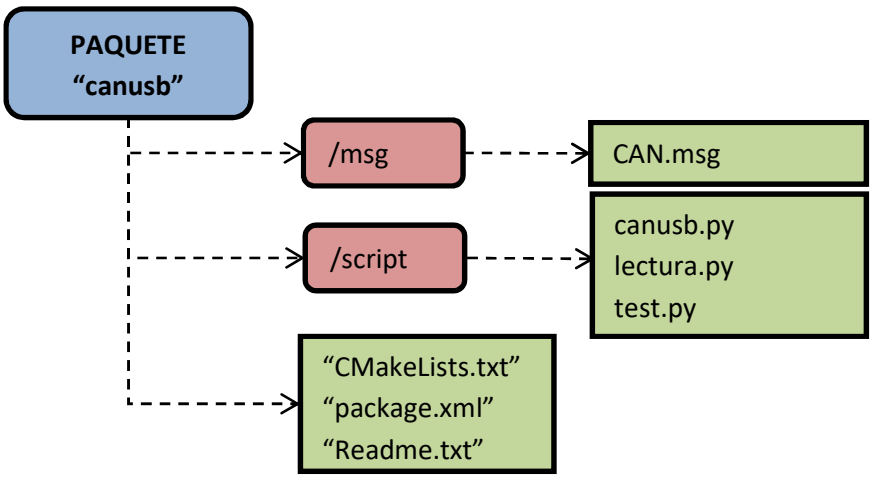


Figura 23 - Descripción topológica de la integración física para SARA de este TFG.

En este script además se añadirá todo el código de la interfaz desarrollado a partir de las librerías GTK de *Python* tal y como se explicará en el siguiente capítulo.

Lo primero que se añade a este script "test.py" es la suscripción al *topic* "canrx". De este *topic* únicamente se recogerá la información de la silla que es relevante para la realización del TFG, descartando el resto de tramas que no se utilizan. En caso de que se necesite probar o verificar el resto de información (sobre todo aquella que no se visualiza en la interfaz referida) se puede ejecutar el script "lectura.py" mencionado anteriormente.

En "test.py" existe un método denominado "lectura" que se encarga de desempaquetar los datos que vienen del *topic* "canrx": SDI, SDD, SLDD, SLDT, STD, STI, SLIT, SLID y modoPC. En la Figura 24 se puede ver la implementación de este método.

```
    def lectura(self, rx):

    if rx.stdId == 273:
        (self.modoPC,)= struct.unpack('B', rx.data[:1])
        self.sincronizar(self.modoPC)

    elif rx.stdId == 513:
        (self.STD,) = struct.unpack('H', rx.data[:2])
        (self.SDI,) = struct.unpack('H', rx.data[2:4])
        (self.SLIT,) = struct.unpack('H', rx.data[4:6])
        (self.SLDD,) = struct.unpack('H', rx.data[6:8])

        self.label_STD.set_text("STD = %d cm" % self.STD)
        self.label_SDI.set_text("SDI = %d cm" % self.SDI)
        self.label_SLIT.set_text("SLIT = %d cm" % self.SLIT)
        self.label_SLDD.set_text("SLDD = %d cm" % self.SLDD)

    elif rx.stdId == 514:
        (self.SLDT,) = struct.unpack('H', rx.data[6:8])
        (self.STI,) = struct.unpack('H', rx.data[4:6])
        (self.SLID,) = struct.unpack('H', rx.data[2:4])
        (self.SDD,) = struct.unpack('H', rx.data[:2])

        self.label_SLDT.set_text("SLDT = %d cm" % self.SLDT)
        self.label_STI.set_text("STI = %d cm" % self.STI)
        self.label_SLID.set_text("SLID = %d cm" % self.SLID)
        self.label_SDD.set_text("SDD = %d cm" % self.SDD)
```

Figura 24 – Código de la implementación del método “lectura”.

Como se aprecia en el fragmento de código anterior se ha añadido la trama CAN que viaja entre el bajo y el alto nivel, con id=0x111 (273) y que no estaba implementada en el TFG de [1]. Esta trama contiene únicamente el mensaje “modoPC”, que se envía una sola vez desde el bajo nivel cuando se accede al modo PC o se sale de este modo de funcionamiento (ver en el capítulo 3).

En realidad, esta trama también se ha añadido al script “lectura.py” descrito en el apartado anterior, resultando finalmente 23 *topics* el número de *topics* que este fichero define en vez de los 22 descritos allí.

La estructura de la trama incluida en el trasiego de datos por el bus CAN se puede ver en la Figura 25.

	LSB(8b)							MSB(8b)
ETIQUETA	MENSAJE A				MENSAJE B			
0X111(273)	modoPC	(Vacío)			(Vacío)			

Figura 25 – Estructura de la trama que contiene el mensaje “modoPC”.

Para realizar la sincronización en TFG [13] se indica que se debe enviar una trama cualquiera, con el identificador de PC, como respuesta al mensaje “modoPC”. Si el mensaje es recibido correctamente en la pantalla del joystick de SARA aparecerá el mensaje “sincronizado” que indica que existe comunicación entre el PC y la silla de ruedas.

La implementación del sincronismo se realiza mediante un método llamado "sincronizar", tal y como se muestra a continuación (Figura 26), que es llamado desde el método "lectura" (ver Figura 24).

```
    def sincronizar(self, sinc):
    if sinc == 3:
        self.enviar(0,0)
```

Figura 26 – Código de la implementación del método "sincronizar".

Por otro lado, se implementa el método "enviar" diseñado para preparar, empaquetar y publicar la trama con los comandos de velocidad para los motores derecho e izquierdo.

En la Figura 26 se muestra la implementación del método "enviar", y en la Figura 25 una llamada a este método en la que se ha pasado como argumentos dos ceros como comandos para conseguir el sincronismo y que las ruedas no se pongan en marcha.

```
    def enviar(self,datoD,datoI):
    pub = rospy.Publisher('cantx', CAN, queue_size=100)
        if not rospy.is_shutdown():
            msg = CAN()
            msg.stdId = 288
            msg.extId = -1
            msg.data = struct.pack('B', datoI) +
struct.pack('B', datoD) + struct.pack('B', 0) + struct.pack('B',
0) + 'A' + struct.pack('B', 0) + struct.pack('B', 0) +
struct.pack('B', 0)

            pub.publish( msg )
            print msg
```

Figura 27 – Código de la implementación del método "enviar".

El envío de comandos por el bus CAN se lleva a cabo preparando una trama con el identificador 0x120 (288), en la que se empaquetan tres mensajes ROS: "datoI", "datoD" (comandos de velocidad para los motores izquierdo y derecho respectivamente) y "lazo" (abierto o cerrado).

La trama así montada (ver Figura 28) se publica como *topic* con el nombre de "cantx".

	LSB(8b)							MSB(8b)
ETIQUETA	MENSAJE A			MENSAJE B				
0X120(288)	datoI	datoD	(vacío)		lazo	(vacío)		

Figura 28 – Estructura de la trama que contiene los mensajes "datoI", "datoD" y "lazo"

La velocidad de los motores se envía codificada en módulo y magnitud en un byte, de modo que el valor 255 corresponde a la velocidad máxima hacia delante, el 1 a la velocidad máxima de retroceso y por tanto el valor 127 detiene los motores.

En el joystick incluido en la GUI se muestra y se controlar el valor de los motores en metros por segundo, por lo que es necesario convertir y escalar estos datos (en el método "scale"), tal y como se muestra en la Figura 29. En el siguiente apartado se justifica la conversión.

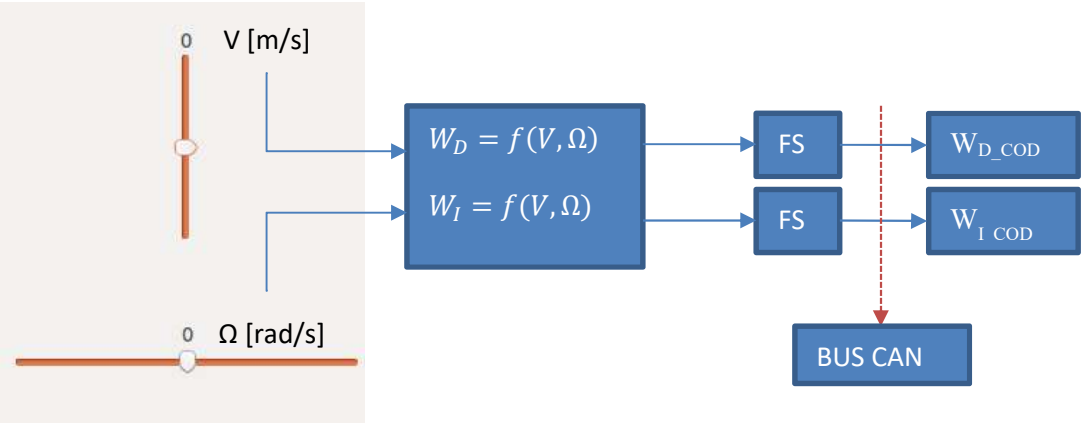
```
    def scale(self, widget):
    V = self.v_scale.get_value()
    R = self.h_scale.get_value()
    wD= ((V/0.155) + (R*1.693))
    wI= ((V/0.155) - (R*1.693))

    #"{0:.0f}".format(wI*10)
    datoD= int(wD*10)
    datoI= int(wI*10)
    if datoD < 0:
        datoD= datoD + 256
    if datoI < 0:
        datoI= datoI + 256
    self.enviar(datoD,datoI)
```

Figura 29 - Código de la implementación del método "scale".

4.4 Escalado

Los valores de velocidad de los motores de la silla de ruedas que se muestran en la interfaz desarrollada en este TFG serán la velocidad lineal y angular de SARA. Estos valores han de convertirse a velocidad angular de cada una de las ruedas de la silla por separado y escalarse, según se ha comentado en el apartado anterior para convertirlos al código de 8 bits que se envía por el bus CAN (ver Figura 30).

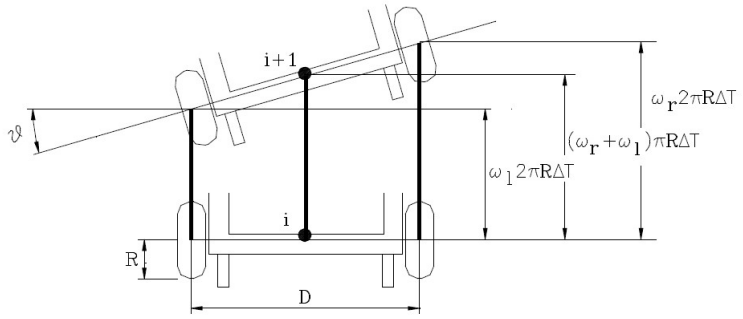


V → Velocidad lineal del móvil [m/s]
Ω → Velocidad angular [rad/s]
FS → Fondo de escala

W_D y W_I →Velocidad angular de la rueda derecha e izquierda respectivamente.
W_{D_COD} y W_{I_COD} → Código de velocidad de la rueda derecha e izquierda respectivamente.

Figura 30 – Esquema de conversión y escalado de los valores de velocidad

En [18] se describen en detalle las ecuaciones y parámetros de la cinemática de SARA (con cinemática diferencial de 2 ruedas) que permiten realizar las transformaciones descritas (distancia entre ruedas, diámetro de éstas, velocidad máxima, etc.) y que se resumen en la Figura 31. Además, esta información se ha comprobado empíricamente, de manera que no se sobrepasen los límites de velocidad confortable o se envíe un comando erróneo.



$$V = \frac{W_D + W_I}{2} \cdot R$$

$$\Omega = \frac{W_D - W_I}{D} \cdot R$$

V → Velocidad lineal del móvil [m/s]
Ω → Velocidad angular [rad/s]
R → Radio de la rueda [m]
D → Distancia entre rueda [m]
W_D y W_I →Velocidad angular de la rueda derecha e izquierda respectivamente.

Figura 31 - Ecuaciones y parámetros de la cinemática de SARA

Concretamente, desarrollando las dos expresiones de la Figura 31, se llega a que la velocidad angular a enviar a cada rueda, a partir de un comando de velocidad lineal y angular de la silla (cinemática directa) puede calcularse de la siguiente manera:

$$W_D = \frac{V}{R} + \frac{\Omega \cdot \mathrm{D}}{2 \cdot R}\ ; \quad W_I = \frac{V}{R} - \frac{\Omega \cdot \mathrm{D}}{2 \cdot R}$$

Además, se ha validado experimentalmente el factor de escala (FS) existente entre el código de velocidad que se empaqueta en la trama CAN y la lineal y angular que selecciona el usuario en la interfaz, teniendo en cuenta que cuando se envía el código máximo la silla se mueve a 2 m/s.

Debido a ello, y a través de las fórmulas de cinemática anteriores (sabiendo que el radio de las ruedas es de R=155 mm y que la distancia entre estas es de D=525mm), se ha decidido fijar la velocidad lineal máxima en V=1m/s y la velocidad angular máxima en Ω=3rad/s en la interfaz gráfica diseñada.

En la Figura 32 se muestra un diagrama de flujo que aclara el funcionamiento del código del archivo “test.py” desarrollado en el TFG.

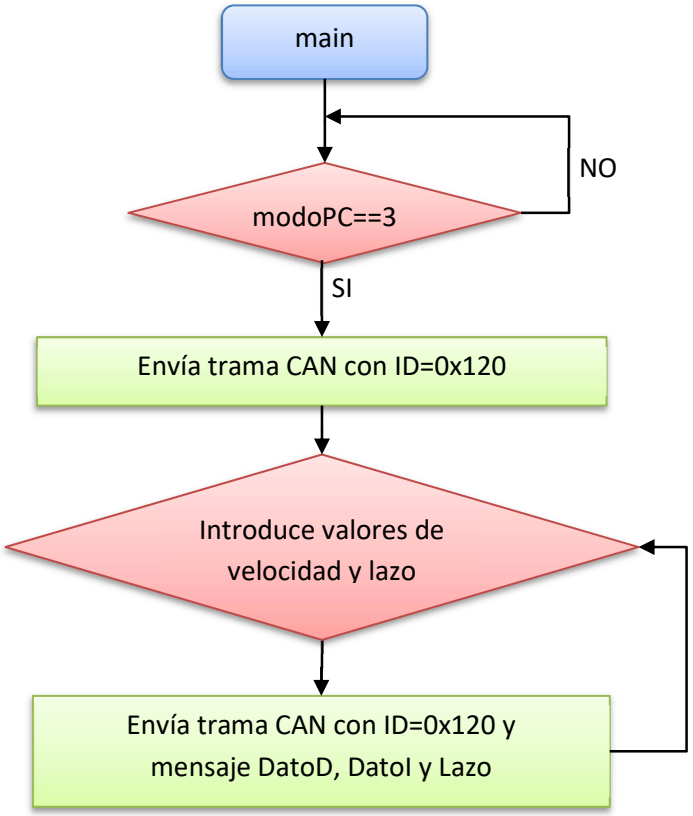


Figura 32 – Diagrama de bloques del funcionamiento del archivo "test.py"

Por último, para ejecutar el paquete "canusb" es necesario indicar el puerto y la velocidad de transmisión de la conexión CAN. Para simplificar y facilitar lo máximo posible el arranque del paquete, se ha creado un archivo *bash* denominado "canusb.sh" y ubicado en el directorio "catkin_ws", con las líneas de código que hacen esta configuración y arrancan el ejecutable "canusb.py" (ver Figura 33).

```
#! /bin/bash

rosparam set /canusb/port /dev/ttyUSB0
rosparam set /canusb/baud 1m
rosrun canusb canusb.py
```

Figura 33 – Código del archivo "canusb.sh"

4.5 Resultados

A continuación, se mostrarán los resultados obtenidos con las diferentes implementaciones explicadas en este capítulo.

Para poder observarlas en el funcionamiento de SARA, se conecta al PC donde está instalada la aplicación a través de un puerto USB (ver **Figura 34**), y se elige en su joystick el modo de funcionamiento (ver Figura 35).

Figura 34 – Fotografía de SARA conectada a través del puerto USB al PC.

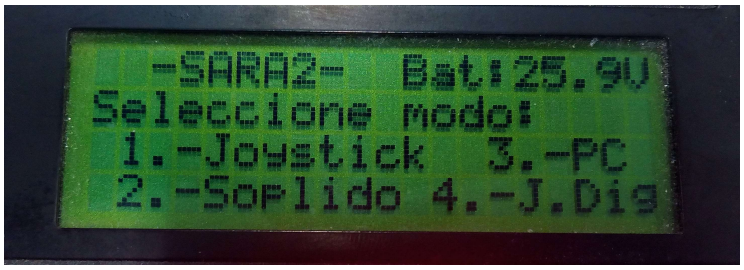


Figura 35 – Menú principal de SARA donde se elige el modo de funcionamiento.

Como ya se sabe en este TFG se está trabajando sobre el modo PC que como podemos ver en la Figura 35 es el modo 3. Si se pulsa el 3 en el teclado del joystick se ve el mensaje "Sincronizando…" (ver Figura 36).

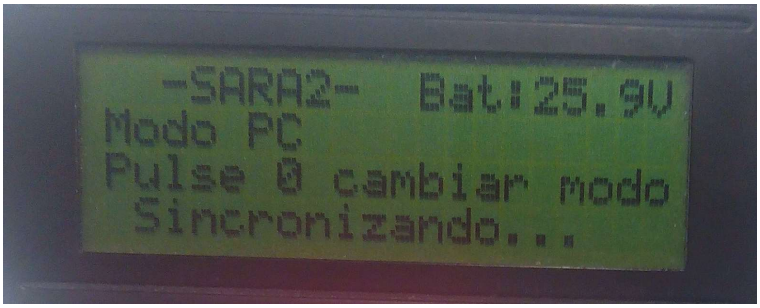


Figura 36 – Mensaje "Sincronizando" de SARA al entrar al modo PC.

Este mensaje indica que la silla está intentando comprobar si hay comunicación con el PC. Como aún no se ha ejecutado el paquete "canusb" en el ordenador, después de un par de segundos se cambia el mensaje y aparece "No se detecta App" (ver Figura 37).

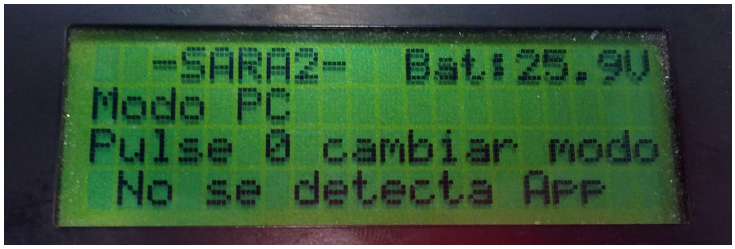


Figura 37 – Mensaje "No se detecta App" de SARA cuando no existe comunicación con el nivel alto (PC).

Cuando aparece este mensaje, SARA indica que no se ha podido detectar ninguna app, es decir, no ha recibido una trama por el bus CAN del nivel alto (PC) y por tanto hay falta de comunicación.

Ahora se vuelve al menú principal pulsando la tecla 0 y se intenta arrancar el paquete implementado en el TFG para comprobar la comunicación y el funcionamiento de los scripts.

Antes de arrancar ningún paquete se debe de ejecutar el ROS Master. Para ello se abrirá un terminal de Linux y dando permisos con "sudo -s" introducimos el comando "roscore".

A continuación, en otro terminal y dando permisos como antes, se ejecuta el archivo *bash* "canusb.sh" que está en la carpeta "./catkin_ws" de ROS el cual primero establece el puerto, luego fija la velocidad del mismo y por último, pone en marcha el script "canusb.py" encargado de recibir y enviar las tramas. A partir de este momento, se puede observar en esta consola todas las tramas que se están recibiendo del puerto serie y que corresponden a la silla de ruedas (ver Figura 38).

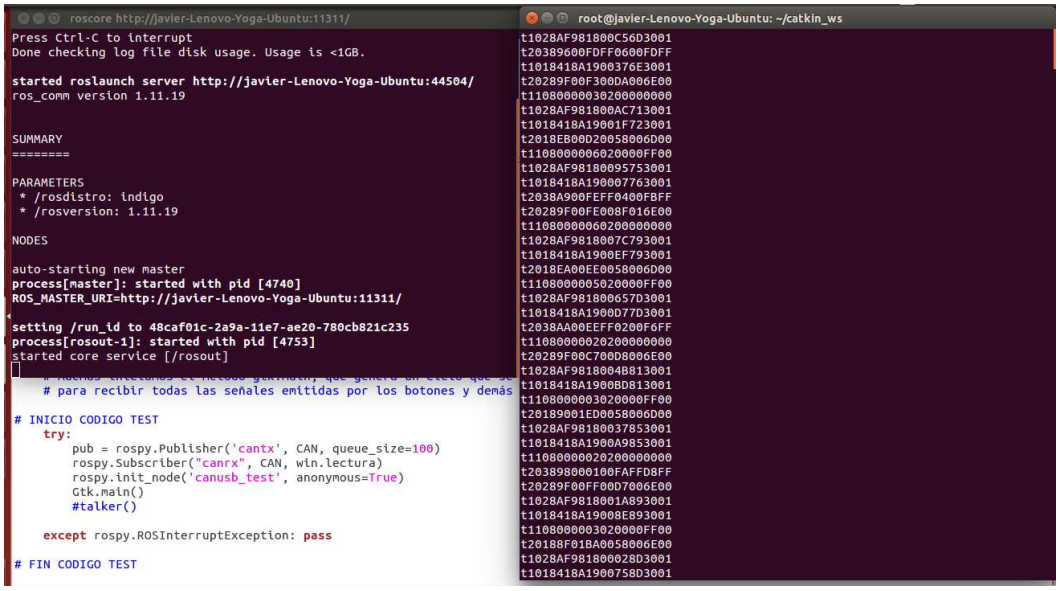


Figura 38 – Tramas recibidas del bus CAN.

En este momento se ejecuta el script "test.py" en otra consola (con todos los permisos) escribiendo el siguiente comando: "*rosrun canusb test.py*".

Una vez ejecutado el script, este se queda a la espera de que se seleccione el modo PC en la silla para enviarle una trama y poder sincronizarse con ella.

Se vuelve a pulsar en el teclado de la silla el botón 3 y ahora el test.py hace su función enviando una trama al bajo nivel de tal forma que la silla detecta comunicación y aparece el siguiente mensaje “Sincronizado” como se aprecia en la Figura 39.

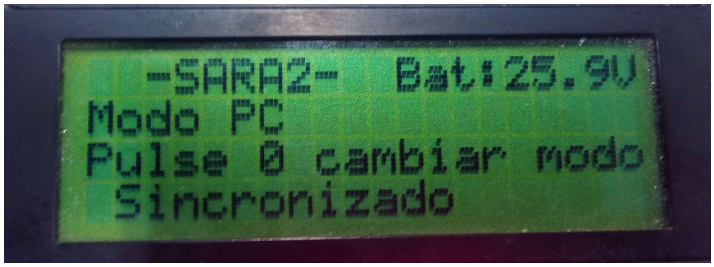


Figura 39 - Mensaje “Sincronizado” de SARA cuando detecta comunicación con el nivel alto (PC).

Este último mensaje indica que ahora la silla si esta sincronizada con el PC y que existe una comunicación bidireccional entre el bajo nivel y el alto. Para comprobar el correcto funcionamiento se varía el valor que enviamos a los motores (ver Figura 40) y se observa como estos toman velocidad en función del valor indicado.

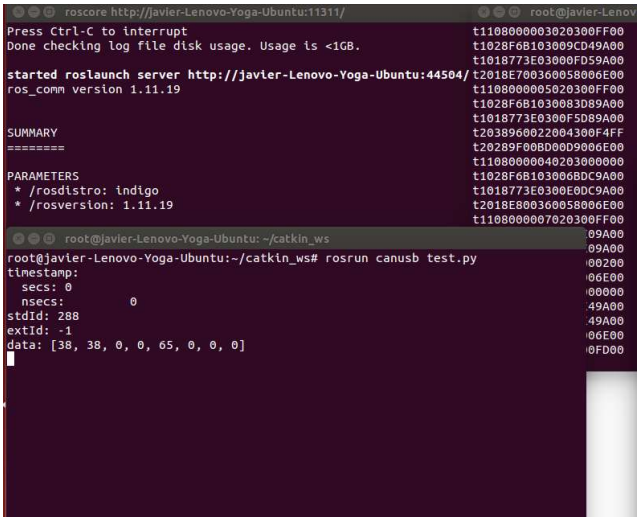


Figura 40 – Ejecución de “test.py” con el valor de 38 en “datoI” y “datoD”

Por otro lado, si se desea leer el valor de un mensaje de una trama cualquiera, como puede ser un sensor, el nivel de la batería etc., se puede ejecutar el script lectura.py que como ya se ha explicado en este capítulo se encarga de publicar en *topics* cada uno de los datos enviados por SARA.

Se ejecuta lectura.py en un terminal nuevo escribiendo “*rosrun canusb test.py”* y *u*na vez lanzado, se abre otra consola y se comprueba los *topics* que actualmente están publicados en ROS con el siguiente comando: “*rostopic list”* (ver Figura 41).

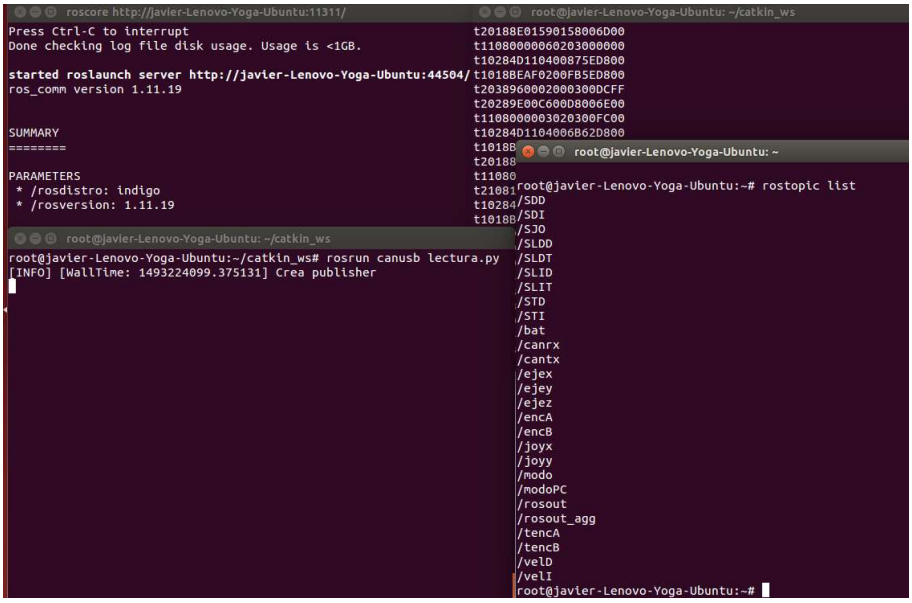


Figura 41 – Ejecución de "lectura.py" y listado de *topics.*

Si se quiere visualizar uno de ellos en concreto y ver el valor que se recibe de este, entonces se utiliza el siguiente comando: *"rostopic echo"* seguido del nombre del *topic* que aparece en la lista. Por ejemplo: *"rostopic echo / bat"* (ver Figura 42).

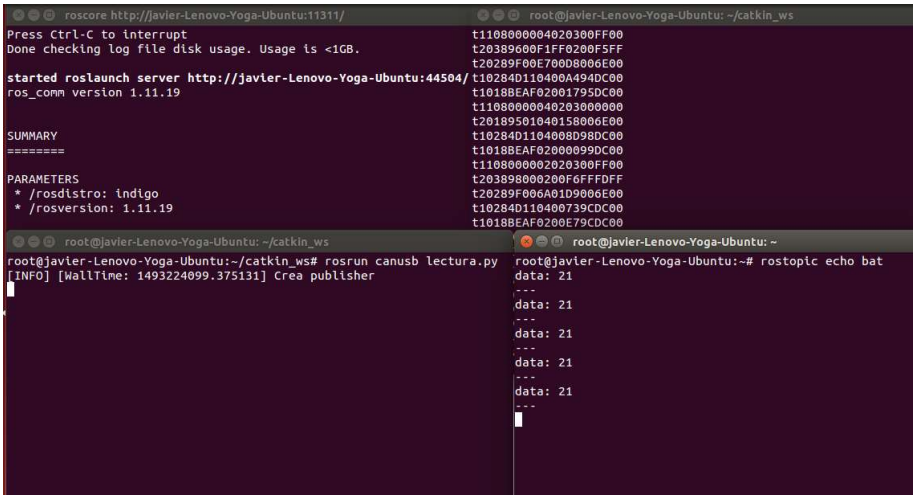


Figura 42 – Datos recibidos del *topic* "bat" (bateria).

Capítulo 5

Creación de la interfaz gráfica

En este capítulo se explicará cómo se ha creado la interfaz gráfica de usuario (GUI), para visualizar la información recibida del bajo nivel de SARA en tiempo real (distancia medida por los sensores de ultrasonidos, etc.) y permitir al usuario moverla a través de unos mandos virtuales.

5.1 Desarrollo

Para crear esta GUI se ha utilizado GTK, que es una biblioteca en *Python* que permite desarrollar entornos gráficos de manera sencilla (ver en Capítulo 2).

En primer lugar, se crea una ventana donde se colocarán todos los elementos que van a componer la interfaz. Se le añade el título (SARA) a la ventana y se le da dimensiones de ancho y alto (en este caso se ha fijado a 500 x 470 pixeles por facilitar la visibilidad y usabilidad de la GUI). El código empleado se muestra a continuación, y el resultado obtenido de este primer paso en la Figura 43.

```
main_win = Gtk.Window()
main_win.set_title("SARA")
main_win.set_default_size(500,470)
```

La biblioteca GTK obliga a utilizar cajas (*.Box*) para ubicar cada uno de los elementos gráficos, es decir, no se puede incluir más de uno de estos elementos en una misma caja.

Las cajas se usan como tablas y las hay de dos tipos: verticales y horizontales. Las primeras permiten la creación de filas y las segundas la creación de columnas, dentro del concepto matricial de las cajas. Además, las cajas se pueden anidar, es decir, se puede incluir una caja dentro de otra, pudiendo desarrollar de esta manera una tabla compleja con tantas filas y columnas como se necesiten.

Figura 43 – Creación de una ventana con el título SARA

El diseño concreto de la GUI desarrollada en este TFG incluye cuatro cajas verticales y tres horizontales, distribuidas según se muestra en la Figura 44, y según se indica en el código siguiente, donde el nombre de las variables se muestra también en la figura para facilitar su identificación.

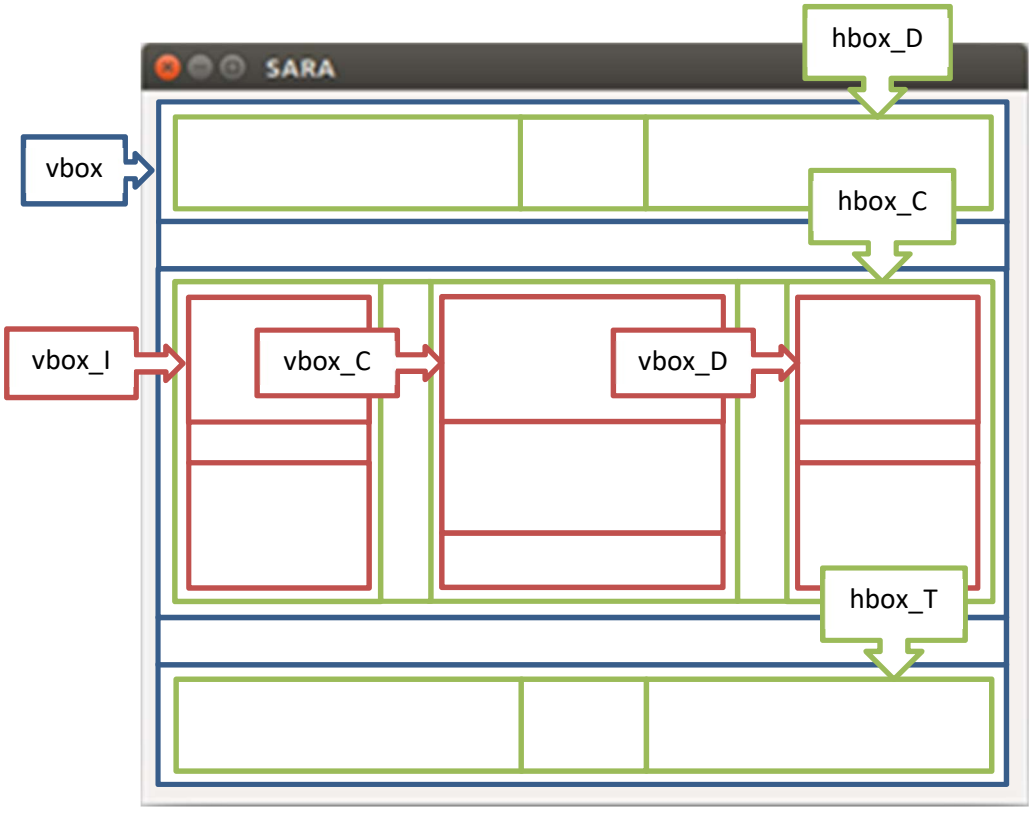


Figura 44 – Diseño de la distribución de las cajas en la ventana de la interfaz

```
        vbox = Gtk.VBox(False,0)

        hbox_D = Gtk.HBox(True,0)
        hbox_D.set_size_request(500,50)

        hbox_C = Gtk.HBox(False,0)
        hbox_C.set_size_request(500,350)
        hbox_C.set_border_width(10)

        vbox_I = Gtk.VBox(True,0)

        vbox_C = Gtk.VBox(False,0)
        vbox_C.set_size_request(250,350)
        vbox_C.set_border_width(10)

        vbox_D = Gtk.VBox(True,0)

        hbox_T = Gtk.HBox(True,0)
        hbox_T.set_size_request(500,50)
```

Una vez creadas las cajas, se crean los separadores que permiten visualizar las líneas de división entre los elementos. En este caso, siguiendo el esquema de la Figura 44, se han creado cuatro separadores horizontales y cuatro verticales, que una vez colocados en su posición, quedarían tal y como se muestran en la Figura 45, mediante el código siguiente.

Figura 45 – Visualización de la ventana SARA con separadores.

```
        sepaV1 =Gtk.VSeparator()
        sepaV2 = Gtk.VSeparator()
        sepaV3 = Gtk.VSeparator()
        sepaV4 = Gtk.VSeparator()
        sepaH1 = Gtk.HSeparator()
        sepaH2 = Gtk.HSeparator()
        sepaH3 = Gtk.HSeparator()
        sepaH4 = Gtk.HSeparator(
```

Después, una vez creado el marco de la GUI, se definen las etiquetas a utilizar para mostrar la distancia medida por los sensores de ultrasonido en centímetros, según la información recibida del bajo nivel de SARA. Para esta GUI se crean ocho etiquetas para corresponder con los ocho sensores de los que dispone la silla de ruedas, cuya definición se realiza como muestra el código incluido a continuación, y que resulta en el diseño mostrado en la Figura 46.

```
        self.label_SDI = Gtk.Label("SDI = 255")
        self.label_SDI.set_halign(Gtk.Align.END)
        self.label_SDD = Gtk.Label("SDD = 255")
        self.label_SDD.set_halign(Gtk.Align.START)

        self.label_SLID = Gtk.Label("SLID = 255")
        self.label_SLIT = Gtk.Label("SLIT = 255")

        self.label_SLDD = Gtk.Label("SLDD = 255")
        self.label_SLDT = Gtk.Label("SLDT = 255")

        self.label_STI = Gtk.Label("STI = 255")
        self.label_STI.set_halign(Gtk.Align.END)
        self.label_STD = Gtk.Label("STD = 255")
        self.label_STD.set_halign(Gtk.Align.START)
```

Figura 46 – Visualización de la venta SARA con las 8 etiquetas que corresponden al valor de los sensores de ultrasonidos en cm.

Para poder incluir un joystick en la GUI de la silla, se han implementado dos nuevos elementos llamados “escalas” que indican el comando de desplazamiento de la silla hacia delante o hacia atrás (velocidad lineal), así como el giro de la misma, hacia la derecha o la izquierda (velocidad angular). A los valores marcados por el usuario en estas dos “escalas”, se les aplican las fórmulas explicadas en el capítulo anterior para obtener el comando de velocidad a enviar a través del bus CAN a cada una de las ruedas.

Para cada una de las dos “escalas” es necesario configurar sus atributos, tal y como se indica en el código siguiente: posición inicial, valor mínimo, valor máximo, incremento por pulsación, área visible, decimales, visualización del valor actual, etc. En la Figura 47 se puede ver la configuración de los atributos para la “escala” vertical.

```
        ad1 = Gtk.Adjustment(0, -1, 1, 0.05, 0.05, 0)
        self.v_scale = Gtk.VScale(adjustment=ad1)
        self.v_scale.set_digits(2)
        self.v_scale.set_draw_value(True)
        self.v_scale.set_inverted(True)
        self.v_scale.set_show_fill_level(True)
```

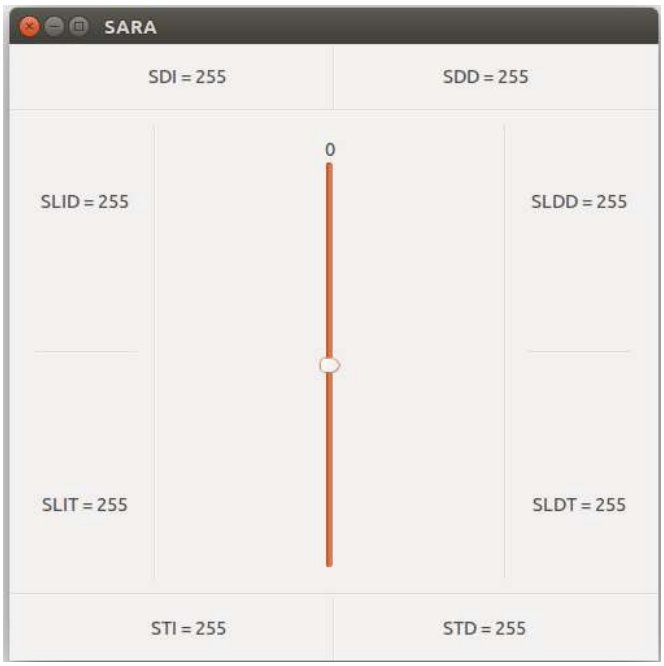


Figura 47 – Resultado de incorporar la “escala” vertical a la interfaz.

De la misma forma, se definen los atributos necesarios para generar la “escala” horizontal, tal y como se observa en la Figura 48 y mediante el código siguiente.

```
        ad2 = Gtk.Adjustment(0, -3, 3, 0.1, 0.1, 0)
        self.h_scale = Gtk.HScale(adjustment=ad2)
        self.h_scale.set_digits(1)
        self.h_scale.set_draw_value(True)
        self.h_scale.set_inverted(False)
        self.h_scale.set_show_fill_level(True)
```

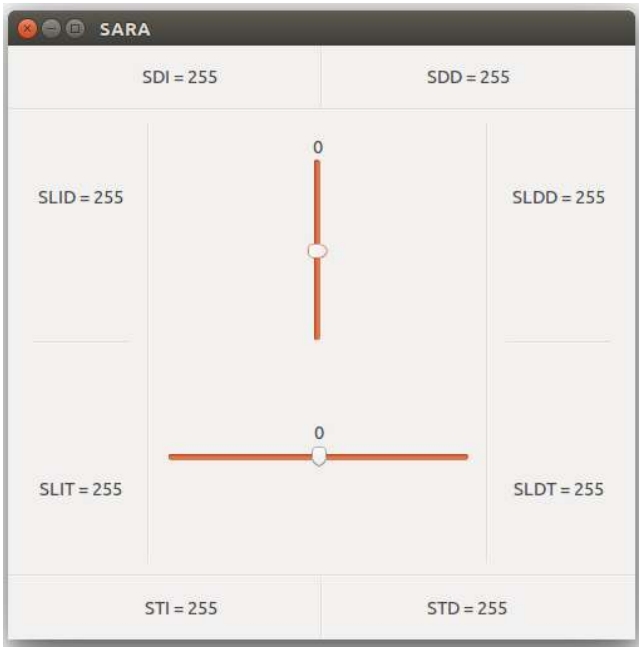


Figura 48 – Resultado de incorporar la "escala" horizontal a la interfaz.

Finalmente, se ha añadido a la GUI un botón que permite detener simultáneamente los dos motores en caso de necesidad de parada de emergencia (ver Figura 49, y el código relacionado). Cuando es pulsado este botón, los cursores del joystick se mueven a su posición de movimiento nulo.

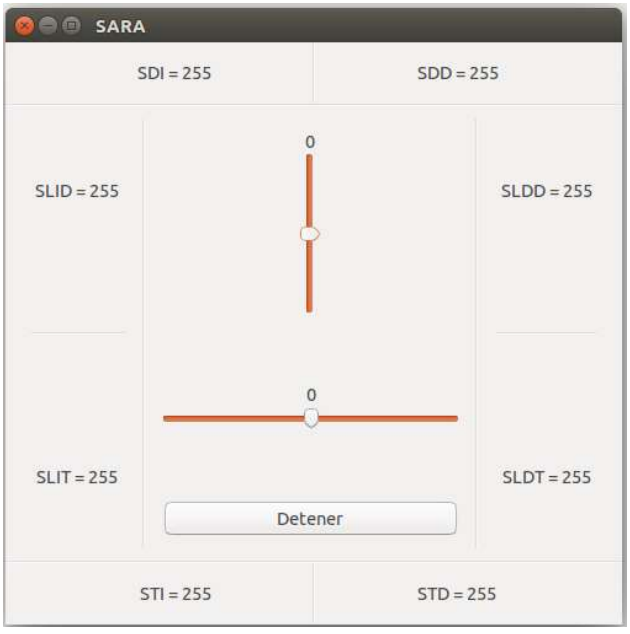


Figura 49 – Resultado final de la GUI para SARA.

```
        button = Gtk.Button(stock=Gtk.STOCK_STOP)
        button.connect("clicked", self.on_button_clicked)
```

Una vez definidos todos los elementos necesarios para la creación de la GUI, se colocan dentro de la ventana diseñada previamente (ver Figura 44). Con el método "pack_start" de

GTK se indica qué elemento va dentro de cada caja. Este proceso se debe hacer siguiendo el orden jerárquico de arriba hacia abajo y de izquierda a derecha.

En el siguiente fragmento de código se puede ver en detalle cómo se han colocado los distintos elementos para conseguir así la interfaz mostrada en la Figura 49.

```
        vbox.pack_start(hbox_D, True, True, 0)

        hbox_D.pack_start(self.label_SDI, True, True, 0)
        hbox_D.pack_start(sepaV1, True, True, 0)
        hbox_D.pack_start(self.label_SDD, True, True, 0)

        vbox.pack_start(sepaH1, True, True, 0)
        vbox.pack_start(hbox_C,True,True,0)

        hbox_C.pack_start(vbox_I, True, True, 10)

        vbox_I.pack_start(self.label_SLID, True, True, 0)
        vbox_I.pack_start(sepaH2, True, True, 0)
        vbox_I.pack_start(self.label_SLIT, True, True, 0)

        hbox_C.pack_start(sepaV2, True, True, 0)
        hbox_C.pack_start(vbox_C, True, True, 0)


        vbox_C.pack_start(self.v_scale, True, True, 0)
        vbox_C.pack_start(self.h_scale, True, True, 0)
        vbox_C.pack_start(button, False, False, 0)

        hbox_C.pack_start(sepaV3, True, True, 0)
        hbox_C.pack_start(vbox_D, True, True, 10)

        vbox_D.pack_start(self.label_SLDD, True, True, 0)
        vbox_D.pack_start(sepaH3, True, True, 0)
        vbox_D.pack_start(self.label_SLDT, True, True, 0)

        vbox.pack_start(sepaH4, True, True, 0)
        vbox.pack_start(hbox_T, True, True, 0)

        hbox_T.pack_start(self.label_STI, True, True, 0)
        hbox_T.pack_start(sepaV4, True, True, 0)
        hbox_T.pack_start(self.label_STD, True, True, 0)
```

5.2 Resultados

Una vez que se tiene la interfaz y el software de integración física, que hace posible la comunicación bidireccional con la silla, se pasa a comprobar su funcionamiento conjunto.

Para ello se debe ejecutar el software de integración física, tal y como se explica en el capítulo anterior, lanzando el script “test.py” del paquete “canusb”. De ese modo se abre también la interfaz ya que todo el código de ésta se ha añadido en este script (ver Figura 50).

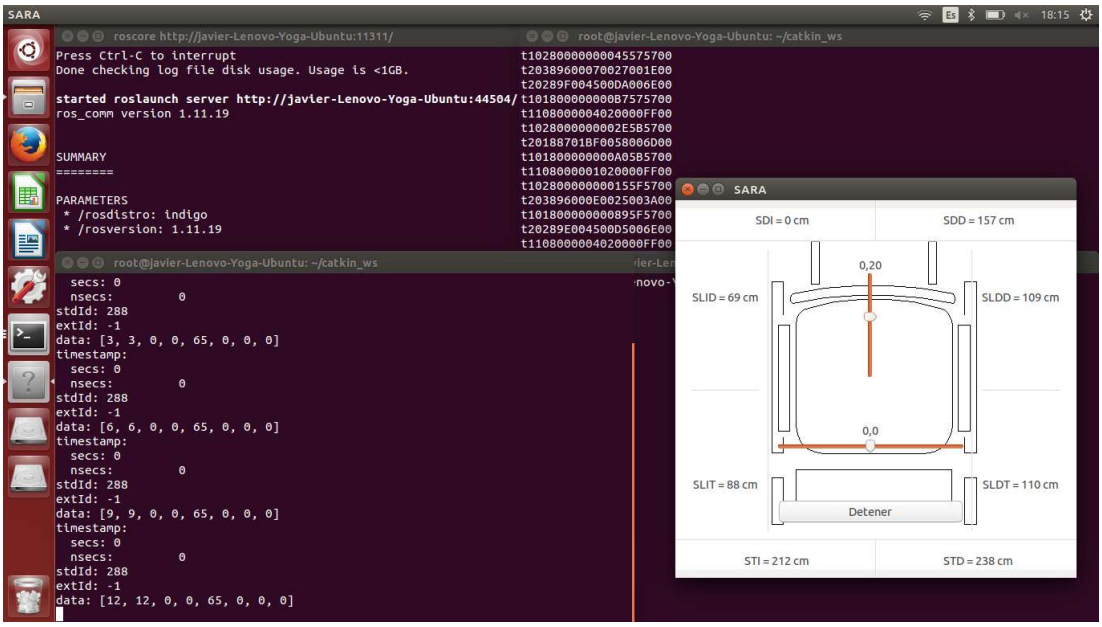


Figura 50 – Ejecución del archivo "test.py" con el código de la GUI.

Una vez lanzada la interfaz se observan en ella los datos de los sensores de ultrasonidos, pudiéndose entonces comprobar que estos muestran una medida correcta y que esta medida cambia en función de la distancia a la que se encuentren los objetos de los sensores.

Además, si se desplazan las "escalas" tanto horizontal como vertical del joystick de la GUI se consigue que se muevan los motores de la silla de acuerdo con el comando indicado en las "escalas".

Concretamente, si se modifica únicamente la "escala" vertical ambos motores aceleran progresivamente en función del dato introducido y si se toca la "escala" horizontal la silla gira hacia la dirección indicada.

Finalmente, cuando se manipulan ambas escalas se determina el valor de consigna para cada motor que consigue que la silla avance a la velocidad lineal ("escala" vertical) y angular ("escala" horizontal) que se ha fijado.

En la Figura 51 se observa un pantallazo de un ejemplo de movimiento de este tipo, destacándose con un recuadro amarillo la trama CAN que se va a enviar a la silla en valor decimal en este caso, en el que se muestran:

- *stdId*, que indica el identificador de la trama CAN, en este caso es 288 (0x120), correspondiéndose con el del nodo de motores.
- *extId*, que indica que la trama CAN no es extendida, es decir que los datos serán 8 bytes en total (4 bytes el mensaje A y 4 bytes el mensaje B)
- *data*, que se corresponde con el mensaje que se va a transmitir e incluye el comando de velocidad de los motores y su configuración (lazo abierto o cerrado). Estos datos están divididos en 8 de 1 byte cada uno. En la imagen se ve que el primero es 254 y el segundo 28, correspondiéndose con los comandos de velocidad de los motores. El

quinto dato es 65 (en ASCII una ‘A’) ya que en esta ocasión se ha configurado el sistema en lazo abierto.

Figura 51 – Visualización de la GUI y de las tramas que está enviado.

Capítulo 6

Conclusiones y trabajos futuros

En este capítulo se exponen las conclusiones obtenidas al finalizar el trabajo y los posibles trabajos futuros.

6.1 Conclusiones

Para poder enfrentarse a este trabajo ha sido necesario adquirir unos conocimientos básicos de *Python,* así como del sistema operativo ROS ya que es la línea que se ha querido seguir y utilizar en trabajos pasados. Esto es así, porque ROS es muy utilizado hoy en día para todo tipo de aplicaciones robóticas al disponer de una gran cantidad de información y de modelos desarrollados por otros usuarios.

Debido a que lo desarrollado hasta el momento no permitía comunicarse con el bus CAN de forma bidireccional, surge la necesidad de complementar y ampliar el proyecto de manera que se pudieran enviar datos a la silla de ruedas y esta pudiera moverse a través del PC con la plataforma ROS.

Haber conseguido esta comunicación, deja finalizada esta tarea permitiendo a futuros proyectos incorporar algoritmos de navegación reactiva u otros. Además, se ha simplificado el proceso y recopilado la información más relevante para facilitar la labor a los futuros investigadores.

Lo más costoso de esta parte ha sido sin duda enfrentarse a un trabajo que se ha estado desarrollando por terceras personal a lo largo de muchos años. Se ha tenido que recopilar información de todos estos proyectos y adaptarlos a las nuevas necesidades para su correcto funcionamiento. Además, enfrentarse a un lenguaje de programación orientado a objetos como es *Python* y que no se conocía, así como a una plataforma y sistema operativo desconocido (ROS).

Antes de ponerse a desarrollar cualquier paquete para la silla como pueden ser algoritmos de navegación reactiva o similares surge una necesidad de saber si todo funciona correctamente para descartar que el problema o el error que surja sea del propio desarrollo.

Al comiendo de este TFG para verificar que los sensores funcionan correctamente había que introducir un “echo” en la consola por cada sensor o dato que se deseara verificar. Esto ralentiza mucho el tiempo de chequeo de la silla en su totalidad.

Para evitar esto se ve necesario la creación de una GUI que con un simple vistazo se pueda verificar que todo está funcionando correctamente.

Además, se añade al script una simulación de un joystick que permite comandar los motores de la silla de ruedas verificando su funcionamiento y que también por otro lado servirá para movernos por un entorno mediante el PC como alternativa al joystick físico que incluye SARA.

6.2 Trabajos futuros

A partir del desarrollo de este TFG se exponen las posibles mejoras o trabajos futuros que se pueden realizar.

- Una vez creada esta primera interfaz, se pueden introducir algunos otros valores, tales como nivel de batería, endoders, etc. que sean de interés para otras aplicaciones que se desarrollen en un futuro.
- Otra posible mejora sería poder indicarle desde la propia interfaz el modo de funcionamiento con el que queremos operar en SARA, es decir, poder pasar por ejemplo al modo soplido desde la GUI. Para esto se necesitaría modificar tanto del bajo nivel de la silla de ruedas como el alto nivel.
- Con el paquete desarrollado “canusb” de este TFG, y los paquetes de integración visual del TFG de [1] se pueden empezar a probar y aplicar algoritmos de navegación reactiva local tanto en simulaciones como tiempo real.

Pliego de condiciones

A continuación, se detallan los elementos físicos y las herramientas software utilizadas durante el desarrollo de este proyecto.

7.1 Equipos Físicos

- Silla de ruedas motorizada

Dimensiones	108 cm x 58 cm x 110 cm (largo - ancho - altura)
Radio de la rueda	15,5 cm
Distancia entre ejes	52,5 cm
Velocidad máxima	2 m/s

- Encoders “HEDS-5500”.
- Batería 24V
- 2 x shimastu npc26-12(12V26AH)
- Display LCD serie “i2c S310118”.
- Teclado matricial 4 x 3 teclas “S310119”.
- Joystick
- Sensor de soplido ”awm2000v”.
- Cargador de baterías ”EMS Power 7969”.
- Convertidor dc-dc ”Mascot 8862000079”.
- Tarjeta de potencia Roboteq AX-3500.
- Tarjeta micro controladora Philips LPC2129.
- Sensores de ultrasonidos “SRF02”.
- Acelerómetro” ADXL330”.
- Conversor de puerto USB a bus CAN “Lawicel CANUSB”.
- Ordenador portátil Lenovo Yoga 500.

7.2 Software

- Sistema operativo GNU/Linux distribución Ubuntu 14.04

- ROS Indigo
- Sistema operativo Windows 10.
- Procesador de textos (Microsoft Word).

Presupuesto

En esta parte del trabajo se hará una estimación del coste total que supone la ejecución del mismo. En los apartados siguientes aparecen los gastos agrupados según su origen, y en el último apartado se detalla el presupuesto total.

8.1 Recursos hardware

CONCEPTO	PRECIO UNIT.	CANTIDAD	SUBTOTAL
Ordenador portátil Intel Core, i7-5500U	700,00€	1	700,00€
Silla de ruedas SARA	16.000,00€	1	16.000,00€
		TOTAL	16.700,00€

8.2 Recursos software

CONCEPTO	PRECIO UNIT.	CANTIDAD	SUBTOTAL
Microsoft Office Word 2010	60,00€	1	60,00€
Material de oficina	120,00€	1	120,00€
		TOTAL	180,00€

8.3 Coste de la mano de obra

La realización de este proyecto ha sido llevada a cabo por las siguientes personas:

CONCEPTO	PRECIO UNIT.	CANTIDAD	SUBTOTAL
Ingeniero	50,00€	600	30.000,00€
		TOTAL	30.000,00€

8.4 Presupuesto de ejecución material

Es la suma total de los importes del coste de materiales y de la mano de obra.

CONCEPTO	PRECIO
Coste recursos hardware	16.700,00€
Coste recursos software	180,00€
Coste mano de obra	30.000,00€
TOTAL	46.880,00€

8.5 Importe de la ejecución por contrata

A continuación, se incluyen los gastos derivados del uso de las instalaciones donde se ha llevado a cabo el proyecto, cargas fiscales, gastos financieros, tasas administrativas y derivados de las obligaciones de control del proyecto. De igual forma se incluye el beneficio industrial. Para cubrir estos gastos se establece un recargo del 22% sobre el importe del presupuesto de ejecución material.

CONCEPTO	PRECIO
22% coste total de ejecución material	10.313,60€

8.6 Honorarios facultativos

Se ha fijado en este proyecto un porcentaje del 7% sobre el coste total de ejecución por contrata.

CONCEPTO	PRECIO
7% coste total de ejecución material	3.281,60€

8.7 Presupuesto total

CONCEPTO	PRECIO
Presupuesto de ejecución material	46.880,00€
Importe de la ejecución por contrata	10.313,60€
Honorarios facultativos	3.281,60€
TOTAL (sin IVA)	60.475,20€
IVA (22%)	13.304,54€
TOTAL	73.779,74€

El presupuesto total del proyecto asciende a la cantidad de setenta y tres mil setecientos setenta y nueve euros con setenta y cuatro céntimos.

Alcalá de Henares a 21 de Septiembre de 2017.

Firmado: Javier León Almazán

Graduado en Ingeniería Electrónica de Comunicaciones

Manual de usuario

Actualmente existen varias distribuciones de ROS disponibles: ROS *JadeTurtle*, ROS *Indigo Igloo*, ROS *Hydro Medusa*, ROS *Groovy Galapagos*, etc.

Se recomienda instalar ROS Indigo, ya que junto a Jade es de los últimos que se desarrollaron y es el más estable. En este apartado se puede encontrar un manual de usuario que explica cómo instalar y configurar ROS para poder ejecutar los paquetes creados correctamente.

9.1 Configuración previa de Ubuntu

Antes de instalar ROS, es recomendable realizar una configuración previa de Ubuntu.

Para ello, se debe configurar el equipo para aceptar el software desde "packages.ros.org":

```
>> sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
$(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

9.2 Instalación de ROS Indigo

Para instalar ROS Indigo hay que seguir los siguientes pasos:

1) Se debe poner al día el índice de paquetes debían.

```
sudo apt-get update
```

2) Se instala la versión completa de ROS Indigo.

```
sudo apt-get install ros-indigo-desktop-full
```

3) Una vez instalado, hay que inicializar rosdep. Permite instalar fácilmente las dependencias del sistema que desea compilar y que se requieren para ejecutar algunos componentes básicos en ros.

```
sudo rosdep init
rosdep update
```

4) A continuación, se configuran las variables de entorno

```
echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

5) Por último, se recomienda instalar “rosinstall”. Esta herramienta permite descargar muchos árboles de código fuente para los paquetes ROS con un comando.

```
sudo apt-get install python-rosinstall
```

9.3 Configuración directorio de trabajo (ROS *workspace*)

Una vez instalado ROS, hay que configurar el directorio en el que se trabajará con ROS.

Antes de crear el directorio de trabajo, es recomendable añadir la siguiente línea en el fichero “.bashrc”:

```
source /opt/ros/indigo/setup.bash
```

El directorio de trabajo usado en ROS se conoce como “catkin_ws” y está formado por tres carpetas: /build , /devel y /src.

Para crear el directorio de trabajo se deben introducir los siguientes comandos en el terminal de Linux:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
```

Una vez creado, se compila con la orden “catkin_make”. Esta orden también se debe realizar cada vez que se crea o se añade un paquete al directorio de trabajo.

```
$ cd ~/catkin_ws/
$ catkin_make
```

9.4 Ejecutar un paquete de ROS.

Antes de ejecutar un paquete en ROS debemos configurar el directorio de trabajo como se ha explicado anteriormente en el apartado 9.3.

```
$ cd ~/catkin_ws/
$ source ./devel/setup.bash
```

Estas sentencias hay que introducirlas en cada sesión y se puede evitar, modificando el archivo ".bashcr". Para ello se tenemos que ejecutar el siguiente código:

```
$ gedit .bashrc
```

Ahora se abre una ventana con el código de ".bashrc" donde se debe colocar al final de todo el código las siguientes líneas:

```
source /opt/ros/indigo/setup.bash
source home/(usuario)/devel/setup.bash
```

Finalmente, para ejecutar un paquete en ROS se tiene que introducir el comando "rosrun" seguido del paquete y el script desde la carpeta "catkin_ws" como se muestra a continuación:

```
$ cd catkin_ws/
$ rosrun (paquete) (script).py
```

9.4.1 Ejecutar interfaz gráfica del TFG

Para lanzar nuestra interfaz gráfica en ROS y que se comunique con SARA se ha explicado en el capítulo 5 que se deben de ejecutar 2 scripts ("canusb.py" y "test.py"). Para ello se abrirán 3 terminales de Linux (con permisos sudo -s) ejecutando en cada terminal un nodo y en el siguiente orden: "roscore", "canusb.py" y "test.py"

Consola 1:

```
$ roscore
```

Consola 2:

```
$ cd catkin_ws/
$ rosrun canusb canusb.py
```

Consola 3:

```
$ cd catkin_ws/
$ rosrun canusb test.py
```

9.4.2 Leer mensajes de los nodos de SARA

Por último, para leer cualquier mensaje de un nodo de SARA se tienen que ejecutar las aplicaciones “canusb” y “lectura.py” del paquete “canusb” tal y como se ha descrito en el capítulo 4. En el siguiente código se muestran los pasos a seguir:

Consola 1:

```
$ roscore
```

Consola 2:

```
$ cd catkin_ws/
$ rosrun canusb canusb.py
```

Consola 3:

```
$ cd catkin_ws/
$ rosrun canusb lectura.py
```

Una vez ejecutados estos tres nodos, se pueden ver la lista de *topics* existentes y visualizar los valores de uno en concreto de la siguiente forma:

Consola 4:

```
$ rostopic list
$ rostopic echo /(nombre del topic)
```

Referencias

[1] D. Pinedo, Diseño de hardware/software para la navegacion reactiva de un sistema avanzado robotico de asistencia a la movilidad (SARA), UAH, 2015.

[2] D. L. Jaffe, "An ultrasonic head position interface for wheelchair control," *Journal of Medical Systems,* vol. 6, no. 4, pp. 337-342, 1982.

[3] H. A. Yanco, Initial Report on Wheelesley: A Robotic Wheelchair System, Canada, 1995.

[4] R. C. Simpson, S. P. Levine, D. A. Bell, L. A. Jaros, Y. Koren and J. Borenstein, "The NavChair assistive wheelchair navigation system," *IEEE Transactions on Rehabilitation Engineering ,* vol. 7, no. 4, pp. 443-451, 1999.

[5] M. Marzo and J. Garcia, "Experiences in assisted mobility: the SIAMO project," *International Conference on Control Applications,* vol. 2, pp. 766-771, 2002.

[6] J. C. Garcia, M. Marron, J. Ureña and D. Gualda, "Intelligent Wheelchairs: Filling the Gap between Labs and People," in *Assistive Technology: From Research to Practic*, vol. 33, 2013, pp. 202-209.

[7] E. B. A. Y. N. Morgan Quigley, "STAIR: Hardware and Software Architecture," *Robotics Workshop,* 2007.

[8] "ROS.org," [Online]. Available: http://wiki.ros.org/Source%20code. [Accessed 2016].

[9] "Python," [Online]. Available: https://www.python.org/. [Accessed 2016].

[10] "GTK," [Online]. Available: https://www.gtk.org/. [Accessed 2016].

[11] "Calcifer.org," 2016. [Online]. Available: http://calcifer.org/documentos/librognome/gtk.html.

[12] "Gnome developer," 2016. [Online]. Available: https://developer.gnome.org/gtk3/.

[13] Á. G. Morcillo, Beca de investigación SARA, Universidad de Alcalá, 2014.

[14] J. B. Álvarez, Guiado de una silla de ruedas mediante joystick y soplido, 2009.

[15] J. B. García, Guiado semiautomático de una silla de ruedas comandada por un, 2009.

[16] Spiralray, Abril 2016. [Online]. Available: https://github.com/spiralray.

[17] "The Python Standard Library," [Online]. Available: https://docs.python.org/2/library/struct.html. [Accessed 2017].

[18] E. S. Martínez, Guiado semiautomático de una silla de ruedas, UAH, 1999.