Proyecto Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Sistema de supervisión y telemetría para un robot móvil con pila de combustible

Autor: Pablo Bergmann Guerra
Tutor: Carlos Bordons Alba

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Proyecto Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Sistema de supervisión y telemetría para un robot móvil con pila de combustible

Autor:
Pablo Bergmann Guerra

Tutor:
Carlos Bordons Alba
Profesor catedrático

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Proyecto Fin de Grado: Sistema de supervisión y telemetría para un robot móvil con pila de combustible

Autor: Pablo Bergmann Guerra

Tutor: Carlos Bordons Alba

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A todas las personas que quiero

Agradecimientos

No parece aparentemente que un Grado en Ingeniería sea el estudio más indicados para una persona que realmente no sabe hacia dónde quiere enfocar su vida. Lo que nadie sabe al empezar unos estudios universitarios es la cantidad de cosas que pueden pasar y cómo van a cambiarle la vida para siempre esos años como estudiante. Conocerás compañeros brillantes, profesores maravillosos y por supuesto amigos para toda una vida. Es algo evidente, el cómo cursar estos estudios ha cambiado mi vida y habría sido imposible sin la ayuda de mi familia, mis amigos, mis compañeros y mis profesores.

A mi familia le agradezco todo el apoyo y cariño que han sabido darme durante toda mi vida. Entre todos han sabido crear una situación de paz y bienestar gracias a la cual solo he tenido que preocuparme por mis estudios durante estos años. Siempre interesados en mis estudios y aficiones. Muy entusiasmados y orgullosos de que por fin haya conseguido esta meta en mi vida.

A mis amigos les agradezco ser el punto de apoyo sobre el que sostenerme cuando los estudios me sobrepasan. No siempre es fácil el camino del estudiante pero con buenos amigos es posible desahogarse y seguir adelante.

A mis compañeros de clase por compartir su conocimiento y siempre estar dispuestos a ayudar a otros en lo que sea necesario. Sé que habría sido imposible finalizar el Grado sin unos compañeros tan generosos y un ambiente de hermandad tan fuerte como el que me enorgullezco de haber vivido estos años. Por otra parte también he de agradecer su tiempo a los compañeros de laboratorio del departamento ya que sin sus conocimientos no sé cómo habría salido adelante todo este trabajo.

Por último pero no menos importante agradecerle todo el tiempo dedicado, así como el interés y la ayuda que me ha estado proporcionando mi tutor Carlos Bordons en el último año de curso. Junto a su compañero de departamento Miguel Ángel Ridao ambos se han encargado de darme las directrices necesarias para llevar a cabo este proyecto.

Pablo Bergmann Guerra

Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Sevilla, 2018

Resumen

Este trabajo forma parte de un proyecto conjunto de la Universidad de Sevilla con el CSIRO (Commonwealth Scientific and Industrial Research Organization) y el INTA (Instituto Nacional de Técnica Aeroespacial). El proyecto está basado en la incorporación de pilas de combustible a robots para mejorar su autonomía por defecto con las baterías de litio $LiFePO_4$. El proyecto completo abarca el estudio de un robot submarino Bleeper Sport Underwater Robot y de un robot móvil terrestre Summit XL.

El trabajo realizado trata concretamente del diseño e instalación de un sistema de telemetría que haga posible la posterior incorporación de la pila de combustible para mejorar la duración de las baterías del Summit XL. Dicho sistema de telemetría, diseñado en este proyecto, consta de un sensor que medirá la intensidad que el robot demanda a las baterías, además esta medición será transmitida junto con otros datos de importancia, relativos a la pila de hidrógeno, vía WiFi a una estación de monitorización donde se usarán estos parámetros para hacer efectivo el control de la pila de combustible.

Abstract

THis project is a part of a bigger project developed by the University of Seville, the CSIRO (Commonwealth Scientific and Industrial Research Organization) and the INTA (National Institute of Aerospace Technology). The aim of such a project is to introduce the incorporation of fuel cells in robots as a method to improve their default ranges with lithium batteries. The whole work covers the investigation of a submarine robot (Bleeper Sport Underwater Robot) and a mobile land robot (Summit XL).

A significant aspect on this topic is also included in this document: the design and installation of a telemetry system which makes possible the successful work of the fuel cell, increasing the range of the Summit XL batteries. This system, which is designed in this project, has an electronic sensor that measures the intensity of the batteries, also this measure will be transmitted via WiFi with some electrical parameters of the fuel cell to an external station where all the parameters and measures will be used for do the fuel cell control.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN Y OBJETIVOS

El Proyecto "Improving efficiency and operational range in low-power unmanned vehicles through the use of hybrid fuel-cell power systems" (IUFCV) tiene como objetivo la evaluación de la viabilidad técnica de sistemas híbridos de potencia, basados en baterías y pilas de combustible, en aplicaciones reales de vehículos no tripulados y plataformas robóticas, comparando las prestaciones de estos sistemas de potencia con los utilizados actualmente, basados exclusivamente en baterías, en términos de disponibilidad de energía y potencia, duración de las misiones y rango operativo, peso y volumen, fiabilidad y vida útil, etc. Para ello se diseñarán e integrarán sistemas de estas características en tres plataformas existentes: un AUV (Autonomous Underwater Vehicle) y dos UGVs (Unmanned Ground Vehicles).

En este proyecto participan el grupo de robótica del organismo de I+D australiano CSIRO (Commonwealth Scientific and Industrial Research Organization), el Área de Energía del Instituto Nacional de Técnica Aeroespacial (INTA), a través de su Laboratorio de Energía en El Arenosillo (Huelva) y el Departamento de Ingeniería de Sistemas y Automática de la Universidad de Sevilla. INTA es responsable del diseño, desarrollo e integración del sistema de potencia basado en pila de combustible, incluyendo el almacenamiento y suministro de reactivos a estos sistemas, así como de su integración en uno de los vehículos terrestres no tripulados y su evaluación. La Universidad de Sevilla participará en el diseño e implementará el sistema de monitorización, control y gestión de energía a bordo de las plataformas, junto con INTA; en tanto que CSIRO es responsable del desarrollo de dos plataformas robóticas, una para un vehículo terrestre y otra para un vehículo submarino, de la integración final de todos los sistemas, incluyendo nuevos sensores y cargas útiles, y de la evaluación de estos vehículos no tripulados en condiciones reales de operación, de acuerdo a misiones diseñadas conjuntamente con diversos usuarios finales interesados en la propuesta.

Este proyecto está cofinanciado por el programa Science for Peace and Security (SPS) de la OTAN, enfocado a fomentar la cooperación científica y la investigación entre países pertenecientes a la Alianza y terceros países.

En lo que concierne a este Proyecto de Fin de Grado se tratará uno de los UGVs , modelo Summit XL, de la empresa Robotnik con la colaboración del Área de Energía del INTA. En el proyecto del INTA se pretende incrementar la autonomía de esta plataforma hasta 7 horas de uso nominal efectivo, manteniendo sus principales capacidades en cuanto a carga útil. Para ello se ha diseñado y ensayado un sistema híbrido de potencia basado en pila de combustible de polímero sólido (PEFC), con una potencia nominal de 200 W, de cátodo abierto y refrigerada por aire. El hidrógeno para estas pilas se almacenará en depósitos a presión o en hidruros metálicos, dependiendo de la disponibilidad y la logística para transportar hidrógeno a presión hasta el emplazamiento de la misión. En ambos casos se plantea una configuración "plug & play", de forma que los depósitos vacíos se reemplazarían en el lugar de uso por depósitos cargados de hidrógeno en las instalaciones del usuario final. El oxígeno necesario para la reacción electroquímica en la pila de combustible se toma directamente del aire ambiente.

El objetivo concreto del trabajo será el diseño y la instalación de un sistema capaz de medir la intensidad demandada por el robot a las baterías en tiempo real. Esto se llevará a cabo gracias a un sensor de intensidad, conectado a una placa Arduino. Además esta medición, junto con otros parámetros del sistema EMS(sistema cuyas funciones básicas son las propias de un sistema de protección y control de baterías de ion-litio (BMS), así como la gestión y control de la pila de combustible), instalado en el Summit junto a la pila de combustible, deberán ser enviados vía WiFi a una estación para ser monitorizados a tiempo real. Esto requerirá el diseño de un sistema de telemetría sobre el que correrá una capa ROS y el cual enviará vía WiFi todos los datos requeridos

para ser monitorizados. Finalmente, como aclaración, se diseñará un sistema que por una parte sea capaz de medir la intensidad demandada a las baterías y por otra parte adquiera todos los parámetros relacionados con la pila, provenientes del EMS. Dicho sistema enviará todos los datos adquiridos a una estación que monitorizará en tiempo real todos los parámetros necesarios.

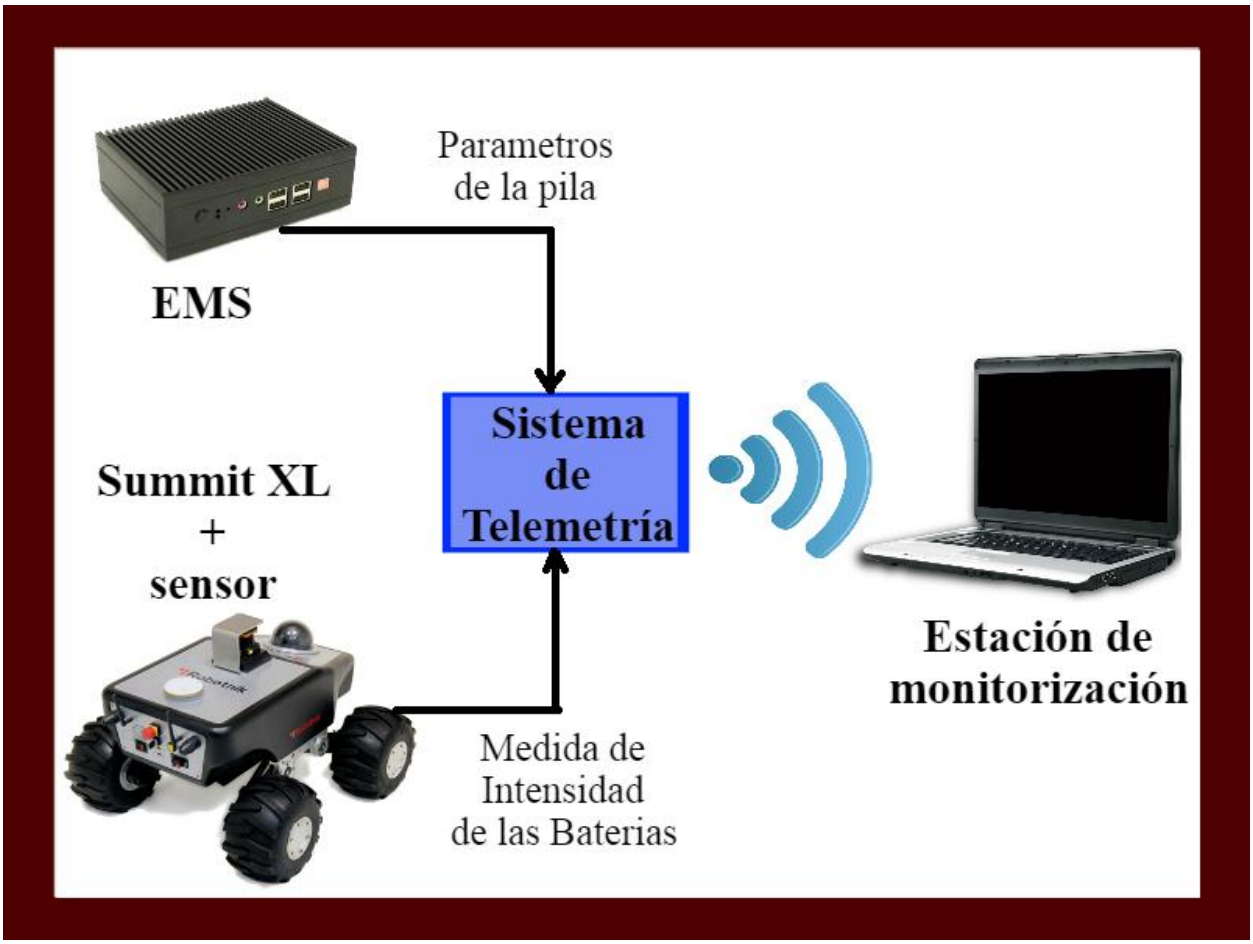


Figura 1-1. Esquema global del proyecto.

2 ENERGÍA Y ALIMENTACIÓN

El desarrollo principal del proyecto conjunto del CSIRO, el INTA y la US está basado en la incorporación de pilas de combustible sobre robots que trabajan con baterías de litio de manera que se pueda aumentar su autonomía mediante el uso de dichas pilas de hidrógeno. Concretamente este proyecto se centrará en el robot Summit XL por lo que se hará referencia a las especificaciones concretas de este robot y a su sistema de alimentación. Durante todo el proyecto ha habido un trabajo en conjunto con el INTA a la hora de recapitular toda la información necesaria para la realización del mismo.

2.1 Baterías de ion-litio

Las baterías de iones de litio (Li-ion) son dispositivos electroquímicos que acumulan carga eléctrica de forma reversible un limitado número de veces, perdiendo su capacidad en el proceso de carga y descarga de la batería. Emplean como electrolito una sal de litio que consigue los iones necesarios para causar una reacción electroquímica entre ánodo y cátodo.

En el caso del Summit XL se dispone de 8 baterías de 3.3V $LiFePO_4$ con las siguientes características:

Tabla 2-1. Características de las baterías

• **Marca:**	**Kokam**
• **Modelo:**	**Ultra High Energy NMC SLPB080085270 cell battery**
• **Configuración del pack:**	**6 celdas en serie**
• **Capacidad (Ah):**	**27**
• **Tension nominal (6S) (V):**	**21,69**
• **Peso del pack (6S) (kg):**	**2,37**
• **Energía especifica (Wh/kg):**	**247,10**

Actualmente y con estas baterías, de serie el Summit XL alimentan sus cuatro motores brushless de 250W y tiene una autonomía de cinco horas de funcionamiento y de unas veinte horas de uso estándar en laboratorio.

La idea de incorporar un sistema híbrido con una pila de combustible servirá para aumentar la autonomía del robot hasta 7 horas.

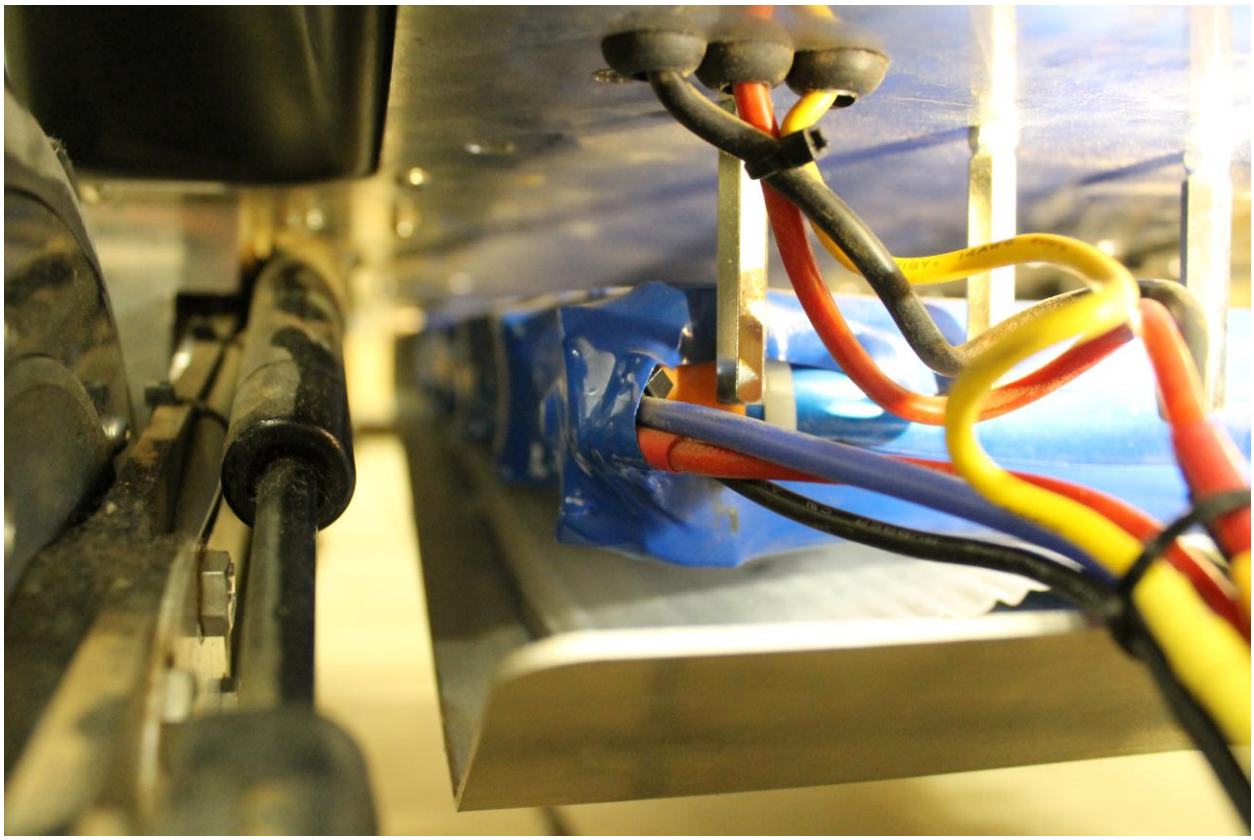


Figura 2-1. Batería del Summit XL

2.2 Pila de combustible

La pila de combustible es un dispositivo electroquímico dentro del cual se produce una reacción química controlada por la cual un flujo continuo de combustible y un oxidante dan lugar a una corriente eléctrica y a producto sobrante.

Este proceso electroquímico tiene una gran eficiencia energética, por encima de las máquinas térmicas y el proceso de obtención de energía de estas pilas al estar exentos de procesos térmicos o mecánicos tienen un impacto medioambiental mínimo. En contraposición la capacidad de almacenamiento de energía que posee una célula de combustible es nula en comparación con las baterías convencionales. Por estas razones se lanzan continuamente líneas de investigación de tecnologías híbridas entre motores térmicos o baterías con sistemas de pila de combustible.

En el caso del Summit XL, el INTA decidió instalar una pila con las siguientes características:

Tabla 2-2. Características de la Pila de hidrógeno

• **Marca:**	**BCH**
• **Modelo:**	**TH-200**
• **Potencia nominal:**	**200 W a 24 V**
• **Peso:**	**0,66 kg (pila con ventilador)**

Figura 2-2. Pila de combustible BCH 200W

2.3 Método de funcionamiento híbrido

Dependiendo del grado y tipo de acoplamiento entre las baterías y la pila de combustible, pueden considerarse tres configuraciones principales:

- Sistema inicialmente desacoplado (configuración 1). En esta configuración, el peso principal de la generación de energía eléctrica para el vehículo no tripulado recae sobre la pila de combustible, por lo que la autonomía vendrá dada sobre todo por el sistema de almacenamiento de hidrógeno. La batería se utiliza fundamentalmente para suministrar energía cuando la demanda de la carga es superior a la potencia que puede alcanzar la pila para el voltaje de operación del bus que alimenta a la carga.
- Sistema permanentemente acoplado (configuración 2). En esta configuración, el peso principal del suministro de energía eléctrica a la carga recae sobre las baterías, operando la pila de combustible como un sistema "range extender" que genera energía eléctrica cuando el estado de carga y la tensión de baterías alcanzan un determinado valor. A partir de ese momento, toda la energía generada por la pila de combustible se inyecta en el bus común de continua que conecta el sistema de potencia con la carga, de forma que parte de esta energía va a la carga y parte, dependiendo del balance de potencia, a las baterías.
- Sistema parcialmente acoplado (configuración 3). Esta configuración puede considerarse como un caso intermedio entre los dos casos anteriores. Su principal característica consiste en incluir la opción de carga de baterías, pero regulando la intensidad de carga destinada a las baterías.

A partir del análisis de los datos experimentales obtenidos de ensayar las configuraciones 1 y 2, se observa que los resultados obtenidos con la opción de recarga directa (configuración 2) son más favorables desde el punto de vista de estabilidad en la operación de la pila de combustible y en la tensión total del sistema, lo que puede redundar en una mayor durabilidad y eficiencia del sistema de potencia. Sin embargo, para garantizar una adecuada operación en estas condiciones, es preciso un cuidadoso diseño, selección y desarrollo conjunto de las baterías y la pila de combustible. Por tanto se ha optado por esta configuración (configuración 2, Figura 2-3) para el sistema híbrido pasivo con baterías y pila de combustible que se integrará en las dos plataformas terrestres del proyecto.

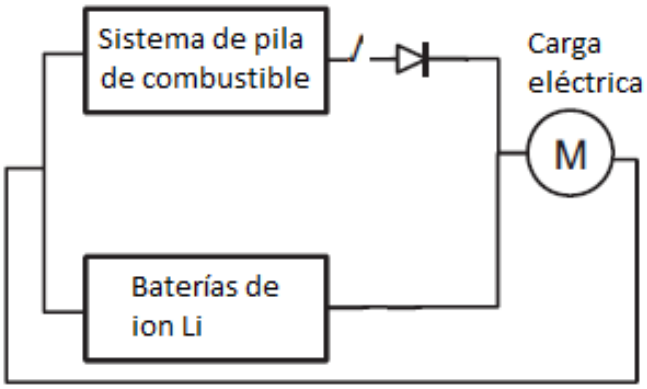


Figura 2-3. Esquema del sistema de potencia híbrido

Con motivo de esclarecer el sistema de potencia que se instalará en el Summit XL, el INTA puso a disposición de la US un gráfico explicativo del bus de potencia que regirá el nuevo diseño del sistema del robot.

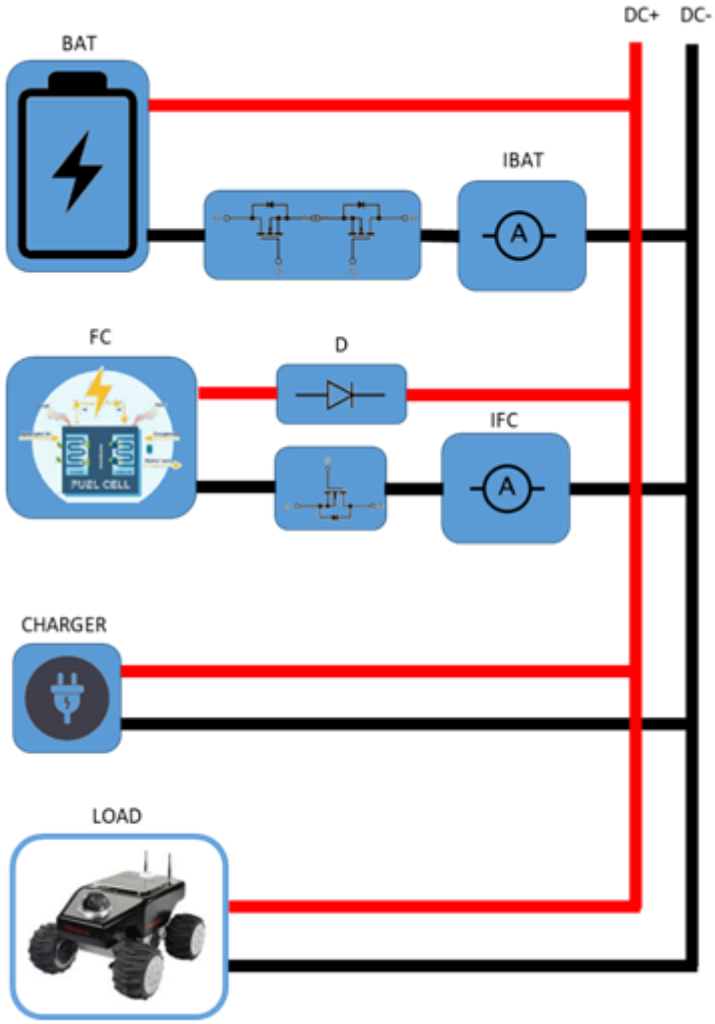


Figura 2-4 Bus de potencia del Summit XL

3 SISTEMA OPERATIVO Y SOFTWARE

En este capítulo se tratará de argumentar las múltiples ventajas que se obtienen al usar un sistema operativo en tiempo real como es Ubuntu. Además de estar más orientado a la programación que otros sistemas operativos, Ubuntu es el soporte sobre el cual trabaja la herramienta ROS. Hace pues, de vital importancia en este proyecto que en todo momento se trabaje con este sistema operativo ya que el robot móvil (Summit XL) trabaja con Ubuntu y ROS en su sistema empotrado.

Se tratará por último el software utilizado para el diseño del sistema de telemetría, se hablará de Arduino, su lenguaje Energía y cómo trabaja este microcontrolador con una capa ROS, de vital importancia para la funcionalidad IoT (Internet of Things – Concepto que engloba la conexión de dispositivos independientes que trabajan en conjunto comunicándose entre ellos) buscada en este proyecto.

3.1 Sistema Operativo Ubuntu 14.04LTS

Como se ha mencionado anteriormente es imprescindible para la compatibilidad con el AGV que todo el proyecto se efectúe bajo Ubuntu. En el caso concreto del Summit XL la versión escogida para el robot fue la 14.04LTS (Trusty Tahr). Las versiones LTS de Ubuntu salen al mercado cada 2 años de forma gratuita y tienen soporte técnico durante los siguientes 5 años. Es importante trabajar sobre Ubuntu, por otro lado, porque a priori es el sistema operativo sobre el que trabaja ROS.

Figura 3-1. Logos ROS y Ubuntu .

Se instaló una versión de escritorio de Ubuntu 14.04LTS sobre un portaril HP Pavilion g6, en el cual se ha realizado toda la programación relativa al proyecto, cedido por el Departamento de Ingeniería de Sistemas y Automática de la US. Se probó la compatibilidad en el conexionado vía WiFi con el computador empotrado del Summit XL que también portaba la misma versión.

Hubo un proceso de familiarización con el modo de trabajo por comando desde el terminal del Ubuntu. A diferencia de otros S.O., Ubuntu realiza muchas de sus acciones directamente desde el terminal.

3.2 Entorno ROS

3.2.1 Introducción a ROS

Robot Operative System (ROS) es un framework flexible que permite la escritura de software para robots. Es un conjunto de herramientas y librerías que tienen como objetivo simplificar la tarea de creación de sistemas

robustos y complejos de robótica. Este entorno posee un cierto nivel del abstracción de hardware, múltiples librerías herramientas de visualización, controladores, comunicación entre dispositivos, administración de paquetes y mucho más. Ha sido originalmente desarrollado por Willow Garage en Stanford y es un sistema que actualmente se encuentra bajo licencia open source.

3.2.2 ¿Por qué usar ROS?

En robótica la creación de un software robusto y para uso general a veces resulta realmente difícil. Los robots son objetos físicos y suelen tener que lidiar con el entorno que los rodea. Empezar un proyecto de esta envergadura desde cero sería sumamente costoso. ROS proporciona una forma de desarrollo de software abierto y colaborativa, de la que todos podemos aprovecharnos. En su plataforma existe un listado de todos los robots sobre los que se han realizado o se realizan actualmente investigaciones y software, esto permite descargar los paquetes de contenido de otras personas para no tener que empezar a programar desde cero y agilizar así el proceso de creación de software.

Existe, por lo tanto, una gran comunidad detrás de la expansión y utilización de ROS. En el caso del Summit XL, como veremos más adelante se aprovecharon los paquetes ya existentes en los repositorios de GitHub de la comunidad ROS para adelantar parte del proceso de hacer funcionar el robot.

3.2.3 Estructura y funcionamiento de ROS

El funcionamiento de ROS se basa en la ejecución en paralelo de varios nodos. Estos nodos cuentan con sockets TCP para comunicarse entre sí, dentro del mismo dispositivo o entre dispositivos diferentes conectados mediante una red WiFi. Estos nodos se comunican entre ellos por medio de topics, que podría decirse que son como "temas de conversación".

El ejemplo más claro serían dos nodos o procesos que desean comunicarse entre sí, habrá un nodo emisor y un nodo receptor. Bien, el nodo emisor o talker publica un topic y el nodo receptor o listener lee el topic que está publicando el emisor. La forma más clara de entender cómo interactúan los nodos ROS es a través de un gráfico proporcionado por la misma herramienta. En la siguiente figura se aprecia gráficamente el ejemplo expuesto. En ella podemos observar los procesos (cuadrados listener y talker) que están interactuando mediante el intercambio de un topic (cuadrado /chatter).

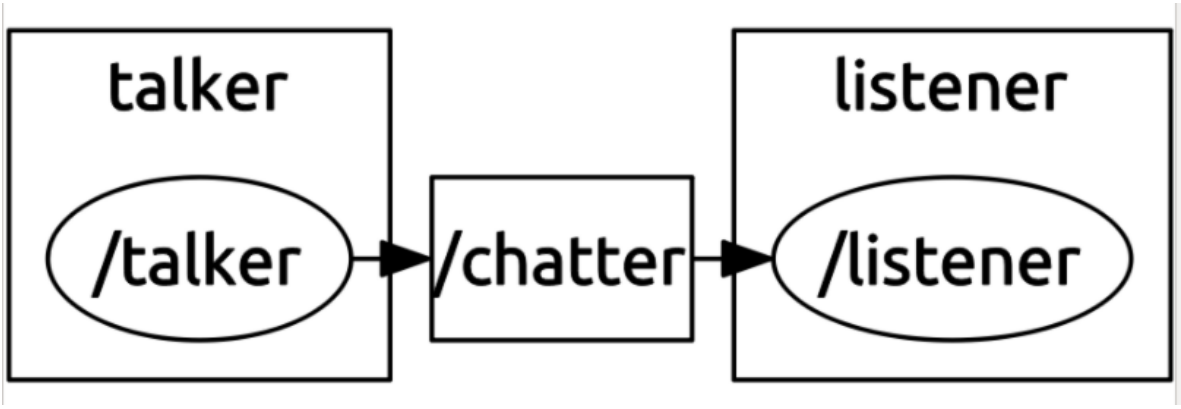


Figura 3-2. Grafo chatter ROS

ROS puede ejecutarse tanto en una computadora de sobremesa como en el sistema embebido de un robot. La forma de funcionar de esta herramienta, al estar basada en nodos independientes que trabajan en paralelo, solo necesita una conexión WiFi o Ethernet para comunicar los distintos nodos de todos los dispositivos implicados en el experimento.

En el caso del Summit XL existen múltiples nodos ejecutándose en paralelo en ROS y corriendo en el sistema empotrado que incorpora el robot. Estos nodos se encargan de manejar todos los dispositivos internos que tiene el robot y además mandar información de todo ello a una estación base para monitorizarlos. Como ejemplo se puede ver a continuación el gráfico de ROS del sistema Summit XL. Como en el caso anterior más sencillo, sabemos que cada uno de los bloques cuadrados representa un proceso que se está llevando a cabo en

el entorno ROS, en cualquiera de los dispositivos conectados, y que cada uno de los arcos está indicando que nodos están comunicándose entre sí y compartiendo alguna información por medio de topics.

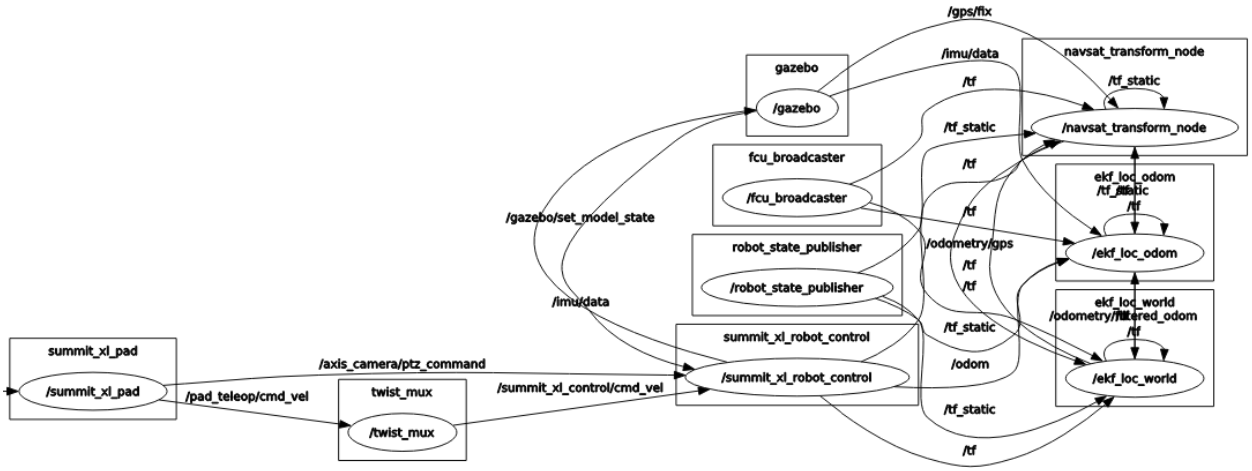


Figura 3-3. Grafo Summit XL

3.2.4 ROS desde el terminal

La parte más difícil de la utilización de la herramienta ROS es, sin duda, el hecho de que no tiene interfaz gráfica. Siguiendo un poco la tradición de Ubuntu, ROS necesita ser utilizado desde el terminal del sistema para la realización de cualquier acción. Cada uno de los procesos que se lancen necesitará una nueva ventana de terminal para poder ejecutarse en paralelo con lo que es recomendable el uso de alguna herramienta de organización del terminal para Linux.

Se necesitará el Shell de Linux como interfaz para utilizar el sistema ROS. Existen una serie de comandos gracias a los cuales, desde el terminal de Ubuntu podremos realizar ciertas acciones con ROS:

- **roscd**: cambiar de directorio a un paquete o pila (ej. *roscd stage*)
- **roscore**: Es necesario ejecutarlo en primera instancia para dar paso al soporte de ejecución del sistema completo de ROS. Siempre tiene que estar ejecutándose para permitir los nodos se comuniquen. Permite ejecutarse en cualquiera de los dispositivos conectados y que estén corriendo ROS (ej. *roscore* o *roscore -p 1234*)
- **roscreate-pkg**: crea e inicializa un paquete. Se tiene que ejecutar desde uno de los directorios válidos para que contengan paquetes. El formato de ejecución es: roscreate-pkg paquete [depen1 ...] donde depen1 es una dependencia. Por ejemplo, si el paquete que estamos creando va a usar los mensajes estándar y va a usar código c++, debemos indicar las dependencias std_msgs y roscpp.
- **rosnode**: Proporciona información sobre algún nodo y nos permite realizar las siguientes acciones:
    - rosnode info nodo (aporta información sobre el nodo)
    - rosnode kill nodo (deja de ejecutar el nodo)
    - rosnode list (muestra la lista de todos los nodos en ejecución)
    - rosnode machine dispositivo1 (Muestra todos los nodos que se están ejecutando concretamente en el dispositivo1).
    - rosnode ping nodo (comprueba la conectividad del nodo).
- **rosrun**: permite ejecutar cualquier aplicación de un paquete sin necesidad de cambiar a su directorio. Es la acción por defecto para lanzar los nodos del sistema ROS para trabajar con él.
- **rostopic**: permite obtener información sobre un topic. Y tiene las siguientes opciones:

- rostopic bw (muestra el ancho de banda consumido por un topic)
- rostopic echo (imprime datos del topic por la salida estándar)
- rostopic find (encuentra un topic)
- rostopic info (imprime información de un topic)
- rostopic list (Crea una lista con todos los topics activos)
- rostopic pub (publica datos a un topic activo)
- rostopic type (imprime el tipo de información de un topic)

- **roswtf**: permite comprobar si hay algún error. Ejecutamos roscd y después roswtf.

En la figura 2-4 podemos ver un ejemplo de comando ROS en la consola de Ubuntu, es este caso se ha introducido la acción "rostopic list" en el Shell y el sistema ha respondido con todos los topics activos que el robot estaba publicando en ese momento.

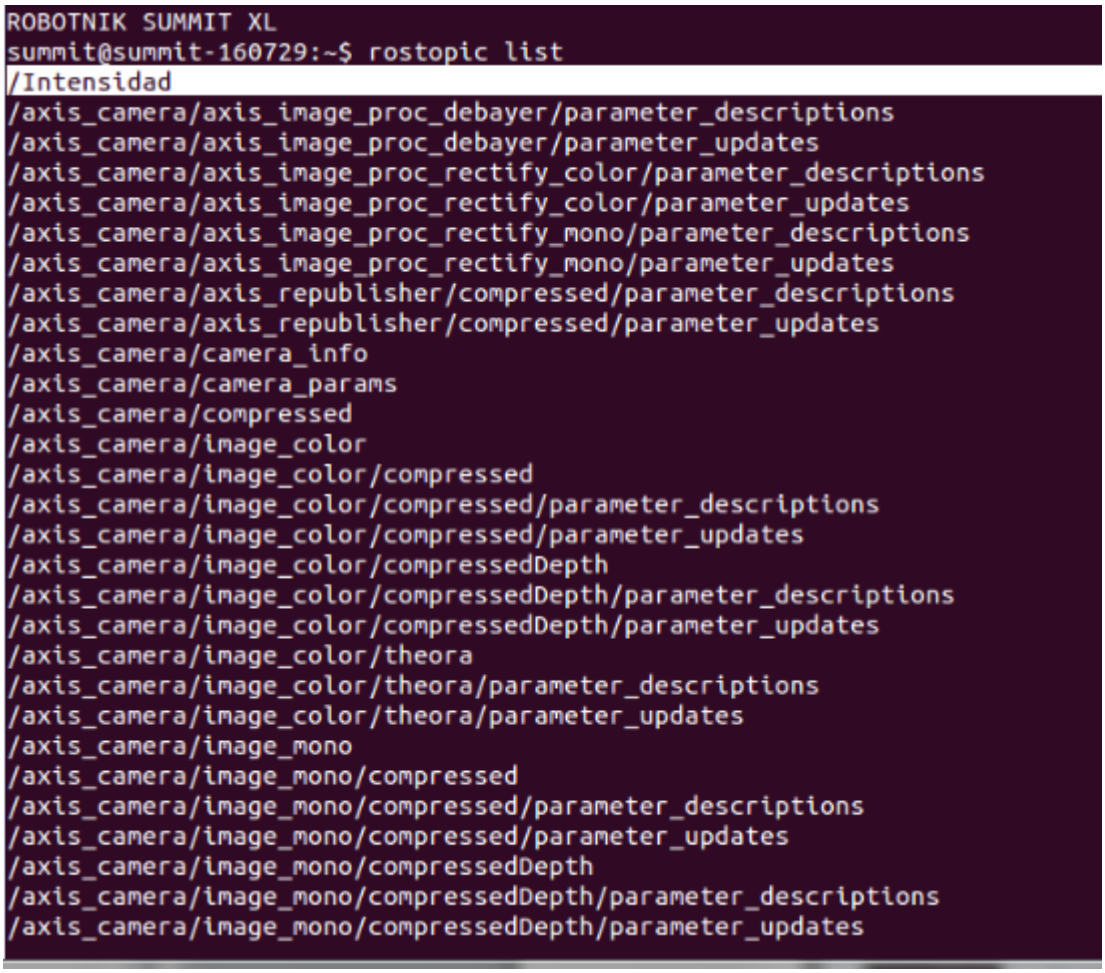


Figura 3-4. Shell Linux y uso de ROS

3.2.5 Útiles de ROS: Gazebo y Rviz

Gazebo y Rviz son dos de las herramientas más importantes incluidas en ROS que aparecen como paquetes en las librerías por defecto cuando descargamos ROS y podemos ejecutarlas a través del comando rosrun en la consola de comandos.

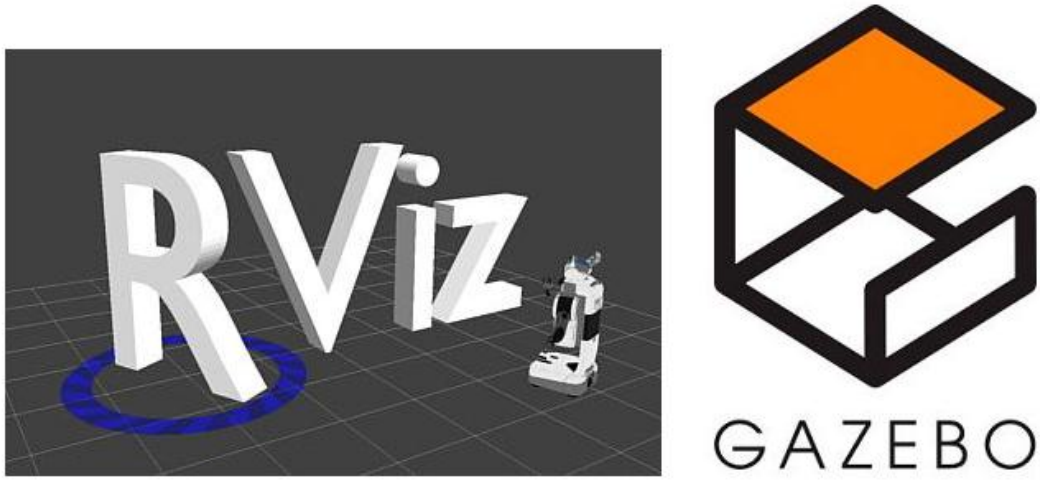


Figura 3-5. Logo Rviz y Gazebo

Gazebo es una herramienta de simulación muy potente que nos permite crear un entorno virtual para realizar una simulación completa con nuestro robot. En el caso del Summit XL existe un modelo diseñado en gazebo específico que permite recrear simulaciones con gran exactitud y teniendo en cuenta los parámetros físicos del robot. Existen a su vez entornos previamente creados por la comunidad ROS que permiten agilizar el proceso de modelado, no teniendo que crear de cero ni el entorno de simulación ni el modelo físico del robot y sus restricciones holonómicas que ya son tenidas en cuenta en el modelo creado por la comunidad.

En la figura que se observa a continuación se ha simulado al Summit XL sobre un entorno predeterminado de muestra de Gazebo.

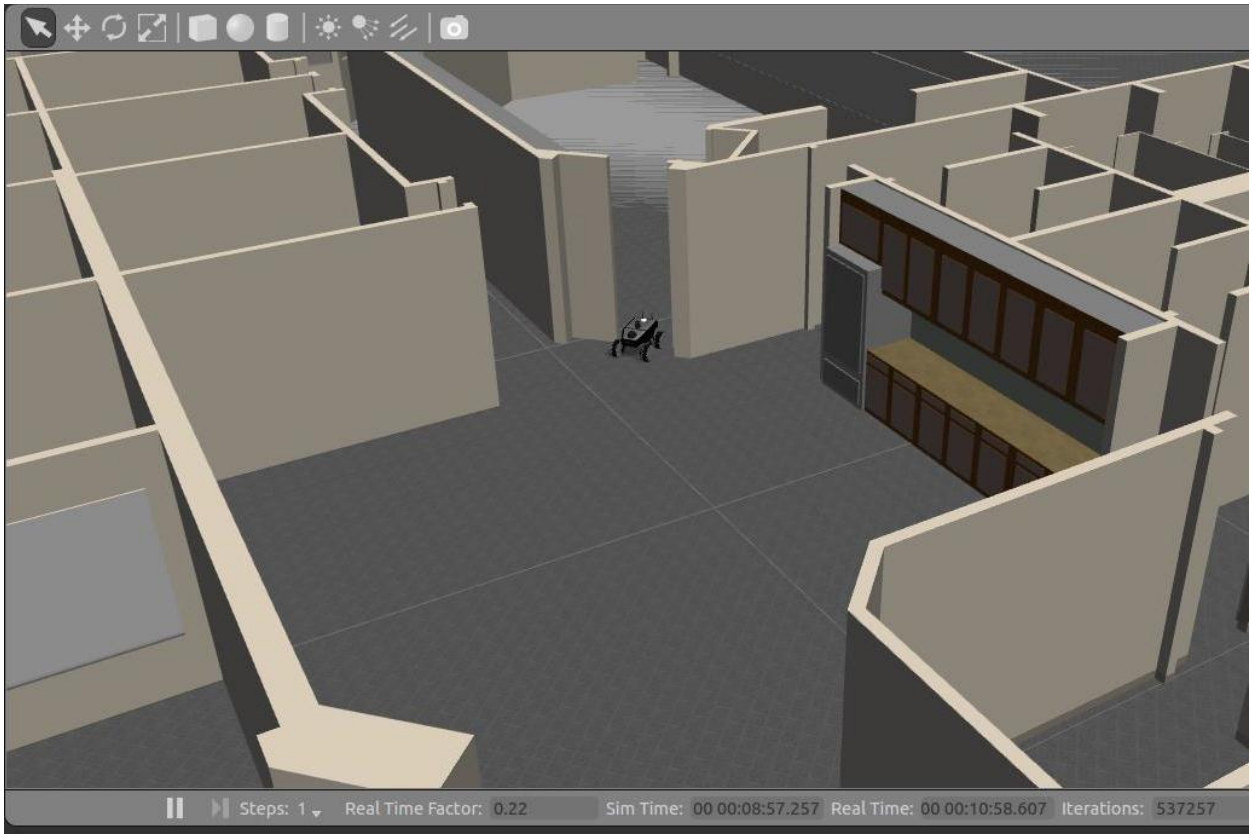


Figura 3-6. Simulación Summit XL en Gazebo

Realmente para la realización del proyecto no fue necesaria la simulación y la utilización de Gazebo en demasiadas ocasiones debido a que se dispuso del robot real durante gran parte de la realización de las pruebas.

La segunda herramienta utilizada en el proyecto fue Rviz, utilizada en conjunto con ROS permite observar en tiempo real la información publicada en los diferentes topics. Rviz posee un software que permite interpretar los topics de los diferentes robots y dispone de una interfaz gráfica para interactuar añadiendo o eliminando los topics que son o no interesantes en cada momento. En el caso del Summit XL se pueden visualizar multitud de topics con la herramienta Rviz y además es posible visualizar en directo la imagen de la cámara del robot y la lectura del sensor para la creación de un mapa virtual del entorno.

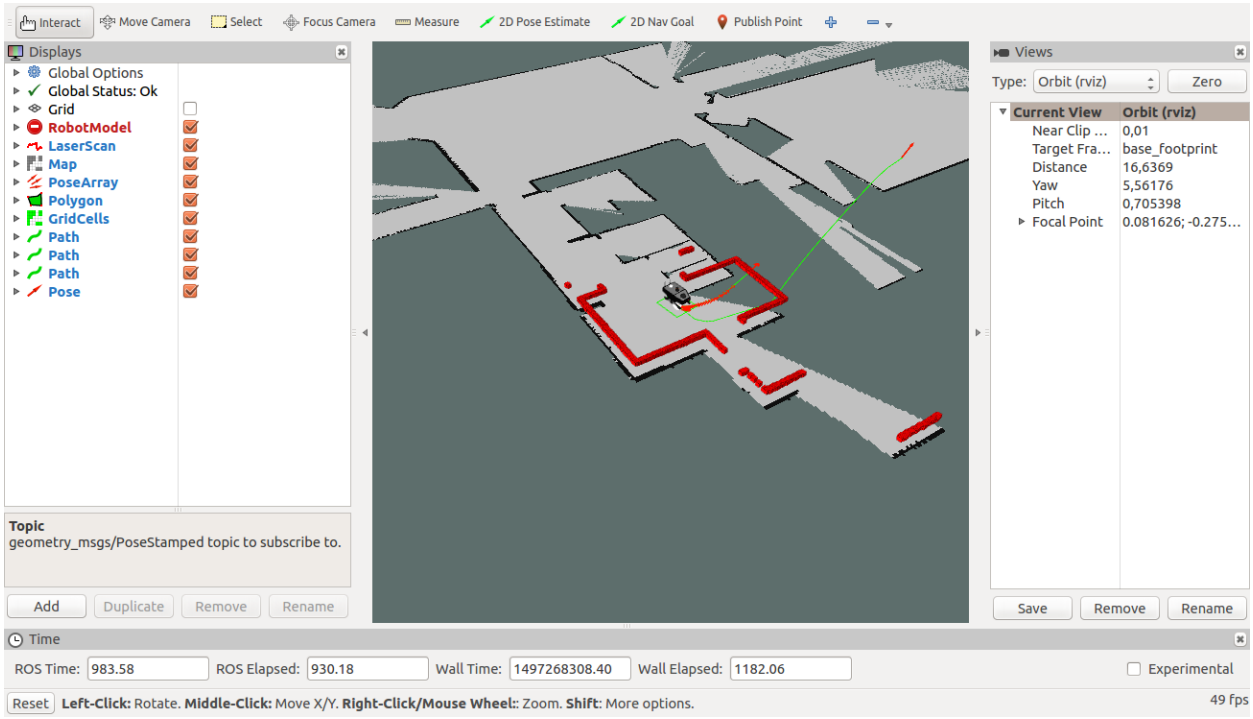


Figura 3-7. Visualización Summit XL en Rviz

3.3 Electrónica con Arduino

Es importante dedicar un apartado en la memoria de este proyecto a explicar el motivo por el cual se escogió Arduino como hardware para el microcontrolador del sistema de telemetría que se realizó en el proyecto. Realmente el trabajo de la placa de Arduino en este proyecto es leer por una entrada analógica la lectura del sensor de intensidad, por una entrada digital como puerto serie comunicarse con el EMS y por último por un segundo puerto serie comunicarse con el ESP8266 para poder transmitir vía WiFi todos los parámetros necesarios para ser monitorizados por una computadora.

Figura 3-8. Logo Arduino

3.3.1 Introducción a Arduino

Arduino es una plataforma de creación de prototipado de código abierto. Está basada en una arquitectura hardware y software libre y de fácil utilización. Las placas de Arduino contienen un microcontrolador, periféricos hardware, puertos de entrada y salida analógicos y digitales y los distintos componentes según el modelo de Arduino a adquirir. Es una gran herramienta para la creación de proyectos de electrónica y robótica y cuenta con una gran comunidad activa de diseñadores que suben contenidos y nuevos proyectos diarios. Este último concepto permite, al igual que la comunidad ROS, no tener que empezar a escribir un software de cero y, en cambio, poder basar una investigación en software ya creado.

Este opensource también consta de un software gratuito que se puede descargar desde la página web de Arduino. El Arduino IDE es una herramienta de programación para cargar sketches (Códigos fuente para Arduino) a cualquier placa de la marca.

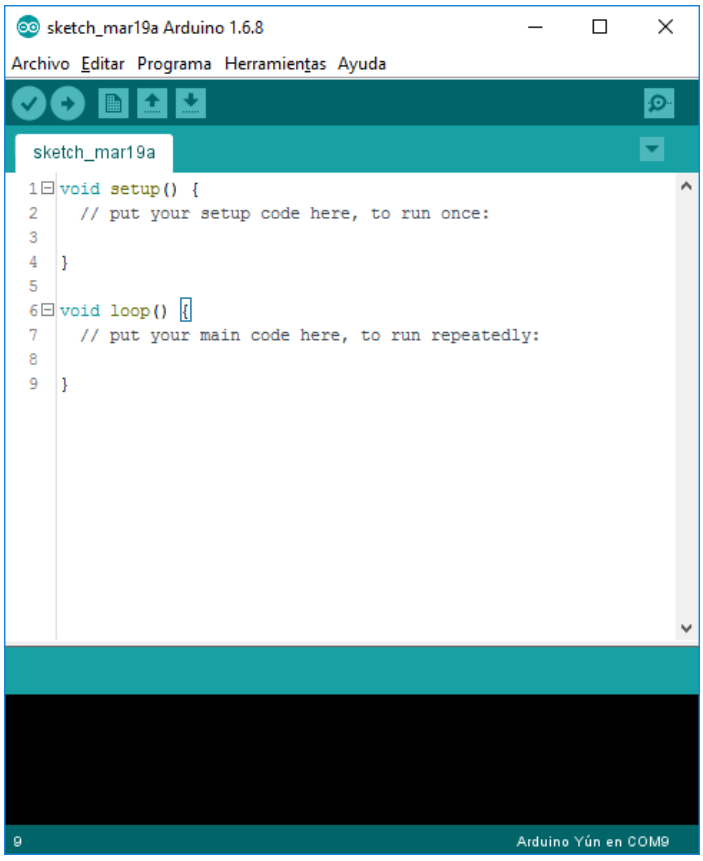


Figura 3-9. Arduino IDE

En el presente proyecto se ha optado por la utilización del Arduino UNO, debido a su disponibilidad en el Departamento de Ingeniería de Sistemas y Automática y a que realmente no era necesaria una placa con mayor número de recursos para el objetivo a desempeñar por el Arduino.

Figura 3-10. Arduino UNO

En esencia el Arduino UNO es una placa basada en un micro Atmel ATmega328 con una serie de componentes integrados para su fácil e intuitiva utilización:

- Botón de Reset. Sirve para reiniciar el programa cargado en la placa desde Arduino IDE
- Puertos de entrada y salida digitales. Pines con la finalidad de conectar sensores y actuadores para interactuar con ellos mediante señales digitales. Seis de ellos con opción PWM.
- Puerto USB. Es en principio la alimentación del Arduino y tienen la función de cargar los programas así como de comunicarse por protocolo serie con el ordenador.
- Reloj oscilador. Mantiene el flujo de ejecución de cualquier programa para la placa.
- Puertos de energía. Son los encargados de dar las salidas de 3.3V y 5V características de Arduino para alimentar los periféricos así como de alimentar por Vin al Arduino el caso de no poder hacerlo mediante el USB.
- Puertos analógicos. Cinco puertos para lectura de señales analógicas proveniente de sensores y actuadores externos.
- Regulador de tensión. Circuito que protege a la placa de sobretensiones y se encarga de la salida correcta de voltaje por los pines en cada momento.

3.3.2 Arduino con capa ROS

Al ser Arduino una gran plataforma de software libre orientada a la investigación en electrónica y ROS por su parte ser una plataforma para la investigación en robótica, existe una creciente ola de trabajos relativos a la introducción de placas Arduino en sistemas robóticos controlados por ROS. Para llevar a cabo esta tarea es necesario programar nuestro Arduino para que ejecute uno o varios nodos ROS y pueda formar parte del sistema robótico deseado.

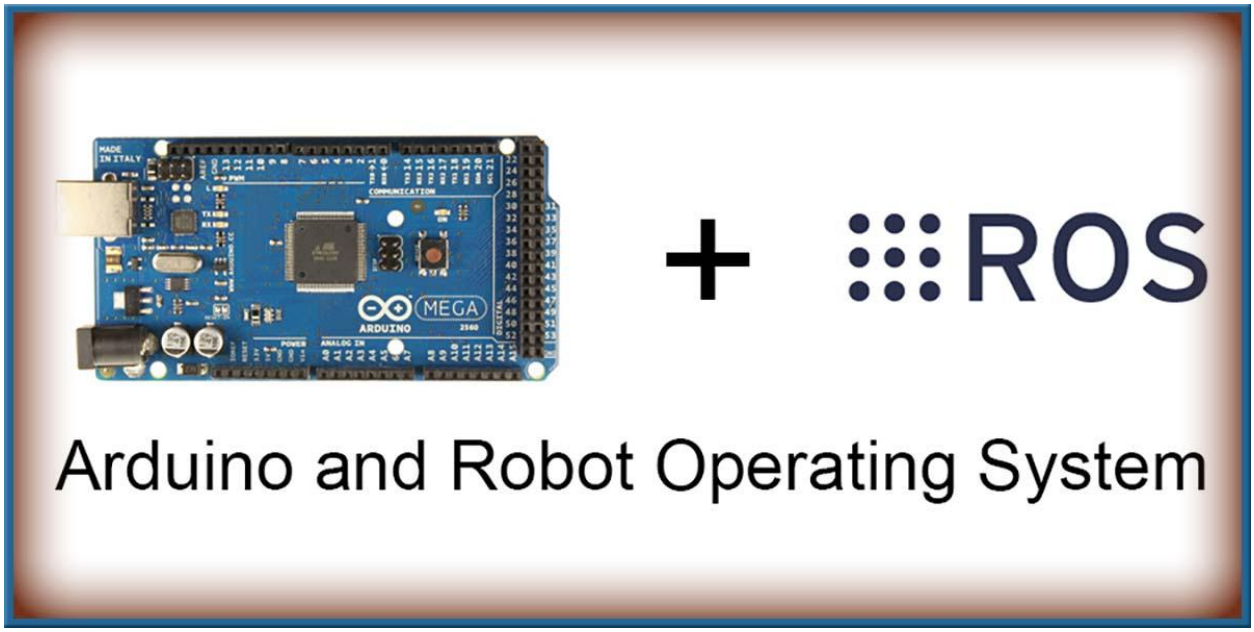


Figura 3-11. Arduino y ROS

Por parte de ROS existe un paquete de código llamado rosserial_arduino y que se puede descargar de manera gratuita desde la página web oficial de ROS. Este paquete permite lanzar programas que ejecuten un nodo en el sistema ROS que sea capaz de comunicarse con Arduino mediante protocolo serie en el caso de estar conectado por USB o mediante protocolo TCP IP en el caso de estar conectado vía Wifi.

En el caso de Arduino la librería rosserial se encuentra en GitHub por parte de la comunidad ROS. Necesitaremos añadir las librerías en nuestro código fuente a la hora de codificar nuestro Arduino para poder correr la capa ROS sobre la UART de la placa Arduino. Esto permitirá a Arduino ser un nodo del sistema ROS y poder subscribirse o publicar topics propios, además integrado en esta librería está la funcionalidad de adquirir por parte de Arduino los tiempos del sistema de ROS para poder sincronizar de manera perfecta el nodo que corre en Arduino con los demás nodos del sistema ROS que corren en otros dispositivos o en un ordenador.

4 ROBOT MÓVIL SUMMIT XL

En el presente capítulo introduciremos al Robot Summit XL, robot sobre el cual se ha trabajado en todo momento para la realización de este Trabajo de Fin de Grado. Se intentara dar una información general del robot móvil y se profundizará en los aspectos técnicos del Summit XL como robot autónomo.

4.1 Introducción a los robots autónomos

"Los robots autónomos son máquinas inteligentes capaces de realizar tareas en su entorno, sin un control explícito de los humanos. Dichos robots tienen que ser capaces de desenvolverse en entornos completamente desconocidos. Para ello deben coordinar todos sus sensores de forma que puedan orientarse, moverse (si tienen capacidad para ello) y poder interactuar con su entorno para realizar la tarea que tienen asignada." (2002-09 Unidad de Desarrollo Tecnológico en Inteligencia Artificial, IIIA-CSIC).

Estas máquinas debes cumplir una serie de principios para considerarse autónomas:

Figura 4-1 Robot autónomo de Boston Dynamics

- Captar información del medio que les rodea (1ª regla)
- Ser capaces de realizar su tarea sin intervención de los humanos (2ª regla)
- Capacidad de moverse total o parcialmente por su entorno operativo sin ayuda humana (3ª regla)
- No dañar a personas o seres vivos a no ser que su diseño lo exija expresamente (4ª regla)
- Posibilidad de aprendizaje para optimización de su tarea o mejora de interpretación del entorno.

4.2 Descripcion del Summit XL

La plataforma móvil Summit XL es un robot móvil autónomo diseñado y distribuido por la compañía Robotnik. Es el dispositivo elegido por el INTA y la US para la instalación de la pila de combustible de hidrógeno que da sentido a este proyecto. El modelo del que se dispone en el laboratorio consta del cuerpo del robot que describiremos en los siguientes sub-apartados y de cuatro ruedas de goma con configuración skid-steering. El robot pesa unos cuarenta y cinco kilos y puede cargar unos veinte kilos más a una velocidad máxima de tres metros por segundo. Actualmente goza de una autonomía de 5 horas gracias a sus pilas Li-ion y el rango de temperaturas de funcionamiento es de cero a cincuenta grados centígrados.

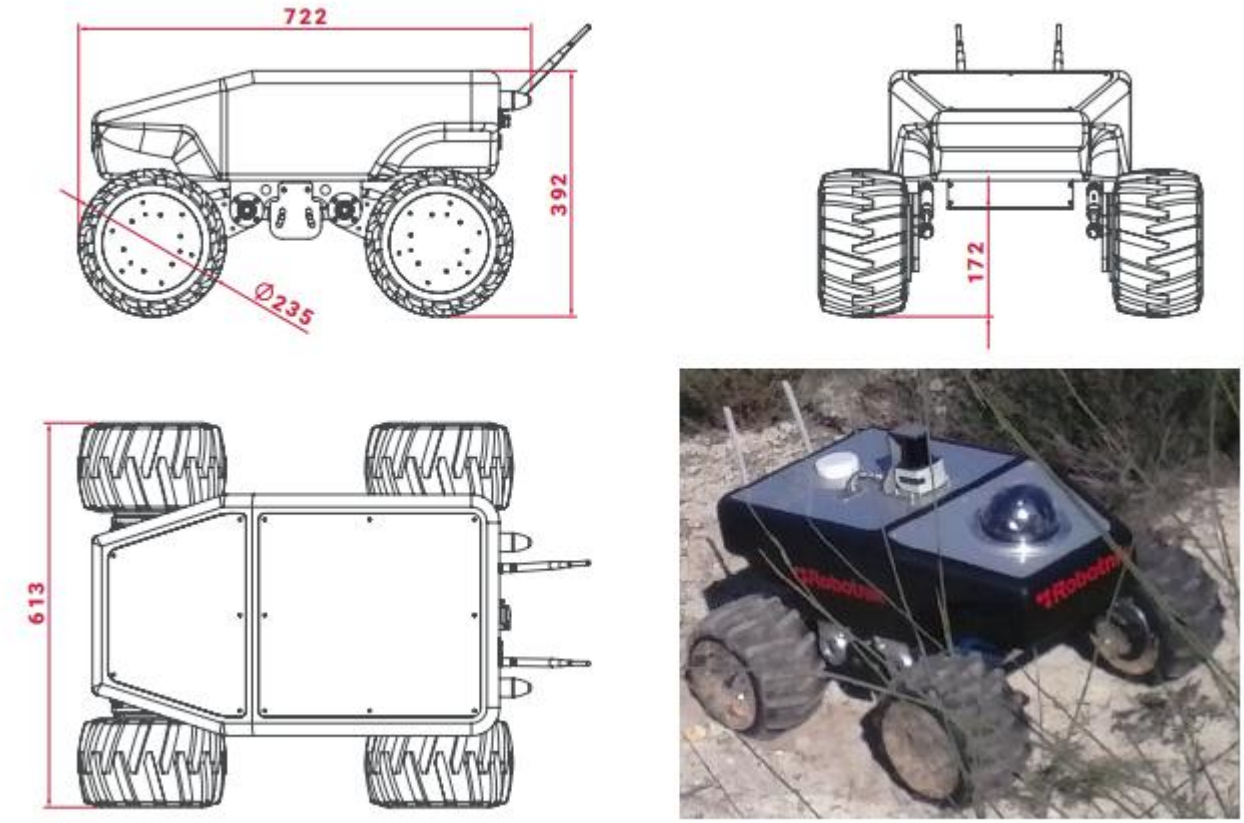


Figura 4-2 Summit XL

4.2.1 Características hardware

Tabla 4-1. Hardware Summit XL

Motores	**brushless 4x250W 8 polos**
Driver motor	**DZCANTE 020L080 y placa MC1DZC**
Tamaño	**722x613x392 mm**
Peso	**45 Kg**
Velocidad	**3 m/s**
Cámara	**Axis Camara (AXIS p5514 PTZ Dome Network Camera)**

Sensor	**Stick laser range finder, amplitude 270º, rango 10m**
Placa base	**Mitac PD10B1 MT con Quad core Intel Bay Trail J1900**
Autopiloto	**Pixhawk FPU PX4 (giroscopio y acelerómetro)**
Controlador	**Ps3 Bluetooth remote controller**
Batería	**8x3.2V $LiFePO_4$ Kokam batteries**
Router	**Router Belkin N300 Wi-Fi N**

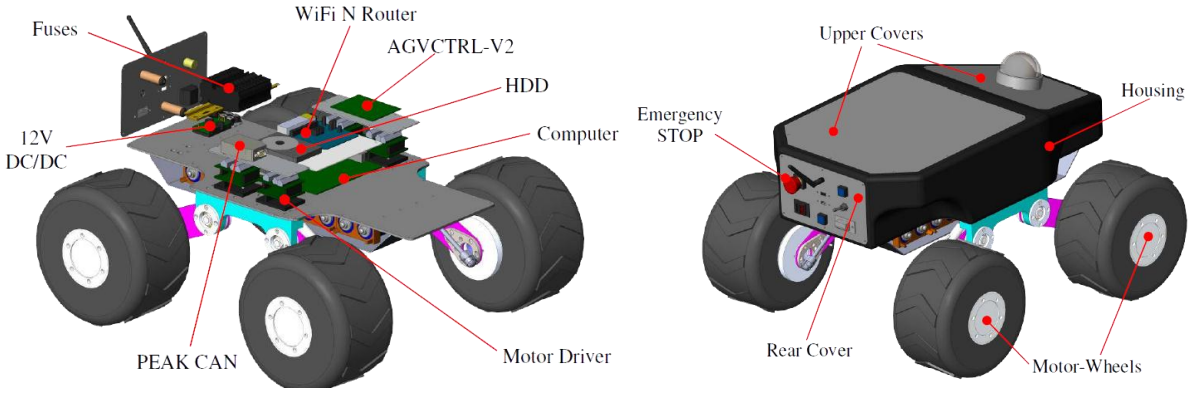


Figura 4-3 Esquema Summit XL

4.2.2 Características software

Como se ha mencionado con anterioridad en el proyecto el Summit XL incorpora un ordenador empotrado en el cual corre el sistema operativo Ubuntu 14.04 LTS de Linux. En dicha computadora se encuentra instalado ROS en si versión con soporte hasta 2019 ROS Indigo. Por motivos de compatibilidad se decidió instalar el mismo S.O. y la misma versión de ROS en la computadora que fue proporcionada por el Departamento de Ingeniería de Sistemas y Automática con el fin de monitorizar los datos transmitidos por el sistema diseñado de telemetría que también trabaja sobre una capa ROS para optimizar la compatibilidad.

El Summit XL tiene ya instalados todos los paquetes necesarios para su utilización mediante ROS y los mismos paquetes fueron instalados en la computadora HP Pavilion g6 donde se monitorizará todo el sistema ROS.

Es importante destacar que al no ser una tarea trivial el instalar y configurar el sistema ROS desde cero en un PC, se ha decidido incluir un apéndice en este proyecto para futuros alumnos que deseen continuar trabajando con el robot. Por lo tanto en el **Apéndice A** se ha recopilado toda la información necesaria y se ha creado un tutorial sencillo para empezar desde cero a investigar con el robot sin invertir tiempo en entender ROS o buscar información acerca del robot móvil.

4.3 Conexiones inalámbricas con el robot

El Summit XL lleva incorporado su propio router WiFi de banda N, que a priori para este proyecto no necesitaría conexión a internet, si no solo a nivel local. La idea principal es conectar todos los dispositivos que están funcionando en paralelo a la misma red WiFi, la red del router del robot, para que todos trabajen como nodos en el mismo sistema ROS. Este concepto está muy de moda en los últimos años y es muy aplicable al mundo de la robótica, hablamos del IoT (Internet of Things). En el **Apéndice A** de este proyecto está

perfectamente definido y explicado todos los pasos a seguir para configurar desde el Summit XL y el router que lleva a bordo las nuevas direcciones IP de cada uno de los dispositivos que estarán conectados a la red y que el sistema ROS tendrá que reconocer para funcionar todo al unísono.

Como el S.O. Ubuntu que corre el sistema empotrado en el Summit XL carece de pantalla para poder interactuar con él es necesario acceder al sistema del robot desde un computador auxiliar conectado a su red WiFi, y a su vez este ordenador también deberá estar corriendo la misma versión de Ubuntu. Desde la consola de comandos del portátil con Ubuntu 14.04 LTS accedemos a la sesión que está corriendo dentro del robot como se aprecia en la figura 4-4.

Figura 4-4 Acceso a la consola del Summit XL

Una vez dentro de la sesión del Summit podremos comprobar todos los archivos que contiene el robot: el sistema ROS instalado, todos los paquetes que usa para navegación autónoma y los controladores para los diferentes dispositivos con los que interacciona como la cámara, el radar o el mando.

Como medida de precaución y ya que parte de la idea de este trabajo es ser continuado por otros alumnos en los siguientes años es necesario hacer un especial hincapié en esta parte del proyecto. Cuando se inicia sesión con ssh en summit@summit, se le está pidiendo a Ubuntu, a través de la consola de comando, acceso al interior del robot. Es importante recordar que existe un trabajo previo de alumnos pasados y por parte del INTA sobre el robot móvil: Instalación de paquetes, configuración de parámetros, etc. Se podría modificar por error cualquier pequeño parámetro a través de la sesión del Summit que perjudicase el avance del proyecto con lo cual a modo de aviso para próximos usuarios, es necesario asegurarse antes de modificar el robot.

5 MEDICIÓN DE INTENSIDAD EN LAS BATERÍAS

Este es el capítulo en el cual se hablará del proceso de elección, diseño e instalación del sistema de medida que da sentido a este proyecto. En este capítulo solo se tendrá en cuenta la parte que compete a la medida de la intensidad de la batería del Summit XL no a la telemetría que permite enviar esta información para que sea de utilidad para el resto del sistema.

5.1 Estudio y elección de sensor

Para el desempeño de esta tarea es primordial conocer cuál es la corriente máxima que el sistema demanda a las baterías, para ello se llevó a cabo un estudio exhaustivo de los manuales del robot Summit XL. A priori la mejor idea parecía escoger un sensor que cubriese un rango de medidas incluso mayor que el rango de intensidades sobre el que trabaja el Summit XL. Dentro de los manuales se habla de la existencia de un circuito de protección contra sobre intensidad a partir de 60A, es decir, la intensidad que demanda el Summit jamás llegará a esta cantidad. El método más rápido para saber el orden de magnitud sobre el que trabaja el robot era a través de una pinza amperimétrica. Como se puede apreciar en la Figura 5-1 se utilizó este utensilio de medida para poder saber la intensidad máxima que se demanda a la batería en condiciones normales de utilización gracias al modo "peak mode" de este modelo de pinza que te detecta el pico máximo en amperios tras la realización de una prueba.

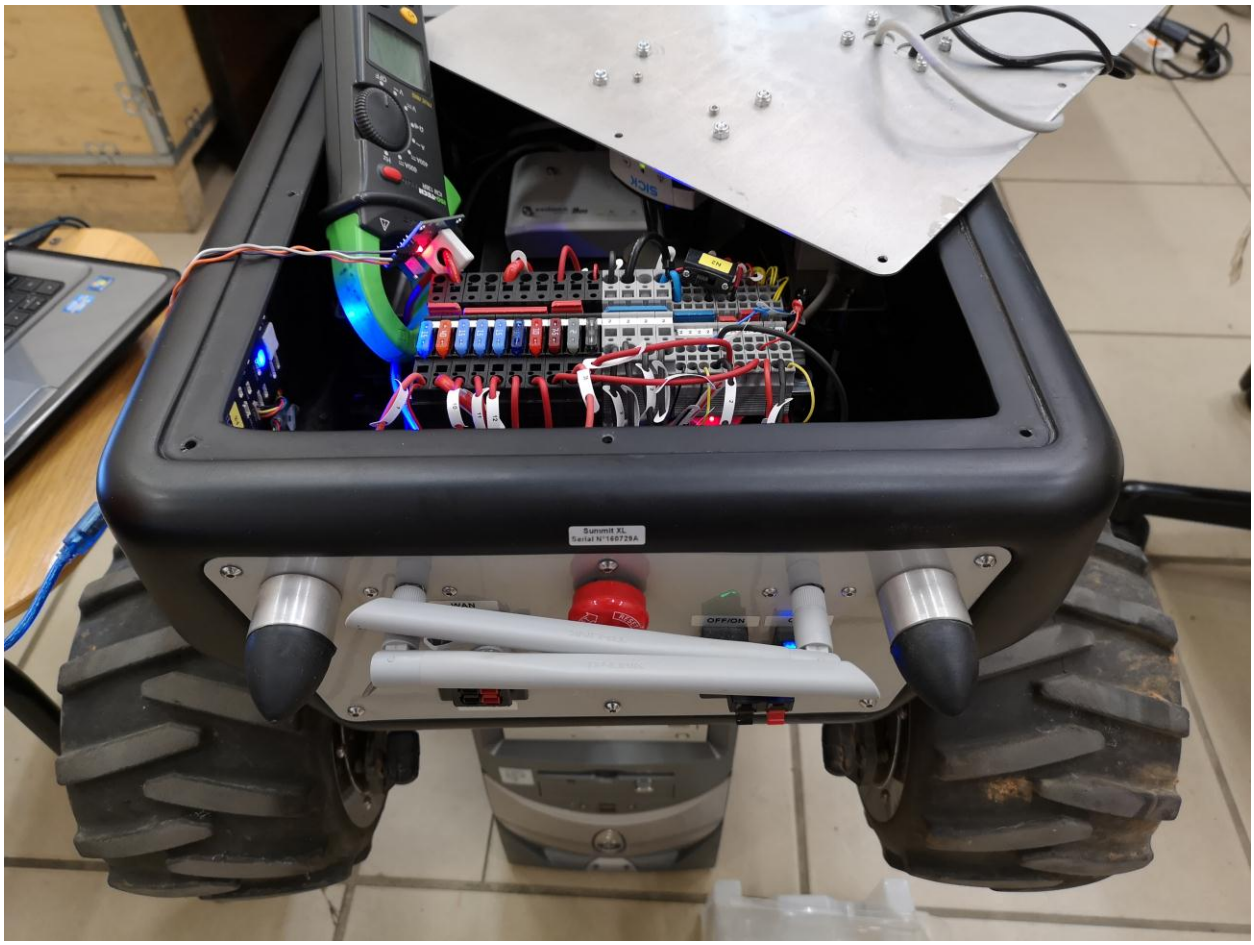


Figura 5-1 Colocación de Pinza Amperimétrica

Para saber dónde colocar la pinza amperimétrica solo fue necesario buscar el circuito de potencia en el manual del Summit XL y remitirse a la misma numeración de cable en el robot real, gracias a que todos los cables vienen numerados y ordenados.

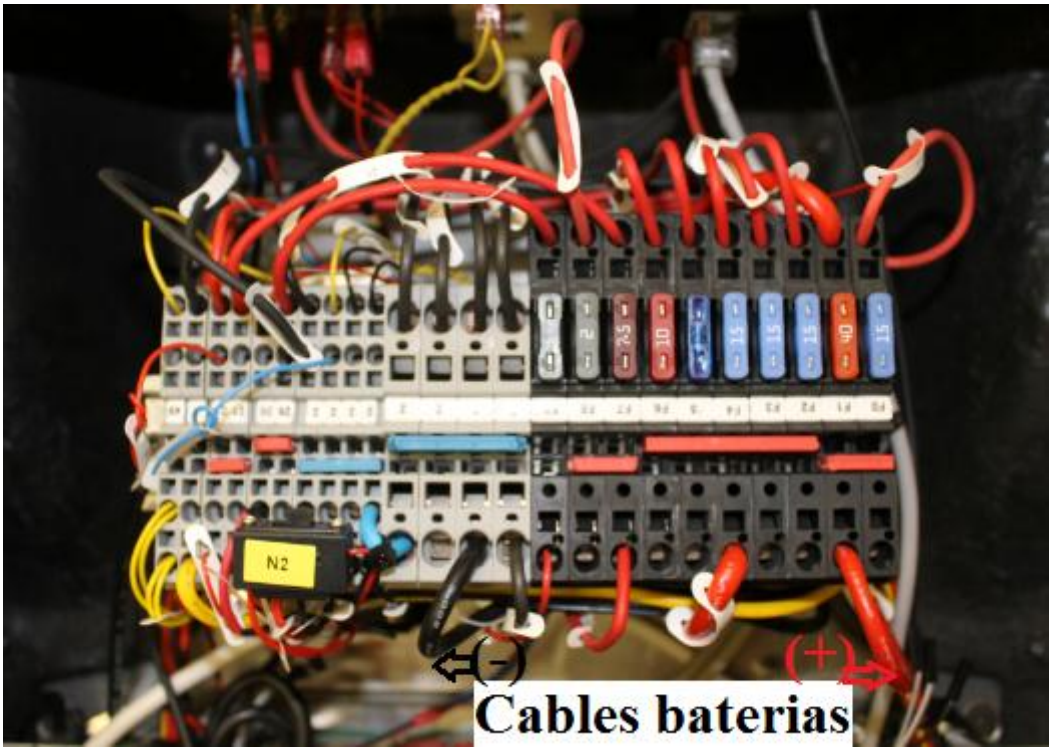


Figura 5-2 Cables de las baterías del Summit XL

Se llegó por lo tanto a la conclusión de que para un uso estándar del robot los amperios demandados a las baterías del mismo no superan los 20A y por tanto un sensor de intensidad de esas características (35Amax en continua) será suficiente para nuestro proyecto.

Una vez sabido el rango de medida necesario para la elección del sensor queda el planteamiento de qué tipo de sensor es el más indicado para esta tarea. A priori, como se buscaba usar el microcontrolador de Arduino para la medición por las múltiples comodidades expresadas anteriormente, se buscaron sensores con las especificaciones concretas para ser leídos por los puertos analógicos de Arduino UNO.

• **Tipo de tecnología**	**TTL**
• **Rango de valores de entrada**	**0-5 Voltios**
• **Resolución**	**10 Bits**
• **Valores digitales**	**0 a 1023**

Tabla 5-1. ADC Arduino UNO

Fue preferible un sensor de intensidad de tipo no invasivo, es decir, que no hubiese que cortar literalmente ningún cable del robot para la medida de la intensidad. Esto redujo bastante las opciones de búsqueda de sensor y redujo las posibilidades a sensores no invasivos o de efecto Hall. Finalmente se escogió un sensor WCS1800 de marca Winson con estas características:

• **Diámetro del sensor**	**9mm**
• **Tipo de corriente a medir**	**DC-AC**
• **Rango de medida**	**0-35A (DC), 0-25A (AC)**
• **Sensibilidad**	**66mV/A**
• **Ancho de Banda**	**23KHz**
• **Vdd**	**5V**
• **Rango de Temperatura**	**-20ºC a +125ºC**

Tabla 5-2. Características del sensor Winson WCS1800

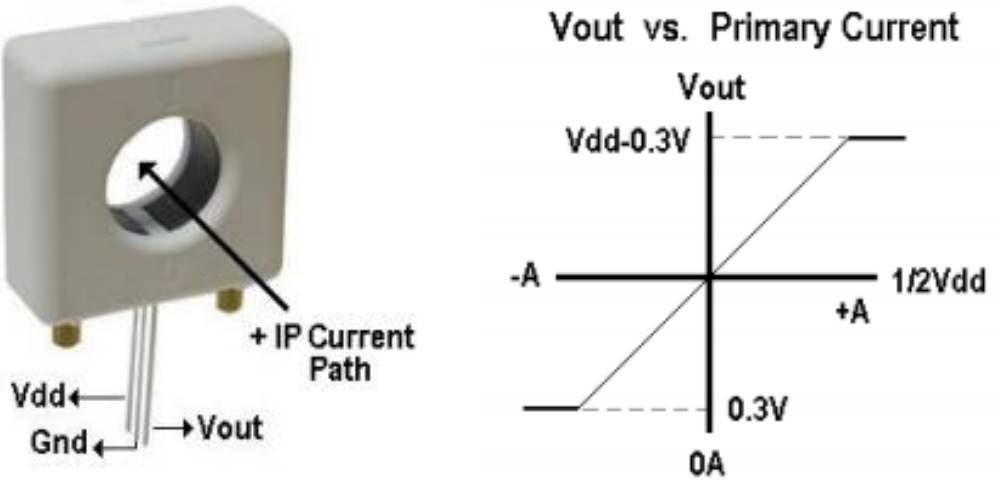


Figura5-3 Esquema y curva de sensibilidad del sensor

Como se puede observar en la figura 5-3 el sensor no es de tipo pinza, es decir habrá que abrir una vez el circuito de la batería del Summit para su instalación pero sin la necesidad de cortar ningún cable. Gracias al cuadro de potencia que lleva instalado el robot se mostrará en próximos apartados como fue instalado.

5.2 Diseño de circuito de acondicionamiento

Para los múltiples sensores de intensidad del mercado, y concretamente para las características que se buscaban, existen sencillos circuitos de acondicionamiento para la señal como: Divisores de tensión, filtros para mejorar la lectura del sensor, etc. Todo depende de la alimentación que necesite el sensor y del tipo de señal que genere al medir la intensidad por el conductor. Se trata de adecuar estos parámetros a los parámetros de los pines ADC que tiene Arduino instalados en su la placa UNO.

Por suerte existe un modelo del sensor WCS 1800 que lleva incorporado su circuito de acondicionamiento compatible con los pines de Arduino, a un precio que realmente no se podía mejorar comprando los componentes necesarios para realizar a mano dicho circuito de acondicionamiento. Este Módulo lleva el sensor incorporado, necesita una alimentación Vdd de cinco voltios y la señal de medida en voltios a la salida del sensor va desde cero a cinco.

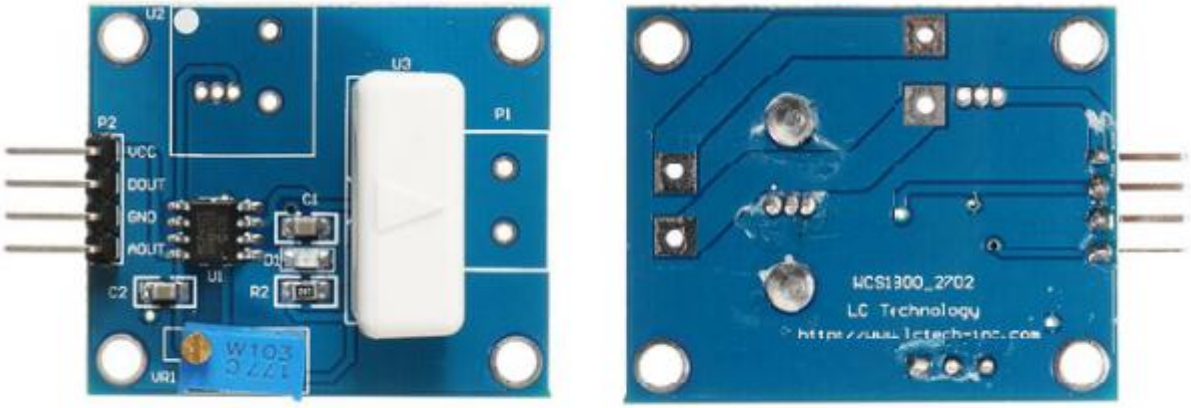


Figura5-4 Módulo WCS1800

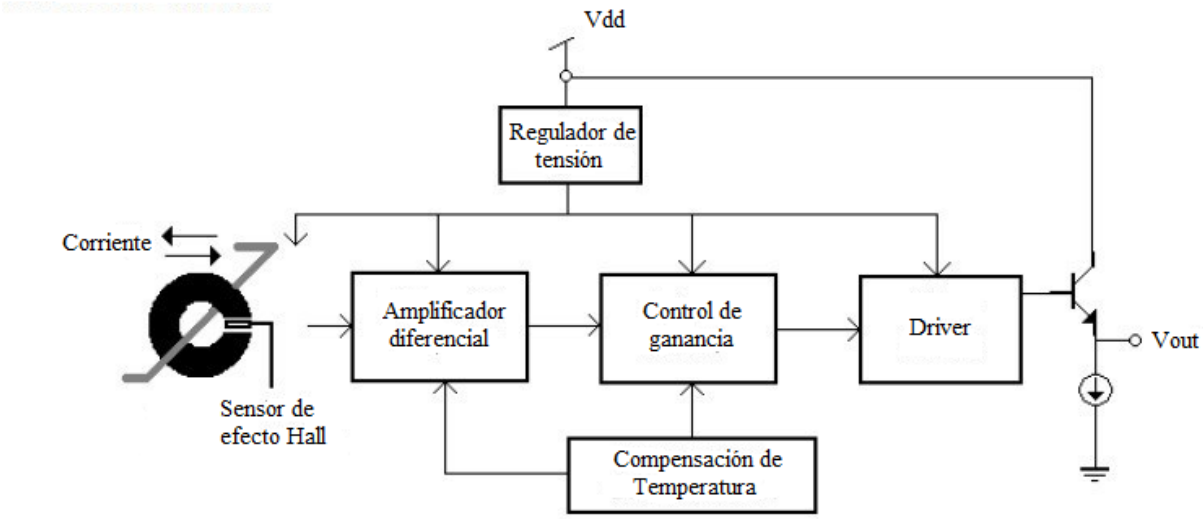


Figura 5-5 Esquema del circuito WCS 1800

5.3 Software y conexión con Arduino

Como se ha explicado en el capítulo anterior, el sensor ya viene incorporado en su propio circuito de acondicionamiento con lo que solo es necesario conectarlo con la placa Arduino Uno para obtener una medida útil de la intensidad después de escribir el código correspondiente para el Arduino.

La conexión resulta sencilla gracias al módulo, Arduino tiene salida de cinco voltios y de tierra que serán Vdd y Vgnd del sensor. Por último conectaremos el pin de salida de señal del sensor (0V a 5V DC) al pin de lectura analógica de Arduino (pin ADC – A0).

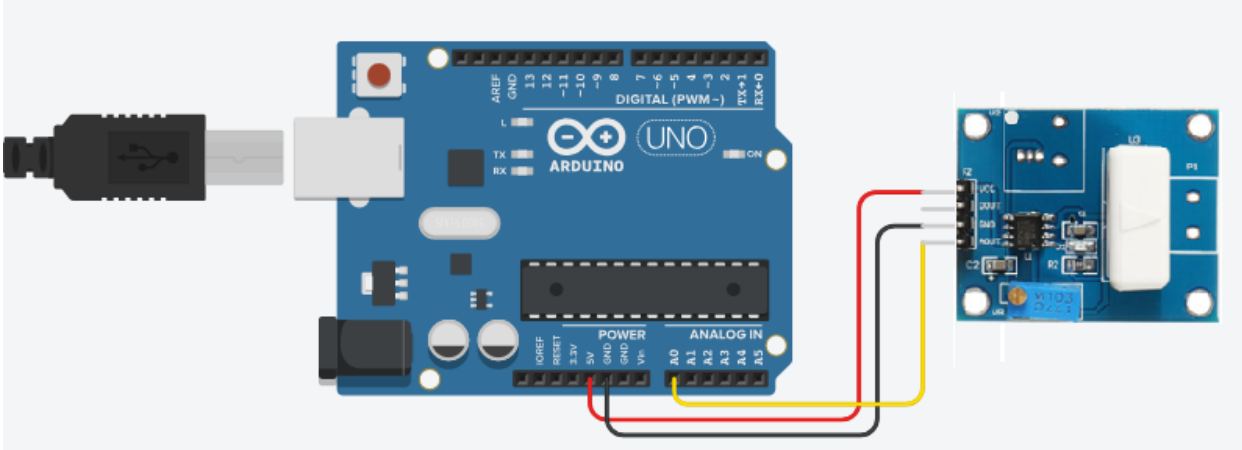


Figura5-6 Conexionado Arduino con sensor

5.4 Instalación del sistema de medida en el Summit XL

Para acceder a la parte interior del Summit XL es necesario desmontar la tapa superior del robot, para ello desatornillaremos los ocho tornillos que la fijan al chasis del mismo. Una vez se tiene acceso al interior del robot se procederá a desconectar el cable de alimentación de las baterías positivo (rojo) de la caja de potencias para así poder introducirlo por el sensor del que hablamos en este capítulo.

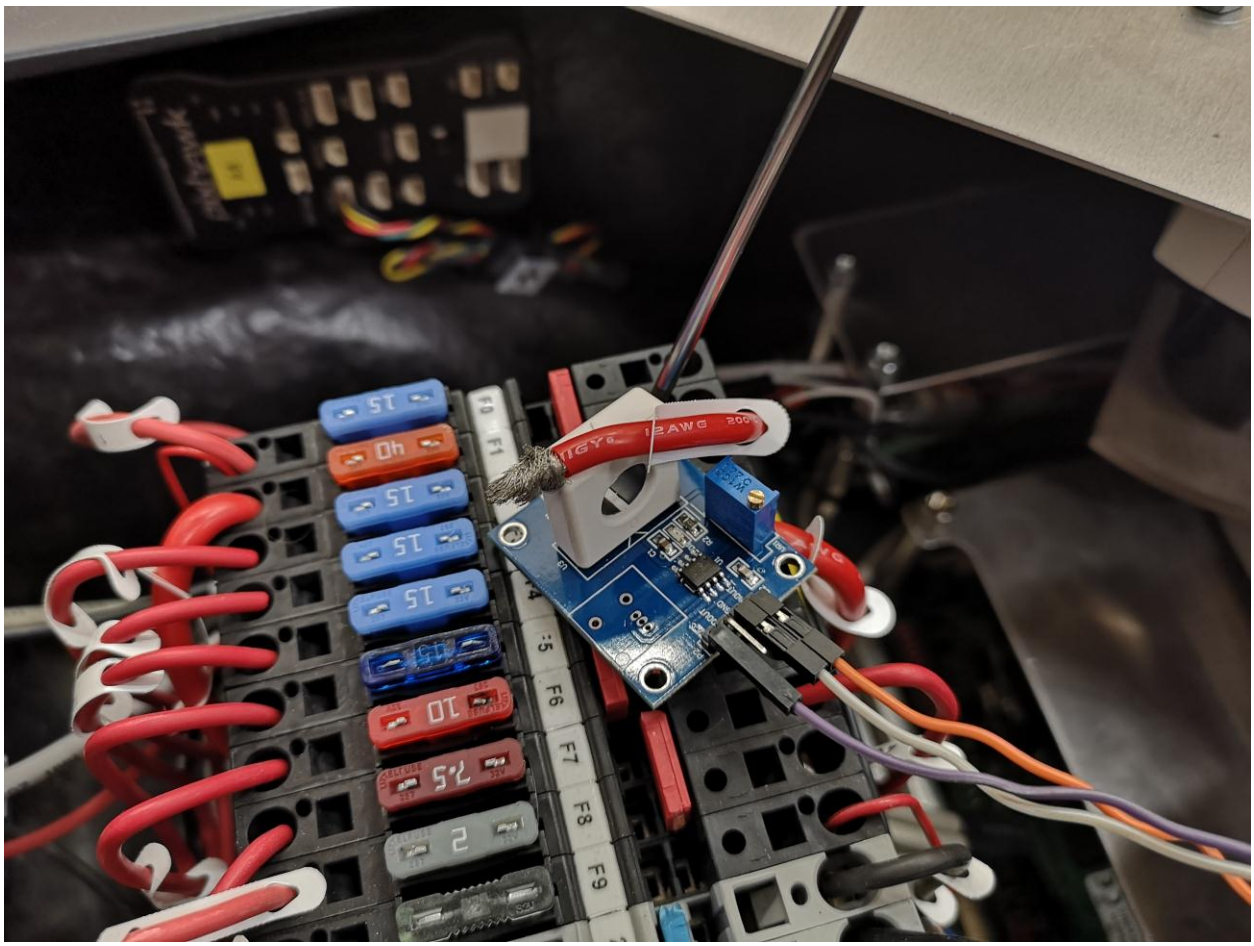


Figura 5-7 Instalación del sensor en el Summit XL (1 de 2)

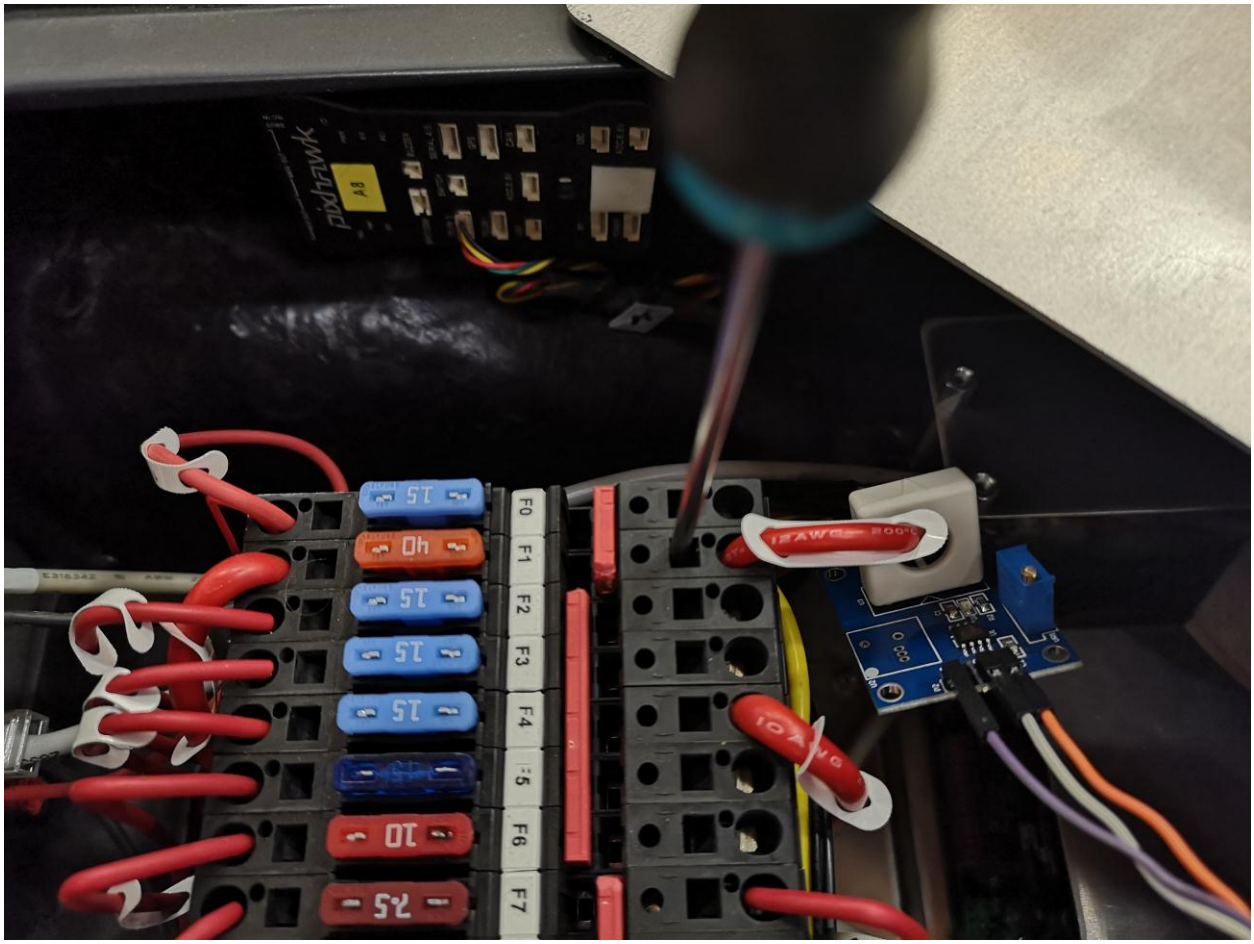


Figura 5-8 Instalación del sensor en el Summit XL (2 de 2)

6 TELEMETRÍA Y COMUNICACIONES

En el capítulo de Telemetría y Comunicaciones se trata el tema de cómo se solucionó el problema de enviar las mediciones de intensidad del sensor con el Arduino a través de la red WiFi del Summit XL para ser monitorizadas por el pc base.

6.1 Protocolo TCP en lenguaje C

En primer lugar se pensó como solución a este problema que Arduino y el módulo WiFi ESP8266 se comunicasen con el PC base sin usar ROS y que fuese el PC el que transformara las mediciones del sensor y el Arduino a un nodo ROS que publicase un topic /Intensidad en el sistema. Para ello se requiere un protocolo de comunicación entre el ESP8266 y el PC, ambos conectados a la red WiFi del Summit.

El protocolo de comunicaciones escogido fue TCP (Transmission Control Protocol) que permite la comunicación entre dos dispositivos que pertenecen a una red. En este protocolo se definen todas las reglas de comunicación para internet y se basa en la existencia de la dirección IP es decir en dar a cada dispositivo conectado una dirección única por la que poder comunicarse con él y enrutar paquetes de datos. Este protocolo detecta además también automáticamente errores en la transmisión de datos.

Desde su diseño, el protocolo TCP/IP está dividido en distintas capas de funcionamiento, como apreciamos en la figura 6-1, aunque lo realmente importante a la hora del desarrollo del proyecto es sin duda su relación con el lenguaje C.

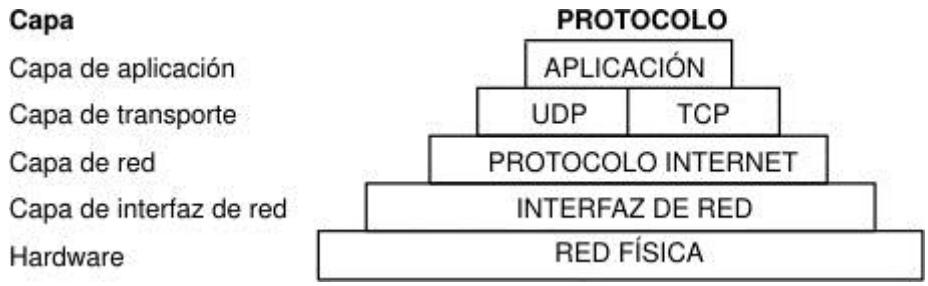


Figura 6-1 Capas de protocolo TCP/IP

Para el desarrollo de un programa en C que hiciese las veces de servidor para recibir los datos del sistema de telemetría vía WiFi y los guardase en el PC para ser introducido a posteriori al entorno ROS, fue necesario consultar algunos manuales sobre Sockets TCP.

Finalmente en el **Apéndice B** aparece el código final usado para las pruebas de conexión y envío de paquetes del módulo WiFi con la información del sensor a través del Arduino hasta el PC base con la capa de protocolo de comunicaciones TCP/IP. En el código de prueba simplemente se establece una escucha en un puerto concreto del PC y cuando el ESP8266 transmite un dato a ese puerto, el programa en C avisa con un mensaje por pantalla de que el mensaje ha sido recibido con éxito.

No se hará mucho más hincapié en el lenguaje C ni en los sockets TCP/IP debido a que finalmente la solución fue que el módulo WiFi que transmite los datos del sensor, lo hiciese directamente en una capa ROS sobre la cual también corre una capa TCP pero es totalmente transparente al usuario y solo se deben tener en cuenta las funciones propias de Arduino y ROS para la programación del sistema de telemetría.

6.2 Nodo ROS y comunicaciones WiFi

Como se comenta en el apartado anterior la forma más cómoda y con menos intermediarios para que un topic /Intensidad se transmitido al entorno ROS, donde están conectados todos los dispositivos, es mediante la utilización de la capa ROS en Arduino. Esta capa incorpora una librería donde se ubican las funciones necesarias para la publicación y lectura de topics en un entorno ROS directamente desde el módulo WiFi conectado a nuestro Arduino.

Una vez más el código que está ejecutándose en la placa WiFi ESP8266 se encuentra en el **Apéndice B**, y se explicara en los siguiente apartados del capítulo. Para entender cuan bien se integra nuestro dispositivo de telemetría en el sistema ROS, se mostrará un grafo donde aparecen todos los nodos que están actuando en el sistema ROS una vez conectado el Summit XL, el sistema de telemetría y el PC base para monitorizar los parámetros deseados (Figura 6-2)

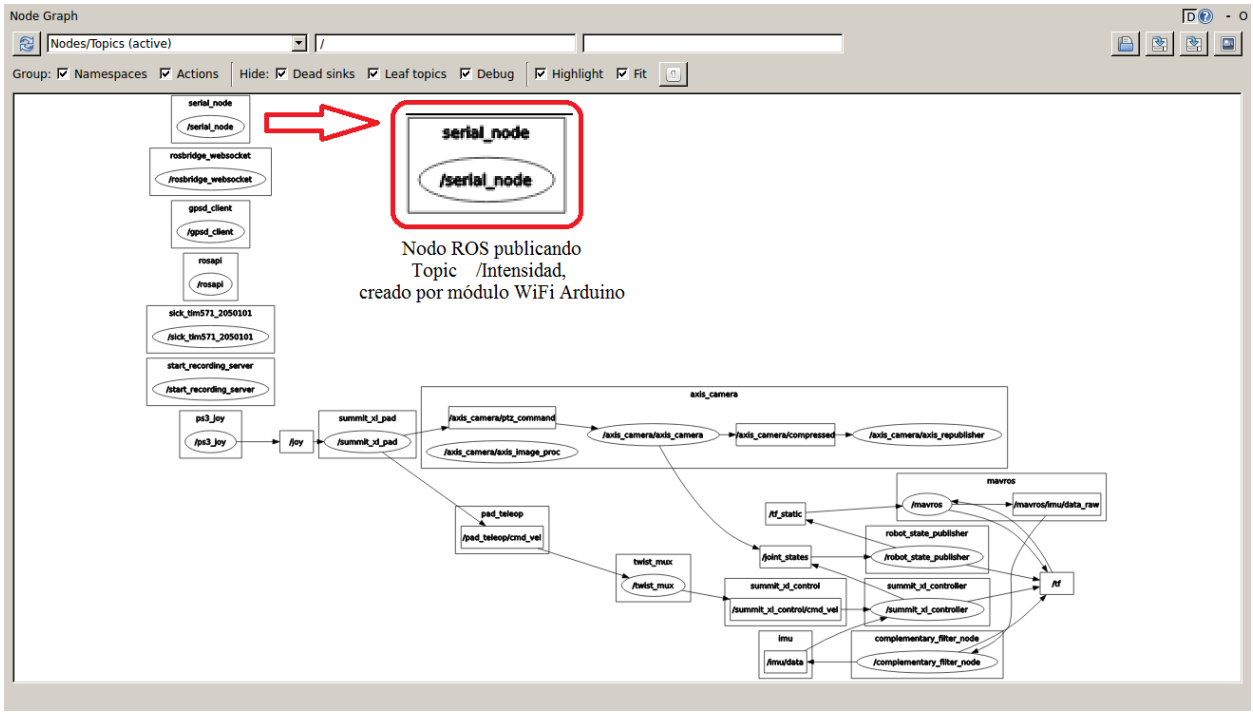


Figura 6-2 Serial Node Arduino

6.3 ESP8266 módulo WiFi

La solución proporcionada para el envío de la medida de intensidad al entorno ros de manera inalámbrica fue un módulo WiFi ESP8266 compatible con Arduino. En primera instancia parecía suficiente con la adquisición únicamente del chip WiFi, por su precio y tamaños eran ideales para el proyecto.

El chip necesita un circuito de acondicionamiento para poder tener una compatibilidad total con Arduino. Esto es debido a que el chip se alimenta con 3,3V pero necesita más intensidad que la que proporciona el pin de 3,3V de Arduino. Para solucionarlo, o bien alimentamos el chip con una batería externa o bien hacemos un divisor de tensión con el pin 5V de Arduino que si tiene la intensidad necesaria para alimentar al ESP8266 pero habría que reducir el voltaje.

Se probó la configuración con el montaje como en el esquema de la figura 6-3 con éxito. El problema residía en que la funcionalidad de la placa ESP8266 por sí sola no era suficiente para establecer una comunicación TCP con un sistema ROS porque resultaba imposible programarla con funciones ROS desde cero. Esto se debe a que este módulo solo puede ser programado con los llamados “Comandos AT”, que ofrecen una funcionalidad limitada para conexiones TCP/IP y no admiten librerías externas.

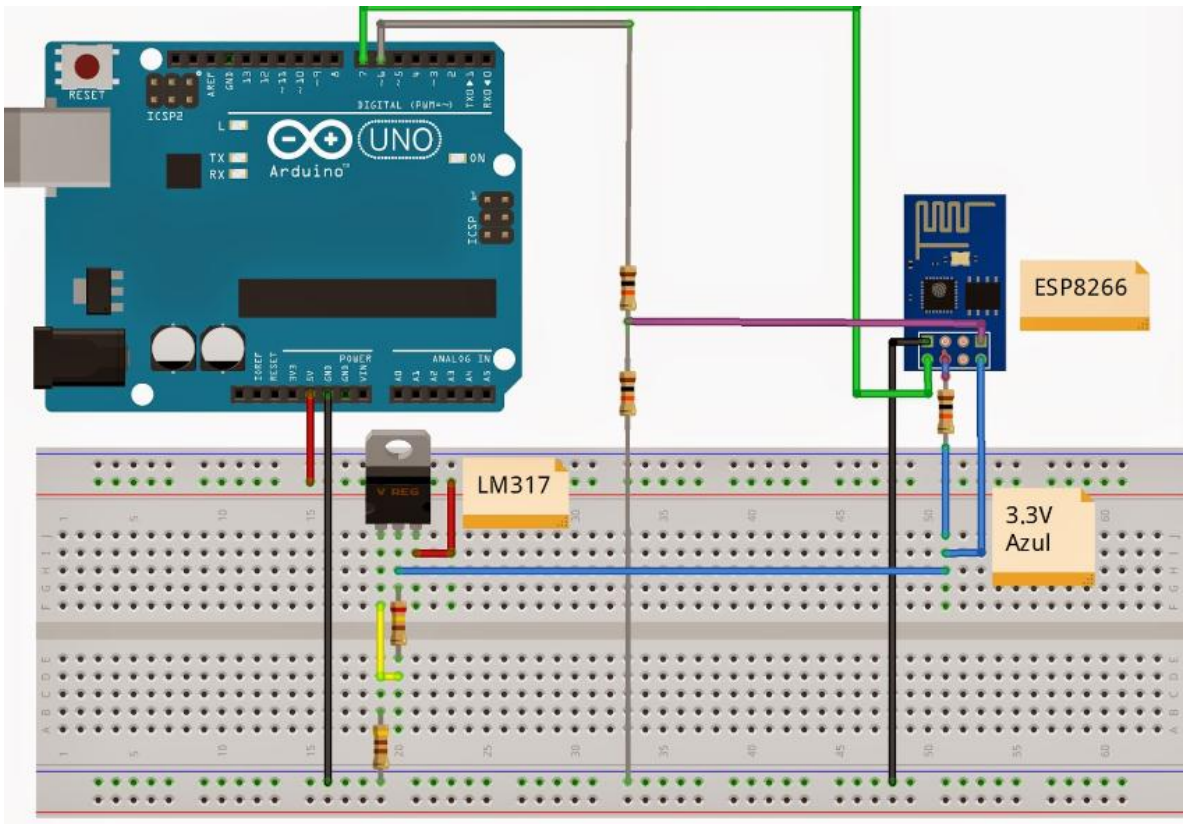


Figura 6-3 Montaje ESP866 con Arduino

Por este motivo se consultó con el Departamento de Ingeniería Electrónica de la US una alternativa al chip ESP8266, de esta manera se prefirió optar por el módulo ESP8266 Feather Huzzah. A priori este módulo tiene el mismo chip WiFi y la misma antena para las comunicaciones pero lleva en la placa instalado algo más de hardware que le permite trabajar sobre una capa Arduino en la que ya si es posible incluir librerías para trabajar expresamente sobre entorno ROS y librerías para facilitar el uso de sockets TCP/IP y conexiones inalámbricas.

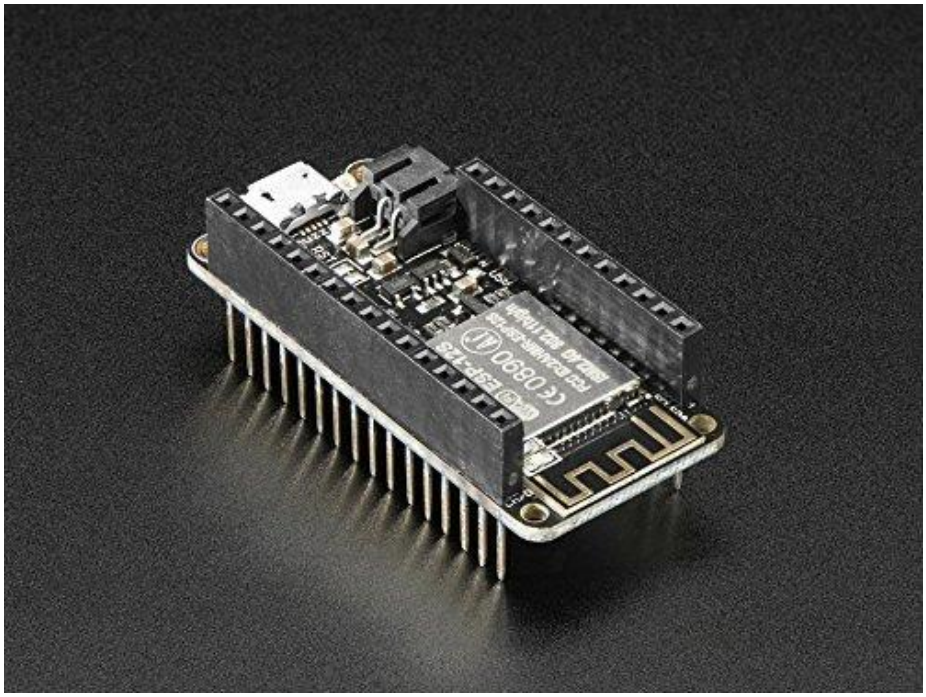


Figura 6-4 ESP8266 Feather Huzzah

6.3.1 Comunicación Arduino con ESP8266 por Puerto Serie

Esta tarea es necesaria para enviar la lectura de intensidad que obtiene Arduino, del sensor que está conectado a él, hasta el módulo ESP8266 que comentamos en el punto anterior. Gracias a que tanto Arduino como el ESP8266 Feather Huzzah corren una capa Arduino en su interior, es posible comunicarlos con Software Serial de Arduino, es decir, con la librería que utiliza Arduino para la comunicación serie por sus puertos digitales con otros dispositivos. El código necesario para la comunicación aparece en este documento en el **Apéndice B**, y la conexión cableada aparece en el esquema de la figura 6-5.

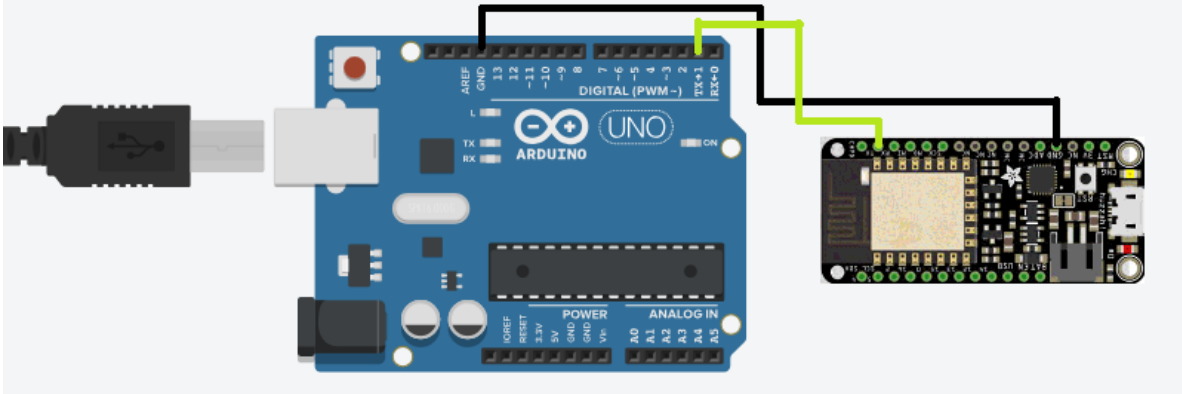


Figura 6-5 Conexión Arduino con ESP8266 Feather Huzzah

6.3.2 Comunicación ESP8266 con Sistema ROS

Una vez que el sensor obtiene una medida, la plaza Arduino transforma la señal analógica en un dato digital de intensidad y se lo transfiere al ESP8266 por protocolo serie. Cuando el módulo WiFi está recibiendo las medidas de intensidad por parte del Arduino es hora de enviarlas por la antena WiFi para ser leídas por un nodo del sistema ROS. Para ello el ESP utiliza las librerías ROS, crea un nodo Publisher desde el cual envía reiteradamente las medidas de intensidad a tiempo real.

Una vez funcionando al unísono el sistema de telemetría diseñado solo quedaría abrir el PC y crear un nodo ROS donde exista un Listener o receptor del mensaje que adquiera los datos del topic /Intensidad por el puerto concreto, en nuestro caso 11411 de la dirección IP asignada por el router del Summit XL a nuestro PC.

Para ello hará falta lanzar 3 ventanas distintas de consola de comandos para desplegar los procesos ROS correspondiente para el desempeño de la tarea:

- Terminal 1
  - export ROS_MASTER_URI=http://summit:11311 (lanzamos ROSCORE en el computador del Summit)
  - rosrun rosserial_python serial_node.py tcp (Lanzamos nodo de lectura de Puerto serie por protocolo TCP)
- Terminal 2
  - ssh summit@summit (Para entrar en la sesión de consola del robot desde nuestro PC)
  - rostopic list (Para ver todos los topic activos que se están publicando y donde debe aperecer /Intensidad)
- Terminal 3

- rostopic echo /Intensidad (Para monitorizar los valores de intensidad en tiempo real

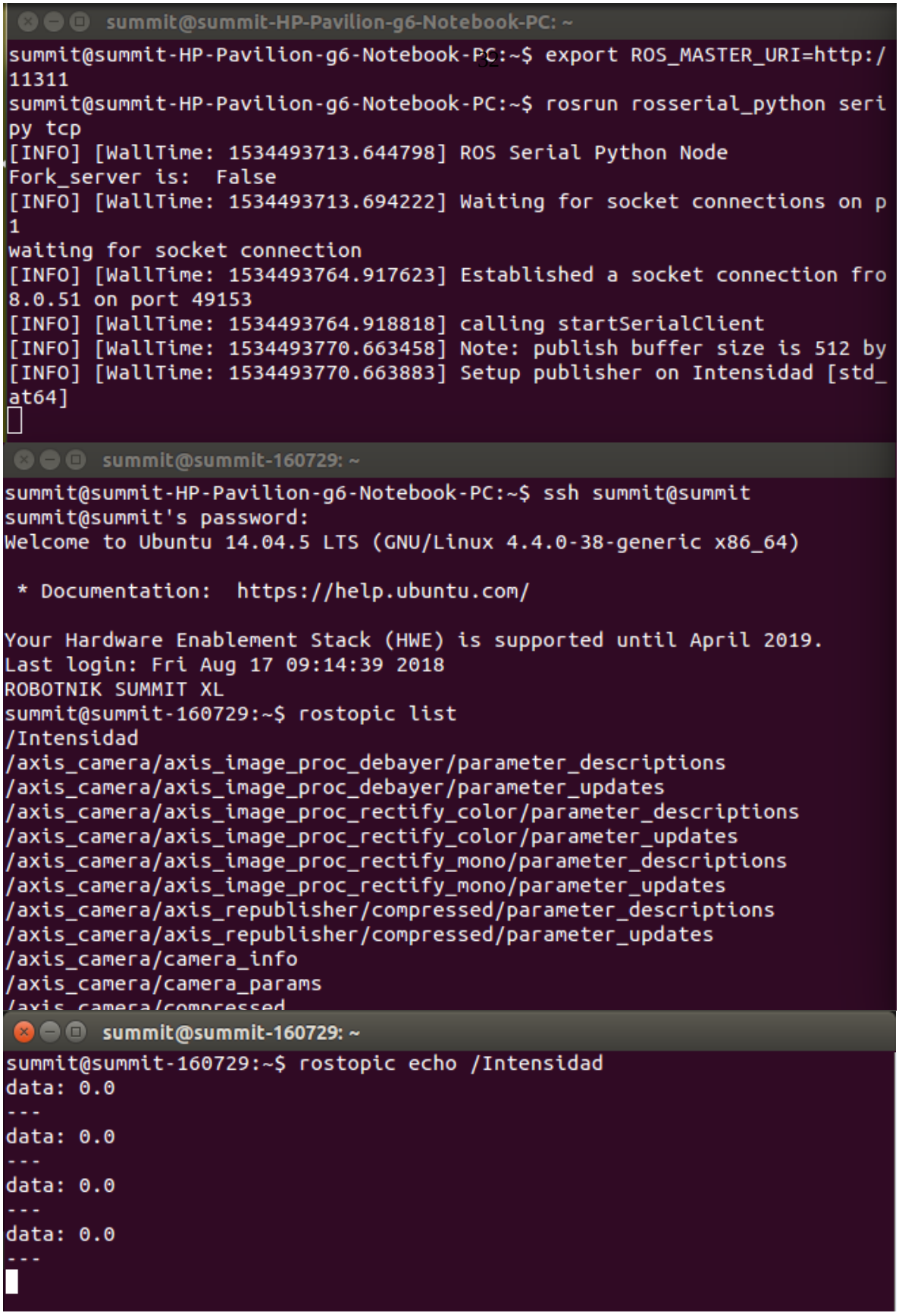


Figura6-6 Terminales ROS

7 PRUEBAS DE MEDICIÓN

Este capítulo incluye toda la información relativa a las pruebas de medición del sistema de telemetría sobre el Summit XL. Se explicarán minuciosamente tanto las simulaciones en el laboratorio como las pruebas reales sobre el robot y se expondrán los datos obtenidos de una manera clara y concluyente.

Todas las pruebas se han realizado en el laboratorio de Sistemas y Automática, y tanto las herramientas como los dispositivos usados para su realización forman parte de este laboratorio.

7.1 Ensayos en bancada

En primer lugar se probó el sistema de telemetría en un entorno controlado y sin tener que abrir en robot para hacer las pruebas pertinentes para calibrar el sensor de intensidad. Para la realización de estas pruebas se necesitó el siguiente material:

- Fuente de Alimentación DC Keysight N8946A para generar una corriente de un amperaje conocido y poder ajustar el sensor de intensidad lo mejor posible.

- Carga activa SBS para cerrar el circuito donde conectar nuestro sensor, formado por fuente de alimentación y carga.

7.1.1 Metodología

La metodología para la simulación de una medida por el conductor es muy sencilla. Se conectará la fuente de alimentación a la carga mediante dos cables, positivo y negativo. A continuación se colocará el sensor de intensidad de nuestro sistema de telemetría sobre uno de los cables para medir la intensidad que pasa a través de él y por último manejaremos la cantidad de amperios que queremos que circulen por el circuito para comprobar la fiabilidad del sensor de intensidad.

Se realizarán dos pruebas bajo el mismo criterio, duración de un minuto y aumentando el valor de la intensidad por el circuito cada diez segundos. En la primera prueba se comenzara en cero Amperios hasta un máximo de ocho aumentando de dos en dos. En la segunda se comenzará por diez Amperios para ir subiendo de uno en uno hasta quince amperios.

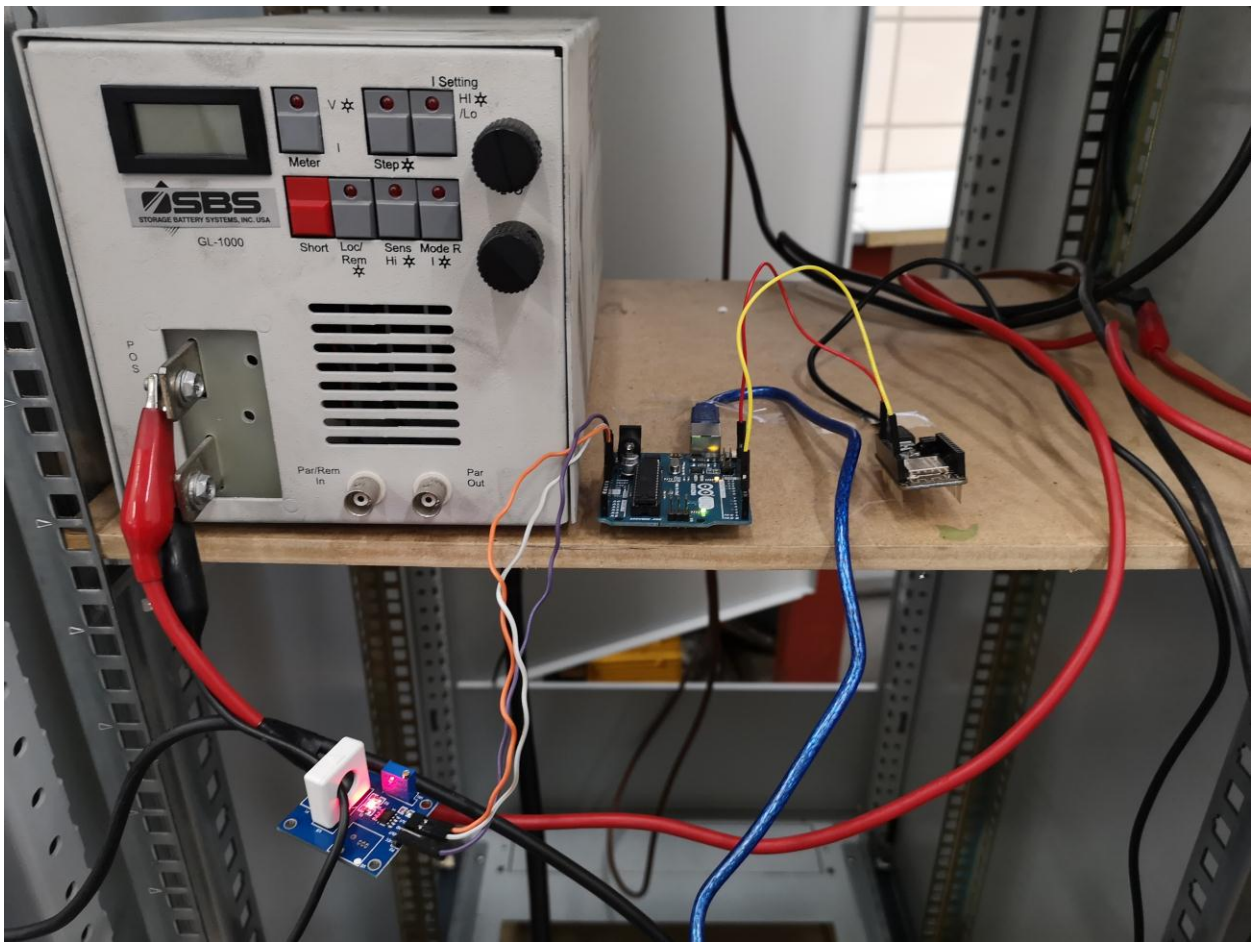


Figura 7-1. Montaje de circuito de prueba

7.1.2 Resultados del ensayo

Los resultados de la primera prueba se encuentran en la Figura 7-1:

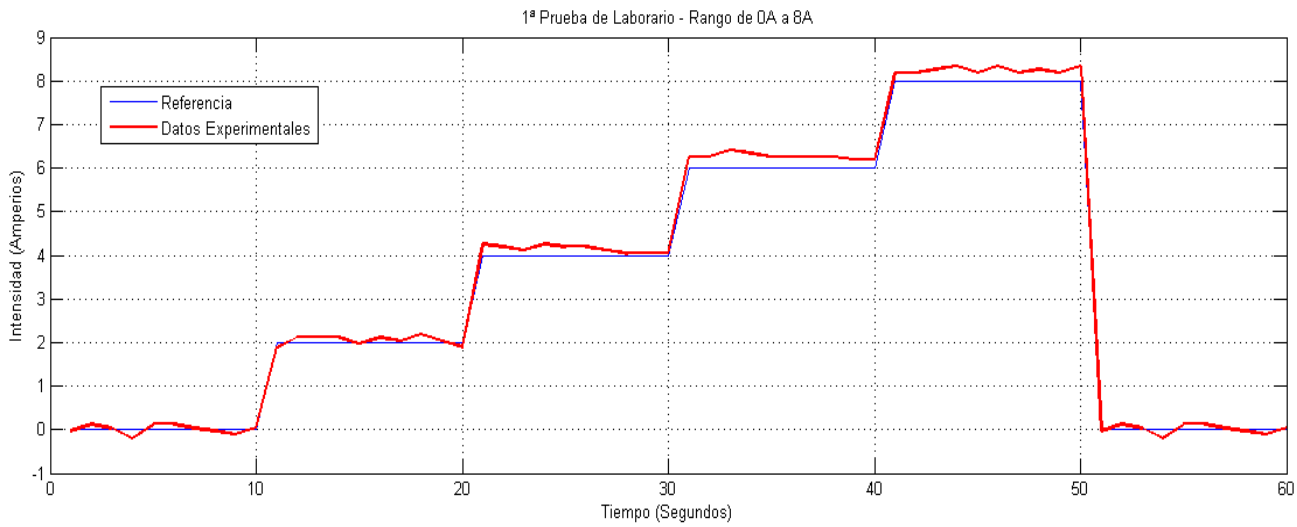


Figura 7-2 Resultados primera prueba de laboratorio

Se comprueba a simple vista que el sensor ha sido calibrado a intensidad cero y realiza una medición bastante coherente con la referencia a medir a pocos Amperios, a medida que subimos la intensidad la calidad de la medida se va desvirtuando un poco de errores de medida del ±5% en intensidades próximas a cero hasta errores del +10% a intensidades próximas a diez Amperios.

Los resultados de la segunda prueba se encuentran en la Figura 7-2:

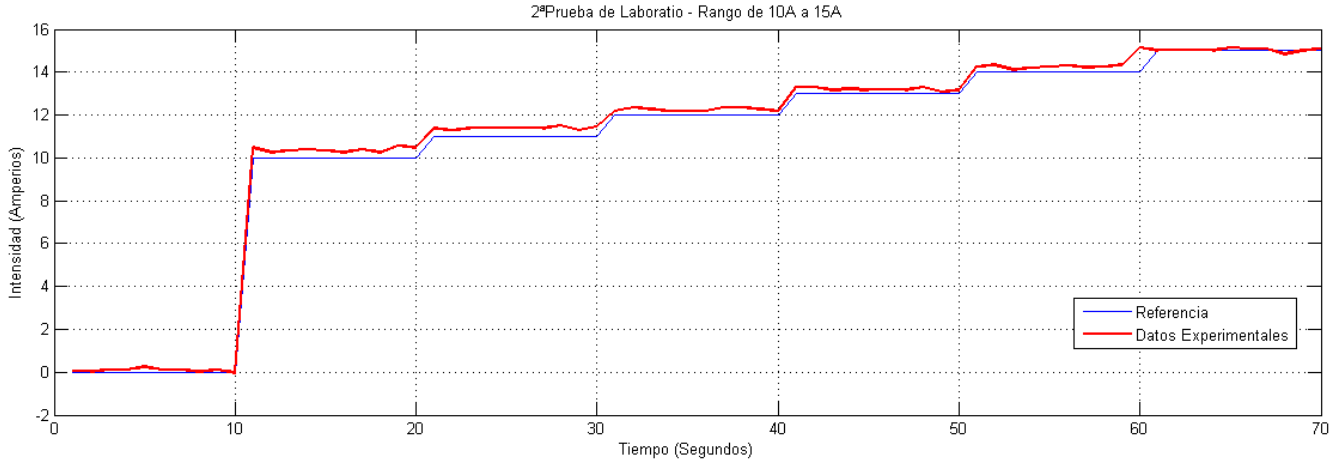


Figura 7-3. Resultados segunda prueba de laboratorio

En este caso se puede observar que el error en la medida sigue siendo mayor (en torno a un +10%) en el área de medida de unos diez Amperios pero esta vez disminuye el error cuando seguimos incrementando la corriente. En la Figura 7-4 podemos ver un detalle para apreciar mejor la distancia entre la medida y su referencia.

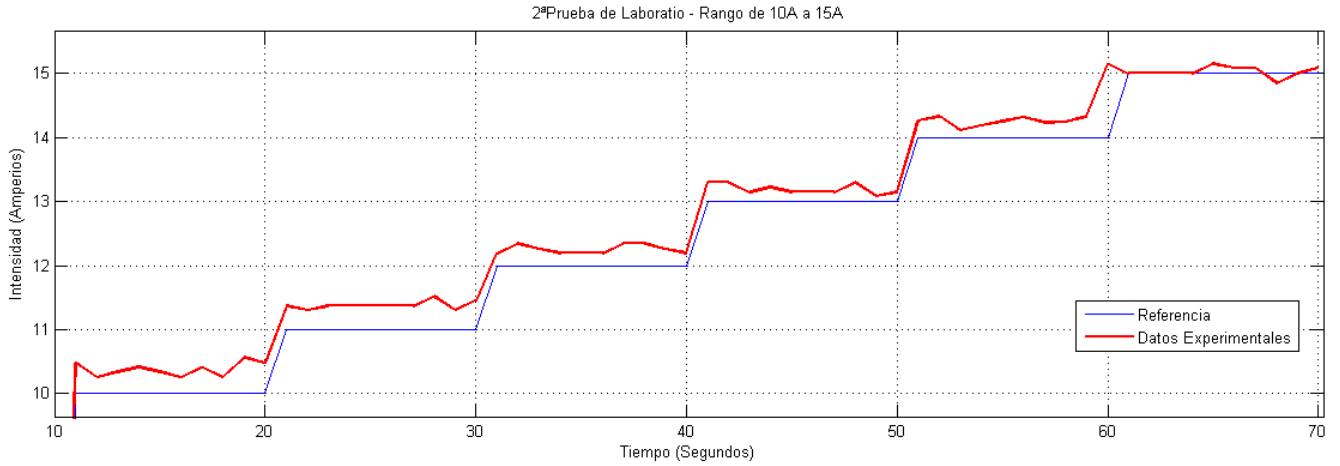


Figura 7-4. Detalle de la segunda prueba de laboratorio

7.2 Ensayos en el robot

En este caso se procedió a abrir el robot e instalar directamente el sistema de telemetría en su interior. Para alimentarlo se usó uno de los puertos USB de sobra con los que cuenta el Summit XL y desde el Arduino alimentado, basta con el pin de cinco voltios para alimentar al ESP8266. Una vez instalado el sensor se procedió hacer otras dos pruebas de veracidad de la medida en el Summit XL.

Para ello se subió al robot a una plataforma dejando sueltas y al aire sus cuatro ruedas de modo que pudiesen girar y el robot permanezca en el sitio todo el tiempo para adquirir los datos con el PC.

Hay que tener en cuenta que la intensidad usada como referencia de lo que el sensor debe medir es una media de lo que el robot consume en ese modo de funcionamiento. Haremos ambas pruebas con 7 modos de funcionamiento diferentes, durarán setenta segundos e iremos cambiando cada diez segundos entre: parado, círculos a la derecha, círculos a la izquierda, hacia adelante, hacia atrás, recto derecha, recto izquierda. La

única diferencia entre ambas pruebas será que el Summit funcionará a velocidad media (5/10) o a velocidad alta (10/10) dentro de sus diez velocidades posibles.

7.2.1 Condiciones de las pruebas

Para esta prueba se ha abierto el robot y colocado el sensor y la pinza amperimétrica sobre el cable positivo que proviene de la batería del Summit como se aprecia en la Figura 7-5.

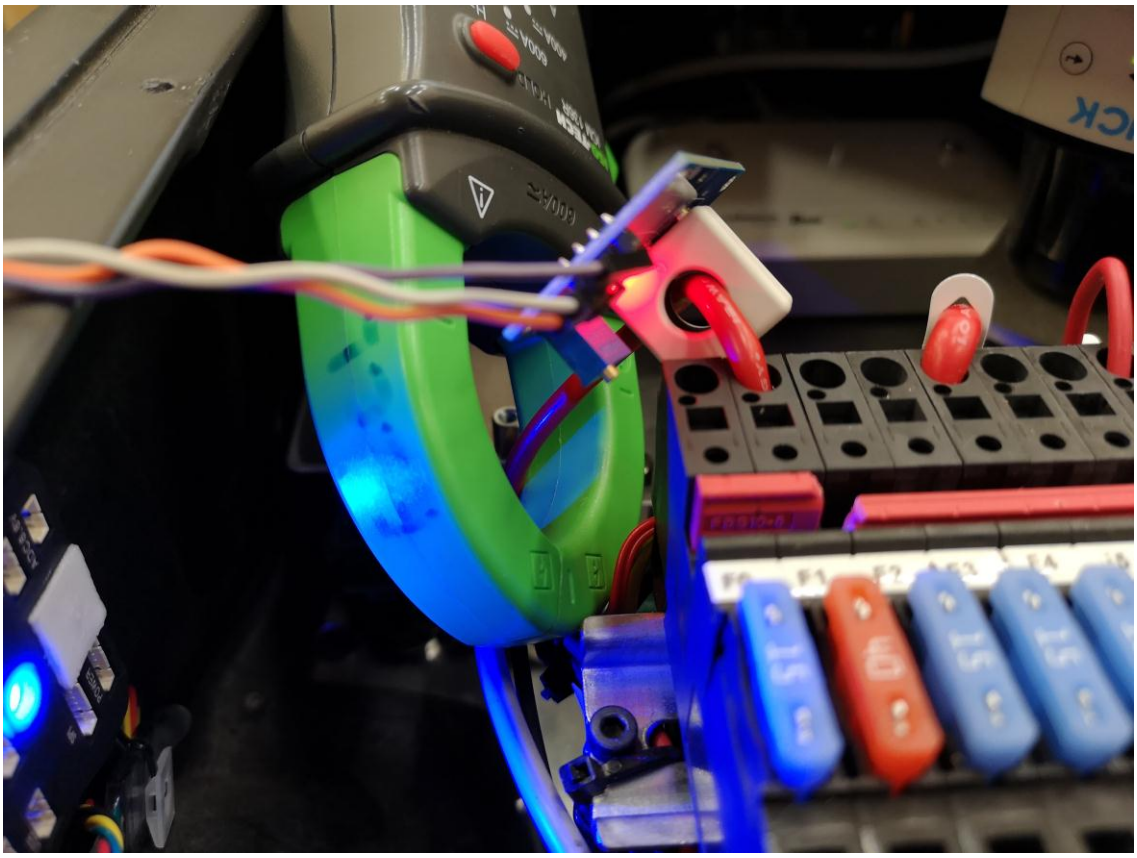


Figura 7-5. Instalación de sistema de telemetría

7.2.2 Resultados experimentales

Los resultados de la primera prueba se encuentran en la Figura 7-6:

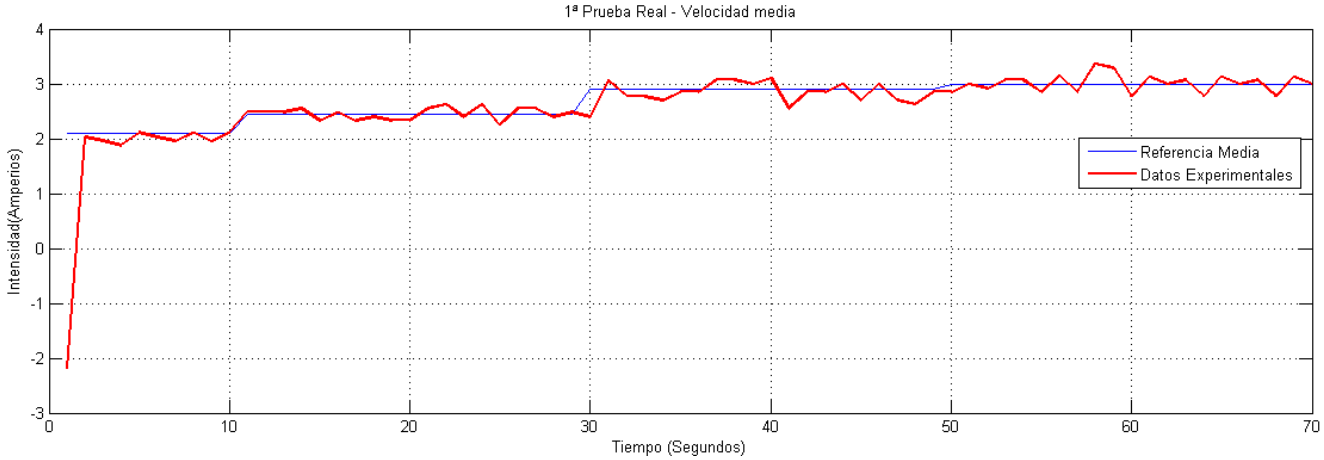


Figura 7-6. Resultados primera prueba real

Como se ha comentado anteriormente la referencia es la media de los valores que toma el Summit en los diez segundos de marcha continuada, pero la medida real del robot con la pinza amperimétrica también muestra variaciones que no se han podido digitalizar como datos para transmitirlos a esta gráfica.

Los resultados de la segunda prueba se encuentran en la Figura 7-7:

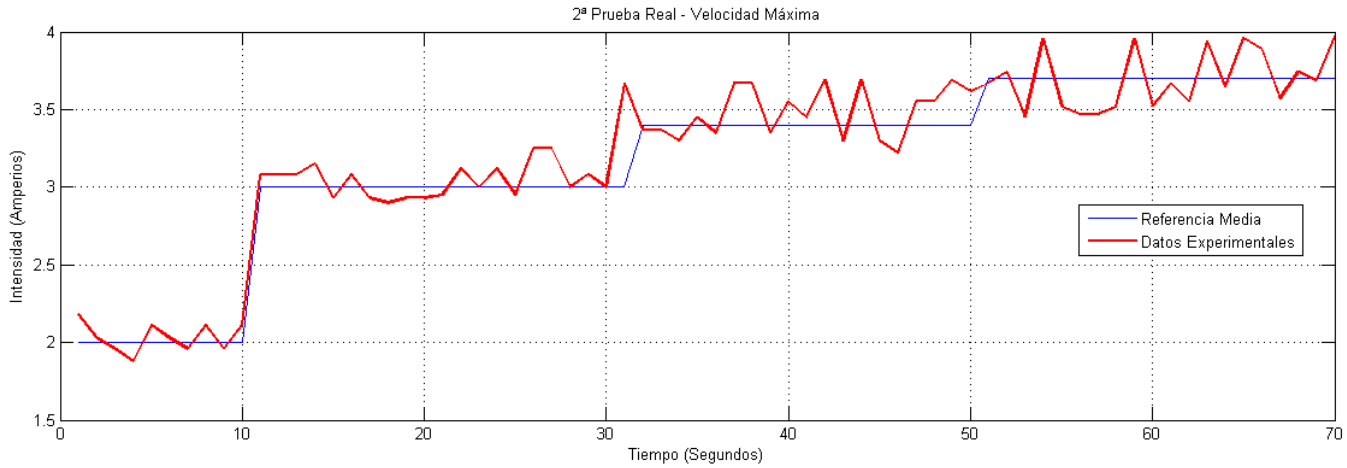


Figura 7-7. Resultados segunda prueba real

Realmente y como en el último gráfico se aprecia el seguimiento de la media referencial de intensidad que marca el Summit XL según los valores promedio tomado por la pinza amperimétrica en las diferentes marchas del robot. Los diferentes picos en la medida no se deben a errores en la medida si no a saltos reales de intensidad que circulan por los conductores cuando el robot demanda instantáneamente una intensidad u otra.

8 SISTEMA EMS

En el capítulo octavo de este proyecto se tratará el tema de las comunicaciones con el sistema EMS (Sistema de gestión de energía) del INTA. Es importante recordar que el sistema de telemetría que se ha diseñado en el proyecto tiene dos funciones principales, la primera es la que se ha solventado en los últimos capítulos y que está relacionada con el sensor de intensidad para las baterías del Summit XL y la segunda función del sistema diseñado es la comunicación con el sistema EMS diseñado por el INTA. La información recibida por el sistema de telemetría que proviene del EMS también tendrá que ser transmitida por el módulo ESP8266 de manera inalámbrica como se hizo con la medida de la intensidad.

8.1 Introducción a la placa EMS del INTA

El EMS se diseñó para el desarrollo e implementación de un sistema de control y monitorización para un sistema híbrido compuesto por batería y pila de combustible en configuración pasiva. Las funciones básicas del sistema de control son las propias de un sistema de protección y control de baterías de ion-litio (BMS), así como la gestión y control de la pila de combustible.

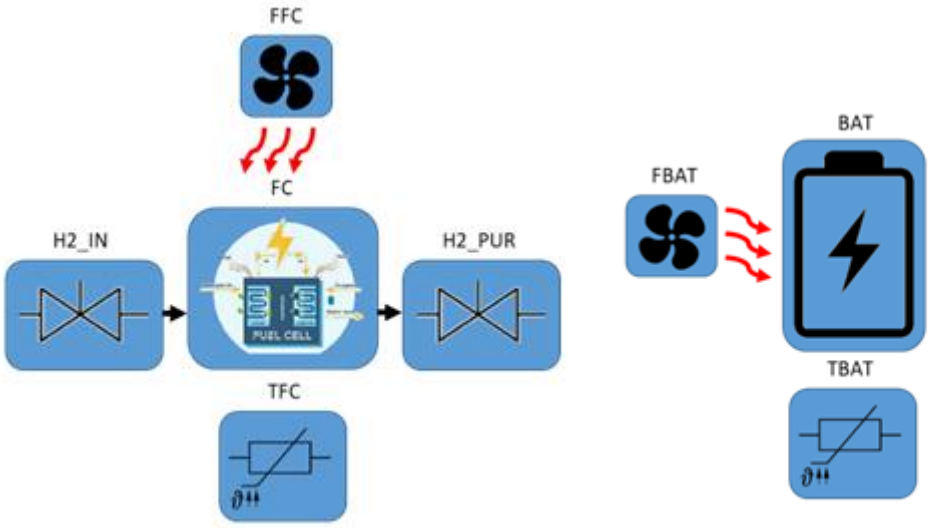


Figura 8-1 Sistema de control de potencia

El sistema de control incorporará todas las funciones básicas asociadas a sistemas de protección y gestión de baterías de ion-litio (BMS), entre las que se destacan, lectura de tensión de batería, lectura de corriente, protección contra sobretensión, sobre corriente, baja tensión y alta temperatura. Para ello, el sistema implementa una configuración basada en Mosfet de muy baja resistencia serie que permiten habilitar o deshabilitar la carga/descarga en función de las condiciones de operación.

El control sobre la pila de combustible incluye todas las acciones necesarias para la gestión de las electroválvulas de suministro y purga de hidrógeno, así como la gestión térmica a través del control de caudal del ventilador de refrigeración de la pila de combustible. De forma adicional, se incluyen sensores para la monitorización de tensión de pila de combustible, así como presión del tanque de hidrógeno para la estimación de Autonomía.

Para poder realizar el proceso de comunicación con este sistema EMS, el INTA facilitó algunos esquemas para el entendimiento del mismo, como el que se muestra en la Figura 8-2.

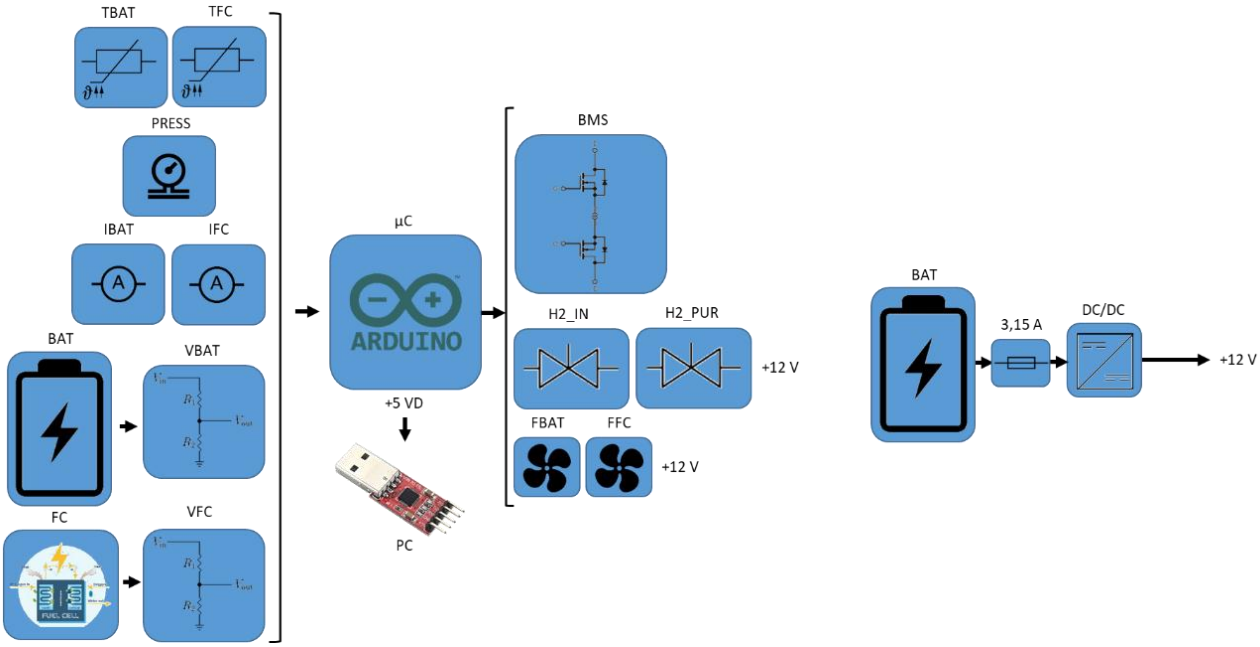


Figura 8-2 Esquema de la placa de control del INTA

8.2 Comunicación con el EMS

Realmente, la parte que concierne a este proyecto es la parte de comunicaciones, más que el funcionamiento en sí o el diseño del EMS. En cuanto a las comunicaciones con el módulo, sabemos que consta de una sería de parámetros de entrada de los que irá adquiriendo información. Posteriormente intentaremos comunicarnos con el EMS, a lo que el módulo responderá con un conjunto de parámetros de salida que tendremos que interpretar con nuestro Arduino UNO. A continuación se detallan todos los parámetros del sistema de control EMS:

- **Parámetros de entrada:** Los parámetros de entrada incluyen todas las variables analógicas asociadas con las variables físicas y eléctricas del sistema híbrido compuesto por baterías y pilas de combustible.
    - $\mathbf{I_{BAT}}$: Corriente de batería (A)
    - $\mathbf{V_{BAT}}$: Tensión de batería (V)
    - $\mathbf{T_{BAT}}$: Temperatura de batería (ºC)
    - $\mathbf{I_{FC}}$: Corriente de pila de combustible (A)
    - $\mathbf{V_{FC}}$: Tensión de pila de combustible (A)
    - $\mathbf{T_{FC}}$: Temperatura de pila de combustible (A)
    - $\mathbf{P_{FC}}$: Presión tanque de hidrógeno (Bar)

- **Parámetros de salida:** Los parámetros de salida incluyen todas las variables de actuación sobre el sistema las cuales en su mayoría se basan en variables digitales.

- **C_{BAT}:** Activación/Desactivación de la carga de batería
- **D_{BAT}:** Activación/Desactivación de la descarga de batería
- **$H2_{IN}$:** Activación/Desactivación de la electroválvula de suministro de hidrógeno
- **PURGE:** Activación/Desactivación de la electroválvula de purga
- **FAN_{BAT}:** Activación/Desactivación del ventilador de batería
- **FAN_{FC}:** Activación/Desactivación del ventilador de pila de combustible
- **FAN_{FC_SPEED}:** Señal de control de velocidad del ventilador de pila de combustible

- **Parámetros internos:** El conjunto de parámetros internos son aquellas variables de tipo digital que permiten determinar el tipo de error que ha provocado la parada del sistema.
  - **OV**: Fallo por sobretensión de baterías
  - **UV**: Fallo por baja tensión de baterías
  - **OTC:** Fallo térmico durante la carga de batería
  - **OTD**: Fallo térmico durante la descarga de batería
  - **OC:** Fallo por sobre corriente de batería
  - **$ERROR_{H2}$:** Fallo por falta de suministro de hidrógeno
  - **$ERROR_{Ta}$:** Fallo térmico de pila de combustible

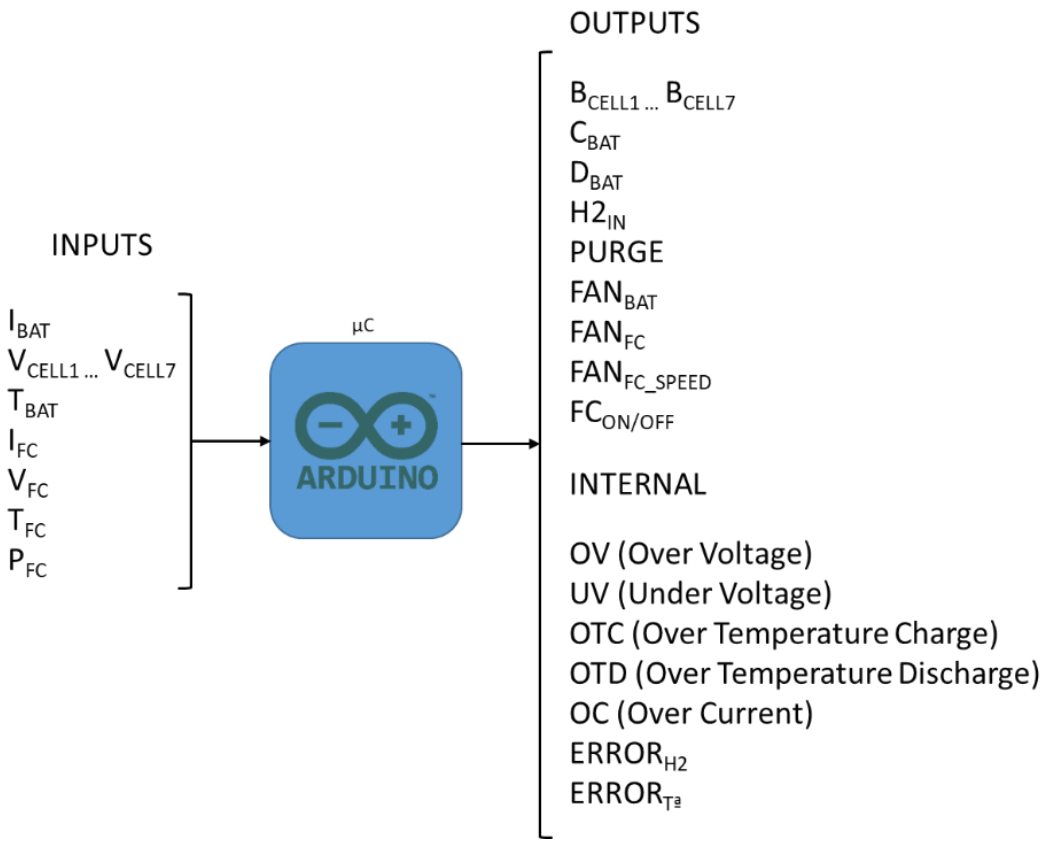


Figura 8-3 Entradas y salidas del sistema EMS

Es el momento de explicar la interfaz de comunicación. Por suerte, el EMS del INTA también está basado en Arduino con lo que la comunicación a priori podría ser bastante sencilla. Se comunicará el módulo del INTA con el Arduino Uno del sistema de telemetría a través de la UART de ambos micro controladores gracias a la implementación del protocolo serie propio de las placas Arduino. Las características de este entramado de comunicación son:

Tabla 8-1. Características protocolo de comunicación

• **Baud Rate:**	**9600**
• **Data bits:**	**8**
• **Paridad:**	**Ninguna**
• **Bits de parada:**	**1**
• **Control de flujo:**	**Ninguno**

Esta trama de comunicación se basa en un protocolo de llamada respuesta en el cuál el Arduino UNO del sistema de telemetría hace una petición de parámetros por medio del envío del comando de llamada "&", al ser recibida la petición por el EMS, este enviará la ristra de respuesta para cada petición "#Vbat/Ibat/Temp_bat/Vfc/ifc/Temp_fc/Presion/Charge/Discharge/OV/UV/OT/Fan_bat/Fan_fc/PWM/Purge/Fc/H2/Error_tª/Error_H2".

El código de Arduino utilizado para programar la trama de comunicación de esta parte del proyecto se encuentra escrito y detallado en el **Apéndice B**.

9 CONCLUSIONES Y FUTURAS MEJORAS

9.1 Sistema de telemetría

Es importante aclarar que el sistema completo desarrollado en este trabajo es en esencia un prototipo funcional que cumple todas las características requeridas para el proyecto. Se ha de mencionar, no obstante, algunas posibles mejoras que podrían ser desarrolladas para concluir el sistema satisfactoriamente.

La primera de ellas sería crear una carcasa robusta que albergue todo el sistema montado sobre una protoboard. El sistema actual está unido con cables de prototipado de electrónica para crear el conexionado entre los módulos y el sensor. Si la finalidad del sistema de telemetría es viajar en el interior del Summit XL sería necesaria la creación de esta carcasa para evitar también contactos indeseados entre los pines de la placa Arduino y los materiales conductores del interior del robot.

La segunda mejora sería alimentar el sistema Arduino directamente por sus pines de alimentación a partir de encontrar en el robot una fuente de 5 Voltios con la que alimentar el Arduino de forma directa al encender el robot y poder prescindir de la conexión USB que ocupa un puerto del robot solo para la alimentación de la placa Arduino.

9.2 Conexión con EMS

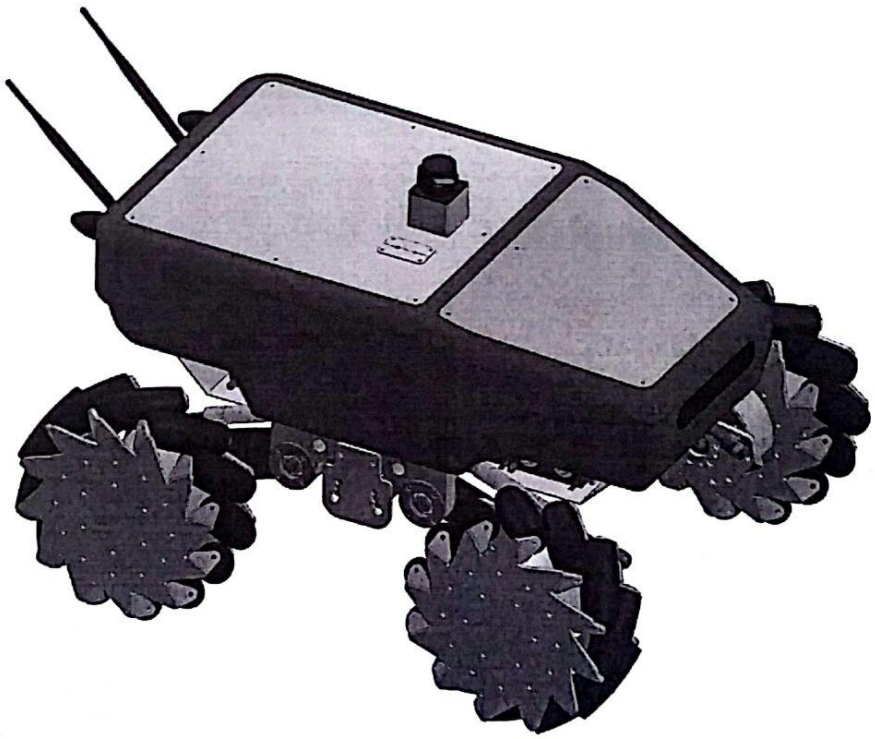
Por último se ha de especificar que debido a que nunca se tuvo en el laboratorio de Sistemas y Automática el módulo de control EMS físicamente, no se han podido realizar los ensayos necesarios para demostrar el funcionamiento del sistema completo, por lo que este proyecto podrá concluirse en futuras líneas de investigación.

Harían falta comprobaciones y ensayos de comunicación entre la placa Arduino y el EMS del INTA para comprobar el correcto funcionamiento del protocolo de comunicación incorporado en ambas placas. A priori se han seguido estrictamente las especificaciones del INTA para adaptarse de la mejor manera a la comunicación con su módulo pero debido a tiempos de entrega de la empresa nunca se pudieron hacer ensayos con el módulo real.

APÉNDICE A
GUÍA DE INICIO RÁPIDO DEL SUMMIT XL

Robotnik

SUMMIT XL
MOBILE PLATFORM



GUÍA DE INICIO

RBTNK-DOC-160802A
Robotnik Automation SLL, Spain

ÍNDICE

1) Introducción

Este documento es una guía rápida para la comprensión y manejo del robot Summit XL de la compañía Robotnik®. Está enfocado principalmente a aquellos alumnos que pretendan retomar el trabajo que se ha venido realizando en años anteriores sobre el Summit XL.

La guía contiene desde la instalación del sistema operativo Ubuntu (necesario para correr ROS) hasta la explicación de los comandos necesarios de control del robot, pasando por todo los pasos e instalaciones intermedias. La idea principal de esta guía es agilizar al alumno y que no pierda tiempo en tener que buscar y entender la información relativa al vehículo.

Para comenzar es necesario saber que existen junto a esta guía, otras 4 guías sobre las que haremos referencia a lo largo del documento.

-HARDWARE MANUAL: En ella se explica todo lo relativo al hardware instalado en el summit: Sensores, motores, placa, router, cámara, etc. Todo ello junto a unas detalladas especificaciones.

-SOFTWARE MANUAL: En este documentos encontraremos una ayuda para la familiarización con el entorno ROS y una introducción a la conexión entre el PC y el Summit XL

-QUICKSTART MANUAL: Esta guía está enfocada a empezar a manejar el Summit sin necesidad de conectarlo a ningún PC externo, simplemente utilizarlo con un control remoto. El control remoto elegido será un Dualshock 3.

-ESQUEMA ELECTRICO ROBOT SUMMIT XL INTA: Como su propio nombre indica este último manual contiene todos los planos electrónicos del vehículo. Es de suma utilidad si se desea ampliar el Summit con otros sensores o saber cómo funciona el conexionado de sus componentes.

En esta guía nos limitaremos a poner el PC y el Summit a punto para que cualquier persona sin ningún conocimiento de Linux ni ROS pueda empezar a trabajar en el robot sin ningún tipo de problema.

2) Descarga e instalación de Ubuntu 14.04-Linux

El ordenador que posee el Summit, es un sistema embebido que corre Ubuntu 14.04, por temas de compatibilidad intentaremos instalar en nuestro PC el mismo sistema, necesario para la conexión con el robot.

2.1) Preparación de USB de arranque

En primer lugar descargaremos la versión Desktop del SO Ubuntu 14.04 LTS desde el enlace oficial y gratuito.
https://www.ubuntu.com/download/desktop
Esto descargara una imagen iso relativamente grande que deberemos tener ubicada para poder encontrarla cuando sea necesario.

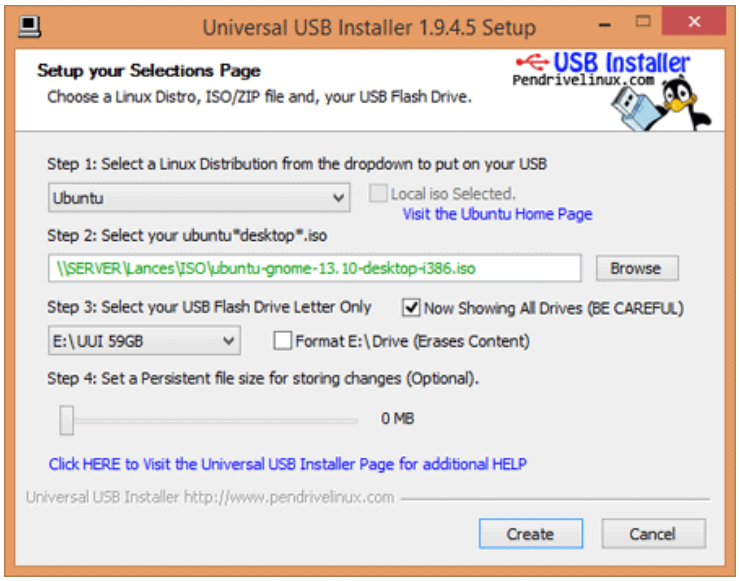
Descargamos a continuación el programa que creara el USB instalador. Lo instalamos según el wizard de manera trivial.
http://www.pendrivelinux.com/universal-usb-installer-easy-as-1-2-3/

Abrimos el programa y nos aparece una panatalla con 3 pasos a seguir:

-1- Desplegamos el menú y seccionamos 'Ubuntu'

-2- Hacemos click en 'Browse' y seleccionamos la imagen de Ubuntu que hemos descargado

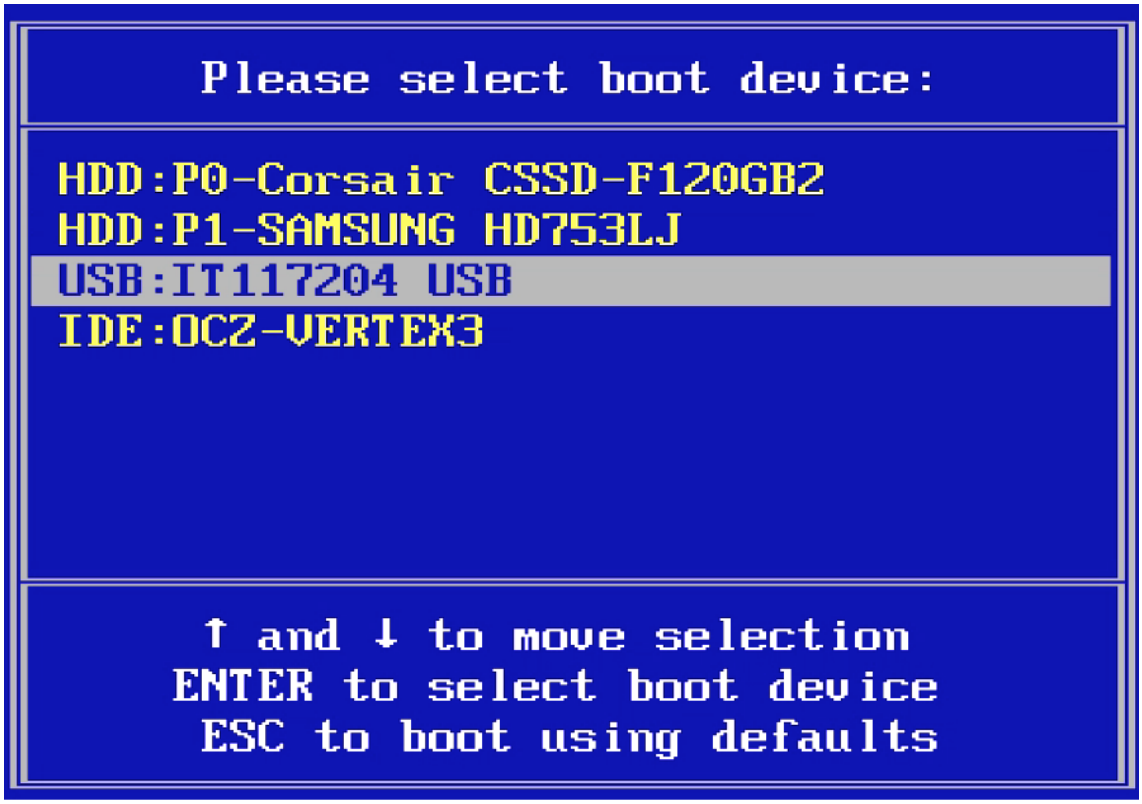
-3- Debemos seleccionar la unidad USB que vamos a usar y activar la casilla 'Format' en 'Step 3' esto eliminar todo el contenido de la unidad USB que tengamos insertada, debemos asegurarnos que está seleccionado la unidad USB en la que deseamos instalar Ubuntu y hacemos click en 'Create'



2.2) Instalación de Ubuntu

Una vez tenemos preparado nuestra memoria USB con la imagen de arranque de sistema operativo Ubuntu 14.04 es necesario reiniciar el ordenador con la memoria USB conectada. Según el modelo y la marca de nuestro PC necesitaremos usar esc, F8, F9 o alguna tecla similar para evitar que arranque el sistema operativo Windows por defecto.

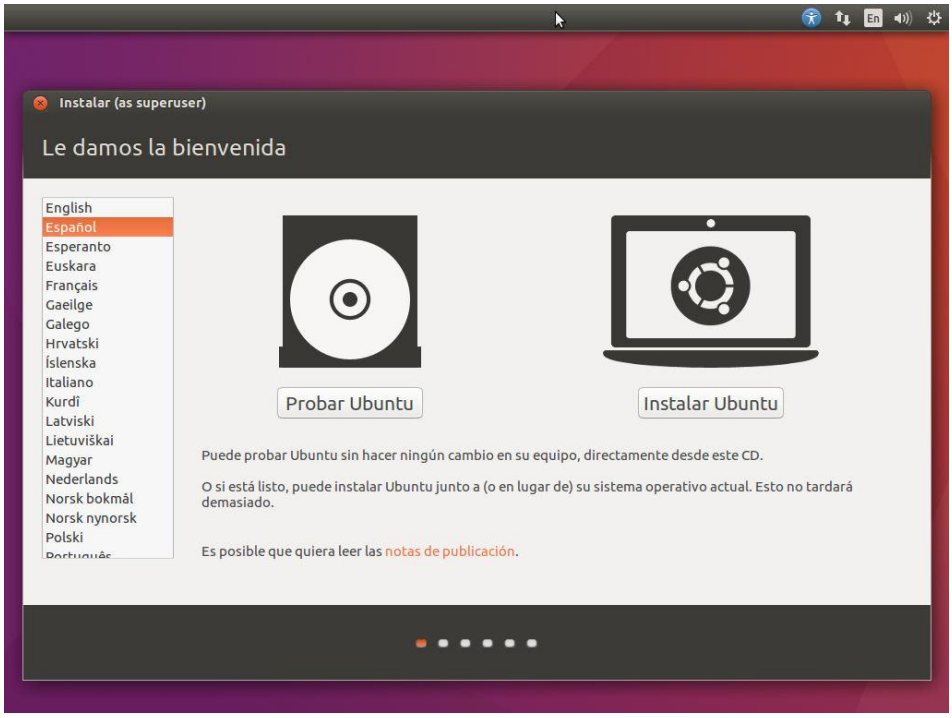
Cuando consigamos con éxito llegar al menú de boot de nuestra máquina nos aparecerá algo como esto:



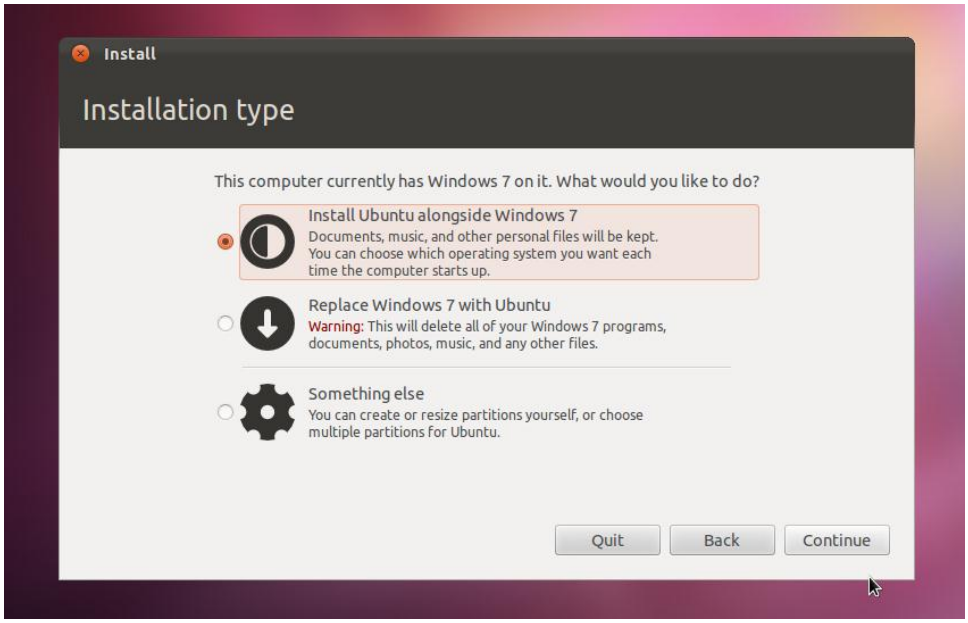
Es el menú de arranque (boot menu) donde tendremos que arrancar Ubuntu 14.04 desde la unidad de memoria USB donde lo hayamos instalado con anterioridad.

Dejamos que se inicie el SO Ubuntu y pasamos a la instalación.

Seleccionaremos el idioma que deseamos, podemos elegir entre probar Ubuntu o realizar una instalación. En nuestro caso necesitamos instalar Ubuntu en el PC. Intentando con ello no eliminar los otros SO que teníamos ya instalado en el pc como Windows en la mayoría de los casos



Seguimos paso a paso el wizard de instalación de Ubuntu hasta el paso más importante: Si nos interesa mantener otros sistemas operativos en el pc a la par que Ubuntu tendremos que escoger la opción que vemos en la imagen inferior. Instalará Ubuntu junto a en este caso en particular Windows 7.



Nos aseguraremos de ir completando el asistente de instalación sin más dificultad, escogeremos un nombre de usuario y aceptaremos el reinicio del pc cuando sea necesario. Y ya tendremos instalado Ubuntu.

3) Descarga e instalación de ROS Índigo

ROS (Robot Operative Sistem) es un entorno de programación que facilita la manipulación software del entorno de la robótica. Es aconsejable familiarizarse con el entorno de ROS y entender bien el concepto de cómo trabaja con múltiples nodos. Para lo cual existe una web http://wiki.ros.org/es de la que podemos obtener toda la información y tutoriales para ponernos al día con este entorno.

Para instalar ROS es necesario abrir el terminal de Ubuntu y desde ahí iremos tecleando comando para completar la instalación de ROS.

Todos estos pasos están descritos en http://wiki.ros.org/indigo/Installation/Ubuntu

En primer lugar hay que configurar la computadora para aceptar software de parte de 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'
```

En segundo lugar introduciremos las claves.

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

En tercer lugar actualizaremos el software necesario para correr ROS.

```
sudo apt-get update
```

La instalación por defecto viene con algunos paquetes de ROS básicos, en cualquier momento podremos descargar mas paquetes desde la web oficial como haremos más adelante con los paquetes del Summit XL.

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

La siguiente instalación es necesaria para utilizar rqt y rviz, imprescindibles en el Summit.

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

```
sudo apt-get install ros-indigo-ros-base
```

También será necesario instalar rosdep.

```
sudo rosdep init
rosdep update
```

Añadir las variables de trabajo de ROS

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

Es necesario instalar el instalador de ros por comando para hacer más fácil las instalaciones en el futuro

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

Con este último paso ya estaría en principio instalado ROS en nuestro PC, sería conveniente realizar alguno de los tutoriales que están en la página de wiki ROS para familiarizarnos con los comandos y la terminología de este entorno.
http://wiki.ros.org/ROS/Tutorials

Si en algún momento durante la instalación nos pidiera la contraseña debemos tener en cuenta que es la misma que se puso en el proceso de instalación y la misma que se utiliza para el inicio de sesión en Ubuntu.

Durante la instalación de ROS hay ciertas ocasiones en las que pide permiso al usuario de Ubuntu para realizar descargas, obviamente debemos aceptar cada petición y finalizar la instalación correctamente.

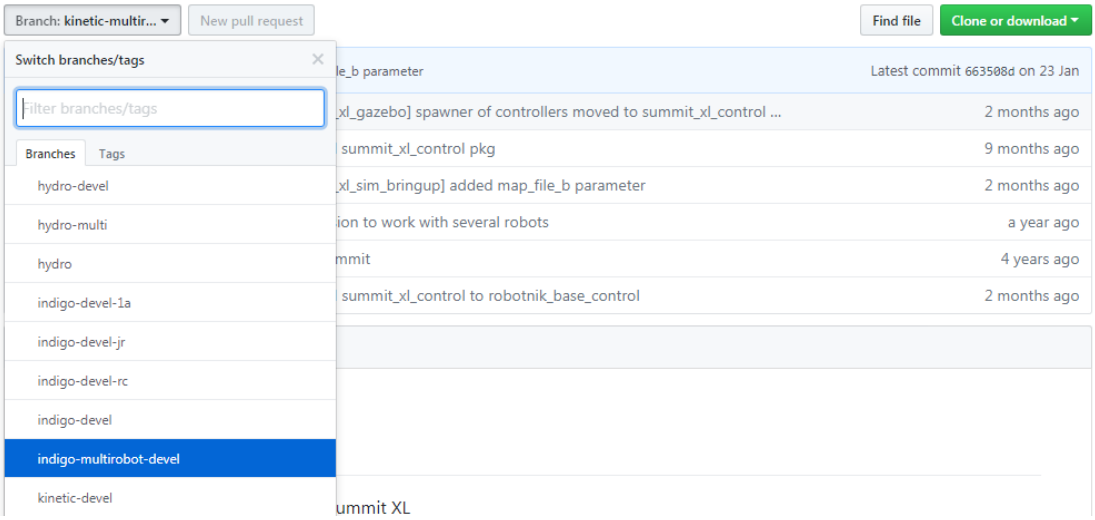
En particular es necesario hacer los tutoriales debido a que la terminología usada en esta guía da por hecho algunos conceptos de ROS que se explican y ejemplifican en sus tutoriales.

4) Descarga e instalación Paquetes Summit-XL

Para realizar simulaciones y control real sobre el robot es necesario descargarse algunos paquetes ROS exclusivos del Summit. Son: Summit_xl_robot_common y Summit_xl_sim. Y nos los podemos descargar oficial y gratuitamente desde: http://wiki.ros.org/Robots/SummitXL

Es importante descargar ambos paquetes. En esta página web se explica ampliamente en que consiste cada paquete, y en esta guía solo se explicará cómo instalarlos correctamente. Deberás llegar a un repositorio github como el del este enlace: https://github.com/RobotnikAutomation/summit_xl_sim

Una vez en la página de descarga por separado de cada paquete se deberá escoger la versión de ROS que está instalada, en el caso de este tutorial, ROS Índigo.



Los archivos descargados, es decir, los dos paquetes descomprimidos deberán guardarse en la carpeta *src* del directorio */home/catkin_ws* que hemos creados gracias a los tutoriales de ROS. Para más adelante poder instalarlos y usarlos con nuestro robot.

Antes de lanzar el paquete de simulación o el de control real de nuestro Summit XL es necesario instalar desde el terminal algunos paquetes ROS que también serán usados en el control del robot.

```
sudo apt-get install ros-indigo-mavlink ros-indigo-mavros ros-indigo-
    realtime-tools ros-indigo-robot-localization ros-indigo-controller-
    manager ros-indigo-transmission-interface ros-indigo-joint-limits-
    interface ros-indigo-control-toolbox ros-indigo-twist-mux ros-indigo-
    slam-gmapping ros-indigo-gazebo-ros-pkgs ros-indigo-robotnik-msgs
    ros-indigo-robotnik-sensors ros-indigo-uuid-msgs ros-indigo-
    geographic-msgs
```

Compilaremos a continuación todos los paquetes que guardamos anteriormente en nuestro directorio catkin_ws:

```
cd ~/catkin_ws
catkin_make
```

Con estos pasos ya estaría listo todo para la simulación del Summit XL

5) Introducción a la simulación

Después de seguir todos los pasos marcados anteriormente todo debería estar listo para lanzar la simulación. La simulación se lanza abriendo un terminal y lanzando la siguiente línea:

```
roslaunch summit_xl_sim_bringup summit_xl_complete.launch
```

Si nos aparece un error de existencia de paquete, algo así: "[] is neither a launch file in package []…" deberemos ubicar el setup.bash en la carpeta catkin del siguiente modo:

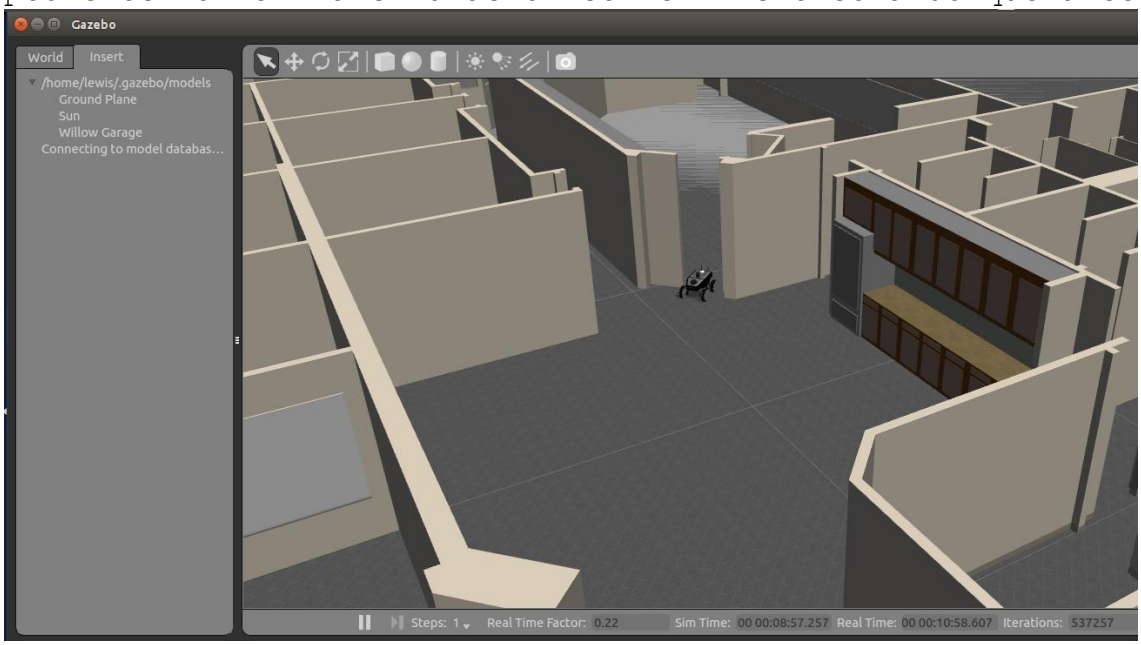
En el terminal escribiremos :

```
$ cd /directorio_donde_esta_catkin_ws
```

y una vez movidos al directorio

```
$ source devel/setup.bash
```

Una vez movido el setup.bash a nuestro directorio catkin_ws podremos lanzar la simulación con el mismo comando que antes



Se lanzara Gazebo que es un entorno de simulación donde gracias a los paquetes ROS "Summit_xl_sim" estará implementado el Summit_XL y podremos añadirle entorno para interactuar y simular lo que necesitemos.

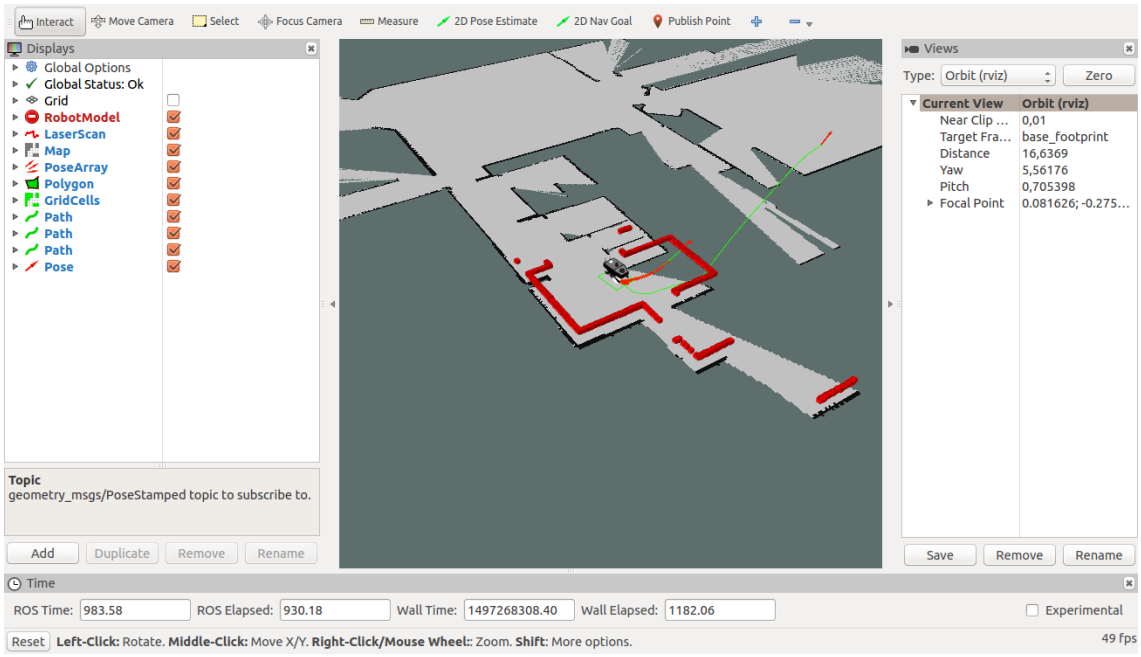
Una vez abierto Gazebo el siguiente paso sería abrir el Rviz(necesita que Gazebo siga abierto para funcionar) donde podremos visualizar el entorno donde se está moviendo el Summit, la imagen de la cámara, el par motor y en definitiva todos los

topics que están enviando continuamente los sensores del robot y que son visibles gracias a la herramienta Rviz

Para correr este programa solo necesitamos abrir un nuevo terminal y escribir en él

```
$  rosrun rviz rviz
```

Y aparecerá algo como esto:



Desde Rviz, con el botón Add (abajo a la izquierda) se desplegará una ventana de Topics, que no son más que todas las señales que nos ofrece el Summit XL para ser leídas.

Podríamos añadir un mapa, una imagen de cámara y un robot model para visualizar al robot en el entorno de Rviz

Se profundizará algo más en la herramienta Rviz cuando lleguemos a la parte de control real del Summit.

5) Control del Summit XL

En este apartado se explicarán los conceptos necesarios para poder controlar el robot, bien sea de manera manual o autónoma

Se empezara explicando el encendido del Summit:

Como se muestra en la imagen superior el Summit consta de varios interruptores y puertos, todos ellos explicados en la guía HARDWARE MANUAL. En este caso se hará énfasis sobre los elementos necesarios para el arranque de sistema.

En primer lugar deberemos pulsar el interruptor rojo, que no es más que un pulsador de emergencia que inhabilita los motores. Este paso es necesario por si ocurre algún problema en el encendido, que el robot no se mueva.

En segundo lugar giramos el interruptor verde a ON. Con esto la batería estará suministrando energía al sistema.

Por último pulsamos el botón azul, que es el encendido de la cpu y esperamos aproximadamente un minuto para que arranque el sistema correctamente.

Tendríamos encendido entonces el Summit XL y todo listo para empezar a controlarlo.

5.1) Control manual

Para controlar el Summit manualmente solo necesitamos el dualshock controller:

En primer lugar encendemos el controlador pulsando dos segundos el botón "Start button". Con esto ya tendremos encendido nuestro mando. No se debe olvidar girar el botón rojo de emergencia para desbloquear los motores y poder dirigir al robot manualmente con el mando.

En cuanto a los controles, vienen especificados en el QUICKSTART MANUAL y son bastante intuitivos.

No debemos olvidar en ningún caso que para que el Summit funcione con el mando bluetooth es necesario dejar pulsado siempre el "Dead man button". Botón de seguridad para que el robot no sea movido por error.

5.2) Control autónomo

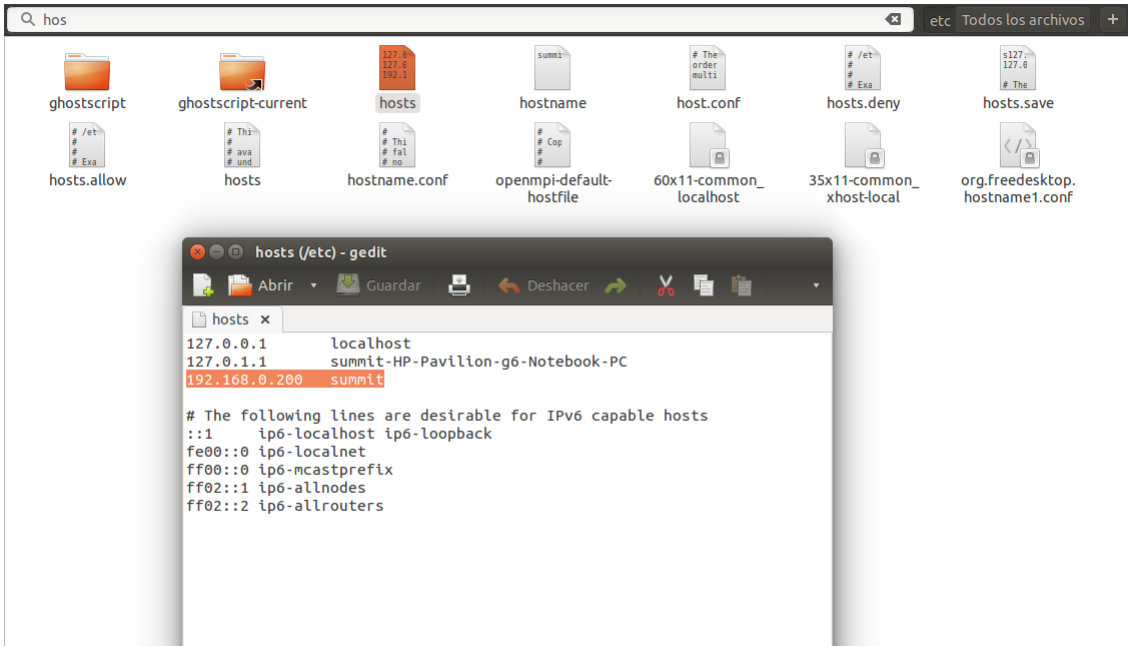
Para la realización de un control autónomo del robot será necesaria una conexión previa con nuestro pc. Para conectar el Summit XL al pc Ubuntu solo necesitamos configurar una conexión WIFI. Esto se debe a que el robot posee un router WIFI al que podemos conectarnos con nuestro ordenador para ver qué ocurre en el pc interno del Summit y correr en el los paquetes de ROS que necesitemos como el de navegación autónoma que veremos en este capítulo.

Necesitaremos además un último paquete de ROS para la navegación, escribimos en un terminal:

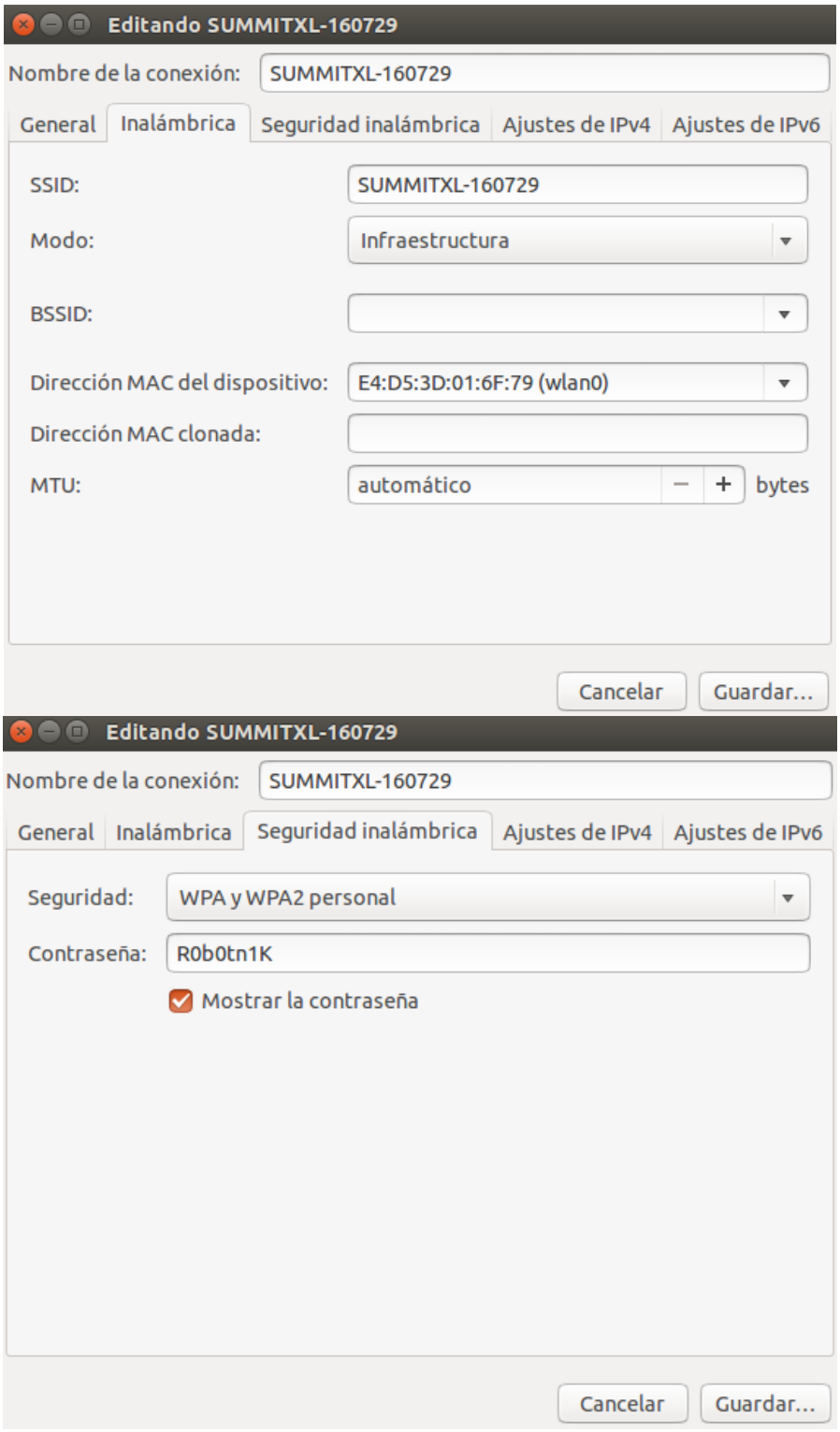
```
sudo apt-get install ros-indigo-robotnik-trajectory-suite
```

5.2.1) Conexión WIFI

En primer lugar será necesario modificar el archivo /etc/host que se encuentra en nuestro pc Ubuntu.

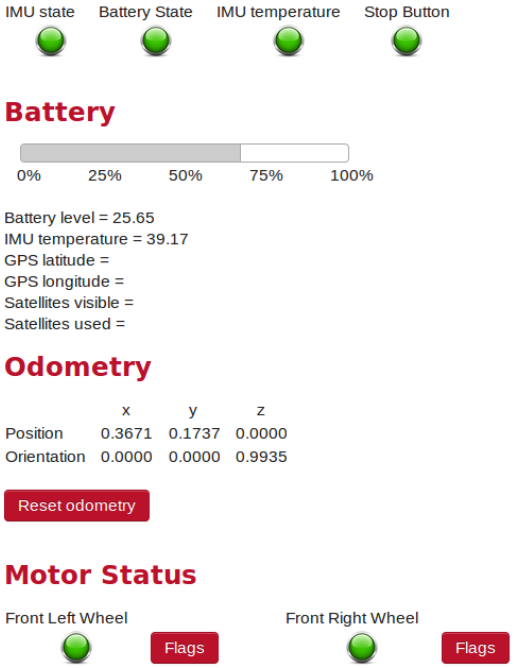


Le añadiremos la dirección ip del Summit como se muestra en la imagen.
A continuación escaneamos las redes WIFI disponible desde nuestri pc, deberemos conectarnos a SUMMITXL-160729, creando una conexión inalámbrica pulsando en "editar las conexiones", opción que se encuentra clicando sobre el icono de WIFI de ubuntu.
La configuramos de la siguiente manera:



A partir de ahora tendremos en cuenta que la contraseña para todo en el Summit XL será R0b0tn1K.

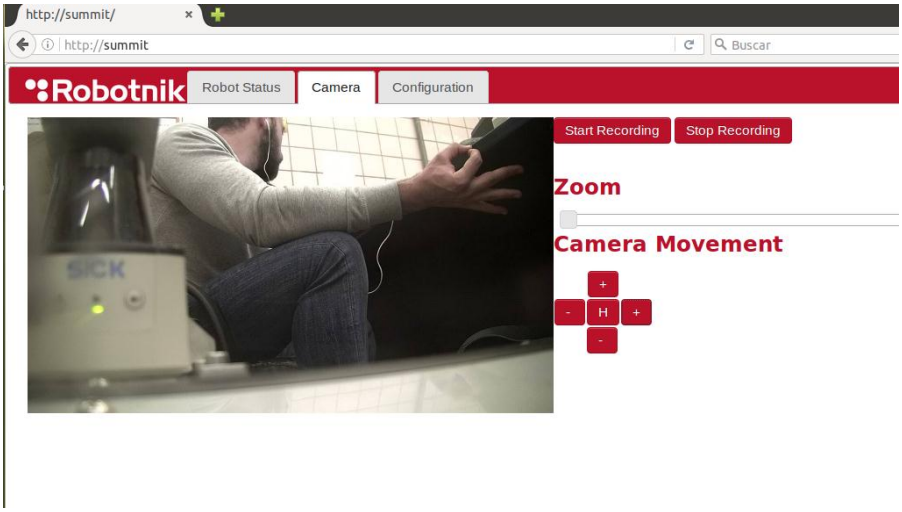
Una vez conectados a la red WIFI del Summit se procederá a comprobar si efectivamente estamos conectados y el robot nos está mandando la información necesaria. Vía HTML el Summit nos está enviando una serie de datos que podemos comprobar accediendo a http://summit donde se nos mostrara desde el navegador la siguiente información:



Comprobamos gracias a la imagen de la izquierda que el Summit nos está enviando el estado de la batería, la IMU y el botón de stop, así como los niveles de batería, las medidas de odometría y el estado de los motores.

En la parte superior encontramos 3 pestañas que podremos clicar e ir navegando por los distintos menús que ofrece el Summit

En la imagen de la derecha se aprecia la captura de la cámara de video del Summit XL en tiempo real con posibilidad de grabación y con botones para mover la cámara desde este servidor web



Una vez estamos seguros que se ha establecido una conexión WIFI con el robot procedemos a abrir un terminal y escribimos lo siguiente:

```
export ROS_MASTER_URI = http://summit:11311

rostopic list
```

Esto hará que ROS corra el nodo core o nodo principal en el pc del Summit XL, si todo ha salido bien seremos capaces de listar todos los topics que el robot está transmitiendo vía WIFI.

A continuación en el terminal la siguiente orden:

```
ssh summit@summit
sudo vim /etc/hosts
... // Adding the remote PC hostname and ip address here
```

Al escribir ssh estamos intentando monitorizar desde nuestro pc lo que pasa dentro del pc del Summit es decir a partir de ahí nuestro ordenador será un reflejo de lo que pasa en el Summit. **Cambiará el usuario que tengamos en nuestro pc por "summit@summit"** en el ámbito del terminal.
Con lo que ahora deberemos modificar el archivo /etc/hosts con el comando vim, es **importante entender que estamos modificando el Summit** cuando iniciamos la sesión en el con el comando ssh.

Como se ha trabajado en años anteriores con el Summit, es muy posible que ya haya otros ordenadores de otros alumnos en el hosts del Summit, valdrá con dejar estos hostnames comentados y escribir nuestro hostname en el archivo hosts del Summit.

También es importante saber cómo funciona el comando Vim que utilizamos para modificar un archivo del Summit desde el terminal. Para usarlo de manera adecuada es recomendable buscar información en internet de como modificar archivos de texto desde el terminal de Ubuntu.

Para este paso y todos los demás pasos dentro del Summit desde el terminal **la contraseña sigue siendo R0b0tn1K.**

5.2.1) Conexión WIFI

El siguiente paso sería abrir rqt graph y Rviz para visualizar con más claridad que está ocurriendo en ROS entre nuestro PC y el Summit.

Para ello necesitamos escribir en el mismo terminal:

```
rosrun gmapping slam_gmapping &
rosrun map_server map_saver <map> // It saves the map with the name
   given in <"map">
```

Este comando ira creando un mapa en 2D de lo que capta el sensor Sick del robot en sus múltiples barridas.

Lo siguiente será **abrir un nuevo terminal** y dejar corriendo en el terminal anterior el slam_gmapping. En el nuevo terminal tendremos que escribir:

```
$ ssh –X summit@summit
```

Con este comando entraremos en el servidor X del Summit, desde el cual si podremos lanzar rqt graph y Rviz como se pretendía.

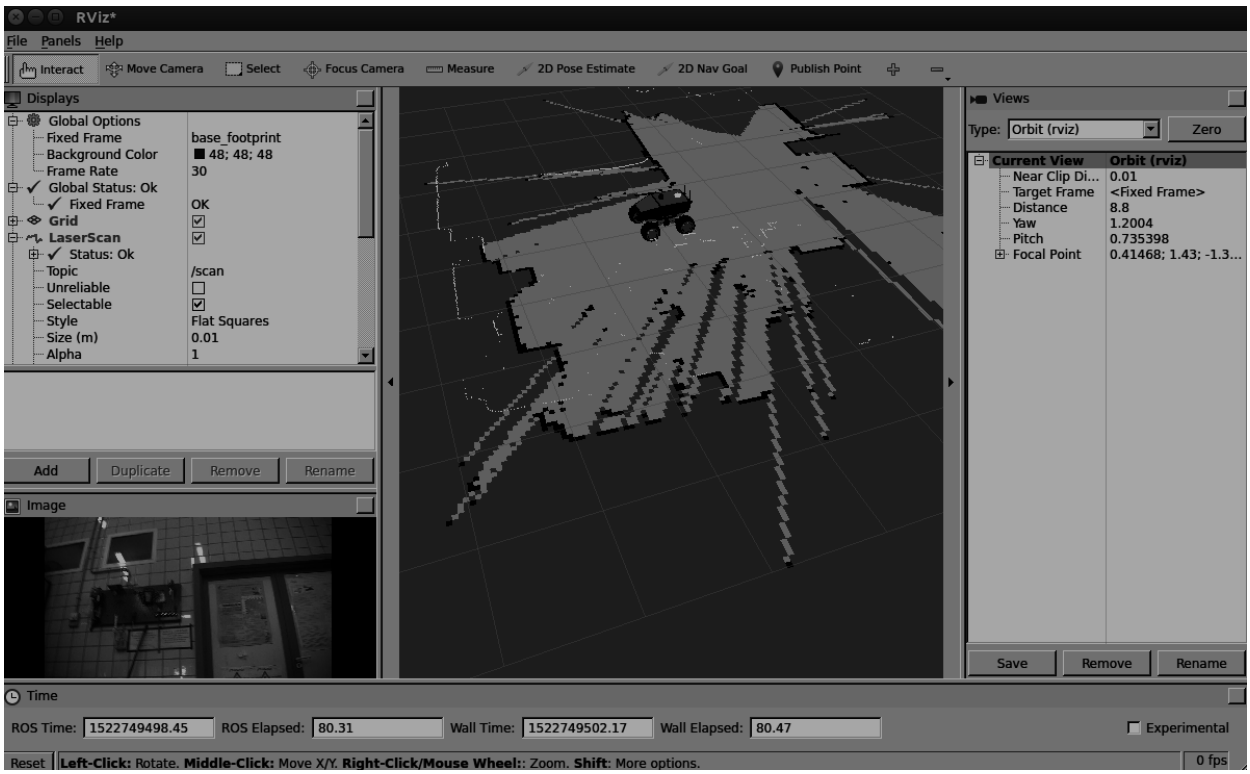
```
rqt &
rosrun rviz rviz
```

Tendremos entonces dos terminales abiertos: uno corriendo el slam_gmapping y otro corriendo Rviz con rqt graph. Abrimos a continuación un nuevo terminal donde correremos el guiado autónomo del robot con control de trayectoria y esquive de obstáculo hasta una meta.

Lo primero al abrir el terminal será volver a hacer ssh summit@summit para estar dentro del ordenador del robot y entonces tecleamos:

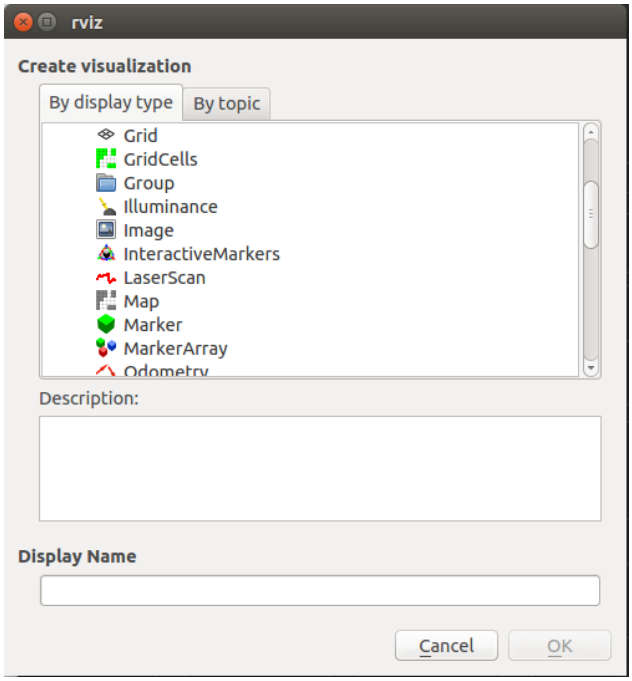
```
roslaunch summit_xl_navigation move_base_nomap.launch
```

Cuando todos los programas y paquetes están lanzados en los 3 terminales correspondientes podemos usar el programa Rviz para la navegación autónoma de la siguiente manera:



Se visualizara la interfaz gráfica de Rviz donde podremos añadir todos los topics que queramos para visualizarlos junto a nuestro Summit XL.

Deberemos añadir cada topic para su visualización pulsando el botón "add", es recomendable añadir:



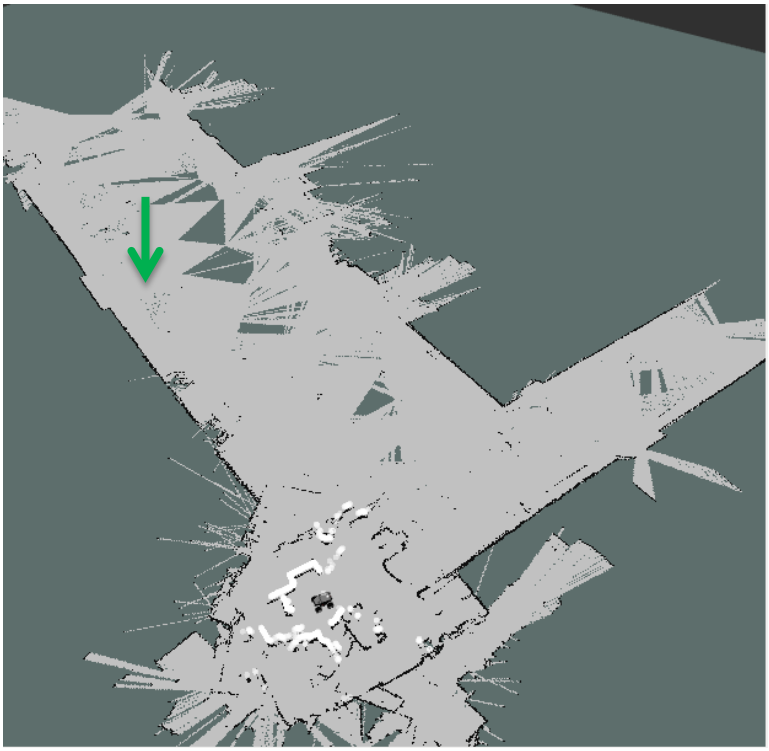
-Cámara (Deberemos clicar la opción "by topic" en la ventana de diálogo que se abre al pulsar el botón "add" y elegir el topic de dato de cámara que nos interese, por ejemplo, imagen a color).

-Robot model (Simplemente al pulsar "add" descendemos hasta encontrar lo que necesitamos dentro de la ventana "by display type").

-Mapa (En la ventana "by display type").

Una vez añadidas estas visualizaciones de los topics que envía el robot pasaremos a la navegación autónoma.

Para este último paso de Navegación autónoma necesitaremos haber recorrido previamente el entorno del Summit de manera manual con el mando, para así crear un mapa del entorno por el cual se realizará la navegación autónoma. Al menos la primera vez que el robot se exponga a ese entorno en concreto, más tarde podremos guardar el entorno y cargarlo en otro momento con los comandos que se han explicado.



Una vez tenemos el entorno registrado en el mapa solo necesitamos pulsar el botón "2D Nav Goal", y a continuación pulsar en la parte del mapa a la que deseamos ir. Si todo está correcto, nuestro robot se encargará de llegar al punto descrito y esquivar los obstáculos espontáneos que surjan en su camino.

APÉNDICE B
CÓDIGOS ARDUINO Y ROS

Código B.1: Lectura de sensor de intensidad (Arduino UNO)

```
/*********************************************************
Código para Arduino UNO
Lee la intensidad del sensor Winson WCS1800
Envía por el puerto 2 Digital la medida leída
*********************************************************/

#include <SoftwareSerial.h>

SoftwareSerial BT1(3, 2);  // recibe / envía

void setup()
{
     Serial.begin(115200);
     BT1.begin(115200);
}

void loop()
{
  float SensorValue = analogRead(A0);
  float SensorValuefin;

  SensorValuefin=((((SensorValue)*5/1023.0)-2.43)/0.066)+0.5;

  //
  // 5=Vdd / 1023 CAD Arduino/
  //-2,41 Offset (mitad de 0 a 5V) / 0.066V/A Sens
  //

  BT1.print(SensorValuefin); //Envio a ESP8266
  Serial.println(SensorValuefin);
  delay(500);

}
```

Código B.2: Comunicaciones WiFi (ESP8266 Feather Huzzah)

```
/*****************************************************
  Programa para ESP8266 Huzza Feather módulo WiFi.
  Conecta con una red WiFi, busca la IP asignada y el puerto asignado
  para transmitir un topic ROS (float64) repetidamente vía WiFi a un pc
  el cual corre rosserial_pyton TCP.
  **quitar el pin rx cuando se está introduciendo un programa
*****************************************************/

//*********************************
//Parámetros WiFi
//const char* ssid     = "SUMMITXL-160729";
//const char* password = "R0b0tn1K";
//const char* host = "192.168.0.50";
//*********************************

#include <SoftwareSerial.h>
#include <ESP8266WiFi.h>
#include <ros.h>
#include <std_msgs/String.h>
#include <std_msgs/Int32.h>
#include <std_msgs/Float64.h>



////////////////////////// AJUSTES WIFI //////////////////////////

const char* ssid = "SUMMITXL-160729";
const char* password = "R0b0tn1K";

IPAddress server(192, 168, 0, 50); // ip de tu Servidor ROS
IPAddress ip_address;
int status = WL_IDLE_STATUS;

WiFiClient client;

class WiFiHardware {
  public:
  WiFiHardware() {};
  void init() {
    client.connect(server, 11411);
    //Puerto de escucha de rosserial TCP
  }

  int read() { // Función read es -1 = failure
    return client.read();         //Devuelve -1 si todo va bien
  }

  // Escribe datos a ROS
  void write(uint8_t* data, int length) {
    for(int i=0; i<length; i++)
      client.write(data[i]);
  }
```

```
  // Devuelve los ms desde el inicio del programa
  unsigned long time() {
     return millis();
  }

};

int i;
//////////////////////////////////////////////////////////

/////////////////////////// ROS//////////////////////////////
std_msgs::Float64 amp;
ros::Publisher chatter("Intensidad", &amp);
ros::NodeHandle_<WiFiHardware> nh;
//////////////////////////////////////////////////////////

////////////////////PROGRAMACIÓN WIFI/////////////////////
void setupWiFi()
{
  WiFi.begin(ssid, password);
  Serial.print("\nConnecting to "); Serial.println(ssid);
  uint8_t i = 0;
  while (WiFi.status() != WL_CONNECTED && i++ < 20) delay(500);
  if(i == 21){
    Serial.print("Could not connect to"); Serial.println(ssid);
    while(1) delay(500);
  }
  Serial.print("Ready! Use ");
  Serial.print(WiFi.localIP());
  Serial.println(" to access client");
}
//////////////////////////////////////////////////////////

////////////////////////SETUP///////////////////////////////
void setup() {
  Serial.begin(115200);
  setupWiFi();
  delay(2000);
  nh.initNode();
  nh.advertise(chatter);
}
//////////////////////////////////////////////////////////

//////////////////////////LOOP////////////////////////////
void loop() {

  float data = Serial.parseFloat();
  amp.data=(float)data;
  Serial.println(amp.data);
  chatter.publish( &amp );
  nh.spinOnce();
  delay(500);
}
```

Código B.3: Lectura de sensor de intensidad y comunicación con EMS (Arduino UNO)

```
/*************************************************************
Código para Arduino UNO.
1 - Lee la intensidad del sensor WCS1800
2 - Envía por el puerto 2 Digital la medida leída
3 - Envía petición al EMS por puerto 4
4 - Recibe del EMS parámetros separados por '/' y termina en '*'
Decodifica el mensaje y manda por puerto 2 Digital todas las medidas
*************************************************************/

#include <SoftwareSerial.h>
SoftwareSerial BT1(3, 2); // recibe / envía CONEXION CON ESP8266
SoftwareSerial BT2(5, 4); //               CONEXION CON EMS

void setup()
{
     Serial.begin(115200);
     BT1.begin(115200);
     BT2.begin(9600); //Baudios del EMS 9600 para EMS
}

void loop()
{
 int i;

  ////////////////////SENSOR INTESIDAD////////////////////////

  float SensorValue = analogRead(A0);
  float SensorValuefin;

  SensorValuefin=((((SensorValue)*5/1023.0)-2.43)/0.066);

  //
  // 5=Vdd / 1023 CAD Arduino/
  //-2,41 Offset (mitad de 0 a 5V) / 0.066V/A Sens
  //


//////////////////////////////////////////////////////////////////


//////////////////////////////////EMS/////////////////////////////

  BT2.print("&");// Comando de peticiónn de trama de respuesta al EMS

  if (BT2.available() > 0) //Solo se mete si hay datos en el buffer
  {
    BT1.print(0.0000); //Enviamos un cero para avisar de que vamos a
                       //enviar una ristra de valores del EMS
    for(i=0;i<20;i++)
    {
      while (BT2.find("/"))
```

```
        {
          float buf = BT2.read(); //Guardamos la cadena de números
                                  //separados por '/' y enviamos de uno
                                  //en uno
          BT1.print(buf);
        }
      }
    }


////////////////////////////////////////////////////////////////////////
/////////////

    BT1.print(SensorValuefin);// Mandamos también la señal de
                              // intensidad del sensor




    delay(500);


}
```

Código B.4: Comunicaciones WiFi versión EMS (Arduino UNO)

```
/*****************************************************
Programa para ESP8266 Huzza Feather módulo WiFi.
Conecta con una red WiFi, busca la IP asignada y el puerto asignado
para transmitir un topic ROS (float64) repetidamente vía WiFi a un pc
el cual corre rosserial_pyton TCP.
**quitar el pin rx cuando se está introduciendo un programa
Version EMS: Recibe de Arduino UNO el parámetro Intensidad y otros 20
parámetros más con los que creará 20 topics
*****************************************************/

 //*********************************
 //Parámetros WiFi
 //const char* ssid     = "SUMMITXL-160729";
 //const char* password = "R0b0tn1K";
 //const char* host = "192.168.0.50";
 //*********************************

 #include <SoftwareSerial.h>
 #include <ESP8266WiFi.h>
 #include <ros.h>
 #include <std_msgs/String.h>
 #include <std_msgs/Int32.h>
 #include <std_msgs/Float64.h>



 ////////////////////////// AJUSTES WIFI ////////////////////////

 const char* ssid = "SUMMITXL-160729";
 const char* password = "R0b0tn1K";

 IPAddress server(192, 168, 0, 50); // ip de tu Servidor ROS
 IPAddress ip_address;
 int status = WL_IDLE_STATUS;

 WiFiClient client;

 class WiFiHardware {
   public:
   WiFiHardware() {};
   void init() {
     client.connect(server, 11411);
     //Puerto de escucha de rosserial TCP
   }

   int read() { // Función read es -1 = failure
     return client.read();         //Devuelve -1 si todo va bien
   }

   // Escribe datos a ROS
   void write(uint8_t* data, int length) {
     for(int i=0; i<length; i++)
```

```
        client.write(data[i]);

    }

    // Devuelve los ms desde el inicio del programa
    unsigned long time() {
       return millis();
    }

};

int i;
//////////////////////////////////////////////////////

/////////////////////////// ROS//////////////////////////////
std_msgs::Float64 amp;
ros::Publisher chatter("Intensidad", &amp);

std_msgs::Float64 Vbat;
ros::Publisher chatter1("Vbat", &Vbat);

std_msgs::Float64 Ibat;
ros::Publisher chatter2("Ibat", &Ibat);

std_msgs::Float64 Temp_bat;
ros::Publisher chatter3("Temp_bat", &Temp_bat);

std_msgs::Float64 Vfc;
ros::Publisher chatter4("Vfc", &Vfc);

std_msgs::Float64 Ifc;
ros::Publisher chatter5("Ifc", &Ifc);

std_msgs::Float64 Temp_fc;
ros::Publisher chatter6("Temp_fc", &Temp_fc);

std_msgs::Float64 Presion;
ros::Publisher chatter7("Presion", &Presion);

std_msgs::Float64 Charge;
ros::Publisher chatter8("Charge", &Charge);

std_msgs::Float64 Discharge;
ros::Publisher chatter9("Discharge", &Discharge);

std_msgs::Float64 OV;
ros::Publisher chatter10("OV", &OV);

std_msgs::Float64 UV;
ros::Publisher chatter11("UV", &UV);

std_msgs::Float64 OT;
ros::Publisher chatter12("OT", &OT);

std_msgs::Float64 Fan_bat;
ros::Publisher chatter13("Fan_bat", &Fan_bat);
```

```
  std_msgs::Float64 Fan_fc;
  ros::Publisher chatter14("Fan_fc", &Fan_fc);

  std_msgs::Float64 PWM;
  ros::Publisher chatter15("PWM", &PWM);

  std_msgs::Float64 Purge;
  ros::Publisher chatter16("Purge", &Purge);

  std_msgs::Float64 Fc;
  ros::Publisher chatter17("Fc", &Fc);

  std_msgs::Float64 H2;
  ros::Publisher chatter18("H2", &H2);

  std_msgs::Float64 Error_t;
  ros::Publisher chatter19("Error_t", &Error_t);

  std_msgs::Float64 Error_H2;
  ros::Publisher chatter20("Error_H2", &Error_H2);

  ros::NodeHandle_<WiFiHardware> nh;

  ////////////////////////////////////////////////////////////

  ////////////////////PROGRAMACIÓN WIFI///////////////////////
  void setupWiFi()
  {
    WiFi.begin(ssid, password);
    Serial.print("\nConnecting to "); Serial.println(ssid);
    uint8_t i = 0;
    while (WiFi.status() != WL_CONNECTED && i++ < 20) delay(500);
    if(i == 21){
      Serial.print("Could not connect to"); Serial.println(ssid);
      while(1) delay(500);
    }
    Serial.print("Ready! Use ");
    Serial.print(WiFi.localIP());
    Serial.println(" to access client");
  }
  ////////////////////////////////////////////////////////////

  ////////////////////SETUP///////////////////////////////////
  void setup() {
     Serial.begin(115200);
    setupWiFi();
    delay(2000);
    nh.initNode();
nh.advertise(chatter);nh.advertise(chatter1);nh.advertise(chatter2);nh
.advertise(chatter3);nh.advertise(chatter4);nh.advertise(chatter5);nh.
advertise(chatter6);nh.advertise(chatter17);nh.advertise(chatter8);nh.
advertise(chatter9);nh.advertise(chatter10);nh.advertise(chatter11);nh
.advertise(chatter12);nh.advertise(chatter13);nh.advertise(chatter14);
nh.advertise(chatter15);nh.advertise(chatter16);nh.advertise(chatter17
);nh.advertise(chatter18);nh.advertise(chatter19);nh.advertise(chatter
```

```
20);}  //Advertir de los 20 nuevos chatter (publicadores de TOPIC)
  ////////////////////////////////////////////////////////////

  ////////////////////////////LOOP//////////////////////////////
  void loop() {

    float data = Serial.parseFloat();

  //Bifurcamos la decisión según si tenemos o no disponibles los datos
del EMS o si solo tenemos el dato de intensidad. Publicaremos todos o
solo el de intensidad.

    if (data==0.0000) //Leemos y publicamos los 20 datos
    {

      float data = Serial.parseFloat();
      Vbat.data=(float)data;
      chatter1.publish( &Vbat );
      nh.spinOnce();

       data = Serial.parseFloat();
      Ibat.data=(float)data;
      chatter2.publish( &Ibat );
      nh.spinOnce();

        data = Serial.parseFloat();
      Temp_bat.data=(float)data;
      chatter3.publish( &Temp_bat );
      nh.spinOnce();

        data = Serial.parseFloat();
      Vfc.data=(float)data;
      chatter4.publish( &Vfc );
      nh.spinOnce();

        data = Serial.parseFloat();
      Ifc.data=(float)data;
      chatter5.publish( &Ifc );
      nh.spinOnce();

        data = Serial.parseFloat();
      Temp_fc.data=(float)data;
      chatter6.publish( &Temp_fc );
      nh.spinOnce();

        data = Serial.parseFloat();
      Presion.data=(float)data;
      chatter7.publish( &Presion );
      nh.spinOnce();

        data = Serial.parseFloat();
      Charge.data=(float)data;
      chatter8.publish( &Charge );
      nh.spinOnce();

        data = Serial.parseFloat();
```

```
Discharge.data=(float)data;
chatter9.publish( &Discharge );
nh.spinOnce();

 data = Serial.parseFloat();
OV.data=(float)data;
chatter10.publish( &OV );
nh.spinOnce();

  data = Serial.parseFloat();
UV.data=(float)data;
chatter11.publish( &UV );
nh.spinOnce();

  data = Serial.parseFloat();
OT.data=(float)data;
chatter12.publish( &OT );
nh.spinOnce();

  data = Serial.parseFloat();
Fan_bat.data=(float)data;
chatter13.publish( &Fan_bat );
nh.spinOnce();

  data = Serial.parseFloat();
Fan_fc.data=(float)data;
chatter14.publish( &Fan_fc );
nh.spinOnce();

  data = Serial.parseFloat();
PWM.data=(float)data;
chatter15.publish( &PWM );
nh.spinOnce();

  data = Serial.parseFloat();
Purge.data=(float)data;
chatter16.publish( &Purge );
nh.spinOnce();

  data = Serial.parseFloat();
Fc.data=(float)data;
chatter17.publish( &Fc );
nh.spinOnce();

  data = Serial.parseFloat();
H2.data=(float)data;
chatter18.publish( &H2 );
nh.spinOnce();

  data = Serial.parseFloat();
Error_t.data=(float)data;
chatter19.publish( &Error_t );
nh.spinOnce();

  data = Serial.parseFloat();
Error_H2.data=(float)data;
```

```
      chatter19.publish( &Error_H2 );
      nh.spinOnce();
    }

    else //Leemos la intensidad de la bateria
    {
      amp.data=(float)data;
      chatter.publish( &amp );
      nh.spinOnce();
    }

    delay(1000);}
```

Código B.5: Prueba de servidor TCP (Computadora)

```
 #include<stdio.h>
 #include<string.h>
 #include<stdlib.h>
 #include<unistd.h>
 #include<sys/socket.h>
 #include<arpa/inet.h>
 #include<netinet/in.h>
 #include<netdb.h>

 int main(int argc, char **argv){

   if(argc<2)
   {
     printf("%s <puerto> ,prueba con 55555\n",argv[0]);
                      //Especifica los argumentos
     return 1;
   }
   int conexion_servidor, conexion_cliente, puerto;
                      //declaramos las variables

   socklen_t longc;  //Debemos declarar una variable que contendrá
                     //la longitud de la estructura

   struct sockaddr_in servidor, cliente;
   char buffer[100]; //Declaramos una variable que contendrá
                     //los mensajes que recibamos

   puerto = atoi(argv[1]);
   conexion_servidor = socket(AF_INET, SOCK_STREAM, 0);
                      //creamos el socket

   bzero((char *)&servidor, sizeof(servidor));
                     //llenamos la estructura de 0's

   servidor.sin_family = AF_INET; //asignamos a la estructura
   servidor.sin_port = htons(puerto);
   servidor.sin_addr.s_addr = INADDR_ANY;
                     //esta macro especifica nuestra dirección


   if(bind(conexion_servidor,    (struct    sockaddr    *)&servidor,
sizeof(servidor)) < 0)
   {                 //asignamos un puerto al socket
     printf("Error al asociar el puerto a la conexion\n");
     close(conexion_servidor);
     return 1;
   }

   listen(conexion_servidor, 3); //Estamos a la escucha
   printf("A     la     escucha     en     el     puerto     %d\n",
ntohs(servidor.sin_port));
```

```
    longc = sizeof(cliente); //Asignamos el tamano de la estructura

    conexion_cliente  =  accept(conexion_servidor,  (struct  sockaddr
*)&cliente, &longc); //Esperamos una conexión

    if(conexion_cliente<0)
    {
      printf("Error al aceptar trafico\n");
      close(conexion_servidor);
      return 1;
    }

    printf("Conectando                con                %s:%d\n",
inet_ntoa(cliente.sin_addr),htons(cliente.sin_port));

  while(1) //PARA NO CERRAR LA CONEXIÓN y seguir recibiendo
  {
    if(recv(conexion_cliente, buffer, 100, 0) < 0)
    {
      //Comenzamos a recibir datos del cliente
      //Si recv() recibe 0 el cliente ha cerrado la conexion. Si es
      //menor que 0 ha habido algún error.

      printf("Error al recibir los datos\n");
      close(conexion_servidor);
      return 1;
    }
    else
    {
      printf("%s\n", buffer);
      bzero((char *)&buffer, sizeof(buffer));
      send(conexion_cliente, "Recibido\n", 13, 0);
    }
  }
    //close(conexion_servidor);
    return 0;


}
```

REFERENCIAS Y BIBLIOGRAFÍA

INTA. (2018) ,"Sistema de control y gestión de energía en plataformas robóticas Summit XL y Husky", Edición l.

Robotnik. (2003). "Summit XL Datasheet".

Robotnik. (2003). "Summit XL mobile platform: Hardware Manual".

Robotnik. (2018). https://www.robotnik.es/robots-moviles/summit-xl/

Willows Garage. (2018). https://www.wiki.ros.org/

Universidad de Alicante (2015). https://moodle2015-16.ua.es/

Arduino (2018). https://playground.arduino.cc

Arduino (2018). https://www.arduino.cc

Kokam (2018). https://www.kokam.com/ess-3/

BCH (2018). http://www.bchpower.com/en/QingNengDianDui_v109.html

W. Kennedy, National Institutes of Health (2002). "NIH SBIR 2 R44 HD041781-02"

Winson (2017). "Winson WCS1800 Datasheet"

IBM (2018). https://www.ibm.com/knowledgecenter

Michael J. Donahoo, Kenneth L.Calvert , "TCP/IP Sockets in C". The Practical Guide Series

Espressif inc (2018) , "ESP8266 AT Instruction Set". IOT Team

Keysight (2018). "https://www.keysight.com"

ROS (2018). "https://www.wiki.ros.com"

GLOSARIO

CSIRO: Commonwealth Scientific and Industrial Research Organization xi

INTA: Instuto Nacional de Tecnología Aeroespacial xi

US: Universidad de Sevilla xi

$LiFePO_4$: Batería de litio-ferrofosfato xi

S.O.: Sistema Operativo xv

EMS: Sistema de gestión de energía de INTA para Summit XL xv

ROS: Robot Operative System 3

SLAM: Simultaneous Localization and Mapping -

IUFCV: Improving efficiency and operational range in low-power unmanned vehicles through the use of hybrid fuel-cell power systems 1

AUV: Autonomous Underwater Vehicle 1

AGV: Autonomous Ground Vehicle 1

IoT: Internet of Things. Concepto relativo a la conexión de pequeños sistemas de forma inalámbrica formando un gran sistema con múltiples nodos 5

GitHub: Repositorio para la comunidad de desarrolladores de software de ROS 6

Rviz: Herramienta de visualización de ROS 6

Gazebo: Herramienta de simulación de ROS 6

IDE: Entorno de Desarrollo Integrado 11

PWM: Modulación por ancho de pulsos 12

skid-steering: Configuración de un vehículo con un motor por rueda que permite hacer giros sobre sí mismo semejantes a las de configuración de oruga 20

ADC: Conversor analógico digital 21

TCP: Transmission Control Protocol 28

EMS: Sistema de Gestión de energía 38