Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA EN TECNOLOGÍA INDUSTRIAL

TRABAJO FIN DE GRADO

CONTROL DE LA NAVEGACIÓN DE UNA PLATAFORMA MÓVIL KOBUKI CON UN BRAZO MANIPULADOR WIDOWX

Alumno: Alonso, Tejeda, Asier

Director: Casquero, Oyarzabal, Oskar

Codirector: Orive, Revillas, Darío

Curso: 2018-2019

Fecha: jueves, 27, 06, 2019

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

Resumen:

En este trabajo se va a desarrollar el control de una base móvil Kobuki que monta encima un brazo robótico WidowX para introducirlo en un sistema de fabricación flexible en el que pueda ofrecer operaciones de transporte y manipulación. Como framework robótico que maneja el conjunto, se utiliza ROS (Robot Operating System) y se realiza un análisis de su funcionamiento e implementación en los equipos. Para efectuar el control se deben conocer las características de los equipos y por ello, se describen sus especificidades y las posibilidades que ofrecen. Además, por tratarse de una plataforma robótica móvil, las comunicaciones entre los distintos equipos que participan deben ser inalámbricas, así que se diseñan e implementan módulos de transmisión inalámbrica de datos.

Palabras clave: ROS, Kobuki, WidowX, fabricación flexible.

Laburpena:

Lan honetan, gainean WidowX beso robotiko bat daraman Kobuki base mugikorraren kontrola garatuko da. Honen helburua, fabrikazio malgu sistema batean robot hauek sartzea da, garraio eta manipulazio ariketak egin ditzaten. Framework robotiko gisa, ROS (Robot Operating System) erabiliko da eta haren inplementazioa eta lan egiteko erak aztertuko dira. Kontrola garatzeko ekipamenduaren ezaugarriak ezagutzea beharrezkoa da, beraz, haren ezaugarriak eta ematen dituzten aukerak deskribatzen dira. Horrez gain, robot mugikor bat denez, elementu guztien arteko komunikazioak haririk gabekoak izatea ezin hobea izango litzake. Hortaz, haririk gabeko datuen transmisioa izateko zenbait modulu sartu eta garatu egiten dira.

Gako hitzak: ROS, Kobuki, WidowX, fabrikazio malgua.

Abstract:

In this work, the control of a Kobuki mobile base which carries a WidowX robotic arm on top of its structure will be developed with the aim of introducing it in a flexible manufacturing system in order to perform manipulation and delivery actions. This project will use ROS (Robot Operating System) and its implementation as well as its working structure will be analyzed. A description of the equipments and the possibilities they offer is done so that the control is performed with as much resources as possible. In addition, the communications between different elements of the work environment must be wireless since this project's mobile base independence has to be optimal. As a consequence, some wireless data transmission tools are developed in the project.

Key words: ROS, Kobuki, WidowX, flexible manufacturing.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 Introducción

El desarrollo industrial llevado a cabo durante las últimas décadas es más que evidente en todos los sectores. Esta continua evolución nos sitúa en la actualidad, en la época de la denominada Industria 4.0 o Cuarta revolución industrial. Debido a estos progresivos avances, las líneas de producción, fabricación, organización y otros muchos ámbitos están tendiendo a grados de automatización elevados, mediante los cuales se pretende optimizar la precisión de las operaciones industriales y sus tiempos de producción.

Estas circunstancias han provocado una masiva introducción de robots en la industria, lo que acarrea el obligado desarrollo de todos los ámbitos relacionados con la inteligencia artificial y la robótica, abriéndose así múltiples líneas de investigación. La implementación de células robóticas automatizadas provoca la necesidad de personal especializado en el sector de la línea de trabajo para supervisar el funcionamiento y la correcta realización de las tareas robóticas. Es por ello que surge la necesidad de formar profesionales capaces de programar y controlar estos robots y de dotar a los trabajadores de conocimientos suficientes en estos ámbitos, para que puedan ejercer sus nuevas labores sin verse apartados.

Uno de los principales inconvenientes identificados a la hora de realizar la programación de algoritmos de control y seguimiento de las máquinas, es la falta de estandarización en estos métodos. Por norma general, cada fabricante proporciona para sus robots sus propios algoritmos privativos con los que busca una situación de dominancia en el sector, lo cual dificulta el aprendizaje. Debido a esto, gracias a investigaciones desarrolladas principalmente en los últimos años, han empezado a aparecer nuevos meta-sistemas operativos y agrupaciones de librerías que facilitan el control, siguiendo una estructura de trabajo y comunicación común e independiente del fabricante.

2 Contexto

Una de las líneas de investigación y desarrollo en la que más fuerza cobran los aspectos presentados anteriormente, es la correspondiente a la fabricación flexible. Un sistema de fabricación flexible consta de unas estaciones de trabajo interconectadas entre sí a través de un sistema de transporte automatizado de piezas o materiales. Actualmente, este sistema de transporte se trata en la mayoría de los casos de una cinta transportadora. Los principales objetivos de la fabricación flexible son optimizar la fabricación respecto a otros procesos como máquinas transfer, aportar versatilidad a las líneas de producción, optimizar la gestión de la producción y los recursos y poder adaptarse a las exigencias impuestas por los clientes.

En el ámbito de ingeniería, siempre se tiende a buscar el mayor grado de eficiencia y la optimización de los procesos. Es por ello, que alrededor del ámbito de la fabricación flexible existen muchas ideas, con vistas a un futuro cercano, que pretenden que su manera de funcionar sea la óptima. Una de ellas, en concreto la que se está llevando a cabo desde el Grupo de Control e Integración de Sistemas de la Escuela de Ingeniería de Bilbao, propone la sustitución de la cinta transportadora por robots móviles de transporte. Estos robots, equipados con distintas herramientas como cámaras o manipuladores, realizarían el transporte de materiales y piezas entre estaciones. Para esto, dichos robots deberán poder comunicarse entre ellos y decidir en cada momento quien será el encargado de realizar cada acción.

La principal restricción de un sistema de fabricación flexible la impone la cinta transportadora, ya que es condición necesaria colocar las estaciones de trabajo a su alrededor y en un orden determinado. Su eliminación supone dar una mayor libertad a la disposición de las estaciones dentro de una célula de fabricación y, por lo tanto, amplía las posibilidades de adaptarse a las exigencias de los clientes, viéndose notablemente aumentada la capacidad de producción de lotes diferentes entre sí. De esta forma, el orden en el que se realizan las operaciones no es fijo y los tiempos de producción se ven optimizados.

Esta eliminación de la cinta transportadora no sería posible sin la introducción de robots móviles. Dichos robots pueden estar equipados con cualquier tipo de herramienta y pueden moverse libremente, lo que aporta muchas posibilidades a la hora de adaptarse a

diferentes tipos de procesos de producción y buscar siempre el tiempo óptimo de trabajo. Para poder llevar a cabo esta modificación, es necesario disponer de los recursos necesarios para alcanzar tales grados de automatización, ya que la comunicación deberá ser en todo momento inalámbrica. Además, para poder realizar el trabajo de forma eficiente, las unidades robóticas deberán negociar entre ellas y colaborar con otras entidades dentro del proceso de fabricación.

3 Alcance

Este TFG forma parte de otro proyecto, realizado con anterioridad [9], que tenía como objetivo la creación de servicios robóticos de alto nivel para atender las peticiones de abastecimiento de materia prima y transporte de material entre máquinas en el marco de un sistema de fabricación flexible coordinado mediante un sistema multi-agente. Concretamente, a través de este trabajo se pretende diseñar y realizar el desarrollo de nodos que controlen una base móvil Kobuki que lleva acoplada un brazo robótico WidowX con un sensor térmico en la pinza.

4 Objetivos

El objetivo principal del trabajo es el control de una plataforma móvil Kobuki y la implementación de una comunicación inalámbrica para la gestión de los datos de un sensor de la pinza del brazo en entorno ROS.

Para completar este objetivo principal será necesario cumplir con los siguientes objetivos secundarios:

- Análisis de funcionamiento de la plataforma móvil Kobuki.
- Conocimiento y documentación de Linux y de diseño de alto nivel de ROS.
- Definición de una serie de casos de uso para el diseño de trayectorias.
- Programación de trayectorias simples e independientes del brazo robótico en ROS.
- Programación de trayectorias dependientes del brazo robótico y movimiento conjunto del sistema base+brazo.
- Análisis de funcionamiento de los módulos de comunicación inalámbrica.
- Implementación física y de software de módulos de comunicación inalámbrica.
- Incorporación de las comunicaciones inalámbricas en ROS.

5 Beneficios que aporta el trabajo

La introducción de robots móviles con manipuladores robóticos montados sobre los mismos en procesos de fabricación flexible mejora el proceso en varios aspectos y ofrece unos beneficios muy diversos.

Desde un punto de vista puramente económico, la eliminación de la cinta transportadora en torno a la cual se organizan las células de trabajo supone dotar al proceso productivo de una mayor flexibilidad. Esto se traduce en multiplicar las posibilidades a la hora de diseñar el proceso de producción que van a seguir los productos solicitados. Una consecuencia directa de aumentar la flexibilidad es que también aumenta la capacidad de adaptarse a demandas variables y poder acoger bajo una misma cadena de producción, la demanda de diversos tipos de productos. De esta forma, puede llegar a asumirse la producción de tiradas cortas de un mismo producto, ofreciendo una cantidad de productos a la venta superior, traduciéndose esto en ingresos o beneficios monetarios. Pero esta modificación del proceso, no se limita sólo a las mejoras económicas en el ámbito de ventas. Cuando en el proceso de fabricación ocurre un fallo en la cinta de transporte, la producción se para hasta que pueda arreglarse el error y es entonces, cuando se retoma el trabajo. Es evidente que un paro en la producción acarrea grandes pérdidas económicas, por lo que habitualmente se realizan diversos tipos de mantenimiento para evitarlos. Sin embargo, el funcionamiento del sistema de fabricación con plataformas móviles sería diferente. Al eliminar la cinta, el trasvase de piezas de una célula de trabajo a otra lo realizarían los robots, que mantienen una comunicación constante entre ellos. En caso de avería de un robot, otro robot que no estuviera trabajando asumiría la función del averiado. De esta manera, se optimizan notablemente los tiempos de producción ya que en caso de fallo, la solución es relativamente rápida. Cuanto menor sea el tiempo de respuesta de las máquinas, menor es el tiempo de parada y por lo tanto, menores también las pérdidas en caso de avería.

A nivel técnico, el mantenimiento de los robots móviles no es complicado y no detiene el proceso en ningún momento, ya que la función que cumple un robot en un momento dado, la puede cumplir otro que cumpla los requisitos hardware solicitados. Además, con una correcta sensorización, se garantiza una precisión alta.

Teniendo en cuenta otra serie de factores, cabe destacar que al dotar al proceso de una gran flexibilidad, las células de trabajo pueden ser redistribuidas de tal forma que se optimice el proceso y se reduzcan los costes energéticos asociados al transporte entre estaciones. Por lo tanto, habrá ocasiones en las que la utilización de robots móviles mejore la eficiencia energética del proceso.

6 Descripción de requerimientos

En este trabajo se han impuesto ciertas especificaciones y requerimientos de partida. El robot a controlar es una base móvil Kobuki de la empresa Yujin Robots. Dicho robot está orientado a la investigación y formación en robótica. El brazo robótico que debe llevar montado sobre el mismo, es un WidowX distribuido por Trossen Robotics. Como el proyecto del que forma parte este trabajo [9] se desarrolla en un entorno de ROS, es condición indispensable que la programación del control de la base se realice utilizando este framework para robots. Además, se deberá implementar un módulo de comunicación inalámbrico para poder comunicar un sensor en el brazo con el procesador del Kobuki.

A nivel de control, como se van a integrar distintos elementos hardware, estos deben poder interactuar entre ellos, es decir, se debe aprovechar el entorno de ROS para establecer los intercambios de información entre brazo, base y sensor. Como se quiere implementar el conjunto como parte de un sistema multi-agente introducido en un ámbito de fabricación flexible, será necesario separar las secuencias de movimiento en módulos, para poder invocar dichos módulos como servicios en función de las necesidades que presente el proceso.

A continuación, se describen los requerimientos funcionales que se derivan del proceso que debe seguir el conjunto del manipulador y base. En un primer momento, la base y el manipulador se deben encontrar en situación de reposo. Al ejecutarse el inicio del programa, el brazo robótico se incorporará y se situará directamente en frente de la pieza a evaluar, teniendo en cuenta las distancias hasta la mesa en la que se encuentra. Una vez aquí, el sensor térmico mide tanto la temperatura del objeto que tiene delante como la temperatura ambiente. Si la temperatura del objeto es inferior a la temperatura ambiente, el brazo debe volver a su posición de reposo y terminar la ejecución; sin embargo, si la temperatura del objeto es superior a la del ambiente, se iniciará la siguiente secuencia. La pinza se cierra y el manipulador tiene que volver a su situación de reposo. A partir de aquí entra en juego la base móvil, ya que se requiere transportar la pieza de una mesa del laboratorio a la que está inmediatamente en frente de ella. Para ello, la base debe realizar un giro de 180 grados sobre el eje vertical y al terminar el giro comenzar un movimiento frontal que terminará al llegar a la mesa en la que depositará la pieza. Una vez allí, el brazo robótico se tiene que incorporar hasta depositar la pieza en la mesa y después volver a su estado de reposo. Para finalizar, la base volverá a girar otros 180 grados respecto el eje

vertical y avanzará hasta llegar al punto de partida, quedando todo en las circunstancias iniciales para poder volver a ejecutar el programa y realizar una nueva evaluación. En la figura 1 se ha representado esquemáticamente la parte correspondiente al funcionamiento del robot móvil Kobuki, destacando en rojo las condiciones que se imponen por parte del manipulador robótico, cuyo control es ajeno a este trabajo.

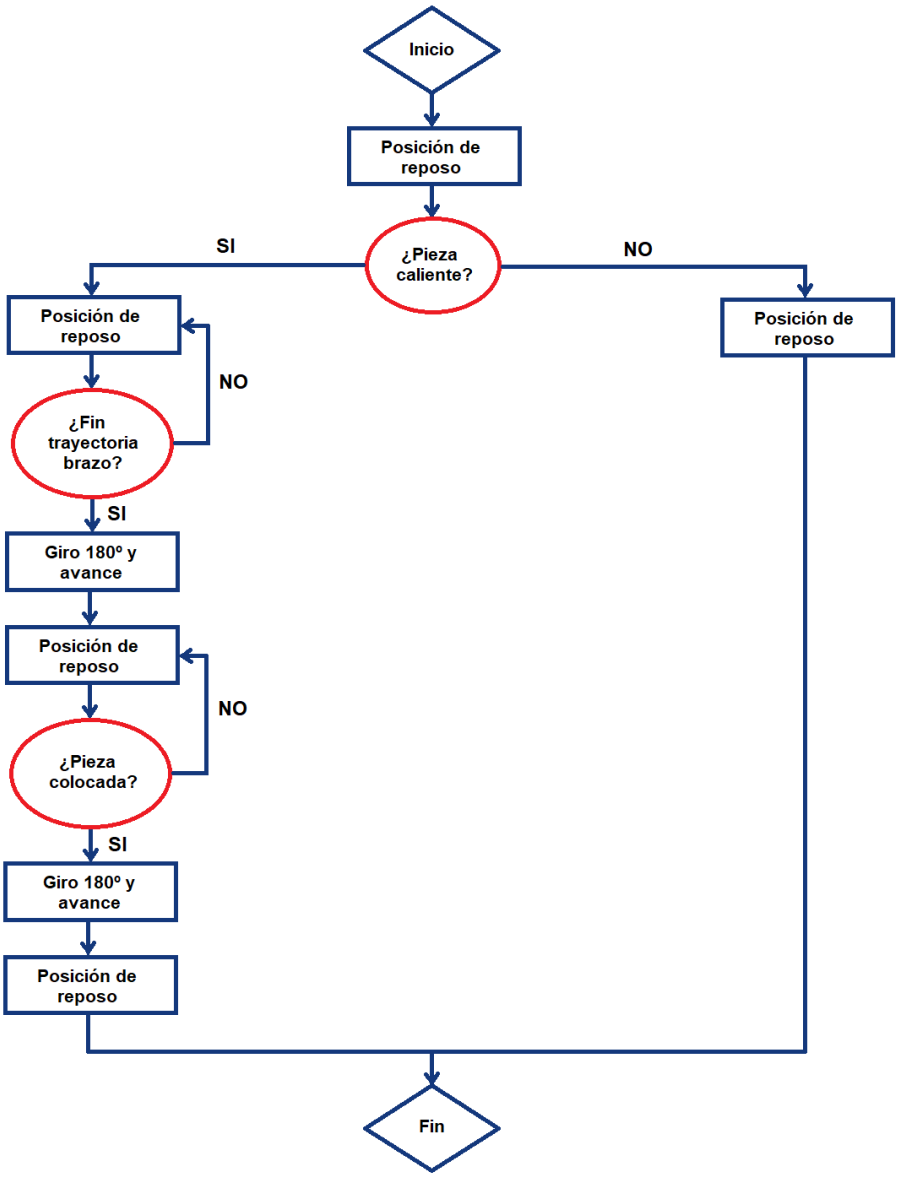


Figura 1 Esquema de funcionamiento de las trayectorias del robot Kobuki del que se derivan los requisitos funcionales.

7 Análisis de alternativas

A la hora de realizar el desarrollo del trabajo planteado, se han tenido que tomar diversas decisiones. A continuación se expondrán en cada uno de los casos las alternativas planteadas y el análisis de su viabilidad mediante diferentes ponderaciones, con el cual se fundamenta la decisión tomada. Para cada alternativa se tendrán en cuenta unos criterios diferentes. Cada criterio tendrá asignado un coeficiente de ponderación debido a la importancia relativa que presenten entre ellos y la nota de cada uno estará comprendida entre 1 y 10 puntos.

7.1 Versión de ROS

Para realizar el estudio en base al cual se tomará la decisión de la versión con la que trabajar se van a tener en cuenta los siguientes criterios:

- **Soporte**: Se trata de la participación de los desarrolladores de ROS en la comunidad para proporcionar ayuda y resolver problemas que se encuentren los usuarios. Se le asigna un coeficiente de ponderación de 0.25.

- **Disponibilidad de paquetes**: Para que se pueda trabajar con un robot es necesario disponer de paquetes que contengan información sobre su modelo, controlador… Escribir estos paquetes desde cero es complicado y por ello, los fabricantes o distribuidores proporcionan estos paquetes ya creados, para que a partir de ellos se pueda programar el control de un equipo. Se le asigna un coeficiente de ponderación de 0.4.

- **Lenguajes de programación**: Los scripts que se van a crear en ROS pueden estar escritos en diferentes lenguajes de programación, en función de las librerías disponibles en cada versión. Se le asigna un coeficiente de ponderación de 0.1.

- **Herramientas y recursos**: ROS dispone de ciertas herramientas que pueden ser de gran utilidad en algunas aplicaciones, por ejemplo, interfaces gráficas. En cada versión el funcionamiento de las mismas y su puesta a punto puede ser diferente. Por lo tanto, atendiendo a la dificultad de puesta en marcha y robustez se le asigna un coeficiente de ponderación de 0.25.

Robot Operating System (ROS) es un framework robótico. Cada versión de ROS funciona sobre una distribución Ubuntu de Linux diferente. ROS publica una nueva versión con una periodicidad anual. Las versiones lanzadas en años pares tienen soporte para 5 años (versiones denominadas tradicionalmente LTS, Long Term Support), mientras que las lanzadas en años impares solo tienen soporte durante dos años. Las versiones que se van a llevar a estudio son las siguientes:

- **ROS Melodic** (figura 2): Se trata de la versión más reciente (lanzada el 23 de mayo de 2018) y es de tipo LTS. Para poder hacer funcionar esta distribución, es necesaria la versión Ubuntu 18.04. Al tratarse de una versión relativamente reciente, es complicado encontrar paquetes actualizados ofrecidos por los fabricantes de robots que sirvan como base de la programación del control de estos. Contiene librerías para programar en Python, C++ y Java entre otros. Algunos paquetes de herramientas que ofrece el entorno ROS no están correctamente adaptadas a la nueva versión y su puesta a punto puede ser costosa. Por lo tanto, se le asignan las siguientes puntuaciones:
    - Soporte: 10
    - Disponibilidad de paquetes: 5
    - Lenguajes de programación: 10
    - Herramientas y recursos: 7.5

Figura 2 Logo ROS Melodic.

- **ROS Lunar** (figura 3): Su lanzamiento data del 23 de mayo de 2017. El soporte de esta distribución termina en mayo de 2019 y para funcionar necesita Ubuntu 17.04. Se trata de una distribución reciente y que además tiene un soporte corto, por lo que la cantidad de trabajo desarrollado con ella es considerablemente baja, de tal forma que puede ser complicado encontrar paquetes compatibles con la versión. Contiene librerías para programar en Python, C++ y Java entre otros. Debido a la poca profundización en el desarrollo de herramientas con esta versión, pueden presentarse inconvenientes a la hora de utilizar algunas de ellas, por ejemplo, falta de estabilidad. Por lo tanto, se le asignan las siguientes puntuaciones:

- Soporte: 8
- Disponibilidad de paquetes: 6.5
- Lenguajes de programación: 10
- Herramientas y recursos: 7

Figura 3 Logo ROS Lunar.

- **ROS Kinetic** (figura 4): Su lanzamiento data del 23 de mayo de 2016. El soporte de esta distribución termina en abril de 2021 y para funcionar necesita Ubuntu 16.04. Es posiblemente una de las versiones que más se han utilizado en investigaciones puramente relacionadas con control de robots. El hecho de que lleve tres años activa como distribución hace que la gran mayoría de paquetes y herramientas estén bien revisadas y que se puedan encontrar fácilmente alternativas para solucionar posibles problemas. Al igual que en los casos anteriores, contiene librerías para programar en Python, C++ y Java entre otros. Por lo tanto, se le asignan las siguientes puntuaciones:
  - Soporte: 5
  - Disponibilidad de paquetes: 9.5
  - Lenguajes de programación: 10
  - Herramientas y recursos: 9

Figura 4 Logo ROS Kinetic.

Tabla 1 Análisis de alternativas: ROS.

	Coeficiente de ponderación	ROS Melodic	ROS Lunar	ROS Kinetic
Soporte	0,25	10	8	5
Disponibilidad de paquetes	0,4	5	6,5	9,5
Lenguajes de programación	0,1	10	10	10
Herramientas y recursos	0,25	7,5	7	9
	Resultado final	7,375	7,35	8,3

Por lo tanto, la versión de ROS con la que se va a realizar este trabajo atendiendo a los resultados obtenidos en el estudio anterior es ROS Kinetic.

7.2 Lenguaje de programación

Para realizar el estudio que determine la elección del lenguaje de programación con el que se trabajará en el entorno ROS se van a tener en cuenta los siguientes criterios:

- **Robustez**: Los lenguajes de programación son robustos si presentan una buena capacidad para hacer frente a un error mientras el programa se está ejecutando. Por lo tanto, cuanto más robusto sea el lenguaje, más estable y solido será el resultado obtenido al ejecutar lo programado. Se le asigna un coeficiente de ponderación de 0.2.

- **Sencillez para prototipado**: Los lenguajes que presentan sencillez para realizar prototipos suelen generalmente presentar tiempos de respuesta rápida a costa de perder algo de estabilidad. Es por ello que se utilizan mayoritariamente en entornos de investigación donde se busca hacer pruebas de concepto para comprobar si una idea funciona y, a posteriori, desarrollarla. Se le asigna un coeficiente de ponderación de 0.3.

- **Manejabilidad en ROS**: La versión de ROS seleccionada ofrece librerías para programar en cualquiera de los dos lenguajes, pero en este apartado se pretende evaluar la dificultad de utilizarlos en este entorno. Se le asigna un coeficiente de ponderación de 0.5.

Los lenguajes de programación que se valoran como alternativas para la realización del trabajo son C++ y Python.

- **C++**: Se trata de un lenguaje de programación muy estable y potente en cuanto a su capacidad. Es por ello, que su utilización está bastante extendida a nivel industrial. Se necesita cierto nivel y dominio de programación, puesto que su uso y organización pueden resultar complejos. Por lo tanto, es un lenguaje robusto, pensado para proyectos en producción de gran magnitud. Las puntuaciones que se le asignan son las siguientes:
    - Robustez: 10
    - Sencillez para prototipado: 6
    - Manejabilidad en ROS: 7

- **Python**: Se trata de un lenguaje de programación de utilización sencilla. Cuenta con gran cantidad de librerías que facilitan su manejo y el desarrollo de los scripts. Su uso y organización también son sencillos y el aprendizaje es rápido. Es un lenguaje de programación idóneo para llevar a cabo trabajos de prototipado. Por lo tanto, se le asignan las siguientes puntuaciones:
    - Robustez: 5
    - Sencillez para prototipado: 10
    - Manejabilidad en ROS: 9

Tabla 2 Análisis de alternativas: Lenguaje de programación.

	Coeficiente de ponderación	C++	Python
Robustez	0,2	10	5
Sencillez para prototipado	0,3	6	10
Manejabilidad en ROS	0,5	7	9
	Resultado final	7,3	8,5

Por lo tanto, el trabajo se va a llevar a cabo usando Python como lenguaje de programación.

8 Descripción de la solución

El sistema como conjunto está formado por una base móvil Kobuki que lleva montado encima un manipulador robótico de 5 grados de libertad de modelo WidowX MK II. Además, se ha colocado un sensor térmico MLX90614 en la pinza del brazo. Finalmente, para eliminar un posible exceso de cables, se utilizan unos módulos de comunicación inalámbrica XBee que envían los datos recogidos por el sensor al procesador.

Cabe destacar que todo el trabajo gira en torno al ordenador de placa reducida (SCB, Simple Computer Board) Odroid. Es alrededor de éste donde se organizan los demás elementos del conjunto. Esta placa Odroid puede comunicarse de manera inalámbrica vía WiFi a un ordenador (figura 5). En dicho ordenador se realiza el desarrollo de programas que se implementan en los equipos y se organiza la información que contiene la placa Odroid.

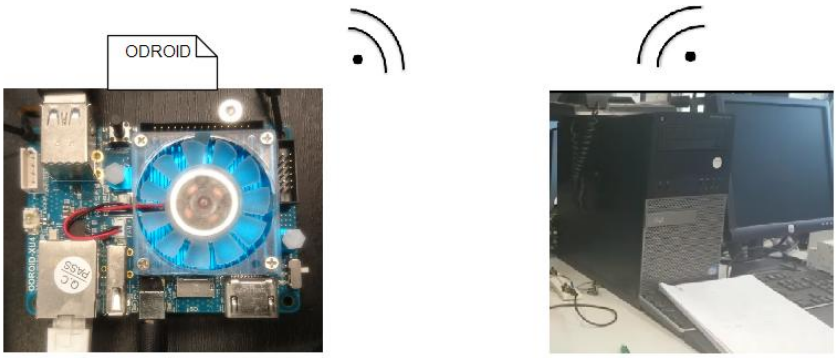


Figura 5 Conexión entre Odroid y ordenador.

La base móvil Kobuki lleva un microcontrolador incorporado (encargado de la gestión de bajo nivel de las E/S), que no puede funcionar sin conectarse a un microprocesador, por lo que se conectan la placa Odroid y la plataforma Kobuki vía USB. Algo similar ocurre con el manipulador robótico. Su controlador es una placa Arbotix, la cual se gestiona gracias al programa Arduino, que se conecta vía USB-FTDI al procesador Odroid (figura 6).

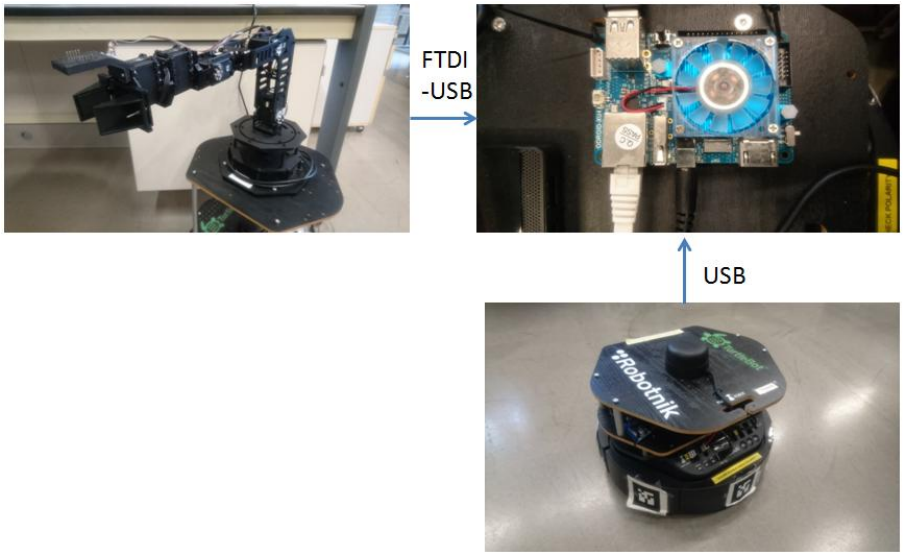


Figura 6 Conexión entre Odroid, Kobuki y WidowX.

El sensor térmico necesita de un microcontrolador Arduino para realizar la lectura de muestras. Además, también necesita que los diferentes circuitos integrados estén correctamente conectados, por lo que está unido a una pequeña PCB (Printed Circuit Board, placa de circuito impreso) que contiene el Arduino. Los módulos de comunicación XBee se organizan por pares. Uno de ellos, envía la información y el otro la recibe. Además, para realizar su función correctamente también es necesario que un Arduino se encargue de gestionar el intercambio de información. Por lo tanto, el emisor se conectará a la PCB que contiene el Arduino que gestiona el sensor y el receptor, a su vez, se conectará a la placa Odroid con un adaptador a puerto USB (figura 7).

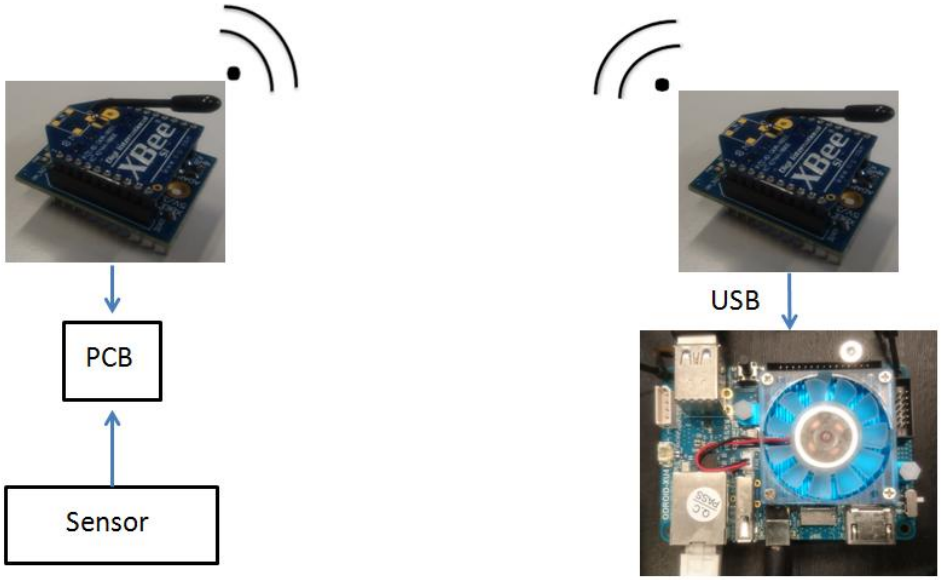


Figura 7 Conexión entre Odroid, sensor y XBees.

En cuanto al software, tanto en el ordenador como en el Odroid están instalados Ubuntu y ROS. Desde el ordenador se carga el programa correspondiente y el procesador Odroid se comunica con quien corresponda. En la plataforma Kobuki se inician los drivers para comunicarse con ROS y en la placa Arbotix está cargado un programa de Arduino que traduce las órdenes de ROS a órdenes que pueda interpretar el brazo robótico. Además, ya se ha comentado que los XBees necesitan de un Arduino para funcionar.

Teniendo en cuenta las conexiones hardware y las necesidades de software, una representación del conjunto para entender el funcionamiento del sistema es la que se muestra en la figura 8.

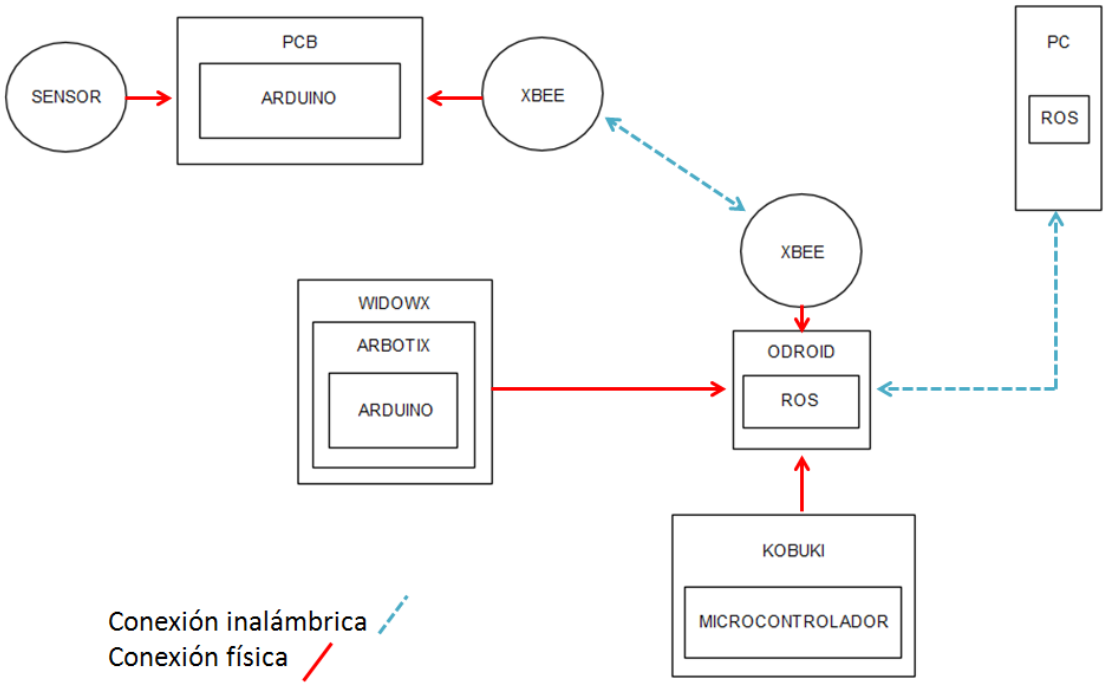


Figura 8 Esquema de comunicación del sistema total.

9 Diseño

En primer lugar, se va a proceder a explicar de forma básica qué es ROS y cuál es su forma de funcionar, ya que todo el proyecto girará en torno a este framework robótico. En segundo lugar, se explicará la parte de hardware de la cual está compuesto el equipo y por último, se analizará la parte de software, tanto a alto como a bajo nivel.

9.1 ROS

9.1.1 Conceptos básicos

ROS es un meta sistema operativo que contiene recursos como librerías y herramientas para la creación de programas para control de robots. Sus objetivos son la estandarización de código y la reutilización del mismo para proyectos de investigación y desarrollo en el ámbito de la robótica. Para ello, se emplean una estructura y forma de trabajar muy particular que es necesario describir para comprender el diseño propuesto en este trabajo.

Para que ROS pueda funcionar, es necesario tener un sistema operativo libre proporcionado por Linux como Ubuntu en cualquiera de sus versiones compatibles con cada versión de ROS. De esta forma, sobre el ordenador se instala una distribución de Ubuntu y sobre ella, ROS. ROS se considera un meta-sistema operativo ya que ofrece una abstracción de hardware similar a la de un sistema operativo al uso, y se instala sobre un sistema operativo. Una representación sencilla de estos conceptos sería la siguiente (figura 9):

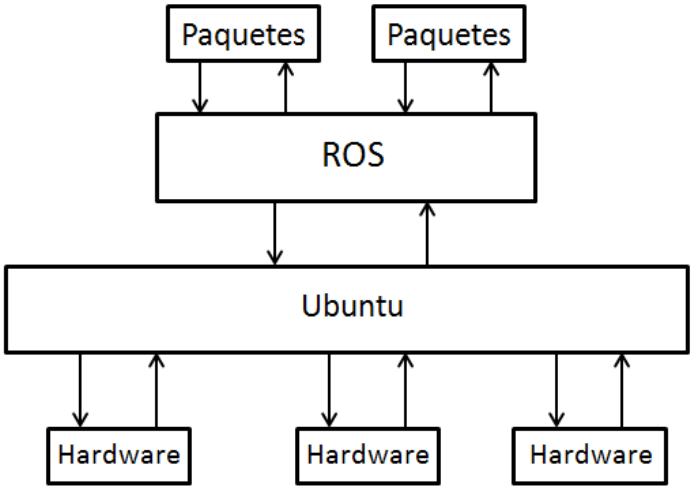


Figura 9 Relación entre ROS y hardware.

A nivel de organización, es muy importante el uso de paquetes. Son el elemento más básico y contienen información relevante como pueden ser librerías, nodos y otros conjuntos de datos que se explicarán a continuación. Estos paquetes pueden almacenarse en repositorios teniendo en cuenta la compatibilidad entre versiones, de tal forma que puedan quedar varios paquetes para una misma aplicación guardados conjuntamente.

Dentro de los paquetes creados, pueden encontrarse tipos de servicios y tipos de mensajes. Los tipos de servicios definen una estructura de datos de petición y respuesta para la implementación de servicios. Por otro lado, los tipos de mensajes definen las estructuras de datos de mensajes que se mandan en ROS. La importancia de las estructuras de datos es muy grande, puesto que habrá que consultarlas cada vez que se quiera utilizar un servicio o mandar un mensaje.

En cuanto a aspectos relacionados con la comunicación y manejo de datos, el concepto fundamental es el *nodo*. Un nodo es el proceso que ejecuta la computación. A nivel práctico, se trata de un archivo con líneas de código que define una función concreta de un robot, por ejemplo, una secuencia de movimientos. Es por ello, que para realizar un control completo de cierto robot será necesaria la implementación y correcta comunicación de diferentes nodos. ROS proporciona varias librerías como Roscpp y Rospy, para utilizar C++ y Python, respectivamente, como lenguaje de programación de nodos.

La comunicación entre nodos se realiza a través de mensajes. Son tipos de estructuras de datos que pueden contener booleans, enteros, reales... y otros datos simples. Además de esto, pueden contener arrays de tipos de datos como los anteriores o estructuras más complejas compuestas, a su vez, de otras estructuras complejas y datos simples. Lo habitual es que los mensajes manejen estructuras complejas, ya que el intercambio de información de este modo es superior y ordenado. Un ejemplo ilustrativo y bastante habitual en todos los robots es el mensaje que devuelve el estado de los servomotores. Este mensaje se define como *sensor_msgs/JointState.msg* y dentro de sí incluye estos campos:

- *std_msgs/Header header*
- *string[] name*
- *float64[] position*
- *float64[] velocity*

- *float64[] effort*

Como se observa, los mensajes tienen una estructura definida y hay que tratar los datos tal cual vengan definidos, para evitar errores. En este ejemplo concreto, el mensaje devolvería su cabecera, seguido de una string que defina el nombre del servomotor del que va a proveer la información. Los siguientes tres parámetros serían la posición, velocidad y esfuerzo como Float64, es decir, como un número real con espacio reservado de 64 bits. Estos 3 parámetros tendrán que ser interpretados en función de los rangos que maneje el servomotor estudiado y las unidades de trabajo.

La comunicación entre nodos no se realiza mediante el envío directo de mensajes entre nodos, sino que se realiza una comunicación indirecta a través de unas entidades de comunicación llamadas *tópicos*. Esto, que a priori puede parecer una desventaja, aporta una versatilidad muy grande. Los nodos publican sus mensajes en los tópicos y también tienen la posibilidad de suscribirse a ellos para recibir información de mensajes publicados por otros nodos. De esta manera, aparecen los conceptos de *publisher* y *subscriber*. Un *publisher* no es más que un nodo que publica mensajes en un tópico, mientras que un *subscriber*, es aquel nodo que accede al tópico para leer dichos mensajes. Por lo tanto, para establecer una comunicación robusta, la cual es una de las principales características de este framework robótico, un nodo tiende a ser *publisher* y *subscriber* al mismo tiempo. La idea principal es que diferentes nodos que publican y se suscriben no sepan nada el uno del otro y simplemente accedan a la información que necesiten, en los correspondientes tópicos.

Para una correcta comunicación y coordinación, es imprescindible la existencia de un *maestro*. El maestro tiene por función permitir y supervisar la comunicación entre nodos. Este maestro mantiene un registro constante de todos los tópicos, servicios y mensajes activos y permite que se encuentren unos con otros. Por lo tanto, las ejecuciones no podrían realizarse sin su existencia y por ello, el primer paso para realizar cualquier función mediante ROS es tenerlo activo. El maestro se puede ejecutar tanto en un ordenador externo como en el propio robot a controlar.

Para facilitar la comprensión de todos estos conceptos se procede a explicar un ejemplo práctico y habitual en robots móviles (figura 10).

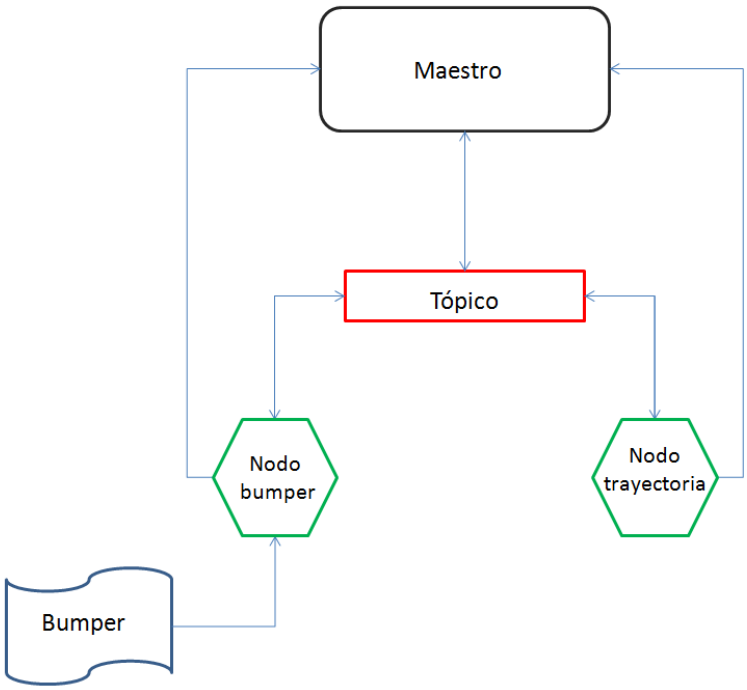


Figura 10 Esquema de comunicación básica en entorno ROS.

Un bumper es un sensor que se activa en caso de que contacte con algún obstáculo. En el nodo bumper se encontraría el programa que procesa la información de este bumper, mientras que en el nodo trayectoria, se encontraría la definición de una trayectoria dependiente de la información proporcionada por dicho sensor. Por otro lado, estaría el tópico, donde el nodo bumper publicaría el estado del sensor y el nodo trayectoria lo leería. Para que esto sea posible, tanto los nodos como el tópico deben estar comunicados con el maestro en todo momento. Este proceso comenzaría con la solicitud de registro de los nodos en el maestro cuando son ejecutados por primera vez. Una vez están registrados, cuando el nodo bumper quiera publicar su información, se lo comunica al maestro, que registra el tópico. Cuando el nodo trayectoria quiera suscribirse al tópico para leer la información publicada, se comunica con el maestro y éste permite la suscripción. De esta forma, se habilita la comunicación indirecta entre los nodos.

En última instancia, cuando nodos y tópicos están registrados y las publicaciones y subscripciones coordinadas, la comunicación se da sin la aparición del maestro como se puede observar en la siguiente figura, la cual sería un grafo simple de la comunicación en el robot previamente explicada (figura 11):

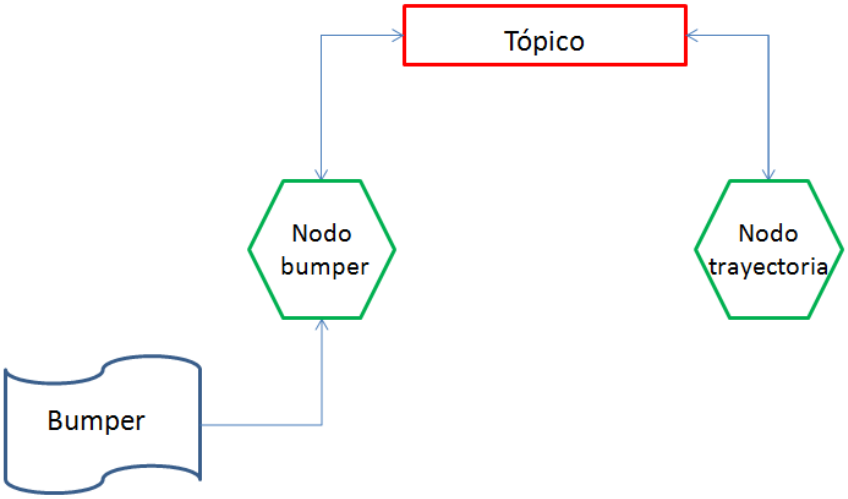


Figura 11 Esquema de comunicación en ROS en el que se ha omitido el maestro para mostrar el esquema de comunicación una vez se han registrado todos los nodos.

Si el maestro estuviera arrancado en el robot y se quisiera visualizar la información del estado del bumper en un ordenador externo, en dicho ordenador tendría que haber un nodo que contenga el programa para realizarlo. Este nodo debería comunicarse con el maestro que se encuentra en el robot, para poder así suscribirse al tópico y leer su información. El maestro también podría arrancarse en el ordenador y desde ahí supervisar nodos y tópicos encontrados en el robot y en el ordenador. Un esquema representativo del primer caso es el siguiente (figura 12):

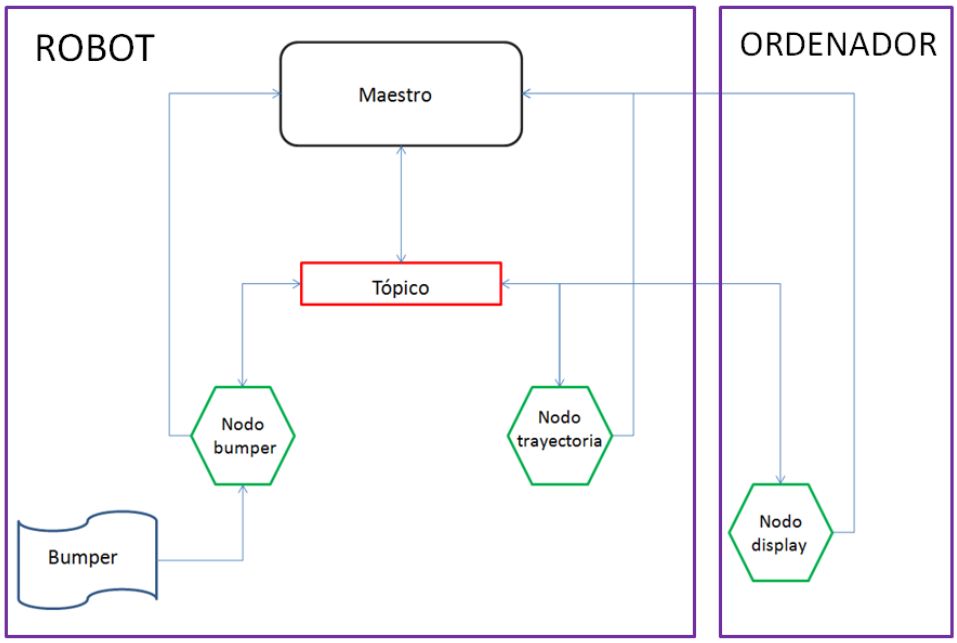


Figura 12 Esquema de relación entre nodos ubicados en diferentes equipos, pero conectados contra el mismo maestro.

El principal inconveniente que presenta la interacción de diferentes equipos con un mismo maestro es la posibilidad de que dicho maestro llegue a saturarse si hay múltiples equipos registrándose contra él. Además, también se sobrecarga la red por el tráfico derivado de la comunicación entre nodos en distintos equipos.

Hay ocasiones en las que se deben utilizar interacciones del tipo petición-respuesta. Para ello, ROS ofrece la posibilidad de utilizar servicios, que se definen mediante un par de mensajes que deben manejar los mismos tipos de datos; uno para la petición y otro para la respuesta. En este caso la comunicación es directa entre nodos, siendo un nodo el cliente y el otro el servidor (figura 13).

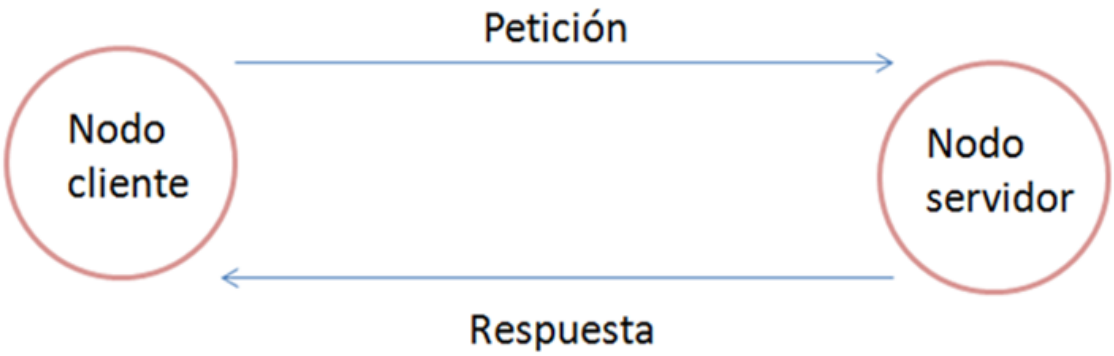


Figura 13 Esquema básico de funcionamiento de servicios en ROS.

Un aspecto diferencial de que ROS se trate de una herramienta Open Source, es la estructura de comunidad que presenta. Esta comunidad está formada por desarrolladores, investigadores, empresas y más usuarios con conocimientos de distintos ámbitos y en ella, se comparte y distribuye código. En primer lugar, cabe destacar la existencia de páginas web con foros formadas por desarrolladores que ofrecen soporte, información y ayuda. Por otro lado, el código compartido online se almacena en repositorios. Cualquier usuario puede ser partícipe pero, sobre todo, es habitual que los fabricantes de robots que ofrecen la posibilidad de usar ROS en ellos, hayan compartido código en un repositorio que defina fundamentalmente el robot y las características y funcionalidades de su controlador, herramientas… Esto facilita la puesta en marcha de los robots a los usuarios, ya que ese desarrollo de código inicial puede ser costoso si no se tiene un dominio considerable del entorno. Sin embargo, para poder usar este código correctamente e implementar desarrollo de código propio es indispensable comprenderlo y saber cómo está definido cada elemento y cómo acceder a él.

9.1.2 Comandos en ROS

A continuación se van a explicar varios de los comandos más utilizados en el entorno de ROS, que se ejecutan en la ventana de comandos de Linux, por lo que se puede decir que son la vía para acceder a ROS.

- **Roscore**: es una colección de nodos que constituyen un requisito fundamental para poder funcionar en un entorno ROS. Su ejecución es imprescindible para que los nodos en ROS se puedan comunicar, ya que entre otras cosas, gracias al comando *Roscore* se arrancan el maestro y el servidor que recoge los parámetros. Por lo tanto, se deberá ejecutar en una unidad capaz de soportar las funciones del maestro.

- **Roslaunch**: permite lanzar varios nodos al mismo tiempo tanto de forma local como inalámbrica y prefijar diferentes parámetros en su servidor. Uno de los aspectos más importantes a la hora de ejecutar el comando *Roslaunch*, es que si no se ha ejecutado un *Roscore* previamente, este ejerce la función de *Roscore* y arranca el *master* y el servidor de parámetros. Por ello, no es necesario ejecutar un comando *Roscore* si se va a utilizar *Roslaunch*.

- **Rosrun**: tiene la función de arrancar un ejecutable, generalmente un nodo, situado en cualquier ubicación del equipo. La principal ventaja es que evita tener que buscar el directorio en el que se encuentra un ejecutable para poder activarlo.

- **Rosnode info**: imprime en pantalla las principales características de un nodo, incluyendo suscripciones y publicaciones.

- **Rosnode list**: imprime en pantalla una lista de nodos que se encuentran activos en el momento de ejecución del comando.

- **Rqt**: como ROS se basa en un sistema de grafos, devuelve un grafo en el que se encuentran las comunicaciones entre nodos activos.

- **Rostopic info**: imprime en pantalla información acerca de un tópico.

- **Rostopic list**: imprime en pantalla una lista de tópicos activos en el momento de ejecución del comando.

- **Rostopic pub**: permite publicar sobre un tópico siempre que se siga la estructura del mensaje a publicar de manera correcta.

- **Rostopic echo**: imprime en pantalla los mensajes que se publican en un tópico, por lo que permite conocer los datos que maneja.

- **Catkin_create_pkg**: sirve para crear un paquete dentro del espacio de trabajo deseado, añadiendo al mismo tiempo las librerías de las cuales el paquete va a ser dependiente, por ejemplo, la librería que permitiría programar en lenguaje python.

- **Catkin_make**: se ejecuta situándose sobre el directorio que represente el espacio de trabajo (*catkin_ws*) y su funcionalidad es poder compilarlo al completo.

9.2 Hardware

Como ya se ha explicado con anterioridad, este es un trabajo que se centra solo en el control de la base y las comunicaciones, pero como forma parte de un proyecto más amplio, es necesario conocer todos los elementos que lo conforman y sus características, ya que habrá componentes con dependencias o que impongan limitaciones a los que concierne este proyecto. Los principales elementos que forman el proyecto como conjunto son los siguientes:

1. Manipulador robótico WidowX MK II.
2. Base móvil Kobuki.
3. Sensor térmico MLX90614.
4. Módulos de comunicación inalámbrica XBee.

A continuación, se desglosarán uno por uno sus características y aspectos que se deben tener en cuenta.

9.2.1 WidowX MK II:

El fabricante del robot es Trossen Robotics – InterbotiX. La estructura física se compone de piezas de aluminio y piezas de metacrilato, que son materiales ligeros pero lo suficientemente rígidos como para soportar los esfuerzos generados durante los movimientos. En esta estructura pueden diferenciarse dos partes, base y estructura. En la base se alojarán el controlador y uno de los servomotores que permitirán al manipulador rotar en torno a su eje z. Esta rotación se produce gracias a un rodamiento de bolas de 14

cm de diámetro. La base está diseñada para soportar el peso y mantener el robot en equilibrio ante cualquier configuración.

Como se trata de un manipulador robótico de 5 grados de libertad dispone de 6 servomotores fabricados por la empresa Robotis.

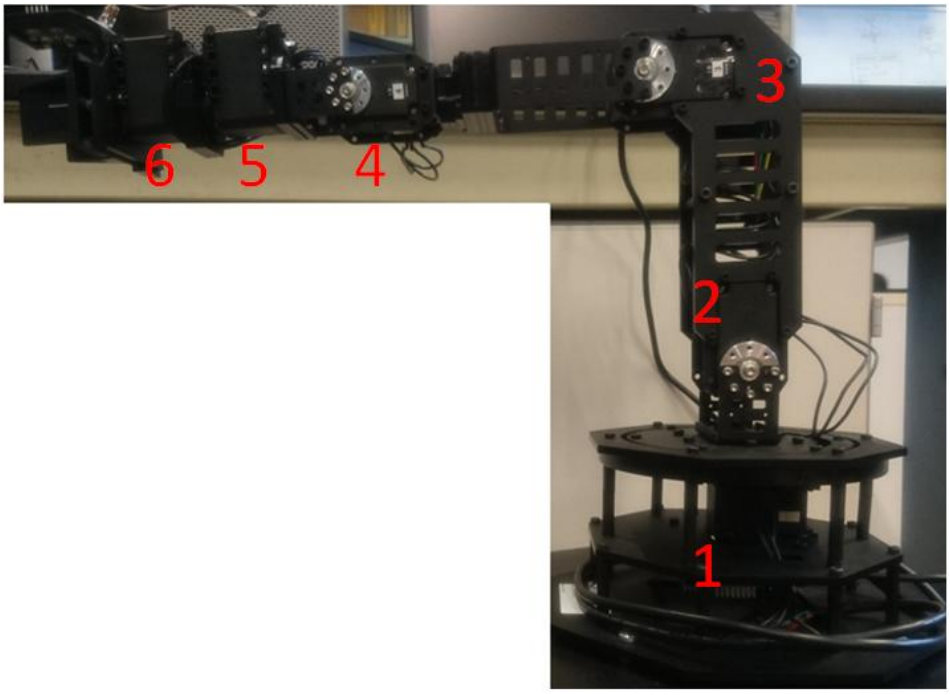


Figura 14 Servomotores sobre brazo robótico.

El segundo y el tercero son del tipo MX-64 Dynamixel, el primero y el cuarto son del tipo MX-28 Dynamixel y los dos últimos son del tipo AX-12A Dynamixel. En la siguiente tabla se recogen las principales características que presentan estos servomotores:

Tabla 3 Características de los servomotores del brazo robótico.

	Características		
Tipo	MX-64	MX-28	AX-12A
Voltaje de operación	12V	12V	12V
Velocidad máxima sin carga	63 RPM	55 RPM	59 RPM
Peso	126 g	72 g	55g
Tamaño	40.2 x 61.1 x 41.0 mm	35.6 x 50.6 x 35.5 mm	32 x 50 x 40 mm
Resolución	0.088°	0.088°	0.29°
Ángulo de operación	360°	360°	300°
Temperatura de operación	-5°C ~ 80°C	-5°C ~ 80°	-5°C ~ 70°C
Protocolo de comunicación	TTL	TTL	TTL
Material	Engranajes: metal Chasis: plástico	Engranajes: metal Chasis: plástico	Engranajes: plástico Chasis: plástico

Las principales características que posee el manipulador como conjunto son las siguientes:

Tabla 4 Características manipulador robótico al completo.

Peso del conjunto	1400 g
Alcance máximo vertical	55 cm
Alcance máximo horizontal	41 cm
Capacidad máxima de la pinza	500 g

El rango de actuación del manipulador se verá influenciado por la posición y por el peso a soportar. En la siguiente gráfica se presentan para diferentes configuraciones la relación entre peso y distancia que puede llegar a soportar:

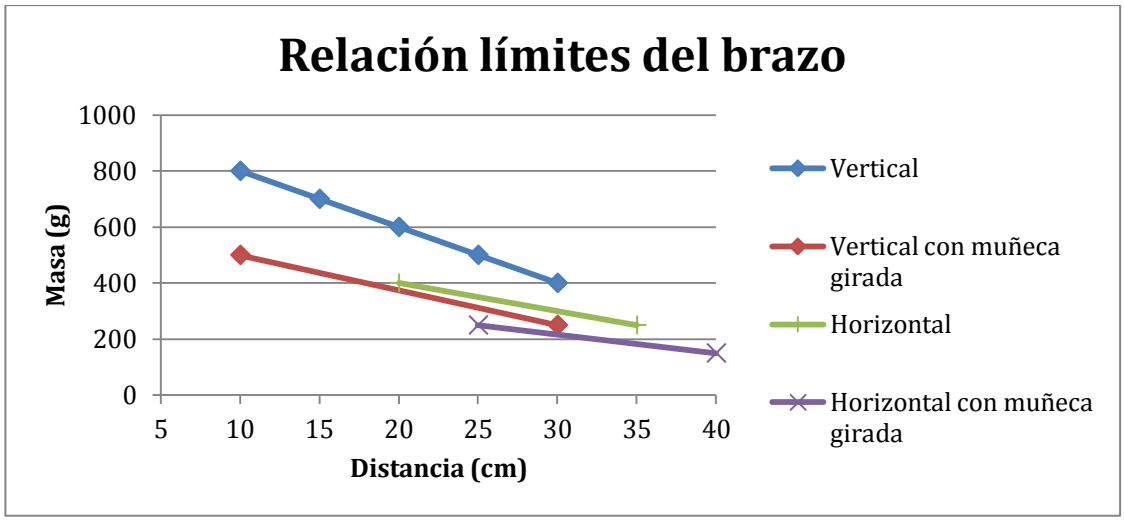


Figura 15 Gráfica de relación de los límites del brazo robótico.

El brazo robótico funciona gracias a un controlador Arbotix-M (figura 16), al cual se deben conectar todos los servomotores vía TTL serie y la alimentación del equipo. Los programas deseados se implementan en el controlador gracias a la plataforma Arduino, que ofrece múltiples posibilidades. Para poder comunicar este controlador con una unidad que contenga un procesador, es necesario emplear un cable FTDI, que en este caso opere a 5 voltios.

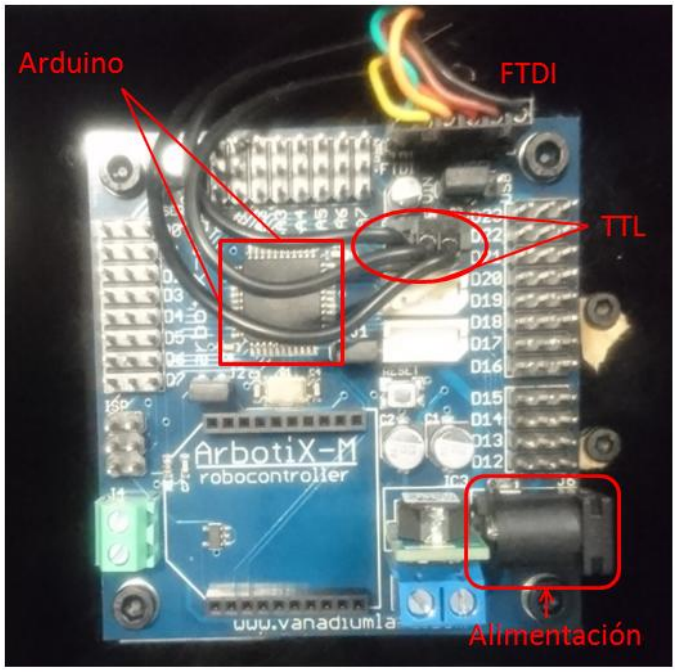


Figura 16 Placa Arbotix

9.2.2 Kobuki:

El fabricante del robot es Yujin Robot. Hay que destacar que el nombre Kobuki hace referencia únicamente a la base, mientras que la extensión Turtlebot 2, referencia a la estructura que lleva incorporada encima. Dicha estructura es necesaria para colocar sobre la base el brazo previamente descrito. Para ello, simplemente hubo que taladrar una de las placas que incorpora la estructura y atornillar sobre ella la base del WidowX (figura 17).

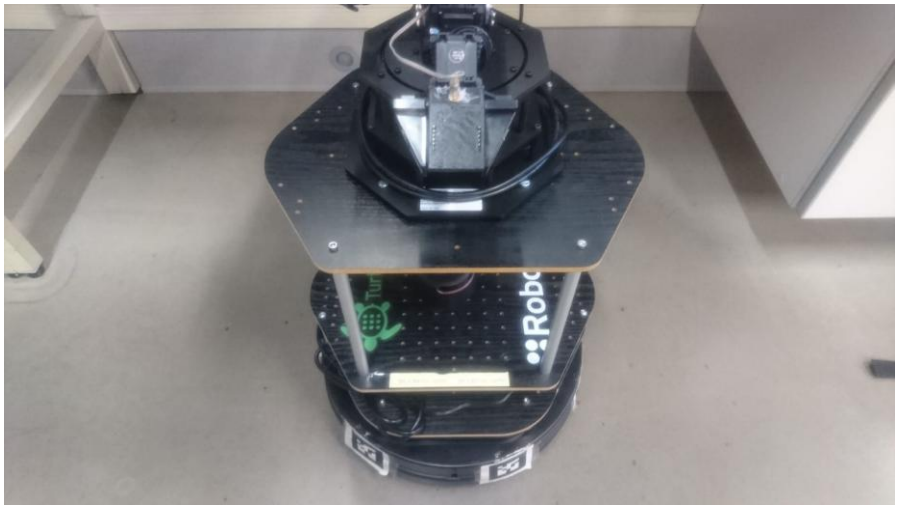


Figura 17 Estructura Kobuki + WidowX.

La base móvil Kobuki funciona gracias a un ordenador de placa reducida Odroid (figura 18) que se encarga de las funciones de control de alto nivel. La palabra Odroid proviene de juntar Open y Android, por lo que puede hacer funcionar tanto distribuciones Android como otras libres, por ejemplo, distribuciones de Linux. Para que se pueda realizar cualquier tipo de control sobre los sensores y actuadores que presenta el robot la base incorpora un controlador en su interior que está en constante comunicación con el procesador Odroid.

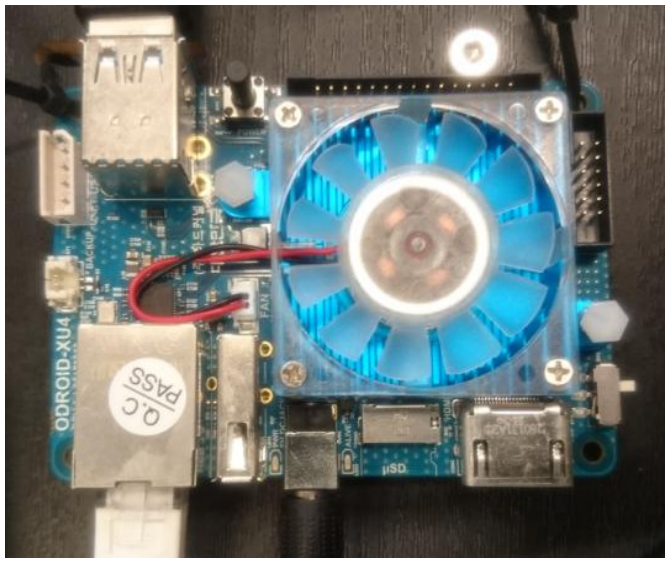


Figura 18 Procesador Odroid.

A pesar de que la placa Odroid es el núcleo de la base, Kobuki incorpora herramientas que amplían sus posibilidades de funcionamiento. Posee un acelerómetro que le permite hacer uso de la función de odometría, con la cual se puede estimar la posición del robot en comparación con la posición inicial. Por lo tanto define 3 ejes de coordenadas: se mueve sobre el plano XY y gira sobre el plano Z (figura 19). Solo tiene la posibilidad de moverse de forma recta en la dirección del eje X y hacer giros en torno al eje Z, limitados a 70 cm/s y 180º/s respectivamente. Para la evaluación de los giros trae incorporado un giróscopo sencillo.

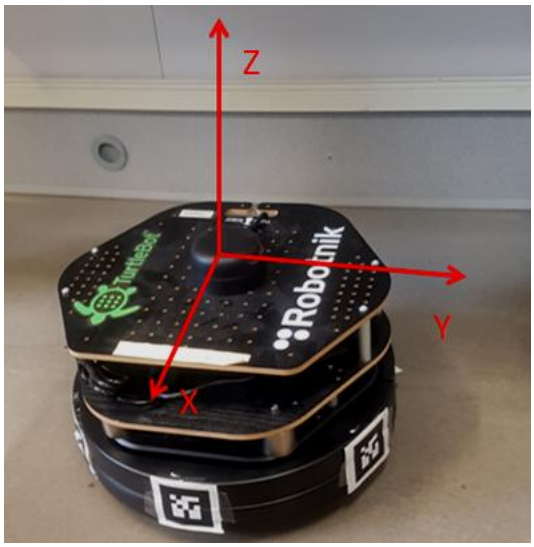


Figura 19 Base Kobuki con eje cartesiano.

Entre los sensores que incorpora, los más destacables son el sensor de caída de las ruedas y el bumper. El sensor de detección de caída de las ruedas informa de que alguna de las ruedas deja de estar en contacto con el suelo. En cuanto a los bumpers, la base Kobuki incorpora tres en su parte frontal (figura 20): izquierdo, derecho y central. Estos se activan en el momento en el que alguno de ellos contacta con una superficie.

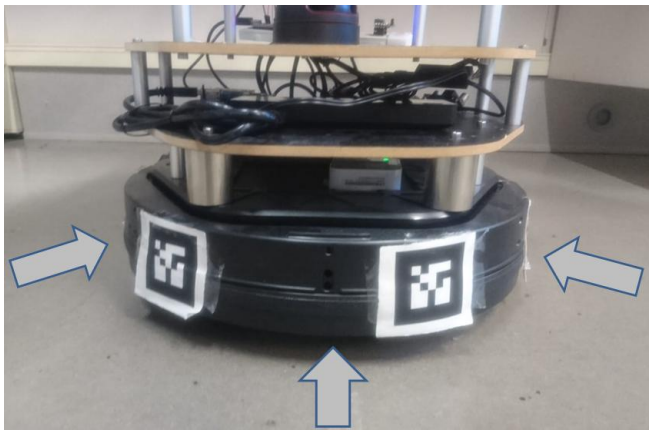


Figura 20 Bumpers Kobuki.

La autonomía del robot es dependiente de la batería y en estado normal ronda las 5 horas. Un led informa del estado de la batería en función de los siguientes casos: parpadea si está cargando, es verde si tiene batería suficiente y naranja si la batería es baja.

Al robot Kobuki se le pueden añadir diferentes elementos (sensores infrarrojos, cámaras u otros robots) haciendo insuficientes el número de puertos USB disponibles. Es por ello que se le ha incorporado un hub USB, que es un concentrador que extiende el número de conexiones vía USB mediante un solo bus.

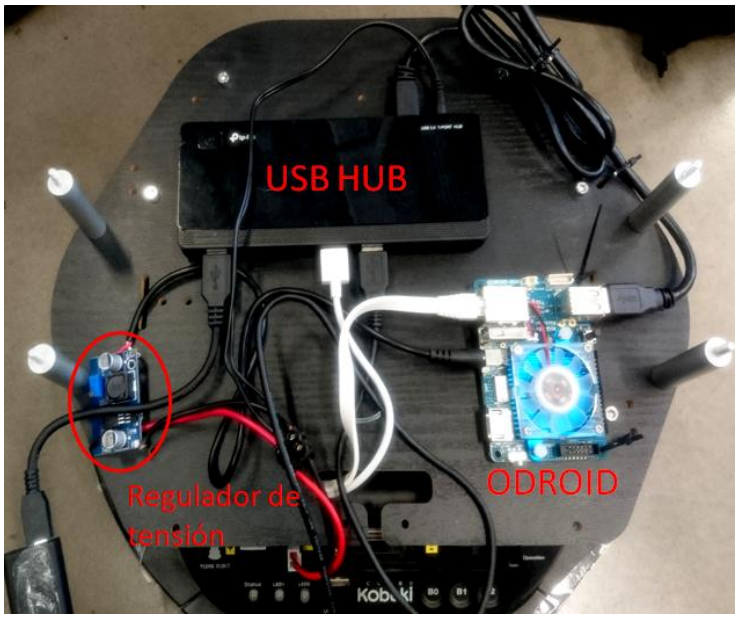


Figura 21 Hardware interno del Kobuki.

9.2.3 Sensor térmico MLX90614:

Se trata de un sensor que ofrece la posibilidad de medir tanto la temperatura de un objeto como la temperatura del ambiente. La resolución teórica del sensor es de 0.02 ºC y puede dar resultados entre rangos de 5 a 80 ºC según el fabricante. Para funcionar necesita estar conectado a un Arduino, generalmente Arduino nano para aprovechar el pequeño tamaño del sensor. El campo de medida tiene forma cónica, situándose la punta del cono en la base del sensor, de tal forma que la medición será menos precisa cuanto más cerca del objeto se encuentre el sensor. Tiene un ángulo máximo de medida de 90º.

9.2.4 Módulos de comunicación inalámbrica XBee:

Un XBee es un chip utilizado para crear pequeñas redes inalámbricas punto a punto, punto multipunto o malladas usando el protocolo IEEE 802.15.4. La interfaz de comunicación habitual de un XBee suele ser una interfaz serie, lo cual hace de estos dispositivos elementos de comunicación conectables a casi cualquier tipo de microcontrolador o

microprocesador (en el caso de este proyecto, un Arduino). Su alimentación es de 5 voltios y sus pines para comunicaciones son Rx y Tx, alimentados a 3.3 V, que habrá que tenerlos en cuenta a la hora de realizar las conexiones físicas en función de si se trata del XBee destinado a la emisión de datos o a la recepción de los mismos. El Arduino a emplear se trata de un Arduino Nano, que se encuentra en un XBee USB explorer. Este elemento, se alimenta a los 5 Voltios necesarios para el funcionamiento del microcontrolador, y presenta el inconveniente de que incluye un conector USB mini-B, por lo que es necesario un cable que permita la conexión entre USB y USB mini. Para hacer la adaptación a la tensión de los pines se ha seguido el siguiente tutorial: https://www.parallax.com/sites/default/files/downloads/32400-XBee-USB-Adapter-Documentation-v1.0.pdf.

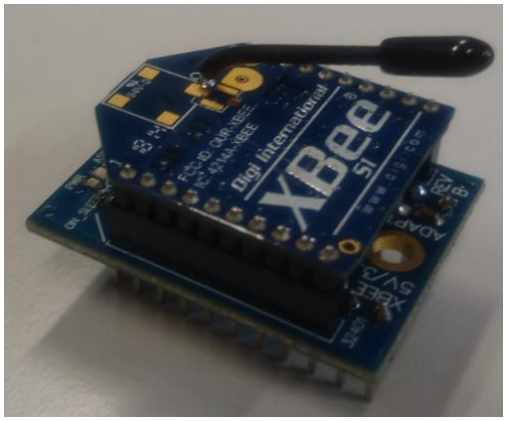


Figura 22 Módulo de comunicación inalámbrica XBee.

9.3 Software

9.3.1 Gestión del sistema

Desde el ordenador de placa reducida Odroid se gestionan todos los controladores del sistema. Por eso, es fundamental que sobre dicha placa esté instalado ROS, ya que a cada equipo se le van a asignar sus drivers correspondientes con los que poder trabajar en este entorno. En líneas generales, desde un ordenador se le cargan los programas desarrollados haciendo uso de ROS y éstos corren sobre la placa. Por lo tanto, el monoprocesador es el encargado de arrancar todos los controladores y adicionalmente, el maestro que permite la comunicación. Analizando el proyecto conjuntamente, deberá gestionar el intercambio de información entre los XBees, el nodo que gestiona el sensor

térmico, el controlador que permite los movimientos del brazo y el controlador que permite gestionar la base.

9.3.2 Software Kobuki

La estructura de software del robot Kobuki es muy sencilla. Dispone de una API (Application Programming Interface) que ofrece la posibilidad de programar en C++ haciendo uso de Linux o directamente desde Windows, de trabajar en ROS o de usar la plataforma Gazebo que sirve para mapear el entorno y poder hacer simulaciones en 3D. Por ello, para poder realizar este trabajo al robot Kobuki en su procesador Odroid se le han instalado los drivers para funcionar con el entorno de ROS.

9.3.3 Nodos y tópicos de Kobuki

Una vez están incorporados los drivers de Kobuki para funcionar en ROS, el usuario tiene la posibilidad de desarrollar sus propios paquetes con los nodos que desee para realizar el control del robot o de utilizar la serie de paquetes que proporciona el fabricante. En ocasiones, estos paquetes pueden resultar insuficientes para realizar ciertas tareas ya que incorporan lo estrictamente necesario para hacer funcionar el hardware que proporcionan, por lo que, en general, es necesario desarrollar código adicional para implementar la funcionalidad deseada. En este caso, al paquete instalado se puede acceder a través de este link: https://github.com/yujinrobot/kobuki.

Entre los paquetes que se descargan el que hace la función de controlador y permite la ejecución de los nodos de los que se vale el robot para poder moverse es el llamado *Kobuki_node*. Cuando dicho controlador es lanzado, se activan ciertos parámetros y nodos, con lo cual se activan también gran cantidad de tópicos a los cuales es posible suscribirse o publicar en ellos. El nodo principal y gracias al cual funciona el controlador recibe el nombre de *Kobuki Nodelet*. Este nodo es el que gestiona toda la información de la base móvil. A continuación, se procede a explicar los tópicos publicados o suscritos a este nodo de mayor utilidad para el desarrollo de este trabajo, así como los tipos de mensaje que maneja cada uno.

1. *Mobile_base/commands/velocity*

El nodo *Kobuki Nodelet* está suscrito a este tópico. Se trata del tópico en el que se publican los mensajes que definen las velocidades a las que se mueve el robot, tanto lineales como angulares. Para ello, hay que tener en cuenta el triedro cartesiano definido en la figura 19 y que este robot solo proporciona la opción de moverse en línea recta y de girar en torno a su eje vertical. La estructura del mensaje es la indicada en la figura 23. En la misma figura se observa que vector3 lo forman números reales de 64 bits, uno por cada eje. Por lo tanto, se pueden publicar tanto la velocidad lineal como la velocidad angular deseada para cada eje.

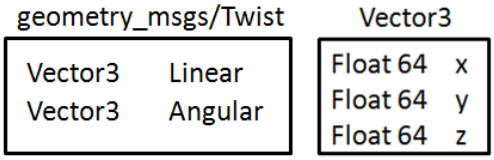


Figura 23 Tipo de mensaje para establecer la velocidad del Kobuki.

2. *Mobile_base/events/bumper*

El nodo *Kobuki Nodelet* está publicando en este tópico. Se trata del tópico en el que se publican los mensajes generados cuando uno de los tres bumpers se encuentra en contacto con algún obstáculo. Al tratarse en este caso de un evento, no está constantemente publicando información, sino que solo se publica cuando la acción ocurre indicando cual es el bumper que se encuentra activo. Como se observa en la figura 24 en el mensaje se define cuál es el bumper que contacta y su estado.

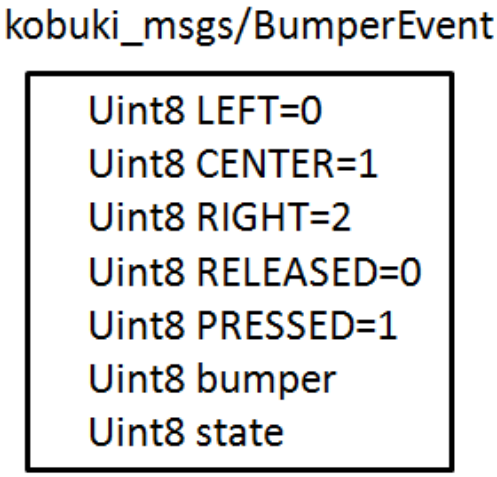


Figura 24 Tipo de mensaje que informa del estado del bumper.

3. Odom

El nodo *Kobuki Nodelet* está publicando en este tópico. Se trata del tópico que proporciona información sobre la odometría del robot. Al ejecutar el controlador por primera vez, se define un origen de coordenadas junto con un triedro cartesiano. En este robot en concreto, la odometría devuelve la posición y orientación del robot respecto del triedro definido en su arranque. Además, la presencia de un acelerómetro junto a un giróscopo en el robot permite a través de este tópico acceder a los valores de velocidad lineal y angular en un instante del movimiento. Existe otro tópico que permite resetear esta función para definir el origen en otro lugar teniendo el controlador arrancado. Un factor importante a tener en cuenta es que este tópico solo ofrece la posibilidad de suscribirse a él, ya que su función es analizar los movimientos realizados y en ningún momento actuar sobre el robot.

Atendiendo a la figura 25 la estructura del mensaje que hay que manejar para manipular el tópico *odom* es bastante compleja en relación a las demás y para poder entenderlo es necesario ir punto por punto. El mensaje presenta una cabecera concreta que hay que tener en cuenta pero los campos más interesantes son *PoseWithCovariance* y *TwistWithCovariance*.

Dentro del apartado *PoseWithCovariance* se pueden encontrar los campos *Pose* y *covariance*. El campo *covariance* devuelve un número real con la covarianza pero es especialmente importante el contenido de *Pose*. Dentro de él se puede acceder a un apartado que devuelve la posición y a otro que devuelve la orientación. Internamente, tanto *Position* como *Orientation*, presentan una subestructura similar: se tratan como números reales; en el primer caso la posición de cada eje y en el segundo la orientación de cada eje. Por lo tanto, para acceder a la posición u orientación del robot en un determinado momento, habrá que ir avanzando progresivamente hacia el interior del mensaje siguiendo la estructura descrita.

Dentro del apartado *TwistWithCovariance* se puede acceder a los apartados *Twist* y *covariance*. De nuevo ocurre como en el caso anterior, *covariance* devuelve un número real con la covarianza pero es especialmente interesante conocer el contenido del apartado *Twist*. En su interior se encuentran todos los valores de velocidades lineales y angulares del robot. De nuevo, a cada eje le corresponde un número real que en el apartado linear representará su velocidad lineal y en el

apartado angular, su velocidad angular. De esta forma y al igual que para acceder a la posición y orientación del robot, para conocer sus velocidades lineales y angulares habrá que seguir está estructura definida y saber moverse en ella.

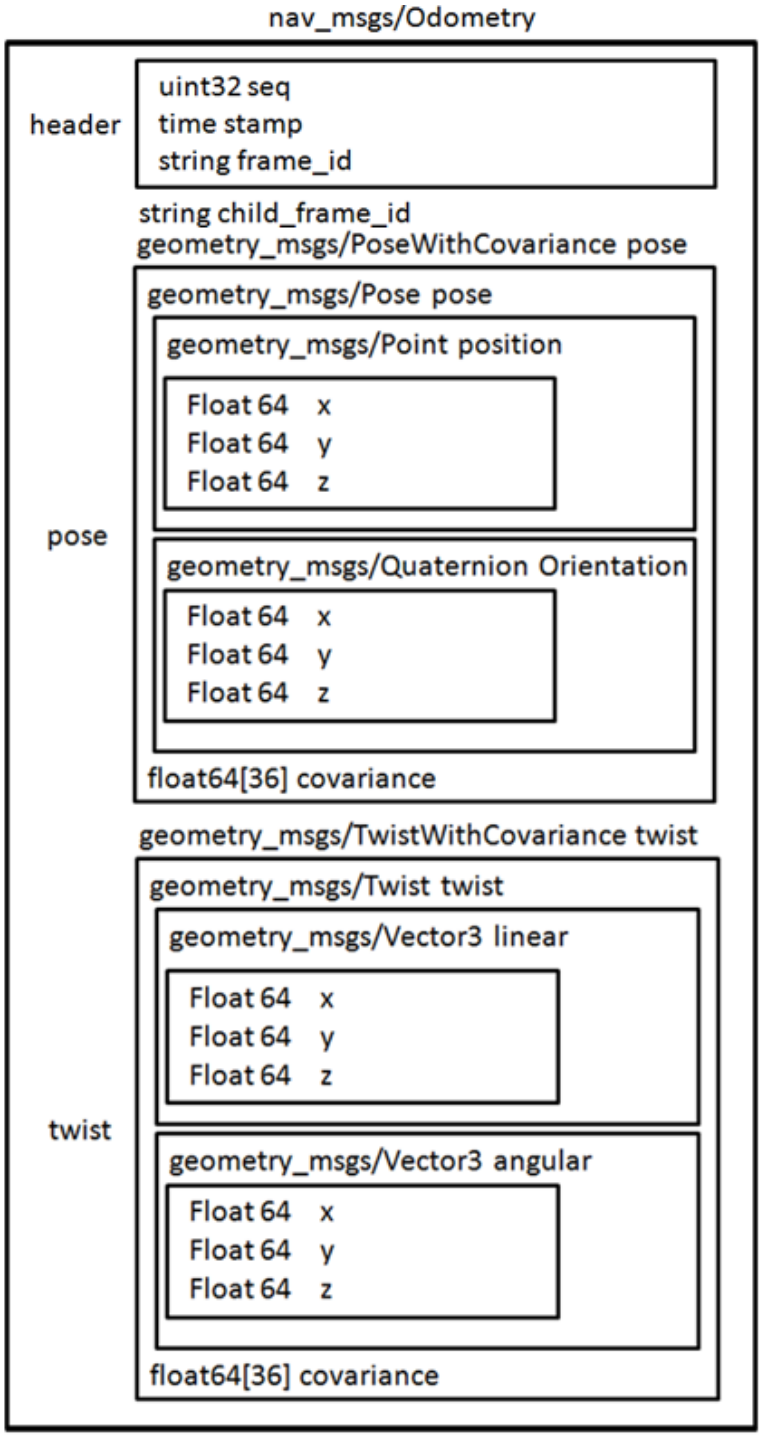


Figura 25 Tipo de mensaje para interpretar el tópico *Odom*.

9.3.4 Diseño de bajo nivel

Una vez realizada la descripción de los nodos y tópicos accesibles, procede explicar cómo se ha realizado el control del movimiento. En primer lugar, se presentan dos posibilidades para realizarlo.

La primera de ellas, consistiría en planificar los movimientos en función de la velocidad y el tiempo. Como ya se ha comentado, las velocidades pueden publicarse en el tópico *Mobile_base/commands/velocity* y además, la librería Rospy para programar en python permite obtener la fecha y hora del sistema en un momento dado. De esta forma, puede plantearse para el movimiento lineal que el espacio recorrido es igual al producto de la velocidad por el tiempo transcurrido. Para el giro del robot, la idea sería la misma, sin más que hacer que el giro realizado es igual a la velocidad angular por el tiempo transcurrido. A nivel de programación esto podría llevarse a cabo registrando el instante en que se inicia la trayectoria en una variable y realizando un bucle (por ejemplo, *while*) en el que publicar la velocidad de movimiento del Kobuki. La condición sería que mientras la distancia calculada dentro del ciclo fuera inferior a la deseada, dentro del ciclo se actualizase una segunda variable para actualizar el tiempo y se registrara después la distancia calculada, como el producto de la velocidad por la diferencia de tiempo entre el instante actual (segunda variable tiempo que se actualiza con cada ejecución del ciclo) y el instante inicial (primera variable tiempo).

La segunda de ellas se trata de aprovechar la disponibilidad de la función Odometry. La idea consiste en definir dos variables, una para conocer la posición y otra para conocer la orientación del robot, en el nodo que contiene el código. El siguiente paso sería suscribirse al tópico en el que se encuentra dicha información y almacenar en las variables de posición y orientación sus valores correspondientes en cada instante teniendo en cuenta el tipo de mensaje que se maneja (figura 26). A nivel de programación, simplemente habría que programar un bucle (por ejemplo, *while*) dentro del cual se publica la velocidad de movimiento del robot, usando la comparación entre la distancia que se quiere recorrer y la devuelta por el tópico *Odom* para la condición de desplazamiento. Para ello, habrá que tener en cuenta el sistema de referencia creado a partir del cual la función Odometry evalúa su posición.

```
def callback(mensaje):
    global orientacion
    global distancia
    orientacion = mensaje.pose.pose.orientation.z
    distancia = mensaje.pose.pose.position.x
```

Figura 26 Definición de variables para acceder al tópico *Odom*.

Valorando las dos posibilidades, la más apropiada para realizar el control de la base a pesar de ser más compleja, es la que aprovecha la forma de funcionar en ROS y se suscribe al tópico *Odom*. Además, de esta manera el control será más preciso y presentará menos error, debido a que únicamente depende del acelerómetro y del giróscopo que se encuentran bien calibrados y a que posibles variaciones en el tiempo de ejecución no afectarán al programa. Además del uso de Odometry, para dotar al programa de mayor seguridad se va a incorporar la condición de que, en el caso de colisión, es decir, en caso de que cualquiera de los 3 bumpers se active, el movimiento se bloquee hasta que el obstáculo desaparezca. De este modo, en el entorno ROS, el script desarrollado sería un nodo el cual se está suscribiendo al tópico *Odom* para evaluar su posición y al tópico *Mobile_base/events/bumper* para asegurarse de que no hay choques. Al mismo tiempo, dicho nodo después de hacer uso de la información extraída de los tópicos a los que está suscrito, publica la velocidad a la que se va a mover el robot en el tópico *Mobile_base/commands/velocity*.

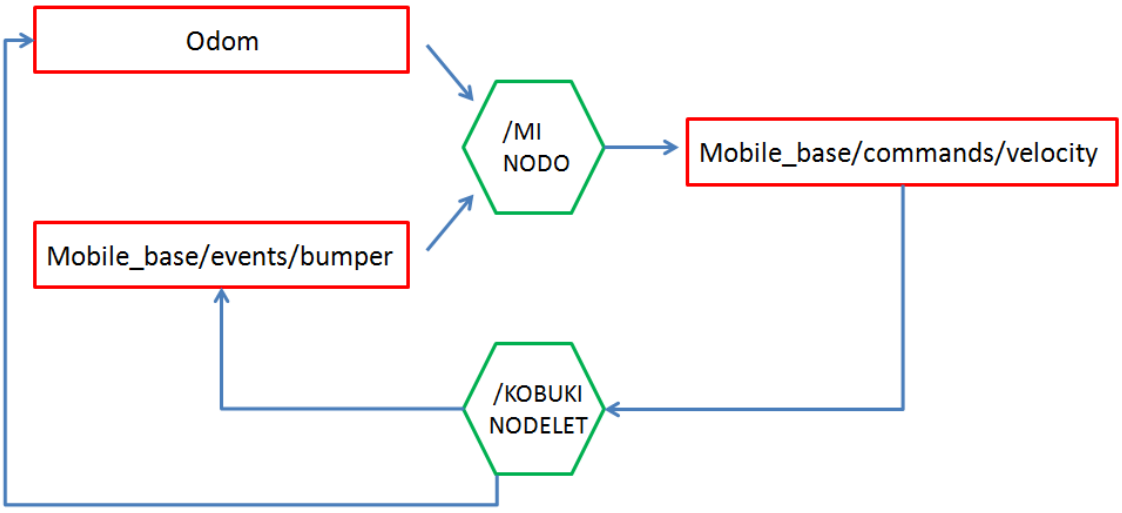


Figura 27 Esquema de funcionamiento en ROS para controlar el movimiento del Kobuki.

9.3.5 Comunicación inalámbrica XBee

A nivel de software son necesarias varias operaciones de configuración para hacer funcionar los diferentes dispositivos que integran la plataforma robótica que se maneja en este proyecto. En primer lugar, para poder conectar el cable adaptador USB a USB mini se deben instalar los drivers oportunos. En el siguiente enlace se puede encontrar un manual informativo:
https://www.ftdichip.com/Support/Documents/DataSheets/Modules/DS_UB232R.pdf.

Una vez está todo preparado para funcionar, es necesario hacer uso del programa XCTU (XBee Configuration and Test Utility) proporcionado por el fabricante Digi. Este software permite identificar y configurar cada XBee además de proporcionar información de sus características. Se debe conectar cada XBee a un puerto serie del ordenador y dentro del programa identificar en qué puerto está conectado cada uno. Aquí se comprueba que la configuración de cada uno es correcta, se comprueba que los XBees se encuentran y se procede a actualizar el firmware de cada uno. Cuando están ambos actualizados, se puede comprobar si la comunicación entre ellos es correcta gracias a una consola que ofrece XCTU.

Con los XBees funcionando, hay que configurar el microcontrolador Arduino que va a gestionar la función de cada módulo. Por ello, habrá que cargar un programa diferente a cada Arduino en función de si se encarga de gestionar la transmisión o la recepción de información. Los programas cargados inicializan un puerto serie virtual en los pines Rx y Tx, a través de los cuales se recibe o envía información. En el caso del emisor recibe la información proporcionada por el sensor y la manda por dicho puerto. Cabe destacar que el Arduino que se encarga de gestionar al XBee emisor y el funcionamiento del sensor es el mismo, por lo que en él se cargará un programa conjunto que lea los datos del sensor y después los mande. En el caso del receptor, recibe la información por el puerto creado y la envía por el puerto serie hardware. De esta forma, primero se comprobó que el XBee encargado de mandar datos funcionaba, mostrando los resultados en el monitor serie del que dispone el programa Arduino. Después, se pudo comprobar que desde un módulo la información le llegaba al otro y este último, la mostraba en el monitor serie.

Habiendo comprobado la comunicación y configurados los Arduinos, solo falta integrar el proceso en el entorno de trabajo ROS. Teniendo todos los pasos anteriores bien realizados,

este apartado resulta sencillo, ya que simplemente se trata de realizar un script que lea la información recibida en el puerto serie del Odroid al que está conectado el XBee que recibe los datos. Para ello, se ha realizado un script, es decir, un nodo que lee los datos del sensor vía serie y los publica en un tópico al que posteriormente el programa principal se suscribirá para poder usar dicha información (figura 28).

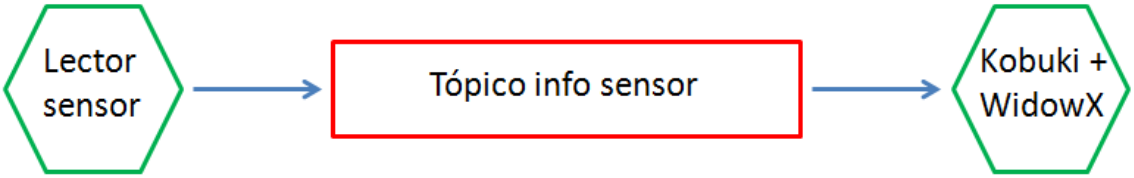


Figura 28 Esquema en ROS de intercambio de información con el sensor.

10 Descripción de los resultados

En este apartado se van a evaluar los resultados obtenidos en el trabajo y si se ha cumplido con los objetivos y requerimientos establecidos.

Un aspecto importante es que se ha aprovechado el aspecto de comunidad Open Source con el que trabaja ROS y se ha compartido el código desarrollado durante todo el trabajo en este repositorio de Github: https://github.com/JulenCuadra/TFG_Kobuki_WidowX. Además se dispone de una guía de puesta en marcha en el Anexo 1 de este documento.

El objetivo principal era realizar el control de la plataforma móvil Kobuki y establecer las comunicaciones inalámbricas para gestionar la información del sensor de la pinza. Los requerimientos establecían varias secuencias de movimiento debiendo respetar las distancias que imponen la recogida y deposición de la pieza por parte del brazo. Atendiendo a estos hechos, se ha conseguido desarrollar un script en python que mediante el uso de ROS interacciona con el brazo y permite al robot realizar giros y traslaciones para realizar el transporte prefijado.

Con respecto a las comunicaciones inalámbricas, se ha logrado realizar la implementación y configuración de los módulos XBee, permitiendo así gestionar los datos del sensor desde la placa Odroid con facilidad y eliminando la limitación que supondría para el brazo que esa conexión fuese física. Además, se ha introducido en ROS, haciendo la información intercambiada muy accesible para el script principal.

11 Plan de trabajo

11.1 Descripción de equipos

Al tratarse de un trabajo que va a formar parte de un proyecto más grande que lo que engloba el alcance de este mismo, el equipo de proyecto va a estar formado por todas las personas que sean participes en el proyecto final. Es por ello, que el equipo estará formado por un director de proyecto, dos ingenieros junior y una persona encargada de la administración y gestión del proyecto.

El director de proyecto es Oskar Casquero Oyarzabal y su función es dirigir el proyecto y orientar a los ingenieros junior en su trabajo.

Los ingenieros junior son Asier Alonso Tejeda y Julen Cuadra Gómez y su función es desarrollar su correspondiente trabajo para conseguir combinar los movimientos del robot móvil y el brazo robótico teniendo en cuenta la lectura del sensor térmico.

La encargada de la administración es Maialen González Jaio y se encarga de la gestión del proyecto y todo lo relacionado con el mismo.

11.2 Descripción de fases y tareas

A continuación, se van a desglosar todas las fases del proyecto y las tareas que hay incluidas dentro de cada una de estas fases. En cada caso, se dará una breve descripción y se asignará un responsable, unos participantes y se reflejará la duración.

Fase 1- Gestión: esta fase se encarga de asegurar que el proyecto funciona correctamente y que se cumplen todas las fases, tareas y requisitos.

Responsable: Maialen González Jaio.

Participantes: Asier Alonso Tejeda y Julen Cuadra Gómez.

Duración: 109 días.

Fase 2- Puesta a punto de los equipos: esta fase está dedicada a realizar todos los posibles montajes y configuraciones necesarias a los equipos que se disponen para poder ponerse a trabajar en el desarrollo de programación.

Responsables: Asier Alonso Tejeda y Julen Cuadra Gómez.

Participante: Maialen González Jaio.

Duración: 19 días.

> **Tarea 2.1- Montaje WidowX:** esta tarea contempla todo lo relacionado con el montaje del brazo robótico. Para ello, es necesario realizar la identificación de cada servomotor que compone el robot, configurar su controlador Arbotix adecuadamente y montar toda su estructura.
>
> **Responsable**: Julen Cuadra Gómez.
>
> **Participantes**: Maialen González Jaio y Asier Alonso Tejeda.
>
> **Duración**: 12 días.
>
> **Tarea 2.2- Configuración Turtlebot 2:** el objetivo de esta tarea es configurar el robot Kobuki y montar sus componentes. Por ello, dentro de esta tarea se incluye la puesta en marcha de la base, el montaje de su estructura y la colocación del brazo robótico sobre ella.
>
> **Responsable**: Asier Alonso Tejeda
>
> **Participantes**: Maialen González Jaio y Julen Cuadra Gómez.
>
> **Duración**: 13 días.
>
> **Tarea 2.3- Instalación software**: se trata de realizar la puesta a punto de los equipos a nivel de software, por ello contempla la instalación de Linux y ROS en los equipos de trabajo, ya sean ordenadores como procesadores.
>
> **Responsables**: Asier Alonso Tejeda y Julen Cuadra Gómez.
>
> **Participante**: Maialen González Jaio.
>
> **Duración**: 6 días.

Fase 3- Formación: esta fase está dedicada a aprender a manejar los equipos a nivel de hardware y software, es decir, a familiarizarse con las herramientas de trabajo.

Responsable: Maialen González Jaio.

Participantes: Asier Alonso Tejeda, Julen Cuadra Gómez y Oskar Casquero Oyarzabal.

Duración: 32 días.

Tarea 3.1- Familiarización con Linux: se trata de aprender a utilizar Linux, es decir, conocer el sistema operativo y los comandos más habituales.

Responsable: Asier Alonso Tejeda.

Participantes: Maialen González Jaio y Julen Cuadra Gómez.

Duración: 7 días.

Tarea 3.2- Familiarización con ROS: se trata de aprender a usar ROS, es decir, conocer su forma de funcionar, sus posibilidades y sobre todo su estructuración para gestionar la comunicación.

Responsables: Julen Cuadra Gómez.

Participantes: Maialen González Jaio y Asier Alonso Tejeda.

Duración: 10 días.

Tarea 3.3- Aprendizaje python: se trata de adquirir el conocimiento suficiente para poder escribir scripts en lenguaje python en el entorno de ROS.

Responsables: Oskar Casquero Oyarzabal.

Participantes: Maialen González Jaio, Asier Alonso Tejeda y Julen Cuadra Gómez.

Duración: 15 días.

Fase 4- Implementación de equipos en ROS: en esta fase se va a realizar el control de los equipos mediante ROS.

Responsables: Maialen González Jaio, Asier Alonso Tejeda y Julen Cuadra Gómez.

Duración: 57 días.

Tarea 4.1- Instalación paquetes WidowX: en esta tarea se deben instalar los paquetes necesarios para poder controlar el brazo WidowX con ROS en todos los ordenadores y procesadores que se vaya a trabajar.

Responsables: Julen Cuadra Gómez.

Participantes: Maialen González Jaio y Asier Alonso Tejeda.

Duración: 5 días.

Tarea 4.2- Instalación paquetes Kobuki: en esta tarea se deben instalar los paquetes necesarios para poder controlar la base Kobuki con ROS en todos los ordenadores y procesadores que se vaya a trabajar.

Responsables: Asier Alonso Tejeda.

Participantes: Maialen González Jaio y Julen Cuadra Gómez.

Duración: 3 días.

Tarea 4.3- Desarrollo programas WidowX: esta tarea consiste en realizar el desarrollo de los programas que sean necesarios para poder llevar a cabo el control de todas las trayectorias que debe realizar el brazo robótico.

Responsables: Julen Cuadra Gómez.

Participantes: Maialen González Jaio y Asier Alonso Tejeda.

Duración: 15 días.

Tarea 4.4- Desarrollo programas Kobuki: esta tarea consiste en realizar el desarrollo de los programas que sean necesarios para poder llevar a cabo el control de todas las trayectorias que debe realizar la base móvil.

Responsables: Asier Alonso Tejeda.

Participantes: Maialen González Jaio y Julen Cuadra Gómez.

Duración: 15 días.

Tarea 4.5- Implementación sensor de temperatura: el objetivo de esta tarea es implementar sobre la pinza del manipulador un sensor de temperatura. Para ello, en primer lugar habrá que elegir el sensor apropiado y diseñar y fabricar el soporte sobre el que incluirlo. Después, habrá que implementarlo sobre WidowX y comprobar su funcionamiento.

Responsables: Julen Cuadra Gómez.

Participantes: Asier Alonso Tejeda y Maialen González Jaio.

Duración: 12 días.

Tarea 4.6- Implementación XBees y Arduinos: esta tarea está dedicada a implementar los XBees y Arduinos en el sistema de trabajo. Para realizarlo, hay que configurar los XBees, realizar el diseño de todos los programas de Arduino e implementar estos dispositivos tanto en la base móvil Kobuki como en el brazo WidowX, comprobando que todas las comunicaciones son correctas.

Responsables: Asier Alonso Tejeda.

Participantes: Maialen González Jaio y Julen Cuadra Gómez.

Duración: 10 días.

Tarea 4.7- Combinación de todos los equipos: en esta tarea se van a combinar todos los módulos diseñados y programados individualmente para conseguir un control conjunto del sistema.

Responsables: Asier Alonso Tejeda, Maialen González Jaio y Julen Cuadra Gómez.

Duración: 15 días.

Fase 5- Validación de resultados: esta fase tiene como objetivo que los resultados obtenidos en el desarrollo del proyecto reciban el visto bueno final del director.

Responsables: Oskar Casquero Oyarzabal.

Duración: 4 días.

Fase 6- Documentación del proyecto: fase que consiste en documentar el proyecto y redactar el trabajo final a presentar.

Responsables: Maialen González Jaio, Asier Alonso Tejeda y Julen Cuadra Gómez.

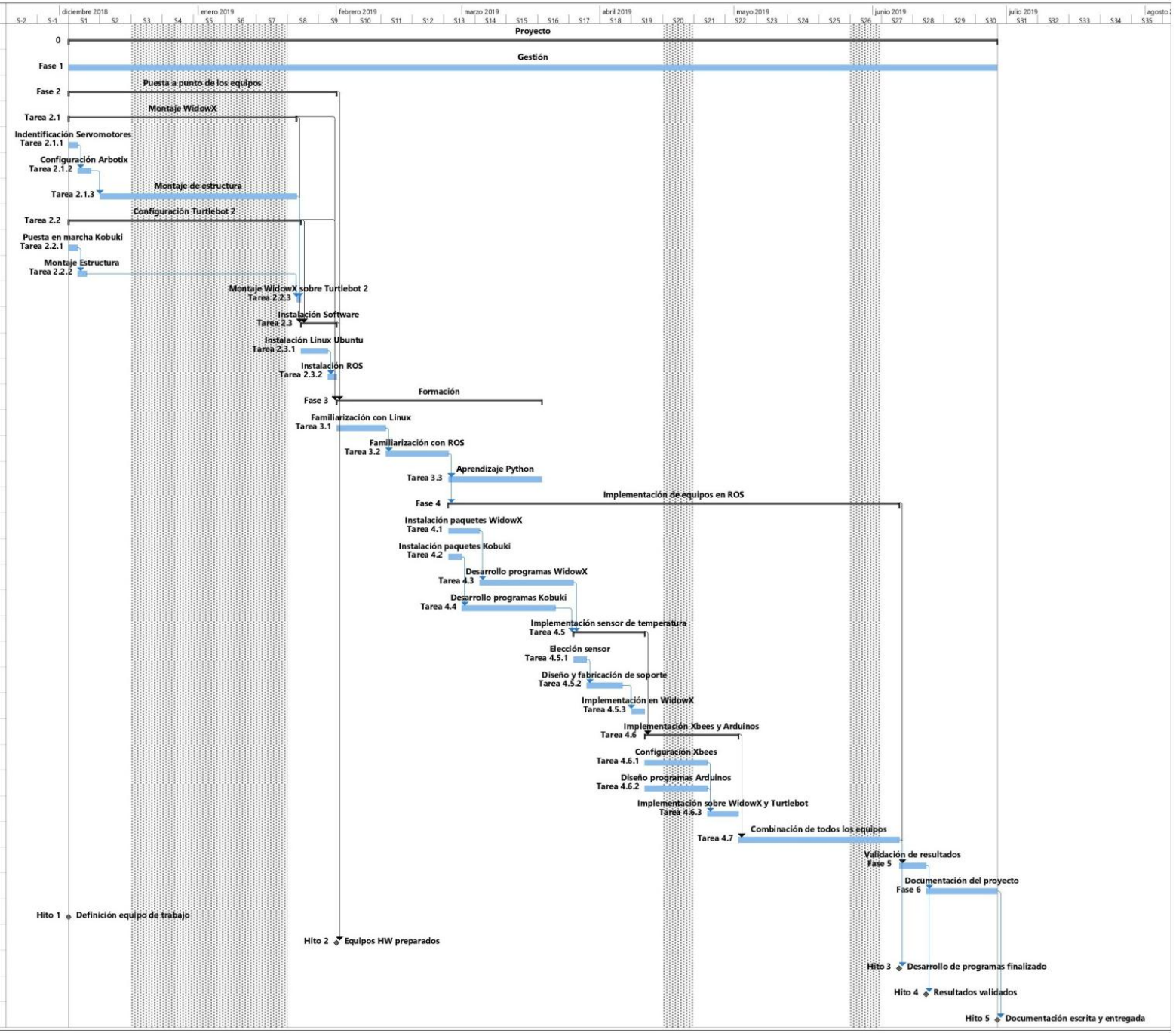
Duración: 12 días.

En adición, se han definido varios hitos para hacer un seguimiento correcto del proyecto. Estos son los siguientes:

1. Definición de equipo de trabajo.
2. Equipos hardware preparados.
3. Desarrollo de programas finalizado.
4. Resultados validados.
5. Documentación escrita y entregada.

11.3 Diagrama de Gantt

Id	EDT	Nombre de tarea	Duración
1	**0**	**Proyecto**	**109 días**
2	Fase 1	Gestión	109 días
3	**Fase 2**	**Puesta a punto de los equipos**	**19 días**
4	**Tarea 2.1**	**Montaje WidowX**	**12 días**
5	Tarea 2.1.1	Indentificación Servomotores	2 días
6	Tarea 2.1.2	Configuración Arbotix	3 días
7	Tarea 2.1.3	Montaje de estructura	7 días
8	**Tarea 2.2**	**Configuración Turtlebot 2**	**13 días**
9	Tarea 2.2.1	Puesta en marcha Kobuki	2 días
10	Tarea 2.2.2	Montaje Estructura	2 días
11	Tarea 2.2.3	Montaje WidowX sobre Turtlebot 2	1 día
12	**Tarea 2.3**	**Instalación Software**	**6 días**
13	Tarea 2.3.1	Instalación Linux Ubuntu	4 días
14	Tarea 2.3.2	Instalación ROS	2 días
15	**Fase 3**	**Formación**	**32 días**
16	Tarea 3.1	Familiarización con Linux	7 días
17	Tarea 3.2	Familiarización con ROS	10 días
18	Tarea 3.3	Aprendizaje Python	15 días
19	**Fase 4**	**Implementación de equipos en ROS**	**57 días**
20	Tarea 4.1	Instalación paquetes WidowX	5 días
21	Tarea 4.2	Instalación paquetes Kobuki	3 días
22	Tarea 4.3	Desarrollo programas WidowX	15 días
23	Tarea 4.4	Desarrollo programas Kobuki	15 días
24	**Tarea 4.5**	**Implementación sensor de temperatura**	**12 días**
25	Tarea 4.5.1	Elección sensor	3 días
26	Tarea 4.5.2	Diseño y fabricación de soporte	6 días
27	Tarea 4.5.3	Implementación en WidowX	3 días
28	**Tarea 4.6**	**Implementación Xbees y Arduinos**	**10 días**
29	Tarea 4.6.1	Configuración Xbees	5 días
30	Tarea 4.6.2	Diseño programas Arduinos	5 días
31	Tarea 4.6.3	Implementación sobre WidowX y Turtlebot	5 días
32	Tarea 4.7	Combinación de todos los equipos	15 días
33	Fase 5	Validación de resultados	4 días
34	Fase 6	Documentación del proyecto	12 días
35	Hito 1	Definición equipo de trabajo	0 días
36	Hito 2	Equipos HW preparados	0 días
37	Hito 3	Desarrollo de programas finalizado	0 días
38	Hito 4	Resultados validados	0 días
39	Hito 5	Documentación escrita y entregada	0 días



12 Descripción del presupuesto y del presupuesto ejecutado

En este apartado se van a presentar dos presupuestos bien diferenciados. Por un lado, uno que recoge la parte conceptual del desarrollo del proyecto, es decir, las horas internas de ingeniería que han sido necesarias para llevar a cabo las ideas principales para la consecución del objetivo final. Por otro lado, un presupuesto que recoge los costes de ejecución del proyecto, ya sean de montaje, fabricación o los debidos a pruebas de funcionamiento.

Tabla 5 Presupuesto desarrollo conceptual del proyecto.

Concepto	Unidades	Coste unitario	Nº unidades	Coste total	Total partida
Horas internas					9.400,00 €
Director de proyecto	horas	35,00 €	30	1.050,00 €	
Ingeniero junior	horas	25,00 €	238	5.950,00 €	
Gestor	horas	20,00 €	120	2.400,00 €	
Amortizaciones					77,60 €
Ordenador	horas	0,20 €	388	77,60 €	
Subtotal 1					9.477,60 €
Costes indirectos (10%)					947,76 €
Subtotal 2					10.425,36 €
Imprevistos (10%)					1.042,54 €
Costes totales					11.467,90 €

Como se puede observar, este presupuesto recoge fundamentalmente los costes del desarrollo de la idea por el equipo de proyecto. Cabe destacar, que al tratarse de un proyecto cuyo fin es controlar un conjunto robótico haciendo un desarrollo de software, se incluye como amortización el coste del equipo informático. El resto del presupuesto lo componen las horas internas de cada uno de los participantes. Además de esto, se ha dedicado un 10% de los costes directos a cubrir posibles costes indirectos. Por último se ha destinado un añadido calculado como el 10% de ese segundo subtotal a tener una reserva para imprevistos.

Tabla 6 Presupuesto de ejecución del proyecto.

Concepto	Unidades	Coste unitario	Nº unidades	Coste total	Total partida
Horas internas					2.205,00 €
Director de proyecto	horas	35,00 €	5	175,00 €	
Ingeniero junior	horas	25,00 €	50	1.250,00 €	
Gestor	horas	20,00 €	24	480,00 €	
Técnico de laboratorio	horas	30,00 €	10	300,00 €	
Amortizaciones					19,75 €
Ordenador	horas	0,20 €	65	13,00 €	
Impresora 3D	horas	0,30 €	20	6,00 €	
PCB	horas	0,15 €	5	0,75 €	
Gastos					3.650,91 €
WidowX		1.512,75 €	2	3.025,50 €	
Kobuki		499,00 €	1	499,00 €	
Sensor MLX90614		17,75 €	1	17,75 €	
XBee		50,00 €	2	100,00 €	
Arduino nano		4,33 €	2	8,66 €	
Subtotal 1					5.875,66 €
Costes indirectos (10%)					587,57 €
Subtotal 2					6.463,23 €
Imprevistos (10%)					646,32 €
Costes totales					7.109,55 €

En este presupuesto se recogen los costes imputables a la ejecución del proyecto. En primer lugar, en la partida de horas internas se ha añadido al equipo de trabajo definido, el técnico de laboratorio, que ha sido necesario en diversos momentos como realizar una impresión en 3D o fabricar una PCB. Dentro de las horas internas se recogen tanto horas destinadas al montaje de los equipos como horas destinadas a la realización de pruebas para comprobar o validar las ideas desarrolladas. En el montaje y preparación de los equipos, esta vez también ha sido necesario el uso de ordenadores, por lo que el apartado de amortizaciones recoge los equipos informáticos, además de los equipos requeridos para realizar impresiones 3D y fabricar la PCB. En la partida de gastos, se han recogido todos los elementos que forman parte del conjunto robótico. Durante el trabajo, se disponía de dos manipuladores WidowX, una base Kobuki, un sensor térmico, una pareja de XBees y su correspondiente pareja de Arduinos. Al igual que en el presupuesto anterior, como puede observarse se han destinado un par de apartados para cubrir costes indirectos e imprevistos que pudieran surgir durante la realización del proyecto.

13 Conclusiones

Tras la realización del trabajo y la consecución de los objetivos planteados, las conclusiones extraídas son las siguientes:

El entorno de trabajo que establece ROS ofrece grandes posibilidades no sólo en cuanto al control puro de robots. Su estructura y el hecho de que la comunicación entre nodos sea indirecta permite su integración en otro tipo de entidades heterogéneas. Es un framework robótico bastante robusto y eficaz debido a su estructura, pero sin embargo, esta estructura también provoca que sea complicado iniciarse en él. Para manejarlo, requiere conocimientos de Linux y un notable dominio de alguno de los lenguajes de programación para los que ofrece APIs. Además, la necesidad de empezar a trabajar a partir de código desarrollado por otros usuarios obtenido a través de repositorios en ocasiones ha dificultado el avance debido a que se han encontrado errores e incompatibilidades que ha habido que corregir. Por ello, trabajar con software proporcionado por terceros supone una ventaja si está bien revisado, pero cuando esto no ocurra, provocará que haya que invertir tiempo en conocer a fondo el código y parchearlo para adaptarlo al trabajo a desarrollar.

El robot móvil Kobuki tiene unas prestaciones limitadas que lo convierten en un robot académico pero apropiado para investigación y realizar pruebas de concepto. Su relativa sencillez en ese aspecto, hace que la integración de la base en otros proyectos fundamentados en ROS sea fácil a partir del desarrollo realizado en este trabajo.

Con respecto a las comunicaciones inalámbricas, se puede decir que la implementación de XBees es una solución apropiada para las necesidades de este trabajo. Sin embargo, para aplicaciones más potentes o industriales podría resultar insuficiente.

Con vistas a futuro, es necesaria la realización de una PCB que integre el Arduino que controla el sensor térmico y el XBee que transmite la información al XBee ubicado en la placa Odroid. Además de esto, con las bases del funcionamiento de la plataforma Kobuki, podrían ampliarse sus funcionalidades con la integración de elementos de hardware externos a los que proporciona para dotarle de una mayor versatilidad e independencia.

14 Bibliografía

[1] Learn.trossenrobotics.com. *WidowX Arm with ROS Getting Started Guide*. [En línea] Disponible en: https://learn.trossenrobotics.com/projects/186-widowx-arm-with-ros-getting-started-guide.html [Último acceso: 15 de diciembre de 2018].

[2] wikiHow. *Cómo instalar Ubuntu*. [En línea] Disponible en: https://es.wikihow.com/instalar-Ubuntu [Último acceso: 25 de enero de 2019].

[3] Cruz Martín, A. *ResearchGate* [En línea] Disponible en: https://www.researchgate.net/publication/291789058_Turtlebot_2_with_a_WidowX_arm_first_steps [Último acceso: 10 de abril de 2019].

[4] Anónimo [En línea] Disponible en: https://learn.sparkfun.com/tutorials/xbee-shield-hookup-guide [Último acceso: 2 de junio de 2019].

[5] Digi.com. *Digi XBee Ecosystem - Everything you need to explore and create wireless connectivity*. [En línea] Disponible en: https://www.digi.com/xbee [Último acceso: 2 de junio de 2019].

[6] Ehu.eus. [En línea] Disponible en: http://www.ehu.eus/manufacturing/docencia/66_ca.pdf [Último acceso: 10 de junio de 2019].

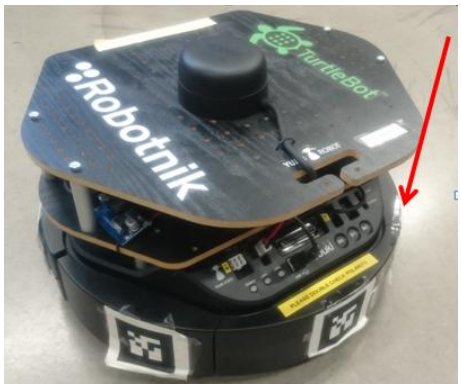
[7] KOBUKI. *About | KOBUKI*. [En línea] Disponible en: http://kobuki.yujinrobot.com/about2/ [Último acceso: 10 de junio de 2019].

[8] Wiki.ros.org. *ROS Wiki*. [En línea] Disponible en: http://wiki.ros.org/es [Último acceso: 21 de junio de 2019].

[9] Alfonso Aberasturi, G. y Armentia, A. *Puesta en marcha del control de robots de transporte.*

15 Anexo I: Manual de usuario.

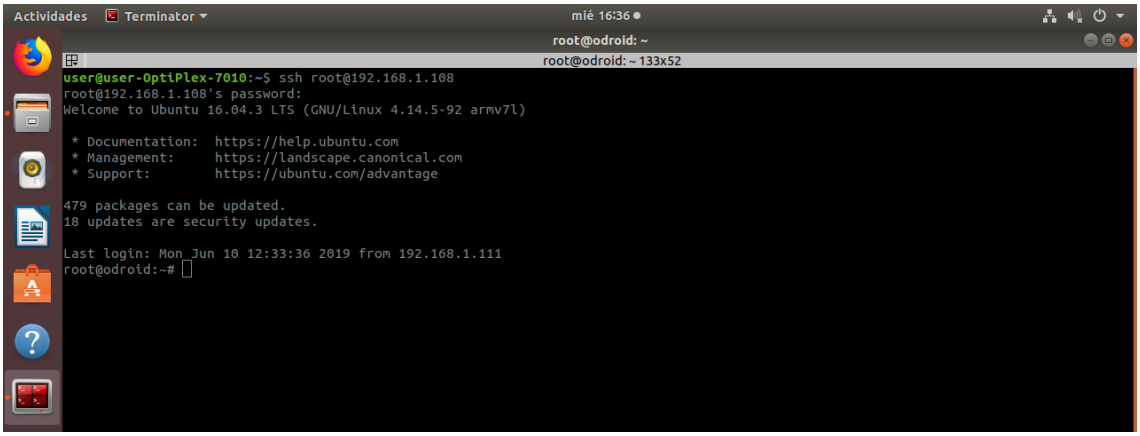
En este manual se van a explicar paso por paso las acciones que se deben realizar para poder poner en funcionamiento el conjunto Kobuki-WidowX con su correspondiente lectura del sensor térmico.



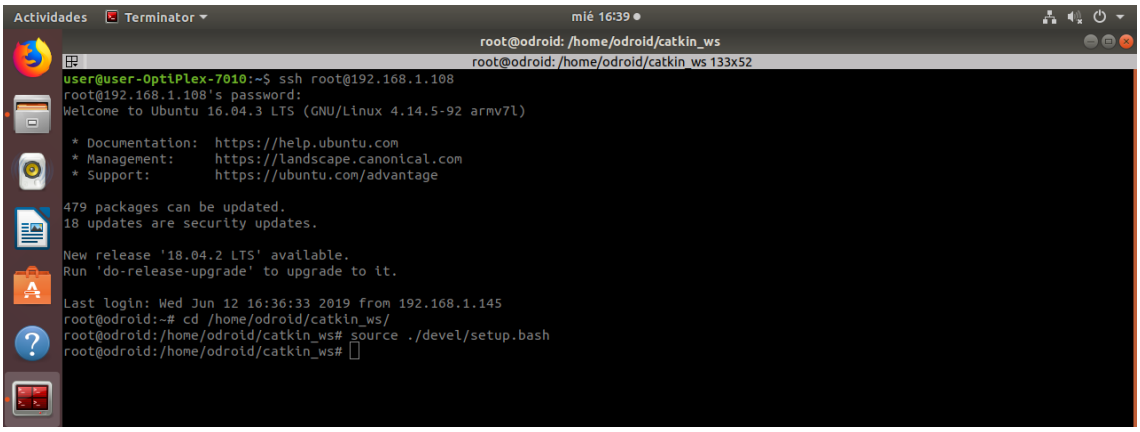
En primer lugar, suponiendo que el robot móvil Kobuki tiene batería suficiente, hay que proceder a encenderlo. El botón de encendido se encuentra donde está señalado en la figura adjunta.

Habiendose encendido el led que informa del estado del robot, es necesario preparar el ordenador. Para ello, al encenderlo hay que entrar en la distribución Ubuntu instalada. Una vez ahí, se debe acceder a la ventana de comandos. Esto puede hacerse directamente pulsando *Control+Alt+T* pero, sin embargo, como se va a trabajar con varias ventanas de comandos abiertas simultáneamente, se recomienda buscar la aplicación Terminator en el ordeandor y ejecutar los comandos desde ahí.

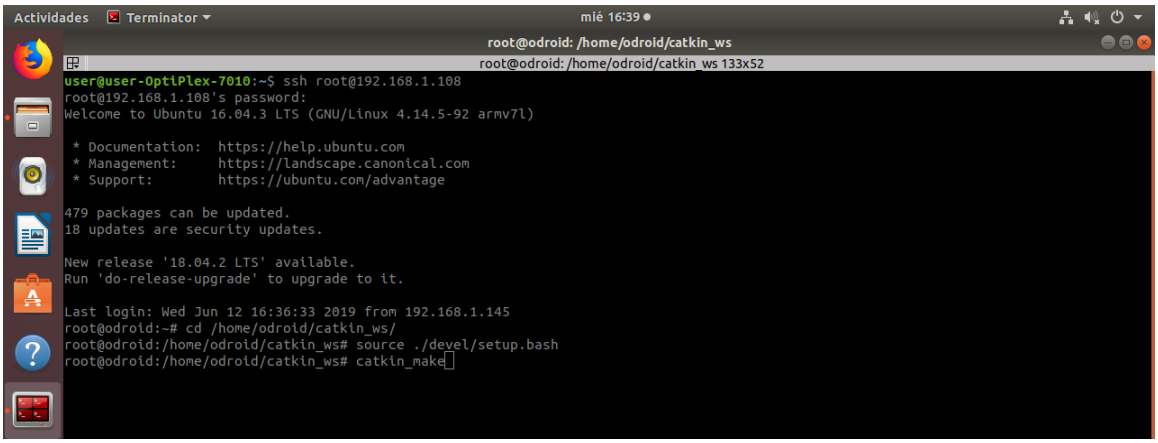
Una vez aquí, lo primero de todo es establecer la comunicación con la placa Odroid. Si el ordenador no admite una conexión inálambrica directa, se debe conectar al puerto ssh un dispositivo para realizarla. Se va a acceder al procesador como usuario root, tecleando lo siguiente: *ssh root@IP*. Es necesario conocer la IP de la placa para poder realizar la conexión. Como se le ha asignado una IP fija, en el robot hay una pegatina donde está escrita. En el caso de usar el que se ha montado para este trabajo, habría que ejecutar lo siguiente:



En este caso, como la IP era 192.168.1.108, el comando ejecutado ha sido ssh *root@192.168.1.108*. Una vez realizado esto y escrita la contraseña de acceso, hay que acceder a la carpeta de trabajo. Para ello, en la placa odroid se encuentra tecleando *cd /home/odroid/catkin_ws*. A fin de que funcione todo correctamente, es necesario ejecutar el siguiente comando una vez ubicados en el espacio de trabajo (catkin_ws): *source ./devel/setup.bash*. En la ventana de comandos se vería lo siguiente:



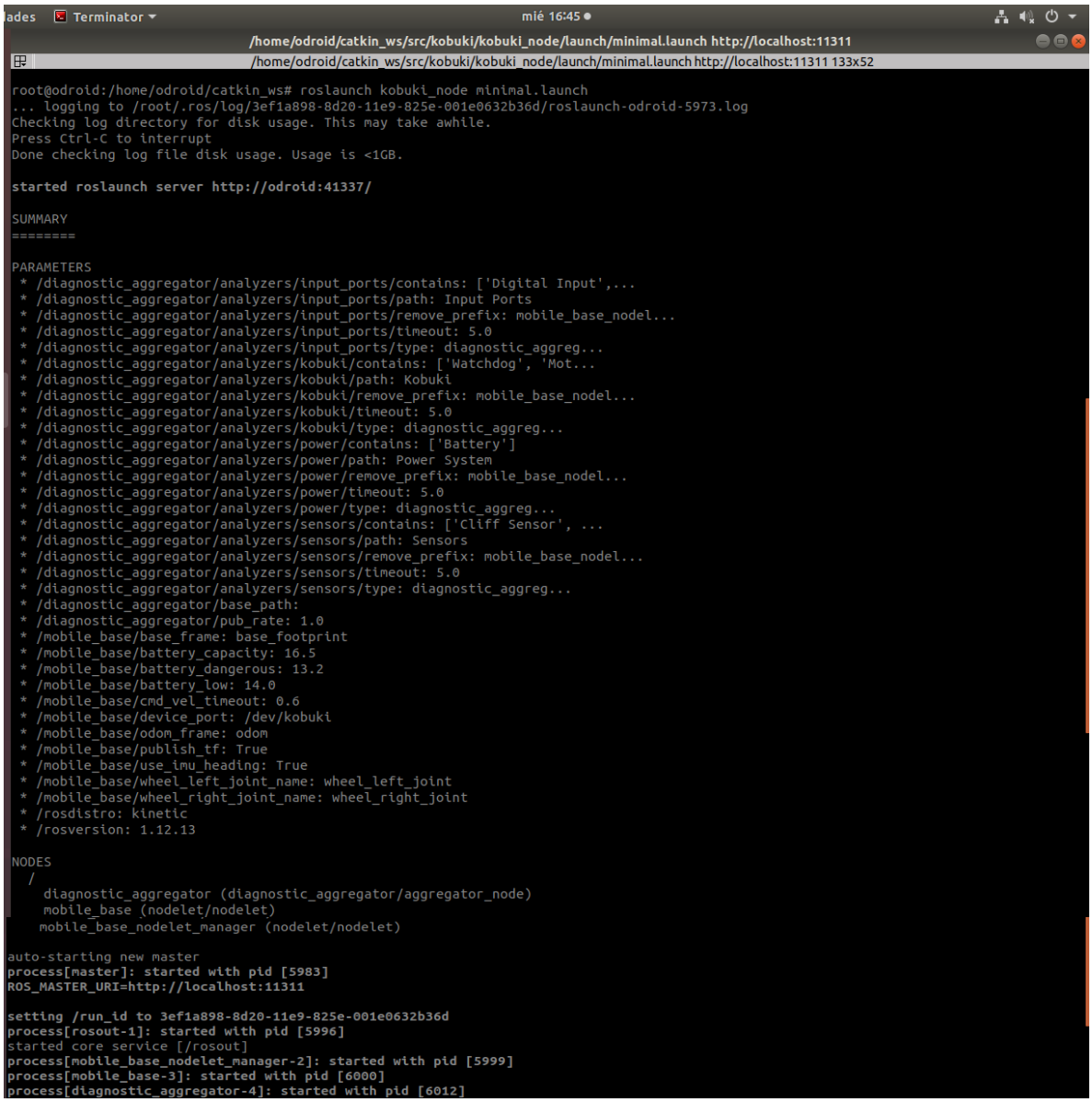
El siguiente paso es compilar el catkin workspace. Para ello basta con escribir *catkin_make*.



A partir de aquí, ya está todo preparado para ejecutar los controladores y los programas necesarios. Es importante destacar que el proceso anterior debe seguirse cada vez que se abre una ventana nueva.

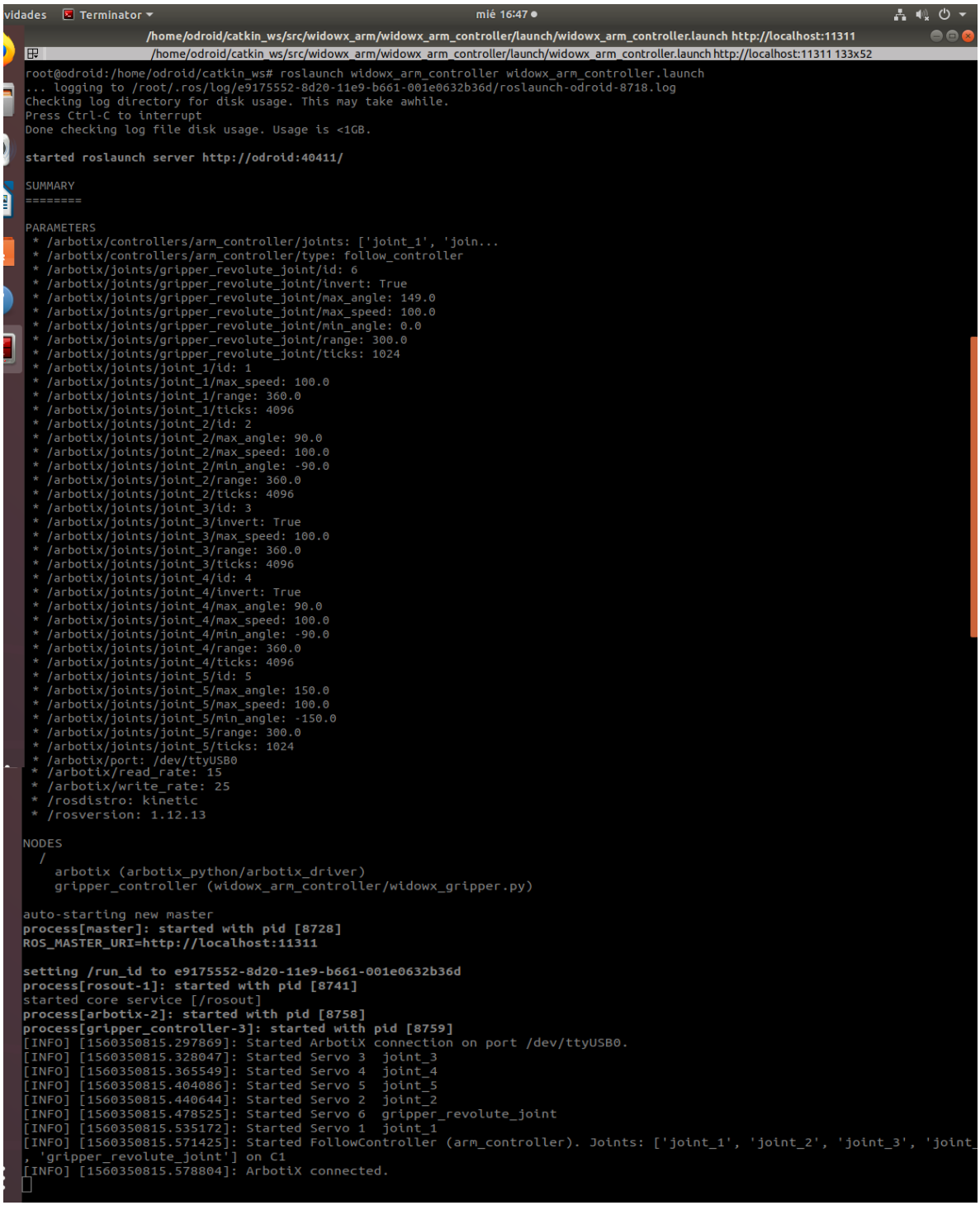
En primer lugar, se va a lanzar el controlador del robot Kobuki y con él sus nodos. Cabe destacar que cuando se lance este controlador, se arrancará el nodo que evalúa la posición del robot en todo momento. Por ello, para lanzar este maestro hace falta colocar la base

móvil en el lugar de partida de la aplicación y con la orientación correcta. El comando a ejecutar sería *roslaunch kobuki_node minimal.launch.* En la ventana de comandos se ve lo siguiente si todo está correcto:



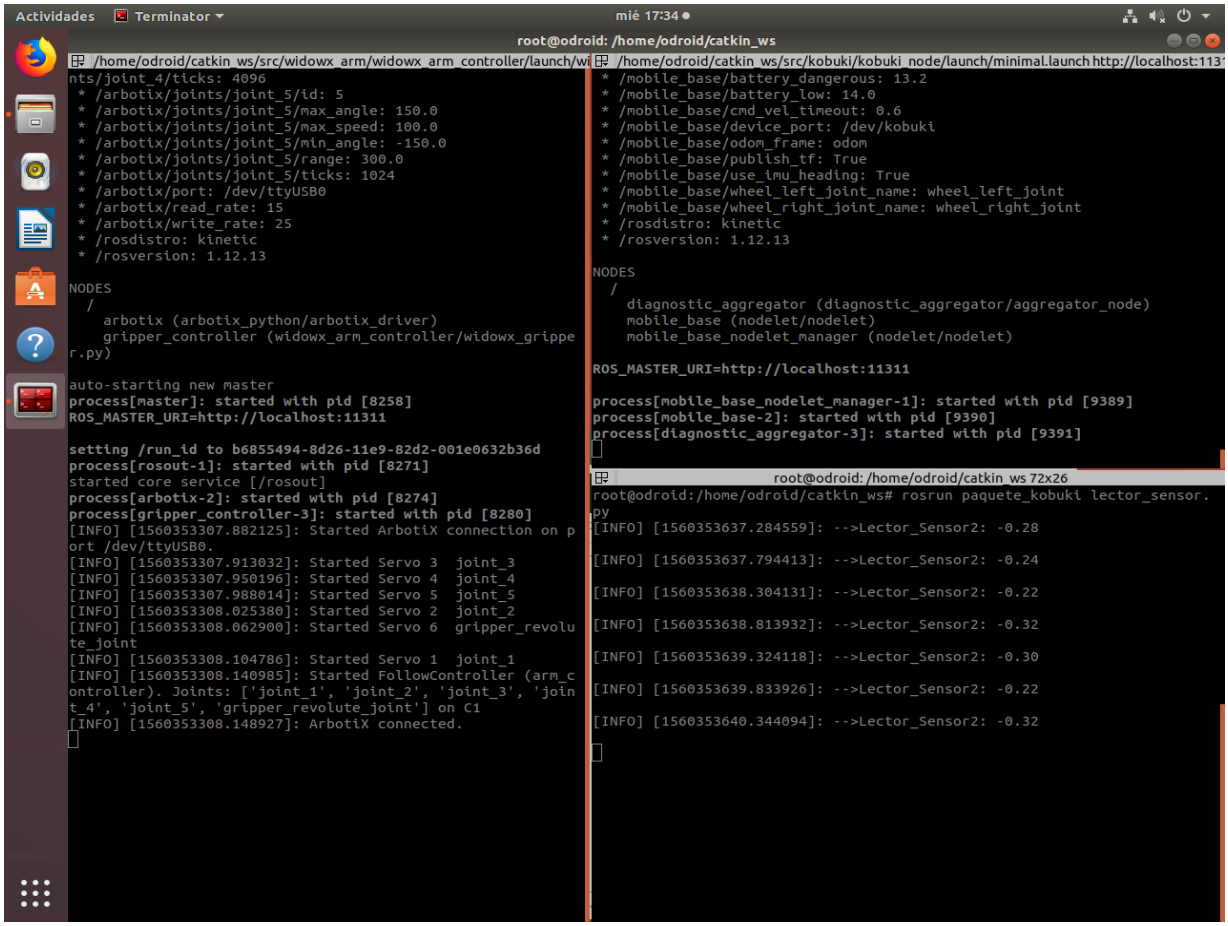
El siguiente paso es arrancar el controlador del brazo robótico. Esto se debe realizar en otra ventana y para ello, basta con hacer click derecho sobre la actual y seleccionar la opción dividir horizontalmente o verticalmente. En esta pantalla hay que repetir todo el proceso inicial, hasta tener compilado el workspace. Antes de lanzar ningún comando, hay que asegurarse de que la alimentación del brazo es correcta. Para ello, basta con observar que en la placa Arbotix situada en la base, se ilumina un led verde. Después de esto, el controlador del brazo se lanza escribiendo *roslaunch widowx_arm_controller*

widowx_arm_controller.launch. El resultado obtenido en la ventana de comandos debe ser tal que así:



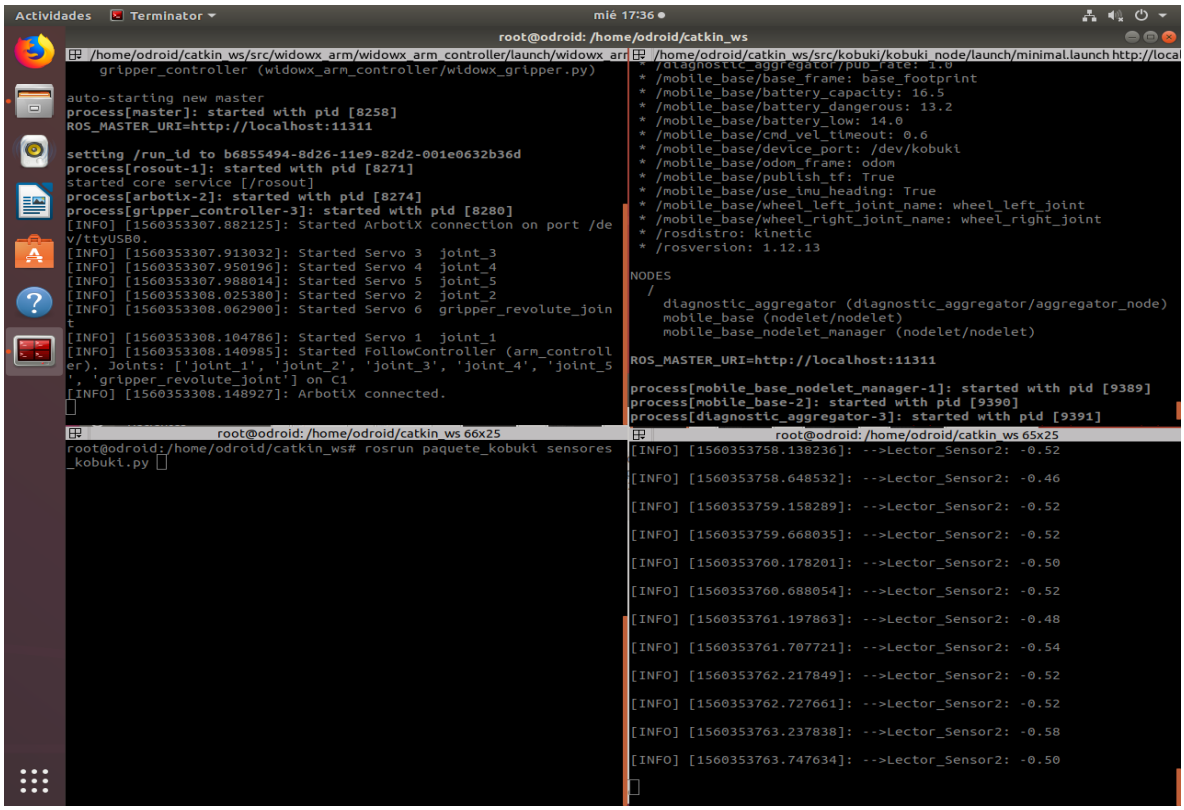
Para poder realizar la evaluación de la temperatura de la pieza es necesario arrancar el nodo que gestiona dicha información. Para ello, abriendo otra ventana y en ella realizando la configuración inicial hasta compilar el workspace, hay que ejecutar el comando *rosrun*

paquete_kobuki lecotr_sensor.py. Debe devolver como resultado la diferencia de temperatura entre el objeto que tiene en el ángulo de visión y el ambiente, como se muestra en la siguiente imagen:



Si el resultado no es el indicado, hay que asegurarse de que las conexiones de los XBees están bien realizadas. Para ello, se debe comprobar que tanto los leds colocados en el emisor como en el receptor están iluminados.

El último paso, sería abrir una ventana nueva, hacerle la configuración inicial y en ella ejecutar el programa que combina movimiento de la base, brazo y lectura de datos del sensor. El comando a ejecutar sería *rosrun paquete_kobuki sensores_kobuki.py*. Después de esto, el brazo robótico debería comenzar a moverse para comenzar con la evaluación de la temperatura. La ejecución será correcta si sobre la pantalla aparecen las 4 terminales abiertas de la manera que se expone:


Actividades
Terminator
mié 17:36
root@odroid: /home/odroid/catkin_ws
/home/odroid/catkin_ws/src/widowx_arm/widowx_arm_controller/launch/widowx_arm
gripper_controller (widowx_arm_controller/widowx_gripper.py)
auto-starting new master
process[master]: started with pid [8258]
ROS_MASTER_URI=http://localhost:11311
setting /run_id to b6855494-8d26-11e9-82d2-001e0632b36d
process[rosout-1]: started with pid [8271]
started core service [/rosout]
process[arbotix-2]: started with pid [8274]
process[gripper_controller-3]: started with pid [8280]
[INFO] [1560353307.882125]: Started ArbotiX connection on port /dev/ttyUSB0.
[INFO] [1560353307.913032]: Started Servo 3 joint_3
[INFO] [1560353307.950196]: Started Servo 4 joint_4
[INFO] [1560353307.988014]: Started Servo 5 joint_5
[INFO] [1560353308.025380]: Started Servo 2 joint_2
[INFO] [1560353308.062900]: Started Servo 6 gripper_revolute_joint
[INFO] [1560353308.104786]: Started Servo 1 joint_1
[INFO] [1560353308.140985]: Started FollowController (arm_controller). Joints: ['joint_1', 'joint_2', 'joint_3', 'joint_4', 'joint_5', 'gripper_revolute_joint'] on C1
[INFO] [1560353308.148927]: ArbotiX connected.
/home/odroid/catkin_ws/src/kobuki/kobuki_node/launch/minimal.launch http://local
* /diagnostic_aggregator/pub_rate: 1.0
* /mobile_base/base_frame: base_footprint
* /mobile_base/battery_capacity: 16.5
* /mobile_base/battery_dangerous: 13.2
* /mobile_base/battery_low: 14.0
* /mobile_base/cmd_vel_timeout: 0.6
* /mobile_base/device_port: /dev/kobuki
* /mobile_base/odom_frame: odom
* /mobile_base/publish_tf: True
* /mobile_base/use_imu_heading: True
* /mobile_base/wheel_left_joint_name: wheel_left_joint
* /mobile_base/wheel_right_joint_name: wheel_right_joint
* /rosdistro: kinetic
* /rosversion: 1.12.13
NODES
/
diagnostic_aggregator (diagnostic_aggregator/aggregator_node)
mobile_base (nodelet/nodelet)
mobile_base_nodelet_manager (nodelet/nodelet)
ROS_MASTER_URI=http://localhost:11311
process[mobile_base_nodelet_manager-1]: started with pid [9389]
process[mobile_base-2]: started with pid [9390]
process[diagnostic_aggregator-3]: started with pid [9391]
root@odroid: /home/odroid/catkin_ws 66x25
root@odroid:/home/odroid/catkin_ws# rosrun paquete_kobuki sensores_kobuki.py
root@odroid: /home/odroid/catkin_ws 65x25
[INFO] [1560353758.138236]: -->Lector_Sensor2: -0.52
[INFO] [1560353758.648532]: -->Lector_Sensor2: -0.46
[INFO] [1560353759.158289]: -->Lector_Sensor2: -0.52
[INFO] [1560353759.668035]: -->Lector_Sensor2: -0.52
[INFO] [1560353760.178201]: -->Lector_Sensor2: -0.50
[INFO] [1560353760.688054]: -->Lector_Sensor2: -0.52
[INFO] [1560353761.197863]: -->Lector_Sensor2: -0.48
[INFO] [1560353761.707721]: -->Lector_Sensor2: -0.54
[INFO] [1560353762.217849]: -->Lector_Sensor2: -0.52
[INFO] [1560353762.727661]: -->Lector_Sensor2: -0.52
[INFO] [1560353763.237838]: -->Lector_Sensor2: -0.58
[INFO] [1560353763.747634]: -->Lector_Sensor2: -0.50