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 UN BRAZO MANIPULADOR DE 5 EJES DE LIBERTAD SOBRE UNA PLATAFORMA ROBOTINO

Alumno/Alumna: González Jaio, Maialen>

Director/Directora (1): Casquero Oyarzabal, Oskar

Director/Directora (2): Orive Revillas, Darío

Curso: 2018-2019

Fecha: jueves, 27 de junio de 2019

RESUMEN

A lo largo de este Trabajo Fin de Grado se realiza el diseño y desarrollo de bajo nivel para el control de la plataforma móvil Robotino y del brazo manipulador WidowX que va acoplado a él. Se analizan los equipos utilizados desde los puntos de vista hardware y software para luego programar aplicaciones en el framework robótico ROS (Robot Operating System) en lenguaje Python. Con este proyecto se pretende ofertar servicios robóticos básicos de transporte y manipulación en el contexto de un sistema de fabricación flexible.

Palabras clave: Robotino, plataforma móvil, brazo manipulador, WidowX, ROS

ABSTRACT

Thorough the completion of this Final Degree Project the low-level design of the Robotino mobile platform and the WidowX robotic arm attached to it is developed in order to control them. The used equipment will be analysed from the hardware and software point of views to later program applications in ROS (Robot Operating System) using the Python language. With this project, basic robotic services are meant to be offered regarding transport and manipulation in the context of a flexible fabrication system.

Keywords: Robotino, mobile platform, robotic arm, WidowX, ROS

LABURPENA

Gradu Amaierako Lan honetan zehar Robotino plataforma mugikorra eta harengan akoplatuta dagoen WidowX beso manipulatzailea kontrolatzeko maila bajuko diseinua garatuko da. Erabilitako ekipamendua hardwareko eta softwareko ikuspuntuetatik analizatuko dira, geroago ROS (Robot Operating System) sistema eragilean aplikazioak programatzeko Python hizkuntza erabilita. Proiektu honekin, oinarrizko zerbitzu robotikoak eskaini nahi dira garraio eta manipulaziorako fabrikazio malguko sistemetako testuinguruan.

Gako-hitzak: Robotino, plataforma mugikorra, beso manipulatzailea, WidowX, ROS

Índice de Contenido

Índice de Figuras

Índice de Tablas

1. Introducción

Con la adopción de tecnologías emergentes en el campo de la automatización, la industria convencional que se conocía hasta hace unas décadas ha sufrido cambios drásticos. Esto ha concluido en lo que actualmente se conoce como la cuarta revolución industrial o Industria 4.0. En ésta se adoptan y adaptan, principalmente, tecnologías como big data, internet de las cosas, redes de sensores inteligentes, conjuntos de recursos compartidos de procesamiento reconfigurables (cloud computing), la impresión 3D, los robots autónomos, el gemelo y la puesta en marcha virtual (digital twin y virtual commisioning) [1].

Asimismo, los sistemas de fabricación también han evolucionado desde la producción en masa hacia una fabricación personalizada que se ajusta a todos los requerimientos y preferencias de los clientes [2]. Esto ha aumentado la competitividad entre los sistemas productivos incluyendo factores y características que antes no se valoraban, entre ellos la flexibilidad, la calidad, la adaptabilidad y la rapidez de respuesta [3].

En consecuencia, la Industria 4.0 y la búsqueda de la fabricación personalizada han hecho que los sistemas de fabricación evolucionen desde las distribuciones lineales (Flow-Shop) y funcionales (Job-Shop), a la distribución flexible [4], entendiendo por sistema de fabricación flexible un conjunto de estaciones de trabajo conectadas por un sistema de transporte automatizado basado en robots de transporte. La característica más importante de estos sistemas es que todos los equipos están completamente automatizados por lo que su control se puede realizar de forma distribuida, dotando de inteligencia a la fábrica.

2. Contexto

La justificación de la utilización de robots de transporte se basa en su flexibilidad frente a la cinta de transporte tradicional alrededor de la cual se deben situar las estaciones de trabajo secuencialmente. En este sentido, el uso de robots de transporte permite disponer y hacer uso de las estaciones de trabajo de forma más eficiente posible de acuerdo con las condiciones de espacio y de trabajo (fallos de máquina y entrada de nuevos pedidos)

Los robots móviles aumentan la flexibilidad de un sistema gracias a sus posibilidades de movimiento y la variabilidad atendiendo a diferentes circunstancias. De esta forma, son capaces de satisfacer diferentes requerimientos que un cliente puede tener y no parar la línea de producción en caso de que alguna máquina se averíe. Además, optimizan significativamente los tiempos no productivos, permitiendo así la reducción de los lotes de fabricación que hoy en día tienen lugar como consecuencia de una producción cada vez más personalizada

Por lo tanto, los robots móviles juegan un papel fundamental en la constitución de un sistema de fabricación flexible. Dichos robots pueden ir equipados tanto con brazos manipuladores, como con cámaras u otros tipos de herramienta que se adecúen a las necesidades de la planta. Por otro lado, hace falta un sistema de colaboración entre los distintos robots para que sean capaces de responder ante cambios en la producción y decidir cuál de ellos es el más adecuado para hacerles frente.

Se debe tener en cuenta, además, que el ámbito de la robótica requiere de conocimientos de distintas áreas como lo pueden ser la automática, el control, la informática, etc. Por lo tanto, para la correcta integración de este tipo de sistemas a gran escala se debe partir del conocimiento del diseño de nivel bajo de ellos. Es aquí donde se sitúa este Trabajo Fin de Grado (TFG), ya que utilizando un framework robótico como ROS, se pretende controlar una plataforma móvil junto con un brazo manipulador.

Concretamente, este TFG forma parte de otro proyecto, realizado con anterioridad [5], 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, este proyecto tiene como objetivo dotar de funcionalidad a dichos servicios robóticos mediante el diseño y desarrollo de nodos funcionales en ROS que controles diferentes partes de robot a bajo nivel.

3. Objetivos

El objetivo principal de este proyecto es el control de una plataforma móvil Robotino, que tiene acoplado un brazo manipulador, a través de la programación a bajo nivel de nodos funcionales en ROS.

Para la consecución de este objetivo principal, será necesario completar los siguientes objetivos secundarios:

1. Conocimiento a nivel de usuario de la distribución Linux Ubuntu y sus comandos básicos.
2. Análisis de la arquitectura hardware y software de la plataforma móvil y del brazo manipulador.
3. Comprensión del diseño de alto nivel de framework robótico ROS.
4. Conocimiento del lenguaje de programación de alto nivel, Python para la programación a bajo nivel de nodos funcionales en ROS.
5. Identificación y análisis de los paquetes disponibles en ROS y los servicios que éstos ofrecen para el manejo de la plataforma móvil y el brazo manipulador.
6. Diseño de unos casos de uso que permitan identificar los requisitos funcionales.
7. Separación de intereses para dividir el programa final en secciones distintas, de forma que cada sección enfoca un interés delimitado, a saber: movimientos del brazo manipulador y movimientos de la plataforma móvil.
8. Integración de los programas correspondientes al brazo manipulador y la plataforma móvil para realizar un movimiento conjunto.

4. Beneficios que aporta el trabajo

La utilización de robots móviles en la industria actual proporciona grandes beneficios que, principalmente, se pueden clasificar en tres categorías, a saber: beneficios sociales, beneficios técnicos y beneficios económicos.

En este apartado analizaremos cada uno de esos beneficios.

4.1. Beneficios sociales

No es sencillo distinguir los beneficios sociales que podría acarrear la implementación de robots móviles en la industria, ya que se podría pensar que arrebatan el trabajo del personal. Sin embargo, aunque sustituyan a los trabajadores en ciertas tareas, especialmente en aquellas con baja complejidad y repetitivas, estas serán sustituidos por nuevos puestos de trabajo [6].

Por otro lado, la calidad de empleo será mayor, ya que los robots serán los encargados de realizar los trabajos más pesados, reduciendo así el riesgo de accidentes y lesiones del personal. En caso de que las tareas a realizar sean de alta peligrosidad, también podrían utilizarse robots para velar de esta forma por la seguridad de los trabajadores.

Por último, queda mencionar que año tras año los clientes exigen productos cada vez más personalizados y que se ajusten mejor a sus gustos. Mediante el uso de los robots la capacidad de personalización de los fabricantes aumenta y por lo tanto se tiene mayor capacidad para satisfacer las necesidades, cada vez con un mayor nivel de personalización, del consumidor.

4.2. Beneficios técnicos

Los beneficios técnicos en este caso esta relacionados con el hecho de que los robots de transporte se pueden utilizar como sustitutivo de la cinta transportadora en cadenas de ensamblaje o producción en serie. Esta posibilidad repercutiría en la cadena proporcionando una gran flexibilidad de trabajo. Esto se debe a que al introducir robots se elimina la cinta transportadora por lo que el área de trabajo gana posibilidades de movilidad.

Por lo tanto, el aumento de flexibilidad es un aspecto positivo que se traduce posteriormente en la posibilidad de hacer tiradas más pequeñas y un menor impacto de los fallos. Las tiradas de fabricación pueden ser más pequeñas debido al hecho de que, al no depender de la velocidad de la cinta transportadora, se puede adecuar el número de robots a las necesidades de fabricación requeridas. Por otro lado, el fallo de una máquina tendría un menor impacto, ya que, en caso de que ocurriera, se podría modificar la trayectoria de los robots para que se dirigiesen a otra máquina

que pudiese realizar las funciones de la máquina que ha fallado. Esto beneficiaría a la empresa ya que no tendría que parar la cadena de montaje en su totalidad, si no adaptarla.

4.3. Beneficios económicos

Cuando consideramos los beneficios económicos de incluir robots móviles en la industria cabe mencionar, en primer lugar, la reducción de los costes fijos de la empresa. Esto se debe a que los robots se pueden utilizar para realizar tareas monótonas o de alta repetitividad. Son capaces de realizar dichas tareas de forma continua sin reducir el rendimiento de ejecución.

Otro coste que también se ve reducido al implementar robots móviles es el coste de instalación. Esto es una consecuencia directa de la eliminación de la cinta transportadora. Al utilizar cintas todas las máquinas deben ir dispuestas de forma secuencial para que la cinta tenga acceso a todas ellas, pero al eliminarla las máquinas se pueden disponer de la forma más conveniente y económica. De esta forma las máquinas se adecuarían al espacio disponible y no al revés.

5. Descripción de requerimientos

En este proyecto se realizará el control de una plataforma móvil que tiene acoplada un brazo manipulador. Para ello, se detallan unos requerimientos y especificaciones con los que se deberá trabajar para completarlo. Se diferenciarán para ello dos tipos de requerimientos: requerimientos del sistema y requerimientos funcionales.

5.1. Requerimientos del sistema

Se trata de los requerimientos que los componentes del sistema deberán cumplir. En primer lugar, la plataforma móvil a utilizar será el robot denominado Robotino de la empresa Festo Didactic que está enfocado para su utilización en las áreas de investigación y formación. Asimismo, el robot deberá llevar acoplada una torre de montaje con las tres plataformas metálicas para la realización de trabajos en altura.

El framework robótico que se utilizará será ROS (Robot Operating System) que consiste en un entorno informático de propósito general para el desarrollo de programas para todo tipo de robots. Proporciona una base común para todos ellos, posibilitando así la combinación de robots de diferentes empresas.

La aplicación que se implementará junto a estas dos especificaciones también tiene que incluir el uso de un brazo manipulador con una pinza en uno de los extremos.

5.2. Requerimientos funcionales

En este apartado se describirán aquellos requisitos que definen los servicios que el sistema robótico (en este caso, el formado por la plataforma móvil y el brazo manipulador) debe proporcionar.

Para definir los servicios se ha descrito un caso de uso: El sistema parte de la situación de reposo y cuando se ejecuta el programa el brazo manipulador deberá ir a la posición de defecto. Una vez hecho esto la base del Robotino deberá realizar movimiento de avance y giro para aproximarse al punto de recogida del objeto. Cuando la plataforma haya llegado a su destino, se procederá a realizar la aproximación y recogida del objeto. Posteriormente, la base deberá realizar un movimiento diagonal, al final del cual el brazo deberá depositar el objeto recogido. Finalmente, el sistema regresará a la posición de partida.

Definido así el caso de uso, la Figura 5.1 muestra las acciones que debe realizar el sistema plataforma móvil y manipulador. En cambio, la Figura 5.2 muestra las características y elementos que se deberán tener en cuenta en la programación para realizar dichas acciones.

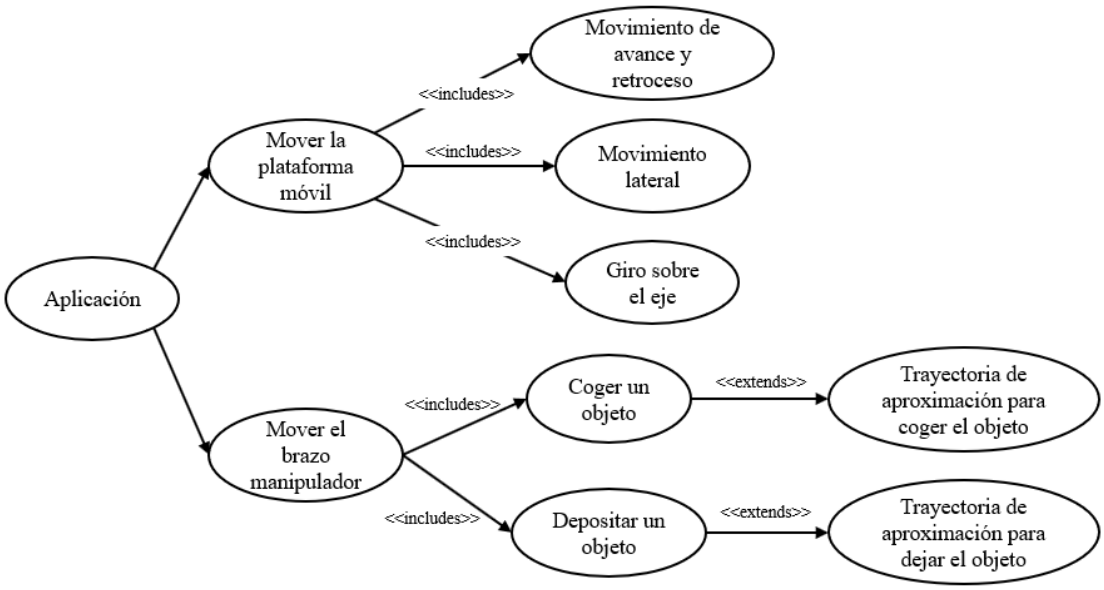

Figura 5.1 - Caso de uso: Movimientos que debe incluir la aplicación

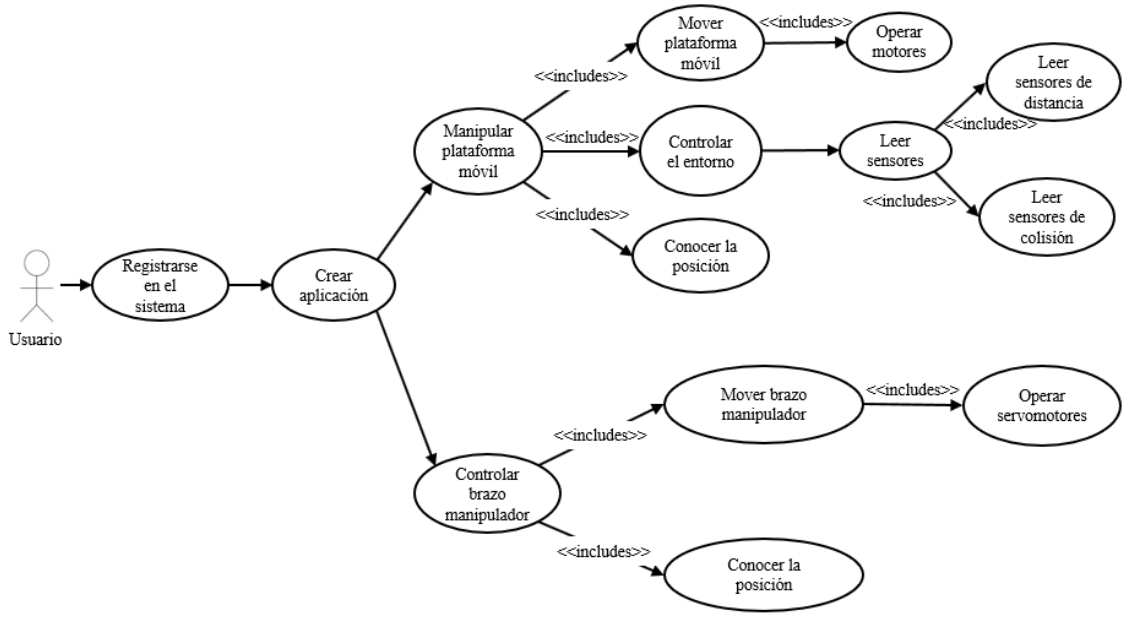

Figura 5.2 – Caso de uso: Elementos que debe incluir la aplicación

6. Análisis de alternativas

Debido a los avances en el sector de la robótica e informática, cada vez son mayores las posibilidades que se ofrecen a la hora de controlar un robot móvil o realizar aplicaciones.

En este sentido, una vez definidos y detallados los requerimientos del sistema, se presentan una serie de alternativas para llevar a cabo la ejecución del proyecto. A continuación, se detallará cada una de dichas alternativas y se terminará eligiendo la más adecuada de forma objetiva en base a una serie de criterios de selección. Los criterios escogidos deberán permitir alcanzar los objetivos propuestos de forma satisfactoria y, además, respetar los requerimientos impuestos.

Finalmente, para tomar la decisión final se utilizará una tabla de ponderaciones donde se incluirán las diferentes alternativas de las que se partía y los criterios de selección escogidos. Cada uno de estos criterios tendrá asignado un coeficiente de ponderación para poder distinguir los factores más importantes. Los coeficientes de ponderación consistirán en un número del 1 al 3, siendo el 1 el menos importante y el 3 el más importante. Asimismo, se calificará el grado de cumplimiento de cada uno de los criterios de acuerdo con la siguiente escala ordinal:

- **5:** Muy alto grado de cumplimiento (Muy bueno)
- **4:** Alto grado de cumplimiento (Bueno)
- **3:** Grado medio de cumplimiento (Medio)
- **2:** Bajo grado de cumplimiento (Malo)
- **1:** Muy bajo grado de cumplimiento (Muy malo)

6.1. Alternativas de brazo manipulador

Para la aplicación que se pide realizar con la plataforma móvil se requiere la utilización de un brazo manipulador. Dicho brazo deberá ser compatible con el resto de los requisitos lo que se reflejará a la hora de definir los criterios de selección.

En cuanto a los brazos manipuladores, teniendo en cuenta lo dicho anteriormente, se considerarán las siguientes alternativas:

Brazo prensor electrónico de Festo Didactic

El brazo que se ha considerado en primer lugar es de la empresa de Festo Didactic (ver Figura 6.1) que es la misma que se encarga de la fabricación del Robotino. El brazo consta de tres servomotores Dynamixel conectados en serie entre sí y que vienen configurados de fábrica. Estos están conectados a su vez a un microcontrolador propio del brazo. El controlador se conecta, por un lado, a la interfaz de E/S del Robotino y por otro, al

procesador mediante USB. La pinza está accionada por otro servomotor y para detectar la presencia de piezas está dotada de cuatro sensores ópticos.

Figura 6.1 – Brazo prensor electrónico de Festo Didactic

Este brazo permite la programación desde diferentes plataformas de diseño de sistemas como MATLAB o LabVIEW. El brazo también podría programarse para ser utilizado desde ROS, sin embargo, la autora de este proyecto, después de realizar una búsqueda exhaustiva, no ha encontrado ningún repositorio disponible para poder hacerlo.
En la Tabla 6.1 se muestran algunos de los parámetros más característicos de brazo electrónico prensor.

Tabla 6.1 – Parámetros característicos del brazo electrónico prensor de Festo Didactic

PARÁMETROS	VALOR
Capacidad de carga	Hasta 200 g
Carrera	30 – 60 mm
Posiciones de agarre	2
Alimentación	24 V DC

WidowX Robot Arm

El segundo brazo que se ha considerado es el brazo fabricado por la empresa Trossen Robotics (ver Figura 6.2). El brazo está formado por 5 servomotores de Dynamixel y otro servomotor encargado del movimiento de la pinza. Estos servomotores también están conectados en serie entre sí y a su vez se conectan a una placa con microcontrolador basada en Arduino de diseño propio [7]. La conexión entre la base y el brazo se realiza por USB.

Figura 6.2 – WidowX Robot Arm

La programación de este brazo puede realizarse mediante Arduino, ROS o MoveIt. La Tabla 6.2 muestra algunos de sus parámetros característicos.

Tabla 6.2 – Parámetros característicos del WidowX Robot Arm

PARÁMETROS	VALOR
Capacidad de carga	Hasta 500 g
Carrera horizontal	37 – 41 cm
Carrera vertical	51 – 55 cm
Alimentación	12 V DC

Para elegir el brazo a utilizar en este proyecto, se considerarán los siguientes criterios:

- **Compatibilidad con ROS:** Uno de los requerimientos impuestos es que la solución final haga uso del framework robótico ROS. Este aspecto hace, de este criterio el más importante a la hora de guiar la elección del brazo.
- **Capacidad de carga:** Este criterio tendrá en cuenta la capacidad de carga de la pinza, ya que ésta limitará las características de los objetos que se podrán manipular en las aplicaciones que se realicen.
- **Grados de libertad:** La valoración de este criterio se traduce en que cuanto mayor sea el número de servomotores de los que disponga el brazo mayor será su libertad de movimiento.
- **Carrera:** La importancia de este criterio está relacionada con la capacidad de llegar a coger objetos que estén a una determinada distancia de la base del brazo.
- **Sensores ópticos:** Este criterio tiene su importancia en relación a la capacidad de detectar objetos delante de la pinza. Esto está directamente relacionado con la posibilidad de plantear diferentes alternativas de programación y proporcionar robustez al programa.

Antes de realizar la tabla de decisión final, a modo de resumen se realizará una breve descripción, Tabla 6.3, de cada uno de los brazos en base a los criterios seleccionados.

Tabla 6.3 – Características de las alternativas de los brazos manipuladores según los criterios de selección

	Compatibilidad con ROS	**Capacidad de carga**	**Número de servomotores**	**Carrera**	**Sensores ópticos**
BRAZO PRENSOR ELECTRÓNICO	No	Hasta 200 g	3	30-60mm	Sí
WIDOWX ROBOT ARM	Sí	Hasta 500 g	5	37-41cm horizontal 51-55cm vertical	No

A continuación, una vez hecha la descripción de cada uno de los criterios de los brazos se completará la tabla de decisión final, Tabla 6.4.

Tabla 6.4 – Tabla de decisión de las alternativas de brazo manipulador

	Compatibilidad con ROS	**Capacidad de carga**	**Número de servomotores**	**Carrera**	**Sensores ópticos**	**TOTAL**
Ponderación	**3**	**1**	**2**	**2**	**1**	
BRAZO PRENSOR ELECTRÓNICO	1	2	3	2	5	**20**
WIDOWX ROBOT ARM	5	4	4	4	1	**36**

Una vez rellenada la tabla de decisión, observamos como el WidowX Robot Arm sería el brazo más adecuado para su utilización en el robot móvil. Esto es coherente con lo ya mencionado anteriormente en el ámbito de la compatibilidad con ROS, ya que este brazo es el único que cumple dicho requerimiento.

6.2. Alternativas de distribución de ROS

Entre los requerimientos solicitados consta el uso del framework ROS para el control y manipulación tanto de la plataforma móvil como del brazo. Sin embargo, es necesario realizar un análisis de alternativas entre las diferentes versiones de ROS, ya que éste es un framework robótico que actualiza frecuentemente (concretamente, anualmente) y puede haber cambios significativos entre versiones.

A continuación, se detallarán las características de las versiones que se considerarán en este proyecto:

ROS Kinetic

Aunque esta versión fue lanzada en 2017, se trata de una versión LTS (Long Term Support) que se alarga hasta el 2021. Este amplio periodo de soporte hace que sea la versión con mayor comunidad y, por tanto, mayo disponibilidad de paquetes y ayuda. Dispone de programas como MoveIt y Rviz para la manipulación de robots.

Para la instalación de Kinetic es necesario trabajar en Ubuntu 16.04 y no es posible hacerlo desde Ubuntu 18.04. Esto no presentaría ningún problema debido al hecho de que Robotino tiene instalada la versión Ubuntu 16.04.

ROS Lunar

Se trata de la versión lanzada en 2017 y de la que ROS realiza el soporte hasta 2019, lo que significa que a partir de este mismo año no habrá ninguna actualización de la versión ni se proporcionará ayuda desde la base de desarrolladores de ROS. Debido a este corto periodo de soporte la comunidad de ROS es reticente a la hora de usarlo.

Esta versión de ROS debe ser instalada sobre el sistema operativo de Ubuntu 16.04.

ROS Melodic

Es la última versión lanzada de ROS. Esto hace que sea la plataforma más robusta de las tres, sin embargo, al ser una versión muy reciente no dispone de los recursos de ROS Kinetic ya que la comunidad que la respalda todavía está en crecimiento. Su soporte se realizará hasta 2023.

Otra desventaja a tener en cuenta es que debido a su reciente lanzamiento las versiones de MoveIt y Rviz para el control de robots no han sido lanzadas. Además, para su instalación es necesaria la utilización de Ubuntu 18.04.

Una vez descritas las últimas versiones, se valorarán los siguientes criterios de selección para realizar la decisión final:

- **Soporte:** Este criterio es de fundamental importancia ya que, a mayor tiempo de soporte, más actualizaciones de la distribución habrá para solucionar los problemas que surjan.

- **Compatibilidad con Ubuntu 16.04:** Debido a que Robotino viene programado por defecto con Ubuntu 16.04, tendrá una gran importancia que la distribución escogida sea compatible. Esto reducirá tiempos de instalación y actualización.
- **Comunidad:** Cuanto mayor sea la comunidad se podrá recibir mayor ayuda y soluciones a problemas que aparezcan a lo largo del proyecto.
- **Paquetes:** Este criterio obtiene su importancia de la disponibilidad de aplicaciones y ejemplos para las distintas distribuciones.

En la Tabla 6.5, se realiza un breve resumen de las características de cada distribución para los criterios de selección valorados.

Tabla 6.5 – Características de las distribuciones de ROS según los criterios de selección

	Soporte	**Compatibilidad con Ubuntu 16.04**	**Comunidad**	**Paquetes**
ROS KINETIC	Hasta 2021	Sí	Amplia	Muchos
ROS LUNAR	Hasta 2019	Sí	Reducida	Pocos
ROS MELODIC	Hasta 2023	No	En crecimiento	Pocos

Por último, en la Tabla 6.6 se muestra la tabla de decisiones realizada considerado los aspectos mencionados.

Tabla 6.6 – Tabla de decisión de las alternativas de distribución de ROS

	Soporte	**Compatibilidad con Ubuntu 16.04**	**Comunidad**	**Paquetes**	**TOTAL**
Ponderación	**2**	**3**	**2**	**1**	
ROS KINETIC	4	5	4	3	**34**
ROS LUNAR	2	5	2	2	**25**
ROS MELODIC	5	1	2	2	**19**

De acuerdo con los resultados obtenidos, la decisión más adecuada es la utilización de ROS Kinetic para este proyecto. El soporte y la compatibilidad con Ubuntu 16.04 se observa que son los factores decisivos en esta decisión.

6.3. Alternativas de lenguaje de programación

El control del robot móvil y el brazo, como ya se ha dicho debe ir implementado en el framework robótico ROS. Sin embargo, dentro de este framework robótico se pueden realizar programas en diferentes lenguajes de programación. A la hora de valorar alternativas de lenguaje, sólo se valorarán lenguajes de alto nivel que permitan reutilizar el mismo código en diferentes robots y utilizar librerías existentes. Es por esta razón por lo que no se ha escogido el lenguaje RobotinoView que solo sería útil para la programación de la plataforma móvil Robotino [8].

A continuación, se describirán cada una de las alternativas escogidas por separado para poder analizar los beneficios que proporciona cada una de ellas y así determinar cuál es la mejor opción.

Python

Python es un lenguaje de programación que fue desarrollado con el objetivo de proporcionar un aprendizaje sencillo y una lectura fácil. Una de sus principales características es la capacidad de desarrollar programas de prototipado de forma rápida.

Presenta una interfaz sencilla ya sea tanto para programadores como no-programadores y muestra el programa de forma explícita. Además, tiene una amplia variedad de librerías que se adaptan a las necesidades de uso requeridas con una gran comunidad que la respalda.

Sin embargo, una gran desventaja de Python es que no tiene una gran robustez, por lo que no se utiliza para proyectos en producción.

C++

C++ es uno de los lenguajes programación más utilizados principalmente debido a su robustez, lo cual lo hace muy apropiado para el despliegue de soluciones en producción. Entre sus puntos fuertes podemos encontrará la gran estabilidad de ejecución y la rapidez. Además, es un lenguaje ampliamente utilizado en la industria, por lo que tiene acceso a una gran variedad de librerías.

Por otro lado, una de las principales desventajas de este lenguaje es su complejidad tanto en programación como en aprendizaje. Como consecuencia, programar correctamente utilizando este lenguaje lleva mucho tiempo y horas de aprendizaje.

MATLAB

MATLAB es una plataforma de diseño de sistemas que ofrece distintos tipos de interfaces para su programación entre los que destacan CLI (Command Line Interface), es decir, scripts y GUI (Graphical User Interface), es decir, Simulink. Se trata de un instrumento que se distingue especialmente a la hora de analizar los datos recogidos.

Sin embargo, MATLAB presenta problemas de portabilidad de programas al trabajar con diferentes sistemas.

LabVIEW

Se trata de una plataforma de diseño cuyo objetivo consiste en visualizar cada uno de los elementos que forman parte de un sistema. Este programa controla y manipula el robot a través de bloques funcionales utilizando el lenguaje Graph7, por lo que tiene una interfaz sencilla.

Aunque tiene una sintaxis sencilla de comprender, requiere de tiempo a la hora de depurar el código, por lo que el desarrollo de prototipos no es muy ágil. Además, es un software privativo, es decir, no proporciona acceso a su código fuente y de pago, lo que dificulta la colaboración con otros usuarios.

Los criterios de selección en los que se basará la elección final de lenguaje de programación serán los siguientes:

- **Tiempo de desarrollo:** Este criterio tendrá una gran importancia dado el carácter de educativo del proyecto en desarrollo y el corto plazo en el que se debe desarrollar (el TFG es un proyecto de 6 créditos, es decir, 150 horas). A la hora de proyectar y desarrollar una idea se debe realizar de forma rápida para así tener tiempo de analizar el mayor número de funciones posible.
- **Robustez:** La importancia de este factor está relacionado con dotar de la mayor estabilidad posible al sistema para no tener problemas en la ejecución y asegurar un comportamiento especifico en sucesivas pruebas.
- **Tiempo de aprendizaje:** Por limitaciones de tiempo se deberá aprender a utilizar el programa con la mayor brevedad posible.
- **Sintaxis:** Mediante este criterio se valorará la facilidad de lectura y comprensión del código para tanto programadores como no programadores.

Teniendo en cuenta los criterios escogidos se realizará una tabla descriptiva, Tabla 6.7, de las diferentes alternativas.

Tabla 6.7 – Características de las alternativas de lenguaje de programación según los criterios de selección

	Tiempo de desarrollo	**Robustez**	**Tiempo de aprendizaje**	**Sintaxis**
PYTHON	Bajo	Media	Bajo	Sencilla
C++	Alto	Alta	Alto	Compleja
MATLAB	Medio	Alta	Medio	Sencilla
LABVIEW	Alto	Media	Medio	Muy sencilla

Considerando los parámetros que relacionan cada una de las alternativas con los criterios de selección se realizará la tabla de decisión final, Tabla 6.8, considerando su grado de cumplimiento.

Tabla 6.8 – Tabla de decisión de las alternativas de lenguaje de programación

	Tiempo de desarrollo	**Robustez**	**Tiempo de aprendizaje**	**Sintaxis**	**TOTAL**
Ponderación	**3**	**2**	**2**	**1**	
PYTHON	4	2	5	4	**30**
C++	1	4	2	2	**17**
MATLAB	3	4	3	3	**23**
LABVIEW	2	3	3	5	**23**

En base a los resultados obtenidos en la tabla de decisión final se concluye que el lenguaje de programación más adecuado para utilizar en este proyecto es Python. Esta elección está principalmente condicionada tal y como se observa por su rapidez en el tiempo de desarrollo y aprendizaje.

7. Descripción de la solución propuesta

Para cumplir los objetivos propuestos para este proyecto, se han considerado en primer lugar los requerimientos y especificaciones impuestos. Para los aspectos que se quedaban sin definir se ha realizado el análisis de alternativas y se han tomado las decisiones objetivas para escoger la más adecuada entre ellas. El objetivo de este apartado es detallar y describir cómo se integran los requerimientos y las especificaciones con las alternativas seleccionadas. En este sentido, la Figura 7.1 muestra el diagrama de bloques con las entidades hardware de las que consta el proyecto y las relaciones entre ellas.

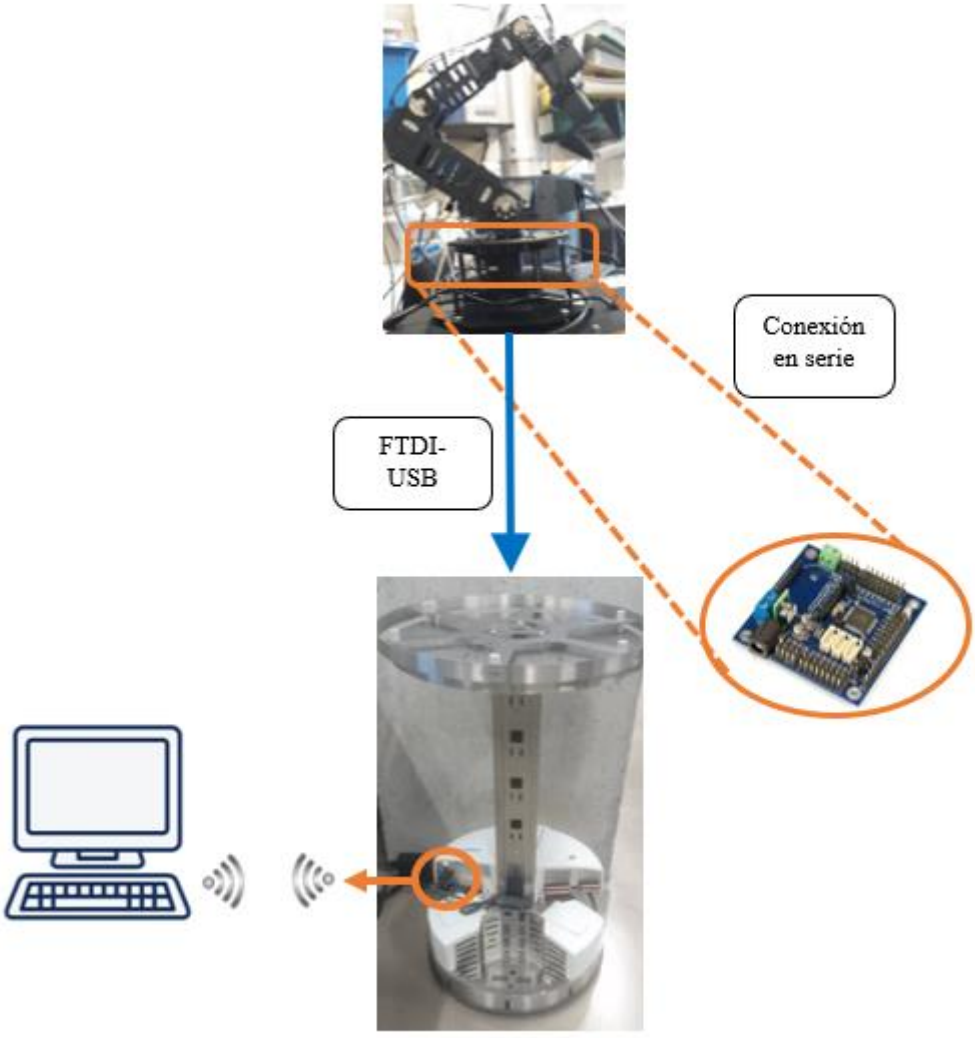

Figura 7.1 – Elementos y conexiones de la solución propuesta

Tal y como se observa en la Figura 7.1 se distinguen cuatro equipos principales: el ordenador, Robotino, la placa controladora Arbotix-M Robocontroller y el brazo manipulador. En el centro de esta configuración está Robotino ya que la CPU que va instala en el será la encargada de

manipular todas las entradas y salidas del Robotino como del brazo. Para acceder a dicha CPU se utilizará una conexión WLAN entre el Robotino y un ordenador externo.

Por otro lado, la conexión entre el Robotino y el brazo pasará por el controlador Arbotix-M Robocontroller. La conexión entre el controlador y el Robotino se hace mediante un cable FTDI-USB. Este cable atravesará la torre central del Robotino hasta conectar el USB a uno de los 6 puertos del Robotino.

La conexión ente el controlador y los servomotores del brazo se realiza vía serie mediante cables de 3 pines.

Si se analiza esta estructura desde el punto de vista del software (ver Figura 7.2), se observa que la distribución básica necesaria para el funcionamiento de todos los elementos es Ubuntu. Por defecto, Linux Ubuntu 16.04 LTS es la distribución que viene instalada en el Robotino. Para la versión de Ubuntu instalada, el análisis de alternativas realizado concluye que la mejor opción es la instalación de la distribución Kinetic de ROS.

En lo que respecta al controlador Arbotix-M Robocontroller, a través de Arduino se cargará en el un programa para interpretar los comandos enviados a través de ROS y enviarlos vía serie a todos los servomotores.

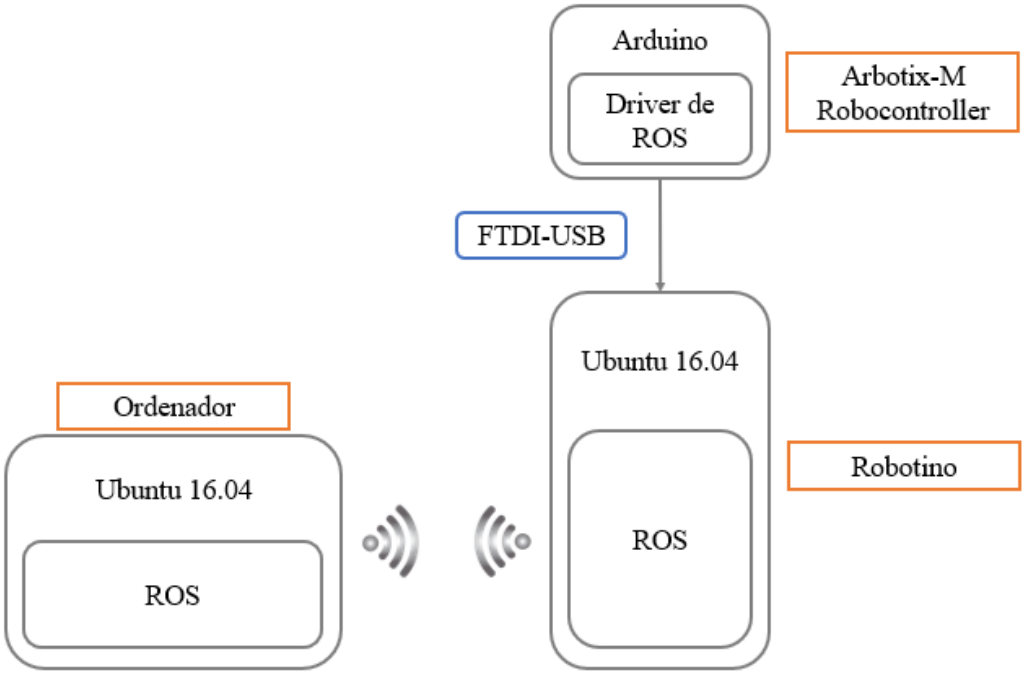

Figura 7.2 – Software de los elementos que forman la solución propuesta

8. Diseño

En este apartado se describirán detalladamente cada uno de los equipos con los que se ha trabajado y su diseño software tanta de alto como de bajo nivel.

8.1. Hardware

8.1.1. Hardware del Robotino

Robotino es un sistema robótico que consiste en una plataforma móvil con actuadores omnidireccionales, es decir, es capaz de realizar movimientos simultáneos en todas las direcciones: hacia adelante, atrás, lateralmente y de rotación sobre sí mismo.

Está equipado con una cámara conectada través de un puerto USB y con diversos sensores tanto analógicos como digitales. Se trata de una plataforma con interfaces mecánicas y eléctricas abiertas para la posible integración de elementos mecánicos, sensores y motores adicionales. Las dimensiones y los parámetros característicos de este robot pueden observarse en la Tabla 8.1 y en la Tabla 8.2, respectivamente.

Tabla 8.1 – Dimensiones del Robotino

DIMENSIONES	
Diámetro	370 mm
Altura	210 mm
Peso	11 kg aprox.

Tabla 8.2 – Parámetros característicos del Robotino

PARÁMETROS	**VALOR**
Alimentación de tensión	24 V DC, 4.5 A
Entradas digitales	8
Salidas digitales	8
Entradas analógicas	8 (0 – 10 V)
Salidas por relé	2

8.1.1.1. Control

El control del sistema se realiza mediante dos componentes: un PC embebido y un microcontrolador. El PC embebido se encarga del control de robot y está conectado directamente

con el microcontrolador cuyas funciones son las siguientes: supervisar la alimentación de tensión, regular el motor y controlar las entradas y salidas tanto analógicas como digitales. Los datasheets tanto del PC embebido como del microcontrolador se encuentran en las siguientes referencias respectivamente,[9], [10].

Tabla 8.3 – Estructura del hardware del Robotino, incluyendo el PC embebido y el microcontrolador

<table>
<tr><td colspan="4">PC Embebido</td><td colspan="3">Microcontrolador</td></tr>
<tr><td colspan="3">CPU</td><td rowspan="2">mSATA</td><td colspan="3">Microcontrolador de 32 bits</td></tr>
<tr><td>6 x USB</td><td>Puerto Serie</td><td>VGA</td><td>Analog Digital Converter</td><td>Puerto Serie</td><td>Puerto Serie</td></tr>
<tr><td>2 x PCI Express</td><td>USB para WLAN</td><td>Ethernet</td><td colspan="2">USB</td><td>Ethernet</td><td>Controller Area Network</td></tr>
</table>

En la Tabla 8.3 se muestran las estructuras de tanto el PC embebido como del microcontrolador y en ella se observa como ambas partes se comunican entre ellas mediante una conexión USB de forma interna. Por otro lado, no se puede acceder a la entrada ethernet del microcontrolador por lo que solo hay un único puerto ethernet. Sin embargo, dentro de este puerto se emulan dos entradas diferentes la eth0.0 y la eth0.1.

La conexión WiFi nos permite comunicarnos con el Robotino como maestro o como cliente. Cuando el Robotino está configurado como maestro actúa como punto de acceso (AP, Access Point), es decir, se crea una red WiFi a la que un ordenador externo se puede conectar. Si el Robotino funciona como cliente, este se conectará a una red externa. La elección de maestro o cliente se realiza en la interfaz web del Robotino al realizar la configuración.

8.1.1.2. Configuración

Robotino viene configurado en modo maestro con una dirección IP fija, es decir, generando una red WiFi propia. Para manejar el Robotino con comodidad en el laboratorio de trabajo, es más sencillo asignarle una dirección IP fija y hacer que trabaje en modo cliente, conectándose a la red WiFi disponible en el laboratorio.

Para realizar dichas configuraciones hay que acceder a la interfaz web de Robotino. Para ello, una vez encendido el Robotino y conectarse a la red generada hay que introducir la dirección IP por defecto del Robotino en el buscador.

- **Nombre de red generada:** Robotino.003.123
- **Contraseña de red:** robotino

- **Dirección IP:** 172.26.1.1

Una vez dentro de la interfaz web, hay que acceder a la pestaña *Network* para en ella editar el apartado "wlan0" cambiando el modo de Master a Client y realizando los cambios necesarios de IP y de red.

- **Nueva IP:** 192.168.1.160
- **Máscara:** 255.255.255.0

Una vez realizados los cambios para comprobar que la configuración se ha hecho de forma correcta hay que introducir la nueva dirección IP en el buscador. En caso de no haber configurado la red de forma correcta y, en consecuencia, perder la conexión del Robotino, habría que resetear el Robot apretando el botón de reseteo situado en la parte posterior del Robotino durante 3 segundos aproximadamente.

8.1.1.3. Sistema de transmisión

Robotino está dotado de un sistema de accionamiento omnidireccional conocido como *Omnidrive* que le permite realizar movimientos en todos los sentidos y girar sobre sí mismo. Este sistema está compuesto por tres unidades independientes formando ángulos de 120º entre sí. Cada uno de ellos incluye un motor, un *encoder* incremental, un reductor y las ruedas. La velocidad de los motores es controlada mediante PIDs implementados en microprocesadores que están conectados con el microcontrolador. La velocidad máxima que puede alcanzar es de 10 km/h. El *encoder* incremental permite comparar la velocidad real del motor con la velocidad deseada.

Para que las ruedas tengan un movimiento omnidireccional están formadas por ruedas más pequeñas pegadas a la periferia de rueda principal. De esta forma las ruedas traccionan en la dirección normal al eje del motor y se deslizan sin fricción en la dirección del eje del motor.

8.1.1.4. Sensores y elementos

El Robotino está compuesto por los siguientes sensores y elementos [11]:

1. **Paragolpes/Bumper**

El paragolpes o bumper es un sensor anticolisión que consiste en una banda de goma ajustada alrededor del chasis inferior del Robotino. Mediante este sensor se manda una señal al microcontrolador de Robotino que nos permite actuar dependiendo de ella para para la ejecución de un programa o calcular rutas una ruta variable.

2. Sensores de distancias

Los sensores de distancia son sensores de luz infrarroja situados a lo largo del chasis inferior del Robotino cada 40º. Hay un total de nueve sensores. Estos sensores tienen un rango de medición de entre 4 y 30 centímetros y para cada distancia devuelven una salida analógica de voltaje entre 0 y 3.3 V.

Para conocer con precisión la distancia a la que están los objetos, la web de Robotino proporciona una gráfica de tensión-distancia (ver Figura 8.1). Para distancias inferiores a 3 centímetros los valores de tensión pierden precisión.

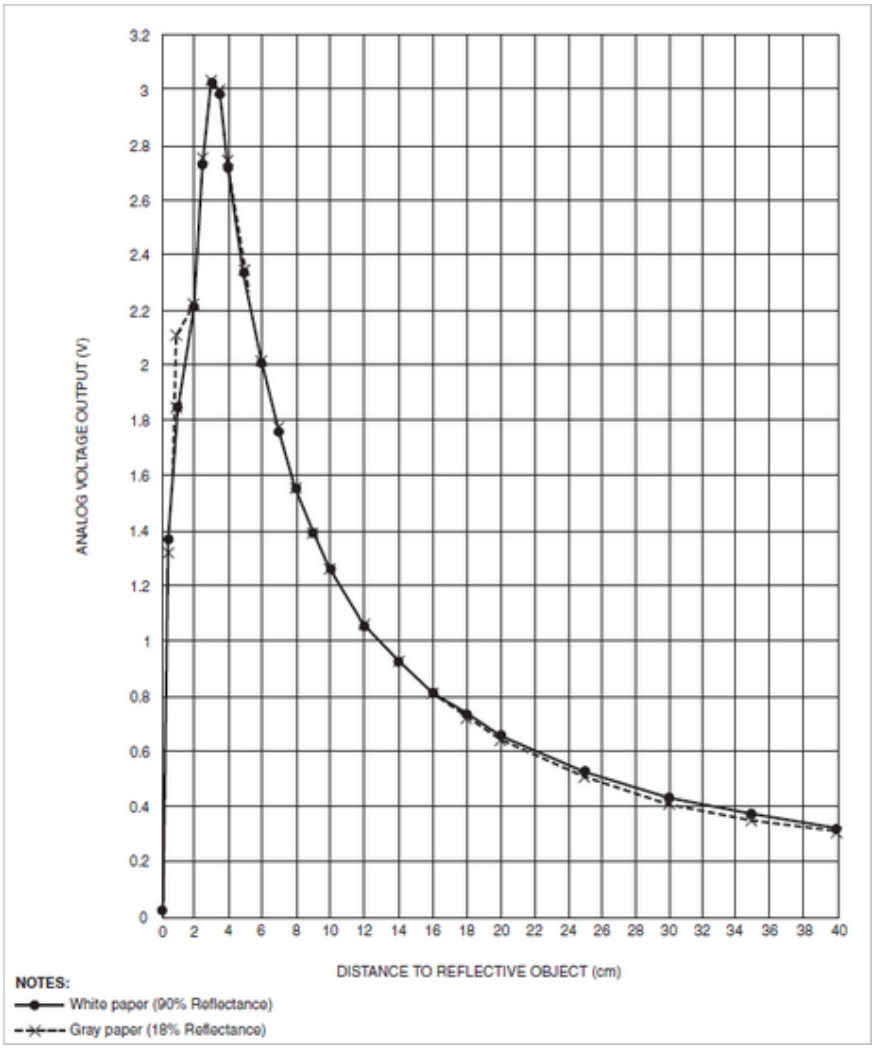

Figura 8.1 – Gráfica tensión-distancia proporcionada por Festo

3. Giroscopios

Los giroscopios se utilizan para aumentar la precisión a la hora de programar posiciones concretas del Robotino. Mediante ellos se realizan las mediciones de velocidades angulares, aunque si está en un ángulo concreto durante un periodo largo empieza a acumular un error denominado, *deriva*.

Además, para las aceleraciones lineales se dispone de acelerómetros que son dispositivos muy exactos a largo tiempo, pero que no trabajan bien cuando hay ruido de alta frecuencia.

4. **Cámara**

La cámara del Robotino está conectada mediante un USB a la CPU de este. Esta transmite imágenes en tiempo real para luego utilizarlas para navegar o para evitar obstáculos. La posición, altura e inclinación de la cámara pueden ajustarse dependiendo de las necesidades concretas de la aplicación.

8.1.1.5. Interfaz de Entradas/Salidas

En esta interfaz se pueden conectar los diferentes sensores que se utilizaran en elementos externos del Robotino con el microcontrolador. La numeración, cantidad y características de las entradas y salidas se observan en la Tabla 8.4.

Tabla 8.4 – Características de la interfaz de entradas/salidas del Robotino

CONEXIÓN	TIPO
DI1 … DI8	Entradas digitales (24 V)
DQ1 … DQ8	Salidas digitales (24 V, máximo 1 A)
AI1 … AI8	Entradas analógicas, (0 – 10 V)
NO1 … NC2	Relé, 24 V
+/-	Alimentación (24 V, máximo 3 A)

8.1.1.6. Suministro

El suministro eléctrico del Robotino se realiza a través de una unidad de alimentación que alimenta a su vez dos baterías de 12 V cada una que alimentan el Robotino a 24 V. Para comprobar el estado de las baterías se puede acceder a la interfaz web de Robotino. Estas baterías proporcionan un tiempo de funcionamiento de hasta dos horas.

8.1.2. Hardware del brazo

Tal y como se ha mencionado el apartado de “Alternativas” el brazo que se utilizará en este proyecto es el WidowX Robot Arm. El brazo está compuesto de 5 servomotores, cada uno de los cuales dota al brazo de un grado de libertad. Además, existe un sexto servomotor que acciona la pinza. El conjunto del brazo va sujeto a una base que aporta robustez y estabilidad al brazo al realizar los movimientos. La placa controladora del robot se sitúa en dicha base.

Los servomotores utilizados son de Dynamixel y cada uno de ellos dispone de un controlador PID para controlar las posiciones y velocidades de estos. En la Tabla 8.5 se muestran las características de cada uno de los servomotores y tal y como se observa, tienen un rango de operación de 360º, aunque por cuestiones estructurales del brazo se definen unos límites de utilización. Asimismo, en la Figura 8.2 se observa la secuencia de posiciones en los que deberán ir colocados los servomotores.

Tabla 8.5 – Parámetros característicos de los servomotores del brazo

	MX-64	**MX-28**	**AX-12 (BRAZO)**	**AX-12 (PINZA)**
Dimensiones (mm)	40.2 x 61.1 x 41	35.6 x 50.6 x 35.5	32 x 50 x 40	32 x 50 x 40
Peso (g)	126	72	54.6	54.6
Ángulo de operación	360º	360º	300º	300º
Límites de utilización (radianes)	$[-\pi/2, \pi/2]$	$[-\pi/2, \pi/2]$	$[-\pi/2, \pi/2]$	$[0, 2.6]$
Velocidad (rpm)	78	97	59	59
Resolución	0.088º	0.088º	0.29º	0.29º
Par torsor máximo (n·m)	78	97	1.5	1.5

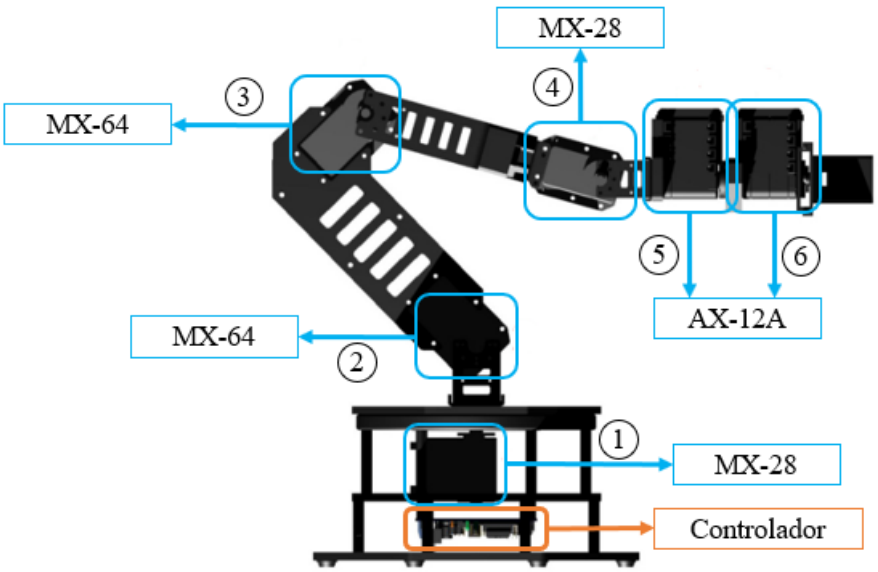

Figura 8.2 – Secuencia y modelo de los servomotores del brazo

Los seis servomotores están conectados en cascada (estructura *daisy chain*) mediante un cable de 3 pines la placa controladora (Arbotix-M Robocontroller) que, a su vez, se conecta con el Robotino mediante USB. El controlador está diseñado para trabajar en robots que disponen de servomotores Dynamixel en el entorno Arduino. Los elementos principales de la placa se observan en la Figura 8.3, siendo las principales características del controlador las siguientes:

- Microprocesador ATMega644p
- 28 entradas y salidas digitales, 8 de las cuales pueden funcionar como entradas analógicas
- Compatibilidad con Arduino IDE
- 3 puertos TTL para la conexión de los servomotores

Para alimentar el brazo no se utiliza la batería que proporciona Robotino si no que se alimenta el controlador a 12 V en corriente directa para que los servomotores funcionen correctamente.

Una vez realizadas las configuraciones necesarias que se detallarán a continuación, para el montaje de brazo Trossen Robotics proporciona una guía detallada en el siguiente enlace: https://www.trossenrobotics.com/productdocs/assemblyguides/widowx-robot-arm-mk2.html.

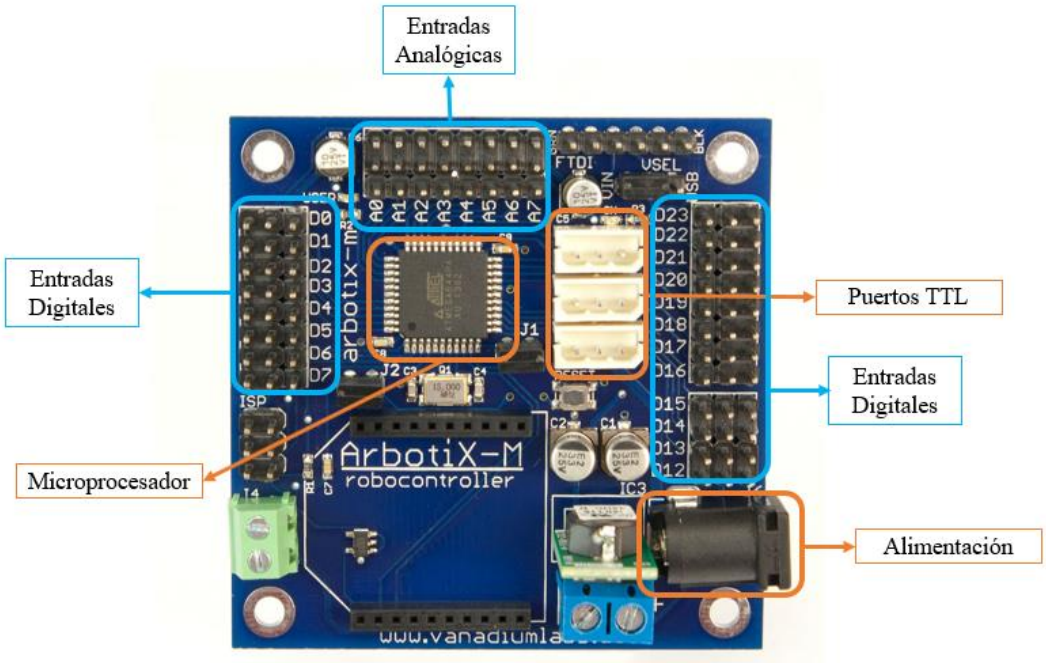

Figura 8.3 – Elementos característicos de la placa controladora Arbotix-M Robocontroller

8.1.2.1. Configuración del controlador

Arbotix-M Robocontroller es compatible con ROS como ya se ha mencionado anteriormente. Sin embargo, es necesario cargar dicho programa en el controlador. Para ello, a continuación, se describen los pasos que se deben seguir.

Para empezar, es necesario descargar la versión 1.0.6 del software Arduino IDE e instalar los drivers para el cable FTDI. Una vez hecho esto habrá que instalar las librerías, hardware y sketches proporcionados por Trossen Robotics para el controlador Arbotix-M Robocontroller. El hardware permite la selección del controlador utilizado y las librerías y sketches permiten la carga de ejemplos de programas proporcionados por ellos, así como de un driver del robot para ROS.

El siguiente paso consiste en la conexión física del controlador al ordenador. Para ello se utiliza un cable FTDI, que consiste en un cable con extremos USB y TTL serie. Cuando conectemos este cable, se debe tener en cuenta que la posición del jumper debe estar del lado del USB y que la orientación de extremo TTL serie debe ser la mostrada en la Figura 8.4.

A continuación, debe comprobarse que la comunicación entre el ordenador y el controlador se ha realizado correctamente. Para hacer esto hay que seleccionar el modelo del controlador en Arduino, elegir el puerto COM correspondiente y cargar el ejemplo *ArbotixBlink*. Si la conexión es correcta, el LED verde de la placa debería estar parpadeando.

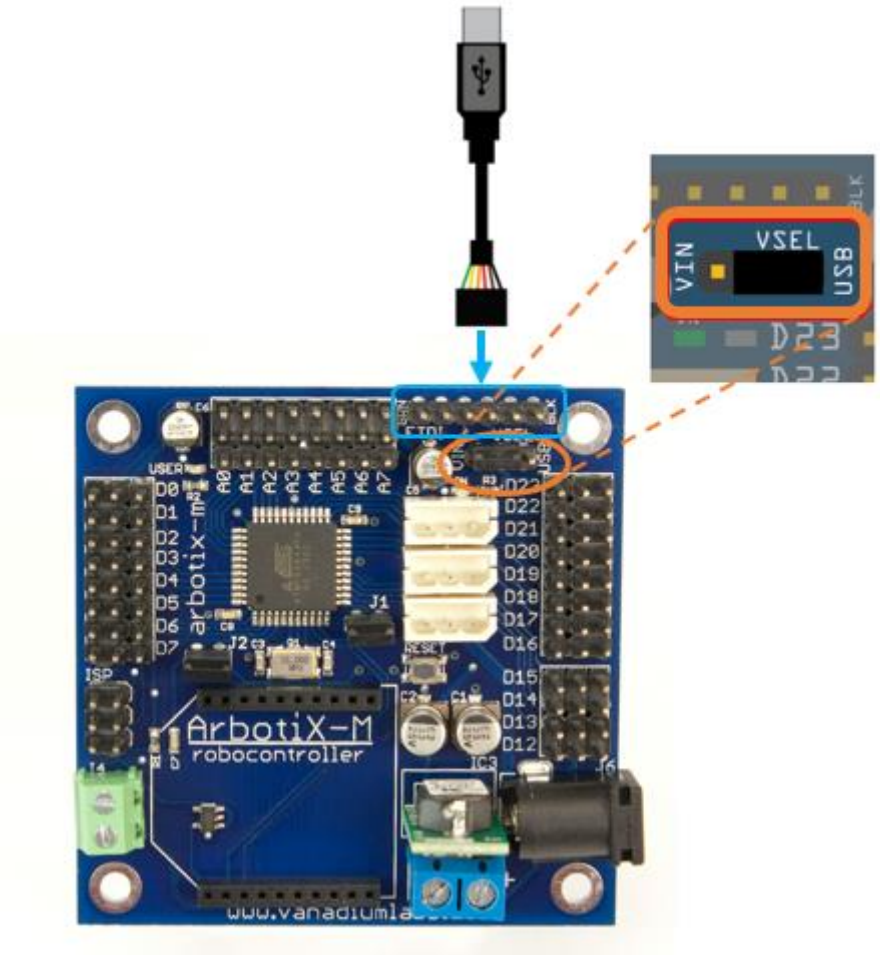

Figura 8.4 – Orientación y posición de los elementos necesarios para la configuración del controlador del brazo

Comprobada la comunicación, hay que volver a colocar en la posición original (lado de VIN) y probar el ejemplo de AXSimpleTest con el servomotor AX-12A conectado a la placa. Si el servomotor comienza a moverse la comunicación es correcta.

Para terminar la configuración del controlador habría que cargar el driver de brazo para ROS. Cuando este programa se carga en la placa actúa como intérprete del protocolo de comunicación empleado entre el paquete ROS y los servomotores, es decir, recibe paquetes de datos que traduce a instrucciones para los servomotores. En el siguiente enlace se describe dicha estructura de paquete de datos empleada: https://learn.trossenrobotics.com/arbotix/arbotix-communication-controllers/31-arm-control#packetStructure

8.1.2.2. Configuración de los servomotores

Tal y como se ha explicado, los servomotores deben ir conectados en unas posiciones muy concretas en la estructura del brazo. Para ello será necesario configurar dichos servomotores con identificadores. Los servomotores vienen identificados con el identificador ID #1 por defecto, por lo que será necesario asignarle a cada uno de ellos un número de identificación diferente de acuerdo con la posición que ocupen.

Para realizar la configuración se han seguido los pasos que se detallan en el siguiente enlace: https://learn.trossenrobotics.com/index.php/getting-started-with-the-arbotix/1-using-the-tr-dynamixel-servo-tool#&panel1-1.

Antes de comenzar, hay que tener en cuenta que no se puede realizar la identificación de diferentes servomotores conectados a la vez; el proceso se debe realizar de forma individual para cada servomotor.

En primer lugar, se debe instalar el software DynaManager que sirve para identificar los servomotores, para lo cual será necesario tener una máquina virtual de Java (JVM, Java Virtual Machine) instalada. Además, el controlador deberá tener cargado ROS.

Una vez realizadas estas operaciones, se procederá a trabajar con el software de DynaManager. Habrá que empezar conectado el controlador al programa siendo la forma más sencilla de hacerlo es darle al botón de "*Auto Search*" con el que el mismo programa localiza el puerto en el que está conectado el controlador. Se accederá así a una segunda pantalla en la que pulsando el botón de "*Scan*" se busca el servomotor conectado. El servomotor se caracteriza por un número de identificación y el modelo de éste. Será ese número de identificación el que será necesario modificar de acuerdo con la posición que ocupe. Por último, para el correcto montaje del servomotor en la estructura es también necesario centrar su posición.

8.2. Diseño de alto nivel

El siguiente apartado está estructurado de forma que en primer lugar se detallarán los aspectos característicos del software de funcionamiento de Robotino y del brazo, en un segundo lugar se analizará la estructura de funcionamiento de ROS de forma genérica, y, por último, se analizará la combinación del framework robótico ROS aplicada en los casos de Robotino y el brazo.

Sin embargo, antes de entrar en detalle con el software de cada elemento hay que mencionar que a lo largo de todo el proyecto se trabajará en Ubuntu16.04. Esto se debe a que la CPU de Robotino viene configurado con dicha versión del sistema operativo.

8.2.1. Software de Robotino

En este apartado se analizará la estructura de software de Robotino para poder entender dónde se sitúa ROS y qué elementos es necesario instalar para que a comunicación pueda funcionar.

Tal y como se puede observar en la Figura 8.5, Robotino se puede programar de diferentes formas y a través de diferentes lenguajes. En primera instancia se diferencian dos bloques: API2 y Lighttpd Webserver.

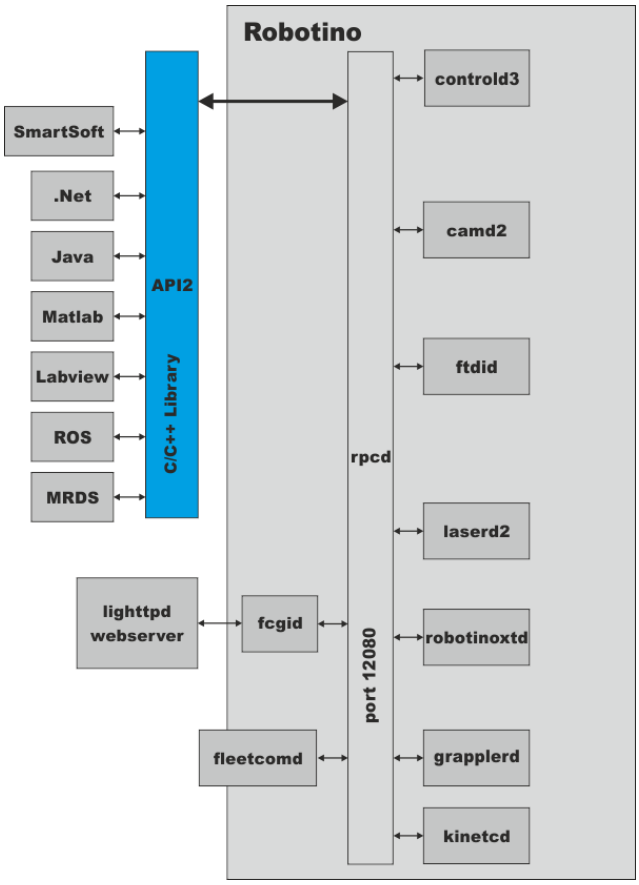

Figura 8.5 – Arquitectura general de funcionamiento del Robotino

Lighttpd Webserver es la interfaz web que proporciona Festo para el control, la configuración y el mantenimiento del Robotino cuyo acceso se realiza escribiendo la dirección IP del Robotino en la barra de direcciones del navegador. Esta interfaz está compuesta de las siguientes secciones:

1. **Program:** Permite controlar los programas creados a través de Robotino LabVIEW.
2. **Control:** Permite controlar manualmente el Robotino y hacer una comprobación inicial de las conexiones.
3. **Battery:** Proporciona información acerca del estado de las baterías.
4. **Network:** Permite configurar los ajustes de la red de Robotino.
5. **Settings:** Permite modificar distintos parámetros relacionados con Robotino.

API2 (Application Programming Interface) es un intérprete que contiene una librería de C/C++ que permite el acceso completo a los sensores y actuadores del Robotino. Se trata de una nueva versión desarrollada para reemplazar a OpenRobotinoAPI [Enlace: http://wiki.openrobotino.org/index.php?title=API2].

Mediante la instalación de esta librería se pueden utilizar diferentes lenguajes de programación para controlar el Robotino entre los que se pueden destacar MATLAB, LabVIEW y ROS.

Dado que uno de los requerimientos de este proyecto es hacer uso de ROS, la arquitectura software de Robotino a utilizar se simplifica a los elementos mostrados en la Figura 8.6. Sin embargo, hay muchos otros interfaces por los que se seguiría la misma metodología, por ejemplo: Java, MATLAB, LabVIEW, etc.

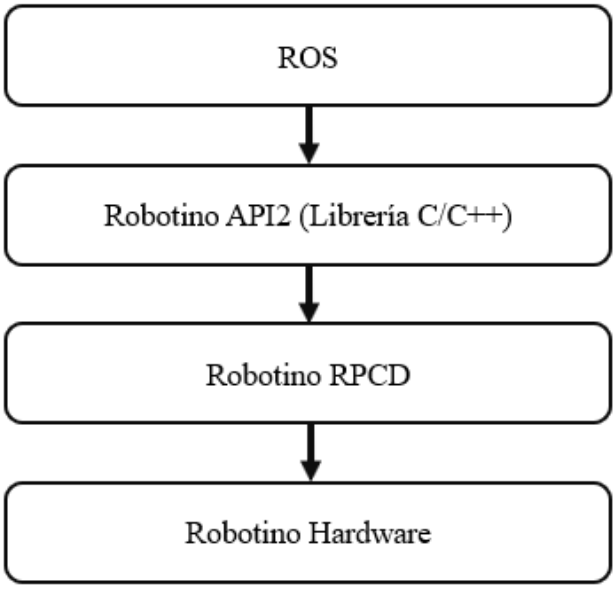

Figura 8.6 – Arquitectura simplificada del funcionamiento de Robotino mediante ROS

Robotino rpcd es elemento central del sistema para administrar los mensajes entre el hardware de Robotino y la librería de acceso de bajo nivel API2. Este elemento está conectado al puerto 12080 por defecto.

El rpcd forma parte de los Robotino-daemons que contienen las aplicaciones necesarias para la abstracción del hardware y la comunicación. Para poder instalar el paquete Robotino-daemons en el Robotino, éste debe estar actualizado a la última versión. La actualización puede descargarse a través del siguiente enlace: https://doc.openrobotino.org/download/tinycore/robotino4image/.

Una vez hecha la actualización, lo último que habría que hacer para para poder comunicarse mediante ROS sería descargar el paquete de ROS para Robotino. Los pasos necesarios para la instalación de ROS se proporcionarán en apartados posteriores.

8.2.2. ROS (Robot Operating System)

ROS es un framework robótico o meta-sistema operativo que consiste en un sistema de código abierto desarrollado para la reutilización de código. Se puede utilizar en todo tipo de robots mediante pequeños cambios en el código fuente. Entre los servicios que ofrece ROS se encuentran la abstracción del hardware del dispositivo, la posibilidad de implementar funciones comunes, el envío de mensajes de un proceso a otro, la gestión de diferentes paquetes, etc. Al tratarse de un

sistema de código abierto no hay ninguna restricción de acceso o uso, siendo el único requerimiento la instalación en el soporte Ubuntu.

A continuación, se explicarán el funcionamiento de general de ROS y la forma en la que está organizado.

8.2.2.1. Conceptos básicos

Para entender el funcionamiento del sistema se mostrarán dos niveles de conceptos básicos de ROS: el nivel del sistema de archivos y el nivel de elementos de computación.

Sistemas de archivos:

- **Packages (Paquetes):** Los paquetes son la unidad principal de organización de software en ROS. En ellos reside la estructura mínima y el contenido necesario para la creación de un programa en ROS. Un paquete puede contener los siguientes elementos: procesos ejecutables, bibliotecas dependientes, conjuntos de datos, archivos de configuración, etc.
- **Manifests (Manifiestos):** Los manifiestos contienen la información relacionado con cada paquete. Entre dicha información podemos encontrar la información de licencia, las dependencias, el idioma, etc.
- **Stacks (Pilas):** Se trata de una colección de paquetes mayormente utilizada en proyectos de gran alcance.
- **Stack Manifest (Manifiesto de pilas):** Al igual que en el caso anterior, estos proporcionan la información relacionada con una pila en concreto
- **Message (msg):** Un mensaje es la información que se envía entre procesos. En ellos se caracteriza la estructura de datos que se recibe o se publica.
- **Service (srv):** Estos archivos contienen la descripción relacionada con un servicio.

En la Figura 8.7 se muestra la estructura de los archivos en ROS donde un repositorio (Repository) es una red federada de código. En los repositorios, diferentes instituciones desarrollan y lanzan el software del robot correspondiente.

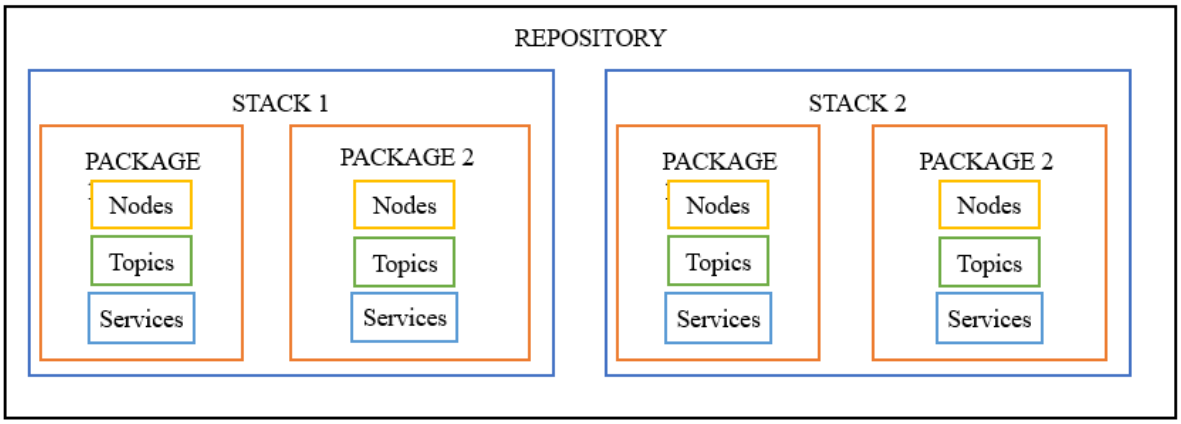

Figura 8.7 – Estructura de los archivos en ROS

Elementos de computación:

- **Nodes (Nodos):** Se trata de procesos de cálculo que se pueden encontrar en el sistema de control del robot, es decir, son ejecutables dentro de paquetes de ROS. El sistema suele estar compuesto por múltiples nodos que pueden estar escritos en leguajes como C++, Python, Lisp, Java o Lua.
- **Master (Maestro):** Es un elemento que pone en contacto diferentes nodos y realiza el registro de nombres. Sin un maestro no habría comunicación entre los nodos. Sin embargo, una vez realizado el registro el maestro no es participe de las publicaciones y suscripciones de los nodos. Por otro lado, un mismo maestro se puede utilizar para registrar nodos de diferentes máquinas posibilitando la opción de robótica colaborativa bajo un único sistema de ROS.
- **Parameter Server (Servidor de parámetros):** Permite el almacenamiento de datos que posteriormente configuren los nodos en mitad de su ejecución.
- **Messages (Mensajes):** Son estructuras de datos que se envían entre nodos, es decir, es el contenido de un topic publicado por un nodo y que otro nodo puede recibir. Tienen una estructura concreta ya que el tipo de mensaje publicado por un nodo debe ser coincidente con el mensaje recibido por el otro.
- **Topics (Temas - Tópicos):** Son los elementos a través de los cuales se envían los mensajes, es decir, forman un sistema de transporte de mensajes. Un mismo nodo puede publicar y suscribirse a diferentes nodos de forma simultánea. Esto permite hace que los nodos estén desacoplados unos de otros, es decir, que desconozcan la situación o estado del resto de nodos.
- **Services (Servicios):** Consiste en un sistema distribuido de comunicación entre nodos para cuando los topics no sean válidos.

- **Bags (Bolsas):** Estos elementos son un formato para poder guardar y posteriormente reproducir información correspondiente a los mensajes.

En consecuencia, el principio de funcionamiento general de ROS consiste en que un nodo envía un mensaje a otro a través de un tópico. Para hacer el envío del mensaje el primer nodo deberá publicar la información en un tópico al que el nodo receptor se suscriba.

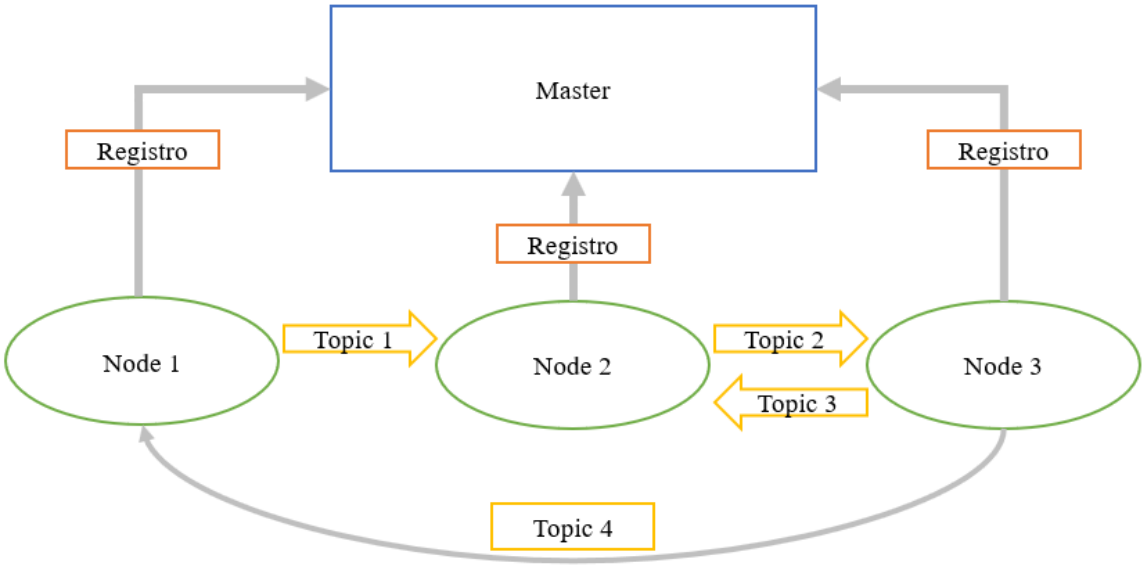

Figura 8.8 – Arquitectura de la comunicación en ROS

Por lo tanto, tal y como se muestra en la Figura 8.8, los nodos se comunican entre si sin transmitir la información ni los mensajes por el maestro. Esto se debe a que el maestro sólo aporta información de búsqueda de los nodos ya que todos los nodos que formen parte del sistema deben registrarse en él para recibir información del resto de nodos registrados y hacer las conexiones correspondientes para comunicarse mediante los topics.

La Figura 8.9 muestra el detalle de la comunicación entre diferentes nodos de forma simplificada. Sin embargo, se debe tener en cuenta que un nodo puede publicar y subscribirse a un tópico de forma simultánea y por lo tanto la información podría fluir en ambos sentidos a través de tópicos diferentes.

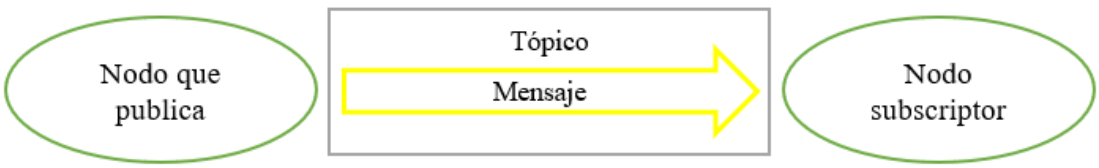

Figura 8.9 – Estructura de comunicación entre el nodo que publica y el nodo que se subscribe

Para acceder a la interfaz gráfica que muestra los nodos y tópicos que estén siendo utilizados, es decir, que estén activados, en un momento en concreto, hay que introducir el comando "*rqt_graph*". El resultado obtenido es una figura similar a la mostrada en la Figura 8.10, cuyos nombres de nodos y tópicos dependerá de los nodos concretos que se hayan arrancado en ese momento.

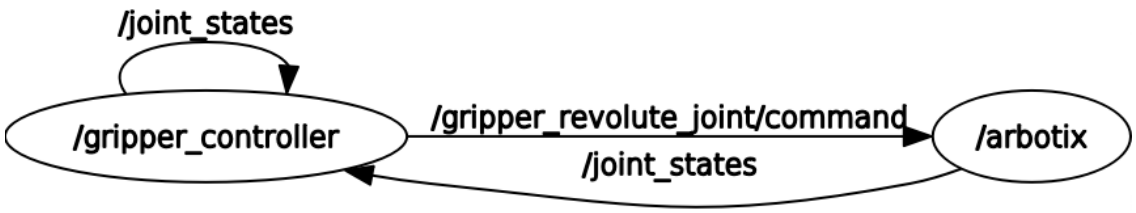

Figura 8.10 – Ejemplo del gráfico que se obtiene al utilizar "rqt_graph"

8.2.2.2. Comandos

Tal y como se ha mencionado en apartados anteriores, el framework robótico ROS se ejecuta sobre el sistema operativo Ubuntu. En consecuencia, la forma más habitual para invocar la funcionalidad de ROS es mediante CLI (Command Line Interface). A continuación, se presentarán los comandos básicos de ROS:

- **roscd:** Este comando se utiliza para navegar a través de distintos directorios de paquetes o pilas de paquetes.
    - **Estructura:** `roscd <nombre_paquete> (/subdirectorio)`
- **roscore:** Ejecución que da soporte al sistema, es una colección de nodos y programas que deben lazarse para que el sistema funcione. Permite la comunicación entre los nodos ya que pone en marcha el maestro, el servidor de parámetros y el nodo *rosout.* Sin embargo, la utilización de este comando se convierte innecesaria cuando se utiliza el comando *roslaunch*
- **roscreate-pkg:** Creación de un paquete junto con las dependencias correspondientes. Debe ejecutarse una vez se está dentro del directorio deseado.
- **rosls:** Muestra el contenido de un paquete
    - **Estructura:** `rosls <nombre_paquete> (/subdirectorio)`
- **rosnode:** Este comando proporciona información sobre un nodo. Ofrece los siguientes comandos para operar sobre los nodos:
    - **rosnode info:** Muestra información sobre el nodo
    - **rosnode kill:** Detiene el proceso ejecutado por el nodo
    - **rosnode list:** Muestra la lista de todos los nodos en ejecución
- **rosrun:** Comando que sirve para la ejecución de programas de un paquete, es decir, nodos. Para ejecutar este comando no hace falta encontrarse dentro del directorio del

nodo. Si se van a ejecutar diferentes nodos simultáneamente utilizando este comando las operaciones deben realizarse en pestañas diferentes de la consola.

  - **Estructura:** `rosrun <nombre_paquete> <nombre_nodo>`

- **roslaunch:** Con este comando se ejecuta el maestro mediante un archivo tipo launch. Además, ejecuta una colección de varios nodos que previamente han sido definidos en el fichero launch.
  - **Estructura:** `roslaunch <nombre_paquete> <nombre_archivo.launch>`
- **rostopic:** Comando que proporciona información sobre un tópico. Se utiliza por medio de alguna de las siguientes combinaciones:
  - **rostopic echo:** Escribe los datos que hay en el topic.
  - **rostopic find:** Encuentra un topic.
  - **rostopic info:** Muestra información correspondiente al topic.
  - **rostopic list:** Muestra la lista de todos los topics activos.
  - **rostopic pub:** Publica datos en un topic.
- **catkin_make:** Compila los paquetes situados en el espacio de trabajo (workspace)

8.2.3. Instalación de ROS en el Robotino

Este apartado pretende resumir de forma clara y concisa los pasos necesarios para la correcta instalación de ROS en el Robotino. Al tratarse ROS de un framework robótico abierto para instalar los componentes necesarios se recurrirá a la página RobotinoWiki y a la comunidad que respalda el programa.

Resumiendo, la instalación de ROS se ha realizado siguiendo los siguientes pasos:

1. Actualización a la última imagen de Robotino. Enlace: https://doc.openrobotino.org/download/tinycore/robotino4image/
2. Descarga de la última versión del paquete Debian del API2 a partir del RobotinoWiki. Enlace: http://packages.openrobotino.org/xenial/pool/main/r/robotino-api2/
3. Instalación de ROS Kinetic. Enlace: http://wiki.ros.org/kinetic/Installation/Ubuntu
4. Instalación del paquete de nodos de Robotino para ROS. Enlace: https://doc.openrobotino.org/ROS/
5. Creación del espacio de trabajo, catkin_ws, para lo que se acude a los tutoriales de ROS. Enlace: http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment
6. Instalación de los Robotino daemons para obtener el elemento administrador de mensajes rpcd. Enlace: http://packages.openrobotino.org/xenial/pool/main/r/robotino-daemons/

7. Sustitución del espacio de trabajo proporcionado al descargar ROS por el proporcionado en el siguiente enlace: https://github.com/SQRSMN/RIC/

8.2.4. Interfaz gráfica MoveIt

Hecha la instalación de ROS, existe la posibilidad de realizar la programación de trayectorias del brazo a través de la herramienta MoveIt, cuyas funciones son las siguientes:

- Planificación de movimientos y trayectorias
- Manipulación de entorno
- Resolución de la cinemática inversa
- Control y ejecución de trayectorias
- Percepción tridimensional del entorno
- Comprobación de colisiones

Para la instalación de MoveIt los pasos a seguir se detallan en el siguiente enlace: http://wiki.ros.org/kinetic/Installation/Ubuntu.

En la Figura 8.11 se muestra la arquitectura del sistema MoveIt. Tal y como se observa toda la comunicación se realiza a través del nodo `move_group` a la que el usuario se puede referir a través de diferentes interfaces entre las que se distinguen dos tipos: las interfaces que se comunican con el `move_group` a través de un lenguaje de programación como C++ o Python y la interfaz gráfica de usuario, Rviz Plugin.

Dado que la implementación mediante lenguajes de programación puede realizarse sin utilizar la interfaz MoveIt, se explicará más detalladamente la parte de la interfaz gráfica, Rviz Plugin. Se trata de un software de visualización que permite crear entornos y escenarios virtuales en los que trabajar con el robot. La principal funcionalidad de Rviz consiste en visualizar diferentes posibilidades de trayectorias una vez introducidas las posiciones iniciales y finales de los servomotores.

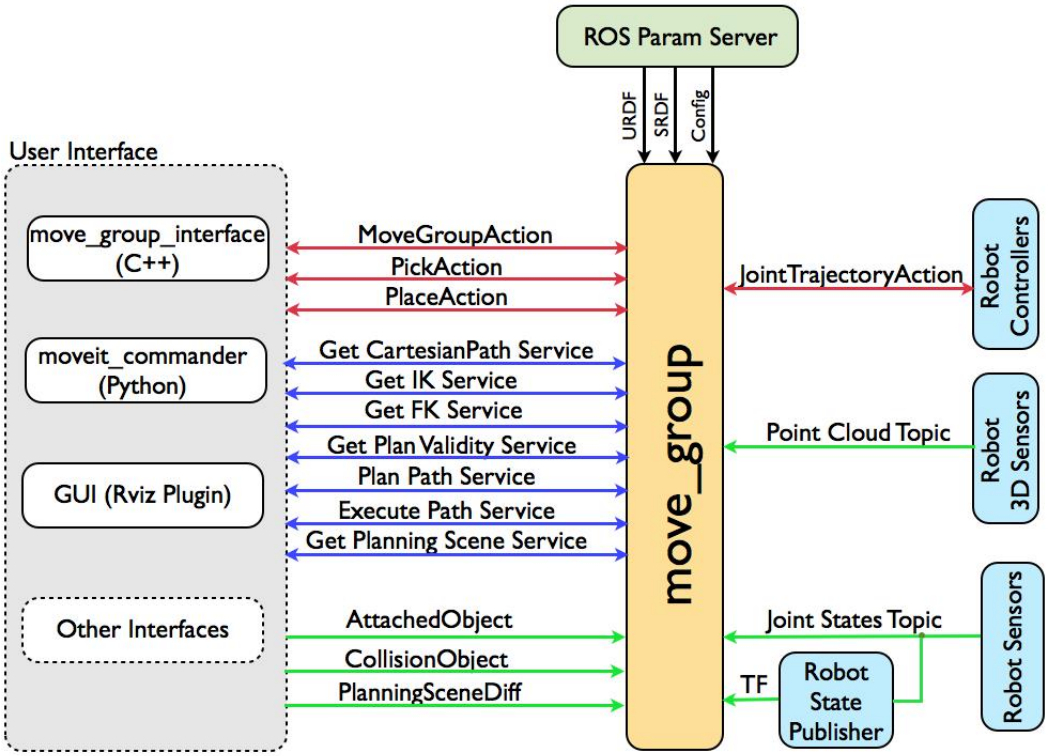

Figura 8.11 – Arquitectura de funcionamiento del sistema MoveIt

En consecuencia, cuando la instalación esté hecha, se lanza el archivo tipo launch que de acceso al programa. Con el robot conectado al ordenador, una vez que se está dentro se observa un modelo virtual del brazo de robot. De esta forma, se pueden probar distintas posiciones del robot de forma que no se ejecutarán hasta pulsar el botón encargado de dicha funcionalidad. También existe la opción de planificación que muestra en pantalla la trayectoria que realizaría el robot.

Sin embargo, no se ha encontrado la opción para que el usuario controle las velocidades de los servomotores ni el tiempo de ejecución del programa desde MoveIt, por lo que se concluye que se trata de un programa con limitaciones para lograr los objetivos del proyecto. Esto se debe al hecho de que no se puede acelerar una etapa genérica del movimiento y ralentizar la velocidad una vez se llegue a la etapa de aproximación y agarre de la pieza. Aun así, se seguirá utilizando cuando sea necesario obtener los datos de una posición exacta de los servomotores y valorar si la trayectoria planteada es válida.

8.3. DISEÑO DE BAJO NIVEL

En este siguiente apartado, se procederá a la descripción detallada del diseño de bajo nivel realizado en el proyecto. Para ello se explicarán los diferentes nodos a los que se puede acceder tanto en el Robotino como en el brazo y como publicar información o subscribirse a ellos.

8.3.1. Robotino y ROS

Una vez realizadas las instalaciones de ROS en Robotino, se puede acceder a todas las posibilidades que este ofrece. Para ello en primer lugar se debe activar el maestro y lanzar poner en funcionamiento varios nodos mediante el archivo tipo launch *robotino_node.launch* del directorio *robotino_node* disponible una vez instalado ROS.

8.3.1.1. Tópicos de robotino_node

Tras lanzar el archivo se tiene acceso a los diferentes tópicos que éste ofrece. A continuación, se analizarán los tópicos más utilizados.

1. **cmd_vel**

Se trata del tópico en el que el usuario publica las velocidades de los 3 motores encargados del movimiento de Robotino. El tipo del mensaje que se publica es el tipo de dato complejo que se muestra en la Figura 8.12. Por otra parte, también deben tenerse en cuenta los ejes cartesianos mostrados en la Figura 8.13 en los que se basa Robotino para realizar los movimientos principales: movimiento lineal de avance y retroceso, movimiento lineal lateral y giro en sobre sí mismo.

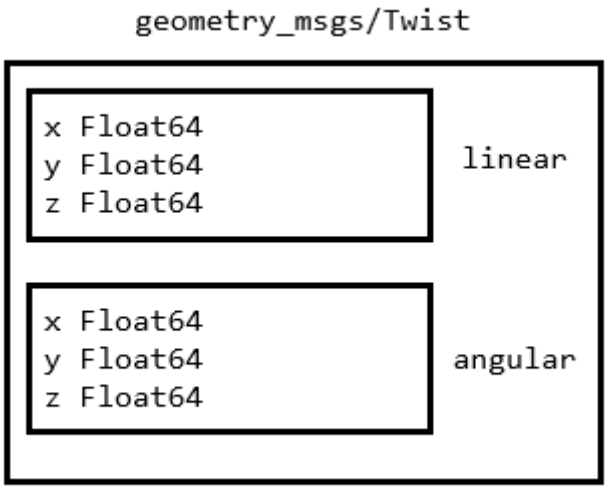

Figura 8.12 – Estructura del mensaje de velocidad del Robotino

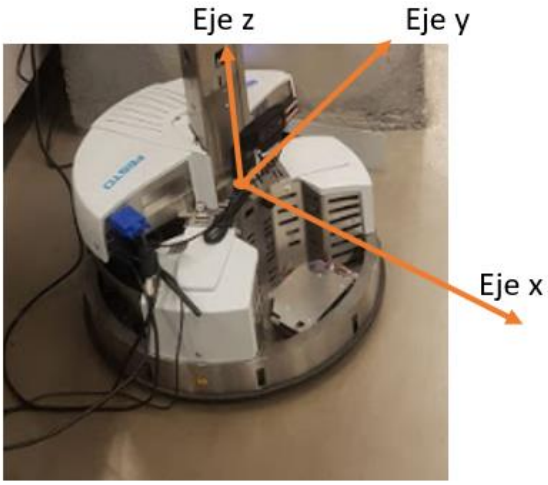

Figura 8.13 – Ejes cartesianos para el movimiento del Robotino

Este tipo de mensaje posibilita la opción de mover el robot en dos ejes de forma simultánea para realizar, por ejemplo, trayectorias diagonales. Sin embargo, se debe tener en cuenta que al realizar dos movimientos a la vez el robot debe ir interpolando los ejes por lo que pierde precisión en la dirección.

2. **bumper**

Este tópico proporciona información sobre el estado del bumper de forma constante, es decir, no manda información en el momento en el que ocurre un cambio de estado del bumper, sino que lo hace de forma continua, independientemente de los cambios que ocurran. Por la tanto, el usuario deberá subscribirse a él.

La información que envía es una variable tipo Bool (Figura 8.14) que manda el valor 0 cuando no hay nada en contacto con el bumper y el valor 1 cuando este esté activo por culpa de un choque, por ejemplo.

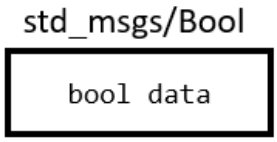

Figura 8.14 – Estructura del mensaje del nodo bumper del Robotino

3. **distance_sensors**

Es el tópico que recibe la información correspondiente de cada uno de los sensores de distancia. Por lo tanto, el usuario deberá subscribirse a él.

Por defecto, las distancias que miden los sensores se descomponen en dos ejes devolviendo la distancia en metros lo que no es práctico para este proyecto ya que se prefiere obtener un solo

valor en la dirección de medida del sensor. Para realizar este cambio se debe modificar el archivo *DistanceSensorArrayROS.cpp* de la configuración de los sensores. Para acceder a él la línea de comando necesaria es:

```
$ cd /catkin_ws/src/ros/robotino_node/src
```

En este archivo se observa cómo se definen las siguientes líneas para dar valor a los parámetros x e y:

```
distance_msg_.points[i].x = ( distances[i] + 0.2 ) * cos(0.698 * i);
distance_msg_.points[i].y = ( distances[i] + 0.2 ) * sin(0.698 * i);
distance_msg_.points[i].z = 0.05; // 5cm above ground
```

Para obtener la información deseada hay que sustituir o comentar las dos líneas superiores por las siguientes:

```
distance_msg_.points[i].x = distances[i] * 100;
distance_msg_.points[i].y = distances[i] * 100;
distance_msg_.points[i].z = 0.05; // 5cm above ground
```

Además, para obtener un valor de medida más adecuado se han multiplicado los valores por 100 para así trabajar con las unidades en centímetros y no en metros.

El tipo de mensaje que se recibe tiene la estructura compleja mostrada en la Figura 8.15. Antes de realizar el cambio mencionado las variables x e y, proporcionaban el valor descompuesto de la medida del sensor, pero tras el cambio, como se ha decidido no descomponer ese valor, las variables devolverán el mismo valor duplicado. Por lo tanto, solo será necesario utilizar uno de los datos. La variable z, por su parte, da todo el tiempo un valor constante que hace referencia a la altura a la que están situados los sensores. En el apartado *channels* se devuelve un array que indica a que sensor hacen referencia las medidas realizadas. La numeración y disposición de los sensores se muestra en la Figura 8.16

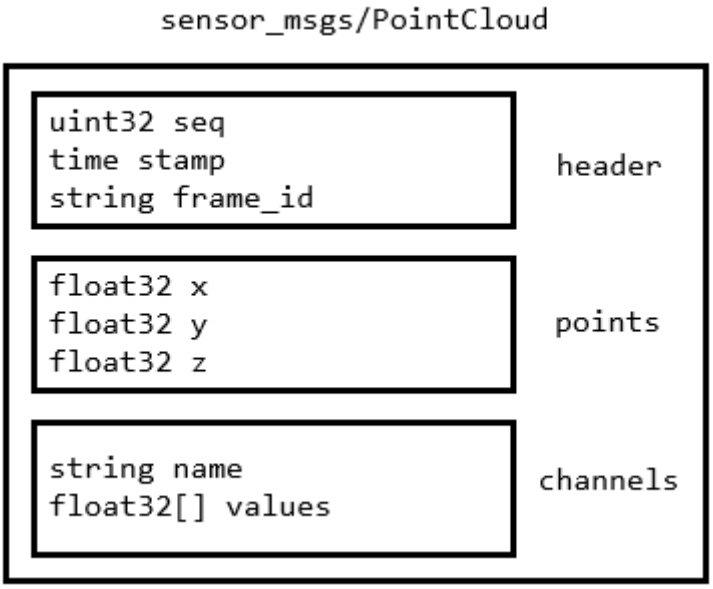

Figura 8.15 – Estructura de mensaje de los sensores de distancia del Robotino

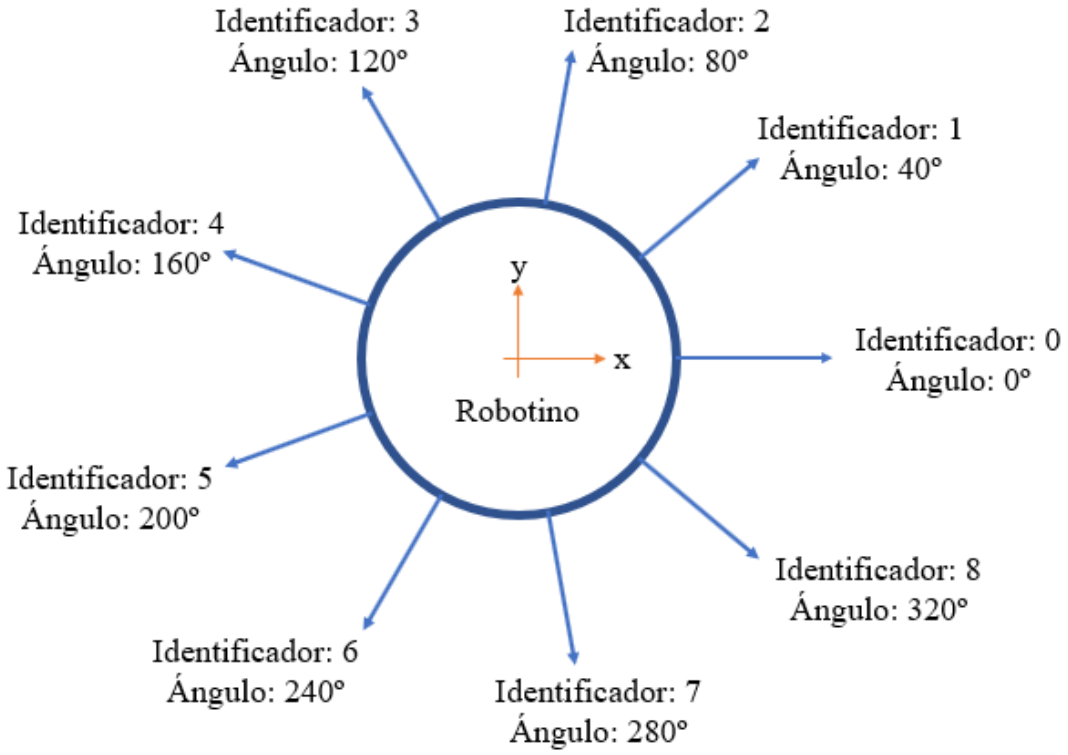

Figura 8.16 – Posicionamiento de los sensores de distancia del Robotino

4. odom

Este tópico proporciona datos sobre la odometría del robot. La odometría es el conjunto de datos utilizados para definir una posición en distancia y ángulo respecto a origen de coordenadas. Se utiliza, por tanto, para analizar los movimientos realizados por lo que, el usuario se subscribe a

él. En el caso de Robotino, el origen de coordenadas queda definido en el punto en el que se enciende la máquina. El tipo de mensaje a tratar se muestra en la Figura 8.17.

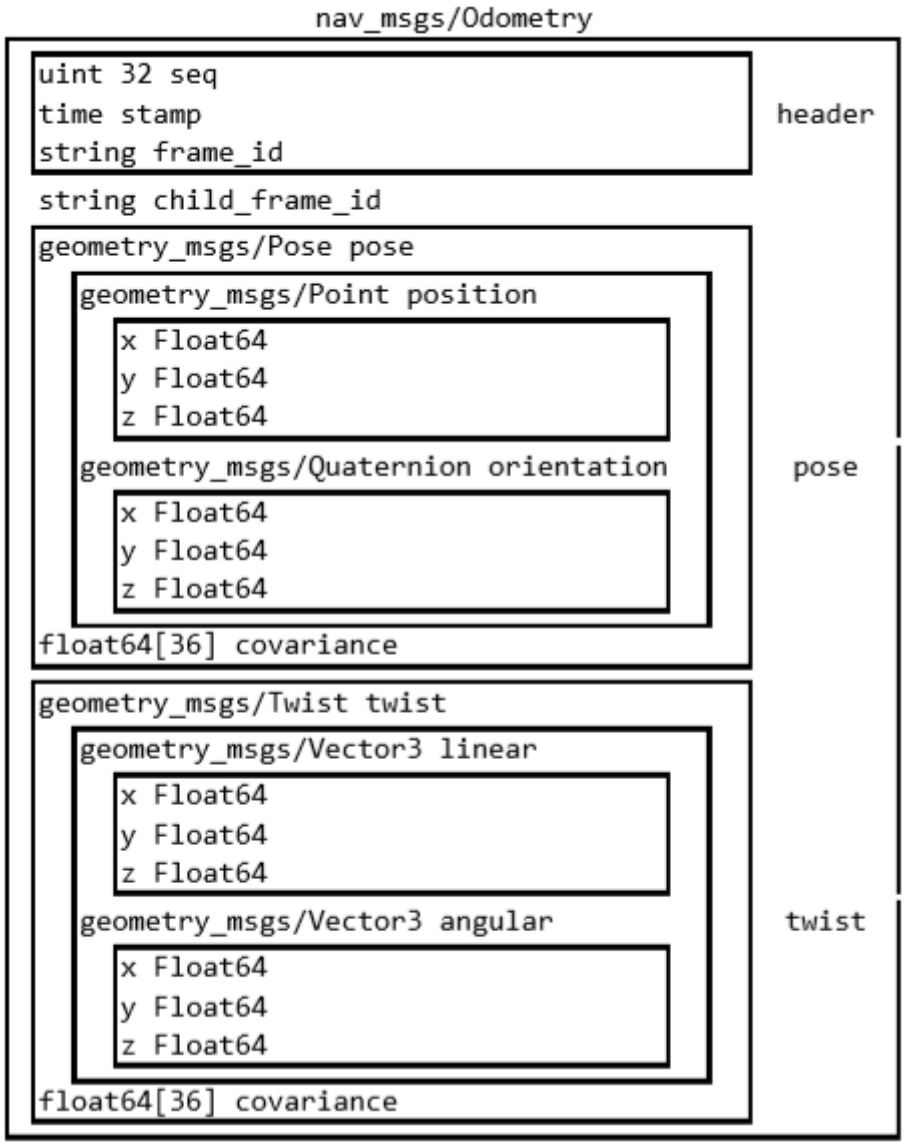

Figura 8.17 – Estructura del mensaje del tópico odom del Robotino

Tal y como se puede observar en la Figura 8.17 la estructura del mensaje es compleja. Aun así, los datos de interés son los referentes a la posición (*pose*) ya que en ellos se nos da la información de la distancia en los ejes cartesianos y la orientación del robot respecto del origen.

8.3.1.2. Análisis del movimiento

Conocidos los nodos principales de Robotino, en este apartado se describirán las características y conceptos a tener en cuenta para moverlo.

En el caso de calcular las distancias recorridas y los giros realizados se distinguen dos posibilidades:

1. **Velocidad y tiempo:** Para que el robot se mueva es necesario publicar las velocidades deseadas en el tópico *cmd_vel*. Por lo tanto, se puede calcular la distancia lineal recorrida o el ángulo girado multiplicado la velocidad publicada por el tiempo de ejecución. Para determinar el tiempo de ejecución hay que tomar el tiempo antes de empezar el movimiento en una variable, calcular la diferencia y definir un ciclo "*while*" en el que se actualice otra variable de tiempo. Las unidades para definir las distancias y los ángulos son metros y radianes, respectivamente.
2. **Odometría:** La segunda de las opciones se basa en el tópico *odom*. El cálculo de distancias y ángulos en este caso se realiza utilizando condicionantes de orientación en el eje z y de posición en los ejes x e y. La orientación depende del coseno del ángulo que forma con respecto al origen, pero no llega a alcanzar los valores 0 y 1 por lo que tiene un pequeño error.

Por razones de seguridad, cada vez que se realice un movimiento del robot se introducirán condicionantes de forma que si el bumper se activa o los sensores de distancia detectan un objeto próximo se detenga el movimiento. La elección acerca de cómo calcular las distancias y los ángulos, tiene una gran relevancia en este aspecto. Si se calcula mediante velocidades y tiempos (opción 1), el movimiento se detiene al activarse alguno de los parámetros de seguridad y a partir de ahí el tiempo de ejecución proporcionaría un dato incorrecto. Por lo tanto, se debe asumir que el área de trabajo está libre de obstáculos u objetos y que los parámetros de seguridad se activan si ha llegado a los finales de recorrido. En caso de trabajar con el nodo de odometría se evitaría está problemática ya que no tiene en cuenta el tiempo de ejecución.

8.3.2. Brazo manipulador y ROS

A continuación, se detallarán los elementos con los que se puede manipular el brazo desde ROS. En este caso, se debe tener en cuenta que el brazo utilizado no está equipado con ningún tipo de sensor por lo que no es posible comprobar ni asegurar que la trayectoria escrita en el programa se ha cumplido sin errores. Por lo tanto, tampoco se podrán utiliza parámetros para garantizar la seguridad del brazo, personas u objetos y evitar que haya colisiones.

En consecuencia, el único tópico que se utilizará en este proyecto será el que hace referencia a las posiciones de los servomotores. El tópico, denominado **'/arm_controller/command'**, permite definir arrays de posiciones para todos los servomotores del brazo de forma simultánea y también, controlar el tiempo de ejecución de cada trayectoria, si se concatenan diferentes posiciones. Para

describir de forma correcta estos conceptos en la Figura 8.18 se muestra el tipo de mensaje con el que se trabaja.

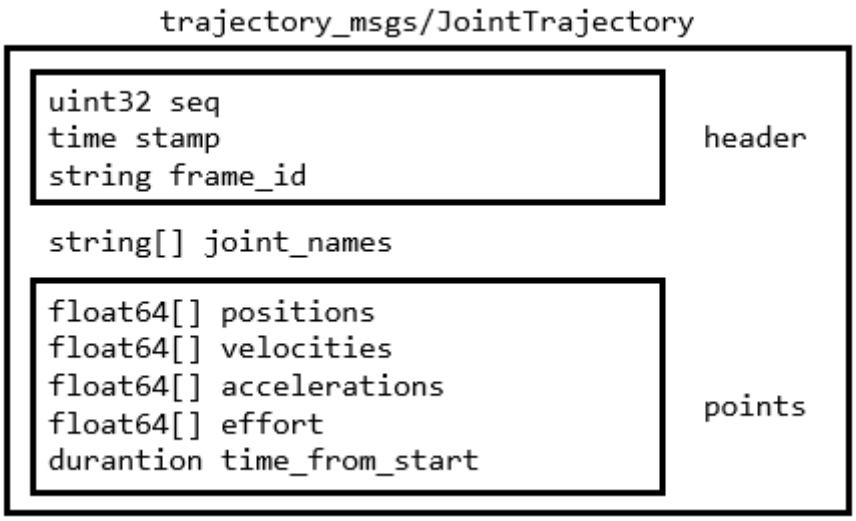

Figura 8.18 – Estructura del mensaje de publicación de trayectorias en el brazo

De acuerdo con la estructura de mensaje mostrada, en primer lugar, se debe definir la variable que es de tipo *JointTrayectory*. En la variable definida se debe guardar el tiempo antes de comenzar la ejecución de la trayectoria en la variable `stamp` y definir el parámetro *joint_names* con los nombres de los servomotores impuestos en la configuración. Además, debe definirse la variable *frame_id* de acuerdo a la configuración. Todos estos parámetros tendrán una estructura similar cada vez que ejecute una trayectoria siendo la única variable que cambiar la variable de tipo *JointTrayectory*. En la Figura 8.19 se muestra la configuración inicial descrita que se utiliza a lo largo del proyecto.

```
trayectory_<número> = JointTrayectory()
trayectory.joint_names = [“joint_1”, “joint_2”, “joint_3”, “joint_4”, “joint_5”,
                          “gripper_revolute_joint”]
trayectory.header.stamp = rospy.Time.now()
trayectory.header.frame_id = “base_footprint”
```

Figura 8.19 – Configuración inicial necesaria para publicar una trayectoria en el brazo

En segundo lugar, quedaría definir los distintos puntos por los que se hace pasar a los servomotores para realizar la trayectoria. Para hacerlo se definen las posiciones correspondientes a cada servomotor en un instante concreto en el orden de correspondiente a *joint_names* y luego se define el parámetro *duration* con una variable de tipo entero. Este parámetro *duration* hace referencia al tiempo que debe pasar desde que comienza la trayectoria, es decir, el tiempo desde que se define el parámetro *stamp*. Por lo tanto, si se concatenan diferentes posiciones, este

parámetro debe ir siempre en aumento. Hay que considerar también que controlando duración del movimiento se controla la velocidad con la que se moverán los servomotores. Cuanto mayor sea su valor más lento será el movimiento y mejor se controlarán las aproximaciones, por ejemplo.

Por último, queda mencionar que tras concatenar las diferentes posiciones estas deben publicarse en la variable *trayectory_<número>* creada en la configuración inicial y posteriormente, hay que parar el programa durante un tiempo igual o superior a la duración total de la trayectoria para que el brazo tenga tiempo de ejecutar todos los movimientos correctamente.

9. Descripción de los resultados

Una vez realizado el diseño y análisis tanto de alto como de bajo nivel de los sistemas tratados en el proyecto, en este apartado se evaluarán los resultados obtenidos y si a través de estos resultados se han logrado completar los objetivos propuestos al inicio del proyecto.

En primer lugar, se debe mencionar que todo el código y los programas escritos a lo largo de este proyecto están disponibles en el siguiente repositorio de github: https://github.com/JulenCuadra/TFG_Robotino_WidowX

El código desarrollado está escrito en lenguaje Python y en él se han utilizado los datos de los sensores de distancias y del bumper cuando se realiza el movimiento de base de la plataforma móvil. Por otra parte, se han planteado dos trayectorias de movimiento para el brazo para la recogida y devolución de un objeto.

La aplicación programada está formada por los siguientes módulos de movimiento:

1. Movimiento de avance positivo en el eje x.
2. Giro de 90º positivo
3. Movimiento de avance positivo en el eje x
4. Trayectoria de aproximación al objeto
5. Cierre de pinza y trayectoria de vuelta a la posición defecto
6. Movimiento diagonal negativo en el eje x y positivo en el eje y
7. Giro de 180º positivo
8. Trayectoria para deposición del objeto
9. Giro de 90º positivo
10. Movimiento de retroceso en el eje x

La Figura 9.1 muestra una representación gráfica de estos movimientos incluyendo los ejes principales del Robotino en cada movimiento de traslación de la base.

Todos los movimientos de la plataforma móvil se han programado con parámetros de seguridad que incluyen la parada de la plataforma por la activación por colisión del bumper y la detección de distancias demasiado pequeñas. Para la adquisición de datos del bumper ha sido necesario programar una variable global.

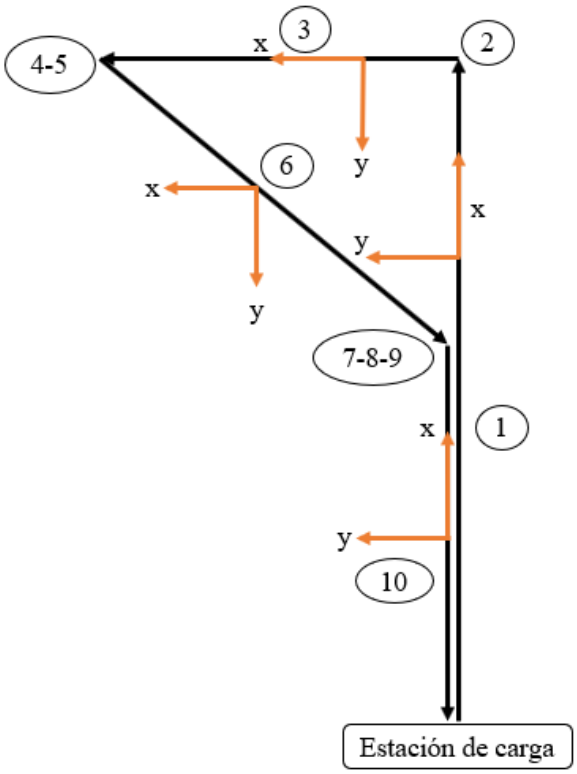

Figura 9.1 – Secuencia de los movimientos programados

Para el cálculo de las distancias recorridas se ha utilizado el método de velocidades y tiempos descrito en el apartado de diseño. Esto significa que cuando el bumper se activa o se detecta una distancia pequeña se pasa a la siguiente etapa del programa. No se ha utilizado el nodo odom por el hecho de que almacena la posición de origen al encender el robot y eso ralentiza el desarrollo del prototipo al trabajar por prueba y error.

Teniendo en cuenta los movimientos realizados y los nodos utilizados, el gráfico de los nodos y tópicos activos, de forma simplificada, tendría la forma mostrada en la Figura 9.2. Tal y como se mencionaba en apartados anteriores el programa se subscribe a los tópicos de *bumper* y *distance_sensors* para recibir la información correspondiente y en cambio, publica en los tópicos *cmd_vel* y */arm_controller/command* para mandar órdenes a lo Robotino y al brazo manipulador respectivamente.

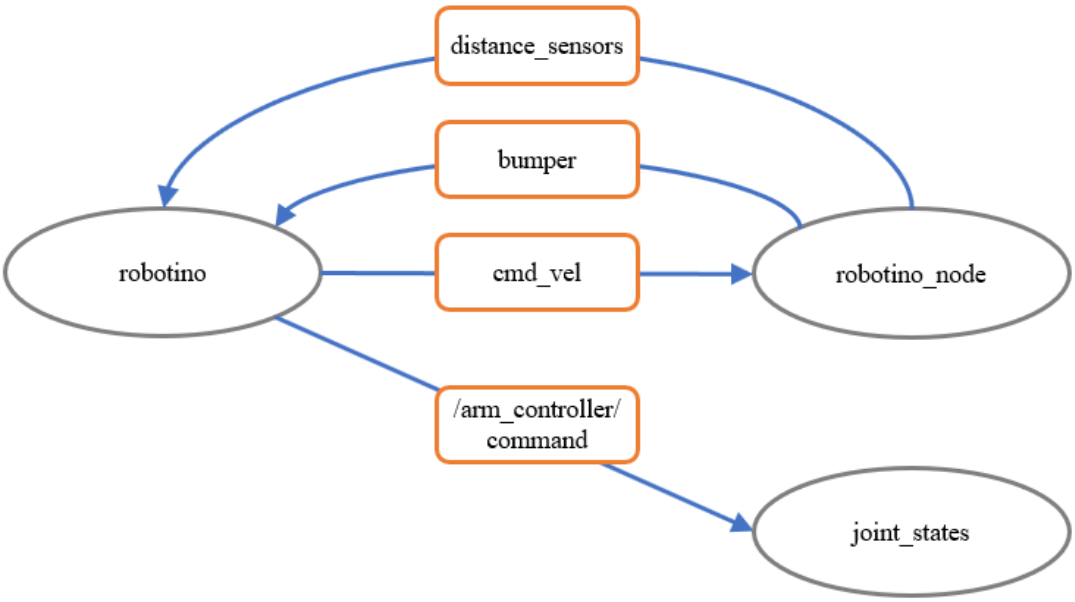

Figura 9.2 – Configuración simplificada de los nodos activos en el programa creado

10. Plan de Trabajo

Teniendo el diseño del proyecto definido se procederá en el siguiente apartado a describir y detallar el plan de trabajo especificando el equipo encargado de llevar a cabo el proyecto y las fases y tareas de las que este se compone.

10.1. Descripción del equipo

Para comenzar a describir el plan de trabajo se mencionarán en primer lugar los integrantes de equipo encargado de completar el proyecto y el cargo que cada uno de ellos ocupa (ver Tabla 10.1).

Tabla 10.1 – Integrantes y cargos del equipo de proyecto

Nombre	Cargo
OSKAR CASQUERO OYARZABAL	Director de proyecto
MAIALEN GONZÁLEZ JAIO	Ingeniero junior
JULEN CUADRA GÓMEZ	Ingeniero junior
ASIER ALONSO TEJEDA	Gestor del proyecto (Administrativo)

10.2. Descripción de Fases y Tareas

Una vez descrito el equipo que formará parte del proyecto, se diferenciarán las fases y tareas que serán necesarias cumplimentar para completarlo.

Para ello, se ha dividido el plan de proyecto en cinco paquetes de trabajo. Cada paquete de trabajo tendrá definido una duración máxima y una serie de entregables y tareas. En la Tabla 10.2 se pueden observar cada uno de los paquetes de trabajo definidos con sus correspondientes características.

Tabla 10.2 – Paquetes de trabajo que conforman el proyecto

Paquete de trabajo	Título	Comienzo	Final	Entregables
WP1	Gestión del Proyecto	Día 1	Día 112	
WP2	Estudios previos	Día 1	Día 5	D2.1
WP3	Montaje y Configuración de los equipos	Día 5	Día 53	D3.1, D3.2
WP4	Formación	Día 53	Dia 74	D4.1, D4.2, D4.3
WP5	Desarrollo del proyecto	Día 74	Día 102	D5.1
WP6	Documentación del proyecto	Día 102	Día 112	D6.1

Los entregables definidos en algunos de los paquetes de trabajo consisten en lo siguiente:

- **D2.1:** Informe sobre los estudios de alternativas y descripción de la solución final
    - Fecha de entrega: Día 5
- **D3.1:** Informe sobre el montaje y configuración del Robotino desde el punto de vista del hardware
    - Fecha de entrega: Día 20
- **D3.2:** Informe sobre el montaje y configuración del brazo manipulador desde el punto de vista del hardware
    - Fecha de entrega: Día 40
- **D4.1:** Informe sobre la estructura general de ROS
    - Fecha de entrega: Día 57
- **D4.2:** Redacción de la documentación correspondiente a la arquitectura de funcionamiento del Robotino y la correlación con ROS
    - Fecha de entrega: Día 64
- **D4.3:** Redacción de la documentación sobre el funcionamiento del brazo manipulador y la relación con ROS
    - Fecha de entrega: Día 74
- **D5.1:** Documentación de cada uno de los tópicos accesibles en el Robotino y el brazo manipulador
    - Fecha de entrega: Día 77
- **D6.1:** Documentación completa de proyecto realizado
    - Fecha de entrega: Día 112

Además de estos entregables, se definirán una serie de hitos a cumplir, Tabla 10.3, para controlar la organización del proyecto y probar que se están cumpliendo los plazos estimados.

Tabla 10.3 – Hitos de la planificación del proyecto

N.º de hito	Título	Fecha
H1	Estudio de alternativas entregado	Día 5
H2	Informe sobre el Robotino entregado	Día 20
H3	Informe sobre el brazo manipulador entregado	Día 40
H4	Instalaciones de software realizados	Día 53
H5	Informe sobre ROS entregado	Día 64
H6	Desarrollo de programas completado y aceptado	Día 102
H7	Entrega de la documentación escrita	Día 112

A continuación, se procederá a explicar la razón de ser de cada uno de los paquetes de trabajo y a definir las tareas a realizar en cada paquete.

WP1: Gestión de proyecto

El objetivo de este paquete de trabajo consistirá en la correcta organización y ejecución del proyecto para lo que será necesario coordinar el trabajo a realizar para completar los entregables e hitos que se han impuesto. Para lograr este objetivo se organizarán reuniones quincenales en las que se requerirá la presencia de todos los integrantes del equipo. En dichas reuniones se tratarán los problemas que surjan durante el proyecto y los elementos necesarios para seguir progresando.

Responsables: O. Casquero, A. Alonso

Participantes: Todos.

WP2: Estudios Previos.

La razón de ser de esta etapa es el conocimiento de los recursos disponibles para llevar a cabo el proyecto. Para ello será necesario evaluar cada uno de los requerimientos especificados al inicio del proyecto y ponerlos en contexto en el entorno en el que se trabajará.

Responsable: M. González.

Participantes: Todos.

T2.1: <u>Estudio de alternativas</u>

El principal objetivo de esta tarea consiste en conocer los requerimientos desde los que se parte y, en primer lugar, identificar los elementos que quedan por especificar. Una vez hecho eso, será necesario valorar las diferentes posibilidades de las que se dispone para poder definir de la forma más adecuada posible cada uno de los elementos. Finalmente, será necesario conocer cómo se combinarán e implementarán los equipos.

Responsable: O. Casquero.

Participantes: Todos.

Resultados: Informe sobre los estudios de alternativas y descripción de la solución final (D2.1).

Recursos: No se necesitarán recursos específicos.

Duración: 5 días. Desde el día 1, hasta el día 5

WP3: Montaje y configuración de los equipos

Como ya se ha mencionado anteriormente, en este proyecto se dispondrá de dos elementos principales: Robotino y el brazo manipular. Por lo tanto, a lo largo de esta fase será necesario realizar el montaje y configuración inicial de cada uno de los equipos.

Responsable: M. Gonzalez

Participantes: Todos

T3.1: <u>Montaje y análisis del hardware de Robotino</u>

Esta tarea consistirá principalmente en el montaje del Robotino y configuración de todas las conexiones iniciales. Será necesario además asignar una dirección IP para la correcta distinción del equipo dentro del entorno de trabajo. En este contexto, se analizará el hardware de Robotino y se identificarán los distintos elementos que componen en sistema: controladores, sensores, CPU, etc.

Responsable: M. González.

Participantes: J. Cuadra, A. Alonso.

Resultados: Informe sobre el montaje y configuración del Robotino desde el punto de vista del hardware (**D3.1**)

Recursos: Estación de montaje, equipo del Robotino, estación de investigación

Duración: 15 días. Desde el día 5, hasta el día 20.

T3.2: Montaje y configuración del brazo manipulador

Al igual que en la Tarea 3.1, será necesario realizar el montaje, en este caso, del brazo manipulador WidowX Robot Arm para lo que se deberán seguir las pautas indicadas en el manual de montaje que proporciona Trossen Robotics [12]. Sin embargo, antes de realizar el montaje se tendrán que identificar cada uno de los servomotores y configurar el controlador Arbotix-M Robocontroller.

Responsables: M. González, J. Cuadra.

Participantes: A. Alonso

Resultados: Informe sobre el montaje y configuración del brazo manipulador desde el punto de vista del hardware (**D3.2**)

Recursos: Estación de montaje, equipo del brazo manipulador, estación de investigación

Duración: 20 días. Desde el día 20, hasta el día 40.

T3.3: Instalación de ROS en todos los equipos

Una vez realizados los montajes correspondientes a los diferentes equipos, se continuará con su configuración para que todos ellos dispongan del framework robótico ROS Kinetic. Para llevar a cabo esta tarea se debe comprender la arquitectura de funcionamiento y comunicación de cada elemento y una vez hecho esto identificar los diferentes programas que se deben instalar. Al tratarse ROS de un framework de acceso abierto para la solución de problemas de compatibilidad que surjan se recurrirá a la comunidad que lo respalda.

Responsables: O. Casquero, M. González.

Participantes: Todos.

Resultados esperados: Implementación del framework robótico ROS en todos los equipos en los que sea necesario.

Recursos: Estaciones de desarrollo.

Duración: 10 días. Desde el día 40, hasta el día 50.

T3.4: Instalación de MoveIt

Cuando se disponga del framework robótico ROS en todos los equipos necesarios, se precisará la instalación de la interfaz gráfica MoveIt (Rviz) en el ordenador de trabajo. La razón de ser de esta

tarea es que dicha interfaz se utilizará posteriormente en una primera aproximación para la compresión del funcionamiento del brazo manipulador y después, para su planificación de trayectorias.

Responsables: M. González, J. Cuadra.

Participantes: A. Alonso.

Resultados esperados: Funcionamiento adecuado del programa MoveIt en el ordenador de trabajo

Recursos: Estaciones de desarrollo

Duración: 3 días. Desde el día 50, hasta el día 53.

WP4: Formación

Una vez llegados a este paquete de trabajo, los equipos han sido montados y configurados y en todos ellos se ha instalado ROS. Por lo tanto, el siguiente paso a dar está relacionado con la formación de los integrantes del equipo de proyecto. Para poder desarrollar funcionalidades para los robots se deben comprender lenguajes de programación y estructuras de comunicación.

Responsable: O. Casquero

Participantes: Todos

T4.1: Conocimiento de Linux a nivel usuario

La meta de esta tarea consiste en que los integrantes del equipo comprendan el funcionamiento a nivel usuario de Linux. Para ello se les formará de forma que sepan navegar a través de la consola y conozcan los comandos básicos que más se utilizan.

Responsable: A. Alonso, M. González.

Participantes: J. Cuadra.

Resultados esperados: Capacidad de navegar a través de la terminal y utilización de comandos básicos

Recursos: Estación de desarrollo, plataforma Linux Ubuntu 16.04, taller de formación

Duración: 4 días. Desde el día 53, hasta el día 57.

T4.2: <u>Comprensión de la estructura organizativa y comunicación de ROS</u>

El objetivo de esta tarea consiste en analizar ROS de forma genérica para así llegar a comprender que elementos son fundamentales en la comunicación. Esta tarea tiene una gran importancia ya que en esos elementos se basará la funcionalidad que se deberá programar.

Responsable: M. González.

Participantes: Todos.

Resultados: Informe sobre la estructura general de ROS (**D4.1**)

Recursos: Estación de desarrollo, plataforma ROS.

Duración: 7 días. Desde el día 57, hasta el día 64.

T4.3: <u>Aprendizaje del lenguaje de programación Python</u>

Tal y como se ha venido explicando en apartados anteriores el código para implementar la funcionalidad que tienen que realizar los robots debe ir programado en lenguaje Python. Para poder llegar a completar la programación será necesario que los integrantes del equipo de proyecto adquieran conocimientos de programación en Python.

Responsable: O. Casquero.

Participantes: Todos

Recursos: Estación de desarrollo, herramientas de desarrollo de software, cursos y talleres de formación

Duración: 10 días. Desde el día 64, hasta el día 74.

WP5: Desarrollo del proyecto

Llegados a esta fase del proyecto, queda realizar el diseño de bajo nivel de éste. Para este momento ya se conoce la estructura hardware tanto del Robotino como del brazo manipulador y se tiene instalado el framework robótico ROS en todos los equipos listo para utilizarse. Por lo tanto, a base de prueba y error se deberán ir conociendo cada una de las posibilidades de funcionamiento tiene cada equipo y realizando programas que las implementen.

Responsable: A. Alonso

Participantes: Todos.

T5.1: Identificación de los tópicos y nodos de los que se dispone en los equipos

La primera tarea a realiza, por tanto, en esta fase será identificar y analizar cada uno de los nodos de los equipos. Además, se deberán conocer todos los recursos en lo referente a tópicos de los que se dispone para controlar y manipular los robots, es decir, hay que identificar los tópicos disponibles y conocer la estructura de sus mensajes para lo que será necesario desglosarlos.

Responsable: M. González.

Participantes: Todos.

Resultados: Documentación de cada uno de los tópicos accesibles en el Robotino y el brazo manipulador (**D5.1**).

Recursos: Estación de desarrollo, herramientas de desarrollo de software

Duración: 3 días. Desde el día 74, hasta el día 77.

T5.2: Diseño de programas para Robotino

En esta fase, habrá que diseñar los programas correspondientes para la manipulación y el control de la plataforma móvil. Se probará la utilización de todos los tópicos para evaluar cuáles de ellos se ajustan mejor al proyecto actual.

Responsable: M. González.

Participantes: J. Cuadra, A. Alonso.

Resultados esperados: Movimiento de la plataforma móvil cumpliendo los requisitos de la trayectoria impuesta

Recursos: Estaciones de desarrollo, Robotino, herramientas de desarrollo de software, área para el movimiento de la plataforma

Duración: 10 días. Desde el día 77, hasta el día 87.

T5.3: Diseño de programas para el brazo

Al igual que se ha hecho con la plataforma móvil, en esta tarea se deberá lograr llevar a cabo una trayectoria con el brazo manipulador. Al no estar el brazo sensorizado se deberá tener mayor cuidado a la hora de realizar movimientos de aproximación por lo que se tendrán que considerar y valorar cada una de las posiciones a programar a través de la plataforma MoveIt.

Responsables: M. González, J. Cuadra.

Participantes: A. Alonso.

Resultados esperados: Seguimiento de la trayectoria programada con el brazo manipulador.

Recursos: Estaciones de desarrollo, brazo manipulador WidowX Robot Arm, herramientas de desarrollo de software, área de movimiento del brazo

Duración: 10 días. Desde el día 87, hasta el día 97.

T5.4: <u>Manipulación del Robotino y del brazo de forma conjunta</u>

Probada la capacidad de controlar y manipular cada uno de los robots por separado, el objetivo de esta tarea consistirá en combinar ambas funcionalidades en un solo programa. Se deberá realizar un programa que coja un objeto de un punto A y lo deposite en otro punto B.

Responsable: M. González.

Participantes: Todos

Resultados esperados: Correcto funcionamiento del movimiento conjunto de la plataforma móvil y el brazo manipulador. Creación de un repositorio github en el que se compartan los programas escritos.

Recursos: Estación de desarrollo, Robotino, brazo manipulador WidowX Robot Arm, herramientas de desarrollo de software.

Duración: 5 días. Desde el día 97, hasta el día 102.

WP6: Documentación del proyecto

La última fase del proyecto consiste en la cumplimentación de un documento explicativo de todo lo realizado en él. El documento deberá ser claro y conciso. Además, tiene que ser aprobado por el director de proyecto. La razón para realizar este documento es que el proyecto realizado pueda servir de guía o base en futuros proyectos.

Responsable: O. Casquero.

Participante: M. González.

Resultados: Documentación completa de proyecto realizado (**D6.1**).

Recursos: No se necesitan recursos específicos para este apartado.

Duración: 10 días. Desde el día 102, hasta el día 112.

11. Diagrama de Gantt

EDT	Nombre de tarea	Duración
Fase 1	**Gestión de Proyecto**	**112 días**
Fase 2	**Estudios Previos**	**5 días**
T2.1	Estudio de alternativas	5 días
Fase 3	**Montaje y configuración de los equipos**	**48 días**
T3.1	Montaje y análisis del hardware de Robotino	15 días
T3.2	Montaje y configuración del brazo manipulador	20 días
T3.3	Instalación de ROS en todos los equipos	10 días
T3.4	Instalación de MoveIt	3 días
Fase 4	**Formación**	**21 días**
T4.1	Conocimineto de Linux a nivel usuario	4 días
T4.2	Comprensión de la estructura organizativa y comunicación de ROS	7 días
T4.3	Aprendizaje del lenguaje de programación, Python	10 días
Fase 5	**Desarrollo del proyecto**	**28 días**
T5.1	Identificación de los tópicos y nodos de los que se dispone en los equipos	3 días
T5.2	Diseño de programas para Robotino	10 días
T5.3	Diseño de programas para el brazo	10 días
T5.4	Manipulación del Robotino y del brazo de forma conjunta	5 días
Fase 6	**Documentación del proyecto**	2 sem.
H1	Estudio de alternativas entregado	0 días
H2	Informe sobre el Robotino entregado	0 días
H3	Informe sobre el brazo manipulador entregado	0 días
H4	Instalaciones de software realizados	0 días
H5	Informe sobre ROS entregado	0 días
H6	Desarrollo de programas completado y aceptado	0 días
H7	Entrega de la documentación escrita	0 días

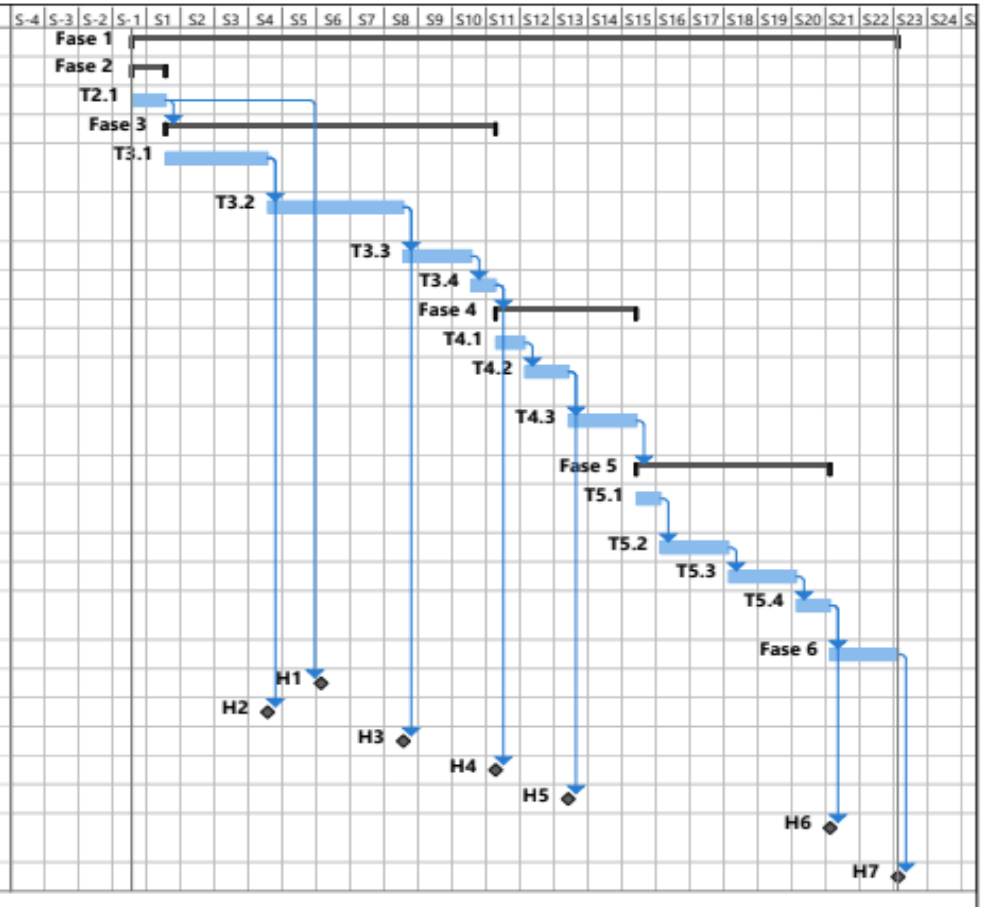

Figura 11.1 – Diagrama Gantt del proyecto

12. Aspectos Económicos

Una vez que se ha definido en qué consistirá el proyecto y la duración y fases de este, se procederá a valorar los aspectos económicos derivados de su consecución. Para hacer esto se calcularán dos presupuestos distintos.

El primer presupuesto considera los costes de desarrollo del proyecto, es decir, su total se obtiene de las horas necesarias por el equipo de proyecto para comprender el funcionamiento de los elementos del sistema y ser capaces de desarrollar una funcionalidad.

Por otro lado, el segundo presupuesto calculará su total de los costes de ejecución de un proyecto, es decir, se trata del presupuesto necesario para una vez asimilados todos los conceptos necesarios, desarrollar una funcionalidad especifica. Como en este caso se requieren unos resultados y unas funcionalidades concretas deberán ajustarse los parámetros de los movimientos a ellas y por lo tanto será necesario el uso de los robots. Esto hace que en este segundo presupuesto se valoren los gastos de los robots. En cambio, como en el primer presupuesto se consideraban los costes conceptuales no se aplicarán estos costes.

Analizando los conceptos incluidos en el primer presupuesto (ver Tabla 12.1) se incluyen como partidas de horas internas a cada uno de los integrantes del equipo de proyecto. El número de horas es elevado debido al hecho de que serán las necesarias para analizar y comprender cada uno de los componentes del proyecto y ser capaces de realizar una funcionalidad. En el caso de las amortizaciones solo se incluye el uso de un ordenado porque como ya se ha explicado tanto ROS como Python son sistemas operativos y lenguajes de acceso abierto por lo que no tienen ningún gasto de licencia.

El presupuesto de desarrollo del proyecto "Control de un brazo manipulador de 5 ejes de libertad sobre una plataforma Robotino" asciende a la cantidad de *diecisiete mil seiscientos cincuenta y nueve euros con noventa y cinco céntimos*.

Tabla 12.1 – Presupuesto de desarrollo

Concepto	Unidades	Coste Unitario	N.º unidades	Coste total	Total
Horas internas					14.500,00 €
Director de proyecto	horas	60,00 €	50	3.000,00 €	
Ingeniero junior	horas	35,00 €	200	7.000,00 €	
Gestor	horas	30,00 €	150	4.500,00 €	
Amortizaciones					70,00 €
Ordenador	horas	0,20 €	350	70,00 €	
Gastos					25,00 €
Material				25,00 €	
Costes directos					14.595,00 €
Costes indirectos	10%				1.459,50 €
Subtotal					16.054,50 €
Imprevistos	10%				1.605,45 €
TOTAL					**17.659,95 €**

Si se analiza ahora el segundo de los presupuestos (ver Tabla 12.2) se observa que en este caso es mucho inferior principalmente debido a la partida de horas internas. Esto se debe a que una vez comprendidos los conceptos y se necesita un menor número de horas para programar una funcionalidad concreta. Sin embargo, tal y como se ha explicado antes en este presupuesto se han de incluir el precio de los robots encargados de ejecutar dicha funcionalidad.

El presupuesto para la ejecución del proyecto "Control de un brazo manipulador de 5 ejes de libertad sobre una plataforma Robotino" asciende a la cantidad de *diecisiete mil novecientos noventa y nueva euros con veintiún céntimos.*

Tabla 12.2 – Presupuesto ejecutado

Concepto	Unidades	Coste Unitario	N.º unidades	Coste total	Total
Horas internas					3.525,00 €
Director de proyecto	horas	60,00 €	5	300,00 €	
Ingeniero junior	horas	35,00 €	75	2.625,00 €	
Gestor	horas	30,00 €	20	600,00 €	
Amortizaciones					70,00 €
Ordenador	horas	0,20 €	350	70,00 €	
Gastos					11.280,38 €
Material				25,00 €	
Robotino			1	9.766,02 €	
WidowX Robot Arm			1	1.489,36 €	
Costes directos					14.875,38 €
Costes indirectos	10%				1.487,54 €
Subtotal					16.362,92 €
Imprevistos					1.636,29 €
TOTAL					**17.999,21 €**

13. Conclusiones

En este último apartado del Trabajo Fin de Grado se recogerán las principales conclusiones que se han extraído al completarlo:

1. ROS es un framework robótico robusto que permite la integración de robots de diferentes empresas alrededor de una misma funcionalidad.
2. ROS tiene el soporte fuerte de una comunidad de usuarios y desarrolladores que constituye una gran fuente de ayuda en la resolución de problemas.
3. Como el software se desarrolla por otros usuarios no se garantiza un funcionamiento correcto.
4. El lanzamiento anual de distribuciones nuevas de ROS hace que los problemas de compatibilidad aumenten.
5. Se puede modificar el código de los equipos de acuerdo con las necesidades requeridas.
6. El programa MoveIt y su interfaz gráfica Rviz son aptos para la visualización de trayectorias y posiciones, pero para la programación, los lenguajes de alto nivel se comprenden y escriben con mayor facilidad.
7. Es necesario un alto conocimiento de lenguajes de programación para hacer un prototipado rápido de las funcionalidades.
8. El nodo odom del Robotino presenta mayor precisión, pero no se recomienda su uso para la programación a base de prueba y error en un entorno de aprendizaje. Sin embargo, su uso presenta mayor interés y ventajas a la hora de realizar una aplicación de mayor nivel ya que permite la posibilidad del movimiento mediante mapas.
9. Si se pretende calcular las distancias de giro del Robotino mediante velocidad y tiempo se comete un error variable en cada caso que debe ser corregido a base de prueba y error.
10. El brazo manipulador debería estar sensorizado para en el caso más simple, poder detectar si la pinza se encuentra frente a un objeto o no.
11. Los sensores de distancia del Robotino tienen un pequeño error por lo que se recomienda que sus valores se parametricen a base de prueba y error.

Extraídas las conclusiones de este proyecto, se pretende continuar aumentando las funcionalidades de estos equipos en futuros proyecto. En este sentido, destaca la utilización del nodo *odom* del Robotino y su aplicación con mapas del entorno de trabajo y el desarrollo de aplicaciones que hagan uso de la cámara para obtener imágenes en tiempo real.

14. Bibliografía

[1] D. Trotta and P. Garengo, “Industry 4 . 0 Key Research Topics : A Bibliometric Review,” *2018 7th Int. Conf. Ind. Technol. Manag.*, no. March 2018, pp. 113–117, 2019.

[2] J. A. Araúzo, J. J. Laviós, and J. J, “Programación y Control de Sistemas de Fabricación Flexibles: un Enfoque Holónico,” *Rev. Iberoam. Automática e Informática Ind.*, vol. 12, no. 1, pp. 58–68, 2015.

[3] Steven Nahmias, “Production and Operations Analysis, 6th Revised edition,” *London McGraw Hill High. Educ.*, 2013.

[4] M. Narciso, M. A. Piera, and A. Guasch, *A MODELING AND SIMULATION APPROACH: TOWARDS TRUE MANUFACTURING FLEXIBILITY*, vol. 35, no. 1. IFAC, 2002.

[5] G. Alfonso Aberasturi, “PUESTA EN MARCHA DEL CONTROL DE ROBOTS DE TRANSPORTE,” 2018.

[6] C. F. Mora and O. P. Calder, “Robotización y transformación del empleo,” 2018.

[7] “WidowX Strength Chart (grams).”

[8] “Robotino® View 2 ES.”

[9] N. X. P. Semiconductors, “LPC2377/78 Single-chip 16-bit/32-bit microcontrollers; 512 kB flash with ISP/IAP, Ethernet, USB 2.0, CAN, and 10-bit ADC/DAC,” no. October, 2013.

[10] I. Core, “SOM-5788,” pp. 7–8, 2012.

[11] “Diseño e implementación de un sistema de teleoperación y evasión de obstáculos en un robot móvil mediante el uso del entorno ROS (Robotic Operating System).,” no. September 2014, 2018.

[12] T. Robotics, “WidowX MKII Robot Arm,” *https://www.trossenrobotics.com/productdocs/assemblyguides/widowx-robot-arm-mk2.html*.

15. ANEXO I : Manual del Usuario

El objetivo de este manual es explicar de forma sencilla los pasos necesarios a dar para la puesta en marcha del Robotino y del brazo. Por lo tanto, se asumirá que las instalaciones necesarias de ROS Kinetic y de los paquetes necesarios para el correcto funcionamiento de los equipos se ha realizado previamente.

Debido a que se necesita trabajar en diferentes terminales se utilizará el programa Terminator para la gestión y visualización de las pestañas.

15.1. Configuración inicial

En cada terminal será necesario acceder como usuario root para lo que hay que introducir el siguiente comando:

```
$ ssh root@192.168.1.160
Contraseña: $ dorp6
```

Una vez dentro del Robotino será necesario acceder al espacio de trabajo:

```
$ cd /home/robotino/catkin_ws/
```

En la primera terminal que se acceda a este directorio será necesario introducir la siguiente línea de comando:

```
$ route add default gw 192.168.1.1
```

Para comprobar que la ruta se ha creado correctamente se debe introducir el siguiente comando que muestra todas las rutas de nuestro equipo:

```
$ route -n
```

Posteriormente, en la primera terminal al igual que en el resto hay que ejecutar los siguientes comandos:

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

Esta última línea consiste en un compilador, por lo tanto, analizará todo el equipo para comprobar que todos los paquetes están bien instalados. Sin embargo, no realizará las compilaciones del código escrito en Python. Este se comprobará al ejecutar el propio programa.

Una vez que se haya compilado el equipo y no haya errores se procede a la activación del maestro y de los nodos necesarios tanto para el Robotino como en el brazo. Para ello será necesario ejecutar dos archivos de tipo launch en terminales diferentes.

Comando de la primera terminal para la activación del Robotino:

```
$ roslauch robotino_node robotino_simple_node.launch
```

Comando de la segunda terminal para la activación del brazo:

```
$ roslauch widowx_arm_controller widowx_arm_controller.launch
```

15.2. Ejecutar un programa

Ahora que están activos los nodos y el maestro se pueden ejecutar los códigos programados en una tercera terminal. Por lo tanto, tras realizar de nuevo los pasos de la configuración inicial, se podrá ejecutar el programa principal introduciendo el siguiente comando:

```
$ rosrun paquete_proyecto base_robotino.py
```

Al ejecutar este comando la plataforma móvil empezará a moverse y a ejecutar el código.

Si en lugar de esta aplicación se quiere ejecutar cualquiera del resto de los códigos creados los pasos a realizar serían los siguientes:

1. Acceder al directorio del código desde el catkin_ws:

```
$ cd /src/paquete_proyecto/scripts/
```

2. Mostrar los ejecutables disponibles:

```
$ ls
```

3. Modificar o leer el programa:

```
$ nano <nombre_del_ejecutable>.py
```

4. Ejecutar el programa:

```
$ rosrun paquete_proyecto <nombre_del_ejecutable>.py
```

15.3. Crear nuevas funcionalidades

En este apartado se detallarán los pasos a realizar si se desea crear una nueva aplicación para el conjunto de la plataforma con el brazo. En caso de que se desee modificar una aplicación ya existente ya sea para reajustar los parámetros o introducir otros comandos, los pasos a realizar han sido definidos en el apartado anterior.

Si se quiere crear una nueva funcionalidad en Python desde cero en primer lugar hay que posicionarse en el directorio en el que se desee guardar la aplicación, partiendo del catkin_ws:

```
$ cd /src/paquete_proyecto/scripts/
```

Una vez posicionados dentro del directoria hay que ejecutar la siguiente línea:

```
$ touch <nombre_del_ejecutable>.py
```

Para comprobar que el ejecutable se ha creado correctamente hay que introducir el siguiente comando y observar que entre los resultados obtenidos hay un archivo con el nombre que se ha escrito.

```
$ ls
```

El nombre del archivo estará escrito en color gris por lo que lo siguiente es darle permisos de ejecución al archivo:

```
$ sudo chmod 775 <nombre_del_ejecutable>.py
```

Una vez hecho esto, si se vuelve a ejecutar el anterior comando el color del nombre del archivo pasará a ser verde.

Otra de las opciones para crear o modificar un archivo consiste en realizar una copia de un archivo ya existente. En este caso, no haría falta darle permisos al documento. La línea de comando sería la siguiente:

```
$ cp <nombre_del_ejecutable_orginal>.py <nombre_de_la_copia>.py
```