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

DISEÑO Y DESARROLLO DE UN PAQUETE ROS PARA EL CONTROL DEL BRAZO MANIPULADOR TINKERKIT BRACCIO

Documento 1 - Memoria

Alumna: Gracia Zarraga, Nerea
Director: Casquero Oyarzabal, Oskar
Codirector: Orive Revillas, Darío

Curso: 2018-2019

Fecha: 2019-07-22

RESUMEN

El brazo robótico Tinkerkit Braccio está formado por seis servomotores controlados por una placa Arduino Mega 2560. Hasta el momento, la programación de los movimientos articulados del brazo se ejecuta desde el programa de Arduino. La idea de este proyecto es conseguir hacer el mencionado control utilizando la conocida plataforma de ROS, para lo que será necesario crear un *driver* que comunique esta plataforma con la mencionada placa.

Palabras clave: robot, brazo, ROS, Arduino

LABURPENA

TinkerKit Braccio beso robotikoa sei serbomotorez osatuta dago, Arduino Mega 2560 plaka batek kontrolatuak. Gaur egunera arte, besoaren mugimendu artikulatuen programazioa Arduino programatik exekutatu izan da. Proiektu honen helburua kontrol hori ROS plataforma ezagunetik egin ahal izatea da, horretarako *driver* bat sortu beharko da plataforma honen eta aipatutako plakaren artean komunikazioa ahalbidetuko duena.

Gako-hitzak: robota, besoa, ROS, Arduino

ABSTRACT

The robotic arm TinkerKit Braccio is made up of six servomotors which are controlled by Arduino Mega 2560 board. Until nowadays, the jointed movement of the arm is programmed by Arduino platform. The finality of this project is to do this control of movements with the well-known platform ROS, for which will be necessary to create a driver which will communicate the platform with the mentioned board.

Keywords: robot, arm, ROS, Arduino

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1.INTRODUCCIÓN

La adopción de la automatización industrial en empresas, en un comienzo, se enfocó en sistemas aislados básicos. Sin embargo, a principios de los años 80, las empresas comenzaron a integrar estos sistemas, creando un conjunto, ya que la competencia industrial de hoy en día requiere que la fabricación incorpore todos los elementos necesarios para que la empresa funcione como una entidad única. Dos de las razones más concluyentes para adoptar tecnologías, son los altos costes salariales y la baja disponibilidad de mano de obra. [1] Automatizando tareas repetitivas y/o tareas específicas con necesidad de una alta precisión, se consigue reducir tanto el tiempo como el coste de producción.

A día de hoy, los robots forman una parte muy importante de la automatización. La Asociación de Industrias Robóticas (RIA) define un robot industrial como: "un manipulador multifuncional reprogramable capaz de mover materias, piezas, herramientas, o dispositivos especiales, según trayectorias variables, programadas para realizar tareas diversas". Dentro de la clasificación del robot industrial se encuentra el robot con control por computador. Este control hace uso de un lenguaje específico constituido por las instrucciones que recibirá el robot, con la ventaja de poder crear un programa de aplicación desde una computadora sin tener que utilizar el brazo manipulador. Esto facilita el control de toda una planta ya que podría hacerse desde un mismo ordenador. [2]

2.CONTEXTO

Un brazo robótico puede tener distintas arquitecturas, entre ellas se encuentra el robot con estructura articulada que dispone de movimientos rotacionales. Este tipo de estructura permite ordenar que varios movimientos se ejecuten a la vez, uno por cada articulación.

Un robot de este tipo es el llamado 'TinkerKit Braccio', sacado al mercado por Arduino en 2016. Es de montaje ventajoso, ya que puede ser montado de diversas maneras a gusto del usuario, según la utilidad que este le quiera dar. También tiene la capacidad de manejar diferentes objetos, desde algo tan simple como sujetar una cámara fotográfica para hacer una foto, hasta mantener un panel solar que seguirá la trayectoria del sol.[3]

El control de esos movimientos mencionados está a cargo de la placa 'Arduino Mega 2560', a la que están conectados los seis servomotores del brazo gracias al *shield*, compatible con Arduino, que incluye Braccio. La placa es la encargada de ejecutar un programa que dará la orden de los movimientos al brazo, utilizando la plataforma de Arduino. [4]

Es aquí donde entra este trabajo de fin de grado (TFG) que consistirá en buscar una alternativa para la programación. El objetivo es controlar el brazo robótico desde la plataforma ROS (Robot Operating System). Esta plataforma es proveedora de librerías y herramientas que son de ayuda para los desarrolladores de software a la hora de crear aplicaciones robóticas.[5] El problema es que no existe un *driver* que ponga en contacto la plataforma ROS con el robot TinkerKit Braccio, y conseguir esto será la base del proyecto.

3.OBJETIVOS Y ALCANCE DEL TRABAJO

El objetivo principal de este trabajo es el diseño y desarrollo de un *driver* que haga capaz la comunicación entre el brazo robótico TinkerKit Braccio y la plataforma de ROS, el cual se creará con base en la utilización de nodos funcionales escritos en lenguaje Python. Para conseguir este objetivo final habrá que cumplir los siguientes objetivos secundarios:

- Conocimiento básico del sistema operativo de código abierto Ubuntu de Linux.
- El usuario deberá conocer el funcionamiento de la plataforma de ROS, comandos básicos, creación de un paquete y creación y edición de nodos entre los más importantes.
- Conocimiento de programación en Python para crear los nodos funcionales en ROS.
- Integración del objeto serial en el nodo programado en lenguaje Python.
- Conocimiento de la programación básica en la plataforma de Arduino.
- Diseñar un protocolo propio para mandar la orden de los movimientos de todas las articulaciones en un mismo *array*.
- Diseñar un entorno de validación que permita probar el software sin necesidad de usar el robot.
- Lograr la posición deseada de las seis articulaciones del robot en una sola orden.

4.BENEFICIOS QUE APORTA EL TRABAJO

A la hora de implementar robots en una empresa es una condición indispensable la facilidad de que puedan comunicarse entre ellos sean o no de la misma marca o modelo. Por tanto, será importante conseguir que todos tengan posible la comunicación con la misma plataforma, aportando así beneficios puramente técnicos. Facilitar la comunicación entre los dispositivos los hará más accesibles y permitirá un mejor resultado final, puesto que se podrá conseguir un producto de igual precisión en un tiempo menor. De esta manera, se elimina la dificultad de utilizar dos robots diferentes con acciones complementarias, como podría ser el ejemplo de un brazo robótico encima de una plataforma transportadora. En este ejemplo la acción principal sería transportar un objeto de un sitio a otro, donde el brazo sería el encargado de la sujeción del objeto, y la plataforma en cambio, del transporte del mismo. Es claramente necesaria una comunicación entre los dos robots para ejecutar la acción deseada correctamente. Por tanto, crear un *driver* que comunique el brazo robótico TinkerKit Braccio con la plataforma de ROS, será meramente beneficioso para todo aquel que quiera darle a este robot la funcionalidad de complementar a otro robot o viceversa.

5.DESCRIPCIÓN DE REQUERIMIENTOS

5.1 Requisitos del sistema.

Para el control del brazo robótico de este proyecto, estarán impuestas varias especificaciones previas al inicio del trabajo. A estas especificaciones se las denomina requisitos del sistema:

- Se va a utilizar el brazo robótico manipulador TinkerKit Braccio, fabricado por Arduino. Se compone de seis articulaciones de movimiento rotacional, movidas cada una por un servomotor.
- En comunicación con los servomotores del robot estará la placa Arduino Mega 2560, también creada por Arduino.
- Para el control de los movimientos se va a utilizar la plataforma de ROS, la cual simplifica el control de diferentes robots, facilitando la complementación entre ellos.
- La plataforma de ROS ha sido diseñada principalmente para Ubuntu, por tanto, el sistema operativo que se utilizará sera Ubuntu 16.04 .

5.2 Requisitos funcionales.

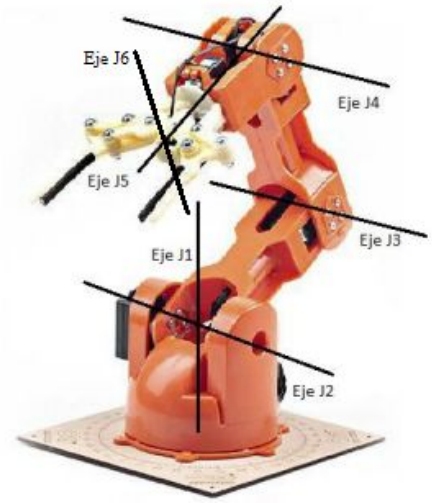


Figura 1: Brazo robótico TinkerKit Braccio con seis ejes articulados

Por otro lado, están los requisitos funcionales, que son las especificaciones que ha pedido el cliente, es decir, las posiciones deseadas para el robot. El requisito, en este caso, es el movimiento de un objeto. Este tendrá que describir una trayectoria específica evitando los obstáculos hallados en el camino. Este requisito se cumplirá ordenándole al sistema que adopte las siguientes posiciones:

- El brazo partirá de una posición llamada “Posición de seguridad” en la que se coloca automáticamente al iniciar el robot. Posición: [90,45,180,180,90,10]
- La primera posición que le indicaremos tomar será la de acercarse al objeto deseado y abrir la pinza. Posición: [0,90,0,0,0,10]
- Para la segunda posición, se requiere que el brazo cierre la pinza lo justo para agarrar el objeto con algo de fuerza. Posición: [0,90,0,0,0,50]
- En la tercera posición se gira el brazo hacia el sitio elegido para dejar el objeto. Posición: [180,15,0,90,0,50]
- Por último, se abre la pinza para soltar el objeto. Posición: [180,15,0,90,0,10]

6.ANÁLISIS DE ALTERNATIVAS

6.1 Alternativa del lenguaje de programación

Teniendo en cuenta los requisitos asignados, las alternativas posibles surgen en el lenguaje de programación. En este proyecto se han tenido en cuenta solamente dos tipos: Python y C++.

Los criterios utilizados para hacer una buena elección, son los siguientes:

- Tiempo de aprendizaje: Puesto que la implementación de los robots es cada día más rápida, es necesario que el proceso de aprendizaje dure lo menos posible, y esto se dará con una menor dificultad de la sintaxis del lenguaje.
- Robustez del código: La robustez, es la capacidad del código de hacerle frente a un error mientras este se está ejecutando. Se requiere un buen comportamiento del sistema aun haciendo todas las pruebas necesarias.
- Tipos de solución: Este apartado da importancia a los tipos de solución posibles que permite el lenguaje. Facilitará la programación la posibilidad de tener diferentes salidas al mismo problema, ya que cada programador tiene su forma de planteárselo.

Analizando estos criterios en cada uno de los lenguajes:

- Python:
  - Es un lenguaje de lectura fácil, es decir, la escritura que se usa tiene una forma explícita. Por tanto, es de sencillo aprendizaje tanto para gente con experiencia como sin ella. No requiere un tiempo desmesurado el aprenderlo, aun no habiendo programado nunca.
  - No tiene una robustez especialmente buena.
  - Contiene una gran variedad de soluciones para un mismo problema por lo que nos permite programar de distintas maneras llegando al mismo resultado.
- C++ :
  - Tiene un lenguaje muy amplio por lo que la dificultad de sintaxis resulta mayor. Esto concluye en un tiempo de aprendizaje alto y en este caso no contaremos con la facilidad de que un no-programador lo aprenda sin problema.

- Es un lenguaje bastante robusto. También es estable y rápido a la hora de ejecutar.
- Da una variedad más restringida de solución de problemas, por lo que para llegar al resultado existirán menos posibilidades de diseño.

Para una comparativa más clara pueden reducirse los datos anteriores a una tabla:

Tabla 1: Resumen de características de los lenguajes

Lenguaje de programación	**Tiempo de aprendizaje**	**Robustez del código**	**Tipos de solución**
Python	Corto	Mala	Muchos
C++	Largo	Buena	Pocos

Finalmente, haciendo uso de otra tabla comparativa y una ponderación de los criterios elegidos, tomaremos la decisión sobre que lenguaje de programación utilizar en el proyecto. La ponderación se hará aportando entre 1 y 3 puntos a las características, dándole un 1 al de menor importancia para el proyecto y un 3 al de máxima importancia. Se le da una mayor importancia al tiempo de aprendizaje necesario puesto que repercute en aspectos tanto económicos como sociales. Le daremos el segundo rango de importancia a la variedad de soluciones posibles ya que esto iría unido a lo anterior, cuantas más soluciones posibles más fácil será aprender a programar. Por último, por el tipo de proyecto, le daremos la menor importancia a la robustez del código.

Tabla 2: Ponderación de los criterios para la elección del lenguaje

Criterio	Tiempo de aprendizaje	Robustez del código	Tipos de solución
Ponderación	3	1	2

Tabla 3: Decisión final del lenguaje

	Tiempo de aprendizaje	**Robustez del código**	**Tipos de solución**	**Puntuación total**
Ponderación	3	1	2	
Python	3	1	2	14
C++	1	3	1	8

Observando la tabla llegamos a la conclusión de que la mejor elección de lenguaje de programación para este proyecto es Python.

6.2 Alternativas de la distribución de ROS

Aun cuando la plataforma de ROS ha sido una condición impuesta, dentro de esta plataforma se encuentran diferentes distribuciones sobre las que se hará un análisis decidiendo finalmente la mejor opción. Los criterios utilizados para la elección en este caso son:

- Soporte : Cuanto mayor sea el tiempo de soporte, habrá más actualizaciones de la distribución y más tiempo para crear una comunidad mayor, lo que será útil a la hora de solventar los problemas que surjan.
- Comunidad : A la hora de recibir ayuda con dudas que puedan surgir y soluciones ante problemas que irán apareciendo durante el proyecto, será importante la amplitud de la comunidad que respalde cada distribución.
- Compatibilidad con Ubuntu 16.04 : Puesto que la utilización de Ubuntu ha sido impuesta junto a los requisitos del sistema, habrá que comprobar la compatibilidad existente con cada uno de los paquetes ya que con esto podrá ahorrarse tiempo en instalaciones y actualizaciones.

Analizando estos criterios en cada una de las alternativas:

- ROS Kinetic : Es una versión de ROS publicada en 2017 y que se extenderá hasta el 2021, lo que supone un amplio periodo de soporte, que a su vez permite formar una amplia comunidad con una gran disposición de paquetes. Por otro lado, para su instalación necesita que su espacio de trabajo sea Ubuntu 16.04.
- ROS Lunar : Esta versión también fue lanzada en 2017, pero esta solo se extenderá hasta 2019, por tanto a partir de ese año no se dispondrá de nuevas actualizaciones. Esta versión también debe ser instalada sobre Ubuntu 16.04.
- ROS Melodic : Es la versión lanzada en 2018 que se ampliará hasta 2023, la última hasta el momento. Al ser tan reciente no dispone de muchos recursos, su comunidad está en crecimiento. Además para su uso es necesario disponer de Ubuntu 18.04.

Para una comparativa más clara pueden resumirse las ideas anteriores en una tabla:

Tabla 4: Resumen de características de las versiones de ROS

Version de ROS	Soporte	Comunidad	Compatibilidad con Ubuntu 16.04
ROS Kinetic	Hasta 2021	Amplia	Sí
ROS Lunar	Hasta 2019	Reducida	Sí
ROS Melodic	Hasta 2023	En crecimiento	No

Para tomar una decisión acertada al igual que en el apartado anterior se hará uso de una ponderación para los criterios, asignando valores del 1 al 3 según la importancia que tenga cada criterio para el proyecto.

Tabla 5: Ponderación de los criterios para la elección de distribución de ROS

Criterio	Soporte	Comunidad	Compatibilidad con Ubuntu 16.04
Ponderación	1	2	3

Haciendo uso de la ponderación asignada en la tabla 5, se formará una nueva tabla en la que puntuaremos en su totalidad cada criterio, comparándolos finalmente para hacer la elección más optima.

Tabla 6: Decisión final de la distribución de ROS

	Soporte	**Comunidad**	**Compatibilidad con Ubuntu 16.04**	**Puntuación total**
Ponderación	1	2	3	
ROS Kinetic	3	3	3	18
ROS Lunar	1	1	3	12
ROS Melodic	3	1	1	8

Observando los resultados de la tabla 6 se puede concluir que la mejor decisión es instalar la versión ROS Kinetic.

7.DESCRIPCIÓN DE LA SOLUCIÓN

Una vez especificados los requisitos y elegidos tanto el lenguaje de programación como la versión de ROS con la que se va a trabajar, en este apartado se pretende describir la metodología a seguir para llegar al objetivo de nuestro proyecto, el cual era controlar el brazo robótico desde la plataforma ROS. Para la descripción de la solución propuesta, quedan próximamente especificados el hardware necesario para llevar el proyecto a cabo y el software de los elementos a utilizar.

7.1 Hardware

En esta primera figura se pueden identificar las diferentes partes del hardware del sistema:

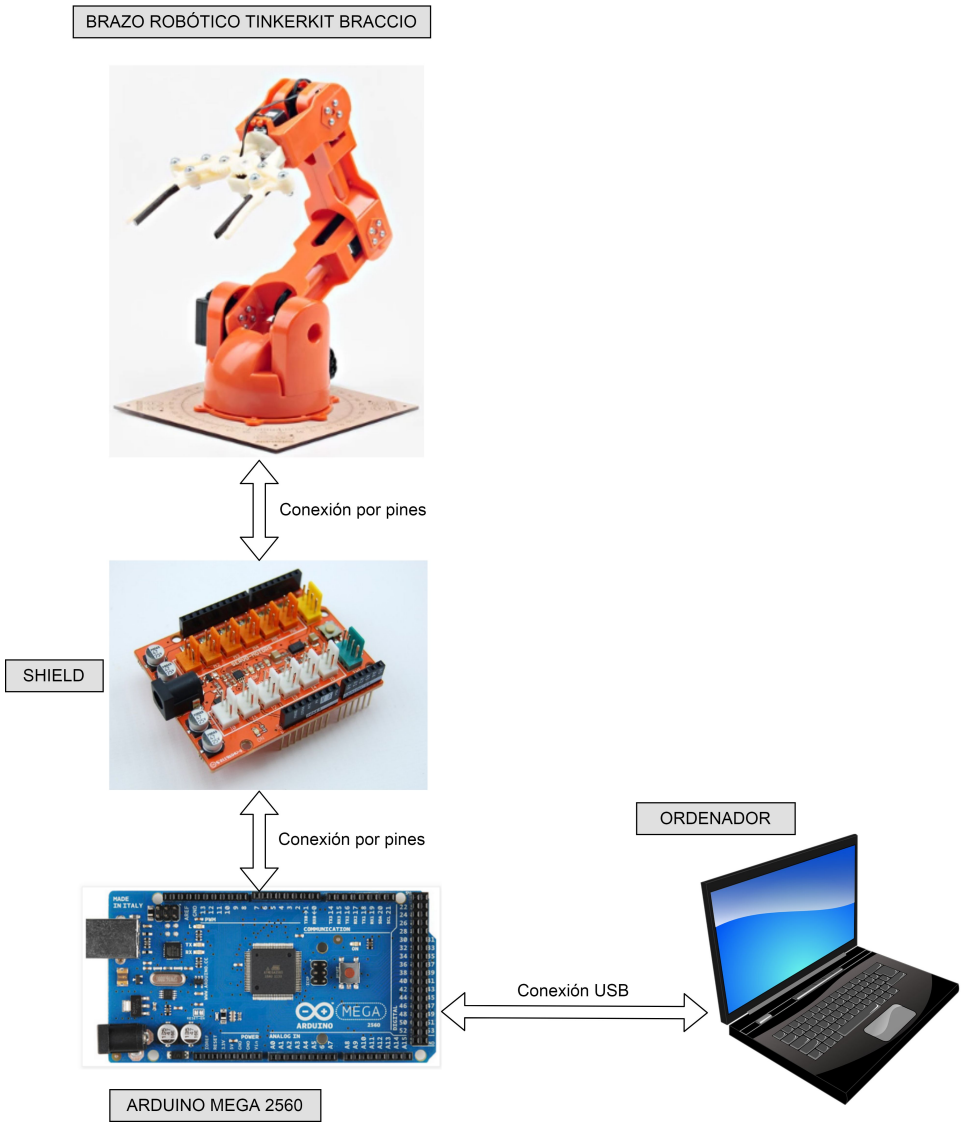


Figura 1: Entorno de ejecución

Pueden presenciarse un ordenador, desde el que tomará parte el programador; una placa Arduino Mega 2560, que ejecutará las ordenes en los servos del brazo; y, por último, el brazo robótico TinkerKit Braccio, el cual contiene seis servomotores que serán los encargados de realizar el movimiento programado. La conexión entre el ordenador y la placa de Arduino se hará físicamente mediante un cable USB como se indica en la figura. En cambio, la conexión entre la placa y el robot se hará gracias a que el robot contiene una *shield* compatible con Arduino, que permite hacer la conexión sin necesidad de soldaduras, siendo la *shield* un conjunto de pines que encaja en la placa de Arduino. [6]

Esta *shield* es necesaria por dos razones:

- La placa de Arduino no dispone de la potencia necesaria para mover el brazo y esto se lo proporciona la *shield* puesto que consta de alimentación propia.
- Cada uno de los servomotores necesita la conexión de tres pines: uno para la alimentación, otro para conectarse a tierra y el último para las señales de control. La placa de Arduino solo es capaz de proporcionar una conexión de tierra y una de alimentación, lo que en este caso no sería suficiente ya que disponemos de seis servomotores. Además, el cable de conexión que poseen los servos es comúnmente uno que incluye los tres pines por lo que para conectarlo a Arduino habría que modificar este cable y en cambio, la *shield* se compone de entradas para recibir este tipo de conexiones.

Figura 2: Servomotor y shield para TinkerKit Braccio

Por otro lado, cabe destacar que el robot no se añadirá al proyecto hasta el último momento ya que no será necesario para el diseño del *driver*. Para este diseño el entorno utilizado constará además de un ordenador conectado a la placa de Arduino y un *shield* conectado a esta placa, de un adaptador TTL-USB con el que se conectará la placa de Arduino con otro ordenador para la comprobación de la correcta llegada de los datos desde el primer ordenador a la placa. Es necesaria la utilización de este adaptador puesto que la placa solo tiene un puerto serie hardware y de esta manera conseguiremos emular un puerto serie software posibilitando dos conexiones instantáneas a la placa.

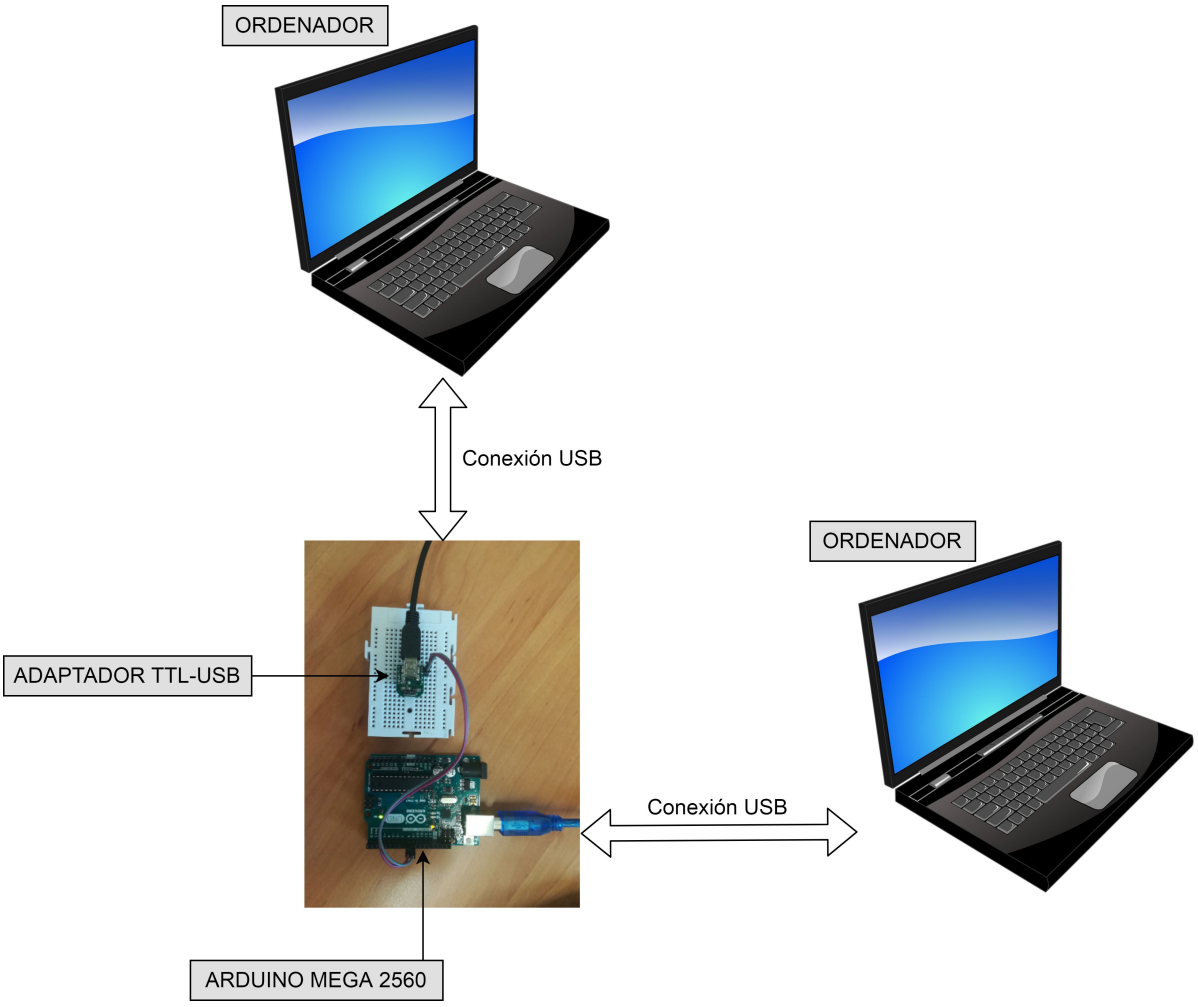


Figura 3: Entorno de desarrollo

7.2 Software

En la siguiente figura se describe el diagrama de bloques correspondiente al software*:*

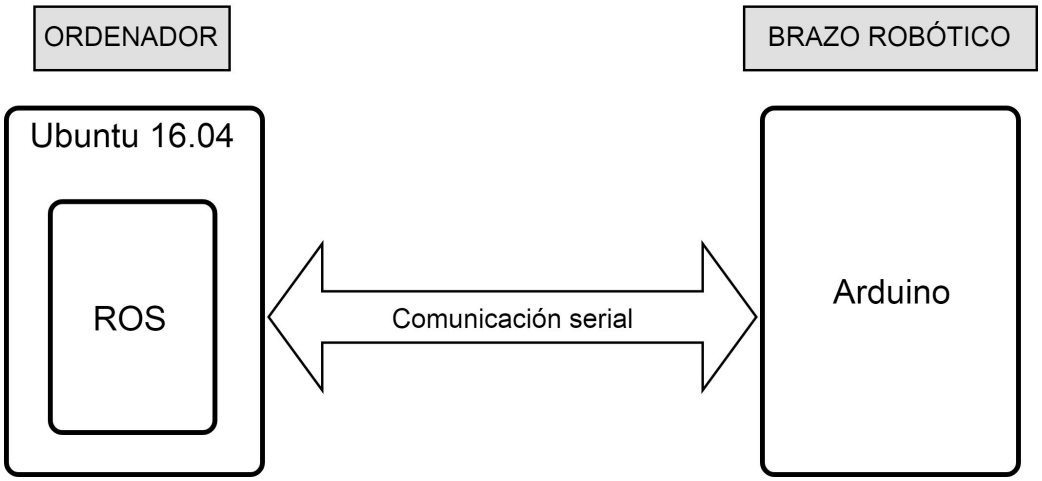


Figura 1: Diagrama de bloques del software

En esta figura se aprecian dos partes bien separadas, por una parte, está el software del ordenador y por la otra, el del brazo.

- Por la parte del ordenador, observamos Ubuntu 16.04, la cual es una distribución del sistema operativo Linux e irá instalada en el ordenador para poder trabajar en la misma. Sobre esta, tenemos ROS, el cual sera nuestro espacio de trabajo para el diseño del software del robot. Por medio de esta plataforma, crearemos los nodos funcionales necesarios:
  - Un nodo publicador, que será necesario para hacer pruebas. Este nodo describirá los movimientos deseados y los mandará a un tópico.
  - Un nodo suscriptor, que estará suscrito al mismo tópico en el que publica el publicador. Este nodo, será el encargado de ordenar los ángulos en un *array* para facilitar la lectura en Arduino. Para ello se diseñará un protocolo propio, cuya estructura sea: una 'S' como carácter de inicio de trama y una '#' como carácter de introducción al ángulo. Por último, se le tendrá que añadir el objeto serial para que pueda mandar la información hasta Arduino.
- Por parte del robot, más específicamente en la placa de Arduino, se utilizará el editor de Arduino en el que se diseñará un pequeño programa para recibir los datos

del nodo suscriptor. A este programa le llegará una cadena de datos como la descrita en el punto anterior y tendrá que ser capaz de separar la cadena e identificar el ángulo de cada articulación. Seguidamente, mediante una librería que conecta Arduino con TinkerKit Braccio, el programa será capaz de controlar los servomotores y se conseguirá la posición deseada.

Para una mayor comprensión de esta explicación, puede resumirse todo en un simple esquema, que podría añadirse dentro de la imagen anterior, puesto que es un desglose del software interno de ROS.

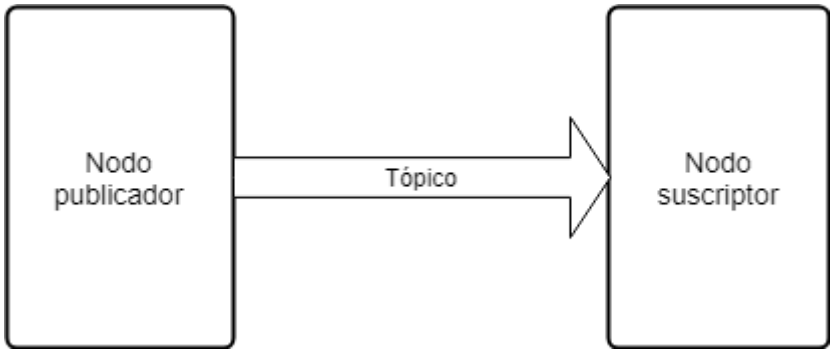


Figura 2: Desglose del software interno de ROS

8.DISEÑO

En este apartado del documento se describirán con detalle tanto los elementos de hardware que se han utilizado como el software de estos elementos, especificando el tipo de comunicación entre ellos y concretando el tipo de mensaje que ejecutará cada uno.

8.1 Hardware.

El hardware del ordenador no tomará parte en el proyecto, por lo tanto, este punto se centrará solamente en describir el brazo robótico TinkerKit Braccio, el elemento principal del proyecto.

TinkerKit Braccio es un brazo manipulador que se vende desmontado para que el usuario tenga libertad de montarlo a su antojo. Para este proyecto se ha montado en su forma más común, utilizando los seis servomotores que proporciona y por tanto montándolo de manera que sea capaz de mover al mismo tiempo seis articulaciones diferentes. En la siguiente imagen se muestran varias posiciones posibles del brazo, siendo la primera la elegida.

Figura 1: Diferentes montajes de TinkerKit Braccio

La estructura del robot se compone por:

- Una base circular que se mueve sobre un eje vertical y que puede rotar entre 0 y 180 grados.
- Una articulación identificada como hombro que une la base con la pieza siguiente y permite movimientos de entre 15 y 165 grados.

- Una articulación llamada codo, la cual une dos piezas semiidénticas y permite movimientos de entre 0 y 180 grados.
- Otra articulación llamada en este caso muñeca vertical, que une una pieza algo más pequeña que las anteriores en una articulación que se mueve entre 0 y 180 grados.
- Una quinta articulación, a la que se acoplará la pinza, identificada como muñeca giratoria y que permitirá un giro de entre 0 y 180 grados sobre un eje en dirección de las pinzas.
- Por último, está la pinza en sí, a la que se le asignará un valor de 10 grados cuando esté completamente abierta y un valor de 73 grados cuando esté cerrada.

A parte de los límites de movimiento y nombres de las articulaciones, el kit aporta las siguientes características y especificaciones: [7]

Tabla 7: Características y especificaciones del TinkerKit Braccio

Distancia de funcionamiento máxima	80 cm
Altura máxima	52 cm
Anchura de la base	14 cm
Ancho de la pinza	9 cm
Peso total	0,792 kg
Capacidad de carga máxima/peso a 32 cm distancia de funcionamiento	150 g
Peso máximo en la base de configuración Braccio	400 g

El movimiento de cada articulación está controlado por un servomotor. Los hay de dos tipos, dos unidades del tipo SR 311 y cuatro unidades del tipo SR 431. Los servomotores son catalogados por numero ascendente al igual que las articulaciones siendo M1 el motor en la base y M6 el que abre y cierra la pinza.

Tabla 8: Resumen de los servomotores del brazo

Nombre del servomotor	Nombre de la articulación	Movimiento permitido (en angulos)	Tipo
M1	Base	0-180	SR 431
M2	Hombro	15-165	SR 431
M3	Codo	0-180	SR 431
M4	Muñeca vertical	0-180	SR 431
M5	Muñeca giratoria	0-180	SR 311
M6	Pinza	10-73	SR 311

El kit en el que viene el brazo robótico también incluye una placa *shield* que permite a los servomotores conectarse directamente con la placa de Arduino. Está compuesto con una serie de pines que encajan directamente en la placa, y una serie de puertos preparados para conectar cada uno de los servomotores. Estos puertos se componen de tres pines, dos de ellos conectados a las conexiones de alimentación y tierra y un tercero conectado al control del correspondiente servomotor. Observando la siguiente figura pueden apreciarse el nombre de cada uno de los motores (M1-M6) alado de los puertos mencionados, protegidos por un plástico de color naranja. [8]

Figura 2: Shield para TinkerKit Braccio

Tabla 9: Características del Braccio Shield

Tensión de funcionamiento	5V
Consumo de potencia	20 mV
Corriente máxima	1,1 A desde los conectores M1 a M4 750 mA desde los conectores M5 a M6

La placa de Arduino Mega 2560 será la encargada de mandar las órdenes de movimiento a los servomotores, órdenes que se procesarán a través de un programa diseñado en la aplicación de Arduino. Esta placa tiene las siguientes características descritas en la tabla 10. [9]

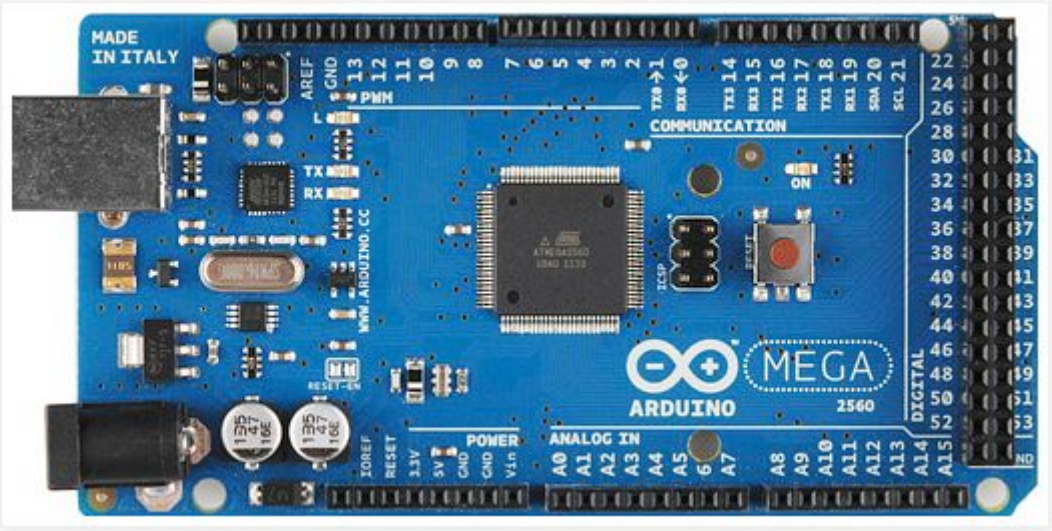


Figura 3: Placa Arduino Mega 2560

Tabla 10: Características de la placa Arduino Mega 2560

Nombre del microcontrolador	AT Mega 2560
Tensión de trabajo	5 V
Tensión de entrada (recomendada)	7-12 V
Tensión de entrada (límite)	6-20 V
Pines digitales I/O	54 (de los cuales 15 proporcionan salida PWM)
Pines de entradas analógicas	16
DC corriente por pin I/O	20 mA
DC corriente por pin 3.3 V	50 mA
Memoria Flash	256 kB de los cuales 8 se usan por el bootloader
SRAM	8 kB
EPROM	4 kB
Velocidad del reloj	16 MHz
Largo	101.52 mm
Ancho	53.3 mm
Peso	37g

Por último, queda mencionar el adaptador TTL-USB que se usará para emular un puerto serie software, ya que el único puerto hardware que contiene Arduino estará conectado al ordenador recibiendo datos del suscriptor. Este puerto serie software se utilizará para mostrar en una nueva pantalla los datos que recibe la placa de Arduino y así hacer comprobaciones durante las pruebas para el diseño.

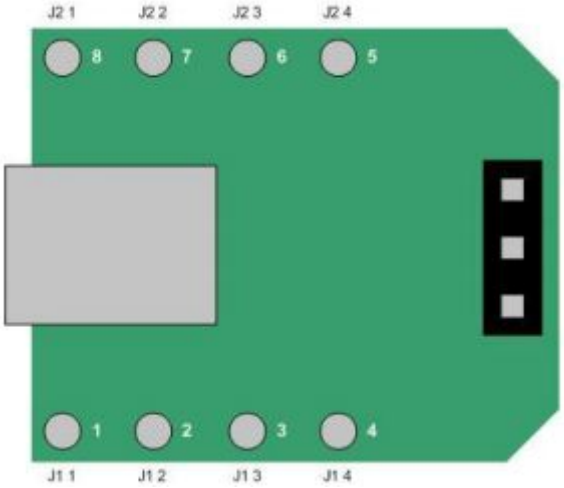


Figura 4: Esquema de conexiones del adaptador TTL-USB

Tabla 11: Conexiones del adaptador TTL-USB

Número de pin	Conexión	Nombre	Descripción
1	J1-1	GND	Pin de 0V de potencia
2	J1-2	VCC	Potencia de 5V desde el interface del USB
3	J1-3	CTS#	FT232R CTS pin
4	J1-4	RTS#	FT232R RTS pin
5	J2-4	CBUS1	FT232R CBUS1 pin
6	J2-3	CBUS0	FT232R CBUS0 pin
7	J2-2	RXD	FT232R RXD pin
8	J2-1	TXD	FT232R TXD pin

El adaptador contiene un puerto USB que se usará para hacer la conexión con el nuevo ordenador, y varios pines, donde dos de ellos se usarán como receptor y transmisor, el 7 y el 8 respectivamente. El pin receptor del adaptador (7) se conectará al pin de la placa Arduino que corresponda a la transmisión de datos (el pin número 9) y el pin transmisor (8) se conectará al pin

de la placa que corresponda con la recepción de datos (el pin número 8). De esta manera se completará una nueva conexión vía serie.

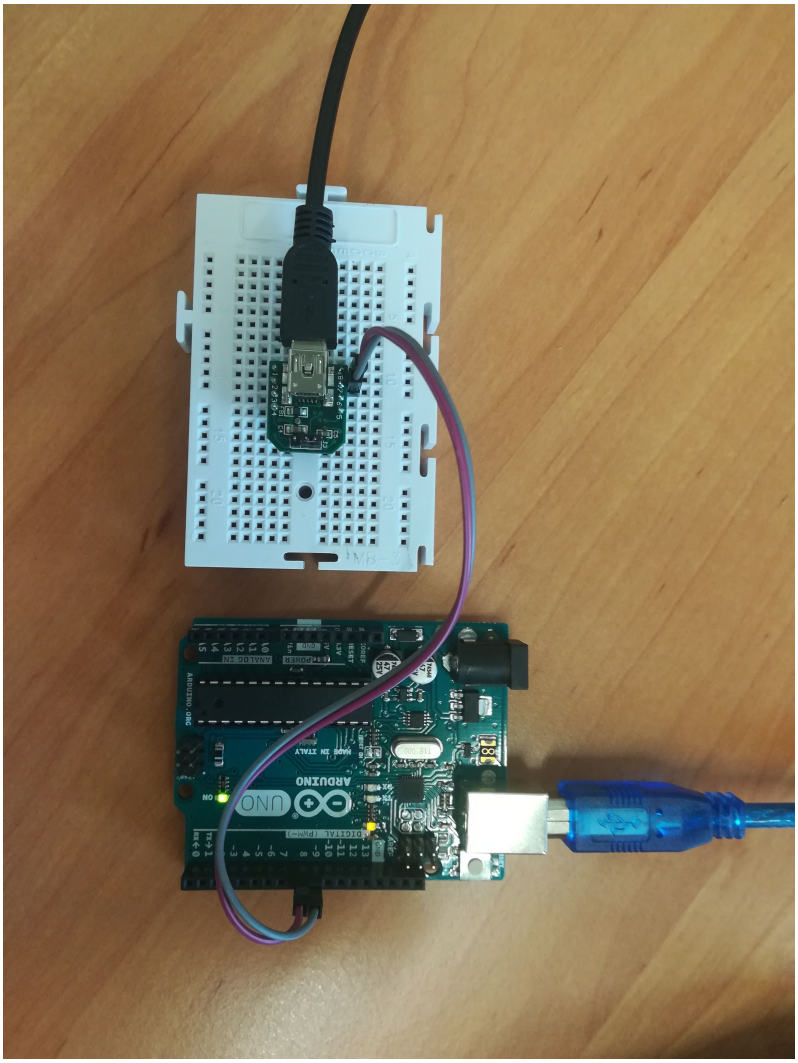


Figura 5: Adaptador TTL-USB conectado a la placa Arduino Mega 2560

8.2 Software

La parte con mayor peso de todo el proyecto se encuentra en el diseño del software. Para ello será importante por una parte conocer el software de ROS, para poder crear los nodos necesarios, y por otra parte habrá que tener también conocimiento del software de Arduino, para poder configurar la recepción de datos en la placa y la comunicación con los servomotores.

8.2.1 ROS

8.2.1.1 Marco teórico.

ROS es un sistema operativo que contiene gran variedad de librerías y permite el diseño de software de robots. Para comprender el uso de esta plataforma en el proyecto, primero se van a explicar detalladamente los conceptos más importantes para su utilización y los comandos básicos que se vayan a utilizar.

- Tipos de archivo
  - Paquetes: Son la unidad más básica de la organización del software y contienen la información de los nodos. Es lo primero que se debe crear cuando se comunican nodos entre si. Dentro del paquete creado se encontrarán todos los códigos de los nodos, será el espacio de trabajo.
  - Mensajes: Son archivos de diferentes tipos que se envían entre nodos. El tipo de mensaje será importante a la hora de crear un nodo o un tópico. Cada tópico solo será capaz de transmitir un tipo de mensaje y los nodos trasmitirán o leerán el tipo de mensaje que se le ordene. Los diferentes tipos de mensajes según su complejidad:
    - Simples: números enteros, números reales o *strings*.
    - Complejos: Por ejemplo, *arrays*, que pueden estar a su vez compuestos de tipos simples o tipos complejos.
- Comunicación
  - Nodos funcionales: Son programas capaces de ejecutar diferentes acciones. Pueden distinguirse dos grupos relacionados entre sí. Esto no quiere decir que todos los nodos tengan que pertenecer a algunos de estos dos grupos, pueden también ejecutarse por separado.
    - Publicadores: Estos nodos son de código secuencial, es decir, cada instrucción sigue a otra anterior. Se les denomina nodos publicadores porque publican mensajes en un tópico, con intención de que un suscriptor los reciba.
    - Suscriptores: Utilizan un código basado en eventos, es decir, su ejecución dependerá de un suceso ocurrido en el sistema que estará definido por el

usuario. Se les denomina nodos suscriptores porque están suscritos a un tópico y esperan a que éste reciba mensajes para poder leerlos.

- Tópicos: Los dos grupos de nodos expuestos anteriormente utilizan este canal para compartir los mensajes. Es de gran importancia que tanto el nodo publicador como el suscriptor estén comunicados al mismo tópico para que exista una comunicación.
- Mensajes: Son los datos que se quieren transmitir entre nodos. En un caso general, el publicador los creará, los escribirá en el tópico y en cuanto lleguen a este, el suscriptor los leerá.
- Maestro: En ROS definido como "ROS Master", es el encargado de permitir la comunicación entre todos los nodos, es decir, si no arrancamos el *master* no será posible tal comunicación. El maestro será el que permita la suscripción de los nodos en el sistema. Una vez creados dentro del sistema, el nodo publicador tendrá que pedir permiso para convertirse en publicador de un tópico y a su vez el suscriptor pedirá permiso para suscribirse al tópico. Una vez dichos permisos han sido concedidos por el Maestro el sistema está listo para funcionar, las comunicaciones entre nodos estarán creadas.

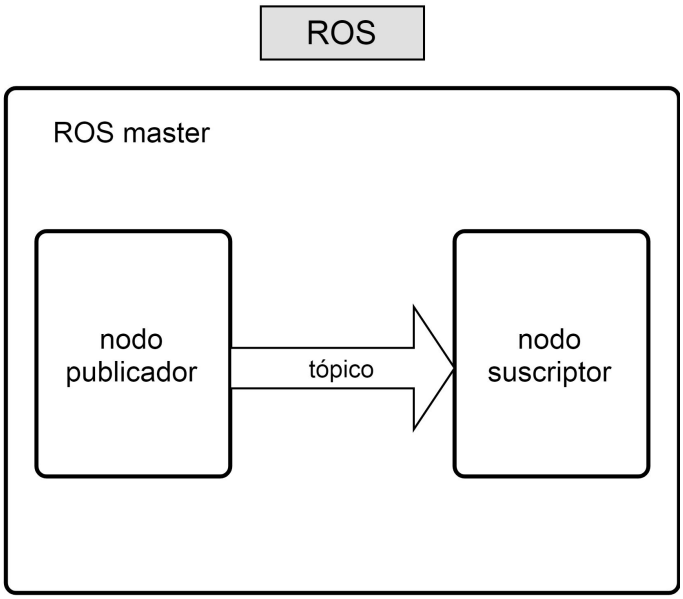


Figura 1: Comunicación interna de ROS

- Comandos útiles para el proyecto
  - Antes de empezar a utilizar algún comando de ROS, para poder acceder a los directorios donde se almacenan los datos de la plataforma, se debe ejecutar en la terminal de Ubuntu el siguiente código:

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

Figura 2: Código necesario para utilizar ROS

  - *roscore*: Una vez que es posible ejecutar cualquier comando de ROS, el primero debe ser siempre este. Esto se debe a que este es el comando que arranca el master. Una vez ejecutado, será necesario abrir una nueva terminal y a esta primera solo se volverá si se quiere detener el ros master, acción que se llevará acabo haciendo 'ctrl+C'. A parte de arrancar el master, este comando también ejecuta un nodo llamado /rosout que estará comunicado con todos los nodos que estén en ejecución.
  - *catkin_create_pkg*: Con este comando crearemos un nuevo paquete, tendremos que especificar el nombre del paquete y a continuación las dependencias que

tendrá (en nuestro caso el tipo de mensaje y la librería de Python, puesto que es la que utilizaremos más adelante en los nodos).

- *catkin_make*: Se utiliza para compilar la carpeta de catkin_ws y se utilizará tanto después de crear un paquete como después de crear un nodo. Este comando se ejecutará siempre estando dentro de la carpeta catkin_ws.

```
cd /home/usuario/catkin_ws/src
catkin_create_pkg [nombre del paquete] [tipo de mensaje] rospy

cd/home/usuario/catkin_ws
catkin_make
```

Figura 3: Código para crear un paquete

- Crear un nodo implica dos pasos muy diferentes:
  - Por un lado, se creará un documento en blanco, que si se hace desde la terminal se utilizará el comando *gedit* seguido del nombre del nodo (ej.: listener.py) con terminación .py ya que se escribirá el código en lenguaje Python. Es necesario que, para dar aviso que a partir de ahí se escribe en lenguaje Python, al principio del documento y antes de escribir el código se escriba lo siguiente:

    #!/usr/bin/env python

  - Por otro lado, es importante hacer el nodo ejecutable, y para ello utilizaremos el comando *chmod.* Por último, no puede ser olvidado ejecutar catkin_make.

```
cd /home/usuario/catkin_ws/src/[nombre de mi paquete]
gedit listener.py

chmod +x listener.py

cd /home/usuario/catkin_ws
catkin_make
```

Figura 4: Crear un nodo ejecutable

- *rosrun*: Este comando será utilizado cada vez que se quiera arrancar un nodo, para ello, será necesario especificar el nombre del paquete donde se encuentra seguido del nombre del nodo que queremos ejecutar.
- *rosed*: Se utiliza para editar un nodo. Al igual que en *rosrun*, se especificará el nombre del paquete en el que se encuentra y el nombre del nodo.
- *rosnode*: Este es un comando con varias utilidades según la orden que le proceda.
  - *rosnode* info [nombre del nodo] : Da información sobre el nodo, como a qué tópicos está suscrito, en qué tópicos publica y el tipo de mensajes que trata.
  - *rosnode* list : Da la lista de nodos que están en ejecución.
  - *rosnode* cleanup : Sirve para resetear la lista anterior
- *rostopic*: Este comando es muy parecido al anterior y asimismo, depende de lo que le proceda. Hay que tener en cuenta que un tópico se creará cuando se cree un publicador y un suscriptor, los cuales lo necesitarán para comunicarse, es decir, se crea automáticamente.
  - *rostopic* list : Muestra la lista de los tópicos en ejecución.
  - *rostopic* type: Muestra el tipo de mensaje que puede contener.
  - *rostopic* echo [nombre del tópico]: Muestra lo que le llega al tópico en ese momento determinado.
- Existe un nodo llamado rqt_graph que muestra un pequeño esquema de los nodos y tópicos que están en ejecución y las uniones que hay entre ellos.

```
rosrun rqt_graph rqt_graph
```

Figura 5: Ejecución de nodo rqt_graph

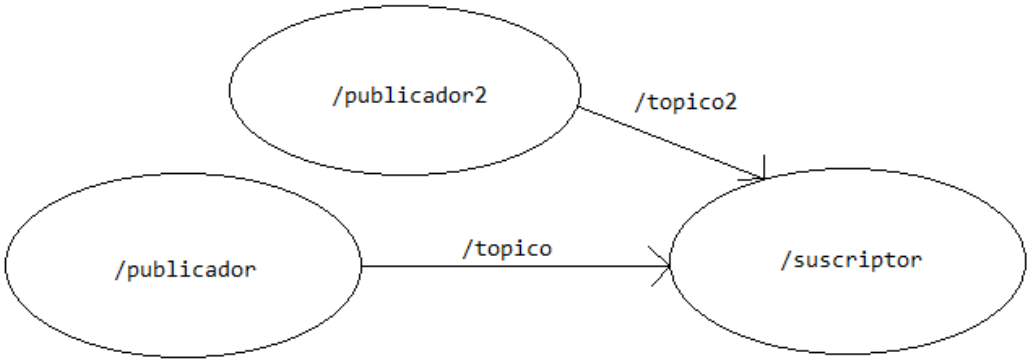


Figura 6: Ejemplo de nodo rqt_graph

En el ejemplo que se visualiza en la figura 6, hay en ejecución un nodo suscriptor suscrito a dos tópicos, 'topico' y 'topico2', asimismo se han ejecutado dos publicadores, el llamado 'publicador' que publica mensajes en 'topico' y 'publicador2' que lo hace en 'topico2'. Deteniendo uno de los nodos publicadores, desaparecería tanto el nodo como su correspondiente tópico. En cambio, al detener el suscriptor, desaparecería tanto el nodo suscriptor como los dos tópicos.

- Al igual que para detener el ROS master, para la detención de cualquier otro nodo, también se ejecutará el comando 'ctrl+C' en la terminal en la que el nodo este abierto, ya que por terminal solo podrá haber un nodo en ejecución.

8.2.1.2 Aspectos prácticos.

Una vez expuestos los aspectos teóricos más importantes para el uso de ROS se pasa a diseñar los nodos necesarios para el proyecto, en los que se incluye un suscriptor y un publicador, existiendo entre ellos un tópico por el que se comunicarán.

- Antes programar los nodos hay que comprender el **tipo de mensaje** con el que se va a trabajar. El mensaje serán los movimientos que tenga que hacer cada una de las articulaciones del brazo robot, y esto se traduce en un mensaje de tipo JointTrajectory. Este es el tipo de mensaje que mandará el publicador, que manejará el tópico y que recibirá el suscriptor. Es un mensaje de estructura compleja, compuesto por tres campos principales:

```
trajectory_msgs/JointTrajectory.msg{
                                    std_msgs/Header header
                                    string[] joint_names
                                    trajectory_msgs/JointTrajectoryPoint[] points
                                   }
```

Figura 1: Estructura del mensaje tipo JointTrajectory

  - header: Una cabecera, en la que se describen tres datos simples. Para este proyecto se dejara este apartado en blanco y ROS lo autorellenará por defecto.

```
std_msgs/Header {
                 uint32 seq
                 time stamp
                 string frame_id
                }
```

Figura 2: Estructura de JointTrajectory/Header

  - Joint_names: Un *array* compuesto de mensajes simples de tipo *string*. Aquí se describirán los nombres de las seis articulaciones del brazo robot.

- JointTrajectoryPoints: Un *array* compuesto de: 4 *arrays* a su vez compuestos de 6 mensajes simples cada uno, uno por cada articulación, y de un mensaje simple que definirá la duración para que concluya el movimiento. Para este proyecto, solo se mandará información del *array* de 'positions' y del mensaje simple 'time_from_start'.

```
trajectory_msgs/JointTrajectoryPoint.msg {
                                         float64[] positions
                                         float64[] velocities
                                         float64[] accelerations
                                         float64[] effort
                                         duration  time_from_start
                                         }
```

Figura 3: Estructura de JointTrajectory/JointTrajectoryPoint

Uniendo todos los datos se consigue la estructura completa del mensaje:

```
trajectory_msgs/JointTrajectory.msg{
                                    header{
                                           uint32 seq
                                           time stamp
                                           string frame_id
                                           }
                                    string[]joint_names
                                    points {
                                             points[1]{
                                                        float64 positions
                                                        float64 velocities
                                                        float64 accelerations
                                                        float64 effort
                                                        duration time_from_start
                                                        }
                                             points[2]{
                                                        float64 positions
                                                        float64 velocities
                                                        float64 accelerations
                                                        float64 effort
                                                        duration time_from_start
                                                        }
                                             (...)

                                             points[n]{
                                                        float64 positions
                                                        float64 velocities
                                                        float64 accelerations
                                                        float64 effort
                                                        duration time_from_start
                                                        }
                                           }
                                 }
```

Figura 4: Estructura completa de JointTrajectory

Es relevante comentar que en 'joint_names' habrá un nombre por cada articulación (6 en el caso que ocupa) y por cada uno de esos se definirá un mensaje simple en cada *array* que contiene 'points[]', es decir, el *array* de 'positions', tendrá 6 mensajes simples y cada uno equivaldrá a la posición deseada para cada articulación. Por tanto, en cada mensaje podrá compartirse al mismo tiempo, más de una posición del robot deseada para diferentes momentos y cada una de ellas se describirá en uno de los 'points[]'. En el caso del proyecto se mandarán solamente los *arrays* que corresponden a la posición y al tiempo. Para la completa comprensión del tipo de mensaje se continuará con un ejemplo simple que

valdrá para el desarrollo del proyecto. Además, como se puede comprobar prestando atención en 'time_from_start', cada posición tardará 3s en ejecutarse.

```
trajectory_msgs/JointTrajectory.msg{
                                    header{}
                                    joint_names["M1","M2","M3","M4","M5","M6"]
                                    points{
                                          points[1]{
                                                    positions[0.0,90.0,0.0,0.0,0.0,10.0]
                                                    time_from_start_3s
                                                   }
                                          points[2]{
                                                    positions[0.0,90.0,0.0,0.0,0.0,50.0]
                                                    time_from_start_6s
                                                   }
                                          points[3]{
                                                    positions[180.0,15.0,0.0,90.0,0.0,50.0]
                                                    time_from_start_9s
                                                   }
                                          points[4]{
                                                    positions[180.0,15.0,0.0,90.0,0.0,10.0]
                                                    time_from_start_12s
                                                   }

                                          }
                                   }
```

Figura 5: Ejemplo de mensaje tipo JointTrajectory

- El **nodo publicador** se diseñará con un código secuencial y escrito en lenguaje Python, para lo que se necesitará importar la librería de Python. Para poder trabajar en este código será necesario también importar la librería del tipo de mensaje con el que se va a tratar, en este caso el tipo JointTrajectory.

```
import rospy
from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint
```

Figura 6: Importación de librerías necesarias para el publicador

Una vez entendida la estructura del mensaje de tipo JointTrajectory será sencillo comprender el código del nodo, puesto que su función será asignar valores a cada campo del mensaje y hacer el envío de este. Antes de nada, se creará el publicador, asociándolo al

tópico y designándole el tipo de mensaje que va a publicar, y seguidamente se inicializará el nodo. El siguiente paso sería dar valor a cada sección del mensaje, en este caso :

- 'header{}' quedará vacío ya que ROS lo autocompletará.
- 'joint_names' se completará con los nombres de las articulaciones:
  jnames = [M1, M2, M3, M4, M5, M6]
- Dentro de 'points[]' sólo se rellenaran los campos 'positions', donde se meterán los *arrays* llamados 'jvalues' que contienen las posiciones de las articulaciones, y 'time_from_start', que extraerá el dato para cada punto del *array* llamado 'jtime'.

  Posición 1: jvalues[1]=[0,90,0,0,0,10]

  Posición 2: jvalues[2]=[0,90,0,0,0,10]

  Posición 3: jvalues[3]=[180,15,0,90,0,50]

  Posición 4: jvalues[4]=[180,15,0,90,0,10]

  jtime = [3,6,9,12]

En el siguiente diagrama, figura 7, pueden verse esquematizados los pasos descritos anteriormente.

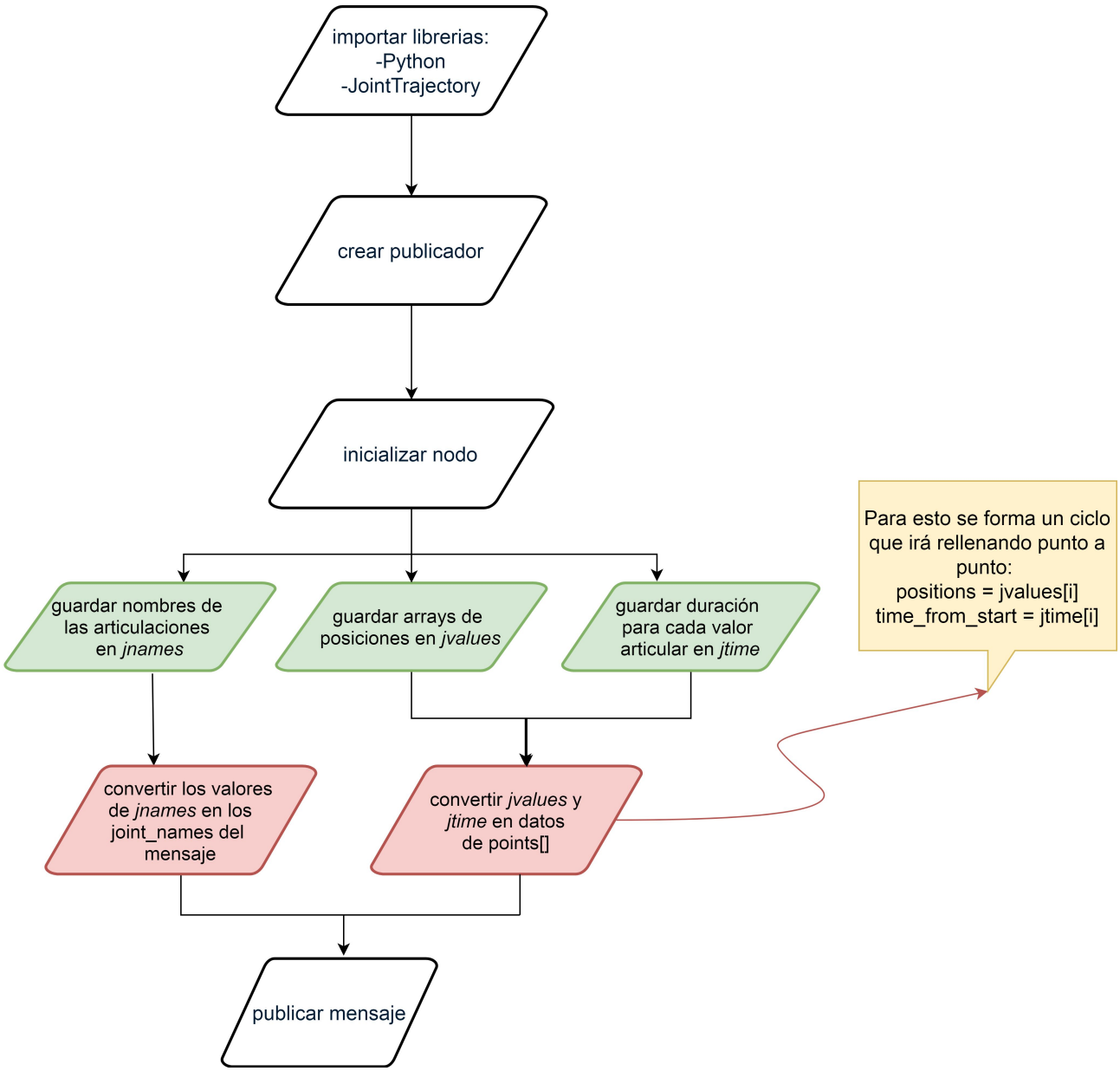


Figura 7: Diagrama del código del publicador

Al ejecutar el nodo publicador y si todo va bien, aparecerán en la terminal los mensajes de la figura siguiente, que serán los que se hayan mandado imprimir al escribir el código. Para hace pruebas, es recomendable ir imprimiendo mensajes por cada paso y finalmente imprimir el mensaje a mandar para comprobar que todo es correcto.

```
header:
  seq: 0
  stamp:
    secs: 1563463078
    nsecs: 276429891
  frame_id: ''
joint_names: [M1,M2,M3,M4,M5,M6]
points:
  -
    positions: [0, 90, 0, 0, 0, 10]
    velocities: []
    accelerations: []
    effort: []
    time_from_start:
      secs: 3
      nsecs:         0
  -
    positions: [0, 90, 0, 0, 0, 50]
    velocities: []
    accelerations: []
    effort: []
    time_from_start:
      secs: 6
      nsecs:         0
  -
    positions: [180, 15, 0, 90, 0, 50]
    velocities: []
    accelerations: []
    effort: []
    time_from_start:
      secs: 9
      nsecs:         0
  -
    positions: [180, 15, 0, 90, 0, 10]
    velocities: []
    accelerations: []
    effort: []
    time_from_start:
      secs: 12
      nsecs:         0
```

Figura 8: Envío de datos desde el publicador

- El **nodo suscriptor** se diseñará con un código basado en eventos. Al igual que en el nodo publicador, el código se escribirá en lenguaje Python y el tipo de mensaje tratado será JointTrajectory, por tanto, necesitará las librerías de Python y de JointTrajectory. Además de estas dos librerías este nodo también necesitará la librería que contiene el objeto serial, para poder integrar la comunicación serial, y la librería del tiempo.

```
import rospy
import time
import serial
from trajectory_msgs.msg import JointTrajectory
```

Figura 9: Importación de librerías necesarias para el suscriptor

Después de haber importado las librerías necesarias hay que llamar al objeto serial, que solo podrá ser llamado una vez. En esta línea de código se especificarán el puerto en el que se conectará la placa de Arduino y la velocidad de transmisión elegida. Es importante que esta velocidad sea la misma en el suscriptor que en el código de Arduino para que exista comunicación entre los dos. En este caso se ha elegido una velocidad de 9600 bps y el puerto por el que se conecta la placa es '/dev/ttyACM0'.

```
arduino = serial.Serial("/dev/ttyACM0",9600)
```

Figura 10: Objeto serial del suscriptor

Una vez especificada la conexión serial, es hora de crear una estructura propia para mandar el mensaje a Arduino de una forma óptima. Para esto se ha decidido crear un *array* que para cada posición comience con una 'S' y a cada ángulo le proceda una '#'. Cuando la cadena formada por una de las posiciones este completa se mandará al Arduino y se esperará el tiempo indicado en 'time_from_start' antes de comenzar a escribir otra cadena y enviarla.

```
def callback(data):
	for point in data.points:
		punto='S'
		time_from_start = point.time_from_start.to_sec();
		for position in point.positions:
			punto += '#'+ str(int(position))
		arduino.write(punto)
		time.sleep(time_from_start)
```

Figura 11: Callback del suscriptor

Por último, se define el suscriptor, lo que significa iniciar el nodo y suscribirlo al mismo tópico donde esta publicando el publicador, a la vez que se le designa el tipo de mensaje JointTrajectory.

```
def suscriptor():
	rospy.init_node('suscriptor',anonymous=True)
	rospy.Subscriber('topic', JointTrajectory, callback)
```

Figura 12: Definición de suscriptor en el código

Al ejecutar el suscriptor no debe aparecer nada en la terminal puesto que hasta que el publicador no publique en el tópico el suscriptor no podrá leer nada. Por tanto, se ejecuta el suscriptor y acto seguido el publicador y si se le ha dado orden al suscriptor de imprimir en pantalla los datos que le va a enviar a Arduino, deberá aparecer lo siguiente:

```
nere@ubuntu:~$ rosrun tfg2b suscriptor.py
suscriptor activo
Cadena de punto a enviar: S#0#90#0#0#0#10
Cadena de punto a enviar: S#0#90#0#0#0#50
Cadena de punto a enviar: S#180#15#0#90#0#50
Cadena de punto a enviar: S#180#15#0#90#0#10
```

Figura 13: Datos a enviar desde el suscriptor a Arduino

En el siguiente diagrama de bloques, figura 14, se describe visualmente el código del suscriptor.

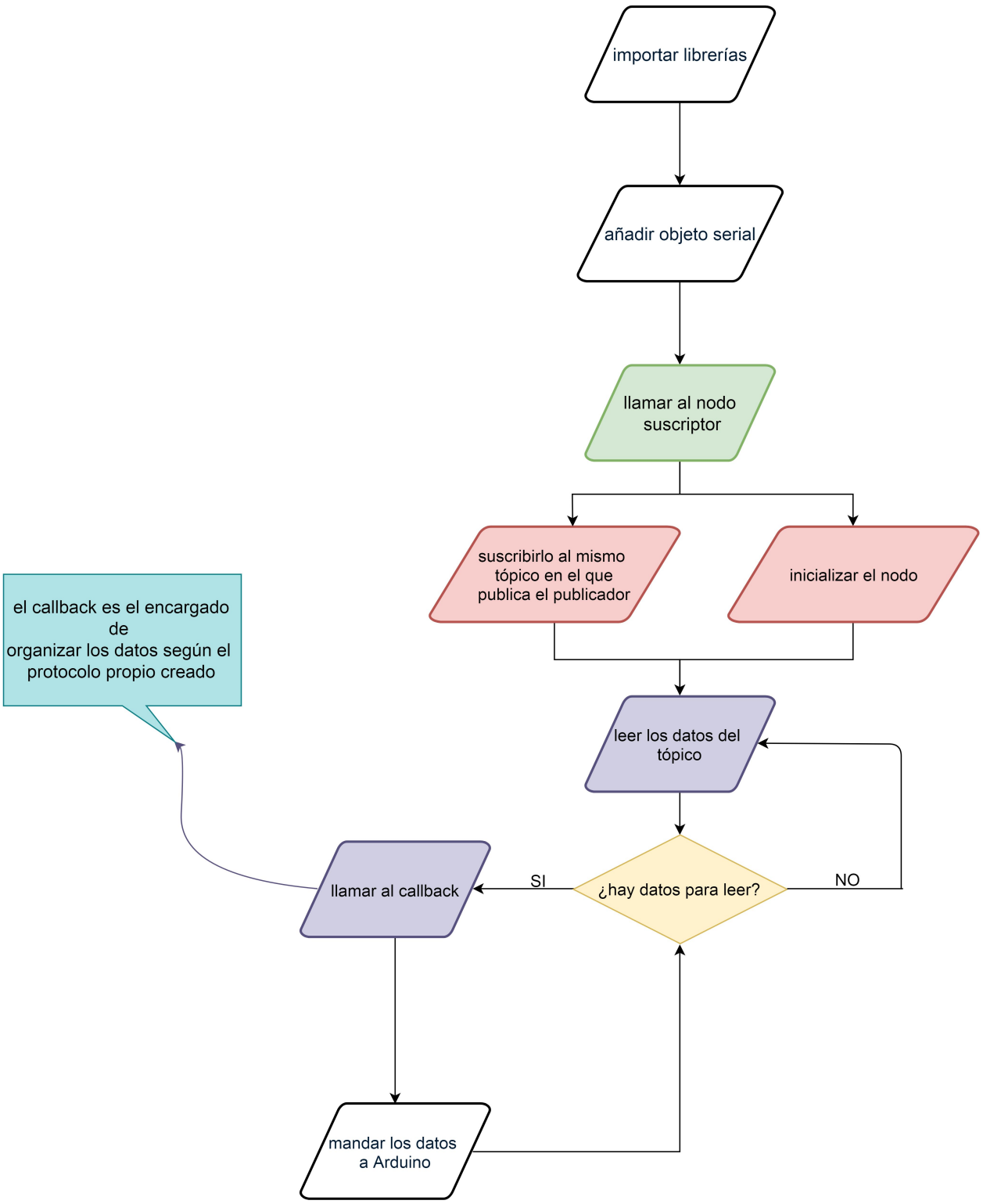


Figura 14: Diagrama del código del suscriptor

8.2.2 Arduino

8.2.2.1 Marco teórico

El lenguaje de programación de Arduino está basado en el lenguaje C++ y a la hora de escribir código será posible utilizar comandos de C++. Aun así es necesario conocer los comandos básicos de la programación en Arduino.[10]

- Estructura: Tiene una estructura simple basada en mínimo dos secciones.
  - *setup()* : Contiene la configuración, donde se declararán las variables. Es la primera función que se ejecutará y lo hará solo una vez.
  - *loop()* : Contiene el código que se ejecutará en bucle, como la lectura de datos en este caso.
- Funciones básicas en comunicación serie:
  - *Serial.begin(rate)* : Esta función es la encargada de abrir el puerto serie a una velocidad especificada entre paréntesis. Para la comunicación con un ordenador el valor de velocidad más común es de 9600. Se escribirá dentro de *setup()* puesto que el puerto serie se abrirá sola una vez.
  - *Serial.println(data)* : Se encarga de imprimir los datos en el puerto serie. La terminación 'ln' significa un salto de línea, por tanto si lo suprimimos se imprimirán todos los mensajes sin separación alguna.
  - *Serial.available()* : Obtiene un número correspondiente a la cantidad de bytes disponibles en el puerto serie para ser leídos, es decir, si en el puerto no hay ningún mensaje su valor será nulo.
  - *Serial.read()* : Se encarga de leer los datos del puerto serie.
  - *Serial.parseInt()* : Esta función es capaz de analizar una cadena de caracteres y devolver sólo los números que haya en ella. Lee caracteres hasta encontrarse con uno que no sea un número, en ese momento para y devuelve lo leído, dejando su 'cursor' en este nuevo caracter desde el que empezará a analizar la siguiente vez.

8.2.2.2 Aspectos prácticos.

Dirigiendo la programación de Arduino al proyecto en cuestión, habrá que diferenciar dos situaciones diferentes, la de antes de incluir el brazo robótico y la de después.

- Diseño sin el brazo robótico: Entorno de validación.

  Para esta primera parte se necesitarán dos puertos serie ya descritos con anterioridad. El puerto serie hardware es conocido por la placa y el programa. Para el puerto serie software en cambio, será necesario incluir su librería e identificar los pines que emularán este nuevo

puerto, los pines 8 y 9 de la placa Arduino en este caso, que serán el pin receptor y el transmisor respectivamente. Una vez declarado el nuevo puerto serie, se declarará una función *setup()* en la que se abrirán los dos puertos y se configurará a su vez la velocidad de transmisión de datos. Después se creará una función *loop()* que solo se ejecutará si hay datos en el puerto, es decir, si ' Serial.available()>0 '. Dentro de este bucle se definirá un protocolo para la lectura del mensaje, en el que se recibirá una cadena y esta función tendrá que ser capaz de separar la cadena de tal forma que se obtengan los ángulos deseados para las articulaciones del brazo robótico.

La comunicación serial hardware estará utilizándose para enviar los datos a la placa, y desde el puerto serie emulado, se imprimirán los ángulos mencionados con anterioridad. Así se estará comprobando si los datos llegan y se clasifican correctamente antes de integrar el robot en el programa.

Siguiendo todos los pasos descritos, el código tendría que quedar de la siguiente manera, donde el comando 'Serial' se referirá al puerto serie hardware y 'puertoSerieSW' en cambio, hará referencia al puerto software

```
#include "SoftwareSerial.h"

SoftwareSerial puertoSerieSW(8, 9); // Rx, Tx

void setup() {
  Serial.begin(9600);
  puertoSerieSW.begin(9600);
}

void loop () {
  if(Serial.available()>0){
    if(Serial.read() == 'S'){
      byte values[6];
      for(int i=0; i < 6; i++) {
        values[i] = Serial.parseInt();
        puertoSerieSW.println("Angulo " + String(i+1) + ": " + String(values[i]));
      }
      puertoSerieSW.println("####################################");
    }
  }
}
```

Figura 1: Código en Arduino

Para mayor comprensión de la estructura del código diseñado en Arduino puede hacerse uso de un diagrama de flujo, donde se resumirán los pasos anteriores de una forma esquemática.

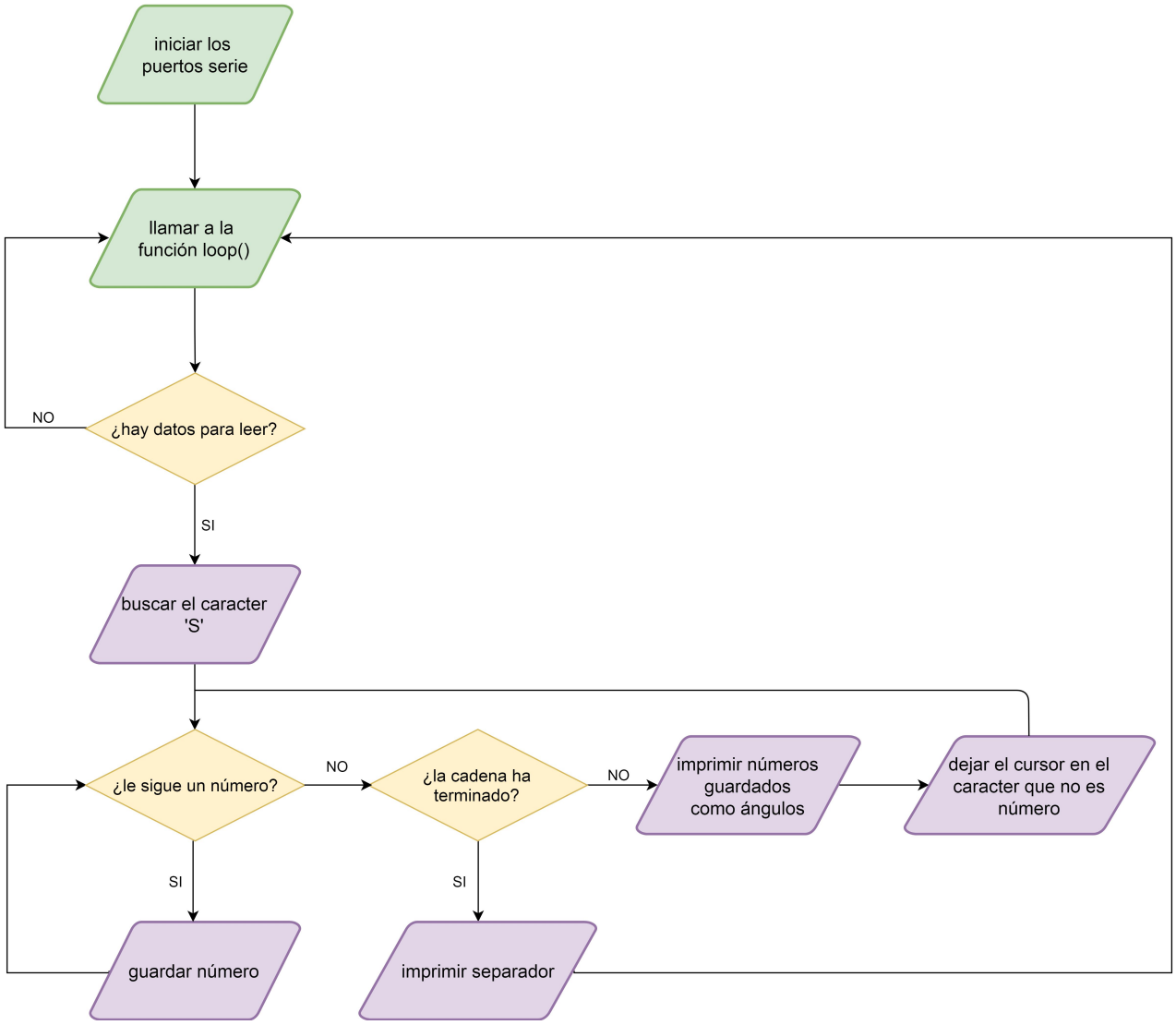


Figura 2: Diagrama de flujo del código en Arduino

Por último, ejecutando este código en el ordenador conectado por el puerto hardware, se imprimirán en el ordenador conectado al otro puerto los datos recibidos y procesados en Arduino.

```
Angulo 1:0
Angulo 2:90
Angulo 3:0
Angulo 4:0
Angulo 5:0
Angulo 6:10
##################################
Angulo 1:0
Angulo 2:90
Angulo 3:0
Angulo 4:0
Angulo 5:0
Angulo 6:50
##################################
Angulo 1:180
Angulo 2:15
Angulo 3:0
Angulo 4:90
Angulo 5:0
Angulo 6:50
##################################
Angulo 1:180
Angulo 2:15
Angulo 3:0
Angulo 4:90
Angulo 5:0
Angulo 6:10
##################################
```

Figura 3: Procesamiento de cadena en Arduino

- Integración de la comunicación con el brazo robótico en el programa de Arduino.

  Se partirá del programa anterior y se añadirán unas pocas líneas de código. Por una parte, será necesario incluir las librerías del brazo robótico y de los servos. Por otro lado, se definirán todas las articulaciones del brazo, cada una conectada a un servomotor. Al igual que se ha hecho con los puertos serie, será necesario inicializar la conexión con el brazo, configurando a su vez la velocidad de transmisión. Por último, quedará añadir al final del la función *loop()* un comando que enviará los ángulos al robot. Hay que tener en cuenta que el primer digito de este comando es 'step delay' y define los milisegundos que deben pasar entre el movimiento de cada servo, permitiendo valores de entre 10 y 30 ms.

```
#include <Braccio.h>
#include <Servo.h>
#include "SoftwareSerial.h"

Servo base;
Servo shoulder;
Servo elbow;
Servo wrist_rot;
Servo wrist_ver;
Servo gripper;

SoftwareSerial puertoSerieSW(8, 9); // Rx, Tx

void setup() {
  Serial.begin(9600);
  puertoSerieSW.begin(9600);

  Braccio.begin();
}

void loop() {
  if(Serial.available()>0){
    if(Serial.read() == 'S'){
      byte values[6];
      for(int i=0; i < 6; i++) {
        values[i] = Serial.parseInt();
        puertoSerieSW.println("Angulo " + String(i+1) + ": " + String(values[i]));
      }
      Braccio.ServoMovement(50, values[0], values[1], values[2], values[3], values[4], values[5]);
      puertoSerieSW.println("####################################");
    }
  }
}
```

Figura 4: Código de Arduino con conexión al robot integrada

Puede resultar interesante comparar este código final con el código utilizado para el entorno de validación y ver la simplicidad de incluir el brazo robótico en el proyecto una vez diseñado el control.

9.DESCRIPCIÓN DE LOS RESULTADOS

Una vez el proyecto haya finalizado es conveniente prestar atención a los objetivos propuestos al principio y hacer un análisis de cuales se han cumplido y cuales no. Volviendo al apartado 3, donde se definen los objetivos del trabajo se ha indicado que el objetivo principal del trabajo era el diseño y desarrollo de un *driver* que hiciera capaz la comunicación entre el brazo robótico TinkerKit Braccio y la plataforma de ROS. Como bien se ha visto en el apartado de diseño, este objetivo se ha alcanzado con existo.

Para llegar a este objetivo principal, se indicaron a su vez varios objetivos secundarios y sobre estos, se han conseguido los siguientes resultados:

- Se han adquirido los suficientes conocimientos del sistema operativo Ubuntu, de la plataforma de ROS, del lenguaje de Python y de Arduino como para diseñar un *driver* que comunique dos plataformas como lo son ROS y el brazo robótico.
- A su vez se ha conocido la conexión serial y se ha trabajado con ella hasta conseguir enviar datos desde un programa escrito en Python hasta uno escrito en Arduino.
- Se ha diseñado un protocolo propio en el que los ángulos especificados forman un conjunto de caracteres empezando con una 'S' y separando los números que componen cada ángulo con una '#'.
- Se ha diseñado un entorno de validación en el que poder comprobar si los datos llegados a Arduino son correctos, pudiendo así corregir fallos antes de comenzar a hacer pruebas con el robot.
- Se puede concluir que con un buen diseño se ha logrado que el robot coja la posición deseada en el momento que se manda la orden, ya que gracias al uso de mensaje tipo JointTrajectory se ha conseguido el control de cada articulación.

10.PLAN DE PROYECTO Y PLANIFICACIÓN

Cuando el diseño del proyecto ha concluido se procede a crear un plan de trabajo en el que se especificarán las diferentes tareas a llevar a cabo y el grupo encargado de hacerlo.

10.1 Descripción del equipo

En la siguiente tabla se describirá el equipo encargado del proyecto y el cargo de cada uno de ellos.

Tabla 12: Equipo encargado del proyecto

Nombre	Cargo
Oskar Casquero Oyarzabal	Director del proyecto
Nerea Gracia Zarraga	Ingeniera junior

10.2 Descripción de fases y tareas

El plan de proyecto se divide en diferentes paquetes de trabajo y cada paquete estará definido por un tiempo límite y diferentes tareas. Con el fin de comprobar que los plazos se cumplen correctamente se definirán varios hitos durante el proyecto.

Tabla 13: Paquetes de trabajo

Paquete de trabajo	Título	Comienzo	Final
WP1	Gestión del proyecto	Día 1	Día 98
WP2	Estudios de alternativas	Día 1	Día 9
WP3	Puesta a punto de los equipos	Día 9	Día 13
WP4	Formación	Día 13	Día 42
WP5	Desarrollo del proyecto	Día 42	Día 86
WP6	Documentación del proyecto	Día 86	Día 98

WP1: Gestión de proyecto

Este paquete tendrá como objetivo organizar y ejecutar correctamente el proyecto. Para ello se coordinarán los trabajos a realizar con la finalidad de completar a tiempo cada hito. También habrá que asegurar la correcta comunicación entre tareas.

Responsable: Oskar Casquero Oyarzabal

Participantes: Nerea Gracia Zarraga

WP2: Estudio de alternativas

Los aspectos que quedan fuera de los requisitos del sistema y que pueden optar por varias alternativas, tendrán que ser identificados y posteriormente adaptados a la opción más adecuada para el proyecto.

Responsable: Nerea Gracia Zarraga

Participante: Nerea Gracia Zarraga

Duración: 9 días

WP3: Puesta a punto de los equipos.

Esta fase se encargará de la instalación software. Se empezará instalando el sistema operativo Ubuntu de Linux en un ordenador, seguido de la instalación de la plataforma ROS y por último el paquete de Arduino. El lenguaje de programación elegido también tendrá que ser instalado.

Responsable: Nerea Gracia Zarraga

Participante: Nerea Gracia Zarraga

Duración: 4 días

WP4: Formación

En esta fase del proyecto los participantes dedicarán tiempo a adquirir conocimientos que posteriormente necesitarán para el desarrollo del trabajo.

Responsable: Nerea Gracia Zarraga

Participante: Nerea Gracia Zarraga

Duración: 29 días

- Tarea 4.1: Familiarización con Linux y Ubuntu.

En esta tarea se tratarán de conocer los comandos básicos necesarios para poder posteriormente operar con facilidad en este servidor.

Responsable: Nerea Gracia Zarraga

Participante: Nerea Gracia Zarraga

Duración: 6 días

- Tarea 4.2: Familiarización con ROS

  Será de gran importancia comprender con exactitud como se compone ROS y las diferentes herramientas de las que dispone. El punto más importante será entender como interacciona esta plataforma con Linux y con otros equipos.

  Responsable: Nerea Gracia Zarraga

  Participante: Nerea Gracia Zarraga

  Duración: 10 días

- Tarea 4.3: Aprendizaje del lenguaje Python.

  Puesto que Python será el lenguaje de casi todo el código a diseñar, en esta fase se tratará de manejar con soltura la programación.

  Responsable: Nerea Gracia Zarraga

  Participante: Nerea Gracia Zarraga

  Duración: 13 días

<u>WP5: Desarrollo del proyecto</u>

En esta fase viene todo el diseño necesario para cumplir el objetivo principal del proyecto.

Responsable: Oskar Casquero Oyarzabal

Participante: Nerea Gracia Zarraga

Duración: 44 días

- Tarea 5.1: Identificación de tópicos y nodos.

  Esta primera tarea del quinto paquete trata de decidir cuantos nodos y de que tipo serán necesarios a lo largo del proyecto, junto a la elección del tipo de mensaje que se transmitirá.

  Responsable: Oskar Casquero Oyarzabal

  Participante: Nerea Gracia Zarraga

Duración: 2 días

- Tarea 5.2: Diseño y programación de nodos.

  El diseño de los nodos se basará en la tarea anterior. Será importante en este punto diseñar un protocolo con el que asegurar la correcta transmisión de los datos y la buena comunicación entre los nodos.

  Responsable: Oskar Casquero Oyarzabal

  Participante: Nerea Gracia Zarraga

  Duración: 23 días

- Tarea 5.3: Diseño y programación en Arduino.

  El programa en Arduino deberá ser capaz de recibir datos y procesarlos para poder después tanto imprimirlos como enviárselos al robot.

  Responsable: Oskar Casquero Oyarzabal

  Participante: Nerea Gracia Zarraga

  Duración: 12 días

- Tarea 5.4: Comunicación entre ROS y Arduino mediante el objeto serial.

  Habrá que descubrir en este punto la manera de conectar vía serie los nodos de ROS con el programa en Arduino.

  Responsable: Oskar Casquero Oyarzabal

  Participante: Nerea Gracia Zarraga

  Duración: 6 días

- Tarea 5.5: Comunicar Arduino con el brazo robótico.

  En esta última tarea se implementará el brazo robótico a todo lo conseguido hasta ahora, consiguiendo así que los datos lleguen desde ROS hasta cada uno de los servomotores.

  Responsable: Oskar Casquero Oyarzabal

  Participante: Nerea Gracia Zarraga

  Duración: 1 días

WP6: Documentación del proyecto

En la última fase del proyecto se debe crear un documento en el que se explique al detalle todo el proceso, puesto que esta documentación podrá servir posteriormente de guía para otro nuevo proyecto.

Responsable: Oskar Casquero Oyarzabal

Participante: Nerea Gracia Zarraga

Duración: 12 días

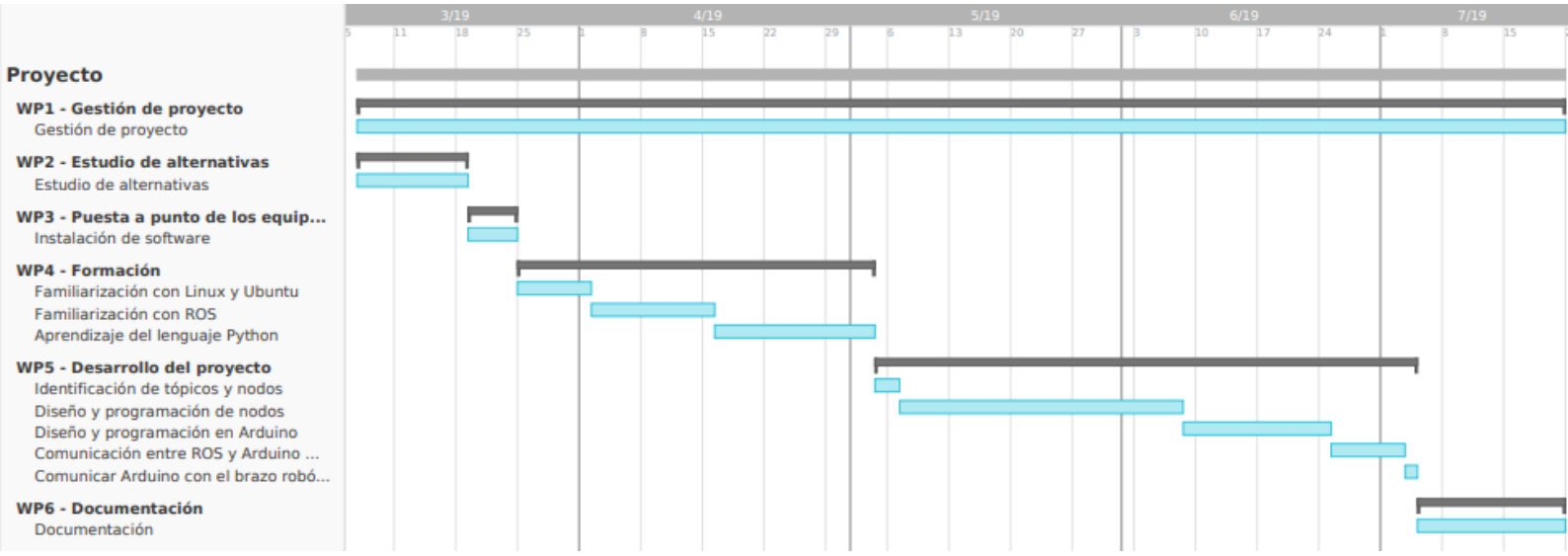
10.3 Hitos de la planificación

Además de las seis fases descritas con anterioridad, para la comprobación del correcto seguimiento del plan de trabajo, se definirán varios hitos a realizar.

Tabla 14: Hitos de la planificación

Hito	Título	Fecha
1	Estudio de alternativas	Día 9
2	Instalación del software	Día 13
3	Informe sobre ROS	Día 26
4	Diseño de nodos completado	Día 67
5	Diseño en Arduino completado	Día 79
6	Conexión serial resuelta	Día 85
7	Entrega de la memoria teórica	Día 98

11.DIAGRAMA DE GANTT



12.ASPECTOS ECONÓMICOS

Una vez asignadas las tareas y la duración del trabajo, se valorarán los aspectos económicos del proyecto. Esto se hará calculando dos presupuestos diferentes. Por un lado, un presupuesto considerará los costes de desarrollo, es decir, se basará en las horas que necesita el equipo de trabajadores para adquirir los conocimientos necesarios para ser capaces de desarrollar el proyecto. Por otro lado, el segundo presupuesto calculará los costes de ejecución, esto es, se basará en el desarrollo del proyecto.

Tabla 15: Presupuesto de desarrollo

CONCEPTO	UNIDADES	COSTE UNITARIO	NÚMERO DE UNIDADES	COSTE TOTAL
Horas internas				8.250,00€
Director	horas	60,00€	50	3.000,00€
Ingeniero junior	horas	35,00€	150	5.250,00€
Amortizaciones				40,00€
Ordenador	horas	0,20€	200	40,00€
COSTES DIRECTOS				8.290,00€
COSTES INDIRECTOS (10% de los directos)				829,00€
SUBTOTAL				9.119,00€
IMPREVISTOS (10% del subtotal)				911,90€
TOTAL				**10.030,90€**

Tabla 16: Presupuesto de ejecución

CONCEPTO	UNIDADES	COSTE UNITARIO	NÚMERO DE UNIDADES	COSTE TOTAL
Horas internas				1,175,00€
Director	horas	60,00€	5	300,00€
Ingeniero junior	horas	35,00€	25	875,00€
Amortizaciones				6,00€
Ordenador	horas	0,20€	30	6,00€
Gastos				239,49€
TinkerKit Braccio		199,00€	1	199,00€
Arduino Mega 2560		34,50€	1	34,50€
Adaptador TTL-USB		5,99€	1	5,99€
COSTES DIRECTOS				1.420,49€
COSTES INDIRECTOS (10% de los directos)				142,05€
SUBTOTAL				1.562,54€
IMPREVISTOS (10% del subtotal)				156,25€
TOTAL				**1.718,79€**

13.CONCLUSIONES

Este apartado detallará las diferentes conclusiones a las que se ha llegado durante la preparación y realización del proyecto.

La primera conclusión adquirida es que la implementación de ROS en un entorno robótico facilitará mucho el trabajo, puesto que ayudará en la conexión entre diferentes robots. Además, el conocimiento básico necesario para poder utilizarlo es de rápida comprensión. A esto se le añade una amplia comunidad de usuarios y tutoriales que ayudan a acelerar ese proceso.

Como segunda conclusión consta que es de total importancia conocer bien el lenguaje de programación a utilizar. Programando con conceptos entendidos a medias, puede pasar fácilmente que no se sepa de donde viene un fallo o en un momento dado no saber como programar un apartado del código. Todo esto puede ralentizar considerablemente el proyecto, siendo un inconveniente en la búsqueda de un trabajo óptimo.

Una tercera conclusión sería que el robot TinkerKit Braccio y las placas de Arduino fueros creados para el aprendizaje, puesto que hay una gran cuantía de información sobre ello en la web y no son de precios relativamente caros.

Por último, se concluye que habiendo hecho un grado de Ingeniería en Tecnología Industrial, no se adquieren los conceptos necesarios para poder llevar a cabo este trabajo de fin de grado, puesto que no se profundiza en temas como la robótica o la programación, lo que ha podido ser un problema al comienzo.

BIBLIOGRAFIA

[1] *La automatización de la industria*. 1993, de AITIM. Sitio web: https://infomadera.net/uploads/articulos/archivo_2116_17668.pdf

[2]R. González, V. *Robots industriales*. 2002. Sitio web: http://platea.pntic.mec.es/vgonzale/cyr_0204/ctrl_rob/robotica/industrial.htm

[3] *TinkerKit Braccio robot,* de Arduino. Sitio web: https://store.arduino.cc/tinkerkit-braccio

[4]*Braccio: Brazo Robótico controlable con arduino*. Sitio web: https://www.mcielectronics.cl/shop/product/braccio-brazo-robotico-controlable-con-arduino-25457

[5] *ROS.* Última edición el 18 de febrero de 2018, por Recio, Igor. Sitio web: http://wiki.ros.org/es

[6]*Brazo Robot Arduino Braccio*, de Brico Geek. Sitio web: https://tienda.bricogeek.com/robots/1157-brazo-robot-arduino-braccio.html

[7] *Braccio Quick Start Guide*, de RS Components Ltd. Sitio web: https://docs-emea.rs-online.com/webdocs/14da/0900766b814da22f.pdf

[8] *Shield LTE/GPS SIM7000E para Arduino*, de Brico Geek. Sitio web: https://tienda.bricogeek.com/shields-arduino/1197-shield-lte-gprs-gps-sim7000e-para-arduino.html

[9] Delgado, M. (2017). *Arduino y su documentación en español*. Sitio web: http://manueldelgadocrespo.blogspot.com/p/arduino-mega-2560.html

[10] Ruiz, J.M. (2007). *Manual de Programación Arduino*. Sitio web: https://arduinobot.pbworks.com/f/Manual+Programacion+Arduino.pdf