Máster Universitario en Automática e Informática Industrial

Curso: 2018/2019

Trabajo Fin de Máster

“Guiado de un cobot con joystick para la grabación de trayectorias.”

Alumno: Vizcaíno Espejo, Juan Rodrigo

Director: Mellado Arteche, Martín

Septiembre 2019

ai2 INSTITUTO DE AUTOMÁTICA E INFORMÁTICA INDUSTRIAL

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Resumen

Este Trabajo Fin de Máster tiene como objetivo desarrollar una aplicación que permita el guiado del robot colaborativo UR5e mediante un dispositivo externo para la grabación de trayectorias. La aplicación diseñada permite mover el robot mediante un joystick, grabar dos tipos de trayectorias y reproducirlas cuando se desee.

Durante el desarrollo del trabajo se definieron los objetivos a alcanzar. Una vez fijados se analizaron los requerimientos del trabajo para lograr dichos objetivos. Se vieron, por una parte, los requerimientos hardware necesarios como son el UR5e, una placa Arduino y un mando de la Wii™. Y, por otra parte, se analizaron diferentes opciones de programación que presentan estos dispositivos para poder implementar los algoritmos necesarios.

A continuación, se analizaron diferentes alternativas. Estas alternativas se estudiaron para diseñar la aplicación que permitiera lograr los objetivos y que finalmente se descartaron por diversos motivos. Luego se presentó la primera solución implementada que consiste en la construcción de un mando con una placa Arduino MEGA. En este apartado se analizaron diferentes posibilidades para establecer la comunicación entre el UR5e y el Mando Arduino y se explicaron el funcionamiento y los algoritmos diseñados, tanto para la placa Arduino como para el UR5e. También se analizaron las diferentes opciones para alimentar el mando, se presentaron los diferentes dispositivos que se conectan a la placa Arduino para formar el mando y se presentó el mando diseñado. Seguidamente se presentó la segunda solución implementada que consiste en utilizar un mando de la Wii™. Se explicó cómo se conecta el mando Wii™ al UR5e mediante un PC como dispositivo intermedio y se explicó el funcionamiento y los algoritmos diseñados, en el PC y en el UR5e. A continuación, se presentó la interfaz de usuario implementada, para la aplicación del Mando Wii™, y su funcionamiento. Para terminar, se analizaron las diferencias entre las dos soluciones desarrolladas y las ventajas e inconvenientes de cada una de ellas.

Para finalizar, se mostró el presupuesto del proyecto realizado y se analizaron las conclusiones y posibles mejoras para el trabajo realizado. Por último, se mostró el manual de usuario diseñado para el Mando Arduino y para el Mando Wii™. Se pretende que mediante este manual el usuario aprenda a utilizar las aplicaciones diseñadas para controlar el UR5e. Para ello se muestra como establecer la comunicación entre el mando y el UR5e y las diferentes funciones que ofrecen los mandos.

Palabras clave: robot colaborativo, dispositivos guiados, guiado robot, grabación de trayectorias, cobot, dispositivo externo, joystick, UR5e.

Resum

Aquest Treball Fi de Màster té com a objectiu desenvolupar una aplicació que permeta el guiat del robot col·laboratiu UR5e mitjançan un dispositiu extern per a la gravació de trajectòries. L' aplicació dissenyada permet moure el robot mitjançan un joystick, gravar dos tipus de trajectòries i reproduirlas quan es desitge.

Durant el desenvolupament del treball es van definir els objectius a assolir. Una vegada fixats es van analitzar els requeriments del treball per aconseguir aquest objectius. Es van veure, d' una banda, els requeriments hardware necessaris com són el UR5e, una placa Arduino y un comandament de la Wii. I, d' altra banda, es van analitzar diferents opcions de programació que presenten aquests dispositius per a poder implementar els algoritmes necessaris.

A continuació, es van analitzar diferents alternatives. Aquestes alternatives es van a estudiar per dissenyar l' aplicació que permetés assolir els objectius i que finalment es van descartar per diversos motius. Després es va presentar la primera solució implementada que consisteix en la construcció d 'un comandament amb una placa Arduino MEGA. En aquest apartat es van analitzar diferents possibilitats per establir la comunicació entre el UR5e i el Comandament Arduino i es va explicar el funcionament i els algoritmes dissenyats, tant per a la placa Arduino com per al UR5e. També es van analitzar les diferents opcions per alimentar el comandament, es va presentar els diferents dispositius que es connecten a la placa Arduino per formar el comandament y es va presentar el comandament dissenyat. Seguidament es va presentar la segona solució implementada que consisteix a utilitzar un comandament de la Wii. En va explicar com es connecta el comandament Wii al UR5e mitjançan un PC com a dispositiu intermedi i es va explicar el funcionament i els algoritmes dissenyats, en el PC i en el UR5e. A continuació, es va presentar la interfaz d' usuari implementada, per a l' aplicació del Comandament Wii, i el seu funcionament. Per acabar, es van analitzar les diferències entre les dues solucions desenvolupades i les avantatges i inconvenients de cadascuna d' elles.

Per finalitzar, es va mostrar el pressupost del projecte realitzat i es van analitzar les conclusions i possibles millores per al treball realitzat. Finalment es va mostrar el manual d' usuari dissenyat per al Comandament Arduino i per al Comandament Wii. Es pretén que mitjançan aquest manual l' usuari aprenga a utilitzar les aplicacions dissenyades per controlar el UR5e. Per a això es mostra com establir la comunicació entre el comandament i el UR5e i les diferents funcions que ofereixen el comandaments.

Paraules Clau: robot col·laboratiu, dispositius guiats, guiat de robot, gravació de trajectòries, cobot, dispositiu extern, joystick, UR5e.

Abstract

This Master´s Final Project aims to develop an application that allows the guidance of the UR5e collaborative robot through an external device for recording path. The designed application allows you to move the robot using a joystick, record two types of path and play them provided that you want.

During the work development, the objectives to be achieved were defined. Once the objectives has been set the work requeriments were analyzed to reach these objectives. On the one hand, the necessary hardware requirements we have seen have been the UR5e, an Arduino board and a Wii remote control. And, on the other hand, different programming options presented by these devices were analyzed in order to implement the necessary algorithms.

After, we analyzed different alternatives. These alternatives were studied to design the application that would allow us to achieve the objectives and have been discarded for various reasons. Then we presented the first solution implemented which consists of the construction of a remote with an Arduino MEGA board. In this section, different possibilities for establishing communication between the UR5e and the Arduino Remote were analyzed and the operation and algorithms designed for both the Arduino board and the UR5e were explained. The different options to power the remote were also analyzed, the different devices that are connected to the Arduino board to form the remote and the designed were presented. Later, the second solution implemented was presented, which consists of using a Wii remote. It was explained how the Wii remote is connected to the UR5e by a PC as an intermediate device and we explained the operation and the algorithms designed, on the PC and on the UR5e. Next, the implemented user interface was presented, for the Wii Remote application, and its operation. In the end, the differences between the two solutions developed and the advantages and disadvantages of each of them were analyzed.

Finally, the project budget was shown and the conclusions and possible improvements for the work done were analyzed. At last, the user manual designed for the Arduino Remote and for the Wii Remote were displayed. It is intended that through this manual the user learn to use the applications designed to control the UR5e. For this reason it is shown how to establish the communication between the remote control and the UR5e and the different functions offered by the remote controls.

Keywords: Collaborative robot, guided devices, guided robot, recording path, cobot, external device, joystick, UR5e.

1. Introducción.

En el presenta Trabajo Fin de Máster se trabajará con un robot colaborativo o cobot, en concreto el UR5e de la empresa Universal Robots. Y se analizarán diferentes alternativas para:

- Guiado del cobot mediante un dispositivo externo, en concreto mediante un joystick.
- Grabación de trayectorias para su posterior reproducción por parte del cobot.

1.1 Antecedentes.

El campo de la robótica ha evolucionado velozmente en los últimos años, debido a este veloz crecimiento de los robots es cada vez más complicado encontrar una definición para ellos que consiga incluir los a todos.

El término *"robot"* aparece por primera vez en la obra de teatro "*Robots Universales de Rossum"* de Karel Capek en 1921, en esta los robots son una especie de humanos artificiales creados para reducir la carga de trabajo al resto de personas. El termino proviene de la palabra checa *"robota"* que significa *"trabajo"* y se puede entender como *"trabajo duro".*

Sin embargo, mucho antes la humanidad ya se había interesado por la creación de máquinas que pudieran ayudar al hombre, en el siglo VIII a. de C. Homero en su célebre Ilíada ya menciona a unos sirvientes mecánicos dotados de inteligencia. En 1495 Leonardo da Vinci realizó los planos de un robot humanoide, un caballero mecánico, que podría realizar ciertos movimientos como mover la cabeza, la mandíbula, los brazos e incluso sentarse.

En el siglo XVIII se desarrollan una serie autómatas que pretendían reproducir fielmente los movimientos de los seres vivos. Por ejemplo, el *Canard digérateur* (1738) , del ingeniero francés Jacques de Vaucanson, se trataba de un pato que era capaz de hacer la digestión. El relojero suizo Pierre Jaquet-Droz es el responsable de los famosos autómatas conocidos como *"el pianista"* , *"el dibujante"* y *"el escritor"* construidos entre 1768 y 1774*.* La pianista toca el órgano pulsando las teclas con sus propios dedos, está compuesta por 2.500 piezas y puede mover los ojos mirando al piano y a los dedos y al finalizar realiza una reverencia. El dibujante está compuesto por 2.000 piezas y era capaz de realizar hasta cuatro dibujos distintos, además mueve los ojos, las manos y sopla al papel para eliminar los restos de lápiz. El escritor consiste en un mecanismo de más de 6.000 piezas que es capaz de escribir textos de unas cuarenta palabras de longitud, para ello mojaba la pluma en tinta, escurría el sobrante, levantaba la pluma como si estuviera pensado, respetaba los espacios y puntos y aparte y seguía con los ojos el papel y la pluma mientras escribía.

Figura 1. Autómatas de Jaquet Droz.

Fue ya en 1950 cuando Isaac Asimov publica su obra "*Yo Robot"* que populariza la robótica . En esta obra se establecen las famosas tres leyes de la robótica que son las siguientes [1]:

1. *"Un robot no puede hacer daño a un ser humano o, por inacción, permitir que un ser humano sufra daño."*

2. *"Un robot debe obedecer las órdenes dadas por los seres humanos, excepto si estas órdenes entrasen en conflicto con la primera Ley."*

3. *"Un robot debe proteger su propia existencia en la medida en que esta protección no entre en conflicto con la primera o segunda Ley."*

En 1954 George Devol patenta el que se considera el primer robot industrial llamado Unimate. La idea surge al darse cuenta de que en la industria había tareas repetitivas y por eso quiso construir una máquina que hiciera más fácil el trabajo dentro de la industria.

Figura 2. Unimate. Primer robot industrial.

En 1956 George Devol junto a Josef Engelberger crean la primera empresa de robótica llamada Unimation (Universal Automation). En 1961 instalan el primer robot industrial en una cadena de montaje de General Motors. En 1978

Unimation junto con General Motors desarrolla el robot PUMA (Programmable Universal Machine Assembly) este robot inició la era de la robótica industrial.

Figura 3. Robot PUMA

Como conclusión, la robótica aparece en los años 50, pero no es hasta finales de los años 70 cuando empieza el crecimiento y la comercialización de robots industriales. Hoy en día la robótica ha evolucionado mucho y está presente en muchos sectores industriales, además de estos la gran evolución de la robótica ha dado paso a nuevos robots como los robots de servicio o los robots colaborativos, de los que se hablará con mayor profundidad más adelante.

1.2 Robots.

En la actualidad existen diferentes definiciones, una de las más aceptada es la propuesta por la Asociación de Industrias Robóticas (RIA) que dice lo siguiente [2]:

"Un robot industrial es un manipulador multifuncional reprogramable, capaz de mover materias, piezas, herramientas o dispositivos especiales, según trayectorias variables, programadas para realizar tareas diversas".

Esta definición con algunas pequeñas modificaciones ha sido aceptada por la Organización Internacional de Estándares (ISO) que dice lo siguiente [2]:

"Manipulador multifuncional reprogramable con varios grados de libertad, capaz de manipular materias, piezas, herramientas o dispositivos especiales según trayectorias variables programadas para realizar tareas diversas."

Aunque existen muchas más definiciones se va a terminar con la propuesta por la Federación Internacional de Robótica (IFR) que distingue entre robot industrial de manipulación y otros robots [2]:

"Por robot industrial de manipulación se entiende una máquina de manipulación automática, reprogramable y multifuncional con tres o más ejes que pueden posicionar y orientar materias, piezas, herramientas o dispositivos especiales para la ejecución de trabajos diversos en las diferentes etapas de la producción industrial, ya sea en una posición fija o en movimiento."

Como se dijo en el apartado anterior hoy en día los robots industriales están presentes en mucho de los sectores de la industria como:

- Alimentación, donde realizan tareas de embalaje, paletizado y despaletizado, etiquetado, etc.
- Químico y farmacéutico, donde realizan tareas de paletizado y despaletizado, etc.
- Plásticos y polímeros, donde realizan tareas de desmoldeado, carga y descarga de circuitos impresos, tareas de pick&place, etc.
- Automoción, los primeros robots industriales se instalaron en cadenas de fabricación de automóviles, hoy en día los robots siguen presentes en este sector realizando diversas tareas como: soldadura, pulido, pintura, atornillado, control de calidad, etc.

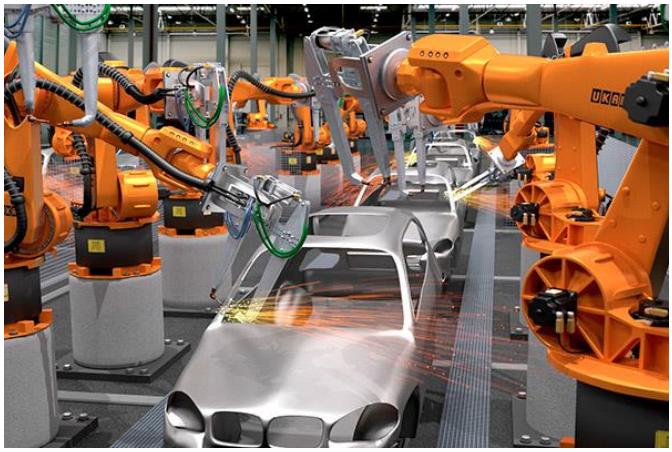


Figura 4. Robot industrial en el sector de automoción.

Además de los robots industriales la gran evolución de la robótica ha dado paso a nuevos robots como los robots de servicio, que la IFR define de la siguiente manera [3]:

> *"Un robot que opera semi o totalmente autónomamente, para realizar servicios útiles para el bien estar de los seres humanos y equipos, con exclusión de las operaciones de fabricación."*

Estos están presentes en diversos campos como: doméstico, agricultura, submarina, militar, entretenimiento, educación, aéreo, médico, etc. Algunos ejemplos son:

- Robots domésticos: sustituyen a las personas en tareas del hogar como aspirar, fregar, limpia cristales, etc.

Figura 5. Robot aspirador.

- Robots médicos: Se están utilizando robots para realizar ciertas tareas en la que se puede destacar la telecirugía y la cirugía mínimamente invasiva. La gran ventaja que estos aportan son la precisión y la miniaturización.

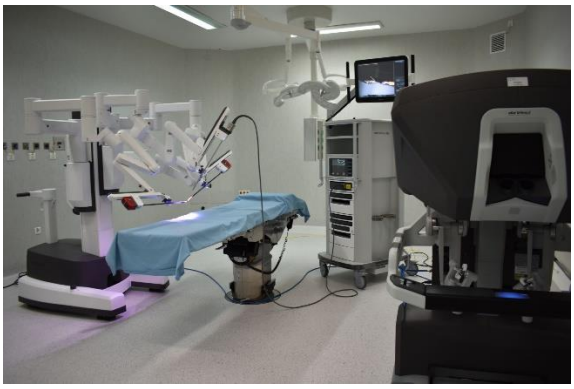


Figura 6. Robot de cirugía.

Los últimos avances en robótica tienen que ver con los robots colaborativos o cobots, estos están diseñados para trabajar junto a las personas. Actualmente estos robots colaborativos se encuentran en multitud de sectores industriales desarrollando diversas tareas. Esto supone un importante avance en la industria y supone ciertas ventajas que se comentarán más adelante.

1.3 Robots colaborativos.

Los robots colaborativos o cobots se extienden en el mercado de la robótica rápidamente y cada vez son más los cobots instalados en fábricas. Los últimos avances en robótica industrial se dirigen a la Industria 4.0 y los cobots se posicionan como la mejor opción para la automatización industrial convirtiéndose en una pieza importante para el desarrollo de esta nueva industria.

El termino cobot nace de la unión de la palabra *"Colaboración"* y *" Robot"*, como se puede intuir por el nombre la principal característica de los cobots, y lo que lo diferencia de los robots industriales convencionales, es que permiten la interacción entre los robots y las personas.

Estos robots permiten interactuar con personas sin el peligro que supondría la interacción con un robot industrial, por lo que no se requiere de forma obligatoria la instalación de vallas de seguridad, ni de otros sistemas. Además, la interacción entre el robot y el operario en el proceso de producción permite unir las ventajas que aportan cada uno de ellos, por ejemplo, el operario realiza alguna tarea más especializada y el robot se encarga de las tareas más repetitivas. De esta forma se consigue automatizar procesos que serían imposibles con robots tradicionales (se estima que solo el 10% de los trabajos son totalmente automatizables). Según un estudio realizado por el MIT (Instituto Tecnológico de Massachusetts) la colaboración entre humanos y robots es un 85% más productiva que la podrían conseguir estos por separado.

Además de las comentadas anteriormente los robots colaborativos presentan otras ventajas como el pequeño tamaño de su base que hace que el espacio necesario para llevar a cabo sus tareas sea mucho menor que el que necesita un robot industrial, sumándole que no necesitan vallado de seguridad el tamaño necesario en la planta se reduce aún más, por lo que se ahora espacio sin perder seguridad. Además, los cobots son muy sencillos de instalar y programar lo que los hacen más accesibles a la pequeña y mediana empresa. También es sencillo cambiar la herramienta para adaptarlo a diferentes tareas dentro de la línea de fabricación y son muy ligeros por lo que tienen mayor facilidad de recolocación. En lo que se refiere al precio los robots colaborativos suponen un menor esfuerzo económico tanto por el precio del robot como el coste de los sistemas de seguridad. Por otra parte, ofrecen un rápido retorno de inversión (ROI).

Sin embargo, los cobots presentan ciertas desventajas, la mayoría de ellas son limitaciones impuestas por el hecho de trabajar junto a personas. Por este motivo, la cargas que pueden manipular son menores que las de un robot industrial para evitar que, en caso de colisión, la inercia sea grande y cause daños graves. También la velocidad de movimiento está limitada. Aunque los cobots dispongan de sensores para detectar colisiones con su entorno y frenar o realizar alguna otra acción de seguridad en caso de colisión, estas limitaciones son necesarias para que la fuerza del impacto no sea lo suficientemente grande para causar daños graves. A pesar de estas limitaciones que lo hacen apto para trabajar junto a personas, en ocasiones puede ser necesario aplicar otras medidas de seguridad, por ejemplo, en ciertas aplicaciones donde la herramienta pueda resultar peligrosa se pueden añadir ciertas medidas de seguridad como delimitar áreas de seguridad mediante sensores externos conectados al cobot. Por ejemplo, si la herramienta es un elemento punzante se puede conectar un sensor que detecte la presencia de personas y pare el robot mientras la persona permanezca en el interior y que cuando el operario haya salido de la zona el robot podrá continuar con su tarea sin ningún problema. Además de esto los cobots permiten configurar diferentes opciones de seguridad como definir zonas del espacio restringidas donde el robot no puede operar, crear esfera de protección alrededor de la herramienta para evitar que esta traspase ciertos límites, botón de parada de emergencia, etc.

Los cobots fueron inventados en 1996 por J.Edward Colgate y Michael Peshkin, dos profesores de la Northwestern University, por lo que son una patente estadounidense de 1997. En el año 2004 la empresa alemana KUKA lanzó al mercado el primer robot colaborativo. En 2008 se sumó al mercado la empresa danesa Universal Robots, que es hoy en día el mayor fabricante de robots colaborativos. A partir de 2010 el uso de cobots empezó a extenderse y otras empresas del sector lanzaron sus modelos de robot colaborativo. A continuación, se presentarán algunas de las empresas que desarrollan cobots y algunas características de estos.

1.3.1 KUKA

Empresa alemana fundada en 1898 es una de las pioneras en robótica industrial a nivel mundial. Además de sus robots industriales han desarrollado el *LBR iiwa* el robot colaborativo de KUKA. Existen dos versiones de este cobot una con capacidad de 7 Kg y un alcance de 800 mm y la otra con una capacidad de 14 Kg y un alcance de 820 mm.

Figura 7. Robot colaborativo LBR iiwa de KUKA.

Algunas de las características a destacar de este cobot son la facilidad de aprendizaje, dispone de sensores que le permite regular la fuerza en función del tipo de pieza que esté manejando y dispone de siete ejes que lo dotan de mayor flexibilidad y liberta de movimientos.

1.3.2 ABB

La empresa ABB nace de la fusión de ASEA y BBC en 1988 con sede en Zúrich, Suiza. Esta se ha convertido en una empresa líder mundial en ingeniería eléctrica

y automatización, la empresa ha instalado más de 400.000 robots en todo el mundo

En el año 2015 ABB presenta oficialmente su robot colaborativo llamado YuMi. Se trata de un robot cobot con dos brazos y manos flexibles diseñado para el ensamblaje de piezas pequeñas.

Figura 8. Robot colaborativo YuMi de ABB.

Las manos servo controladas incluyen cámara de visión que le permite localizar piezas, además presenta sistemas de alimentación de pinzas y puede ser programado sin la necesidad de código, el YuMi tiene la capacidad de aprender la trayectoria tras ser guiado de manera manual. Tiene una capacidad de carga de 0.5 Kg en cada brazo y un alcance de 559 mm. La empresa define este cobot de la siguiente manera *"YuMi es una visión de futuro"*, *"YuMi cambiará nuestro concepto de la automatización del ensamblaje"* y *"YuMi es como trabajar con un compañero con posibilidades ilimitadas"* [4].

En 2018 ABB presentó un nuevo miembro de la familia YuMi, se trata de una versión con un solo brazo.

Figura 9. Robot colaborativo YuMi single arm de ABB.

1.3.3 FANUC

Empresa fundada en 1956 por Dr. Seiuemon Inaba en Japón, actualmente es un grupo de compañías con sede en Japón, EEUU y Luxemburgo que ofrecen servicios de automatización industrial. Disponen de diferentes modelos de robots colaborativos como el CR-4Ai, el CR-7Ai que son robots colaborativos compactos y ligeros con una capacidad entre 4 y 7 Kg y un alcance entre 4 y 7 Kg. FANUC dispone de una versión media el CR-15Ai es un robot de seis ejes con capacidad de 15 Kg y alcance de 1441 mm. Por último, dispone de una versión que dista del resto de robots colaborativos que hay en el mercado en lo que se refiera a carga máxima y alcance, ya que este funciona con cargas de hasta 35 Kg y alcance superior a los 1800 mm. El inconveniente es su peso que le hace menos flexible y le resta movilidad.

Figura 10. Robot colaborativo CR-35iA de FANUC.

1.3.4 Universal Robots

La empresa Universal Robots se funda en 2005 en Dinamarca por los ingenieros Esben Østergaard, Kasper Støy y Kristian Kassow, al darse cuenta de que en el mercado de la robótica dominaban los robots pesados y caros pensaron en crear un robot innovador que se adaptara a la industria actual. En el año 2008 Universal Robots vende el primer robot colaborativo, el UR5. Hoy en día Universal Robots es una de las empresas que más se ha expandido y continúa liderando el sector, vende más cobots que todos sus competidores juntos.

Con el lanzamiento del UR5 en 2008 se abrió un nuevo mercado, hasta entonces la robótica era algo caro y complejo, estos nuevos robots estaban destinados a la pequeña y mediana empresa. Según Universal robots: *"es innegable que la empresa ha sabido dar respuesta a las exigencias del mercado que necesita un robot flexible, fácil de utilizar, capaz de trabajar junto al resto del personal y que ofrezca un retorno de inversión rápido y eficaz".* Actualmente, los robots colaborativos de UR están presentes en muchos sectores industriales como: alimentación y agrícola, mobiliario y equipamiento, electrónica y tecnología, metal y mecanizado, automoción, plásticos y polímeros, farmacéutica y química y científica e investigación

Universal Robots en 2008 lanzan el UR5, en el año 2012 la empresa lanza al mercado su segundo cobot el UR10. En el 2014 la empresa saca una versión revisada de sus dos modelos y al año siguiente lanza al mercado el último robot de la familia el UR3. En el año 2018 la empresa lanza una nueva generación de cobots *e-series* que incluye avances tecnológicos que mejoran las aplicaciones que se pueden desarrollar con esta familia de cobots, además la programación es todavía más sencilla que en los modelos anteriores, dispone de una interfaz más simple e intuitiva y cumple con los últimos estándares de seguridad ISO (EN ISO 13849-1 y EN ISO 10218-1).

En la Tabla 1 se muestran algunas de las características de esta familia de robots colaborativos.

Cobot	Carga útil (Kg)	Alcance (mm)	Peso (Kg)	Huella (mm)
UR3e	3	500	11	128
UR5e	5	850	18.4	149
UR10e	10	1300	33.5	190

Tabla 1. Características cobots de Universal Robots.

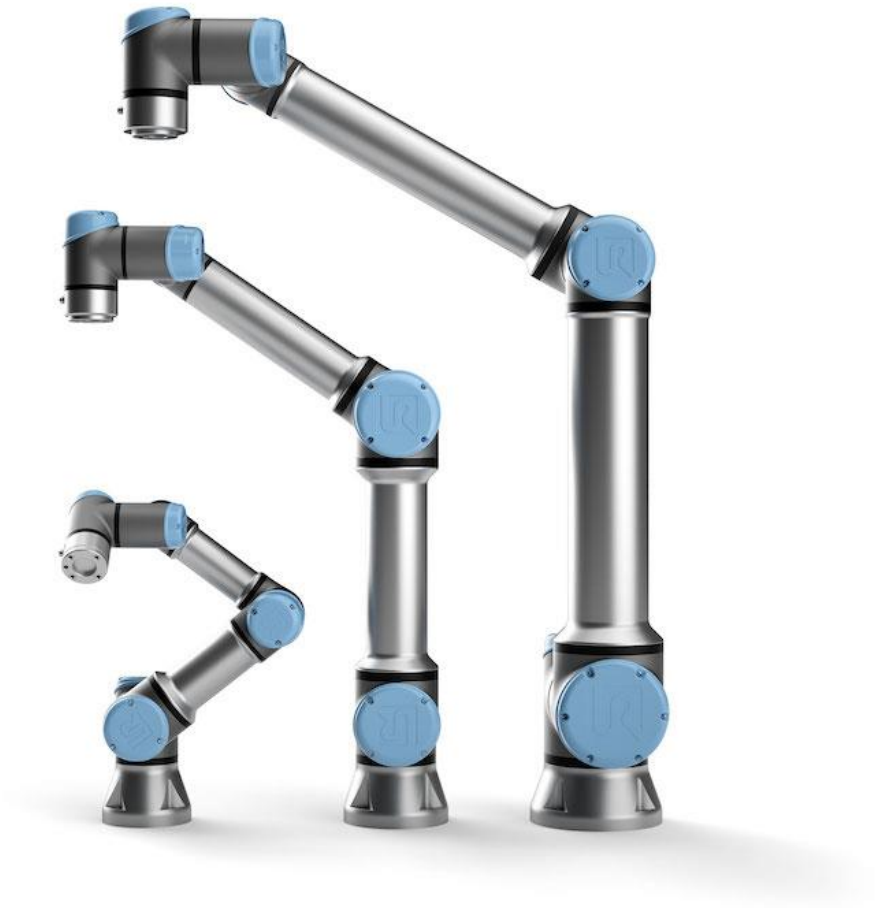


Figura 11. Robot colaborativo UR3, UR5 y UR10 de la empresa Universal Robots.

Para el desarrollo del presente Trabajo Fin de Máster se ha utilizado el cobot UR5e adquirido por el instituto ai2, disponible en el Laboratorio *Pedro Albertos de sistemas de Fabricación inteligente para la Industria 4.0.*

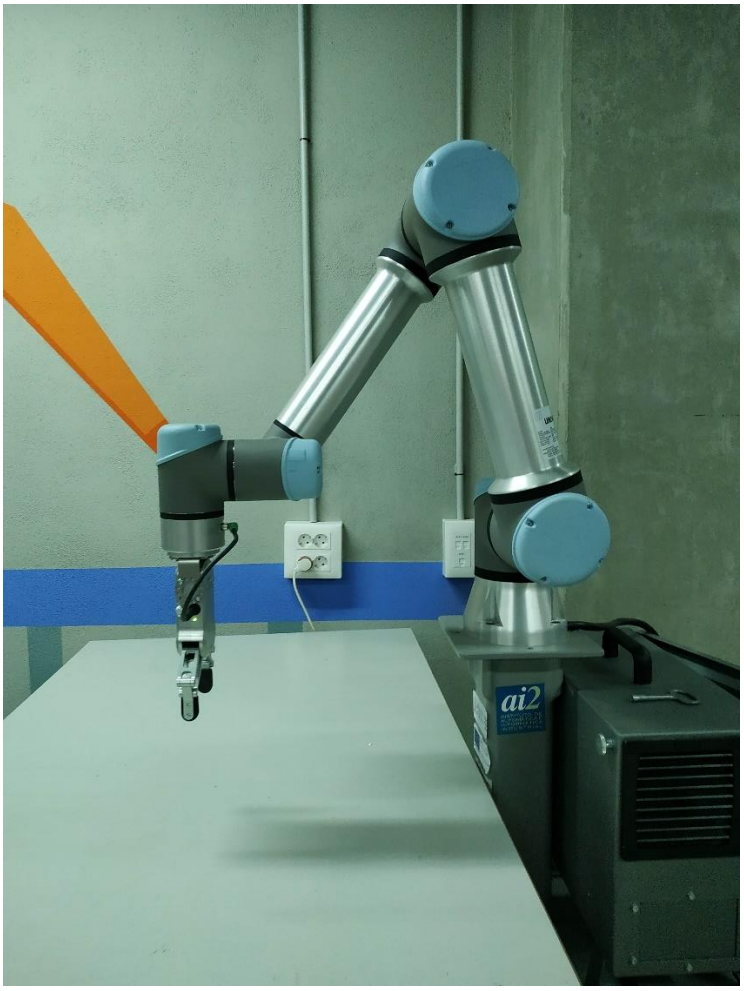


Figura 12. Robot UR5e del instituto ai2 para el desarrollo del TFM.

1.4 Objetivos.

Como se ha visto en los apartados anteriores los robots colaborativos están adquiriendo mucha importancia en la industria y el número de cobots presentes en la industria va a seguir aumentando de forma considerable en los próximos años. Uno de los motivos es la facilidad que presentan para ser programados.

Existen diferentes métodos de programación, una posible clasificación es atendiendo al sistema utilizado para indicarle la secuencia de acciones que debe reproducir. Según este criterio un robot puede ser programado por: programación textual o programación por guiado.

La programación textual utiliza un lenguaje de programación específico para describir las tareas a realizar por el robot. Dentro de este método existen tres niveles: nivel de robot, nivel de objeto y nivel de tarea.

Por otro lado, la programación por guiado consiste en hacer al robot realizar la tarea e ir almacenando las configuraciones para que el robot pueda repetirlas de forma automática. Dentro de este método existen 3 formas diferentes: guiado

pasivo (programador mueve físicamente el robot), guiado pasivo por maniquí (programador mueve un maniquí del robot más pequeño y ligero y el robot reproduce los movimientos) y guiado activo (programador mueve el robot mediante un joystick).

El objetivo del Trabajo Fin de Máster es diseñar una aplicación que permita una programación del robot colaborativo UR5e sencilla e intuitiva para el usuario, esta programación se realizará mediante un dispositivo externo, en concreto un joystick. Se pretende que esta aplicación pueda ser utilizada por personas que no estén familiarizadas con la programación de robots. Para lograr este objetivo final se han planteado los siguientes objetivos intermedios:

- Analizar las posibilidades de comunicación que ofrece el UR5e.

- Diseñar e implementar una aplicación sencilla e intuitiva que permita mover el cobot y grabar diferentes trayectorias para su posterior reproducción utilizando un mando de la consola Nintendo® Wii™. Este se comunica con el PC mediante bluetooth y este se comunica con el robot por TCP/IP.

- Diseñar y construir un mando que permita mover el cobot y grabar diferentes trayectorias. Para ello se utilizará un Arduino Mega que se comunicará directamente con el robot mediante el protocolo TCP/IP.

2. Requerimientos del trabajo.

En el presente apartado se van a presentar los elementos necesarios para el desarrollo del Trabajo Fin de Máster.

2.1 Robot colaborativo UR5e.

En la caja de Universal Robots encontrarás el UR5e junto a la caja de control y la consola de programación. En la página web de Universal Robots hay disponible mucha información sobre sus productos. Además, hay una formación interactiva con 8 módulos que va desde desembalar el robot de la caja, presentándote los elementos que hay en su interior y sus características, hasta realizar una pequeña aplicación de pick & place [5].

Una de las ventajas de trabajar con Universal Robots (UR) es que al ser una empresa líder en el sector de robots colaborativos existe una comunidad muy grande a su alrededor. Esto hace que la cantidad de material sobre sus robots sea elevada, material proporcionado tanto por el propio UR como de terceros. Es sencillo encontrar ejemplos de aplicaciones o encontrar la solución a problemas que puedan surgir, además el foro de UR es bastante activo. En definitiva, UR pone gran empeño por su parte para que utilizar sus robots te resulte sencillo. Para la realización de este trabajo se han consultado algunos de los manuales que ofrece UR en su página web: manual de usuario de los robots e-series[6], manual de software (PolyScope v5.1)[7] y manual de URScripts[8].

2.1.1 UR5e.

El UR5e se trata de un robot colaborativo de la empresa Universal Robots. Este es el de tamaño mediano de los que ofrece la empresa UR en el mercado, pesa 18.4 Kg, la carga útil es de 5 Kg, su alcance es de 850 mm y su huella es de 149 mm, construido con aluminio y plásticos de PP.

Dispone de 6 articulaciones, en la siguiente tabla se muestra el radio de acción y la velocidad máxima de cada una de ellas.

Eje del brazo robot	Radio de acción	Velocidad máxima
Base	± 360°	± 180°/s
Hombro	± 360°	± 180°/s
codo	± 360°	± 180°/s
Muñeca 1	± 360°	± 180°/s
Muñeca 2	± 360°	± 180°/s
Muñeca 3	± 360°	± 180°/s

Tabla 2. Radio de acción y velocidad máxima que presentan las articulaciones del UR5e.

El UR5e puede trabajar en un rango de temperaturas entre 0-50°C. Su repetibilidad es de ± 0.1 mm. Presenta 15 funciones avanzadas de seguridad regulables, probadas de acuerdo con las normas EN ISO 13849:2008 PL d.

Consume una energía mínima de 90 W, una estándar de 150 W y una máxima de 325 W.

La calificación IP (*Ingress Protection)* es IP54, el 5 indica que el dispositivo es estanco al polvo. El 6 indica que está protegido frente a chorros potentes de agua. Además, el robot debe estar en una sala limpia de clase 5 según la ISO. El UR5e es relativamente silencioso, produce un ruido de 72 dB.

Dispone de puertos E/S en herramienta con 2 entradas digitales, 2 salidas digitales y 2 entradas analógicas. También dispone de fuente de alimentación en herramienta de 12 V / 24 V y 600 mA. El conector de la herramienta es de tipo M8.

Puertos de E/S en herramienta	
Entradas digitales	2
Salidas digitales	2
Entradas analógicas	2
Salidas analógicas	0

Tabla 3. Puertos E/S en herramienta.

2.1.2 Caja de control.

La caja de control tiene una clasificación IP de IP20, el 2 significa que el tamaño del objeto entrante debe se <50 mm. El 0 indica que no está protegido frente a líquidos, por lo que el agua entrará fácilmente. Además, la caja de control debe estar en una sala limpia de clase 6 según la ISO. Esta puede trabajar en un rango de temperaturas entre 0-50°C y se considera que es silenciosa, ya que produce un ruido de <65dB. La caja de control debe estar conectada a una fuente de alimentación de 100 – 240 V CA, 50 – 60 Hz. Esta pesa 15 Kg y es de acero.

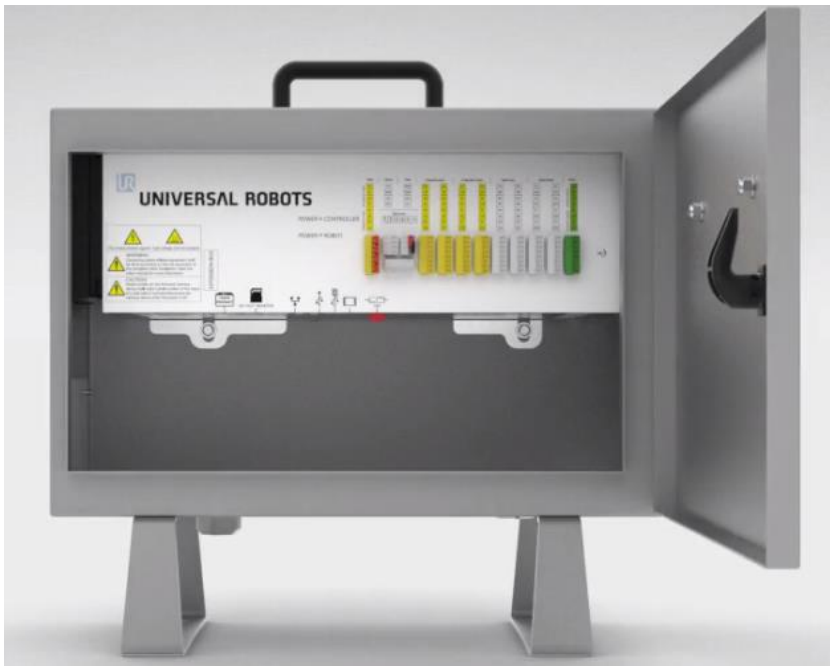


Figura 13. Caja de control del UR5e.

Dentro de la caja se encuentran la placa madre, la tarjeta SD y el tablero de seguridad. Este último maneja todas las E/S desde el controlador y la conectividad con los equipos periféricos como: dispositivos de seguridad, sensores, interfaz de la máquina y pulsadores. Además, hay una entrada de seguridad redundante dedicada que sirve para conectar la parada de seguridad externa y la parada de protección. El tablero de control de seguridad también dispone de un microordenador de computación rápida, ethernet y conexión USB. Por su parte la tarjeta SD contiene todo el software desde el S.O Linux , la interfaz de programación de UR (PolyScope) y todos los programas.

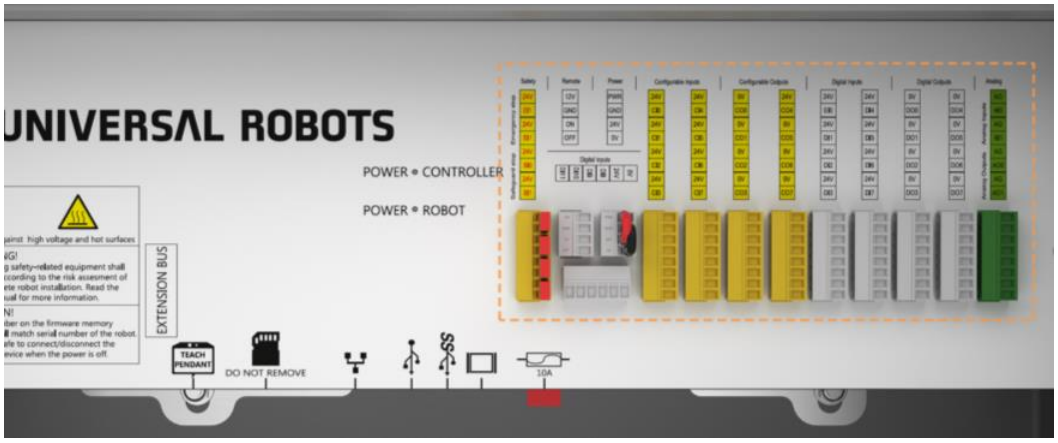


Figura 14. Tablero de seguridad de la caja del controlador.

La caja de control dispone de los puertos e interfaces de comunicación que dispone el robot, este puede comunicarse mediante: TCP/IP 100 Mbit, Modbus TCP, Profinet, EthernetIP.

Dispone de puertos E/S con 16 entradas digitales, 16 salidas digitales, 2 entradas analógicas y 2 salidas analógicas. También dispone de E/S de fuente de alimentación de 24 V 2 A.

Puertos de E/S	
Entradas digitales	16
Salidas digitales	16
Entradas analógicas	2
Salidas analógicas	2

Tabla 4. Puertos de E/S en la caja de control.

2.1.3 Consola de programación.

La consola de programación es una pantalla táctil de 12”, pesa 1.5 Kg y es de aluminio y plástico PP. Su clasificación IP es IP20, igual que la caja de control. La consola de programación nos permite encender el robot, al encenderse nos muestra la interfaz gráfica de usuario PolyScope que es muy intuitiva y permite una programación sencilla del robot y controlar las señales de los dispositivos periféricos. El funcionamiento de la interfaz de usuario PolyScope se explicará más adelante.

Figura 15. Consola de programación de UR5e.

2.1.4 Herramienta del cobot.

El presente trabajo consiste en guiar y grabar trayectorias en el cobot UR5e mediante un joystick, para hacer una aplicación más completa se ha añadido una herramienta cuyo estado también se puede modificar y grabar para su posterior reproducción. Este trabajo no tiene como objetivo realizar ninguna tarea concreta con el UR5e, sino como se dijo anteriormente poder guiarlo y grabar trayectorias, por lo no se ha utilizado una herramienta específica para la tarea deseada, sino la que estaba disponible en el Instituto ai2. En definitiva, se ha tenido en cuenta en la aplicación diseñada la posibilidad de que haya una herramienta conectada y se han creado los métodos necesarios para poder accionarla durante el guiado y grabar su estado para que el robot lo reproduzca.

La herramienta utilizada es la pinza RG2 de OnRobot, esta es un efector final que está diseñada para integrase perfectamente con los robots colaborativos de Universal Robots. Una de sus ventajas es que no necesita cables externos, por lo que el brazo robot puede moverse libremente sin preocuparse de dónde se encuentran los cables. Además, su instalación es muy sencilla y es muy fácil de programar desde la interfaz de usuario PolyScope, desde donde se puede controlar el movimiento y la fuerza de la pinza. También, la RG2 puede medir el ancho del objeto y detectar si ha agarrado el objeto, esta información se envía al robot que puede realizar ciertas acciones en función de las señales recibidas.

Figura 16.Pinza RG2 de OnRobot.

Del datasheet de la pinza se obtiene las siguientes especificaciones técnicas:

Technical data	Min	Typical	Max	Units
Total stroke (adjustable)	0	-	110	[mm]
Finger position resolution	-	0,1	-	[mm]
Repetition accuracy	-	0,1	0,2	[mm]
Reversing backlash	0,2	0,4	0,6	[mm]
Gripping force (adjustable)	3	-	40	[N]
Gripping force accuracy	± 0,05	± 1	± 2	[N]
Operating voltage*	10	24	26	[V DC]
Power consumption	1,9	-	14,4	[W]
Maximum Current	25	-	600	[mA]
Ambient operating temperature	5	-	50	[°C]
Storage temperature	0	-	60	[°C]
Product weight	-	0,65	-	[kg]

*At 12V the gripper runs at approximately half the normal speed

Figura 17. Especificaciones técnicas de la pinza RG2.

Sus dimensiones mecánicas:

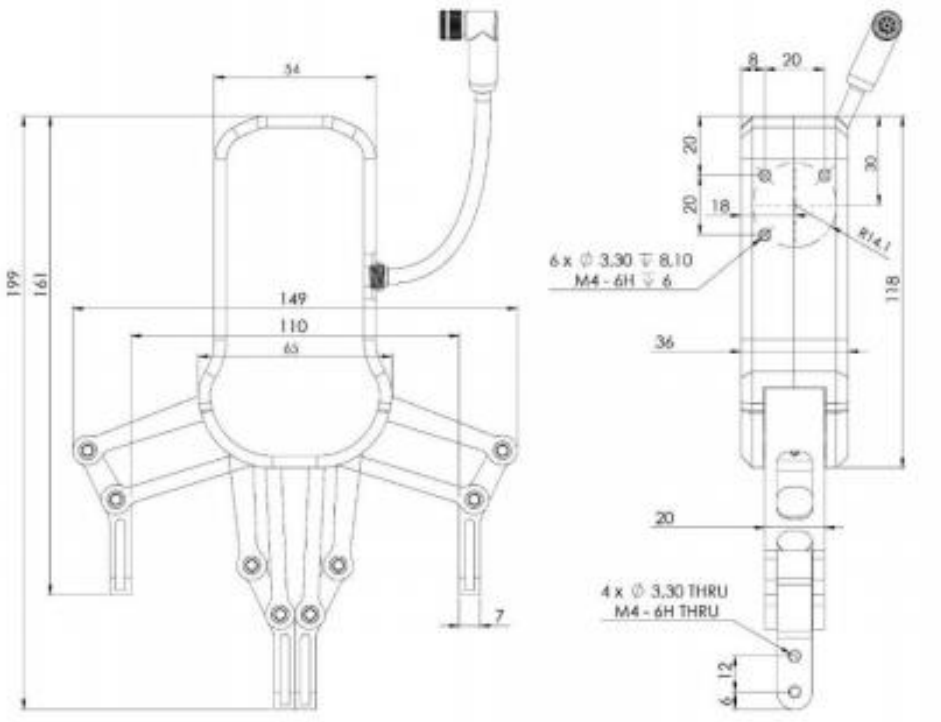


Figura 18. Dimensiones mecánicas de la pinza RG2.

Y el conector de la herramienta, PinOut:

Pin	Wire	UR I/O	UR I/O V3
1	White	AI2	Tool analog input 2
2	Brown	AI3	Tool analog input 3
3	Green	DI9	Tool input 1
4	Yellow	DI8	Tool input 0
5	Gray	Power	24V DC
6	Pink	DO9	Tool output 1
7	Blue	DO8	Tool output 0
8	Red	GND	0V DC

Figura 19. PinOut de la pinza RG2.

2.2 Arduino.

Arduino es una compañía de código abierto basada en software y hardware muy sencillo de utilizar. Arduino se basa en una placa electrónica de hardware libre que integra un microcontrolador reprogramable y unos pines que le permiten comunicar al microcontrolador con diferentes sensores y actuadores de manera sencilla.

Figura 20.Logo de Arduino.

Arduino nace en el año 2005 en el Instituto de Diseño Interactivo de Ivrea como un proyecto enfocado para estudiantes sin experiencia en electrónica y programación. En un principio la idea era fabricar una placa barata para uso interno en la escuela, más tarde este proyecto se abrió al público para que todo el mundo pudiera participar en la evolución de este. La apertura del proyecto a todo el mundo de la placa Arduino es una de las causas de su éxito. Hoy en día existe una gran comunidad entorno a Arduino que acoge a estudiante, aficionados y profesionales. Esta gran comunidad hace que realizar un proyecto con esta placa resulte sencillo, ya que se añade nuevo contenido casi a diario, hay mucha divulgación y foros activos donde se resuelven las dudas que te puedan surgir. Además, en internet existen infinidad de cursos, proyectos, tutoriales, documentos de consulta y mucho más que hace que trabajar con Arduino sea muy sencillo.

Otras de las grandes ventajas que aporta Arduino y que son motivo de su éxito es que todas las placas Arduino son completamente de código abierto, por lo que los usuarios pueden construirlas de manera independiente para adaptarlas a sus necesidades, motivo por el cual existen numerosas placas no oficiales para diferentes propósitos. El software también es código abierto y además existe mucho material gracias a las aportaciones de los usuarios, hay disponibles numerosas librerías de software de terceros que permiten realizar infinidad de aplicaciones.

Por otro lado, el software de Arduino es sencillo de utilizar por principiantes, lo que lo hace atractivo para nuevos usuarios, pero es lo suficientemente amplio para usuarios más avanzados. Arduino dispone de un entorno de programación sencillo e intuitivo para poder programar aplicaciones para sus diferentes placas, este es el Arduino IDE. Además, su lenguaje de programación basado en C++ resulta muy sencillo. Sobre este entorno de programación y sobre el lenguaje de programación se hablará más adelante.

A continuación, se van a mostrar algunas ventajas importantes de Arduino:

- El precio, las placas Arduino son baratas en comparación con otras plataformas de microcontroladores.
- Software de código abierto.
- Hardware de código abierto.
- Entorno de programación sencillo e intuitivo (Arduino IDE).
- Multiplataforma, el software de Arduino IDE se puede ejecutar en Windows, Mac OS x y Linux.
- Lenguaje de programación sencillo.
- Gran comunidad, por lo que existe una gran cantidad de conocimiento accesible para poder desarrollar infinidad de proyectos.
- Reutilizable, es sencillo montar y desmontar tus proyectos, lo que facilita utilizar la placa Arduino para diferentes proyectos.

2.2.1 Funcionamiento.

En este aparado se va a explicar de forma muy resumida cómo funciona Arduino. La placa de Arduino se basa en un microcontrolador ATMEL, un microcontrolador es básicamente un circuito integrado programable en el que se pueden grabar instrucciones. Este está formado por diferentes bloques funcionales que cumplen con una tarea concreta. Un microcontrolador dispone es su interior de las tres principales unidades funcionales de un computador: unidad central de procesamiento, memoria y periféricos de E/S.

El conjunto de instrucciones que se graban en el microcontrolador se debe escribir en el lenguaje de programación de Arduino, para lo que se puede usar el entorno de programación Arduino IDE. Una vez creado el programa con las diferentes instrucciones se cargan al microcontrolador donde estas interactuaran con los circuitos de la placa para llevar a cabo la tarea para la que se ha diseñado.

El microcontrolador de Arduino dispone de una interfaz de entrada que permite la conexión de diferentes periféricos, cuya información podrá ser utilizada por el microcontrolador. También dispone de una interfaz de salida que permite llevar información que se ha procesado en el microcontrolador a otros periféricos.

2.2.2 Modelos de Arduino.

El hecho de que el hardware de Arduino sea abierto ha provocado que se pueda encontrar diferentes tipos de placas, aunque estas comparten su diseño básico. Existen placas de Arduino de diferentes tamaños, colores y formas y con características especiales según el proyecto que se quiera realizar. Hay Arduinos para aplicaciones IoT, impresión 3D, etc.

A parte de los modelos generados por la comunidad existen diferentes placas Arduino oficiales. Estas están pensadas para diferentes propósitos y tienen una serie de características que hace que unas sean más adecuadas que otras según el proyecto que se esté desarrollando. Alguna de estas características son el tamaño, el número de pines de E/S, el modelo del microcontrolador, etc. A pesar de estas diferencias todas las placas llevan un microcontrolador AVR marca ATMEL, lo que significa que tienen en común muchas de sus características software, así como arquitectura y librerías. Algunos de los modelos oficiales son: Arduino NANO, Arduino MICRO, Arduino UNO, Arduino MEGA, Arduino LEONARDO, Arduino YUN, etc.

Arduino UNO REV3.

Esta es la placa recomendada por Arduino para introducirse en el mundo de la electrónica. Este es el modelo de referencia de Arduino a partir de la cual se ha evolucionado a otras versiones.

El microcontrolador de la placa Arduino UNO es el ATmega328. La memoria flash es de 32 KB y su tensión de funcionamiento es de 5 V. La placa dispone de 14 pines de E/S digital (6 de ellos se pueden utilizar como salidas PWM) y 6 entradas analógicas. También dispone de un cristal de cuarzo de 16 MHz, un encabezado ICSP (In Chip Serial Programmer), un botón de reinicio, conexión USB y un conector de alimentación.

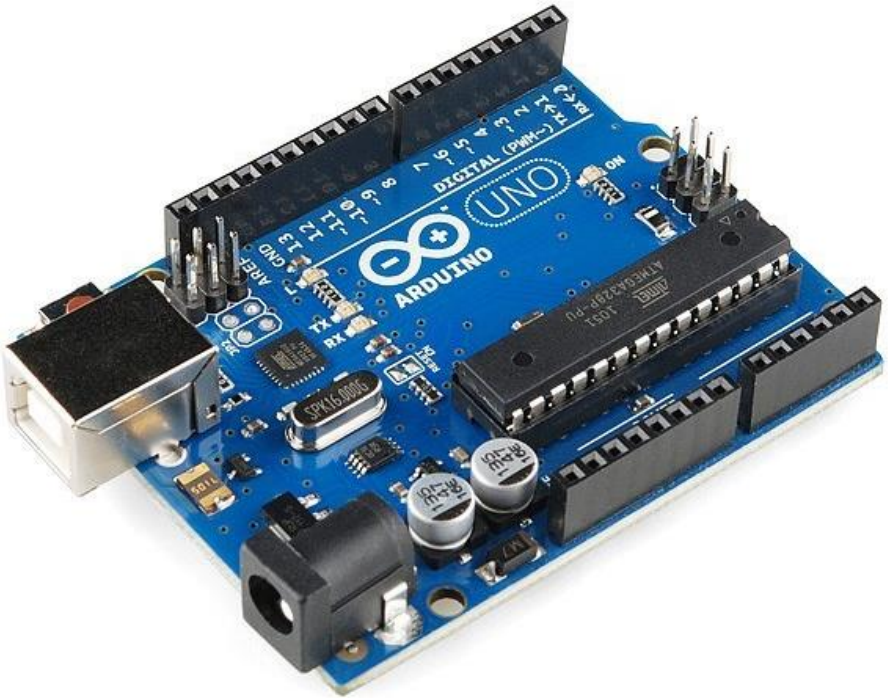


Figura 21. Arduino UNO.

Arduino NANO

La placa de Arduino NANO es una placa similar a la de Arduino UNO, pero más compacta. Esta placa lleva el mismo microcontrolador que el Arduino UNO, ATmega238, pero en su versión SMD que reduce su tamaño de forma considerada. Dispone de 14 pines de E/S digitales (de los cuales 6 se pueden utilizar como salidas PWM),8 entradas analógicas, un encabezado ICSP y un botón de reinicio. Este modelo no dispone de conector de alimentación, por lo que funciona con un cable USB Mini-B.

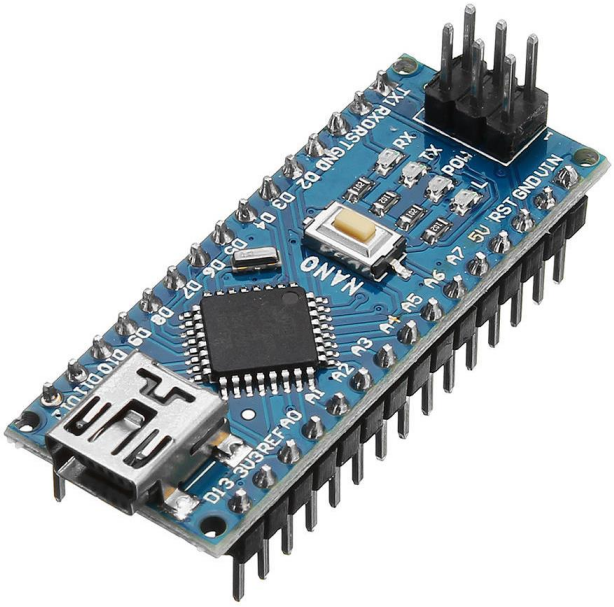


Figura 22. Arduino NANO.

Arduino MEGA 2560.

Esta placa está diseñada para la realización de proyectos más complejos. Su microcontrolador es en ATmega2560. Dispone de 54 pines de E/S digitales (de los cuales 15 se pueden utilizar como salida PWM, 16 entradas analógicas, 4 UART (puertos serie de hardware), un oscilador de cristal de 16 MHz, conexión USB, conector de alimentación, un encabezado ICSP y un botón de reinicio. El Arduino MEGA 2560 es una versión mejorada del Arduino MEGA. Su memoria flash es de 256 KB (de los cuales 8 KB utilizados por el gestor de arranque). Esta es una de las placas más potentes de la familia Arduino. Además, esta presenta una gran ventaja y es que es compatible con la mayoría de los shields para Arduino UNO.

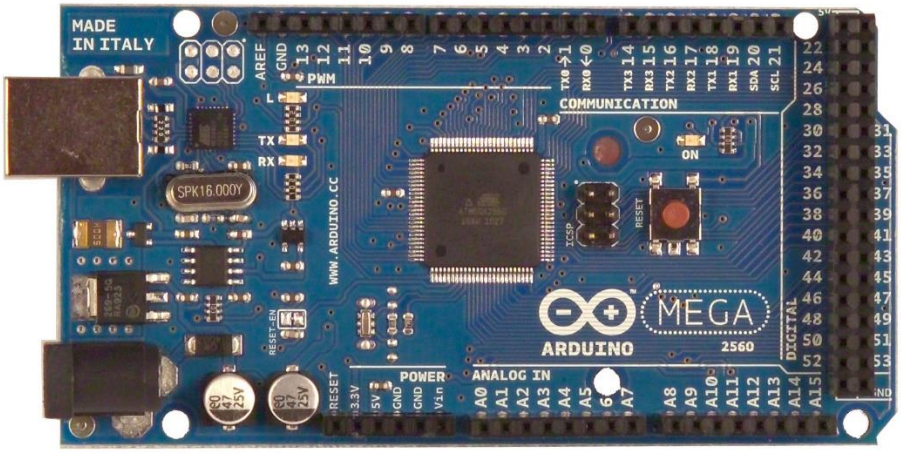


Figura 23. Arduino MEGA 2560.

En la Tabla 5 se muestra una comparativa entra las tres placas de Arduino que se acaban de presentar.

	Arduino UNO	Arduino NANO	Arduino MEGA
Microcontrolador	ATmega328	ATmega328	ATmega2560
Tensión de funcionamiento	5 V	5 V	5 V
Voltaje de entrada (recomendado)	7-12 V	7-12 V	7-12 V
Voltaje de entrada (límite)	20 V	-	20 V
Pines E/S digitales	14	14	54
Pines PWM	6	6	15
Pines de entrada analógica	6	8	16
Corriente DC por pin E/S	40 mA	40 mA	40 mA
Corriente DC para Pin de 3.3 V	50 mA	-	50 mA
Memoria flash	32 KB (0.5 KB gestor de arranque)	32 KB (2 KB gestor de arranque)	256 KB (8 KB gestor de arranque)
SRAM	2 KB	2 KB	8 KB
EEPROM	1 KB	1 KB	4 KB
Velocidad reloj	16 MHz	16 MHz	16 MHz
LED_BUILTIN	13	-	13
Longitud	68.6 mm	45 mm	101.52 mm
Anchura	53.4 mm	18 mm	53.3 mm
Peso	25 g	7 g	37 g

Tabla 5. Comparación entre la placa de Arduino UNO, NANO y MEGA.

Como se puede observar las placas tienen las mismas características eléctricas y sus principales diferencias son el número de pines de E/S digital, el número de pines PWM, el número de pines de entrada analógica, la memoria flash y su tamaño y peso.

Para poder seleccionar una placa se debe tener en cuenta el proyecto que se va a realizar y analizar qué placa se adapta mejor a las necesidades de este. En este proyecto la placa Arduino se va a utilizar para construir un mando para poder guiar y grabar trayectorias con el UR5e. Para ello la placa se comunicará directamente con el cobot mediante TCP, decisión que se explicará con más detalle más adelante, por lo que se necesitará conectar un módulo Ethernet. Además, el mando está formado por 10 botones para realizar las diferentes operaciones y un joystick. En resumen, se va a necesitar 10 entradas digitales para los botones, 2 entradas analógicas para el joystick y el módulo ethernet elegido se comunica con la placa mediante SPI por lo que se necesitarán 4 entradas digitales para su conexión (las placas Arduino disponen de soporte hardware vinculados a ciertos pines, estos varían según el modelo). Otra cosa para tener en cuenta al conectar el módulo de ethernet es que Arduino se utilizan

el pin 10 y el 4 para seleccionar el módulo ethernet y la tarjeta SD (ambos son usados como SS en la comunicación), por lo que no pueden trabajar simultáneamente y por tanto como en el presente proyecto no se va a utilizar la tarjeta SD el pin 4 debe quedar libre para evitar problemas de comunicación con el módulo ethernet.

Teniendo en cuenta todo lo dicho anteriormente se concluye que para la realización del proyecto se necesitan:

- 15 pines de entrada digital (10 para los botones, 4 para el módulo ethernet y el pin 4 debe quedar libre.
- 2 entradas analógicas.
- Alimentación a 5 voltios.

Teniendo en cuenta las necesidades del proyecto es fácil darse cuenta de que la principal limitación viene impuesta por el número de entradas digitales necesarias ya que la única placa Arduino que puede tener 15 entradas digitales es el arduino Mega. Además, utilizar los pines digitales 0 y 1 no es muy cómodo porque para programar se usan esos pines por lo que se debe desconectar el circuito conectado en estos. Teniendo en cuenta que los circuitos conectados al Arduino estarán dentro del mando, lo que dificulta su accesibilidad, se ha considerado dejar estos pines libres, lo que reduce el número de pines digitales disponibles.

Por tanto, para el presente proyecto se va a utilizar la placa **Arduino MEGA 2560.** Aunque su tamaño es un inconveniente a la hora de la construcción del mando, una placa como Arduino UNO o NANO son más adecuadas para poder construir un mando de tamaño más reducido, esta es la única placa que cumple con los requisitos necesarios para la realización del proyecto. Además, el tamaño del mando no es una característica crítica, y utilizando la placa de Arduino MEGA el tamaño del mando es adecuado para poder utilizarlo cómodamente. Otras características como la memoria no se han tenido en cuenta porque cualquiera de las placas cumple en este aspecto con los requisitos del proyecto.

2.2.3 Modulo ethernet.

Como se ha comentado en el apartado anterior para la realización del presente proyecto se va a comunicar el mando realizado con la placa Arduino MEGA con el UR5e mediante el protocolo TCP/IP. Para ello se debe utilizar un módulo que permita la conexión, ya que el Arduino por sí solo no puede. Existen diferentes alternativas para establecer una conexión por TCP/IP con Arduino. A continuación, se presentarán los principales módulos encontrados y se justificará la elección tomada.

ENC28J60

Un controlador de Ethernet de Microchip Technology Inc que permite dar acceso a la placa Arduino a una red.

Figura 24.Módulo ENC28J60.

Tras investigar sobre las características del módulo ENC28J60 se encontraron las siguientes ventajas:

- Se controla a través del bus SPI, por lo que la conexión es sencilla.
- Soporta velocidades de 10 Mbits/s
- Soporta modo Full-Duplex y Half-Duplex con la capacidad de detectar errores de polaridad y corrección automática.
- Opera con 3.3 V, pero tolera señales de 5 V.
- Incorpora filtrados de paquetes para limitar número de paquetes entrantes.
- Existen varias opciones de librerías para hacerlo trabajar con Arduino.
- Es más barato que las otras alternativas analizadas. Se ha encontrado este módulo por un precio sobre los 4 €.

Y las siguientes desventajas:

- No dispone de pila TCP/IP por hardware, por lo que es más complejo de utilizar.
- Consume mucha más memoria que el W5100.

Por tanto, la principal ventaja de este módulo respecto al resto de alternativas es su precio. Es una buena opción para aplicaciones básicas y existe mucho material en internet para trabajar con él, pero se descartó por no ser la mejor opción para el desarrollo del presente proyecto.

W5100

El W5100 es un controlador de ethernet diseñado por Wiznet. Este nos permite conectar la placa de Arduino a una red.

Algunas de las ventajas del W5100 son:

- Dispone de pila TCP/IP por hardware y buffer interno de 16 KB para Tx/Rx lo que libera de carga al procesador. Esto es una gran ventaja respecto al ENC28J60.

- Permite la conexión por SPI, lo que hace que su conexión sea muy sencilla.

- Soporta velocidades de 10/100 Mbits/s.

- Soporta modo Full-Duplex y Half-Duplex con la capacidad de detectar errores de polaridad y corrección automática.

- La pila TCP/IP soporta conexiones TCP, UDP, IPv4 y hasta 4 conexiones de forma simultánea.

- Librería para su uso integrada en el entorno de programación Arduino IDE.

Como desventaja en comparación con el ENC28J60 es que su precio es un poco más elevado, unos 6€, pero las ventajas que aporta son lo suficientemente importantes como para compensar esta pequeña subida de precio.

Por tanto, el controlador W5100 es la una buena opción para el desarrollo de este proyecto, ya que ofrece muchas ventajas frente a otras alternativas analizadas.

El W5100 se puede encontrar en distintos módulos en el mercado, estos módulos están diseñados para facilitar la conexión del W5100 con la placa de Arduino. Uno de los más importantes es el Ethernet shield cuya principal ventaja es su facilidad de montaje, ya que se acopla directamente sobre la placa de Arduino.

Figura 25. Módulo Ethernet Shield montado sobre la placa de Arduino UNO.

Dispone del conector estándar para ethernet el RJ45 y lector de tarjeta micro SD. Este módulo solo es compatible con la placa de Arduino UNO y MEGA, por lo que se puede utilizar en nuestro proyecto. Otra opción es utilizar un módulo independiente con el W5100, este se puede utilizar con cualquier placa.

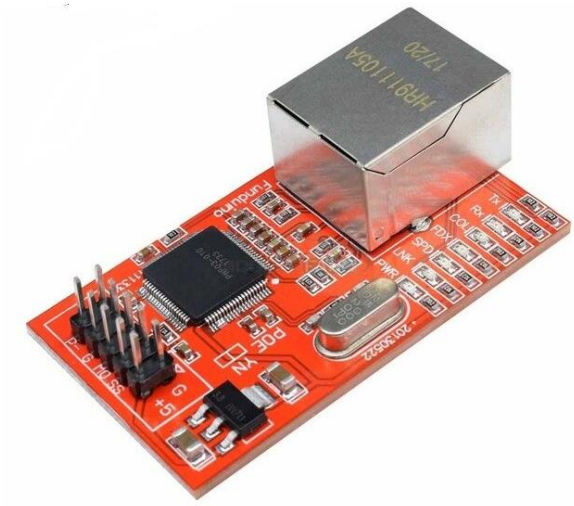


Figura 26.Modulo independiente con W5100.

Aunque la conexión es más complicada que el módulo de ethernet shield, ya que habrá que cablearla, el tamaño reducido de este módulo independiente se amolda mejor a las necesidades del proyecto. Además, al no estar obligado a estar en una posición determinada para conectarlo permite elegir la posición del conector RJ45, esto junto con su menor tamaño permite una mayor flexibilidad a la hora de colocarlo en el interior del mando que se va a construir. Por estos motivos se ha elegido este **módulo independiente con W5100.**

2.3 Mando Nintendo® Wii™.

Uno de los objetivos del presente trabajo es analizar diferentes alternativas para el guiado y la grabación de trayectorias del UR5e mediante un

dispositivo externo. Para ello, además del mando construido con la placa de Arduino se ha utilizado el mando de la Nintendo® Wii™ que ofrece diferentes posibilidades (las diferencias entre ambas soluciones desarrolladas, así como las ventajas y desventajas de la una y la otra se verán en el apartado 3).

La videoconsola Wii™ de Nintendo® se lanzó al mercado en 2006. El objetivo de Nintendo® con el lanzamiento de esta consola era "revolucionar el entretenimiento en casa con nuevas experiencias". Esta videoconsola pretendía ofrecer una experiencia de juego muy diferente de lo que se había visto hasta el momento, y una de las grandes revoluciones fueron sus mandos el wiimote y el nunchuk.

Figura 27. Mando de la Nintendo® Wii™, Nunchuk (izquierda) y wiimote (derecha).

El wiimote dispone un control en forma de cruz y diversos botones, además en su interior incluye un acelerómetro que permite medir la aceleración del mando en los tres ejes. También dispone de un sensor óptico que se utiliza para conocer el punto de la pantalla al que se está apuntando (se ayuda de una barra de infrarrojos conectada a la consola). Este mando también dispone de un altavoz y un pequeño motor que le permite vibrar. Además, tiene una pequeña memoria interna para almacenar información del usuario. El wiimote funciona con 2 pilas AA y dispone de un puerto de expansión donde se puede conectar diversos accesorios como el nunchuk.

El nunchuk dispone de un joystick analógico, un par de botones y un acelerómetro.

Este mando es una buena opción para utilizarlo en el desarrollo de este trabajo ya que incluye diferentes botones, que permiten configurar los diferentes modos de funcionamiento en la aplicación diseñada, y un joystick que permite el guiado del UR5e. Pero una de las características más importante de este mando y que

lo diferencia de otros es que el wiimote se conecta a la Wii™ mediante bluetooth, esto permite conectar el mando al PC de manera sencilla. Además, existen librerías que permiten conectar el mando de la Wii™ con una aplicación Java, lenguaje que se va a utilizar para implementar la comunicación por TCP entre el PC y el UR5e. Por estos motivos se va a utilizar **el wiimote y el nunchuk.**

2.4 Requerimientos de software.

En los apartados anteriores se han presentado los dispositivos necesarios para el desarrollo del presente trabajo. En este apartado se van a presentar los diferentes métodos de programación que se van a utilizar para programar estos dispositivos.

2.4.1 Programación del UR5e.

Existen diferentes métodos para la programación del robot, en este apartado se presentarán alguno de ellos. En concreto, se explicará la programación del UR5e mediante la interfaz PolyScope y mediante scripts ya que son los métodos de programación utilizados para la realización del presente trabajo.

PolyScope

Es la interfaz gráfica de usuario diseñada por Universal Robot y se ejecuta al arrancar la consola de programación. Desde la consola de programación se enciende la caja de control y se arranca el software PolyScope. Una vez encendido PolyScope permite encender y apagar el robot, realizar configuraciones de seguridad como establecer límites del robot, establecer límites de los ejes o establecer una contraseña de seguridad entre otras muchas opciones. También permite configurar la instalación de la herramienta, planos, etc. Una vez se han realizado todas estas configuraciones, en caso de que sean necesarias, la interfaz permite crear nuevos programas, para lo que dispone de un árbol de programa al que se le van añadiendo diferentes nodos de programa. También se puede cargar programas desde un USB. Una vez creado o cargado el programa permite la ejecución de este por parte del robot.

Al arrancar la consola se muestra una pantalla en la que se distinguen tres zonas:

- Zona azul: en la que están los iconos que permiten moverse entre las diferentes pantallas, de Izquierda a derecha son:
    - Ejecutar: permite cargar un programa.
    - Programa: editor para crear o modificar los programas del robot.
    - Instalación: realizar las configuraciones necesarias del brazo.
    - Mover: permite mover el robot desde la consola.

- I/O: permite visualizar y modificar las señales de E/S.
- Log: proporciona datos sobre el robot como temperatura, voltaje e intensidad en cada una de las articulaciones, etc. Además, muestra los mensajes de error o advertencia.
- Nuevo: crear un nuevo programa.
- Abrir: permite ver los programas guardados y abrir uno.
- Guardar: permite guardar el programa y la instalación con el que se está trabajando.
- Modo remoto: permite poner el robot en modo remoto (este icono sale porque se ha habilitado desde ajustes).
- Suma de comprobación: muestra la configuración de seguridad del robot.
- Menú hamburguesa: permite obtener ayuda y realizar ciertos ajustes.

- Zona verde: es la pantalla de inicio, que irá cambiando conforme se navega entre las diferentes pantallas.
- Zona amarilla: botones que controlan la ejecución del programa cargado, permite ajustar la velocidad de movimiento, iniciar el programa, pararlo. En esta zona también está el botón de encendido y apagado del robot.

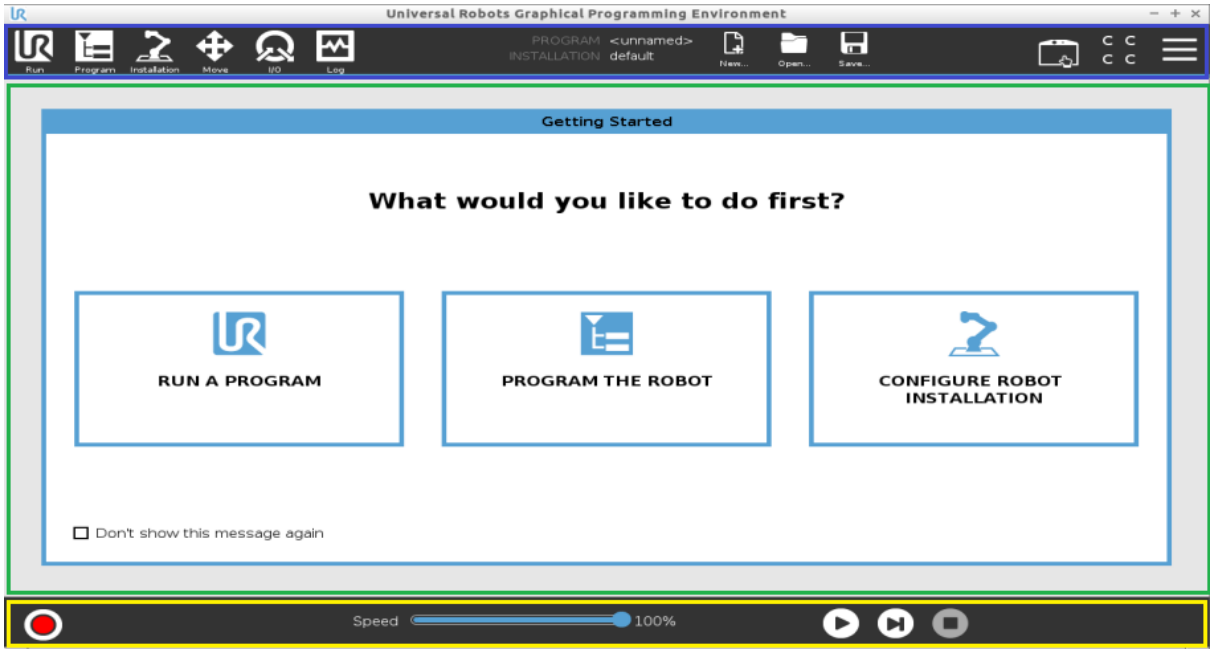


Figura 28. Pantalla inicial PolyScope.

Antes de comenzar a programar es necesario encender el robot, para ello se pulsa el botón rojo situado en la esquina inferior izquierda de la pantalla. Lo que nos lleva al menú de inicialización.

El color del botón nos indica el estado del brazo, este es rojo cuando está apagado.

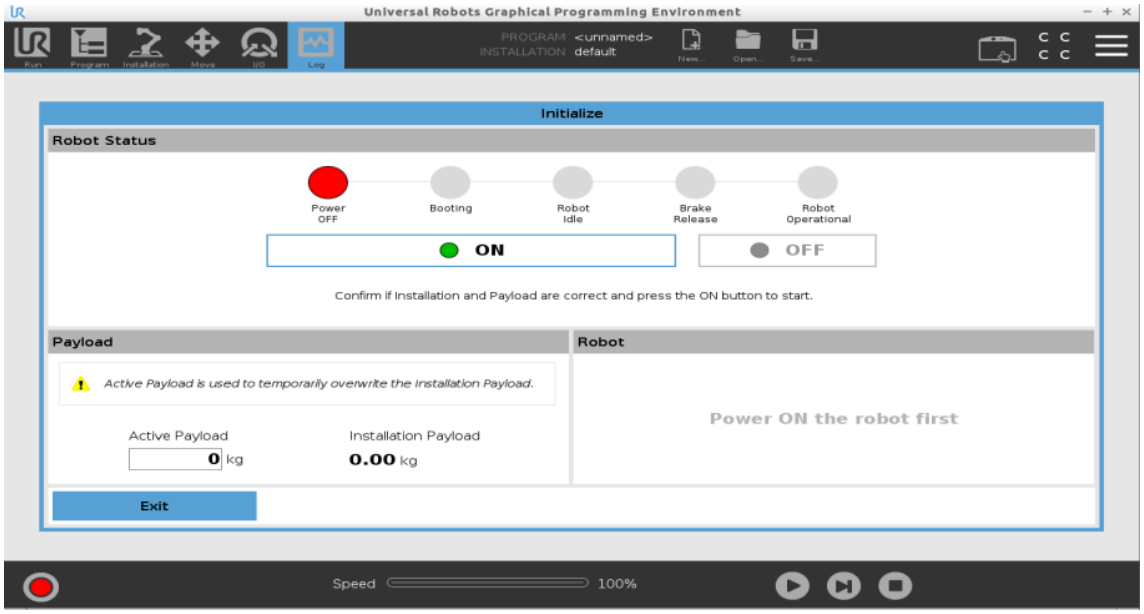


Figura 29.Secuencia de encendido del UR5e, robot apagado.

Amarillo cuando está encendido, pero no está listo para funcionar, recibe tensión, pero los frenos están activados.

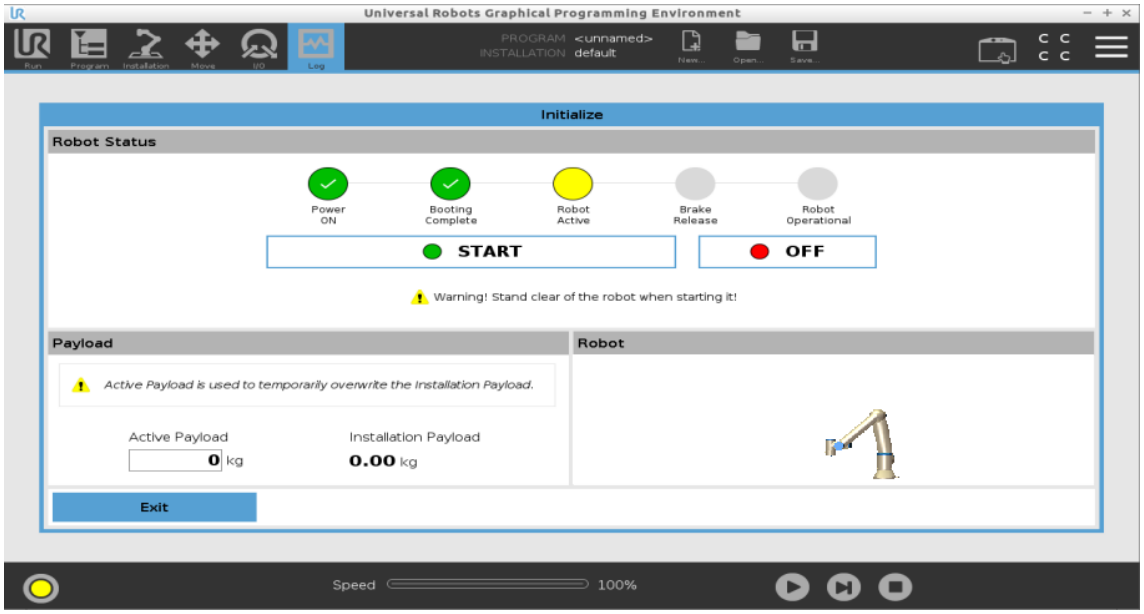


Figura 30.Secuencia de encendido del UR5e, robot encendido, pero no listo para funcionar.

Cuando se vuelve a pulsar el botón de iniciar el brazo libera los frenos, lo que produce unos sonidos y unos movimientos, y el botón se pone verde lo que indica que está encendido y listo para funcionar.

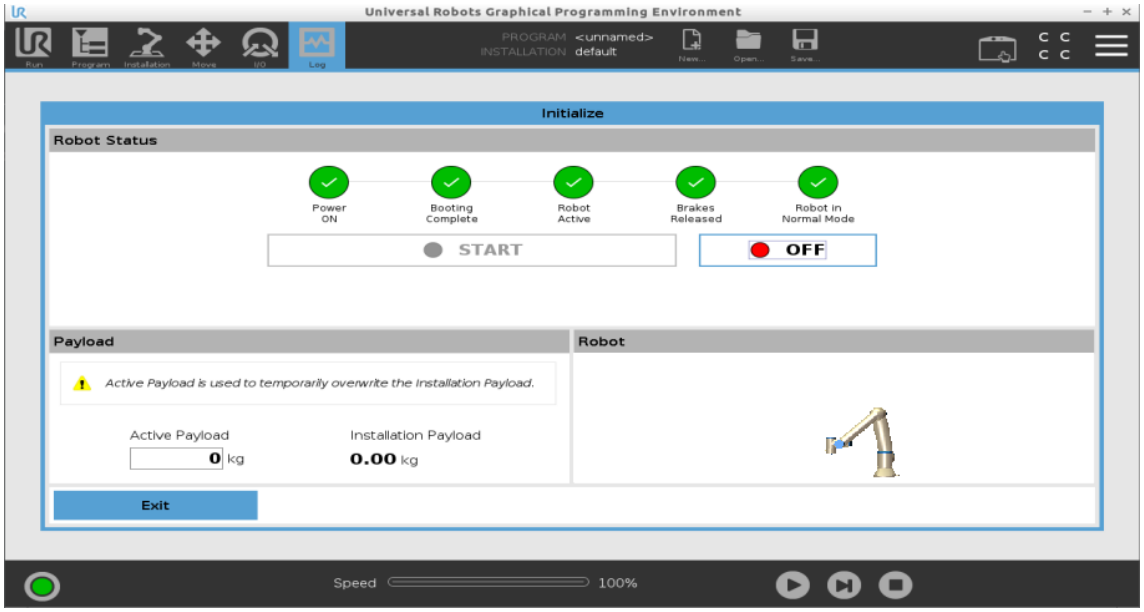


Figura 31.Secuencia de encendido del UR5e, robot encendido y listo para funcionar.

Una vez encendido se puede comenzar a programar el brazo, para ello hay que dirigirse a la pestaña de programación. En esta aparece un árbol de programa vacío donde se van a insertar las diferentes instrucciones y debajo de este una serie de herramientas para editar el código. A la izquierda aparece un menú desplegable con los diferentes comandos agrupados en tres grupos: básicos, avanzados y plantillas. A la derecha aparece una pestaña que aporta información sobre los comandos, información gráfica sobre el estado del robot o información sobre las variables.

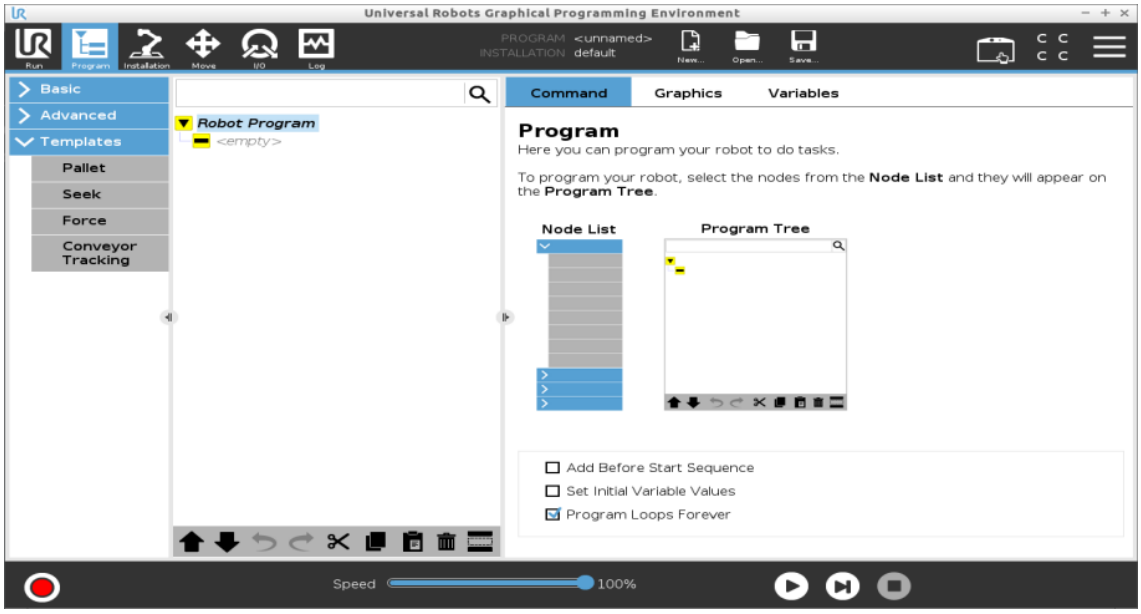


Figura 32. Pestaña de programación de PolyScope.

En la pestaña de comandos se puede añadir un *Before Start sequence,* que es una secuencia de código que solo se ejecutará la primera vez. También se puede inicializar las variables y permite seleccionar si el bucle principal se ejecuta una sola vez o de manera infinita.

En lo que se refiere a los comandos disponibles a continuación se van a presentar de forma resumida:

Nivel básico:

- *Move:* mueve el robot a través de los puntos de paso que se añadan. El movimiento puede ser *MoveJ* , *MoveL* y *MoveP.*
    - *MoveJ:* movimientos calculados respecto al espacio articular. Trayectoria curva de la herramienta.
    - *MoveL:* movimientos calculados respecto al punto central de la herramienta. Trayectoria lineal de la herramienta entre los puntos de paso.
    - *MoveP:* movimiento lineal de la herramienta a velocidad constante. Este permite la opción de realizar un movimiento circular, para lo que hay que definir el punto inicial, un punto medio y el punto final.
- *Waypoint:* son los puntos de paso, se pueden definir moviendo el robot a la posición deseada o mediante una variable.
- *Wait:* para la ejecución del programa durante un tiempo, hasta que se produzca un cambio en una entrada digital, hasta que una variable analógica alcance cierto valor o hasta que se cumpla una condición.
- *Set:* ajustar salidas digitales o analógicas a el valor deseado.
- *Popup:* permite mostrar un mensaje emergente en la pantalla, permite detener la ejecución del programa si se desea.
- *Halt:* detiene la ejecución del programa en este punto.
- *Comment:* permite añadir comentarios.
- *Folder:* sirve para organizar el código, no tiene efecto sobre la ejecución del programa.

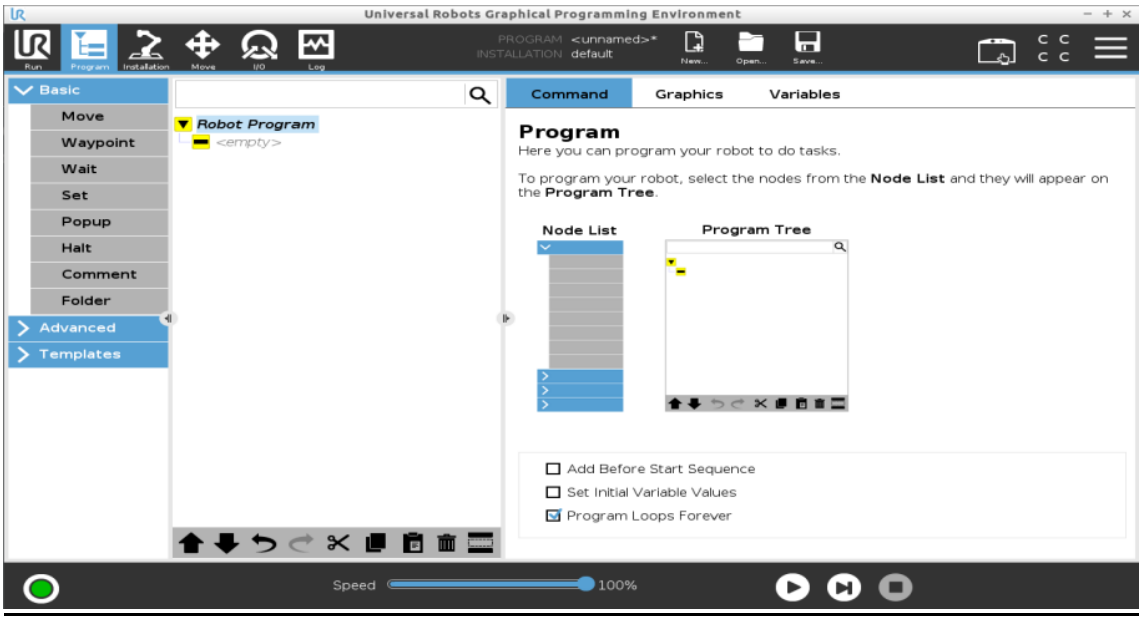


Figura 33. Menú de comandos nivel básico.

Nivel Avanzado:

- *Loop:* permite crear bucles que repetirán los comandos que haya en su interior siempre, X veces o hasta que una expresión se haga verdadera.

- *SubProg:* aquí se puede añadir líneas de comandos que se repiten en varios lugares del código o partes que se quieran proteger.

- *Assignment:* asignar valores a las variables.

- *If:* típica estructura condicional que ejecuta una parte del código cuando se cumple la condición, permite añadir *elseif* y *else.*

- *Script:* permite añadir líneas o archivos de programa en el lenguaje URScript. Esta función es muy útil ya que permite combinar la facilidad de programar en PolyScope con las posibilidades que ofrece los comando de URScript.

- *Event:* si ocurre un evento concreto realiza una serie de instrucciones. Mientras se ejecuta el código correspondiente al evento que ha ocurrido la ejecución del programa continua.

- *Thread:* permite realizar diferentes procesos de forma simultánea. Esto permite una programación concurrente y se ha aprovechado en la realización del presente trabajo.

- *Switch:* estructura switch que realiza unas acciones u otras en función del valor de la condición.

- *Timer:* es un temporizador.

- *Home:* se utiliza para mover el brazo robot a una posición de origen que debe definirse previamente.

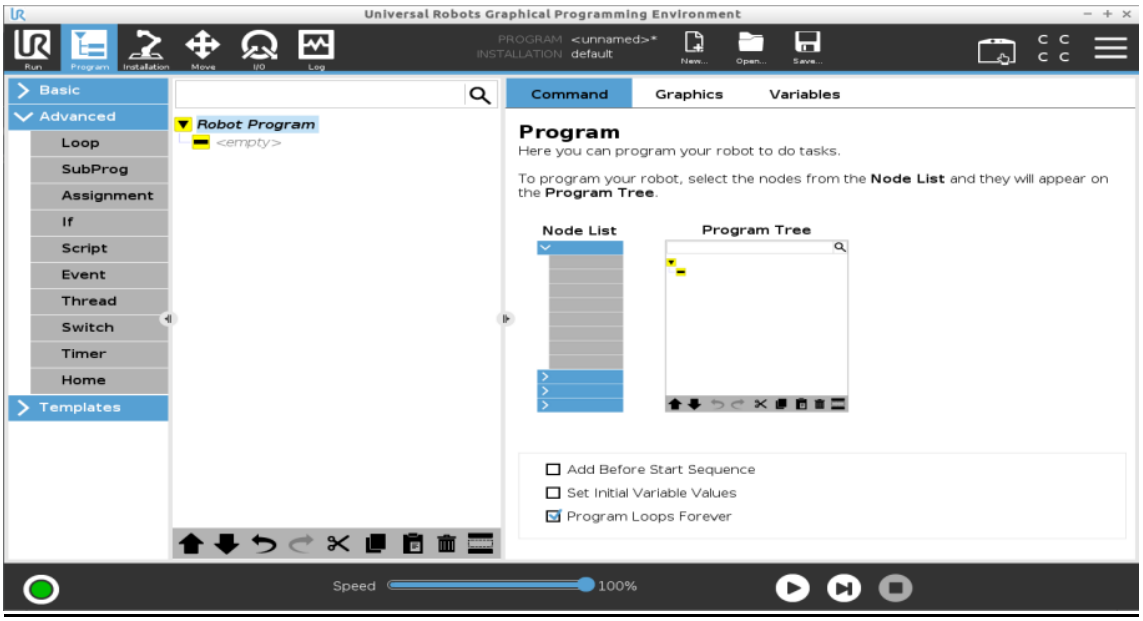


Figura 34. Menú de comandos nivel avanzado.

Nivel de plantilla:

Son unas plantillas que pueden usarse como base para construir ciertos programas. Este menú desplegado se muestra en la Figura 32.

En cuanto a las variables existen dos tipos de variables, las variables de instalación que se pueden utilizar por varios programas y que almacenan su valor cuando el robot y la caja del robot se apagan. Las otras son las variables normales del programa, solo se pueden ejecutar en el programa donde se han definido y su valor se pierde cuando se para el programa. En PolyScope existen diferentes tipos de variables: boolean, int, float, string, pose, list. Estas son las típicas que existen en otros lenguajes de programación, exceptuando el tipo *pose* que es un vector que describe la posición y la orientación en el espacio cartesiano y que presenta la siguiente estructura:

$$p[x,y,z,rx,ry,rz]$$

donde (x,y,z) es el vector posición y (rx,ry,rz) es un vector de rotación.

Esto es un breve resumen de algunas de las muchas posibilidades que ofrece la interfaz PolyScope, Universal Robots dispone de un "*Manual de PolyScope"* donde hay mucha información sobre esta interfaz.

Scripts

Una alternativa a programar el UR5e con PolyScope es programar a nivel de scripts. Estos son programas o líneas de programas que se pueden cargar directamente en el robot o también se pueden cargar en un programa creado en PolyScope.

Los scripts funcionan como un lenguaje de programación estándar, Universal Robots ha desarrollado su propio lenguaje de programación para controlar el robot llamado URScripts. Este es similar a Python. URScripts dispone de variables, instrucciones de control de flujo, permite monitorear las E/S del robot y los movimientos del robot, etc.

Universal Robots proporciona el manual *"The URScripts Programming Language"* donde se explica cómo programar el robot mediante un Script que se ejecute directamente en el robot, explica las diferentes características de este lenguaje, sus posibilidades, su sintaxis, etc. En este apartado no se va a entrar a explicar de forma detallada las diferentes características del URScripts, sino que se va a presentar brevemente algunas de las funciones que se van a utilizar para el desarrollo de la aplicación.

En nuestra aplicación no se va a programar directamente con scripts, sino que se van a utilizar diferentes funciones que se incluirán en el programa desarrollado en PolyScope. A continuación, se mencionarán algunas de las funciones más interesantes que dispone URScripts:

- *force_mode:* pone el robot en modo fuerza. Se debe definir los argumentos de vector de posicion de la herramienta, un vector de seis posiciones con 0 o 1 indicando en qué eje se realiza la tarea, vector de seis posiciones con la fuerza que se aplica en cada eje, el tipo y la máxima velocidad permitida en cada eje.

- *Freedrive_mode:* pone el robot en modo libre.

- *movej:* mueve el robot en el espacio de las articulaciones. Debe definirse argumentos de la posicion de las articulaciones (q), la aceleración, la velocidad, el tiempo y el radio de la curva que puede doblar el robot para alcanzar el punto objetivo.

- *movel:* mueve el robot respecto a la herramienta. Debe definirse argumentos de la posicion, la aceleración, la velocidad, el tiempo y el radio de la curva que puede doblar el robot para alcanzar el punto objetivo.

- *servoj:* mueve el robot en el espacio de las articulaciones. Es útil para el control online del robot, ya que suaviza la trayectoria. Deben definirse algunos argumentos como la posición de las articulaciones (q), aceleración y velocidad.

- *Speedj:* velocidad de las articulaciones. Se definen como argumentos la velocidad de las articulaciones (rad/s), la aceleración de las articulaciones (rad/s^2) y el tiempo que tarda en acabar la función.

- *speedl:* velocidad de la herramienta. Se definen como argumentos la velocidad de la herramienta (m/s), la aceleración de la herramienta (m/s^2), el tiempo que tarda en acabar la función y de manera opcional la velocidad de rotación (rad/s^2).

- *stopj:* parar el movimiento de las articulaciones. Se debe definir la desaceleración.

- *get_actual_joint_posotions:* devuelve la posición angular actual de las articulaciones.

- *get_actual_tcp_pose:* devuelve la posición actual de la herramienta.

- *get_forward_kin:* calcula la transformación cinemática hacia delante que sirve para transformar la posición de las articulaciones al espacio de la herramienta. Como argumentos hay que poner el vector *q* con la posicion de las articulaciones y de manera opcional un offset para la posición de la herramienta.

- *get_inverse_kin:* calcula la transformación cinemática inversa que sirve para transformar la posición del espacio de la herramienta al espacio de las articulaciones. Como argumentos se debe añadir la posición de la herramienta y de manera opcional un error máximo para la posición y la orientación, lista con la posicion de las articulaciones y un offset para la posición de la herramienta.

- *textmsg:* permite enviar mensajes al Log. Hay que incluir el mensaje a enviar como argumento.

Uno de los requisitos más importante que debe cumplir el método de programación elegido para el UR5e es que debe permitir la comunicación mediante el protocolo TCP/IP, ya que el robot debe comunicarse por este método tanto con la placa de Arduino como con el PC para el mando de la Wii™. URScripts dispone de una serie de funciones que permiten que el robot se conecte como cliente a un servidor e intercambiar datos con este, estas son:

- *socket_open:* abre un socket de comunicación ethernet TCP/IP. Para ello hay que definir como argumentos la dirección del servidor (string), el número de puerto (int) y se puede poner un nombre al socket que se va a abrir para identificar la conexión. Esta función devuelve *false* si no ha logrado establecer la conexión y *true* si la conexión se ha establecido correctamente.

- *socket_read_ascii_float:* lee números de tipo *float* en formato ascii enviados desde el servidor. Como argumentos se define el número de datos que se espera recibir (máximo 30 cada vez) y deforma opcional el nombre de la conexión y el timeout (tiempo que espera a recibir la información hasta que termine la acción de leer, si se pone 0 o un numero negativo no termina hasta que no se hayan leído todos los numero).

- *socket_send_line:* envía un *string* en formato ascii terminado en *n* al servidor. Como argumentos hay que poner la frase que se desea enviar y de manera opcional el nombre de la conexión.

Cabe destacar que se han mencionado solamente algunas de las funciones que ofrece URScripts y de manera resumida. Si se desea mayor información el manual de scripts mencionado anteriormente dispone de más y mejor información.

Una vez vistos estos dos métodos que ofrece UR para programar sus robots se puede concluir que PolyScope es una interfaz muy sencilla que permite programar el UR5e fácilmente. En esta el programa construido se ejecuta en el robot tan solo apretando el botón de ejecutar, sin necesidad de configuraciones extra por parte del robot. Además, dispone de funciones típicas de los lenguajes de programación (bucles, condicionales, etc.), también permite la creación de hilos para ejecutar varios procesos de forma simultánea y una de las funciones que lo hacen todavía más potente es que permite añadir scripts, lo que hace que se combinen las ventajas de ambos métodos. Por su parte los scripts ofrecen muchas funciones específicas para trabajar con el robot, pero lo más importante es que nos ofrece funciones para que el robot se comunique por TCP/IP como cliente. En resumen, la combinación de PolyScope con los scripts ofrecen un método de programación muy sencillo de utilizar que dispone de las herramientas necesarias para la realización del presente trabajo. Por este motivo se decidió programar el UR5e utilizando este método.

Existen otros métodos de programación, ya que el cobot UR5e permite ejecutar programas que hay que encapsular en el robot en Python o C++. También se puede utilizar otros lenguajes que tengan el protocolo XML-RPC. Sin embargo, la utilización de este método no aporta ninguna ventaja significativa con respecto al método elegido.

2.4.2 Programación Placa Arduino MEGA 2560.

El entorno de programación que se va a utilizar es el ARDUINO 1.8.9, este IDE (Integrated Development Environment) de Arduino está escrito en Java y se puede ejecutar en Windows, Mac OS X y Linux.

El software de Arduino IDE tiene como base el entorno de Processing, lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java. Este entorno de programación dispone de un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Además, permite seleccionar la placa sobre la que se van a cargar los programas y elegir la versión de microcontrolador que esta incorpora. Una vez seleccionada la placa y la versión del microcontrolador este entono dispone de las herramientas para cargar el programa compilado en la memoria flash del microcontrolador de la placa.

Los programas de Arduino son un fichero de extensión *.ino*, el fichero principal debe estar en una carpeta con su mismo nombre.

Al abrir en entono nos aparece el editor de texto donde escribir el código, una consola donde se muestran los errores, una serie de botones y menús desplegables y el modelo de la placa Arduino con el puerto al que está conectada.

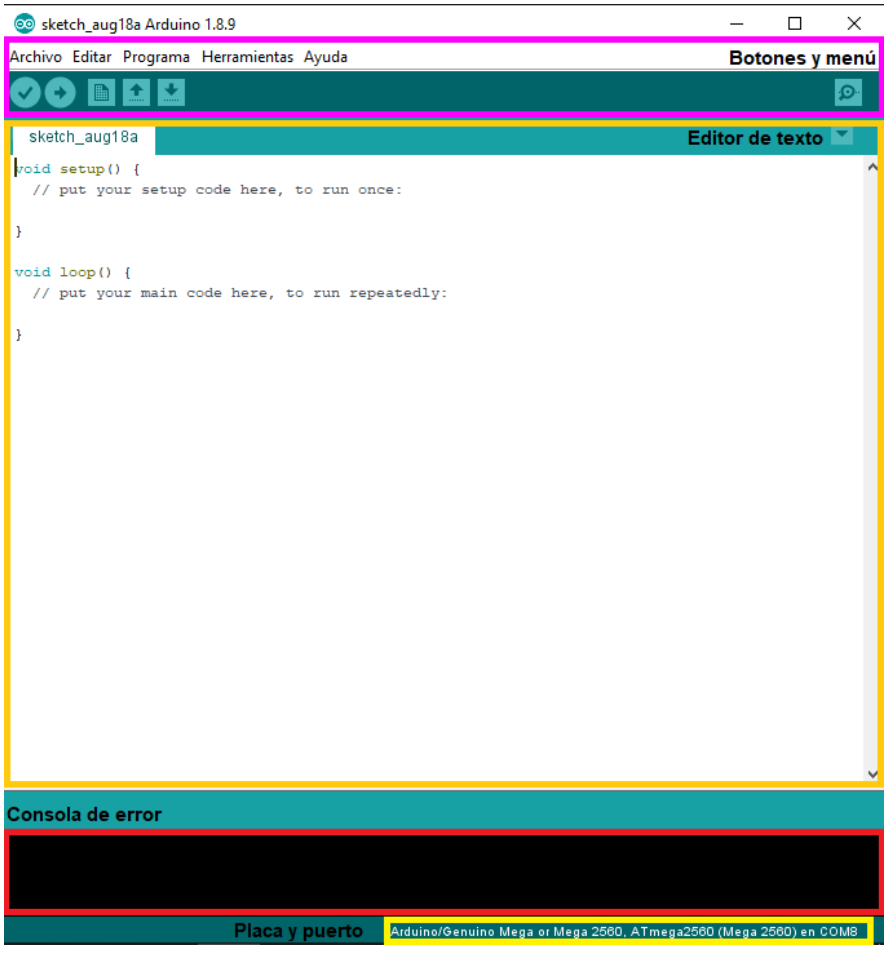


Figura 35. Entorno de programación Arduino IDE.

Los botones disponibles son los siguientes (de izquierda a derecha según aparecen en la Figura 28):

- Verificar: sirve para detectar errores en el código, es recomendable utilizarlo antes de subir el código a la placa.
- Subir: copia el programa en el microcontrolador de la placa.
- Nuevo: abre una nueva pestaña para realizar un nuevo programa.
- Abrir: permite abrir un programa guardado.
- Salvar: sirve para guardar el programa que se está realizando.
- Monitor serie: abre la consola del puerto serie que muestra los datos que recibe el Arduino por el puerto serie y permite enviar datos.

De las diferentes pestañas cabe destacar la de *Herramientas* que es donde se debe configurar la placa Arduino y la versión del microcontrolador que se va a utilizar y el puerto al que está conectada.

Cuando se abre un nuevo programa aparecen dos funciones *setup ()* y *loop ().* Todo el código que se escriba dentro de la primera solo se ejecutará la primera vez, esto se utiliza para realizar las configuraciones iniciales necesarias. A diferencia de este, todo el código que se escriba dentro de *loop ()* se ejecutara sin parar, por lo que aquí es donde debe estar el código que la placa debe ejecutar para realizar la tarea para la que ha sido programada.

Una vez se ha escrito el código se verifica para comprobar que no existen errores, una vez verificado se pulsa el botón *subir* que compila y carga el programa en la placa. Al cargar el programa se utiliza el gestor de arranque (o bootloader) que es un programa cargado en el microcontrolador que hace posible subir el código sin hardware adicional. Este se acciona durante un tiempo cuando se enciende o reinicia la placa y luego empieza a correr el programa que tenga cargado, para saber si el gestor de arranque se está ejecutando hay que fijarse en el led integrado que parpadea cuando este se ejecuta.

Además de todas estas características, que hacen de este entorno de programación una herramienta muy útil y sencilla de usar que permite construir y cargar programas en diferentes placas de Arduino de manera rápida, sencilla y segura, Arduino IDE incluye muchas bibliotecas muy interesantes para la realización de ciertos proyectos. Por ejemplo, en el presenta trabajo se va a utilizar el un módulo independiente para la conexión ethernet, Arduino dispone de la biblioteca *Ethernet* que incluye diferentes funciones que permiten configurar la placa como cliente o servidor, enviar o recibir diferentes tipos de mensajes, etc. Además, este módulo se comunica con la Placa por SPI y también se dispone de una librería de SPI que permite a la placa de Arduino comunicarse con dispositivos por este método. Estas librerías son muy sencillas de usar,

además en la web de Arduino se pueden encontrar muchos ejemplos de proyectos donde se utilizan estas librerías y documentación explicando cada una de las funciones.

La librería Ethernet dispone de 5 clases cada una de ellas con sus diferentes funciones:

- Clase ethernet: inicializa la biblioteca y la configuración de red. Dispone de algunas funciones como: *begin(), dnsServerIP(), localIP(),* etc.

- Clase IPAdress: se utiliza para direccionamiento IP. Dispone de la función *IPAdress().*

- Clase servidor: con esta se crean servidores que pueden enviar y recibir información de clientes conectados a ellos. Dispone de funciones como: *sever, EthernetServer(), begin(), accept(), write(), println(),* etc.

- Clase cliente: con esta se crean clientes que pueden conectarse a servidores e intercambiar información con ellos. Dispone de funciones como: *client, EthernetClient, connected(), print(),* etc.

- Clase ethernetUDP: permite enviar y recibir mensajes UDP. Dispone de funciones como: *begin(), read(), beginPacket(), endPacket, avaliable(),* etc.

El uso de esta biblioteca es muy importante en nuestro trabajo para poder establecer la comunicación TCP con el robot.

El lenguaje de programación de Arduino es un lenguaje propio que se basa en un lenguaje de alto nivel *Processing* que es parecido a C++. Se trata de una adaptación que deriva de avr-libc (herramientas necesarias para programar microcontroladores AVR de Atmel) que dispone de una librería de C de alta calidad para usarse con GCC (conjunto de compiladores estándar para los S.O que derivan de UNIX) en los microcontroladores AVR de Atmel.

Este lenguaje soporta las funciones estándar de C y algunas de C++. A continuación, se va a exponer de manera resumida las tres partes principales en las que se puede dividir el lenguaje de programación de Arduino:

<u>Funciones</u>

Se utilizan para controlar la placa y realizar cálculos. Algunas de las principales son :

- E/S digital que tiene funciones como: *pinMode(), digitalRead()* y *digitalWrite().*

- E/S analógica que tiene funciones como: *analogRead(), analogReference()* y *analogWrite().*
- Hora que tiene funciones como: *delay(), delayMicroseconds(),*etc.
- Matemáticas que tiene funciones como: *abs(), max(), min(), sqrt(),* etc.

Existen muchas más que se pueden consultar en la página web de Arduino.

Variables

El lenguaje de Arduino dispone de diferentes tipos de datos y algunas constantes.

Algunas de las constantes que se utilizan en este lenguaje son : HIGH, LOW, true, false, INPUT, OUTPUT, INPUT_PILLUP, etc.

En cuanto a los tipos de datos dispone de: String, array, bool, bolean, byte, char, int, float, double, long, short, size_t, unsigned char, unsigned int, unsigned long, void, Word.

También dispone de funciones para conversión de datos.

Estructuras de control

Dispone de las estructuras de control típicas como son:

- Condicionales: *if, if…else, switch…case.*
- Bucles: *for, while, do…while.*
- Bifurcaciones y saltos : *break, continue, return, goto.*

Este lenguaje es muy parecido a C por lo que mucha de la sintaxis que se utiliza es igual como los operadores aritméticos, de comparación, etc.

En resumen, tanto el entorno como el lenguaje de programación de Arduino son muy sencillos de utilizar y disponen de las herramientas necesarias para el desarrollo de nuestra aplicación.

2.4.3 Programación Mando Wii™.

Como se dijo anteriormente para controlar el Cobot UR5e mediante un mando de la Wii™ este se conecta por bluetooth al PC y es este quien se comunica con el UR5e por TCP. Para implementar esta aplicación se ha utilizado el lenguaje de programación Java.

Hoy en día java es uno de los lenguajes más utilizados, este nace en 1991 bajo el nombre de “OAK” que más tarde se cambió a “Green” y finalmente a Java.

Este surge con el objetivo de crear un lenguaje que se pareciera al C++ en estructura y sintaxis, que estuviera orientado a objetos, pero con la diferencia de que tuviera una máquina virtual propia.

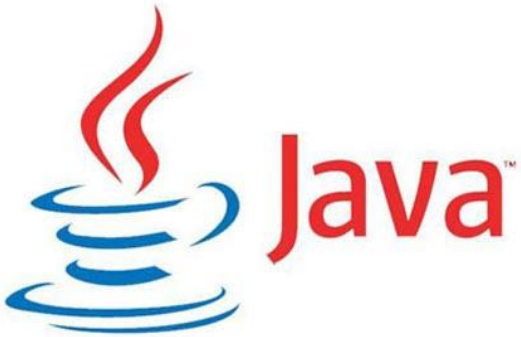


Figura 36. Logo de Java.

Java es un lenguaje orientado a objetos, parcialmente compilado/interpretado (compilador produce bytecode que es interpretado por una máquina virtual), tiene una sintaxis similar a C/C++, no tiene punteros y es 100% portable.

Como se ha dicho es un lenguaje orientado a objetos, que son instancias a una clase. Una clase es un tipo de dato abstracto que dispone de una serie de métodos definidos que permiten manipularlo.

El lenguaje de programación Java integra una serie de librerías estándar para: interfaz de usuario, objetos distribuidos y Threads.

La clase Threads permite crear diferentes hilos de ejecución lo que permite una programación concurrente con más de una tarea realizándose de manera simultánea. Esto se ha utilizado en nuestra aplicación para poder intercambiar información entre el UR5e y el PC de forma simultánea (se explica más detalladamente en el apartado 3).

Uno de los requisitos indispensables para el desarrollo de la aplicación es la comunicación entre el PC y el UR5e por TCP. Java dispone de la API Sockets que es un mecanismo que permite acceder a los servicios IPC que proporcionan los protocolos TCP y UDP. Un socket es como un “enchufe” al que otros procesos se pueden “enchufar un cable”, este representa una conexión. Esta API puede trabajar en modo datagrama que proporciona el acceso al transporte de datos del protocolo UDP. O también puede trabajar en modo stream que proporciona acceso al transporte de datos del protocolo TCP. Esta última es la que se va a utilizar para el desarrollo de la aplicación y dispone de diferentes clases con sus respectivos constructores y métodos:

- *ServerSocket:* Es el que acepta las conexiones de forma pasiva, se le llama socket de conexión o servidor. Alguno de los métodos de los que dispone son: *accept()* y *close().*

- *Socket:* Puede intercambiar datos con otros sockets, se le llama socket de datos, o cliente cuando es el proceso que inicia la comunicación. Alguno de los métodos de los que dispone son: *connect(), getInputStream()* y *getOutputStream()*

Existen diversas alternativas de lenguajes de programación para poder comunicar el PC con el UR5e mediante el protocolo TCP/IP. Uno de los motivos por lo que se decidió utilizar Java es que se encontró la librería *wiiusej* [9] que dispone de los constructores y los métodos necesarios para controlar el mando de la Wii™ desde Java. Esta librería dispone de diferentes clases que permiten acceder a diferentes funciones del mando (como la batería, la vibración, los leds, etc.), manejadores de eventos para los diferentes botones del mando, etc. A partir de los ejemplos de la librería se ha creado una clase con nuevos métodos más específicos para nuestra aplicación (todo esto se explicarán de forma detallada en el apartado 3).

Para la programación se ha utilizado el entorno de desarrollo eclipse 2018-12, este es una plataforma de desarrollo para diferentes lenguajes como Java y C/C++ entre otros. Este fue desarrollado inicialmente por IBM y en la actualidad por Fundation Eclipse. El uso más común de este software es con el lenguaje de programación Java, eclipse dispone de un asistente de código para Java que hace más sencilla la programación. Una de las ventajas de este entorno es que es gratis y de código abierto.

En eclipse es sencillo crear un nuevo proyecto Java y dentro de este crear los paquetes y las clases necesarias para dicho proyecto. Al iniciar el software hay una serie de menús desplegables y botones que sirven para crear un nuevo proyecto, cargar uno existente, ajustar las configuraciones de ejecución, etc. A la izquierda aparece un explorador de paquetes donde se muestra los diferentes proyectos con sus diferentes paquetes, clases y las librerías incluidas en el proyecto. Hay una zona de editor de texto que sirve para escribir el código. También dispone de una consola donde se muestran los errores.

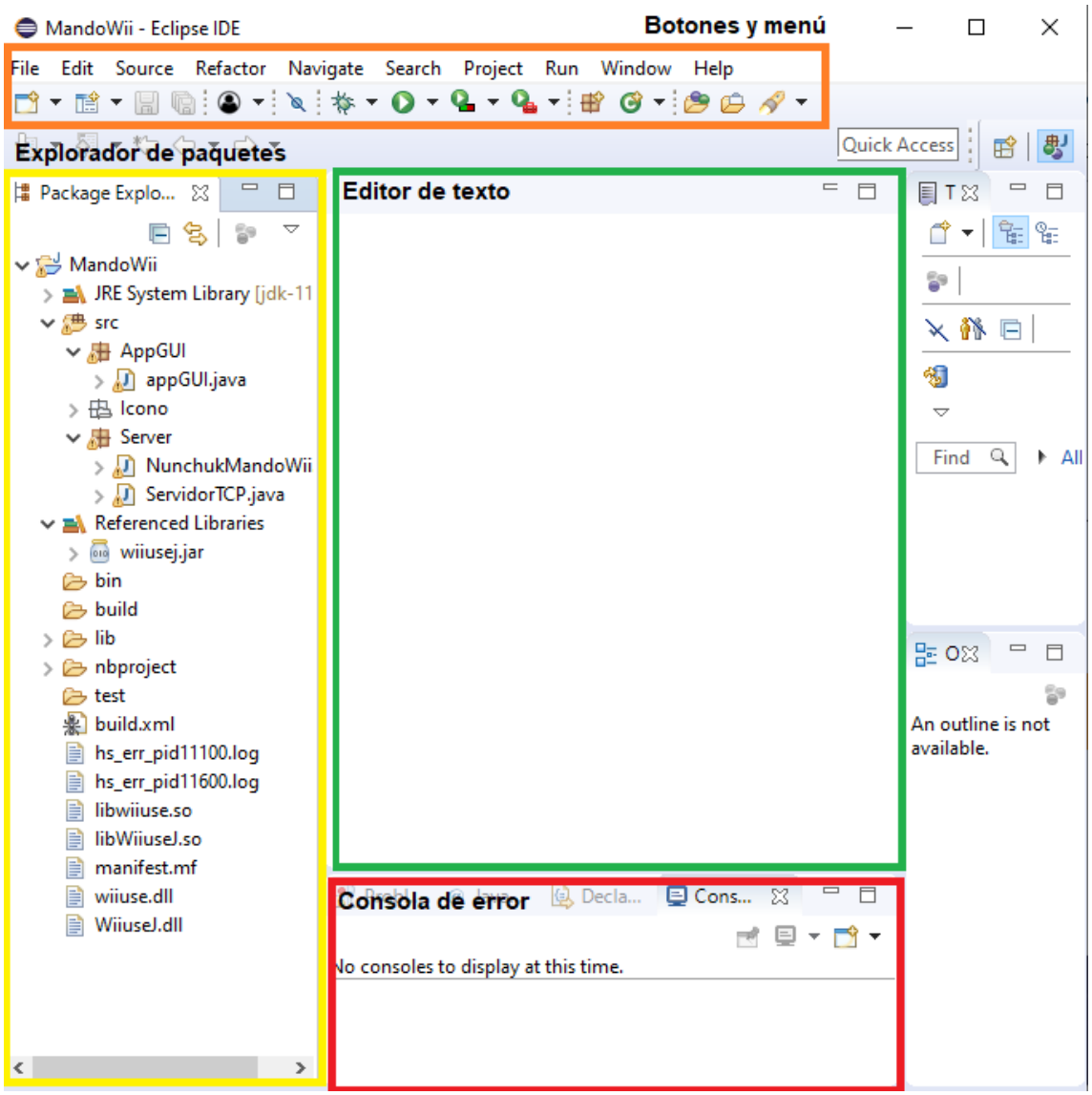


Figura 37. Eclipse IDE.

En definitiva, como se ha visto en este apartado tanto Eclipse IDE como el lenguaje de programación Java cumple con los requisitos necesarios para poder desarrollar el presente trabajo.

3. Diseño e implementación de las aplicaciones desarrolladas.

Tal y como se dijo en el apartado *"1.4 Objetivos"* el trabajo consiste en diseñar una aplicación que permita una programación del UR5e sencilla e intuitiva mediante un dispositivo externo. Esta aplicación debe permitir el guiado del robot y la grabación de trayectorias. En este apartado se mostrará las diferentes soluciones desarrolladas. En primer lugar, se analizarán algunas alternativas que se estudiaron para lograr el objetivo y que finalmente se descartaron. En segundo lugar, se presentará la solución desarrollada con la placa de Arduino. A continuación, se desarrollará la solución desarrollada con el mando de la Wii™. Por último, se compararán las dos soluciones desarrolladas y se expondrán sus ventajas e inconvenientes.

3.1 Análisis de alternativas.

En primer lugar, se comenzó analizando las diferentes opciones que ofrece el UR5e para poder desarrollar una aplicación que permitiera cumplir con los objetivos propuestos. Antes de llegar a las soluciones implementadas se probaron diferentes métodos que finalmente se descartaron por diversas causas. En este apartado se van a presentar dos de estos métodos y se explicaran los motivos de su descarte.

Primera solución descartada.

El UR5e puede trabajar de forma remota o de forma local, cuando el cobot está en modo remoto se disponen de unos puertos reservados para el control remoto a través de TCP/IP. Hay un puerto destinado para la conexión de un cliente primario (puerto 30001), otro para un cliente secundario (puerto 30002) y otro de tiempo real (puerto 30003). Es en el controlador donde están los diferentes sockets de servidor.

A través de estos puertos, cuando el cobot está en modo remoto, se puede transmitir información sobre el estado del cobot como posiciones, temperatura, etc. (en la página web de UR hay disponible un documento Excel con toda la información que se puede recibir desde el cobot). Además, se puede recibir comandos URScripts. La frecuencia de actualización es de 10 Hz, pero no siempre se puede garantizar esta frecuencia, en caso de que el tiempo real sea crítico el cobot dispone de un puerto específico para el intercambio de datos en tiempo real RTDE (puerto 30004) (guía disponible en la web de UR). Por otro lado, si el cobot está en modo local no se puede recibir comandos URScripts, pero si se desea obtener información del estado el cobot se deben usar los puertos 30004, 30011, 30012 y 30013.

e-Series							
	Primario		Secundario		Tiempo real		Intercambio de datos en tiempo real (RTDE)
Puerto no.	30001	30011	30002	30012	30003	30013	30004
Frecuencia [Hz]	10	10	10	10	500	500	500
Recibir	Comandos URScript	-	Comandos URScript	-	Comandos URScript	-	Varios datos
Transmitir	Ver archivo adjunto desde abajo		Ver archivo adjunto desde abajo		Ver archivo adjunto desde abajo		Ver la guía RTDE

Figura 38.Puertos disponibles para controlar y monitorear el UR5e de forma remota.

Por lo tanto, se pensó utilizar estos puertos para guiar y grabar las trayectorias del UR5e. La idea era enviar la información del estado del joystick por un puerto y otro puerto para guardas las posiciones del cobot.

Para realizar esta prueba es necesario poner el robot en modo remoto. Para ello se deben realizar los siguientes pasos desde la consola de programación:

1. *Installation → Fieldbus → Ethernet /IP (Enable).*
2. *Hamburger menu → Settings → System → Remote Control (Enable).*
3. Activar el control remoto en la nueva pestaña que aparece en la interfaz.

En el menú hamburguesa en la pestaña *about* se muestra la IP del robot.

Una vez activado el modo remoto se implementó en el PC un cliente en java que construía un comando URScripts a partir de la posición del joystick (para realizar estas pruebas se utilizó el mando de la Wii™) y lo enviaba al robot por TCP al puerto 30001. Cuando se realizaron las pruebas se consiguió mover el robot con el mando, pero este se movía a tirones. En un principio se pensó que la causa podría ser el tipo de comando enviado, para intentar solucionarlo se enviaron diferentes instrucciones como *movel, movej, speedl* obteniéndose el mismo resultado. El problema es que el robot ejecuta la instrucción y el programa termina, esto es lo que provoca que el robot se mueva a tirones. Cuando se utilizan los comandos *movel* o *movej* el robot se mueve hasta alcanzar la posición y se para, además en este método lo interesante es que el incremento de las coordenadas sea pequeño para poder alcanzar cualquier posición por lo que esto empeoraba el comportamiento. Por otra parte, cuando se utiliza el comando *speedl* el robot se mueve hasta alcanzar la velocidad definida para cada eje, el robot también funcionaba a tirones y además con movimientos más bruscos debido a los frenazos. Se pensó la posibilidad de aumentar la velocidad de forma que el robot no llegara a alcanzarla antes de que se cambiara su valor mediante movimientos de joystick, pero el robot tardaba mucho en reaccionar a los cambios de dirección y llevarlo a una posición concreta resultaba casi imposible, además en algunas de las pruebas realizadas el robot realizaba una parada de seguridad.

```
else if(input.equals(input8)) {
        output =  cl.enviar_y_esperar_respuesta("movel(p[" +x +"," +y +"," +z +"," +rx +"," +ry +"," +rz +"], a=1,v=0.25,t=0,r=0)");
}
else if(input.equals(input9)) {
        output =  cl.enviar_y_esperar_respuesta("movej(p[" +x +"," +y +"," +z +"," +rx +"," +ry +"," +rz +"], a=1,v=0.25,t=0,r=0)");

}
else if (input.equals(input3)) {
        output =  cl.enviar_y_esperar_respuesta("speedl([" +vx +",0.0,0.0,0.0,0.0,0.0],0.1)");
}
```

Figura 39.Fragmento del código de la prueba de cliente en Java.

Además de estas complicaciones, la recepción de la información del estado del robot no resultó ser tan sencilla, ya que el robot envía esta información cifrada. En la siguiente figura se muestra los mensajes recibidos en Java por parte del robot.

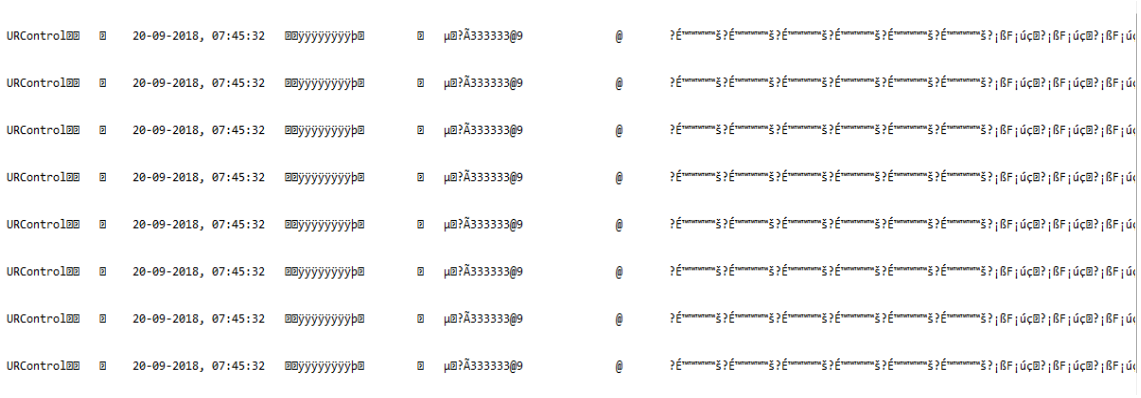


Figura 40.Mensaje recibido del estado del robot.

Existen diferentes opciones para descifrar esta información. En la propia web de Universal Robots proporcionan un código en Python para descifrarla. Otra opción que se pensó es implementar un cliente Modbus para leer los registros y guardar los valores de posición para grabar la trayectoria. Debido a los problemas que se encontraron en el guiado del robot esta parte no se desarrolló más y se decidió investigar otras posibilidades.

Segunda solución descartada.

Tras los problemas encontrados en la opción anterior se pensó que disponer de un programa dentro del robot podría resolver los problemas encontrados y facilitar el uso de los datos del robot.

La idea inicial era crear un programa en el UR5e que se ejecutara en el propio robot. En el PC se implementó un servidor en Java que construía un mensaje a partir de la posición del joystick y lo enviaba a un cliente conectado a él. El UR5e funcionaba como cliente y recibía las instrucciones a realizar desde el PC, para ello se iniciaba la comunicación con el servidor en el apartado *BeforeStart* del programa del robot. Con esto se pretendía que el robot recibiera el comando URScripts y lo ejecutará. De esta forma, como el programa del robot se ejecuta de manera infinita, solucionábamos el problema de que al realizar la instrucción

se acabara el programa que era el motivo por el cual el robot se movía a tirones. Además, se pretendía que el robot enviara directamente la información sobre su posición al servidor, lo que evitaría tener que descifrarla.

Sin embargo, al enviar los comandos URScripts al robot estos no se ejecutaban, se probó guardarlos en una variable, añadirlos en un script, también se probó a utilizar diferentes funciones. El UR5e recibía los mensajes correctamente, pero no los ejecutaba. Al guardarlos en una variable se vio que el mensaje se guardaba con las comillas, para comprobar si esto era el problema se definieron símbolos para indicar el comienzo y el final del mensaje (esto es una opción que te ofrece la instrucción del URScripts utilizada para la recepción de mensajes desde un servidor).

En la siguiente figura se muestra un fragmento de código de prueba donde se observan diferentes opciones que se probaron para poder ejecutar la instrucción recibida, todos ellos sin éxito.

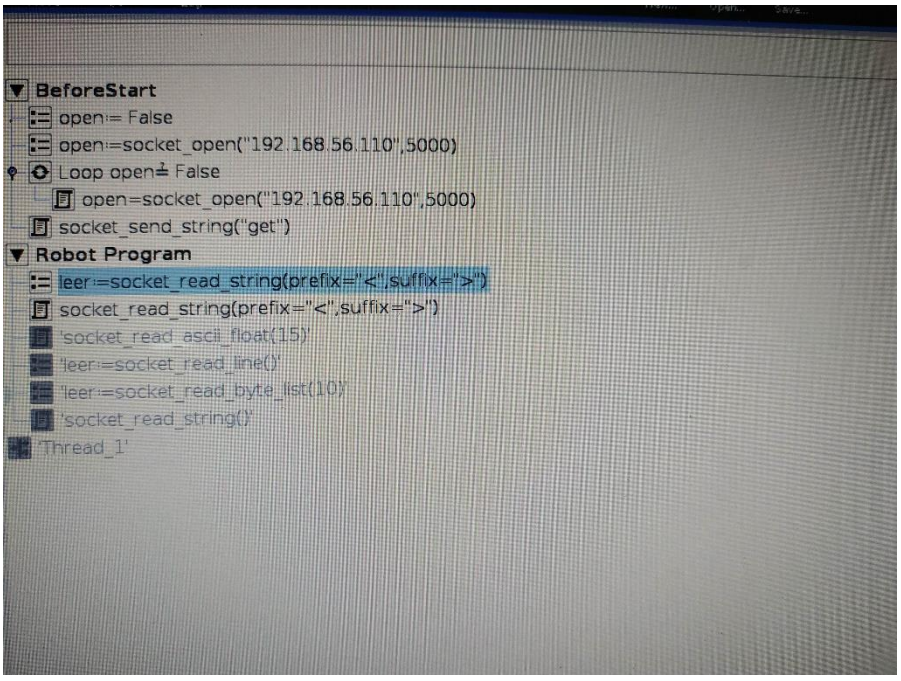


Figura 41. Recepción de una instrucción desde un servidor en Java.

Por lo tanto, esta opción tampoco funcionaba, ni si quiera se logró mover el robot, ya que no ejecutaba las instrucciones.

Los resultados negativos obtenidos en la implementación de estas opciones sirvieron para sacar algunas conclusiones que facilitaron la búsqueda de nuevas alternativas. De estas pruebas se sacó en claro que un programa ejecutándose en el robot facilita las cosas, ya que los movimientos de robot son más suaves debido a que no terminar el programa cada vez que se ejecuta la instrucción. Y, además, es más sencillo guardar y utilizar las diferentes posiciones del robot, sin necesidad de programas externos para descifrarla ni de leer los registros. Por esto se planteó la opción de un programa que se ejecutará en el robot y se

conectará a un servidor para recibir la información del dispositivo externo y a partir de esta moverse y poder grabar las trayectorias.

3.2 Mando Arduino.

En el presente apartado se va a presentar una de las aplicaciones diseñadas para lograr los objetivos, que son el guiado del UR5e y la grabación de trayectorias mediante un dispositivo externo. Para ello se va a construir un mando utilizando una placa de Arduino que dispone de diferentes botones para realizar las configuraciones necesarias como tipo de movimiento, grabar trayectorias, reproducir trayectorias, etc. También dispondrá de un joystick para guiar al robot.

3.2.1 Comunicación UR5e y placa Arduino.

Para poder controlar el UR5e mediante el mando construido en la placa de Arduino primero se debe establecer la comunicación entre ellos. En un principio, para establecer la comunicación se pensó en comunicar el Arduino con el PC mediante bluetooth y que este se comunicara por TCP/IP con el UR5e. Para ello se encontró la librería *PanamaHitek_Arduino 3.0.0* que permite recibir en un programa Java datos enviados por bluetooth desde Arduino. Sin embargo, esta opción requería la utilización del PC como dispositivo intermedio. Para evitar esto se decidió comunicar la placa de Arduino directamente con el UR5e mediante TCP/IP, para ello se necesita un dispositivo externo, ya que el Arduino MEGA por sí solo no permite conectarse a una red. Para esto se utilizó un módulo independiente con W5100.

TCP/IP es el nombre que toman un conjunto de protocolos para la comunicación de datos. El nombre se debe a sus dos protocolos más importantes TCP (Transmission Control Protocol) e IP (Internet Protocol). Hoy en día este protocolo es el más utilizado en la mayoría de las aplicaciones telemáticas. Algunas de las ventajas de este protocolo es que los estándares del protocolo son abiertos y son desarrollados de forma independiente del hardware, además funcionan sobre cualquier tipo de medio (en esta aplicación sobre una red ethernet).

En esta aplicación se va a utilizar a nivel de transporte el protocolo TCP, este está orientado a conexión, este protocolo permite la detección de errores y se asegura que los datos lleguen completos y en el orden correcto del emisor al receptor. Este protocolo es más lento que el protocolo UDP, pero es más fiable en la transmisión de datos. Como la velocidad de transmisión de datos no es un factor crítico es esta aplicación se va a utilizar el protocolo TCP.

En la comunicación entre el UR5e y la placa de Arduino cada uno tiene un papel diferente en la comunicación. La placa de Arduino es el servidor, esta ofrece unos servicios y espera a que se le soliciten. El UR5e actúa como cliente, es el

que inicia la comunicación y solicita los servicios y espera respuestas. Como se vio en el apartado "*2.4 Requerimientos de software"* la placa Arduino dispone de la librería *Ethernet* para poder implementar un servidor y el UR5e dispone de comandos URScripts para implementar un cliente.

3.2.2 Funcionamiento y algoritmos diseñados.

El funcionamiento de la aplicación diseñada es muy sencillo, una vez establecida la comunicación entre el mando y el UR5e desde el mando se envía un mensaje que se construye a partir del estado de los botones y de la posición del joystick. A partir de este mensaje el UR5e realiza unas acciones como moverse, grabar la trayectoria, reproducir una trayectoria grabada o activar la herramienta.

Para poder llevar a cabo esta función se debe implementar dos programas. El primero se ejecuta en la placa de Arduino en este se inicia un servidor que espera a la conexión de un cliente, momento en el que comienza a leer los valores de los 10 botones que lo forman y los valores del joystick. A partir de esta información se construye un vector que se envía al cliente.

El segundo programa se ejecuta en el UR5e, este es el encargado de iniciar la comunicación con el servidor. Una vez establecida la comunicación recibe el vector, guarda el valor de las diferentes posiciones y realiza las acciones asociadas a los valores recibidos. En el vector recibido el robot encuentra información sobre el modo de funcionamiento, sobre la grabación o reproducción de trayectorias, velocidades en cada eje y sobre el estado de la herramienta.

En la aplicación diseñada se puede guiar el robot mediante el joystick, los diferentes modos de funcionamiento nos permiten mover el robot en los planos XY, XZ e YZ y realizar giros en X, Y Z. Todos estos movimientos para guiar al robot se realizan ajustando la velocidad en cada eje de la herramienta con respecto al sistema de coordenadas de la base. También se pueden grabar 2 tipos de trayectoria:

- Una trayectoria de movimientos rectilíneos en la que la posición y la orientación de la herramienta sigue una evolución lineal. Este tipo de trayectorias puede ser de utilidad para tareas en que la precisión de la posición y la velocidad de la herramienta pueda resultar importante, por ejemplo, cuando el robot trabaja en un entorno delicado donde hay objetos cercanos.

- Una trayectoria de movimientos libres o punto a punto. En este tipo de trayectoria el movimiento del robot es más rápido. Este tipo de trayectoria es adecuado cuando no existe peligro de colisión con otros objetos mientras el robot realiza su tarea.

Una vez grabadas las trayectorias estas se pueden reproducir cuando se desee. Además, se ha añadido una función de seguridad que al detectar que la fuerza

en el eje Z de la herramienta es mayor de 50 N impide el movimiento del robot, si estaba en movimiento o realizando alguna trayectoria se detiene. Cuando el robot entra en este modo no se puede mover con el joystick ni realizar ninguna trayectoria grabada hasta que se reestablecen los valores al pulsar el botón *home.* Esto impide que en caso de colisión el robot siga avanzando. También se ha añadido un botón de parada de emergencia que detiene el robot si está en movimiento e impide realizar trayectorias, para salir de este modo y restablecer los valores hay que pulsar el botón *home*. Por último, se ha añadido la posibilidad de utilizar la herramienta del UR5e, para ello se ha añadido un botón para activarla. Esta se puede activar durante el guiado del robot y también durante la grabación de trayectorias, en este caso se grabará también el estado de la garra y, por tanto, durante la reproducción de la trayectoria también se reproducirán los movimientos de la herramienta.

A continuación, se explicará de forma detallada cada uno de los algoritmos diseñados.

Programa Arduino.

La función de este programa es enviar por TCP/IP al UR5e información del estado de los botones y del joystick. Para ello se debe incluir la librería *Ethernet.h,* además el módulo independiente con W5100 se conecta a la placa de Arduino mediante SPI por lo que es necesario incluir la librería *SPI.h.*

Una vez incluidas las librerías necesarias para la realización del proyecto se declaran e inicializan las variables a utiliza, en la Tabla 6 se muestran las variables utilizadas en este programa.

Nombre de la variable	Tipo de dato	Explicación
Vector [9]	float	Vector de 9 posiciones. Se construye a partir de la información del estado de los botones y joystick. Es el vector que se envía al UR5e.
i	Int	Variable auxiliar. Contador.
btn1	bool	Botón para seleccionar el modo.
btn2	bool	Botón para seleccionar el modo.
btn3	bool	Botón para seleccionar el modo.
btn4	bool	Botón para seleccionar el modo.
btnT1	bool	Botón para grabar la trayectoria 1.
btnT2	bool	Botón para grabar la trayectoria 2.
btnDT	bool	Botón para realizar la trayectoria grabada. Se debe combinar con btnT1 para realizar la trayectoria 1 y con el btnT2 para realizar la trayectoria 2.
btnHerr	bool	Botón para activar la herramienta.
btnHome	bool	Botón para reestablecer los valores iniciales. Sirve para salir de una situación de bloqueo del robot ya sea porque ha superado la fuerza

		permitida o porque se ha producido una parada de emergencia.
btnEmergencia	bool	Botón de parada de emergencia.
VRX	int	Valor analógico eje X del joystick.
VRY	int	Valor analógico eje Y del joystick.
joyX	float	Valor de velocidad cuyo valor depende de la entrada analógica del eje X del joystick.
joyY	float	Valor de velocidad cuyo valor depende de la entrada analógica del eje Y del joystick.
enviar	String	Para poder enviar el vector al UR5e se necesita enviarlo como cadena de bytes. Como paso intermedio se debe convertir a String.
buf [48]	byte	Buffer que guardar la cadena de bytes que se debe enviar al UR5e.
Mac []	byte	Dirección Mac del dispositivo. Parámetro para definir la conexión Ethernet.
ip	IPAddress	Dirección IP del dispositivo. Parámetro para definir la conexión Ethernet.
gateway	IPAddress	Dirección IP puerta de enlace de la red. Parámetro para definir la conexión Ethernet.
subnet	IPAddress	Máscara de subred. Parámetro para definir la conexión Ethernet.

Tabla 6. Variables utilizadas en el programa de la placa Arduino.

Una vez declaradas las variables se inicia un servidor indicando el puerto al que escucha, en este caso el puerto 5000. Para ello se declara de la siguiente manera:

EthernetServer server = EthernetServer(5000);

A continuación, en la función *setup()* del código Arduino se establece el modo de funcionamiento de los diferentes pines. Luego se inicializa la biblioteca *Ethernet* y se configura la red, indicando la mac, la ip, la puerta de enlace y la máscara de subred. Para esto se utiliza la siguiente función:

Ethernet.begin(mac,ip,Gateway,subnet);

El siguiente paso es empezar a escuchar clientes. Utilizando la función:

server.begin();

En el programa se han añadido algunas funciones que pueden servir de utilidad en caso de que se produzcan problemas en la comunicación. Se ha iniciado la comunicación serie y en el monitor serie del IDE de Arduino se muestra un mensaje cuando el servidor se ha conectado, te muestra su IP, su MAC y un mensaje cuando el servidor comienza a escuchar clientes. Para hacer esta comprobación el Arduino debe estar conectado al PC. Esto permite comprobar si la comunicación se ha establecido de forma correcta fácilmente, si se produce un problema puedes comprobar si la comunicación está bien solo conectando el Arduino al PC y abriendo el monitor serie.

Una vez realizadas todas estas configuraciones iniciales comienza el bucle *loop()* de Arduino. En este el primer paso es comprobar si se ha conectado un cliente, mediante la función:

EthernetClient client =server.available();

Mientras no se haya conectado ningún cliente la función anterior devolverá *false* y el programa continuará esperando a la conexión de un cliente. Cuando se establezca la comunicación la función devuelve *true* lo que permite comenzar el intercambio de mensajes entre el servidor (mando Arduino) y el cliente (UR5e).

Una vez conectado un cliente se lee el valor de las diferentes entradas de la placa Arduino y se asigna su valor a la variable correspondiente. En la Tabla 7 se muestra los diferentes pines que se van a utilizar, con su modo de funcionamiento y a los botones a los que están asociados cada uno de ellos.

Variable	Pin (PinMode)	Tipo de entrada	Dispositivo asociado
btn1	2 (INPUT)	Digital	Botón 1
btn2	3 (INPUT)	Digital	Botón 2
btn3	5 (INPUT)	Digital	Botón 3
btn4	6 (INPUT)	Digital	Botón 4
btnHerr	12 (INPUT_PULLUP)	Digital	Botón joystick
btnHome	7 (INPUT)	Digital	Botón Home
btnT1	9 (INPUT)	Digital	Botón T1
btnT2	11 (INPUT)	Digital	Botón T2
btnDT	13 (INPUT)	Digital	Botón DT
btnEmergencia	8 (INPUT)	Digital	Botón Emergencia
VRX	A0	Analógica	Joystick
VRY	A1	Analógica	Joystick

Tabla 7. Asignación del valor de las variables en función del botón pulsado.

Para asignar los valores leídos en las entradas a las variables se utilizan las siguientes funciones para las entradas digitales:

btn1 = digitalRead(2);

Y para las analógicas:

VRX = analogRead(A0);

Se puede observar que no se ha utilizado los pines digitales 0,1,4 y 10. El 0 y el 1 se han dejado libres porque para programar la placa se utilizan y habría que desconectar los circuitos conectados a estos. Por otro lado, el módulo de ethernet se comunica con la placa de Arduino mediante SPI, este utiliza el pin 10 como SS en la comunicación por lo que este está reservado para el módulo con W5100. El pin 4 se utiliza como SS cuando se selecciona la tarjeta SD por lo que el pin 10 y el 4 no pueden trabajar simultáneamente. En este proyecto no habrá SD por lo que se deja el pin 4 libre para evitar problemas de comunicación con el módulo ethernet. Además, hay que señalar que la entrada digital asociada al

botón del joystick (pin 12) se ha configurado como *INPUT_PULLUP* de esta forma se utiliza la resistencia pull-up interna de Arduino y no es necesario añadir resistencias adicionales.

Una vez leídos los valores de los botones y joystick se llama a la función *CalculoVector().* Esta función se ha diseñado para construir el vector a enviar en función del estado de los botones. En esta el primer paso es poner a 0 las nueve posiciones del vector para que no se mantengan valores anteriores, para ello se recorre el vector mediante un buble *for.*

Luego se utiliza la función *CalculoJoystick()* diseñada para actualizar los valores leídos de las entradas analógicas. En esta función se asigna el valor a las variables *joyX* y *joyY* en función de los valores del joystick leídos que se guardan en las variables *VRX* y *VRY.* El joystick utilizado no dispone de mucha precisión, por lo que las posibilidades que ofrece son reducidas. Se pensó que la velocidad de movimiento del UR5e fuera proporcional a la posicion del joystick, pero con el joystick utilizado no fue posible. Finalmente se implementó esta función que permite conocer en la posición que se encuentra el joystick. Para ello:

- *VRX* < 416 → *joyX = - 0.1*
- 416 ≤ *VRX* ≤ 616 → *joyX = 0*
- *VRX* > 616 → *joyX = 0.1*

Para el caso de *joyY* se realiza el mismo proceso, comparando en este caso el valor de *VRY.*

En este momento se comienza a rellenar las diferentes posiciones del vector. A continuación, se explicará la función de las diferentes posiciones.

Posición 0:

Esta sirve para modificar el estado de la herramienta. Depende únicamente del estado del botón del joystick que es el asignado para activar la herramienta. El botón del joystick está en modo *INPUT_PULLUP* por lo que hay que tener en cuenta que cuando el botón está pulsado *btnHerr = 0* y cuando no está pulsado *btnHerr=1* en cambio en la posición 0 del vector se escribe un 1 cuando el botón se ha pulsado y un 0 cuando no está pulsado. En la siguiente tabla se muestran los diferentes valores que puede tomar el vector en esta posición y la combinación de botones para cada uno de ellos.

Posición 0	Función	Combinación de botones		Valor Vector [0]	Significado
	Cambiar estado de la herramienta	Ninguno		0	No cambia estado herramienta
		btnHerr		1	Sí cambia estado herramienta

Tabla 8. Posición 0 del vector que se envía al UR5e desde Arduino.

Posición 1:

La posición 1 sirve para seleccionar el modo de funcionamiento. El valor de esta posición del vector depende del botón 1 (*btn1),* botón 2 (*btn2),* botón 3 (*btn3*), botón4 (*btn4*) , botón Home (*btnHome*) y botón de Emergencia (*btnEmergencia*). Se han configurado 9 modos diferentes. Los botones 1, 2 y 3 activan el movimiento del robot en diferentes planos. Cuando estos botones se combinan con el botón 4 indica giros sobre diferentes ejes. Por su parte el botón Home sirve para indicar al robot que debe restablecer ciertos valores. Y por último el botón de Emergencia que indica al robot que debe realizar una parada de emergencia. En la siguiente tabla se explica los diferentes modos de funcionamiento y la combinación de botones para lograrlo:

Posición 1	Función	Combinación de botones	Valor vector[1]	Significado
	Seleccionar el modo de funcionamiento	Ningún botón pulsado	0	El robot no se mueve
		btn1 (resto de botones sin pulsar)	1	Movimientos en el plano XY
		btn2 (resto de botones sin pulsar)	2	Movimientos en el plano XZ
		btn3 (resto de botones sin pulsar)	3	Movimientos en el plano YZ
		btn1 + btn4 (resto de botones sin pulsar)	4	Giros en el eje X
		btn2 + btn4 (resto de botones sin pulsar)	5	Giros en el eje Y
		btn3 + btn4 (resto de botones sin pulsar)	6	Giros en el eje Z
		btnHome (resto de botones sin pulsar)	7	Se ha pulsado el botón Home
		btnEmergencia (sin importar el estado del resto de botones)	8	Se ha pulsado el botón de parada de emergencia.

Tabla 9. Posición 1 del vector que se envía al UR5e desde Arduino.

Nota: Cuando se dice “resto de botones sin pulsar” se refiere a los botones de los que depende este modo, el estado del resto de botones no se contempla y no afecta.

Posición 2:

La posición 2 sirve para indicarle al robot que debe comenzar a grabar una trayectoria y cuando debe terminar la grabación, también le indica si debe realizar alguna de las trayectorias grabadas. El valor de esta posición del vector depende del botón T1 (*btnT1),* botón T2 (*btnT2)* y botón DT (*btnDT).* En la siguiente tabla se muestran los diferentes valores que puede tomar esta posicion del vector, su significado y la combinación de botones para lograrlo.

Posición 2	Función	Combinación de botones	Valor vector[2]	Significado
	Indicarle al robot que comience o termine una grabación de una trayectoria y que realice alguna de las trayectorias grabadas	Ninguno	0	Ninguna acción
		bntT1 (resto de botones sin pulsar)	1	Iniciar o terminar la grabación de la trayectoria 1 (cuál de las dos acciones se debe realizar se gestiona en el UR5e)
		bntT2 (resto de botones sin pulsar)	2	Iniciar o terminar la grabación de la trayectoria 2 (cuál de las dos acciones se debe realizar se gestiona en el UR5e)
		bntT1 + btnDT	3	Realizar la trayectoria 1
		bntT2 + btnDT	4	Realizar la trayectoria 2

Tabla 10. Posición 2 del vector que se envía al UR5e desde Arduino.

Nota: Cuando se dice “resto de botones sin pulsar” se refiere a los botones de los que depende este modo, el estado del resto de botones no se contempla y no afecta.

Posición 3,4,5,6,7 y 8:

Se ha visto que existen diferentes modos de funcionamiento que se indican en la posicion 1 del vector. Cuando el modo es el 7 o el 8, que se corresponde a que los botones de Home o Emergencia se han pulsado, no hay que gestionar nada más, ya que es en el propio UR5e donde se llevarán a cabo las acciones correspondientes.

Sin embargo, el resto de los modos (0,1,2,3,4,5 y 6) que indica el plano sobre el que se va a mover la herramienta del robot o el eje de giro de esta, se gestiona en este código. Para ello se rellenan estas posiciones del vector con la velocidad de cada eje de la herramienta con respecto al sistema de coordenadas de la base, en función de los valores del joystick. Para llevar a cabo esta función se utiliza la estructura *switch* que comprueba el valor de la posición 1 del vector para conocer el modo en el que se encuentra y así poder rellenar las posiciones

del vector correctas. En la siguiente tabla se presenta la función de estas posiciones.

Posición	Función
Posición 3	Velocidad desplazamiento en el eje X
Posición 4	Velocidad desplazamiento en el eje Y
Posición 5	Velocidad desplazamiento en el eje Z
Posición 6	Velocidad giro en el eje X
Posición 7	Velocidad giro en el eje Y
Posición 8	Velocidad giro en el eje Z

Tabla 11. Posición 3,4,5,6,7 y 8 del vector que se envía al UR5e desde Arduino.

Las diferentes posiciones se rellenan con el valor de *joyX* y *joyY* según el modo seleccionado. A continuación, se muestra cómo se rellena el vector en función del modo seleccionado:

- *Case 0:* el robot no se va a mover por lo que todas las posiciones son 0.
- *Case 1*: el robot se mueve en el plano XY. Por tanto, *vector[3] = joyX* y *vector[4] = joyY* .
- *Case 2:* el robot se mueve en el plano XZ. Por tanto, *vector[3] = joyX* y *vector[5] = joyY.*
- *Case 3:* el robot se mueve en el plano YZ. Por tanto, *vector[5] = joyX* y *vector[4] = joyY*
- *Case 4:* el robot gira en el eje X. Por tanto, *vector[6] = joyX.*
- *Case 5:* el robot gira en el eje Y. Por tanto, *vector[7] = joyX.*
- *Case 6:* el robot gira en el eje Z. Por tanto, *vector[8] = joyX.*

De esta forma al UR5e le llegan directamente el valor de la velocidad de cada eje, en función de los valores del joystick, en el que se debe mover según el modo seleccionado.

Una vez se ha construido el vector se debe convertir a bytes, ya que este es el formato de mensaje que se va a enviar al UR5e. Para ello se debe realizar un paso intermedio que es convertir el vector a un String (variable *enviar)*, este paso intermedio es la única opción que se encontró para pasar el vector a bytes. Una vez se tiene el vector en la variable *enviar* como un Sting se convierte esta a una cadena de bytes y se almacena en la variable *buf,* esto se consigue mediante la siguiente función:

enviar.getbytes(buf,sizeof(buf));

Ahora ya se dispone del mensaje a enviar y en el formato deseado, por tanto, el servidor envía el mensaje al UR5e que con la información recibida llevará a cabo las acciones pertinentes. Para enviar un mensaje en bytes se utiliza la siguiente función:

server.write(buf,sizeof(buf));

Todo este proceso que se realiza cuando se ha conectado un cliente se repite cada 100 ms mientras haya un cliente conectado. Por tanto, cada 100 ms se lee el valor de los botones y del joystick se construye el vector y se envía al UR5e.

En este apartado se han presentado los diferentes botones y como se construye el vector en función de su estado con el objetivo de explicar el funcionamiento del código diseñado. Sin embargo, en el apartado *Anexo I. Manual de usuario* se explicará detalladamente como utilizar el mando diseñado.

En la siguiente figura se muestra el diagrama de flujo del funcionamiento del programa que se ejecuta en la placa de Arduino.

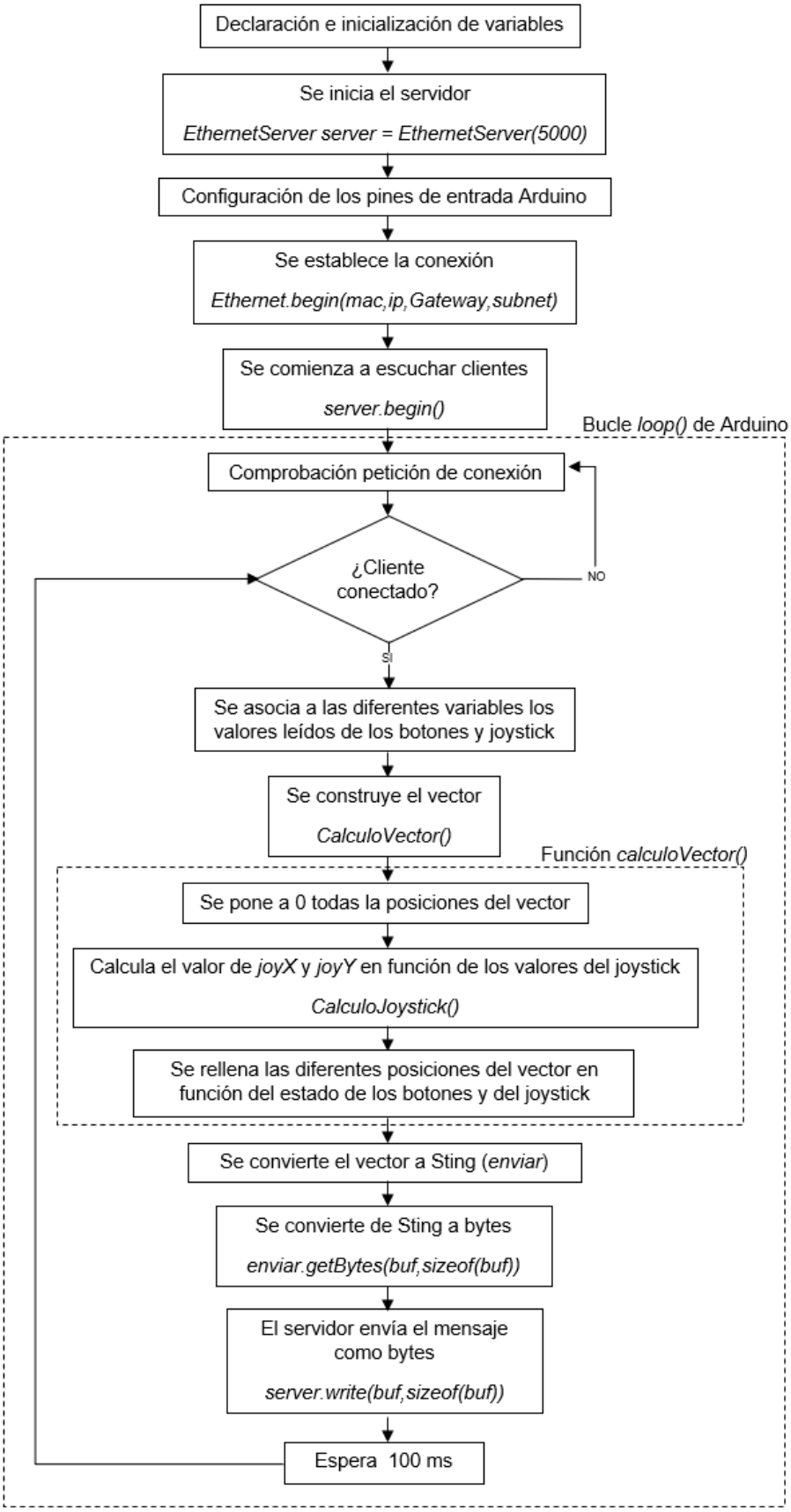


Figura 42. Diagrama de flujo programa Arduino.

En el programa de Arduino se pensó que se pudiera recibir información desde el robot y de esta manera realizar alguna acción como activar un zumbador al recibir un mensaje de que el robot está en parada de emergencia o se ha superado la fuerza máxima permitida. También unos leds que indicaran que el robot está grabando o realizando alguna trayectoria. Sin embargo, la programación secuencial que permite la placa de Arduino y el hecho de que las funciones para recibir mensajes desde un cliente son bloqueantes, dificulta la implementación de estas funciones, el programa se queda bloqueado hasta que recibe un mensaje del UR5e. Por tanto, el programa se quedaría esperando a recibir el mensaje y no enviaría el vector. Se podría gestionar que el robot enviara mensajes al Arduino todo el rato y en función del mensaje recibido se realizaran unas acciones u otras, sin embargo, esto una mayor carga computacional en una función que no aporta grandes ventajas. Como se verá para poder ver información sobre el estado del robot se ha gestionado que se pueda ver información sobre las actividades que está llevando a cabo el robot en la consola de programación.

Programa UR5e.

La función de este programa es recibir por TCP/IP desde el mando Arduino la información enviada por este. El UR5e se conecta como cliente por lo que es el encargado de iniciar la comunicación. Una vez establecida, el robot comenzará a recibir el vector construido en arduino. A partir de esta información el robot debe gestionar su movimiento, el estado de la garra, la grabación de trayectorias, la reproducción de trayectorias grabadas, gestionar las acciones asociadas a una parada de emergencia y a que se haya pulsado el botón home y realizar una parada de seguridad si se supera la fuerza máxima permitida.

Para comenzar se ha añadido una secuencia de inicio al programa, la programación en PolyScope permite añadir un trozo de código que se ejecuta antes de comenzar a ejecutar el programa principal, este se llama *BeforeStart.* En este se declaran e inicializan las variables y se inicia la comunicación con el mando Arduino. En la siguiente tabla se muestran las variables utilizadas en este programa.

Variable	Tipo de variable	Explicación
vec_tool1 [ ]	bool	Vector de 100 posiciones inicializado a *False.* Este se utiliza para guardar el estado de la herramienta durante la grabación de la trayectoria 1. Durante la realización de la trayectoria 1 se va recorriendo el vector para reproducir el estado de la herramienta. Es decir, este vector sirve para grabar el estado de la garra y su posterior reproducción.
vec_tool2 [ ]	bool	Misma función que *vec_tool1*, pero para la trayectoria 2.

tool	int	Valor del estado de la herramienta leído de la posición 0 del vector recibido.
toolbool	bool	Variable auxiliar para controlar la herramienta que depende del valor de la variable *tool.* Representa el estado de la garra, cuando es *True* la garra se abre y cuando es *False* la garra se cierra
nposiciones1	int	Indica la longitud del vector *trayec1.* Se utiliza para gestionar la grabación de trayectorias, que no se grabe más si el vector está lleno.
nposiciones2	int	Misma función que *nposiciones1* ,pero para *trayec2.*
grabar	int	Valor recibido desde el mando de Arduino (posición 2 del vector). A partir de este valor el robot gestiona el inicio o el fin de la grabación de la trayectoria 1 o 2 y la reproducción de alguna de las trayectorias grabadas.
grabarbool1	bool	Variable que depende del valor de *grabar.* Se utiliza para gestionar el comienzo y el final de la grabación de la trayectoria 1. Cuando su valor es *True* se inicia la grabación y cuando es *False* se finaliza.
grabarbool2	bool	Misma función que grabarbool1, pero para gestionar la trayectoria 2.
hacertray1	bool	Variable que depende del valor de *grabar*. Cuando el valor recibido indique que se debe reproducir la trayectoria 1 esta variable se pondrá a *True.*
hacertray2	bool	Mismo funcionamiento que *hacertray1*, pero para realizar la trayectoria 2.
fuerzaz	int	Guarda el valor de la fuerza ejercida en el TCP de la herramienta.
fuerzaparar	bool	Variable para gestionar la parada de seguridad cuando se ha superado la fuerza máxima permitida. Cuando esto ocurre se pone a *True.*
P_Emergencia	bool	Esta variable se activa cuando se ha pulsado el botón de parada de Emergencia en el mando. Cuando está *True* impide el movimiento del robot y la grabación y realización de trayectorias.
Pr [ ]	pose	Vector que describe la posición y orientación en el espacio cartesiano. En esta se ha guardado una posición de reposo del robot.
trayec1 [ ]	pose	Es un vector de pose donde se almacenan las diferentes posiciones del robot durante la grabación de la trayectoria 1. Durante la reproducción de la trayectoria 1 se recorre este vector para conocer las diferentes posiciones a las que se debe mover el robot. Este es un vector de 100 posiciones que se ha inicializado

		con la posición *pr* (medida extra de seguridad que se explicará más adelante).
trayec2 [ ]	pose	Misma función que *trayec1*, pero para gestionar la trayectoria 2.
datos	float	Variable donde se guarda el mensaje recibido
x	float	Valor recibido, posición 3 del vector. Velocidad desplazamiento eje X.
y	float	Valor recibido, posición 4 del vector. Velocidad desplazamiento eje Y.
z	float	Valor recibido, posición 5 del vector. Velocidad desplazamiento eje Z.
rx	float	Valor recibido, posición 6 del vector. Velocidad de giro eje X.
ry	float	Valor recibido, posición 7 del vector. Velocidad de giro eje Y.
rz	float	Valor recibido, posición 8 del vector. Velocidad de giro eje Z.
modo	int	Valor recibido, posición 1. Indica el modo de funcionamiento. En el robot se gestiona únicamente el modo 7 (botón home pulsado) y el modo 8 (botón parada de emergencia pulsado).
i	int	Indica el número de posiciones que se han grabado en *trayec1.* De esta forma se conoce hasta que posición se debe leer cuando se reproduce la trayectoria 1.
k	int	Es un contador que indica las posiciones del vector *tray1* que ya se han leído durante la realización de la trayectoria 1.
w	int	Misma función que *i,* pero para la trayectoria 2.
j	int	Misma función que *k,* pero para la trayectoria 2.
open	bool	Se utiliza para comprobar si la conexión con el mando se ha establecido. Su valor inicial es *False* y cuando inicia la comunicación se vuelve *True.*
mover	pose	Variable auxiliar que permite guardar el valor de una posición del vector de la trayectoria grabada en cada instante. Esta variable se le pasa como punto destino a la función *moveL.*
moverq	list	Variable auxiliar que permite guardar el valor de la posición de las articulaciones en cada instante a partir de los valores grabados en el vector de trayectoria. Esta variable se le pasa como argumento a la función *servoj.*

Tabla 12. Variables utilizadas en el programa del UR5e.

Como se dijo la programación en PolyScope permite ejecutar varios procesos de forma simultánea. Por esto se ha dividido el programa en diferentes partes que se ejecutan al mismo tiempo, esto permite gestionar el funcionamiento del robot de forma muy sencilla. Además del programa principal se han creado 6 hilos de

ejecución, cada uno de ellos encargado de una tarea específica. A continuación, se explicarán de forma detallada todas las partes del programa y su función. Las diferentes partes del programa son: *BeforeStart, Robot Program, thread1, thread2, thread3, thread4, thread5* y *thread6.*

BeforeStart

Esta parte del código se ejecuta antes de comenzar el programa principal y una única vez. En este se definen todas las variables que se van a utilizar, que se han presentado en la Tabla 12. A continuación se establece la comunicación con el mando, para ello se utiliza el comando URScripts que permite abrir un socket para la comunicación TCP/IP, esta función devuelve *False* cuando no se ha establecido la comunicación y *True* cuando se ha establecido correctamente. Para saber el estado de la comunicación se ha igualado esta función a la variable *open.* La instrucción es la siguiente:

open := socket_open("192.168.56.120",5000)

Para asegurar que la conexión se ha establecido antes de comenzar la ejecución del programa principal, se introduce la instrucción anterior en un bucle que se ejecutara hasta que la variable *open = True,* lo que significa que la conexión se ha establecido correctamente. En ese momento se envía el mensaje *"get"* al servidor para iniciar la comunicación, mediante la siguiente instrucción:

socket_send_string("get")

Y se muestra el mensaje *"Conexión Establecida"* en la pestaña Log de la consola de programación para informar al usuario del estado del robot. Para enviar un mensaje a la pestaña Log se utiliza el siguiente comando:

textmsg("Conexión Establecida")

En este momento el robot comenzará a recibir información del mando y empezarán a ejecutarse el programa principal y los diferentes hilos.

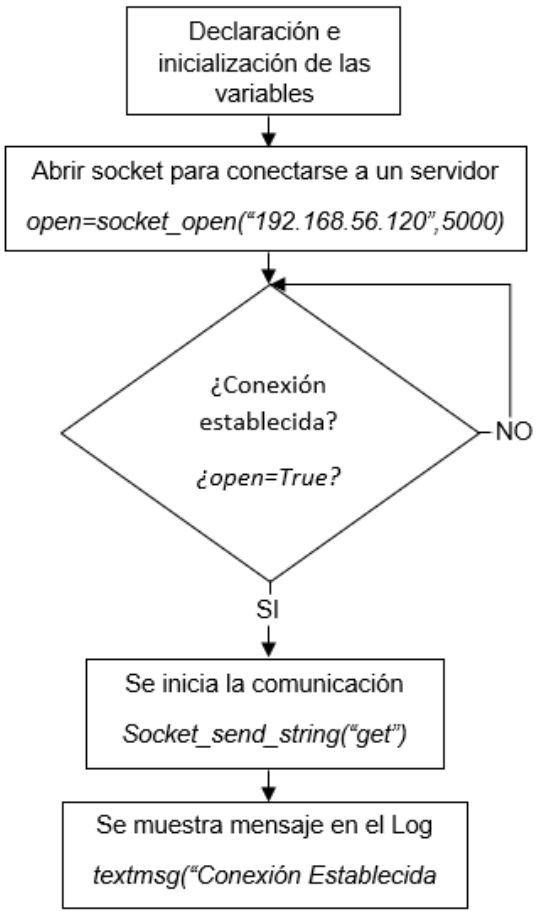


Figura 43. Diagrama de flujo de la parte BeforeStart del programa del UR5e.

Thread1

Se va a comenzar explicando este hilo porque es el encargado de recibir el vector desde el mando y guardar la información recibida en las variables correspondientes. Este hilo se encarga únicamente de recibir el mensaje del servidor, para ello se utiliza un comando URScirpts específico sabiendo que se va a recibir un vector de floats e indicándole el número de datos que se espera recibir, en este caso 9. Esta función devuelve el vector recibido y se guarda en la variable *datos.*

datos := socket_read_ascii_float(9)

Con el vector en la variable *datos* se guarda el valor de cada posición del vector en la variable correspondiente, a la hora de realizar las asignaciones hay que tener en cuenta que los vectores en PolyScope empiezan en la posición 1 y no en 0 como en Arduino. Sabiendo esto se hacen las siguientes igualaciones:

- *tool := datos [1],* estado de la herramienta.
- *modo := datos[2],* modo de funcionamiento.

- *grabar := datos[3],* gestión de la grabación y reproducción de trayectorias.
- *x := datos[4],* velocidad desplazamiento eje X.
- *y := datos[5],* velocidad desplazamiento eje Y.
- *z := datos[6],* velocidad desplazamiento eje Z.
- *rx := datos[7],* velocidad de giro eje X.
- *ry := datos[8],* velocidad de giro eje Y.
- *rz := datos[9],* velocidad de giro eje Z.

a los valores (x,y,z) no se les hace ninguna modificación, pero a los valores (rx,ry,rz) se multiplican por 1.5 para aumentar la velocidad de giro (que ahora será de ±0.15 en lugar de ±0.1). Si se quisiera modificar alguna de las 6 velocidades solo habría que multiplicar estos valores.

Con este hilo se tiene un proceso dedicado exclusivamente a recibir la información del servidor y guardar dicha información en la variable correspondiente para que esta pueda usarse por el robot. Para poder ejecutar varios hilos estos deben terminar con el comando *sync().*

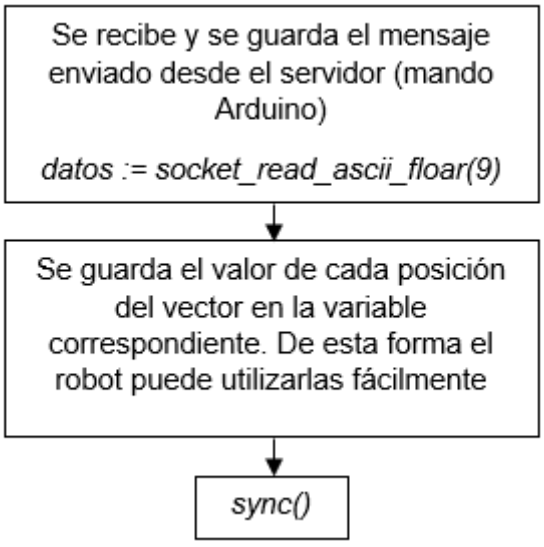


Figura 44. Diagrama de flujo de la parte Thread1 del programa del UR5e.

Robot Program (o programa principal)

Este es el programa principal, en este están las instrucciones que permiten mover el robot en función de los valores recibidos del mando, parar el movimiento del robot y realizar las trayectorias. Para saber que acción debe llevar a cabo se analizan una serie de variables que indican el trabajo a realizar, el estado de estas variables se gestiona en otros hilos.

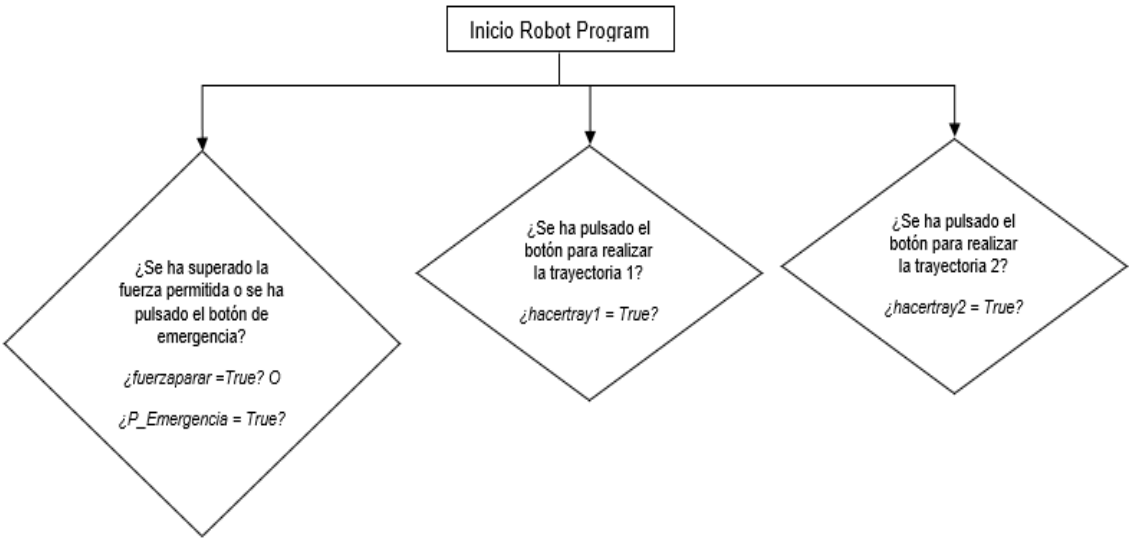


Figura 45. Diagrama de flujo de la parte Robot Program del programa del UR5e (parte1).

Si la variable *fuerzaparar = False* y *P_Emergencia = False*, lo que significa que no se ha superado la fuerza máxima permitida y que no se ha pulsado el botón de parada de emergencia, el robot utilizara los valores de velocidad de cada eje recibidos para moverse. Para ello se utiliza el comando *speedl* al que se le debe pasar un vector con la velocidad de la herramienta en m/s (vector espacial) y la aceleración de la herramienta en m/s^2. La instrucción utilizada es la siguiente:

speedl([x,y,z,rx,ry,rz],0.5)

En cambio, si *fuerzaparar = True* o *P_Emergencia = True*, lo que significa que se ha superado la fuerza máxima permitida o que se ha pulsado el botón de emergencia, el robot se parará. Para ello se utiliza el comando *stopl* que frena el robot (en el espacio de la herramienta) y hay que indicarle el valor de desaceleración en m/s^2. La instrucción es la siguiente:

stopl(50)

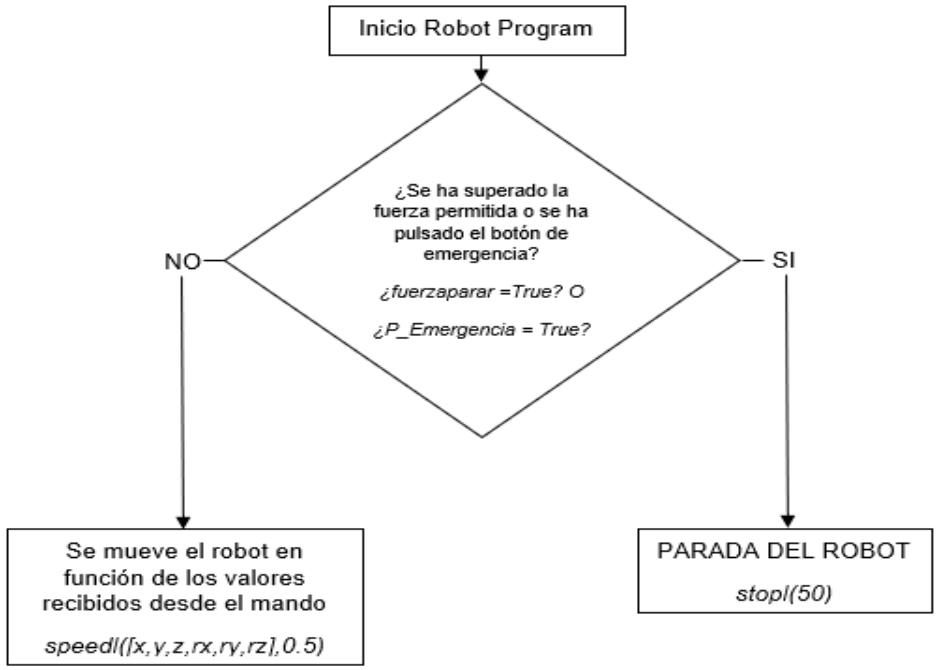


Figura 46.Diagrama de flujo de la parte Robot Program del programa del UR5e (parte2).

En esta parte del código también se gestiona la reproducción de las trayectorias grabadas. Cuando la variable *hacertray1 = True* se comienza a realizar la trayectoria 1, esta trayectoria es de movimiento libre. En este momento se muestra en la pestaña del Log el mensaje *"Realizando Trayectoria 1…"*. A continuación, comienza un bucle que terminará cuando se haya terminado la trayectoria grabada (se haya recorrido el vector *trayec1* hasta la última posición grabada, indicada por la variable *i)*, si se produce una parada de emergencia (*P_Emergencia = True)* o si se ha superado la fuerza máxima admitida (*fuerzaparar = True)*. Dentro del bucle el primer paso es guardar en la variable *mover* la posición que se debe alcanzar en ese instante que se corresponde al valor de *trayec1* en dicho instante (se utiliza la variable *k* para recorrer el vector). Esta trayectoria es de movimiento libre por lo que se aplica cinemática inversa para transformar la posición de la herramienta al espacio de articulaciones que se almacena en la variable *moverq*.

mover := trayec1[k]

moverq := get_inverse_kin(mover)

Para mover el robot a partir de los ángulos de las articulaciones se utiliza la instrucción *servoj*, con esta los movimientos del robot son más bruscos que si se utiliza *movel*. Esto puede suponer un problema si el robot se encuentra muy alejado de la posición de inicio de la trayectoria, ya que se moverá bruscamente hasta la posición inicial desde donde comenzará a ejecutarla, lo que puede resultar peligroso. Para solucionar esto cuando se inicia la reproducción de la trayectoria 1 (*k = 0)* el robot se mueve hasta la posición de inicio mediante un *movel.* Una vez alcanzada la posición inicial se utilizará para reproducir las

diferentes posiciones grabadas el comando *servoj* al que hay se le han añadido como parámetros la posición de las articulaciones, el tiempo de búsqueda anticipada y la ganancia (sirve para agudizar o suavizar la trayectoria).

servoj(moverq,0,0,0.5,0.5,1)

y se modificará el estado de la garra *toolbool* en función de los diferentes estados de la garra grabados en la variable *vec_tool1.*

toolbool := vec_tool1[k]

una vez realizadas estas acciones se aumentará el valor de *k* para ir recorriendo los diferentes vectores (posición y estado de la herramienta) y así reproducir la trayectoria grabada.

Una vez ha finalizado la realización de la trayectoria (se han recorrido los vectores hasta la última posición grabada, *k=i)* se muestra *"Finalizada Trayectoria 1"* en la pestaña Log y se pone *k = 0,* para poder volver a reproducirla, y *hacertray1 = False* para indicar que ha terminado. La realización de la trayectoria también puede terminar si se pulsa el botón de parada de emergencia o se supera la fuerza máxima permitida, ya que esto provocara que el buble termine.

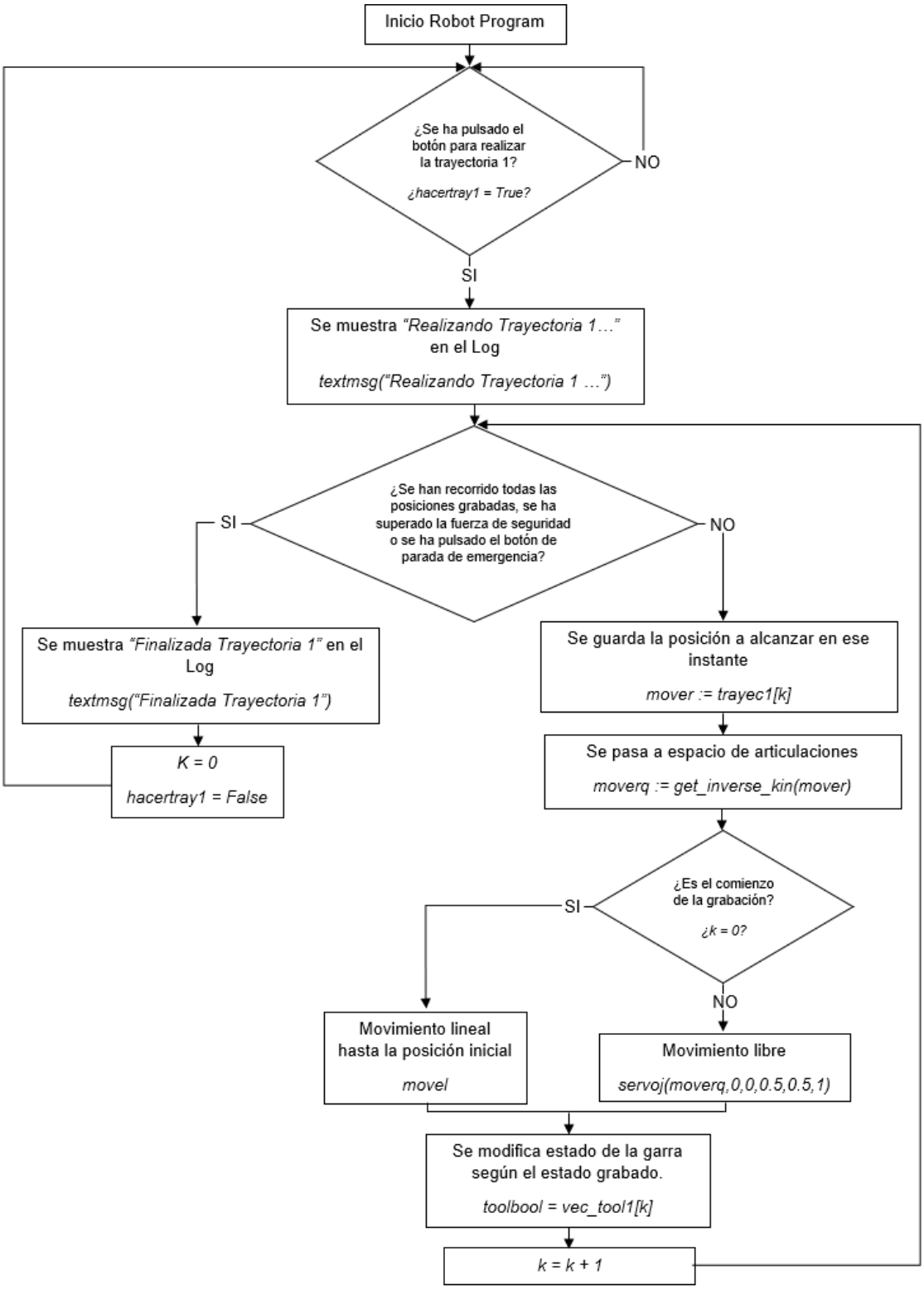


Figura 47Diagrama de flujo de la parte Robot Program del programa del UR5e (parte3).

Para terminar también se gestiona la realización de la trayectoria 2. El proceso es muy similar al seguido para la realización de la trayectoria 1, teniendo en cuenta que esta trayectoria es de movimientos rectilíneos, por lo que para mover el robot solo se utilizará la instrucción *movel.* Para comenzar a realizar la trayectoria 2 la variable *hacertray2 = True*, en este momento se muestra por la pantalla del Log el mensaje *"Realizando Trayectoria 2 …".* A continuación, se entra en el bucle encargado de reproducir la trayectoria grabada el cual se termina cuando se ha terminado de reproducir la trayectoria 2 (se haya recorrido el vector *trayec2* hasta la última posición grabada, indicada por la variable *w)* o también si se ha superado máxima fuerza de seguridad permitida (*fuerzaparar = True)* o si se ha pulsado el botón de emergencia (*P_Emergencia = True)*. Dentro del bucle se guarda en la variable *mover* el valor de la posición que hay que alcanzar en cada instante y se le pasa como parámetro a la función *movel.*

mover := trayec2[j]

Luego se modificará el estado de la garra *toolbool* en función de los diferentes estados de la garra grabados en la variable *vec_tool2.*

toolbool := vec_tool2[j]

Antes de volver a realizar el bucle se espera 0.5 s. Este tiempo es el tiempo cada cuanto se guarda la posición del robot durante la grabación. Al utilizar *movel* el robot se mueve para alcanzar los diferentes puntos, pero si se ha permanecido un tiempo en un punto concreto el robot salta al siguiente punto, por lo que no se respetaban los tiempos. Para solucionar esto se añade esta espera que obliga a estar al robot en la posición correcta el tiempo que ha permanecido en ella durante la grabación.

Una vez esperado el tiempo se aumenta el valor de *j* para ir recorriendo los diferentes vectores y reproducir la trayectoria.

Cuando se ha finalizado la realización de la trayectoria (se han recorrido los vectores hasta la última posición grabada, j=*w)* o se sale de bucle de reproducción por algunas de las otras causas indicadas anteriormente, se muestra *"Finalizada Trayectoria 2"* en la pestaña Log y se pone *j = 0* para poder volver a reproducirla y *hacertray2 = False* para indicar que ha terminado.

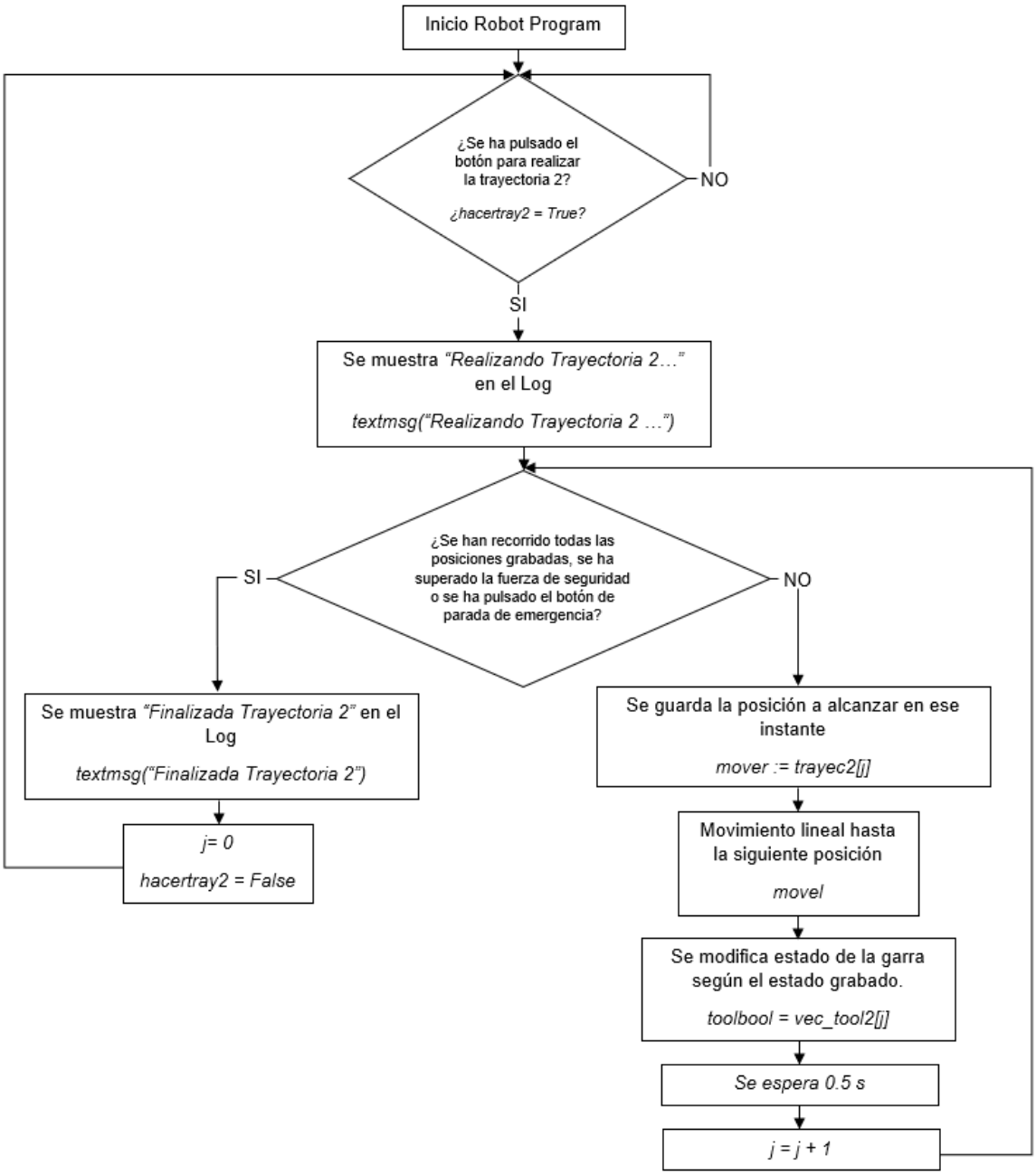


Figura 48.Diagrama de flujo de la parte Robot Program del programa del UR5e (parte4).

Thread2

En este hilo se gestiona la grabación de la trayectoria 1. Para comenzar la grabación se debe cumplir *grabarbool1 = True*, el valor de esta variable se gestiona en otro hilo. Mientras no se cumpla la condición el hilo no realiza ninguna acción, solo comprueba el valor de la condición. Una vez se cumple se comienza a grabar la trayectoria 1. Para ello se entra en un bucle en el que se permanecerá mientras que la variable *grabarbool1 = True*, lo que quiere decir que la grabación de la trayectoria no ha terminado, cuando se quiera finalizar la grabación bastará con que *grabarbool1 = False*. También se comprueba que no

se ha llegado a la última posición del vector *trayec1* (*i* < *nposiciones1)*, ya que, si se llega a la última posición y se continua el bucle al querer grabar en la posicion 101 el robot detiene su ejecución para informar del fallo, por eso se ha gestionado que se detenga al llegar a la última posición. Mientras se cumplan estas condiciones el este bucle se ejecutará. En el bucle se guarda el valor de la posición actual del robot en el vector *trayec1* para ello se utiliza una función disponible en PolyScope *get_actual_tcp_pose()* que devuelve el valor de la posicion de la herramienta en el instante actual.

trayec1[i] = get_actual_tcp_pose()

También se guarda el estado de la herramienta en el vector *vec_tool1*.

vec_tool1[i] = toolbool

Para ir recorriendo el vector se utiliza la variable *i* (por lo que al finalizar la grabación esta variable indica el número de posiciones grabadas) que va aumentando su valor cada vez que se guarda una posicion y se almacena el estado de la herramienta. A continuación, se realiza una espera de 0.5 s antes de guardar la siguiente posición, este tiempo se puede modificar, pero se realizaron pruebas con diferentes tiempos y con este se obtuvieron buenos resultados.

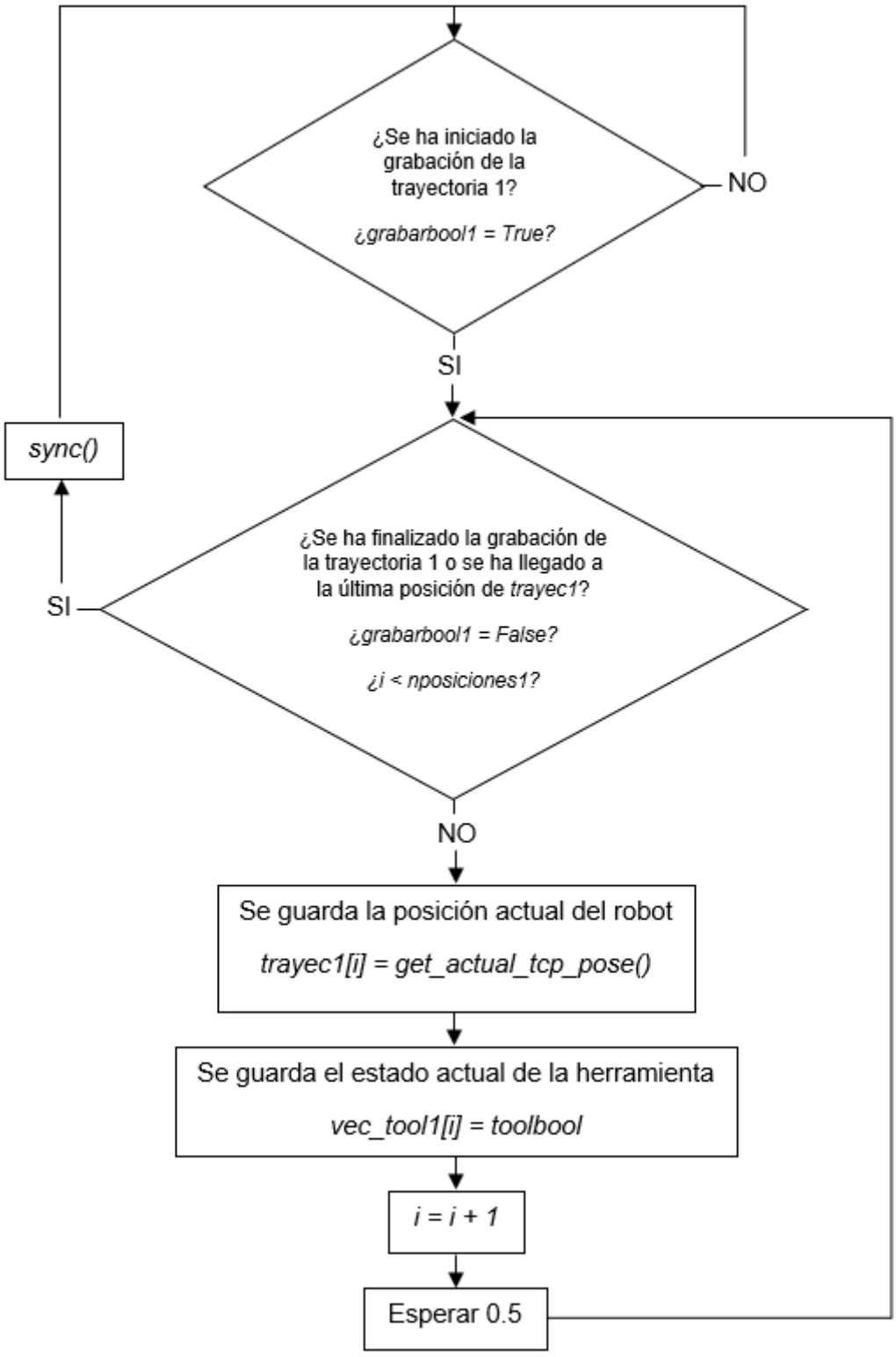


Figura 49.Diagrama de flujo de la parte Thread2 del programa del UR5e.

Thread5

Este hilo es el encargado de la grabación de la trayectoria 2. El funcionamiento es idéntico al del hilo 2, pero con las variables asociadas a la trayectoria 2 que son: *nposiciones2, grabarbool2, w, trayec2* y *vec_tool2.*

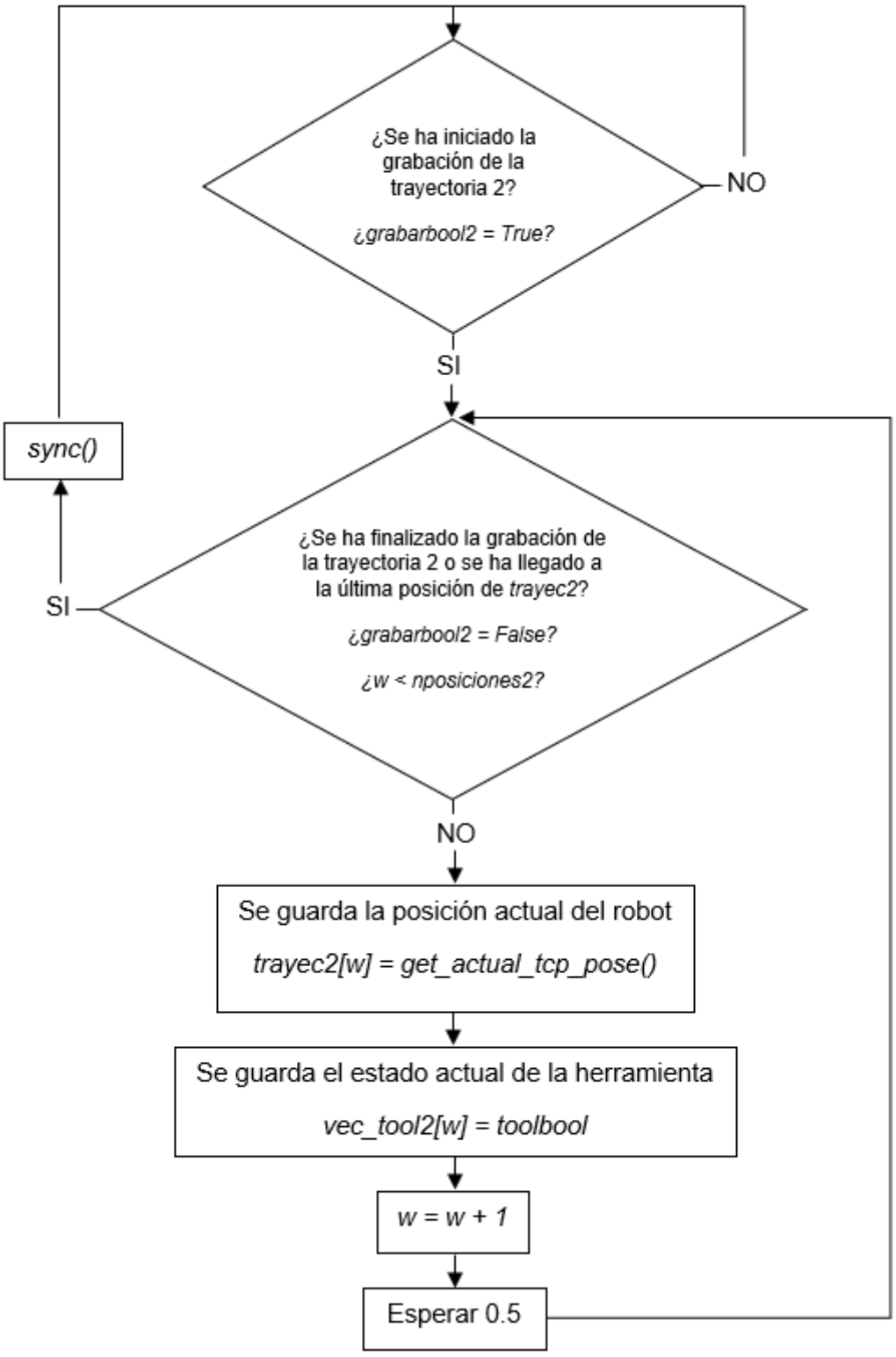


Figura 50. Diagrama de flujo de la parte Thread5 del programa del UR5e.

Thread4

En este hilo se define el estado de las variables que gestionan el inicio y el final de la grabación de las dos trayectorias y el comienzo de la realización de estas, es decir: *grabarbool1, grabarbool2, i, w, hacertray1 y hacertray2.* Para ello se utiliza el valor guardado en la variable *grabar* cuyo valor depende del vector recibido. Recordemos que esta variable podía tomar cuatro valores: el 1 para grabar la trayectoria 1, el 2 para grabar la trayectoria 2, el 3 para reproducir la trayectoria 1 y el 4 para reproducir la trayectoria 2.

El funcionamiento es el siguiente:

Cuando se recibe un 1 (*grabar = 1)* y no se está ya grabando la trayectoria 2 (*grabarbool2 = False)*, ni se está realizando ninguna trayectoria (*hacertray1 = False* y *hacertray2 = False)*, ni se ha superado la fuerza máxima permitida (*fuerzaparar = False)* ni se ha pulsado el botón de parada de emergencia (*P_Emergencia = False)* el robot está en condiciones de grabar la trayectoria 1. Desde el mando se envía un 1 tanto para comenzar como para finalizar la grabación de la trayectoria, por eso es aquí donde se gestiona cuál de las 2 acciones se debe llevar a cabo. Para ello se comprueba el valor de *grabarbool1* que será *True* si ya se estaba grabando, y por tanto el nuevo 1 recibido es para finalizar la grabación, o si es *False* y por tanto debe iniciarse la grabación.

En el caso *grabarbool1 = False* se inicia el contador *i = 0* para comenzar a rellenar el vector *trayec1* desde la primera posición. A continuación, se muestra el mensaje *"Grabando Trayectoria 1…"* en el Log y se cambia el valor de la variable *grabarbool1 = True* que indicará que se está grabando y provoca el inicio de la grabación en el Thread2. En el caso *grabarbool1 = True* se muestra el mensaje *"Trayectoria 1 Grabada Correctamente"* en el Log y se cambia el estado de la variable *grabarbool1 = False* que indicará que la grabación ha terminado. Para cambiar el estado de la variable se utiliza la siguiente instrucción:

grabarbool1 = not grabarbool1

Luego se espera 0.5 segundos, esta espera es debida a que se recibe un mensaje desde el mando cada 0.1 s por lo que mientras se pulsa el botón se pueden mandar varios 1, por lo que el estado de *grabarbool1* cambiaba rápidamente y no se realizaba la tarea correctamente. Al añadir este tiempo nos aseguramos de que el botón se ha dejado de pulsar antes del robot vuelva a comprobar el estado de la variable *grabar*.

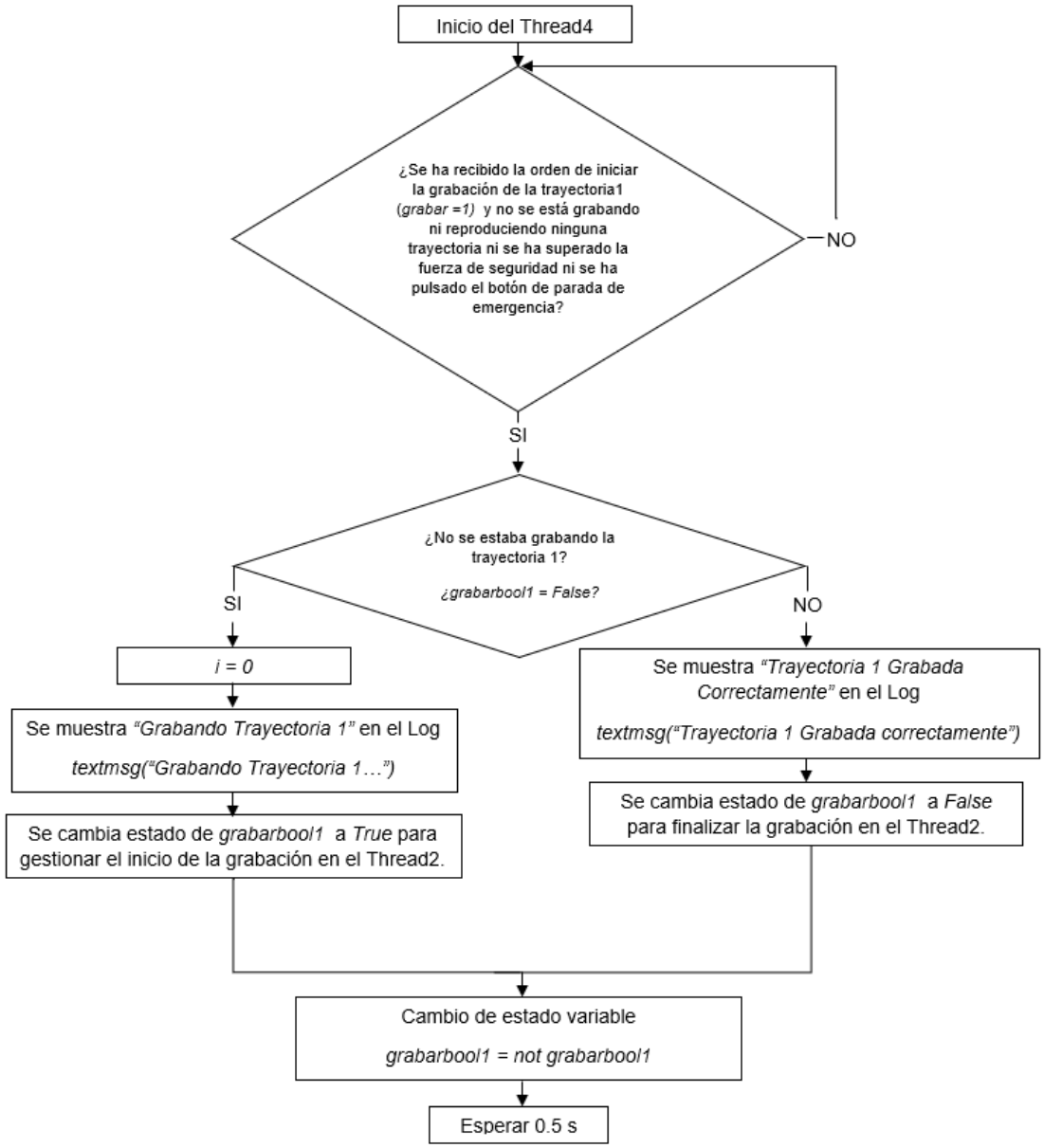


Figura 51. Diagrama de flujo del funcionamiento del Thread4 cuando grabar = 1.

Cuando se recibe un 2 (*grabar = 2)* y no se está grabando la trayectoria 1 (*grabarbool1 = False),* ni se está realizando ninguna trayectoria (*hacertray1 = False* y *hacertray2 = False)*, ni se ha superado la fuerza de seguridad (*fuerzaparar = False)* ni se ha realizado una parada de emergencia (*P_Emergencia = False)* el robot puede grabar la trayectoria 2. Al igual que pasaba en grabación de la trayectoria 1, cuando desde el mando se envía un 2 este sirve tanto para comenzar como para finalizar la grabación de la trayectoria 2 y es aquí donde se gestiona qué acción se debe realizar. Si al recibir el 2 *grabarbool2 = False*, lo que indica que no se estaba grabando la trayectoria, se inicia el contador *w = 0* para comenzar a grabar el vector *trayec2* desde el principio, se muestra el mensaje *"Grabando Trayectoria 2 …"* en el Log y se cambia el valor de la variable *grabarbool2 = True* para que se inicie la grabación en el thread5. En el caso que *grabarbool2 = True* el 2 recibido es para finalizar la grabación por lo que se muestra el mensaje *"Trayectoria 2 Grabada*

Correctamente" en el Log y se cambia el valor de la variable *grabarbool2 = False.* Para cambiar el estado de la variable se utiliza la misma función vista en el caso anterior. Luego se realiza una espera de 0.5 s antes de volver a comprobar el valor de la variable *grabar.*

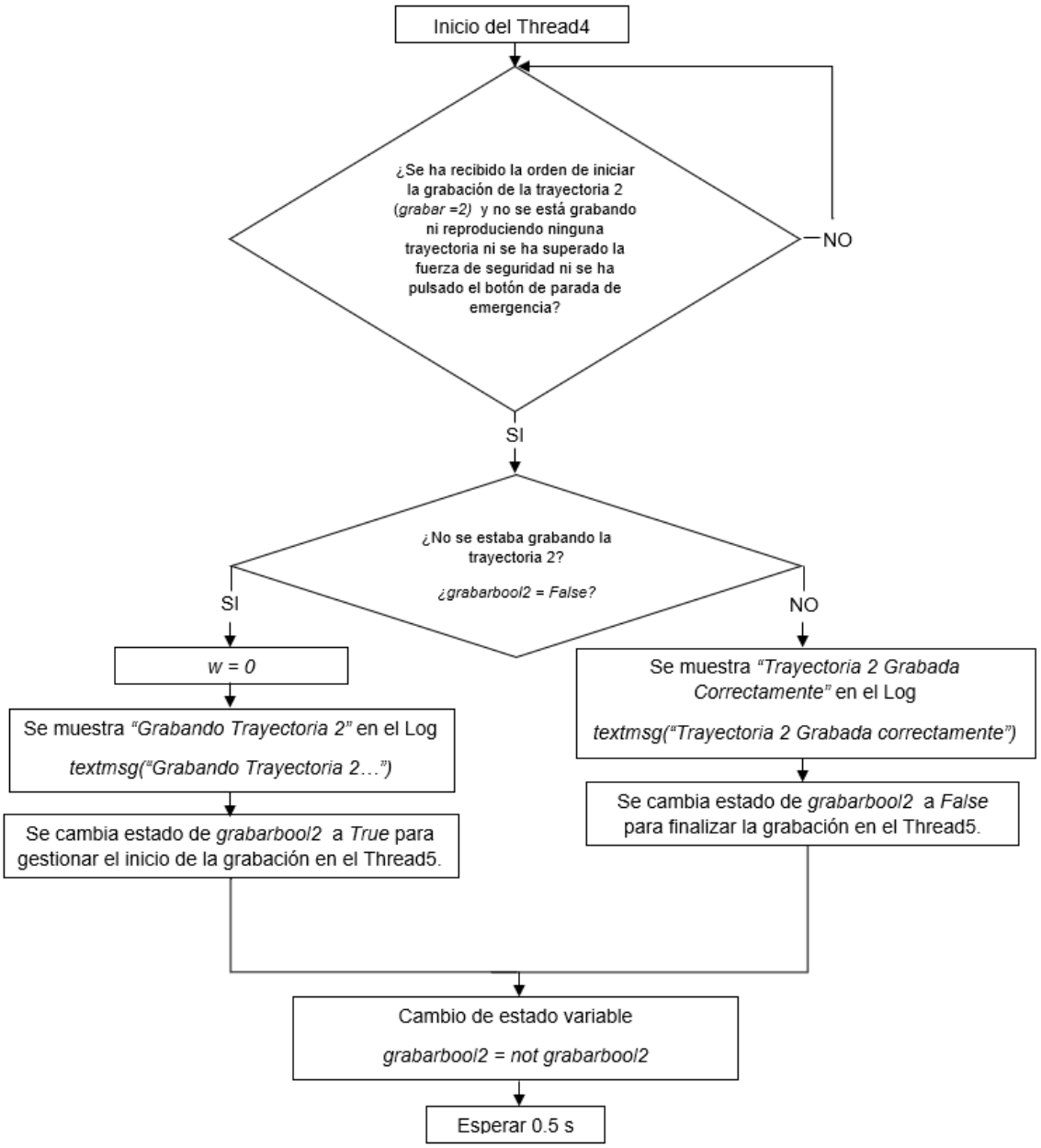


Figura 52. Diagrama de flujo del funcionamiento del Thread4 cuando grabar = 2.

Cuando se recibe un 3 (*grabar = 3)* y no se está realizando la trayectoria 2 (*hacertray2 = False)*, ni se está grabando ninguna de las 2 trayectorias (*grabarbool1 = False* y *grabarbool2 = False)*, ni se ha superado la fuerza máxima permitida (*fuerzaparar = False)* y no se ha pulsado el botón de parada de emergencia (*P_Emergencia = False)* el robot está en condiciones de poder realizar la trayectoria 1. Para ello simplemente hay que hacer *hacretray1 =True*

lo que provocará en el programa principal que se comience a ejecutar la trayectoria 1 grabada. En este caso no hay que gestionar nada más porque será en el programa principal donde se haga *hacertray1 = False* cuando se haya finalizado la realización de la trayectoria.

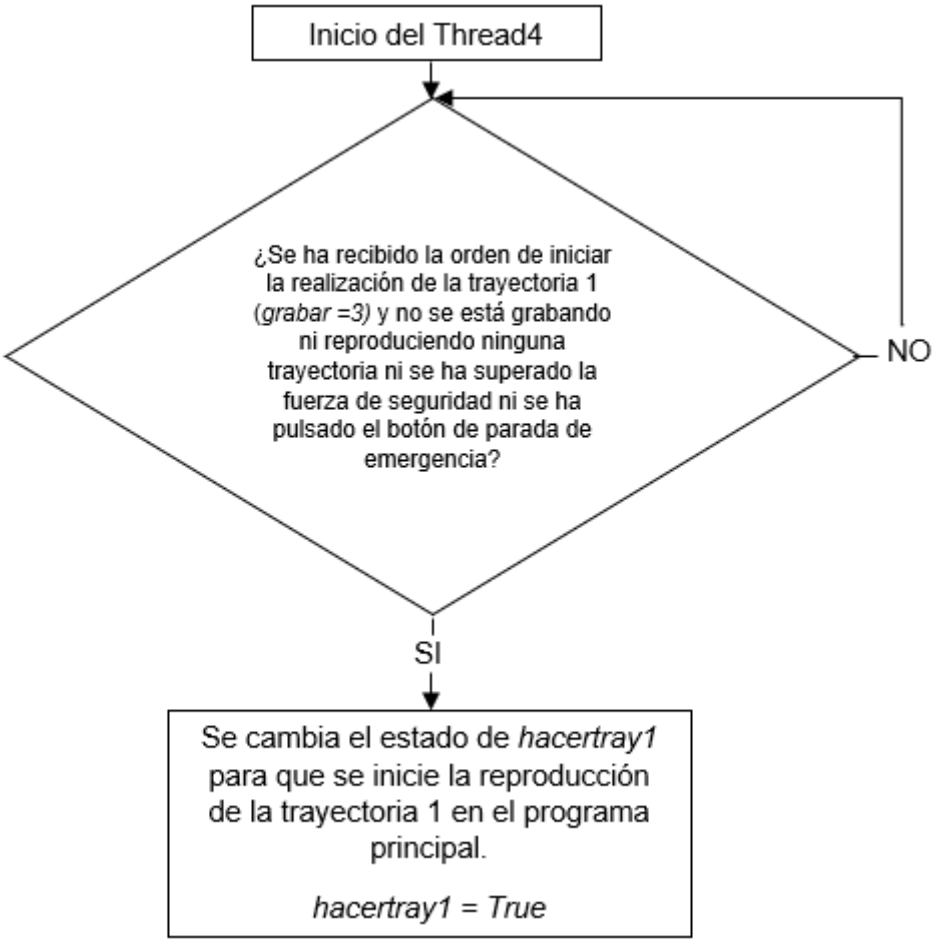


Figura 53.Diagrama de flujo del funcionamiento del Thread4 cuando grabar = 3.

Cuando se recibe un 4 (*grabar = 4)* y no se está realizando la trayectoria 1 (*hacertray1 = False)*, ni se está grabando ninguna de las 2 trayectorias (*grabarbool1 = False* y *grabarbool2 = False)*, ni se ha superado la fuerza máxima permitida (*fuerzaparar = False)* y no se ha pulsado el botón de parada de emergencia (*P_Emergencia = False)* el robot está en condiciones de poder realizar la trayectoria 2. Para ello simplemente hay que hacer *hacretray2 =True* lo que provocará en el programa principal que se comience a ejecutar la trayectoria 2 grabada. También es en el programa principal donde se hará *hacertray2 = False* cuando la trayectoria grabada haya finalizado.

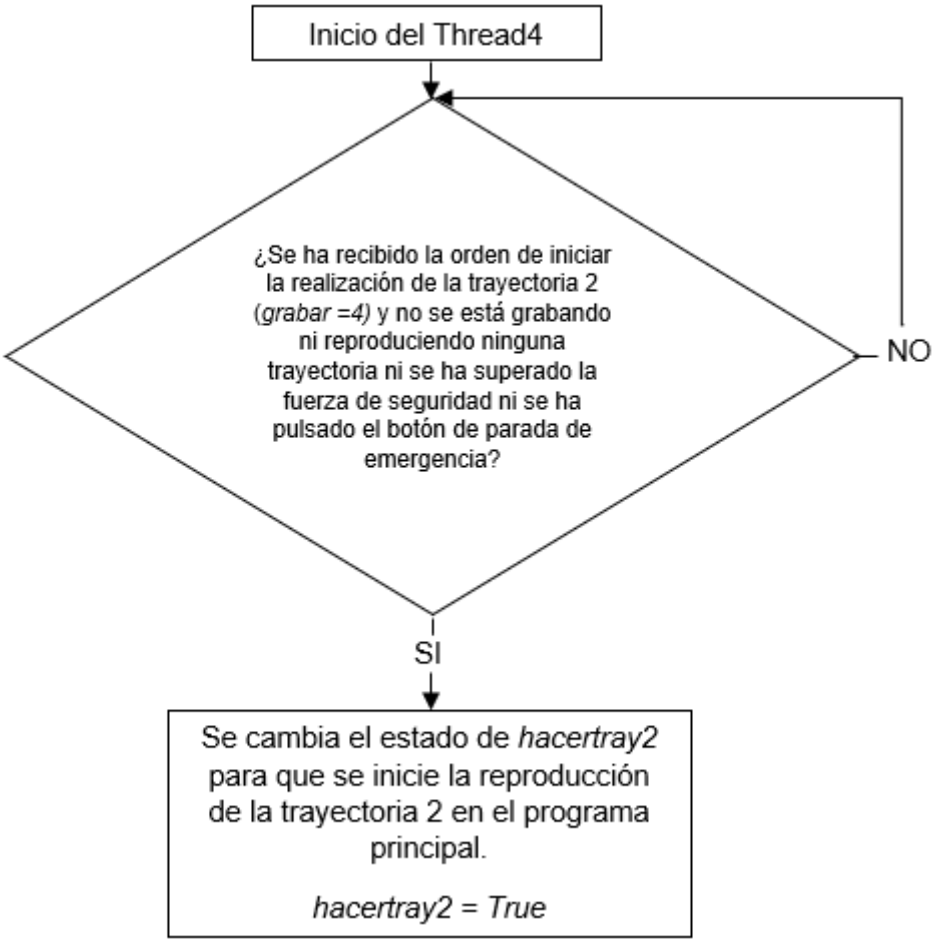


Figura 54. Diagrama de flujo del funcionamiento del Thread4 cuando grabar = 4.

Tal y como se ha gestionado el funcionamiento no hay ningún problema de que desde el mando se pulse algún botón mientras se está llevando a cabo alguna grabación o reproducción de trayectoria, ya que antes de modificar la variable encargada de gestionar la realización dichas acciones se comprueba el estado del resto. Por ejemplo, si se está grabando la trayectoria 1 y se recibe desde el mando un 4 para empezar a reproducir la trayectoria2, esto no producirá ningún problema en la grabación que se estaba llevando a cabo, ya que como *grabarbool1 = True* el valor de *hacertray2* no se modificará a pesar de que *grabar = 4,* de esta forma si se pulsa por error un botón que no se debe no se pierde el trabajo que se estaba realizando. Además, cuando se ha superado la fuerza de seguridad o se ha pulsado el botón de parada de emergencia no se permite comenzar a grabar ni a reproducir ninguna trayectoria. En caso de que esto ocurra mientras se estaba grabando una trayectoria tampoco se permite finalizar la grabación, pero cuando se pulse el botón *home*, necesario para salir de esta situación de bloqueo, se finaliza la grabación que se estaba ejecutando cuando ocurrió el problema.

Thread3

En este hilo se define el estado de las variables que gestionan las funciones de seguridad que se han incluido en la aplicación, como son la parada de emergencia (*P_Emergencia)* y la fuerza máxima permitida (*fuerzaparar)*.

También se gestiona las acciones que se deben llevar a cabo para que el robot vuelva a funcionar con normalidad de una forma segura después de que se haya producido uno de estos problemas. Por tanto, las tareas que realizan este hilo tienen que ver con las diferentes funciones de seguridad que se han incluido en la aplicación.

Por un lado, este hilo se encarga de observar el valor de la fuerza que se está ejerciendo en el eje Z de la herramienta y lo almacena en la variable *fuerzaz* mediante la siguiente instrucción:

fuerzaz = force()

Si esta variable supera el valor máximo permitido, que se ha establecido en 50 N, la variable *fuerzaparar* se vuelve *True* lo que provoca que no se pueda mover el robot ni libremente ni realizando una trayectoria (esto se gestiona en el programa principal, como se explicó anteriormente) y tampoco permite grabar trayectorias, ni comenzar a realizar trayectorias (esto se gestiona en el Thread4, como se explicó anteriormente). Luego el programa espera 0.5 segundos antes de volver a comprobar el valor de la variable, esto es porque si no se realiza esta espera el valor de la fuerza puede estar por encima del valor permitido en más de un instante y el mensaje que se va a mostrar por pantalla se muestra más de una vez. A continuación, se muestra el mensaje *"Fuerza de Seguridad Superada (N) = X"* (siendo X el valor de la fuerza alcanzada) en la pestaña Log de la consola de programación.

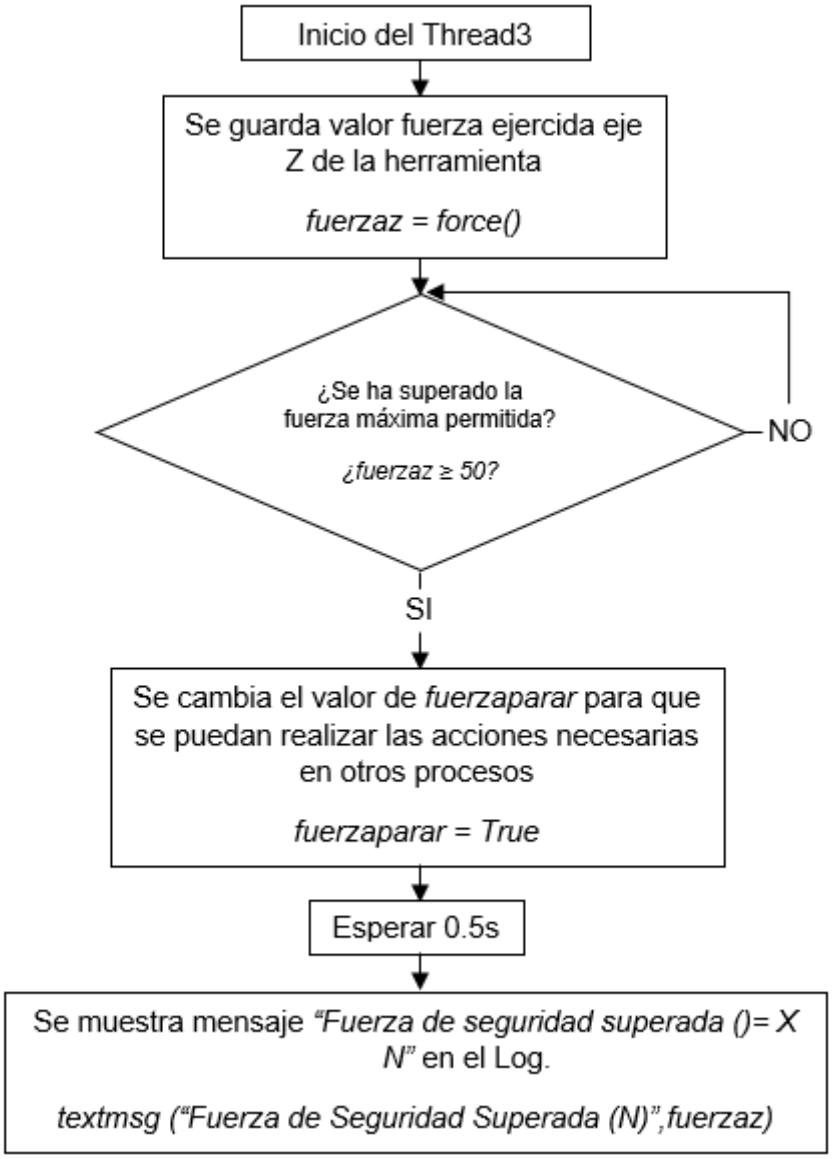


Figura 55.Diagrama de flujo del funcionamiento del Thread3 cuando se supera la fuerza de máxima permitida.

Por otro lado, este hilo comprueba si se ha pulsado en el mando el botón de emergencia, se recuerda que la posición del vector recibido con esta información se ha guardado en la variable *modo.* Por tanto, si se ha pulsado el botón de emergencia (*modo = 8)* la variable *P_Emergencia* se pone a *True* lo que tiene las mismas consecuencias que las vistas anteriormente para cuando *fuerzaparar* =*True.* Luego el programa muestra el mensaje *"PARADA DE EMERGENCIA"* en el Log. A continuación, se espera 0.5 segundos antes de volver a comprobar el valor de la variable, este tiempo es porque se envía un mensaje desde el mando cada 0.1 segundos por lo que al pulsar el botón puede que lleguen varias veces este mensaje, con este tiempo de espera se soluciona este error.

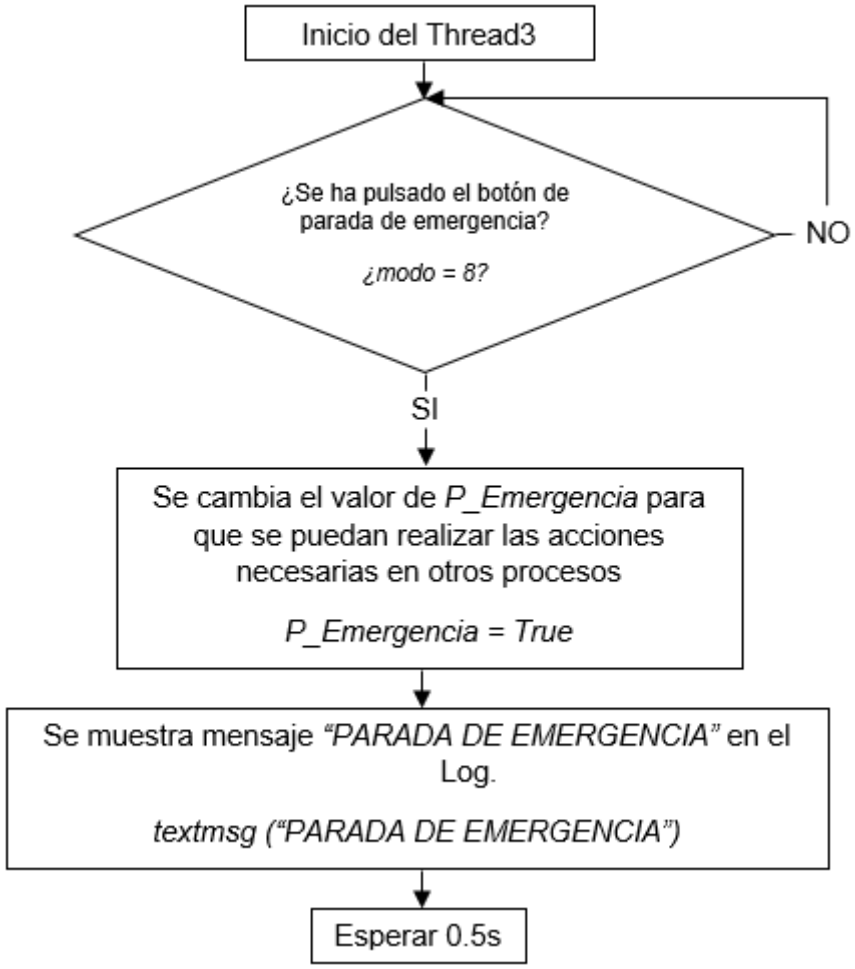


Figura 56.Diagrama de flujo del funcionamiento del Thread3 cuando se pulsa el botón de parada de emergencia (modo = 8).

Por último, cuando se produce alguna de estas dos situaciones el robot se queda en un estado de bloqueo donde no puede realizar ninguna acción. Para salir de esta situación se ha incluido el botón *home* que lleva al robot a un estado de equilibrio, donde no estará ni grabando ni realizando ninguna trayectoria, sino que permanecerá quieto hasta recibir nuevas órdenes desde el mando. Esta información también se ha guardado en la variable *modo*. El botón *home* debe pulsarse cuando la situación de peligro que causo el problema ha desaparecido, y, por tanto, cuando este se pulsa (*modo = 7)* se reestablecen los valores de las variables *fuerzaparar = Fase, P_Emergencia = False.* Además, para mayor seguridad si se estaba realizando una trayectoria cuando ocurrió el problema esta no continuará para ello se hace *hacertray1 = False* y *hacertray2 = False.* Además, si cuando ocurrió el problema se estaba grabando una trayectoria se finalizará la grabación, ya que durante el tiempo que el robot a estado bloqueado se ha continuado grabando y la trayectoria guardada no será correcta, para ello *grabarbool1 = False* y *grabarbool2 = False.* Después se muestra el mensaje *"Valores Restablecidos. Robot en funcionamiento normal"* en el Log. Para finalizar se espera 0.5 segundos antes de comprobar de nuevo el valor de la variable porque cuando se pulsa el botón *home* se envía más de un 7, lo mismo que pasaba con el botón de emergencia.

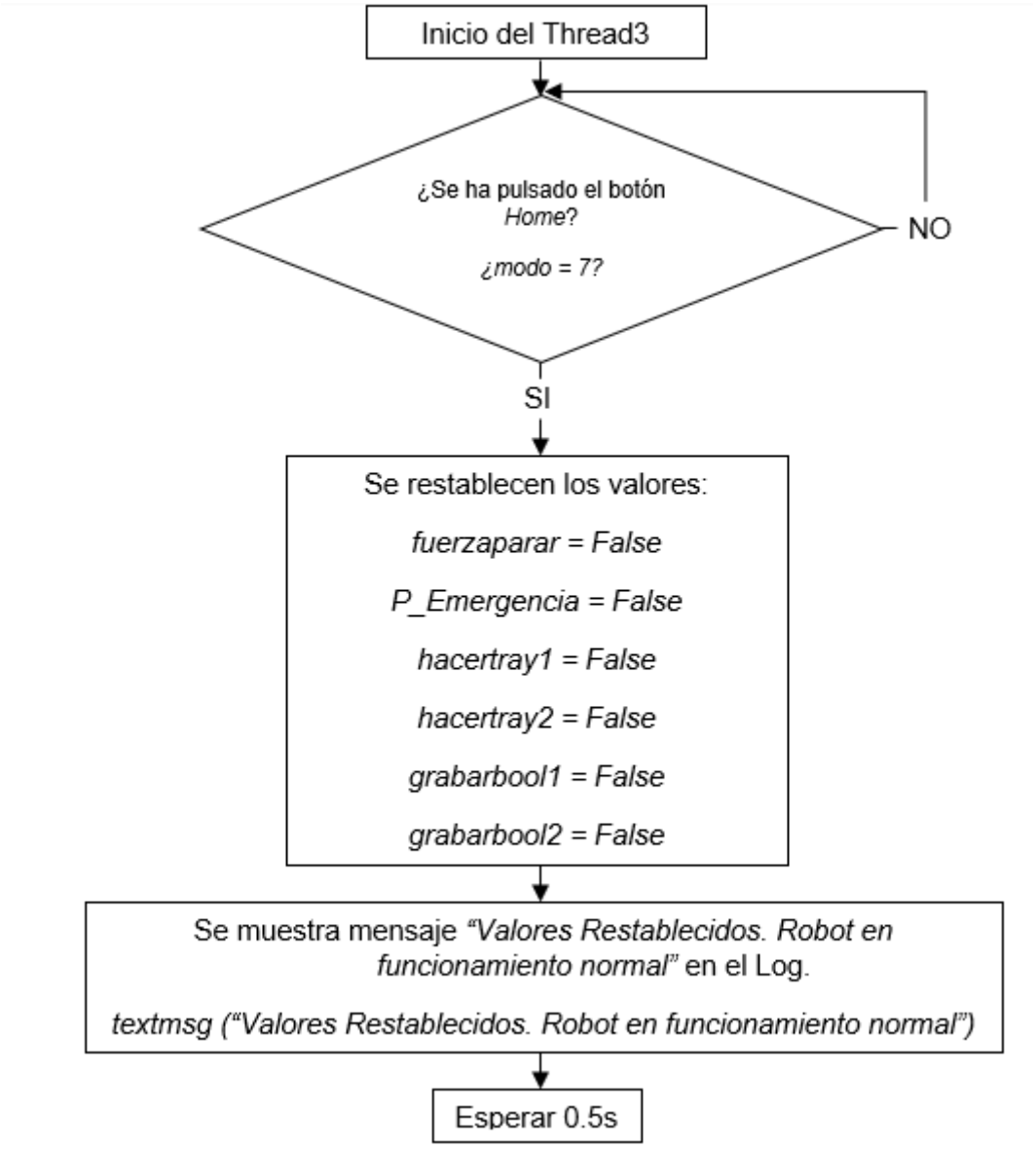


Figura 57.Diagrama de flujo del funcionamiento del Thread3 cuando se pulsa el botón Home (modo = 7).

Thread6

En este hilo se gestiona el estado de la herramienta, como se dijo, en este trabajo se ha utilizado una garra. Esta puede estar abierta o cerrada y para cambiar su estado el robot recibe desde el mando un mensaje, este está en la primera posición del vector y su valor se guarda en la variable *tool.* Si se ha pulsado el botón del joystick (*tool = 1)* y no se está realizando ninguna trayectoria, ni se ha superado la fuerza máxima permitida y tampoco se ha pulsado el botón de parada de emergencia la garra cambia de estado. El estado de la garra lo define el valor de la variable *toolbool* que cuando es *True* la garra se abre y cuando es *False* se cierra.

toolbool = not toolbool

Para abrir o cerrar la garra hay que indicarle los milímetros de apertura. Las funciones utilizadas aparecen en PolyScope cuando se conecta la garra. Para la garra abierta:

RG2(80)

Y para la garra cerrada:

RG2(0)

Tal y como se ha gestionado la garra solo se puede manipular cuando se mueve el robot libremente o cuando se está grabando una trayectoria. Esto es para que si se pulsa el botón del joystick mientras se realiza una trayectoria grabada no influya y la garra siga los estados grabados (esto es porque se ha utilizado la misma variable *toolbool* y si no se hacia esta restricción se podía manipular su estado). Además, cuando el robot está bloqueado porque ha superado la fuerza máxima permitida o porque se ha pulsado el botón de parada de emergencia la garra se quedará quieta en el estado en el que se encontrara cuando se produjo el problema.

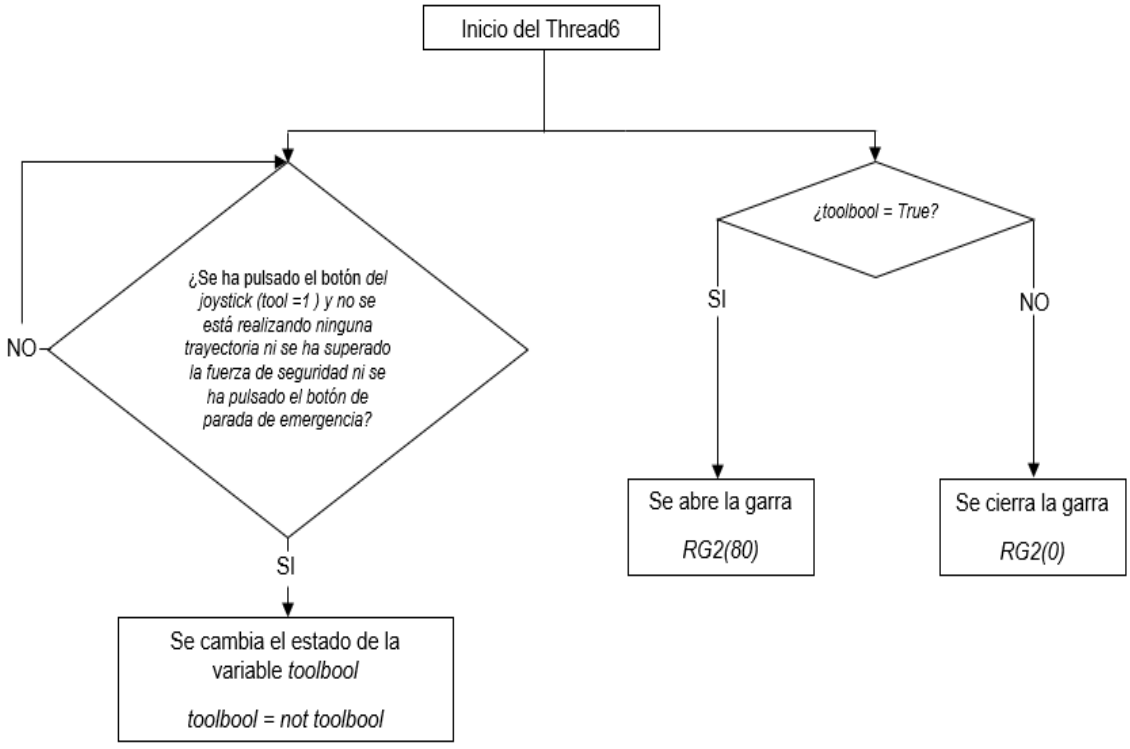


Figura 58.Diagrama de flujo del funcionamiento del Thread6.

3.2.3 Opciones de alimentación.

Para poder utilizar el mando construido se debe alimentar la placa Arduino. Existen diferentes opciones para realizar esta tarea, en este apartado se analizarán alguna de ellas y se presentará la opción elegida y las ventajas que presenta.

La placa de Arduino permite diferentes opciones de alimentación: dispone de un clavija Jack donde se pueden aplicar voltajes de entre 6 -12 V (utiliza el regulador de voltaje incluido en la placa), también se puede aplicar 6 -12 V entre el pin GND y el pin Vin de la placa (utiliza el regulador de voltaje incluido en la placa), se permite alimentar mediante USB y aplicar 5 V en el pin de 5V (es importante que estén regulados y sean estables ya que aquí no se está usando el regulador de voltaje y una variación de la tensión dañará la placa). En la aplicación diseñada la placa arduino no va a estar conectada al PC por lo que hay que buscar una alternativa, el uso de baterías es una solución adecuada para este proyecto.

Una de las opciones analizadas es utilizar pilas, estas son sencillas de usar ya que se dispone de portapilas con clavija Jack que permite conectarlas fácilmente con la placa. Dentro de esta opción se pueden utilizar pilas de 9V o 4 pilas de 1.5V (entre otras). Dentro de estas el uso de 4 pilas de 1.5V presenta más ventajas ya que se puede obtener mayor intensidad máxima, su vida es mayor, son más comunes. Aunque estas cumplen con los requisitos necesarios para alimentar el mando diseñado se prefirió utilizar otros métodos, ya que el hecho de que no sean recargables es un inconveniente. Existen otras opciones como pilas recargables, pero tampoco aportan ninguna ventaja sobre la solución elegida.

Para alimentar el mando diseñado se va a utilizar una batería USB. Las principales ventajas es que se puede conectar a la placa directamente por USB, sin preocuparse de regular el voltaje, ya que estas baterías proporcionan 5V. Con esta tensión se pueden alimentar todos los componentes del mando por lo que cumple con los requisitos del proyecto. Otra ventaja es que estas baterías son recargables, por lo que, aunque la inversión inicial es más grande que si se utilizaran pilas a largo plazo es más rentable. Además, al alimentar el Arduino por USB en el mando se tendrá disponible este puerto y en caso de que se acabe la batería se puede conectar este cable a un PC o a un cargador de móvil para alimentar la placa, también al poderse conectar al PC permite cargar alguna actualización del programa o utilizar las funciones que se implementaron para comprobar que la conexión es correcta, de una manera sencilla. Por las facilidades que esta ofrece con respecto a las otras opciones ha sido finalmente la solución adoptada.

3.2.4 Conexión de los diferentes dispositivos a la placa de Arduino.

El mando diseñado dispone de 9 botones, un joystick y el módulo con W5100. Todos estos dispositivos se deben conectar a la placa de forma correcta. En este apartado se explicará cómo se debe conectar los diferentes dispositivos a la placa.

Por un lado, los botones se conectan a las entradas digitales de la placa, para ello se realizará un montaje con una resistencia Pull Down de 4.7 kΩ. Para la conexión se utiliza una tensión de referencia de 5V. Tal y como se ha conectado el botón cuando este esté pulsado se leerá HIGH.

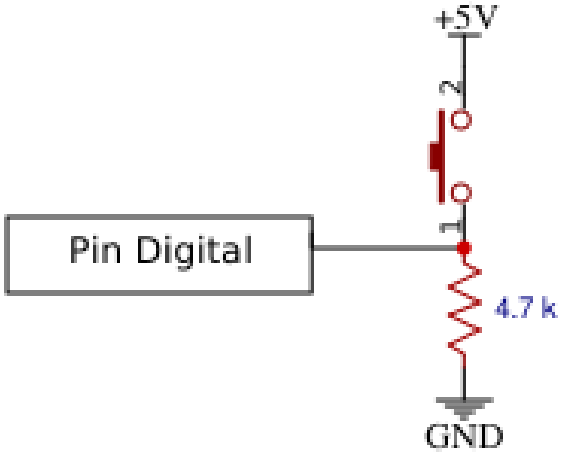


Figura 59. Esquema montaje botón.

Por otro lado, el joystick se conecta a las entradas analógicas de la placa, para conocer la posición de sus ejes, y el pulsador que incorpora a una entrada digital. El joystick dispone de 5 patas: Vcc, GND, VRX, VRY y SW. Para alimentar el módulo se conecta la pata Vcc al pin de 5V y el GND al pin GND de la placa. Para la lectura de los valores analógicos se conecta el VRX al pin A0 y el VRY al pin A1. Para utilizar el botón del joystick se ha conectado directamente a la entrada digital 12 (que como se dijo se ha configurado de modo INPUT_PULLUP para evitar el uso de resistencias externas). Por tanto, todos los pines del módulo se conectan directamente al pin correspondiente de la placa de Arduino.

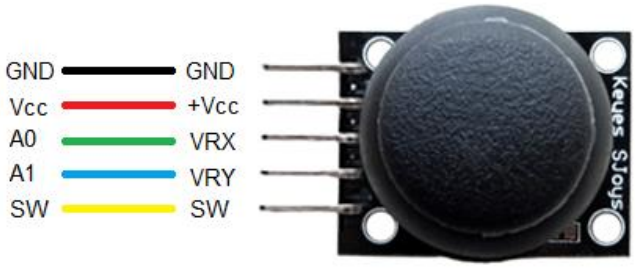


Figura 60. Esquema conexión joystick a la placa de Arduino.

Por último, la conexión del módulo con W5100 se realiza a través de SPI. Este dispone de 10 pines: Vcc, GND, SS, MOSI, RST, SCK, MISO, P+ y P-. El pin Vcc se conecta a 5V y GND al pin GND de la placa. Para la conexión SPI la placa de Arduino MEGA tiene unos pines concretos reservados para esta función. Para este proyecto se va a conectar: el SS al pin 10, el MOSI al pin 51, el SCK al pin 52 y el MISO al pin 50. El resto de los pines se dejan libres. Por tanto, todos estos pines se conectan directamente a la placa.

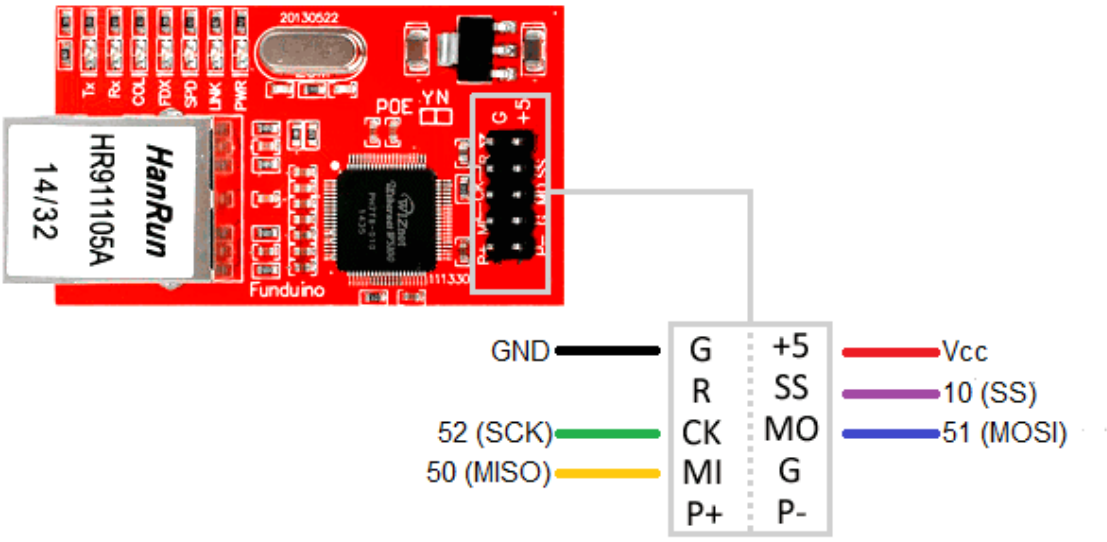


Figura 61.Esquema conexión módulo W5100 a la placa de Arduino.

Todos los dispositivos conectados se alimentan a 5V mediante el pin de 5V de la placa y se conectan al pin GND de la placa. En la Tabla 13 se muestra un resumen del resto de conexiones.

Dispositivo	Pin	Modo
Botón 1	2	Entrada digital
Botón 2	3	Entrada digital
Botón 3	5	Entrada digital
Botón 4	6	Entrada digital
Botón Home	7	Entrada digital
Botón Emergencia	8	Entrada digital
Botón T1	9	Entrada digital
Modulo W5100	10	SS
Botón T2	11	Entrada digital
Botón joystick	12	Entrada digital
Botón DT	13	Entrada digital
Modulo W5100	50	MISO
Modulo W5100	51	MOSI
Modulo W5100	52	SCK
Joystick	A0	Entrada analógica
Joystick	A1	Entrada analógica

Tabla 13.Pines de conexión de los dispositivos del mando a la placa Arduino.

Los esquemas de conexión se adjuntan en el ANEXO II.

3.2.5 Mando construido.

El diseño del mando se ha hecho con la idea de que su uso sea lo más cómodo posible para el usuario, teniendo en cuenta las restricciones impuestas por los diferentes dispositivos que debe albergar en su interior.

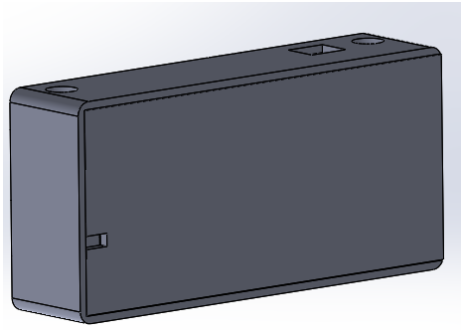


Figura 62. Parte trasera mando diseñado.

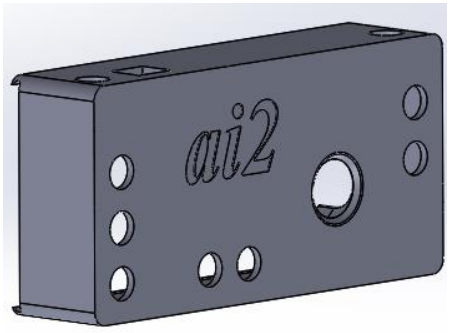


Figura 63. Parte delantera mando diseñado.

El tamaño del mando está muy condicionado por el tamaño de la placa Arduino MEGA (101.6 mm x 53.3 mm), además de la placa debe incluir el módulo W5100, el joystick y los 9 botones. Las medidas del mando diseñado son 175 mm x 84 mm x 40 mm. El diseño del mando se ha realizado con *SolidWorks* y se ha impreso con filamento PLA (ácido poliláctico), la estructura del mando pesa 112 g más los 44 g de la tapa.

La disposición elegida para los botones permite un manejo fácil de las diferentes opciones que nos ofrece el mando, ya que se podrá realizar cualquier función tan solo utilizando los dedos pulgar e índice. El joystick está en la parte derecha para poder manipularse con dicha mano.

El conector ethernet del módulo W5100 está en la parte frontal, de manera que el cable no moleste durante el manejo del mando.

Para no hacer el mando más grande la batería que se va a usar para alimentar la placa no irá en el interior de este. La batería va enganchada a la parte inferior del mando mediante un velcro y para realizar la conexión el cable USB conectado a la placa Arduino sale al exterior del mando por la parte inferior de este. Este método aporta ciertas ventajas como la facilidad para intercambiar una batería agotada por otra. Además, disponer del cable USB en el exterior permite ampliar las opciones para alimentar la placa ya que se puede alimentar conectando el cable al PC o a un enchufe mediante un transformador. También permite conectar la placa al PC por si fuera necesario una modificación del código o para comprobar que la conexión TCP/IP se ha establecido de forma correcta (como se dijo anteriormente se han implementado unas funciones para esto).

En la siguiente figura se muestra el mando diseñado una vez impreso y montado.

Figura 64. Mando diseñado impreso y montado.

Los planos del mando se adjuntan en el ANEXO III.

3.3 Mando Wii™.

Además de la solución desarrollada con el mando Arduino se ha diseñado una aplicación para controlar el robot mediante el mando de la Wii™. Esto se ha realizado para comparar las diferentes posibilidades que ofrecen estos dispositivos. Para poder manipular el UR5e desde este mando se necesita un PC como dispositivo intermedio, ya que el mando de la Wii™ se comunicará con el PC mediante bluetooth y es el PC el que establece la comunicación con el robot mediante TCP/IP. El mando de la Wii™ dispone de diferentes botones que permiten configurar los diferentes modos de funcionamiento y un joystick para guiar al robot.

3.3.1 Comunicación entre el mando Wii™ y el UR5e.

Para controlar el UR5e mediante el mando de la Wii™ el primer paso es establecer la comunicación. Como se ha visto anteriormente establecer una comunicación por TCP/IP es muy sencillo ya que el robot dispone de funciones específicas para conectarse como cliente a un servidor mediante este protocolo. Por tanto, se debe construir un servidor que ofrezca información relativa al estado de los botones y del joystick del mando. Para lograrlo se debe utilizar un dispositivo intermedio que permita la implementación de un servidor, ya que el mando no dispone de las herramientas necesarias para ello, y por tanto el mando deberá conectarse al dispositivo para ofrecerle la información necesaria. Para ello, se va a utilizar como dispositivo intermedio un PC, este se comunica con el mando mediante bluetooth y con el UR5e mediante TCP/IP. El PC cumple con los requisitos necesarios, ya que permite la conexión bluetooth y dispone de herramientas para implementar un servidor (se implementará en Java).

Para conectar el mando con el PC mediante bluetooth se ha utilizado el programa *Dolphin* que facilita la conexión de este.

Para implementar el servidor en Java que se comunique con el robot es necesario que este sea capaz de leer la información recibida desde el mando. Esto se logra gracias a la librería *wiiusej* que es una clase con diferentes métodos que permite conocer el estado del mando de la Wii™ desde Java, a partir de los ejemplos de uso de esta clase se ha desarrollado la aplicación utilizando los métodos que ofrece y creando nuevos métodos más específicos para la aplicación desarrollada.

En esta aplicación se utiliza a nivel de transporte el protocolo TCP, por los mismos motivos que se utilizó en el mando de Arduino.

3.3.2 Funcionamiento y algoritmos diseñados.

El funcionamiento de esta aplicación es muy sencillo, una vez establecida la comunicación entre los diferentes dispositivos se inicia un intercambio de mensajes entre el UR5e y el PC, este intercambio de mensajes es posible ya que ambos permiten una programación concurrente (se tiene diferentes hilos para enviar y para recibir por lo que no importa que las funciones sean bloqueantes, a diferencia de lo que ocurría en la aplicación de Arduino). Desde el PC se envía al robot el vector construido a partir del estado de los botones y de la posición del joystick del mando de la Wii™. A partir de este mensaje el UR5e realiza la acción correspondiente (las acciones que se pueden realizar con el robot son las mismas que para la aplicación del mando de Arduino). Por su parte el robot envía mensajes al PC con información sobre las acciones que está realizando, esta información se muestra al usuario en la interfaz diseñada y también se muestra en el mando de la Wii™ (los diferentes mensajes que se reciben, así como la forma en la que estos se muestran al usuario se explicará más adelante).

Para poder llevar a cabo está aplicación se debe implementar dos programas. El primero se ejecuta en el PC en este se muestra una interfaz de usuario donde se configura la comunicación y se inicia el servidor a la espera de que se conecte un cliente. Cuando se conecta un cliente se inicia un hilo que construye el vector con la información del estado de los botones y joystick del mando que se envía al UR5e. Y también se inicia un hilo encargado de recibir la información enviada desde el robot y realizar las acciones asociadas al mensaje recibido. Además, desde la interfaz diseñada se permite finalizar la comunicación, entre otras funciones.

El segundo programa se ejecuta en el UR5e, este es el encargado de iniciar la comunicación con el PC Una vez establecida recibe el vector, guarda los valores recibidos y realiza las acciones asociadas a los valores recibidos. En este vector se muestra información sobre el modo de funcionamiento, sobre la grabación y reproducción de trayectorias, estado de la herramienta y velocidades de cada eje. Además, durante la realización de las diferentes acciones envía un mensaje al servidor con información sobre la tarea que se está realizando.

En la aplicación diseñada se puede guiar el robot mediante el joystick, los diferentes modos de funcionamiento permiten mover el robot en los ejes XY y Z y realizar giros en YZ y X (esto es una diferencia con el mando de Arduino que tiene que ver con la disposición de los botones del mando Wii™). Todos estos movimientos se realizan ajustando los valores de velocidad de cada eje de la herramienta con respecto al sistema de coordenadas de la base. En esta aplicación también se pueden grabar dos tipos de trayectorias una de movimientos rectilíneos y otra de movimientos libres. Además, una vez grabadas se pueden reproducir cuando se desee. En esta aplicación también se han añadido las funciones de seguridad asociadas a que se supere la fuerza máxima permitida en la herramienta y a que se pulse el botón de parada de emergencia. Para salir de esta situación de bloqueo se debe pulsar el botón *home* del mando. También se ha gestionado la posibilidad de manipular la garra, mediante un botón del mando, y grabar su estado para que se reproduzca junto con la trayectoria.

A continuación, se explicará de forma detallada cada uno de los algoritmos diseñados.

Programa PC

Se ha construido un proyecto java al que se le ha incluido la librería *wiiuse.jar,* que nos permite interactuar con el mando desde la aplicación Java, en el proyecto se diferencian 2 paquetes:

- *AppGUI:* En este está el programa *appGUI,* es el programa principal en este se define la interfaz de usuario y las acciones que se llevan a cabo cuando se pulsan los diferentes botones de la interfaz.

- *Server:* En este paquete se ha implementado la clase *servidorTCP* que dispone del constructor y los métodos necesarios para iniciar un servidor y establecer la conexión con un cliente y enviar y recibir mensajes. También se ha implementado la clase *NunchukMandoWii* que dispone de los métodos para interactuar con el mando, tanto para obtener la información deseada como para modificar algunos de sus actuadores.

A continuación, se explicará cada uno de los programas que se han implementado centrándose en los métodos y funciones utilizadas para el desarrollo del proyecto. Se comienza explicando la clase *NunchukMandoWii*, luego se explicará la clase *ServidorTCP* y por último *appGUI.*

NunchukMandoWii

Esta clase sirve para interaccionar con el mando de la Wii™, en esta se definen una serie de métodos que serán utilizados en el resto de los programas.

En esta clase se definen como variables los diferentes botones del mando que se van a utilizar y los valores del joystick. Y se crea un objeto de la clase Wiimote

que da acceso a diferentes funciones que permiten interaccionar con el mando. Además, en esta clase se dispone de diferentes manejadores de eventos que permiten saber que un botón se ha pulsado, que el nunchuk se ha conectado y que el joystick se ha movido.

En la siguiente tabla se muestran las variables que se van a utilizar.

Variable	Tipo	Función
ControlActivo	Boolean	Sirve para indicar cuando se ha conectado el Nunchuk. Esto se debe saber porque hasta que no esté conectado ciertos botones y el joystick no estarán disponibles
botonC	Boolean	Estado del botón C del nunchuk
botonZ	Boolean	Estado del botón Z del nunchuk
botonA	Boolean	Estado del botón A del wiimote
botonB	Boolean	Estado del botón B del wiimote
boton1	Boolean	Estado del botón 1 del wiimote
boton2	Boolean	Estado del botón 2 del wiimote
botonHome	Boolean	Estado del botón Home del wiimote
botonArriba	Boolean	Estado del botón de la flecha de arriba del wiimote
botonAbajo	Boolean	Estado del botón de la flecha de abajo del wiimote
botonDerecha	Boolean	Estado del botón de la flecha de la derecha del wiimote
botonIzquierda	Boolean	Estado del botón de la flecha de la izquierda del wiimote
joyX	float	Guardar el valor de la posición del joystick. Valor que se envía al UR5e. Velocidad del eje.
joyY	float	Guardar el valor de la posición del joystick. Valor que se envía al UR5e. Velocidad del eje.
joyMin	float	Variable auxiliar para calcular el valor de las variables joyX y joyY.
joyMax	float	Variable auxiliar para calcular el valor de las variables joyX y joyY.

Tabla 14. Variables del programa NunchukMandoWii.

Para poder utilizar el mando para controlar el robot es necesario que el nunchuk esté conectado al wiimote, ya que es este el que dispone del joystick. En esta clase hay un manejador de eventos que informa de que se ha conectado, pone la variable *controlActivo* = *true* y enciende los leds 2 y 3 del wiimote.

Una vez se ha conectado el wiimote y el nunchuk es necesario conocer el estado de los diferentes botones y del joystick para construir el vector que se debe enviar al UR5e. Para ello se gestiona las acciones que se deben llevar a cabo cuando ocurren los diferentes eventos. Por ejemplo, cuando se pulsa el botón A del mando la variable *botonA* = *true* y si se suelta *botonA* = *false.* Esto se hace para todos los botones que se van a utilizar.

```
public void onButtonsEvent(WiimoteButtonsEvent arg0) {
    if (controlActivo && estadoWii.isConnected()) {

        //Boton A
        if (arg0.isButtonAJustPressed()) {
            botonA = true;
        } else if (arg0.isButtonAJustReleased()) {
            botonA = false;
        }
        //Boton Arriba
        if (arg0.isButtonUpJustPressed()) {
            botonArriba = true;
        } else if (arg0.isButtonUpJustReleased()) {
            botonArriba = false;
        }
```

Figura 65. Fragmento del código NunchukMandoWii.

Una vez se dispone de la información del mando se puede construir el vector a enviar. Para ello se dispone del método ***getControlArray()*** en este se construye el vector de 9 posiciones a partir del estado de los botones y del joystick. A continuación, se explicará la función de las diferentes posiciones.

Posición 0

En esta posición se manda la información relativa a la grabación y reproducción de trayectorias. El valor de esta posición depende de los botones de las flechas (*botonArriba, botonAbajo, botonDerecha* y *botonIzquierda).* En la siguiente tabla se muestran los diferentes valores que puede tomar esta posicion del vector, su significado y la combinación de botones para lograrlo.

Posición 0	Función	Combinación de botones	Valor vector[0]	Significado
	Indicarle al robot que comience o termine una grabación de una trayectoria y que realice alguna de las trayectorias grabadas	Ninguno	0	Ninguna acción
		Botón Arriba	1	Iniciar o terminar la grabación de la trayectoria 1 (cuál de las dos acciones se debe realizar se gestiona en el UR5e)
		Botón Abajo	2	Iniciar o terminar la grabación de la trayectoria 2 (cuál de las dos acciones se debe realizar se gestiona en el UR5e)
		Botón Derecha	3	Realizar la trayectoria 1
		Botón Izquierda	4	Realizar la trayectoria 2

Tabla 15. Posicion 0 del vector a enviar al UR5e desde el PC.

Posición 1

La posición 1 sirve para seleccionar el modo de funcionamiento. El valor de esta posición del vector depende del botón B (*botonB),* botón Z (*botonZ),* botón A (*botonA*), botón Home (*botonHome*) y botón de Emergencia (*boton1* y *boton2*). Se han configurado 6 modos de funcionamiento diferentes. Los botones B, Z y A sirven para mover el robot y realizar giros en diferentes ejes. Por su parte el botón Home sirve para indicar al robot que debe restablecer ciertos valores. Y por último el botón de Emergencia que indica al robot que debe realizar una parada de emergencia (se han configurado los botones 1 y 2 como botón de parada de emergencia). En la siguiente tabla se explica los diferentes modos de funcionamiento y la combinación de botones para lograrlo:

Posición 1	Función	Combinación de botones	Valor vector[1]	Significado
	Seleccionar el modo de funcionamiento	Ningún botón pulsado	0	El robot no se mueve
		botonB y no parada de emergencia	1	Movimientos en el plano XY
		botonB + botonZ y no parada de emergencia	2	Movimientos en el eje Z
		botonB + botonA y no parada de emergencia	3	Giros en los ejes Y Z
		botonB + botonA + botonZ y no parada de emergencia	4	Giros en el eje X
		botonHome	5	Se ha pulsado el botón Home
		boton1 o boton2 (botones de parada de emergencia)	6	Se ha pulsado el botón parada de emergencia.

Tabla 16.Posicion 1 del vector a enviar al UR5e desde el PC.

Posición 2,3,4,5,6 y 7:

Los diferentes modos de funcionamiento, que se guardan en la posición 1 del vector indican el movimiento que debe realizar el robot. Cuando los modos son el 5 o el 6 no se debe gestionar nada más, ya que es el UR5e el que gestiona las acciones a llevar a cabo. Sin embargo, el resto de los modos indican el eje sobre el que se va a mover el robot por lo que en función del modo se rellenan unas posiciones u otras (igual que se hizo en el mando Arduino). Antes de comenzar a rellenar estas posiciones se pone a 0 todas para evitar que se envíen valores antiguos.

Para llevar a cabo esta función se utiliza la estructura *switch* que comprueba el valor de la posición 1 del vector para conocer el modo en el que se encuentra y así poder rellenar las posiciones del vector correctas. En la siguiente tabla se presenta la función de estas posiciones.

Posición	Función
Posición 2	Velocidad desplazamiento en el eje X
Posición 3	Velocidad desplazamiento en el eje Y
Posición 4	Velocidad desplazamiento en el eje Z
Posición 5	Velocidad giro en el eje X
Posición 6	Velocidad giro en el eje Y
Posición 7	Velocidad giro en el eje Z

Tabla 17.Posición 2,3,4,5,6 y 7 del vector que se envía al UR5e desde el PC.

Las diferentes posiciones se rellenan con el valor de *joyX* y *joyY* , que se calculan en mediante la función *calcularJoystick()* que define el valor de estas variables en función de la posición del joystick. A continuación, se muestra cómo se rellena el vector en función del modo seleccionado:

- *Case 0:* el robot no se va a mover por lo que todas las posiciones son 0.
- *Case 1*: el robot se mueve en el plano XY. Por tanto, *vector[2] = joyX* y *vector[3] = joyY* .
- *Case 2:* el robot se mueve en el eje Z Por tanto *vector[4] = joyX.*
- *Case 3:* el robot gira sobre los ejes YZ. Por tanto, *vector[6] = joyX* y *vector[7] = joyY*
- *Case 4:* el robot gira sobre el eje X. Por tanto, *vector[5] = joyX.*

De esta forma al UR5e le llegan directamente el valor de la velocidad de cada eje, en función de los valores del joystick, en el que se debe mover según el modo seleccionado.

Posición 8

En esta posición se envía la información necesaria para manipular la herramienta. El valor de esta posición depende únicamente del botón C que al pulsarse (*botonC =true)* enviará un 1.

Posición 8	Función	Combinación de botones		Valor Vector [8]	Significado
	Cambiar estado de la herramienta	Ninguno		0	No cambia estado herramienta
		botonC		1	Sí cambia estado herramienta

Tabla 18.Posicion 8 del vector a enviar al UR5e desde el PC.

Además de este método que permite construir el vector, se han implementado otros como ***bateriaRestante()*** que devuelve el valor del nivel de batería del mando. También los métodos ***Vibrar()*** y ***NoVibrar()*** que permiten activar y desactivar la vibración del mando. Otro método implementado es el ***ConfiguracionLeds()*** que sirve para definir el estado de los 4 leds del mando, para ello se debe incluir como argumentos *true* para que esté encendido y *false* para que esté apagado.

```
public void ConfiguracionLeds(Boolean led1, Boolean led2, Boolean led3, Boolean led4 ) {
    wiimote.setLeds(led1,led2,led3,led4);
}
```

Figura 66. Método ConfiguracionLeds() de la clase NunchukMandoWii.

Todos estos se utilizan en los métodos implementados en la clase *ServidorTCP* y en *appGUI* por lo que en ambos programas se creará un objeto de la clase *NunchukMandoWii.*

SevidorTCP

Esta clase permite iniciar un servidor, para ello se utiliza la clase *ServerSocket* y *Socket* que dispone Java para realizar esta función. Esta clase dispone de los métodos necesarios para iniciar el servidor e intercambiar información con el UR5e por TCP/IP. El primer paso es definir las variables, en la siguiente tabla se presentan las variables utilizadas en este programa y las instancias a otras clases (objetos).

Variables / Objetos	Tipo	Función
serverSocket	ServerSocket	Socket que acepta conexiones TCP de forma pasiva. Es el socket de conexión o servidor
clientSocket	Socket	Socket que permite intercambiar datos con otro socket. El servidor al recibir la petición de un cliente crea un nuevo socket y lo conecta al remoto para poder intercambiar información.
port	int	Puerto al que va a escuchar el servidor
is	BufferReader	Reader que permite leer texto enviado desde el cliente a través de un buffer intermedio. Permite leer líneas enteras con el método String *readline()*
os	OutputStream	Canal para enviar información al cliente. Este escribe caracteres de uno en uno utilizando el método *write()*
datosRecibidos	String	Sting para guardar la cadena recibida desde el cliente.

escucharcliente	boolean	La aplicación Java estará escuchando los mensajes recibidos desde el UR5e mientras esta variable sea *true*
enviaracliente	boolean	La aplicación Java estará enviando mensajes al UR5e mientras esta variable sea *true*
textField	JTextField	Variable que permite escribir en el cuadro de texto de la interfaz de usuario.
led	JPanel	Variable que permite modificar el led presente en la interfaz de usuario
mandoWii	NunchukMandoWii	Objeto de la clase NunchukMandoWii para poder utilizar los métodos de esta clase.
controlArray	float[]	Vector donde se va a guardar el vector construido que se debe enviar al UR5e.

Tabla 19. Variables y objetos utilizados en la clase servidorTCP.

Esta clase será utilizada en el programa principal, donde se creará un objeto de esta y se utilizarán los diferentes métodos para realizar las funcionas para las que se ha diseñado esta aplicación. A continuación, se van a explicar los diferentes métodos que incorpora esta clase y la función de cada uno de ellos.

El primer método que se va a presentar es ***Iniciar_servidor(int port)*** a este se le pasa como argumento el puerto al que va a escuchar el servidor. El primer paso es crear el servidor y asociarle un puerto, este espera la conexión de un cliente. Para ello se utiliza la función:

serverSocket = new ServerSocket(port)

una vez se recibe la petición del cliente (UR5e) se crea un socket y se conecta al UR5e para intercambiar información:

clientSocket = serverSocket.accept()

Luego se definen los canales para intercambiar información, para recibir mensajes:

is = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()))

y para enviar:

os = clientSocket.getOutputStream()

Por tanto, este método sirve para iniciar el servidor y los diferentes canales que permiten la comunicación entre cliente y servidor.

El siguiente método es ***Desconectar()*** en esta se cierra el servidor, que deja de esperar la conexión de nuevos clientes:

serverSocket.close()

Se cierran los canales y el socket por lo que se cierra la comunicación y se deja de intercambiar datos:

is.close() os.close() clientSocket.close()

Y se pone a *false* el valor de la variable *escucharcliente* y *enviaracliente* lo que finaliza los hilos creados para enviar y recibir.

Otro de los métodos que ofrece esta clase es ***floatArrayToString(float [] datos)*** que se le pasa como argumento el vector construido y devuelve el vector convertido en String. Esto es un paso intermedio para poder convertir el vector a bytes que es el formato en el que se va a enviar al robot.

Para recibir los mensajes del UR5e se ha implementado el método ***Recibir(JTextField textField, JPanel led, NunchukMandoWii mandoWii)*** a este método se le pasa como argumentos el cuadro de texto de la interfaz para poder escribir sobre él, también el led de la interfaz para poder modificar su estado y el objeto mandoWii para poder utilizar los métodos de esta clase, ya que además de mostrar la información recibida desde el UR5e en la interfaz también se realizan algunas acciones sobre el mando en función del mensaje recibido. En este método se crea el hilo *RecibirInformacion* al que se le pasan los mismos argumentos y se inicia. Este hilo se encarga de recibir el mensaje del UR5e:

datosRecibidos = is.readLine()

y a partir del mensaje recibido se realizan las acciones correspondientes. Las diferentes acciones que se realizan en este hilo, en función del mensaje recibido son: modificar el estado del led que puede ponerse rojo o verde, mostrar el mensaje recibido en la interfaz (y si es el mensaje de parada de emergencia escribirlo en rojo), mostrar una pantalla emergente en la interfaz (cuando se ha superado la fuerza máxima permitida o se ha pulsado el botón de parada de emergencia), modificar el estado de los 4 leds del mando Wii™ y activar o desactivar la vibración del mando. Este hilo permanecerá en funcionamiento ejecutando estas acciones mientras la variable *escucharcliente = true.*

Por último, se ha implementado el método ***Enviar(NunchukMandoWii mandoWii)*** al que se le pasa como argumento el objeto mandoWii para poder utilizar los métodos de esta clase. En este método se crea y se inicia el hilo *EnviarInformacion* al que se le pasa el mismo argumento. Este hilo se encarga de obtener el vector a partir del estado de los botones y del joystick del mando y enviarlo al UR5e para que lleve a cabo las acciones correspondientes. Para ello guarda en la variable *controlArray* el vector mediante la siguiente instrucción:

controlArray = mandoWii.getControlArray()

a continuación, se envía el mensaje al robot, teniendo en cuenta que este se debe enviar como bytes. Para ello se realizan las siguientes instrucciones:

os.write(floatArrayToString(controlArray).getBytes("US-ASCII"))

os.flush()

a continuación, se espera 100 ms hasta volver a enviar la información. Este hilo permanecerá enviando el vector construido cada 100 ms mientras la variable *enviaracliente = true.*

appGUI

Esta es la clase principal, en ella está el método *main* por lo que es la encargada de realizar todas las acciones necesarias para el correcto funcionamiento de la aplicación diseñada. Además, en esta se define la interfaz de usuario.

En la siguiente tabla se muestra las variables y los objetos que se han utilizado en este programa:

Variable / objeto	Tipo	Función
frame	JFrame	El frame de la interfaz de usuario
Port	JTextField	Cuadro para introducir texto. En este se introduce el puerto al que debe escuchar el servidor
server	ServidorTCP	Objeto de la clase *ServidorTCP* que permite utilizar los métodos de esta
ComunicacionEstablecida	boolean	Variable que indica el estado de la comunicación con el cliente.
textField	JTextField	Cuadro de texto donde se mostrarán los mensajes recibidos desde el UR5e.
mandoWii	NunchukMandoWii	Objeto de la clase *NunchukMandoWii* que permite utilizar los métodos de esta
textMando	JTextField	Cuadro de texto donde se muestra el nivel de batería del mando
wiimotes	Wiimote	Permite saber si el wiimote y el nunchuk están conectados
lblPuerto	JLabel	Texto de la interfaz *"puerto"*
led	JPanel	Se utiliza como un led indicador del estado del robot.
imagenes	JLabel	Variable para mostrar la una imagen en la interfaz
btnComenzar	JButton	Botón de la interfaz que permite iniciar y finalizar la comunicación

barraBateria	JProgressBar	Barra que indica el nivel de batería del mando
btnMando	JButton	Botón que permite actualizar el valor de la batería del mando
lblControlRemotoCobot	JLabel	Título de la interfaz *"Control Remoto cobot UR5e – Mando Wii"*
lblConexion	JLabel	Texto interfaz *"Configuración de la conexión"*
separador	JSeparador	Barra separadora en interfaz
Separador_1	JSeparador	Barra separadora en interfaz
lblEstadoCobot	JLabel	Texto interfaz *"Estado Cobot:"*

Tabla 20.Variables y objetos utilizados en la clase appGUI.

En esta clase se definen todos los elementos que forman la interfaz de usuario, así como su disposición en la pantalla y las acciones asociadas a cada uno. En este apartado se explican las diferentes funciones que se realizan en este programa sin entrar en mucho detalle ni en el diseño ni en el funcionamiento de la interfaz de usuario que se presentará de manera detallada más adelante.

Para empezar, se hacer visible el frame y se asegura que ambos mandos estén conectados. Como se ha dicho se inicializa todos los elementos de la pantalla. El primer paso para controlar el UR5e mediante el mando de la Wii™ es iniciar el servidor, para ello se debe configurar el puerto que va a escuchar que se guarda en la variable *Port* (se ha establecido por defecto el valor 5000) y a continuación pulsar el botón *"Iniciar Servidor TCP".* Este botón sirve para establecer y para finalizar la conexión, cuál de las dos acciones debe llevar a cabo depende del valor de la variable *ComunicacionEstablecida.* En un primer momento como no se ha establecido la comunicación el valor de esta variable será *false* y por lo tanto se inicia el servidor utilizando el método correspondiente de la clase *ServidorTCP*:

server =new ServidorTCP ()

server.Iniciar_servidor(Integer.parseInt(Port.getText()))

Una vez se ha conectado un cliente se muestra cierta información en la interfaz, el nombre del botón cambia a *"Desconectar",* se inician los hilos de enviar y de recibir, que como se ha explicado son los encargados de enviarle el vector al UR5e y de recibir los mensajes que se envían desde este:

server.Enviar(mandoWii);

server.Recibir(textField, led,mandoWii)

Luego se cambia el valor de la variable *ComunicacionEstablecida* a *true* para indicar que se está comunicando con el robot.

Una vez iniciados los hilos se comenzará a enviar el vector al UR5e construido a partir de los valores del mando y a recibir los mensajes desde el UR5e, estos mensajes se muestran en el cuadro de texto asociado a la variable *textField.* Además, en función del mensaje recibido se modifica el estado del led (*led).*

Durante el funcionamiento se puede consultar el valor de la batería del mando al pulsar el botón *"Bateria Mando"* este valor se presentará en texto en el cuadro de texto asociado a la variable *textMando* y en la barra de proceso asociado a la variable *barraBateria.* Una vez pulsado el botón se cambia el texto a *"Actualizar valor"* pero la función sigue siendo la misma.

Si se desea finalizar la comunicación se debe pulsar el botón *"Desconectar"* que comprobará el valor de la variable *ComunicacionEstablecida*, que será *true* porque se ha establecido la comunicación, y por tanto finalizará la comunicación cerrando a la conexión establecida utilizando la siguiente instrucción:

sever.Desconectar()

en este momento se cambiará el valor de la variable *ComunicacionEstablecida* a *false,* se cambiará el texto del botón a *"Iniciar Servidor TCP"* y se mostrará en la interfaz información que indica que la comunicación ha terminado.

Como se ha visto en este programa se utilizan los métodos implementados en las dos clases vistas anteriormente (*NunchukMandoWii* y *ServidorTCP*), lo que le permite llevar a cabo la función para la que se ha diseñado esta aplicación, que es enviar un vector con información del estado de los botones y del joystick al UR5e para que este realice las acciones pertinentes. Y por otro lado recibir información sobre las acciones que está llevando a cabo el robot e informar al usuario mediante una interfaz.

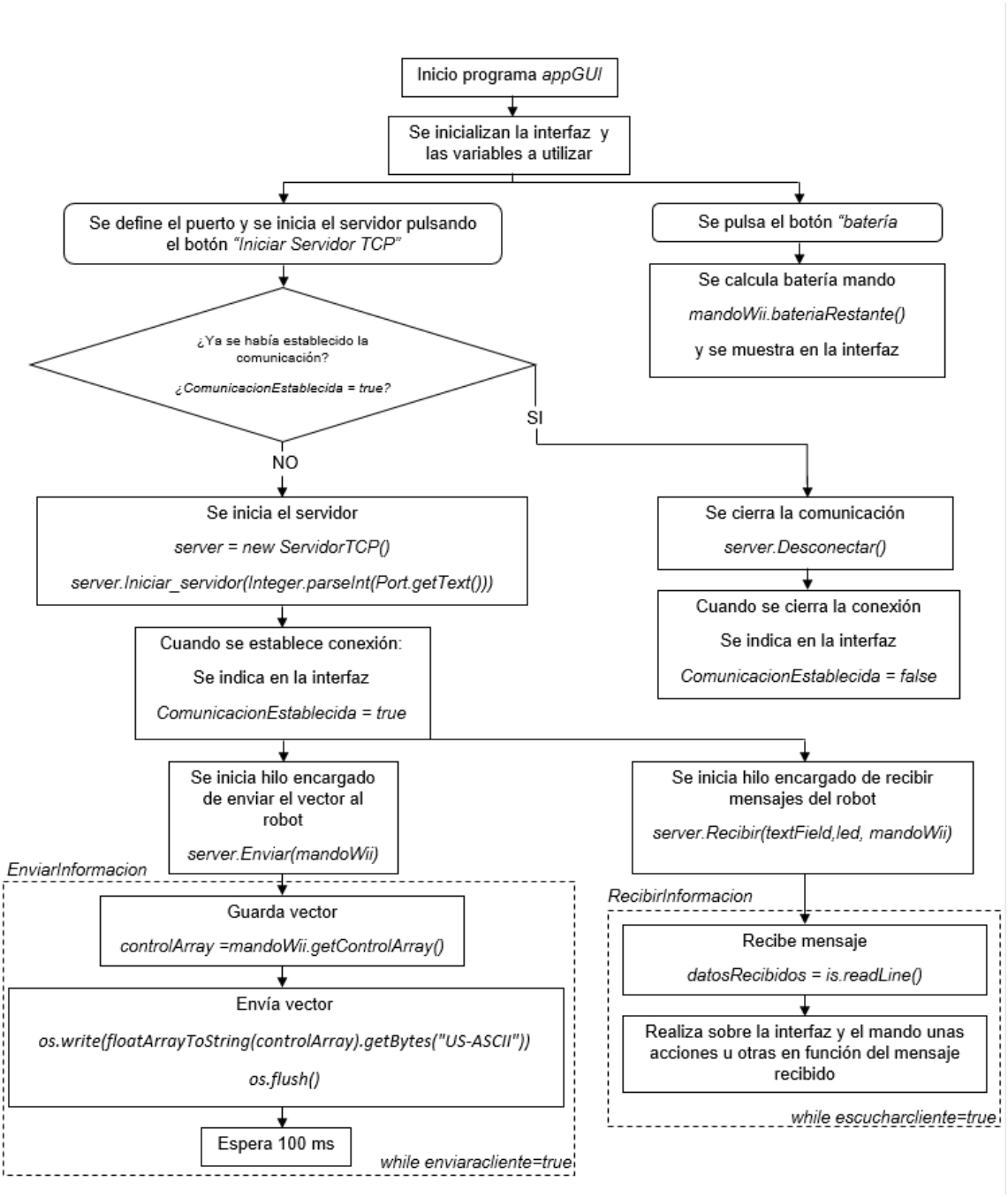


Figura 67.Diagrama de flujo del funcionamiento del programa del PC.

Programa Robot

El programa que se ejecuta en el robot en esta aplicación es muy similar al que se ejecuta en la aplicación del mando de Arduino, ya que el robot realiza las mismas funciones en ambas aplicaciones. Es por esto por lo que el programa es casi igual, salvo algunas pequeñas modificaciones relacionadas con el hecho de que ahora se envía la información al PC en lugar de mostrarla en el Log. Por lo tanto, en este apartado se presentarán las diferentes partes y sus funciones, sin entrar en mucho detalle ya que se explicaron anteriormente, y se hará hincapié en las diferencias entre ambos programas.

El programa presenta las mismas partes más un hilo añadido para la gestión de los mensajes enviados. Por tanto, se encuentran las siguientes partes: *BeforeStart, Robot Program, Thread1, Thread2, Thread3, Thread4, Thread5, Thread6* y *Thread7.*

BeforeStart

En este se declaran e inicializan las variables y se establece la comunicación con el servidor. Las variables utilizadas en este programa se presentaron en la Tabla 12. Cuando se ha establecido la conexión con el PC se envía un mensaje para iniciar la comunicación, para ello se utiliza la función:

socket_send_line("Robot Conectado")

esta función envía el mensaje terminado en "\n" que es el formato que se debe recibir en la aplicación Java del PC. Una vez establecida la comunicación y enviado el mensaje se espera 1 segundo antes de comenzar a ejecutarse el programa principal y los diferentes hilos, esto es porque el mensaje enviado se muestra en la interfaz durante este tiempo (si no se realiza esta espera el mensaje no se muestra porque el robot envía otro mensaje al iniciar el funcionamiento normal).

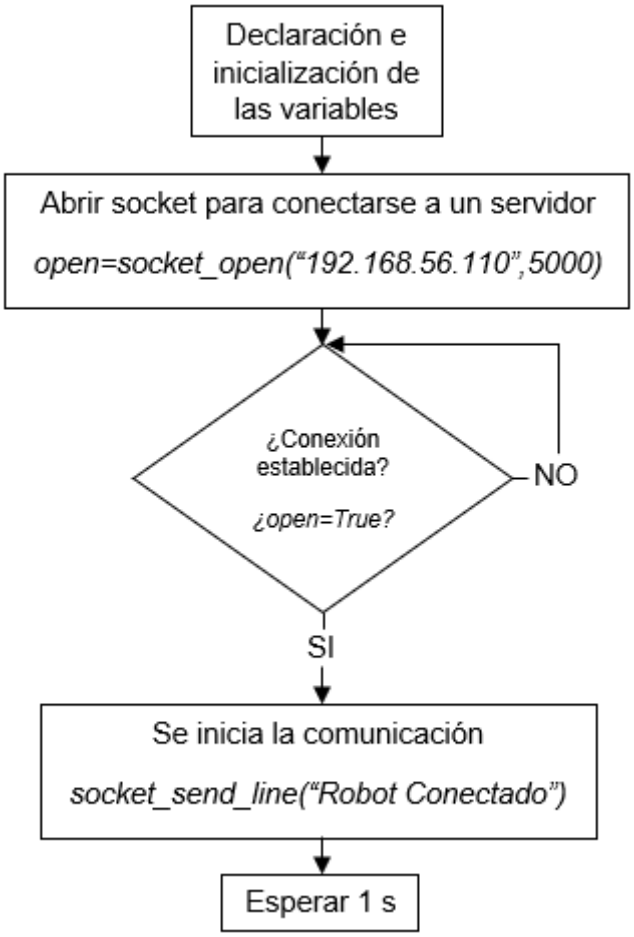


Figura 68.Diagrama de flujo de la parte BeforeStart del programa del UR5e en aplicación mando Wii™.

Thread1

Este hilo es el encargado de recibir el vector desde el PC y guardar la información de cada posición del vector en la variable correspondiente. El funcionamiento es el mismo que el del *Thread1* de la aplicación del mando de Arduino, ya que se recibe un vector de floats de 9 posiciones por lo que se utiliza la misma función para recibirlo. En este caso se hacen las siguientes igualaciones:

- *grabar := datos[1],* gestión de la grabación y reproducción de trayectorias.
- *modo := datos[2],* modo de funcionamiento.
- *x := datos[3],* velocidad desplazamiento eje X.
- *y := datos[4],* velocidad de desplazamiento eje Y.
- *z := datos[5]*, velocidad de desplazamiento eje Z.
- *rx := datos[6],* velocidad de giro eje X.

- *ry :=datos[7]*, velocidad de giro eje Y.
- *rz := datos[8]*, velocidad de giro eje Z.

En este caso todos los valores de velocidad se dividen entre 8.5 para transformar el ±0.85 recibido en ±0.1.

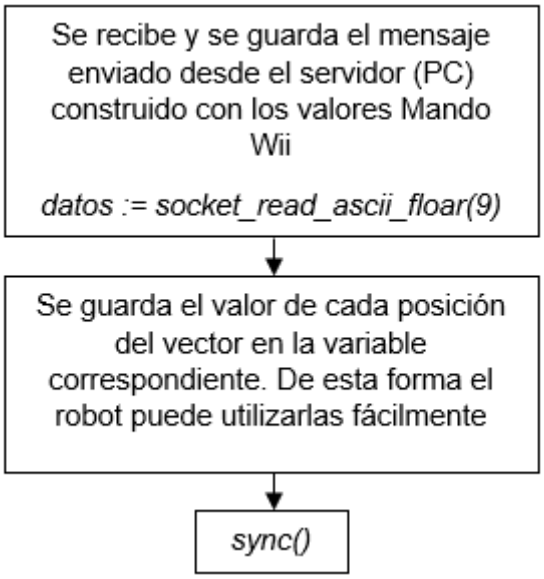


Figura 69.Diagrama de flujo de la parte Thread1 del programa del UR5e en la aplicación Mando Wii™.

Robot Program (o programa principal)

En este programa se gestiona el movimiento del robot en función de los valores recibidos desde el mando, parar el robot y realizar los dos tipos de trayectorias disponibles. El funcionamiento es exactamente el mismo que el de *Robot Program* de la aplicación del mando Arduino, con la única diferencia que el lugar de mostrar un mensaje en el Log cuando se iniciaba alguna de las dos trayectorias ahora se envía al PC. Los mensajes enviados son *"Realizando Trayectoria 1"* y *"Realizando Trayectoria 2"* y para enviarlos se utiliza la función:

socket_send_line("Realizando Trayectoria 1")

Thread2

Este hilo se encarga de realizar la grabación de la trayectoria 1. Su funcionamiento es igual que el del *Thread2* de la aplicación del mando Arduino.

Thread7

Este hilo se encarga de realizar la grabación de la trayectoria 2. Su funcionamiento es igual que el del *Thread5* de la aplicación del mando Arduino.

Thread4

En este hilo se define el estado de las variables que gestionan el inicio y el final de la grabación de las dos trayectorias y el inicio de la realización de estas. Su funcionamiento es igual que el del *Thread4* de la aplicación del mando Arduino, con la diferencia de que en lugar de enviar los mensajes al Log ahora se envían al PC y que en esta aplicación solo se envía mensaje al iniciar la grabación y no al finalizarla. Los mensajes que se envían son *"Grabando Trayectoria 1"* y *"Grabando Trayectoria 2"*, para ello se utiliza la función:

socket_send_line ("Grabando Trayectoria 1")

Thread3

En este hilo se define el estado de las variables que gestionan las funciones de seguridad que se han incluido en la aplicación que son la parada de emergencia y la fuerza máxima permitida. También se gestiona las acciones que se deben llevar a cabo para que el robot vuelva a funcionar con normalidad de una forma segura después de que se haya producido uno de estos problemas. El funcionamiento es igual que el del *Thread3* de la aplicación Mando Arduino, teniendo en cuenta que ahora el botón de parada de emergencia hace *modo = 6* y que el botón home hace *modo = 5.* Además, se envían los mensajes al PC, en lugar de al Log, mediante la función *socket_send_line().* Cuando se supera la fuerza de seguridad se envía *"Fuerza de Seguridad Superada"*, cuando se pulsa el botón de parada de emergencia se envía *"PARADA DE EMERGENCIA"* y cuando se pulsa el botón home se envía *"Valores Restablecidos"* y se espera 1 segundo, este es el tiempo que se muestra este mensaje en la interfaz.

Thread6

En este hilo se gestiona el estado de la herramienta, se ha utilizado una garra. Su funcionamiento es idéntico al del *Thread6* de la aplicación Mando Arduino.

Thread5

Este hilo se encarga de enviar un mensaje al PC cuando el robot se encuentra moviéndose libremente sin realizar ninguna acción. Este hilo es necesario para gestionar de forma correcta los mensajes que se muestran en la pantalla. Si no estuviera este hilo y, por el ejemplo, el robot estuviera haciendo la trayectoria 1, por la interfaz se muestra el mensaje *"Realizando Trayectoria 1"* y al terminar de

realizarla se continuaría mostrando este mensaje, ya que no se envía otro mensaje hasta que no se realice otra acción, por lo que la interfaz mostraría información errónea. Para solucionar esto, este hilo se encarga de que cuando el robot no está grabando ninguna trayectoria (*grabarbool1 = False* y *grabarbool2 = False)*, ni realizando ninguna trayectoria grabada (*hacertray1 = False* y *hacertray2 = False)*, ni se ha superado la fuerza máxima de seguridad (*fuerzaparar = False)* y tampoco se ha pulsado el botón de parada de emergencia (*P_Emergencia = False)* se envía al PC el mensaje *"Escuchando Ordenes"* esto se repite cada 0.5 segundos.

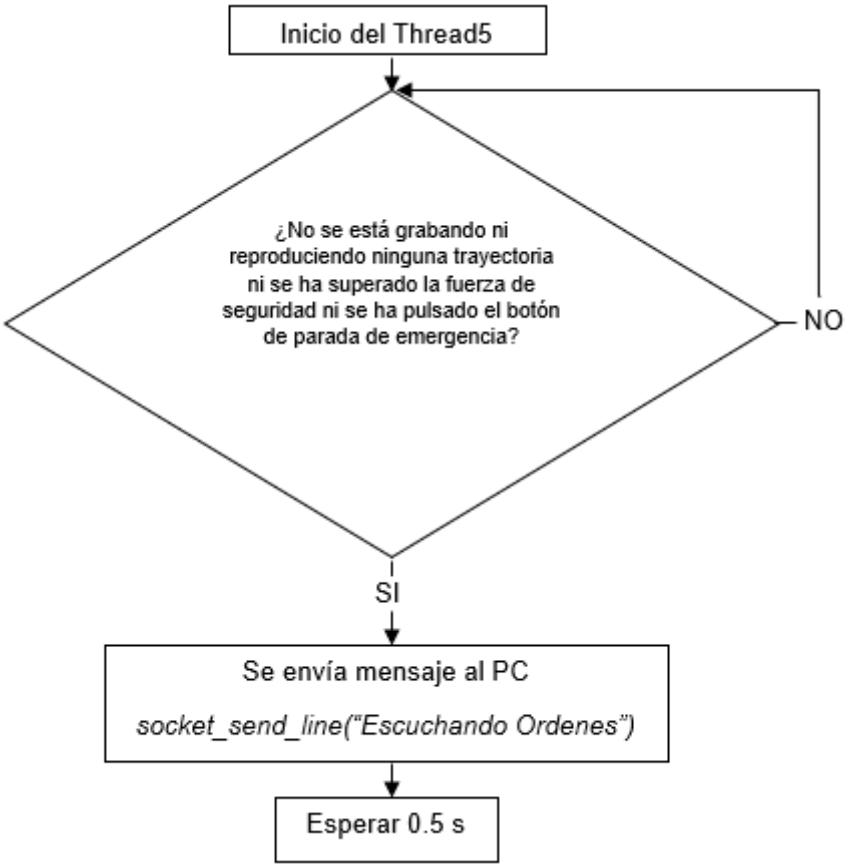


Figura 70.Diagrama de flujo de la parte Thread5 del programa del UR5e en la aplicación Mando Wii™.

3.3.3 Interfaz de usuario.

Para facilitar el uso de la aplicación se ha diseñado una interfaz de usuario cuya función es facilitar a este la interacción con el robot. La interfaz se ha diseñado de forma que sea fácil e intuitiva.

Desde la interfaz diseñada el usuario puede establecer la comunicación con el robot de una manera sencilla y ver la información sobre el estado y la actividad que está realizando el robot. También se muestra el estado de la batería del mando Wii™. En la figura 68 se muestra el aspecto de la interfaz diseñada.

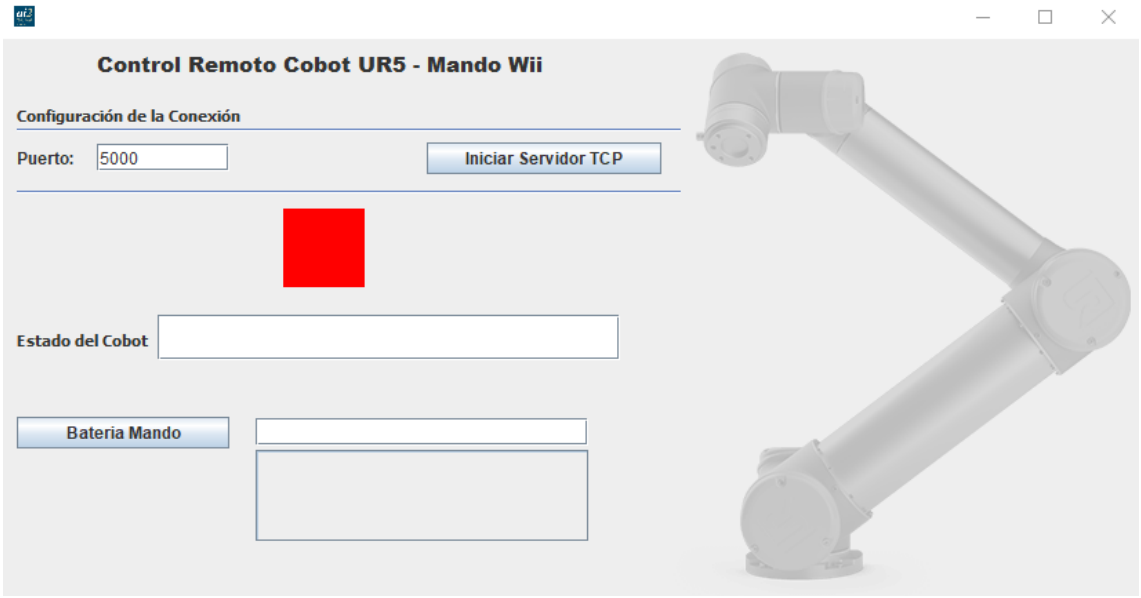


Figura 71. Interfaz de usuario.

Como se puede observar la interfaz dispone de un apartado para configurar la conexión, un led para indicar el estado del robot, una imagen del robot que indica si hay comunicación con el robot, un cuadro de texto donde se muestra las acciones que está realizando el robot y un apartado para consultar la batería del mando Wii™.

Cuando se inicia el programa la interfaz tiene el aspecto mostrado en la Figura 68. Para establecer la comunicación se configura el puerto deseado y se inicia el servidor pulsando el botón *"Iniciar Servidor TCP"* en este momento se está esperando la conexión del robot. Cuando el robot se conecta se muestra el mensaje en la interfaz, el led se pone verde para indicar que el robot puede realizar las acciones para las que ha sido programado, la imagen del robot se vuelve a color para indicar que el robot está conectado y el texto del botón cambia a *"Desconectar"*.

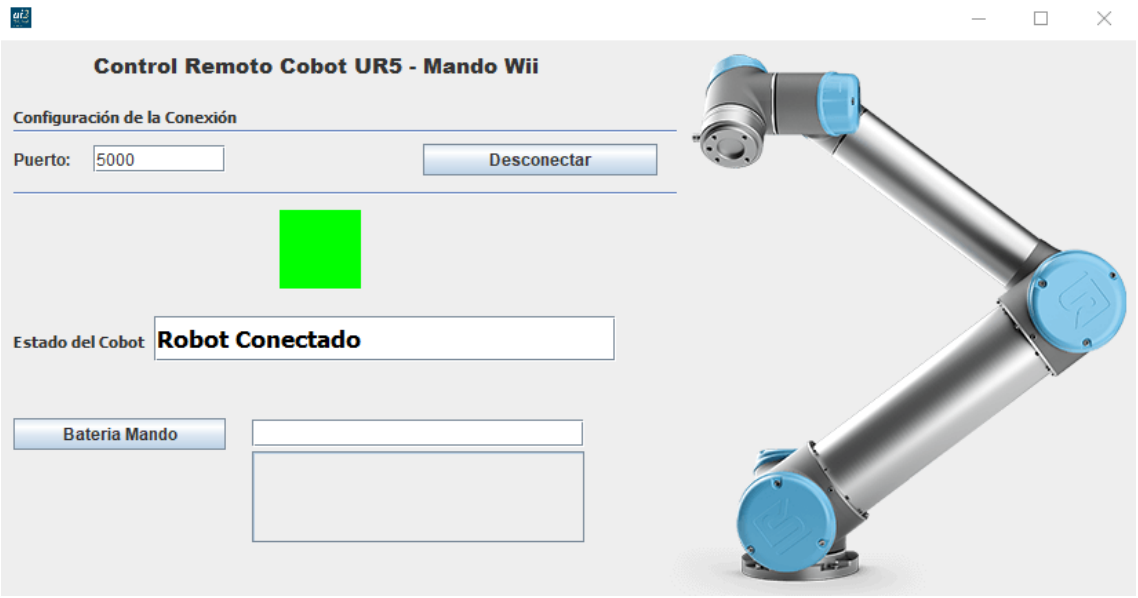


Figura 72. Interfaz con robot conectado.

Este mensaje se muestra durante 1 segundo y a continuación el robot comienza a escuchar las órdenes recibidas desde el mando.

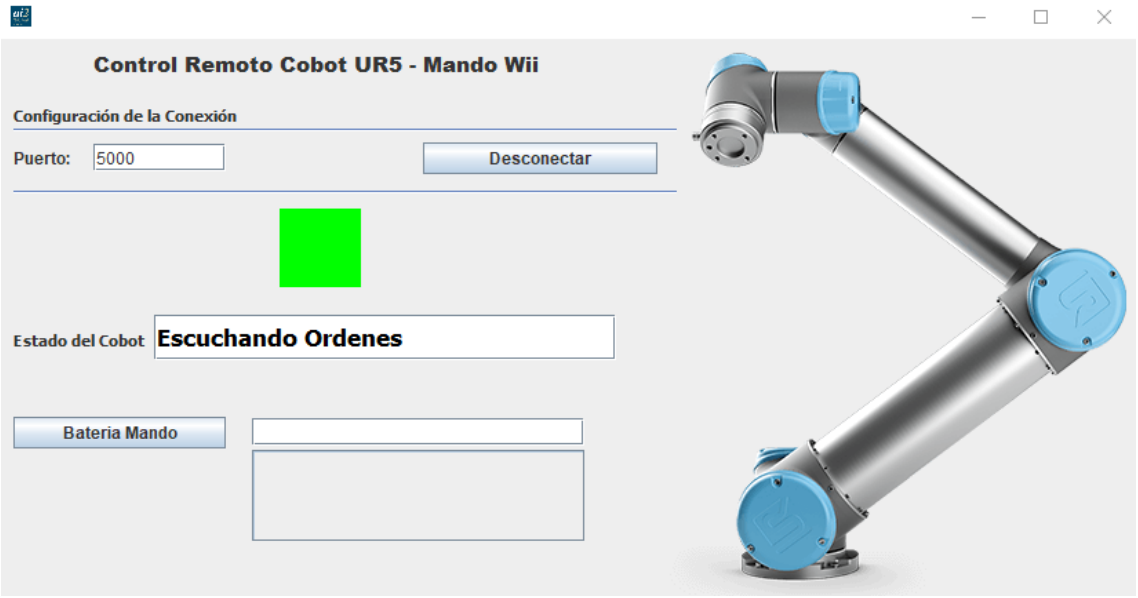


Figura 73. Interfaz robot esperando órdenes del mando Wii™.

En este momento el robot se puede mover libremente o realizar las acciones para las que ha sido programado, grabación y realización de trayectorias. Cuando el robot está realizando alguna de estas acciones lo muestra por pantalla.

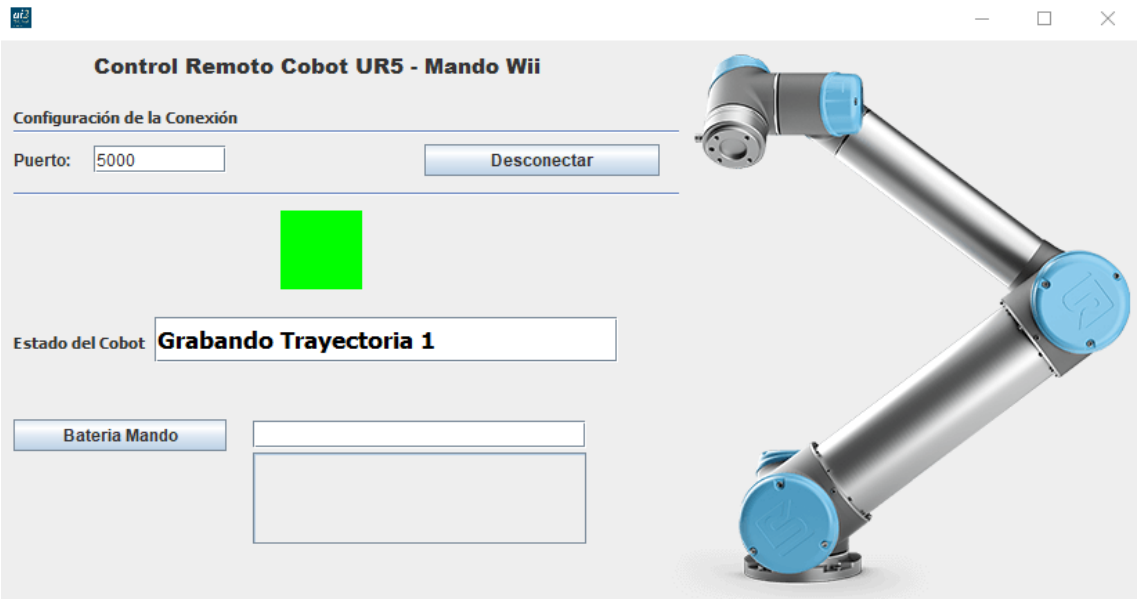


Figura 74. Interfaz robot grabando la trayectoria 1.

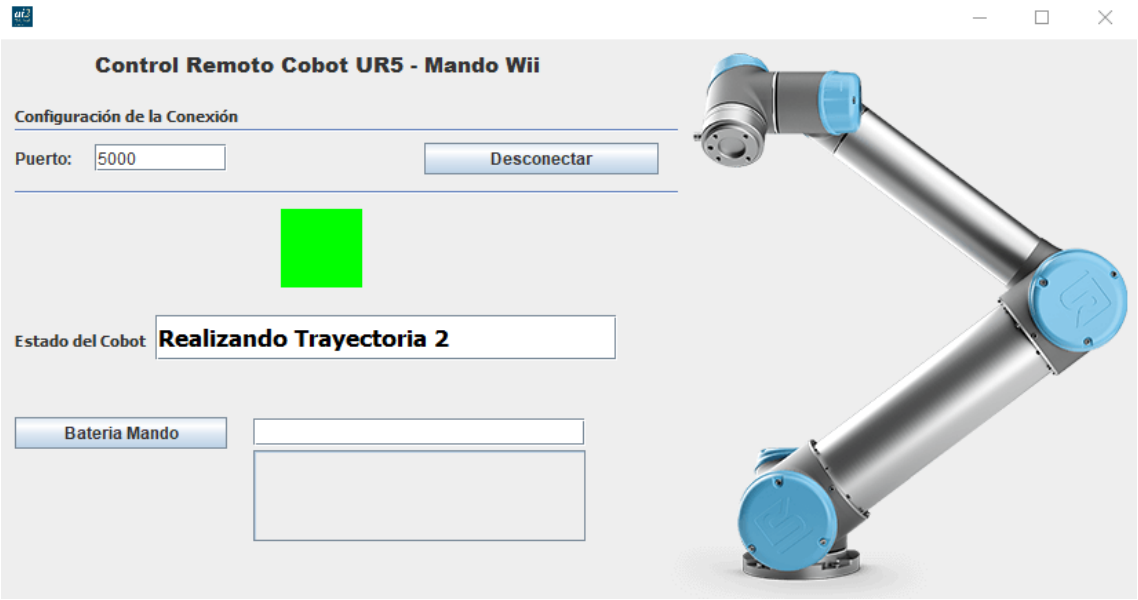


Figura 75. interfaz robot realizando la trayectoria 2.

Cuando se supera la fuerza máxima permitida el robot no puede moverse ni grabar ni realizar trayectorias por lo que el led se pone de color rojo para indicarlo. Además, se muestra el mensaje por pantalla y sale una pestaña emergente con un mensaje de advertencia.

Figura 76. Interfaz cuando se ha superado la fuerza máxima permitida.

Por otro lado, cuando se ha pulsado el botón de parada de emergencia, al igual que cuando se supera la fuerza, el robot pasa a un estado de bloqueo indicado por el led. En este caso también se muestra por pantalla el mensaje, con color rojo para llamar la atención, y se muestra un mensaje de error en una pestaña emergente.

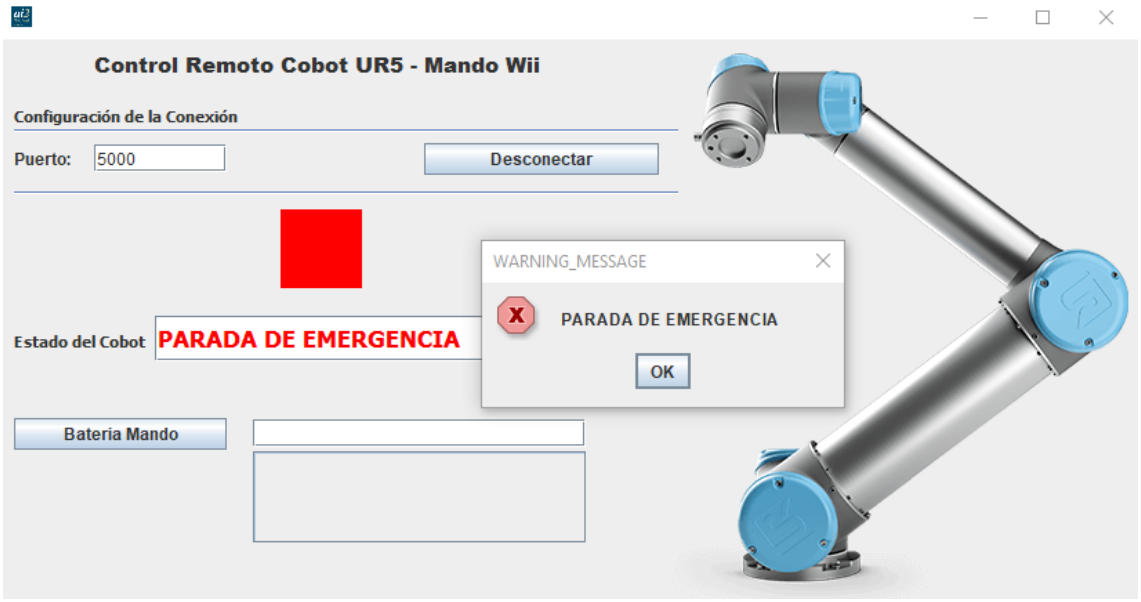


Figura 77. Interfaz cuando se ha pulsado el botón de parada de emergencia.

Para sacar al robot de esta situación de bloqueo se debe cerrar la pestaña emergente y pulsar el botón home. El led se volverá a poner verde ya que el robot volverá a su estado normal y se muestra un mensaje por pantalla durante un segundo, después de este tiempo se volverá a escuchar órdenes y la interfaz quedará como en la Figura 70.

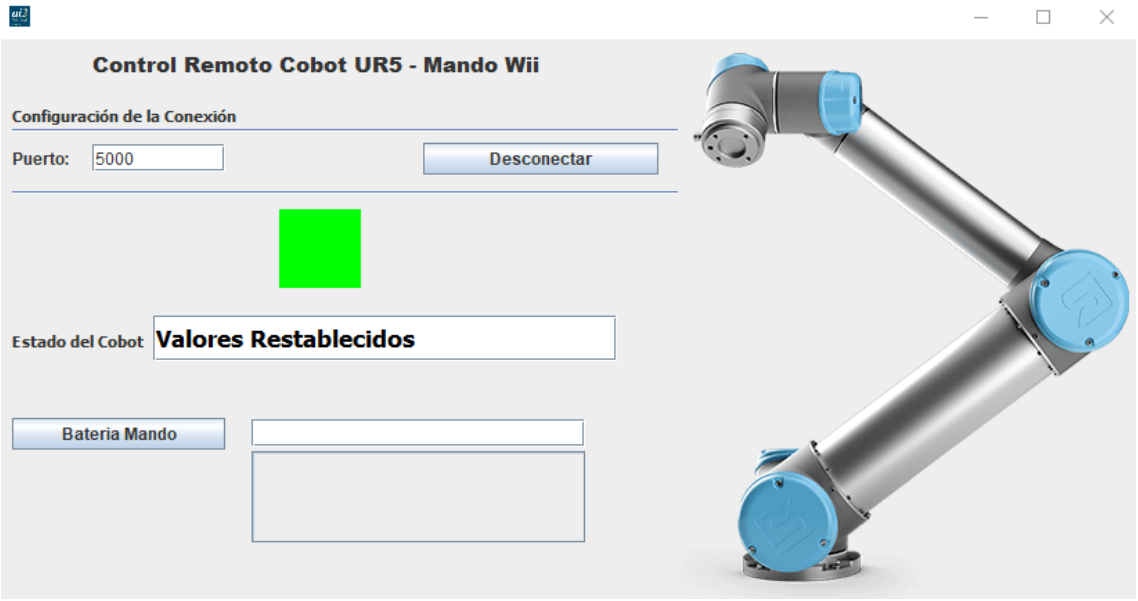


Figura 78. Robot al pulsar botón home.

Además de obtener información sobre el estado de la conexión y sobre el estado del robot y de las acciones que se está llevando a cabo, se puede consultar el nivel de batería del mando. Al pulsar el botón *"Batería Mando"* la primera vez y luego *"Actualizar valor"* se muestra el valor de la batería restante. Cuando el nivel de batería es inferior a 10% se muestra en color rojo.

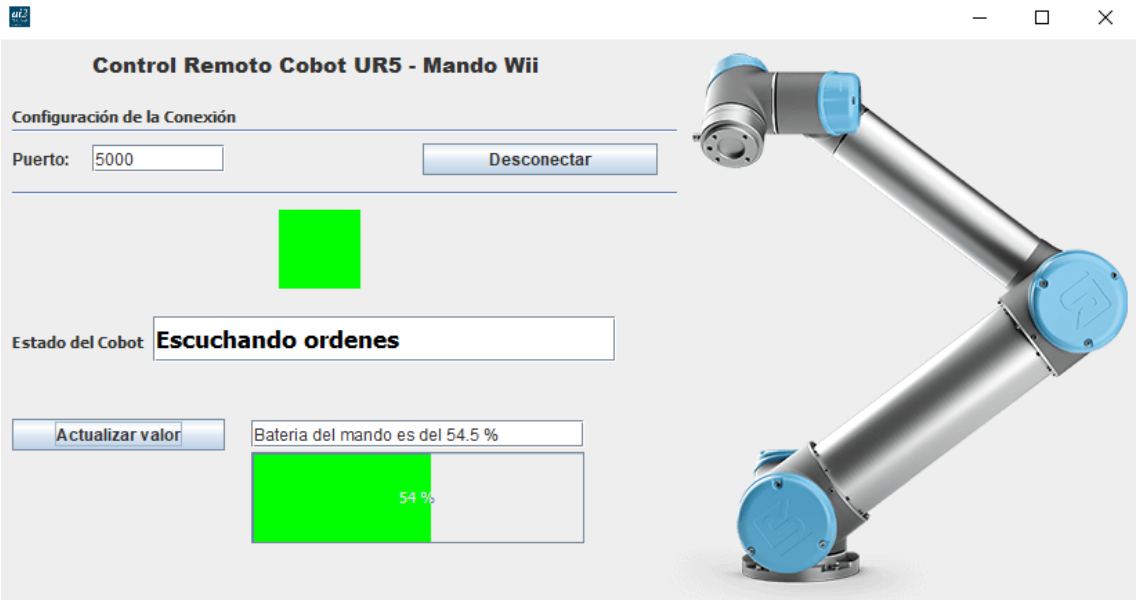


Figura 79. Interfaz mostrando batería del mando Wii™.

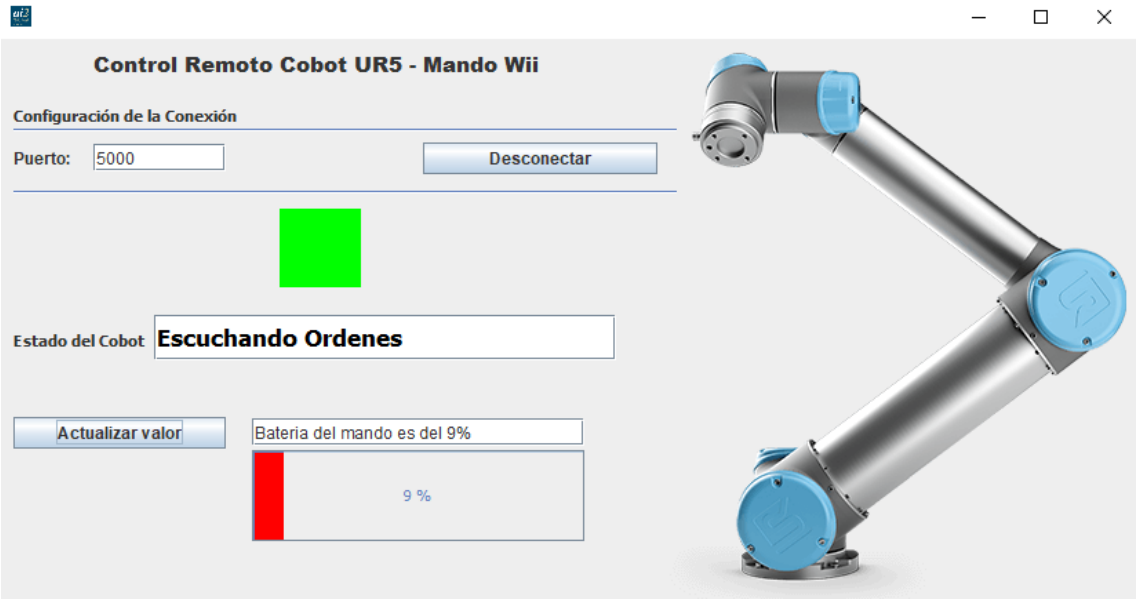


Figura 80. Interfaz mostrando batería baja mando Wii™.

Cuando se finaliza la conexión la imagen del robot vuelve a blanco y negro para indicar que no hay comunicación con el robot, el led se vuelve rojo y cambia el texto del botón a *"Iniciar Servidor TCP"* por lo que la interfaz vuelve a tener el aspecto de la Figura 68.

Además de esta información mostrada por la pantalla del PC, también se ofrece información sobre el estado del robot a través del mando Wii™. Para indicar las diferentes acciones que está llevando a cabo se utilizan los 4 leds disponibles en el wiimote. Cuando el robot se conecta y comienza a escuchar órdenes se encienden los leds 2 y 3 del mando. Cuando se está realizando la grabación de la trayectoria 1 se enciende el led 1 y cuando se graba la trayectoria 2 el led 2. Por su parte, cuando se realiza la trayectoria 1 se encienden los leds 1 y 4, y cuando es la trayectoria 2 la que se está realizando se encienden los leds 2 y 4. Cuando en el robot se supera la fuerza de seguridad se indica en el mando encendiendo los leds 1,2 y 3. Por último, cuando se pulsa el botón de parada de emergencia se encienden los 4 leds y además en este caso se activa la vibración del mando, que no cesará hasta que se cierre la pestaña con el mensaje de error mostrada en la interfaz. Además de indicar las acciones que está llevando a cabo el robot, el mando también indica que su nivel de batería es inferior al 10%, ya que cuando esto ocurre vibra. En la siguiente tabla se presentan los diferentes mensajes que se reciben desde el robot, y por tanto se muestran en la interfaz, y la información que muestra el mando en cada caso.

Mensaje enviado desde el UR5e	Estado del mando
Robot Conectado	Encendidos leds 2 y 3
Grabando Trayectoria 1	Encendido led 1
Grabando Trayectoria 2	Encendido led 2
Realizando Trayectoria 1	Encendidos leds 1 y 4
Realizando Trayectoria 2	Encendidos leds 2 y 4

Valores Restablecidos	Encendidos leds 2 y 3
Fuerza de seguridad superada	Encendidos leds 1,2 y 3
PARADA DE EMERGENCIA	Encendidos leds 1,2,3 y 4

Tabla 21. Mensajes enviados desde UR5e e indicaciones del mando Wii™.

3.3.4 Comparación de las aplicaciones desarrolladas.

Ambas aplicaciones, mando Arduino y mando Wii™, se han diseñado para poder llevar a cabo las mismas tareas. Sin embargo, existen algunas diferencias entre las dos soluciones desarrolladas, en este apartado se presentarán estas diferencias y las ventajas y desventajas que presentan la una sobre la otra.

Una de las ventajas de la aplicación mando Wii™ es que el programa se ejecuta en un PC por lo que es una plataforma mucho más potente que la placa de Arduino. El programa en Java del PC permite una programación concurrente, lo que facilita el intercambio de mensajes entre el mando y el UR5e, ya que se crea un hilo para enviar, que enviará la información con la frecuencia deseada, y otro para recibir por lo que no importa que las funciones sean bloqueantes. Este intercambio de mensajes permite disponer de mucha más información. Además, permite diseñar una interfaz donde mostrarla de manera clara. Sin embargo, la placa de Arduino no permite la programación concurrente, y por lo tanto si se espera a recibir mensajes desde el UR5e, al ser las llamadas bloqueantes, puede que no se envíe el vector al robot con la frecuencia deseada. Por esto, en la aplicación Mando Wii™ durante el funcionamiento se muestra información del estado de la conexión y de las acciones que se están llevando a cabo tanto en la interfaz como en el mando, y en la aplicación Mando Arduino solo se muestra la información en la consola de programación.

Sin embargo, esto que se ha presentado como una ventaja se puede considerar como una desventaja, ya que la aplicación Mando Wii™ necesita un PC como dispositivo intermedio, mientras que el Mando Arduino no necesita dispositivo intermedio ya que se comunica con el UR5e directamente. Esto es una gran ventaja ya que solo se necesita el mando para controlar el UR5e. Además, la conexión es mucho más sencilla, ya que para el mando de Arduino solo se debe conectar el cable ethernet a la red y alimentar la placa, cuando se establezca la comunicación se mostrará en el Log y ya se puede manipular el UR5e. En cambio, para conectar el mando Wii™ se debe conectar el PC a la red, conectar el mando al PC por bluetooth y ejecutar la aplicación en la cual se configura el puerto y se inicia la comunicación (los procesos para conectarse se explican con más detalle en el anexo I).

Por otro lado, el mando Arduino se ha diseñado específicamente para controlar el UR5e, por lo que se ha diseñado el mando con los botones necesarios y en una disposición adecuada que permita un manejo sencillo. Por ejemplo, tal y como se ha montado no es necesario pulsar más de dos botones de forma simultánea, ya que con dos botones se puede realizar todas las acciones. En cambio, el mando Wii™ no se ha diseñado específicamente para esta aplicación, por lo que hay que amoldarse a los botones que dispone y a su disposición. Por

eso en el mando Wii™ se ha configurado la selección de modo solo con 3 botones (en el mando Arduino con 4), esto se ha hecho pensando en la manejabilidad del mando.

4. Validación de las aplicaciones.

Para validar el funcionamiento de las aplicaciones diseñadas se han realizado una serie de pruebas para comprobar que el robot es capaz de realizar todas las tareas para las que ha sido programado.

4.2 Validación de la aplicación Mando Arduino.

Para validar la aplicación Mando Arduino se ha construido el mando y se ha comprobado que las conexiones se han realizado correctamente. Para realizar esta comprobación se ha verificado que los diferentes dispositivos, que forman el mando, están conectados al pin correcto de la placa Arduino, ya que en caso contrario el comportamiento del mando no sería el esperado. Y además, se ha analizado el mensaje enviado desde el mando en función del estado de sus botones y joystick.

Una vez comprobado el funcionamiento del mando se debe conectar el UR5e y el Mando Arduino a la misma red y verificar que se establece la conexión de forma correcta.

Finalmente se han ejecutados los programas del robot y del mando para comprobar el funcionamiento de la aplicación. Para ello se han realizado los diferentes tipos de movimiento que puede realizar el robot en función del modo seleccionado desde el mando, se ha realizado la grabación de los dos tipos de trayectorias y se han reproducido las trayectorias grabadas. Además, Se ha probado las funciones de seguridad añadidas, parada de emergencia y fuerza de seguridad máxima permitida, en diferentes situaciones como con el robot escuchando órdenes y grabando o realizando trayectorias. En todos los casos se han obtenido la respuesta esperada por parte del robot. También se ha comprobado que el botón Home devuelve el robot a su estado normal de una forma segura. Por otra parte, se ha verificado que la información mostrada en la pestaña del Log de la consola de programación se corresponde con las acciones que está llevando a cabo el robot, y por tanto, los mensajes mostrados son correctos.

Una vez realizadas todas estas comprobaciones se puede concluir que la aplicación diseñada funciona correctamente, ya que con ella se logra alcanzar los objetivos planteados al principio de este trabajo.

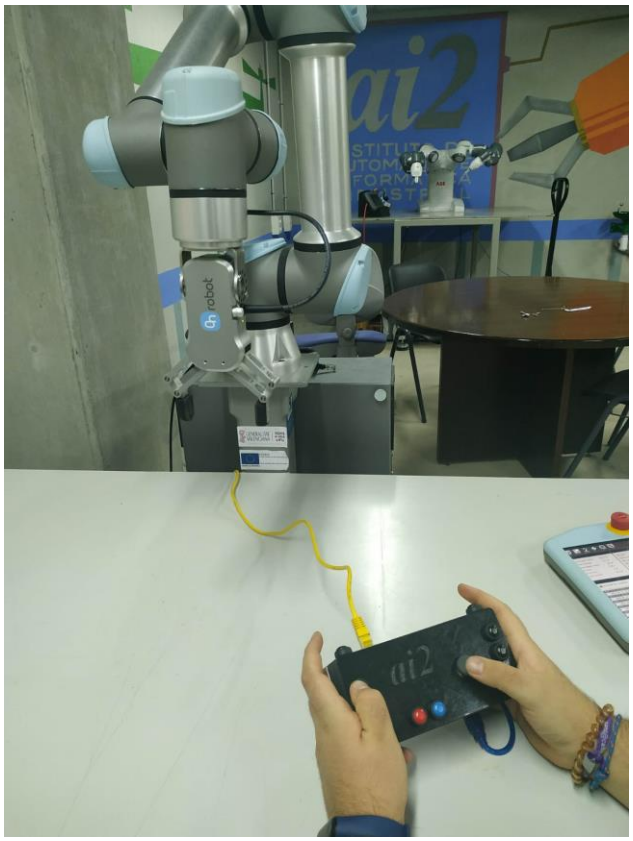


Figura 81. Mando Arduino y UR5e.

4.3 Validación de la aplicación Mando Wii™.

Para validar la aplicación Mando Wii™ se ha comprobado que la conexión entre el PC y el mando Wii™ se ha establecido correctamente. Luego se debe conectar el PC y el UR5e a la misma red.

Una vez las conexiones se han realizado correctamente se han ejecutado los programas del PC y del UR5e y se realizaron las mismas pruebas de funcionamiento realizadas para la aplicación mando Arduino. Los resultados obtenidos fueron satisfactorios, ya que, el robot realiza los movimientos y giros en los diferentes ejes en función del estado del mando, se graban y se reproducen los dos tipos de trayectorias de forma correcta y el comportamiento del robot frente a que se supere la fuerza máxima de seguridad permitida, a que se pulse el botón de parada de emergencia o a que se pulse el botón Home es el esperado. Por otra parte, se ha comprobado que la información mostrada en la interfaz de usuario diseñada y en el mando Wii se corresponde con el estado del robot, y por tanto, los mensajes enviados desde el UR5e al PC son correctos.

Tras la realización de todas estas comprobaciones se concluye que la aplicación diseñada funciona correctamente, ya que con ella se logra alcanzar los objetivos planteados al principio de este trabajo.

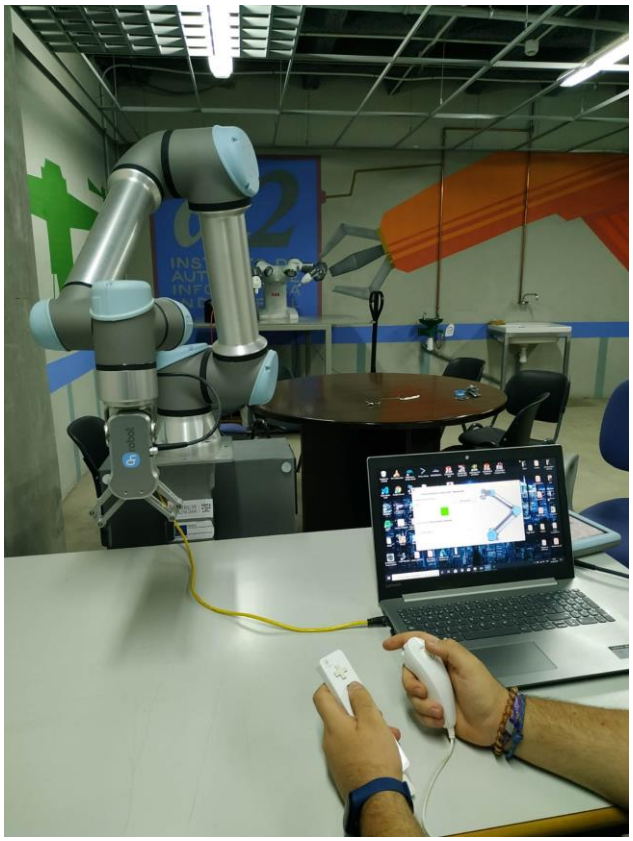


Figura 82. Mando Wii™ y UR5e.

5. Presupuesto.

En el presupuesto se muestran los gastos de los materiales, licencias de programas y mano de obra necesarios para llevar a cabo el proyecto. Como el proyecto no tiene un fin empresarial no se tendrá en cuenta los impuestos (en el precio de los materiales adquiridos se incluye el IVA) ni los beneficios.

En las horas de mano de obra se incluyen todas las horas dedicadas a la realización del trabajo. Estas se dividen en varias tareas:

- Familiarizarse con el UR5e para conocer las herramientas de las que dispone para la realización del proyecto. Y centrándose en aprender las posibilidades que ofrece la programación en PolyScope y los comandos URScripts.

- Estudio de las diferentes posibilidades para desarrollar el proyecto, obtener información sobre las diferentes alternativas que permiten llevar a cabo la realización del proyecto como: diferentes alternativas para guiar el UR5e mediante un dispositivo externo, comunicación TCP/IP con Arduino, comunicación del mando de la Wii™ con una aplicación java, etc.

- Programación y pruebas realizadas para la implementación de los algoritmos que permitan lograr un correcto funcionamiento de las aplicaciones.

- Diseño y construcción del mando Arduino. El diseño se ha realizado mediante *SolidWorks* software que no se había utilizado antes por lo que el tiempo de diseño es más elevado del necesario ya que hubo que aprender a utilizar esta plataforma.

Para realizar el presupuesto se va a separar las dos aplicaciones diseñadas, pero existen tareas de investigación que se realizaron previamente y se muestran en la siguiente tabla.

Concepto	Cantidad (horas)	Precio unitario (€/horas)	Precio (€)	Total (€)
Pruebas con UR5e	30	20	600	600
Estudio y pruebas de las diferentes alternativas	50	20	1000	2000
TOTAL	80 horas			1600 €

Tabla 22. Tareas de investigación inicial.

Mando Arduino

Para el presupuesto del mando Arduino se van a mostrar diferentes tablas con los gastos relativos a la mano de obra en la realización de diversas tareas, con los gastos en el hardware necesario y finalmente se mostrará el presupuesto total. En la siguiente tabla se muestra las diferentes tareas realizadas, las horas de dedicación y el precio:

Concepto	Cantidad (horas)	Precio unitario (€/horas)	Precio (€)	Total (€)
Investigación comunicación TCP/IP	10	20	200	200
Programación y pruebas	120	20	2400	2600
Diseño y construcción del mando	45	20	900	3500
TOTAL	175 horas			3500 €

Tabla 23. Tareas realizadas para la aplicación Mando Arduino.

En la siguiente tabla se muestra el hardware necesario y su precio:

Producto	Precio unitario (€)	Cantidad	Total (€)
Botones	1.1	9	9.9
Joystick	1.62	1	11.52
Resistencias	2.80	1	14.32
Arduino MEGA	19.93	1	34.25
Módulo con W5100	6.76	1	41.01
Cableado	3.45	1	44.46
Batería externa (5000 mah)	10.45	1	54.91
Placa preperforada	2.58	1	57.49
Pines	0.98	1	58.47
Cable ethernet	7	1	65.47
Impresión 3D (pieza 156 g)	4.37	1	69.84
TOTAL			69.84 €

Tabla 24. Hardware necesario construcción del mando Arduino.

Para terminar, se va a mostrar una tabla con el presupuesto total para el desarrollo de la aplicación Mando Arduino.

APLICACIÓN MANDO ARDUINO			
Concepto	Cantidad	Precio	TOTAL
Hardware para mando	1	69.84	69.84 €
Licencia SolidWorks (3 meses)	1	1010 €	1079.84 €
Mano de obra	255	5100 €	6179.84 €
TOTAL			6179.84 €

Tabla 25. Presupuesto aplicación Mando Arduino.

Este es el presupuesto total para la creación de un primer ejemplar. Para realizar más ejemplares el precio se reduciría considerablemente, debido a que las horas de mano de obra necesarias serían mucho menor. Esta reducción se debe a que se eliminan las horas de investigación, de diseño e implementación de algoritmos y diseño del mando, por lo que tampoco es necesario la licencia de *SolidWorks*. Por lo que el presupuesto queda reducido al hardware y la mano de obra

necesaria para la construcción del mando y puesta en marcha de la aplicación. El precio de una unidad podría ser de unos 130 € (69.84 € del hardware + 60 € por 3h de mano de obra).

Mando Wii™

Para la aplicación mando Wii™ se van a mostrar en diferentes tablas el presupuesto relativo a la mano de obra, al hardware necesario y finalmente el presupuesto final.

En la siguiente tabla se muestra las diferentes tareas realizadas, las horas de dedicación y el precio:

Concepto	Cantidad (horas)	Precio unitario (€/horas)	Precio (€)	Total (€)
Investigación comunicación	15	20	300	300
Programación y pruebas	100	20	2000	2300
TOTAL	115 horas			2300 €

Tabla 26.Tareas realizadas para la aplicación Mando Wii™.

En la siguiente tabla se muestra el hardware necesario y su precio:

Producto	Precio unitario (€)	Cantidad	Total (€)
wiimote	49.99	1	49.99
nunchuk	24.95	1	74.94
Portátil Lenovo ideapad330	599.99	1	674.93
Cable ethernet	7	1	681.93
TOTAL			681.93 €

Tabla 27.Hardware necesario construcción del mando Wii™.

Para terminar, se va a mostrar una tabla con el presupuesto total para el desarrollo de la aplicación Mando Wii™.

APLICACIÓN MANDO WII™			
Concepto	Cantidad	Precio	TOTAL
Hardware	1	681.93 €	681.93 €
Mano de obra	195	3900 €	4581.93 €
TOTAL			4581.93 €

Tabla 28.Presupuesto aplicación Mando Wii™.

Al igual que en el caso de la aplicación mando Arduino este es el presupuesto total para la creación de un primer ejemplar. Para realizar más ejemplares el precio se reduciría debido a que las horas de mano de obra necesaria es menor. Por lo que el presupuesto queda reducido al hardware y la mano de obra necesaria para la puesta en marcha de la aplicación. En este caso el hardware

es más caro por lo que el precio es más elevado que en el caso del mando Arduino. El precio de una unidad podría ser de unos 742 € (681.93 € del hardware + 60 € por 3h de mano de obra). Sin tener en cuenta el PC, ya que este es un dispositivo intermedio y se puede utilizar uno ya disponible u otro más barato, el precio de una unidad sería de unos 142€ (81.94 € del hardware + 60 € mano de obra).

6. Conclusiones y propuestas de mejora.

En el presente trabajo se han logrados los objetivos propuestos ya que se ha logrado guiar el robot mediante un dispositivo externo y realizar la grabación y reproducción de trayectorias.

Para lograr que el UR5e realizara las tareas propuestas a partir de la información enviada desde un dispositivo externo se estudiaron diferentes alternativas. La idea inicial era ejecutar todo de forma remota, es decir, un programa que se ejecutara fuera del robot y que construyera a partir de la información del mando un mensaje para mover el robot. Por otra parte, se pretendía que el robot enviara información sobre su posición al programa ejecutándose remotamente y esta información se usaría para reproducir la trayectoria. Sin embargo, se realizaron algunas pruebas basándose en esta idea inicial y no se obtuvieron resultados satisfactorios por lo que se tuvo que cambiar la estrategia. Debido a estos malos resultados se analizó la posibilidad de que hubiera un programa ejecutándose en el UR5e, esto aporta muchas facilidades a la hora de grabar trayectorias y los movimientos del robot son más suaves, cuando se ejecutaba todo de forma remota y se enviaba el comando al robot este se movía a tirones. A partir de esta nueva estrategia se diseñaron las aplicaciones de Mando Arduino y Mando Wii™ con las que se lograron los objetivos.

En la aplicación Mando Arduino en un principio se pensó que este se comunicara con el PC mediante bluetooth y este a su vez con el UR5e, de esta manera se podría diseñar una interfaz de usuario para esta aplicación. Sin embargo, esta opción se descartó ya que no aportaba ninguna ventaja sobre la aplicación Mando Wii™. Por ello, se eliminó la necesidad de un dispositivo intermedio estableciendo la comunicación mediante el protocolo TCP/IP directamente entre la placa Arduino y el UR5e. Un problema encontrado fue la necesidad de utilizar el Arduino MEGA, debido al número de pines necesarios, ya que el tamaño de esta placa es grande y dificulta la construcción de un mando manejable. Para minimizar este problema se utilizaron pulsadores pequeños, un módulo independiente con W5100 (en lugar del shield de Arduino que es de mayor tamaño) y se decidió que la batería fuera en el exterior.

Por su parte en la aplicación Mando Wii™ se necesita una librería para poder utilizar el mando en un programa en Java, por lo que ha sido necesario estudiar la librería y entender las posibilidades que ofrece. Mediante el uso de esta librería se ha podido implementar los algoritmos necesarios para utilizar el mando Wii™ para controlar el robot.

A lo largo del desarrollo del trabajo aparecieron diferentes problemas, que se fueron subsanando de la mejor manera posible. En el programa del robot no se permite crear un vector y que este vaya creciendo conforme se vayan introduciendo valores, por tanto, los vectores para guardar las posiciones del robot y el estado de la herramienta deben definirse al principio con su tamaño. Por tanto, el vector es finito y puede que se alcance el final y no continúe grabando. Para solucionarlo se ha creado un vector de 100 posiciones, este

problema no es grave, ya que se puede hacer el vector lo grande que se desee, aunque no es una forma “bonita” funciona. Por otro lado, la precisión del joystick utilizado es mala por lo que no se ha podido ajustar la velocidad de movimiento de los ejes del robot en función de la posición del joystick. También se planteó la opción de utilizar las variables de instalación para guardar las trayectorias grabadas y así que estas no se borren cuando se apaga el robot. Al intentar guardar estos vectores daba errores al intentar guardar la instalación, como daba problemas y no es algo critico no se investigó más sobre el tema.

En definitiva, se ha logrado diseñar dos aplicaciones que permiten controlar las acciones del robot desde un dispositivo externo. Las acciones que se pueden controlar son: el movimiento libre del robot, la grabación de dos tipos diferentes de trayectorias y la reproducción de trayectorias. Además, para hacer más completas las aplicaciones se han añadido funciones de seguridad como detener el movimiento del robot si se ha superado la fuerza máxima de seguridad permitida en la herramienta o si se ha pulsado el botón de emergencia. También se ha gestionado la vuelta a la normalidad de forma segura tras esta situación de bloqueo.

Para trabajos futuros se podría realizar ciertas mejoras como realizar una URCap que permitiera ajustar los valores de velocidad de movimiento, esto se podría realizar fácilmente, por ejemplo, haciendo que el valor introducido multiplicara a los valores de velocidad de cada eje recibidos desde el mando. También se podría modificar el tiempo cada cuanto se guarda la posición del robot durante la grabación de trayectorias, actualmente se guarda cada 0.5 segundos y es posible que para ciertas aplicaciones este no sea el tiempo adecuado. Estos ajustes en la aplicación del Mando Wii™ se podrían realizar desde la interfaz de usuario diseñada.

En el mando Arduino se podría utilizar el módulo ESP8266, este es un módulo WIFI que permitiría al mando conectarse a la red de forma inalámbrica.

Una de las mejoras más interesantes sería la de realizar un mando utilizando una placa como la Raspberry Pi o la BeagleBone Black. Esta última es una placa programable con la capacidad de convertirse en un ordenador de bajo consumo que dispone de un sistema operativo en base Linux. Con esta se podría realizar un mando que incluye las ventajas de las dos aplicaciones diseñadas. Por un lado, permite una programación concurrente, por lo que facilita el intercambio de mensajes y podría incluir dispositivos como leds o zumbadores para dar información sobre el estado y las acciones que está llevando a cabo el robot, mientras envía a este, con la frecuencia deseada, la información relativa al estado de sus botones y joystick. Por otra parte, esta se puede comunicar con el UR5e mediante el protocolo TCP/IP sin la necesidad de dispositivos intermedios y además tampoco necesita módulos externos ya que esta placa incluye un puerto ethernet. Además, permite el diseñar un mando específico para esta aplicación, y como el tamaño de esta placa es menor que la placa Arduino MEGA y al no necesitar modulo externo para la comunicación TCP el tamaño del mando se podría reducir considerablemente.

También se podrían añadir ciertas funciones como poder grabar una tercera trayectoria combinando los dos tipos de movimientos (rectilíneo y libre), por ejemplo, habilitando un botón que al pulsarlo indicara que esos movimientos se deben reproducir con un movimiento rectilíneo.

En este trabajo no se ha diseñado una aplicación para que el robot realice un trabajo concreto, por eso hay ciertas funciones que no se han implementado pero que se podrían implementar fácilmente si la tarea a realizar por el robot lo requiriera. Estas funciones son, por ejemplo, si la tarea a realizar debe repetirse varias veces se podría hacer que la trayectoria grabada se repitiera cíclicamente hasta que se volviera a pulsar el botón de realizar trayectorias, esto no sería muy complicado ya que se realizaría igual que la gestión de inicio y fin de grabación de trayectoria que también se realiza mediante un solo botón. Si fuera necesario se podría añadir alguna función más de seguridad como por ejemplo definir una zona de trabajo y que si entra alguna persona en ella el robot reduzca su velocidad o se detenga hasta que la persona salga de la zona, para ello haría falta utilizar un sensor externo.

Todas estas mejoras citadas anteriormente son una buena opción para ampliar y mejorar la aplicación diseñada y son un buen punto de partida para posibles trabajos futuros.

7. Bibliografía y referencias.

[1]. Leyes de la robótica de Isaac Asimov.

https://hipertextual.com/2017/01/isaac-asimov-robotica

[2]. Definición de robot industrial.

http://platea.pntic.mec.es/vgonzale/cyr_0204/ctrl_rob/robotica/industrial.htm

[3]. Definición robótica de servicio. Tema 0 asignatura Robótica de servicios.

[4]. Robot colaborativo YuMi empresa Abb.

https://new.abb.com/products/robotics/es/robots-industriales/yumi

[5]. Formación como programador de robots de Universal Robots.

https://www.universal-robots.com/es/academy/

[6]. Manual de usuario UR5e.

https://cfzcobots.com/wp-content/uploads/2018/06/UR5e_User_Manual_es_Global-reducido.pdf

[7]. Manual de PolyScope.

https://s3-eu-west-1.amazonaws.com/ur-support-site/46190/Software_Manual_es_Global.pdf

[8]. The URScript Programming Language.

https://s3-eu-west-1.amazonaws.com/ur-support-site/44052/scriptManual_en.pdf

[9]. Librería que permite la utilización del mando Wii™ desde Java.

https://code.google.com/archive/p/wiiusej/downloads

Antecedentes e historia de robots:

- https://www.areatecnologia.com/electronica/tipos-de-robots.html
- https://es.wikipedia.org/wiki/R.U.R._(Robots_Universales_Rossum)
- https://quasar106.odoo.com/page/10-robots-de-la-historia
- https://proyectoidis.org/automatas-de-jaquet-droz/
- https://robotsinaction.com/el-primer-robot-industrial-unimate/
- https://www.universal-robots.com/es/industrias/

- http://www.udesantiagovirtual.cl/moodle2/mod/book/view.php?id=24914

Tipos de robots y robots colaborativos:

- https://www.areatecnologia.com/electronica/tipos-de-robots.html
- https://www.fib.upc.edu/retro-informatica/avui/salut.html

Robots colaborativos y principales marcas en el mercado:

- https://blog.universal-robots.com/es/beneficios-robots-colaborativos
- https://movicontrol.es/robots-colaborativos/?gclid=Cj0KCQjwkK_qBRD8ARIsAOteukAx0s4JaMS62PYeBLdPQBkx4j7sNe-w_RAB2Mqp59kdMaYMAraKvccaAuDOEALw_wcB
- https://hablemosdeempresas.com/grandes-empresas/que-son-los-cobots/
- https://www.kuka.com/es-es/acerca-de-kuka/historia
- https://www.kuka.com/es-es/future-production/cooperaci%C3%B3n-hombre-robot
- https://new.abb.com/es/abb-in-spain/quienes-somos/historia
- https://new.abb.com/products/robotics/es
- https://www.fanuc.eu/es/es/quienes-somos/fanuc-historia
- http://www.reporteroindustrial.com/temas/Robots-colaborativos-para-estaciones-de-trabajo-con-humanos+119806

Cobots de Universal Robots:

- https://www.universal-robots.com/es/acerca-de-universal-robots/noticias/historia-de-los-cobots/
- https://www.universal-robots.com/es/productos/robot-ur10/
- https://www.universal-robots.com/es/productos/robot-ur5/
- https://www.universal-robots.com/es/productos/robot-ur3/
- https://www.universal-robots.com/es/industrias/

Métodos programación de robots

- https://es.wikibooks.org/wiki/Rob%C3%B3tica/M%C3%A9todos_de_programaci%C3%B3n_de_un_robot
- Tema 1. Sistemas robotizados (asignatura 4º de GIEIA)

Requerimientos del trabajo UR5e:

- https://www.universal-robots.com/media/1801300/esp_semea_199912_ur5_tech_spec_web_a4.pdf
- https://www.universal-robots.com/media/50591/ur5_es.pdf
- https://www.universal-robots.com/plus/end-effectors/rg2-gripper/
- https://onrobot.com/es/node/1
- https://onrobot.com/sites/default/files/documents/RG2_datasheet_V1.9.1.pdf

Requerimientos del trabajo Arduino:

- https://www.arduino.cc/en/Guide/Introduction
- http://arduino.cl/que-es-arduino/
- https://store.arduino.cc/arduino-uno-rev3
- https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno
- https://store.arduino.cc/arduino-nano
- https://store.arduino.cc/mega-2560-r3
- https://www.luisllamas.es/arduino-ethernet-enc28j60/
- https://www.luisllamas.es/arduino-ethernet-shield-w5100/

Requerimientos del trabajo mando Nintendo® Wii™:

- https://www.fayerwayer.com/2013/10/un-poco-sobre-la-nintendo-wii/
- https://www.xataka.com/videojuegos/especial-controles-de-videojuegos-wiimote

Requerimientos software Arduino:

- https://aprendiendoarduino.wordpress.com/category/ide/
- https://www.arduino.cc/en/main/software
- https://www.arduino.cc/en/Reference/Ethernet
- https://www.arduino.cc/en/reference/SPI
- https://aprendiendoarduino.wordpress.com/2015/03/26/lenguaje-de-programacion-c/
- https://www.arduino.cc/reference/en/
- https://aprendiendoarduino.wordpress.com/2015/03/26/lenguaje-de-programacion-de-arduino-estructura-de-un-programa/
- https://es.wikibooks.org/wiki/Lenguaje_de_programaci%C3%B3n_Arduino
- Temas de Redes y sistemas Distribuidos para el control (asignatura MUAII).

Requisitos software Mando Wii™:

- http://www.tuprogramacion.com/programacion/historia-de-java/
- https://www.eclipse.org/eclipseide/

Requerimiento software UR5e:

- https://www.universal-robots.com/how-tos-and-faqs/how-to/ur-how-tos/remote-control-via-tcpip-16496/

Opciones de alimentación:

- https://www.luisllamas.es/alimentar-arduino-baterias/

Anexos

Anexo I. Manual de usuario.

En el presente apartado se muestra la información necesaria para permitir al usuario utilizar las aplicaciones diseñadas con el fin de controlar el UR5e. Se explicará como establecer la comunicación entre el mando y el UR5e, y las diversas funciones que presenta.

Manual de usuario Mando Arduino.

El primer paso es establecer la comunicación entre el Mando Arduino y el UR5e, para ello ambos deben estar conectados a la misma red. Una vez están conectados a la misma red se alimenta la placa Arduino lo que inicia el servidor, a continuación, se ejecuta el programa del robot que establecerá la conexión como cliente. Esta información se mostrará al usuario en la pestaña Log de la consola de programación.

Nota: En el programa de Arduino se ha definido la IP del dispositivo, esta se utiliza por el robot para conectarse al servidor. Si no se establece la conexión, se puede conectar el Arduino al PC para comprobar que la IP es correcta (abriendo el monitor serie se muestra la IP y la MAC de la placa).

Una vez establecida la comunicación el robot está listo para escuchar órdenes enviadas desde el mando. A continuación se va a presentar los diferentes elementos que forman el Mando Arduino y la función de cada uno de ellos.

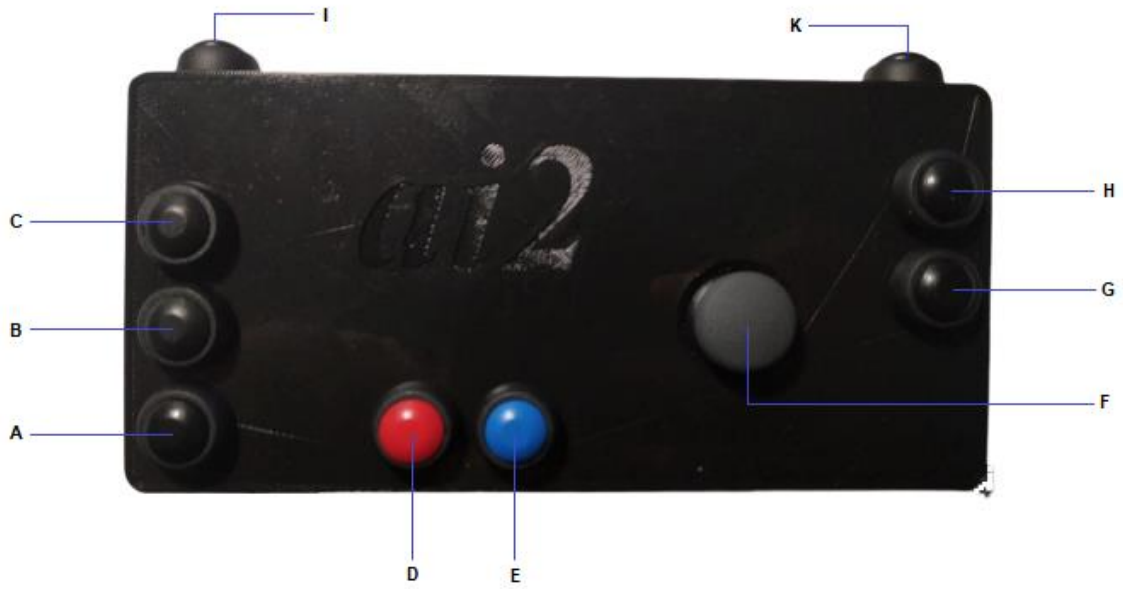


Figura 83.Mando Arduino vista frontal.

Figura 84.Mando Arduino vista superior.

A) Botón 1

B) Botón 2

C) Botón 3

D) Botón Parada de emergencia

E) Botón Home

F) Joystick y Botón herramienta

G) Botón T1

H) Botón T2

I) Botón 4

J) Ethernet

K) Botón DT

Para seleccionar los ejes en los que se realizan movimientos se utilizan los botones 1, 2, 3 y 4.

- Botón 1: Movimientos en el plano XY.
- Botón 2: Movimientos en el plano XZ.
- Botón 3: Movimientos en el plano YZ.
- Botón 1 + Botón 4: Giros en el eje X.
- Botón 2 + Botón 4: Giros en el eje Y.

- Botón 3 + Botón 4: Giros en el eje Z.

Una vez seleccionados el tipo de movimiento se utiliza el joystick para mover el robot.

Para gestionar la grabación y la realización de trayectorias se utilizan los botones T1 ,T2 y DT.

- Botón T1: Inicio y final grabación trayectoria 1.
- Botón T2: Inicio y final grabación trayectoria 2.
- Botón T1 + Botón DT: Realización trayectoria 1.
- Botón T2 + Botón DT: Realizando trayectoria 2.

Para realizar una parada de emergencia se utiliza el botón de Parada de emergencia.

- Botón Parada de emergencia: Parada de emergencia.

Para salir de la situación de bloqueo debida a una parada de emergencia o a que se ha superado la fuerza máxima de seguridad se pulsa el botón Home.

- Botón Home: Restablecer valores, devolver al robot a su funcionamiento normal de forma segura.

Por último, para activar la garra se ha habilitado el botón del Joystick.

- Botón Herramienta: Modifica estado de la herramienta.

Durante la ejecución de la aplicación el usuario puede observar información relativa al estado del robot y de las acciones que está llevando a cabo en la pestaña Log de la consola de programación. En la siguiente tabla se presentan los mensajes que se muestran por pantalla en función de la acción que se esté realizando.

Acción del UR5e	Información mostrada en la pestaña Log
Robot inicia la comunicación con el servidor	Mensaje: *"Conexion Establecida"*
Inicia grabación trayectoria 1	Mensaje: *"Grabando Trayectoria 1…"*
Finaliza grabación de la trayectoria 1	Mensaje: *"Trayectoria 1 Grabada Correctamente"*
Comienza a realiza la trayectoria 1	Mensaje: *"Realizando Trayectoria 1…"*
Termina de realizar la trayectoria 1	Mensaje: *"Finalizada Trayectoria 1"*
Inicia grabación trayectoria 2	Mensaje: *"Grabando Trayectoria 2…"*
Finaliza grabación de la trayectoria 2	Mensaje: *"Trayectoria 2 Grabada Correctamente"*
Comienza a realiza la trayectoria 2	Mensaje: *"Realizando Trayectoria 2…"*
Termina de realizar la trayectoria 2	Mensaje: *"Finalizada Trayectoria 2"*
Se supera la fuerza máxima permitida	Mensaje: *"Fuerza de Seguridad Superada (N) = X"* (X = valor fuerza alcanzada)
Se realiza una parada de emergencia	Mensaje: *"PARADA DE EMERGENCIA"*
Se restablecen los valores	Mensaje: *"Valores Restablecidos. Robot en funcionamiento normal"*

Tabla 29.Información mostrada en la pestaña Log de la consola de programación sobre las actividades realizadas por el robot

Manual de usuario Mando Wii™.

El primer paso es comunicar el PC con el mando Wii™ mediante bluetooth. Para ello se debe seguir las siguientes indicaciones:

1. Desde el PC se va a *Panel de control* → *Hardware y sonido* → *Dispositivos e impresoras* → *Agregar un dispositivo*

2. En este momento se comienza a buscar los dispositivos, para que el mando Wii™ este visible se debe mantener pulsados los botones 1 y 2 del mando, los 4 leds parpadearán lo que indica que está buscando dispositivo para establecer la comunicación.

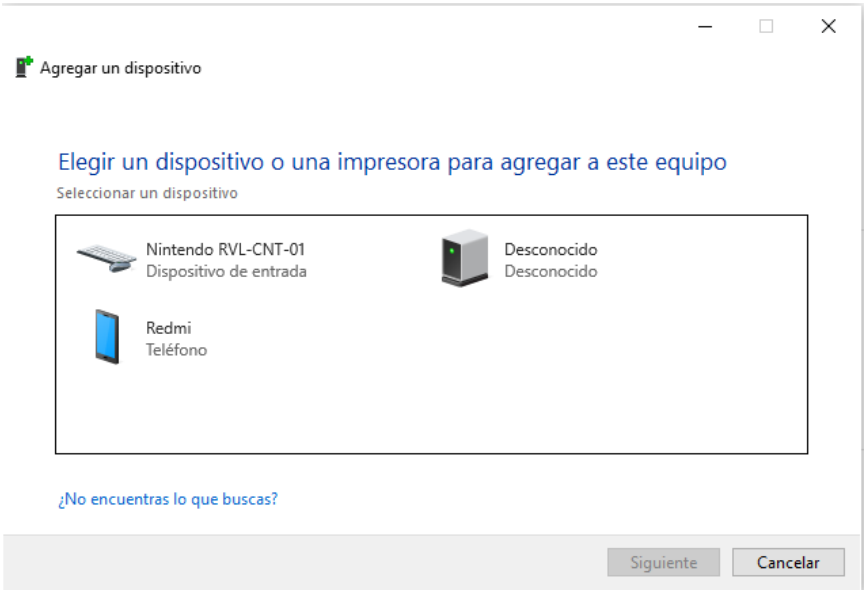


Figura 85.Conexión mando Wii™ a PC (1).

3. Una vez encontrado se selecciona el dispositivo, se pide una contraseña, pero este método permite saltar al siguiente paso (si se intenta conectar desde *configuraciones* no se permite saltar este paso y se debe introducir la contraseña para establecer la comunicación). A continuación, el dispositivo queda agregado.

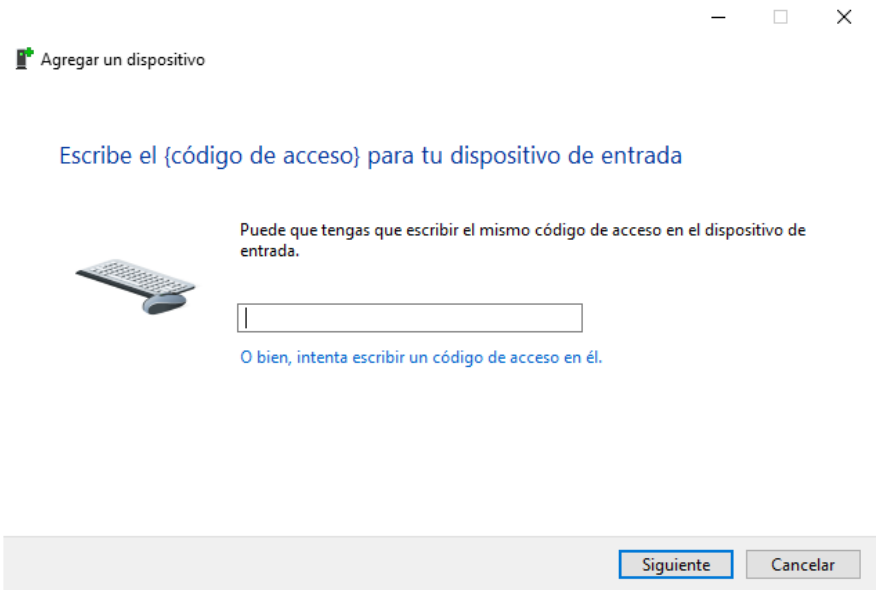


Figura 86.Conexión mando Wii™ a PC (2).

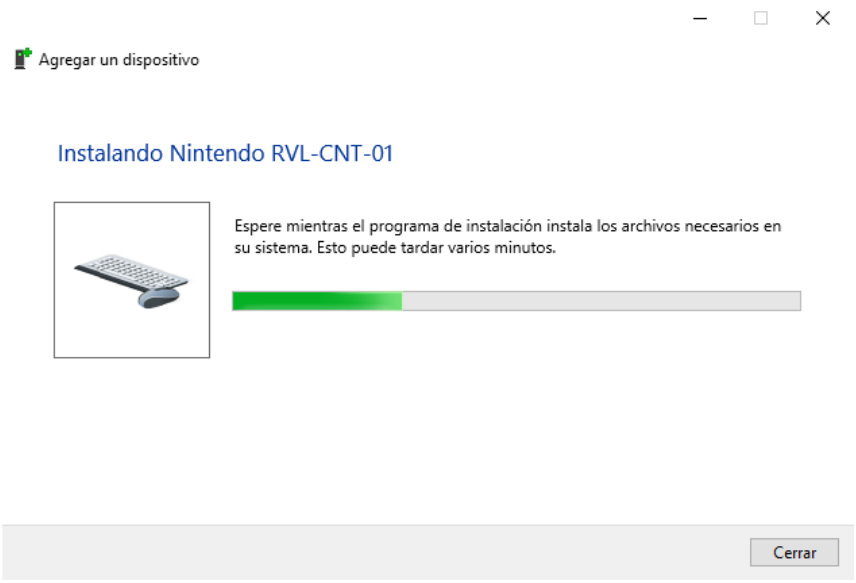


Figura 87.Conexión mando Wii™ a PC (3).

4. Para poder utilizar el mando de la Wii™ se va a utilizar el programa *Dolphin* que emula un adaptador bluetooth de la Wii™ (aunque existen otros programas para esta función). Al iniciar el programa se abre la pestaña *Mandos*.

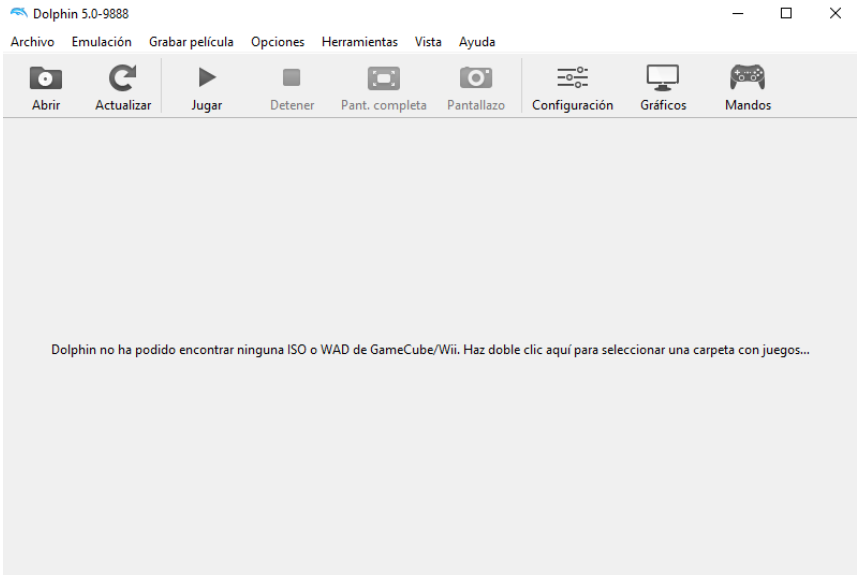


Figura 88.Conexión mando Wii™ a PC (4).

5. En esta pestaña se indica que se va a utilizar un mando Wii™ real se pulsa *Actualizar* para buscar el dispositivo. El mando Wii™ vibra y se enciende el led 1 para indicar que se ha establecido la comunicación.

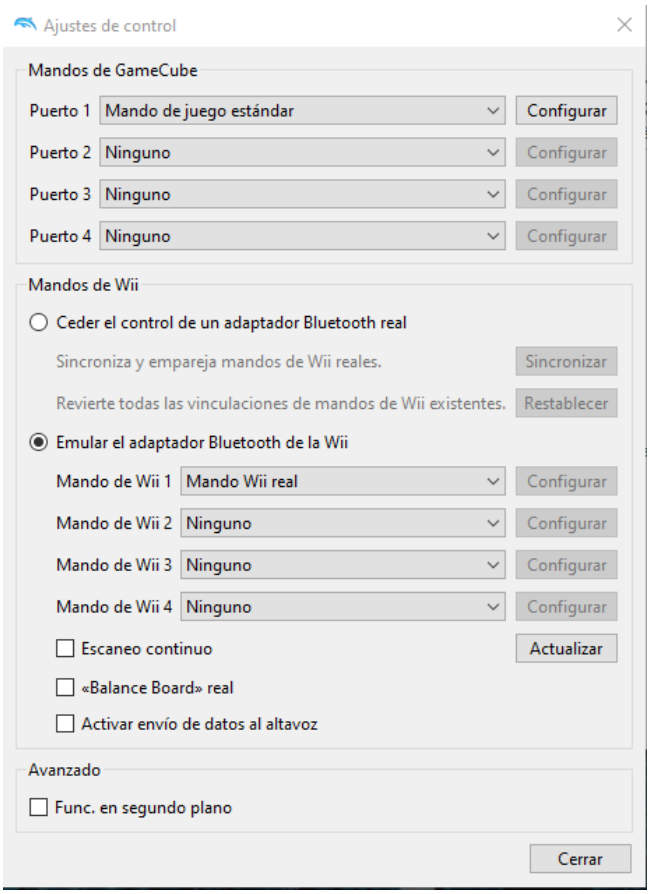


Figura 89.Conexión mando Wii™ a PC (5).

De esta forma el mando Wii™ está conectado y listo para funcionar.

Una vez conectado el mando se ejecuta la aplicación Java diseñada. Mediante la interfaz de la aplicación se define el puerto que va a escuchar el servidor y se inicia, el PC debe estar conectado a la misma red que el UR5e. Cuando comience a ejecutarse el programa del robot se establecerá la conexión y el robot estará listo para obedecer las órdenes recibidas.

Nota: En este caso no hay que configurar nada más para establecer la comunicación ya que el programa del robot se ha configurado la IP del PC que se ha utilizado como servidor. Para poder utilizar otro PC como servidor habrá que cambiar la direccion IP en el programa del robot o cambiar la IP del PC a la establecida en el programa del robot.

A continuación se va a presentar los diferentes elementos que forman el Mando Wii™ y la función de cada uno de ellos.

Wiimote y nunchuk vista frontal y posterior.

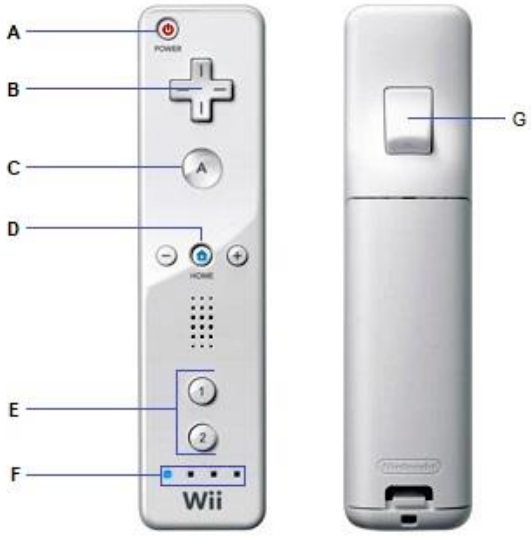


Figura 90.Wiimote.

Figura 91.Nunchuk.

A) Botón Encendido

B) Cruz de control

C) Botón A

D) Botón Home

E) Botón 1 y 2

F) Leds indicadores

G) Botón B

H) Joystick

I) Botón C

J) Botón Z

Para seleccionar los ejes en los que se realizan movimientos se utilizan los botones B, A y Z.

- Botón B: Movimientos plano XY.
- Botón B + Botón Z : Movimiento eje Z.

- Botón B + Botón A: Giros en eje Y Z.
- Botón B + Botón A + Botón Z: Giros eje X.

Una vez seleccionados el tipo de movimiento se utiliza el joystick para mover el robot.

Para gestionar la grabación y la realización de trayectorias se utiliza la cruz de control.

- Cruz Arriba: Inicio y final grabación trayectoria 1.
- Cruz Derecha: Realización trayectoria 1.
- Cruz Abajo: Inicio y final grabación trayectoria 2.
- Cruz Izquierda: Realización trayectoria 2.

Para realizar una parada de emergencia se han habilitado los botones 1 y 2.

- Botón 1: Parada de emergencia.
- Botón 2 : Parada de emergencia.

Para salir de la situación de bloqueo debida a una parada de emergencia o a que se ha superado la fuerza máxima de seguridad se debe cerrar la pestaña emergente que aparece en la interfaz y se pulsa el botón Home.

- Botón Home: Restablecer valores, devolver al robot a su funcionamiento normal de forma segura.

Por último, para activar la garra se ha habilitado el Botón C.

- Botón C: Modifica estado de la herramienta.

Durante la ejecución de la aplicación el usuario puede observar información relativa al estado del robot y de las acciones que está llevando a cabo tanto en la interfaz como en el mando. En la siguiente tabla se presentan los mensajes que se muestran por pantalla y la información ofrecida por el mando en función de la acción que se esté realizando.

Acción del UR5e	Información interfaz	Información Mando Wii™
Robot conectado	Mensaje *"Robot Conectado"*	Se encienden led 2 y 3
	Imagen robot a color	
	Led verde.	
Grabar trayectoria 1	Mensaje: *"Grabando Trayectoria 1"*	Se enciende led 1
	Led verde	
Grabar trayectoria 2	Mensaje: *"Grabando Trayectoria 2"*	Se enciende led 2
	Led verde	
Realizar trayectoria 1	Mensaje: *"Realizando Trayectoria 1"*	Se encienden led 1 y 4
	Led verde	
Realizar trayectoria 2	Mensaje: *"Realizando Trayectoria 2"*	Se encienden les 2 y 4
	Led verde	
Parada de emergencia	Mensaje: *"PARADA DE EMERGENCIA"*	Se encienden led 1, 2, 3 y 4
	Led rojo	
Fuerza de seguridad superada	Mensaje *"Fuerza de seguridad superada"*	Se encienden led 1, 2 y 3
	Led rojo	
Restablecer valores	Mensaje: *"Valores Restablecidos"*	Se encienden led 2 y 3
	Led verde	

Tabla 30. Información mostrada en la interfaz y en el mando sobre las actividades realizadas por el robot.

Anexo II. Esquemas de conexiones eléctricas.

Se muestra el esquema general con todos los dispositivos que forman el mando conectados. Para ver las conexiones de manera más detallada se adjuntan por separado: un esquema de conexión de los diferentes botones, un esquema de la conexión del joystick y un esquema de la conexión del módulo independiente con W5100.

Joystick

Vcc GND VRX VRY SW

W5100

Vcc GND SS MO G P- R CK MI P+

Arduino MEGA 2560

3V3 5V Vin

RESET AREF IOREF A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15

D13 PWM D12 PWM D11 PWM D10 PWM D9 PWM D8 PWM D7 PWM D6 PWM D5 PWM D4 PWM D3 PWM D2 PWM D1 TX0 D0 RX0

D14 TX3 D15 RX3 D16 TX2 D17 RX2 D18 TX1 D19 RX1 D20 SDA D21 SCL

D38 D39 D40 D41 D42 D43 D44 D45 D46 D47 D48 D49 D50 D51 D52 D53

D22 D23 D24 D25 D26 D27 D28 D29 D30 D31 D32 D33 D34 D35 D36 D37

GND

BDT 1 2 4K7

BT2 1 2 4K7

BT1 1 2 4K7

BEmergencia 1 2 4K7

BHome 1 2 4K7

B4 1 2 4K7

B3 1 2 4K7

B2 1 2 4K7

B1 1 2 4K7

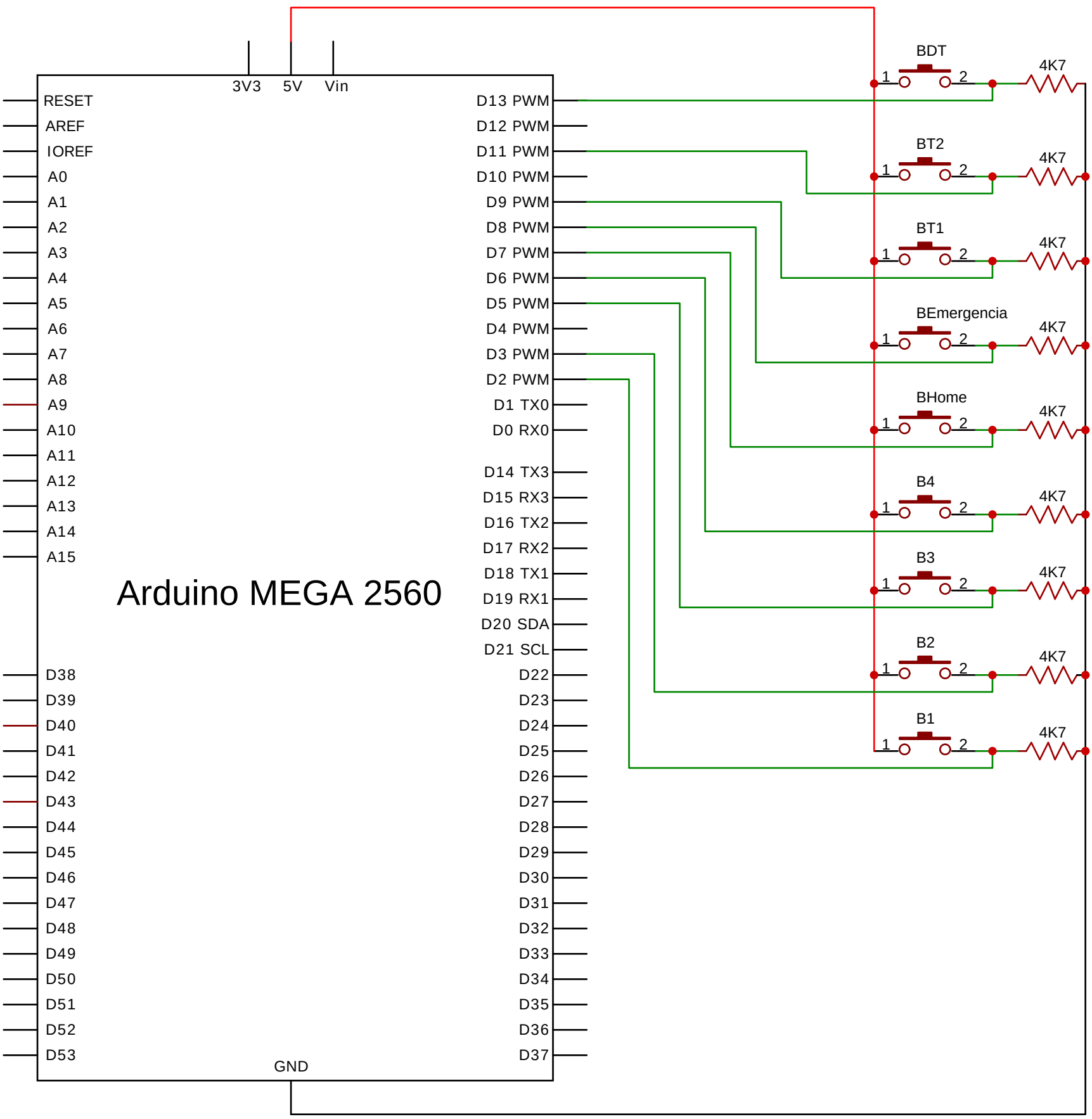
TITULO:
Esquema de conexiones

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Fecha: Septiembre 2019

Hoja: 1/4

Autor: Juan Rodrigo Vizcaino Espejo



TITULO:

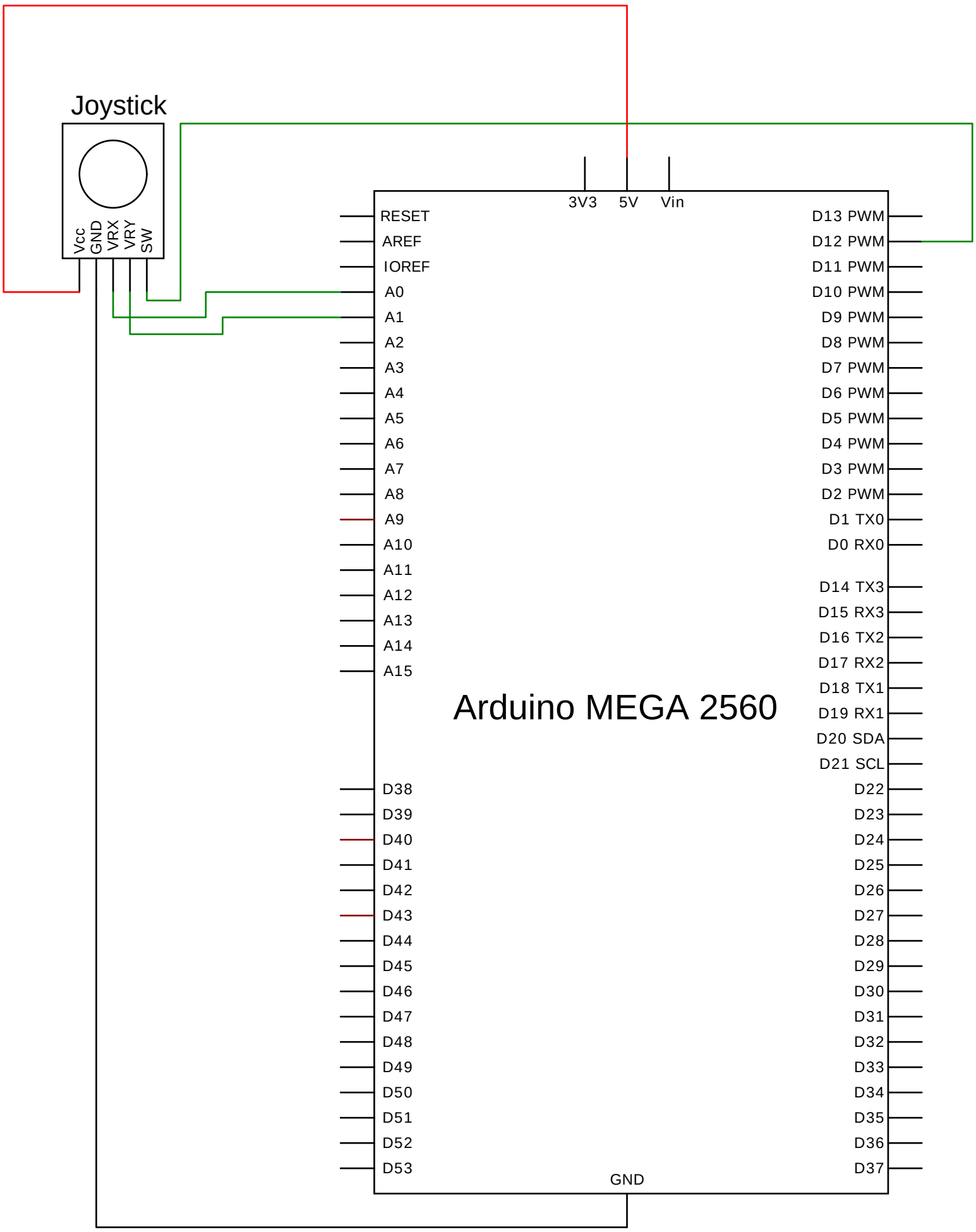
Esquema de conexion de botones

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Fecha: Septiembre 2019

Hoja: 2/4

Autor: Juan Rodrigo Vizcaino Espejo



TITULO:

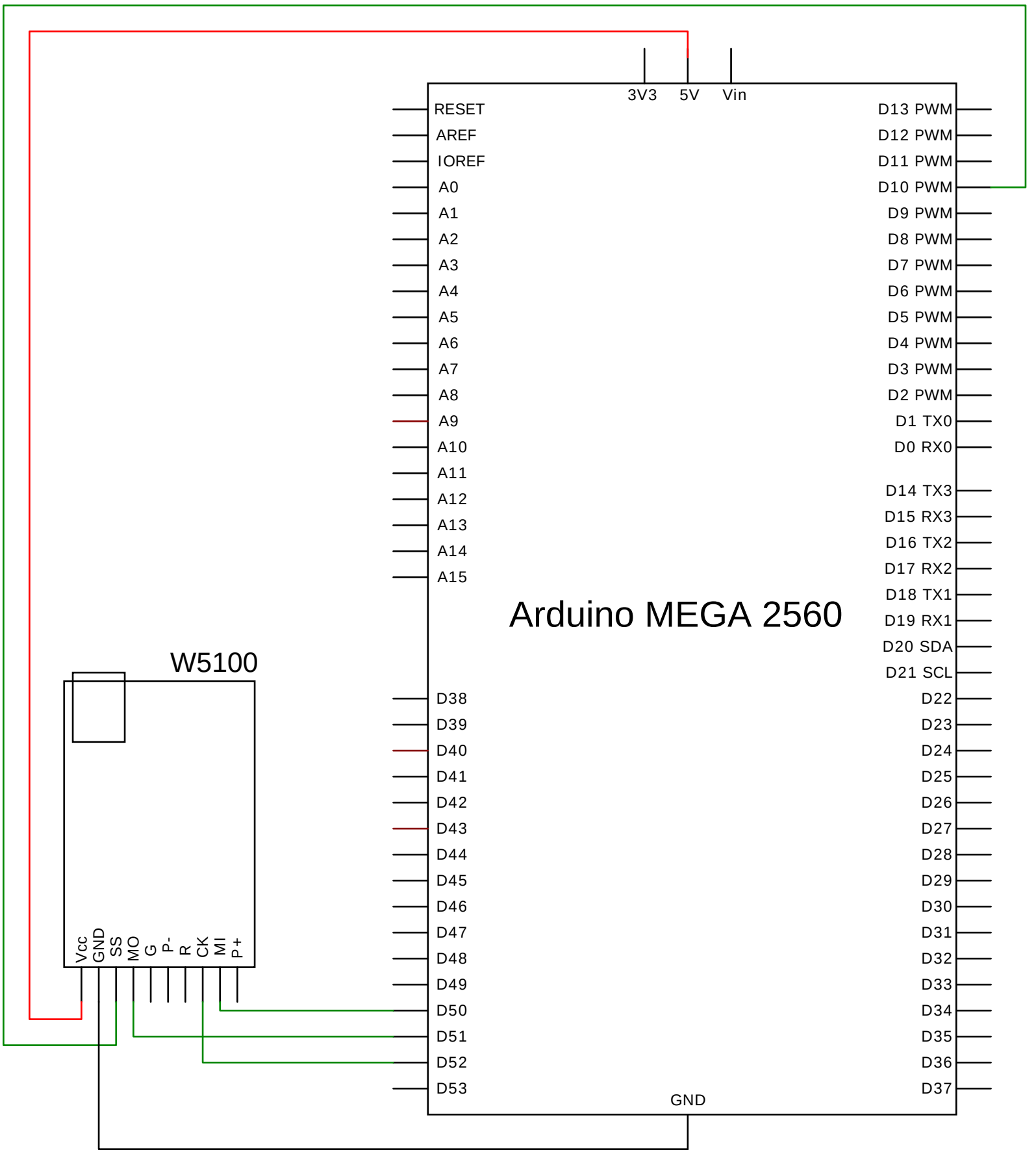
Esquema de conexion del joystick

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Fecha: Septiembre 2019

Hoja: 3/4

Autor: Juan Rodrigo Vizcaino Espejo



TITULO:

Esquema de conexion del modulo con W5100

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

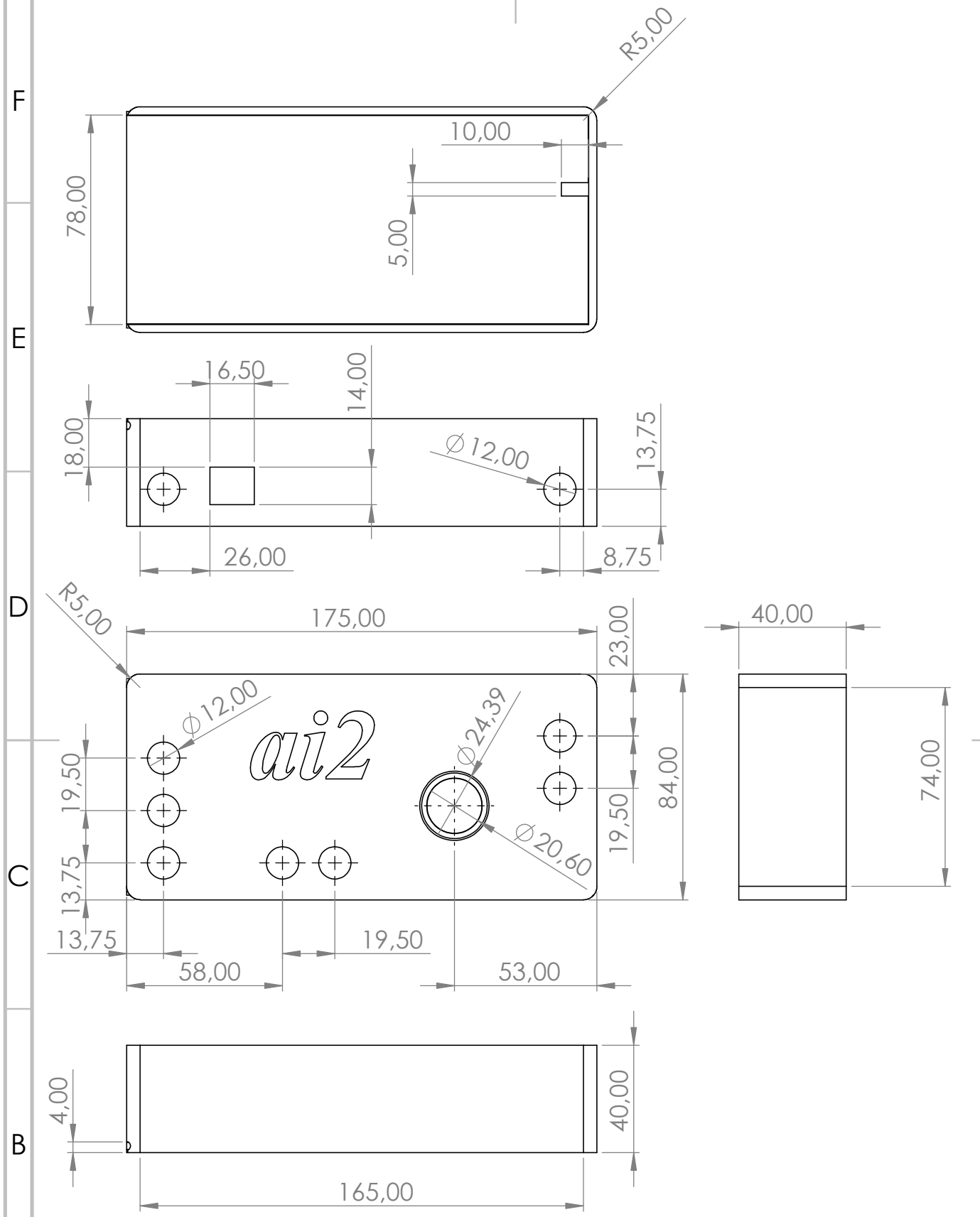
Fecha: Septiembre 2019

Hoja: 4/4

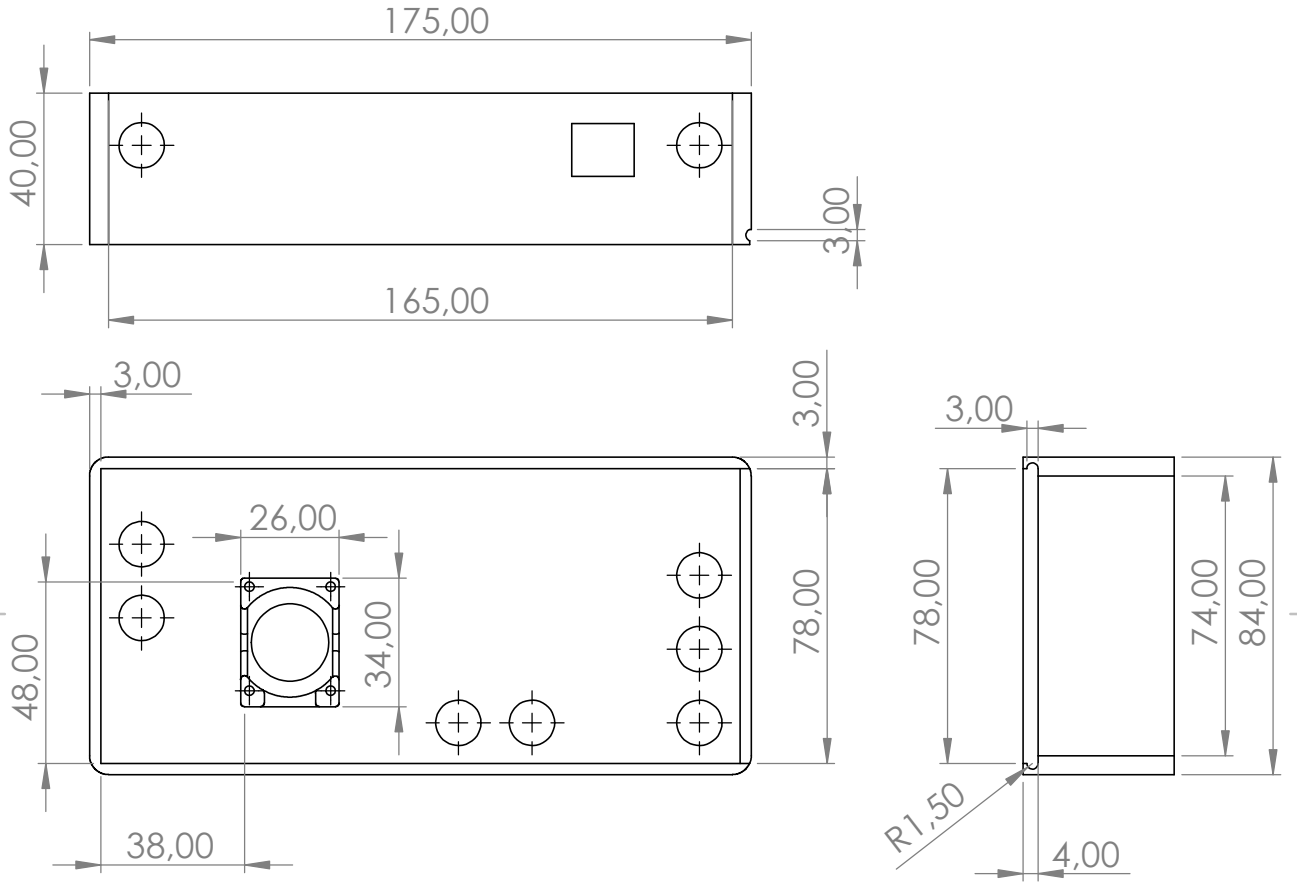
Autor: Juan Rodrigo Vizcaino Espejo

Anexo III. Planos del mando Arduino.

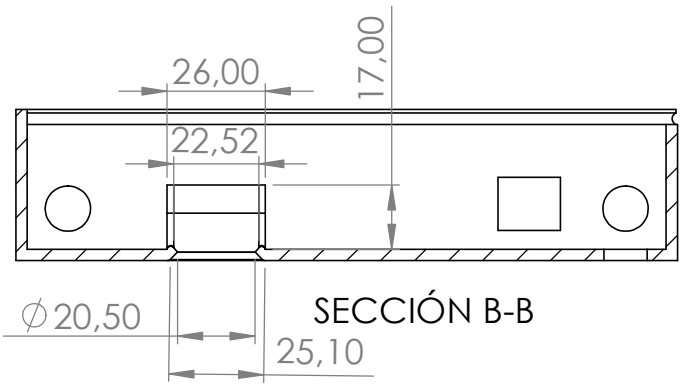
Se muestran los planos del mando Arduino diseñado y construido



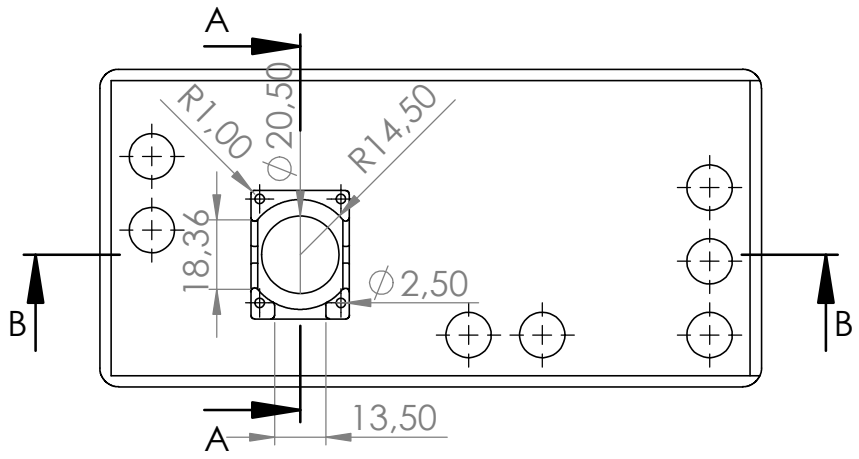
TITULO: Mando Arduino con tapa		Escala: 1:2
Autor: Juan Rodrigo Vizcaino Espejo		Hoja: 1/3
Fecha: Septiembre 2019		Unidades: mm

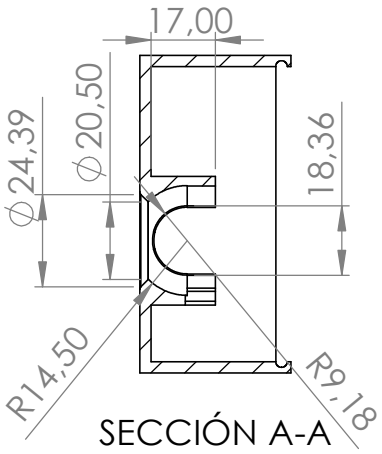


TITULO: Mando Arduino sin tapa		Escala: 1:2
Autor: Juan Rodrigo Vizcaino Espejo		Hoja: 2/3
Fecha: Septiembre 2019		Unidades: mm



SECCIÓN B-B





SECCIÓN A-A

TITULO: Mando Arduino estructura joystick	Escala: 1:2
Autor: Juan Rodrigo Vizcaino Espejo	Hoja: 3/3
Fecha: Septiembre 2019	Unidades: mm