UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

TRABAJO FIN DE GRADO

Grado en Ingeniería Electrónica Industrial y Automática

Universidad Politécnica de Valencia

Escuela Técnica Superior de Ingeniería del Diseño

Diseño y programación de un sistema robotizado para el juego del *Tres en raya*

Autor: María Dolores Tíscar López

Tutor: Ranko Zotovic Stanisic

Curso académico: 2019/2020

Resumen

En el presente Trabajo de Fin de Grado, se muestra el desarrollo del diseño y programación de un sistema robotizado colaborativo para un juego de estrategia permitiéndole jugar de forma autónoma con un humano.

El desarrollo del juego cuenta con una variedad de inteligencia artificial como es la visión artificial, así como un sensor de fuerza permitiendo, gracias a esto, alcanzar el grado de autonomía deseado. Así mismo, a medida que se va desarrollando, se explican los diferentes problemas que han ido surgiendo y las posibles soluciones a adoptar.

La finalidad de este trabajo es dejar ver como este tipo de robots pueden colaborar con humanos incluso interactuar sin necesidad de grandes medidas de seguridad.

Para la realización de este proyecto se ha escogido el robot colaborativo UR3 de Universal Robots, el equipo de visión Wrist Camera de Robotiq y el sensor de fuerza HEX-EB165 de OnRobot, de los que dispone el departamento de Ingeniería de Sistemas y Automática.

Palabras clave: programación, robot colaborativo, visión artificial, sensor de fuerza y tres en raya.

Resum

En el present Treball de Fi de Grau, es mostra el desenvolupament del disseny i programació d'un sistema robotitzat col·laboratiu per a un joc d'estratègia permetent-li jugar de forma autònoma amb un humà.

El desenvolupament del joc compta amb una varietat d'intel·ligència artificial com és la visió artificial, així com un sensor de força permetent, gràcies a això, aconseguir el grau d'autonomia desitjat. Així mateix, a mesura que es va desenvolupant, s'expliquen els diferents problemas que han anat surgint i les possibles solucions adoptades.

La finalitat d'aquest treball és deixar veure com aquesta mena de robots poden col·laborar amb humans fins i tot interactuar sense necesitat de grans mesures de seguretat.

Per a la realització d'aquest projecte s'ha triat el robot col·laboratiu UR3 de Universal Robots, el equip de visió Wrist Camera de Robotiq i el sensor de força HEX-EB165 de OnRobot, dels quals disposa el departamento d'Enginyeria de Sistemes i Automàtica.

Paraules clau: programació, robot colaboratiu, visió artificial, sensor de força i tres en ratlla.

Abstract

In the present End of Degree Project, is shown the development of the design and programming of a collaborative robotic system for a strategy game this afford autonomous play with human.

The game development has a variety of artificial intelligence such as computer vision, as well as a force sensor allowing, thanks to this, to achieve the wanted level of autonomy. In the same way, as it develops, the differents problems that have arisen and the possible solutions adopted are explained.

The purpose of this project is to show how this type of robots can collaborate and interact with humans without great security measures.

For the realization of this Project, have been chosen the collaborative robot UR3 of Universal Robots, the vision team Wrist Camera of Robotiq and the force sensor HEX-EB165 of OnRobot, of which the Departamento f Systems Engineering and Automation has.

Keywords: programming, collaborative robot, computer vision, force sensor and tic-tac-toe.

DOCUMENTOS DEL TFG

- **DOCUMENTO I** MEMORIA DESCRIPTIVA
- **DOCUMENTO II** PLANOS
- **DOCUMENTO III** PRESUPUESTO
- **DOCUMENTO IV** PLIEGO DE CONDICIONES

ÍNDICE DE LA MEMORIA DESCRIPTIVA

ÍNDICE DE FIGURAS

ÍNDICE PLANOS

ÍNDICE PRESUPUESTO

ÍNDICE PLIEGO DE CONDICIONES

DOCUMENTO I

MEMORIA DESCRIPTIVA

CAPÍTULO 1

1. Introducción

En primer capítulo se encuentra dividido en tres secciones. En la *sección 1.1* se exponen los motivos por los que se decidió llevar a cabo este trabajo. A continuación, en la *sección 1.2* se presentan los objetivos de éste. Y, por último, cuenta con la *sección 1.3* donde se explica la estructura del presente documento.

1.1. Motivación del TFG

Hoy en día la sociedad está constantemente rodeada de algún tipo de dispositivo electrónico y conectada a Internet. De esta forma es fácil encontrar en las casas lavadoras, robots de cocina, robots que limpian el polvo, juguetes que interactúan con el usuario, etc. Todo esto empieza a surgir con las ganas de tener una vida más cómoda y sin grandes esfuerzos, lo que provoca un avance a velocidad vertiginosa de la tecnología. Pero no sólo encontramos esto en los particulares, también se puede observar en hospitales, industrias, restaurante, etc.

Todos estos aparatos tienen en común un gran desarrollo de técnicas pertenecientes a la inteligencia artificial. Gracias a estas técnicas se permiten las mejoras de las funciones que llevan a cabo como, por ejemplo, evitar que el robot que limpia se choque con algún obstáculo.

Dentro del área de la robótica, se puede apreciar como el hardware evoluciona mucho más rápido que el software lo que provoca que se encuentren robots robustos pero torpes o poco inteligentes. Las líneas de investigación de la robótica se centran en esta brecha para conseguir un comportamiento inteligente o similar al del ser humano, pero no en todos es posible. Sin embargo, es fácil de encontrar robots que se acerquen a esto por sus características comunicativas con el entorno y sus formas de actuar, éstos son los conocidos como robots sociales.

Con este trabajo se pretende estudiar si es imposible que un robot no destinado a cierta tarea sea capaz de llevarla a cabo. Teniendo los medios y las técnicas necesarias para proporcionarle esa inteligencia artificial que necesita para hacer dicha tarea, seguramente sea capaz de aprender y realizarla. De esta forma, aunque sus acciones no sean perfectas como las de cualquier otro robot especializado en este campo, se obtendrá un robot mucho más versátil.

1.2. Objetivo

El trabajo recoge el desarrollo de un sistema realizado para el robot colaborativo UR3. El objetivo de dicho sistema es que el robot sea capaz de jugar de forma autónoma al *Tres en raya* sobre un tablero.

Mediante el equipo de visión artificial el robot reconocerá e identificará las diferentes piezas y lugares de estas sobre el tablero. En primer lugar, el robot reconocerá con qué pieza va a jugar, la cual dibujará. A continuación, será capaz de identificar las fichas colocadas por el rival en el tablero y, finalmente, tendrá la capacidad de saber en qué casilla dibujar en cada turno.

Para el desarrollo de este proyecto es fundamental familiarizarse con un entorno real y conocer cómo funciona el sistema robotizado, así como de la visión artificial y del sensor de fuerza.

1.3. Estructura del documento

El presente documento se ha estructurado en capítulos a fin de ofrecer una visión global del documento. A continuación, se enuncia el contenido de cada uno de estos capítulos.

En el *capítulo 1* se relatan cuáles han sido las motivaciones por las que se decidió realizar este trabajo, los objetivos de éste y la estructura que va a seguir el presente documento.

En el *capítulo 2* se realiza un estudio de la base teórica relacionada con el trabajo desarrollado.

En el *capítulo 3* se muestra una breve descripción de las características de los elementos utilizados para llevar a cabo el desarrollo del proyecto.

En el *capítulo 4* se expone el procedimiento seguido de las piezas a utilizar en el trabajo.

En el *capítulo 5* se explica el procedimiento que se ha seguido para el diseño de esta aplicación y se muestran los problemas más relevantes durante la realización de ésta.

En el *capítulo 6* se muestra y explica la estructura del programa que se ha realizado.

En el *capítulo 7* se exponen los posibles errores que han ido surgiendo durante la realización del proyecto y las posibles soluciones adquiridas.

En el *capítulo 8* se muestran los resultados finales que se han obtenido.

En el *capítulo 9* se enuncian las conclusiones extraídas de la realización del trabajo.

Por último, se encuentran la bibliografía y anexos utilizados para la búsqueda de información para el buen desarrollo del proyecto.

CAPÍTULO 2

2. Estado del arte

Para poder comprender mejor el trabajo que se ha realizado es importante conocer y entender el funcionamiento de los sensores de fuerza, de visión y de los brazos robots colaborativos ya que son los dispositivos que se utilizan para el desarrollo de este proyecto.

Aunque la utilización de los robots en procesos industriales se considera algo nuevo o incluso futurista, en verdad, no es como se conoce coloquialmente. Hace miles de años, aunque la robótica no era reconocida como ciencia y la palabra robot no surgió mucho después, ya se habían declarado los primeros robots con el nombre de autómatas. Sin embargo, las primeras máquinas herramientas que se construyeron para ayudar al hombre en su trabajo recibieron el nombre de artefactos o simples máquinas.

Desde el principio de los tiempos, el hombre ha deseado crear vida artificial para diferentes tareas como hacer simple compañía, tareas repetitivas, pesadas o incluso difíciles de realizar por un ser humano.

Se empezaron a crear autómatas como un pasatiempo con el fin de entretener construidos con materiales al alcance de todos (maderas, cobre, material moldeable, etc.). Estos primeros autómatas utilizaban la fuerza bruta para poder realizar los movimientos.

2.1. Historia de la robótica

A lo largo de los años el ser humano ha intentado imitar partes del cuerpo humano construyendo máquinas para ello. Los antiguos egipcios crearon brazos mecánicos y los griegos construyeron estatuas que funcionaban con sistemas hidráulicos.

El inicio de la robótica industrial que conocemos actualmente puede fijarse en torno al siglo XVIII dentro del sector de la producción textil. Este inicio fue impulsado durante la revolución industrial donde se pueden citar varios inventos como la hiladora giratoria (1770), la hiladora mecánica (1779), el telar mecánico (1785), entre otros, pero fue en 1801 cuando Joseph Jacquard inventó una máquina textil programable mediante tarjetas perforadas. Sin embargo, ya durante los siglos XVII y XVIII en Europa se construyeron muñecos mecánicos muy ingeniosos con grandes características de los robots, pero estas creaciones mecánicas con forma humana deben considerarse como inversiones aisladas que reflejan el genio humano que se anticiparon a su época.

El término robot empezó a surgir gracias a una obra checoslovaca publicada en 1917 escrita por el dramaturgo Karel Kapek, denominada "*Rossum's Universal Robots*". La palabra checa "*robota*" significa servidumbre o trabajador forzado y, cuando se tradujo al inglés, se convirtió en el término que hoy conocemos como robot [1].

Entre los escritores de ciencia ficción, Isaac Asimov contribuyó en muchas narraciones relativas a robots donde la imagen de éstos son máquinas bien diseñadas y con una seguridad garantizada que actúan bajo tres principios. Es a él a quien se le atribuye el acuñamiento del término Robótica. Estos principios fueron denominados como las "*Tres Leyes de la Robótica*", y son:

- **Primera ley**: un robot no puede actuar contra un ser humano o, mediante la inacción, que un ser humano sufra daños.
- **Segunda ley**: un robot debe obedecer las órdenes dadas por los seres humanos, salvo que estén en conflictos con la primera ley.
- **Tercera ley**: un robot debe proteger su propia existencia, a no ser que esté en conflicto con las dos primeras leyes.

Estas tres leyes fueron tan bien recibidas que la ficción se convirtió en realidad y Asimov ha sido referenciado en numerosos artículos científicos, películas y libros [2].

Es evidente la gran evolución de la robótica con el paso del tiempo. Este desarrollo de la tecnología incluyendo poderosas computadoras electrónicas, actuadores de control retroalimentados, transmisiones de potencia a través de engranajes y tecnología de sensores, han contribuido a que los autómatas sean más flexibles y lleguen a desempeñar tareas dentro de la industria [1].

Dentro de esta evolución se pueden dividir en cinco grandes generaciones:

- **Primera generación**: conocidos como los robots manipuladores. Están diseñados para repetir una serie de tareas programadas previamente ejecutadas de forma secuencial. Cuentan con un sencillo sistema de control en lazo abierto, es decir, no reciben ninguna información o retroalimentación sobre el entorno. Son útiles para las aplicaciones industriales de tomar y colocar, pero están limitados a un número pequeño de movimientos. En la Figura 1 puede verse un ejemplo de este tipo.

Figura 1. Robot primera generación.

- **Segunda generación**: también llamados robots de aprendizaje. Esta clase de robots son capaces de memorizar y repetir una serie de movimientos que realiza un operador humano. De esta forma, el robot es capaz de realizar actividades que suponen un riesgo para el ser humano sin que este acabe perjudicado. Estos movimientos se almacenan en un disco o cinta mecánica. Este grupo de robots sí que tienen en cuenta el entorno adquiriendo esa información a través de sensores para así poder adaptar su actuación a las mismas, lo que viene siendo lo mismo, dispone de un sistema de control de lazo cerrado. En la Figura 2 puede verse un ejemplo de este tipo.

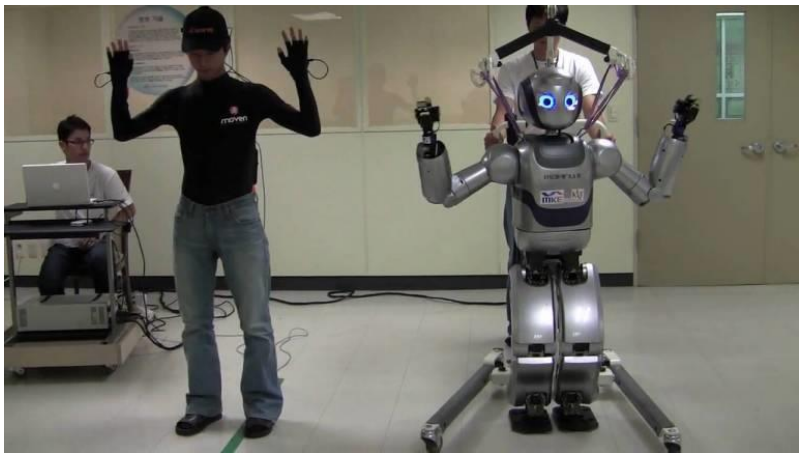


Figura 2. Robot segunda generación.

- **Tercera generación**: también reciben el nombre de robots con control sensorizado. Con esta generación se inicia la era de los robots inteligentes y empiezan a aparecer los lenguajes de programación para escribir los programas de control. Se utilizan computadoras para su estrategia de control llamada ciclo cerrado. Llegan a tener conocimiento del ambiente local, a través de sensores, para poder medirlo y así modificar su estrategia. Para realizar los movimientos necesarios, se ejecutan las órdenes de un programa y se envían a través de una computadora a un manipulador. En la Figura 3 puede verse un ejemplo de este tipo.

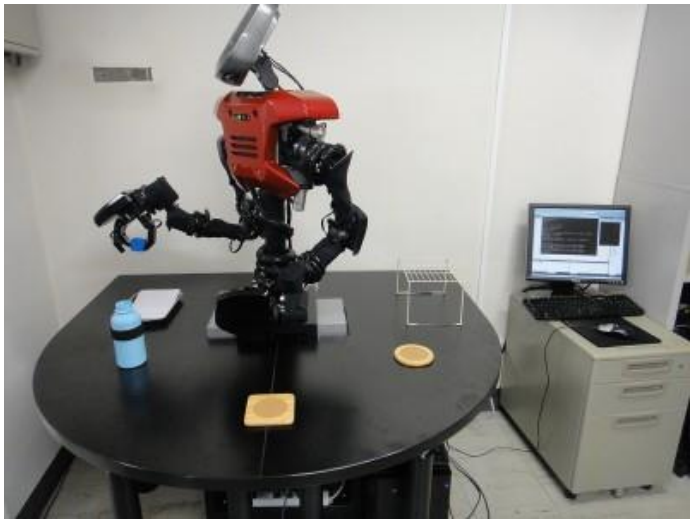


Figura 3. Robot tercera generación.

- **Cuarta generación**: llamados robots inteligentes. Son caracterizados por tener sensores más sofisticados para comprender sus acciones y el mundo que les rodea, los cuales mandan información al controlados y la analizan mediante estrategias complejas de control. Incorpora un concepto de "modelo del mundo" que les permite actuar ante situaciones determinadas. Para adaptarse y aprender de su entorno utilizan el "conocimiento difuso", "redes neuronales" y otros métodos de análisis y obtención de datos. En la Figura 4 puede verse un ejemplo de este tipo.

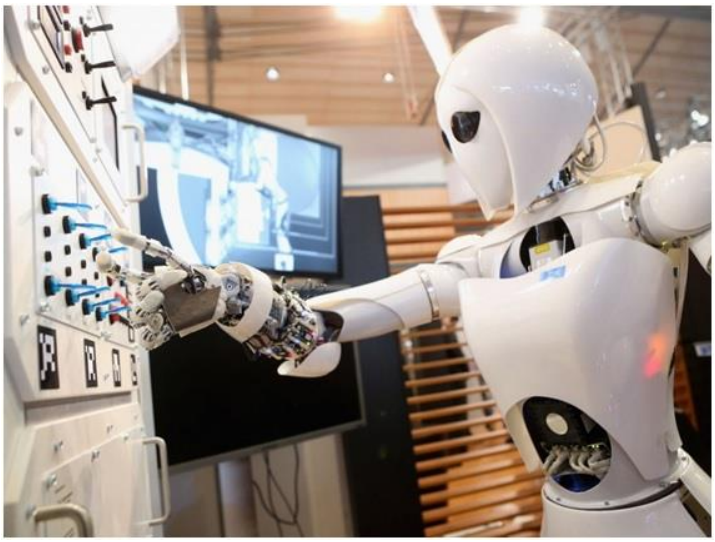


Figura 4. Robot cuarta generación.

- **Quinta generación**: actualmente se encuentran en desarrollo. Se tratará de una nueva tecnología que incorporará al máximo la inteligencia artificial y utilizará modelos de conducta y una nueva arquitectura distribuida en módulos conductores denominada arquitectura de subsunción [3][4]. En la Figura 5 puede verse un ejemplo de este tipo.

Figura 5. Robot quinta generación.

A pesar de la clasificación que se acaba de mostrar, existen innumerables tipos de clasificación de robots como puede ser una clasificación por su nivel de inteligencia, su nivel de control o su nivel de lenguaje de programación.

2.2. Robots colaborativos

"Aparato y método para la interacción física y directa entre una persona y un dispositivo manipulador de carácter genérico controlado por un ordenador" primera definición de robot colaborativo en el año 1999. Esta definición hace referencia a lo que hoy se llamarían dispositivos de asistencia inteligentes.

Tras una larga colaboración entre la empresa *KUKA*, pionera en robótica, y el *Centro Aeroespacial Alemán*, en 2004 se comercializó el LBR 3 (Figura 6), el primer robot colaborativo ligero por fuerza motriz por parte de dicha empresa. Conforme fueron pasando los años se fueron perfeccionando sus capacidades de control del movimiento y surgieron dos versiones actualizadas de este mismo, comercializándose en 2008 y 2013.

Figura 6. Robot colaborativo LBR 3 de KUKA.

También en 2008, el fabricante danés *Universal Robots* comercializó el UR5 (Figura 7), el primer robot capaz de funcionar de forma segura junto a los trabajadores, eliminando la

necesidad de vallas o jaulas de seguridad. Gracias a este nuevo robot se inauguró oficialmente la era de los robots colaborativos flexibles, intuitivos y rentables [5].

Figura 7. Robot colaborativo UR5 de Universal Robots.

Como bien se ha dicho, estos robots, también conocidos como *cobots*, están diseñados para interactuar con humanos en un espacio de trabajo compartido sin necesidad de instalar vallas de seguridad. Están programados para detenerse en el momento que entran en contacto ya que disponen de sensores de fuerza y consumo, basta con que el contacto sea mínimo para que el cobot se detenga y deje de realizar la función para la que ha sido programado. Debido a esto se consigue una reducción de riesgos para los empleados y de los costes de implantar sistemas de seguridad a su alrededor por lo que se optimiza el espacio en las líneas de montaje.

En la industrial 4.0, la robótica colaborativa ha destacado por la gran capacidad del manejo de la interfaz de humano-robot. La programación de los cobots es muy sencilla y no requiere la intervención de un especialista. Su puesta en marcha se puede llevar a cabo en unas pocas horas a diferencia de los robots industriales donde es necesario una gran cantidad de tiempo. Además, se pueden reconfigurar para poder operar en diversos puntos de una línea de producción lo que proporciona una optimización en la productividad. Gracias a todos estos detalles, permite que los propios empleados de una fábrica sean capaces de recibir, instalar, programar y poner en marcha una instalación robótica [6].

Sin embargo, para que los robots colaborativos puedan trabajar cerca de las personas, cuentan con ciertas limitaciones como en la carga a manipular y la velocidad de los movimientos, pero a cambio cuentan con muchas otras ventajas que los hacen ser una opción interesante y que cada vez se utilicen más en las líneas de producción [7].

2.3. Visión artificial

La visión artificial forma parte de la inteligencia artificial concretamente en el sector encargado del reconocimiento y procesamiento de imágenes que permite a un robot o un agente analizar el entorno y comprender qué está pasando. Todo esto se consigue a través de una imagen o vídeo que es capturado por una cámara ya será la propia del robot o una externa.

El origen de la visión artificial industrial se remonta a los años 60 gracias a un prototipo automatizado basado en cámaras de visión y sistemas de procesamiento de las imágenes captadas. Conforme se han ido desarrollando las cámaras fotográficas y la obtención de imágenes desde la perspectiva científica, la evolución y aplicación de la visión artificial ha ido en aumento.

La finalidad de todo ello era tener acceso a cómo eran ciertas estructuras y al análisis de su contenido mediante el procesamiento de imágenes con ayuda de ordenadores y softwares.

Ya en la década de los 80 marca su hito con el desarrollo de la ingeniería informática y la creación de procesadores sofisticados y rápido que dieron lugar a microprocesadores capaces de captar, procesar y reproducir imágenes. A partir de aquí se empieza a investigar cómo captar imágenes de forma automatizada y reproducir las características visuales.

Con ello se consiguen unas actividades básicas como el seguimiento de un objeto, la localización y reconocimientos de éste, modelos tridimensionales, estimar y comparar medidas o crear modelos y patrones. Con el perfeccionamiento de estas actividades se ha conseguido que la visión artificial sea una de las tecnologías que marcan la diferencia en ciertas tareas esenciales en la producción industrial como los controles de calidad o detección de productos defectuosos llegando a ámbitos tan diversos como el sector químico, alimentario o sanitario.

Se puede hablar de diversas categorías destacando algunas características relevantes de cada una de ellas a pesar de que los límites entre productos aún están muy poco definidos. Entre ellos tenemos:

- **Sensores de visión**: sus tareas se limitan más a detectar resultados de paso o fallos dado que cuentan con limitaciones en la toma de decisiones. Son tecnología de visión más sofisticada que los tradicionales sensores fotoeléctricos.
- **Cámaras inteligentes y sistemas de visión integrados**: cuentan con aplicaciones muy variadas gracias a su potencia de cálculo, resolución de imagen y fácil instalación. Además, tiene la capacidad de almacenamiento y de conectar con otros sistemas automatizado gracias a su capacidad de procesamiento. Tanto el sensor, la memoria como el procesador se encuentran en el cabezal remoto para facilitar la conectividad y el manejo de los elementos.
- **Sistemas de visión avanzados**: tienen un hardware más sofisticado y complejo lo que mejora las funciones con el software y procesamiento de datos. Su aplicación está ideada para tecnologías y máquinas automatizadas de mayor complejidad, por lo que el sistema tiene que ser implementado durante la fabricación de la máquina [8].

2.4. Sensor de fuerza

Los sensores existen desde siempre, no es necesario mirar muy lejos porque el ser humano dispone de ellos en su propio cuerpo. La capacidad que tiene de diferenciar entre calor o frío, duro o blando, pesado o no, etc. es una de las características de los sensores, medir ciertas magnitudes físicas del exterior que nos rodea captando información de este, pero con el paso del tiempo se han ido necesitando magnitudes que fueran más exactas.

Los sensores son dispositivos que tienen la facultad de detectar movimientos, ruidos, presión, luces y cualquier tipo de elemento externo para convertirlo en señal eléctrica [9]. Éstos son elementos físicos que pertenecen a unos dispositivos llamados transductores (Figura 8). Los transductores son los encargados de transformar ese elemento externo, es decir, una variable física, en otra diferente [10].

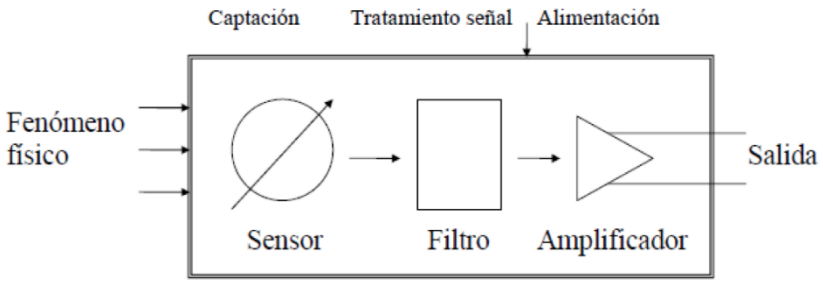


Figura 8. Diagrama explicativo de un transductor.

Para definir las características generales de los sensores, la forma habitual es a través de sus características estáticas y dinámicas debido a que estas mismas definen las características de funcionamiento, en régimen permanente y en régimen transitorio respectivamente. No obstante, también se encuentran otras definiciones que se centran en las propiedades del diseño, fiabilidad o factores medioambientales [11].

En cuanto a la clasificación de los sensores existen múltiples formas, se pueden clasificar por la variable a medir, por el principio físico en el que se basa su funcionamiento, por la tecnología en la que se basan, etc. Dentro de la clasificación de los sensores se encuentran los sensores empleados en robots donde se focalizará en los sensores de fuerza en el desarrollo del presente trabajo.

Los sensores de fuerza son los encargados de controlar la fuerza que ejerce el robot a la hora de manipular algún objeto. Son de gran utilidad ya que gracias a ellos se puede llegar a manipular varios tipos de materiales sin llegar a dañarlos. Dentro de este tipo de sensor se encuentran diferentes clases de las cuales se destacan algunos a continuación:

- **A compresión**: este tipo de sensor sólo permite la medida de fuerza o peso en una dirección siendo esta la de compresión o empuje [12]. Parala medida de la fuerza en con este tipo de sensor es necesario contar con una superficie de contacto, no puede garantizarse la medida en un punto [13]. En la Figura 9 se puede ver varios formatos de este tipo de sensores.

Figura 9. Sensores de fuerza a compresión.

- **A tracción o en tensión**: están diseñados para una medida unidireccional de la fuerza basados en el tiro que se produce en ambos extremos del transductor. En la Figura 10 se muestran varios modelos de este tipo de sensor.

Figura 10. Sensores de fuerza a tracción o en tensión.

- **Sensores de fuerza y par multiaxiales**: gracias a este tipo de sensores obtenemos la medición de forma simultánea de la fuerza y el par en cada una de las componentes x, y, z. Son sensores de fuerza triaxiales para cada una de sus magnitudes. A continuación, en la Figura 11, se muestran diferentes modelos de este sensor.

Figura 11. Sensores de fuerza y par multiaxiales.

- **Piezoeléctricos**: están especialmente indicados para variaciones muy súbitas de fuerza, es decir, para la medida de golpes o picos de fuerza donde ésta caría muy rápidamente en el tiempo. Esto es debido a su rápida deformación en su material interno produciendo un pico de tensión proporcional a esa deformación. Además, son considerados sensores muy dinámicos. En la Figura 12se pueden ver un ejemplo de éste.

Figura 12. Sensor de fuerza piezoeléctrico.

- **Dinamómetos**: este tipo de sensor se emplea para la medida de fuerza, bien sea a tracción, compresión o ambas, pero con la particularidad de integrar en el propio sistema el sensor de fuerza. En la actualidad, la gran mayoría de los dinamómetros son digitales (véase la Figura 13) [12].

Figura 13. Dinamómetro (sensor de fuerza).

CAPÍTULO 3

3. Materiales y dispositivos

En este apartado se realizará una breve descripción de las principales características de los elementos utilizados para llevará a cabo el proyecto.

3.1. Robot colaborativo

Para el desarrollo del este trabajo se ha utilizado el brazo robot UR3 perteneciente a la empresa *Universal Robots* que se encuentra en el edificio de *DISA* de la *Universidad Politécnica de Valencia* en el laboratorio de robótica (véase la Figura 14).

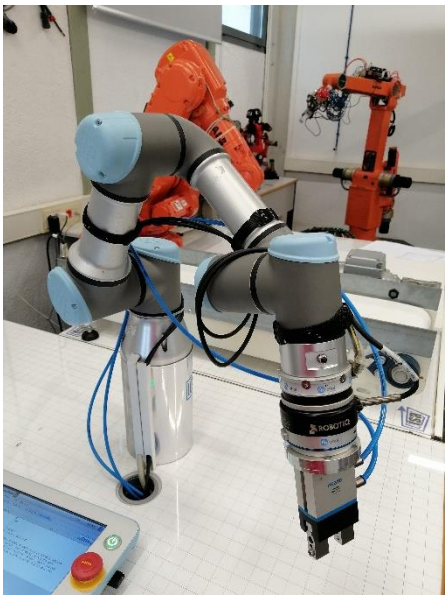


Figura 14. Robot colaborativo UR3

El robot tiene un peso de 11 kg, es el más pequeño de la familia de estos robots, fabricado de aluminio y plástico PP. La carga útil con la que puede llegar a trabajar es de 3 kg con un radio de acción de 500 mm.

Se trata de un robot de sobremesa con gran flexibilidad y precisión. Este tipo de robot ha sido diseñado para entornos de trabajo reducidos por lo que se pueden utilizar en espacios limitados. Gracias a todo esto es una de las mejores opciones para automatizar tareas y trabajos ligeros que requieran de gran precisión como sería en un proceso de molde por inyección, en el control de calidad de una línea de producción, en atornillado y muchas más.

A continuación, se expondrán brevemente los componentes del sistema robotizado de *Universal Robots*.

3.1.1. Brazo robot

El brazo del robot dispone de 6 articulaciones de tipo rotatorio, lo que permite tener 6 grados de libertad:

- **Base**: es la parte del robot que está montada en una superficie. Cuenta con 360 º de acción y una velocidad máxima de 180 º/s.
- **Hombro**: al igual que la base, tiene una acción de 360 º y una velocidad máxima de 180 º/s.
- **Codo**: de la misma forma que las dos articulaciones anteriores, 360 º de acción y velocidad máxima de 180 º/s.
- **Muñeca 1**: sigue teniendo los mismos grados de acción que los anteriores, 360, pero aquí ya se tiene una velocidad máxima de 360 º/s.
- **Muñeca 2**: de la misma forma que la muñeca 1 cuanta con 360 º de acción y 360 º/s de velocidad máxima.
- **Muñeca 3**: esta muñeca, a diferencia de las demás, cuenta con un radio de acción infinito, pero sigue manteniendo una velocidad máxima de 360 º/s.

Cabe destacar que los modelos UR5, UR10 y UR16 no disponen de la característica final de la muñeca 3, lo que convierte a este modelo idóneo para tareas de atornillado.

Para diferenciar las partes del robot véase la Figura 15.

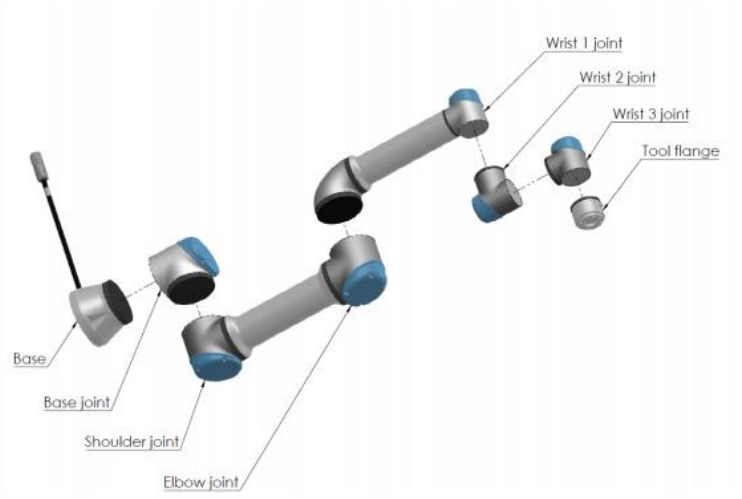


Figura 15. Partes del robot UR3.

Cada movimiento giratorio en las diferentes articulaciones es independiente del anterior. Esto proporciona que el número de articulaciones coincida con el número de grados de libertad (GDL), es decir, 6. Gracias a esto, se dispone de 6 parámetros con los que se puede posicionar (x,y,z) y orientar (α,β,γ) la herramienta u objeto a manipular de cualquier manera en el espacio.

3.1.2. Caja de control

La caja de control es la unidad de control donde se encuentran el tablero de control de seguridad, la placa madre y el USB, es decir, contiene los puertos e interfaces de comunicación que nos permite la programación y comunicación con dispositivos externos (Figura 16).

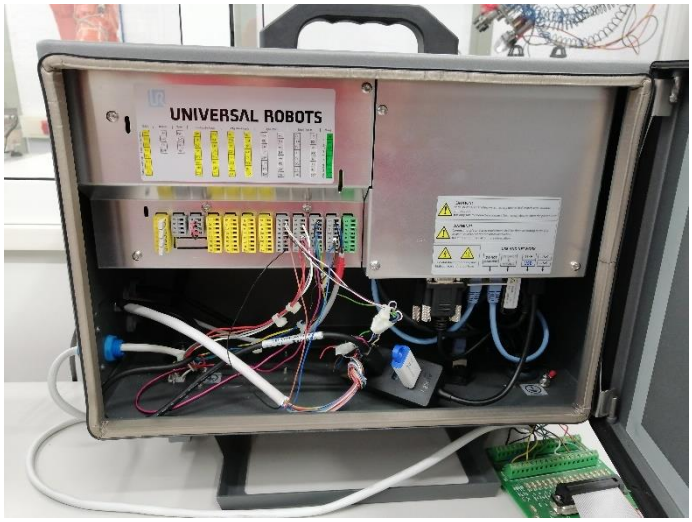


Figura 16. Caja de control abierta.

Tiene un peso de 15 kg fabricado en acero. Cuenta con unas dimensiones de 475 × 423 × 268 mm (an. × al. × la.). Está alimentado por una fuente de 100 – 240 V por corriente alterna a 50 – 60 Hz.

El tablero de control de seguridad maneja todas las entradas y salidas del robot, así como la conectividad de periféricos al robot siendo estos dispositivos de seguridad, sensores, interfaz de la máquina y pulsadores. Todo ello se encuentra distribuido de la siguiente forma:

- **Amarillos**: señales de seguridad.
- **Grises**: entradas y salidas digitales de propósito general. Está formado por 16 entradas y 16 salidas.
- **Verdes**: entradas y salidas analógicas. Formado por 2 entras y 2 salidas.

La placa madre es un microordenador del UR3, encargado de trabajar con los datos para ejecutar las instrucciones programadas. Además, dispone de comunicación mediante TCP/IP 100 Mbit, Modbus TCP, Profinet y EthernetIP.

Finalmente, la memoria USB que se encuentra conectada a la placa madre, contiene todos los softwares necesarios para poder llevar a cabo los programas deseados con el robot. Entre ellos se destacan la interfaz de programación PolyScope y los programas guardados.

3.1.3. Consola de programación

La consola de programación (Figura 17) es un dispositivo con pantalla táctil de 12 ’’ portátil, con interfaz gráfica de usuario (GUI) de programación PolyScope, que permite encender y apagar el robot, programarlo de forma sencilla y rápida para tareas específicas y además controlar las señales de dispositivos periféricos como por ejemplo una pinza.

Figura 17. Consola táctil UR3.

En la parte delantera, se encuentra la pantalla táctil, el botón de encendido y el botón de emergencia (botón rojo). Mientras que, por la parte trasera, se dispone de un botón llamado botón de movimiento libre que permite mover el robot de forma manual al ser presionado. Y, en el lateral derecho, cuenta con una entrada USB que, en el desarrollo de este proyecto, se ha utilizado para cargar y guardar los programas realizados [14].

Los aspectos de programación se tratarán más adelante en el apartado 5.

3.2. Sensor de visión

El equipo de visión que se ha elegido está diseñado para trabajar con los robots colaborativos de *Universal Robots* (Figura 18. Wrist Camera en la muñeca del robot.Figura 18) por ello, son equipos considerados de “*plug-and-play*”, por su fácil configuración e instalación [15].

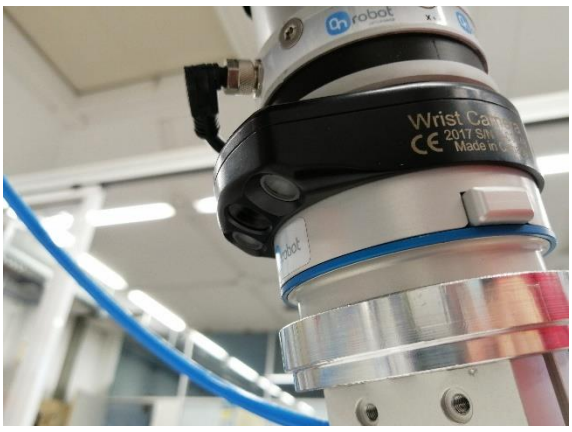


Figura 18. Wrist Camera en la muñeca del robot.

Existen diversos modelos según el robot en el que se vaya a instalar. En este caso, como el modelo del robot escogido es el UR3, el modelo escogido para el equipo de visión es el correspondiente a este.

3.2.1. Hardware de la cámara

Para el montaje de esta en el robot, cuenta con 4 orificios para ser atornillado a la muñeca de este directamente.

Tiene un peso de 160 g y soporta como carga máxima 10 kg o una fuerza de 40 Nm. Se alimenta a 24 V de DC mediante un cable flexible de 10 m que va conectado a la caja de control de robot.

Utiliza un sensor con tecnología CMOS para la captación de imágenes. Las características son las siguientes:

- Resolución: de 0.3 a 5 Mpx.
- Fotogramas por segundo: de 2 a 30 FPS.
- Mínimos campo de visión (cm): 10 × 7.5
- Máximo campo de visión (cm): 36 × 27

A cada lado de la lente de la cámara se encuentran dos luces compuestas por tres leds en cada una (ver Figura 19). La finalidad de esto es proporcionar una luz blanca difusa para cada vez que captura una imagen con opción de desactivarla en el caso necesario.

Figura 19. Hardware de la Wrist Camera.

Como se muestra en la imagen anterior, el equipo de visión cuenta con pines que permiten la conexión con la pinza "*2-Fingers Gripper*" del propio fabricante. En el caso de este proyecto no se utilizarán puesto que se usará la pinza que dispone el laboratorio [16].

3.2.2. Software de la cámara

Para la programación y trabajo de la cámara no son necesarias herramientas extra como un ordenador ya que se puede realizar a través de la propia consola de programación del brazo

robot. Todo esto es gracias a que el software incorporado en el sistema de visión está diseñado para ser usado con robots de *Universal Robots*.

Dispone de una interfaz sencilla para que gente sin grandes conocimientos en el campo de la visión sea capaz de realizar tareas simples de *pick and place*.

Este estilo de equipo de visión está hecho para trabajar en planos 2D por lo tanto, una vez se haya definido el espacio de trabajo deseado, será capaz de reconocer la posición (x, y) y la orientación sobre el eje z del objeto enseñado previamente.

Los aspectos de calibración y programación de la cámara se tratarán más adelante en el apartado 5.2.

3.3. Sensor de fuerza

El modelo escogido trata de un sensor de fuerza par de 6 ejes de *OnRobot*. Los sensores de esta marca están diseñados para adaptarse a la mayoría de los brazos robóticos industriales de hoy en día. Estos modelos disponen de mejoras técnicas para facilitar y agilizar la instalación y el manejo de los sensores [17].

3.3.1. Hardware del sensor

El sensor está compuesto por un cuerpo del sensor, un adaptados y un tapón de sobrecarga. Dentro del cuerpo del sensor se encuentran el conector del cable del sensor, el soporte para los cables, la protección guardapolvo, el número de serie y los marcadores de los marcos de referencia.

Para colocarlo en el robot es necesario un adaptador para así poder colocarlo en el borde de la herramienta del robot mientras que la herramienta se encuentra sujeta al cuerpo del sensor directamente. Ver la Figura 20 para entender las partes del sensor.

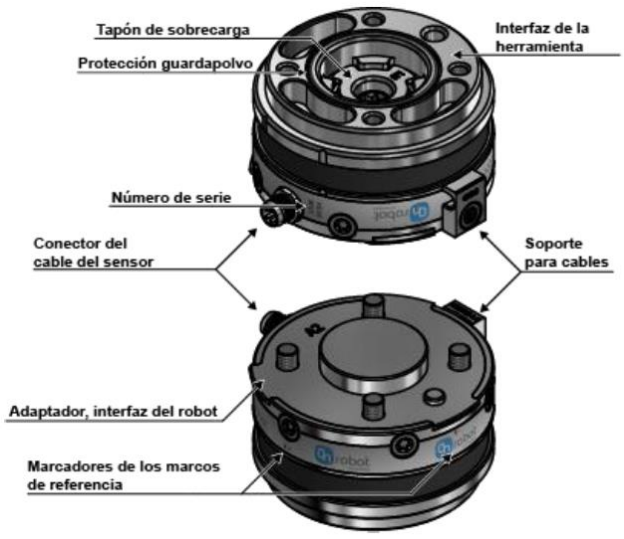


Figura 20. Hardware del sensor de fuerza.

Tiene un peso con placas adaptadoras integradas de 347 g, sus dimensiones son de 50 × 71 × 93 mm (al. × an. × la.) y se alimenta a 7 – 24 V. Cuenta con una capacidad nominal de 200 Nm en la fuerza en x, y, z (Fxyz) mientras que en la torsión en x e y (Txy) 10 Nm y 6.5 Nm en z (Tz).

Durante la carga de un solo eje, el sensor puede ponerse en funcionamiento hasta alcanzar su capacidad nominal. Si se supera dicha capacidad nominal, la lectura será imprecisa e inválida. Además, hay que tener en cuenta la señal de ruido típica, que viene definida como la desviación típica estándar de una señal sin carga de un segundo típica, donde es de 0.035 Nm para Fxy, 0.15 en Fz, 0.002 en Txy y 0.001 en Tz [18].

3.3.2. Software del sensor

Este sensor es capaz de proporcionar mediciones precisas de fuerza par a lo largo de los 6 ejes. Estas mediciones que ofrece están formadas por las coordenadas en los 3 ejes cartesianos de la fuerza y del par que se ejerce sobre el sensor. Esto es una ventaja a la hora de llevar a cabo tareas difíciles como de montaje, pulido, lijado o desbarbado ya que proporciona un control preciso. A parte de que estas sean las aplicaciones más comunes para este sensor, también puede utilizarse para el aprendizaje y detección de choques.

Otra de las características con las que cuenta es que mantiene de forma constante la fuerza y la velocidad durante todo un proceso, lo que ofrece una optimización y productividad sin comprometer la calidad en una línea de producción.

Cuenta con un software preintegrado fácil de instalar y programar, lo que permite que los mismos operarios sin formación técnica puedan configurarlo en poco tiempo [19].

CAPÍTULO 4

4. Diseño de piezas

Este capítulo contiene toda la información relacionada con las piezas utilizadas en la aplicación. Primero se incluirá una breve descripción del programa utilizado para el diseño de éstas. A continuación, el deseño de cada una y, por último, los resultados que se han obtenido en la fabricación de las piezas.

4.1. AutoCAD

Para la realización del diseño de las diferentes piezas se utilizó el programa *AutoCAD* desarrollado por *Autodesk* a través de la plataforma *PoliLabs* que proporciona la universidad. Se escogió este programa por las facilidades que presenta ya que hace posible el dibujo digital de planos o la recreación de imágenes en 3D. Además, este programa ya era conocido previamente ya que se enseña durante la carrera.

Se trata de un programa bastante versátil lo que ha hecho que se convierta en un estándar en el diseño por ordenador puesto que se puede ampliar el programa a base de programación.

Como se ha comentado, al disponer de la recreación en 3D, también cuenta con la ventaja de tener gran compatibilidad con el software de impresión 3D permitiendo exportar los diseños en diferentes formatos, esto es lo que se requiere para este trabajo ya que la intención es fabricar las diferentes piezas. En la Figura 21 se puede observar la ventana inicial del programa.

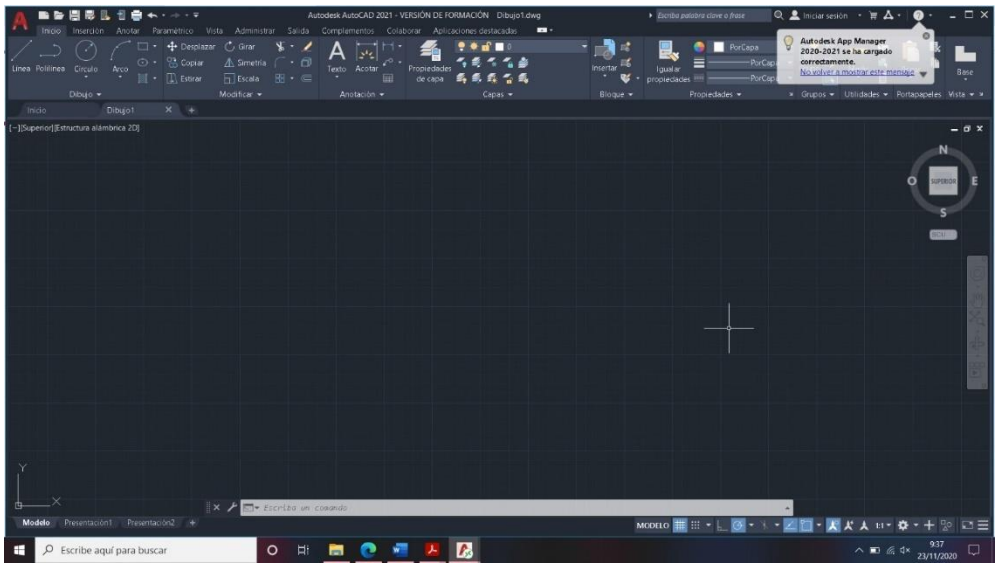


Figura 21. Ventana inicial AutoCAD.

4.2. Piezas diseñadas

Para el desarrollo de este trabajo se realizó el diseño de 8 piezas, 4 corresponden a círculos y otras 4 a cruces, las cuales son necesarias para el juego. A la hora de jugar se decidió que el robot dibujara las piezas que les correspondía ya que, en el caso de los círculos, al tratarse de objetos curvos, la pinza no lograba sujetarlos de forma correcta y acababan cayéndose. A continuación, en las Figura 23 y Figura 22, se muestra una imagen 3D de ellas para poder identificarlas y relacionarlas.

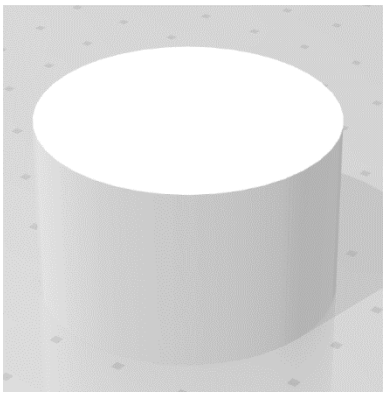


Figura 23. Modelo 3D ficha círculo.

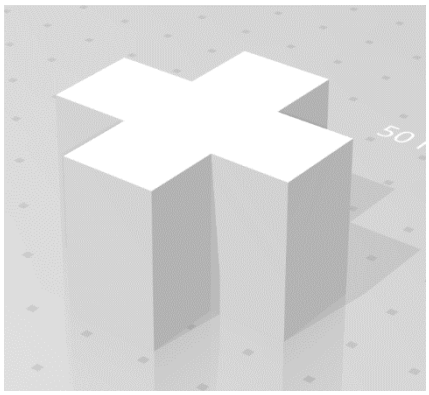


Figura 22. Modelo 3D ficha cruz.

4.3. Impresión 3D

Una vez terminado el modelaje de las piezas, se procedió a la impresión de éstas. Para la impresión solo fue necesario el diseño de una de ellas, es decir, un círculo y una cruz, ya que a la hora de imprimir se le determinaba que imprimiera 4 de cada. Dada a la disponibilidad de los materiales que se tenían, el color a imprimir fue el blanco. Esto supuso algunas complicaciones ya que el fondo también sería de color claro, por lo que se optó por pintarles las caras visibles de color negro y así facilitaría la detección del objeto. A continuación, en las Figura 24 y Figura 25, se muestran las piezas ya terminadas listas para ser utilizadas en la aplicación.

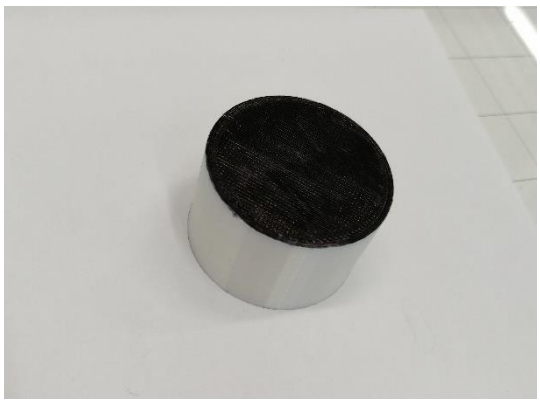


Figura 24. Ficha círculo.

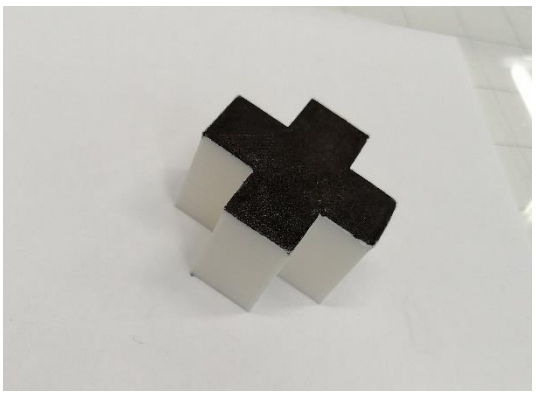


Figura 25. Ficha cruz.

CAPÍTULO 5

5. Metodología empleada

A lo largo de este capítulo se mostrarán y explicará el procedimiento seguido para el diseño de esta aplicación, así como los problemas más relevantes que han ido surgiendo.

5.1. Programación en PolyScope

El conjunto de indicaciones de acciones de forma secuencial que se llevan a cabo durante una tarea específica se puede considerar como la programación de un robot.

El UR3 utiliza la programación por guiado. Esta consiste en guardar las posiciones que tiene que adoptar durante una tarea para su repetición posteriormente. Para el movimiento del robot se puede dar lugar mediante dos formas: a través de las flechas de movimiento que dispone la consola de programación (guiado activo) o de forma manual manteniendo el botón de movimiento libre (guiado pasivo).

A la hora de generar trayectorias, el robot realiza una interpolación de las posiciones definidas (también llamadas puntos de paso) con las características que se hayan especificado como el tipo de trayectoria, la velocidad, aceleración, etc. pudiendo ser modificadas desde la propia consola de programación.

A continuación, se hace una breve explicación de las características de PolyScope más relevantes que se han usado durante este proyecto.

5.1.1. Interfaz de programación

Una vez se haya encendido el robot, se debe seleccionar la opción “Programar robot” (Figura 26) y a continuación “Programa nuevo” (Figura 27) lo que nos llevará a la interfaz principal de programación.

Figura 26. Pantalla principal PolyScope.

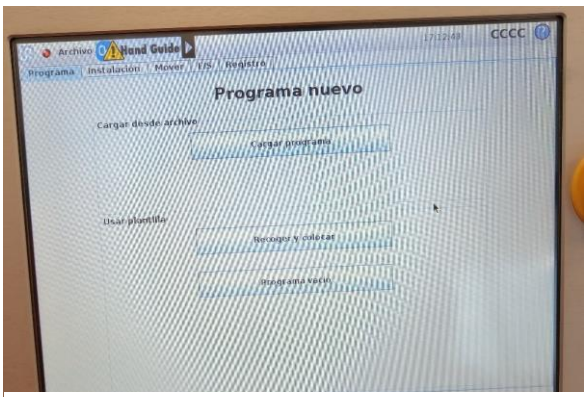


Figura 27. Pantalla secundaria para programar.

Una vez listo para programar, a la izquierda se encuentra lo conocido como árbol de programas principal. Aquí es donde se encuentran todos los comandos que se llevarán a cabo durante la ejecución del programa.

En la parte derecha, se sitúan 4 subpestañas (“Comando”, “Gráficos”, “Estructura”, “Variables”) que permiten la modificación del programa, así como ver información de este.

En el apartado “Estructura” se muestran los comandos que se pueden insertar en el árbol de programas y las opciones para editar de forma sencilla el árbol como son mover, copiar, pegar, cortar y eliminar. Dentro de la parte de insertar se pueden diferenciar diferentes pestañas que nos muestran diferentes comandos diferenciados en básicos, avanzados, asistentes y “URCaps”.

Por otro lado, en “Gráficos” se puede observar las trayectorias que realiza el robot durante la ejecución del programa, así como los puntos de paso. Es una buena opción para ejecutar el programa con la opción de simulación seleccionada, en la esquina inferior izquierda de la pantalla principal, así muestra una idea de lo que realizaría el robot.

Y, por último, cuenta con el apartado “Variables” donde muestra de forma ordenada según el nombre y valor todas las variables definidas en el programa.

5.1.2. Comando Mover

Con este comando es posible controlar las trayectorias que genera el robot a través de los puntos de paso que se han predeterminado especificando la velocidad en mm/s, la aceleración en mm/s^2 y el tipo de movimiento deseado. Los diferentes tipos entre los que se puede mover el robot son:

- MoveJ: es un movimiento no linear calculado en el espacio de articulaciones. Este tipo de movimiento da ligar a una trayectoria curva de la herramienta. Además, es la opción recomendable cuando no importa la trayectoria de la herramienta y se encuentra en un espacio de trabajo libre. En la Figura 28 se muestra un ejemplo de trayectoria utilizando el comando MoveJ.

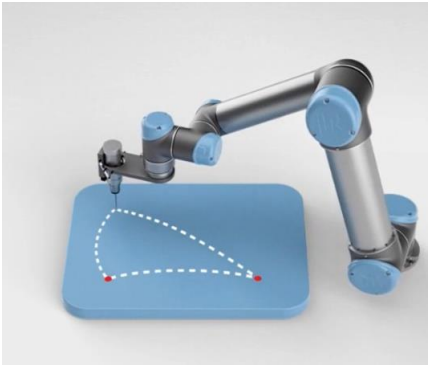


Figura 28. Ejemplo de trayectoria MoveJ.

- MoveL: es un movimiento lineal entre los puntos de paso calculado en el espacio cartesiano, esto supone que el robot debe hacer movimiento más complicados para mantener la herramienta en una trayectoria recta. Esto produce que el movimiento sea más lento que el anterior. En la Figura 29 se muestra un ejemplo de trayectoria utilizando el comando MoveL.

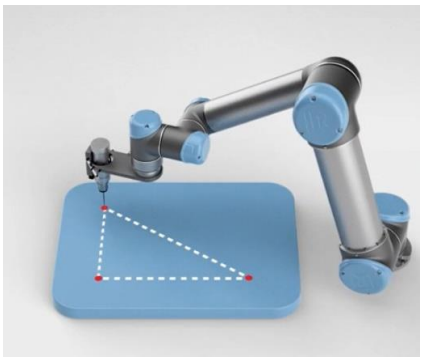


Figura 29. Ejemplo de trayectoria MoveL.

- MoveP: es un movimiento lineal que sea caracteriza por mantener la velocidad constante lo que da lugar a unas transiciones circulares entre los puntos de paso. Este movimiento es recomendado para tareas en las que se deba mantener una distribución regular como es en el caso de una soldadura o en la dispensación de pegamento. En la Figura 30 se muestra un ejemplo de trayectoria utilizando el comando MoveP.

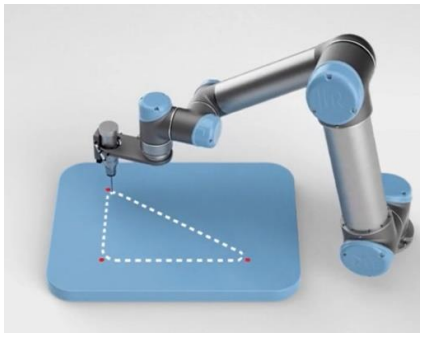


Figura 30. Ejemplo de trayectoria MoveP.

- CircleMove: es un comando que se le puede añadir a un moveP. Está formado por dos puntos de paso lo que permite movimientos circulares, estos dos puntos son conocidos como punto de vía y punto final. Tiene dos opciones de funcionar, interpolando la orientación de la herramienta con las dos posiciones definidas o manteniendo siempre la misma orientación. En la Figura 31 se muestra un ejemplo de trayectoria realizado con CircleMove.

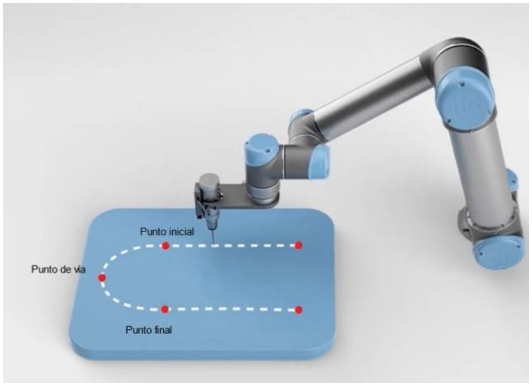


Figura 31. Ejemplo de trayectoria CircleMove.

Para la programación del movimiento del robot con aplicaciones de visión artificial usando la cámara de *Robotiq*, cabe resaltar una característica de MoveL. En este movimiento se dispone de la opción de programar en función a un punto elegido, cambiando la referencia respecto la base. Al elegir como punto el del "Snapshot position" permitirá que, al programar un *pick and place* de una pieza en esa posición, sirva para todas las posiciones en las que se encuentre la pieza detectada.

5.1.3. Comando script

Este comando se encuentra dentro de los comandos tipo avanzado. Gracias a este comando se tiene un acceso directo al lenguaje script que está ejecutando en tiempo real el controlador del robot. En este proyecto se ha utilizado con la finalidad de simplificar el árbol de programas ya que eran muchos comandos. Además, ha servido de ayuda a la hora de controlar la fuerza sin necesidad de repetir grandes comandos (se explica más detalladamente en el punto 6.3).

Cuenta con la opción “Archivo” situado en el desplegable de la esquina superior derecha (véase Figura 32) con el cual se pueden crear y modificar archivos con extensión “.script”. Esto es de gran utilidad ya que facilita el tiempo de programación pudiendo hacerlo desde un ordenador a parte y cargarlo con un USB.

Figura 32. Pantalla de comando script.

Dentro de los diferentes scripts que forman el programa, se destacan diferentes comandos y funciones que se han utilizado:

- **def *programa*( ):** **end** → declara funciones. Se utiliza para realizar una serie de comandos cuando esta es llamada desde el programa. No es necesario introducirle un parámetro, pero en caso de necesitarlo se pondría entre los paréntesis. Una de las ventajas es la función *return* donde, declarando una variable en el programa principal y llamando a la función, obtenemos el valor de una de las variables que tienen lugar dentro del script.
- **thread *programa*( ):** **end** → declara funciones generales. No permite introducir variables y, de la misma forma, no devuelve valores de estas. Esta función puede ser declarada dentro de otra declarando una variable con el comando *run* seguido del nombre de la función. Para terminar la ejecución de ésta se utiliza *kill* seguido del nombre de la variable con la que se ha puesto en marcha.
- **force_mode(*task_frame, selection_vector, wrench, type, limits*)** → se utiliza para controlar el robot mediante la fuerza. Para acabar con la ejecución de este programa se utiliza el comando *end_force_mode( )*. Este comando hace referencia al comando avanzado de fuerza que se encuentra en *PolyScope* (Figura 33).

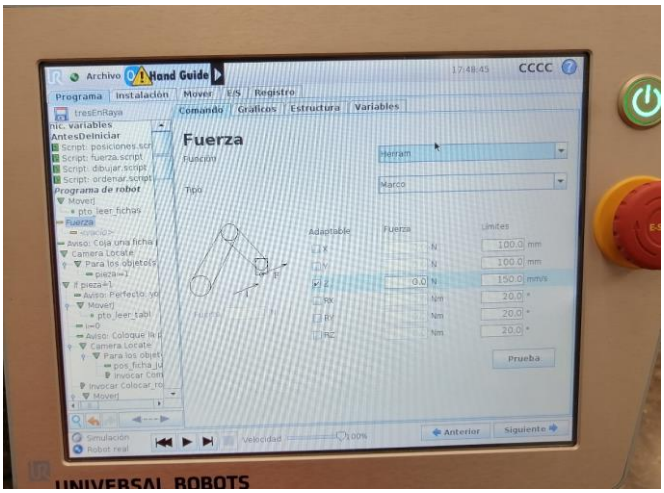


Figura 33. Pantalla de comando Fuerza.

- **pose_add(*position1, position2*)** → suma dos vectores de posición.
- **movec(*pose_via, pose_to, a, v, r, mode*)** → hace referencia al *CircleMove* explicado anteriormente. Con *mode* se indica mediante 0 o 1 si interpolar la orientación o dejarla fija respectivamente.

5.1.4. Comando Subprograma

Un subprograma se encuentra fuera de lo que es el programa principal. Este puede albergar partes de un programa que vaya a necesitarse en diferentes momentos. Si no es invocado dentro del programa principal este no se ejecuta. Además, cuenta con la opción de guardar o cargar un programa tratándose así de un archivo independiente (Figura 34).

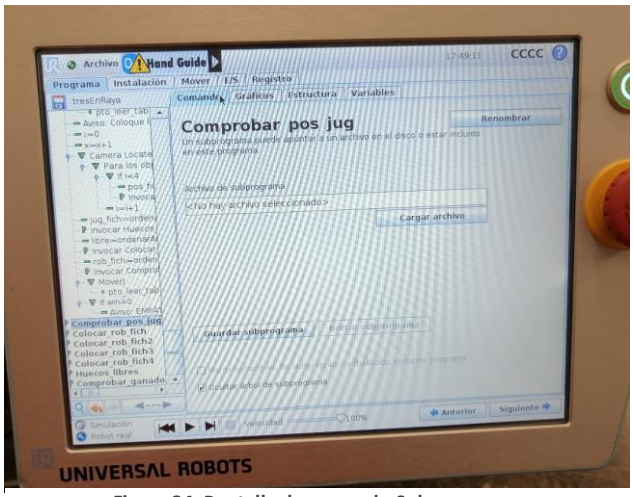


Figura 34. Pantalla de comando Subprograma.

Gracias a este comando también se ha conseguido una reducción de líneas en el programa principal y ha servido de herramienta que ha facilitado la búsqueda de un apartado en el proyecto o de algún posible error.

5.1.5. Variables

Para el desarrollo de este proyecto han sido de gran utilidad el uso de diferentes variables. Para la creación de estas en PolyScope primero debemos abrir la dicha de "Instalación" y seleccionar la opción "Variables" (Figura 35).

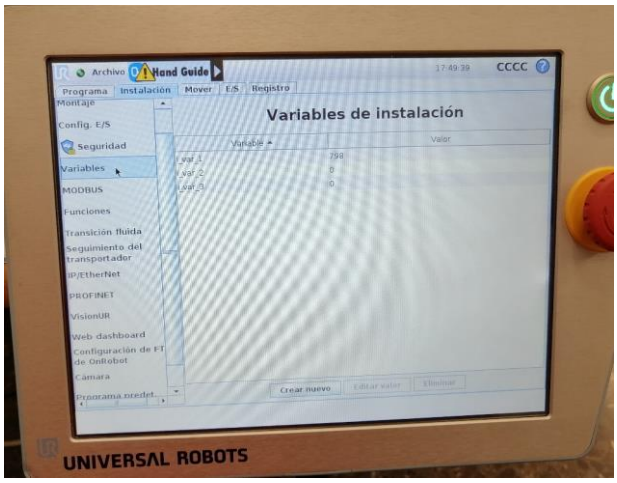


Figura 35. Pantalla de declaración de variables.

Una vez dentro se puede crear, editar y eliminar variables. Las variables definidas en este proyecto y su función son las siguientes:

- **pieza**: toma el valor de 0 o 1 lo que permite diferenciar si el robot juega con círculos o con cruces respectivamente.
- **i**: variable genérica que permite saber qué número de pieza se ha detectado.
- **pos_ficha_jug**: indica el vector posición de la pieza detectada.
- **x**: variable genérica que permite saber qué pieza corresponde al robot colocar.
- **jug_fich**: array donde se almacenan las posiciones en el tablero de las fichas del usuario.
- **libre**: array donde se almacenan las posiciones libres del tablero.
- **rob_fich**: array donde se almacenan las posiciones en el tablero de las fichas del robot.
- **win**: toma el valor de 0, 1 o 2 lo que permite diferenciar entre empate, gana el usuario o gana el robot respectivamente.

Además, a parte de estas, dentro de los scripts se pueden encontrar más variables consideradas locales, en su mayoría, ya que son únicamente utilizadas para el script determinado en el que se encuentren. Estas son las variables y su función:

- **fichas**: array formado por los arrays "jug_fich" y "rob_fich".
- **dim_fichas**: indica el tamaño del array "fichas".
- **tablero**: array formado del 1 al 9, ambos inclusive, que hace referencia a los números asignados a los huecos del tablero.
- **dim_tablero**: indica el tamaño del array "tablero".

- **esq1**, **esq2**, **esq3**, **esq4**: indican las posiciones de los 4 puntos para dibujar la cruz o el círculo.
- **dist1**, **dist2**, **dist3**, **dist4**: indican la posición de seguridad antes de acercarse a los puntos establecidos por “esq1”, “esq2”, “esq3” y “esq4”.
- **thread_force**: llama a la función de fuerza definida en “fuerza.script”.
- **pos**: valor aleatorio que se utiliza para saber en qué posición colocar la primera ficha por parte del robot.
- **pos1**, **pos2**, **pos3**, **pos4**, **pos5**, **pos6**, **pos7**, **pos8**, **pos9**: indican las posiciones de las casillas del tablero.
- **dim_array**: indica la dimensión del array introducido en la función “ordenarArray” de “ordenar.script”.

5.1.6. Cargar/Guardar un programa

Este apartado cabe considerarlo importante ya que a veces se han producido errores al ejecutar el programa por no haberlo guardado correctamente. Por ello, es conveniente tener en cuenta las siguientes consideraciones.

En primer lugar, para guardar y cargar el programa, se debe seleccionar “Archivo”, situado en la esquina superior izquierda, donde se despliega una lista que contiene estas opciones (Figura 36).

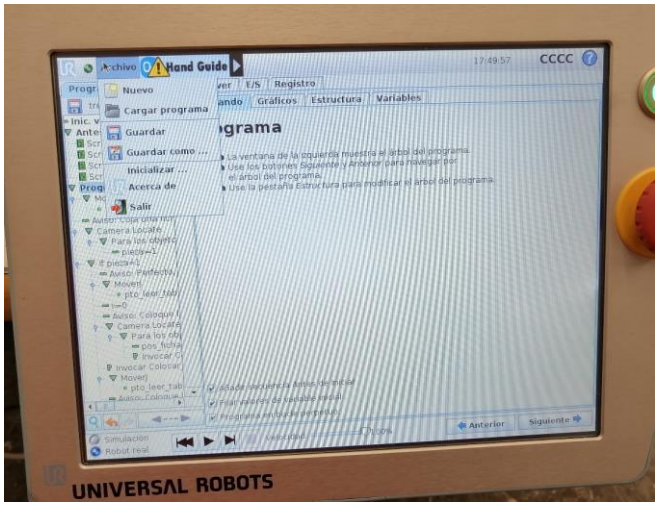


Figura 36. Despliegue opciones para guardar.

A la hora de guardar es recomendable hacer un directorio con cada programa diferente que se vaya a tener para así contar con todos los archivos necesarios en el mismo directorio.

Por otro lado, es aconsejable utilizar la opción “Guardar como” antes que “Guardar” porque esta segunda opción lo que hace es sobrescribir los ficheros dando lugar a que no se actualicen bien causando así problemas. La primera opción proporciona que se guardan todos los archivos generados hasta el momento.

Al guardar se generan varios archivos, son los siguientes:

- **archivo.urp**: archivo manejado por PolyScope que contiene el código del programa a ejecutar.
- **archivo.installation**: archivo manejado por PolyScope que contiene las configuraciones de la instalación del robot (configuración de seguridad, PCH, variables, funciones, conexiones eléctricas, etc.).
- **archivo.variables**: archivo de visualización donde se pueden consultar las variables y su valor.
- **archivo.script**: archivo de visualización que contiene script en lenguaje Python del programa.
- **archivo.txt**: archivo de visualización que contiene la estructura del árbol de procesos.

5.2. Programa de visión

El programa de visión utilizado es el que incorpora el propio robot. A continuación, se detallarán los pasos seguidos para el *teaching* de las piezas y los problemas tratados durante esta etapa.

5.2.1. Posición de captura

Para poder realizar la calibración se le debe definir una posición al robot la cual toma de referencia para dicha calibración y para la captura de imágenes que se harán durante el desarrollo del programa para la búsqueda de las piezas. Dicho punto es conocido como "Snapshot position".

Para ello, desde la consola, se crea una función tipo punto desde el menú "Funciones" dentro de la pestaña "Instalación" y se verificará la casilla de "Variable" (Figura 37). En este caso, se crearán dos puntos, el primero recibirá el nombre de "pto_fichas" el cual será utilizado para saber con qué ficha juega el robot y, el segundo, recibe el nombre de "pto_tablero" que será la posición que tomará el robot para saber qué fichas hay colocadas en el tablero.

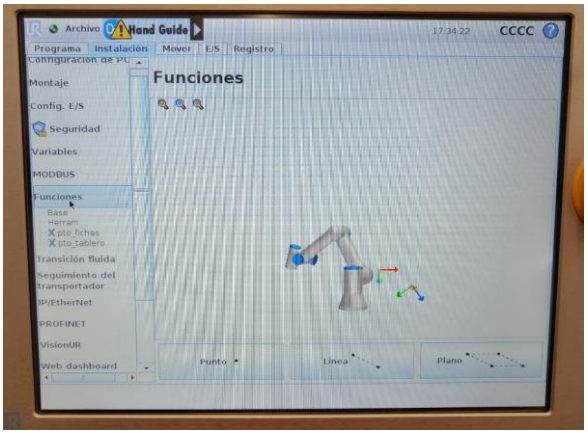


Figura 37. Pantalla declarar variable Snapshot Position.

Una vez están creadas las variables, es decir, se ha realizado lo anterior y se le ha enseñado la posición al robot, en la pestaña “Camera” de la misma venta de instalación, se selecciona la subpestaña de “Snapshot position” (véase en la Figura 38) y se elige la variable que se desea calibrar.

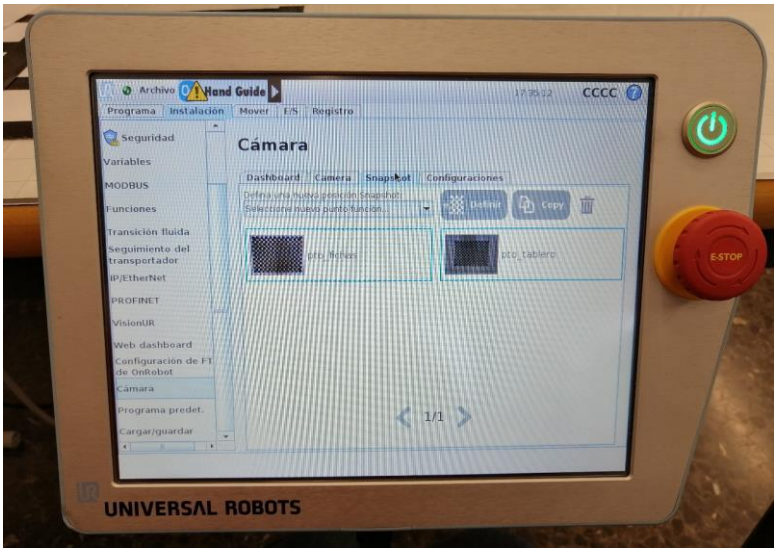


Figura 38. Pantalla elegir punto para definir Snapshot Position.

Tras elegir la opción “Define”, aparecerá la siguiente interfaz (Figura 39):

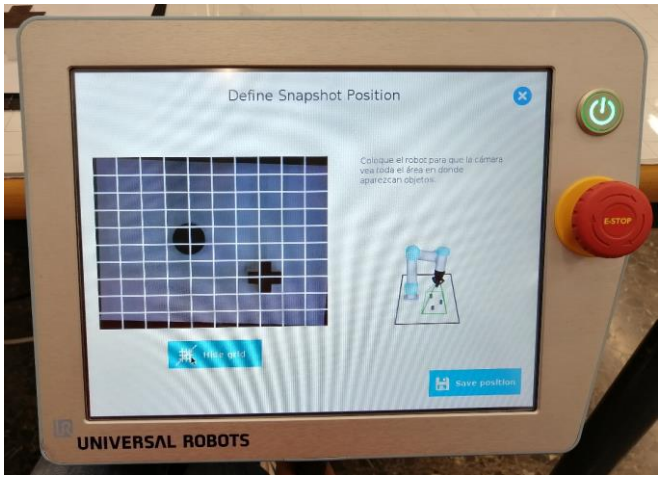


Figura 39. Pantalla definir Snapshot Position.

Hay que tener en cuenta activar el “Grid” ya que, para el buen reconocimiento de las piezas, la cámara debe estar situada de forma que el tamaño de cada una de las piezas sea, como mínimo, un cuadrado de la cuadrícula.

Una vez se ha fijado bien la posición deseada se selecciona “Save position” y procede con la calibración.

5.2.2. Calibración de la cámara

Este proceso es necesario porque establece una relación entre las coordenadas de un sistema de visión y el robot. Gracias a esto se consigue que el software de la cámara transforme las coordenadas recibidas en píxeles en coordenadas equivalentes al del robot. Además, una de las ventajas que muestra este tipo de cámara es que no se necesita establecer una relación entre el origen de coordenadas de la cámara y la base del robot, la posición ya es conocida al encontrarse instalada en el extremo de la muñeca del robot.

En este caso, la calibración de la cámara se hace de forma automática con la plantilla de calibración que proporciona el fabricante o también puede ser impresa desde la web oficial de *Robotiq*. La plantilla debe ser la correspondiente al modelo a utilizar y colocada en la zona de trabajo de tal forma que sea visible la cuadrícula tal y como se muestra en la Figura 40:

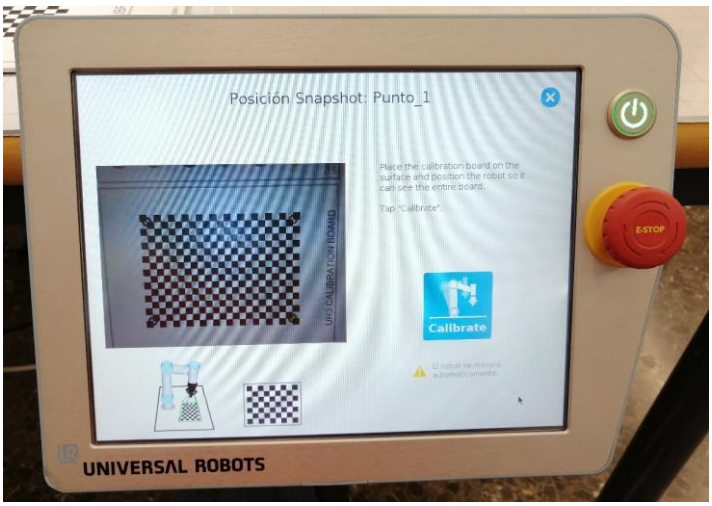


Figura 40. Pantalla de calibración de la cámara.

Este proceso de calibración se lleva a cabo en un tiempo reducido siendo este unos 10 minutos aproximadamente. Realiza, de forma automática, 27 fotos desde diferentes ángulos viendo que coge bien la cuadrícula y otras 9 más para validar la calibración (Figura 41 y Figura 42).

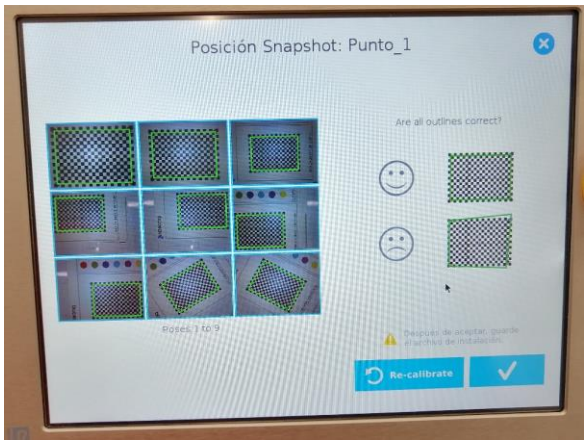


Figura 41. Pantalla de validación de calibración.

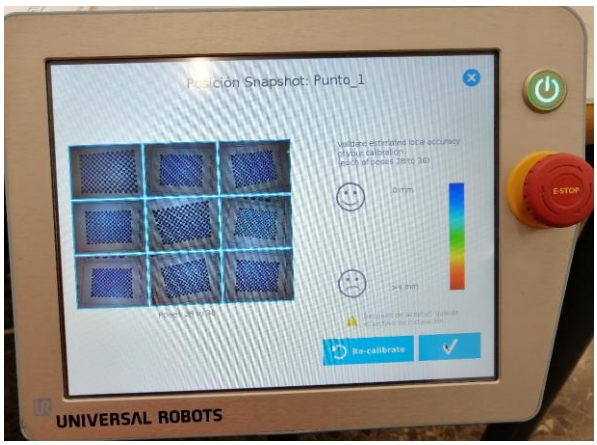


Figura 42. Pantalla de validación de precisión.

En caso de que el robot no pueda acceder a una posición o la calibración no resulte buena, se repetirá el proceso. Para saber si ha sido buena dicha calibración se revisan las imágenes tomadas viendo que coge bien la cuadrícula y, en la parte de precisión, no supere los 4 mm.

Este proceso se repetirá tantas veces como puntos de calibración definidos haya.

5.2.3. "Teaching" de las piezas

Para llevar a cabo este procedimiento se deben haber realizado los pasos anteriores de "Snapshot position" y la calibración. Para ello, se insertará una estructura árbol de programa del robot.

Dentro del menú "Programa" en "Estructura", se selecciona la pestaña "URCaps" donde se encuentra la opción de insertar "Camera locate" (Figura 43).

Figura 43. Pantalla URCaps.

Una vez se haya insertado, habrá que colocarse encima y saldrá la opción del "Teaching". Dentro dará la opción de llevarlo a cabo mediante dos métodos: método automático y método paramétrico.

El método automático construye un modelo basado en la toma de fotos y escaneado del objeto a aprender. Este método es aconsejable cuando se desea utiliza objetos con formas complejas e irregulares.

Por otro lado, el método paramétrico construye un modelo basado en parámetros 2D de figuras simples que se introducen como son círculos, cuadrados o rectángulos. Es aconsejable utilizarlo siempre que se pueda ya que es una opción más robusta debido a que la búsqueda de objetos la realiza directamente por la silueta elegida e ignora los espacios en blanco, como pueden ser los producidos por reflejos.

Después de varias pruebas, se optó por enseñar ambas piezas de forma automática ya que, como se ha comentado, a la hora de utilizar el método paramétrico con el caso de la pieza círculo, este no era bien reconocido en muchas ocasiones debido a los reflejos producidos por la propia luz de la cámara. Además, para la posición de captura se utiliza la misma que se determinó como punto de calibración.

Antes de realizar el "teaching" del objeto, es recomendable colocar un fondo de color que permita realzar al máximo posible las características de la pieza al tratarse de un método que construye el modelo a base de las imágenes que capta.

En primer lugar, se coloca el objeto de manera que sea completamente visible y, con la opción de detección automática, detectará el objeto a enseñar. En el caso de no ser detectado habría que seleccionar el área que contenga la pieza. Con esto hecho se realizará la primera foto haciendo un reconocimiento breve y habrá que hacer 3 más con orientación diferente de 90º en cada una (Figura 44). Una vez hechas todas las fotos se verificará si la detección de las características han sido la correcta.

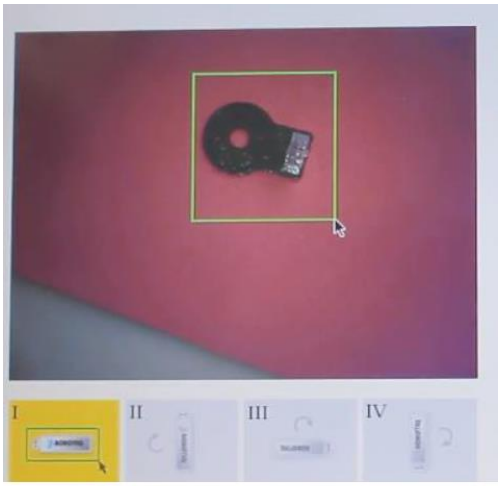


Figura 44. Ejemplo de aprendizaje de una pieza.

Posteriormente, se realiza un escaneo de la pieza mediante imágenes desde diferentes posiciones y ángulos. Esto proporcionará un modelo final que tendrá como referencia el programa.

Por último, tiene lugar el test donde se puede modificar la cantidad de piezas a detectar, el color de estas y el umbral de detección tanto por la forma como por el color (Figura 45). En este caso el umbral no se modificó ya que, tras varias pruebas, se comprobó que no era necesario cambiarlo por las formas tan diferentes que tienen las piezas. Sin embargo, sí que se fue modificando la cantidad de piezas a detectar en la posición "pto_tablero" ya que cada vez

que fuera leyendo encontraría más piezas puestas por el oponente que debería detectar para saber su posición en el tablero.

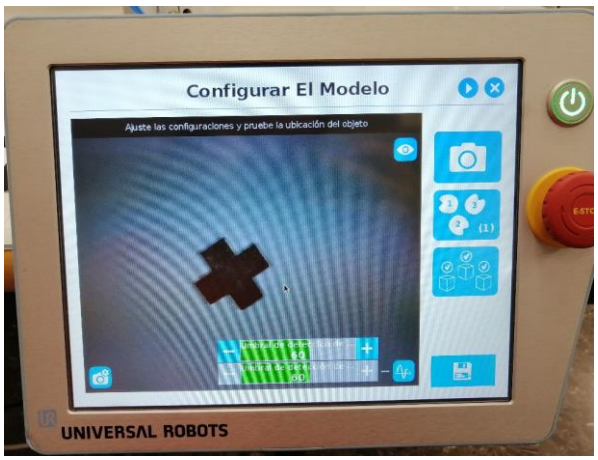


Figura 45. Pantalla test de detección de la ficha.

5.2.4. Problemas surgidos

Durante este proceso han ido surgiendo diferentes problemas que se han ido solucionando conforme se avanzaba con el proceso.

El primero de ellos fue determinar qué método se emplearía para el reconocimiento de las piezas ya que era la primera vez que se trabajaba con este software y era necesario hacer diferentes pruebas para asegurarse qué método escoger. Tras varias pruebas con el método paramétrico para la detección de fichas circulares surgió el problema de no detección de algunas piezas cuando estaban más alejadas del punto central, ya que proporcionaban otra perspectiva y no eran reconocidas bien por este método.

Además, se sumaban los posibles reflejos que surgieron a la hora de activar el flash de la cámara ya que las piezas están hechas de un plástico que produce reflejos al ser iluminado. Se pensó en la opción de desactivarlo, pero tras varias pruebas se puedo comprobar que esto era más problema porque se formaban sombras por la luz ambiente. Por ello, se optó por el método automático que acabó funcionando correctamente.

5.3. Control de fuerza

Para la realización de este proyecto se ha considerado introducir un control de fuerza para cuando el robot se dispone a dibujar la ficha en el tablero. Esto se consideró interesante ya que sin dicho control el robot ejercía demasiada fuerza lo que provocaba paradas de protección o que no llegara a bajar lo suficiente como para dibujar sobre el tablero.

5.3.1. Comando Fuerza

El modo fuerza es el encargado de llevar a cabo un control de la conformidad y las fuerzas en uno o varios ejes seleccionados dentro del espacio de trabajo del robot. El brazo robot ajustará automáticamente su posición para lograr la fuerza deseada. Otra de las posibilidades de este comando es que el propio brazo robot aplique una fuerza a su entorno.

Esto es idóneo para aplicaciones donde se requiera una fuerza a lo largo del eje donde la posición real del PCH (punto central de herramienta) a lo largo de dicho eje no sea importante. Además, permite aplicar determinados pares alrededor de los ejes que se hayan predefinido.

Este comando dispone de diferentes opciones a seleccionar (véase Figura 33) para determinar los parámetros necesarios para el funcionamiento deseado. Las opciones son las siguientes [14]:

- **Funciones**: permite la elección del sistema de coordenadas que usará el robot al funcionar bajo el modo de fuerza.
- **Tipo**: se diferencia 4 que determinan la forma en que se interpretará la función seleccionada.
    - **Simple**: solo un eje se adaptará en modo de fuerza. La fuerza a lo largo de este eje se puede ajustar.
    - **Marco**: pueden seleccionarse de forma independiente la adaptabilidad y las fuerzas en los 6 grados de libertad.
    - **Punto**: el marco de tarea viene definida por el eje y apuntando desde el PCH del robot hacia el punto de partida de la función seleccionada. Hay que tener en cuenta que el marco de tarea cambia conforme lo vaya haciendo la posición del PCH del robot.
    - **Movimiento**: esto permite que el marco de tarea cambie la dirección del movimiento del PCH. Esto puede resultar útil donde haga falta una fuerza perpendicular al movimiento del PCH.
- **Valores**: se puede configurar tanto para ejes adaptables como no adaptables.
    - **Conforme**: el brazo robot ajustará su posición para lograr la fuerza seleccionada.
    - **No conforme**: el brazo robot seguirá su trayectoria ajustada por el programa.
- **Límites**: se puede establecer un límite para los ejes, sin embargo, esto será distinto dependiendo de si los ejes son adaptables o no.
    - **Conforme**: se tratará del límite de velocidad máxima que puede alcanzar el PCH (mm/2 y º/s).
    - **No conforma**: serán los límites de desviación máxima de la trayectoria del programa (mm y º).

5.3.2. Implementación

A la hora de conseguir el control de fuerza, como se necesita cada vez que el robot va a dibujar, se optó por crear un script el cual sería llamado cada vez que se necesitara.

Este comando es conocido como *force_mode* el cual realiza la misma función que el comando "Fuerza" implementado en PolyScope como ya se ha comentado en el apartado 5.1.3. Además, cabe destacar la característica comentada anteriormente de que el brazo robot, con esta función, ya es capaz de corregir automáticamente su posición para la fuerza que se le ha indicado por lo que ahorra la implementación de control de ésta.

Para la declaración del comando *force_mode* es necesario la introducción de ciertos parámetros quedando de la siguiente forma:

force_mode(task_frame, selection_vector, wrench, type, limits)

- ***task_frame***: hace referencia al marco de referencia con el que se va a tomar la fuerza. Vendría a ser la opción de "Funciones" en el comando de PolyScope.
- ***selection_vector***: indica en qué ejes se limitará la fuerza. En este caso corresponde a la selección de ejes que se lleva a cabo en PolyScope.
- ***wrench***: especifica qué fuerza y en qué ejes se quiere controlar. Hace referencia a la opción de "Valores".
- ***type***: hace referencia al tipo de control que se quiere tener (1: para un punto, 2: simple o marco, 3: para el movimiento). Del mismo modo que se seleccionaría con la opción "Tipo" en el comando de PolyScope.
- ***limits***: especifica los límites de desviación de posición o velocidad para el cumplimiento de las especificaciones anteriores. La determinación de estos límites no deben ser demasiados pequeños ya que si se produce una desviación provoca una parada de emergencia. Siendo los "Límites" seleccionados en PolyScope.

Una vez establecidos los parámetros que se van a utilizar, lo cuales fueron elegidos sabiendo las características explicadas en el apartado 5.3.1, se implementará el script "fuerza.script" que estará formado por una función *thread*, llamada "force_properties()", para poder ser llamada desde otro script, en este caso, el script "dibujar.script". Como se ha indicado en el apartado 5.1.3, para llamar y detener a una función *thread* y el control de fuerza se realizará de la siguiente forma:

global thread_force = run force_properties()

end_force_mode()

kill thread_force

Por lo tanto, la secuencia a seguir será la siguiente:

1. Acercarse al tablero con MoveJ.
2. Activar el control de fuerza llamando a la función correspondiente.
3. Bajar con MoveL.

4. Dibujar la línea correspondiente.
5. Desactivar el control de fuerza.
6. Detener la llamada de la función.
7. Subir con MoveL.
8. Alejarse del tablero con MoveJ.

5.3.3. Problemas surgidos

El problema más destacable e importante para este proyecto fue la dificultad de dibujar que mostraba conforme más cerca de la base del robot se encontrara la ficha a dibujar. Estudiando el porqué de esta situación, ya que se estuvo mirando si era problema de los parámetros introducidos, se conoció que era debido a la rigidez que tiene este tipo de robots en las articulaciones. Esto viene definido por la siguiente ecuación:

$$J^T * K_\theta * inv(J) = K_x$$

- J^T : jacobiana transpuesta.
- K_θ : matriz de rigidez de articulaciones.
- $inv(J)$: inversa de la jacobiana.
- K_x : matriz de rigidez cartesiana.
- $\delta(x)$: deformaciones cartesianas.

La cual viene de la combinación de las dos ecuaciones siguientes:

$$[M1, M2, M3, M4, M5, M6] = J^T * [Fx, Fy, Fz, Mx, My, Mz] \quad \text{(ecuación 1)}$$

$$\delta(x) = J * \delta(q) \quad \text{(ecuación 2)}$$

- $[M1, M2, M3, M4, M5, M6]$: par en los motores.
- $[Fx, Fy, Fz, Mx, My, Mz]$: fuerza de 6D.
- $\delta(q)$: deformación en la configuración del robot.

Donde la ecuación 1 hace referencia como el reflejo en los motores de una fuerza 6D en el efector final y la ecuación 2 es para pequeñas deformaciones.

Además, se debe definir la rigidez cartesiana como:

$$[Fx, Fy, Fz, Mx, My, Mz] = K_x * [\delta(x1), \delta(x2), \delta(x3), \delta(x4), \delta(x5), \delta(x6)]$$

Y despejando la deformación cartesiana ($\delta(x)$) y teniendo en cuenta la siguiente fórmula de la rigidez en las articulaciones:

$$M = K_\theta * \delta(q)$$

Se obtiene la ecuación deseada donde, asumiendo que la matriz de rigidez de las articulaciones (K_θ) es constante, nos muestra como la matriz de rigidez cartesiana (K_x) no lo es ya que depende de la matriz jacobiana (*J*) la cual depende da la configuración [q1, q2, q3, q4, q5, q6] del robot. Por ello, el error cartesiano es más grande en unos sitios que en otros.

CAPÍTULO 6

6. Estructura del programa

Para que resulte más sencilla la explicación de la estructura del programa creado se dividirá en 3 partes, una explicación de la estructura general de forma resumida, nodo del juego y los scripts utilizados.

Es importante mostrar, antes de comenzar con la explicación del programa, cómo se han enumerado las casillas del tablero para saber en qué posición se encuentra cada ficha. En la Figura 46 se encuentra dicha enumeración.

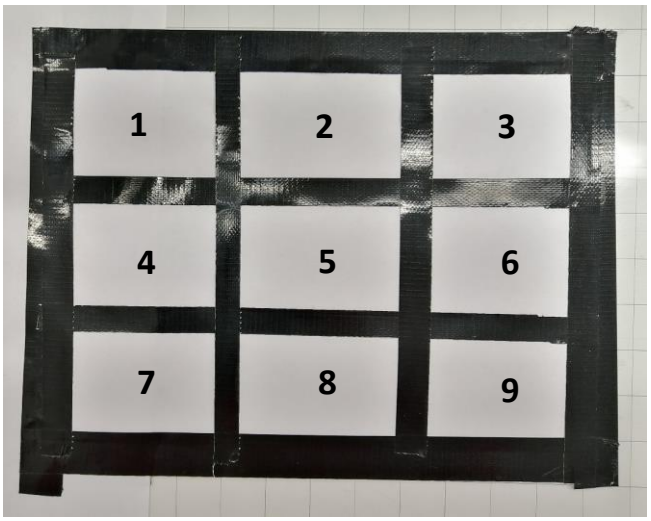


Figura 46. Numeración tablero.

6.1. Estructura general

En este apartado se hará una explicación de los elementos principales que intervienen en la aplicación diseñada (Figura 47).

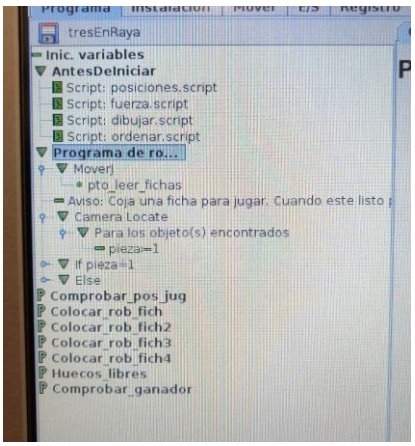


Figura 47. Estructura programa principal simplificado.

En primer lugar, encontramos una inicialización de variables donde se pondrán todas las variables definidas previamente a cero. A continuación, hay una fase de “BeforeStart” donde se encuentran una serie de scripts generales que se llamarán en el momento que se necesiten utilizar (estos scripts serán explicados son explicados en el apartado 6.3).

Una vez dentro del programa principal, lo primero a realizar es colocar el robot en el punto llamado “pto_leer_ficha” mediante un movimiento J para leer con qué ficha jugará el robot. Este punto coincide con el “Snapshot position” lo cual debe cumplirse ya que para ejecutarse las instrucciones en la estructura de “Camera locate”, el robot debe estar situado en la posición de captura previamente o no se podrá ejecutar el programa. Lo siguiente que aparecerá será un mensaje de aviso para el usuario donde informa de que debe coger las fichas (Figura 48).

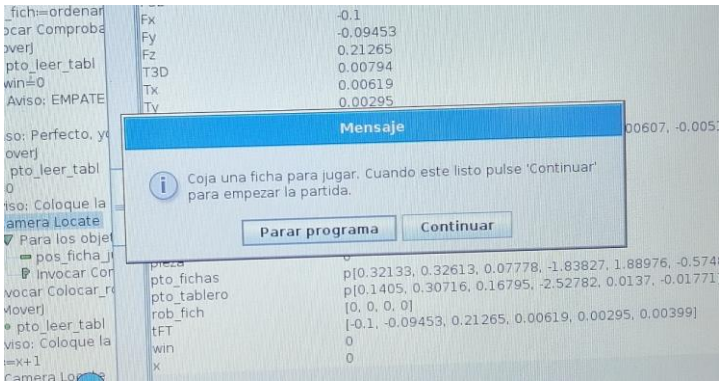


Figura 48. Mensaje emergente de aviso de coger ficha.

Una vez el usuario haya cogido las fichas, éste debe darle a continuar para poder seguir con el programa.

A continuación, tiene lugar la estructura de “Camera locate” detectando la cruz, en el caso de encontrar alguna ficha pondrá la variable “pieza” a 1. Esto significará que el robot jugará con las cruces.

6.2. Nodo del juego

Una vez ha sido detectado con qué pieza va a jugar cada uno, se entraría en la parte del juego donde, mediante un “if” (Figura 47), se diferencia si el robot juega con círculos o con cruces. Ambos casos el procedimiento es el mismo, sin embargo, cambiaría a la hora del “Camera locate” y de dibujar la figura correspondiente.

Una vez sabe el robot con qué ficha va a jugar, se moverá mediante un moveJ al punto llamado “pto_leer_tabl” que corresponde al otro “Snapshot position” definido para la detección de las fichas, en este caso, en el tablero y comenzará el juego.

Cada vez que se coloque en la posición para leer las fichas que se encuentran en el tablero, mandará un aviso como el que se muestra en la Figura 49 indicando que coloque la siguiente ficha.

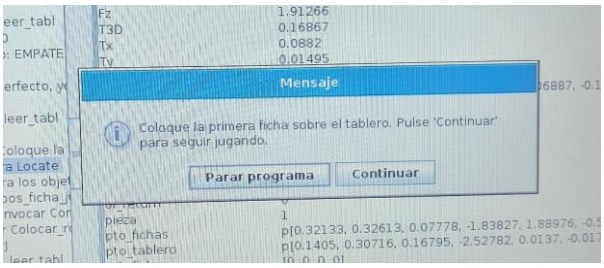


Figura 49. Mensaje emergente de aviso de colocar ficha.

Dentro del "Camera locate", dependiendo en qué momento del juego se encuentre, se repetirán los comandos tantas veces como piezas a detectar haya para así poder guardar todas las posiciones en un array.

Después de cada secuencia de detección se invocan determinados scripts: ordenar el array de las fichas que se encuentran en el tablero, saber qué huecos hay libres y que el robot coloque su ficha.

Cabe destacar que cuando el jugador ha colocado su tercera ficha, después de la detección, se añade una nueva invocación la cual revisa si ha habido algún ganador. De la misma forma lo hará cuando se coloque la cuarta ficha. En el caso de que con el último movimiento no se haya detectado un campeón, se mostrará un aviso de empate.

6.3. Scripts

La elección de utilizar scripts ha sido una opción idónea para evitar la repetición de comandos ya que, como se ha comentado anteriormente, tanto si el robot juega con los círculos como si lo hace con las cruces, los pasos a seguir son iguales.

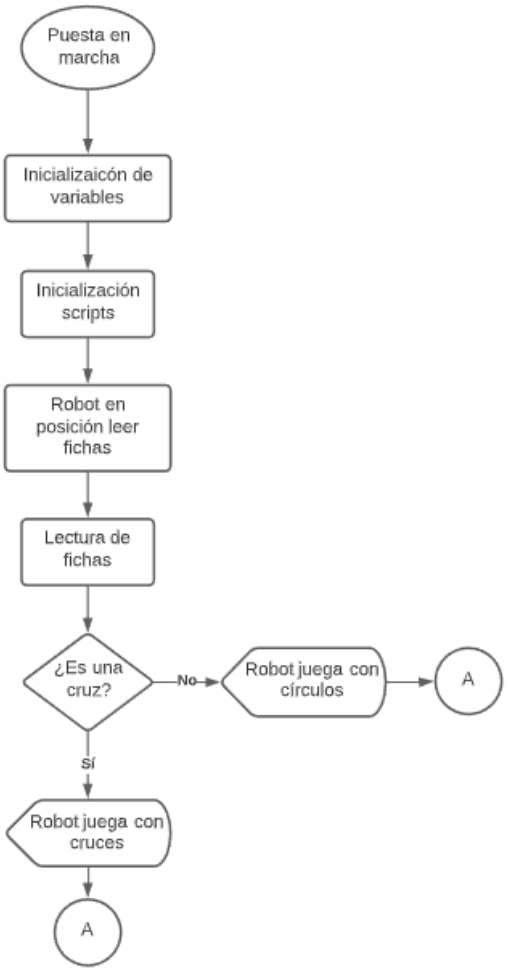
A continuación, se explican más detalladamente la finalidad de cada uno para comprender mejor el procedimiento del programa.

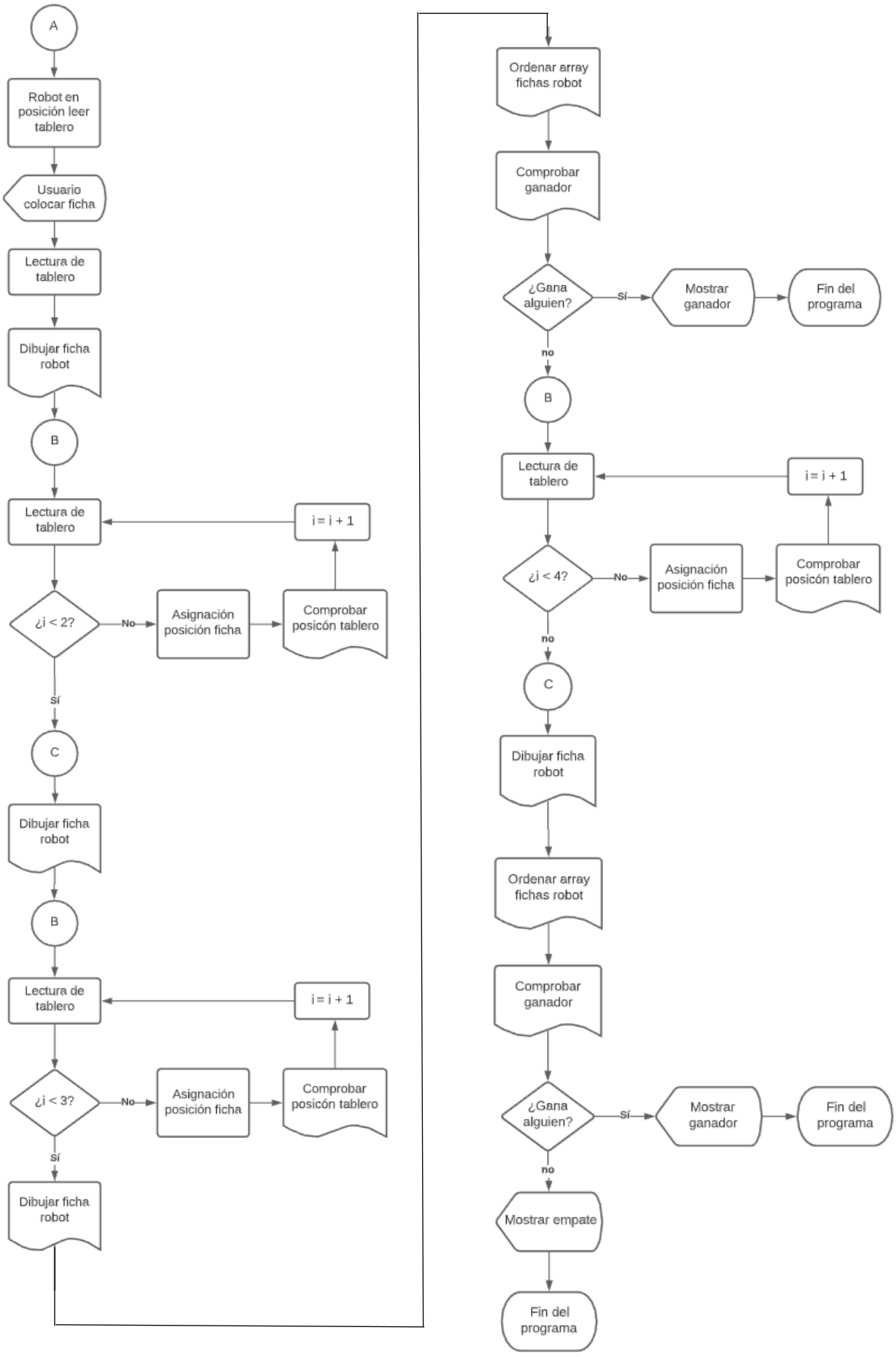
- **posiciones.script**: declaración de variables generales de las posiciones de cada casilla en el tablero.
- **fuerza.script**: función para controlar la fuerza que ejerce el robot. Dicha función recibe el nombre de *force_properties*.
- **dibujar.script**: función que necesita un punto a introducir, el cual será la posición en la que tenga que dibujar el robot. A partir de éste se calculan los puntos necesarios para acercarse al tablero y dibujar. A continuación, detecta el valor de la variable "pieza" y según corresponda dibujará una cruz o un círculo en el punto indicado controlando la fuerza en el momento de contacto. La función de este script está declara como *dibujar(punto)*.

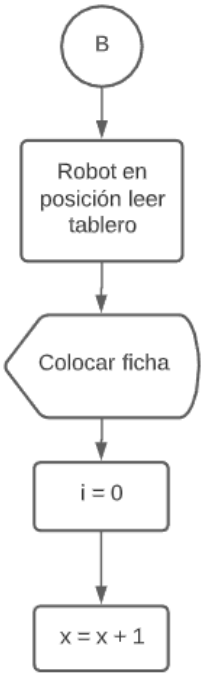
- **ordenar.script**: gracias a este script se consigue que el array introducido sea ordenado de mayor a menor. Esto facilita a la hora de leer el array para saber dónde tiene que colocar la ficha el robot. Este script contiene la función *ordenarArray(array)* la cual se encarga de esto.
- **posicionJug.script**: determina y guarda en un array la posición en la que se encuentra la ficha del usuario. Para este script se declara la función *PosFichaJug()*. Toma la variable global declarada en PolyScope "i" para situar en el array según la ficha leída.
- **move1_rob.script**: se utiliza para determinar dónde dibujará la primera ficha el robot. Para ello se crea un número aleatoriamente. La función de este script recibe el nombre de *mov1rob()*.
- **detectar_libres.script**: está formado por una función para detectar los huecos libres que hay en el tablero. Dicha función es declarada con el nombre de *huecosLibres(jug_fich,rob_fich)*. Para ello, se introducen los dos arrays donde se han ido guardando las posiciones de las fichas del robot y del usuario.
- **move2_rob.script**: se encuentra formado por dos funciones. La primera de ellas (*movLibre()*) será llamada cuando en la segunda (*mov2rob()*)se haya detectado que, en la posición que intenta dibujar, está ocupada. Para ello se tienen en cuenta las dos fichas del jugador en el tablero y se intentará bloquear para que en la siguiente jugada no gane el usuario.
- **move3_rob.script**: es igual que el anterior script pero en este caso teniendo en cuenta tres fichas colocadas en el tablero. La primera función recibe el mismo nombre que el anterior script mientras que la segunda recibe el nombre de *mov3rob()*.
- **ganador.script**: comprueba las posiciones de las fichas tanto las de parte del usuario como las del robot para ver si alguno ha resultado ganador. Para llevar a cabo esto se ha declarado la función *verGanador()*.
- **move4_rob.script**: en este caso, se detectan qué huecos son libres y dibujará ahí la ficha. La función en este script es declarada como *mov4rob()*.

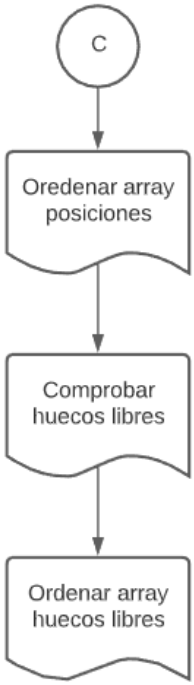
6.4. Flujograma

El flujograma de este programa ayuda a comprender de forma visual e intuitiva cómo funciona la aplicación diseñada de forma sencilla y rápida.




A
Robot en posición leer tablero
Usuario colocar ficha
Lectura de tablero
Dibujar ficha robot
B
Lectura de tablero
i = i + 1
¿i < 2?
No
Asignación posición ficha
Comprobar posicón tablero
Sí
C
Dibujar ficha robot
B
Lectura de tablero
i = i + 1
¿i < 3?
No
Asignación posición ficha
Comprobar posicón tablero
Sí
Dibujar ficha robot
Ordenar array fichas robot
Comprobar ganador
¿Gana alguien?
Sí
Mostrar ganador
Fin del programa
no
B
Lectura de tablero
i = i + 1
¿i < 4?
No
Asignación posición ficha
Comprobar posicón tablero
no
C
Dibujar ficha robot
Ordenar array fichas robot
Comprobar ganador
¿Gana alguien?
Sí
Mostrar ganador
Fin del programa
no
Mostrar empate
Fin del programa


B
Robot en posición leer tablero
Colocar ficha
i = 0
x = x + 1


C
Oredenar array posiciones
Comprobar huecos libres
Ordenar array huecos libres

CAPÍTULO 7

7. Correcciones de errores

Tras realizar el diseño de la aplicación de este proyecto había que comprobar su funcionamiento para corregir los posibles errores que fueran surgiendo.

Antes, cabe comentar que, la mayoría de los problemas surgidos durante su realización, han sido solventados o buscado una solución más apropiada para cada caso y han sido comentados en el apartado 5.

En primer lugar, uno de los errores más importantes que han surgido ha sido a la hora de determinar la posición de las piezas del usuario en el tablero mediante visión. Tras varias tomas de valores para determinar los rangos de los ejes en los que se podía encontrar la ficha, se llegó a la conclusión que sería demasiado complicado establecer de forma exacta estos puntos ya que, a la hora de coger el punto en el que se encuentra la ficha, la posición que se devuelve no siempre coincide y los rangos de error producía que la ficha se detectara en la casilla que no era. Para poder solventarlo se fueron tomando datos a base de prueba y error dando así los rangos que más se ajustaron para la solución de este problema.

En segundo lugar, a la hora de dibujar la ficha en el tablero, al no tratarse de una pinza específica para el agarre de un bolígrafo o rotulador, se daba el caso que en el momento de contacto el rotulado bailaba en la pinza por lo que se tuvo que hacer un pequeño apaño con cinta adhesiva consiguiendo así la inmovilidad del bolígrafo.

Por último, otro de los problemas importantes y a la vez más interesantes, se encontró dibujando, como se comenta en el apartado 5.3.3. Una de las soluciones en las que se pensó para la posible mejora de este problema fue dejar una mayor distancia entre el tablero y la base del robot.

CAPÍTULO 8

8. Resultados

El funcionamiento de la aplicación se puede apreciar en la siguiente Figura 50. En ella se recogen varias capturas del vídeo de demostración donde se aprecia cómo se desenvuelve durante la ejecución del programa.

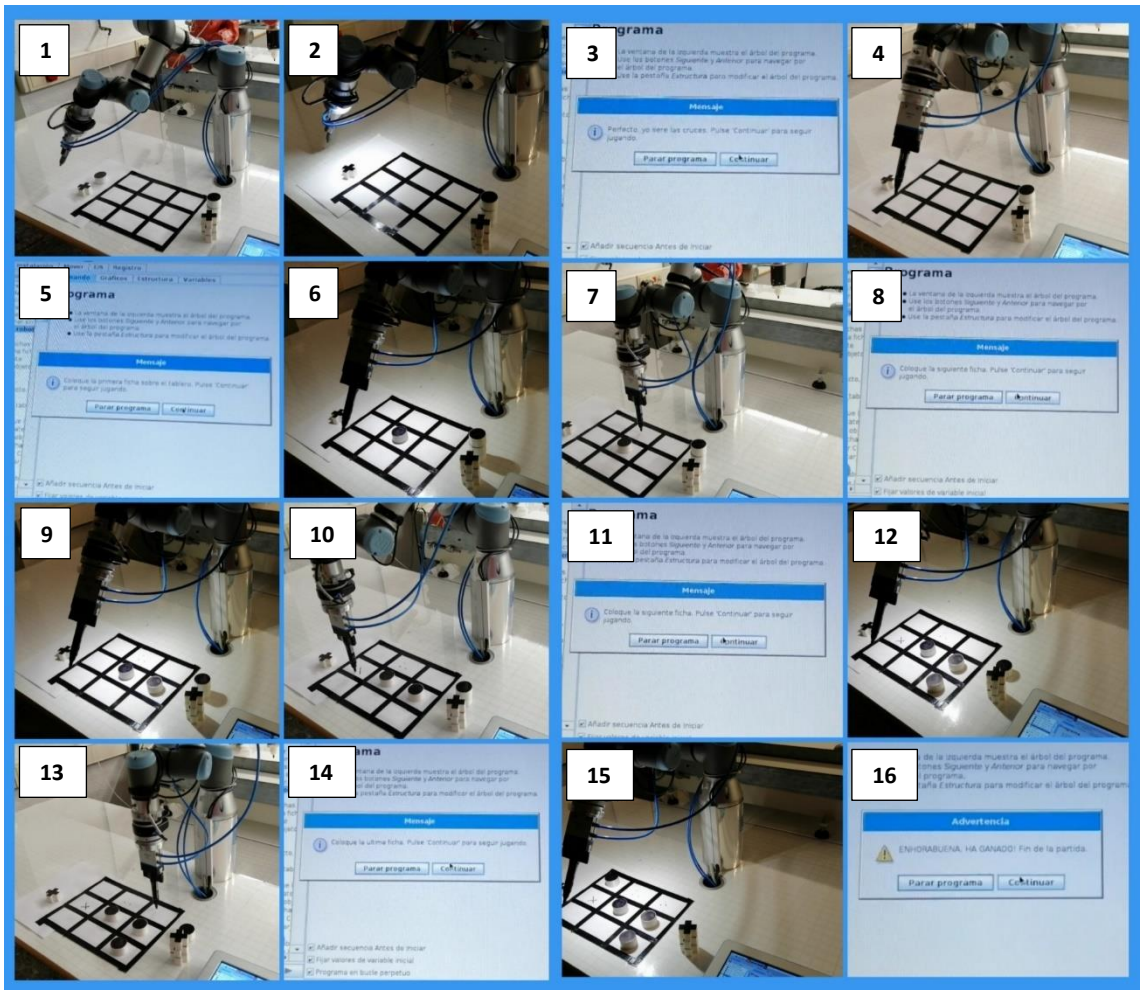


Figura 50. Secuencia de imágenes del programa en marcha.

En las secuencias 1 el robot se posiciona para leer con qué ficha jugará la partida iniciada. Una vez se haya cogido la ficha, escaneará para detectar cual queda como se muestra en la secuencia 2, la cual será con la que juegue el robot y se mostrará con un mensaje (secuencia 3).

En las secuencias 4, 6, 9, 12 y 15, el robot se encuentra en la posición de “Snapshot position” donde tomará las fotos para saber en qué posiciones se encuentran las fichas que ha posicionado el usuario en el tablero. Siempre que el robot se encuentre en esa posición, saldrá un mensaje como el de las secuencias 5, 8, 11 y 14 donde avise al usuario de poner ficha.

Sabiendo las posiciones, el robot dibuja su ficha en el tablero en la posición que más le corresponde para evitar que el usuario gane (secuencias 7, 10 y 13). Se acercará a la posición con un movimiento J y una vez cerca bajará con un movimiento L, de la misma forma lo hará para alejarse de la posición.

Por último, leerá todas las fichas y mostrará un mensaje final (secuencia 16) donde indique si ha ganado el usuario, el robot o han quedado empate.

CAPÍTULO 9

9. Conclusiones

Como uno de los objetivos de este trabajo es de realizar una aplicación de jugar al *Tres en raya* con un robot colaborativo, en base a los resultados obtenidos, se puede concluir como cumplido.

Gracias a los conceptos adquiridos en la asignatura de Sistemas Robotizados, junto con el manual de usuario y programación, se ha conseguido que la utilización del robot colaborativo UR3 y la programación en PolyScope no sean de gran dificultad. Por ello, ha sido posible el aprendizaje casi en su totalidad de todas las funciones que dispone.

Por otra parte, el equipo de visión tampoco cuenta con una programación compleja. Esto es dado a la gran compatibilidad que tiene el sistema de visión de *Robotiq* con el programa PolyScope ya que ha sido diseñado con ese fin. Sin embargo, se encontraron varias limitaciones, no se puede acceder al código de la cámara para aplicaciones más complejas, confunde fácilmente figuras parecidas y las posiciones que se adquieren no llegan a ser exactas.

También se ha conocido las limitaciones que llega a tener el robot controlado por el sensor de fuerza. Este se ve limitado, cuando se desean fuerzas ente ±1 N empieza a causar problemas por lo que se deduce que no está diseñado para actividades de detección sutiles de fuerza.

En general, ha sido un trabajo multidisciplinar donde se ha tenido la oportunidad de experimentar por primera vez con un cobot. Ha servido para la ampliación de conocimientos prácticos en robótica, visión artificial, sensor de fuerza e impresión 3D que serán de gran utilidad en un futuro.

BIBLIOGRAFIA

10. Bibliografía

PÁGINAS WEB

[1] http://www.profesormolina.com.ar/tecnologia/robotica/historia.html

[2] http://inteligencia-artificialrobotica.blogspot.com/p/historia-de-la-robotica.html

[3] http://conozcamoslarobotica.blogspot.com/p/generaciones-de-la-robotica.html

[5]https://www.euautomation.com/cl/automated/magazine/article/robotica-colaborativa-historia-y-tendencias-recientes1

[6] https://blog.universal-robots.com/es/beneficios-robots-colaborativos

[8] https://blog.infaimon.com/imagen-vision-artificial/

[9] https://www.centroestudioscervantinos.es/que-es-un-sensor/

[10]https://bookdown.org/alberto_brunete/intro_automatica/sensores-industriales.html

[12]https://sensores-de-medida.es/medicion/sensores-y-transductores/sensores-de-fuerza/

[13]https://www.sensy.com/fr/tecnologia/sensores-de-fuerza/sensor-de-fuerza-en-compresion

[15]https://insights.globalspec.com/article/7055/new-plug-and-play-robotic-wrist-camera

[17] https://mercadoindustrial.mbzpress.com/2018/07/06/onrobot-usa-tecnologia-de-la-nasa/

[19] https://onrobot.com/es/productos/sensor-de-fuerza-par-hex-de-6-ejes

MANUALES

- **Universal Robots modelo UR3**
  - Manual de instalación del hardware.
  - [14] Manual de usuario.
  - Manual de PolyScope.
  - Manual de programación.

- **Equipo de visión Wrist Camera de Robotiq**
    - [16] Manual de usuario.

- **Sensor de fuerza HEX-EB165 de OnRobot**
    - [18] Manual de usuario.

ARTÍCULOS Y

[4] *Generaciones de la robótica*, Marta M. Conde Canaviri

[7] *Los robots colaborativos: una nueva era en la automatización industrial*, Jacob Pascual Pepe

APUNTES

[11] Apuntes asignatura de *Sistema Robotizados* impartida por el departamento de Ingeniería de Sistemas y Automática. Universidad Politécnica de Valencia.

ANEXOS

11. Anexos

11.1. Ficha técnica UR3

UR3

Rendimiento

Repetibilidad	±0,1 mm / ±0,0039 in (4 mil.)
Intervalo de temperaturas	0–50°*
Consumo de energía	Mín. 90 W, estándar 125 W, máx. 250 W
Operación de colaboración	15 funciones avanzadas de seguridad regulables. Función de seguridad con certificación TÜV NORD Probado de acuerdo con las normas: EN ISO 13849:2008 PL d

Especificación

Carga útil	3 kg / 6,6 lb
Alcance	500 mm / 19,7 in
Grados de libertad	6 articulaciones giratorias
Programación	Interfaz gráfica del usuario PolyScope con pantalla táctil de 12" con soporte

Movimiento

Movim. del eje del brazo robot.	Radio de acción	Velocidad máxima
Base	± 360°	± 180°/s
Hombro	± 360°	± 180°/s
Codo	± 360°	± 180°/s
Muñeca 1	± 360°	± 360°/s
Muñeca 2	± 360°	± 360°/s
Muñeca 3	Infinita	± 360°/s
Herramienta típica		1 m/s / 39,4 in/s

Funciones

Clasificación IP	IP64	
Clase ISO Sala limpia	5	
Ruido	Comparativamente silencioso	
Montaje del robot	Todos	
Puertos de E/S en herramienta	Entrada digital	2
	Salida digital	2
	Entrada analógica	2
	Salida analógica	0
E/S de fuente de aliment. en herramienta	12 V / 24 V 600 mA en herramienta	

Características físicas

Huella	Ø 128 mm
Materiales	Aluminio, plásticos de PP
Tipo de conector para herramientas	M8
Long. cable del brazo robótico	6 m / 236 in
Peso con cable	11 kg / 24,3 lb

*El robot puede trabajar dentro del intervalo de temperaturas 0-50 °C. A alta velocidad continua de las articulaciones, la temperatura ambiente se reduce.

CAJA DE CONTROL

Funciones

Clasificación IP	IP20	
Clase ISO Sala limpia	6	
Ruido	<65 dB (A)	
Puertos de E/S	Entrada digital	16
	Salida digital	16
	Entrada analógica	2
	Salida analógica	2
E/S de fuente de alimentación	24 V 2 A	
Comunicación	TCP/IP 100 Mbit, Modbus TCP, Profinet, EthernetIP	
Fuente de alimentación	100-240 V CA, 50-60 Hz	

Características físicas

Tamaño de la caja de control (an.×al.×la.)	475 × 423 × 268 mm / 18,7 × 16,7 × 10,6 in
Peso	15 kg / 33,1 lb
Materiales	Acero

CONSOLA DE PROGRAMACIÓN

Funciones

Clasificación IP	IP20

Características físicas

Materiales	Aluminio, PP
Peso	1,5 kg
Longitud del cable	4,5 m / 177 in

11.2. Especificaciones Wrist Camera

DIMENSIONES

Specification	Value	
Maximum load	10 kg	40 Nm
Weight (without tool plate)	160 g	
Weight (with tool plate)	230 g	
Added height (without tool plate, for use with 2-Finger Gripper)	13.5 mm	
Global thickness (without tool plate)	22.4 mm	
Added height (with tool plate)	23.5 mm	
Global thickness (with tool plate)	29.5 mm	

CARACTERÍSTICAS ELÉCTRICAS

Specification	Value
Operating supply voltage	24 V DC ±20%
Quiescent power (minimum power consumption)	1 W
Maximum power	22 W
Communication interface	USB 2.0

CARACTERÍSTICAS TÉCNICAS

Specification	Value
Maximum resolution	5 Mpx at 2 fps (2560 X 1920)
Maximum frame rate	30 fps at 0.3 Mpx (640 X 480)
Active array size	2592 X 1944
Focus range	70 mm to infinity
Integrated lighting	6 LED diffuse white light
Autofocus technology	Liquid lense

CARACTERÍSTICAS DE PRECISION

Robot Model	Accuracy
UR3	+/- 2mm
UR5	+/- 3mm
UR10	+/- 3mm

11.3. Especificaciones HEX-EB165

Propiedades generales	Sensor de fuerza/par de 6 ejes				Unidad
	Fxy	Fz	Txy	Tz	
Capacidad nominal (C. N.)	200	200	10	6,5	[N] [Nm]
Deformación de eje único en C. N. (típica)	±1,7 ±0,067	±0,3 ±0,011	±2,5 ±2,5	±5 ±5	[mm] [°] [in] [°]
Sobrecarga de eje único	500	500	500	500	[%]
Señal de ruido* (típica)	0,035	0,15	0,002	0,001	[N] [Nm]
Resolución sin ruido (típica)	0,2	0,8	0,01	0,002	[N] [Nm]
No linealidad a gran escala	< 2	< 2	< 2	< 2	[%]
Histéresis (medida en eje Fz, típica)	< 2	< 2	< 2	< 2	[%]
Diafonía (típica)	< 5	< 5	< 5	< 5	[%]
Clasificación IP	67				
Dimensiones (alto x ancho x largo)	50 x 71 x 93 1,97 x 2,79 x 3,66				[mm] [in]
Peso (con placas adaptadoras integradas)	0,347 0,76				[kg] [lb]

* La señal de ruido se define como la desviación estándar (1 σ) de una señal sin carga de un segundo típica.

Condiciones de funcionamiento	Mínimo	Típico	Máximo	Unidad
Fuente de alimentación	7	-	24	[V]
Consumo de energía	-	-	0,8	[W]
Temperatura de funcionamiento	0 32	- -	55 131	[°C] [°F]
Humedad relativa (sin condensación)	0	-	95	[%]
MTBF calculado (vida útil)	30 000	-	-	[Horas]

11.4. Manual de usuario

A TENER EN CUENTA…

Antes de utilizar esta aplicación, recuerde leer este manual detenidamente para darle un uso de forma correcta. Después de eso, guarde este manual en un lugar seguro y consúltelo cuando sea necesario.

Para utilizar correctamente esta aplicación, se explica el contenido a tener en cuenta, clasificado tal como sigue.

- La gravedad de las lesiones y daños debidos al uso incorrecto por hacer caso omiso del contenido explicado se explica mediante las siguientes indicaciones clasificadas.
- La información incluida en este manual con relación a la seguridad no debe considerarse una garantía de que el manipulador industrial no causará lesiones o daños, aunque se cumplan todas las instrucciones de seguridad.

ADVERTENCIAS

Indica una situación potencialmente peligrosa que, si no se evita, puede provocar la muerte o lesiones graves.

PRECAUCIONES

Indica una situación potencialmente peligrosa que, si no se evita puede provocar lesiones o daños materiales.

- Se explica el contenido a tener en cuenta y se clasifica mediante los siguientes pictogramas:

Esto indica una situación posiblemente peligrosa que, si no se evita, podría provocar lesiones o daños importantes.

Esto indica una superficie caliente posiblemente peligrosa que, si se toca, podría provocar lesiones.

- Asegúrese de que los pernos, el brazo robótico, sensores y herramienta estén correcta y seguramente colocados y no se encuentren dañados.

- Asegúrese de que el brazo robótico tenga espacio suficiente para funcionar libremente.

- No lleve ropa holgada ni joyas cuando trabaje con el robot. Asegúrese de recogerse el pelo si lo tiene largo.

- Guarde y cargue el archivo de instalación junto con el programa.

- Tenga cuidado con el movimiento del robot cuando utilice la consola portátil.

- El robot y la caja del controlador generan calor durante su funcionamiento. No manipule ni toque el robot mientras esté en funcionamiento o inmediatamente después de su funcionamiento. Para refrigerar el robot, apague el robot y espere una hora.

- Asegúrese de que todos los cables estén asegurados.

- Asegúrese de que todas las personas mantengan las cabezas y los rostros alejados del alcance del robot.

BOTONES

- [POWER] Pulse el botón POWER que se encuentra en la consola portátil para encender el robot, una vez pulsado se pondrán en funcionamiento los equipos del UR3.
- [PARA DE SEGURIDAD] Pulse el botón PARADA DE SEGURIDAD cuando se detecte algún fallo o anomalía que pueda dañar a alguien o al propio equipo.
- [MOVIMIENTO LIBRE] Mantenga pulsado el botón MOVIMIENTO LIBRE que se encuentra en la parte trasera de la consola para cambiar la posición del robot de forma manual.

FUNCIONAMIENTO

Una vez encendido aparecerá un texto del sistema operativo subyacente en la pantalla táctil. Al minuto aproximadamente, aparecerán algunos botones en la pantalla y un mensaje emergente para la inicialización donde se pulsará el botón *Ahora no* para cargar el programa previamente.

Desde los botones que aparecen en la pantalla se pulsará el botón *Programar Robot* que llevará a una nueva pantalla de botones donde será posible cargar el programa pulsando el botón *Cargar programa*.

Una vez se haya cargado el programa, bien desde un pendrive o desde la propia consola, se deberá encender el brazo robot desde el menú *Inicialización* pulsando el botón *Encender* que se muestra en la pantalla y, a continuación, pulsando *Iniciar*. Cuando se inicia el robot, emite un sonido y se mueve ligeramente mientras libera los frenos. Al finalizar, se pulsará el botón *Ok* y llevará a la pantalla principal del programa donde pulsando el botón *Play* comenzará la ejecución de este.

11.5. Estructura en PolyScope

```
Programa
  Inic. variables
  BeforeStart
    Script: posiciones.script
    Script: fuerza.script
    Script: dibujar.script
    Script: ordenar.script
  Programa de robot
    MoveJ
      pto_leer_fichas
    Aviso
    Camera Locate
      Para los objeto(s) encontrados
        pieza:=1
    If pieza≟1
      Aviso
      MoveJ
        pto_leer_tabl
      i:=0
      Aviso
      Camera Locate
        Para los objeto(s) encontrados
          pos_ficha_jug:=pto_tablero_var
          Invocar Comprobar_pos_jug
      Invocar Colocar_rob_fich
      MoveJ
        pto_leer_tabl
      Aviso
      x:=x+1
      Camera Locate
        Para los objeto(s) encontrados
          If i<2
            pos_ficha_jug:=pto_tablero_var
            Invocar Comprobar_pos_jug
          i:=i+1
      jug_fich:=ordenarArray(jug_fich)
      Invocar Huecos_libres
      libre:=ordenarArray(libre)
      Invocar Colocar_rob_fich2
      MoveJ
        pto_leer_tabl
      Aviso
      i:=0
      x:=x+1
      Camera Locate
        Para los objeto(s) encontrados
          If i<3
            pos_ficha_jug:=pto_tablero_var
            Invocar Comprobar_pos_jug
          i:=i+1
      jug_fich:=ordenarArray(jug_fich)
      Invocar Huecos_libres
      libre:=ordenarArray(libre)
      Invocar Colocar_rob_fich3
      rob_fich:=ordenarArray(rob_fich)
      Invocar Comprobar_ganador
      MoveJ
        pto_leer_tabl
      Aviso
      i:=0
      x:=x+1
      Camera Locate
        Para los objeto(s) encontrados
          If i<4
            pos_ficha_jug:=pto_tablero_var
            Invocar Comprobar_pos_jug
          i:=i+1
      jug_fich:=ordenarArray(jug_fich)
      Invocar Huecos_libres
      Invocar Comprobar_ganador
      libre:=ordenarArray(libre)
      Invocar Colocar_rob_fich4
      rob_fich:=ordenarArray(rob_fich)
      Invocar Comprobar_ganador
      MoveJ
        pto_leer_tabl
      If win≟0
        Aviso
      Else
        Aviso
        MoveJ
          pto_leer_tabl
        i:=0
        Aviso
        Camera Locate
          Para los objeto(s) encontrados
```

```
    pos_ficha_jug:=pto_tablero_var
    Invocar Comprobar_pos_jug
Invocar Colocar_rob_fich
MoveJ
  pto_leer_tabl
Aviso
x:=x+1
Camera Locate
  Para los objeto(s) encontrados
    If i<2
      pos_ficha_jug:=pto_tablero_var
      Invocar Comprobar_pos_jug
    i:=i+1
jug_fich:=ordenarArray(jug_fich)
Invocar Huecos_libres
libre:=ordenarArray(libre)
Invocar Colocar_rob_fich2
MoveJ
  pto_leer_tabl
Aviso
i:=0
x:=x+1
Camera Locate
  Para los objeto(s) encontrados
    If i<3
      pos_ficha_jug:=pto_tablero_var
      Invocar Comprobar_pos_jug
    i:=i+1
jug_fich:=ordenarArray(jug_fich)
Invocar Huecos_libres
libre:=ordenarArray(libre)
Invocar Colocar_rob_fich3
rob_fich:=ordenarArray(rob_fich)
Invocar Comprobar_ganador
MoveJ
  pto_leer_tabl
Aviso
i:=0
x:=x+1
Camera Locate
  Para los objeto(s) encontrados
    If i<4
      pos_ficha_jug:=pto_tablero_var
      Invocar Comprobar_pos_jug
    i:=i+1
    jug_fich:=ordenarArray(jug_fich)
    Invocar Huecos_libres
    libre:=ordenarArray(libre)
    Invocar Colocar_rob_fich4
    rob_fich:=ordenarArray(rob_fich)
    Invocar Comprobar_ganador
    MoveJ
      pto_leer_tabl
    If win≟0
      Aviso
Comprobar_pos_jug
  Script: posicionJug.script
  jug_fich:=posFichaJug()
Colocar_rob_fich
  Script: move1_rob.script
  posicion:=mov1rob()
  dibujar(posicion)
Colocar_rob_fich2
  Script: move2_rob.script
  posicion:=mov2rob()
  dibujar(posicion)
Colocar_rob_fich3
  Script: move3_rob.script
  posicion:=mov3rob()
  dibujar(posicion)
Colocar_rob_fich4
  Script: move4_rob.script
  posicion:=mov4rob()
  dibujar(posicion)
Huecos_libres
  Script: detectar_libres.script
  libre:=huecosLibres(jug_fich,rob_fich)
Comprobar_ganador
  Script: ganador.script
  win:=verGanador()
  If win≟1
    Aviso
  ElseIf win≟2
    Aviso
```

11.6. Código fuente de programación

```
def tresEnRaya():
  modbus_add_signal("158.42.206.10", 255, 128, 2, "MODBUS_1", False)
  modbus_set_signal_update_frequency("MODBUS_1", 10)
  modbus_add_signal("158.42.206.10", 255, 129, 2, "MODBUS_2", False)
  modbus_set_signal_update_frequency("MODBUS_2", 10)
  set_standard_analog_input_domain(0, 1)
  set_standard_analog_input_domain(1, 1)
  set_tool_analog_input_domain(0, 1)
  set_tool_analog_input_domain(1, 1)
  set_analog_outputdomain(0, 0)
  set_analog_outputdomain(1, 0)
  set_tool_voltage(0)
  set_input_actions_to_default()
  set_runstate_standard_digital_output_to_value(3, 3)
  set_tcp(p[0.0,0.0,0.0,0.0,0.0,0.0])
  set_payload(1.2)
  set_gravity([0.0, 0.0, 9.82])
  global i_var_3=0
  global i_var_2=0
  global i_var_1=798
  global pto_fichas=p[0.3213358748143835,0.32612446468714495,0.07779290654808824,-
1.8380996768848783,1.889792478903238,-0.5746443806640781]
  global pto_tablero=p[0.14048766331351048,0.3071671394532257,0.16794161737473473,-
2.5277982295060917,0.013747643390065246,-0.017677924219300607]
  # begin: URCap Installation Node
  #   Source: Web dashboard, 1.8.1, Kim Nyholm SL
  #   Type: Web dashboard
  # end: URCap Installation Node
  # begin: URCap Installation Node
  #   Source: FT-OnRobot, 4.0.2, OnRobot A/S
  #   Type: ConfiguraciÃ³n de FT de OnRobot
  # end: URCap Installation Node
  # begin: URCap Installation Node
  #   Source: Robotiq_Wrist_Camera, 1.6.6, Robotiq Inc.
  #   Type: CÃ¡mara

  ##############################################
  #######Vision urcap preamble start########

  logging_service = rpc_factory("xmlrpc","http://127.0.0.1:4747")
  # Converts a pose relative to the flange in the base frame.
  def get_T_in_base_from_flange(T_x_in_flange):
```

```
 T_flange_in_base = get_actual_tool_flange_pose()

 T_x_in_base = pose_trans(T_flange_in_base, T_x_in_flange)

 return T_x_in_base
end

# Search pose cartesian (camera pose)
pto_fichas = p[0.321335, 0.326128, 0.0777825, -1.83827, 1.88976, -0.574806]
pto_tablero = p[0.140501, 0.307158, 0.167948, -2.52782, 0.0136984, -0.0177129]
T_camera_in_flange = p[0.0, 0.05, 0.05, -0.5, 0.0, 0.0]
snapshot_position_offset = p[0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
ignore_snapshot_position = False

# Open connection with vision service
xmlrpc_server=rpc_factory("xmlrpc","http://127.0.0.1:4242")

#######Vision urcap preamble end##########
##########################################

# end: URCap Installation Node
# begin: URCap Installation Node
#   Source: WebcamUR, 1.0.0.UPV, Fran Lopez Nacher
#   Type: VisionUR
# end: URCap Installation Node
global jug_fich=[0,0,0,0]
global libre=[0,0,0,0,0,0,0,0,0]
global pieza=0
global rob_fich=[0,0,0,0]
global win=0
global x=0
def Comprobar_pos_jug():
 def posFichaJug():
        if pos_ficha_jug[1]<0.30:
                if pos_ficha_jug[0]>0.14:
                        jug_fich[i]=1
                elif 0.14>pos_ficha_jug[0] and pos_ficha_jug[0]>0.10:
                        jug_fich[i]=2
                elif pos_ficha_jug[0]<0.1:
                        jug_fich[i]=3
                end
        elif 0.3<pos_ficha_jug[1] and pos_ficha_jug[1]<0.38:
                if pos_ficha_jug[0]>0.16:
                        jug_fich[i]=4
```

```
                elif 0.16>pos_ficha_jug[0] and pos_ficha_jug[0]>0.10:
                        jug_fich[i]=5
                elif pos_ficha_jug[0]<0.1:
                        jug_fich[i]=6
                end
        elif pos_ficha_jug[1]>0.38:
                if pos_ficha_jug[0]>0.16:
                        jug_fich[i]=7
                elif 0.16>pos_ficha_jug[0] and pos_ficha_jug[0]>0.10:
                        jug_fich[i]=8
                elif pos_ficha_jug[0]<0.1:
                        jug_fich[i]=9
                end
        end
        return jug_fich
 end
 global jug_fich=posFichaJug()
end
def Colocar_rob_fich():
 def mov1rob():
        pos=random()*10

        if 0<=pos and pos<2:
                if jug_fich[0]!=1:
                        posicion = pos1
                        rob_fich[x]=1
                else:
                        posicion = pos3
                        rob_fich[x]=3
                end
        elif 2<=pos and pos<3:
                if jug_fich[0]!=2:
                        posicion = pos2
                        rob_fich[x]=2
                else:
                        posicion = pos4
                        rob_fich[x]=4
                end
        elif 3<=pos and pos<4:
                if jug_fich[0]!=3:
                        posicion = pos3
                        rob_fich[x]=3
                else:
                        posiscion = pos5
                        rob_fich[x]=5
```

```
        end
elif 4<=pos and pos<5:
        if jug_fich[0]!=4:
                posicion = pos4
                rob_fich[x]=4
        else:
                posicion = pos6
                rob_fich[x]=6
        end
elif 5<=pos and pos<6:
        if jug_fich[0]!=5:
                posicion = pos5
                rob_fich[x]=5
        else:
                posicion = pos7
                rob_fich[x]=7
        end
elif 6<=pos and pos<7:
        if jug_fich[0]!=6:
                posicion = pos6
                rob_fich[x]=6
        else:
                posicion = pos8
                rob_fich[x]=8
        end
elif 7<=pos and pos<8:
        if jug_fich[0]!=7:
                posicion = pos7
                rob_fich[x]=7
        else:
                posicion = pos9
                rob_fich[x]=9
        end
elif 8<=pos and pos<9:
        if jug_fich[0]!=8:
                posicion = pos8
                rob_fich[x]=8
        else:
                posicion = pos1
                rob_fich[x]=1
        end
elif 9<=pos and pos<=10:
        if jug_fich[0]!=9:
                posicion = pos9
                rob_fich[x]=9
```

```
                else:
                        posicion = pos2
                        rob_fich[x]=2
                end
        end
        return posicion
 end

 global posicion=mov1rob()
 dibujar(posicion)
end
def Colocar_rob_fich2():
 def movlibre():
        if libre[1]==1:
                posicion = pos1
                rob_fich[x]=1
        elif libre[1]==2:
                posicion = pos2
                rob_fich[x]=2
        elif libre[1]==3:
                posicion = pos3
                rob_fich[x]=3
        elif libre[1]==4:
                posicion = pos4
                rob_fich[x]=4
        elif libre[1]==5:
                posicion = pos5
                rob_fich[x]=5
        elif libre[1]==6:
                posicion = pos6
                rob_fich[x]=6
        elif libre[1]==7:
                posicion = pos7
                rob_fich[x]=7
        elif libre[1]==8:
                posicion = pos8
                rob_fich[x]=8
        else:
                posicion = pos9
                rob_fich[x]=9
        end
        return rob_fich
 end

 def mov2rob():
```

```
if jug_fich[0]==9:
        if jug_fich[1]==1 and rob_fich[0]!=5:
                posicion = pos5
                rob_fich[x]=5
        elif jug_fich[1]==3 and rob_fich[0]!=6:
                posicion = pos6
                rob_fich[x]=6
        elif jug_fich[1]==5 and rob_fich[0]!=1:
                posicion = pos1
                rob_fich[x]=1
        elif jug_fich[1]==6 and rob_fich[0]!=3:
                posicion = pos3
                rob_fich[x]=3
        elif jug_fich[1]==7 and rob_fich[0]!=8:
                posicion = pos8
                rob_fich[x]=8
        elif jug_fich[1]==8 and rob_fich[0]!=7:
                posicion = pos7
                rob_fich[x]=7
        else:
                rob_fich=movlibre()
        end
elif jug_fich[0]==8:
        if jug_fich[1]==2 and rob_fich[0]!=5:
                posicion = pos5
                rob_fich[x]=5
        elif jug_fich[1]==5 and rob_fich[0]!=2:
                posicion = pos2
                rob_fich[x]=2
        elif jug_fich[1]==7 and rob_fich[0]!=9:
                posicion = pos9
                rob_fich[x]=9
        else:
                rob_fich=movlibre()
        end
elif jug_fich[0]==7:
        if jug_fich[1]==1 and rob_fich[0]!=4:
                posicion = pos4
                rob_fich[x]=4
        elif jug_fich[1]==3 and rob_fich[0]!=5:
                posicion = pos5
                rob_fich[x]=5
        elif jug_fich[1]==4 and rob_fich[0]!=1:
                posicion = pos1
                rob_fich[x]=1
```

```
        elif jug_fich[1]==5 and rob_fich[0]!=3:
                posicion = pos3
                rob_fich[x]=3
        else:
                rob_fich=movlibre()
        end
elif jug_fich[0]==6:
        if jug_fich[1]==3 and rob_fich[0]!=9:
                posicion = pos9
                rob_fich[x]=9
        elif jug_fich[1]==4 and rob_fich[0]!=5:
                posicion = pos5
                rob_fich[x]=5
        elif jug_fich[1]==5 and rob_fich[0]!=4:
                posicion = pos4
                rob_fich[x]=4
        else:
                rob_fich=movlibre()
        end
elif jug_fich[0]==5:
        if jug_fich[1]==1 and rob_fich[0]!=9:
                posicion = pos9
                rob_fich[x]=9
        elif jug_fich[1]==2 and rob_fich[0]!=8:
                posicion = pos8
                rob_fich[x]=8
        elif jug_fich[1]==3 and rob_fich[0]!=7:
                posicion = pos7
                rob_fich[x]=7
        elif jug_fich[1]==4 and rob_fich[0]!=6:
                posicion = pos6
                rob_fich[x]=6
        else:
                rob_fich=movlibre()
        end
elif jug_fich[0]==4:
        if jug_fich[1]==1 and rob_fich[0]!=7:
                posicion = pos7
                rob_fich[x]=7
        else:
                rob_fich=movlibre()
        end
elif jug_fich[0]==3:
        if jug_fich[1]==2 and rob_fich[0]!=1:
                posicion = pos1
```

```
                    rob_fich[x]=1
                elif jug_fich[1]==1 and rob_fich[0]!=2:
                    posicion = pos2
                    rob_fich[x]=2
                else:
                    rob_fich=movlibre()
        end
          elif jug_fich[0]==2:
                if jug_fich[1]==1 and rob_fich[0]!=3:
                    posicion = pos3
                    rob_fich[x]=3
                else:
                    rob_fich=movlibre()
                end
          end
          return posicion
  end

  global posicion=mov2rob()
  dibujar(posicion)
end
def Colocar_rob_fich3():
  def movlibre():
        if libre[1]==1:
                local posicion = pos1
                rob_fich[x]=1
        elif libre[1]==2:
                posicion = pos2
                rob_fich[x]=2
        elif libre[1]==3:
                posicion = pos3
                rob_fich[x]=3
        elif libre[1]==4:
                posicion = pos4
                rob_fich[x]=4
        elif libre[1]==5:
                posicion = pos5
                rob_fich[x]=5
        elif libre[1]==6:
                posicion = pos6
                rob_fich[x]=6
        elif libre[1]==7:
                posicion = pos7
                rob_fich[x]=7
        elif libre[1]==8:
```

```
                posicion = pos8
                rob_fich[x]=8
        else:
                posicion = pos9
                rob_fich[x]=9
        end
        return posicion
end

def mov3rob():
        m=0
        while m<3:
                if jug_fich[m]==9:
                        if jug_fich[m+1]==1 and (rob_fich[0]!=5 and rob_fich[1]!=5):
                                posicion = pos5
                                rob_fich[x]=5
                                m=3
                        elif jug_fich[m+1]==3 and (rob_fich[0]!=6 and rob_fich[1]!=6):
                                posicion = pos6
                                rob_fich[x]=6
                                m=3
                        elif jug_fich[m+1]==5 and (rob_fich[0]!=1 and rob_fich[1]!=1):
                                posicion = pos1
                                rob_fich[x]=1
                                m=3
                        elif jug_fich[m+1]==6 and (rob_fich[0]!=3 and rob_fich[1]!=3):
                                posicion = pos3
                                rob_fich[x]=3
                                m=3
                        elif jug_fich[m+1]==7 and (rob_fich[0]!=8 and rob_fich[1]!=8):
                                posicion = pos8
                                rob_fich[x]=8
                                m=3
                        elif jug_fich[m+1]==8 and (rob_fich[0]!=7 and rob_fich[1]!=7):
                                posicion = pos7
                                rob_fich[x]=7
                                m=3
                        else:
                                if m==2:
                                        posicion=movlibre()
                                        m=3
                                else:
                                        m=m+1
                                end
                        end
```

```
elif jug_fich[m]==8:
        if jug_fich[m+1]==2 and (rob_fich[0]!=5 and rob_fich[1]!=5):
                posicion = pos5
                rob_fich[x]=5
                m=3
        elif jug_fich[m+1]==5 and (rob_fich[0]!=2 and rob_fich[1]!=2):
                posicion = pos2
                rob_fich[x]=2
                m=3
        elif jug_fich[m+1]==7 and (rob_fich[0]!=9 and rob_fich[1]!=9):
                posicion = pos9
                rob_fich[x]=9
                m=3
        else:
                if m==2:
                        posicion=movlibre()
                        m=3
                else:
                        m=m+1
                end
        end
elif jug_fich[m]==7:
        if jug_fich[m+1]==1 and (rob_fich[0]!=4 and rob_fich[1]!=4):
                posicion = pos4
                rob_fich[x]=4
                m=3
        elif jug_fich[m+1]==3 and (rob_fich[0]!=5 and rob_fich[1]!=5):
                posicion = pos5
                rob_fich[x]=5
                m=3
        elif jug_fich[m+1]==4 and (rob_fich[0]!=1 and rob_fich[1]!=1):
                posicion = pos1
                rob_fich[x]=1
                m=3
        elif jug_fich[m+1]==5 and (rob_fich[0]!=3 and rob_fich[1]!=3):
                posicion = pos3
                rob_fich[x]=3
                m=3
        else:
                if m==2:
                        posicion=movlibre()
                        m=3
                else:
                        m=m+1
                end
```

```
        end
elif jug_fich[m]==6:
        if jug_fich[m+1]==3 and (rob_fich[0]!=9 and rob_fich[1]!=9):
                posicion = pos9
                rob_fich[x]=9
                m=3
        elif jug_fich[m+1]==4 and (rob_fich[0]!=5 and rob_fich[1]!=5):
                posicion = pos5
                rob_fich[x]=5
                m=3
        elif jug_fich[m+1]==5 and (rob_fich[0]!=4 and rob_fich[1]!=4):
                posicion = pos4
                rob_fich[x]=4
                m=3
        else:
                if m==2:
                        posicion=movlibre()
                        m=3
                else:
                        m=m+1
                end
        end
elif jug_fich[m]==5:
        if jug_fich[m+1]==1 and (rob_fich[0]!=9 and rob_fich[1]!=9):
                posicion = pos9
                rob_fich[x]=9
                m=3
        elif jug_fich[m+1]==2 and (rob_fich[0]!=8 and rob_fich[1]!=8):
                posicion = pos8
                rob_fich[x]=8
                m=3
        elif jug_fich[m+1]==3 and (rob_fich[0]!=7 and rob_fich[1]!=7):
                posicion = pos7
                rob_fich[x]=7
                m=3
        elif jug_fich[m+1]==4 and (rob_fich[0]!=6 and rob_fich[1]!=6):
                posicion = pos6
                rob_fich[x]=6
                m=3
        else:
                if m==2:
                        posicion=movlibre()
                        m=3
                else:
                        m=m+1
```

```
                end
            end
        elif jug_fich[m]==4:
            if jug_fich[m+1]==1 and (rob_fich[0]!=7 and rob_fich[1]!=7):
                posicion = pos7
                rob_fich[x]=7
                m=3
            else:
                if m==2:
                    posicion=movlibre()
                    m=3
                else:
                    m=m+1
                end
            end
        elif jug_fich[m]==3:
            if jug_fich[m+1]==2 and (rob_fich[0]!=1 and rob_fich[1]!=1):
                posicion = pos1
                rob_fich[x]=1
                m=3
            elif jug_fich[m+1]==1 and (rob_fich[0]!=2 and rob_fich[1]!=2):
                posicion = pos2
                rob_fich[x]=2
                m=3
            else:
                if m==2:
                    posicion=movlibre()
                    m=3
                else:
                    m=m+1
                end
    end
        elif jug_fich[m]==2:
            if jug_fich[m+1]==1 and (rob_fich[0]!=3 and rob_fich[1]!=3):
                posicion = pos3
                rob_fich[x]=3
                m=3
            else:
                if m==2:
                    posicion=movlibre()
                    m=3
                else:
                    m=m+1
                end
            end
```

```
                else:
                        posicion=movlibre()
                        m=3
                end

        end
        return posicion
 end
 global posicion=mov3rob()
 dibujar(posicion)
end
def Colocar_rob_fich4():
 def mov4rob():
        if libre[1]==1:
                local posicion = pos1
                rob_fich[x]=1
        elif libre[1]==2:
                posicion = pos2
                rob_fich[x]=2
        elif libre[1]==3:
                posicion = pos3
                rob_fich[x]=3
        elif libre[1]==4:
                posicion = pos4
                rob_fich[x]=4
        elif libre[1]==5:
                posicion = pos5
                rob_fich[x]=5
        elif libre[1]==6:
                posicion = pos6
                rob_fich[x]=6
        elif libre[1]==7:
                posicion = pos7
                rob_fich[x]=7
        elif libre[1]==8:
                posicion = pos8
                rob_fich[x]=8
        else:
                posicion = pos9
                rob_fich[x]=9
        end
        return posicion
 end
 global posicion=mov4rob()
 dibujar(posicion)
```

```
end
def Huecos_libres():
 def huecosLibres(jug_fich,rob_fich):
        dim_fichas=0
        dim_tablero=0
        r=0
        fichas=[0,0,0,0,0,0,0,0]
        while r<8:
                if r<4:
                        fichas[r]=jug_fich[r]
                else:
                        fichas[r]=rob_fich[r-4]
                end
                r=r+1
        end
        tablero = [1,2,3,4,5,6,7,8,9]
        libre=[0,0,0,0,0,0,0,0,0]
        dim_fichas = get_list_length(fichas)
        dim_tablero = get_list_length(tablero)
        r=0
        p=0
        while r<dim_tablero:
                falta = True
                while p<dim_fichas:
                        if fichas[p]==tablero[r]:
                                falta=False
                        end
                        p=p+1
                end
                if falta:
                        libre[r]=tablero[r]
                         else:
                             libre[r]=0
                end
                p=0
                r=r+1
        end
        return libre
 end
 global libre=huecosLibres(jug_fich,rob_fich)
end
def Comprobar_ganador():
 def verGanador():
        global win=0
        if jug_fich[0]==9 and jug_fich[1]==8 and jug_fich[2]==7:
```

```
            win=1

    elif jug_fich[0]==9 and jug_fich[1]==6 and jug_fich[2]==3:
            win=1
    elif jug_fich[0]==9 and jug_fich[2]==6 and jug_fich[3]==3:
            win=1
    elif jug_fich[0]==9 and jug_fich[1]==6 and jug_fich[3]==3:
            win=1

    elif jug_fich[0]==9 and jug_fich[1]==5 and jug_fich[2]==1:
            win=1
    elif jug_fich[0]==9 and jug_fich[2]==5 and jug_fich[3]==1:
            win=1
    elif jug_fich[0]==9 and jug_fich[1]==5 and jug_fich[3]==1:
            win=1

    elif jug_fich[0]==8 and jug_fich[1]==5 and jug_fich[2]==2:
            win=1
    elif jug_fich[1]==8 and jug_fich[2]==5 and jug_fich[3]==2:
            win=1
    elif jug_fich[0]==8 and jug_fich[2]==5 and jug_fich[3]==2:
            win=1
    elif jug_fich[0]==8 and jug_fich[1]==5 and jug_fich[3]==2:
            win=1

    elif jug_fich[0]==7 and jug_fich[1]==5 and jug_fich[2]==3:
            win=1
    elif jug_fich[1]==7 and jug_fich[2]==5 and jug_fich[3]==3:
            win=1
    elif jug_fich[0]==7 and jug_fich[2]==5 and jug_fich[3]==3:
            win=1
    elif jug_fich[0]==7 and jug_fich[1]==5 and jug_fich[3]==3:
            win=1

    elif jug_fich[0]==7 and jug_fich[1]==4 and jug_fich[2]==1:
            win=1
    elif jug_fich[1]==7 and jug_fich[2]==4 and jug_fich[3]==1:
            win=1
    elif jug_fich[0]==7 and jug_fich[2]==4 and jug_fich[3]==1:
            win=1
    elif jug_fich[0]==7 and jug_fich[1]==4 and jug_fich[3]==1:
            win=1

    elif jug_fich[0]==6 and jug_fich[1]==5 and jug_fich[2]==4:
            win=1
```

```
elif jug_fich[1]==6 and jug_fich[2]==5 and jug_fich[3]==4:
        win=1

elif jug_fich[0]==3 and jug_fich[1]==2 and jug_fich[2]==1:
        win=1
elif jug_fich[1]==3 and jug_fich[2]==2 and jug_fich[3]==1:
        win=1

else:
        if rob_fich[0]==9 and rob_fich[1]==8 and rob_fich[2]==7:
                win=2

        elif rob_fich[0]==9 and rob_fich[1]==6 and rob_fich[2]==3:
                win=2
        elif rob_fich[0]==9 and rob_fich[2]==6 and rob_fich[3]==3:
                win=2
        elif rob_fich[0]==9 and rob_fich[1]==6 and rob_fich[3]==3:
                win=2

        elif rob_fich[0]==9 and rob_fich[1]==5 and rob_fich[2]==1:
                win=2
        elif rob_fich[0]==9 and rob_fich[2]==5 and rob_fich[3]==1:
                win=2
        elif rob_fich[0]==9 and rob_fich[1]==5 and rob_fich[3]==1:
                win=2

        elif rob_fich[0]==8 and rob_fich[1]==5 and rob_fich[2]==2:
                win=2
        elif rob_fich[1]==8 and rob_fich[2]==5 and rob_fich[3]==2:
                win=2
        elif rob_fich[0]==8 and rob_fich[2]==5 and rob_fich[3]==2:
                win=2
        elif rob_fich[0]==8 and rob_fich[1]==5 and rob_fich[3]==2:
                win=2

        elif rob_fich[0]==7 and rob_fich[1]==5 and rob_fich[2]==3:
                win=2
        elif rob_fich[1]==7 and rob_fich[2]==5 and rob_fich[3]==3:
                win=2
        elif rob_fich[0]==7 and rob_fich[2]==5 and rob_fich[3]==3:
                win=2
        elif rob_fich[0]==7 and rob_fich[1]==5 and rob_fich[3]==3:
                win=2

        elif rob_fich[0]==7 and rob_fich[1]==4 and rob_fich[2]==1:
```

```
                win=2
            elif rob_fich[1]==7 and rob_fich[2]==4 and rob_fich[3]==1:
                win=2
            elif rob_fich[0]==7 and rob_fich[2]==4 and rob_fich[3]==1:
                win=2
            elif rob_fich[0]==7 and rob_fich[1]==4 and rob_fich[3]==1:
                win=2

            elif rob_fich[0]==6 and rob_fich[1]==5 and rob_fich[2]==4:
                win=2
            elif rob_fich[1]==6 and rob_fich[2]==5 and rob_fich[3]==4:
                win=2

            elif rob_fich[0]==3 and rob_fich[1]==2 and rob_fich[2]==1:
                win=2
            elif rob_fich[1]==3 and rob_fich[2]==2 and rob_fich[3]==1:
                win=2
            end
        end


        return win
  end
  global win=verGanador()
  if (win == 1):
   popup("ENHORABUENA, HA GANADO! Fin de la partida.", "Advertencia", True, False,
blocking=False)
   halt
  else:
   if (win == 2):
    popup("HE GANADO! Lo siento... Fin de la partida.", "Advertencia", True, False,
blocking=False)
    halt
   end
  end
 end
 $ 2 "BeforeStart"
 $ 3 "Script: posiciones.script"
 pos1=p[0.23323,0.26243,0.07,-3.06902,0.61313,0.01528]
 pos2=pose_add(pos1,p[-0.095,0,0,0,0,0])
 pos3=pose_add(pos1,p[-0.19503,0,0,0,0,0])
 pos4=pose_add(pos1, p[0,0.07892,0,0,0,0])
 pos5=pose_add(pos1, p[-0.095,0.07892,0,0,0,0])
 pos6=pose_add(pos1, p[-0.19503,0.07892,0,0,0,0])
 pos7=pose_add(pos1, p[0,0.14078,0,0,0,0])
```

```
pos8=pose_add(pos1, p[-0.095,0.14078,0,0,0,0])
pos9=pose_add(pos1, p[-0.19503,0.14078,0,0,0,0])
$ 4 "Script: fuerza.script"
thread force_properties():
        while (True):

        force_mode(tool_pose(),[0,0,1,0,0,0],[0.0,0.0,9.0,0.0,0.0,0.0],2,[0.01,0.01,0.15,0.17,0.
17,0.17])
                sync()
        end
end
$ 5 "Script: dibujar.script"
def dibujar(punto):
        esq1=pose_add(punto,p[0.01,-0.01,0,0,0,0])
        esq2=pose_add(punto,p[-0.01,0.01,0,0,0,0])
        esq3=pose_add(punto,p[0.01,0.01,0,0,0,0])
        esq4=pose_add(punto,p[-0.01,-0.01,0,0,0,0])
        dist1=pose_add(esq1,p[0,0,0.033,0,0,0])
        dist2=pose_add(esq2,p[0,0,0.033,0,0,0])
        dist3=pose_add(esq3,p[0,0,0.033,0,0,0])
        dist4=pose_add(esq4,p[0,0,0.033,0,0,0])

        if pieza==1:
                movej(dist1,a=1.4,v=1.05,t=0,r=0)
                global thread_force = run force_properties()
                movel(esq1,1,0.5,0,0)
                movej(esq2,1.4,1.05,0,0)
                end_force_mode()
                kill thread_force
                movel(dist2,1,0.5,0,0)
                movej(dist3,1.4,1.05,0,0)
                global thread_force = run force_properties()
                movel(esq3,1,0.5,0,0)
                movej(esq4,1.4,1.05,0,0)
                end_force_mode()
                kill thread_force
                movel(dist4,1,0.5,0,0)
        else:
                movej(dist1,1.4,1.05,0,0)
                global thread_force = run force_properties()
                movel(esq1,1,0.5,0,0)
                movec(esq4, esq2, a=1.2, v=0.1, r=0.001, mode=0)
                movec(esq3, esq1, a=1.2, v=0.10, r=0.001, mode=0)
                end_force_mode()
                kill thread_force
```

```
        end
end
$ 6 "Script: ordenar.script"
def ordenarArray(array):
        j = 0
        z = 0
        dim_array=0
        dim_array=get_list_length(array)
        while j<dim_array:
                while z<dim_array-1:
                        if array[z]<array[z+1] :
                                pos=array[z]
                                array[z]=array[z+1]
                                array[z+1]=pos
                        end
                        z=z+1
                end
                z=0
                j=j+1
        end
        return array
end
while (True):
  $ 7 "Programa de robot"
  $ 8 "MoveJ"
  $ 9 "pto_leer_fichas"
  movej(get_inverse_kin(p[.321381839247, .326101140929, .077809606357, -
1.838027758918, 1.889863374937, -.574795246001], qnear=[1.1379096508026123, -
2.5379264990436, -1.5295260588275355, -0.34863502184023076, 1.0239168405532837,
1.0820508003234863]), a=1.3962634015954636, v=1.0471975511965976)
  $ 10 "Aviso"
  popup("Coja una ficha para jugar. Cuando este listo pulse 'Continuar' para empezar la
partida.", "Mensaje", False, False, blocking=True)
  # begin: URCap Program Node
  #   Source: Robotiq_Wrist_Camera, 1.6.6, Robotiq Inc.
  #   Type: UbicaciÃ³n de CÃ¡mara
  $ 11 "Camera Locate"
  popup("Unable to initialize Vision Server. Go to Installation tab > Camera > Dashboard for
more information.. Error code: [UCC-5]", warning = False, error = True)
  halt
  # end: URCap Program Node
  $ 14 "If piezaâ‰Ÿ1"
  if (pieza == 1):
    $ 15 "Aviso"
```

```
    popup("Perfecto, yo sere las cruces. Pulse 'Continuar' para seguir jugando.", "Mensaje",
False, False, blocking=True)
    $ 16 "MoveJ"
    $ 17 "pto_leer_tabl"
    movej(get_inverse_kin(p[.140491078503, .307142425909, .167928534585, -
2.527924077601, .013659440073, -.017792940890], qnear=[1.470109462738037, -
2.0002477804767054, -2.216708485280172, 0.11703526973724365, 1.6128531694412231, -
0.07658082643617803]), a=1.3962634015954636, v=1.0471975511965976)
    $ 18 "iâ‰”0"
    global i=0
    $ 19 "Aviso"
    popup("Coloque la primera ficha sobre el tablero. Pulse 'Continuar' para seguir jugando.",
"Mensaje", False, False, blocking=True)
    # begin: URCap Program Node
    #   Source: Robotiq_Wrist_Camera, 1.6.6, Robotiq Inc.
    #   Type: UbicaciÃ³n de CÃ¡mara
    $ 20 "Camera Locate"

    ##############################################
    #######VisionLocate node start########

    # Offset in translation only.
    snapshot_position_offset[3] = 0
    snapshot_position_offset[4] = 0
    snapshot_position_offset[5] = 0
    T_camera_in_flange = p[0, 0, 0, 0, 0, 0] # enlever une fois que l'enseignement du modele
sera fait en faisant un movetool avec la pose de la camera dans le repere de la flange. Pour
l'instant, on suppose que la camera est situee directement sur la flange.
    tool_20765927 = get_T_in_base_from_flange(T_camera_in_flange)
    textmsg("actual tool flange : ", tool_20765927)
    tool_20765927 = pose_sub(tool_20765927, snapshot_position_offset)
    textmsg("tool after offset : ", tool_20765927)
    snapshot_position = p[0.140501, 0.307158, 0.167948, -2.52782, 0.0136984, -0.0177129]
    textmsg("expected snapshot position : ", snapshot_position)
    diff = pose_sub(tool_20765927, snapshot_position)
    textmsg("d765927, snapshot_position)
    textmsg("diff = ", diff)
    textmsg("norm([diff[0], diff[1], diff[2]]) = ", norm([diff[0], diff[1], diff[2]]))
    textmsg("norm([diff[3], diff[4], diff[5]]) = ", norm([diff[3], diff[4], diff[5]]))
    is_at_snapshot_position = norm([diff[0], diff[1], diff[2]]) < 0.002
    textmsg("is_at_snapshot_position = ", is_at_snapshot_position)
    is_at_snapshot_position = is_at_snapshot_position and (norm([diff[3], diff[4], diff[5]]) <
0.005)
    textmsg("is_at_snapshot_position = ", is_at_snapshot_position)
    is_snapshot_position_offset = norm(snapshot_position_offset) != 0
```

```
    is_at_snapshot_position = is_at_snapshot_position or ignore_snapshot_position
    if not(is_at_snapshot_position):
     popup("Robot is not at Snapshot Position. Add Move instruction to Snapshot Position
before Camera Locate node.. Error code: [UCC-8]", warning = False, error = True)
     halt
    end
    f=[0,0.0,0.0,0.0,0.0,0.0,0.0,0]
    object_location=p[0.0,0.0,0.0,0.0,0.0,0.0]
    f_20765927 = xmlrpc_server.findmodel("contextName-79821", tool_20765927[0],
tool_20765927[1], tool_20765927[2], tool_20765927[3], tool_20765927[4],
tool_20765927[5])
    logging_service.publish("FIND_MODEL", f_20765927)
    nb_occurence_20765927 = f_20765927[0]
    i_20765927 = 0
    object_teaching_location = p[0.1354699787603353, 0.3403290536620529, -
0.17657353008930013, -3.13719509763806, 0.03687464062995248, 0.009851919779364723]
    feature_teaching_location = p[0.14048766331351048, 0.3071671394532257,
0.16794161737473473, -2.5277982295060917, 0.013747643390065246, -
0.017677924219300607]
    # begin: URCap Program Node
    #   Source: Robotiq_Wrist_Camera, 1.6.6, Robotiq Inc.
    #   Type: Bucle de ubicaciÃ³n de cÃ¡mara
    $ 21 "Para los objeto(s) encontrados"
    while (i_20765927 < nb_occurence_20765927):
     current_object_location_20765927 =
p[f_20765927[8*i_20765927+1],f_20765927[8*i_20765927+2],f_20765927[8*i_20765927+3],
f_20765927[8*i_20765927+4],f_20765927[8*i_20765927+5],f_20765927[8*i_20765927+6]]
     object_location = current_object_location_20765927
     f[0] = f_20765927[0]
     f[1] = f_20765927[8*i_20765927+1]
     f[2] = f_20765927[8*i_20765927+2]
     f[3] = f_20765927[8*i_20765927+3]
     f[4] = f_20765927[8*i_20765927+4]
     f[5] = f_20765927[8*i_20765927+5]
     f[6] = f_20765927[8*i_20765927+6]
     f[7] = f_20765927[8*i_20765927+7]
     textmsg("current_object_location before offset = ", current_object_location_20765927)
     current_object_location_20765927 = pose_add(current_object_location_20765927,
snapshot_position_offset)
     textmsg("current_object_location after offset = ", current_object_location_20765927)
     pto_tablero = pose_trans(current_object_location_20765927,
pose_trans(pose_inv(object_teaching_location), feature_teaching_location))
     $ 22 "pos_ficha_jugâ‰”pto_tablero_var"
     global pos_ficha_jug=pto_tablero
     $ 23 "Invocar Comprobar_pos_jug"
```

```
      Comprobar_pos_jug()
      i_20765927 = i_20765927 + 1
    end
    # end: URCap Program Node
    # Restore snapshot position
    pto_tablero = p[0.140501, 0.307158, 0.167948, -2.52782, 0.0136984, -0.0177129]

    #######VisionLocate node end########
    ##########################################

    # end: URCap Program Node
    $ 24 "Invocar Colocar_rob_fich"
    Colocar_rob_fich()
    $ 25 "MoveJ"
    $ 26 "pto_leer_tabl"
    movej(get_inverse_kin(p[.140491078503, .307142425909, .167928534585, -
2.527924077601, .013659440073, -.017792940890], qnear=[1.470109462738037, -
2.0002477804767054, -2.216708485280172, 0.11703526973724365, 1.6128531694412231, -
0.07658082643617803]), a=1.3962634015954636, v=1.0471975511965976)
    $ 27 "Aviso"
    popup("Coloque la siguiente ficha. Pulse 'Continuar' para seguir jugando.", "Mensaje",
False, False, blocking=True)
    $ 28 "xâ‰”x+1"
    global x=x+1
    # begin: URCap Program Node
    #   Source: Robotiq_Wrist_Camera, 1.6.6, Robotiq Inc.
    #   Type: UbicaciÃ³n de CÃ¡mara
    $ 29 "Camera Locate"

    ##########################################
    #######VisionLocate node start########

    # Offset in translation only.
    snapshot_position_offset[3] = 0
    snapshot_position_offset[4] = 0
    snapshot_position_offset[5] = 0
    T_camera_in_flange = p[0, 0, 0, 0, 0, 0] # enlever une fois que l'enseignement du modele
sera fait en faisant un movetool avec la pose de la camera dans le repere de la flange. Pour
l'instant, on suppose que la camera est situee directement sur la flange.
    tool_11011158 = get_T_in_base_from_flange(T_camera_in_flange)
    textmsg("actual tool flange : ", tool_11011158)
    tool_11011158 = pose_sub(tool_11011158, snapshot_position_offset)
    textmsg("tool after offset : ", tool_11011158)
    snapshot_position = p[0.140501, 0.307158, 0.167948, -2.52782, 0.0136984, -0.0177129]
    textmsg("expected snapshot position : ", snapshot_position)
```

```
    diff = pose_sub(tool_11011158, snapshot_position)
    textmsg("diff = ", diff)
    textmsg("norm([diff[0], diff[1], diff[2]]) = ", norm([diff[0], diff[1], diff[2]]))
    textmsg("norm([diff[3], diff[4], diff[5]]) = ", norm([diff[3], diff[4], diff[5]]))
    is_at_snapshot_position = norm([diff[0], diff[1], diff[2]]) < 0.002
    textmsg("is_at_snapshot_position = ", is_at_snapshot_position)
    is_at_snapshot_position = is_at_snapshot_position and (norm([diff[3], diff[4], diff[5]]) <
0.005)
    textmsg("is_at_snapshot_position = ", is_at_snapshot_position)
    is_snapshot_position_offset = norm(snapshot_position_offset) != 0
    is_at_snapshot_position = is_at_snapshot_position or ignore_snapshot_position
    if not(is_at_snapshot_position):
     popup("Robot is not at Snapshot Position. Add Move instruction to Snapshot Position
before Camera Locate node.. Error code: [UCC-8]", warning = False, error = True)
     halt
    end
    f=[0,0.0,0.0,0.0,0.0,0.0,0.0,0]
    object_location=p[0.0,0.0,0.0,0.0,0.0,0.0]
    f_11011158 = xmlrpc_server.findmodel("contextName-3947", tool_11011158[0],
tool_11011158[1], tool_11011158[2], tool_11011158[3], tool_11011158[4],
tool_11011158[5])
    logging_service.publish("FIND_MODEL", f_11011158)
    nb_occurence_11011158 = f_11011158[0]
    i_11011158 = 0
    object_teaching_location = p[0.1354699787603353, 0.3403290536620529, -
0.17657353008930013, -3.13719509763806, 0.03687464062995248, 0.009851919779364723]
    feature_teaching_location = p[0.14048766331351048, 0.3071671394532257,
0.16794161737473473, -2.5277982295060917, 0.013747643390065246, -
0.017677924219300607]
    # begin: URCap Program Node
    #   Source: Robotiq_Wrist_Camera, 1.6.6, Robotiq Inc.
    #   Type: Bucle de ubicaciÃ³n de cÃ¡mara
    $ 30 "Para los objeto(s) encontrados"
    while (i_11011158 < nb_occurence_11011158):
     current_object_location_11011158 =
p[f_11011158[8*i_11011158+1],f_11011158[8*i_11011158+2],f_11011158[8*i_11011158+3],
f_11011158[8*i_11011158+4],f_11011158[8*i_11011158+5],f_11011158[8*i_11011158+6]]
     object_location = current_object_location_11011158
     f[0] = f_11011158[0]
     f[1] = f_11011158[8*i_11011158+1]
     f[2] = f_11011158[8*i_11011158+2]
     f[3] = f_11011158[8*i_11011158+3]
     f[4] = f_11011158[8*i_11011158+4]
     f[5] = f_11011158[8*i_11011158+5]
     f[6] = f_11011158[8*i_11011158+6]
```

```
      f[7] = f_11011158[8*i_11011158+7]
      textmsg("current_object_location before offset = ", current_object_location_11011158)
      current_object_location_11011158 = pose_add(current_object_location_11011158,
snapshot_position_offset)
      textmsg("current_object_location after offset = ", current_object_location_11011158)
      pto_tablero = pose_trans(current_object_location_11011158,
pose_trans(pose_inv(object_teaching_location), feature_teaching_location))
      $ 31 "If i<2"
      if (i<2):
        $ 32 "pos_ficha_jugâ‰”pto_tablero_var"
        global pos_ficha_jug=pto_tablero
        $ 33 "Invocar Comprobar_pos_jug"
        Comprobar_pos_jug()
      end
      $ 34 "iâ‰”i+1"
      global i=i+1
      i_11011158 = i_11011158 + 1
    end
    # end: URCap Program Node
    # Restore snapshot position
    pto_tablero = p[0.140501, 0.307158, 0.167948, -2.52782, 0.0136984, -0.0177129]

    #######VisionLocate node end########
    ##########################################

    # end: URCap Program Node
    $ 35 "jug_fichâ‰”ordenarArray(jug_fich)"
    global jug_fich=ordenarArray(jug_fich)
    $ 36 "Invocar Huecos_libres"
    Huecos_libres()
    $ 37 "libreâ‰”ordenarArray(libre)"
    global libre=ordenarArray(libre)
    $ 38 "Invocar Colocar_rob_fich2"
    Colocar_rob_fich2()
    $ 39 "MoveJ"
    $ 40 "pto_leer_tabl"
    movej(get_inverse_kin(p[.140491078503, .307142425909, .167928534585, -
2.527924077601, .013659440073, -.017792940890], qnear=[1.470109462738037, -
2.0002477804767054, -2.216708485280172, 0.11703526973724365, 1.6128531694412231, -
0.07658082643617803]), a=1.3962634015954636, v=1.0471975511965976)
    $ 41 "Aviso"
    popup("Coloque la siguiente ficha. Pulse 'Continuar' para seguir jugando.", "Mensaje",
False, False, blocking=True)
    $ 42 "iâ‰”0"
    global i=0
```

```
    $ 43 "xâ‰”x+1"
    global x=x+1
    # begin: URCap Program Node
    #   Source: Robotiq_Wrist_Camera, 1.6.6, Robotiq Inc.
    #   Type: UbicaciÃ³n de CÃ¡mara
    $ 44 "Camera Locate"

    ##############################################
    #######VisionLocate node start########

    # Offset in translation only.
    snapshot_position_offset[3] = 0
    snapshot_position_offset[4] = 0
    snapshot_position_offset[5] = 0
    T_camera_in_flange = p[0, 0, 0, 0, 0, 0] # enlever une fois que l'enseignement du modele sera fait en faisant un movetool avec la pose de la camera dans le repere de la flange. Pour l'instant, on suppose que la camera est situee directement sur la flange.
    tool_11353532 = get_T_in_base_from_flange(T_camera_in_flange)
    textmsg("actual tool flange : ", tool_11353532)
    tool_11353532 = pose_sub(tool_11353532, snapshot_position_offset)
    textmsg("tool after offset : ", tool_11353532)
    snapshot_position = p[0.140501, 0.307158, 0.167948, -2.52782, 0.0136984, -0.0177129]
    textmsg("expected snapshot position : ", snapshot_position)
    diff = pose_sub(tool_11353532, snapshot_position)
    textmsg("diff = ", diff)
    textmsg("norm([diff[0], diff[1], diff[2]]) = ", norm([diff[0], diff[1], diff[2]]))
    textmsg("norm([diff[3], diff[4], diff[5]]) = ", norm([diff[3], diff[4], diff[5]]))
    is_at_snapshot_position = norm([diff[0], diff[1], diff[2]]) < 0.002
    textmsg("is_at_snapshot_position = ", is_at_snapshot_position)
    is_at_snapshot_position = is_at_snapshot_position and (norm([diff[3], diff[4], diff[5]]) < 0.005)
    textmsg("is_at_snapshot_position = ", is_at_snapshot_position)
    is_snapshot_position_offset = norm(snapshot_position_offset) != 0
    is_at_snapshot_position = is_at_snapshot_position or ignore_snapshot_position
    if not(is_at_snapshot_position):
      popup("Robot is not at Snapshot Position. Add Move instruction to Snapshot Position before Camera Locate node.. Error code: [UCC-8]", warning = False, error = True)
      halt
    end
    f=[0,0.0,0.0,0.0,0.0,0.0,0.0,0]
    object_location=p[0.0,0.0,0.0,0.0,0.0,0.0]
    f_11353532 = xmlrpc_server.findmodel("contextName-78897", tool_11353532[0], tool_11353532[1], tool_11353532[2], tool_11353532[3], tool_11353532[4], tool_11353532[5])
    logging_service.publish("FIND_MODEL", f_11353532)
```

```
    nb_occurence_11353532 = f_11353532[0]
    i_11353532 = 0
    object_teaching_location = p[0.1354699787603353, 0.3403290536620529, -
0.17657353008930013, -3.13719509763806, 0.03687464062995248, 0.009851919779364723]
    feature_teaching_location = p[0.14048766331351048, 0.3071671394532257,
0.16794161737473473, -2.5277982295060917, 0.013747643390065246, -
0.017677924219300607]
    # begin: URCap Program Node
    #   Source: Robotiq_Wrist_Camera, 1.6.6, Robotiq Inc.
    #   Type: Bucle de ubicaciÃ³n de cÃ¡mara
    $ 45 "Para los objeto(s) encontrados"
    while (i_11353532 < nb_occurence_11353532):
      current_object_location_11353532 =
p[f_11353532[8*i_11353532+1],f_11353532[8*i_11353532+2],f_11353532[8*i_11353532+3],
f_11353532[8*i_11353532+4],f_11353532[8*i_11353532+5],f_11353532[8*i_11353532+6]]
      object_location = current_object_location_11353532
      f[0] = f_11353532[0]
      f[1] = f_11353532[8*i_11353532+1]
      f[2] = f_11353532[8*i_11353532+2]
      f[3] = f_11353532[8*i_11353532+3]
      f[4] = f_11353532[8*i_11353532+4]
      f[5] = f_11353532[8*i_11353532+5]
      f[6] = f_11353532[8*i_11353532+6]
      f[7] = f_11353532[8*i_11353532+7]
      textmsg("current_object_location before offset = ", current_object_location_11353532)
      current_object_location_11353532 = pose_add(current_object_location_11353532,
snapshot_position_offset)
      textmsg("current_object_location after offset = ", current_object_location_11353532)
      pto_tablero = pose_trans(current_object_location_11353532,
pose_trans(pose_inv(object_teaching_location), feature_teaching_location))
      $ 46 "If i<3"
      if (i<3):
        $ 47 "pos_ficha_jugâ‰”pto_tablero_var"
        global pos_ficha_jug=pto_tablero
        $ 48 "Invocar Comprobar_pos_jug"
        Comprobar_pos_jug()
      end
      $ 49 "iâ‰”i+1"
      global i=i+1
      i_11353532 = i_11353532 + 1
    end
    # end: URCap Program Node
    # Restore snapshot position
    pto_tablero = p[0.140501, 0.307158, 0.167948, -2.52782, 0.0136984, -0.0177129]
```

```
    #######VisionLocate node end########
    ############################################

    # end: URCap Program Node
    $ 50 "jug_fichâ‰”ordenarArray(jug_fich)"
    global jug_fich=ordenarArray(jug_fich)
    $ 51 "Invocar Huecos_libres"
    Huecos_libres()
    $ 52 "libreâ‰”ordenarArray(libre)"
    global libre=ordenarArray(libre)
    $ 53 "Invocar Colocar_rob_fich3"
    Colocar_rob_fich3()
    $ 54 "rob_fichâ‰”ordenarArray(rob_fich)"
    global rob_fich=ordenarArray(rob_fich)
    $ 55 "Invocar Comprobar_ganador"
    Comprobar_ganador()
    $ 56 "MoveJ"
    $ 57 "pto_leer_tabl"
    movej(get_inverse_kin(p[.140491078503, .307142425909, .167928534585, -
2.527924077601, .013659440073, -.017792940890], qnear=[1.470109462738037, -
2.0002477804767054, -2.216708485280172, 0.11703526973724365, 1.6128531694412231, -
0.07658082643617803]), a=1.3962634015954636, v=1.0471975511965976)
    $ 58 "Aviso"
    popup("Coloque la ultima ficha. Pulse 'Continuar' para seguir jugando.", "Mensaje", False,
False, blocking=True)
    $ 59 "iâ‰”0"
    global i=0
    $ 60 "xâ‰”x+1"
    global x=x+1
    # begin: URCap Program Node
    #   Source: Robotiq_Wrist_Camera, 1.6.6, Robotiq Inc.
    #   Type: UbicaciÃ³n de CÃ¡mara
    $ 61 "Camera Locate"

    ############################################
    #######VisionLocate node start########

    # Offset in translation only.
    snapshot_position_offset[3] = 0
    snapshot_position_offset[4] = 0
    snapshot_position_offset[5] = 0
    T_camera_in_flange = p[0, 0, 0, 0, 0, 0] # enlever une fois que l'enseignement du modele
sera fait en faisant un movetool avec la pose de la camera dans le repere de la flange. Pour
l'instant, on suppose que la camera est situee directement sur la flange.
    tool_31818774 = get_T_in_base_from_flange(T_camera_in_flange)
```

```
    textmsg("actual tool flange : ", tool_31818774)
    tool_31818774 = pose_sub(tool_31818774, snapshot_position_offset)
    textmsg("tool after offset : ", tool_31818774)
    snapshot_position = p[0.140501, 0.307158, 0.167948, -2.52782, 0.0136984, -0.0177129]
    textmsg("expected snapshot position : ", snapshot_position)
    diff = pose_sub(tool_31818774, snapshot_position)
    textmsg("diff = ", diff)
    textmsg("norm([diff[0], diff[1], diff[2]]) = ", norm([diff[0], diff[1], diff[2]]))
    textmsg("noriff[3], diff[4], diff[5]]))
    is_at_snapshot_position = norm([diff[0], diff[1], diff[2]]) < 0.002
    textmsg("is_at_snapshot_position = ", is_at_snapshot_position)
    is_at_snapshot_position = is_at_snapshot_position and (norm([diff[3], diff[4], diff[5]]) <
0.005)
    textmsg("is_at_snapshot_position = ", is_at_snapshot_position)
    is_snapshot_position_offset = norm(snapshot_position_offset) != 0
    is_at_snapshot_position = is_at_snapshot_position or ignore_snapshot_position
    if not(is_at_snapshot_position):
     popup("Robot is not at Snapshot Position. Add Move instruction to Snapshot Position
before Camera Locate node.. Error code: [UCC-8]", warning = False, error = True)
     halt
    end
    f=[0,0.0,0.0,0.0,0.0,0.0,0.0,0]
    object_location=p[0.0,0.0,0.0,0.0,0.0,0.0]
    f_31818774 = xmlrpc_server.findmodel("contextName-35294", tool_31818774[0],
tool_31818774[1], tool_31818774[2], tool_31818774[3], tool_31818774[4],
tool_31818774[5])
    logging_service.publish("FIND_MODEL", f_31818774)
    nb_occurence_31818774 = f_31818774[0]
    i_31818774 = 0
    object_teaching_location = p[0.1354699787603353, 0.3403290536620529, -
0.17657353008930013, -3.13719509763806, 0.03687464062995248, 0.009851919779364723]
    feature_teaching_location = p[0.14048766331351048, 0.3071671394532257,
0.16794161737473473, -2.5277982295060917, 0.013747643390065246, -
0.017677924219300607]
    # begin: URCap Program Node
    #   Source: Robotiq_Wrist_Camera, 1.6.6, Robotiq Inc.
    #   Type: Bucle de ubicaciÃ³n de cÃ¡mara
    $ 62 "Para los objeto(s) encontrados"
    while (i_31818774 < nb_occurence_31818774):
     current_object_location_31818774 =
p[f_31818774[8*i_31818774+1],f_31818774[8*i_31818774+2],f_31818774[8*i_31818774+3],
f_31818774[8*i_31818774+4],f_31818774[8*i_31818774+5],f_31818774[8*i_31818774+6]]
     object_location = current_object_location_31818774
     f[0] = f_31818774[0]
     f[1] = f_31818774[8*i_31818774+1]
```

```
      f[2] = f_31818774[8*i_31818774+2]
      f[3] = f_31818774[8*i_31818774+3]
      f[4] = f_31818774[8*i_31818774+4]
      f[5] = f_31818774[8*i_31818774+5]
      f[6] = f_31818774[8*i_31818774+6]
      f[7] = f_31818774[8*i_31818774+7]
      textmsg("current_object_location before offset = ", current_object_location_31818774)
      current_object_location_31818774 = pose_add(current_object_location_31818774,
snapshot_position_offset)
      textmsg("current_object_location after offset = ", current_object_location_31818774)
      pto_tablero = pose_trans(current_object_location_31818774,
pose_trans(pose_inv(object_teaching_location), feature_teaching_location))
      $ 63 "If i<4"
      if (i<4):
        $ 64 "pos_ficha_jugâ‰”pto_tablero_var"
        global pos_ficha_jug=pto_tablero
        $ 65 "Invocar Comprobar_pos_jug"
        Comprobar_pos_jug()
      end
      $ 66 "iâ‰”i+1"
      global i=i+1
      i_31818774 = i_31818774 + 1
    end
    # end: URCap Program Node
    # Restore snapshot position
    pto_tablero = p[0.140501, 0.307158, 0.167948, -2.52782, 0.0136984, -0.0177129]

    #######VisionLocate node end########
    ###########################################

    # end: URCap Program Node
    $ 67 "jug_fichâ‰”ordenarArray(jug_fich)"
    global jug_fich=ordenarArray(jug_fich)
    $ 68 "Invocar Huecos_libres"
    Huecos_libres()
    $ 69 "Invocar Comprobar_ganador"
    Comprobar_ganador()
    $ 70 "libreâ‰”ordenarArray(libre)"
    global libre=ordenarArray(libre)
    $ 71 "Invocar Colocar_rob_fich4"
    Colocar_rob_fich4()
    $ 72 "rob_fichâ‰”ordenarArray(rob_fich)"
    global rob_fich=ordenarArray(rob_fich)
    $ 73 "Invocar Comprobar_ganador"
    Comprobar_ganador()
```

```
    $ 74 "MoveJ"
    $ 75 "pto_leer_tabl"
    movej(get_inverse_kin(p[.140491078503, .307142425909, .167928534585, -
2.527924077601, .013659440073, -.017792940890], qnear=[1.470109462738037, -
2.0002477804767054, -2.216708485280172, 0.11703526973724365, 1.6128531694412231, -
0.07658082643617803]), a=1.3962634015954636, v=1.0471975511965976)
    $ 76 "If winâ‰Ÿ0"
    if (win == 0):
      $ 77 "Aviso"
      popup("EMPATE, BIEN JUGADO! Fin de la partida.", "Advertencia", True, False,
blocking=False)
      halt
    end
  else:
    $ 78 "Else"
    $ 79 "Aviso"
    popup("Perfecto, yo sere los circulos. Pulse 'Continuar' para seguir jugando.", "Mensaje",
False, False, blocking=True)
    $ 80 "MoveJ"
    $ 81 "pto_leer_tabl"
    movej(get_inverse_kin(p[.140491078503, .307142425909, .167928534585, -
2.527924077601, .013659440073, -.017792940890], qnear=[1.470109462738037, -
2.0002477804767054, -2.216708485280172, 0.11703526973724365, 1.6128531694412231, -
0.07658082643617803]), a=1.3962634015954636, v=1.0471975511965976)
    $ 82 "iâ‰”0"
    global i=0
    $ 83 "Aviso"
    popup("Coloque la primera ficha sobre el tablero. Pulse 'Continuar' para seguir jugando.",
"Mensaje", False, False, blocking=True)
    # begin: URCap Program Node
    #   Source: Robotiq_Wrist_Camera, 1.6.6, Robotiq Inc.
    #   Type: UbicaciÃ³n de CÃ¡mara
    $ 84 "Camera Locate"

    ##########################################
    #######VisionLocate node start########

    # Offset in translation only.
    snapshot_position_offset[3] = 0
    snapshot_position_offset[4] = 0
    snapshot_position_offset[5] = 0
    T_camera_in_flange = p[0, 0, 0, 0, 0, 0] # enlever une fois que l'enseignement du modele
sera fait en faisant un movetool avec la pose de la camera dans le repere de la flange. Pour
l'instant, on suppose que la camera est situee directement sur la flange.
    tool_31656700 = get_T_in_base_from_flange(T_camera_in_flange)
```

```
    textmsg("actual tool flange : ", tool_31656700)
    tool_31656700 = pose_sub(tool_31656700, snapshot_position_offset)
    textmsg("tool after offset : ", tool_31656700)
    snapshot_position = p[0.140501, 0.307158, 0.167948, -2.52782, 0.0136984, -0.0177129]
    textmsg("expected snapshot position : ", snapshot_position)
    diff = pose_sub(tool_31656700, snapshot_position)
    textmsg("diff = ", diff)
    textmsg("norm([diff[0], diff[1], diff[2]]) = ", norm([diff[0], diff[1], diff[2]]))
    textmsg("norm([diff[3], diff[4], diff[5]]) = ", norm([diff[3], diff[4], diff[5]]))
    is_at_snapshot_position = norm([diff[0], diff[1], diff[2]]) < 0.002
    textmsg("is_at_snapshot_position = ", is_at_snapshot_position)
    is_at_snapshot_position = is_at_snapshot_position and (norm([diff[3], diff[4], diff[5]]) <
0.005)
    textmsg("is_at_snapshot_position = ", is_at_snapshot_position)
    is_snapshot_position_offset = norm(snapshot_position_offset) != 0
    is_at_snapshot_position = is_at_snapshot_position or ignore_snapshot_position
    if not(is_at_snapshot_position):
     popup("Robot is not at Snapshot Position. Add Move instruction to Snapshot Position
before Camera Locate node.. Error code: [UCC-8]", warning = False, error = True)
     halt
    end
    f=[0,0.0,0.0,0.0,0.0,0.0,0.0,0]
    object_location=p[0.0,0.0,0.0,0.0,0.0,0.0]
    f_31656700 = xmlrpc_server.findmodel("contextName-47251", tool_31656700[0],
tool_31656700[1], tool_31656700[2], tool_31656700[3], tool_31656700[4],
tool_31656700[5])
    logging_service.publish("FIND_MODEL", f_31656700)
    nb_occurence_31656700 = f_31656700[0]
    i_31656700 = 0
    object_teaching_location = p[0.14457822971951348, 0.3282708363187259, -
0.1772027020943127, -3.137195097638309, 0.03687464060942577, 0.009851919779365375]
    feature_teaching_location = p[0.14048766331351048, 0.3071671394532257,
0.16794161737473473, -2.5277982295060917, 0.013747643390065246, -
0.017677924219300607]
    # begin: URCap Program Node
    #   Source: Robotiq_Wrist_Camera, 1.6.6, Robotiq Inc.
    #   Type: Bucle de ubicaciÃ³n de cÃ¡mara
    $ 85 "Para los objeto(s) encontrados"
    while (i_31656700 < nb_occurence_31656700):
     current_object_location_31656700 =
p[f_31656700[8*i_31656700+1],f_31656700[8*i_31656700+2],f_31656700[8*i_31656700+3],
f_31656700[8*i_31656700+4],f_31656700[8*i_31656700+5],f_31656700[8*i_31656700+6]]
     object_location = current_object_location_31656700
     f[0] = f_31656700[0]
     f[1] = f_31656700[8*i_31656700+1]
```

```
      f[2] = f_31656700[8*i_31656700+2]
      f[3] = f_31656700[8*i_31656700+3]
      f[4] = f_31656700[8*i_31656700+4]
      f[5] = f_31656700[8*i_31656700+5]
      f[6] = f_31656700[8*i_31656700+6]
      f[7] = f_31656700[8*i_31656700+7]
      textmsg("current_object_location before offset = ", current_object_location_31656700)
      current_object_location_31656700 = pose_add(current_object_location_31656700,
snapshot_position_offset)
      textmsg("current_object_location after offset = ", current_object_location_31656700)
      pto_tablero = pose_trans(current_object_location_31656700,
pose_trans(pose_inv(object_teaching_location), feature_teaching_location))
      $ 86 "pos_ficha_jugâ‰”pto_tablero_var"
      global pos_ficha_jug=pto_tablero
      $ 87 "Invocar Comprobar_pos_jug"
      Comprobar_pos_jug()
      i_31656700 = i_31656700 + 1
    end
    # end: URCap Program Node
    # Restore snapshot position
    pto_tablero = p[0.140501, 0.307158, 0.167948, -2.52782, 0.0136984, -0.0177129]

    #######VisionLocate node end########
    ##########################################

    # end: URCap Program Node
    $ 88 "Invocar Colocar_rob_fich"
    Colocar_rob_fich()
    $ 89 "MoveJ"
    $ 90 "pto_leer_tabl"
    movej(get_inverse_kin(p[.140491078503, .307142425909, .167928534585, -
2.527924077601, .013659440073, -.017792940890], qnear=[1.470109462738037, -
2.0002477804767054, -2.216708485280172, 0.11703526973724365, 1.6128531694412231, -
0.07658082643617803]), a=1.3962634015954636, v=1.0471975511965976)
    $ 91 "Aviso"
    popup("Coloque la siguiente ficha. Pulse 'Continuar' para seguir jugando.", "Mensaje",
False, False, blocking=True)
    $ 92 "xâ‰”x+1"
    global x=x+1
    # begin: URCap Program Node
    #   Source: Robotiq_Wrist_Camera, 1.6.6, Robotiq Inc.
    #   Type: UbicaciÃ³n de CÃ¡mara
    $ 93 "Camera Locate"

    ##########################################
```

```
    #######VisionLocate node start########

    # Offset in translation only.
    snapshot_position_offset[3] = 0
    snapshot_position_offset[4] = 0
    snapshot_position_offset[5] = 0
    T_camera_in_flange = p[0, 0, 0, 0, 0, 0] # enlever une fois que l'enseignement du modele sera fait en faisant un movetool avec la pose de la camera dans le repere de la flange. Pour l'instant, on suppose que la camera est situee directement sur la flange.
    tool_9449721 = get_T_in_base_from_flange(T_camera_in_flange)
    textmsg("actual tool flange : ", tool_9449721)
    tool_9449721 = pose_sub(tool_9449721, snapshot_position_offset)
    textmsg("tool after offset : ", tool_9449721)
    snapshot_position = p[0.140501, 0.307158, 0.167948, -2.52782, 0.0136984, -0.0177129]
    textmsg("expected snapshot position : ", snapshot_position)
    diff = pose_sub(tool_9449721, snapshot_position)
    textmsg("diff = ", diff)
    textmsg("norm([diff[0], diff[1], diff[2]]) = ", norm([diff[0], diff[1], diff[2]]))
    textmsg("norm([diff[3], diff[4], diff[5]]) = ", norm([diff[3], diff[4], diff[5]]))
    is_at_snapshot_position = norm([diff[0], diff[1], diff[2]]) < 0.002
    textmsg("is_at_snapshot_position = ", is_at_snapshot_position)
    is_at_snapshot_position = is_at_snapshot_position and (norm([diff[3], diff[4], diff[5]]) < 0.005)
    textmsg("is_at_snapshot_position = ", is_at_snapshot_position)
    is_snapshot_position_offset = norm(snapshot_position_offset) != 0
    is_at_snapshot_position = is_at_snapshot_position or ignore_snapshot_position
    if not(is_at_snapshot_position):
      popup("Robot is not at Snapshot Position. Add Move instruction to Snapshot Position before Camera Locate node.. Error code: [UCC-8]", warning = False, error = True)
      halt
    end
    f=[0,0.0,0.0,0.0,0.0,0.0,0.0,0]
    object_location=p[0.0,0.0,0.0,0.0,0.0,0.0]
    f_9449721 = xmlrpc_server.findmodel("contextName-51154", tool_9449721[0], tool_9449721[1], tool_9449721[2], tool_9449721[3], tool_9449721[4], tool_9449721[5])
    logging_service.publish("FIND_MODEL", f_9449721)
    nb_occurence_9449721 = f_9449721[0]
    i_9449721 = 0
    object_teaching_location = p[0.14457822971951348, 0.3282708363187259, -0.1772027020943127, -3.137195097638309, 0.03687464060942577, 0.009851919779365375]
    feature_teaching_location = p[0.14048766331351048, 0.3071671394532257, 0.16794161737473473, -2.5277982295060917, 0.013747643390065246, -0.017677924219300607]
    # begin: URCap Program Node
    #   Source: Robotiq_Wrist_Camera, 1.6.6, Robotiq Inc.
```

```
    #   Type: Bucle de ubicaciÃ³n de cÃ¡mara
    $ 94 "Para los objeto(s) encontrados"
    while (i_9449721 < nb_occurence_9449721):
     current_object_location_9449721 =
p[f_9449721[8*i_9449721+1],f_9449721[8*i_9449721+2],f_9449721[8*i_9449721+3],f_9449
721[8*i_9449721+4],f_9449721[8*i_9449721+5],f_9449721[8*i_9449721+6]]
     object_location = current_object_location_9449721
     f[0] = f_9449721[0]
     f[1] = f_9449721[8*i_9449721+1]
     f[2] = f_9449721[8*i_9449721+2]
     f[3] = f_9449721[8*i_9449721+3]
     f[4] = f_9449721[8*i_9449721+4]
     f[5] = f_9449721[8*i_9449721+5]
     f[6] = f_9449721[8*i_9449721+6]
     f[7] = f_9449721[8*i_9449721+7]
     textmsg("current_object_location before offset = ", current_object_location_9449721)
     current_object_location_9449721 = pose_add(current_object_location_9449721,
snapshot_position_offset)
     textmsg("current_object_location after offset = ", current_object_location_9449721)
     pto_tablero = pose_trans(current_object_location_9449721,
pose_trans(pose_inv(object_teaching_location), feature_teaching_location))
     $ 95 "If i<2"
     if (i<2):
      $ 96 "pos_ficha_jugâ‰”pto_tablero_var"
      global pos_ficha_jug=pto_tablero
      $ 97 "Invocar Comprobar_pos_jug"
      Comprobar_pos_jug()
     end
     $ 98 "iâ‰”i+1"
     global i=i+1
     i_9449721 = i_9449721 + 1
    end
    # end: URCap Program Node
    # Restore snapshot position
    pto_tablero = p[0.140501, 0.307158, 0.167948, -2.52782, 0.0136984, -0.0177129]

    #######VisionLocate node end########
    ##############################################

    # end: URCap Program Node
    $ 99 "jug_fichâ‰”ordenarArray(jug_fich)"
    global jug_fich=ordenarArray(jug_fich)
    $ 100 "Invocar Huecos_libres"
    Huecos_libres()
    $ 101 "libreâ‰”ordenarArray(libre)"
```

```
    global libre=ordenarArray(libre)
    $ 102 "Invocar Colocar_rob_fich2"
    Colocar_rob_fich2()
    $ 103 "MoveJ"
    $ 104 "pto_leer_tabl"
    movej(get_inverse_kin(p[.140491078503, .307142425909, .167928534585, -
2.527924077601, .013659440073, -.017792940890], qnear=[1.470109462738037, -
2.0002477804767054, -2.216708485280172, 0.11703526973724365, 1.6128531694412231, -
0.07658082643617803]), a=1.3962634015954636, v=1.0471975511965976)
    $ 105 "Aviso"
    popup("Coloque la siguiente ficha. Pulse 'Continuar' para seguir jugando.", "Mensaje",
False, False, blocking=True)
    $ 106 "iâ‰”0"
    global i=0
    $ 107 "xâ‰”x+1"
    global x=x+1
    # begin: URCap Program Node
    #   Source: Robotiq_Wrist_Camera, 1.6.6, Robotiq Inc.
    #   Type: UbicaciÃ³n de CÃ¡mara
    $ 108 "Camera Locate"

    ##########################################
    #######VisionLocate node start########

    # Offset in translation only.
    snapshot_position_offset[3] = 0
    snapshot_position_offset[4] = 0
    snapshot_position_offset[5] = 0
    T_camera_in_flange = p[0, 0, 0, 0, 0, 0] # enlever une fois que l'enseignement du modele
sera fait en faisant un movetool avec la pose de la camera dans le repere de la flange. Pour
l'instant, on suppose que la camera est situee directement sur la flange.
    tool_19940356 = get_T_in_base_from_flange(T_camera_in_flange)
    textmsg("actual tool flange : ", tool_19940356)
    tool_19940356 = pose_sub(tool_19940356, snapshot_position_offset)
    textmsg("tool after offset : ", tool_19940356)
    snapshot_position = p[0.140501, 0.307158, 0.167948, -2.52782, 0.0136984, -0.0177129]
    textmsg("expected snapshot position : ", snapshot_position)
    diff = pose_sub(tool_19940356, snapshot_position)
    textmsg("diff = ", diff)
    textmsg("norm([diff[0], diff[1], diff[2]]) = ", norm([diff[0], diff[1], diff[2]]))
    textmsg("norm([diff[3], diff[4], diff[5]]) = ", norm([diff[3], diff[4], diff[5]]))
    is_at_snapshot_position = norm([diff[0], diff[1], diff[2]]) < 0.002
    textmsg("is_at_snapshot_position = ", is_at_snapshot_position)
    is_at_snapshot_position = is_at_snapshot_position and (norm([diff[3], diff[4], diff[5]]) <
0.005)
```

```
    textmsg("is_at_snapshot_position = ", is_at_snapshot_position)
    is_snapshot_position_offset = norm(snapshot_position_offset) != 0
    is_at_snapshot_position = is_at_snapshot_position or ignore_snapshot_position
    if not(is_at_snapshot_position):
     popup("Robot is not at Snapshot Position. Add Move instruction to Snapshot Position
before Camera Locate node.. Error code: [UCC-8]", warning = False, error = True)
     halt
    end
    f=[0,0.0,0.0,0.0,0.0,0.0,0.0,0]
    object_location=p[0.0,0.0,0.0,0.0,0.0,0.0]
    f_19940356 = xmlrpc_server.findmodel("contextName-70990", tool_19940356[0],
tool_19940356[1], tool_19940356[2], tool_19940356[3], tool_19940356[4],
tool_19940356[5])
    logging_service.publish("FIND_MODEL", f_19940356)
    nb_occurence_19940356 = f_19940356[0]
    i_19940356 = 0
    object_teaching_location = p[0.14457822971951348, 0.3282708363187259, -
0.1772027020943127, -3.137195097638309, 0.03687464060942577, 0.009851919779365375]
    feature_teaching_location = p[0.14048766331351048, 0.3071671394532257,
0.16794161737473473, -2.5277982295060917, 0.013747643390065246, -
0.017677924219300607]
    # begin: URCap Program Node
    #   Source: Robotiq_Wrist_Camera, 1.6.6, Robotiq Inc.
    #   Type: Bucle de ubicaciÃ³n de cÃ¡mara
    $ 109 "Para los objeto(s) encontrados"
    while (i_19940356 < nb_occurence_19940356):
     current_object_location_19940356 =
p[f_19940356[8*i_19940356+1],f_19940356[8*i_19940356+2],f_19940356[8*i_19940356+3],
f_19940356[8*i_19940356+4],f_19940356[8*i_19940356+5],f_19940356[8*i_19940356+6]]
     object_location = current_object_location_19940356
     f[0] = f_19940356[0]
     f[1] = f_19940356[8*i_19940356+1]
     f[2] = f_19940356[8*i_19940356+2]
     f[3] = f_19940356[8*i_19940356+3]
     f[4] = f_19940356[8*i_19940356+4]
     f[5] = f_19940356[8*i_19940356+5]
     f[6] = f_19940356[8*i_19940356+6]
     f[7] = f_19940356[8*i_19940356+7]
     textmsg("current_object_location before offset = ", current_object_location_19940356)
     current_object_location_19940356 = pose_add(current_object_location_19940356,
snapshot_position_offset)
     textmsg("current_object_location after offset = ", current_object_location_19940356)
     pto_tablero = pose_trans(current_object_location_19940356,
pose_trans(pose_inv(object_teaching_location), feature_teaching_location))
     $ 110 "If i<3"
```

```
    if (i<3):
      $ 111 "pos_ficha_jugâ‰”pto_tablero_var"
      global pos_ficha_jug=pto_tablero
      $ 112 "Invocar Comprobar_pos_jug"
      Comprobar_pos_jug()
    end
    $ 113 "iâ‰”i+1"
    global i=i+1
    i_19940356 = i_19940356 + 1
  end
  # end: URCap Program Node
  # Restore snapshot position
  pto_tablero = p[0.140501, 0.307158, 0.167948, -2.52782, 0.0136984, -0.0177129]

  #######VisionLocate node end########
  ##########################################

  # end: URCap Program Node
  $ 114 "jug_fichâ‰”ordenarArray(jug_fich)"
  global jug_fich=ordenarArray(jug_fich)
  $ 115 "Invocar Huecos_libres"
  Huecos_libres()
  $ 116 "libreâ‰”ordenarArray(libre)"
  global libre=ordenarArray(libre)
  $ 117 "Invocar Colocar_rob_fich3"
  Colocar_rob_fich3()
  $ 118 "rob_fichâ‰”ordenarArray(rob_fich)"
  global rob_fich=ordenarArray(rob_fich)
  $ 119 "Invocar Comprobar_ganador"
  Comprobar_ganador()
  $ 120 "MoveJ"
  $ 121 "pto_leer_tabl"
  movej(get_inverse_kin(p[.140491078503, .307142425909, .167928534585, -
2.527924077601, .013659440073, -.017792940890], qnear=[1.470109462738037, -
2.0002477804767054, -2.216708485280172, 0.11703526973724365, 1.6128531694412231, -
0.07658082643617803]), a=1.3962634015954636, v=1.0471975511965976)
  $ 122 "Aviso"
  popup("Coloque la ultima ficha. Pulse 'Continuar' para seguir jugando.", "Mensaje", False,
False, blocking=True)
  $ 123 "iâ‰”0"
  global i=0
  $ 124 "xâ‰”x+1"
  global x=x+1
  # begin: URCap Program Node
  #   Source: Robotiq_Wrist_Camera, 1.6.6, Robotiq Inc.
```

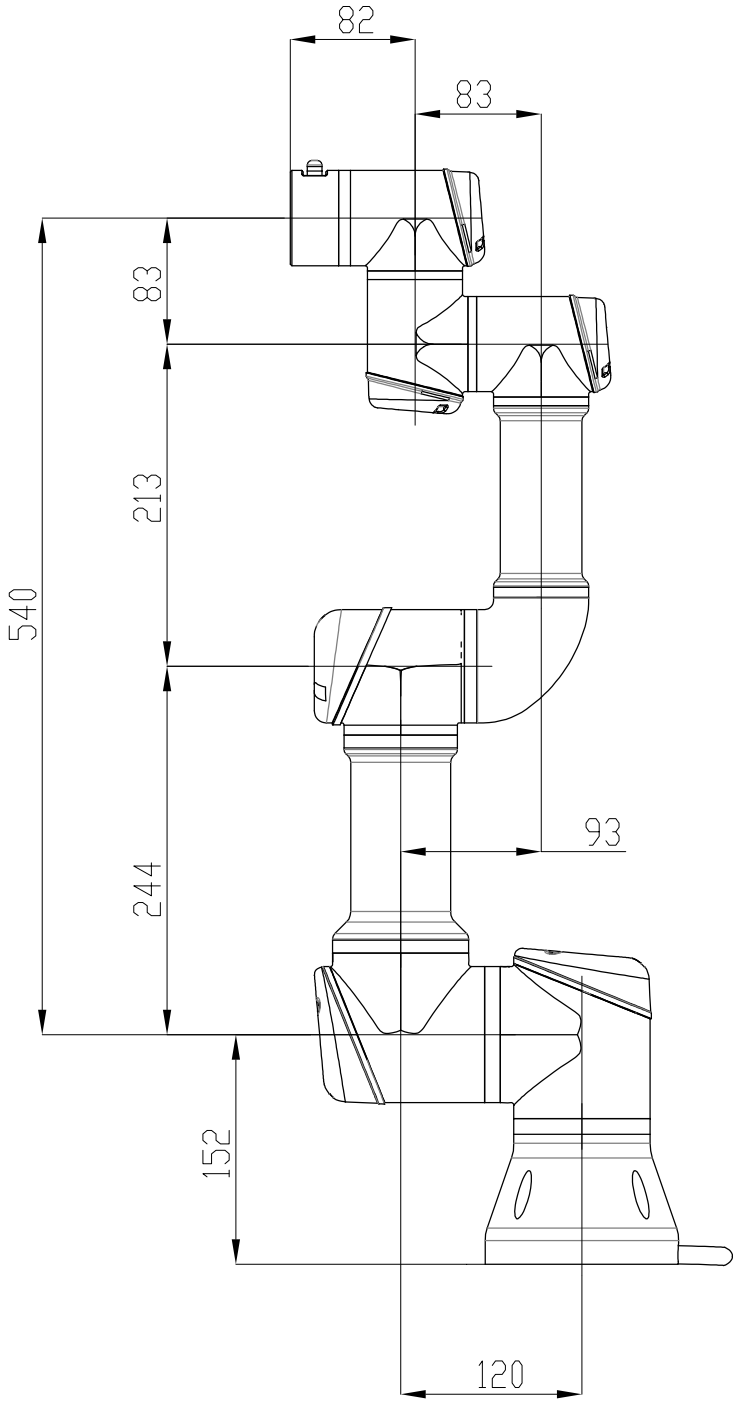
```
    #   Type: UbicaciÃ³n de CÃ¡mara
    $ 125 "Camera Locate"

    ##########################################
    #######VisionLocate node start########

    # Offset in translation only.
    snapshot_position_offset[3] = 0
    snapshot_position_offset[4] = 0
    snapshot_position_offset[5] = 0
    T_camera_in_flange = p[0, 0, 0, 0, 0, 0] # enlever une fois que l'enseignement du modele sera fait en faisant un movetool avec la pose de la camera dans le repere de la flange. Pour l'instant, on suppose que la camera est situee directement sur la flange.
    tool_30771112 = get_T_in_base_from_flange(T_camera_in_flange)
    textmsg("actual tool flange : ", tool_30771112)
    tool_30771112 = pose_sub(tool_30771112, snapshot_position_offset)
    textmsg("tool after offset : ", tool_30771112)
    snapshot_position = p[0.140501, 0.307158, 0.167948, -2.52782, 0.0136984, -0.0177129]
    textmsg("expected snapshot position : ", snapshot_position)
    diff = pose_sub(tool_30771112, snapshot_position)
    textmsg("diff = ", diff)
    textmsg("norm([diff[0], diff[1], diff[2]]) = ", norm([diff[0], diff[1], diff[2]]))
    textmsg("norm([diff[3], diff[4], diff[5]]) = ", norm([diff[3], diff[4], diff[5]]))
    is_at_snapshot_position = norm([diff[0], diff[1], diff[2]]) < 0.002
    textmsg("is_at_snapshot_position = ", is_at_snapshot_position)
    is_at_snapshot_position = is_at_snapshot_position and (norm([diff[3], diff[4], diff[5]]) < 0.005)
    textmsg("is_at_snapshot_position = ", is_at_snapshot_position)
    is_snapshot_position_offset = norm(snapshot_position_offset) != 0
    is_at_snapshot_position = is_at_snapshot_position or ignore_snapshot_position
    if not(is_at_snapshot_position):
      popup("Robot is not at Snapshot Position. Add Move instruction to Snapshot Position before Camera Locate node.. Error code: [UCC-8]", warning = False, error = True)
      halt
    end
    f=[0,0.0,0.0,0.0,0.0,0.0,0.0,0]
    object_location=p[0.0,0.0,0.0,0.0,0.0,0.0]
    f_30771112 = xmlrpc_server.findmodel("contextName-91271", tool_30771112[0], tool_30771112[1], tool_30771112[2], tool_30771112[3], tool_30771112[4], tool_30771112[5])
    logging_service.publish("FIND_MODEL", f_30771112)
    nb_occurence_30771112 = f_30771112[0]
    i_30771112 = 0
    object_teaching_location = p[0.14457822971951348, 0.3282708363187259, -0.1772027020943127, -3.137195097638309, 0.03687464060942577, 0.009851919779365375]
```

```
    feature_teaching_location = p[0.14048766331351048, 0.3071671394532257,
0.16794161737473473, -2.5277982295060917, 0.013747643390065246, -
0.017677924219300607]
    # begin: URCap Program Node
    #   Source: Robotiq_Wrist_Camera, 1.6.6, Robotiq Inc.
    #   Type: Bucle de ubicaciÃ³n de cÃ¡mara
    $ 126 "Para los objeto(s) encontrados"
    while (i_30771112 < nb_occurence_30771112):
     current_object_location_30771112 =
p[f_30771112[8*i_30771112+1],f_30771112[8*i_30771112+2],f_30771112[8*i_30771112+3],
f_30771112[8*i_30771112+4],f_30771112[8*i_30771112+5],f_30771112[8*i_30771112+6]]
      object_location = current_object_location_30771112
      f[0] = f_30771112[0]
      f[1] = f_30771112[8*i_30771112+1]
      f[2] = f_30771112[8*i_30771112+2]
      f[3] = f_30771112[8*i_30771112+3]
      f[4] = f_30771112[8*i_30771112+4]
      f[5] = f_30771112[8*i_30771112+5]
      f[6] = f_30771112[8*i_30771112+6]
      f[7] = f_30771112[8*i_30771112+7]
      textmsg("current_object_location before offset = ", current_object_location_30771112)
      current_object_location_30771112 = pose_add(current_object_location_30771112,
snapshot_position_offset)
      textmsg("current_object_location after offset = ", current_object_location_30771112)
      pto_tablero = pose_trans(current_object_location_30771112,
pose_trans(pose_inv(object_teaching_location), feature_teaching_location))
      $ 127 "If i<4"
      if (i<4):
       $ 128 "pos_ficha_jugâ‰”pto_tablero_var"
       global pos_ficha_jug=pto_tablero
       $ 129 "Invocar Comprobar_pos_jug"
       Comprobar_pos_jug()
      end
      $ 130 "iâ‰”i+1"
      global i=i+1
      i_30771112 = i_30771112 + 1
    end
    # end: URCap Program Node
    # Restore snapshot position
    pto_tablero = p[0.140501, 0.307158, 0.167948, -2.52782, 0.0136984, -0.0177129]

    #######VisionLocate node end########
    ##########################################

    # end: URCap Program Node
```

```
    $ 131 "jug_fichâ‰”ordenarArray(jug_fich)"
    global jug_fich=ordenarArray(jug_fich)
    $ 132 "Invocar Huecos_libres"
    Huecos_libres()
    $ 133 "libreâ‰”ordenarArray(libre)"
    global libre=ordenarArray(libre)
    $ 134 "Invocar Colocar_rob_fich4"
    Colocar_rob_fich4()
    $ 135 "rob_fichâ‰”ordenarArray(rob_fich)"
    global rob_fich=ordenarArray(rob_fich)
    $ 136 "Invocar Comprobar_ganador"
    Comprobar_ganador()
    $ 137 "MoveJ"
    $ 138 "pto_leer_tabl"
    movej(get_inverse_kin(p[.140491078503, .307142425909, .167928534585, -
2.527924077601, .013659440073, -.017792940890], qnear=[1.470109462738037, -
2.0002477804767054, -2.216708485280172, 0.11703526973724365, 1.6128531694412231, -
0.07658082643617803]), a=1.3962634015954636, v=1.0471975511965976)
    $ 139 "If winâ‰Ÿ0"
    if (win == 0):
      $ 140 "Aviso"
      popup("EMPATE, BIEN JUGADO! Fin de la partida.", "Advertencia", True, False,
blocking=False)
      halt
    end
  end
 end
end
)
      halt
    end
  end
 end
end
```

DOCUMENTO II

PLANOS

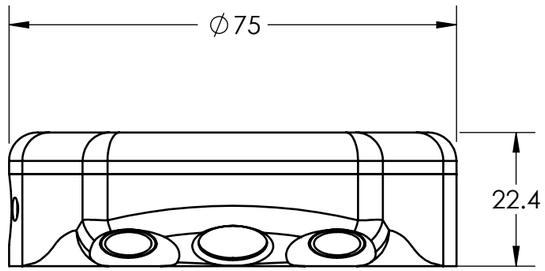


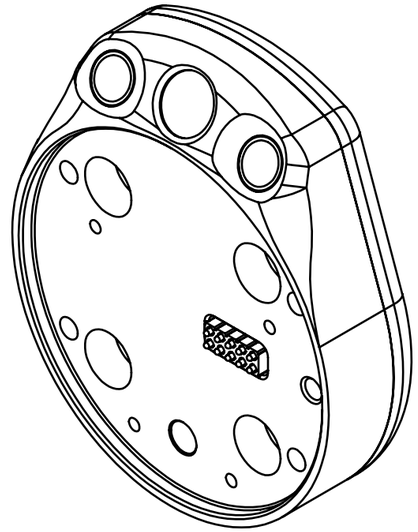
MÉTODO DE PROYECCIÓN DEL PRIMER DIEDRO

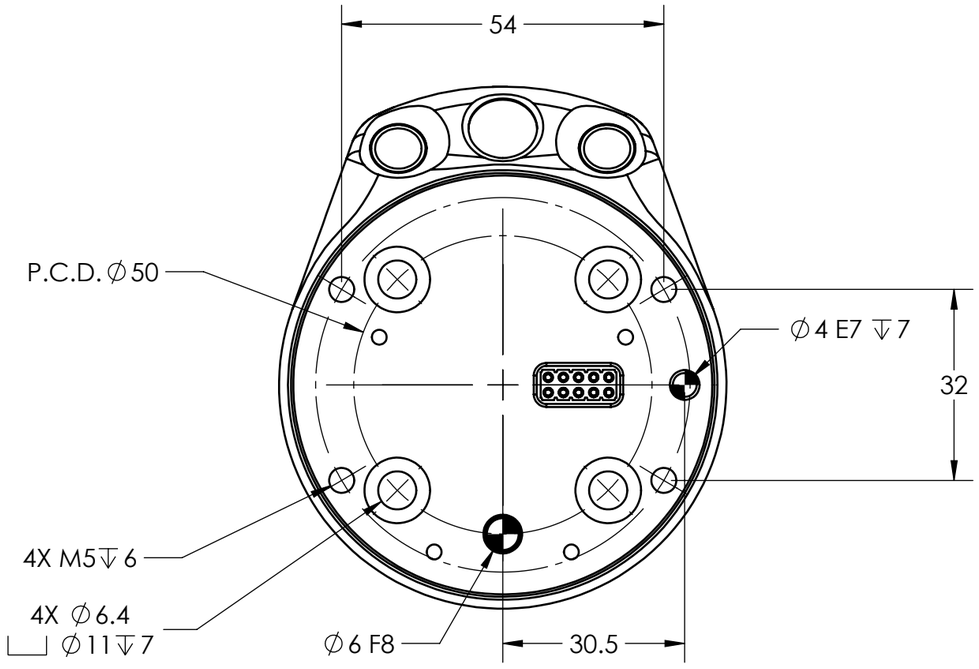
PROYECTO : Diseño y programación de un sistema robotizado para el juego del Tres en raya TITULAR : María Dolores Tíscar López		Fecha : 27/11/2020 Unidad mm
Autor : María Dolores Tíscar López	Plano : Dimensiones UR3	Plano Nº 01

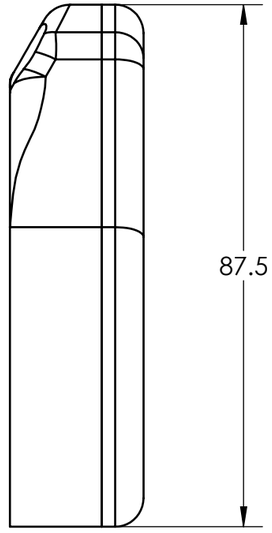
CREADO CON UNA VERSION PARA ESTUDIANTES DE AUTODESK

CREADO CON UNA VERSION PARA ESTUDIANTES DE AUTODESK









REV	DESCRIPTION	DATE	DESSINÉ/ DRAWN	VÉRIFIÉ/ APPROVED
A	PLAN INITIAL/INITIAL DRAWING	2019-07-30		

MATÉRIAU/MATERIAL

NOTES GÉNÉRALES/GENERAL NOTES

FINI SURFACE/ SURFACE FINISH: Ra 1.6 µm

ÉCHELLE/ SCALE: 1:1

FORMAT PAPIER/ SHEET FORMAT: ANSI B

TOLÉRANCES GÉNÉRALES/ GENERAL TOLERANCES

PROJECTION 3e DIÈDRE/ 3rd ANGLE

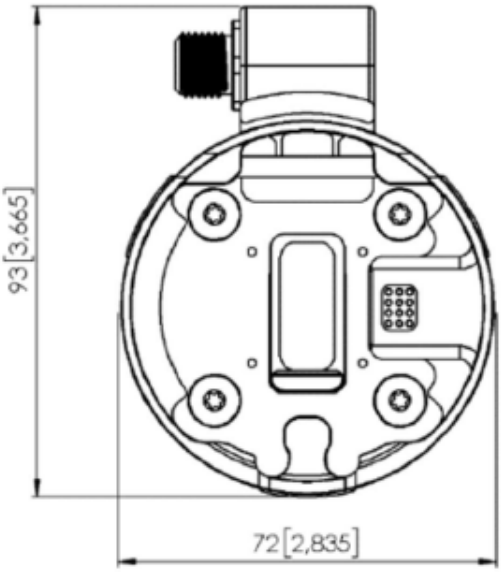
UNITÉS/UNITS: mm

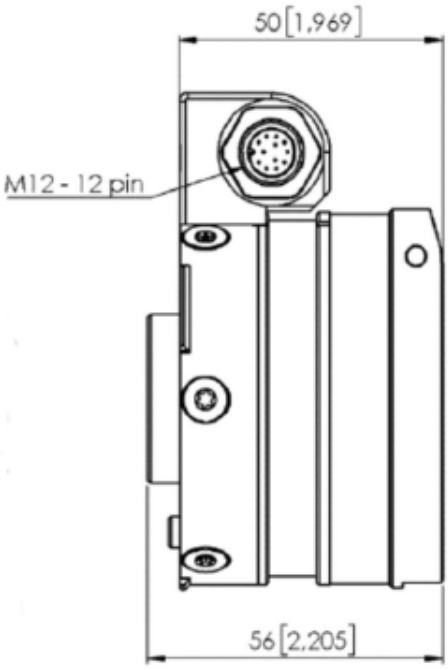
ROBOTIQ ©

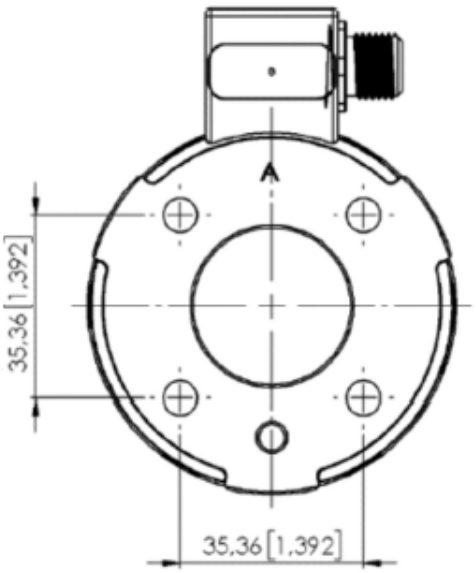
PROJET/ PROJECT

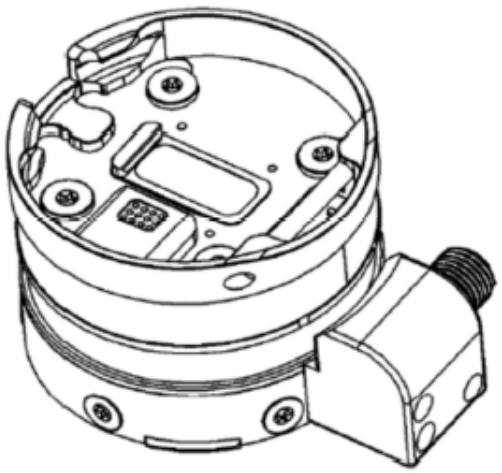
TITRE/ TITLE: **WRIST CAMERA**

DESSIN/DRAWING #	REV.	PAGE/SHEET
Q-A01	A	1 / 1







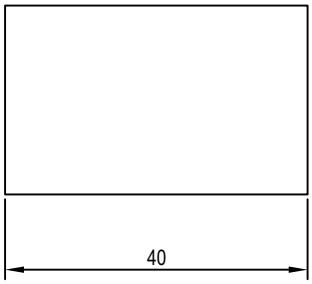


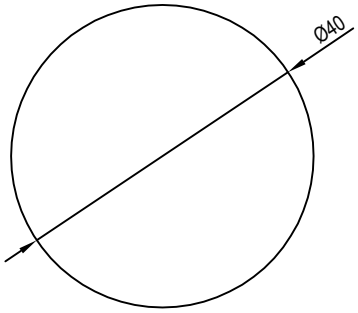
MÉTODO DE PROYECCIÓN DEL PRIMER DIEDRO

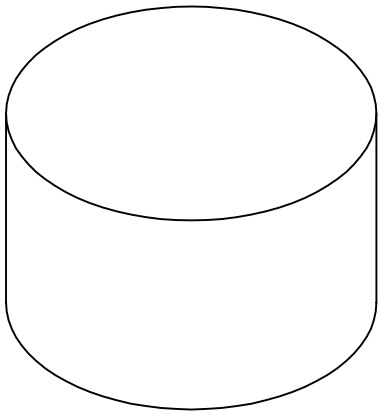
PROYECTO : Diseño y programación de un sistema robotizado para el juego del Tres en raya TITULAR : María Dolores Tíscar López		Fecha : 27/11/2020 Unidades mm [pulgadas]
Autor : María Dolores Tíscar López	Plano : Dimensiones HEX-EB165	Plano Nº 03

CREADO CON UNA VERSION PARA ESTUDIANTES DE AUTODESK

CREADO CON UNA VERSION PARA ESTUDIANTES DE AUTODESK







MÉTODO DE PROYECCIÓN DEL PRIMER DIEDRO

PROYECTO : Diseño y programación de un sistema robotizado para el juego del Tres en Raya

TITULAR : María Dolores Tíscar López

Fecha : 27/11/2020

Escala 1:1

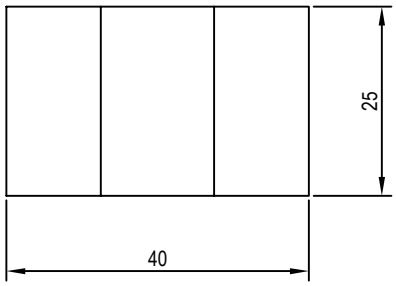
Autor : María Dolores Tíscar López

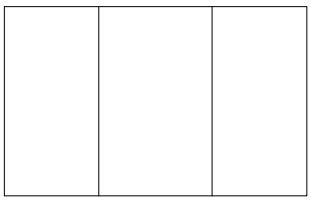
Plano : PIEZA CÍRCULO

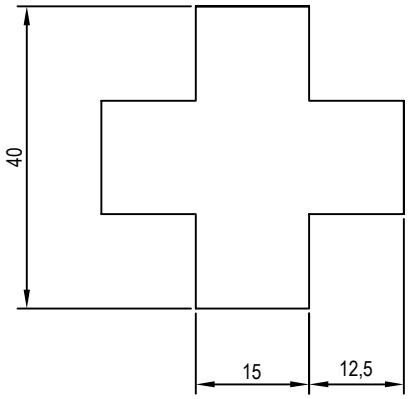
Plano Nº 04

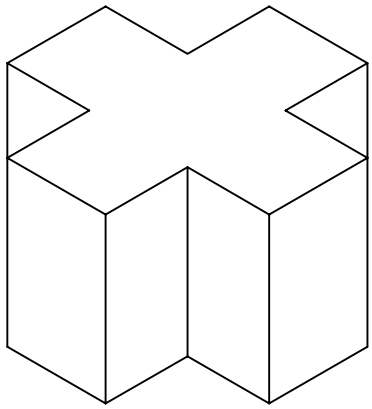
CREADO CON UNA VERSIÓN PARA ESTUDIANTES DE AUTODESK

CREADO CON UNA VERSIÓN PARA ESTUDIANTES DE AUTODESK









MÉTODO DE PROYECCIÓN DEL PRIMER DIEDRO

PROYECTO : Diseño y programación de un sistema robotizado para el juego del Tres en raya TITULAR : María Dolores Tíscar López		Fecha : 27/11/2020 Escala 1:1
Autor : María Dolores Tíscar López	Plano : PIEZA CRUZ	Plano Nº 05

CREADO CON UNA VERSION PARA ESTUDIANTES DE AUTODESK

CREADO CON UNA VERSION PARA ESTUDIANTES DE AUTODESK

CREADO CON UNA VERSION PARA ESTUDIANTES DE AUTODESK

DOCUMENTO III

PRESUPUESTO

1. Introducción

En el presente documento se detallará el desglose del presupuesto sin amortizaciones del proyecto expuesto. Se encuentra dividido en las diferentes partidas a seguir como el hardware, software y juego. Cada partida muestra los diferentes materiales necesarios para el proyecto junto con el coste en cada caso.

2. Capítulo de recursos del hardware

A continuación, se muestran los costes de los materiales y la mano de obra empleados en la parte del hardware para el desarrollo del proyecto.

HARDWARE

MATERIALES						
Referencia	**Uds**	**Denominación**	**Fabricante**	**Cantidad**	**Precio (€)**	**Total**
M1	u	Robot colaborativo UR3	Universal Robots	1	26.332,00	26.332,00
M2	u	Wrist Camera	Robotiq	1	5.500,00	5.500,00
M3	u	Force Sensor HEX-EB165	OnRobot	1	3.885,00	3.885,00
M4	u	Gripper	Festo	1	234,00	234,00
M5	u	Ordenador ASUS VivoBook	ASUS	1	699,00	699,00
					SUBTOTAL MATERIALES	**36.650,00**
MANO DE OBRA						
Referencia	**Uds**	**Denominación**	**Tarea**	**Cantidad**	**Precio (€)**	**Total**
O1	h	Peón	Montaje	5	18,00	90,00
O2	h	Peón	Puesta en marcha	3	20,00	60,00
O3	h	Peón	Pruebas funcionamiento	2	20,00	40,00
					SUBTOTAL MANO DE OBRA	**190,00**

SUBTOTAL MATERIALES	SUBTOTAL MANO DE OBRA	SUBTOTAL SOFTWARE
36.650,00 €	**190,00 €**	**36.840,00 €**

El coste de la partida del hardware es de la expresada cifra de TREINTA Y SEIS MIL OCHOCIENTOS CUARENTA EUROS.

3. Capítulo de recursos del software

A continuación, se muestran los costes de los materiales y la mano de obra empleados en la parte del software para el desarrollo del proyecto.

SOFTWARE						
Referencia	**Uds**	**Denominación**	**Fabricante**	**Cantidad**	**Precio (€)**	**Total**
S1	u	Office Profesional 2019	Mricrosoft 365	1	99,00	99,00
S2	u	Windows 10	Microsoft	1	31,99	31,99
S3	u	Simulador UR3	Universal Robots	1	0,00	0,00
S4	u	AutoCAD	Autodesk	1	2.227,00	2.227,00
					SUBTOTAL MATERIALES	**2.357,99**
MANO DE OBRA						
Referencia	**Uds**	**Denominación**	**Tarea**	**Cantidad**	**Precio (€)**	**Total**
O4	h	Oficial 1º	Investigación y documentación	2	20,00	40,00
O5	h	Oficial 1º	Diseño del programa	30	24,00	720,00
O6	h	Oficial 1º	Diseño de piezas	5	24,00	120,00
O7	h	Peón	Pruebas funcionamiento	2	20,00	40,00
O8	h	Peón	Puesta en marcha	5	20,00	100,00
					SUBTOTAL MANO DE OBRA	**1.020,00**

SUBTOTAL MATERIALES	SUBTOTAL MANO DE OBRA	SUBTOTAL SOFTWARE
2.357,99 €	**1.020,00 €**	**3.377,99 €**

El coste de la partida del software es de la expresada cifra de TRES MIL TRESCIENTOS SETENTA Y SIETE EUROS CON NOVENTA Y NUEVE CÉNTIMOS.

4. Capítulo de recursos del juego

A continuación, se muestran los costes de los materiales y la mano de obra empleados en la parte del juego para el desarrollo del proyecto.

JUEGO

Referencia	Uds	Denominación	Fabricante	Cantidad	Precio (€)	Total
M6	u	Impresora 3D	Creality	1	179,00	179,00
M7	kg	Filamento PLA	Geeetech	1	23,00	23,00
M8	u	Tablero		1	3,00	3,00
M9	u	Rotulador	Staedtler	1	0,85	0,85
					SUBTOTAL MATERIALES	**205,85**
MANO DE OBRA						
Referencia	**Uds**	**Denominación**	**Tarea**	**Cantidad**	**Precio (€)**	**Total**
O9	h	Peón	Imprimir piezas	8	18,00	144,00
					SUBTOTAL MANO DE OBRA	**144,00**

SUBTOTAL MATERIALES	SUBTOTAL MANO DE OBRA	SUBTOTAL SOFTWARE
205,85 €	**144,00 €**	**349,85 €**

El coste de la partida del juego es de la expresada cifra de TRESCIENTOS CUARENTA Y NUEVE EUROS CON OCHENTA Y CINCO CÉNTIMOS.

5. Resumen del presupuesto

Resumen presupuesto	Materiales	Mano de obra	Total
Hardware	36.650,00	190,00	36.840,00
Software	2.357,99	1.020,00	3.377,99
Juego	205,85	144,00	349,85

PRECIO FINAL	40.567,84 €

El coste total del del proyecto es de la expresada cifra de CUARENTA MIL QUINIENTOS SESENTA Y SIETE EUROS CON OCHENTA Y CUATRO CÉNTIMOS.

DOCUMENTO IV

PLIEGO DE CONDICIONES

1. Definición y alcance del pliego

El objetivo del presente documento es establecer las condiciones a cumplir y considerar entre los contratantes durante la realización de este Trabajo Fin de Grado: diseño y programación de un sistema robotizado para el juego del Tres en Raya.

El ámbito de aplicación de este documento se extiende a todos los equipos mecánicos, eléctricos y electrónicos que forman parte de este proyecto.

2. Condiciones generales y normativa

Se considerará el acuerdo de todo lo estipulado en el contrato por ambas partes una vez éste haya sido firmado.

Los materiales y equipos usados serán los nombrados en este proyecto y deberán estar en perfectas condiciones y cumplir con la normativa vigente.

El cliente obtendrá una garantía de 1 año tras recibir los equipos. Esta incluye el reemplazo del elemento en caso de ser defectuoso, sin coste alguno. La garantía no incluirá desperfectos o averías por el uso indebido del equipo o por ser tratado por personal no autorizado.

El contratista tiene la obligación de proporcionar toda información necesaria para su correcta instalación la cual deberá ser llevada a cabo por personal capacitado.

Una vez instalado, se realizarán todas las pruebas necesarias hasta asegurarse que el conjunto trabaja de forma adecuada y sin problemas. Todas las pruebas deben ser realizadas bajo la supervisión de un agente del contratista.

Todos los elementos de este proyecto cumplen con la normativa expuesta en el *Real Decreto 1215/1997* sobre las disposiciones mínimas de seguridad y salud de equipos de trabajo usado por trabajadores.

El robot colaborativo usado UR3, cumple con la normativa vigente reflejada en su manual de usuario adjuntado en el apartado "*Normas aplicadas*". A continuación, se destacan las más relevantes:

- **ISO 13850:2015** Seguridad de la maquinaria. Parada de emergencia. Principios del diseño.
- **ISO 10218-1:2011** Robots y dispositivos robóticos. Requisitos de seguridad para robots industriales. Parte 1: Robots.
- **ISO/TS 15066:2016** Robots y dispositivos robóticos. Requisitos de seguridad para robots industriales. Operación colaborativa.
- **ISO 14118:2017** Seguridad de la maquinaria. Prevención de puesta en marcha inesperada.
- **IEC 60529:2013** Grados de protección proporcionados por las envolventes (IP *Code*).

- **IEC 60320-1:2015** Acopladores de dispositivos para el hogar y fines generales similares. Parte 1: Requisitos generales.
- **ISO 13732-1:2006** Ergonomía del entorno térmico. Métodos para evaluación de las respuestas humanas al contacto con superficies. Parte 1: Superficies calientes.
- **IEC 60068-2:2007** Pruebas ambientales. Pruebas, Parte 2-1: Prueba A. Frío; Parte 2-2: Prueba B. Calor seco; Parte 2-27: Prueba Ea y vibración; Parte 2-67: Prueba Fh. Vibración, banda ancha aleatoria y guía.

3. Condiciones de carácter económico

El precio indicado en este proyecto es fijo y no hay lugar a negociación alguna respecto al precio.

El pago se realizará en dos partes:

1. Se abonará el 75% al firmar el contrato.
2. Se abonará el 10% al recibir todos los equipos.
3. Se abonará el 15% en un periodo de 10 días naturales tras recibir todos los equipos.

En caso del incumplimiento con los pagos dentro de los plazos establecidos, se aplicará un cargo de 10% del precio total al cliente.

Si no se ha realizado la entrega de los equipos en el plazo acordado, el cliente tendrá derecho a una compensación.

El pago se realizará mediante cheque o transferencia a la entidad bancaria indicada.

Una vez transcurrido el tiempo de garantía, cualquier consulta supondrá un coste al cliente correspondiente al número de horas empleadas.

4. Especificaciones de ejecución

Una vez instalados los equipos, la puesta en marcha se llevará a cabo tal y como figura en el manual de usuario adjuntado.

La programación del sistema robotizado, la cámara de visión y el sensor de fuerza deben ejecutar lo descrito en este proyecto.

Para el correcto funcionamiento de la aplicación diseñada se deben tener en cuenta una serie de consideraciones que se muestran a continuación:

- Las piezas deben ser colocadas dentro de las casillas del tablero.
- No importa la orientación en la que ponga la pieza.
- Señalar correctamente la zona de trabajo del robot cuando este se encuentra en movimiento para así evitar colisiones que puedan dañar al robot o al usuario.

El contratista no se hace responsable de los problemas ocasionados por no seguir estas instrucciones.