Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

EXPLORACIÓN DEL SOFTWARE ABIERTO ROBODK

Alumno: Antonio David Acosta Torres

Tutor: Ángel Gaspar González Rodríguez.
Dpto: Ingeniería de Sistemas y Automática.

MAYO, 2022

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Ingeniería Electrónica y Automática

Don ÁNGEL GASPAR GONZÁLEZ RODRÍGUEZ, tutor del Proyecto Fin de Carrera titulado: EXPLORACIÓN DEL SOFTWARE ABIERTO ROBODK, que presenta ANTONIO DAVID ACOSTA TORRES, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Mayo de 2022

El alumno:

ACOSTA TORRES ANTONIO DAVID - 77434683P

Firmado digitalmente por ACOSTA TORRES ANTONIO DAVID - 77434683P
Fecha: 2022.05.23 14:18:44 +02'00'

ANTONIO DAVID ACOSTA TORRES

Los tutores:

GONZALEZ RODRIGUEZ ANGEL GASPAR - 26014077L

Firmado digitalmente por GONZALEZ RODRIGUEZ ANGEL GASPAR - 26014077L
Fecha: 2022.05.23 16:45:18 +02'00'

ÁNGEL GASPAR GONZÁLEZ RODRÍGUEZ

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

En estos tiempos en los que la tecnología se desarrolla de forma desenfrenada, se está viendo como cada vez los robots van formando parte de un mayor grupo de labores, tanto en tareas puramente industriales, como en otros ámbitos tales como el entorno médico, del sector servicios o incluso militar. La industria fue el primer sector que empezó a implementar robots en sus filas, dando lugar, en conjunto con el desarrollo de las tecnologías de la información (TIC) y la electrónica, a una revolución del tejido productivo que es conocida en algunos documentos como la “Tercera Revolución Industrial”.

Es innegable que la inserción de la robótica en la industria ha traído consigo numerosas ventajas, siendo un factor determinante para la mejoría de la calidad, la productividad y la reducción de costos; por ello los robots se han convertido en un elemento esencial que cualquier industria moderna necesita para estar en la vanguardia. Un buen ejemplo de ello es el hecho hoy día casi todas las industrias del sector del automóvil cuentan con algún sistema robótico automatizado en sus líneas de montaje. Como se verán en apartados posteriores, el uso de robots está extendido en distintos tipos de industrias siendo la automotriz la principal.

A raíz de esta inclusión masiva de la automatización en el entorno industrial, se hace necesario el desarrollo de nuevos lenguajes y métodos de programación de robots industriales, siendo también fundamental la prueba y simulación de estos en un entorno virtual para su posterior puesta a punto en un ambiente real.

1.1 Motivación del proyecto

Este proyecto surge a partir de la necesidad de desarrollar una herramienta alternativa al simulador de robots “Roboguide” desarrollado por la empresa FANUC que es el que se utiliza generalmente en prácticas de laboratorio de distintas asignaturas del Grado de Ingeniería Electrónica de la Universidad de Jaén. Para conseguir este objetivo se ha diseñado una aplicación a partir de una API suministrada por la página web de RoboDK, la cuál es capaz de realizar de forma similar las funciones básicas utilizadas en otros softwares de simulación de robots que utilicen el lenguaje TPE.

2. OBJETIVOS

El objetivo principal del presente trabajo es el **desarrollo de una aplicación en lenguaje C# para la ejecución de programas en lenguaje TPE simulando el movimiento de robots e interactuando con objetos y cintas transportadoras.** Se apoya en el software de simulación de robots "RoboDK" y en la API (Interfaz de Aplicación del Programa) proporcionada por RoboDK. Esta aplicación se ha ido construyendo gracias al entorno de desarrollo Microsoft Visual Studio, donde se han ido implementando las distintas fases del proyecto dando lugar por último a la creación de una aplicación que es capaz de admitir e interpretar distintas instrucciones en lenguaje TPE que es principalmente utilizado por los robots de la marca FANUC.

En resumen, durante este proyecto se ha desarrollado una aplicación de usuario que simula una consola de programación real de un robot pudiendo implementar las principales operaciones del lenguaje TPE en un robot simulado, permitiendo, a diferencia de otros tipos de software similares tales como Roboguide, la manipulación de objetos y la interacción con cintas transportadoras. Previamente se ha realizado el estudio del software de simulación de robots RoboDK, que sirve como motor gráfico de la aplicación

2.1 Objetivos del proyecto.

En la realización del presente trabajo se han tenido en cuenta una serie de objetivos que debían ser alcanzados tras la conclusión del mismo. Estos objetivos se pueden clasificar dentro de 5 categorías (Figura 1):

A. Investigación y análisis.
B. Programación de las pestañas de diseño de la aplicación (Pestañas DATA, EDIT y DISEÑO).
C. Desarrollo de un algoritmo de gestión de archivos (Pestaña General).
D. Realización del intérprete del programa TPE.
E. Depuración y creación de un gestor de errores.

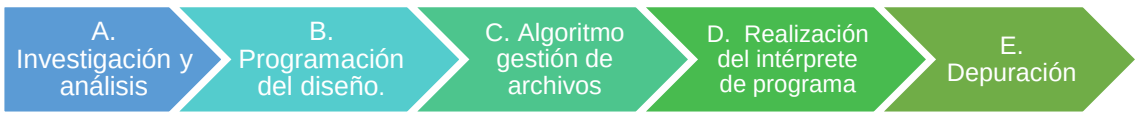


Figura 1: Categorías de objetivos

Los objetivos relativos a estas categorías se presentan en la tabla siguiente:

Categoría A: Investigación y análisis.
- Estudio y análisis del código y del entorno de desarrollo, así como familiarización con la herramienta de simulación RoboDK.
Categoría B: Programación de las pestañas de diseño de la aplicación.

- DATA
  • Lograr almacenar posiciones del robot dentro de un registro de posiciones.
  • Almacenamiento de variables en registros escalares.
- EDIT
  • Desarrollo de la pestaña EDIT, con los elementos necesarios para poder incluir las distintas instrucciones y comandos del lenguaje TPE en un programa.
- DISEÑO
  • Conseguir incluir objetos (mesas y objetos manipulables) dentro de la estación, así como permitir modificaciones de tamaño y color y dejar registro de los mismos.
  • Insertar y/o eliminar cintas transportadoras.
  • Añadir un método para agregar/cambiar un robot, así como seleccionar una herramienta asociada.

Categoría C: Desarrollo de un algoritmo de gestión de archivos.
- Componer un sistema para el guardado del programa y todos los registros asociados en un archivo. - Lograr la apertura y carga de elementos a partir de un archivo de guardado.
Categoría D: Realización del intérprete del programa TPE.
- Desarrollar un algoritmo para la lectura e interpretación de un programa TPE, permitiendo implementar las principales instrucciones del mismo sobre un robot, siendo fundamental la manipulación de objetos sencillos.
Categoría E: Depuración y creación de un gestor de errores.
- Crear un gestor de errores que alerte al usuario de problemas.

Tabla 1: Desglose de objetivos del proyecto.

3. ANTECEDENTES

3.1 Introducción Histórica de la Robótica Industrial

En estos últimos dos siglos, las sucesivas revoluciones industriales han promovido un desarrollo desenfrenado en cuanto a producción industrial se refiere. Se destacan avances como la máquina de vapor en la Primera Revolución Industrial (1820-1840), que mejoró de forma notoria las industrias textiles y metalúrgicas, así como los medios de transporte o bien la aparición de las primeras centrales eléctricas y de los primeros transistores en la Segunda Revolución Industrial (1870-1914) que dieron lugar a la aparición de la producción en masa y a las líneas de montaje y fabricación.

Sin embargo, no es hasta la Tercera Revolución Industrial donde se produce un gran impulso en la industria, pues es en esta era donde gracias al desarrollo de la electrónica, la automatización y las comunicaciones, se empiezan a ver los primeros ordenadores personales y los primeros robots de producción. Esta mejora en el modelo industrial dio lugar a un incremento manifiesto de la producción y un abaratamiento de los costos de forma que determinados productos se hicieron más accesibles para la población ganando así en calidad de vida.

Desde hace unos años ya se empieza a hablar de una cuarta revolución industrial también conocida como industria 4.0 proyectada para la tercera década del siglo XXI, donde las tecnologías de la información unidas a un uso intensivo del Internet y el desarrollo de la inteligencia artificial darán lugar a la creación de dispositivos y máquinas inteligentes, autónomas e interconectadas. El avance de las telecomunicaciones, las redes de internet de alta velocidad y en otros conceptos más vanguardistas como lo son el Cloud Computing, el Big Data o el “Internet of Things” (IoT) hacen posible idear proyectos y modelos de producción que eran impensables hace unos años. La combinación de estos conceptos aplicados a la industria y gracias al desarrollo del mundo de la robótica harán factible lo que se conoce por “Smart Factories”, que cuenten con una mayor adaptabilidad a las distintas situaciones y a los procesos productivos, así como una gestión más eficiente de los recursos pudiéndose cumplir así los objetivos medioambientales. Por último, estas fábricas inteligentes tendrán la labor también de tomar decisiones inteligentes y de la administración de activos en base a la información recopilada y analizada [1][2][3]

Como se puede advertir, a lo largo de la historia la robótica ha tenido una participación notoria en el desarrollo industrial por lo que el estudio de los fundamentos y la

programación de estos dispositivos robóticos es fundamental para un Ingeniero Industrial, ya que se encuentran en la gran mayoría de las industrias sobre todo las relacionadas con la producción en masa y las automovilísticas.

3.2 Tipos de Robots Industriales

Para comenzar, se ha de definir qué se entiende por un robot industrial. Según dos de los organismos más referentes en cuanto a robótica se refiere como lo son la Federación Internacional de Robótica (IRF) y la Asociación de Industrias Robóticas (RIA) [4], se pueden obtener dos definiciones basadas en la normativa ISO 8373 “Robots and Robotic Devices”:

- Un robot industrial es un manipulador multifuncional, controlado automáticamente, reprogramable en tres o más ejes, que puede estar fijo o móvil para uso en aplicaciones de automatización industrial.

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

En un primer momento en la industria se pueden diferenciar entre robots manipuladores y AGVs. Los robots manipuladores (Figura 2) son aquellos que tienen como finalidad realizar tareas que impliquen manejar objetos o piezas de este; dentro de esta categoría se encuentran por ejemplo los brazos robóticos. Por otra parte, también están los AGVs o Vehículos de Guiado Automático (Figura 3), que son robots dedicados exclusivamente al transporte de elementos de un punto a otro de una planta. Estos circulan de forma autónoma recorriendo rutas que previamente han sido definidas. A partir de aquí solo se hará referencia a los robots manipuladores industriales ya que son los que serán utilizados en la aplicación.

Figura 3: Ejemplo de AGVs

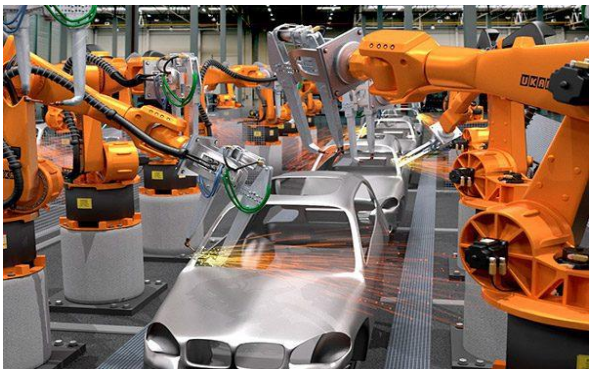


Figura 2 Ejemplo de Robots Manipuladores

Teniendo ahora definido lo que es un robot manipulador industrial, se pueden definir algunas características principales que son determinantes a la hora de clasificar los robots [5]. Estas características son:

- Configuración: empleo de diferentes combinaciones de articulaciones y disposición de las mismas en una posición estandarizada.
- Grados de libertad: se refiere a cada una de las coordenadas independientes necesarias para definir la posición y orientación espacial de los elementos del robot.
- Área de trabajo: es el espacio donde el robot puede realizar sus tareas, así como la distancia mínima y máxima que este puede alcanzar.
- Capacidad de carga: inercias admisibles en objetos y elementos finales y carga máxima de peso que el robot es capaz de movilizar.
- Tipo de programación: es la posibilidad de programar un robot industrial en distintos lenguajes.

Finalmente se puede establecer una clasificación de los robots industriales atendiendo a la configuración del mismo (Figura 4). Se distinguen los siguientes tipos:

- Robot cartesiano: estos robots trabajan con tres ejes prismáticos que se mueven en línea recta perpendicularmente.
- Robot cilíndrico: cuentan con una columna vertical que gira sobre su base y una articulación prismática para el movimiento lineal. Está formado por 2 ejes prismáticos y uno rotacional.
- Robot esférico o polar: permite desplazar su brazo en un área esférica gracias a sus articulaciones donde dos de ellas son rotacionales y una lineal.
- Robot SCARA: SCARA son las siglas de *Selective Compliant Assembly Robot Arm*, cuentan con cuatro grados de libertad y posicionamiento horizontal.
- Robot articulado o angular: este tipo de robot tiene articulaciones tanto lineales como angulares permitiendo un gran número de movimientos y orientaciones distintas.
- Robot paralelo o Delta: están compuestos por dos bases unidas por cadenas cinemáticas cerradas basadas en paralelogramos.

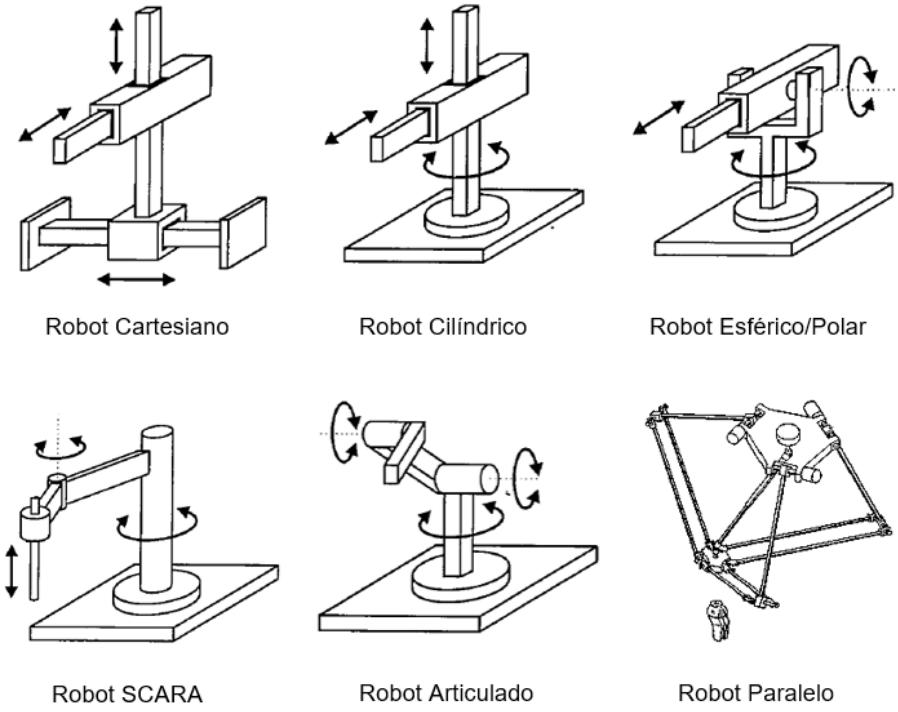


Figura 4: Principales configuraciones de robots industriales

3.3 Aplicaciones de los Robots Industriales

Existen varias organizaciones nacionales e internacionales dedicadas a promover las tecnologías de la automatización y de la robótica dentro del tejido industrial. A nivel español se encuentra la Asociación Española de Robótica y Automatización (AER) [6], que es una asociación sin ánimo de lucro que agrupa a los gremios más importantes dentro del sector de la robótica como son fabricantes, universidades y empresas. A nivel internacional existe la *International Federation of Robotics (IRF)*, que es también una organización sin ánimo de lucro, fundada en 1987 y cuya misión es promover los beneficios de los robots en el crecimiento económico y fomentar la investigación y desarrollo de dispositivos robóticos [7].

Las aplicaciones de un robot industrial son muy diversas: soldadura, paletizado, mecanizado de piezas, adhesivado, montaje, transporte, manipulación de elementos peligrosos, pintura, etc. En la Figura 5 se muestra el número de unidades robóticas, en miles, instaladas en los distintos sectores de la industria en 2018, 2019 y 2020. Estos datos han sido extraídos del *World Robotics 2021 Report* [8]. De ellos se deduce que las principales áreas de aplicación de los robots en el seno de la industria son la electrónica y el sector automotriz.

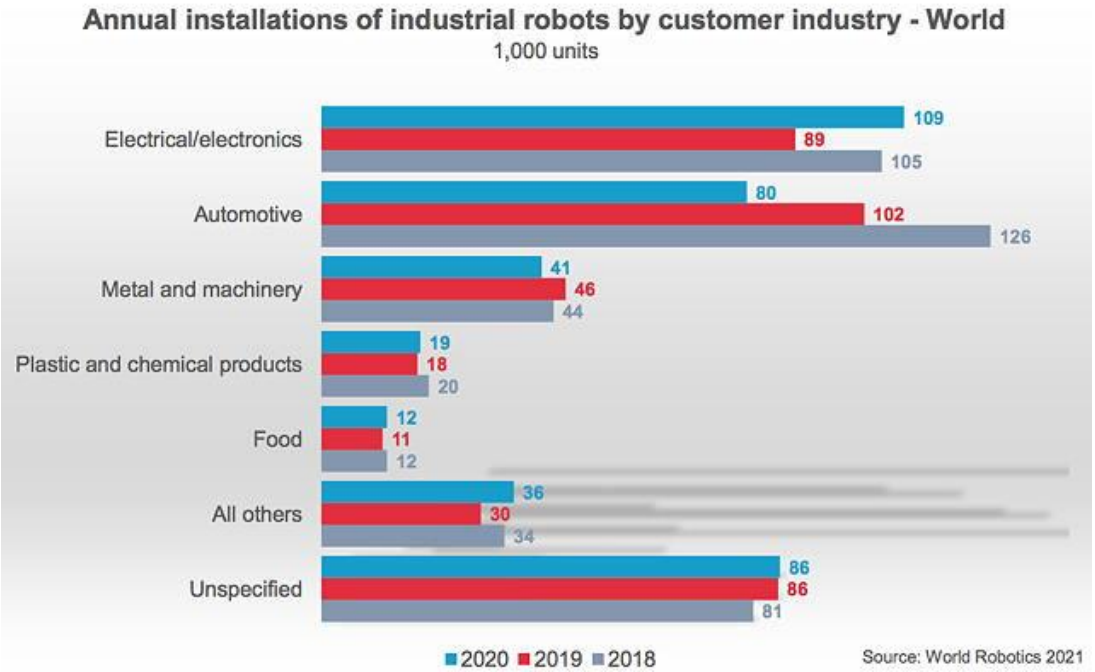


Figura 5: Instalación mundial de robots industriales por sector

3.4 Parque de Robots Industriales

Según el World Robotics 2021 report, la instalación global de robots se espera que aumente en un 13% cada año, sin embargo, como consecuencia de la crisis del Covid-19 producida en 2020 se ha visto una reducción notable en la instalación de robots. Sin embargo, se prevé que en los próximos años el ritmo de crecimiento se vuelva a recuperar a valores pre Covid-19 (Figura 6).

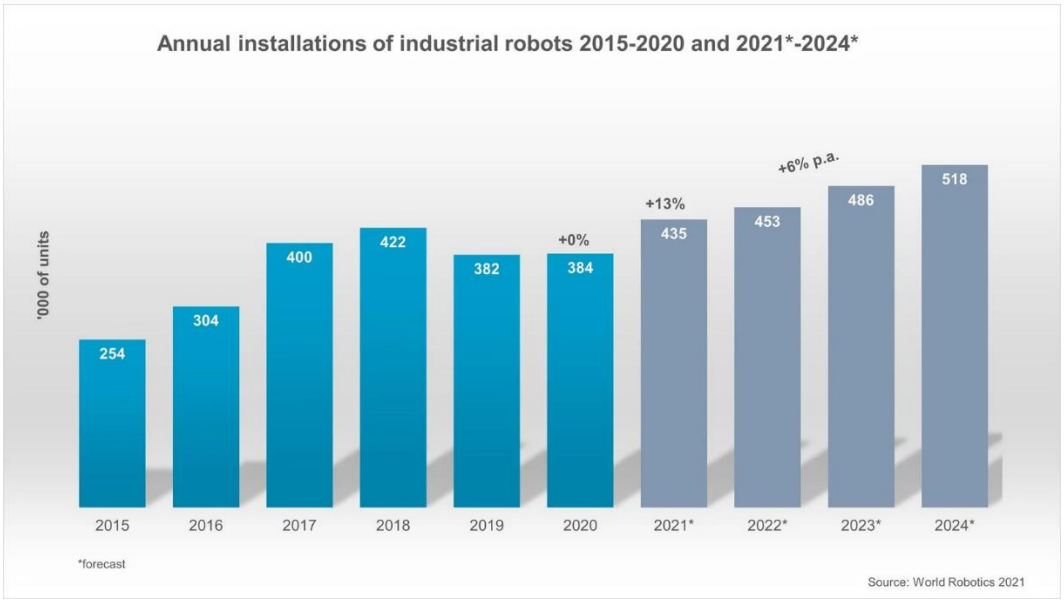


Figura 6: Instalación mundial de robots industriales 2015-2020

Asia se mantiene como el mercado de robots industriales más grande del mundo (Figura 7). Se estima que entorno al 71% de los robots desarrollados en 2020 fueron instalados en la región asiática, siendo fuertemente remarcable la participación de China. Japón se consolida en segunda posición seguida de los Estados Unidos a pesar de las afecciones de la pandemia del Covid-19 [8].

Por otra parte, en Europa la instalación de robots industriales ha decaído en torno al 8-9% hasta solo 67.000 unidades en 2020. De esta manera continua la tendencia a la baja por segundo año consecutivo debido a la caída de la demanda automovilística de un 20%. Alemania es el país europeo que encabeza la lista de instaladores de robots industriales, posicionándose en quita posición en el ranking mundial.

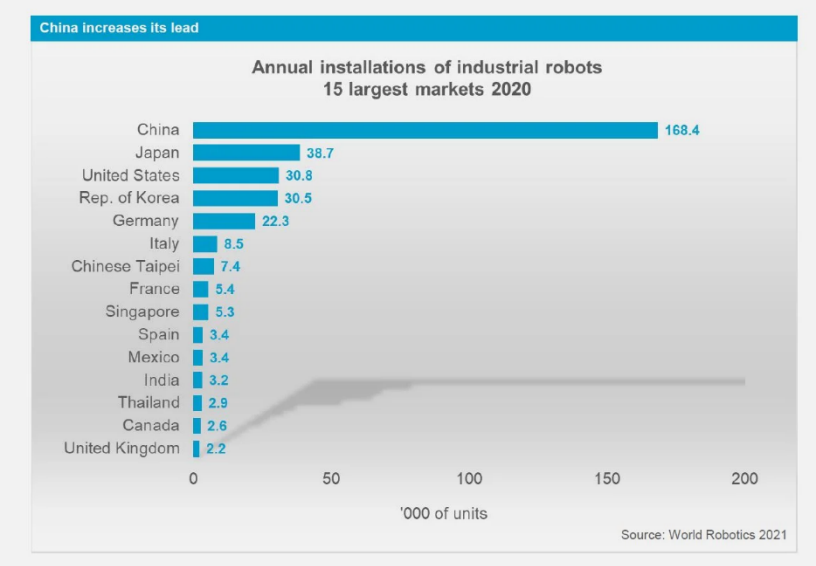


Figura 7: Países con mayor instalación de robots industriales en 2020

4. CONCEPTOS BÁSICOS SOBRE ROBÓTICA

Dentro de este apartado se tratarán algunos de los conceptos básicos referentes a teoría robótica [9] los cuales han sido necesarios estudiar y comprender a fondo para la realización de la aplicación del proyecto. Aunque la teoría de robots se extiende mucho más, solo se tratarán los temas más fundamentales y estrictamente necesarios para la comprensión de algunos métodos usados en la herramienta. Paralelamente al desarrollo de estos conceptos, se indicará dónde se aplican estos dentro de la herramienta.

4.1 Representación de la posición.

Para que un robot pueda recoger una pieza es necesario tener constancia de la posición y orientación del extremo del robot con respecto a su base; por ello es necesario emplear una serie de herramientas matemáticas para poder especificar la posición y orientación en el espacio.
Para representar la posición se pueden usar diferentes métodos como son coordenadas cartesianas, coordenadas cilíndricas y coordenadas esféricas. En este proyecto se utiliza un sistema de referencia en coordenadas cartesianas para denotar las tres componentes necesarias para definir la posición en un espacio tridimensional. Estas tres componentes componen lo que se denomina vector de posición o traslación.

4.2. Representación de la orientación

Por otra parte, también es necesario especificar la orientación para definir una postura única ya que a una misma posición se podría acceder desde distintas orientaciones. El proceso utilizado para describir la orientación de un objeto respecto de un sistema de referencia se basa en asignar un nuevo sistema de referencia al objeto y estudiar posteriormente la relación entre sistemas de referencias. Existen distintos métodos para describir la orientación como son:

- Matrices de Rotación
- Ángulos de Euler
- Par de Rotación
- Cuaternios

En este epígrafe solo se tratarán los dos primeros ya que son los que están implicados en el desarrollo de esta aplicación.

4.2.1 Matrices de Rotación

Las matrices de rotación son el método más utilizado para la descripción de orientaciones y sirven para transformar las coordenadas de un vector de un sistema de referencia en las de otro sistema distinto. En la siguiente tabla se pueden ver las distintas matrices de rotación empleadas:

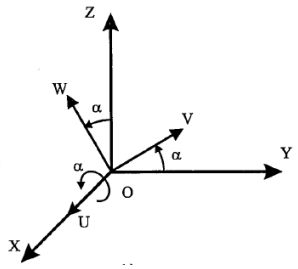
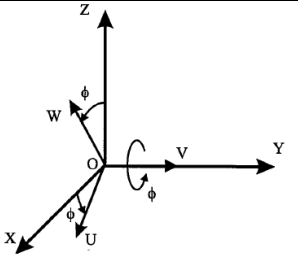
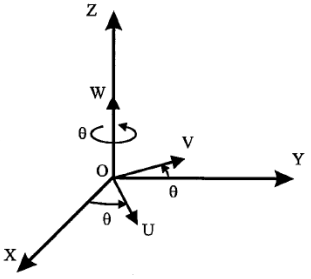
MATRICES DE ROTACIÓN		
ROTACIÓN RESPECTO EJE X	$R(x,\alpha)=\begin{bmatrix}1 & 0 & 0\\ 0 & cos\alpha & -sen\alpha\\ 0 & sen\alpha & cos\alpha\end{bmatrix}$	
ROTACIÓN RESPECTO EJE Y	$R(y,\phi)=\begin{bmatrix}cos\phi & 0 & sen\phi\\ 0 & 1 & 0\\ -sen\phi & 0 & cos\phi\end{bmatrix}$	
ROTACIÓN RESPECTO EJE Z	$R(z,\theta)=\begin{bmatrix}cos\theta & -sen\theta & 0\\ sen\theta & cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}$	

Tabla 2: Matrices de Rotación

Para realizar una aplicación continua de varias rotaciones se utiliza la composición de rotaciones que consiste en la multiplicación de matrices de rotación en el orden en que se producen las mismas. En la siguiente ecuación se describe una rotación respecto a "x" seguida de una rotación respecto "y" y finalmente una rotación respecto a "z" (para

abreviar, el coseno se representa con la letra C y el seno con la S). El conjunto de las rotaciones compone una matriz de rotación que contempla a las anteriores.

$$T = R(x,\alpha)R(y,\phi)R(z,\theta) =$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & C\alpha & -S\alpha \\ 0 & S\alpha & C\alpha \end{bmatrix} \begin{bmatrix} C\phi & 0 & S\phi \\ 0 & 1 & 0 \\ -S\phi & 0 & C\phi \end{bmatrix} \begin{bmatrix} C\theta & -S\theta & 0 \\ S\theta & C\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} C\phi C\theta & -C\phi S\theta & S\phi \\ S\alpha S\phi C\theta + C\alpha S\theta & -S\alpha S\phi S\theta + C\alpha C\theta & -S\alpha C\phi \\ -C\alpha S\phi C\theta + S\alpha S\theta & C\alpha S\phi S\theta + S\alpha C\theta & C\alpha C\phi \end{bmatrix}$$

4.2.2 Ángulos Euler

Este método de representación de la orientación utiliza únicamente tres elementos en lugar de los nueve necesarios en las matrices de rotación. Un sistema puede definir su orientación con respecto a otro mediante tres ángulos ϕ, θ, ψ, denominados ángulos de Euler. Existen diferentes representaciones, dependiendo del eje respecto del cual se produzca el giro como pueden ser la representación ZYZ o la representación ZXZ. Una de las representaciones más utilizadas es la Roll, Pitch and Yaw descrita en la Figura 8. Los ángulos de Euler se pueden convertir en una matriz de rotación mediante la multiplicación de matrices de rotación. Cada una de estas matrices de rotación representarían la rotación respecto a un eje con un valor determinado por cada uno de los ángulos de Euler. Por otra parte, también se puede obtener los ángulos de Euler a partir de la matriz de rotación. Para ello se acuden a una serie de operaciones trigonométricas.

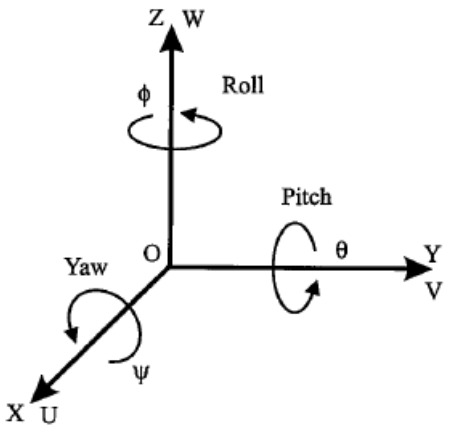


Figura 8: Roll, Pitch and Yaw.

4.3 Matrices de Transformación Homogéneas

Ahora bien, para realizar una representación conjunta de posición y orientación se necesitan las llamadas Matrices de Transformación homogéneas que se definen como una matriz 4 x 4 que representa la transformación de un vector de coordenadas homogéneas de un sistema de coordenadas a otro. Esta matriz está dividida a su vez en 4 submatrices las cuales corresponden a la matriz de rotación, estudiada anteriormente, el vector de translación, una submatriz de escalado global y otra de perspectiva.

$$T = \begin{bmatrix} R_{3x3} & p_{3x1} \\ f_{1x3} & w_{1x1} \end{bmatrix} = \begin{bmatrix} Rotación & Traslación \\ Perspectiva & Escalado \end{bmatrix}$$

Usualmente se utiliza la perspectiva nula y el escalado global unitario teniendo como resultado la siguiente matriz:

$$T = \begin{bmatrix} R_{3x3} & p_{3x1} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} Rotación & Traslación \\ 0 & 1 \end{bmatrix}$$

En la siguiente tabla se muestran las matrices de transformación homogéneas para representar una traslación respecto a cualquier eje y las rotaciones respecto a cada uno de los 3 ejes espaciales:

MATRICES DE TRANSFORMACIÓN HOMOGÉNEA	
TRASLACIÓN	$T(X,Y,Z)=\begin{pmatrix}1&0&0&X\\0&1&0&Y\\0&0&1&Z\\0&0&0&1\end{pmatrix}$
ROTACIÓN RESPECTO EJE X	$T(x,\alpha)=\begin{pmatrix}1&0&0&0\\0&Cos(\alpha)&-Sin\,(\alpha)&0\\0&Sin(\alpha)&Cos(\alpha)&0\\0&0&0&1\end{pmatrix}$
ROTACIÓN RESPECTO EJE Y	$T(y,\phi)=\begin{pmatrix}Cos(\phi)&0&Sin\,(\phi)&0\\0&1&0&0\\-Sin\,(\phi)&0&Cos(\phi)&0\\0&0&0&1\end{pmatrix}$
ROTACIÓN RESPECTO EJE Z	$T(z,\theta)=\begin{pmatrix}Cos(\theta)&-Sin\,(\theta)&0&0\\Sin\,(\theta)&Cos(\theta)&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}$

Tabla 3: Matrices de Transformación Homogénea

La mayor ventaja de utilizar matrices de transformación homogéneas es la capacidad de representación simultánea de orientación y posición, unificando la matriz de rotación y el vector de posición en una sola matriz. En la siguiente formula se presenta un ejemplo en el que se realiza una rotación sobre el eje X seguido de una traslación según el vector P.

$$T((x,\alpha),p)=\begin{pmatrix}1&0&0&p_x\\0&Cos\,\alpha&-Sin\;\alpha&p_y\\0&Sin\,\alpha&Cos\,\alpha&p_z\\0&0&0&1\end{pmatrix}$$

Otra de las grandes virtudes de la utilización de matrices de transformación homogéneas es, la composición de estas para la descripción de traslaciones y giros consecutivos. De este modo un movimiento complejo puede pasar a describirse como la aplicación consecuente de varios movimientos sencillos. Ahora bien, la composición implica la multiplicación de matrices, y es sabido que es una operación no conmutativa, por consecuente, se han de multiplicar en el orden en el que se desean que se realicen las

distintas transformaciones. En el siguiente ejemplo se muestra una traslación según el vector de posición P, seguido de una rotación sobre el eje X y una rotación sobre el eje Z:

$$T = T(p)T(x,\phi)T(y,\theta) =$$

$$= \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & Cos\,\phi & -Sin\,\phi & 0 \\ 0 & Sin\,\phi & Cos\,\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Cos\,\theta & 0 & Sin\,\theta & 0 \\ 0 & 1 & 0 & 0 \\ -Sin\,\theta & 0 & Cos\,\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} Cos\,\theta & 0 & Sin\ \theta & p_x \\ Sin\,\phi\,Sin\ \theta & Cos\,\phi & -Cos\,\theta\,Sin\,\phi & p_y \\ -Cos\,\phi\,Sin\,\theta & Sin\,\phi & Cos\,\phi\,Cos\,\theta & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4.4 Problema Directo e Inverso.

Cuando se quiere estudiar la el modelado del robot se plantean dos problemas que se ven descritos de manera gráfica en la Figura 9:

- El problema directo, el cual determina la posición y orientación del extremo del robot, con respecto de un sistema de coordenadas de referencia siendo conocidos los valores articulares y geométricos de los elementos del robot.
- El problema inverso, que determina la configuración angular que debe adoptar un robot para que su extremo alcance una posición y orientación determinadas.

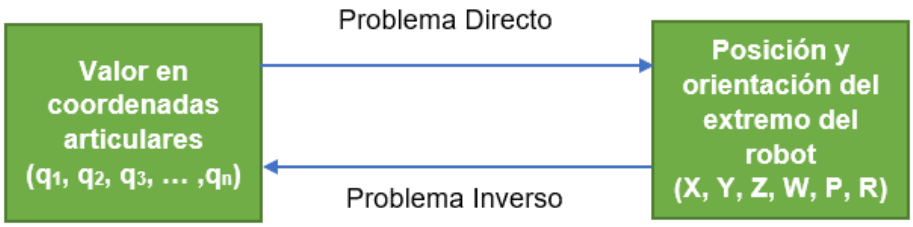


Figura 9: Modelado Directo e Inverso

Se aplican distintas metodologías como el algoritmo de Denavit Hartenberg para resolver el problema directo. Este algoritmo es un método sistemático que describe la geometría de un robot utilizando matrices de transformación homogéneas para describir las relaciones entre elementos rígidos adyacentes. Como un robot es una cadena cinemática compuesta por eslabones y articulaciones, se puede designar un sistema de referencia fijo en la base del robot y describir la ubicación de cada eslabón con respecto

al sistema de referencia base. De este modo, el problema directo queda reducido a la búsqueda de una matriz de transformación homogénea que enlace la posición del extremo del robot con respecto al sistema de coordenadas base propio.

Por otra parte, el problema inverso, como ya se ha indicado antes, consiste en encontrar los valores de las articulaciones del robot para que el extremo de este se posicione en una ubicación específica. Para resolver el problema inverso es más correcto hallar una solución cerrada ya que hay una serie de algoritmos basados en métodos numéricos pero su velocidad de convergencia es reducida e incluso en algunas ocasiones imposible. La utilización de matrices de transformación homogéneas queda restringida ya que el procedimiento de obtención de las ecuaciones es muy dependiente de la configuración del robot dando lugar a múltiples soluciones válidas. Por ello se utilizan las matrices de transformación homogéneas en combinación con el desacoplo cinemático para resolver este problema.

5. DESCRIPCIÓN DEL LENGUAJE TPE

5.1 ¿Qué es el lenguaje TPE?

El fin principal de este proyecto es realizar una herramienta de simulación de un robot industrial en la que se interprete un programa escrito en lenguaje TPE. El lenguaje TPE (Teach Pendant Editor) es el lenguaje de programación robótica utilizado en la programación de robots de la marca Fanuc, a través del terminal Teach Pendant. Es una herramienta de programación sencilla, con un lenguaje no estructurado cuya sintaxis es controlada de forma automática durante su creación.

En este capítulo se expondrá la estructura de un programa en TPE, los registros que contiene y qué estructuras componen su sintaxis más básica reduciéndose los contenidos de esta explicación a los utilizados en el desarrollo de la aplicación.

5.2 Organización de un Programa en TPE

Para crear un programa en TPE en una consola de programación propia de un robot Fanuc se tienen que editar dos tipos de menús:

- Menú DATA: En este menú es donde se almacenan los distintos registros que son utilizados durante la ejecución del programa. Se distinguen dos tipos de registros: los registros de posición (Figura 11) que guardan los vectores de seis elementos que contienen los valores de posición y orientación de cada una de las localizaciones que se desean almacenar, y los registros escalares (Figura 10) que contienen los vectores de un solo elemento cuyo contenido pueden ser dimensiones de un objeto, contadores, valores auxiliares…

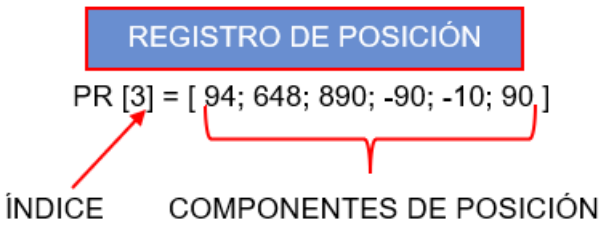


Figura 11: Registro de Posición

Figura 10: Registro Escalar

- Menú EDIT: En este otro menú se compone el programa a ejecutar por el robot. En este espacio se escriben las distintas líneas de instrucción que se van a interpretar y llevar a cabo por el robot industrial.

Para que la herramienta desarrollada en este proyecto fuese lo más parecida posible a la programación en TPE habitual en una consola, se han incluido, como se verá más adelante, dos pestañas EDIT y DATA donde se escribirán los distintos registros y el programa como tal.

Para la manipulación de entradas y salidas digitales también se ha de usar el menú I/O en una consola de programación; como en esta herramienta no se han empleado no se ha desarrollado en este apartado.

5.3 Estructuras de Programación Fundamentales. Sintaxis.

A lo largo del siguiente apartado se tratará de definir las instrucciones básicas del lenguaje de programación TPE. Estas instrucciones son las que interpretará y ejecutará el robot.

5.3.1 Instrucciones de movimiento

Se distinguen tres tipos de movimiento hacia un punto: el movimiento Joint (J) en el que el desplazamiento se realiza de forma angular, el movimiento Linear (L) en el que el desplazamiento se realiza de forma lineal y el movimiento Circular (C) (que no se aplica en esta herramienta) en que se realiza un desplazamiento circular definiendo un punto de paso.

Las instrucciones de movimiento tienen la siguiente estructura (Figura 13):

- El primer elemento describe el tipo de movimiento, Joint, Lineal o Circular (J, L y C).
- El segundo elemento describe el tipo de posición: Posición (P) o Registro de posición (PR) y su índice. También cabe la posibilidad de que en vez de un índice se tenga lo que se conoce como direccionamiento indirecto; es decir, en vez de tener un índice que denote el registro de posición al que se hace referencia, se tiene un registro escalar cuyo valor es el índice que alude al registro de posición deseado (Figura 12).

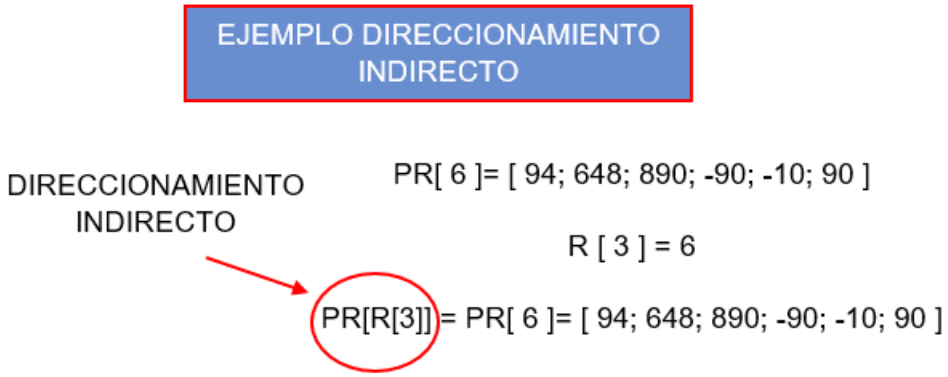


Figura 12: Direccionamiento indirecto

- El tercer elemento describe la velocidad del movimiento, en unidades porcentuales, angulares o lineales en función del tipo de movimiento.
- El cuarto elemento describe la precisión del movimiento pudiendo ser esta FINE o CNT (en esta aplicación solo se utilizará FINE).
- El quinto elemento contiene las opciones especiales, en esta aplicación albergará los offset (Offset o Tool offset).
-

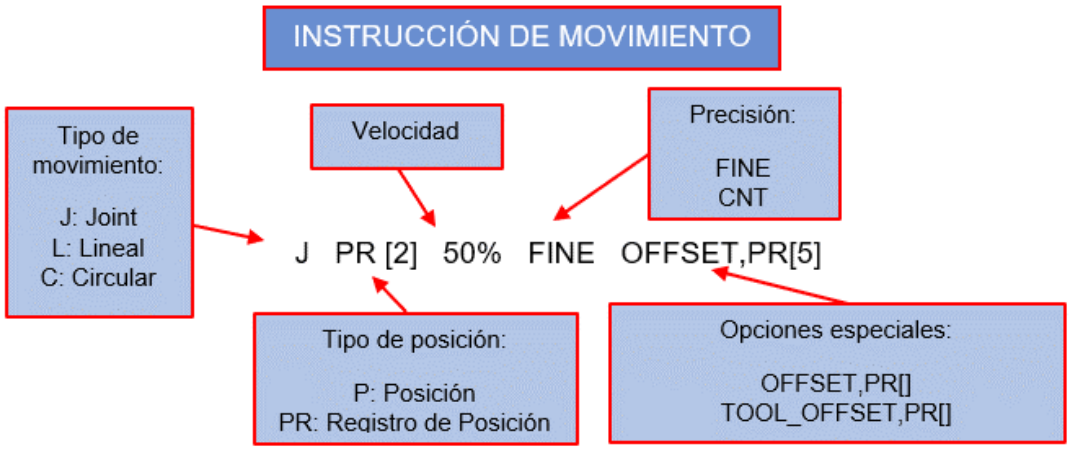


Figura 13: Instrucción de movimiento

5.3.2 Instrucciones de salto condicional e incondicional

La instrucción Label (Figura 14) es fundamental para realizar saltos de forma condicional. Esta instrucción cuenta con un índice que denota el número de etiqueta que es. Las etiquetas sirven como punto de referencia al que volver o avanzar cuando se realiza un salto. Un ejemplo sería una instrucción Label con índice 2 que se encuentra en la línea 5 del programa. Cuando durante la ejecución de un programa aparezca una

Instrucción de salto a la etiqueta 2, el programa irá a la línea 2 y seguirá ejecutándose desde ahí.

Figura 14: Instrucción Label

Por otra parte, se distingue entre instrucción de salto incondicional y salto condicional. Las instrucciones de salto incondicional (Figura 15) permiten ejecutar un salto a una etiqueta del mismo programa para que el programa continúe ejecutándose a partir de ahí.

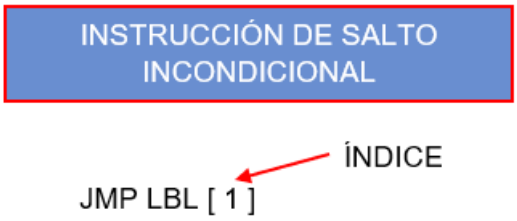


Figura 15: Instrucción salto incondicional

Las instrucciones de salto condicional por su parte, realizan la misma operación que las de salto incondicional, pero en función de si se ha cumplido un criterio o no; es decir, si la condición da un resultado true se realiza el salto a la etiqueta; si no, sigue ejecutándose el programa por la siguiente línea. En la aplicación desarrollada la condición puede ser una comparación entre dos registros escalares o entre un registro escalar y una constante. En la Figura 16 se puede observar un ejemplo de instrucción de este tipo en la que se comprueba si el registro 4 es menor o igual que el registro 2. Si es así, es decir, si se cumple esta condición, se realiza un salto a la etiqueta 1.

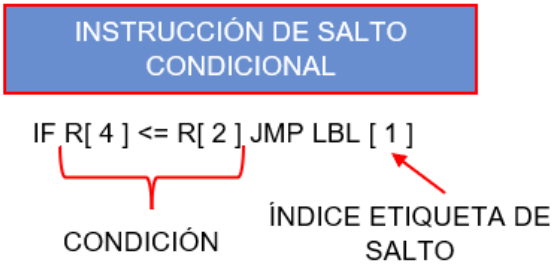


Figura 16: Instrucción salto condicional

5.3.3 Instrucciones de espera

Las instrucciones de espera hacen que la ejecución del programa se demore un tiempo preestablecido o hasta que una condición sea verdadera. Se distinguen dos tipos de instrucciones en función de lo anterior: la temporización, en la que el programa se detiene un tiempo específico; y la espera de una condición verdadera, que detiene la ejecución hasta que un criterio se cumpla. Solo se utiliza el primer tipo de instrucción en el desarrollo de esta herramienta. Un ejemplo de instrucción de espera de temporización se puede ver en la Figura 17.

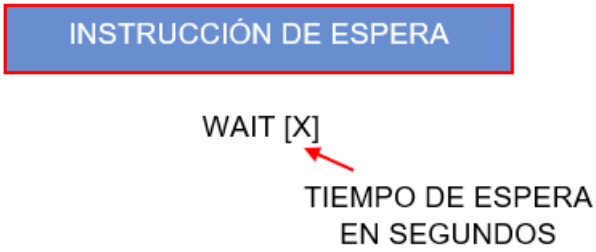


Figura 17: Instrucción de espera

5.3.4 Instrucciones de modificación de registros

Los valores de los registros son susceptibles de ser modificados mediante operaciones de punto que vienen descritas por instrucciones de modificación de registros. Estas operaciones están compuestas por un valor (que es un registro de posición PR, una posición P o un registro escalar R), un operador matemático y otro valor. Además, los registros de posición son accesibles elemento a elemento, lo que permiten que sean modificados por componentes independientes. En la Figura 18, se observa la modificación de un registro de posición y otro escalar. En el primer caso lo que se hace es cambiar la primera coordenada del registro de posición 2 para que sea igual a la del registro de posición 3 menos el valor almacenado en el registro escalar 5. En el segundo caso, lo que se hace es modificar el registro escalar 1 para que sea igual a sí mismo pero aumentado en uno.

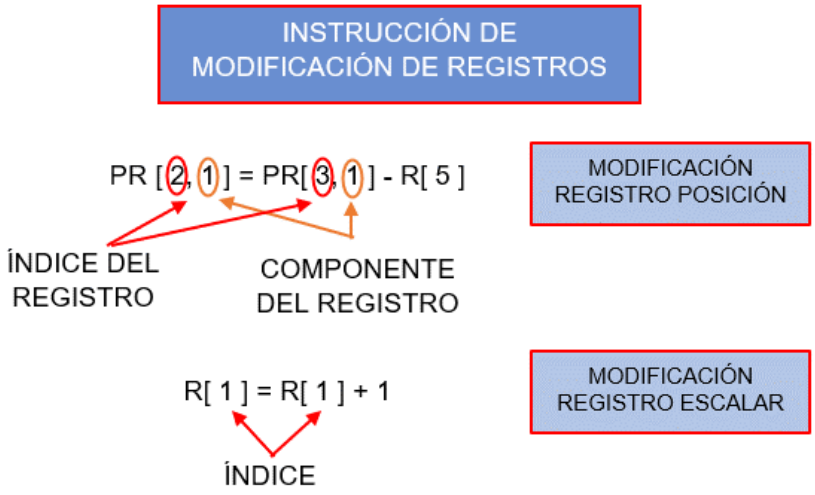


Figura 18: Instrucción de modificación de registros

5.3.5 Instrucciones de llamada

Mediante esta instrucción (Figura 19) se permite llamar a otra rutina o subprograma que ya ha sido establecido en otro apartado. Un ejemplo de esta instrucción de llamada es la llamada a ABRIR y CERRAR para realizar la apertura y cierre de la pinza o la llamada a MOVER CINTA para iniciar la ejecución del programa asociado a la cinta transportadora. En la aplicación las llamadas a ABRIR, CERRAR Y MOVER CINTA generan internamente una llamada a uno o varios programas en Python que ejecutan la simulación y las operaciones necesarias para realizar la captura y dejada de un objeto, así como el movimiento de la cinta transportadora y la detección a su final.

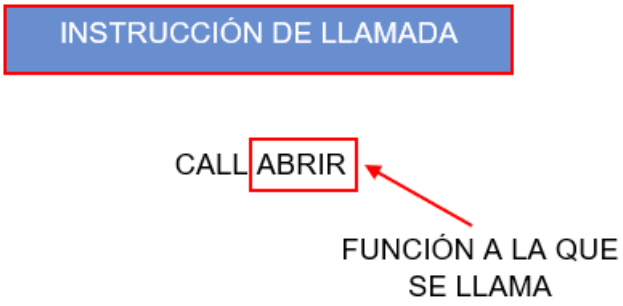


Figura 19: Instrucción de llamada

6. ESTUIDIO DEL SOFTWARE ROBODK

En este capítulo se realizará un estudio sencillo del software ROBODK utilizado como base para el desarrollo del proyecto, así como una breve descripción de la empresa propietaria y del interfaz utilizado.

6.1 ¿Qué es el software RoboDK?

La API utilizada para el desarrollo de este proyecto pertenece al software desarrollado por la empresa RoboDK. RoboDK Sofware es una empresa de origen canadiense más concretamente en la ciudad de Montreal. Surge en 2015 como un renombramiento del laboratorio CoRo de la Escuela de Tecnología Superior (École de technologie supérieure) de la universidad de Montreal. Desde 2015 RoboDK Software se ha desarrollado rápidamente como empresa ayudando a empresas de todo tipo, desde pequeñas organizaciones a instituciones multinacionales, en la tarea de la simulación y programación de robots industriales [10].

RoboDK es un software de simulación de robots industriales utilizado en el desarrollo de programas para los mismos a través de un PC. Dispone de una interfaz relativamente sencilla en la que no hay necesidad de tener conocimientos previos en programación para poder utilizarla. El software de RoboDK también ofrece la posibilidad de que estos programas pueden ser llevados a la práctica en un robot real. RoboDK ofrece una librería de robots industriales con alrededor de 450 elementos robóticos de 50 fabricantes distintos como son ABB, Fanuc, Kuka, Kawasaki o Universal Robots.

6.2 Descarga del programa

El acceso a su descarga (Figura 20) se realiza a través de su página web https://robodk.com/es/download, dentro de la pestaña descargar obtendremos las distintas versiones de RoboDK dependiendo de la arquitectura y sistema operativo utilizado en el ordenador donde se va a realizar la instalación.

Figura 20: Web de descarga de software RoboDK

6.3 Descripción del interfaz RoboDK

La interfaz de RoboDK es simple, gráfica e intuitiva. En la Figura 21 se pueden ver las distintas secciones de las que se compone esta ventana:

- Menú principal: Contiene todas las opciones y acciones posibles del programa.
- Ventana principal: Es la ventana donde se representa el modelado 3D de la estación diseñada.
- Barra de herramientas: En este menú se hallan las principales funciones utilizadas, representadas por el icono correspondiente. Entre las operaciones a destacar se encuentran:
    - Cargar Archivo (local) : Carga una estación almacenada localmente.
    - Abrir biblioteca en línea : Muestra una librería con los distintos elementos a utilizar en la estación (robots, objetos y herramientas).
    - Guardar estación : Almacena la estación RoboDK localmente como archivo RDK.
    - : Añade un nuevo sistema de referencia al que enlazar distintos elementos como robots u objetos.
    - : Añade una nueva posición objetivo (target) para que sea alcanzada por el robot.

- Mover sistema de referencia, mover herramienta: Habilita el movimiento manual (arrastrando con el ratón) de los sistemas de referencia o de la herramienta del robot.
- Añadir programa: Agrega un nuevo programa de simulación.
- Instrucción movimiento axial, lineal: Configura una instrucción de movimiento tipo articular/lineal.

- Árbol de la estación: Contiene cada uno de los elementos agregados a la estación (robots, posiciones, objetos, sistemas de referencia, subprogramas…) y describe la relación de dependencia entre ellos (Parent-Child).

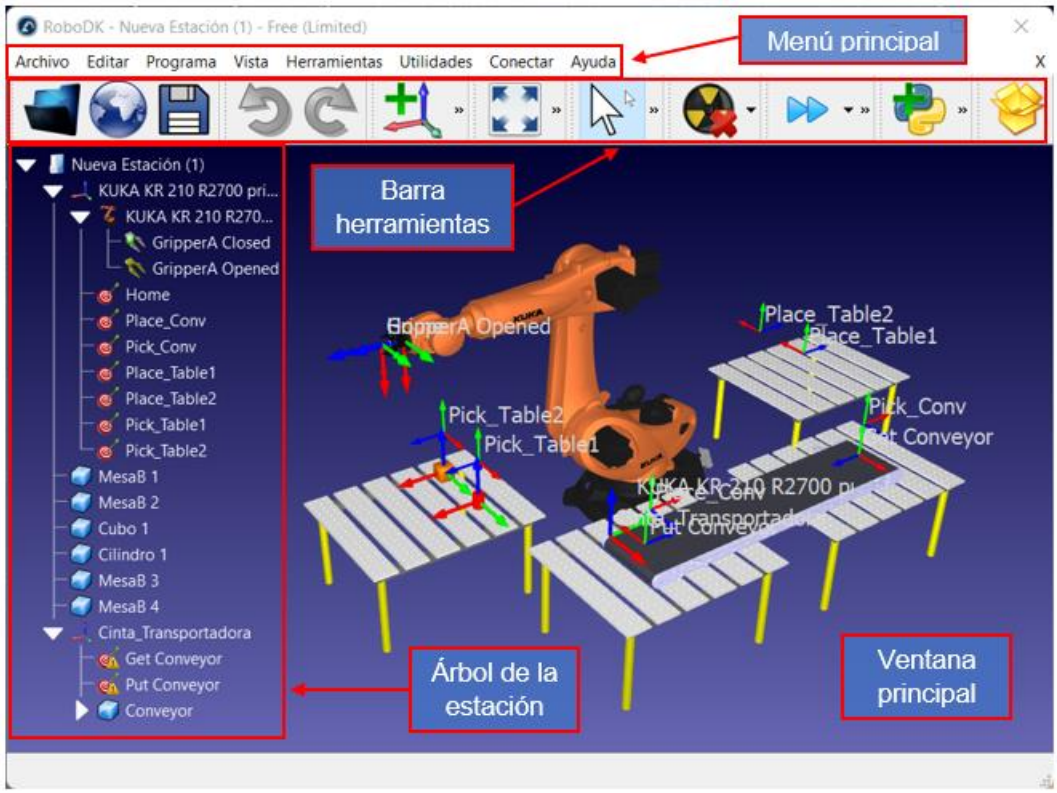


Figura 21: Interfaz RoboDK

Por otra parte, hay otros menús de uso muy recurrente como es el Panel Robot (Figura 23), donde se pueden revisar todos los parámetros del robot y configurarlos como se prefiera y Panel de Objeto/Sistema (Figura 22), donde se pueden ver las coordenadas de posición y orientación del objeto o sistema de referencia, así como modificar color y tamaño.

Figura 22: Panel objeto/sistema

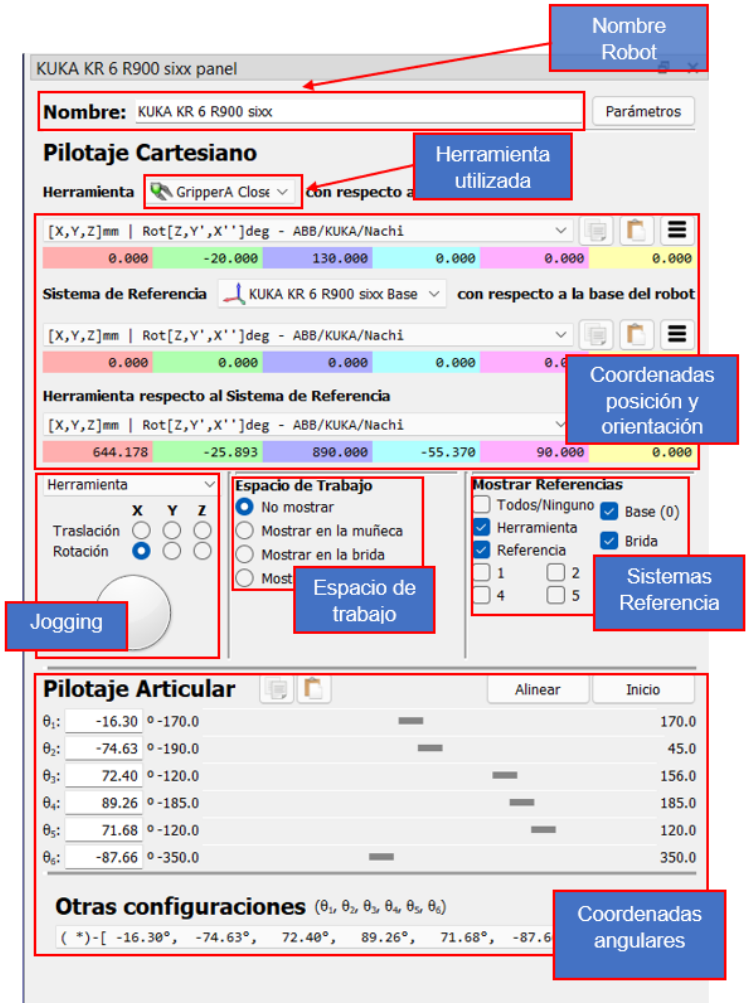


Figura 23: Panel Robot

Para más información, en la página web de RoboDK, (https://robodk.com/doc/es/Basic-Guide.html#Start) se encuentra una guía donde se detallan cada una de las secciones y se explica, con ejemplos incluidos, como dar los primeros pasos con este software.

6.4 Limitaciones de RoboDK

RoboDK es un software de simulación robótica muy completo, sin embargo, hay ciertas carencias dentro de sus posibilidades que se han tratado de cubrir en el desarrollo de la aplicación de este proyecto.

Para programar en RoboDK se hace uso de herramientas de movimiento que se arrastran al árbol de la estación. Concatenando distintas herramientas en el árbol se crea un programa o función. Sin embargo, RoboDK no permite la utilización del lenguaje TPE para escribir el código a implementar por la estación. Además, la aplicación desarrollada en este proyecto permite la configuración de las instrucciones en velocidad además de incluir la opción de añadir offset y tool offset.

La modificación de registros tanto de posición como escalares es una funcionalidad que añade esta herramienta con respecto al programa base de RoboDK. RoboDK no permite las instrucciones condicionales para realizar decisiones y saltos entre líneas de código. En cambio, la aplicación desarrollada sí permite implementar instrucciones condicionales y saltos a líneas de código específicas lo que facilita la reutilización de código.

7. DESARROLLO DE LA APLICACIÓN CREADA

Este capítulo se centrará en explicar el desarrollo de la aplicación creada durante este proyecto. El trabajo que se ha realizado para el desarrollo de esta herramienta se ve reflejado en este apartado. Para realizar la explicación del funcionamiento de la aplicación se ha evitado el uso de código, sustituyendo este por flujogramas que hacen más fácil la comprensión del funcionamiento de los distintos métodos y recursos que se usan en la aplicación.

Este es el capítulo clave para mostrar el trabajo realizado, que será presentado en los apartados 7.3 en lo que se refiere al diseño de la interfaz y 7.4 que detalla las funciones creadas para permitir la simulación del robot en un entorno previamente definido. Previamente, en los apartados 7.1 y 7.2 se explica brevemente

7.1 Entorno de desarrollo y API

7.1.1 Microsoft Visual Studio

Para la realización de esta herramienta se ha utilizado la API disponible para el lenguaje de programación C# suministrada por RoboDK la cual se ha obtenido mediante la descarga a partir de su sitio web [11].

C# es un lenguaje de programación orientado a objetos desarrollado por la compañía Microsoft y diseñado para el desarrollo de gran diversidad de aplicaciones que se ejecutan en .NET Framework. Durante el desarrollo de esta herramienta se han empleado tanto la guía de programación C# [12], como la documentación ofrecida por System Windows Forms de .NET Framework [13], ambas con recursos útiles tales como ejemplos y descripciones detalladas de métodos y funcionalidades propias de Visual Studio. La consulta frecuente de esta documentación ha ayudado significativamente en el desarrollo del código y de la interfaz.

7.1.2 Interfaces de la herramienta

El uso de la herramienta desarrollada precisa del manejo conjunto de dos ventanas (Figura 24), que trabajarán de forma simultánea durante la ejecución de la aplicación:

- La primera ventana es donde el usuario lleva a cabo el diseño de la estación, la programación del robot y la gestión de archivos de guardado. A esta ventana se le conocerá desde ahora como *Interfaz de Diseño*.

- La segunda ventana es donde se generará el modelo 3D de la estación diseñada y el lugar donde se podrán visualizar los resultados de la programación realizada en la *Interfaz de Diseño*. A lo largo de esta memoria se referirá a esta ventana con el nombre de *Interfaz de Visualizado*, o *Interfaz RoboDK.*

Estas dos utilidades, las cuales se inician de forma simultánea, llevan a cabo procesos independientes y se comunican a partir de *sockets*. Las comunicaciones más frecuentes entre estos dos procesos suelen ser la comprobación del estado del robot, instrucciones de movimiento, gestión de ítems, y envío y recepción de posiciones tanto del robot como de los objetos de su entorno entre otras.

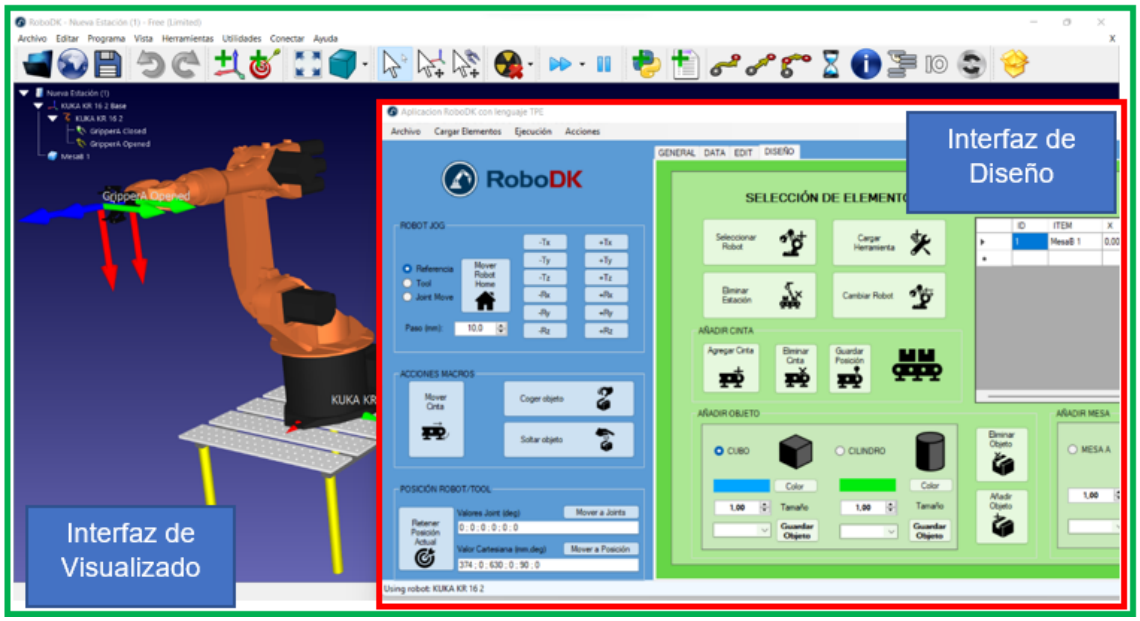


Figura 24: Interfaces de la aplicación

7.1.3 Descripción de la API

La API de RoboDK para C# se compone principalmente de 3 archivos fundamentales para su funcionamiento (Figura 25):

1. El archivo *Program.cs* que contiene la clase *Program* que a su vez tiene el método *main*, encargado de inicializar la aplicación y el formulario.
2. El fichero *RoboDK.cs* que contiene dos clases: Por una parte, la clase *Mat* que cuenta con las herramientas matemáticas necesarias para el tratamiento con matrices de transformación homogéneas (llamadas en la programación como “Pose”) y por otra, la clase *RoboDK*, encargada de gestionar los ítems usados por *RoboDK* y de realizar las comunicaciones entre interfaz de diseño y de visualización. Es importante aclarar lo que es un ítem, que es un tipo de variable

objeto que hace referencia a un elemento que se encuentra en el árbol de desarrollo de la estación; es decir, un ítem puede ser un programa, un robot, una posición, una herramienta etc. Los ítems son comunicados entre los dos procesos que lleva a cabo la aplicación gracias a métodos pertenecientes a la clase RoboDK.

3. El formulario *FromRobot.cs* es el que contiene la programación de las funciones y eventos de la interfaz gráfica. En este formulario es donde se ha llevado a cabo la mayor parte de la programación del proyecto ya que aquí se encuentran designadas las acciones a realizar por los distintos elementos de la interfaz gráfica. El formulario *FromRobot.cs* tiene una versión de diseño llamada *FromRobot.cs[Diseño],* en la que se muestra un diseño modificable de la interfaz. En este formulario de diseño se pueden arrastrar distintos elementos al panel de la aplicación, tales como botones, menús desplegables e imágenes entre otros. Una vez en este panel se pueden cambiar tamaño, posición, color y demás propiedades de estos elementos.

   Por otra parte, dentro del formulario *FromRobot.cs* se encuentra el fichero *FromRobot.Designer.cs* en el que se declaran los elementos añadidos al panel de la aplicación y en el que se encuentra también la programación relativa a las propiedades (tamaño, color, posición…) de los elementos.

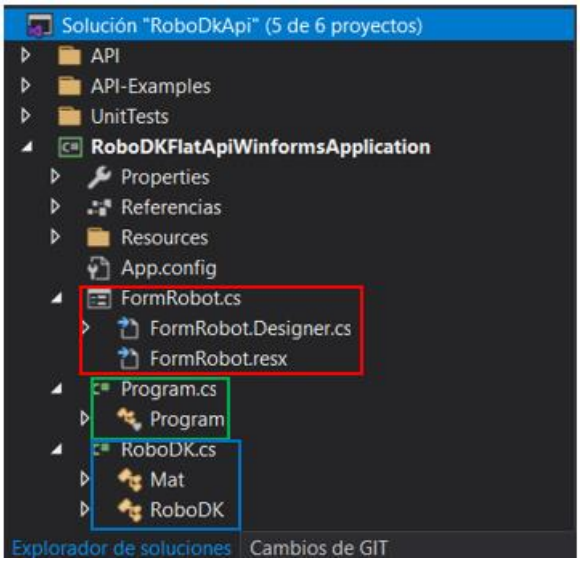


Figura 25: Ficheros de la aplicación

7.2 Interfaz de visualizado, RoboDK

La Interfaz de Visualizado o Interfaz RoboDK (Figura 26) es la encargada de mostrar los modelos 3D de robots, herramientas y objetos que se añaden a la estación. Este

proceso también lleva a cabo internamente la representación del movimiento a una posición determinada de los robots o de los objetos. Realiza las operaciones necesarias para poder visualizar y modificar los modelos de los elementos implicados en los movimientos a partir de las posiciones enviadas por la Interfaz de Diseño. En definitiva, este proceso hace las veces de motor gráfico de la herramienta completa. En la sección 6.3 Descripción del interfaz RoboDK de este documento, se ha desarrollado de forma más extensa las distintas utilidades que provee esta ventana.

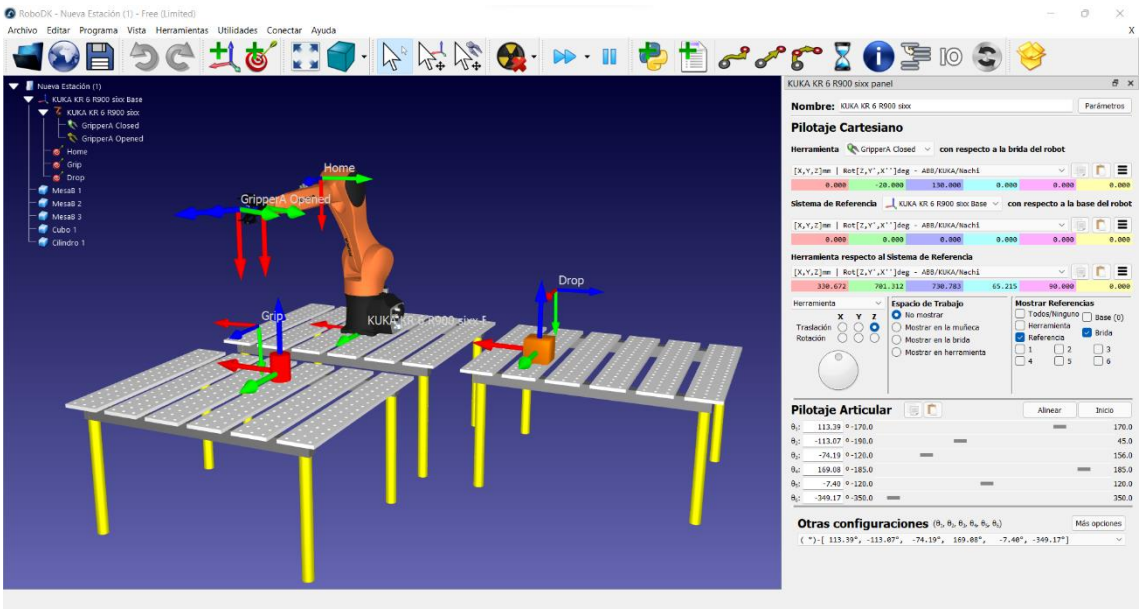


Figura 26: Interfaz de Visualizado

7.3 Interfaz de diseño

La interfaz de diseño (Figura 27) es la que se utiliza para la composición de la estación robótica la cual es representada a posteriori en la interfaz de visualizado. En esta ventana también se realizará la programación de la estación diseñada.

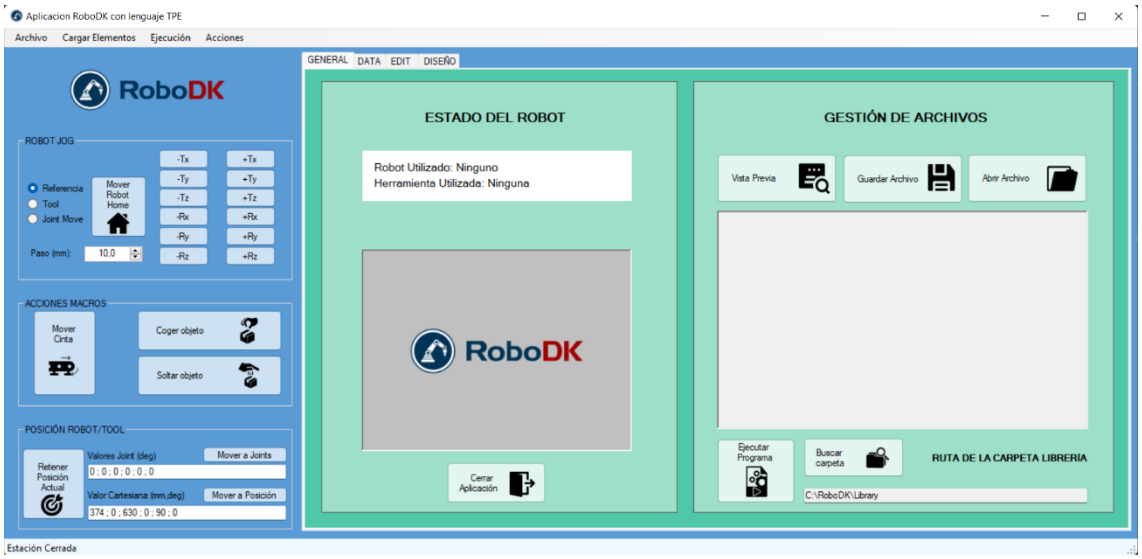


Figura 27: Interfaz de Diseño

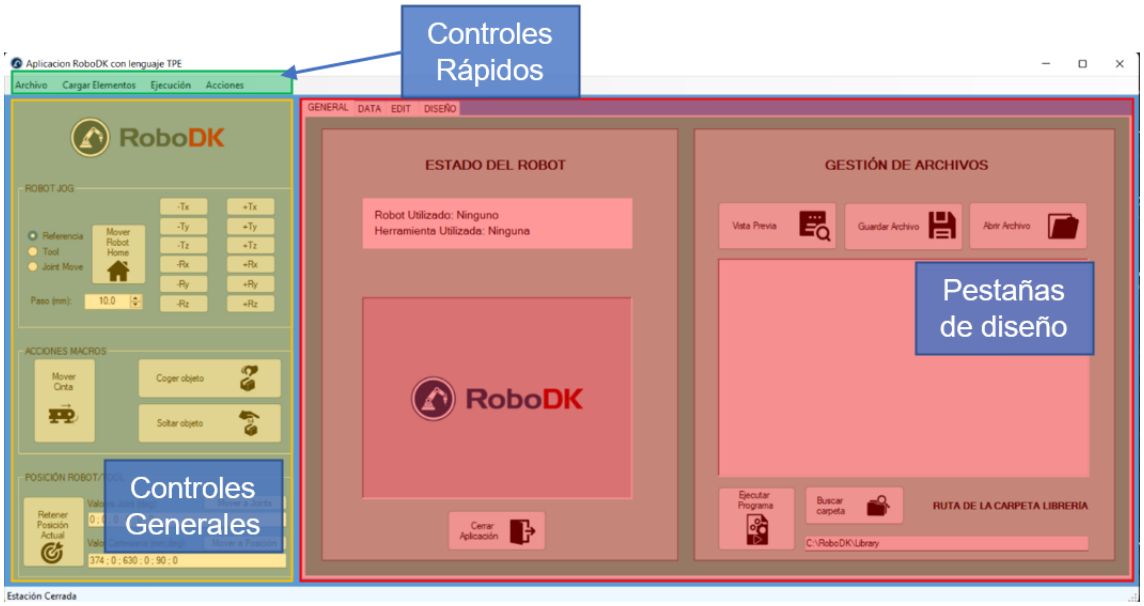


Figura 28: Secciones de la Interfaz de Diseño

Esta interfaz de diseño se puede dividir en tres secciones indicadas en la Figura 28:

1. Sección de Controles Rápidos: En la gran mayoría de aplicaciones, en la parte superior izquierda suele haber una serie de menús desplegables utilizados generalmente para la gestión de archivos y para la realización de otras funciones. Para el desarrollo de esta aplicación se ha visto conveniente integrarlos ya que este menú desplegable (Figura 29) contiene una serie de herramientas rápidas que evitan el cambio constante entre pestañas y hace más sencillo su uso.

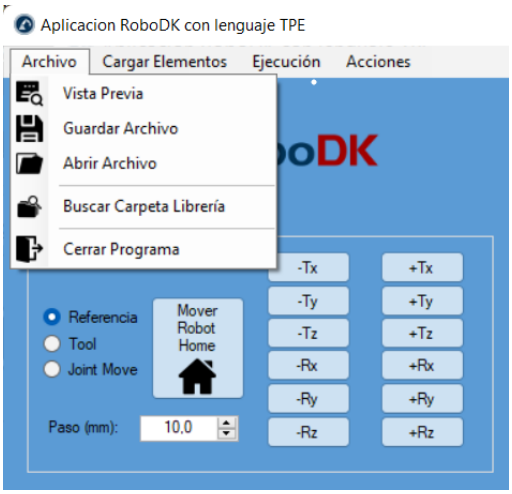


Figura 29: Menú Strip

2. Sección de Controles Generales: Esta área contiene los controles de movimiento que son aplicables a la estación en cualquier momento del proceso de diseño, aunque principalmente será usada durante la fase de programación y grabado de registros. Se subdivide a su vez en otras tres subáreas:

- Robot Jog: Incluye los controles manuales para mover al robot libremente representados de forma similar a como aparecen en una consola de programación real (Figura 30). Se puede ajustar el espacio recorrido en cada pulsación en el selector “Paso”. También hay un botón que devuelve el robot a su posición de referencia (Home) de forma automática. Finalmente, a la izquierda, hay un selector del sistema de referencia respecto al cual se moverá cuando se interactúe con los botones anteriores. Se puede elegir entre moverse respecto a la referencia de la base del robot (Referencia), respecto al sistema de referencia de la herramienta (Tool) o mover las articulaciones del robot por separado (Joint Move) (Figura 31).
- Acciones Macros: Aquí se encuentran las llamadas a las funciones de coger y soltar objeto por parte de la herramienta del robot como también la función de activar la cinta transportadora. El fin de esta subárea es poder comprobar durante la fase de programación, que el robot ancla las piezas de manera adecuada y que no se encuentra demasiado lejos del objeto.

- Posición Robot/Tool: Se trata de una subárea de información que servirá al usuario para determinar las coordenadas y ángulos de Euler (también los valores angulares de las articulaciones) que definen la posición y orientación del robot o de la herramienta en caso de tener alguna asociada. También permite realizar la función contraria y establecer al robot en una configuración específica a partir de coordenadas y ángulos.

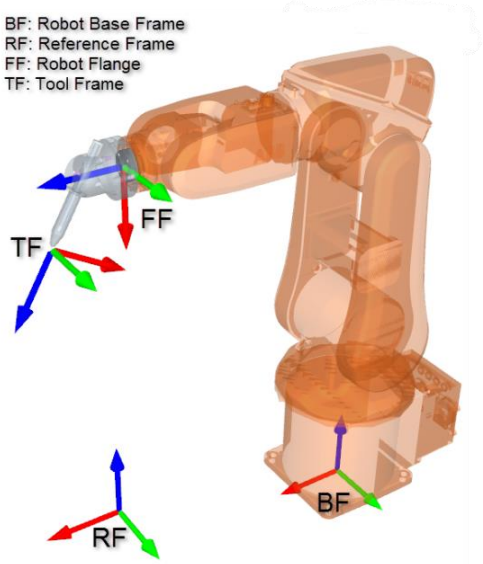


Figura 31: Sistemas de referencia de la estación

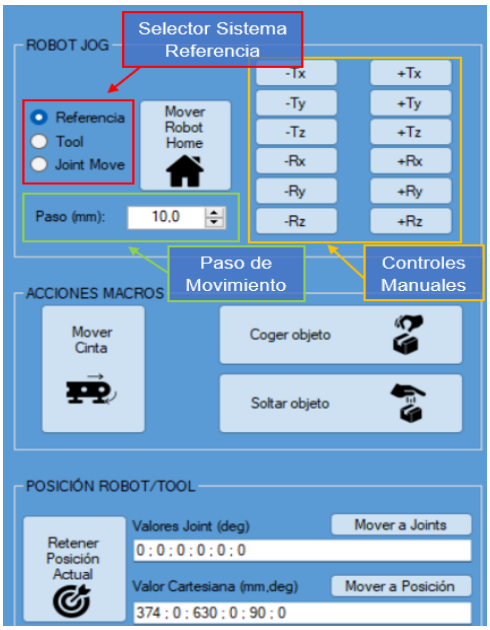


Figura 30: Detalle Sección Controles Generales

3. Sección de Pestañas (Figura 32): Se compone de un gran panel con un selector de pestañas en la parte superior izquierda. En estas pestañas es donde se realizará el diseño y programación de la estación. Las pestañas GENERAL, DATA, EDIT y DISEÑO desarrollaran en los subsiguientes apartados.

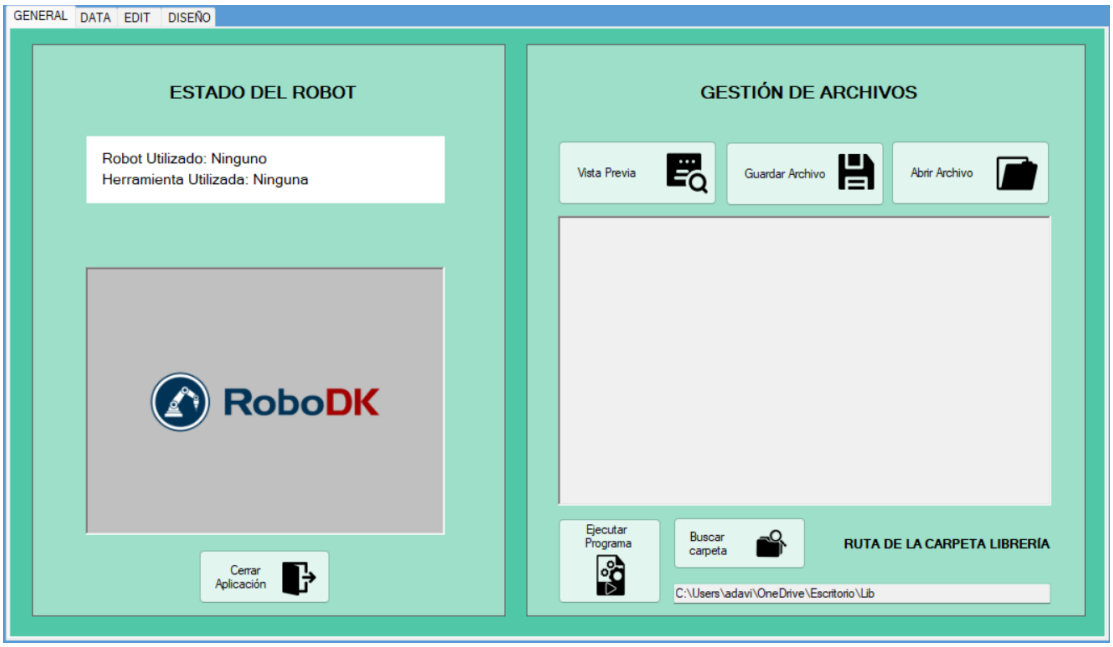


Figura 32: Sección de pestañas

7.3.1 Pestaña General

La pestaña General (Figura 33) es usada para el guardado, la apertura de archivos y la ejecución de programas, así como la visualización general del estado actual del robot. Hay dos zonas bien diferenciadas: el estado del robot y la gestión de archivos.

Estado del robot es una subárea de carácter informativo en la que se representa el robot que se ha seleccionado para la estación como la herramienta (aunque en este proyecto solo se podrá añadir una pinza) y su estado (abierta o cerrada). Cuenta con el botón de cierre de aplicación que concluye los procesos de visualizado y de diseño, y es la forma recomendada de cerrar la herramienta tras su uso.

Por otra parte, está la zona de gestión de archivos. En el centro se observa un panel central donde se visualizará el programa completo al realizar una vista previa. Además de la vista previa, desde esta sección se podrá tanto guardar como abrir archivos, así como ejecutar un programa ya diseñado. Por último, en la parte inferior se halla la zona de escaneo de la ruta de la librería de objetos que se explicará más adelante en el capítulo 8.1.1 Cómo instalar e iniciar la aplicación.

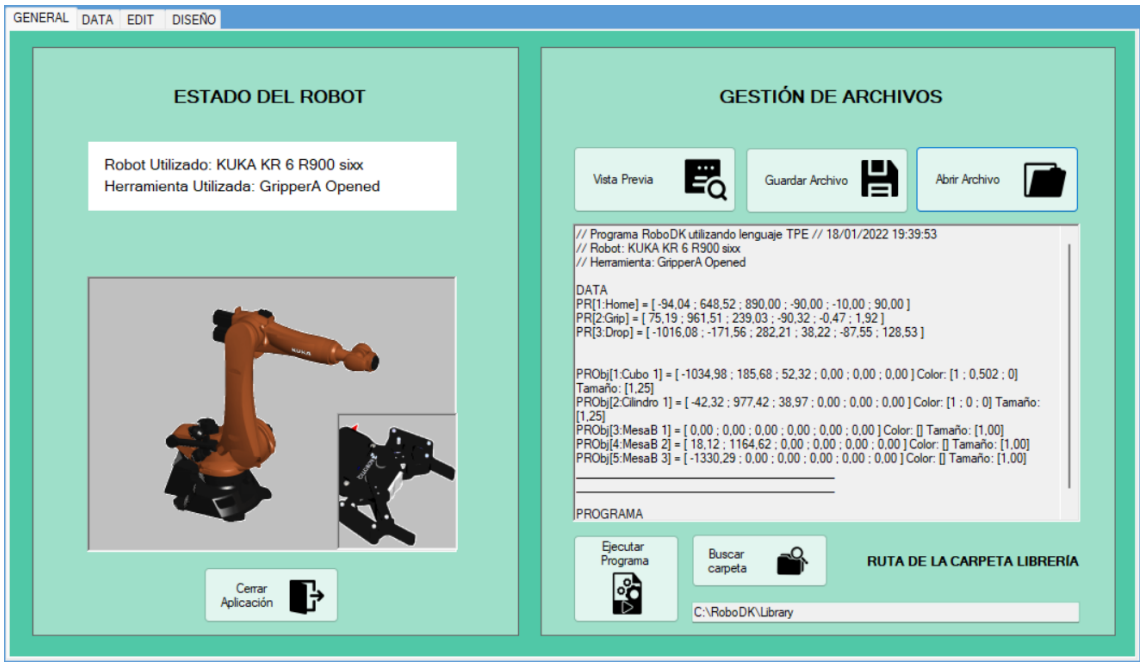


Figura 33: Pestaña General

7.3.2 Pestaña Data

Desde esta pestaña (Figura 34), se llevará acabo el almacenamiento de registros que serán usados más adelante en el programa que se vaya a diseñar. Se distinguen dos secciones:

- Registros de Posiciones PR[ ]: Aquí se lleva a cabo el registro de las distintas posiciones clave que adoptará el robot durante la ejecución del programa en forma de 6 elementos: 3 coordenadas cartesianas, X,Y,Z para la posición y 3 ángulos de Euler, W,P,R (*Ya**w**, **P**itch, **R**oll*) para la orientación. Se puede tanto agregar un registro de posición (u offset) como modificarlo o eliminarlo, además se puede añadir una etiqueta identificativa de cada registro.

- Registros Escalares R[ ]: De una forma muy similar al registro de posiciones, esta zona se encarga de guardar los registros escalares que se necesitarán para la posterior modificación de registros o uso en instrucciones condicionales durante la ejecución. De igual forma, se pueden añadir, modificar y eliminar registros, que esta vez constarán solo de un elemento.

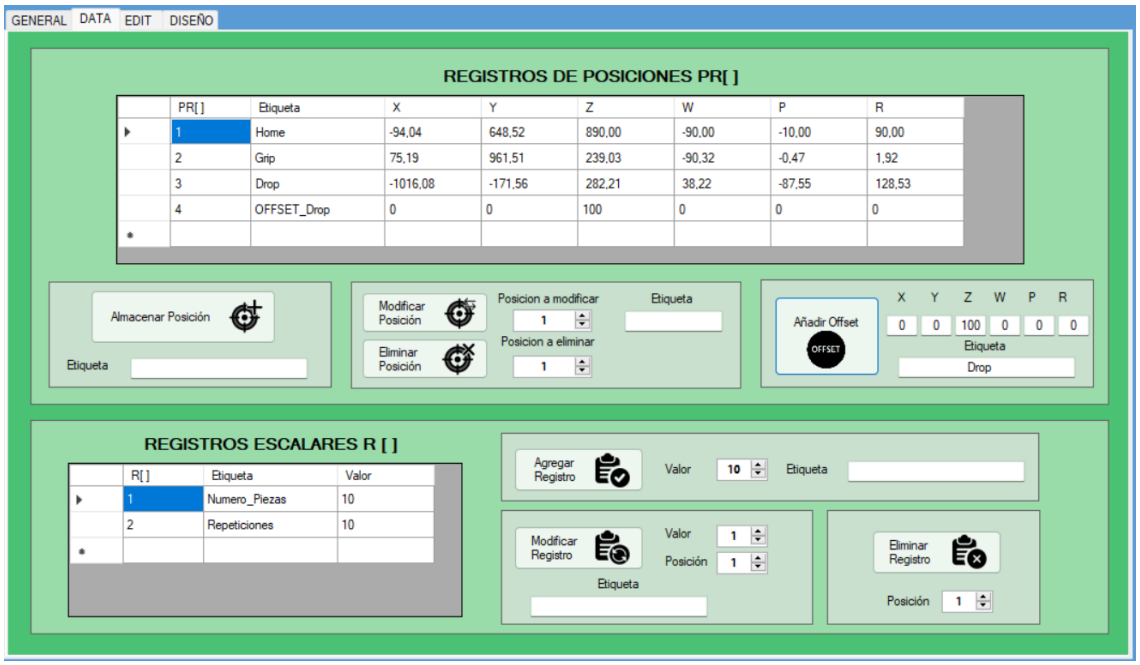


Figura 34: Pestaña Data

7.3.3 Pestaña Edit

La programación de la estación se llevará a cabo en esta pestaña Edit (Figura 35) donde encontramos un panel en la parte inferior en el cual se redactará el programa conforme se vaya diseñando, y una serie de secciones que contienen las instrucciones principales del lenguaje TPE disponibles para configurar el programa. Estas secciones son:

- Sección de instrucciones de movimiento: permite la configuración de los distintos parámetros (tipo de movimiento, velocidad, registro, offset…) para escribir una línea de instrucción de movimiento.
- Sección de instrucciones Label/Jump: desde aquí se pueden añadir etiquetas y saltos a etiquetas específicas.
- Sección de instrucción Wait: sirve para agregar una instrucción de espera de X segundos.
- Sección de instrucciones condicionales: su finalidad es añadir una instrucción de salto condicional al conjunto del programa.
- Sección de instrucciones de modificación de registros: esta sección contiene dos pestañas, una para agregar una instrucción de modificación de registro escalar y otra para los registros de posición.
- Sección de instrucción coger/soltar objeto: permite agregar una instrucción de cerrar o abrir pinza.
- Sección de instrucción cinta transportadora: añade una instrucción que inicia la subrutina de la cinta transportadora.

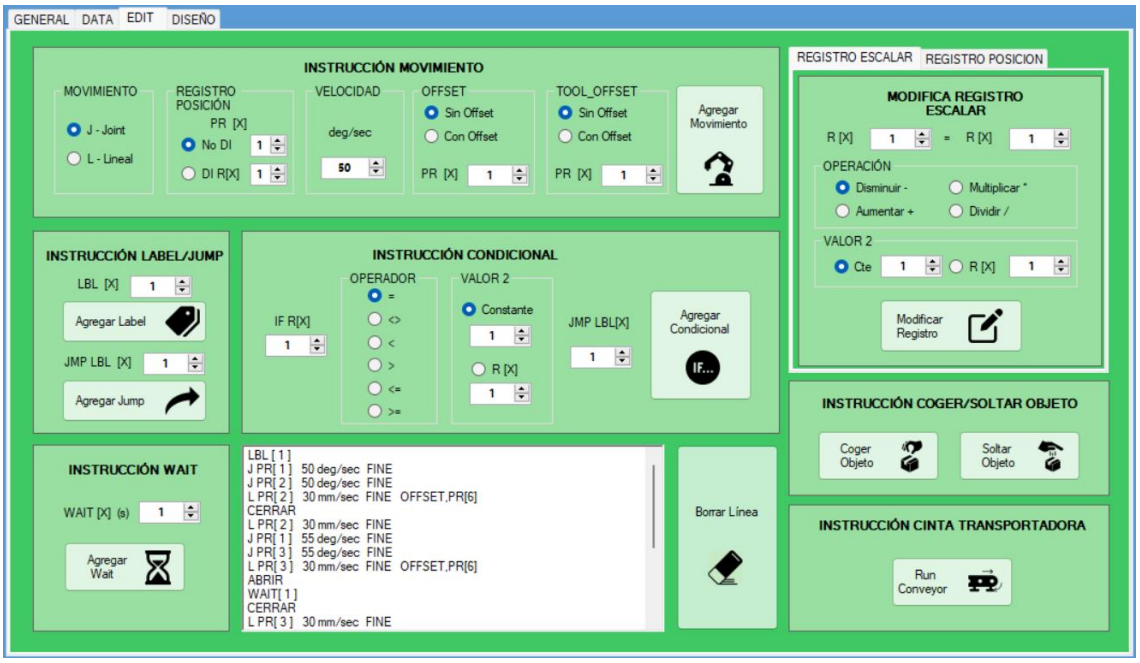


Figura 35: Pestaña Edit

7.3.4 Pestaña Diseño

Esta última pestaña (Figura 36) será la encargada de componer el diseño de la estación como tal y llevar a cabo el registro de posición de elementos auxiliares como lo son los objetos, las mesas y las cintas transportadoras.

En un primer lugar se observan cuatro botones en la parte de Selección de Elementos que como su nombre indica serán de utilidad a la hora de agregar/cambiar un robot, agregar una herramienta o eliminar una estación al completo. Inmediatamente debajo se encuentra la sección dedicada a la carga de cintas transportadoras, desde donde se podrá añadir y eliminar una cinta transportadora. En la parte inferior, las secciones de carga de objetos y mesa nos permiten hacer lo propio con estos elementos y además se tiene la posibilidad de modificar el color y el tamaño de los mismos. Todos estos ítems se guardarán en la tabla de registro de posición, que se encuentra en la parte superior derecha, cuya forma de operar es similar a las tablas de registro de la pestaña Data.

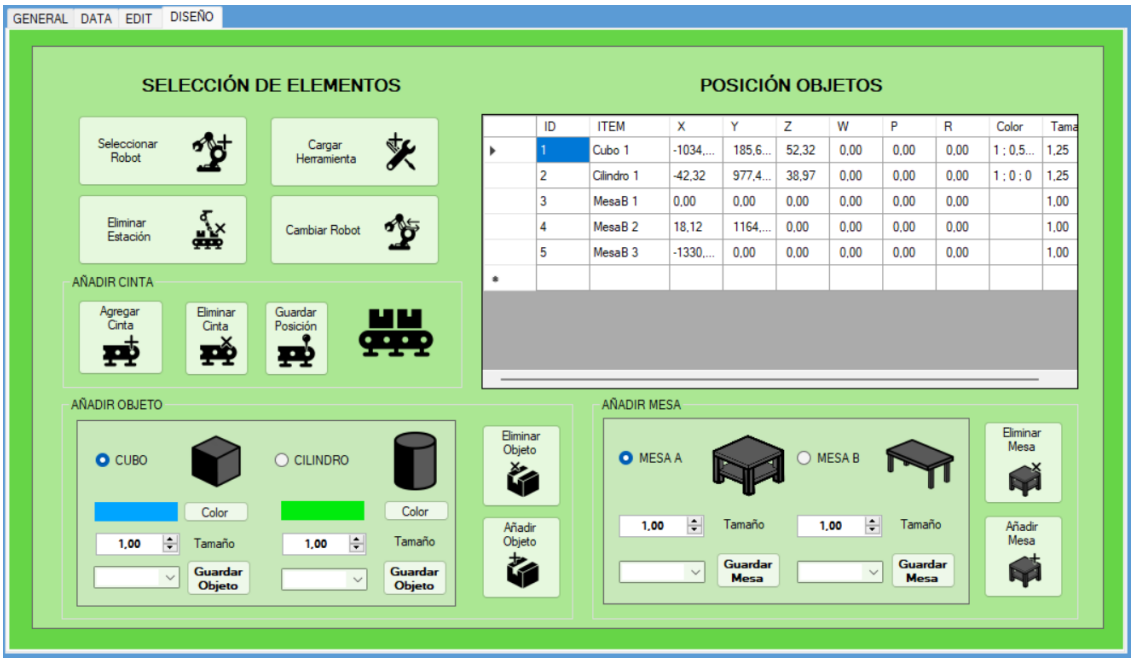


Figura 36: Pestaña Diseño

7.4 Exposición del funcionamiento de los métodos principales del código y su nexo con funciones propias de ROBODK

Una vez realizada la descripción de las interfaces se continuará con la parte más fundamental de este proyecto: el código desarrollado que da funcionalidad a la aplicación. A lo largo de este apartado se expondrán los métodos más importantes que se han diseñado y programado para el correcto funcionamiento de la aplicación. Se hará uso de flujogramas explicativos acompañados de una descripción detallada de las distintas funciones y operaciones que estos representa. El objetivo de este apartado es lograr una compresión del funcionamiento interno de la herramienta de la forma más sencilla posible y sin hacer uso de código salvo excepciones contadas. En el apartado 8.1 Modo de uso se llevará a cabo una explicación del funcionamiento externo, es decir, del modo de uso de la aplicación para poder utilizarla sin tener la necesidad de conocer los mecanismos internos de la misma.

Los métodos principales se pueden subdividir en 4 categorías diferentes (Figura 37):

- Métodos de guardado de archivos: Son aquellos métodos cuyo cometido es codificar, dar formato y componer un conjunto de ficheros que recogen la información del programa creado.
- Métodos de apertura de archivos: Se engloba dentro de esta categoría a los métodos dedicados a abrir los ficheros de guardado, cargar los

registros de posición, escalares y de objetos y representar el modelo de la estación en la interfaz de visualizado.

- Métodos de ejecución de programa: En este apartado se aúnan los métodos encargados de ejecutar un programa escrito previamente. Esta categoría es la más extensa y la más crítica para el correcto funcionamiento de la aplicación
- Otros métodos: En esta categoría se incluyen aquellos métodos con funcionalidad variada, aunque vinculados al diseño de la estación y el programa. Componer las distintas líneas de programa, actualizar posiciones y valores en registros o añadir elementos a la estación son algunas de las funcionalidades que se verán en esta categoría. Debido a su sencillez no se ha visto conveniente que dispongan de una categoría separada.

Figura 37: Métodos principales por categorías

7.4.1 Métodos de guardado de archivos

Para comenzar, se desarrollarán los métodos de guardado de ficheros. Tras el diseño de la estación y la elaboración del programa a ejecutar, es necesario guardar todos estos archivos en algún directorio para poder abrirlos a posteriori, bien sea para modificarlos o para ejecutarlos. Para el guardado, se crea una carpeta con el nombre del programa en el directorio seleccionado, la cual va a albergar 5 ficheros que son los que componen el programa como tal (Figura 38). La estructura de estos ficheros es la siguiente:

Nombre del programa + tipo de fichero + extensión del archivo.

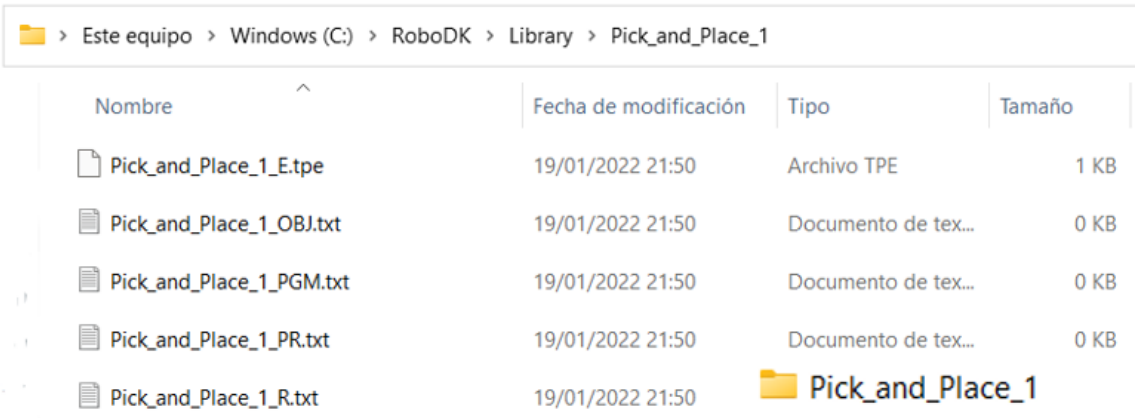


Figura 38: Carpeta y ficheros de guardado

Los 5 ficheros que alberga la carpeta tienen su mismo nombre seguidos de lo que aquí se ha llamado como tipo de fichero. Estos ficheros están protegidos contra escritura, de modo que solo se puedan modificar desde la aplicación desarrollada. Son los siguientes:

A. Tipo de fichero "_E": Este fichero es el único que se guarda con la extensión ".tpe", esto es así ya que al abrir un nuevo programa solo se verán en la ventana de examinar los archivos con esta extensión. El objetivo de esta práctica es que sea el único de los 5 ficheros visible a la hora de abrir un programa evitando así el error de seleccionar otro fichero.

   El contenido de este fichero es la versión estilizada del programa al completo, que es la que se muestra en el panel de vista previa; es decir, es la versión sin codificar del programa junto con todos los registros almacenados.

B. Tipo de fichero "_OBJ": Este y los siguientes ficheros tienen una extensión ".txt" ya que son archivos de texto, aunque tal y como se ha dicho antes están protegidos contra escritura para evitar alterar de forma manual con un editor el texto el contenido del mismo. Aquí se almacenan los registros de posición de los (objetos o mesas) guardados en la pestaña Diseño.
C. Tipo de fichero "_PGM": Contiene las líneas del programa diseñado en la pestaña Edit.
D. Tipo de fichero "_PR": Alberga los registros de posición del robot.
E. Tipo de fichero "_R": Guarda los registros escalares necesarios para el funcionamiento del programa

7.4.1.1 Método Guardar_Archivo()

Este método se ejecuta tras pulsar el botón de guardar archivo de la pestaña General. En la Figura 39 se muestra su funcionamiento. Primero se comprueba si se ha realizado

una vista previa o no y si es así se pide al usuario seleccionar un directorio donde guardar. A continuación, se comprueba si el archivo ya existe en esa ruta, en cuyo caso se preguntará al usuario si quiere sobrescribirlo o no. Si se hace la sobreescritura primero se han de dar permisos de escritura para eliminar los archivos anteriores y escribir los nuevos. Finalmente se crea una carpeta con el nombre del programa, donde se almacenarán los archivos.

A continuación, se realiza una rutina que se repite 5 veces con algunas modificaciones para cada uno de los archivos que se guardan en la carpeta. Esta rutina consiste en crear el archivo con su extensión, atribuir el permiso de solo lectura, llamar a la función que reconstruye el contenido del archivo codificado para su posterior lectura y finalmente la escritura en el archivo creado al principio.

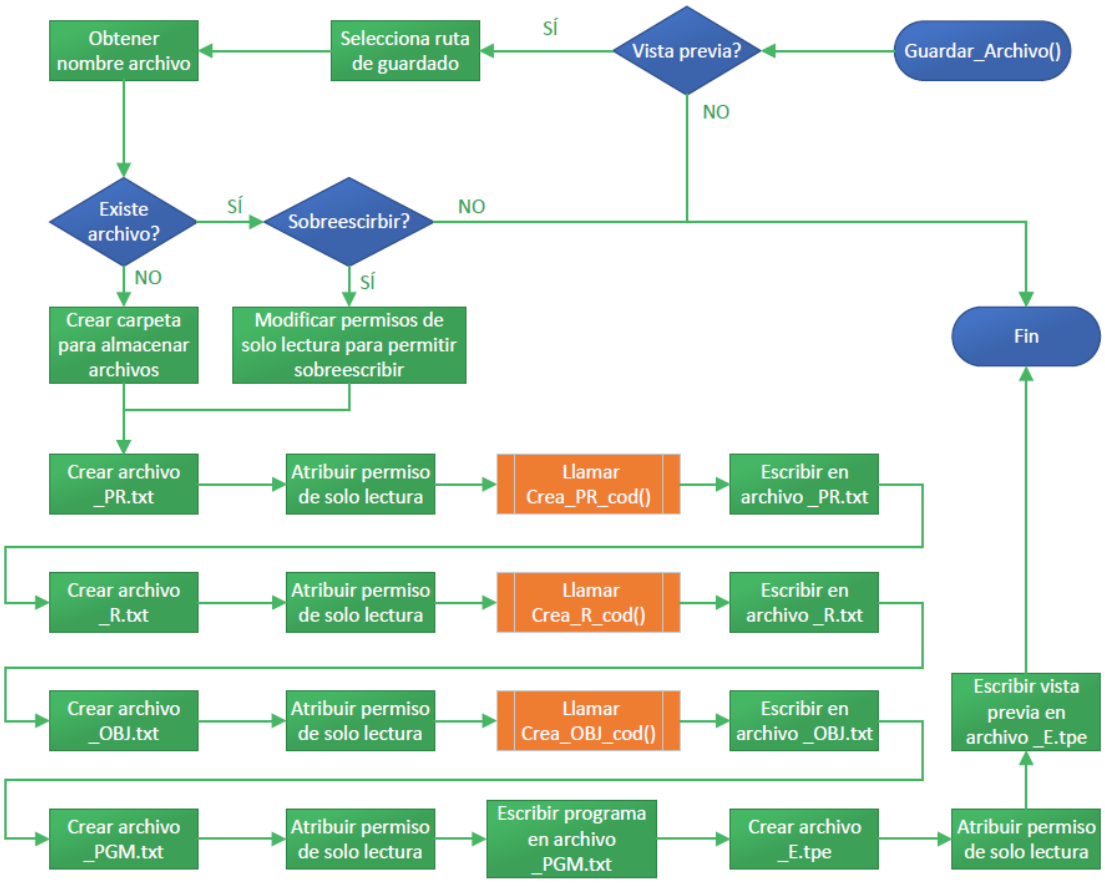


Figura 39: Algoritmo de guardado de archivos

7.4.1.2 Funciones auxiliares de guardado

El algoritmo de guardado de archivos hace uso de una serie de funciones auxiliares cuya finalidad es codificar el contenido de cada fichero a guardar de modo que al abrir el programa se lean estos ficheros codificados. El motivo de por qué se codifican se explicará en el apartado de apertura de archivos 7.4.2.

Las funciones auxiliares utilizadas durante el guardado son las siguientes:

- *Crea_PR_cod()*: Esta función se dedica a leer la tabla del registro de posiciones y escribir cada fila en un fichero con la siguiente configuración:

  PR[fila :# etiqueta] = [$ vector posición y orientación]

- *Crea_R_cod()*: Su cometido es el mismo que la función anterior solo que ahora se lee el registro escalar y lo escribe de la siguiente forma:

  R[fila :# etiqueta] = [$ valor escalar]

- *Crea_OBJ_cod()*: Opera de manera idéntica a las anteriores funciones, pero con el registro de posición de los ítems. Como novedad, añade los elementos de color y tamaño. El valor de color viene definido como código RGB de la forma ([R;G;B]) y el valor de tamaño es un valor escalar que hace referencia a un escalado proporcional del elemento en las tres dimensiones. Cada fila generada adquiere la siguiente estructura:

PRObj[fila :# etiqueta] = [$ vector posición y orientación] $ color $ tamaño

El archivo _PGM también está codificado solo que este no necesita una función exclusiva para ello, sino que se va componiendo el archivo codificado mientras se diseña el código del programa TPE.

7.4.2 Métodos de apertura de archivos

Al igual que con la categoría de métodos de guardado, para la apertura también se hace uso de un método general que en este caso es *Abrir_Archivo()* el cual llama a otras funciones auxiliares con cometidos diversos. La apertura de archivos tiene la finalidad de abrir los ficheros guardados previamente y representar su contenido en la aplicación, por una parte, construyendo las tablas de registro a partir de los datos guardados, así como escribir el programa y la vista previa, y por otra cargar la estación en la interfaz de visualizado RoboDK.

7.4.2.1 Método Abrir_Archivo()

El método *Abrir_Archivo()* (Figura 40) es el método general de apertura que se pone en marcha al pulsar ABRIR ARCHIVO en la pestaña general. El proceso llevado a cabo es el siguiente:

1. Primero se consulta al usuario si se desea borrar la estación que se encuentra actualmente representada en la interfaz de visualizado.
2. Después se pide seleccionar el directorio donde se encuentra el programa TPE que se desea abrir (para evitar errores solo será posible seleccionar archivos de extensión .tpe.
3. Una vez seleccionado el programa, este se abre en el panel de vista previa y carga el robot y la herramienta asociados para proceder finalmente a llamar a las funciones auxiliares que se encargan de abrir los ficheros de los registros. Tras cada ejecución de estas funciones auxiliares se comprueba que la salida de esta no tenga ningún error; en caso de error la apertura del programa abortará y se mostrará un mensaje de error.

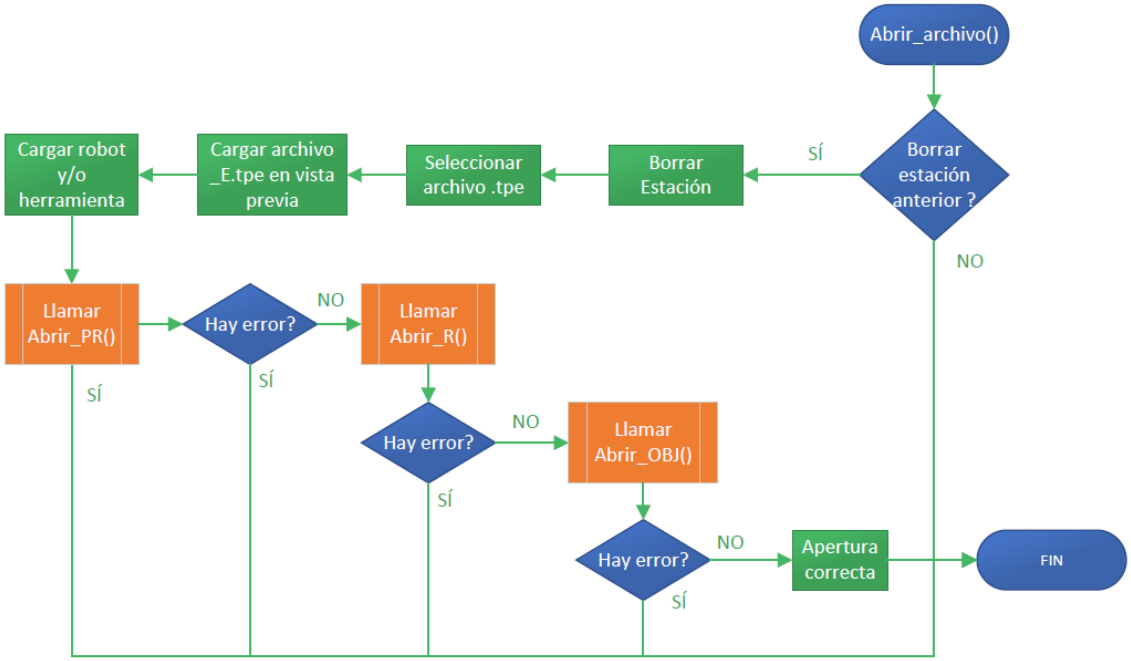


Figura 40: Algoritmo apertura de archivos.

7.4.2.2 Función Abrir_OBJ()

Esta función es la utilizada para cargar los distintos ítems en la interfaz de visualizado y en el registro de la pestaña diseño. Realiza las siguientes acciones:

1. Primero lleva a cabo una serie de operaciones para obtener la ruta del fichero _OBJ a partir del nombre del archivo .tpe seleccionado.
2. A continuación, se limpia la tabla de registro de la pestaña diseño y se comprueba que existe el archivo _OBJ; en caso contrario, devuelve un valor falso a la función general de apertura.
3. Después convierte el fichero .txt en un vector cuyos elementos son cada una de las líneas del fichero, se segmenta atendiendo a los símbolos "$" y "#" del apartado de creación de archivos y se actualiza la tabla de posiciones de objetos.
4. Luego se obtiene la ruta de la librería de objetos que se ha introducido en la pestaña general y se busca el archivo .sld que corresponde a cada objeto en función del nombre.
5. Finalmente, si esta última tarea tiene éxito y se encuentra el archivo .sld se procede a llamar a la función *Cargar_Ítems()*. El flujograma de la Figura 41 expone de manera clara todo este proceso.

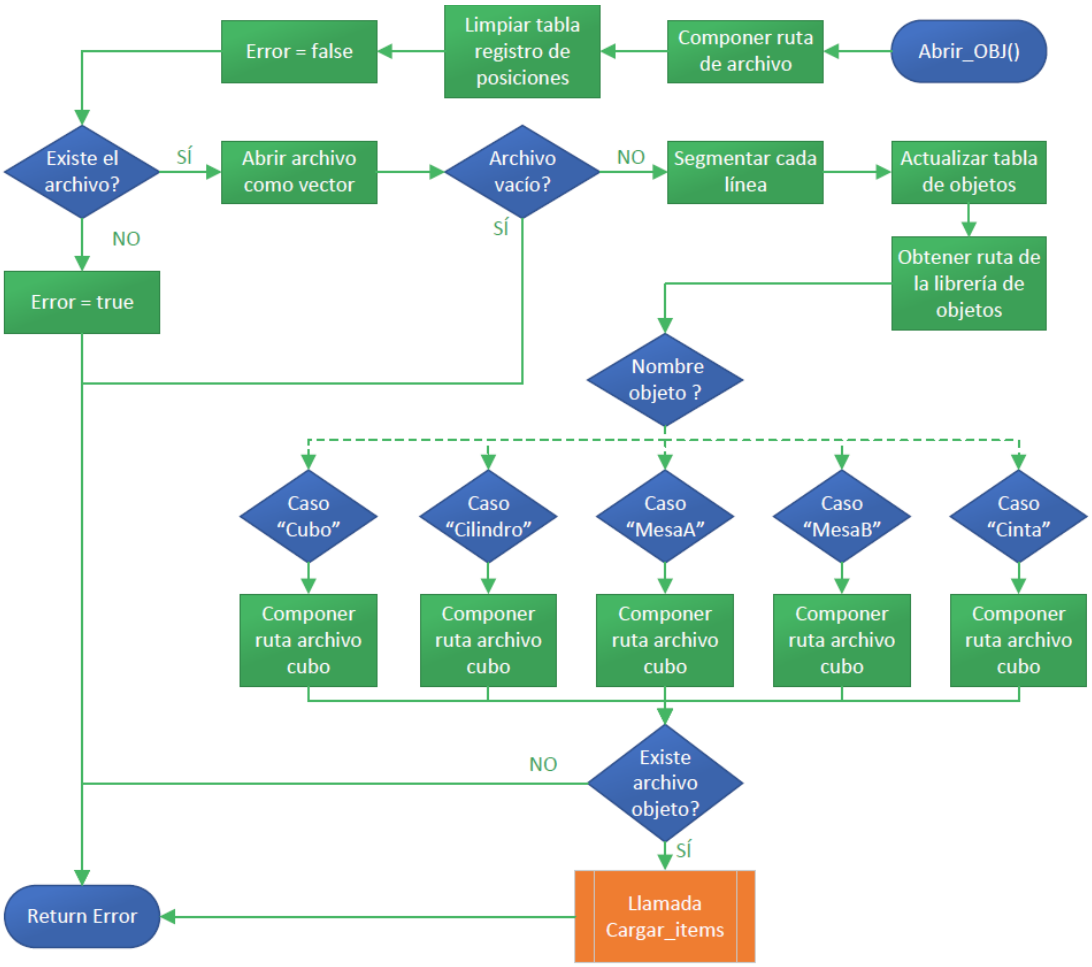


Figura 41: Flujograma Abrir_OBJ()

7.4.2.3 Función Abrir_PGM()

El cometido de esta función (Figura 42) es abrir el archivo que contiene el programa en TPE codificado y mostrarlo en el panel de la pestaña edit.

La forma de operar de la función es la siguiente, primero se obtiene la ruta del fichero _PGM a partir del nombre del archivo .tpe seleccionado. Después se limpia el panel de programa en la pestaña edit y se pasa a comprobar si existe el fichero en la ubicación compuesta. A continuación, se genera un vector cuyos elementos son cada una de las líneas del archivo _PGM de modo que cada elemento del vector es una línea del programa. Finalmente se carga el programa en el panel de programa.

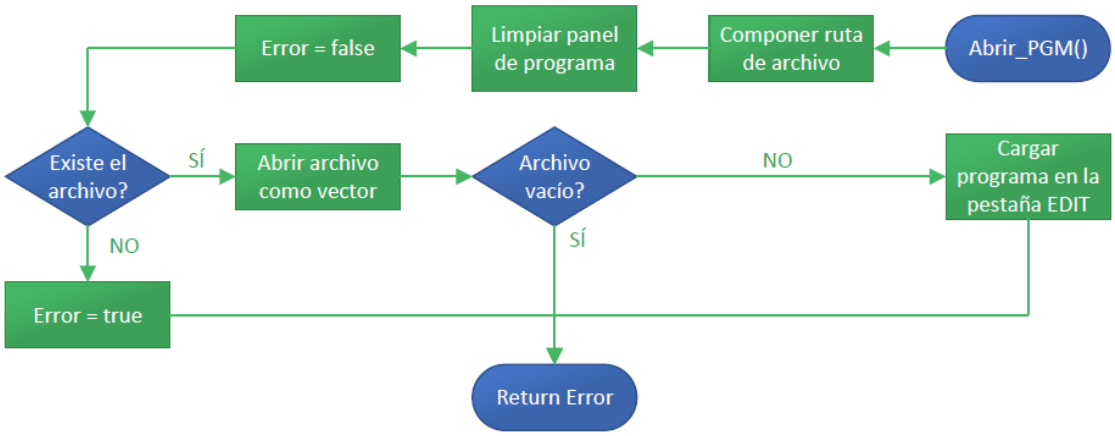


Figura 42: Flujograma Abrir_PGM()

7.4.2.4 Funciones Abrir_PR() y Abrir_R()

Estas dos funciones trabajan de manera similar entre sí y a la anterior. La única diferencia es que estas vuelcan la información del vector compuesto en el registro de posiciones y el registro escalar respectivamente. En la función *Abrir_PR()* (Figura 43) además se representan los sistemas de referencia de cada posición guardada en la interfaz de visualizado.

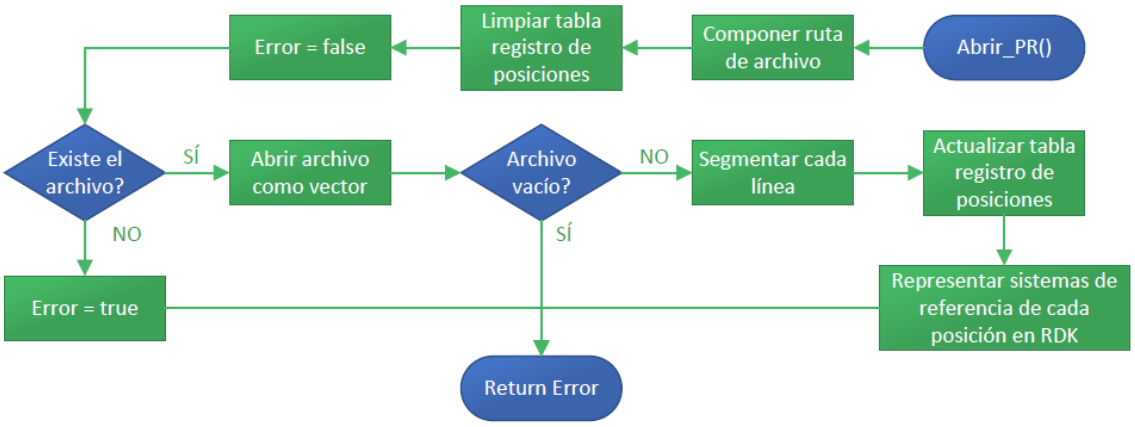


Figura 43: Flujograma Abrir_PR()

7.4.2.5 Método Cargar_Ítems()

La última parte del método de apertura del fichero _OBJ hace una llamada a la función *Cargar_Ítems()* (Figura 44) que es la encargada de cargar cada objeto en la estación y configurar su posición, tamaño y color. El proceso es el siguiente:

1. Lo primero que hace es comprobar que existe el archivo .sld en la ruta configurada como ruta de librería de objetos. En caso afirmativo se llama a la función *Generar_Ítems()* que carga los objetos de manera predeterminada en la posición 0 del sistema de referencia general de la estación (aquí se llama al método *AddFile()* de la clase RoboDK).
2. Luego se pasa a generar la matriz de transformación homogénea (“Pose”) que identifica la posición y orientación a partir de las coordenadas cartesianas y ángulos de Euler que se ha guardado en el fichero. El ítem pasa a tomar la posición indicada mediante la llamada al método de la clase RoboDK *setposeAbs()*.
3. Finalmente se comprueba si es un cilindro o un cubo para establecer el color (método RDK *recolor()*) También verifica si no es una cinta a fin de establecer su tamaño (método RDK *scale()*) ya que solo las mesas y los objetos manipulables pueden variar su tamaño.

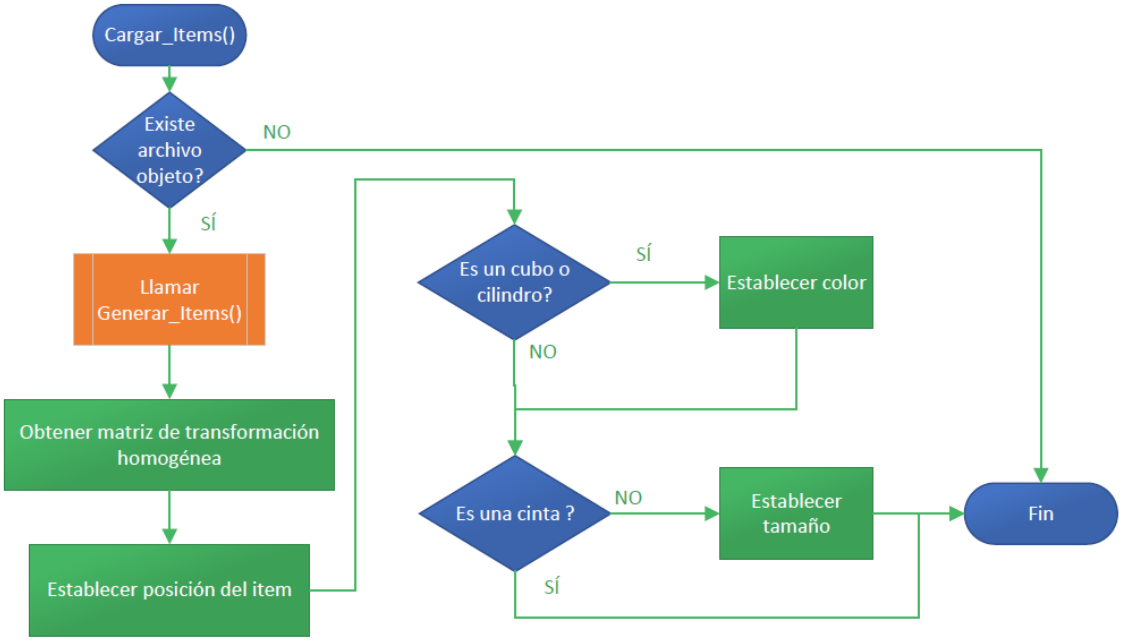


Figura 44: Algoritmo de carga de ítems

7.4.3 Métodos de ejecución de programa

En este subcapítulo se desarrollará la parte de interpretación y ejecución del programa diseñado en lenguaje TPE. Los programas desarrollados tienen dos variantes, una que es la variante estilizada que es la que se presenta al usuario en la pestaña Edit y otra que es la codificada que queda oculta al usuario (Figura 45). Para la interpretación del programa se hará uso de la versión codificada, en la que como se puede apreciar, cada tipo de instrucción viene encabezada por un símbolo identificativo (ID) que la diferencia de las demás. En la Tabla 4 se muestran las distintas instrucciones y el símbolo que les corresponde.

El método principal de esta sección es el método *Run_Program()* el cual llama a otras funciones auxiliares que se encargan de interpretar y ejecutar cada tipo de instrucción.

```
PROGRAMA
 LBL [ 1 ]
 J PR[ 1 ]   50 deg/sec  FINE
 J PR[ 2 ]   50 deg/sec  FINE
 L PR[ 2 ]   30 mm/sec  FINE   OFFSET,PR[6]
 CERRAR
 L PR[ 2 ]   30 mm/sec  FINE
 J PR[ 1 ]   55 deg/sec  FINE
 J PR[ 3 ]   55 deg/sec  FINE
 L PR[ 3 ]   30 mm/sec  FINE   OFFSET,PR[6]
 ABRIR
 WAIT[ 1 ]
 CERRAR
 L PR[ 3 ]   30 mm/sec  FINE
 J PR[ 1 ]   50 deg/sec  FINE
 J PR[ 2 ]   50 deg/sec  FINE
 L PR[ 2 ]   30 mm/sec  FINE   OFFSET,PR[6]
 ABRIR
 L PR[ 2 ]   30 mm/sec  FINE
 J PR[ 1 ]   50 deg/sec  FINE
 R[1] = R[1] - 1
 IF R[ 1 ] =  0  JMP LBL [ 1 ]
 WAIT[ 2 ]
```

```
PROGRAMA
! LBL[1]
$ J PR[1] 50deg/sec FINE ?
$ J PR[2] 50deg/sec FINE ?
# L PR[2] 30mm/sec FINE OFFSET,PR[6] ?
® CERRAR
$ L PR[2] 30mm/sec FINE ?
$ J PR[1] 55deg/sec FINE ?
$ J PR[3] 55deg/sec FINE ?
# L PR[3] 30mm/sec FINE OFFSET,PR[6] ?
© ABRIR
% WAIT[1]
® CERRAR
$ L PR[3] 30mm/sec FINE ?
$ J PR[1] 50deg/sec FINE ?
$ J PR[2] 50deg/sec FINE ?
# L PR[2] 30mm/sec FINE OFFSET,PR[6] ?
© ABRIR
$ L PR[2] 30mm/sec FINE ?
$ J PR[1] 50deg/sec FINE ?
Æ R[1] =R[1] - 1
Ø IFR[1] = 0 JMPLBL[1]
% WAIT[2]
```

Figura 45: Ejemplo de programa sin codificar (izquierda) y codificado (derecha)

IDENTIFICADORES DE CADA INSTRUCCIÓN			
SÍMBOLO (ID)	**INSTRUCCIÓN**	**SÍMBOLO (ID)**	**INSTRUCCIÓN**
“ $“	MOVIMIENTO SIN OFFSET	“ß”	MODIFICAR REGISTRO DE POSICIÓN (REGISTRO)
“#”	MOVIMIENTO CON OFFSET	“«”	MODIFICAR REGISTRO DE POSICIÓN (CONSTANTE)
“µ”	MOVIMIENTO CON TOOL OFFSET	“Ø”	CONDICIONAL CONSTANTE
“%”	WAIT	“¥”	CONDICIONAL R[X]
“&”	JUMP LABEL	"®"	COGER OBJETO
“!”	LABEL	"©"	SOLTAR OBJETO
“£”	MODIFICAR REGISTRO ESCALAR (REGISTRO)	"½"	RUN CONVEYOR
“Æ”	MODIFICAR REGISTRO ESCALAR (CONSTANTE)		

Tabla 4: Identificadores de instrucción

7.4.3.1 Método Run_Program()

El método *Run_Program()* (Figura 46), como se ha dicho antes, es el algoritmo general para la interpretación y ejecución de los programas TPE. Este método llama consecutivamente a una serie funciones auxiliares que se encargan de interpretar y ejecutar cada tipo de instrucción. Los pasos del algoritmo son los siguientes:

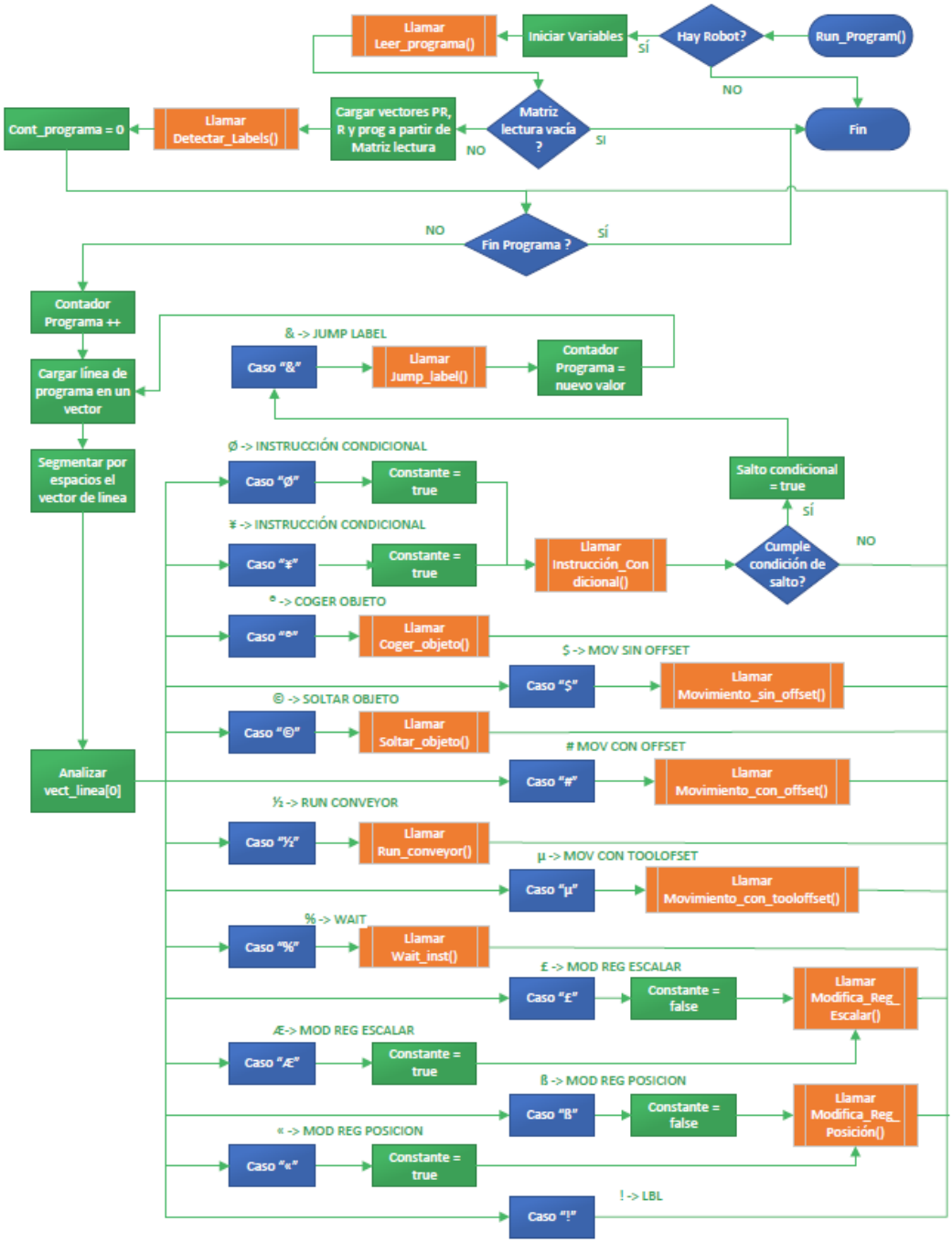


Figura 46: Algoritmo de ejecución de programa

1. El algoritmo comienza comprobando si hay algún robot en la estación ya que es necesario para que se puedan ejecutar las distintas instrucciones.

Si es así continúa iniciando variables auxiliares y llamando a la función *Leer_Programa()* que se encarga de obtener los distintos registros y el programa como tal de una forma específica dentro de una matriz.

2. Luego se comprueba si la matriz leída está vacía en cuyo caso no habría programa y se pasa a finalizar la ejecución del mismo.
3. Posteriormente se generan 3 vectores a partir de la matriz de lectura: uno para las líneas de programa, otro para los registros escalares y otro para los registros de posición (Figura 47).

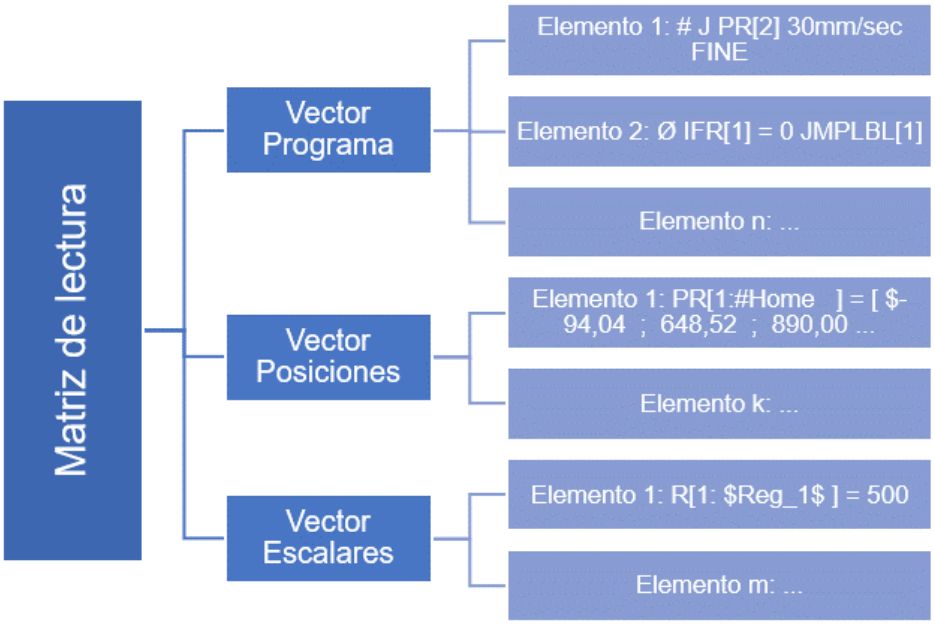


Figura 47: Esquema matriz de lectura

4. Se llama a la función *Detectar_Labels()* que recorre el programa al completo para detectar las etiquetas de salto que haya y las almacena en un vector de etiquetas.
5. Inicia el contador de programa a 0 y comprueba que el programa no ha terminado.
6. Carga la primera línea del programa a partir del vector de programa en un vector auxiliar y se segmenta por espacios (vect_línea).
7. Se comprueba el primer elemento de este vector auxiliar que corresponde con el símbolo identificador de la instrucción y se redirige a la función auxiliar que se encargará de ejecutar la instrucción.
8. Tras finalizar la instrucción, se comprueba si se ha terminado el programa. Si es así se finaliza la ejecución. Si no, se incrementa el contador de programa y se carga en un vector la siguiente línea de programa volviendo al paso 7.

En caso de ser una instrucción de salto condicional el modo de actuar es un poco diferente en el paso 7. Se llama a la *Instrucción_Condicional()* que determina si se cumplen las condiciones para realizar el salto a la etiqueta y si es así se va al caso de salto que llama a la función *Jump_Label().* Esta por su parte, obtiene el número de línea de programa en el que está la etiqueta de salto y modifica el contador de programa para que continúe por la línea de programa en la que está la etiqueta de salto

7.4.3.2 Funciones Detectar_Labels() y Leer_Programa()

Las dos primeras funciones que son llamadas en el método *Run_Program()* son *Leer_Programa()* y *Detectar_labels().* Su método de operación es el siguiente:

- *Leer_Programa()* (Figura 49): Se encarga de construir una matriz con 3 vectores: el vector de programa, el vector de escalares y el vector de posición tal y como se ve en la Figura 47. Estos vectores son resultado de leer las tablas de registros escalares y de posición de la pestaña Data así como de la lectura del programa codificado (oculto al usuario).
- *Detectar_labels()* (Figura 50): Recorre el programa línea a línea para detectar las etiquetas LBL[] que contiene el programa. Para ello aumenta el contador de programa en cada ciclo y comprueba que esa línea de programa tenga el identificador "!". Si es así, se agrega un nuevo elemento al vector de etiquetas labels con la estructura de la Figura 48.

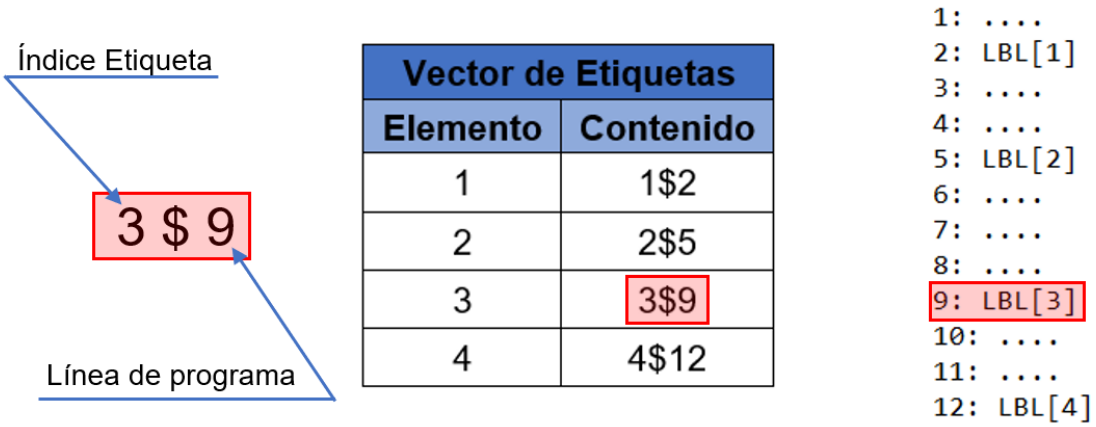


Figura 48: Vector de etiquetas

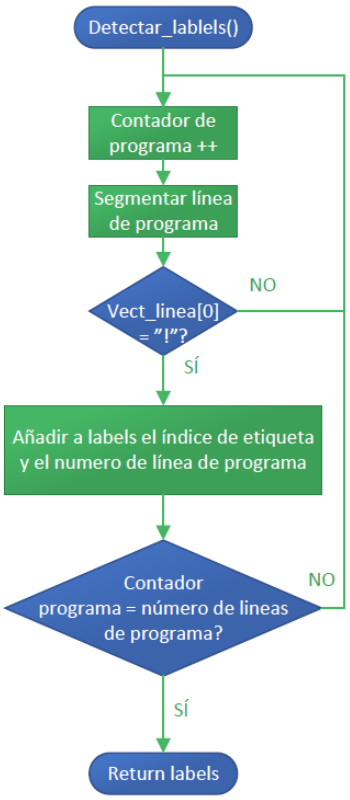


Figura 50: Algoritmo Detectar_Labels()

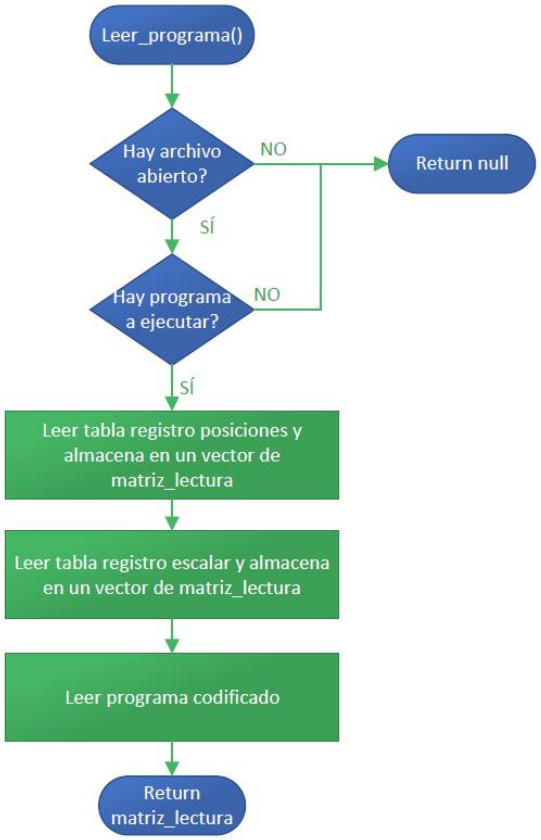


Figura 49: Algoritmo Leer_Programa()

7.4.3.3 Funciones de Movimiento

Se distinguen tres tipos de movimientos (con una función para cada uno) dependiendo del offset que se aplica. Se entiende como offset a un desplazamiento que se aplica a una posición u orientación. Las funciones de movimiento son las siguientes:

- Movimiento sin offset: Este tipo de movimiento es el que no aplica ningún tipo de desplazamiento o rotación respecto a la posición objetivo.
- Movimiento con offset: Movimiento en el que se aplica un desplazamiento o rotación respecto a la posición objetivo y tomando como referencia en el desplazamiento el sistema de coordenadas de la base de la estación.
- Movimiento con tooloffset: El movimiento con tooloffset es similar al movimiento con offset solo que el desplazamiento o rotación se realiza tomando como referencia el sistema de coordenadas de la herramienta.

En las figuras Figura 52, Figura 53 y Figura 54 se muestra el algoritmo aplicado para realizar los distintos tipos de movimiento:

1. Analiza si hay un direccionamiento indirecto (DI)

2. Si hay un direccionamiento indirecto se obtiene el índice del registro escalar R[x], luego se busca este registro dentro del vector de escalares a partir de su índice y se obtiene el índice del registro de posición PR[x] al que apunta. Finalmente se obtiene las coordenadas del registro de posición buscando dentro del vector de posición. En la Figura 51 se describe este proceso de forma más detallada.

   Si no hubiese direccionamiento indirecto se buscaría el registro de posición directamente dentro del vector de posición.

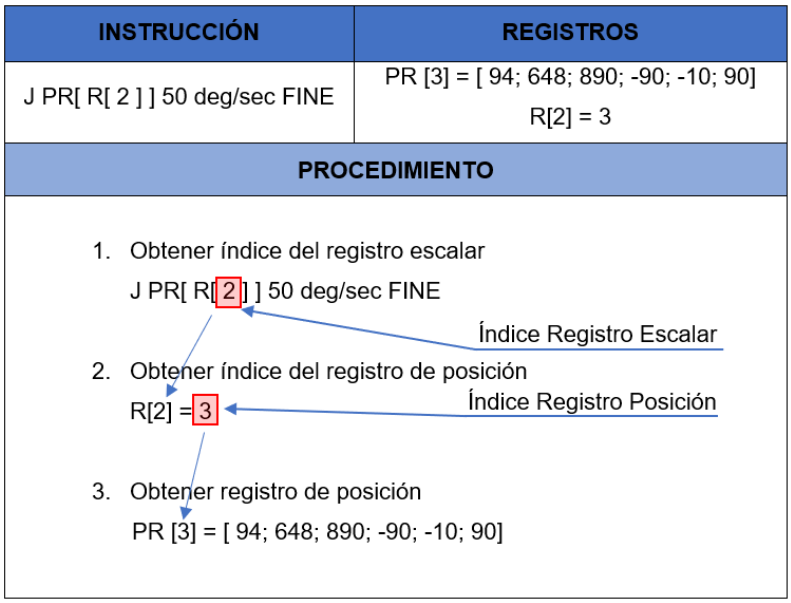


Figura 51: Proceso de obtención de registros

3. Se compone la matriz de transformación homogénea a partir de las coordenadas y los ángulos de Euler obtenidos del registro de posición del paso anterior, haciendo uso de la función de la clase Mat, *FromTxzRxyz().*

   A. En el caso de haber un movimiento con offset, además del registro de posición del movimiento, se obtendrían el índice del registro de posición del offset y el registro de posición del offset como tal. Luego se obtienen las coordenadas del offset y del registro de posición del movimiento y se suman entre sí para obtener un vector de coordenadas único que defina la posición junto con el offset. A partir de este vector se construye la matriz de transformación homogénea.

B. Si el movimiento se realiza con tooloffset, se obtiene el registro de posición de movimiento y el registro de posición del tooloffset, se consiguen los vectores de coordenadas de ambos registros y se componen una matriz de transformación homogénea para cada vector de coordenadas. Finalmente se calcula la posición final mediante la multiplicación de las matrices de transformación obtenidas en el paso anterior.

4. Una vez obtenida la matriz de transformación del movimiento, se comprueba de qué modo se va a realizar el movimiento; para ello se comprueba el segundo elemento del vector de línea (vect_linea).

5. Se obtiene el dato de la velocidad con la que se va a ejecutar el movimiento y se realiza finalmente el movimiento del robot de manera Lineal o Joint en función de lo decidido en el paso anterior.

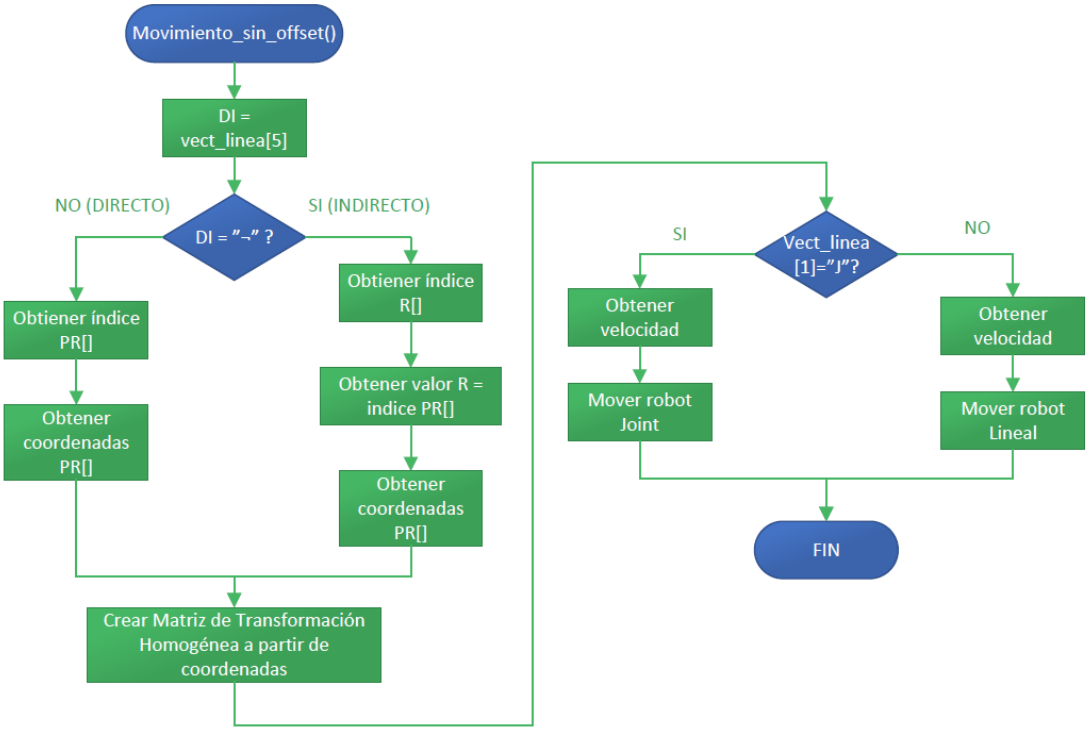


Figura 52: Función Movimiento_sin_offset()

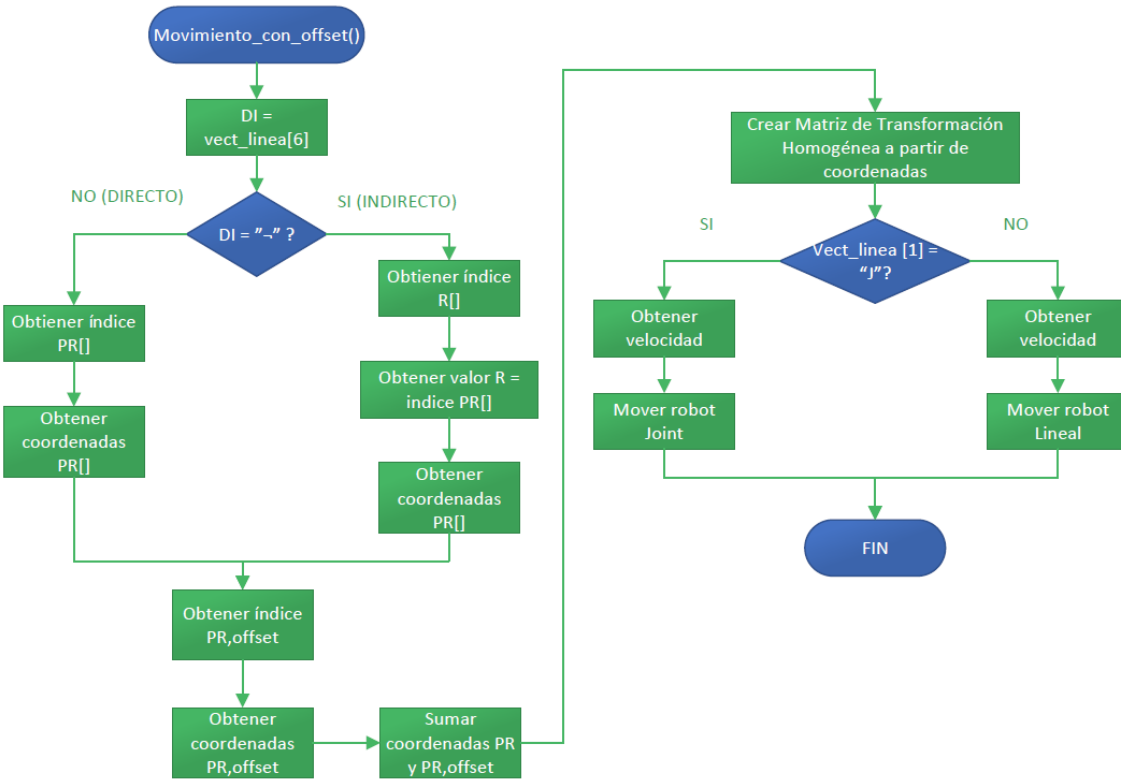


Figura 53: Función Movimiento_con_offset()

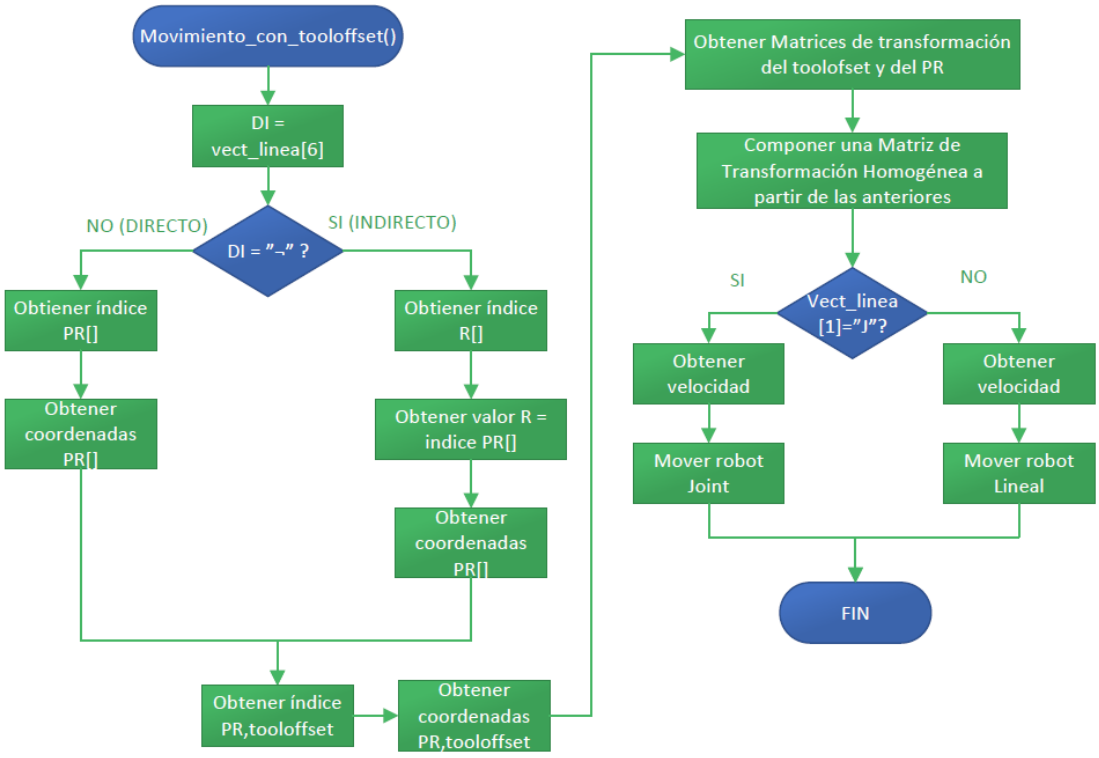


Figura 54: Función Movimiento_con_tooloffset()

7.4.3.4 Funciones de modificación de registros

Estas dos funciones se encargan de realizar modificaciones en los registros durante la ejecución del programa. Los registros afectados pueden ser tanto los registros escalares como los de posición.

- Los registros escalares (Figura 56) pueden modificar su valor utilizando las cuatro operaciones matemáticas básicas (suma, resta, multiplicación y división) en conjunto con otro registro o un valor constante.

- Los registros de posición (Figura 55) modifican el valor de una de sus coordenadas utilizando también una de las cuatro operaciones básicas en conjunto con otro registro escalar o un valor constante.

El procedimiento a seguir para la modificación del registro es el siguiente:

1. Se obtiene el índice de los registros implicados tal y como se ha descrito en la Figura 51.

2. Se analiza si el valor de modificación es un registro o es una constante. Si es un registro, se obtiene el valor del registro.

3. Se comprueba cual es la operación de modificación y finalmente se realiza la actualización del registro. En este paso se verifica que los resultados no sean negativos (en el caso de modificación de un registro escalar) y alertar y detener el proceso si sucede una división por cero.

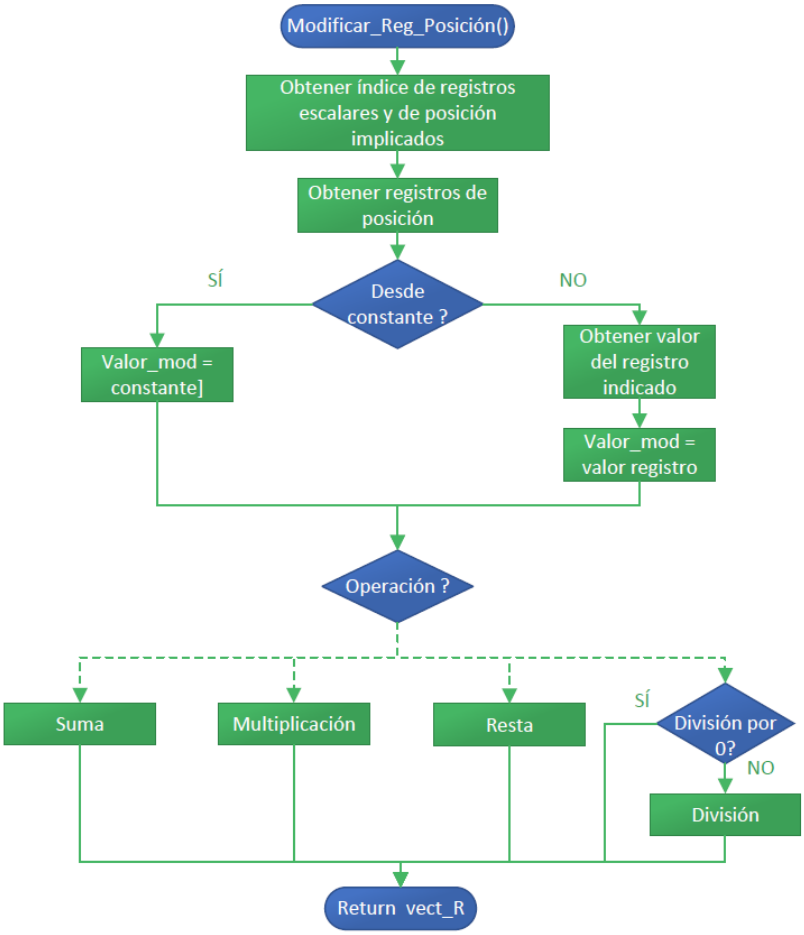


Figura 55: Función Modificar_Reg_Posición()

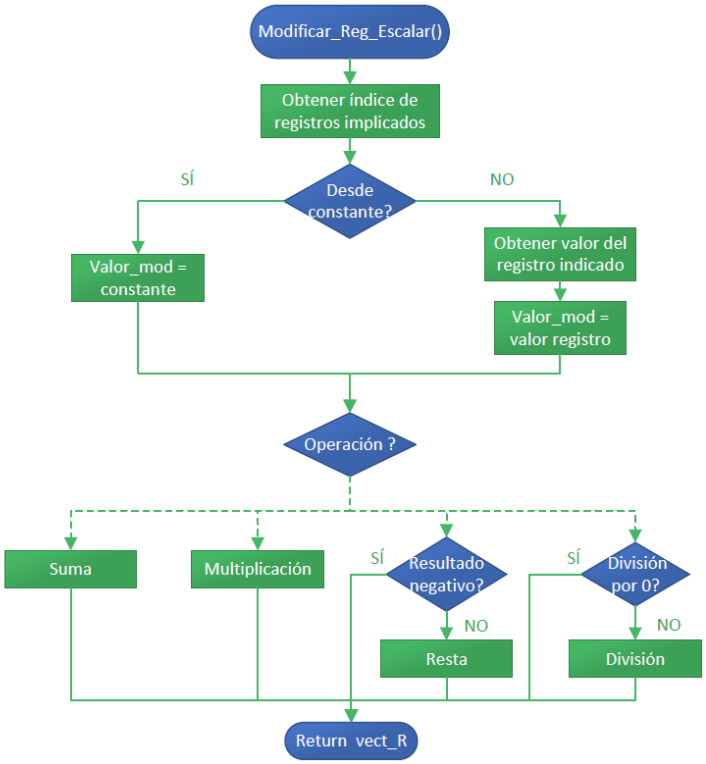


Figura 56: Función Modificar_Reg_Escalar()

7.4.3.5 Funciones de salto condicional

Las dos funciones que componen este apartado son la *Instrucción_Condicional()* y la función *Jump_Label()*. La instrucción *Jump_Label()* (Figura 59) se utiliza para realizar un salto a una etiqueta (LBL[X]) del programa sin tener en cuenta ninguna condición. El comando *Instrucción_Condicional()*, realiza un salto a una etiqueta en función de si se cumple o no una condición determinada.

Jump_Label() obtiene el índice de la etiqueta a la que va a saltar, luego busca la etiqueta a partir de su índice en el vector de etiquetas y consigue de ella el número de línea de programa que es el valor que contiene la etiqueta asociada. Finalmente actualiza el contador del programa al número de línea de programa obtenido en el paso anterior y devuelve este contador a la función general *Run_Program*.

Por otra parte, el algoritmo de *Instrucción_Condicional* (Figura 58) es el siguiente:

1. Se obtiene el índice de los registros implicados y el valor del primer registro.
2. Se analiza si el valor a comparar es un registro o es una constante. Si es un registro, se obtiene el valor del registro.
3. Se comprueba cuál es la operación de comparación (igual que, mayor que, diferente que, menor que …) y finalmente se verifica si se cumple la condición (Figura 57). Si la condición se cumple se devuelve la variable booleana condición_salto como true a la función principal de *Run_program*, la cual volverá a comprobar el estado de esta variable y llamará a la función *Jump_Label* (Figura 59). En caso de que no se cumpla la condición de salto, se pasará a la siguiente línea de programa.

INSTRUCCIÓN	REGISTROS
IF R [1] < R [2] JMP LBL [3]	R [1] = 3 R [2] = 8
PROCEDIMIENTO	
IF R [1] < R [2] JMP LBL [3] Registro 1 (R1) Valor a comparar Como 3 < 8 , cumple condición de salto a etiqueta 3.	

Figura 57: Proceso de obtención de registros y comparación

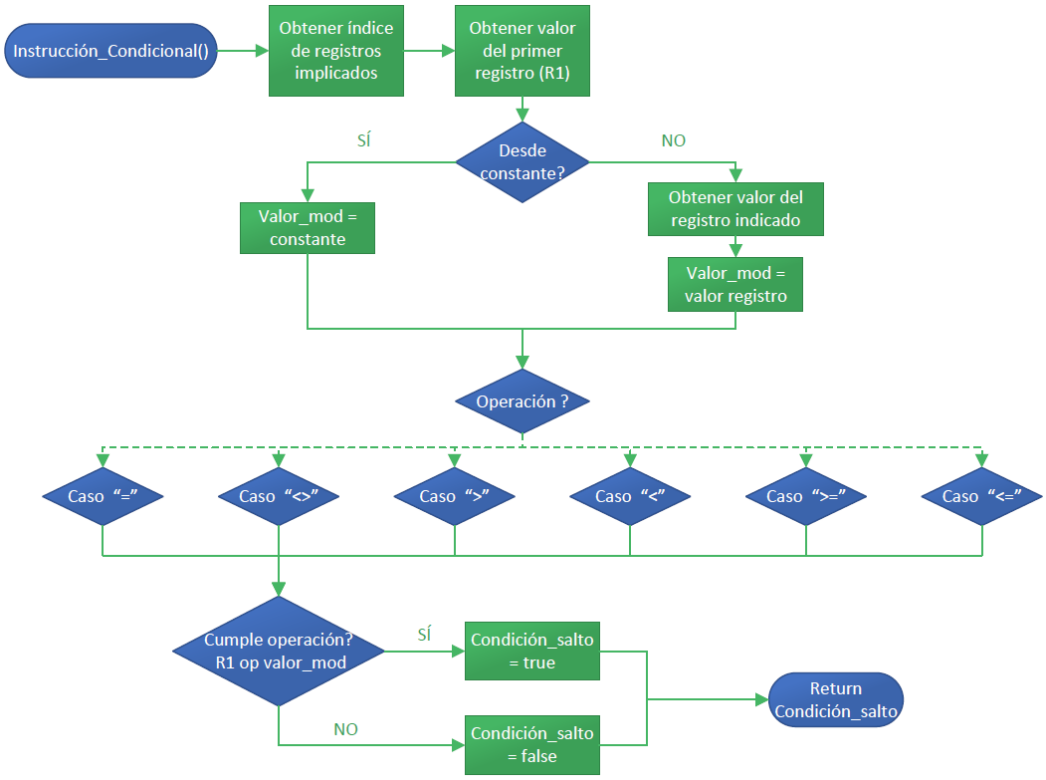


Figura 58: Función Instrucción_Condicional

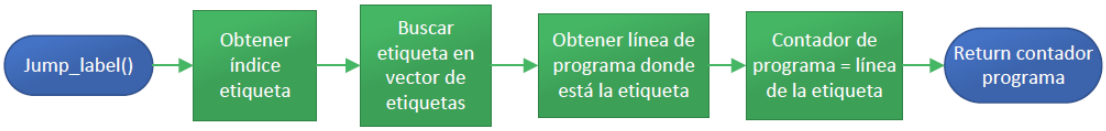


Figura 59: Función Jump_Label

7.4.3.6 Función Run_Conveyor()

Esta función se encarga de poner en marcha y detener la cinta transportadora. Cuando un objeto está cerca en la posición de inicio *Put Conveyor* y se ejecuta una instrucción de Mover Cinta, el objeto pasa a formar parte de la cinta y se mueve con ella hasta la posición final de recogida de piezas *Get Conveyor*¸ lugar donde tanto el objeto como la cinta se detienen.

El algoritmo es el siguiente (Figura 61). Primero se rastrean los elementos hijos de la estación para formar una lista con objetos que son manipulables (cilindros y cubos). Después de entre esa lista se descartan aquellos que se encuentran lejos de la posición de inicio de la cinta *Put Conveyor*. Con los objetos restantes, se llama a dos programas de Python (Figura 60). Estos programas provienen de RoboDK y han sido modificados expresamente para que sean capaces de manejar tanto los cilindros como los cubos:

- *RunConveyor*: se encarga de enlazar los objetos manipulables cercanos a la cinta y de comenzar el movimiento de la misma.
- *WaitForPart*: cuando los objetos anclados a la cinta llegan al punto final de la misma *Get Conveyor*, entra en juego esta función, deteniendo el movimiento de la cinta y desligando los objetos de esta. Este programa funciona como un sensor de final de cinta. Cuando el objeto llega al final de la cinta, atraviesa un plano, el cuál es perpendicular a la cinta y pasa por el punto *Get Conveyor*. Este plano actúa como sensor y hace que se detenga la función *WaitForPart*, lo que detiene también la función *RunConveyor*.

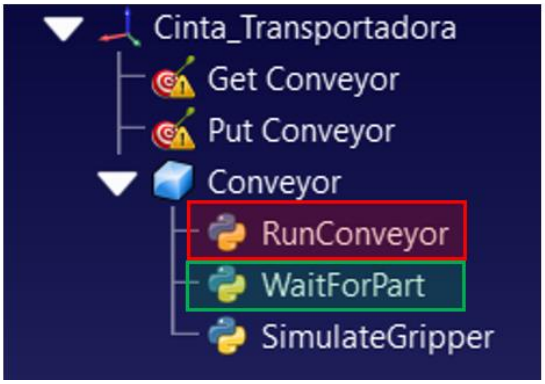


Figura 60: Programas Python para conveyor

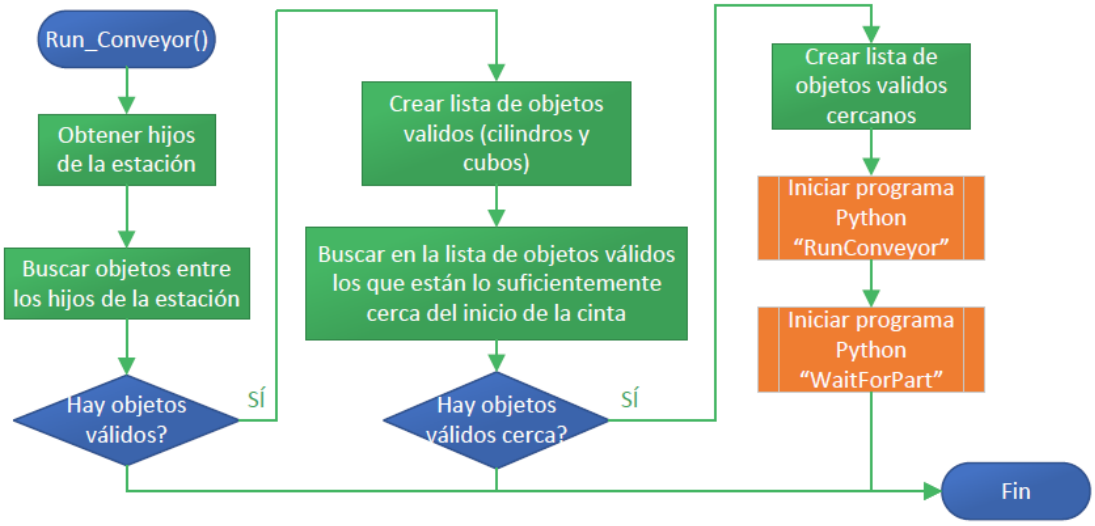


Figura 61: Función Run_Conveyor

8. MANUAL DE USUARIO

Este capítulo tiene la finalidad de servir como guía de usuario, mostrando paso a paso cómo utilizar las diferentes funcionalidades de la aplicación. Por otra parte, también se ha dedicado la segunda mitad del capítulo a desarrollar un índice de posibles errores que pueden acontecer durante el uso de la aplicación y dar solución a estos.

El apartado 7.4 Exposición del funcionamiento de los métodos principales del código y su nexo con funciones propias de ROBODKse centró en desarrollar la metodología y el funcionamiento interno de la herramienta, exponiendo los métodos principales de esta a través de flujogramas explicativos. Por contraparte, durante este apartado se abarcará el cómo usar la aplicación sin necesidad de comprender el funcionamiento interno de cada utilidad. El propósito de esta guía es hacer más sencilla la interacción con la aplicación sabiendo en todo momento cómo realizar cada tarea, mejorando así la experiencia del usuario.

8.1 Modo de uso

Esta guía de modo de uso se describirá las acciones principales para la correcta utilización de la aplicación desarrollada. Se desglosará en cuatro apartados que abarcan las cuatro operaciones fundamentales que se necesitan para trabajar con la herramienta:

- Cómo instalar e iniciar la aplicación.
- Cómo añadir elementos a la estación.
- Cómo crear y guardar un programa.
- Cómo abrir y ejecutar un programa.

8.1.1 Cómo instalar e iniciar la aplicación

Para poder utilizar la aplicación desarrollada en este proyecto es necesario la instalación de dos componentes; por una parte, el programa RoboDK que será el que proporcionará la interfaz de visualizado y por otra parte la herramienta que se ha desarrollado mediante Visual Studio a partir de la API proporcionada desde la web de RoboDK.

El programa RoboDK se ha descargado e instalado directamente desde la web de RoboDk https://robodk.com/es/download , tal y como se expuso en el apartado 6.2 Descarga del programa.

Para la instalación de la herramienta se hará uso de un instalador, que se proporcionará al usuario (Figura 62). Una vez iniciado el asistente de instalación, se pedirá al usuario una ruta donde alojar el programa (Figura 63).

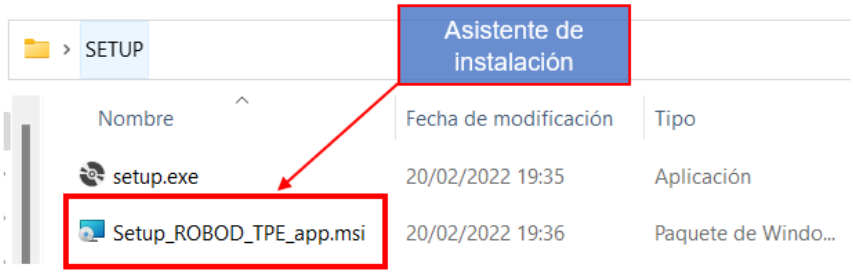


Figura 62: Asistente de instalación

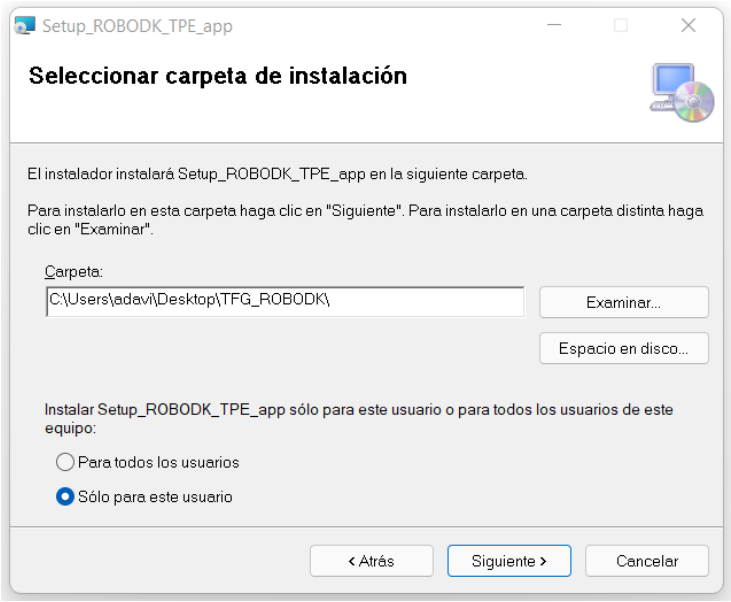


Figura 63: Ventana selección ruta de asistente de instalación

Cuando se ha configurado la instalación, la cual consistirá en elegir la carpeta donde se instalará el programa, se obtendrán los siguientes elementos en la carpeta objetivo; uno de ellos será el archivo ejecutable .exe y otro la librería que hace uso el programa para obtener los modelos 3D de objetos y robots, así como macros programadas en Python. Además, se creará un acceso directo en el escritorio para iniciar el programa (Figura 64).

Figura 64: Carpeta de instalación

Una vez completada la instalación de ambos programas se puede proceder a ejecutar la aplicación desarrollada.

Nada más iniciar la aplicación el usuario se encontrará ante la siguiente ventana que le requerirá seleccionar la carpeta librería (Figura 66) que se ha mencionado anteriormente, la cual contiene los distintos modelados 3D. Se puede hacer una selección manual o bien cancelar esta selección, en cuyo caso se escogerá una ruta por defecto en la carpeta de instalación del programa (Figura 65).

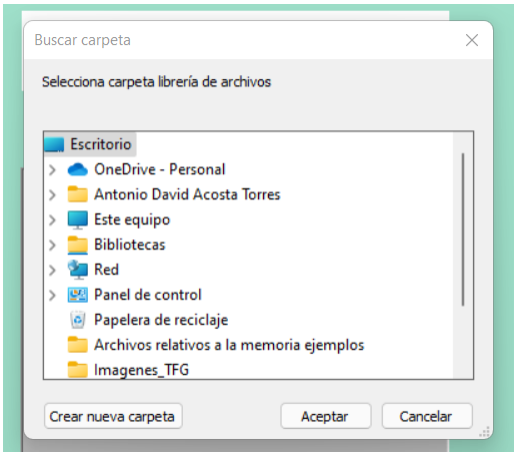


Figura 66: Ventana de selección librería

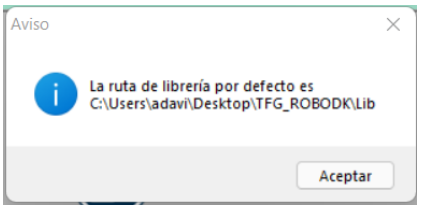


Figura 65: Ruta librería por defecto

Otra posibilidad es realizar esta selección de la librería en cualquier momento haciendo uso del botón “Buscar Carpeta” en la pestaña General, 

.

Realizados todos estos pasos, la instalación e inicialización se da por completa y la aplicación está lista para ser utilizada.

8.1.2 Cómo añadir elementos a la estación

Para añadir elementos a la estación se hará uso de la pestaña de diseño (Figura 67), la cual está dividida en varias secciones:

- Sección general. En ella se puede añadir un robot, cambiarlo por otro, eliminar la estación completa o añadir una herramienta al robot. Una limitación que se ha impuesto es reducir el número de robots por estación a uno solo, ya que el diseño de este programa no contempla el trabajo simultáneo entre robots dejándolo como propuesta a futuro. También conviene aclarar que al añadir herramienta se agregará una pinza de pick and place al robot, dejando del mismo modo como propuesta futura la posibilidad de utilizar herramientas varias.

- Sección conveyor. Se gestiona la cinta transportadora, añadir, eliminar y guardar posición. El programa está limitado a la utilización de una sola cinta transportadora por lo que al intentar añadir una segunda se obtendrá un mensaje de aviso.

- Sección agregar objeto/mesa. Desde aquí se añaden los distintos objetos y se configurarán en tamaño y color. También se deja a esta sección la eliminación y guardado de posición de objetos.

- Sección registro de posiciones. Es el lugar donde se registran las posiciones y características que definen a los objetos añadidos a la estación.

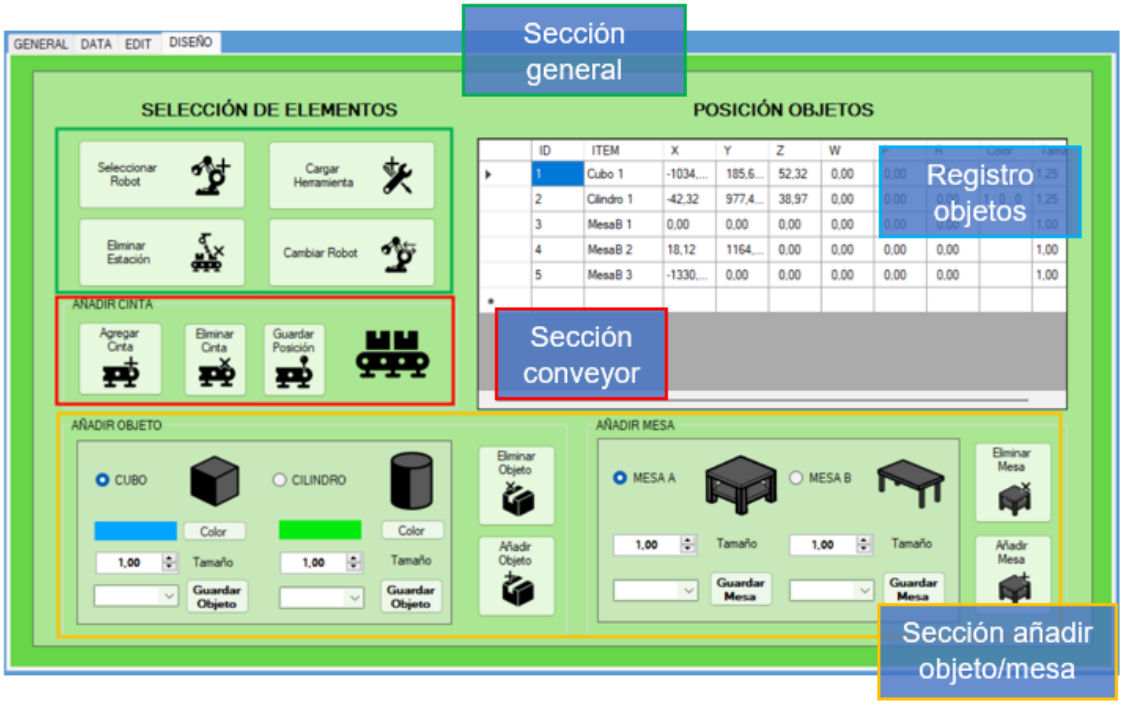


Figura 67: Descripción pestaña Diseño

8.1.2.1 Añadir un ítem

Para añadir una cinta a la estación, se utiliza el botón Agregar Cinta y la cinta aparecerá en la interfaz de visualizado en la posición de referencia. Para eliminarla basta con usar Eliminar Cinta.

Ahora bien, para añadir objetos se sigue el siguiente procedimiento (Figura 68):

1. Se selecciona el tipo de ítem presionando uno de los botones circulares al lado de cada objeto. Se elige entre uno de los objetos o una de las mesas.
2. Se añade el ítem desde el botón Añadir Objeto o Añadir Mesa, dependiendo del ítem que se desee añadir.

Figura 68: Procedimiento añadir ítem

8.1.2.2 Mover un ítem

Una vez añadido el ítem, este aparecerá representado en la interfaz de visualizado en el origen de coordenadas de la estación. Para moverlo de posición se hará clic sobre él y se desplegará un menú lateral en el que poder modificar tanto las coordenadas como los ángulos de giro del objeto para definir su posición y orientación (Figura 69).

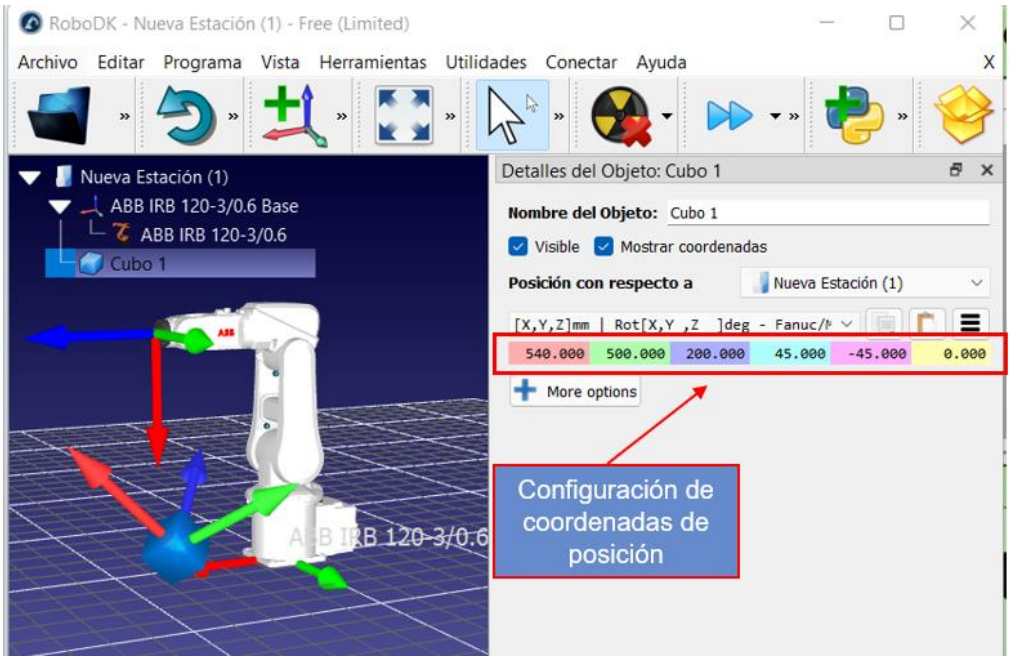


Figura 69: Movimiento de ítems (1)

Otro modo de configurar la posición del objeto de forma más rápida, aunque menos precisa, es mediante la utilidad Mover Sistemas de Coordenadas. Una vez activada esta opción el usuario puede variar la posición del ítem clicando y arrastrando desde uno de

los ejes de coordenadas del objeto o bien haciendo lo mismo en uno de los planos de traslación o flechas de rotación (Figura 70).

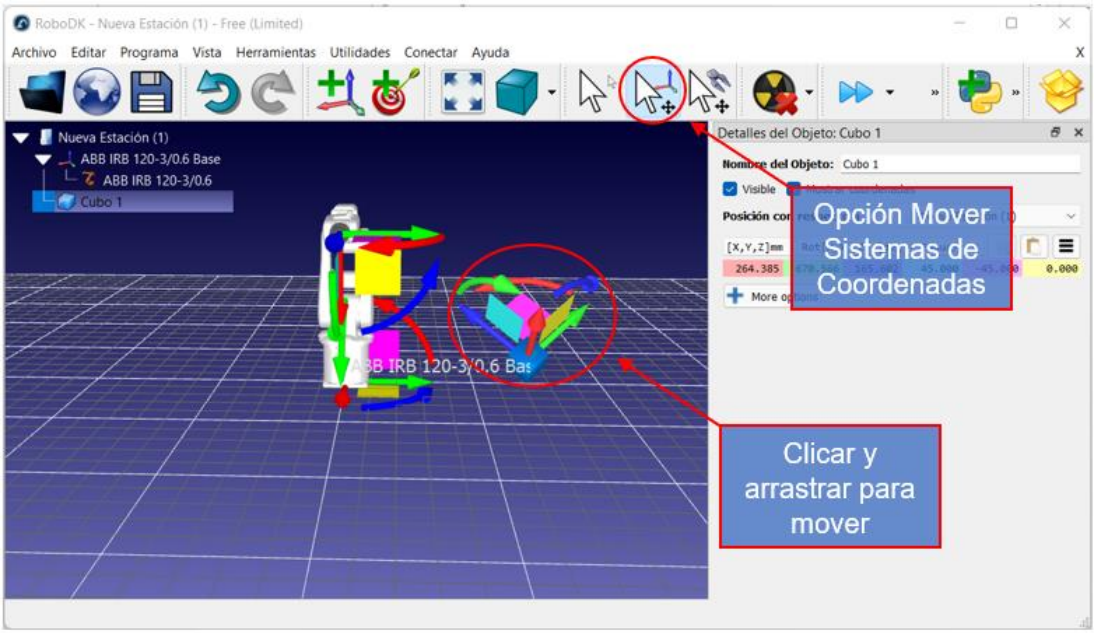


Figura 70: Movimiento de ítems (2)

Como excepción, la cinta transportadora sólo se podrá mover usando su sistema de referencia y no desde el objeto (Figura 71); es decir, se manipulará su posición desde el sistema de referencia y no desde el objeto asociado ya que de esta forma se mantienen la relación entre los sistemas de referencia Put Conveyor y Get Conveyor.

Figura 71: Movimiento de cinta transportadora

8.1.2.3 Configurar y guardar un ítem

La aplicación permite cambiar el color y el tamaño de los objetos y mesas además de modificar su posición (Figura 72). Para ello es necesario primero seleccionar el objeto que se desea modificar en color y/o escala haciendo uso del desplegable. Luego configurar el tono de color y la escala a aplicar. Finalmente guardar el ítem, tras lo cual quedará registrado en el panel de registro de posición tanto la configuración de color y tamaño como el vector de posición del ítem.

Es muy importante realizar el guardado de todos los ítems añadidos a la estación ya que si no se realiza este paso de guardado, al volver a cargar la estación tras finalizar el programa el ítem no aparecerá representado en esta.

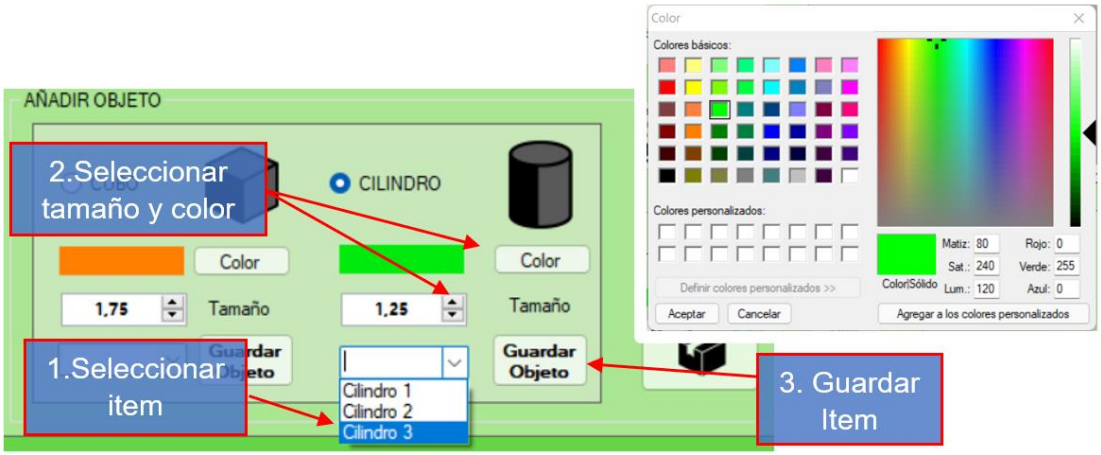


Figura 72: Modificar ítem

8.1.2.4 Eliminar un ítem

Para eliminar un ítem primero se selecciona el ítem en el desplegable y luego se presiona el botón de eliminar mesa/objeto. Una vez hecho esto el ítem desaparecerá tanto de la interfaz de visualizado como de la tabla de registro de posiciones. Se puede operar de manera idéntica para las cintas transportadoras.

8.1.3 Cómo crear y guardar un programa

La creación del programa, es decir, la programación de la rutina que ha de llevar a cabo el robot, se realizará desde la pestaña Edit aunque antes serán necesarias tener registradas las posiciones que el robot debe alcanzar. Se recomienda también haber añadido todos los elementos necesarios a la estación antes de comenzar a programar.

8.1.3.1 Guardar registros

El registro de posiciones y de escalares se llevará a cabo desde la pestaña Data la cual está dividida en dos subsecciones, una para los registros escalares y otra para los registros de posición. En cada sección encontramos utilidades similares de guardado, modificación y eliminación de registros con la diferencia que la sección de registros de posición permite almacenar posiciones de offset.

Para añadir un registro de posición se seguirá el siguiente procedimiento:

1. Se mueve el robot a la posición deseada, bien sea desde los controles generales o bien desde el botón Mover Sistemas de Coordenadas y se operará de manera idéntica al movimiento de objetos.
2. Se añade un nombre o etiqueta al registro de posición.
3. Mediante el botón Almacenar posición, se agrega la posición a la tabla registro de posiciones.

Para añadir un registro escalar se opera de manera similar, pero en la sección de registros escalares.

También se puede modificar un registro. Para ello, solo hay que seleccionar el registro a modificar y mediante el botón modificar posición se asocia la posición actual del robot al registro seleccionado. Si se desea añadir un offset tan solo se tiene que configurar el vector de posición y agregar una etiqueta para añadir un offset.

8.1.3.2 Crear programa TPE

La creación de un programa TPE se lleva a cabo desde la pestaña Edit. En esta pestaña se encuentran las principales funciones que el robot puede llevar a cabo en lenguaje TPE.

Para introducir una instrucción (Figura 73) en el programa primero se ha de configurar los parámetros de la instrucción. Por ejemplo, si se quiere añadir una instrucción de espera se tendrá que modificar el indicador numérico de la sección Wait. Otro ejemplo es si se quiere añadir una instrucción de modificación de registro, para lo que se deberán cambiar los registros implicados, la operación y el valor modificador. Luego, una vez configurados los parámetros de la instrucción de forma correcta, se pulsará el botón de agregar instrucción para que esta se añada al programa. Por otra parte, si se desea borrar una instrucción se puede mediante el botón Borrar Línea, eliminar la última línea de programa.

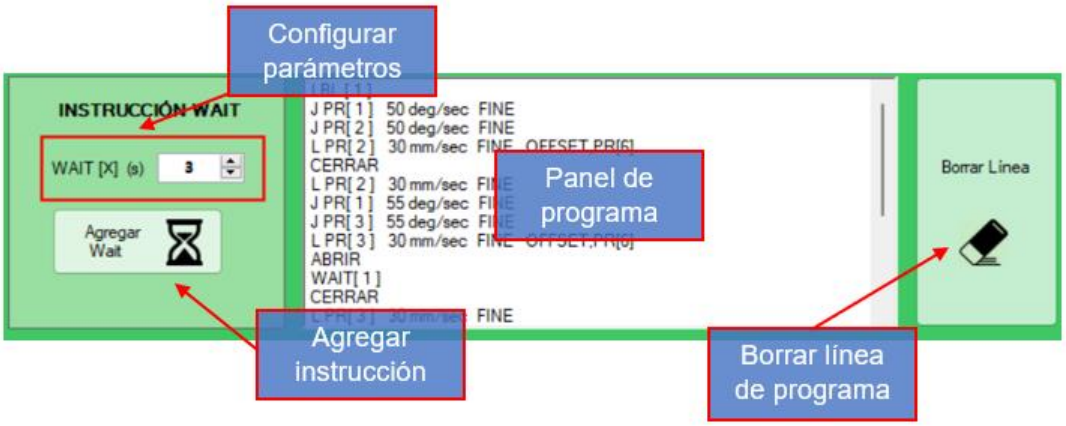


Figura 73: Pasos para añadir instrucción

Se mostrará un ejemplo con una instrucción de movimiento (Figura 74):

INSTRUCCIÓN
J PR [4] 60deg/sec FINE OFFSET,PR [5]
CONFIGURACIÓN

DESARROLLO
1. Se elige el tipo de movimiento, Joint o lineal. 2. Se decide la forma de obtener el registro de posición, bien mediante direccionamiento indirecto (DI) o sin él. Con DI se seleccionará un registro escalar con el control numérico y el valor de ese registro será el índice del registro de posición para el moviemiento. Sin DI se selecciona el índice del registro de posición en el control numérico directamente. 3. Se configura la velocidad de movimiento, en grados por segundo si es Joint o milímetros por segundo si es Lineal. 4. Se conigura si hay o no un offset y se selecciona el registro que contiene la información del offset. 5. De igual modo que el paso anterior se configura si hay un tool offset. A tener en cuenta que si hay un offset no se puede elegir un tool offset a la vez.

Figura 74: Ejemplo instrucción de movimiento

8.1.3.3 Guardar programa completo

Una vez diseñada la estación, almacenando los registros de posición y escalares necesarios y creado el programa que debe ejecutar el robot, es momento de guardar el programa para poder modificarlo o ejecutarlo a posteriori. Para guardar el programa, primero hay que hacer una vista previa en la pestaña General para luego guardar archivo (Figura 75). Aparecerá una ventana para seleccionar la carpeta donde almacenar el proyecto (Figura 76) y añadiendo un nombre se habrá completado el guardado del proyecto. Si se desea sobrescribir un proyecto, hay que buscar la carpeta donde se aloja el anterior y escribir el mismo nombre, pero sin la extensión "_E".

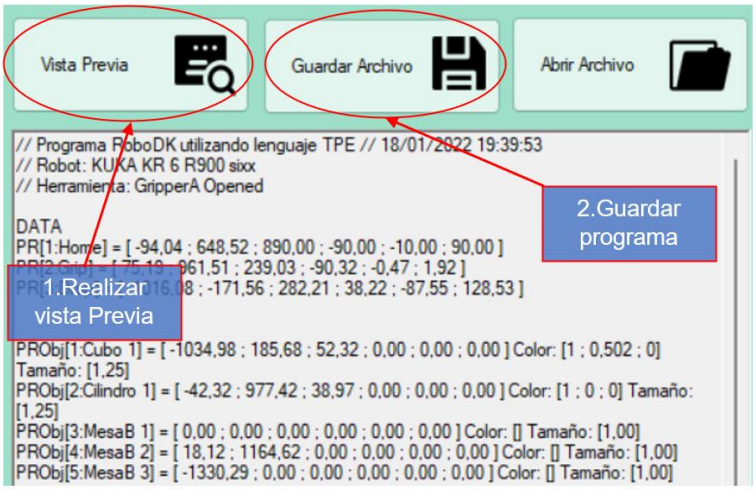


Figura 75: Guardado de programa TPE

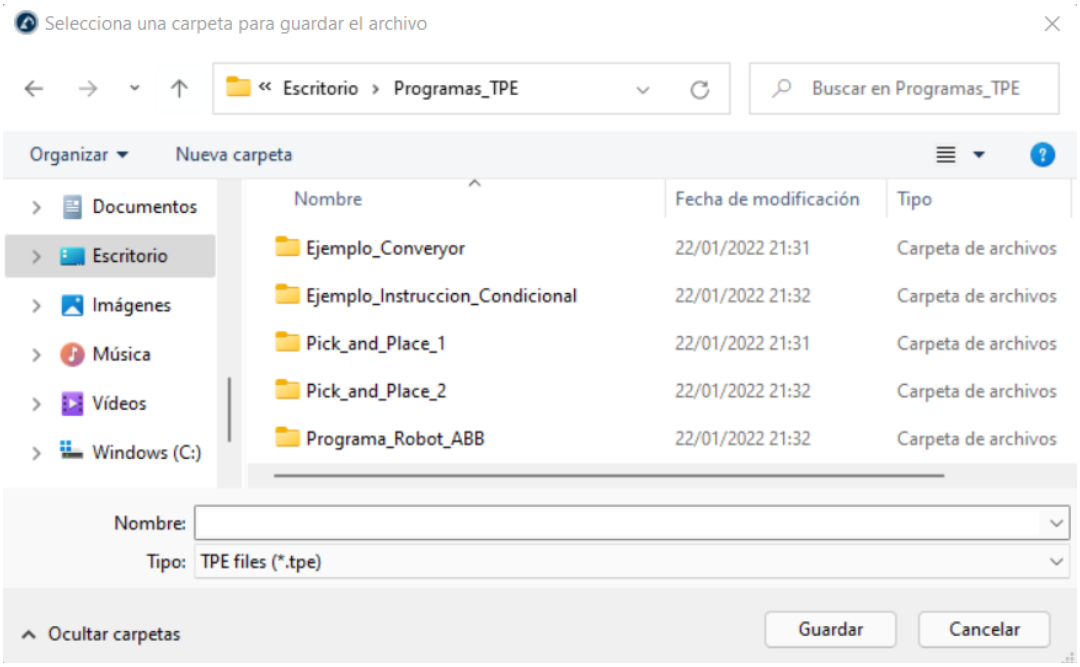


Figura 76: Ventana guardado de proyecto

Como observación, resaltar el hecho de que, si en algún momento se desea cambiar la localización del programa guardado, se deberá mover la carpeta y todo su contenido al completo, ya que de otro modo la aplicación mostrará un mensaje de error ante la ausencia de algunos ficheros como resultado de un copiado erróneo.

8.1.4 Cómo abrir y ejecutar un programa

Para abrir y ejecutar un programa se pulsará el botón de abrir archivo en la pestaña General, se elegirá la ruta donde se almacena el archivo ".tpe" y se clicará en él (FFigura 77). A continuación, se verá una vista general del proyecto en la ventana de vista previa. Esta vista contiene el programa y los tres tipos de registros. Luego, el programa se mostrará en la pestaña Edit y los registros se cargarán en las tablas de registros de las pestañas Data y Diseño. Finalmente, se cargará la estación en la interfaz de visualizado.

Por otro lado, para iniciar la ejecución del programa se pulsará el botón EJECUTAR PROGRAMA en la pestaña General, lo que dará comienzo a la rutina de ejecución anteriormente vista.

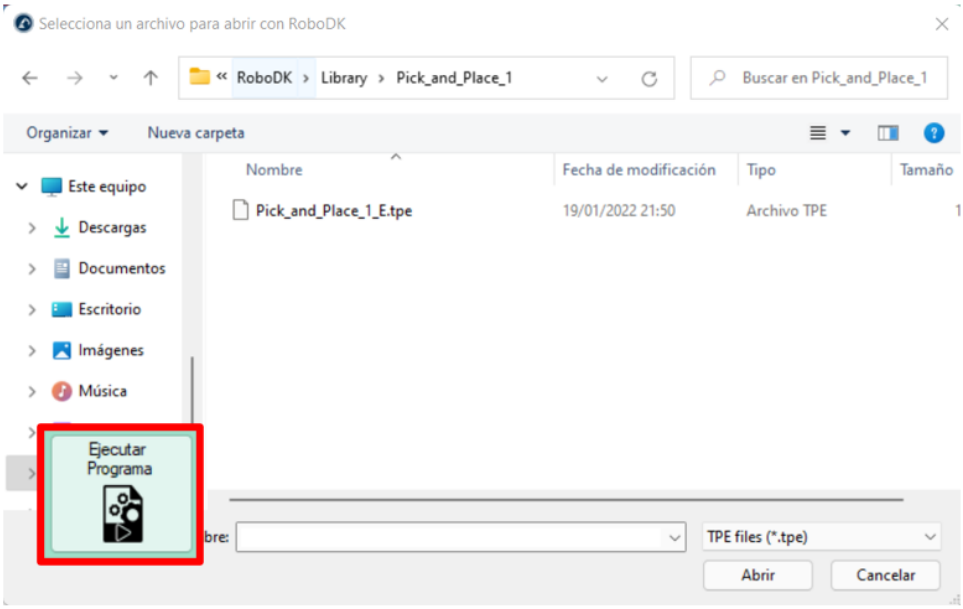


Figura 77: Ventana seleccionar archivo para abrir y botón de ejecución de programa

8.2 Gestor de Errores y Avisos. Guía de solución de errores

Para el manejo y gestión de errores y avisos se ha desarrollado un método llamado "Gestor_Errores()" en el cual se encuentran todos los mensajes de alerta y error que

son mostrados cuando ocurre una falla importante durante el uso de la aplicación (Figura 78). Para no saturar al usuario con la aparición reiterada de ventanas de error, se han delegado a la Barra de notificaciones algunos mensajes de error y alerta, los cuales son categorizados como de menor relevancia. De este modo se pretende tener una mejor experiencia por parte del usuario al utilizar la aplicación. Un ejemplo de estos errores menores sería por la obtención de registros negativos en las funciones de modificación de registro.

A continuación, se puede ver detalladamente un listado con los errores y avisos manejados por el método “Gestor_Errores()” así como una breve explicación de cómo solventarlos:

Figura 78: Error ruta de librería de carga vacía

1: Error librería incorrecta

```
"No se encuentra ruta para la carga de archivo. Por favor, busque la librería en
                        la ventana 'General'."
```

- Seleccione la ruta de la librería en la pestaña general.

2: Error ya hay un robot cargado.

```
"Ya hay un robot cargado."
```

- Debido a limitaciones del programa sólo se puede tener un robot por estación. Si se desea se puede cambiar el robot por otro en la pestaña de diseño.

3: Error se ha cargado un archivo que no tiene extensión .robot.

```
"El archivo seleccionado no tiene el formato robot."
```

- Se ha seleccionado otro tipo de archivo que no contiene un modelado de un robot. Seleccione un archivo con extensión .robot.

4: Error no hay robot cargado.

```
"No se ha cargado ningún robot. Por favor, cargar un robot primero."
```

- Para evitar errores, no se puede cargar ningún elemento antes de disponer de un robot. Cargar primero un robot y luego el elemento deseado.

5: Error ya hay una herramienta cargada.

"No se puede cargar una herramienta cuando y hay otra cargada."

- Solo se puede seleccionar una herramienta por robot

6: Error no se encuentra el archivo .tool.

"No se encuentra el archivo .tool en la ubicación descrita como librería."

- La librería seleccionada no es correcta. Volver a seleccionar librería desde la pestaña general.

7: Error, el robot no puede alcanzar la posición descrita.

"La posición no es alcanzable por el robot."

- La posición seleccionada no es alcanzable debido a la configuración del robot. Mover la ubicación objetivo a una zona alcanzable por el robot.

8: Error al introducir un valor de posición.

"La posición introducida no tiene el formato deseado ([X;Y;Z;W;P;R]) o contiene un carácter no numérico."

- Se ha introducido un valor de posición incorrecto. Comprobar que se escriben exclusivamente valores numéricos.

9: Error al introducir un valor de offset no valido.

"Por favor, introduzca un valor de offset, válido."

- El valor de offset está fuera de límites. Introducir un valor de offset de entre -9999 y 9999.

10: Aviso guardado con éxito.

"Archivo guardado con éxito."

11: Error de guardado.

"Error en guardado."

- Error en guardado. Se recomienda revisar la ruta de guardado, así como que los registros estén correctos.

12: Error no hay archivo a guardar. No hay vista previa.

"No hay archivo a guardar. Por favor hacer vista previa."

- Se debe hacer vista previa antes de realizar el guardado del archivo.

13: Error no se puede encontrar el archivo _PR.

"No se puede encontrar el archivo _PR."

- Comprobar que el archivo no se ha borrado en la carpeta de guardado.

14: Error no se puede encontrar el archivo _R.

"No se puede encontrar el archivo _R."

- Comprobar que el archivo no se ha borrado en la carpeta de guardado.

15: Error no se puede encontrar el archivo _PGM.

"No se puede encontrar el archivo _PGM."

- Comprobar que el archivo no se ha borrado en la carpeta de guardado.

16: Error no se puede encontrar el archivo _OBJ.

"No se puede encontrar el archivo _OBJ."

- Comprobar que el archivo no se ha borrado en la carpeta de guardado.

17: Error librería de objetos incorrecta.

"La librería no es correcta. Por favor, cargar una librería válida."

- Seleccione la ruta de la librería correcta (Lib) en la pestaña general.

18: Aviso apertura correcta de archivo.

"Apertura correcta de archivo."

19: Error fallo de ejecución de programa. Fallo al interpretar.

"Fallo al interpretar."

- Error con múltiples causas, en el próximo párrafo se exponen las principales.

20: Aviso ejecución correcta del programa.

"El programa se ha ejecutado correctamente."

21: Error ejecución de programa, no hay archivo abierto.

"No hay se ha abierto ningún archivo. Por favor, abrir primero un archivo en 'Abrir Archivo'. "

- No se puede ejecutar el programa ya que no está abierto. Abrir el archivo de programa y luego ejecutar.

22: Error al coger objeto. El objeto está muy lejos de la herramienta.

"No hay objeto cerca de la herramienta. Por favor, posicione el robot más cerca del objeto a coger"

- La herramienta no está lo suficientemente cerca como para interactuar con el objeto. Acercar la herramienta a una posición más próxima al objeto

23: Error no hay herramienta seleccionada.

"No hay herramienta seleccionada. Por favor, seleccione una herramienta previamente"

- No hay una herramienta asociada al robot, no se puede realizar la acción de coger/soltar un objeto. Se debería de seleccionar una herramienta antes de realizar la acción.

24: Error no hay objeto en la herramienta para soltar.

"No hay objeto seleccionado por la herramienta. Por favor, compruebe que hay un objeto en la herramienta"

- La herramienta no tiene asociado ningún objeto en el árbol de la estación. Realizar primero la acción de coger objeto antes de soltar.

25: Error el objeto no está lo suficientemente cerca de la cinta transportadora.

```
"El objeto no está lo suficientemente cerca del punto 'Put Conveyor' Por favor,
                    acerque el objeto al punto indicado. "
```

- La herramienta no está lo suficientemente cerca de la posición de dejada de la cinta transportadora. Acercar la herramienta a una posición más próxima al punto Put Conveyor.

26: Error solo una cinta transportadora por proyecto.

```
"Solo se permite una cinta transportadora por proyecto."
```

- Debido a limitaciones de la aplicación, solo es posible utilizar una cintra transportadora por estación.

27: Error de escalado.

```
"No se puede escalar a ese valor."
```

- Se ha introducido un valor de tamaño de objeto incorrecto o es un valor no numérico.

28: Error de ejecución de programa.

```
"No hay programa para ejecutar."
```

- El programa para ejecutar está vacío.

La función Run_Program(), tiene dos posibles resultados, o bien la ejecución del programa se realiza de manera correcta, en cuyo caso se verá reflejado en un mensaje informativo, o bien ha ocurrido algún error durante la ejecución. Debido a la diversidad de errores posibles y a que algunos de ellos no son posibles de predecir o de saber con antelación a poner el programa en marcha, se ha optado por emitir un único mensaje de error de fallo de ejecución. A continuación, se enumeran las posibles causas que dan lugar a la aparición de este error.

1. La posición no es alcanzable por el robot. Este error es resultado de intentar acceder a una posición mediante un tipo de movimiento el cual el robot no puede ejecutar. Probablemente aparezca antes de realizar un movimiento de tipo lineal. Una solución sería cambiar a un tipo de movimiento “Joint” o establecer una posición cercana que admita el movimiento “Lineal”.

2. El índice del elemento PR[X] apunta a una posición que no existe. Aparece cuando se quiere realizar alguna operación en la que hay un PR[X] implicado el cual no está declarado en el registro de posiciones. Probablemente sea un error simple de programación del usuario. Cambiar el

índice del registro de posición al correcto bastará para solucionar el error.

3. El índice del elemento R[X] apunta a un PR[X] que no existe. Este error es idéntico al descrito en el punto 2 solo que ahora el registro o registros afectados son los escalares (R[X]). Por tanto, se procederá de la misma manera que el punto anterior para solucionarlo, solo que ahora se trabajará con los registros escalares.

4. Se ha realizado un salto condicional o un salto simple a una etiqueta (LBL[X]) que no existe. La aparición de este error es debida al intento de acceder a una etiqueta que no existe en el registro. Este error se soluciona comprobando que los saltos se realicen a etiquetas que ya estén en el código.

9. EJEMPLOS DE APLICACIÓN

Teniendo ahora una visión general de los mecanismos internos de la herramienta, se han diseñado una serie de estaciones en las que se tratará de explorar las principales funcionalidades de la aplicación. Este apartado consta de dos ejemplos diferenciados en los que la finalidad principal es la manipulación de objetos. El primero muestra el funcionamiento de un robot en conjunto con una cinta transportadora y el segundo demuestra la utilidad de modificación de registros escalares y de posición, así como el uso de estructuras cíclicas para el apilamiento de objetos en un pallet.

9.1 Ejemplo A, Cinta Transportadora

9.1.1 Análisis del programa del ejemplo A

La estación de este ejemplo consiste en dos mesas (una en la que se recoge los objetos y otra en la que se dejan) comunicadas por una cinta transportadora. El robot se encuentra en posición central y el objetivo de esta estación es coger dos piezas (un cilindro y un cubo) y llevarlas a la segunda mesa, pasando por la cinta transportadora. En la Figura 79 se ve la estación al principio y al final de la ejecución del programa donde los objetos se han trasladado a la segunda mesa. El programa en TPE se puede ver en la Figura 80.

En este ejemplo se tiene un registro de posición para la cogida de cada objeto y para cada dejada, otro para el posicionamiento sobre la cinta y otro para la recogida, a los que se le suma otro para el “home”. Solo hay un registro escalar que indica el número de piezas por recoger.

El algoritmo de este programa es el siguiente:

1. Primero coge una pieza y la deja sobre la cinta.
2. La cinta realiza su movimiento y el robot recoge la pieza al final del recorrido de la cinta y la posiciona en la segunda mesa.
3. Se realiza una comprobación de si hay más piezas para ser recogidas, para ello evalúa un registro escalar que funciona a modo de contador de piezas. Si quedan más piezas, pasa a realizar la misma operación que antes, pero con la segunda pieza; si no, termina la ejecución del programa (Figura 81).

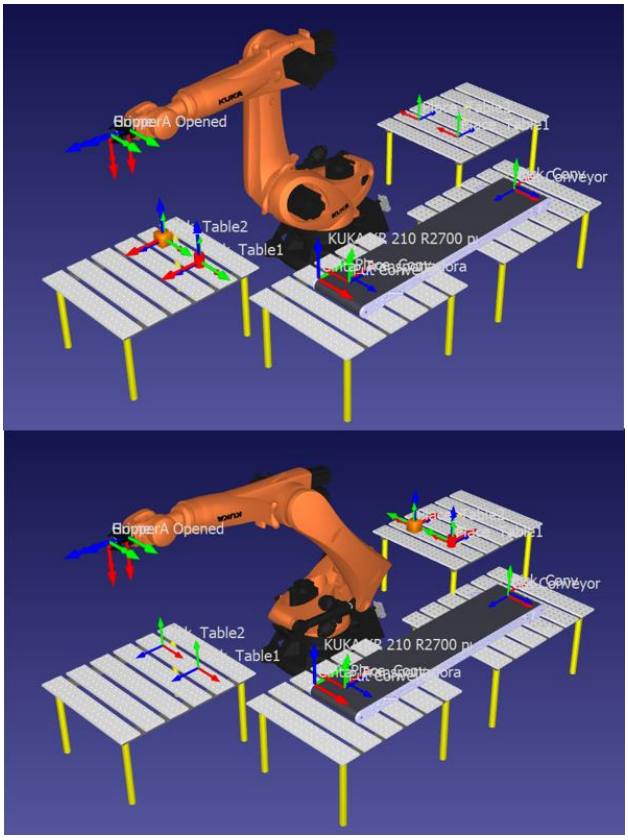


Figura 79: Estación A al comienzo (arriba) y al final (abajo)

```
NºLÍNEA              INSTRUCCIÓN
   1      J PR[6] 80deg/sec FINE OFFSET,PR[8]
   2      L PR[6] 50mm/sec FINE
   3      CALL CERRAR
   4      L PR[6] 50mm/sec FINE OFFSET,PR[8]
   5      LBL[1]
   6      R[1] =R[1] - 1
   7      J PR[2] 80deg/sec FINE OFFSET,PR[9]
   8      L PR[2] 50mm/sec FINE
   9      CALL ABRIR
  10      L PR[2] 50mm/sec FINE OFFSET,PR[9]
  11      J PR[1] 80deg/sec FINE
  12      CALL MOVER CINTA
  13      J PR[3] 80deg/sec FINE OFFSET,PR[9]
  14      L PR[3] 50mm/sec FINE
  15      CALL CERRAR
  16      L PR[3] 50mm/sec FINE OFFSET,PR[9]
  17      IF R[1] = 0 JMPLBL[2]
  18      J PR[4] 80deg/sec FINE OFFSET,PR[8]
  19      L PR[4] 50mm/sec FINE
  20      CALL ABRIR
  21      L PR[4] 50mm/sec FINE OFFSET,PR[8]
  22      J PR[1] 80deg/sec FINE
  23      J PR[7] 80deg/sec FINE OFFSET,PR[8]
  24      L PR[7] 50mm/sec FINE
  25      CALL CERRAR
  26      L PR[7] 50mm/sec FINE OFFSET,PR[8]
  27      J PR[1] 50deg/sec FINE
  28      JMPLBL[1]
  29      LBL[2]
  30      J PR[5] 80deg/sec FINE OFFSET,PR[8]
  31      L PR[5] 50mm/sec FINE
  32      CALL ABRIR
  33      L PR[5] 30mm/sec FINE OFFSET,PR[8]
  34      J PR[1] 50deg/sec FINE
```

Figura 80: Programa TPE del ejemplo A

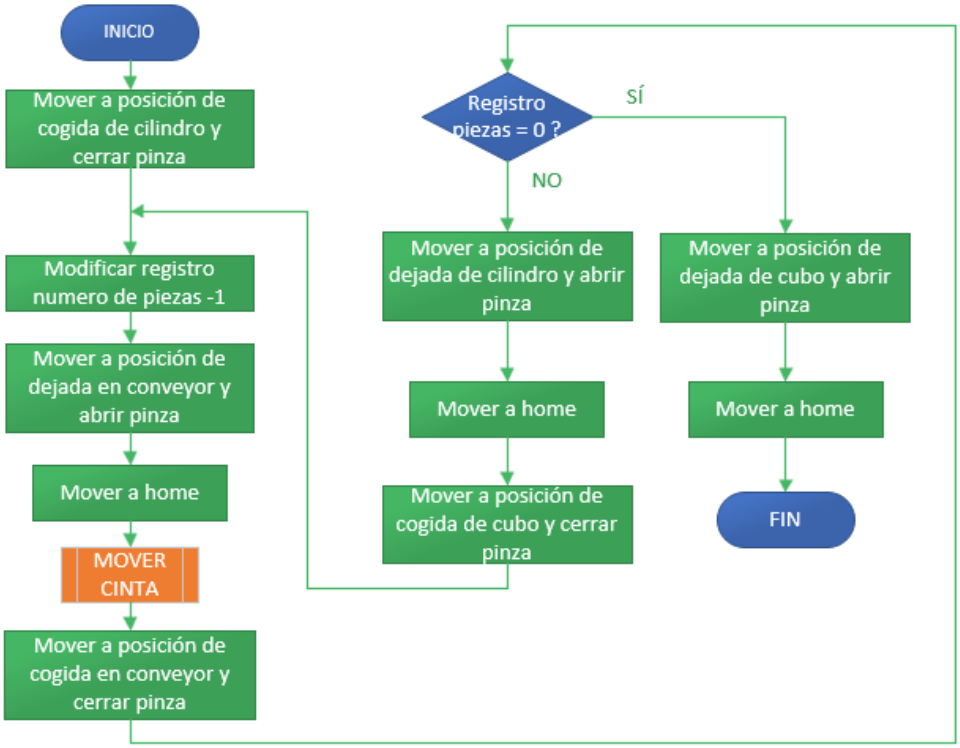


Figura 81: Flujograma del algoritmo del ejemplo A

9.2 Ejemplo B, Paletizado

9.2.1 Análisis del programa del ejemplo B

Este programa consiste en una pequeña estación de paletizado en la cual un robot va cogiendo piezas colocadas en una mesa y las apila en torres de 2 en la otra. El número de piezas es de 12 (6 torres, una fila de 3 y otra de 3, con 2 piezas cada una). y las piezas son cubos. El objetivo de este ejemplo es demostrar cómo, con solo 3 registros de posición (Pick, Place y Home), se puede realizar una operación compleja como es el paletizado. Pare ello se ha hecho uso de la modificación de registros y de la utilización de estructuras cíclicas-condicionales.

La estación (Figura 82) está compuesta por dos mesas, una de recogida y otra de dejada, un robot entre ellas y 12 piezas cúbicas divididas por colores en 6 grupos. El robot moverá cada una de las piezas y la apilará según colores formando un pallet de 3x2 con dos pisos.

En la Figura 83 se muestra el programa empleado para este ejemplo. Con el fin de facilitar la comprensión del mismo, se ha añadido una columna de número de línea a la izquierda de cada instrucción.

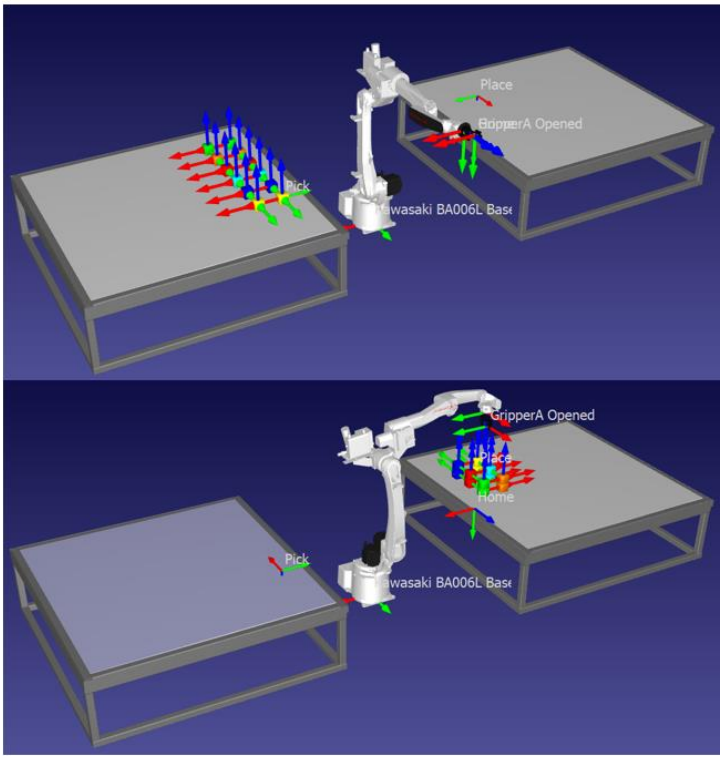


Figura 82: Estación B al comienzo (arriba) y al final (abajo)

```
NºLÍNEA              INSTRUCCIÓN
   1     J PR[1] 85deg/sec FINE
   2     LBL[1]
   3     R[5] =R[3] * R[4]
   4     PR[2,2] =PR[2,2] - R[5]
   5     R[4] =R[4] + 1
   6     IF R[6] = 2 JMPLBL[2]
   7     IF R[6] = 1 JMPLBL[3]
   8     LBL[4]
   9     R[6] =R[6] + 1
  10     JMPLBL[1]
  11     LBL[2]
  12     PR[3,1] =PR[3,1] + R[3]
  13     LBL[3]
  14     J PR[2] 85deg/sec FINE OFFSET,PR[4]
  15     L PR[2] 60mm/sec FINE
  16     CALL CERRAR
  17     L PR[2] 60mm/sec FINE OFFSET,PR[4]
  18     J PR[1] 85deg/sec FINE
  19     J PR[3] 85deg/sec FINE OFFSET,PR[5]
  20     L PR[3] 60mm/sec FINE
  21     CALL ABRIR
```

```
NºLÍNEA              INSTRUCCIÓN
  22     L PR[3] 60mm/sec FINE OFFSET,PR[5]
  23     PR[2,1] =PR[2,1] + R[3]
  24     J PR[2] 85deg/sec FINE OFFSET,PR[4]
  25     L PR[2] 60mm/sec FINE
  26     CALL CERRAR
  27     L PR[2] 60mm/sec FINE OFFSET,PR[4]
  28     J PR[1] 85deg/sec FINE
  29     PR[2,1] =PR[2,1] - R[3]
  30     J PR[3] 85deg/sec FINE OFFSET,PR[5]
  31     PR[3,3] =PR[3,3] + R[1]
  32     L PR[3] 60mm/sec FINE
  33     CALL ABRIR
  34     PR[3,3] =PR[3,3] - R[1]
  35     L PR[3] 60mm/sec FINE OFFSET,PR[5]
  36     PR[2,2] =PR[2,2] + R[5]
  37     IF R[6] = 1 JMPLBL[4]
  38     R[6] =R[6] - 1
  39     PR[3,1] =PR[3,1] - R[3]
  40     PR[3,2] =PR[3,2] + R[3]
  41     IF R[4] < R[2] JMPLBL[1]
```

Figura 83: Programa TPE del ejemplo B

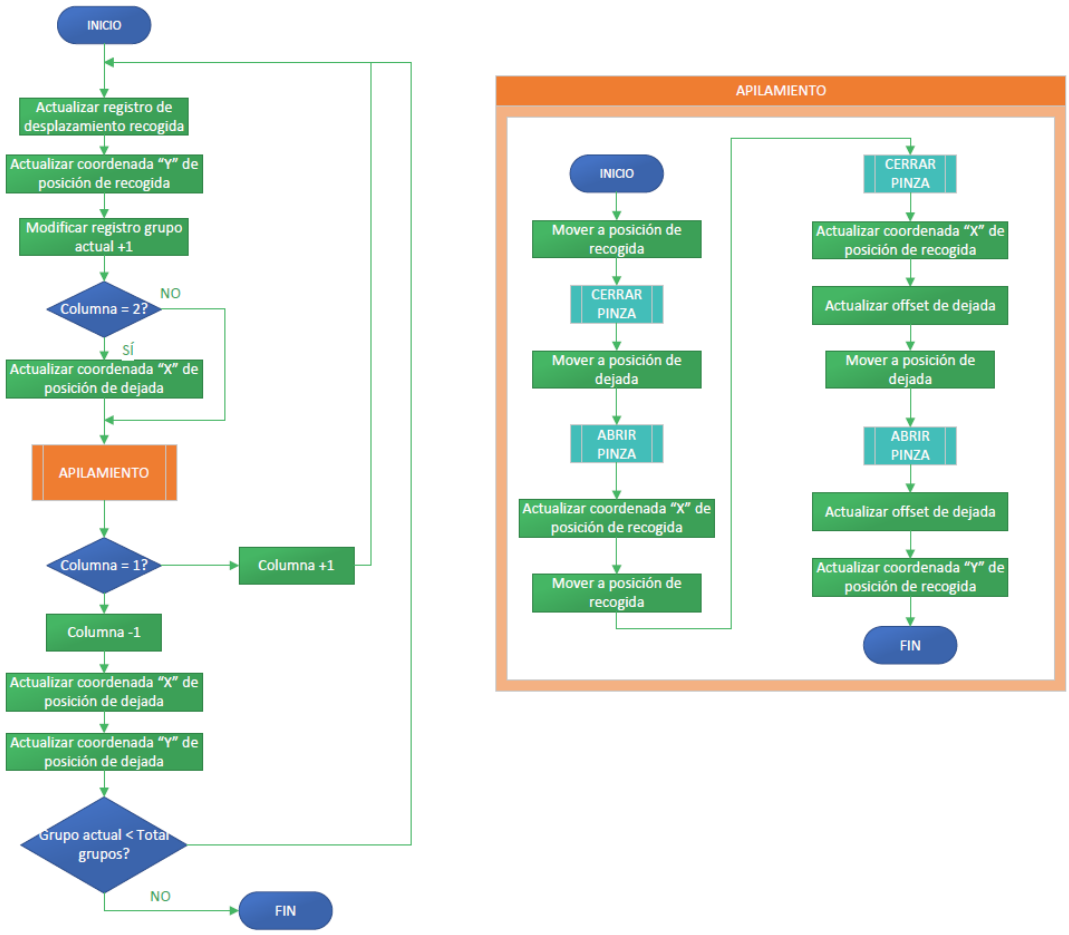


Figura 84: Flujograma del algoritmo del ejemplo B

10. CONCLUSIONES
10.1 Consecución de los objetivos

Tras la conclusión del trabajo se puede decir que se han cumplido la mayoría de los objetivos detallados en la propuesta original, a excepción de parte de una meta de utilización de sensores, proponiéndose así, como reto de cara a un futuro proyecto que prosiga con la labor efectuada en este.

Para comenzar, se ha conseguido desarrollar una aplicación intuitiva y coherente con las necesidades descritas en la propuesta siendo también acorde con el uso al que se va a destinar. Tras el análisis del código fuente proporcionado desde la web y el estudio de las posibilidades del mismo, se comenzó con el diseño y construcción de la interfaz gráfica siendo este un proceso de continua modificación y mejora. Se comenzó añadiendo la funcionalidad de poder almacenar tanto registros de posición como registros escalares dentro de la pestaña “DATA” y no solo eso, sino también la modificación o eliminación de uno de estos registros, cumpliendo así los objetivos A y B propuestos en el apartado 2.1 Objetivos del proyecto.”.

A continuación, se realizó un sistema de gestión de archivos que permite dividir el proyecto en 5 ficheros los cuales se pueden guardar en una ruta determinada para luego poder ser abiertos y modificados a posteriori. Este sistema de archivos permite tanto el guardado como la lectura de los proyectos creados en la herramienta, cumpliendo así el objetivo C.

La tarea que más tiempo llevó conseguir ha sido la de crear el algoritmo capaz de leer un programa en TPE, ser capaz de interpretarlo y ejecutarlo en un robot simulado (Objetivo D). Se han conseguido utilizar las principales instrucciones del lenguaje TPE para aplicarlas en la programación del robot.

Y por último la programación de todos los avisos, alertas y métodos auxiliares que se encargan de evitar la aparición de errores y excepciones, guiando al usuario por el camino correcto para tener una experiencia de uso más agradable (Objetivo E). Algunos de estos mensajes de alerta también son usados para orientar al usuario de cuál es el orden adecuado a la hora de utilizar el programa o indicarle qué hacer en caso de cometer un error.

10.2 Líneas de trabajo futuras.

Durante el desarrollo de la aplicación, las distintas soluciones propuestas para los objetivos del proyecto han dado pie a explorar a fondo la API suministrada y descubrir el potencial que tiene de cara a mejorar e incluir nuevas funcionalidades. Por esta razón se proponen como líneas de investigación futuras las siguientes tareas que podrían enriquecer de manera significativa la experiencia de uso de la aplicación desarrollada pudiendo así aplicarse a un mayor número de trabajos de índole diversa. Las líneas de trabajo futuras propuestas se pueden declarar en las siguientes sentencias:

1. Añadir el control y gestión real de señales digitales, que permitan así la inclusión de sensores, otros robots y su trabajo en colaboración.

   Como es bien sabido, los sensores son un elemento esencial en cualquier actividad industrial, pues son necesarios para obtener los datos necesarios para determinar propiedades diversas del sistema y actuar en consecuencia. Además, sería interesante de cara a futuro poder lograr la incorporación de señales digitales tanto de entrada como salida para poder lograr una comunicación entre robots y robot-sensor.

2. Incluir una funcionalidad que logre el manejo de dos programas operando dos robots diferentes los cuales trabajarían de forma coordinada.

   En la mayoría de aplicaciones industriales en las que se precisa de la robótica no solo se emplea un único robot, sino que son varios robots los que trabajan de forma síncrona sobre una tarea común. Por este motivo una de las líneas de trabajo más importantes debería seguir esta propuesta ya que se conseguirían realizar programas mucho más complejos y más cercanos a lo que sería una actividad industrial.

3. Investigar en cómo aplicar y utilizar macros para emplear diferentes herramientas, tales como otro tipo de garras, ventosas o puntas de soldadura.

   Las aplicaciones de “Pick and Place” forman parte de una gran parte de los usos de un robot, pero no son las únicas que los robots industriales pueden hacer. Incluyendo en el proyecto otro tipo de herramientas y dándoles

funcionalidad se podrían realizar programas aplicables a más áreas como la soldadura, la pintura o el corte de piezas.

4. Incluir nuevas opciones de precisión aparte de FINE como por ejemplo CNT.

5. Ampliar el catálogo de elementos que se pueden usar en el programa, agregando nuevos tipos de objetos y superficies para manipular, así como protecciones de seguridad o pedestales donde colocar piezas.

6. Permitir la aplicación de offset y tool offset en una misma instrucción de movimiento.
7. Desarrollar un sistema de depuración del programa TPE que posibilite la ejecución del programa línea por línea indicando explícitamente que línea se está implementando en el momento.

   Consistiría en añadir botones para la detención, pausa y reanudación del programa. Sería interesante desde el punto de vista del usuario para revisar posibles fallas en el programa y poder enmendarlas de manera más rápida.

11. BIBLIOGRAFÍA

[1] Ekon. (2018 de Enero de 26). *Las cuatro revoluciones industriales y quienes no lo vieron venir*. Recuperado en Enero de 2022, de https://www.ekon.es/las-cuatro-revoluciones-y-quienes-no-lo-vieron-venir/

[2] Solex. (5 de Febrero de 2021). *Conozca las 4 revoluciones industriales y las tecnologías que las impulsaron*. Recuperado en Enero de 2022, de IBM Maximo: https://www.solex.biz/noticias/revoluciones-industriales-tecnologias-impulsaron/

[3] Wikipedia.org. (2022). *Revoluciones Industriales*. Recuperado en Enero de 2022, de Wikipedia, la enciclopedia libre: https://es.wikipedia.org/wiki/Revoluciones_industriales

[4] ESD Robotics. (14 de Diciembre de 2020). *Tipos de robots industriales y sus usos*. Recuperado en Febrero de 2022, de https://www.edsrobotics.com/blog/tipos-robots-industriales-usos/

[5] Revista de Robots. (7 de Diciembre de 2021). *Robots industriales. Qué es un robot industrial, tipos y ejemplos de robots*. Recuperado en Febrero de 2022, de https://revistaderobots.com/robots-y-robotica/robots-industriales-y-robotica-industrial/

[6] AER Automation. (2022). Quiénes Somos. Recuperado en Febrero de 2022, de https://www.aer-automation.com/quienes-somos/oDK.

[7] IFR. (2018). *About IFR*. Recuperado en Febrero de 2022, de https://ifr.org/association

[8] IFR. (28 de Octubre de 2021). *IFR presents World Robotics 2021 reports*. Recuperado en Febrero de 2022, de https://ifr.org/ifr-press-releases/news/robot-sales-rise-again

[9] Universidad Politécnica de Madrid. (1997). *Fundamentos de Robótica* (Segunda ed.). Madrid, España: McGraw-Hill.

[10] RoboDK. (2022). *Acerca de RoboDK*. Recuperado en Marzo de 2022, de https://robodk.com/es/about

[11] RoboDK. (2022). *API de RoboDK*. Recuperado en Marzo de 2022, de https://robodk.com/doc/es/RoboDK-API.html#CsAPI

[12] Microsoft. (2022). *Documentación de C#*. Recuperado en Marzo de 2022, de https://docs.microsoft.com/es-es/dotnet/csharp/

[13] Microsoft. (2022). *System.Windows.Forms Espacio de nombres*. Recuperado en Abril de 2022, de https://docs.microsoft.com/es-es/dotnet/api/system.windows.forms?view=windowsdesktop-6.0