UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL

DISEÑO Y SIMULACIÓN DE APLICACIONES ROBOTIZADAS MEDIANTE ROBODK

DESARROLLO DE UNA CÉLULA INDUSTRIAL DE MECANIZADO, CONTROL DE CALIDAD BASADO EN VISIÓN ARTIFICIAL Y EMPAQUETADO

AUTOR: Fernando Cuello Calvo

TUTOR: Ángel Valera Fernández

Curso Académico: 2020-21

Agradecimientos

A mis padres y a mi hermano, por haberme apoyado durante todos estos años y estar siempre cuando les he necesitado.

A Emma, por escucharme y apoyarme siempre incondicionalmente y haber sido mi sustento durante toda la carrera y el máster.

A todos mis amigos que me han acompañado durante la carrera, en especial a Jorge, sin ti toda esta aventura que comenzamos hace 6 años no hubiera sido lo mismo.

A mi tutor Ángel, por haberme ayudado a llevar este trabajo a cabo.

Resumen

La evolución actual de nuevas tecnologías y la reducción de los costes de fabricación va a hacer posible que, en un futuro próximo, se puedan desarrollar productos para cualquier tipo de necesidad y además con un alto grado de personalización. Esto será posible gracias a la automatización de las fábricas y procesos industriales. Además, la inclusión de la robótica en estos procesos va a permitir aumentar la productividad, reducir los costes de producción y reducir el riesgo para los operarios.

En este Trabajo Fin de Máster se va a llevar a cabo el diseño y la simulación de una célula automatizada basada en el uso de robots. Para ello, se va a hacer uso de RoboDK, que es un software muy potente que permite, a diferencia de otros softwares de simulación conocidos como RobotStudio o KUKASIM, simular cualquier tipo de robot gracias a su librería con más de 500 robots diferentes de las principales marcas que lideran el sector.

Con RoboDK y su programación fuera de línea, es posible programar los robots fuera del entorno de producción y programarlos directamente desde cualquier ordenador, eliminando los tiempos de actividad de la producción debidos a las paradas de programación de los robots.

La propuesta de célula robotizada de este Trabajo Fin de Máster es una célula compuesta por tres robots (ABB IRB140, UR3 y UR3e) y diferente equipamiento como sistemas de visión, cintas transportadoras, elementos finales para los robots, etc. Con todo ello se va a simular una aplicación de mecanizado, control de calidad y empaquetado automático. Cuando se haya validado la simulación en RoboDK, se procederá al desarrollo de la aplicación en la célula robótica real situada en el Laboratorio de Robótica del Dpto. de Ingeniería de Sistemas y Automática.

Para el desarrollo de la célula real cabe destacar que RoboDK cuenta con la capacidad de exportar los programas desarrollados a cualquier tipo de robot sin necesidad de conocer el lenguaje de programación de dicho robot, gracias a los más de 70 post-procesadores que incluye.

Por lo tanto, para la realización de este Trabajo Fin de Máster se van a aplicar conocimientos relacionados con la programación de robots de una célula robotizada con tres robots sincronizados haciendo el uso de software RoboDK, conocimientos de visión artificial para realizar un control de calidad adecuado, conocimientos sobre comunicaciones TCP/IP para la comunicación entre el robot y la cámara, y la transferencia y ejecución de los programas realizados mediante RoboDK a los robots reales del laboratorio.

Abstract

The current evolution of new technologies and the reduction of manufacturing costs will make it possible soon, to develop products for any type of need and with a high degree of customization. This will be possible thanks to the automation of factories and industrial processes. Moreover, the inclusion of robotics in these processes will increase productivity, reduce production costs, and reduce the risk for operators.

In this master's Thesis, the design and simulation of an automated cell based on the use of robots will be carried out. For this purpose, RoboDK will be used, which is a very powerful software that allows, unlike other well-known simulation software such as RobotStudio or KUKASIM, to simulate any type of robot thanks to its library with more than 500 different robots of the main brands that lead the sector.

With RoboDK and its offline programming, it is possible to program robots outside the production environment and program them directly from any computer, removing production downtime due to robot programming stoppages.

The robotic cell proposal of this master's Thesis is a cell composed of three robots (ABB IRB140, UR3 and UR3e) and different equipment such as vision systems, conveyor belts, final elements for the robots, etc. All this will be used to simulate a machining, quality control and automatic packaging application. Once the simulation has been validated in RoboDK, the application will be developed in the real robotic cell located in the Robotics Laboratory of the Dept. of Systems Engineering and Automatics.

For the development of the real cell, it should be noted that RoboDK can export the programs developed to any type of robot without the need to know the programming language of the robot, thanks to the more than 70 post-processors included.

Therefore, in order to carry out this master's Thesis, knowledge related to robot programming of a robotic cell with three synchronized robots using RoboDK software, knowledge of artificial vision to carry out adequate quality control, knowledge of TCP/IP communications for communication between the robot and the camera, and the transfer and execution of the programs carried out using RoboDK to the real robots in the laboratory will be applied.

Resum

L'evolució actual de noves tecnologies i la reducció dels costos de fabricació farà possible que, en un futur pròxim, es puguen desenvolupar productes per a qualsevol mena de necessitat i a més amb un alt grau de personalització. Això serà possible gràcies a l'automatització de les fàbriques i processos industrials. A més, la inclusió de la robòtica en aquests processos permetrà augmentar la productivitat, reduir els costos de producció i reduir el risc per als operaris.

En aquest Treball Fi de Màster es durà a terme el disseny i la simulació d'una cèl · lula automatitzada basada en l'ús de robots. Per a això, es farà ús de RoboDK, que és un programa molt potent que permet, a diferència d'altres programes de simulació coneguts com RobotStudio o KUKASIM, simular qualsevol tipus de robot gràcies a la seua llibreria amb més de 500 robots diferents de les principals marques que lideren el sector.

Amb RoboDK i la seua programació fora de línia, és possible programar els robots fora de l'entorn de producció i programar-los directament des de qualsevol ordinador, eliminant els temps d'activitat de la producció deguts a les parades de programació dels robots.

La proposta de cèl · lula robotitzada d'aquest Treball Fi de Màster és una cèl · lula composta per tres robots (ABB IRB140, UR3 i UR3e) i diferent equipament com a sistemes de visió, cintes transportadores, elements finals per als robots, etc. Amb tot això se simularà una aplicació de mecanitzat, control de qualitat i empaquetat automàtic. Quan s'haja validat la simulació en RoboDK, es procedirà al desenvolupament de l'aplicació en la cèl · lula robòtica real situada en el Laboratori de Robòtica del Dpto. d'Enginyeria de Sistemes i Automàtica.

Per al desenvolupament de la cèl · lula real cal destacar que RoboDK compta amb la capacitat d'exportar els programes desenvolupats a qualsevol mena de robot sense necessitat de conéixer el llenguatge de programació d'aquest robot, gràcies als més de 70 post-processadors que inclou.

Per tant, per a la realització d'aquest Treball Fi de Màster s'aplicaran coneixements relacionats amb la programació de robots d'una cèl · lula robotitzada amb tres robots sincronitzats fent l'ús de programari RoboDK, coneixements de visió artificial per a realitzar un control de qualitat adequat, coneixements sobre comunicacions TCP/IP per a la comunicació entre el robot i la cambra, i la transferència i execució dels programes realitzats mitjançant RoboDK als robots reals del laboratori.

Índice

I MEMORIA **3**

CAPÍTULO I: INTRODUCCIÓN **3**

1.1. Introducción 3

1.2. Objetivos 5

1.3. Estructura del documento 6

1.4. Justificación 7

CAPÍTULO II: DESARROLLO TEÓRICO **9**

2.1. Robótica Industrial 9

2.2. Paquetes de Simulación 12

2.3. RoboDK 14

2.4. Visión Artificial 17

2.5. Sistemas de comunicación industriales 20

CAPÍTULO III: DESARROLLO PRÁCTICO **23**

3.1. Planteamiento general de la célula 23

3.1.1. Justificación 23

3.1.2. Elementos 24

3.1.3. Funcionamiento 31

3.2. Planteamiento célula RoboDK 33

3.2.1. Programa en RoboDK 33

3.2.2. Algoritmo Visión Artificial 49

3.2.3. Comunicación cámara-robot 53

3.2.4. Resultados 55

3.3. Planteamiento célula laboratorio 61

3.3.1. Puesta en marcha 61

3.3.2. Impresión 3D 64

3.3.3. Algoritmo de visión y comunicaciones 67

3.3.4. Programación robots 72

3.3.5. Resultados 74

CAPÍTULO IV: CONCLUSIONES **81**

4.1. Conclusiones 81

4.2. Trabajo Futuros 82

BIBLIOGRAFÍA **83**

II PRESUPUESTO **87**

CAPÍTULO I: PRESUPUESTO **87**

1. Introducción 87

2. Presupuesto detallado 87

2.1. Costes de amortización 87

2.2. Costes materiales 88

2.3. Costes Recursos Humanos 88

2.4. Presupuesto Total 90

III ANEXOS **93**

ANEXO A: Ejecución de simulación HTML de RoboDK 93

ANEXO B: Manual Utilización Célula RoboDK 94

ANEXO C: Manual Utilización Célula Laboratorio 96

ANEXO D: Código de los programas utilizados 98

D.1. Código Programas RoboDK 98

D.2. Código PyCharm 108

D.3. Código Robots Reales 114

Índice de Figuras

Figura 1. Evolución de instalaciones de robots industriales (Fuente: World Robotics 2019) 9

Figura 2. Diversos ejemplos de nuevas aplicaciones robóticas. Fuente (ABB, AIT, IEEE Spectrum, ESCOBI) 10

Figura 4. Principales paquetes de simulación. (Fuente: elaboración propia). 12

Figura 5. Logotipo RoboDK Software. (Fuente: RoboDK) 14

Figura 6. Ejemplo de pick & place con 2 robots y una cinta transportadora. (Fuente: RoboDK) 14

Figura 7. Ejemplo RoboDK programado con Python. (Fuente: RoboDK) 15

Figura 8. Brazo robótico con sistema de visión artificial integrado. (Fuente: Electrónica Edimar) 17

Figura 9. Esquema funcionamiento sensor visión. (Fuente: Cognex) 17

Figura 10. Cámara inteligente integrado en un brazo robótico para inspección. (Fuente: EDS Robotics) 18

Figura 11. Sistema de visión avanzado en una línea de producción. (Fuente: EDS Robotics) 18

Figura 12. Concepto de redes comunicación industrial. (Fuente: Siemens) 20

Figura 13. Funcionamiento estructura cliente servidor. (Fuente: OOSE) 21

Figura 14. UR3, IRB140 y UR3e del laboratorio DISA de izquierda a derecha. 23

Figura 15. Robot UR3 de Universal Robots del laboratorio de Robótica. 24

Figura 16. Robot UR3e de Universal Robots del laboratorio de Robótica 25

Figura 17. Robot IRB140 de ABB del laboratorio de Robótica. 25

Figura 18. Mesa giratoria vista desde arriba y desde el lateral. 26

Figura 19. Cinta transportadora del laboratorio de Robótica. 27

Figura 20. Cámara Hércules del laboratorio de Robótica. 27

Figura 21. Pieza manipulada 28

Figura 22. Piezas manipuladas en color azul, verde y rojo. 28

Figura 23. Rediseño de la pieza manipulada. 29

Figura 24. Diseño doble herramienta. 29

Figura 25. Ensamblaje ventosa, delineador y soporte. 30

Figura 26. Esquema funcionamiento célula robotizada por etapas 31

Figura 27. Interfaz Principal RoboDK. 33

Figura 28. Distribución secciones Interfaz Principal RoboDK. 34
Figura 29. Sistemas de Referencia RoboDK. 35
Figura 30. Despliegue del sistema de referencia World. 35
Figura 31. Despliegue sistema referencia Mesa Giratoria, ... 36
Figura 32. Despliegue sistema referencias de los tres robots. 37
Figura 33. Objetos y robots en posición en RoboDK. 37
Figura 34. Ventana configuración mecanismos RoboDK. 38
Figura 35. Imagen capturada por la cámara virtual en la posición de procesamiento de imagen de la cinta transportadora. 39
Figura 36. Estación Ejemplo Post-Procesadores. 44
Figura 37. Programa pick & place Ejemplo Post-Procesadores. 45
Figura 38. Navegación al Selector de Post-Procesador. 45
Figura 39. Navegación a Generar Programa de Robot. 46
Figura 40. Matrices RGB obtenidas a partir de una imagen. (Fuente: OpenCV) 49
Figura 41. Fotografía RoboDK con tres piezas de distintos colores. 50
Figura 42. Imágenes binarias obtenidas a través de las máscaras. En orden, máscara roja, verde y azul 51
Figura 43. Ejemplo enumeración función minAreaRect. (Fuente: OpenCV). 52
Figura 44. Ejemplo cálculo ángulo función minAreaRect. (Fuente: OpenCV) 52
Figura 45. QR que enlaza con archivo simulación RoboDK. 55
Figura 46. Simulación célula RoboDK (I) 56
Figura 47. Simulación célula RoboDK (II) 56
Figura 48. Simulación célula RoboDK (III) 57
Figura 49. Simulación célula RoboDK (IV) 57
Figura 50. Simulación célula RoboDK (V) 58
Figura 51. Simulación célula RoboDK (VI) 58
Figura 52. Simulación célula RoboDK (VII) 59
Figura 53. Simulación célula RoboDK (VIII) 59
Figura 54. Simulación célula RoboDK (IX) 60
Figura 55. Célula robotizada del laboratorio. 61
Figura 56. Proceso de calibración herramienta ventosa mediante método 4 puntos. .. 62
Figura 57. Resultado final de las piezas manipuladas impresas. 65

Figura 58. Resultado final de la doble herramienta impresa en 3D acoplada a la brida del robot. 66

Figura 59. Imagen capturada por la cámara del laboratorio y su posterior procesado. 67

Figura 60. Imagen obtenida al aplicar máscara de color azul sobre pieza azul. 68

Figura 61. Escala de color HSV. (Fuente: Mathworks) 68

Figura 62. Mejora en la detección color aplicando escala HSV 69

Figura 63. Vista de los componentes HSV. (Fuente: Stackoverflow) 69

Figura 64. Interfaz gráfica diseñada para el servidor del laboratorio. 71

Figura 65. Diagrama de flujo de la programación de la célula. 72

Figura 66. Código QR para visualizar el funcionamiento de la célula del laboratorio. 74

Figura 67. Simulación célula laboratorio (I) 75

Figura 68. Simulación célula laboratorio (II) 75

Figura 69. Simulación célula laboratorio (III) 76

Figura 70. Simulación célula laboratorio (IV) 76

Figura 71. Simulación célula laboratorio (V) 77

Figura 72. Simulación célula laboratorio (VI) 77

Figura 73. Simulación célula laboratorio (IV) 78

Figura 74. Simulación célula laboratorio (IV) 79

Figura 75. Ejemplo de ubicación archivo simulación. 93

Figura 76. Pantalla de visualización previa a la simulación. 93

Figura 77. Indicación de la situación de MainProgram en la célula de RoboDK. 94

Figura 78. Indicación de la situación de StopALL en la célula de RoboDK. 95

Figura 79. Programas cargados en la pantalla táctil del UR3 y UR3e respectivamente. 96

Figura 80. Indicación de la situación del botón para iniciar el servidor en PyCharm. 97

Índice de Tablas

Tabla 1. Utilización de los tipos de aplicaciones en función del sector.. 11
Tabla 2. Parámetros cámara virtual RoboDK. 39
Tabla 3. Instrucciones de programación en RoboDK. 40
Tabla 4. Parámetros RGB para creación máscaras de colores. 50
Tabla 5. Valores calibración doble herramienta. 62
Tabla 6. Conexiones entre robots. Entradas (verde) y salidas (azul). 63
Tabla 7. Señales digitales de los actuadores (azul) y sensores (verde) de la estación.. 64
Tabla 8. Parámetros Impresión 3D de la pieza manipulada. 64
Tabla 9. Parámetros Impresión 3D de la pieza doble herramienta. 65
Tabla 10. Costes de amortizaciones. 87
Tabla 11. Costes de materiales. 88
Tabla 12. Costes Ingeniero Junior. 89
Tabla 13. Costes Ingeniero Senior. 89
Tabla 14. Costes Recursos Humanos. 89
Tabla 15. Presupuesto total proyecto. 90

Parte I

MEMORIA TÉCNICA

CAPÍTULO I: INTRODUCCIÓN

En este primer capítulo se realizará una introducción al trabajo, presentando los principales objetivos y la estructura del documento y la justificación de la realización del trabajo.

1.1. Introducción

Hoy en día, el sector de la robótica está ganando mucha importancia en el ámbito industrial ya que la instalación de robots en una línea de producción ofrece una gran cantidad de ventajas, como pueden ser la reducción de costes de fabricación, reducción de tiempos, aumento de la productividad, etc.

Además de lo anterior, el hecho de tener robots en una línea de producción dispara las oportunidades de personalización, gracias a la gran flexibilidad con la que cuentan. Esto permite diseñar células robotizadas que sean capaces de fabricar cualquier tipo de producto para que se adapte perfectamente a las necesidades de cada cliente.

En este Trabajo Fin de Máster se propone crear una célula automatizada basada en robots, en la que se trabajen diferentes aspectos del desarrollo de una célula robotizada en el ámbito industrial, como el diseño de la célula, la simulación de esta mediante un software de simulación de robots y por último la aplicación del diseño final a una célula real en el Laboratorio de Robótica del Dpto. de Ingeniería de Sistemas y Automática.

Para llevar a cabo la simulación de la célula, se propone usar el software de simulación RoboDK. Este software, a diferencia de otros como RobotStudio o KUKASIM, permite programar cualquier tipo de robot de las principales marcas del mercado y cuenta con una librería de más de 500 robots diferentes. Esto es posible gracias a que RoboDK incluye más de 70 post-procesadores que permiten exportar del lenguaje de programación común que emplea RoboDK al lenguaje de programación de cualquiera de los 40 fabricantes con los que trabaja el programa.

Aparte de ello, RoboDK es un software de programación fuera de línea, lo cual permite programar los robots desde un ordenador y una vez que en simulación todo funcione adecuadamente, se puede transmitir el programa rápidamente a la célula real reduciendo notablemente los tiempos debidos a las paradas de programación de los robots.

La célula robotizada que se pretende desarrollar está compuesta por los tres robots existentes en el laboratorio del departamento (UR3e, UR3 y ABB IRB140) y de diverso equipamiento como sistemas de visión, cintas transportadoras, elementos finales para los robots, etc. Con todo ello, se va a simular una aplicación de mecanizado, control de calidad mediante visión artificial y empaquetado automático.

La pieza con la que se va a trabajar en la célula es una pieza cúbica con un saliente cilíndrico, el cual encaja en las casillas de la mesa giratoria del departamento. El mecanizado de la pieza se simulará usando un delineador en vez de una fresa verdadera

ya que como la pieza está impresa en 3D, para cada ensayo se tendría que imprimir una nueva. Además, para que un mismo robot pueda agarrar la pieza con una ventosa y realizar el mecanizado con un delineador se diseñará un soporte para ambas piezas que permitirá tener una doble herramienta.

El sistema de visión artificial de la célula se encargará de detectar el color de la pieza y la orientación de esta para poder realizar un paletizado automático en función de color y corregir posibles desviaciones de la orientación de la pieza.

Para poder transmitir la información entre el robot y el ordenador que procesa la imagen de la cámara, se va a usar un comunicación Ethernet TCP/IP en la que el ordenador (servidor) envíe la información al robot (cliente) a través de sockets.

De esta forma, en este Trabajo Fin de Máster se van a poner en práctica conocimientos relacionados con la programación de robots industriales, el software RoboDK, sistemas de visión artificial, comunicaciones TCP/IP y ejecución de los programas simulados en una célula real.

1.2. Objetivos

El objetivo principal del presente Trabajo Fin de Máster es el diseño, simulación, desarrollo y análisis de células industriales robotizadas. Aunque el trabajo se ha planteado de forma genérica, por lo que se podría abordar cualquier tipo de célula robótica, se ha propuesto la implementación de una célula de mecanizado, transporte de material y control de calidad puesto que son los tipos de aplicaciones más habituales en sectores como las industrias del automóvil, metalúrgica, electrónica o alimentación.

Para poder validar los resultados obtenidos en este trabajo, se desarrollará primero una célula simulada. Una vez verificado el correcto funcionamiento de ésta, se va a proceder a su implementación en la célula robótica del Dpto. de Ingeniería de Sistemas y Automática de la Universitat Politécnica de Valencia. Dicha célula está compuesta por un robot industrial ABB IRB140, dos robots colaborativos UR3 y UR3e, dos cintas transportadoras y diferentes sistemas de sensorización como cámaras de visión artificial, sensores de fuerza, sensores de presencia, etc.

De esta forma, para lograr este objetivo principal, en el trabajo se proponen los siguientes objetivos específicos:

- Conocer los sectores de mayor uso de los robots manipuladores y las aplicaciones más habituales en las que los sistemas robotizados están presentes.
- Estudiar los requerimientos básicos para el diseño y la implantación de células robotizadas funcionales.
- Estudiar las características y el funcionamiento de un software de alto nivel de simulación de robots, como es RoboDK.
- Desarrollar una célula robotizada con la aplicación software de simulación de robots RoboDK.
- Realizar la programación de los diferentes robots industriales y dispositivos existentes en la célula robótica, así como de los sistemas de comunicación y coordinación entre los equipos de la célula.
- Implementar un sistema de visión artificial que permita obtener información útil sobre un elemento de la célula robotizada.
- Simular la correcta implantación y funcionamiento de la célula robotizada simulada.
- Verificar el correcto funcionamiento de la célula en un entorno real basado en robots colaborativos e industriales.

1.3. Estructura del documento

Con el propósito de cumplir los objetivos descritos en la sección anterior, este Trabajo de Fin de Máster se ha estructurado de la siguiente forma:

- **Capítulo II: Desarrollo teórico.** En este capítulo se realizará un desarrollo teórico sobre distintos aspectos en los que se enfocan este TFM como la robótica en la actualidad y los paquetes de simulación más usados, el software de RoboDK, la visión artificial y los sistemas de comunicación en células robotizadas. Esto servirá como punto de partida para el diseño de la célula robotizada.
- **Capítulo III: Desarrollo práctico.** En este capítulo se describirá como los distintos temas presentados en el desarrollo teórico se han aplicado en el trabajo para poder realizar una célula robotizada que funcione tanto en simulación con el software de RoboDK, como en el laboratorio con robots reales. En concreto, se presentará una propuesta inicial de célula, justificando su diseño y explicando se funcionamiento, se explicará las bases del programa RoboDK y como se ha programado la célula simulada y por último se explicará como se ha llevado a cabo la célula simulada a la célula del laboratorio.
- **Capítulo IV: Conclusiones y Trabajos Futuros.** Este capítulo se centrará en desarrollar las principales conclusiones obtenidas al realizar el trabajo, así como proponer posibles mejoras para completar el trabajo en el futuro.

1.4. Justificación

Desde el punto de vista académico, la realización de este trabajo permite al alumno poner en práctica diferentes conocimientos adquiridos en las asignaturas cursadas en el Máster de Ingeniería Industrial Superior, además de ampliar dichos conocimientos.

El diseño de una célula robotizada y la programación de tres robots, tanto en simulación como en el laboratorio, permite aplicar y expandir lo aprendido en la asignatura de Robótica Industrial, dentro de la especialidad de Control de Procesos, Automatización y Robótica.

El desarrollo de un algoritmo de visión que detecte el color y el ángulo de un objeto, así como las comunicaciones TCP/IP creadas para transmitir información permite profundizar en los conocimientos adquiridos en la asignatura optativa de Visión Artificial.

Además de ello, la programación necesaria para llevar a cabo el trabajo va a permitir mejorar las aptitudes de programación en un lenguaje de alto nivel, como es el caso de Python, las cuales son muy solicitadas dentro del ámbito profesional.

Desde el punto de vista del ámbito industrial, la utilización del software RoboDK para el diseño y programación fuera de línea de una célula robotizada ofrece grandes ventajas dentro de una línea de producción. Este tipo de software hace posible realizar un estudio de diferentes escenarios de una célula robotizada antes de configurar la célula final. Además, se pueden detectar fallos de diseño a tiempo gracias a la posibilidad de probar en simulación diferentes configuraciones.

En la industria, los sistemas de visión artificial y las comunicaciones industriales son aspectos claves en la producción, ya que la visión permite aumentar su productividad mediante el uso de cámaras que detecten defectos o marquen posiciones para un robot, y las comunicaciones industriales son esenciales para que todo este conectado y que las líneas de producción funcionen adecuadamente.

Todo lo anterior, junto con la utilización de robot reales y la configuración de una célula de producción real, son conocimientos muy valorados en el entorno industrial, por lo que la realización de este trabajo va a ser una primera toma de contacto con el sector de la robótica industrial y un punto de partida dentro del ámbito profesional como ingeniero especializado en control de procesos, automatización y robótica.

CAPÍTULO II: DESARROLLO TEÓRICO

En este capítulo, se desarrollará a nivel teórico diferentes aspectos del trabajo, que más adelante se pondrán en práctica.

2.1. Robótica Industrial

En la actualidad, la presencia de robots industriales en el ámbito industrial se ha vuelto casi imprescindible, ya que supone un aumento de la productividad para las empresas y les permiten ser más competitivas.

Si se observan los números, desde el año 2010, las instalaciones anuales de robots industriales se han más que triplicado, llegando a alcanzar la cifra de 422.000 unidades instaladas en fábricas de todo el mundo. Sin embargo, se puede observar un leve desplome en las instalaciones de robots industriales en 2019, que tuvo mayor impacto en Asia y América (-13%) que en Europa (-5%).

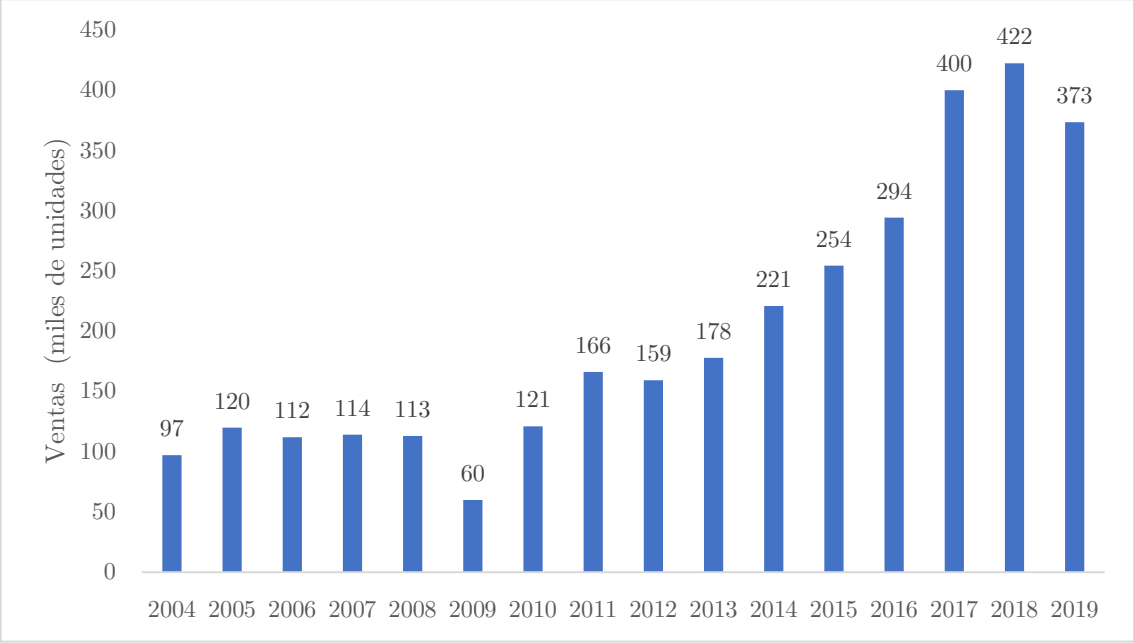

Figura 1. Evolución de instalaciones de robots industriales (Fuente: World Robotics 2019)

Según la Asociación Española de Robótica (AER), dicha caída fue provocada por el aumento de las tensiones comerciales internacionales y por el enfriamiento de la economía mundial. No obstante, la AER asegura que la tendencia de crecimiento se mantiene estable y con vísperas de un crecimiento sostenido [1].

La primera industria que fue pionera en soluciones con robots industriales fue la de la automoción, siendo empleados en las líneas de montaje para aumentar la producción y conseguir mayor calidad en la misma. Además, el uso de robots industriales también es importante en otro tipo de sectores como en la industria alimentaria, manufacturera o química.

Sin embargo, en los últimos años están apareciendo nuevas sectores y formas de obtener provecho a la robótica industrial. A continuación, se muestran algunos ejemplos de nuevos sectores en los que se esta aplicando la robótica [3].

- **Robots colaborativos.** Permiten la posibilidad de trabajar conjuntamente a humanos con robots de forma segura, pudiendo reducir la aportación del humano a la producción de hasta un 50%.
- **Medicina**. En los últimos años, el número de cirugías no invasivas o mínimamente invasivas ha aumentado considerablemente [2] por lo que la introducción de robots capaces de participar en cirugías de este estilo, como por ejemplo una endoscopia, puede hacer frente al aumento de estas cirugías a la vez que reduce la fatiga de los cirujanos, evitando su agotamiento.
- **Robots auto curativos**. Los avances en la tecnología han permitido que algunos robots puedan detectar pequeños errores en su maquinaria y repararlos de forma autónoma. Por lo tanto, en el futuro se podrá ver robots que sean capaces de realizar reparaciones mucho más complejas sin necesidad de presencia humana.
- **Agricultura.** En este sector, la invención de los drones ha supuesto un gran cambio ya que permite monitorear toda la zona de cultivo para detectar plagas o incendios e incluso analizar los mismos mediante visión artificial para saber su estado de maduración. Además, se está empezando a desarrollar tractores automatizados con la capacidad de sembrar, fertilizar y cosechar.

Figura 2. Diversos ejemplos de nuevas aplicaciones robóticas. Fuente (ABB, AIT, IEEE Spectrum, ESCOBI)

Una de las principales ventajas del uso de robots en una línea de producción es la flexibilidad que tienen para emplearse en distintas aplicaciones. Dependiendo del sector en el que nos encontremos habrá algunas aplicaciones que se usen más frecuentemente y con mayor importancia que otras. En la siguiente tabla se muestra la utilización de los tipos de aplicaciones en función del sector.

	TIPO DE APLICACIÓN				
Sector	**Manipulación de piezas**	**Aplicación de Material**	**Mecanizado**	**Soldadura**	**Control de Calidad**
Industrial alimentaria					
Ind. Textil					
Ind. Madera y Mueble					
Ind. Papel e Imprenta					
Ind. Química y Plástica					
Ind. Vidrio y Cerámica					
Ind. Metalúrgica					
Ind. Electrónica y Electricidad					
Ind. Ing. Control, Óptica					
Ind. Automóvil					

Tabla 1. Utilización de los tipos de aplicaciones en función del sector. Uso frecuente (rojo), uso medio (naranja), uso nulo o casi nulo (blanco). (Fuente: Apuntes Robótica Industrial).

2.2. Paquetes de Simulación

La aplicación de los robots industriales en diferentes sectores de la industria mejora considerablemente la productividad y la calidad de las empresas que deciden realizar una inversión en ellos. Si se programan adecuadamente, permiten realizar tareas que son repetitivas para un operario a una mayor velocidad y con una mayor precisión.

A pesar de esto, los robots industriales requieren del paro de la producción para poder programarse correctamente, lo que supone grandes pérdidas en tiempo, y por lo tanto dinero, para las empresas que emplean este tipo de robots.

Sin embargo, existen software de programación fuera de línea que permiten crear un entorno virtual en el que programar el robot para realizar las tareas deseadas, pudiendo añadir diferentes elementos al entorno como cintas transportadoras o paneles de seguridad. De esta forma, se consigue que la simulación sea lo más acorde posible a la realidad.

Los simuladores con los que cuentan algunos softwares pueden ser muy básicos, permitiendo solo la simulación 2D con funciones limitadas, y hay otros que son muy avanzados permitiendo realizar simulaciones en 3D con físicas complejas y entornos realistas.

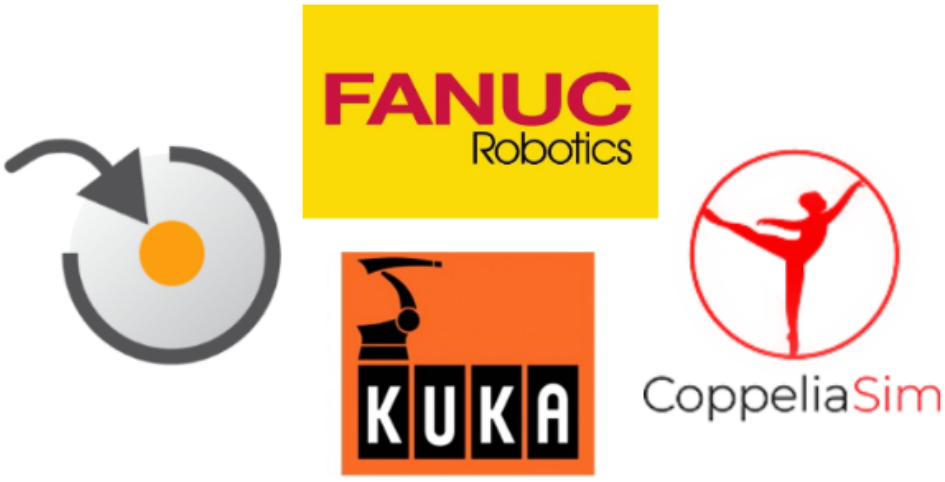

Figura 3. Principales paquetes de simulación. (Fuente: Elaboración propia).

Algunos de los principales paquetes de programación fuera de línea son:

- **RobotStudio**. Es el software de programación fuera de línea que ofrece la compañía ABB, el cual permite realizar simulaciones muy realistas gracias a que utiliza el mismo controlador que el software real de los robots en producción.
- **RoboGuide**. Es el simulador de robots que utiliza Fanuc, capaz de simular los movimientos de los robots y los comandos del programa, reduciendo considerablemente el tiempo dedicado a las configuraciones de movimiento.
- **KukaSim**. Es el programa empleado por la empresa KUKA para la programación fuera de línea de sus robots. Permite crear un gemelo digital del proceso de producción posterior con datos muy consistentes para que la unidad de control virtual y la real trabajen con la misma información para minimizar posibles fallos en la implantación.
- **CoppealianSim.** Es un software muy empleado para el desarrollo de algoritmos rápidos, simulaciones de automatización de fábricas, prototipado y verificación rápida y educación relacionada con la robótica. Este programa usa un entorno de desarrollo integrado basado en una arquitectura de control distribuido que permite que cada objeto se pueda controlar individualmente mediante un script incrustado.

2.3. RoboDK

Este apartado se va a centrar en el software usado en el Trabajo de Fin de Máster llamado RoboDK.

Figura 4. Logotipo RoboDK Software. (Fuente: RoboDK)

RoboDK Software fue creado en 2015 como resultado de una spin-off de un laboratorio pionero en robots industriales de la universidad École de Technologie Supérieure en Montreal, Canadá. Actualmente, la empresa está creciendo rápidamente y ayuda tanto a pequeñas empresas como a grandes multinacionales en tareas de simulación y programación de robots industriales.

El entorno de programación de RoboDK cuenta con una extensa librería de robots de más de 50 marcas, incluyendo a las más conocidas como KUKA, Fanuc, ABB, Universal Robots, etc. Además, en total su librería cuenta con más de 500 tipos de brazos robóticos de distintas características que permiten al usuario elegirlo en función de las características deseadas como la marca, los grados de libertad, los ejes, la carga soportada, el rango, la repetibilidad y el peso del robot. También incluye diferentes elementos típicos de una célula robotizada como pueden ser cintas transportadoras, vallas de seguridad, pedestales, mesas, cámaras y herramientas.

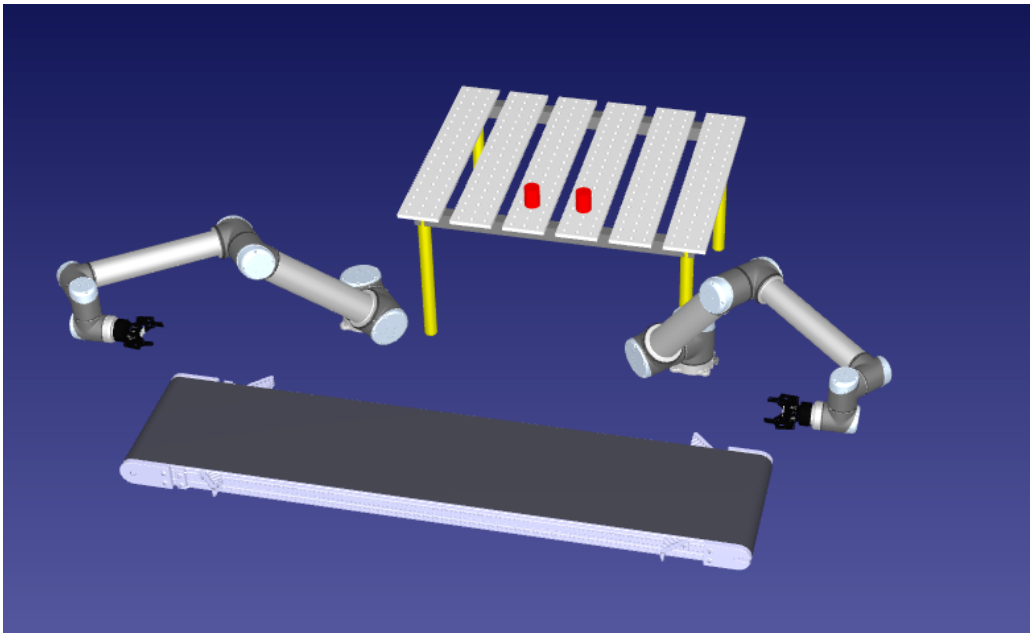

Figura 5. Ejemplo de pick & place con 2 robots y una cinta transportadora. (Fuente: RoboDK)

De esta forma, se consigue que el usuario que se disponga a diseñar una célula robotizada cuenta con la mayor oferta de robots industriales y pueda acercarse al máximo a la simulación real añadiendo elementos al entorno. Además, el software cuenta con la posibilidad de importar archivos de CAD personalizados para aquellos objetos que no se dispongan en la librería de RoboDK o aquellos que el usuario haya diseñado de manera propia.

En cuanto a la programación, RoboDK está diseñado para poder realizar una célula robotizada completa y funcional sin la necesidad de escribir ninguna línea de código. Esto se debe a que RoboDK apuesta por una forma de programación gráfica de manera que usando los distintos iconos que dispone en la barra superior de la interfaz del programa se pueden realizar distintas acciones como programar objetivos, movimientos lineales, articulares y circulares, esperas, cambios en señales digitales, etc.

Sin embargo, para aquellos usuarios más familiarizados con la programación convencional y para aquellos que desean realizar acciones más específicas, RoboDK cuenta con una API que permite realizar programas en lenguaje Python.

Al igual que los paquetes de simulación del apartado anterior, RoboDK cuenta con programación fuera de línea, permitiendo al usuario estudiar diferentes situaciones de una celda de trabajo antes de configurar la célula real. Esto permite eliminar la disminución de la producción debidas a la programación de los robots de planta, así como maximizar el retorno de la inversión realiza en los robots industriales, ya que los tiempos de la parada de producción pasan de ser de semanas a apenas un día.

```python
# Draw a hexagon around the Target 1
from robolink import *    # RoboDK's API
from robodk import *      # Math toolbox for rob
ots

# Start the RoboDK API:
RDK = Robolink()

# Get the robot (first robot found):
robot = RDK.Item('', ITEM_TYPE_ROBOT)

# Get the reference target by name:
target = RDK.Item('Target 1')
target_pose = target.Pose()
xyz_ref = target_pose.Pos()

# Move the robot to the reference point:
robot.MoveJ(target)
```

Figura 6. Ejemplo RoboDK programado con Python. (Fuente: RoboDK)

Además de las ventajas comentadas anteriormente como su extensa librería o su facilidad de programación gracias a si intuitiva interfaz, RoboDK cuenta con una ventaja muy importante y diferenciadora respecto a otros softwares y es el uso de post-procesadores. Gracias a ellos, se puede exportar el programa de cualquier robot de las marcas soportadas por el software al propio lenguaje del procesador real de robot.

De esta forma, se puede tener en una misma estación robots industriales de tres marcas diferentes, y se podrá programar la célula sin ninguna necesidad de saber como funciona cada uno de los lenguajes de programación que utiliza cada marca, ya que RoboDK se encargará de procesar el programa en función del procesador de cada robot.

2.4. Visión Artificial

En los últimos años, la visión artificial dentro del ámbito industrial es una tecnología que ha ido creciendo rápidamente, hasta el punto de volverse casi imprescindible para ciertos sectores. Los sistemas de visión artificial se encargan de adquirir, procesar y analizar imágenes del mundo físico, con la finalidad de adquirir información útil que pueda ser interpretada por un ordenador o por un operario para optimizar un proceso industrial.

Por lo tanto, se podría acoplar una cámara a un brazo robótico para que inspeccione un producto en busca de defectos para conseguir el mejor control de calidad posible, o también podría ayudar en aplicaciones de Pick & Place para agarrar un producto u otro en función de su color, tamaño, posición u otras características.

Figura 7. Brazo robótico con sistema de visión artificial integrado. (Fuente: Electrónica Edimar)

Si nos fijamos en el mercado actual de visión artificial en la industria, se pueden distinguir tres tipos de sistemas [4].

Sensores de visión

Supone una avance tecnológico con respecto a los convencionales sensores fotoeléctricos. Su principal función es detectar anomalías en una muestra, es decir, actúan como mecanismos de pasa o no pasa en una línea de producción. Destacan sobre otros sistemas de visión por su rápida instalación y su bajo coste.

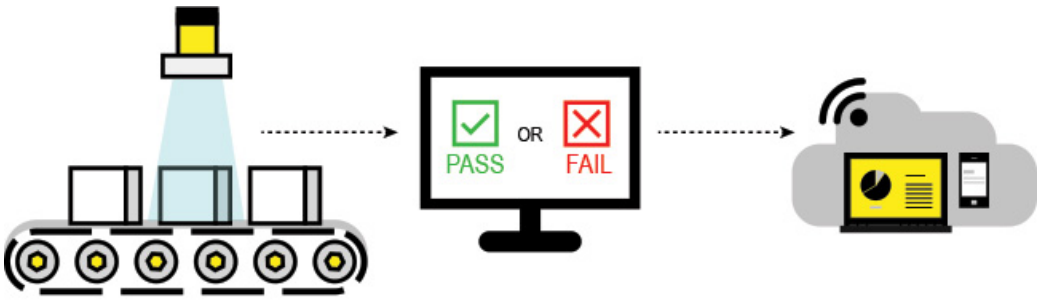

Figura 8. Esquema funcionamiento sensor visión. (Fuente: Cognex)

Cámaras inteligentes y sistemas de visión integrados

Esta tecnología supone un avance con respecto a los sensores de visión y destacan principalmente por su capacidad de cálculo, su fácil integración y la calidad de imagen. Todo ello permite la aplicación de esta tecnología en cualquier punto de la producción.

Además, gracias a su capacidad de procesamiento es capaz de almacenar datos y por lo tanto tiene la posibilidad de conectarse a otros sistemas automatizados gracias a las entradas y salidas de las que dispone.

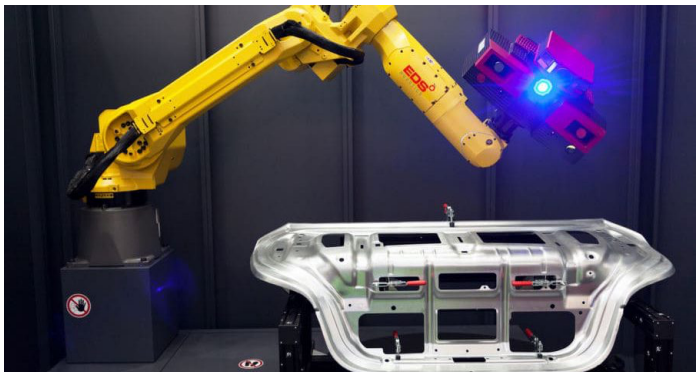

Figura 9. Cámara inteligente integrado en un brazo robótico para inspección. (Fuente: EDS Robotics)

Sistemas de visión avanzados

Este último tipo de sistema de visión industrial es muy parecido al anterior, pero con la diferencia de que su hardware es más complejo, lo cual le permite la optimización de aquellas funciones relacionadas con el procesamiento de la información obtenida a partir de la imagen.

Su principal aplicación es en máquinas que necesitan un mayor uso de recursos de datos, por lo que, en estos casos la integración del sistema de visión se realiza desde el momento en el que se fabrica la máquina.

Figura 10. Sistema de visión avanzado en una línea de producción. (Fuente: EDS Robotics)

Dentro de la industria, las aplicaciones de la visión artificial son innumerables, ya que ofrecen muchas ventajas al ser una tecnología muy variable y adaptable a cualquier necesidad de una línea de producción. Algunas de sus principales aplicaciones son:

- **Envasado y embalajes**. La inspección de la presencia o ausencia de etiquetas, números de lote o fechas de caducidad mediante visión artificial es muy usado en la industrial. También se pueden inspeccionar la correcta colocación de piezas como tapones en botellas, o incluso detectar si por ejemplo en una empresa de bebidas hay alguna botella que no se ha llenado correctamente.
- **Automoción.** Se emplea la visión artificial para la inspección en el proceso de fabricación de los componentes y para asegurar el correcto ensamblado de todas las piezas. Además de ello, en procesos como pintura, soldadura o mecanizado se emplean sistemas de visión para supervisar que el proceso se desarrolle adecuadamente.
- **Electrónica.** Gracias a los sistemas de visión, en los procesos de fabricación ce componentes electrónicos se puede supervisar las soldaduras realizadas por el robot o asegurar un correcto ensamblaje de los componentes.
- **Sector agroalimentario.** Los sistemas de visión en este sector son de gran utilidad, ya que permiten clasificar frutas y verduras en función de su punto de maduración (p.e. las naranjas con su color) o para asegurar que las latas de conserva se prensan correctamente sin presentar ninguna fuga.

Como se puede ver, la visión artificial es una tecnología muy útil para la industria ya que permite su aplicación en infinidad de sectores y que está en constante desarrollo. De hecho, en los últimos años se están implementando algoritmos de inteligencia artificial, para que a través de una red neuronal sea capaz de emular las funciones del ojo y el cerebro humano.

Actualmente, las empresas más importantes que se dedican al desarrollo e instalación de sistemas de visión artificial para el sector de la automatización son Cognex, Keyence, Sensopart, Infaimon y BCN Vision [4].

2.5. Sistemas de comunicación industriales

En la automatización de procesos industriales, los sistemas de comunicación industriales entre los distintos dispositivos que intervienen en un proceso son un aspecto clave, ya que permiten que exista un buen funcionamiento en cualquier parte de este.

Redes comunicación industrial

Una red de comunicación industrial es un sistema de comunicación entre varios dispositivos dentro del ámbito industrial, la cual esta diseñada para manejar grandes cantidades de información con un ancho de banda limitado.

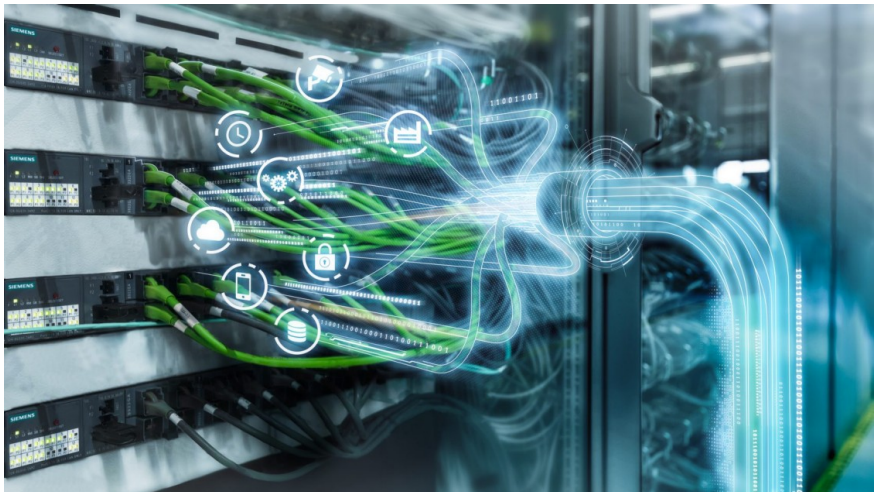

Figura 11. Concepto de redes comunicación industrial. (Fuente: Siemens)

Estas redes pueden usarse dentro de los sistemas de control para la transferencia de datos entre dispositivos de campo y PLCs, entre PLCs o entre PLCs y PCs en los que se muestran las interfaces de usuario que controlan los operarios. Los sistemas de control más utilizados en la arquitectura de la automatización industrial son los PLCs, los sistemas SCADA y los sistemas de control distribuido (DCS).

Para establecer las comunicaciones entre los distintos dispositivos se utilizan diversos protocolos de comunicaciones industriales. Estos protocolos son un conjunto de reglas que hacen posible la transferencia de información entre estos dispositivos. Los protocolos más usados dentro del ámbito industrial [5] son:

- **Ethernet.** Es el protocolo más usado en la industria y es ideal para ambientes hostiles. Permite ofrecer un sistema con una gran velocidad, adaptable y a prueba de fallos. Además, también permite conexión en tiempo real entre los dispositivos conectados.
- **Modbus**. Este protocolo fue creado por Modicon como una estructura de mensajería. Se usa para establecer comunicaciones cliente-servidor entre los dispositivos. Los dos más utilizados en la industria son el Modbus TCP/IP y el RTU.

- **Profinet.** Actualmente, es un protocolo muy usado ya que se basa en el protocolo Ethernet, pero más flexible, permitiendo personalizar dispositivos y programas. También soporte conexión en tiempo real y ofrece un sistema muy preciso y de gran calidad.
- **Profibus.** Este protocolo es un protocolo estándar abierto que permite transmitir datos a altas velocidades y se usa bastante en ambientes hostiles.

El uso de redes de comunicación industrial ofrece diversas ventajas: la reducción del cableado y su simplificación que implica planos más sencillos y menos costes, el uso de dispositivos inteligentes que permiten un mayor rendimiento y cuenta con más funciones y el uso del control distribuido, que permite aumentar el rendimiento y la fiabilidad del control de procesos.

Estructura cliente servidor

La estructura cliente servidor es una arquitectura de software que permite conseguir un procesamiento de la información de forma cooperativa, en la cual las tareas se reparten entre un servidor, que provee recursos y uno o varios clientes que reciben esos recursos [6].

Normalmente, debido a la carga computacional que soportan los servidores suelen contar con un hardware muy potente para poder procesar todas las peticiones de los clientes, a la vez que actúa como un depósito de datos almacenando toda la información necesaria.

Esta estructura esta muy presente en la actualidad en múltiples servicios informáticos, como por ejemplo el Internet, el correo electrónico o la conexión inalámbrica con impresora de papel.

El funcionamiento de esta forma de comunicación es sencillo: el servidor es el que tiene almacenada la información, entonces cuando un cliente quiere acceder a dicha información, se envía una petición al servidor. El servidor recibe la petición del cliente y genera una respuesta con la información solicitada si procede o con alguna advertencia o error por la cual no se ha podido enviar dicha información en el caso contrario.

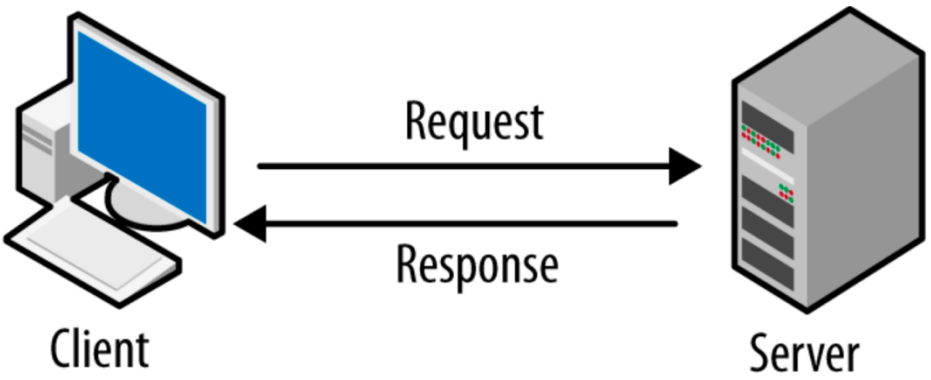

Figura 12. Funcionamiento estructura cliente servidor. (Fuente: OOSE)

CAPÍTULO III: DESARROLLO PRÁCTICO

En este capítulo, se desarrollará el trabajo realizado en el proyecto tanto de la simulación de la célula con RoboDK como la ejecución de esta en los robots del laboratorio.

3.1. Planteamiento general de la célula

En este apartado se va a explicar diferentes aspectos sobre la célula robotizada para el trabajo como su justificación, los elementos que la conforman y el funcionamiento de esta.

3.1.1. Justificación

Para poner en práctica los conocimientos adquiridos sobre la programación de robots industriales en la asignatura de Robótica Industrial y para aprovechar el potencial del software de RoboDK, se va a plantear una célula de trabajo que aproveche tres robots industriales con los que cuenta el Departamento de Ingeniería de Sistemas y Automática (DISA). En concreto el departamento cuenta con un robot UR3 y un UR3e de Universal Robots y con un IRB140 de ABB.

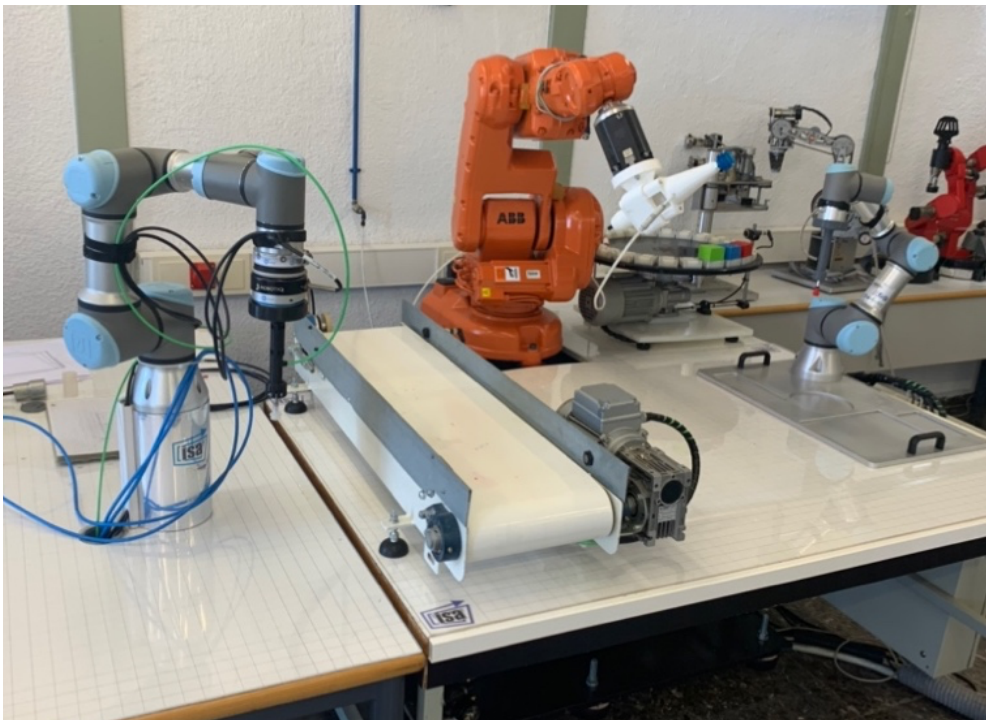

Figura 13. UR3, IRB140 y UR3e del laboratorio DISA de izquierda a derecha.

Para ello, se va a proponer realizar una célula de trabajo de una parte de un proceso productivo en la cual intervengan los tres robots, de forma que primero se programe en RoboDK para asegurar un correcto funcionamiento y por último se ejecute en el laboratorio.

La célula de trabajo se encargará de un proceso de mecanizado y paletizado de un objeto utilizando visión artificial y con comunicación TCP/IP entre uno de los robots y el

ordenador que procese los datos de la cámara. Sin embargo, para simplificar el trabajo el mecanizado no se realizará con una broca sino con un delineador para que quede ejemplificado el funcionamiento de la célula.

De esta forma, se podrán poner en práctica conocimientos relacionados con la programación en RoboDK, programación de robots reales, visión artificial y comunicación cliente-servidor.

3.1.2. Elementos

A continuación, se van a presentar los elementos necesarios para llevar a cabo la propuesta del apartado anterior, indicando cual es la utilidad dentro de la célula de cada uno de ellos.

Robot UR3

El primer robot con el que cuenta la estación es el UR3 de Universal Robots. Este robot colaborativo ultraligero y compacto es ideal para aplicaciones sobre mesa de trabajo, como en este caso. Aunque el robot solo pesa 11 kg, es capaz de trabajar con cargas de hasta 3 kg y cuenta con rotaciones ± 360º en todas sus articulaciones y una rotación infinita en su extremo.

Figura 14. Robot UR3 de Universal Robots del laboratorio de Robótica.

La función de este robot será clasificar los objetos que lleguen por una cinta transportadora en función de su color y pudiendo corregir el ángulo de desviación del objeto, gracias a una cámara situada en la parte superior de la célula.

Robot UR3e

El primer robot de la célula de trabajo es el UR3e de Universal Robots. Este robot tiene las mismas especificaciones que el UR3 con la diferencia de que cuenta con varias mejoras como un sensor de fuerza en el sexto eje, una nueva pantalla táctil capacitiva y con mayor resolución y por último mejoras para realizar la programación y la navegación por la pantalla mucho más sencillas.

Figura 15. Robot UR3e de Universal Robots del laboratorio de Robótica

La principal función de este robot será una tarea de pick & place desde el abastecimiento de las piezas de la mesa giratoria hasta la zona de trabajo del robot que realizará el mecanizado.

Robot IRB140

El IRB140 de ABB es un robot muy compacto y potente capaz de soportar cargas de 6 kg con un alcance de 810 mm. El robot está instalado sobre un pedestal que le permite estar a una altura adecuada con respecto los otros elementos de la célula.

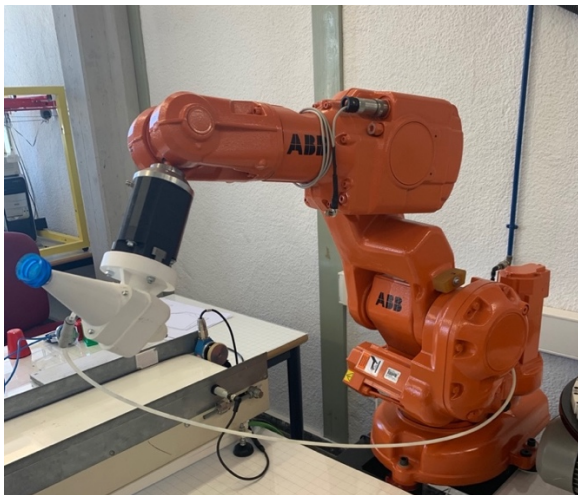

Figura 16. Robot IRB140 de ABB del laboratorio de Robótica.

Cuenta también con una protección frente al polvo y al agua de IP67, aunque para la aplicación que se va a realizar en el laboratorio no será necesario tanto nivel de protección, ya que no se va a realizar un mecanizado real que genere viruta u otro tipo de aplicación que genere residuos.

Este robot se encargará de realizar 2 tareas de pick & place y un mecanizado.

Mesa Giratoria

Esta mesa giratoria será el punto de abastecimiento de la célula. El movimiento giratorio de la mesa es producido por un motor eléctrico situado en la parte posterior de la misma. La mesa cuenta con 24 casillas repartidas equitativamente en las que hay una cavidad de 30 mm de diámetro y 10 mm de alto en el que se aloja la pieza manipulada.

Para el control de su posición la mesa cuenta con un sensor en la parte superior que detecta si hay pieza o no hay pieza, y otro sensor en la parte inferior que detecta el paso de cada casilla gracias a que en cada una de ellas sobresale un tornillo, el cual es detectado por el sensor.

Figura 17. Mesa giratoria vista desde arriba y desde el lateral.

La función de esta mesa será el abastecimiento de la célula, y su posición será controlada por el robot ABB IRB140 mediante los sensores descritos anteriormente.

Cinta transportadora

Para poder llevar los objetos desde la zona de trabajo del ABB IRB140 hasta la zona de trabajo del UR3 es necesario emplear la cinta transportadora con la que cuenta el laboratorio. Además, la cinta está en la posición óptima para que la cámara pueda tomar fotos de los objetos que hay en ella, y como la cinta es blanca, ayudará a mejorar el contraste de las fotos para obtener un mejor procesamiento.

Para su movimiento cuenta también con un motor eléctrico que le permite mover la cinta en dos direcciones.

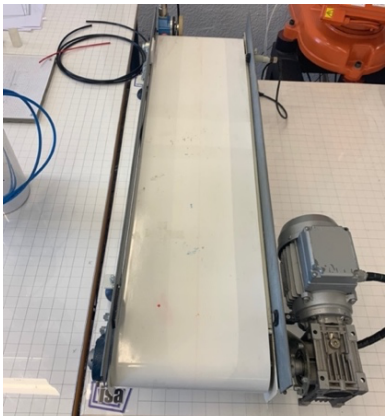

Figura 18. Cinta transportadora del laboratorio de Robótica.

Cámara

Una de las partes más importantes que incorpora esta célula es la cámara, ya que va a permitir procesar imágenes de los objetos y obtener información relevante de ellos.

En concreto, en esta célula la cámara se encargará de obtener una foto del objeto para posteriormente obtener la información relacionada con su color, para paletizarlo en su lugar correspondiente, y del ángulo, para poder corregir alguna posible desviación en la orientación del objeto.

La cámara del laboratorio de Robótica es el modelo Webcam Classic de la marca Hércules, y es capaz de tomar fotos de 1.3 Mpx de un tamaño de 640 x 480 píxeles. La conexión con el ordenador para poder procesar la imagen se realiza mediante un cable USB 2.0.

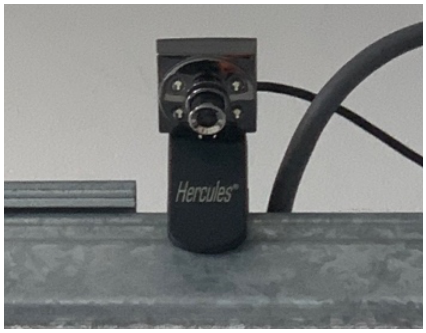

Figura 19. Cámara Hércules del laboratorio de Robótica.

Pieza manipulada

El objeto que se va a utilizar como producto a procesar en la célula robotizada será un cubo de 43 x 43 x 40 cm con un cilindro de 27.5 cm de diámetro y 10 cm de alto en la parte posterior del mismo. Gracias a estas dimensiones, las piezas encajas perfectamente en las casillas de la mesa giratoria.

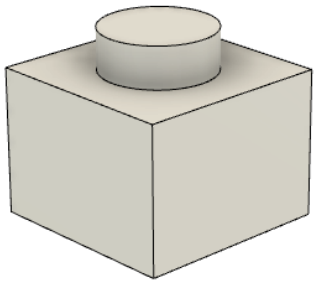

Figura 20. Pieza manipulada

El material de este objeto será PLA ya que va a ser fabricado mediante una impresión 3D. Además, como se ha marcado el objetivo de paletizar estas piezas en función de su color, se va a emplear PLA azul, verde y rojo. El motivo de la elección de estos colores es porque son los colores primarios y va a ser más fácil de distinguir que otra elección de colores.

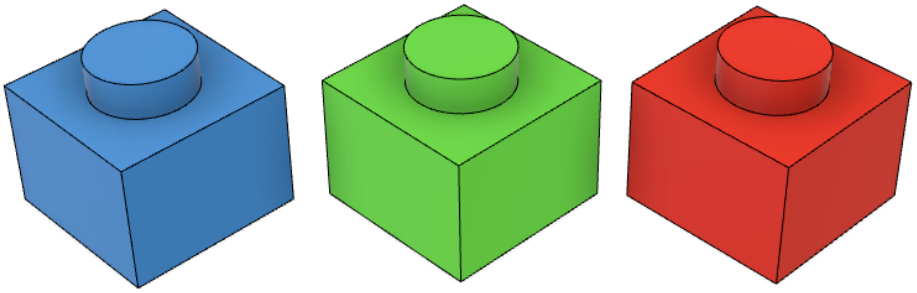

Figura 21. Piezas manipuladas en color azul, verde y rojo.

Originalmente la célula estaba pensada para paletizar cuatros piezas de cada color a una misma cota. Sin embargo, se ha propuesto una mejora al diseño inicial que permitiría apilar las piezas verticalmente de forma que el resultado final sea más compacto.

Para ello, habría que realizar una cavidad cilíndrica en la parte inferior de la pieza de 28.5 mm de diámetro, dejando una holgura de 1 mm, y de 10 mm de alto, para permitir albergar todo el saliente en dicha cavidad.

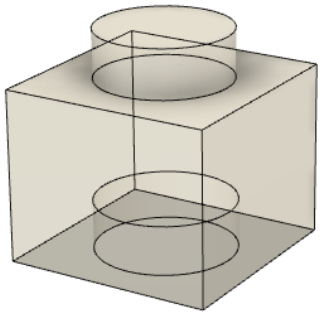

Figura 22. Rediseño de la pieza manipulada.

Sin embargo, debido a que la inclusión de este rediseño supondría más costes para el proyecto, se ha decidido incluir exclusivamente en simulación y en el laboratorio usar las piezas con el diseño original.

Herramienta Doble Ventosa-Delineador

Para que el robot ABB IRB140 pudiese manipular la pieza, moviéndola entre su zona de trabajo y la cinta transportadora y que además pudiera realizar el mecanizado con el delineador, se ha diseñado una doble herramienta que permita poder realizar ambas tareas sin necesidad de un cambio de herramienta o un segundo robot o mecanismo.

Aprovechando los diseños existentes de la pieza que soporta la ventosa y la pieza que soporta el delineador, se ha creado una tercera pieza que servirá de unión de ambas piezas de forma que queden separadas 180º. Gracias a este diseño (véase Figura 23) cuando se quiera cambiar de herramienta el robot simplemente tendrá que hacer un giro de 180 grados sin necesidad de mecanismos más complejos.

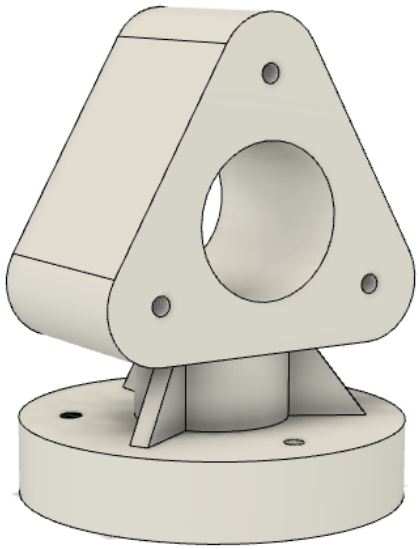

Figura 23. Diseño doble herramienta.

La pieza consta de una parte cilíndrica que iría adjunta a la brida del robot, con 3 agujeros para tornillería M6 unidos a una parte triangular mediante otra pieza cilíndrica más estrecha. En la unión de ambas piezas se ha decidido introducir unos nervios que permitan aumentar la resistencia de dicha unión. La parte triangular cuenta con 3 agujeros para tornillería M6 y un agujero más grande cuya función principal es aligerar el peso de la pieza y reducir en costes de material y energía. Esta pieza va a ser impresa en 3D, debido a su complejo diseño, con PLA blanco.

El resultado final de la pieza diseñada con los dos soportes se muestra a continuación.

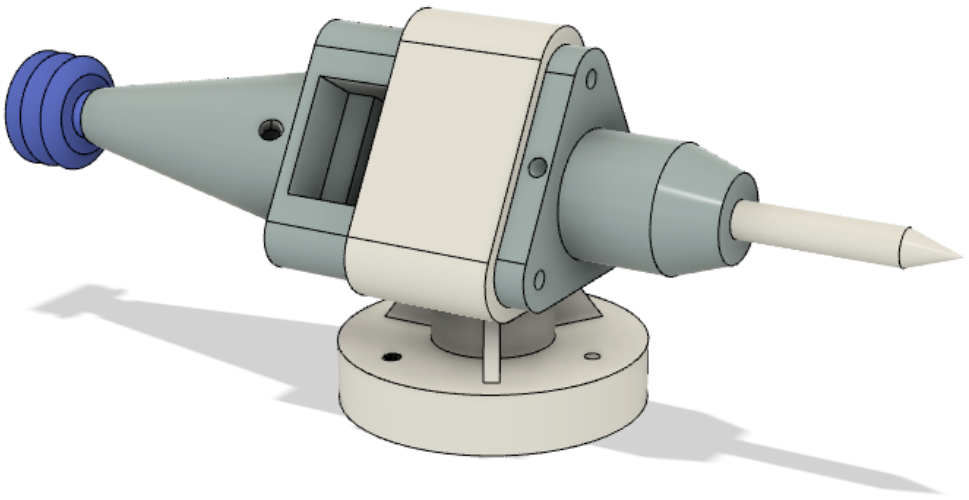

Figura 24. Ensamblaje ventosa, delineador y soporte.

3.1.3. Funcionamiento

Una vez explicadas la justificación de la célula y los elementos que la conforman, se va a explicar la propuesta de funcionamiento del flujo de trabajo.

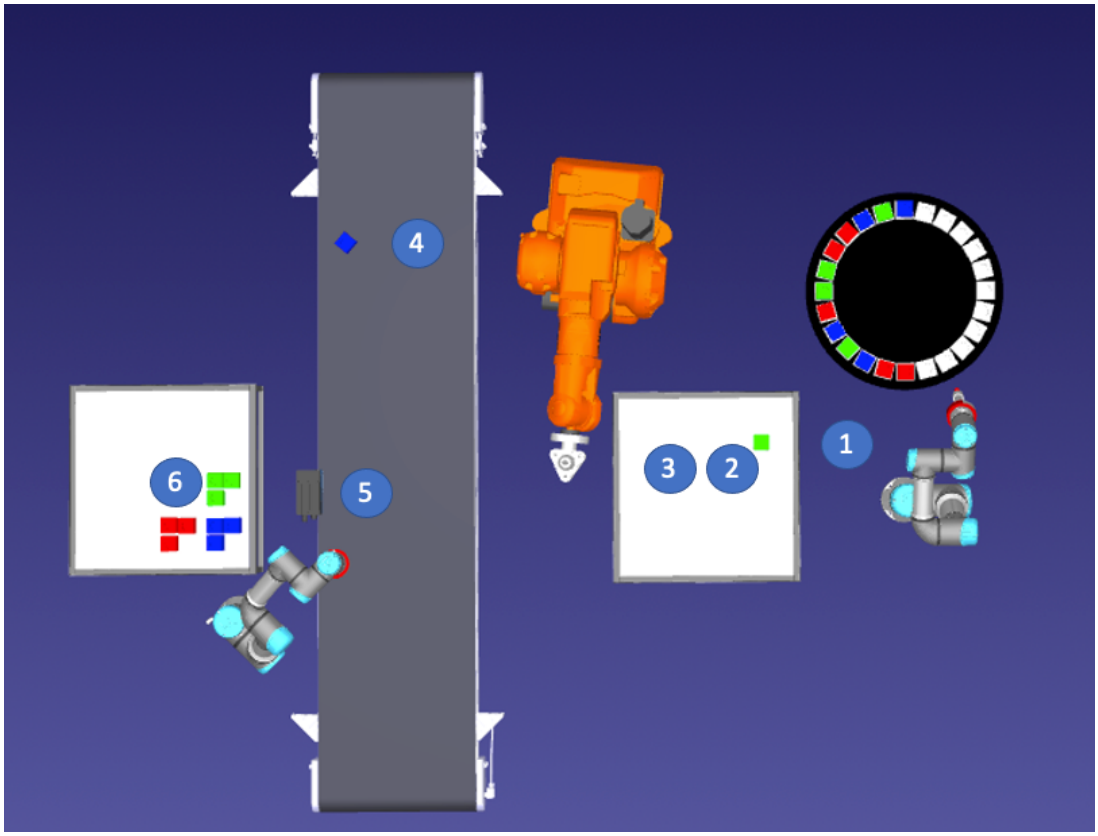

Figura 25. Esquema funcionamiento célula robotizada por etapas.

Se pueden distinguir 5 etapas distintas en el flujo de trabajo de la célula.

1. **Pick & Place UR3e.** En esta etapa el UR3e se encargará de coger las piezas existentes en la mesa giratoria y dejarlas en la zona de trabajo del ABB IRB140.
2. **Pick & Place IRB140 (I).** Una vez en la zona de trabajo, el robot se encargará de situar la pieza en una posición más favorable para realizar el mecanizado, poniendo la doble herramienta con la ventosa apuntando hacia abajo.
3. **Mecanizado pieza IRB140.** Con la pieza en la posición adecuada, el robot cambiará la orientación de la muñeca para usar el delineador, que se encargará de realizar un mecanizado en los contornos superiores de la pieza.
4. **Pick & Place IRB140 (II).** Una vez se haya realizado el mecanizado de la pieza, el robot pondrá la herramienta en la posición de ventosa para realizar una tarea de pick & place que situé la pieza sobre la cinta transportadora. Además, en esta parte final, se ha desarrollado un algoritmo que permita al robot dejar la pieza en un ángulo aleatorio para que luego pueda ser corregido mediante visión artificial por el robot UR3.

5. **Procesado de la imagen.** Cuando la pieza se sitúe en la cinta, esta avanzará hasta llegar al final, donde se realizará una fotografía de la pieza. Esta fotografía será procesada por algoritmos de visión artificial para obtener el color de la pieza y el ángulo de desviación de esta. Una vez obtenida dicha información, se envía al robot UR3 mediante un socket.
6. **Paletizado UR3.** Con la información de la cámara, el robot UR3 realizará la tarea de paletizado en la que en función del color irá agrupando bloques de 4 piezas. Además, si existe algún tipo de desviación en el ángulo de la pieza este será corregido por el robot para tener un paletizado compacto.

3.2. Planteamiento célula RoboDK

Con la propuesta de la célula ya desarrollada, el siguiente paso es configurar la célula en simulación para ver si es viable realizar el proceso en el laboratorio. Por lo tanto, en este apartado se va a desarrollar como se ha implementado la propuesta de célula robotizada en el software de RoboDK y como se ha realizado la programación de los algoritmos de visión y las comunicaciones cliente-servidor.

3.2.1. Programa en RoboDK

Como se había explicado previamente en el desarrollo teórico, RoboDK permite programar de manera gráfica, rápida y muy intuitiva cualquier robot de las marcas más importantes del mercado, como pueden ser ABB, KUKA, Fanuc, Universal Robots, etc.

Para explicar como se ha realizado el programa en RoboDK, en este apartado se va a tratar como funcionan diferentes aspectos del software, como la interfaz principal del programa, los sistemas de referencia y objetos, la cámara virtual, los programas realizados mediante RoboDK, los programas realizados mediante Python y la exportación de los programas mediante post-procesador

La principal fuente para el aprendizaje del programa ha sido leer la documentación que RoboDK recoge en su página web [7].

Interfaz Principal

Antes de comenzar a enseñar como funciona RoboDK, es necesario conocer su interfaz principal y como se estructura. Para ello, se muestra la interfaz principal de un ejemplo creado por RoboDK de una soldadura a una pieza metálica con un ABB IRB 2600.

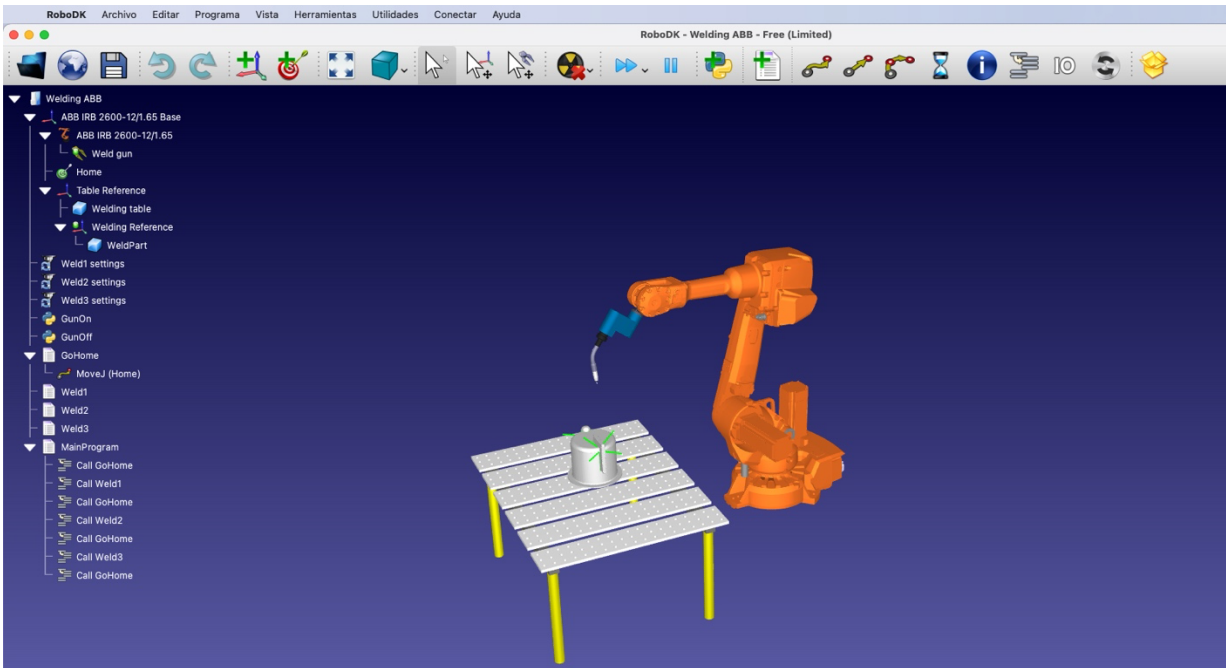

Figura 26. Interfaz Principal RoboDK.

Para que quede más claro las secciones principales de las que se compone la interfaz principal, se ha creado una figura en la que se pueden ver cuáles son dichas secciones.

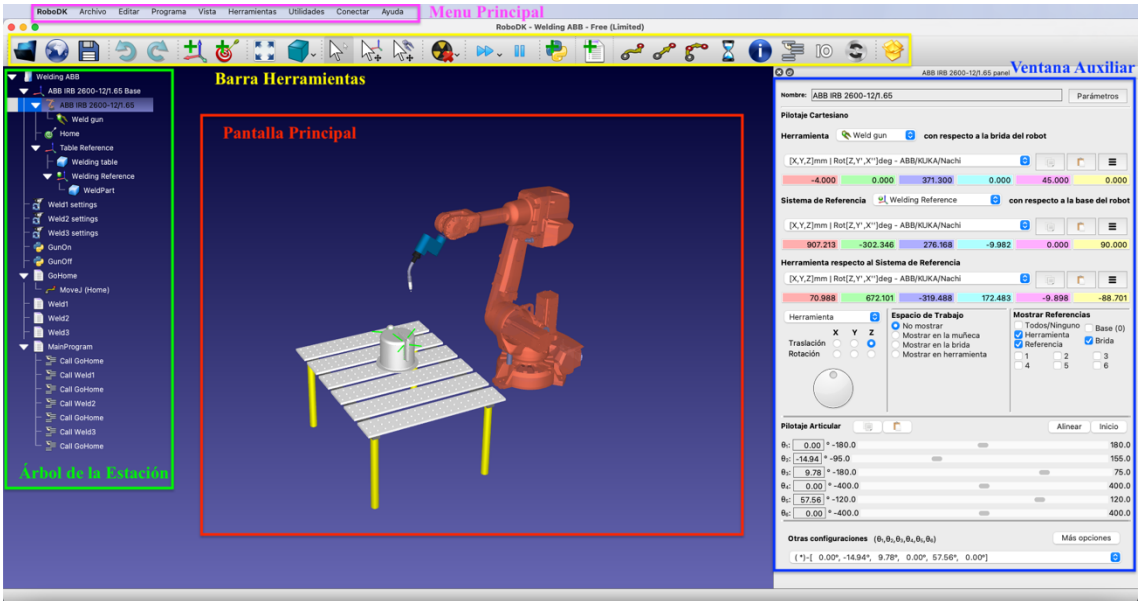

Figura 27. Distribución secciones Interfaz Principal RoboDK.

La interfaz principal de RoboDK se divide en las siguientes secciones:

- **Menú Principal.** Está situado en la parte superior de la interfaz, y en el se muestran diferentes secciones desplegables para interactuar con el programa, en las cuales se encuentran todas las acciones y opciones disponibles.
- **Barra de Herramientas.** Permite acceder rápidamente a las acciones más usadas del menú, como abrir y guardar un archivo, crear sistemas de referencia, puntos, objetos y trayectorias o incluso crear un programa que están representados mediante iconos gráficos.
- **Árbol de la Estación.** En esta parte de la interfaz se encuentran todos los elementos que se encuentran en la estación. Estos elementos pueden ser robots, posiciones, sistemas de referencia, mecanismos, herramientas, programas, etc. El sistema en árbol permite unir elementos a otros, de forma que se pueden adjuntar las posiciones de un robot al mismo, la herramienta a un robot u objetos a otros, como por ejemplo una mesa con piezas encima de ella.
- **Pantalla Principal.** Muestra los elementos del árbol y es la vista principal a la hora de configurar la estación.
- **Ventana Auxiliar.** Al hacer doble clic sobre cualquier elemento de la estación, dependiendo del tipo de elemento que sea aparecerá una ventana auxiliar en la que configurar diferentes parámetros de dicho elemento. Por ejemplo, en el caso de un robot se puede ver su sistema de referencia de y el de la herramienta y, su espacio de trabajo, su configuración, e incluso se puede ver y mover sus articulaciones.

Sistemas de referencia y Objetos

Una vez que se sabe como se estructura la interfaz principal, se va a explicar como se ha desarrollado la célula. En primer lugar, a la hora de crear la estación de trabajo, se deberán definir los sistemas de referencia de los objetos que se vayan a añadir para que, si se necesita realizar algún desplazamiento o giro de un objeto, se tenga una referencia inmóvil del mismo.

En el caso de este trabajo, se han creado un sistema de referencia para cada uno de los tres robots y otro para los objetos de la célula como pueden ser las mesas, la cinta o la cámara y por último un sistema de referencia para la mesa giratoria que alberga las piezas manipuladas. En total se cuenta con 5 sistemas de referencia.

Figura 28. Sistemas de Referencia RoboDK.

RoboDK funciona con estructuras de tipo árbol, por lo que dentro del sistemas de referencia se puede añadir nuevos objetos u otros sistemas referencias como si fueran ramas de un árbol que se van desplegando.

Una vez que hemos definido los sistemas de referencia, se debe añadir los objetos necesarios a cada uno de ellos para configurar la célula. De forma que *World* albergará las dos mesas, la cinta transportadora y un sistema de referencia de la cámara.

Figura 29. Despliegue del sistema de referencia World.

Como se puede observar, se pueden distinguir los objetos estáticos Mesa 1 y Mesa 2 indicados por un bloque azul, el sistema de referencia de la cinta transportadora *Conveyor Belt Base,* el sistema de referencia de la cámara *Camara Reference* en el que esta adjuntado el objeto cámara. Además, si se observa el sistema de referencia de la cinta transportadora, hay un icono de unos engranajes representado un mecanismo, que ha sido creado para que la cinta pueda desplazarse, y también tres objetos con un icono de una diana atravesada por una flecha que indican que son objetivos o posiciones del robot. RoboDK utiliza estos objetivos para definir los puntos por los cuales un robot o mecanismo se puede mover. En este caso hay una posición inicial donde el ABB IRB140 deja la pieza, una posición para que la cámara tome una foto y una posición final donde el UR3 pueda recoger la pieza.

A continuación, se presenta el despliegue del sistema de referencia de la mesa giratoria.

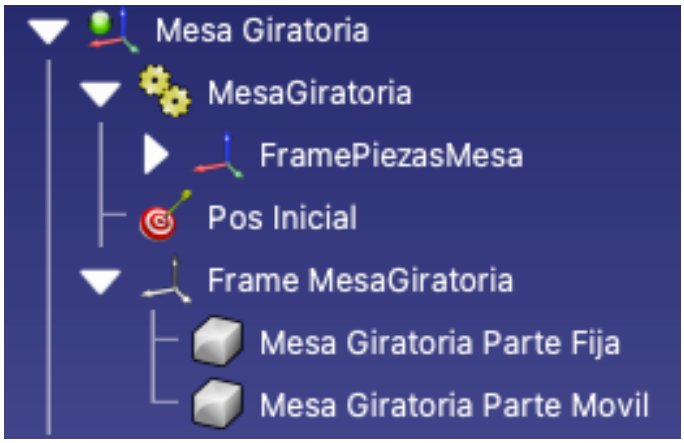

Figura 30. Despliegue sistema referencia Mesa Giratoria,

En este caso se observa que hay un mecanismo *MesaGiratoria* el cual va a permitir que la parte móvil de la mesa pueda girar con respecto a la parte fija. También hay un sistema de referencia *FramePiezasMesa* que va a albergar todas las piezas que inicialmente hay en la mesa.

Finalmente se encuentra un objetivo que marca la posición inicial de la mesa y un sistema de referencia oculto en el que están como objetos estáticos la parte fija y móvil de la mesa que se han utilizado para crear el mecanismo.

A continuación, se muestran a continuación los sistemas de referencia de cada uno de los tres robots.

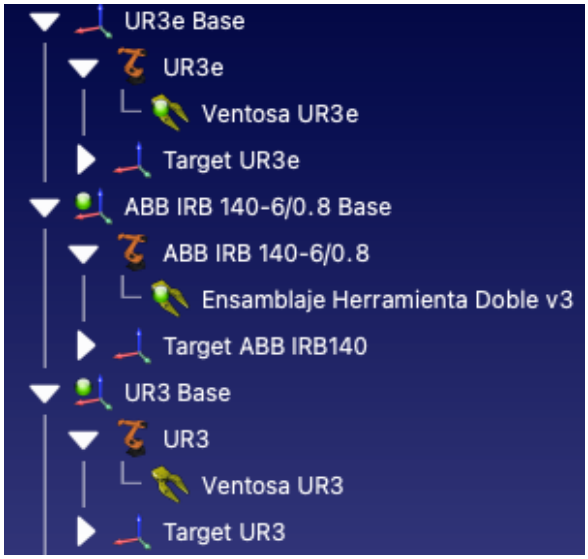

Figura 31. Despliegue sistema referencias de los tres robots.

Cada uno de los sistemas de referencia contiene el propio robot, al cual está adjunto su herramienta correspondiente, y un sistema de referencia en el que se almacenan todos los objetivos/posiciones del robot.

Con todo ello se puede configurar la célula y el resultado final de todos los objetos correctamente situados en la pantalla principal es el siguiente.

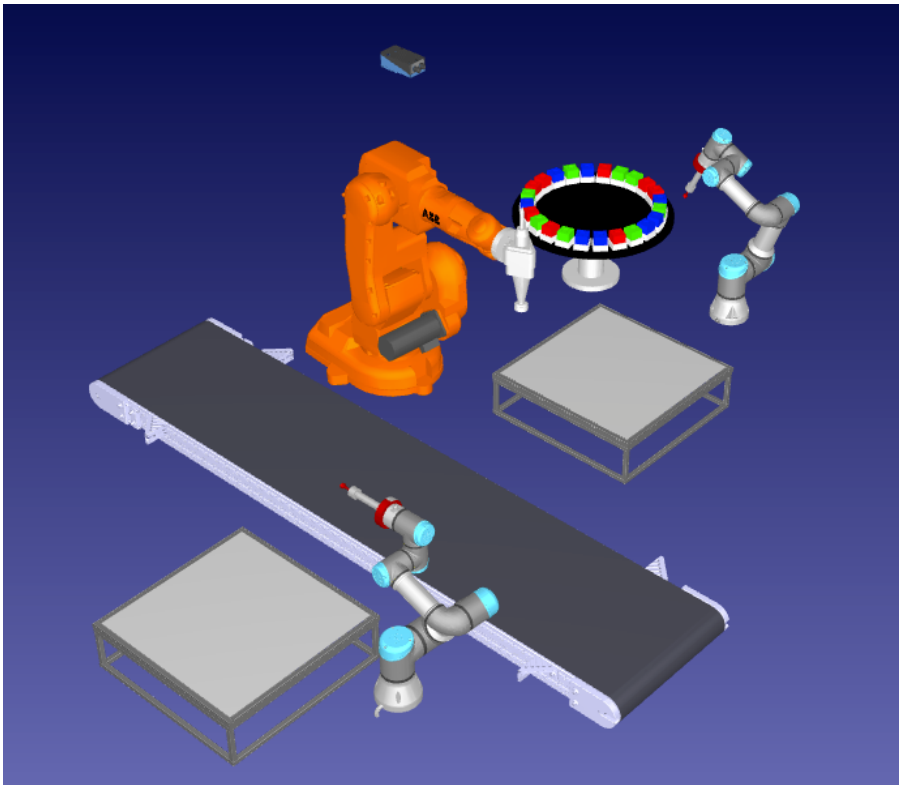

Figura 32. Objetos y robots en posición en RoboDK.

Mecanismos

RoboDK permite también la creación de mecanismos en la estación de trabajo para crear movimientos lineales o rotativos sobre objetos existentes en la estación. Para crear un mecanismo deberemos irnos a la barra de menús y en el apartado de Utilidades seleccionar Modelizar Mecanismo o Robot. Al seleccionar esta opción aparecerá la siguiente ventana.

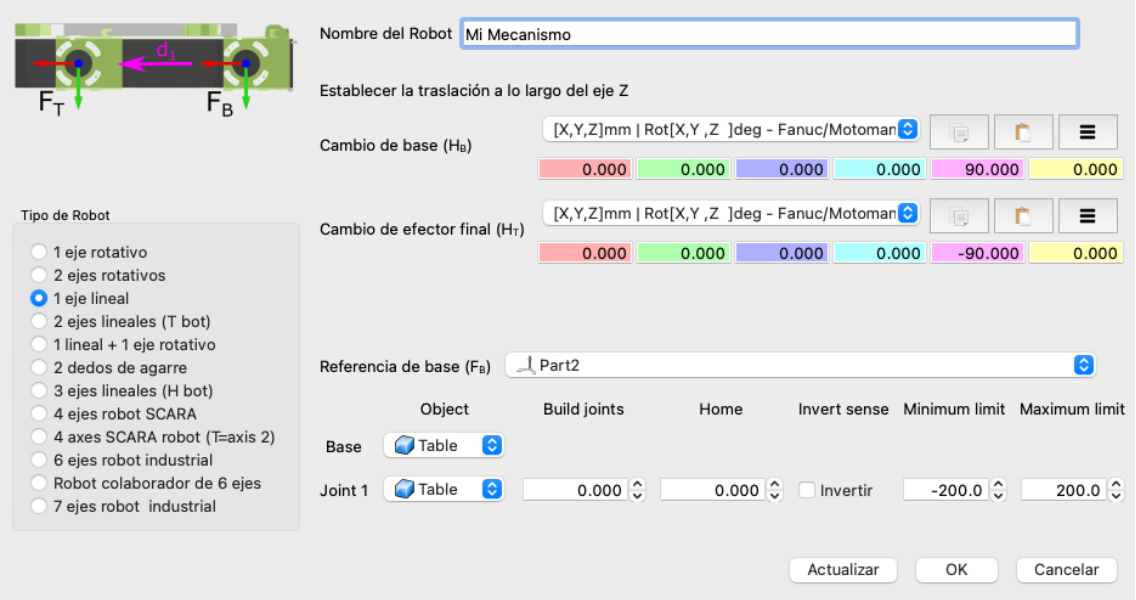

Figura 33. Ventana configuración mecanismos RoboDK.

En este ventana se puede configurar el mecanismo al gusto del usuario. Se puede configurar el tipo de robot o mecanismo definiendo su número de ejes y de que tipos son, se puede establecer la traslación a través del eje Z y definir el sistema de referencia.

Para configurar el mecanismo se selecciona un objeto como base, que será inmóvil, y otro como junta, que permitirá el movimiento. También se define los limites del rango de movimiento, la posición de origen y se tiene la posibilidad también de invertir los sentidos.

Con esta opción, se han dos mecanismos para la célula: la mesa giratoria, que ha sido configurada como un mecanismo de 1 eje rotativo con un rango de 0º a 360º y la cinta transportadora, que ha sido configurada como un mecanismo de 1 eje lineal con un rango desde -500000 mm hasta 500000 mm. Dicho rango se ha establecido de forma tan amplia para poder contar con un movimiento constante de la cinta.

De esta forma tan sencilla, se pueden crear mecanismos en RoboDK en tan solo unos minutos.

Cámara virtual

El software nos permite crear una cámara virtual capaz de tomar imágenes de la estación que se va a desarrollar. Además, RoboDK nos permite ajustar diversos parámetros de la cámara como la distancia focal, el campo de visión, la distancia de trabajo, el tamaño del píxel y el tamaño del sensor de la cámara.

Por lo tanto, en la célula se va a situar una cámara virtual justo encima de la cinta, tal y como esta en el laboratorio, intentando que el sensor de la cámara solo capture la imagen de la cinta para no tener que realizar ningún tipo de post-procesado a la imagen. Los parámetros seleccionados para la cámara se presentan en la siguiente tabla.

Longitud focal (mm)	**5**
Tamaño píxel (µm)	1.727
Campo de visión (grados)	9.48
Distancia de trabajo (mm)	1750
Tamaño sensor cámara (píxeles)	640 x 480

Tabla 2. Parámetros cámara virtual RoboDK.

Situando la cámara a una altura de 1.4 metros con respecto a la cinta transportadora, se obtiene la siguiente imagen en la posición de procesamiento de imagen de la cinta.

Figura 34. Imagen capturada por la cámara virtual en la posición de procesamiento de imagen de la cinta transportadora.

Esta imagen será procesada por un programa que incorpora los algoritmos de visión artificial necesarios para obtener la información del color y el ángulo de la pieza.

Programas RoboDK

Una vez definidos todos los elementos de la estación, el siguiente paso es programar los robots para que realizar las tareas deseadas. En esta sección se va a explicar la función principal de los programas realizados mediante el sistema de programación que propone RoboDK sin escribir ninguna línea de código.

Antes de hablar sobre los programas que se han utilizado para configurar la célula robotizada, se van a explicar como se crear un programa en RoboDK. Para crear en un programa se pulsa sobre el icono de creación de programa y esté aparecerá en el árbol de la estación. Una vez creado, para programar hay que añadir instrucciones sobre lo que se desea realizar. RoboDK ofrece una gran variedad de instrucciones que abarcan las necesidades principales de programación, las cuales se muestran en la siguiente tabla.

Icono	*Nombre*	*Descripción*
	Movimiento Articular	Genera un movimiento articular al objetivo seleccionado.
	Movimiento Lineal	Genera un movimiento lineal al objetivo seleccionado.
	Movimiento Circular	Genera un movimiento circular al objetivo seleccionado.
	Establecer Velocidad	Establece la velocidad a la cual se moverá el robot.
	Mostrar Mensaje	Muestra un mensaje personalizado a través de una ventana emergente.
	Pausa	Realiza una pausa en el programa.
	Llamado a Programa	Permite llamar o iniciar otro programa. En la llamada se espera a que el otro programa finalice, mientras que al iniciar se ejecutan paralelamente.
	Establecer o esperar instrucciones de I/0	Permite resetar o setear señales digitales y realizar esperas al cambio de dichas señales.
	Cambio sistemas de Coordenadas	Establece un nuevo sistema de coordenadas en el robot del programa.
	Cambio de Herramienta	Establece una nueva herramienta en el robot del programa.
	Simulación de Evento	Permite adjuntar o separar a una herramienta, mostrar o ocultar y establecer una posición de origen a los objetos de la célula.

Tabla 3. Instrucciones de programación en RoboDK.

Usando estas instrucciones, se puede crear cualquier programa básico que se use en una célula robotizada.

A continuación, se va a explicar la función de cada uno de los programas creados mediante RoboDK con los que cuenta la célula robotizada.

- **UR3e Programa**. En este programa el robot UR3e realizará la tarea de pick & place de las piezas que están en la mesa giratoria a la mesa de trabajo del ABB IRB140.
- **ABB Parte 1.** El robot ABB IRB140, en primer lugar, realiza la tarea de pick & place para dejar la pieza en posición de trabajo, luego llama al programa *Mecanizado*, vuelve a coger la pieza y finalmente llama al programa *Ángulo Random ABB* para que sitúe la pieza en una orientación aleatoria que posteriormente será detectada por la cámara.
- **Mecanizado.** Este programa es el encargado de realizar el mecanizado de la pieza con el delineador. Para ello, se definen 4 puntos para que el robot pueda mecanizar los contornos de la pieza. Se ha programado aparte ya que si se quieren añadir nuevos recorridos de mecanizados solo habría que cambiar este programa.
- **MoveConveyor.** Mueve la cinta transportadora desde el inicio hasta el procesamiento de la imagen.
- **ConveyorCI.** Mueve la cinta transportadora hasta sus condiciones iniciales, es decir, a la posición de inicio.
- **Reemplazar objetos.** Este programa es de vital importancia a la hora de programar la estación y realizar pruebas, ya que permite devolver todos los objetos a su posición inicial, mover los mecanismos y robots a sus posiciones iniciales y reiniciar las señales digitales.
- **MainProgram.** Con este programa se puede iniciar la simulación de la célula robotizada ya que llama a los programas necesarios para que todo se ejecute correctamente.

Programas Python

A la hora de programar la célula, para realizar ciertas tareas la programación que nos plantea RoboDK era un tanto limitante. Sin embargo, el software permite instalar una API que nos permite ejecutar programas escritos en Python y gracias a la librería Robolink se puede enlazar elementos de la estación como robots, mecanismos, objetos y programas a Python como si fuesen objetos.

Para aclarar el funcionamiento de Python, se propondrá un ejemplo en el cual se quiera mover un robot a una cierta posición, pero usando código. En primer lugar, se deberá importar la librería y crear un objeto de la clase Robolink que sirva como enlace de la estación.

```
from robolink import *        # importación librería robolink
RDK = Robolink()              # se establece un enlace con la estación
```

A continuación, se enlaza el robot con su posición objetivo y se define el valor de las articulaciones en el momento inicial.

```
robot= RDK.Item('UR3e')             # se enlaza el robot a un objeto
robot.setJoints([0,0,0,0,0,0])      # se ponen a cero las articulaciones
target = RDK.Item('Target')         # se enlaza el objetivo
```

Una vez que se ha enlazado los objetos necesarios con la estación ya se puede mover el robot. Para ello se emplea el siguiente comando.

```
robot.MoveJ(target)           # mueve el robot al objetivo indicado
```

Si se quisiera que el robot se trasladase respecto al objetivo definido, se crearía un nuevo objetivo a partir del existen y se desplazaría el robot a dicha posición. Pero para ello, antes se debería importar la librería RoboDK en la que hay una función llamada *Pose* que permite obtener los valores de la posición de un objetivo, para poder aplicarle una traslación a dicho objetivo.

```
from robodk import *
new_target = target.Pose()*transl(20,0,100)
robot.MoveL(new_target)
```

De esta forma, queda ejemplificado el funcionamiento de la API de Python para realizar programas con el entorno de RoboDK.

A continuación, se presentan los programas que escritos en Python usando para programar la estación.

- **MoveMesaGiratoria.** Permite mover la mesa 15º en sentido antihorario cada vez que el robot UR3e deposita una pieza en la zona de trabajo del robot ABB IRB140. El motivo de mover 15º la mesa es porque hay 24 casillas, y si se divide 360º entre el numero de casillas se obtiene que el ángulo entre casilla y casilla es de 15º.
- **CIRobots**. Este programa es llamado por el programa de *"Reemplazar objetos"* y permite mover los 3 robots a sus posiciones iniciales.
- **StopALL.** Detiene todos los programas en ejecución.
- **Angulo Random ABB.** Permite generar un número aleatorio entre 0 y 90 para que cuando el robot llegue a la posición de place en la cinta transportadora, aplique una rotación sobre la pieza para posteriormente ser detectada por la cámara y corregida por el UR3.
- **FinCicloUR3e**. Permite saber al UR3e cuando no hay más piezas para procesar para que finalice la ejecución del programa *"UR3e Programa"*.
- **FinCicloABB.** Permite saber al ABB IRB140 cuando no hay más piezas para procesar para que finalice la ejecución del programa *"ABB Parte 1"*.
- **ProcesadoImagen.** Este programa se encarga de procesar la imagen obtenida por la cámara y obtener el ángulo y color de la pieza. En una versión más avanzada del trabajo se incluyó dentro del código del servidor para facilitar el tratamiento de la información.
- **ServidorVA.** Crea una conexión TCP/IP que conecta la información del procesamiento de la imagen con el robot UR3.
- **PaletizadoUR3.** Este programa paletiza las piezas que llegan por la cinta con la información recibida del servidor en función de su color, y si es necesario corrige la orientación.

Post-procesadores

Una de las principales ventajas de RoboDK es el uso de post-procesadores para generar programas para cualquiera de los 40 fabricantes que soporta el software. El post-procesador define como los programas del robot deben ser generados para el tipo de controlador seleccionado.

Aunque RoboDK incluye más de 70 post-procesadores con funcionalidad total, el software permite la creación de un post-procesador personalizado o modificar alguno existente,

Para ejemplificar como funcionan los post-procesadores en RoboDK se va a crear una aplicación sencilla de pick & place en RoboDK y se va a exportar el programa usando post-procesadores al lenguaje de ABB y al lenguaje de Universal Robots.

El punto de partida será una estación sencilla con un robot de Doosan Robotics, una mesa y una pieza cúbica sobre la mesa.

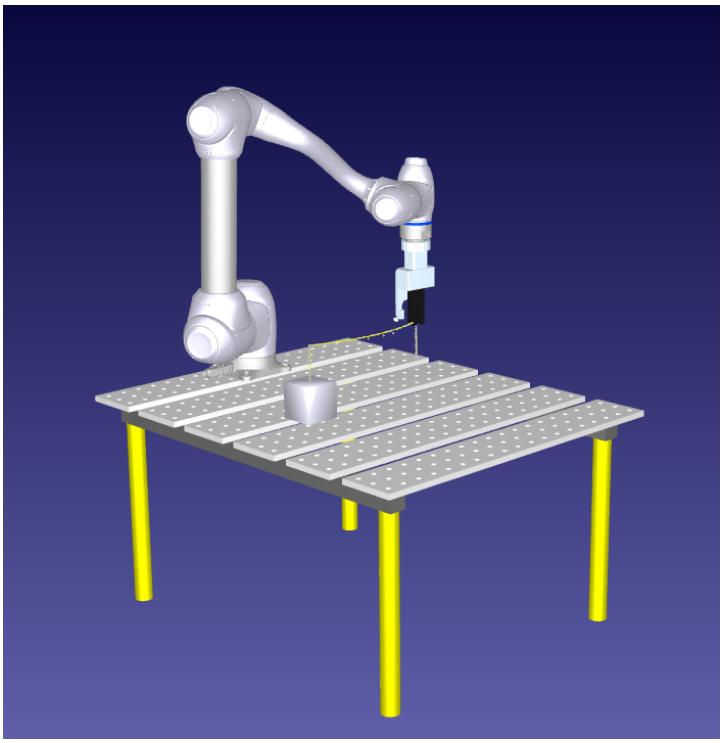

Figura 35. Estación Ejemplo Post-Procesadores.

Una vez añadidos los elementos a la estación, se crean los sistemas de referencia y se programan las posiciones del robot.

A continuación, se crea un programa y se añaden las instrucciones necesarias para crear una tarea de aplicación básica, en la que se establece el sistema de referencia, se define la herramienta que se va a usar, se establece la velocidad de movimiento y se añade los movimientos articulares y lineales necesarios.

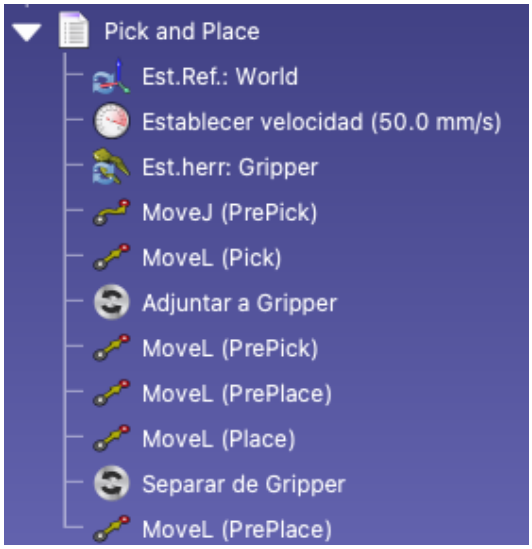

Figura 36. Programa pick & place Ejemplo Post-Procesadores.

Cuando el programar funciona adecuadamente en simulación, se puede generar el código para el robot deseado seleccionando su post-procesador. Para ello, en la interfaz de RoboDK se pulsa clic derecho sobre el programa y se selecciona la opción Seleccionar Post-Procesador. Esto abrirá una ventana en la que poder elegir el que se desee, que en este caso será el de Universal Robots.

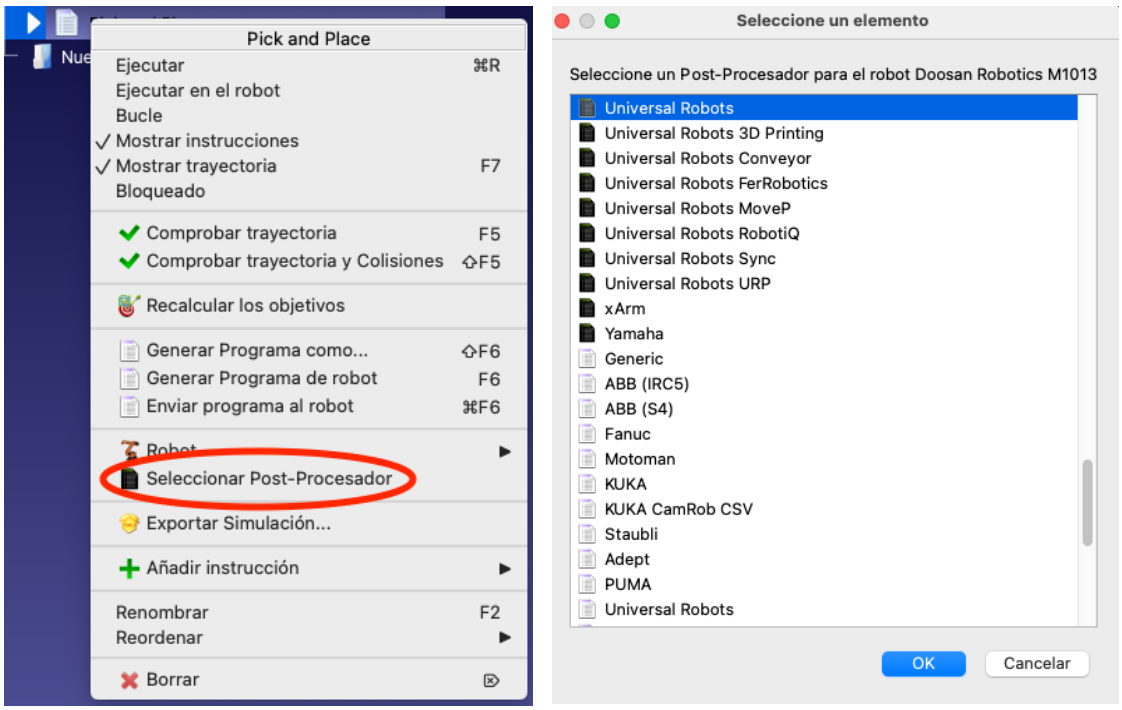

Figura 37. Navegación al Selector de Post-Procesador.

Una vez que se tenga el post-procesador seleccionado, se puede generar el programa en el código de Universal Robots. Para ello, se pulsa clic derecho sobre el programa y a continuación sobre Generar Programa de robot.

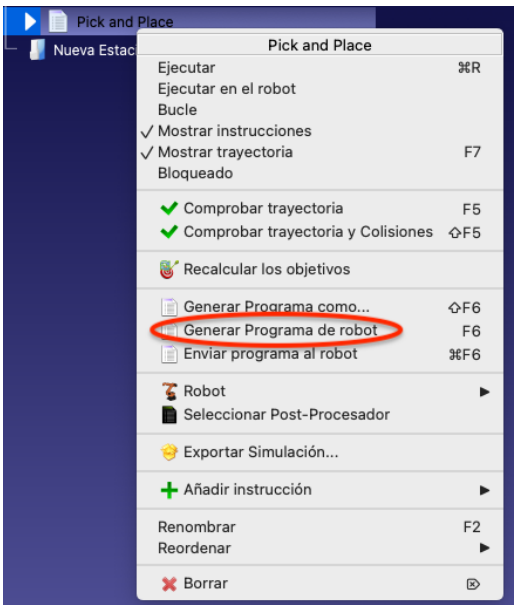

Figura 38. Navegación a Generar Programa de Robot.

Si se ha seleccionado el post-procesador de Universal Robots, se obtiene en este caso el siguiente código.

```
   # Main program:
   speed_ms    = 0.050
   set_tcp(p[0.000000, 0.000000, 0.233370, 0.000000, 0.000000, -0.174533])
   movej([2.825164, -0.036377, -1.934488, 0.000000, -1.170554, -
5.061455],accel_radss,speed_rads,0,0)
   movel(p[0.000000, 0.000000, 0.200000, -2.350499, 2.084305, -
0.000250],accel_mss,speed_ms,0,0.000)
   # Adjuntar a Gripper
   movel(p[0.000048, -0.000039, 0.200000, -2.439180, 1.979795, -
0.000262],accel_mss,speed_ms,0,0.000)
   movel(p[0.000000, 0.000000, 0.200000, -2.350499, 2.084305, -
0.000250],accel_mss,speed_ms,0,0.000)
   movel(p[0.000000, 0.000000, 0.085000, -2.350499, 2.084305, -
0.000250],accel_mss,speed_ms,0,0.000)
   # Separar de Gripper
   movel(p[0.000000, 0.000000, 0.200000, -2.350499, 2.084305, -
0.000250],accel_mss,speed_ms,0,0.000)
   # End of main program
 end
```

Se puede observar, que RoboDK genera el programa estableciendo las coordenadas y orientación de cada posición y definiendo si los movimientos son articulares o lineales. Sin embargo, no tiene control sobre las entradas y salidas del controlador, por lo que, en el código generado, crea un comentario para avisar al usuario que debe programar el comando correspondiente en dicha línea.

Si se genera el programa para un robot ABB con el controlador ABB IRBC5 se obtiene lo siguiente:

```
 MODULE MOD_PickandPlace

      ! Tool variables:
      PERS tooldata Gripper :=
[TRUE,[[0.000,0.000,233.370],[0.99619470,0,0,-
0.08715574]],[1,[0,0,20],[1,0,0,0],0,0,0.005]];

      ! Reference variables:
 PERS wobjdata World := [FALSE, TRUE,
"",[[500,0,0],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];

 PROC Main()
      ConfJ \On;
      ConfL \Off;
      MoveAbsJ [[161.8700,-2.084260,-110.838,0,-67.0678,-
290],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v50,z1,Gripper \WObj:=World;
      MoveL [[0,0,200],[0.00003430,-0.74820352,0.66346928,-
0.00007970],[1,0,-4,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v50,z1,Gripper
\WObj:=World;
      ! Adjuntar a Gripper
      MoveL [[0.048,-0.039,200.000],[0.00003348,-0.77643152,0.63020162,-
0.00008337],[1,0,-4,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v50,z1,Gripper
\WObj:=World;
      MoveL [[0.000,0.000,200.000],[0.00003430,-0.74820352,0.66346928,-
0.00007970],[2,-1,-
3,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v50,z1,Gripper \WObj:=World;
      MoveL [[0.000,0.000,85.000],[0.00003430,-0.74820352,0.66346928,-
0.00007970],[2,-1,-
3,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v50,z1,Gripper \WObj:=World;
      ! Separar de Gripper
      MoveL [[0.000,0.000,200.000],[0.00003430,-0.74820352,0.66346928,-
0.00007970],[2,-1,-
3,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v50,z1,Gripper \WObj:=World;
  ENDPROC

 ENDMODULE
```

Al igual que para el caso del código generado para Universal Robots, RoboDK genera comentarios para indicar que hay que cambiar las líneas relacionadas con las entradas y salidas del controlador.

De esta forma, queda ejemplificado el uso de los post-procesadores de RoboDK para aplicarlos a diferentes robots.

3.2.2. Algoritmo Visión Artificial

En esta sección, se va a tratar principalmente como se ha desarrollado el algoritmo de visión artificial, para que a partir de una fotografía tomada en RoboDK se pueda obtener información de la imagen relevante para la célula como el color de la pieza y su ángulo de desviación. Para que esto fuese posible, se ha utilizado la librería de OpenCV, que es una librería *Open Source* de las más utilizadas hoy en día para la visión artificial.

Detección color

Se puede definir una imagen dentro del ámbito matemático como una matriz numérica de 3 dimensiones, en las que las dos primeras hacen referencia al alto y ancho de la imagen, y la última esta asociada a los 3 colores primarios (rojo, verde y azul), donde cada elemento de la matriz representa un píxel de la imagen, el cual toma un valor desde 0 hasta 255. Por lo tanto, una imagen se puede dividir en 3 matrices diferentes de forma que la superposición de los valores de una matriz de un color primario con una matriz de otro color primario formaría un color nuevo.

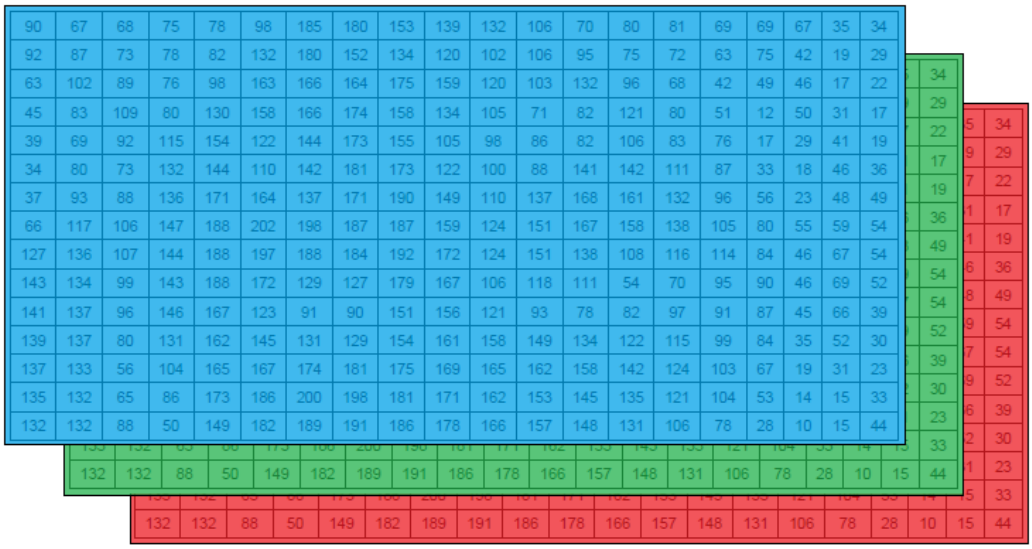

Figura 39. Matrices RGB obtenidas a partir de una imagen. (Fuente: OpenCV)

Sabiendo esto, cuando OpenCV lee la imagen tomada, transforma la imagen a una matriz numérica y con ello se puede obtener información útil de la imagen. Para tomar una fotografía en RoboDK se emplea el siguiente código.

```
camera = RDK.Item('Main Camera')    # se enlaza la cámara de RoboDK
RDK.Cam2D_Snapshot(path)            # se realiza una captura de pantalla
```

Para poder obtener el color de la imagen, en primer lugar, hay que crear máscaras para cada color que se quiera detectar. Estas máscaras son matrices binarias de 2 dimensiones en las que se define un rango de validación de una variable, de forma que, si queremos detectar el color verde, se deberá establecer rangos con valores bajos de rojo y azul, y rangos con valores altos de verde. De esta forma se obtendrá una imagen binaria en la que los 0 representen un píxel que no es verde y los 1 representen algo que si es verde.

A continuación, se muestra el código empleado para crear las máscaras de color.

```
# lectura de la imagen y copia para procesar el angulo posteriormente
imagen=cv2.imread(out_path)
imagen_angulo=imagen.copy()

# creación máscaras de color
verdeBajo = (0, 200, 0)
verdeAlto = (10, 255, 10)
mask_verde = cv2.inRange(imagen, verdeBajo, verdeAlto)

azulBajo = (0, 0, 200)
azulAlto = (10, 10, 255)
mask_azul = cv2.inRange(imagen, azulBajo, azulAlto)

rojoBajo = (200, 0, 0)
rojoAlto = (255, 10, 10)
mask_rojo = cv2.inRange(imagen, rojoBajo, rojoAlto)
```

Para ejemplificar el resultado que se obtiene al aplicar las máscaras, se ha tomado una fotografía en RoboDK con las tres piezas de colores diferentes, que se muestra a continuación.

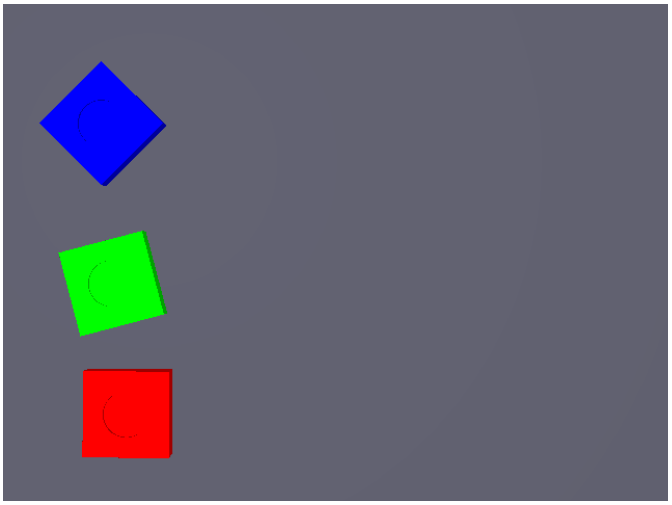

Figura 40. Fotografía RoboDK con tres piezas de distintos colores.

Los rangos utilizados para crear las máscaras de color se muestran a continuación:

	LÍMITE INFERIOR (R,G,B)	LÍMITE SUPERIOR (R,G,B)
PIEZA ROJA	(200, 0, 0)	(255, 10, 10)
PIEZA VERDE	(0, 0, 200)	(10, 10, 255)
PIEZA AZUL	(0, 200, 0)	(10, 255, 10)

Tabla 4. Parámetros RGB para creación máscaras de colores.

Empleando dichos rangos en las máscaras se obtienen los siguientes resultados:

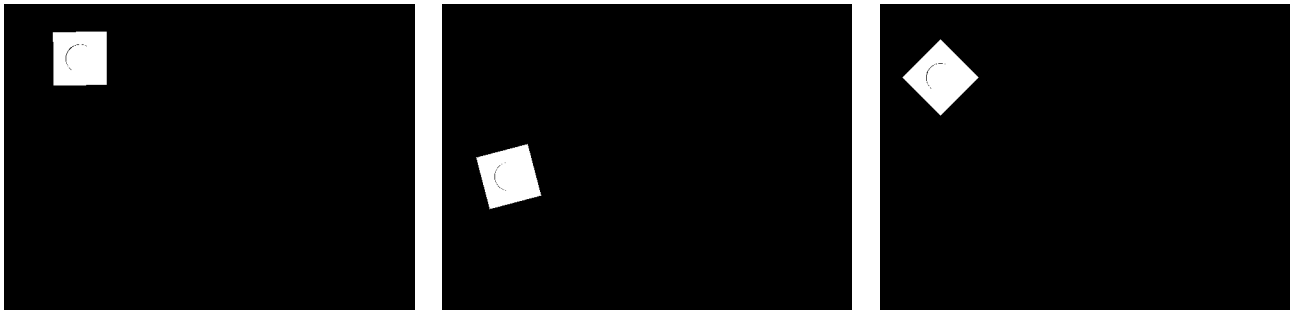

Figura 41. Imágenes binarias obtenidas a través de las máscaras. En orden, máscara roja, verde y azul

Como se puede observar en la figura anterior, cada una de las máscaras esta perfectamente calibrada ya que no hay ningún color que muestre ruido de otros colores. En este punto estamos en disposición de poder detectar el color del objeto que haya en la imagen.

La función findContours de OpenCV permite encontrar contornos de objetos presentes en una imagen, de forma que, si se obtiene los contornos de cada una de las máscaras de color, solo aquella máscara cuyo color coincida con el del objeto tendrá un número de contornos mayor que la unidad, mientras que las otras máscaras tendrán un número de contornos igual a cero. De esta forma, se consigue clasificar la pieza en función de su color. El código empleado para ello se muestra a continuación.

```
# selección de color en función del número de contornos obtenido
for idx,item in enumerate(mask):
    contornos=cv2.findContours(item,cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)
    img_cont = cv2.drawContours(imagen,contornos,-1,(0,0,0),5)
    if idx == 0 and len(contornos)==1:
        color_obj="verde"
    if idx == 1 and len(contornos)==1:
        color_obj="azul"
    if idx == 2 and len(contornos)==1:
        color_obj="rojo"
```

Detección ángulo

Para la detección del ángulo del objeto se ha usado la función minAreaRect que incorpora la librería OpenCV [8]. Dicha función devuelve una serie de parámetros de una forma rectangular como su centro de gravedad, altura, anchura y ángulo de rotación. Para ello la función requiere como entrada el contorno obtenido mediante FindContours.

Esta función en primer lugar localiza los vértices del rectángulo y los numera del 0 al 3 en sentido horario empezando a enumerar por el punto más bajo con respecto al eje vertical (véase Figura 42). Si dos puntos tienen la misma altura, se comienza por el punto que esta más hacia la derecha.

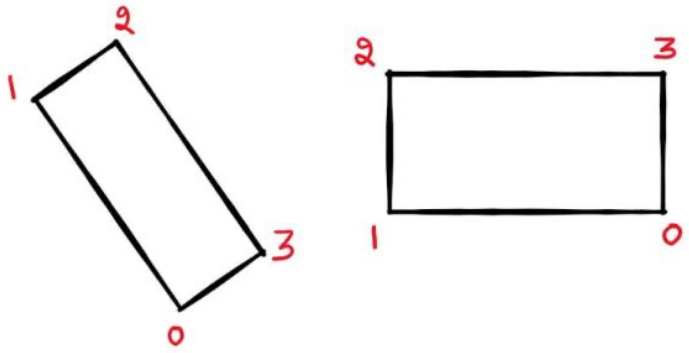

Figura 42. Ejemplo enumeración función minAreaRect. (Fuente: OpenCV).

Una vez numerados los vértices del rectángulo, el ángulo será el que forman la línea que une el punto de inicio con el punto final, con respecto a la línea horizontal como se puede ver en la siguiente figura.

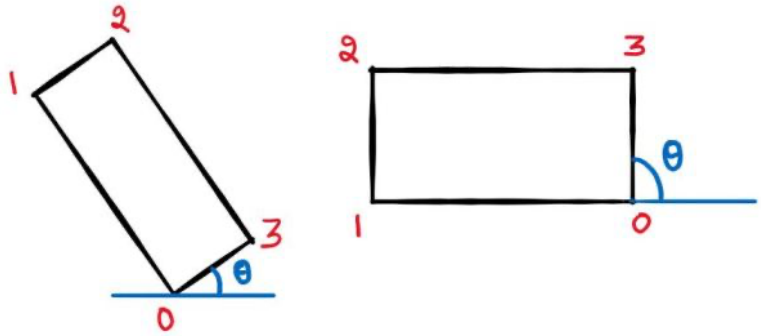

Figura 43. Ejemplo cálculo ángulo función minAreaRect. (Fuente: OpenCV).

El ángulo que se obtiene mediante la función siempre estará acotado entre 0 y 90 grados, ya que si la figura sigue rotando se utilizará el siguiente borde, ya que al girarlo el criterio de enumeración cambiaría. De forma que el ángulo de la pieza quedaría calculado empleando el siguiente código.

```
 rotatedRect = cv2.minAreaRect(cnt)
(cx, cy), (width, height), angle = rotatedRect
```

3.2.3. Comunicación cámara-robot

Una vez que se tiene la imagen procesada y se ha podido obtener la información necesaria de ella, el siguiente paso será transmitir dicha información desde el PC que ha realizado el procesamiento hasta el robot UR3 que se va a encargar de aplicarla.

Para ello, en este trabajo se ha creado una comunicación tipo TCP/IP en la cual el PC actuará como servidor en el que se procesa la imagen y se envía, y el UR3 actúa como cliente que recibe la información y con ello realiza la tarea de paletizado.

Servidor

Para la creación del servidor, en primer lugar, se deberá definir el host y el puerto que se va a emplear para transmitir los datos.

```
host = socket.gethostname()        # se obtiene el nombre del PC
port = 12345
```

Una vez definidos, se crea el servidor empleado la librería socket de Python, y mediante el comando *bind* se asocia el servidor al puerto. Además, también se puede definir el número de conexiones entrantes al servidor mediante el comando *listen*.

```
serv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serv.bind((host, port))
serv.listen(5)
```

Cuando el servidor ya esta activo y escuchando posibles peticiones entrantes de clientes, se crea un bucle infinito que nos permita estar escuchando dichas peticiones constantemente.

```
while True:
    conn, addr = serv.accept()
    from_client = ''
```

Dentro de este bucle, se creará un segundo bucle para que en el caso de que exista información de un cliente, pueda ir recibiendo la información poco a poco en el caso de los datos sean muy grandes.

```
while True:
        data = conn.recv(4096)
        if not data: break
        from_client += data.decode('utf-8')
```

No se puede recibir toda la información de golpe ya que en las comunicaciones TCP/IP existe un límite de bytes que se pueden enviar en un viaje, que viene definido por el tamaño del búfer. Por lo tanto, si se quiere transmitir datos que ocupen más que el tamaño del búfer, se deberá “trocear” la información y enviarla poco a poco.

Sin embargo, en este caso no se va a trabajar con cadenas de texto tan largas ya que solo se recibe una palabra como validación del cliente. De todas formas, se ha diseñado para cadenas más largas por si en trabajos futuros se decide enviar datos de mayor tamaño.

Finalmente, si el servidor recibe del cliente la palabra "Ready", significa que la pieza está en posición de tomar la fotografía para procesarla y por lo tanto deberá ejecutar la función de procesamiento de imagen, que se ha explicado en el apartado anterior. Una vez enviada la información, se cerrará la conexión entre el cliente y el servidor.

```
if from_cliente == "Ready":
            info = procesar_imagen()
            conn.send(info.encode('utf-8'))
            conn.close()
```

Cliente

Mientras que el código usado para el servidor es el mismo tanto en simulación como en el proceso real, el cliente tiene cambios importantes ya que en simulación se usa Python para crear al cliente, pero en el laboratorio se deberá usar la programación de Universal Robots para poder recibir la información de la cámara. En este apartado nos centraremos en el cliente diseñado en simulación.

En el caso del cliente en simulación, el host será el propio PC ya que el programa se ejecuta dentro del mismo.

```
host = socket.gethostname()
port = 12345
```

Se crea el socket y se realiza la conexión cliente-servidor.

```
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect((host, port))
```

Esta función será ejecutada justo cuando la pieza esté en posición de procesamiento de imagen, por lo que en dicho momento se establecerá la conexión con el servidor, se enviará el "Ready" y finalmente el cliente recibirá el mensaje con la información del color y del ángulo en el formato *color_angulo,* y después cerrar el cliente.

```
client.send("Ready".encode('utf-8'))
from_server = client.recv(4096)
msg=from_server.decode('utf-8')
client.close()
```

3.2.4. Resultados

Una vez llegados a este punto, se han definido los sistemas de referencia y elementos de la estación, se ha hablado de los programas necesarios para el funcionamiento de la célula y se ha explicado como se ha desarrollado el algoritmo de visión artificial y las comunicaciones cliente-servidor entre el PC y el robot.

Por lo tanto, en este apartado se van a exponer los resultados obtenidos de la simulación, así como un enlace para poder ejecutarla en un navegador web.

El tiempo total de simulación es de 14 minutos y 58 segundos, por lo que considerando que se t 24 piezas, cada una de ellas tarda en procesarse 37.4 segundos. La simulación no presenta ningún tipo de error y todo funciona sin problemas.

RoboDK ofrece la posibilidad de crear un archivo HTML con la simulación creada para que otro usuario pueda ejecutarlo en su ordenador sin la necesidad de tener RoboDK instalado. Para ello hay que descargar el archivo enlazado en el QR de la **Figura 44** y seguir las instrucciones del ANEXO A: Ejecución de simulación HTML de RoboDK.

Figura 44. QR que enlaza con archivo simulación RoboDK.

Las ventajas de utilizar este tipo de simulación frente a un video convencional es la posibilidad de desplazarse por la célula pudiendo hacer zoom, y además orientar la vista a nuestra elección.

Aun así, se van a mostrar capturas de todas las partes del proceso para que quede más claro el resultado final obtenido en la simulación.

En proceso comienza moviendo a todos los robots a sus posiciones iniciales para evitar colisiones durante el transcurso de este.

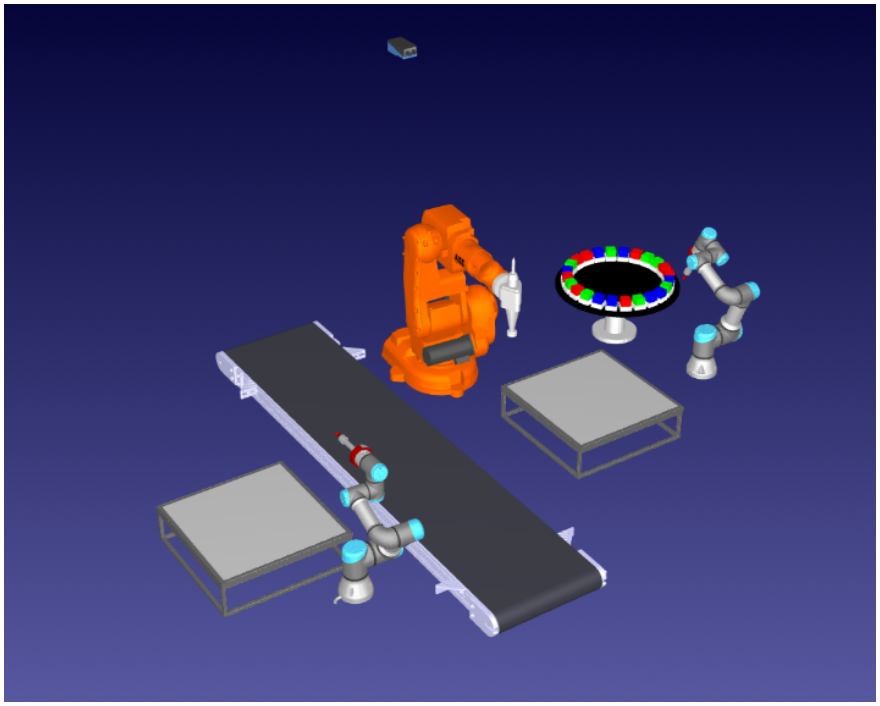

Figura 45. Simulación célula RoboDK (I)

Una vez en posición, el robot UR3e procede a coger la pieza que está en posición en la mesa giratoria y la mueve hasta la mesa, en una posición en la cual el ABB IRB140 tiene el suficiente alcance para llegar.

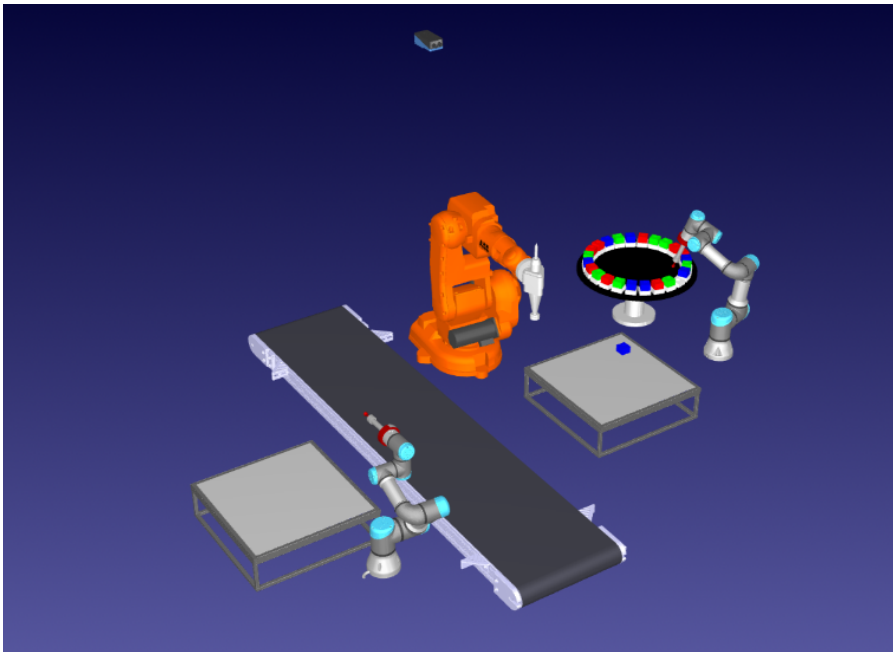

Figura 46. Simulación célula RoboDK (II)

Cuando la pieza está en posición, el UR3e se retira a su posición inicial a la espera de que el ABB mueva la mesa para poder coger la siguiente pieza. Mientras tanto, el ABB utiliza la ventosa de la doble herramienta para colocar la pieza en una posición en la cual tenga un buen rango de movimiento para mecanizar correctamente la pieza.

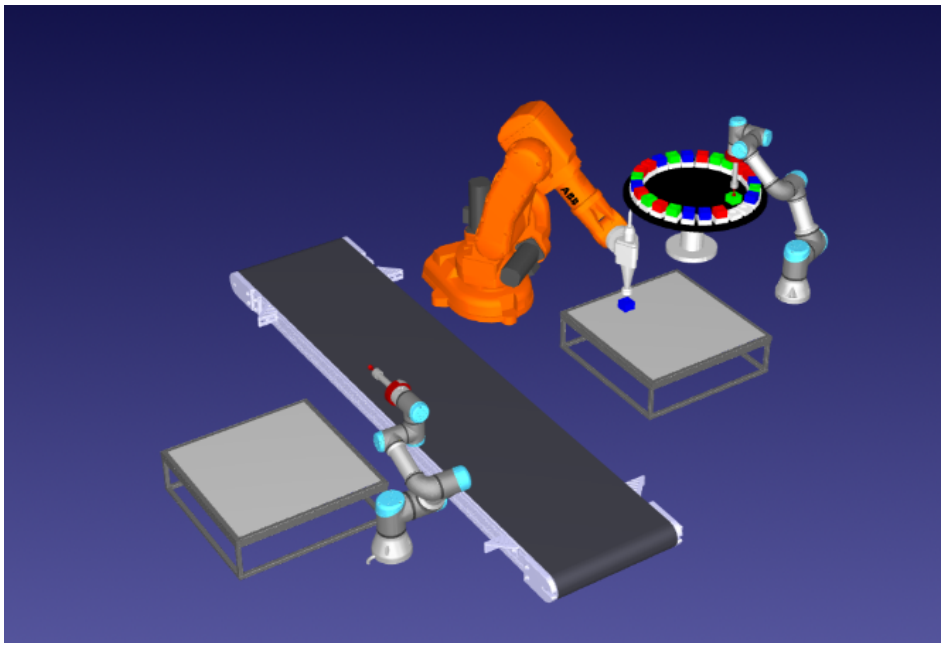

Figura 47. Simulación célula RoboDK (III)

El robot ABB realiza el mecanizado de la pieza y mientras tanto el UR3e está realizando la tarea de pick & place para que en el momento que el ABB termine sus tareas con una pieza, pueda empezar con la siguiente en el acto.

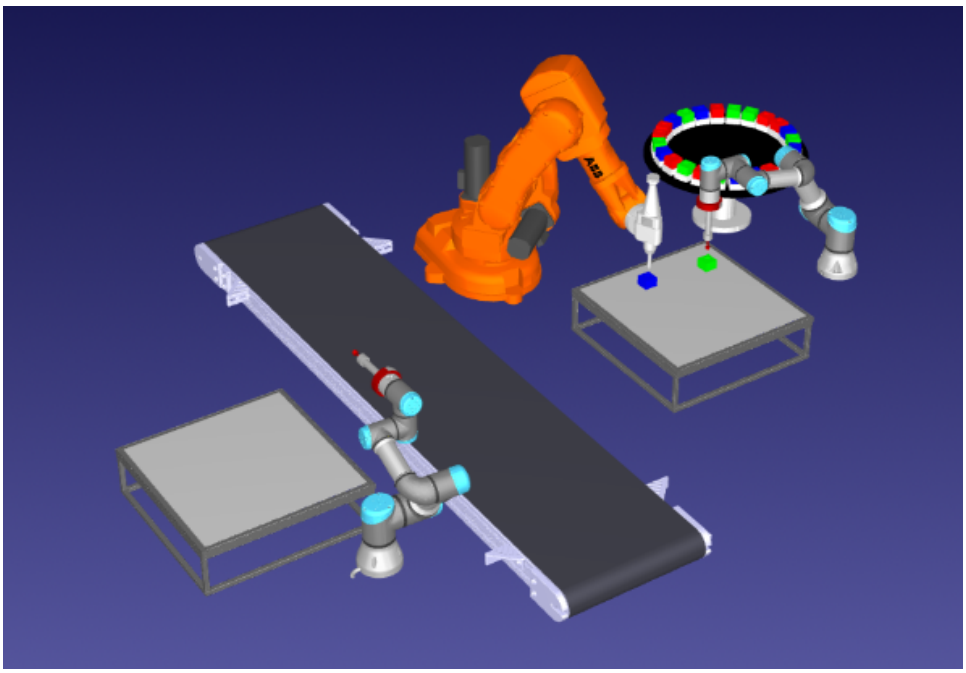

Figura 48. Simulación célula RoboDK (IV)

Cuando el ABB termina el mecanizado, pone la doble herramienta en posición ventosa y realiza una tarea de pick & place desde la mesa hasta la cinta transportadora. En el momento que este deje la pieza, se pondrá en marcha la cinta.

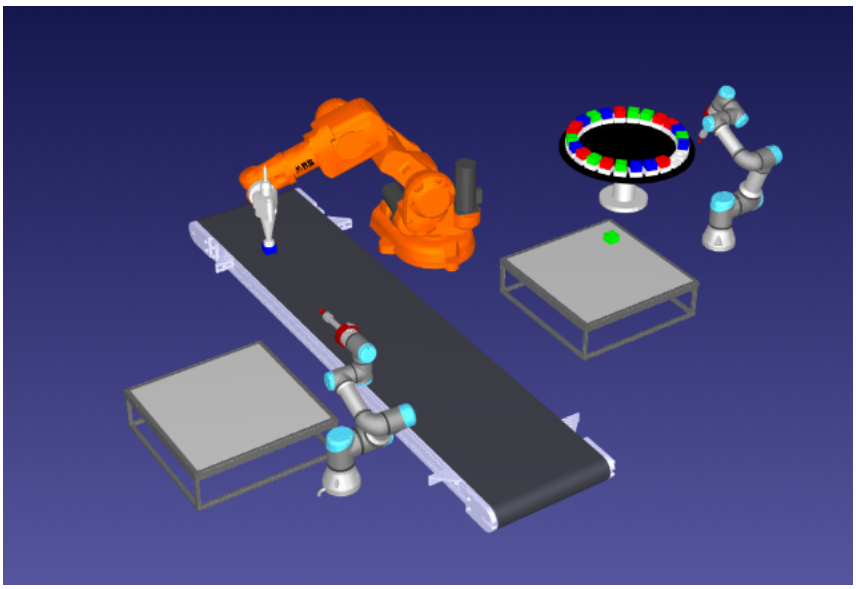

Figura 49. Simulación célula RoboDK (V)

La cinta llevará a la pieza a una posición, en la cual la cámara pueda obtener una imagen sin interferencias de ningún objeto o robot, donde se detendrá para procesar la imagen y una vez obtenida la información necesaria se pondrá la cinta en marcha para llegar al final del recorrido de la cinta. Mientras tanto el ABB ha comenzado a mecanizar la siguiente pieza.

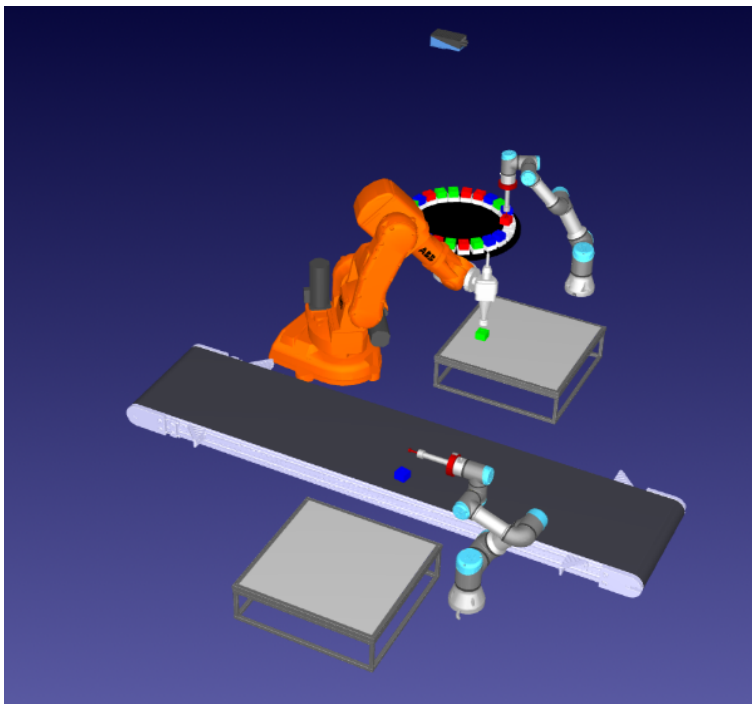

Figura 50. Simulación célula RoboDK (VI)

Cuando la pieza ha llegado al final de la cinta, el UR3 procede a realizar el paletizado con la información que la cámara le ha proporcionado, terminando de esta forma el ciclo de trabajo para una pieza.

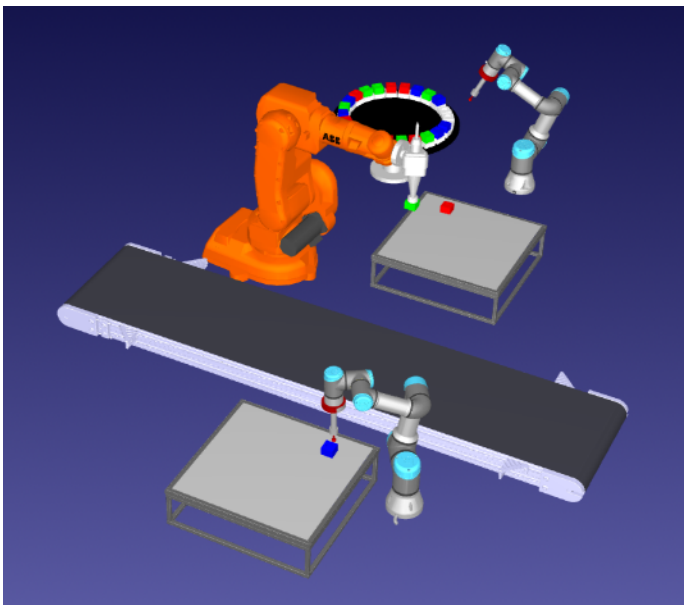

Figura 51. Simulación célula RoboDK (VII)

Si se sigue ejecutando la simulación, a los 8 minutos y 2 segundos, la célula robotizada ya ha paletizado la primera altura de piezas ordenadas por color y con posibles desviaciones en su orientación corregidas.

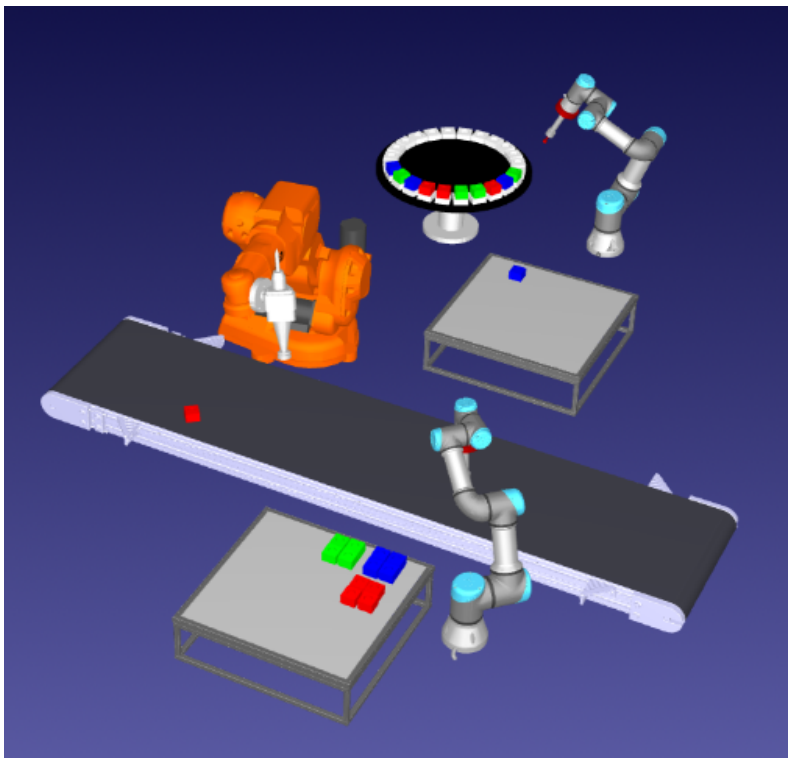

Figura 52. Simulación célula RoboDK (VIII)

Finalmente, a los 14 minutos y 58 segundos desde el comienzo del proceso, se ha conseguido paletizar las 24 piezas clasificadas en tres colores diferentes y con un apilamiento de dos alturas.

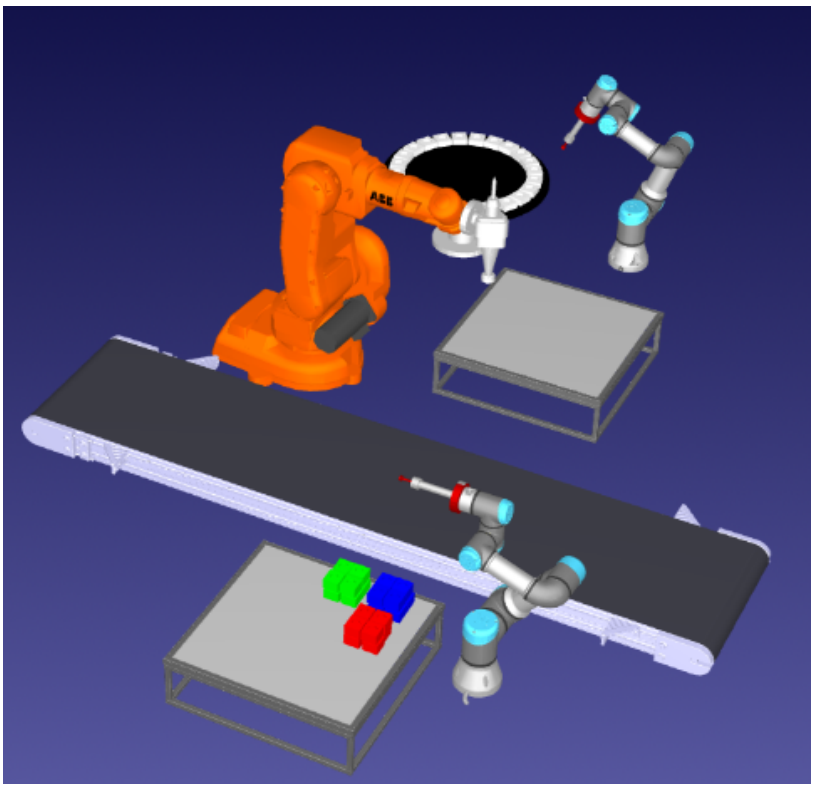

Figura 53. Simulación célula RoboDK (IX)

Para simular esta estación, se ha creado un manual de uso en el ANEXO B: Manual Utilización Célula RoboDK, en el que se explica como hay que proceder para ejecutar la simulación.

En la siguiente sección se explicará que se ha hecho para poder llevar esta simulación a la realidad, usando los robots y recursos disponibles del laboratorio.

3.3. Planteamiento célula laboratorio

En este apartado se va a explicar como se ha llevado a cabo a la realidad la célula simulada mediante RoboDK. En concreto, se centrará en la explicación de la célula del laboratorio, las piezas impresas en 3D, los algoritmos visión y de comunicaciones, la programación de los robots reales y también se mostrará el resultado final obtenido.

3.3.1. Puesta en marcha

Cuando se obtuvieron los resultados deseados en simulación, el siguiente paso era configurar la célula del laboratorio para que tuviese el mismo funcionamiento que en simulación.

Distribución de los elementos

En primer lugar, se tuvo que poner todos los elementos en posición para que el resultado fuese similar en simulación. Sin embargo, solo era posible mover dos elementos de la célula, el robot UR3e, que cuenta con una bandeja con asas que permite moverlo con facilidad, y la mesa giratoria, que era un poco más compleja de mover, pero se requería de cierta orientación para que los sensores estuviesen en posición. En concreto la bandeja UR3e se posicionó a 184 mm desde el borde inferior de la mesa y a 115 mm desde el borde derecho de esta con respecto a la esquina derecha de la bandeja donde se encuentra la manivela.

El resto de los elementos estaban fijados, como es el caso del robot UR3, el robot ABB IRB140 y la cámara, y la cinta transportadora, aunque no esta fijada no era posible moverla ya que estaba en el punto exacto para que ambos robots pudieran alcanzarla y para que la cámara pudiera tomar una buena imagen. Finalmente, la célula robotizada quedo de la siguiente forma:

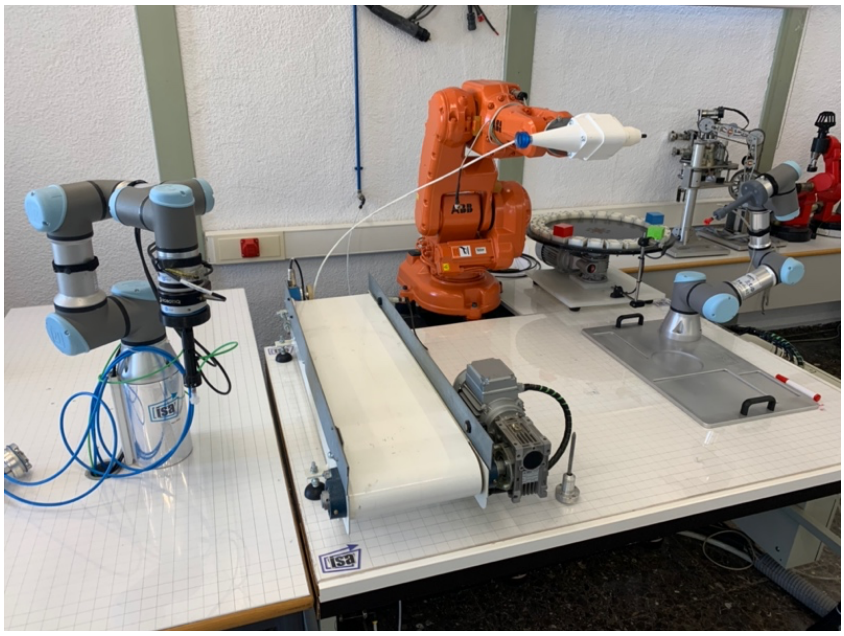

Figura 54. Célula robotizada del laboratorio.

Calibración de la doble herramienta

Al haber creado un nuevo tipo de herramienta, era necesario calibrar dicha herramienta para que el robot ABB IRB140 supiese donde estaba el extremo de esta. Además, había que tener en cuenta de que había que calibrar dos herramientas ya que en el mismo bloque herramienta cuenta con una ventosa y un delineador.

Para ello se crearon dos herramientas nuevas mediante el TeachPendant llamadas tool_ventosa y tool_plotter. El método calibrar las herramientas fue el método de los 4 puntos, el cual consiste en que mediante el uso de un patrón con forma de punta (véase **Figura 55**), se coloque el extremo de la herramienta en cuatro posiciones distintas para que el robot pueda obtener la información necesaria para configurar la herramienta.

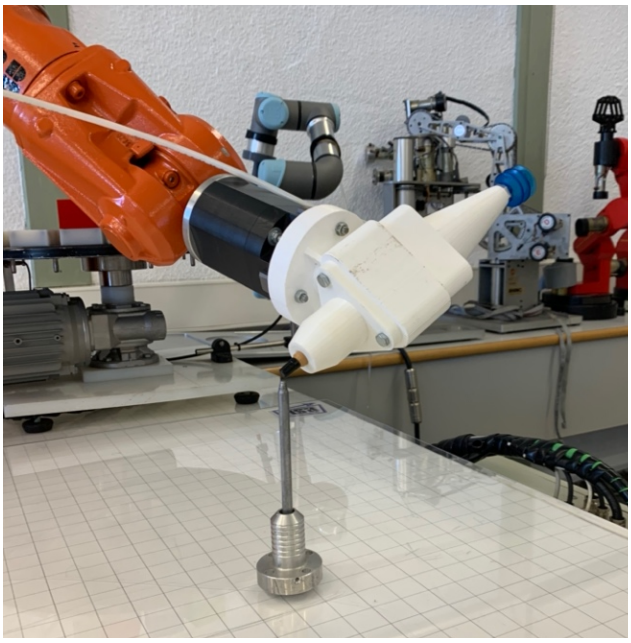

Figura 55. Proceso de calibración herramienta ventosa mediante método 4 puntos.

Dicha información es la distancia XYZ desde el origen de la brida del robot hasta la nueva herramienta, y las orientaciones de los ejes de la nueva herramienta. En este caso ambas herramientas se han configurado para que el eje Z este alineado con la herramienta apuntando hacia fuera.

Los valores de las distancias XYZ y orientaciones se muestran en la siguiente tabla.

	X (mm)	**Y (mm)**	**Z (mm)**	$q_1(rad)$	$q_2(rad)$	$q_3(rad)$	$q_4(rad)$
VENTOSA	492	175	160	0.0052	-0.9713	0.019	0.2367
DELINEADOR	492	-85	160	0.0052	0.9713	0.019	0.2367

Tabla 5. Valores calibración doble herramienta.

Señales digitales

Cuando se trabaja con varios robots en una misma célula, es muy importante establecer conexiones digitales entre los mismos que permitan evitar colisiones entre ellos si comparten espacio de trabajo, o dar el inicio de una tarea a otro robot.

En el caso de esta célula, es necesario establecer conexiones entre el UR3e y el ABB IRB140 para evitar colisiones ya que comparten el mismo espacio de trabajo, y conexiones entre el ABB IRB140 y el UR3, ya que el sensor de objetos al final de la cinta lo controla el UR3e, por lo que deberá enviar una señal digital al ABB, que es el que controla el movimiento de la cinta, para que la detenga.

Sin embargo, cuando se comenzó la programación de estas salidas existía el problema de que el ABB IRB140 no tenía conexión física con el UR3e, y además el ABB tenía todas sus entradas y salidas digitales al completo, por lo que era imposible crear dicha conexión.

Pero a pesar de que no existía este enlace entre estos robots, si que existía una conexión entre el UR3 y el ABB, por lo que se decidió crear una señal de entrada y salida entre el UR3e y el UR3, de forma que haciendo "rebotar" la señal en este último fuera posible establecer la conexión deseada.

También, se realizó una conexión puenteada de dos salidas digitales del ABB IRB140 entre el UR3 y UR3e, de forma que el UR3e pudiera leer directamente las señales generadas por el robot ABB.

A continuación, se muestra una tabla con las señales digitales empleadas para hacer funcionar la célula en el laboratorio.

ABB IRB140	UR3	UR3e
do_To_UR1	di_4	di_4
do_To_UR2	di_5	di_5
	di_2	do_2
di_from_UR_1	do_1	
di_from_UR_2	do_2	
di_from_UR_3	do_3	

Tabla 6. Conexiones entre robots. Entradas (verde) y salidas (azul).

Además de las señales digitales relacionadas con la comunicación entre robots, la estación de trabajo contaba con varias señales que controlan los elementos de esta, como la cinta, la mesa giratoria, etc. Dichas señales se muestran en la siguiente tabla.

ABB IRB140		UR3		UR3e	
Señal	Descripción	Señal	Descripción	Señal	Descripción
GRIPPER_CLOSE	Ventosa	do_4	Pinza	do_5	Ventosa
CONVEYOR_FWD	Cinta hacia adelante	do_5	Ventosa		
CONVEYOR_BWD	Cinta hacia atrás	di_6	Sensor presencia objeto final cinta		
CONVEYOR_OBJ_SEN	Sensor presencia objeto principio cinta				
TABLE_FWD	Rotación mesa giratoria				
TABLE_OBJ_SEN	Sensor presencia objeto mesa giratoria				

Tabla 7. Señales digitales de los actuadores (azul) y sensores (verde) de la estación.

3.3.2. Impresión 3D

Esta sección se centrará en como se han impreso las piezas 3D necesarias para la célula del laboratorio y cual ha sido el resultado final de las mismas.

Pieza manipulada

La pieza manipulada que se va a emplear en la célula del laboratorio va a ser impresa en PLA en tres colores distintos para tener variedad a la hora de probar el algoritmo de visión artificial.

Los parámetros más importantes utilizados para imprimir las piezas se muestran a continuación.

Temperatura Impresión	**210ºC**
Altura de Capa	0.25 mm
Densidad de Relleno	15%
Patrón Relleno	Rejilla
Velocidad Impresión	50 mm/s

Tabla 8. Parámetros Impresión 3D de la pieza manipulada.

Se ha considerado hacer un relleno de un 15% ya que como la pieza no tiene que soportar ningún tipo de carga estructural, de esta forma se puede ahorrar tanto en tiempo de

impresión como en coste de materia prima y energía. Con estos parámetros fijados, la pieza impresa tendrá un peso de 31 gramos, lo cual es casi despreciable dado las cargas máximas que pueden manejar los robots (alrededor de 6 kg).

A continuación, se muestra el resultado final de las piezas impresas.

Figura 56. Resultado final de las piezas manipuladas impresas.

Soporte doble herramienta

El soporte diseñado que une el soporte de la ventosa y el soporte del delineador ha sido impreso con un PLA blanco para que no destaque con respecto a las otras partes. El material ideal para imprimir esta pieza hubiese sido el ABS ya que es un material mucho más resistente que el PLA, pero debido a que la impresora con la que se contaba no podía imprimir en dicho material, finalmente se ha optado por elegir el PLA configurando los parámetros de impresión a conciencia de que tiene que soportar una cierta carga estructural.

Los parámetros más importantes utilizados para imprimir las piezas se muestran a continuación.

Temperatura Impresión	**210ºC**
Altura de Capa	0.2 mm
Densidad de Relleno	60 %
Patrón Relleno	Rejilla
Velocidad Impresión	50 mm/s

Tabla 9. Parámetros Impresión 3D de la pieza doble herramienta.

En este caso, se ha seleccionado una altura de capa menor con respecto a la pieza manipulada, ya que se necesita más precisión en los agujeros para que los tornillos puedan encajar correctamente. Además, se ha aumentado la densidad de relleno de la pieza debido a que va a tener que soportar el soporte de la ventosa y el soporte del delineador.

Junto con lo anterior, habría que tener en cuenta también que los propios movimientos del robot podrían generar una fractura en la pieza si no es lo suficientemente densa debido a desplazamientos rápidos o fuerzas debido a la presión al agarrar el objeto con la ventosa o al "mecanizar" la pieza con el delineador.

Con todos los parámetros anteriores fijados, el peso de la pieza es de 420 gramos. El resultado de la pieza impresa acoplada a la brida del robot con los soportes de la ventosa y del delineador se muestra a continuación.

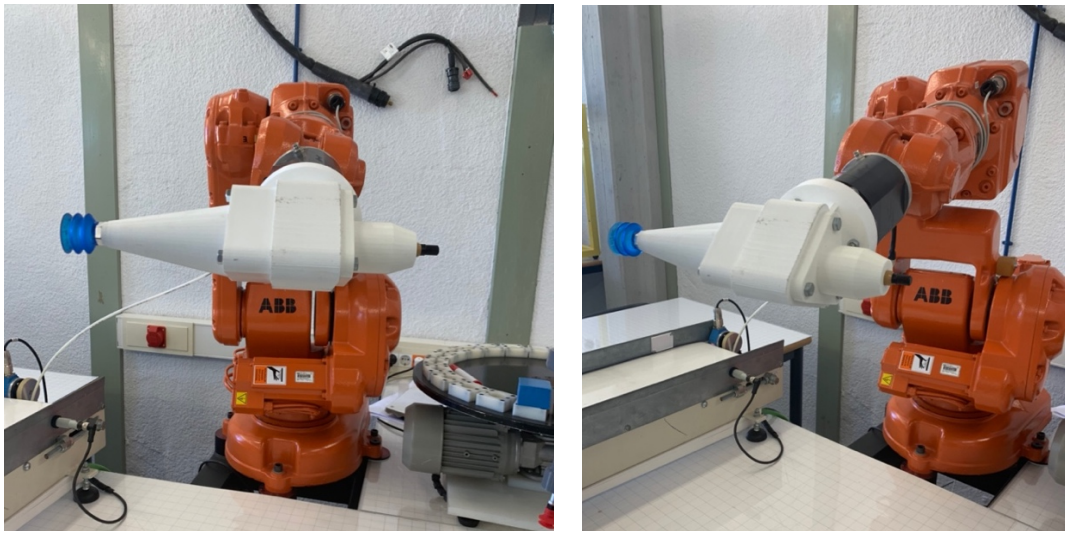

Figura 57. Resultado final de la doble herramienta impresa en 3D acoplada a la brida del robot.

3.3.3. Algoritmo de visión y comunicaciones

En esta sección se va a tratar los cambios realizados en los algoritmos de visión artificial para la detección del color y ángulo de las piezas, y sobre la comunicación TCP/IP con el robot UR3 del laboratorio.

Algoritmo de visión artificial

Al trabajar con la cámara real en el laboratorio, surgieron importantes diferencias que provocaron grandes cambios en el planteamiento del problema de visión artificial.

En primer lugar, la imagen que capturaba la cámara no era de mucha calidad y por ello debía ser recortada a un tamaño adecuado en el que solo estuviese la cinta blanca y la pieza para detectar la información correctamente.

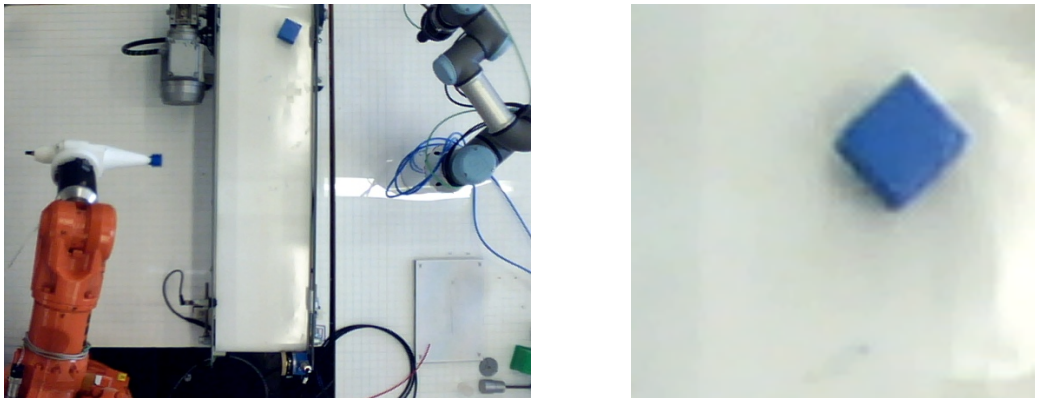

Figura 58. Imagen capturada por la cámara del laboratorio y su posterior procesado.

Esto provocaba que la imagen a procesar estuviese más pixelada por lo que al calcular el ángulo de la pieza, como se ha explicado en apartados anteriores, aparecían errores de precisión.

Pero a pesar de ello, el problema más importante que surgió a raíz de probar el algoritmo en la cámara del laboratorio fue la detección del color. Cuando se trabajaba con la célula de simulación en RoboDK, los colores de las piezas eran planos y sin luces y sombras, pero en el laboratorio las luces y las sombras que genera la pieza provocaba fallos en el algoritmo. A continuación, se muestra un ejemplo del resultado obtenido al aplicar la máscara azul a una pieza azul obtenida mediante la cámara del laboratorio, en el cual aparecen en la máscara contornos distorsionados debidos a las sombras.

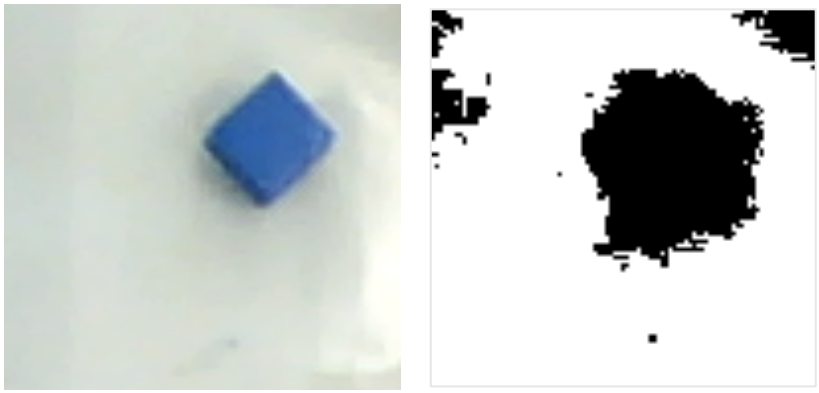

Figura 59. Imagen obtenida al aplicar máscara de color azul sobre pieza azul.

Tras varias pruebas, se llego a la conclusión de que una de las posibles fuentes de error era el rango de color establecido a cada máscara. Por lo tanto, habría que ajustar cada uno de los canales de color hasta conseguir un buen resultado. Sin embargo, ajustar la máscara de color en RGB resulta bastante tedioso por lo que se ha decido usar la escala HSV en lugar de la RGB.

La escala HSV cuenta con tres canales: Matiz (*Hue*), Saturación (*Saturation*) y Valor (*Value*). La principal ventaja con respecto a la escala de color RGB es que esta escala es una transformación no lineal del espacio de color RGB, de forma que una vez ajustado la saturación y el valor, podemos movernos por todos los colores solo cambiando el matiz. Mediante esta escala resulta mucho más fácil detectar los rangos de color ya que solo se debe variar un canal en vez de tres, como en el caso del RGB.

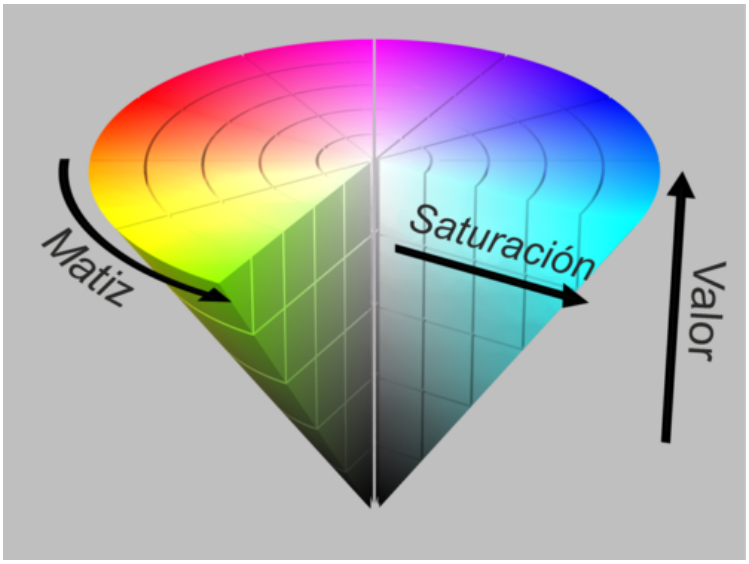

Figura 60. Escala de color HSV. (Fuente: Mathworks)

Aplicando esta escala de color, después de encontrar los rangos adecuados para detectar los colores, se obtuvo este resultado con la máscara azul.

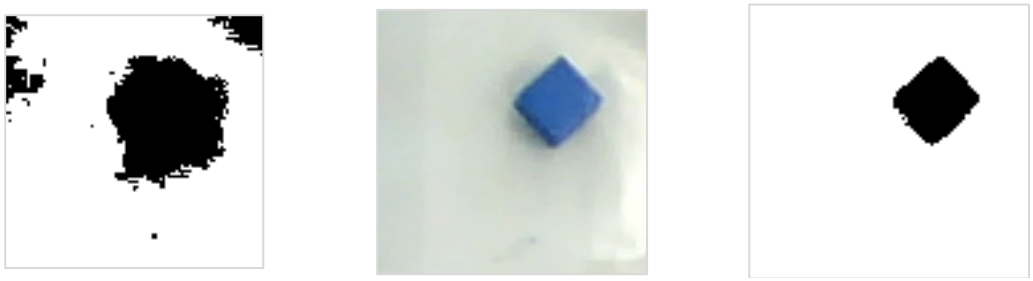

Figura 61. Mejora en la detección color aplicando escala HSV

Con el color verde se obtenían resultados parecidos al azul, pero el color presentaba algunos fallos. Estos fallos se debían a que la escala HSV divide dos tonos de rojo diferente (véase **Figura 62.**) e inicialmente solo se había aplicado un tono, por lo que el problema se solucionó creando una máscara combinada con ambos tonos de rojo.

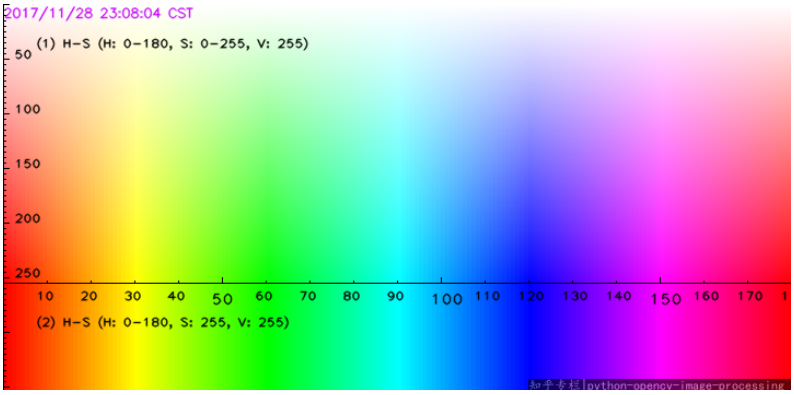

Figura 62. Vista de los componentes HSV. (Fuente: Stackoverflow)

El código utilizado para la creación de estas máscaras se muestra a continuación.

```
# convertimos la imagen a la escala de color HSV
imagenHSV = cv2.cvtColor(imagen, cv2.COLOR_BGR2HSV)

# DEFINICIÓN FILTROS COLORES
verdeBajo = (30, 100, 0)
verdeAlto = (80, 255, 255)
mask_verde = cv2.inRange(imagenHSV, verdeBajo, verdeAlto)

azulBajo = (90, 100, 0)
azulAlto = (120, 255, 255)
mask_azul = cv2.inRange(imagenHSV, azulBajo, azulAlto)

rojoBajo1 = (150, 0, 0)
rojoAlto1 = (255, 255, 255)
mask_rojoHSV_1 = cv2.inRange(imagenHSV, rojoBajo1, rojoAlto1)
rojoBajo2 = (0, 150, 0)
rojoAlto2 = (50, 255, 255)
mask_rojoHSV_2 = cv2.inRange(imagenHSV, rojoBajo2, rojoAlto2)
mask_rojoHSV = cv2.add(mask_rojoHSV_1, mask_rojoHSV_2)
```

En cuanto al problema con la detección del ángulo, tras varias pruebas se llegó a la conclusión de que el algoritmo detectaba contornos muy pequeños, los cuales provocaban errores en la detección del ángulo, incluso en ocasiones devolviendo ángulos completamente distintos al real.

Para solucionar este problema, se incluyó en el código un algoritmo que filtraba los contornos detectados y los que tuvieran un tamaño muy pequeño serían borrados de la imagen. Con ello, la detección del ángulo se corrigió completamente con un alto porcentaje de precisión. El código empleado para el filtrado de los contornos se muestra a continuación.

```
for idx, contour in enumerate(contornos): # recorremos todos los contornos
   area = cv2.contourArea(contour)        # calculamos el área del contorno
   if area > 20:
       cnt = contornos                    # guardamos contorno en variable
```

Comunicaciones TCP/IP con UR3e

Al trabajar con el robot real de la célula y no con la simulación, había que tener en cuenta algunas consideraciones.

El código empleado para el servidor no sufre ningún tipo de modificación, mientras que el código empleado para el cliente ya no sirve debido a que el UR3 no trabaja con código de Python, por lo que se debe adaptar el código del cliente a la programación de Universal Robots.

Para leer la información del socket se emplea el programa lectura_socket que previamente ha sido programado por el tutor del trabajo. Este programa lee la cadena de texto enviada por el servidor, mediante la función *socket_read_ascii_float,* y la separa en los dos valores de información útiles. Dicha cadena debe tener el formato *color_angulo* para poder procesarla correctamente. A continuación, se muestra el código del programa en formato txt.

```
lectura_socket
     socket_send_string("Ready")
     receiveFromServ:=socket_read_ascii_float(2)
     Bucle receiveFromServ[0]≠2
       Esperar: 0.1
       receiveFromServ:=socket_read_ascii_float(2)
     color:=receiveFromServ[1]
     angulo:=receiveFromServ[2]
```

Para iniciar la conexión entre el cliente y el servidor, en el programa principal del UR3 se utiliza la función *socket_open* y después se realiza una llamada al programa *lectura_socket* para obtener los valores del color y del ángulo.

```
open:=socket_open("158.42.206.7",30000)
     Invocar lectura_socket
```

Teniendo esto en cuenta, las comunicaciones entre el robot y el PC funcionan sin problema. Sin embargo, para que sea más fácil para el usuario que maneja el programa se ha creado una interfaz gráfica sencilla en Python, usando la librería Tkinter.

Esta interfaz va a permitir al usuario iniciar el servidor, visualizar la información procesada por el algoritmo de visión artificial como el color y el ángulo en grados y radianes, así como la información relacionada con la comunicación TCP/IP como el puerto y el servidor.

También se puede visualizar la fotografía tomada por la cámara en la parte izquierda de la misma, en la cual está marcado el contorno de la pieza detectado por el algoritmo. Esta imagen se presenta ya recortada para que se pueda apreciar mejor la pieza detectada y su contorno.

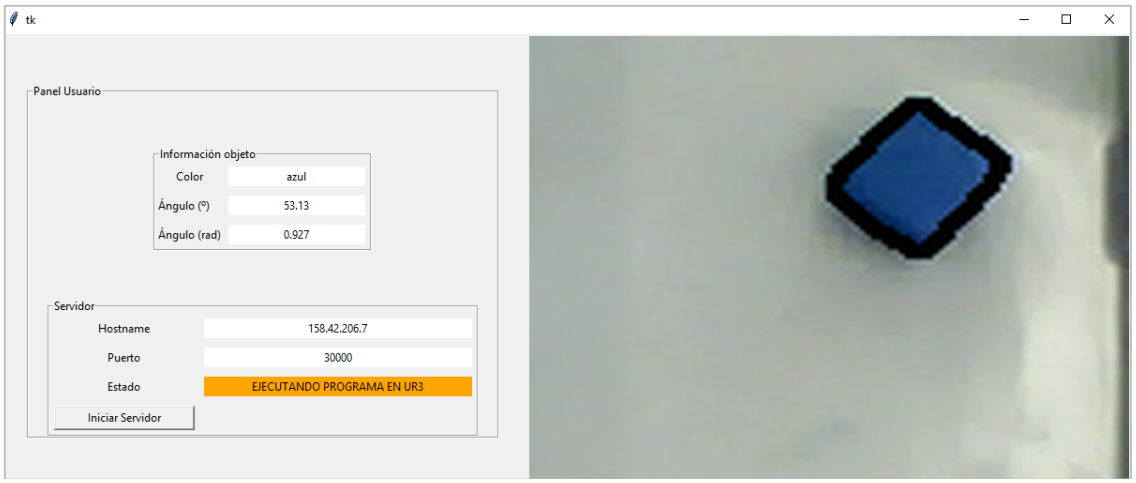

Figura 63. Interfaz gráfica diseñada para el servidor del laboratorio.

Se ha considerado añadir un apartado en la interfaz en la que se muestre el estado en el que se muestra el programa. Al iniciarse el programa, la barra de estado se muestra en rojo indicando que no se ha iniciado la conexión. Cuando se inicia el servidor, la barra cambia a verde indicando que el servidor esta activo y escuchando clientes, y cuando detecta la conexión con el robot y comienza a procesar su imagen la barra se vuelve naranja. Cuando finaliza de procesar vuelve al color verde indicando que esta esperando para procesar una nueva imagen.

3.3.4. *Programación robots*

Una vez ya se tiene todos los elementos en posición, se tienen las piezas impresas, la doble herramienta acoplada y los algoritmos adaptados a la célula del laboratorio, el siguiente paso es la programación de los robots.

La gran ventaja que tiene RoboDK respecto a otros softwares es la posibilidad de exportar sus programas a cualquiera de los robots soportados gracias a los post-procesadores. Esto ha ayudado bastante en el proceso de programación, ya que, aunque el código exportado no estaba completo debido a que había que configurar manualmente las señales digitales y algunas funciones, ha servido para tener un esqueleto a la hora de programar.

Debido a que el código empleado en cada uno de los robots es extenso para discutirlo línea a línea en este apartado, se va a adjuntar en el ANEXO D: Código de los programas utilizados.

Aun así, para explicar el funcionamiento de los tres programas, se ha creado un diagrama de flujo que permite explicar de forma esquemática el modo la programación de los robots.

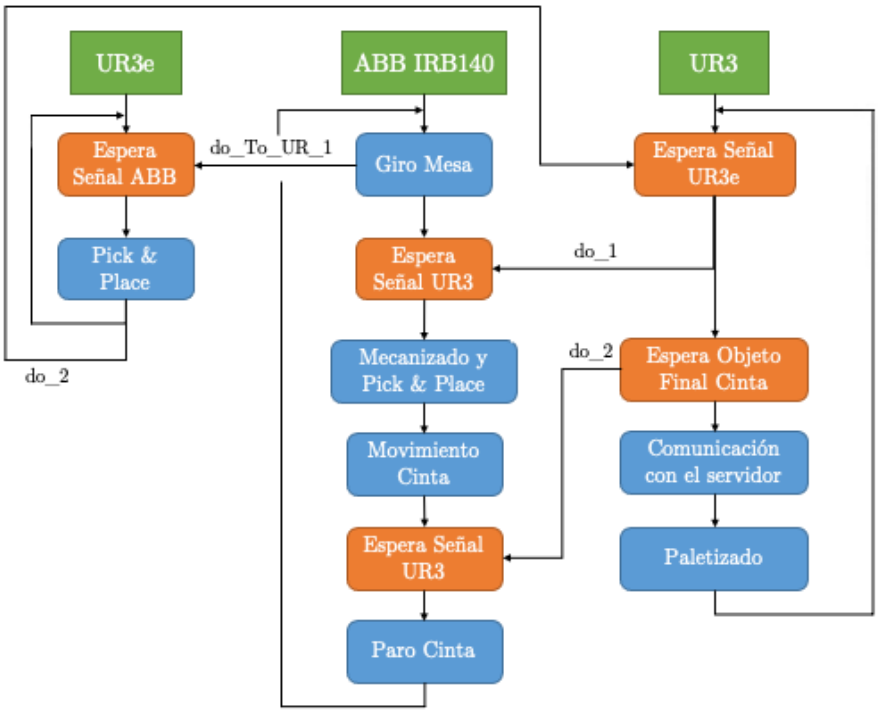

Figura 64. Diagrama de flujo de la programación de la célula.

Antes de iniciar el programa, todos los robots se mueven a una posición de partida donde no interfieren con ningún elemento de la célula.

El programa comienza con el movimiento de la mesa giratoria controlado por el robot ABB IRB140, la cual se detiene cuando se activa la señal digital TABLE_OBJ_SEN.

En ese momento, el robot ABB activa la señal do_To_UR_1, la cual iniciaría el movimiento de pick & place del robot UR3e.

Cuando el UR3e termina la tarea anterior, activa la señal do_2 para indicarle al UR3 que puede activar la señal do_1. La razón de realizar este "rebote" de señales es que, como se ha comentado en el apartado de señales digitales, no existe una conexión directa entre el UR3e y el ABB y por ello se precisa de esta maniobra.

Cuando el ABB detecta la señal do_1 que le manda el UR3 inicia el mecanizado y pick & place de la pieza, ya que en ese momento el UR3e ha terminado su tarea y esta esperando de nuevo a que se vuelva a activar la señal para realizar otra iteración.

Al terminar estas tareas, el ABB ya con la pieza en la cinta, la pone en marcha hasta que al robot UR3 se le activa la señal di_6, indicando que la pieza ha llegado al fin de la cinta, y en ese preciso instante el UR3 activa la señal do_2 que le indica al robot ABB que pare la cinta. Este proceso podría ser más sencillo si el robot ABB tuviese conexión directa con el sensor, sin embargo, para no manipular la distribución existente de señales digitales del laboratorio se ha optado por realizar esta maniobra.

Finalmente, el UR3 procedería a iniciar la comunicación TCP/IP con el PC para poder procesar la información de la pieza de la cinta, y a continuación paletizar la pieza en función del color detectado y pudiendo corregir el ángulo en caso de desviación.

Mientras tanto, el robot ABB ha comenzado a girar la mesa giratoria para reiniciar el proceso con otra pieza, quedando de esta forma definido el ciclo de trabajo de la célula robotizada.

3.3.5. Resultados

Finalmente, una vez configurado todos los robots de la célula, comprobado el buen funcionamiento del algoritmo de visión artificial y las comunicaciones entre el PC y el UR3e y haber programado los tres robots, la célula estaría terminada.

Para poder demostrar el correcto funcionamiento de la célula se ha realizado un vídeo en el cual se muestra el funcionamiento. Dicho vídeo se encuentra en la plataforma YouTube y se puede acceder a el con el siguiente código QR.

Figura 65. Código QR para visualizar el funcionamiento de la célula del laboratorio.

Como se puede ver en el video, el tiempo total para procesar las tres piezas del laboratorio es de 7 minutos y 44 segundos, empleado para cada pieza 2 minutos y 44 segundos aproximadamente.

Estos tiempos podrían bajarse optimizando más en que momento se ejecutan ciertas tareas o se ponen en marcha los actuadores. Pero debido a la complejidad que supone trabajar simultáneamente con los tres robots en el laboratorio, se ha optado por no optimizar dichos tiempos ya que el objetivo del trabajo no es optimizar al máximo el ciclo de trabajo de una célula sino probar su funcionamiento.

En el video se puede observar que el robot ABB IRB140 realiza una serie de movimientos extra en vez de ir directamente a los puntos concretos. Esto se debe a que el alcance limitado con el que cuenta el robot hace que para poder alcanzar ciertos puntos se tenga que llegar a una configuración de articulaciones muy límite, haciendo que para moverse a otro punto diferente el robot tenga que cambiar totalmente la configuración de sus articulaciones.

Al igual que en el apartado de resultados en RoboDK, se va a procesar a mostrar algunas capturas del proceso ejecutado en el laboratorio para que quede más claro el resultado final obtenido.

En primer lugar, el proceso comienza moviendo a todos los robots a su posición sino estaban en dicha posición previamente.

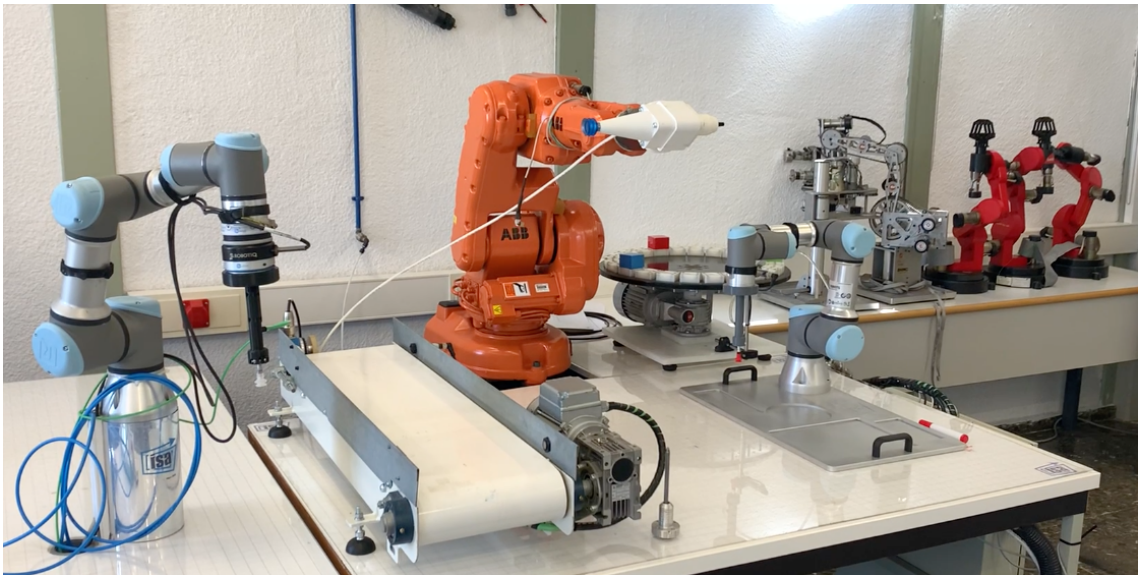

Figura 66. Simulación célula laboratorio (I)

Una vez que están en posición todos los robots, el ABB comienza a mover la mesa giratoria hasta que el sensor de la mesa detecta que hay una pieza. En ese momento se para y el UR3e procede a realizar la tarea de pick & place.

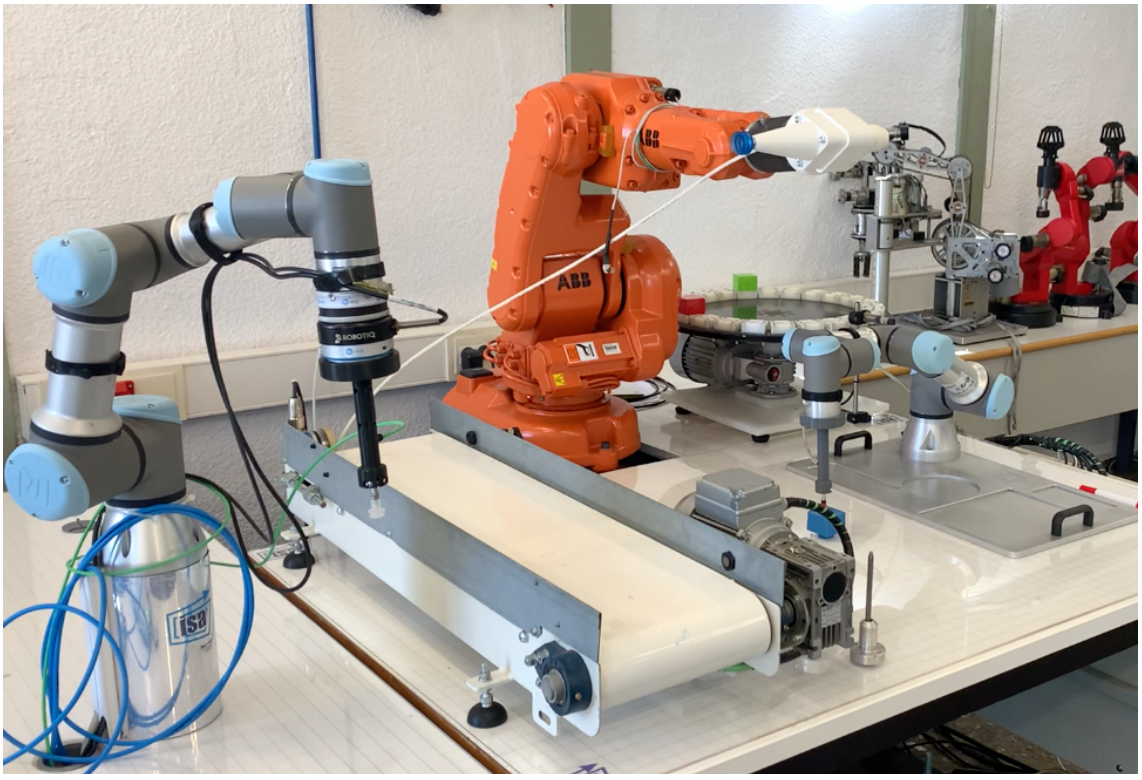

Figura 67. Simulación célula laboratorio (II)

A continuación, se procede a realizar el mecanizado de la pieza. A diferencia de la célula en simulación, en el laboratorio no se puedo ejecutar la maniobra de acercamiento de la pieza a la zona de trabajo ejecutado por el ABB, ya que el alcance estaba muy limitado para ello. La solución fue que el UR3e se estirase más para llegar a una zona en la que el ABB pudiera trabajar sin problema.

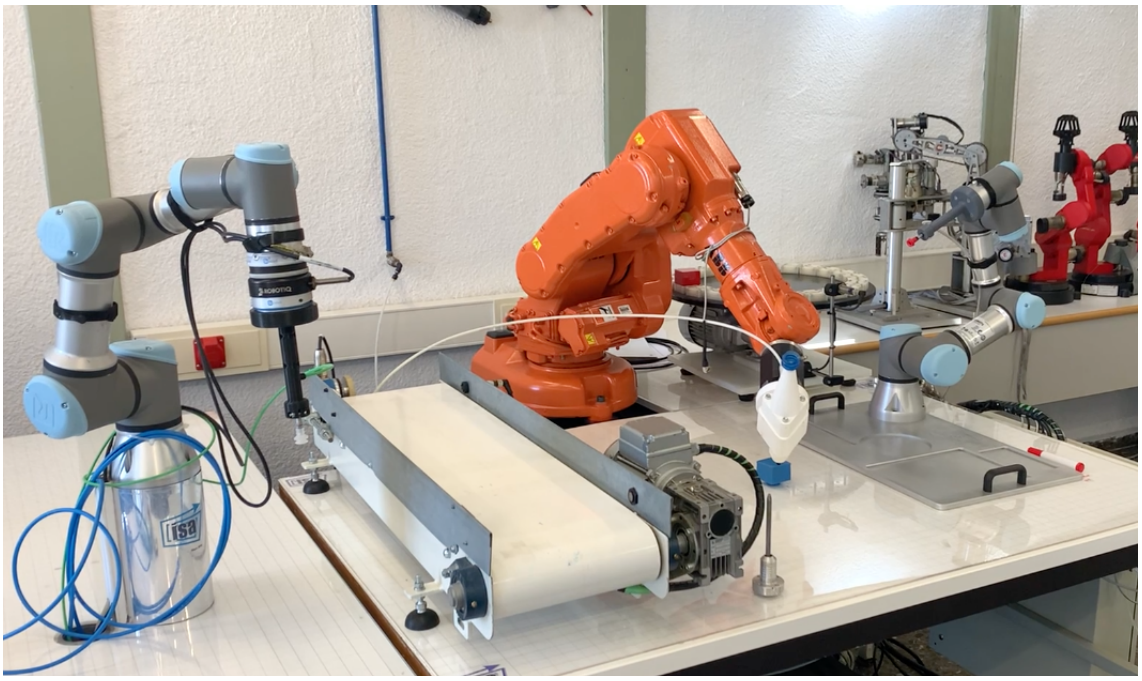

Figura 68. Simulación célula laboratorio (III)

Una vez que finaliza el proceso de mecanizado, el ABB pone la herramienta en posición de ventosa para realizar el pick & place de la pieza a la cinta.

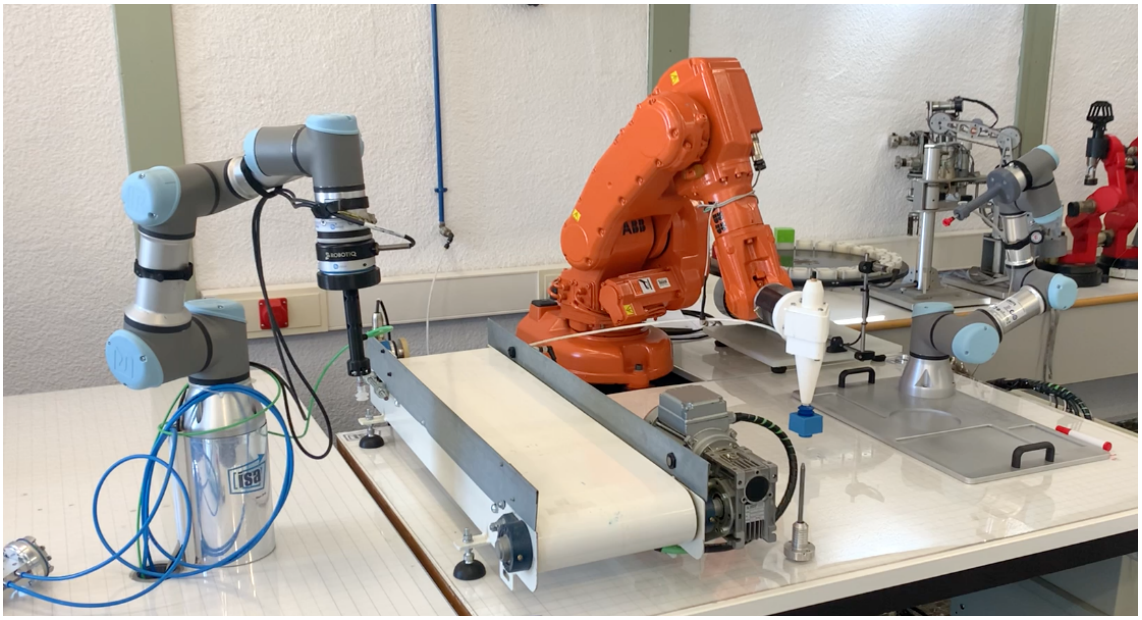

Figura 69. Simulación célula laboratorio (IV)

Cuando el ABB deja la pieza en la cinta, está comienza a moverse hasta que el sensor del final de la cinta se active y entonces se detenga.

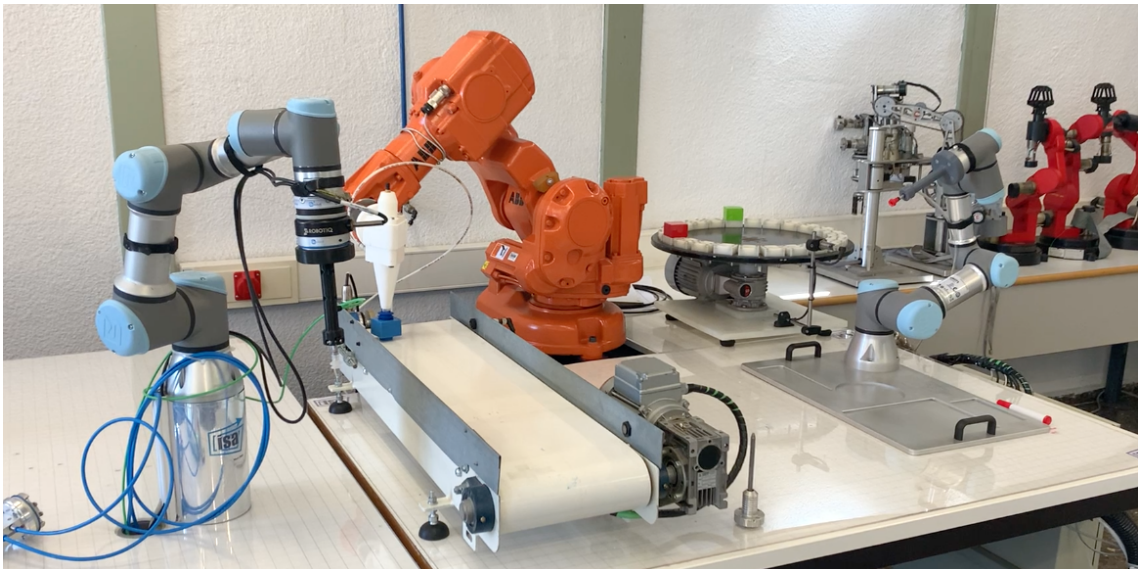

Figura 70. Simulación célula laboratorio (V)

Una vez que la pieza esta situada en el final de la cinta, está en posición óptima para que la cámara tome una fotografía de esta para realizar el procesamiento de imagen. A diferencia de la célula simulada, en el laboratorio no hay posición intermedia para el procesamiento de imagen ya que no existe un sensor en la mitad de la cinta. Sin embargo, recortando la imagen adecuadamente el algoritmo no presenta fallos en el procesamiento.

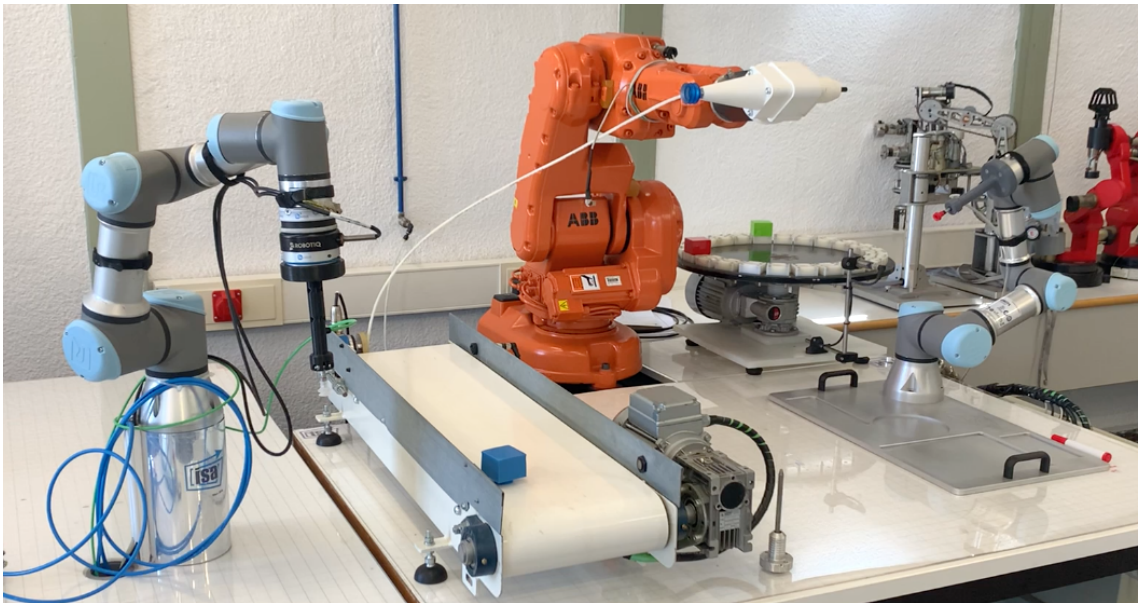

Figura 71. Simulación célula laboratorio (VI)

El robot UR3 espera hasta que el servidor le manda la información necesaria para proceder a realizar el paletizado de la pieza en función del color detectado. Mientras tanto, el robot UR3e se dispone a coger la siguiente pieza de la mesa giratoria para optimizar los tiempos del ciclo de trabajo de la célula.

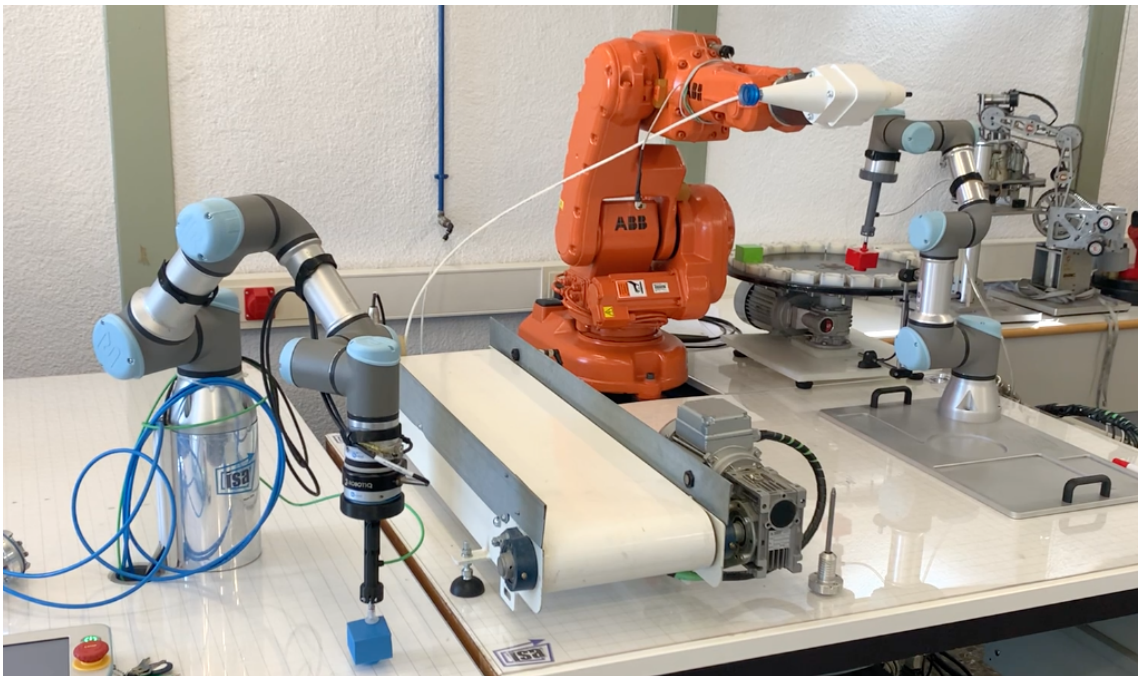

Figura 72. Simulación célula laboratorio (IV)

A los 2 minutos 44 segundos la célula es capaz de procesar la primera pieza. El tiempo es bastante más elevado que en simulación debido a las limitaciones en el alcance del robot ABB IRB140, la complejidad que supone programar sus posiciones reales y que el ABB trabaja a 50 mm/s para que no haya ningún problema en el laboratorio.

Como este trabajo es de ámbito académico y tiene como objetivos principales la adquisición de conocimientos para desarrollar una célula robotizada tanto en simulación como en el laboratorio, no se ha considerado necesario optimizar los tiempos de trabajo de la célula. Aun así, si se quisiera trabajar en un ámbito profesional, la optimización de dichos tiempos sería totalmente necesaria para reducir tiempos y costes.

Finalmente, toda la estación tarda 7 minutos y 44 segundos en procesar las tres piezas, lo cual es un tiempo bastante extenso, pero como se ha comentado antes, la optimización de este no es un objetivo del trabajo.

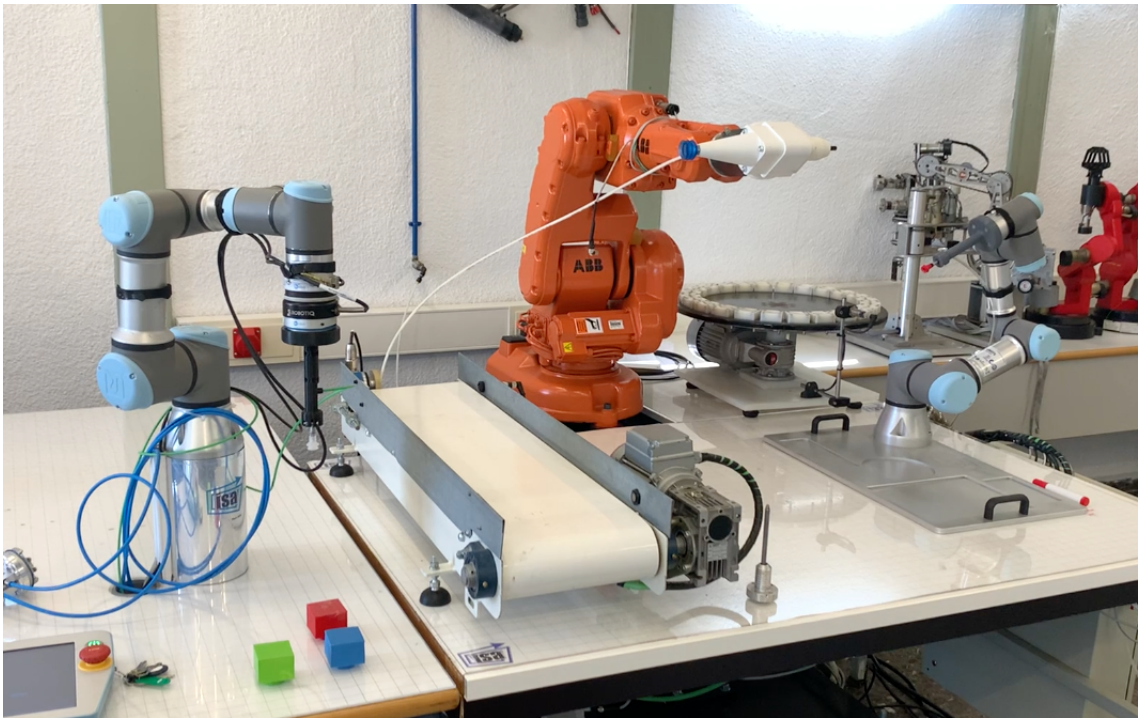

Figura 73. Simulación célula laboratorio (IV)

Por lo tanto, el funcionamiento de la célula robotizada en el laboratorio quedaría totalmente demostrado, sin presentar ningún tipo de problema de colisiones.

Para ejecutar el ensayo en esta estación, se ha creado un manual de uso en el ANEXO C: Manual Utilización Célula Laboratorio, en el que se explica como hay que proceder para ejecutar este ensayo.

Para terminar el documento de la memoria, en el siguiente apartado se expondrán las conclusiones obtenidas de la realización de este.

CAPÍTULO IV: CONCLUSIONES

Para finalizar la memoria del trabajo, se van a exponer las principales conclusiones obtenidas de la realización de este, junto con mejoras posibles para trabajos futuros.

4.1. Conclusiones

Una vez terminado este trabajo, se puede afirmar que todos los objetivos planteados han sido cumplidos exitosamente. Además, se han adquirido y aplicado conocimientos relacionados con la robótica, visión artificial, sistemas de comunicación y programación.

En primer lugar, se ha podido llevar a cabo el diseño y desarrollo de una célula robotizada totalmente funcional usando el software de RoboDK. Su estilo de programación gráfica junto con la posibilidad de escribir ciertos programas en Python ha facilitado mucho la programación de la célula completa. Por lo tanto, RoboDK ha demostrado ser un software bastante útil y potente para el desarrollo de una célula robótica.

En segundo lugar, se ha llevado a cabo la implementación de la célula simulada en la célula del laboratorio, exportando los programas de RoboDK con la ayuda del post-procesador. Sin embargo, aunque esta herramienta es muy útil, hay que programar manualmente a mano algunas cosas como las señales digitales y las posiciones del robot, aunque resulta bastante útil para conformar un código base a la hora de programar el robot.

En tercer lugar, se ha conseguido implementar un sistema de visión artificial que detecte el color y el ángulo de una pieza cuadrada. Hay que destacar las diferencias que se han presentado a la hora de aplicar los algoritmos de visión desarrollados en RoboDK con respecto a la cámara real, que han supuesto realizar diversas correcciones para que el algoritmo funcionase adecuadamente.

En cuarto lugar, se ha podido implementar un sistema de comunicaciones TCP/IP entre un cliente y un servidor, y enviar información a través de sockets. Esto se ha logrado tanto en la simulación como en el laboratorio pudiendo establecer una conexión entre el robot UR3 y el ordenador personal del laboratorio. Además, se ha desarrollado una interfaz gráfica para presentar la información de una manera más visual y poder interactuar con ella.

Por último, la realización de este Trabajo Fin de Máster ha permitido la familiarización con el entorno de la robótica actual, aplicando conocimientos de diversos campos tecnológicos para poder diseñar, desarrollar y aplicar una célula robotizada totalmente funcional.

4.2. Trabajo Futuros

Para concluir esta memoria, se van a plantear posibles trabajos futuros siguiendo la línea de este Trabajo Fin de Máster.

El software de RoboDK ha supuesto una herramienta muy útil para la realización del trabajo, en el cual se han usado muchas de sus herramientas como la programación gráfica, la API de Python o los post-procesadores, pero hay algunas que no se han podido aplicar para no extender demasiado el trabajo. Por ejemplo, RoboDK cuenta con una herramienta que permite conectar directamente los robots reales al software, por lo que, de esta forma, la programación de la célula sería aún más rápida. En futuras mejoras de este trabajo se podría implementar dicha herramienta.

El sistema de visión artificial ha resultado ser muy útil para el paletizado automático de las piezas, por lo que se podría seguir trabajando en este algoritmo para mejorar la calidad el paletizado. Una posible vía sería incluir la detección de la posición de las piezas en la cinta para que el robot agarrase la pieza justo por el centro de esta.

Por último, un posible camino con el que seguir el trabajo es la optimización de los ciclos de trabajo de la célula robotizada. Para ello habría que ajustar más los tiempos entre la activación y desactivación de señales para evitar tiempos ociosos y optimizar las trayectorias de los robots.

La programación de trayectorias con los robots UR3s ha sido bastante sencilla ya que al ser robots colaborativos permiten moverlos manualmente para encontrar la posición óptima. Sin embargo, la programación de trayectorias del ABB ha resultado ser más compleja ya que se debe mover el robot al punto deseado mediante el uso de la pantalla táctil, lo que al ser un robot de 6 ejes en ocasiones genera posiciones demasiado complejas. En el caso de la célula del laboratorio se ha visto que el ABB tenía que hacer trayectorias muy complejas debido a problemas con el alcance de la célula, por lo que generar trayectorias más sencillas podría ayudar a optimizar los tiempos de ciclo de trabajo.

BIBLIOGRAFÍA

[1] *MundoPlast. (1 de octubre de 2020). Las ventas de robots industriales, a la baja en 2019*. https://mundoplast.com/ventas-robots-industriales-2019/

[2] Lalinde, D. E. (s.f.). La cirugía mínimamente invasiva es cada vez más común. *Dr. Lalinde*. Recuperado el 25 de agosto de 2021 de https://drlalinde.es/blog/la-cirugia-minimamente-invasiva-es-cada-vez-mas-comun/

[3] *Aula21 | Formación para la Industria.(2 de enero de 2020).* 7 Tendencias de la Robótica Industrial que debes conocer en 2020. http://www.cursosaula21.com/robotica-industrial-tendencias/

[4] *Revista de Robots. (25 de junio de 2021).* Sistemas de Visión Artificial industrial para robots y empresas. https://revistaderobots.com/robots-y-robotica/vision-artificial-industrial/

[5] Blog Logicbus. (17 de junio de 2019). *Protocolos de comunicación industriales*. https://www.logicbus.com.mx/blog/protocolos-de-comunicacion-industriales/

[6] Capetillo, J. (22 de febrero de 2017). *¿Cómo funciona la arquitectura cliente-servidor?* CEESA, S.A. https://ceesa.com/como-funciona-arquitectura-cliente-servidor/

[7] *RoboDK. (s.f.). Primeros Pasos—Documentación RoboDK*. Recuperado el 29 de agosto de 2021 de https://robodk.com/doc/es/Getting-Started.html

[8] Kang & Atul. (s. f.). *Angle of rotation by cv2.minAreaRect()*. TheAILearner. Recuperado 5 de septiembre de 2021 de https://theailearner.com/tag/angle-of-rotation-by-cv2-minarearect/

Parte II

PRESUPUESTO

CAPÍTULO I: PRESUPUESTO

1. Introducción

En este apartado del trabajo se pretende elaborar un presupuesto detallado de lo que ha supuesto la elaboración del proyecto.

2. Presupuesto detallado

Los costes de realizar el trabajo se pueden catalogar en costes debidos a amortizaciones del equipo necesario para llevar a cabo el proyecto, los costes de la materia prima utilizada y los costes de recursos humanos. La suma de estos costes viene dada por el presupuesto de ejecución material, y junto con los gastos generales, el beneficio industrial y el IVA se obtendrá el presupuesto total del proyecto.

2.1. Costes de amortización

El uso del equipo informático, paquetes de software, robots del laboratorio y mecanismos de este suponen unos cortes de amortización a la hora de realizar el proyecto. Para calcular dicho coste se aplica la siguiente fórmula.

$$Coste\ amortización = Precio(€) \cdot \frac{periodo\ de\ uso\ (meses)}{periodo\ de\ amortización\ (meses)}$$

Los periodos de amortización del equipo usado para realizar el trabajo de fin de máster se basan en la tabla de amortizaciones publicada por el BOE en el Real Decreto 1777/2004.

Descripción	Precio (€)	Uso (Meses)	Amortización (Años)	Importe (€)
Ordenador Portátil	1000	6	6	83,33
Impresora 3d BQ Hephestos 2	800	6	6	66,67
Licencia RoboDK	2995	6	6	249,58
Robot ABB IRB140	20000	6	12	833,33
Robot UR3	15000	6	12	625,00
Robot UR3e	15000	6	12	625,00
Cámara Hercules Classic	50	6	8	3,13
Cinta Transportadora	600	6	10	30,00
Mesa Giratoria	350	6	10	17,50
			TOTAL	**2.533,54**

Tabla 10. Costes de amortizaciones.

2.2. Costes materiales

En esta parte del presupuesto, se incluye toda la materia prima empleada para realizar el trabajo. En concreto, se incluyen las bobinas de filamento PLA empleadas para fabricar las piezas necesarias, distinguiendo entre las bobinas de colores de menor tamaño (300 gramos) con las que se han fabricado las piezas manipuladas, y la bobina blanca de mayor tamaño (1 kg) con la que se ha fabricado el soporte de la doble herramienta.

También se incluye la tornillería necesaria empleada para fijar el soporte ventosa y el soporte delineador al soporte diseñado, y a su vez para fijar el conjunto a la brida del robot. En particular, se han empleado varios tornillos, tuercas y arandelas del tamaño M6.

Descripción	**Precio (€)**	**Unidad**	**Importe (€)**
Bobina Filamento PLA Rojo/Verde/Azul (300 g)	10	3	30,00
Bobina Filamento PLA Blanco (1 kg)	20	1	20,00
Tornillería M6 (tuercas, arandelas y tornillos)	1	1	1,00
		TOTAL	**51,00**

Tabla 11. Costes de materiales.

2.3. Costes Recursos Humanos

Para calcular los costes de los recursos humanos empleados en el trabajo, se asignarán el número de horas empleadas para cada tarea del proyecto y con la tarifa horaria del personal empleado se obtendrán el importe total de cada tarea realizada.

Las tareas realizadas en el proyecto han sido realizas por un Ingeniero Junior, en este caso el alumno que realiza el proyecto, con una tarifa horaria de 35 €/hora según establece el convenio.

Tarea	Tarifa (€/h)	Horas	Importe (€)
Estudio Bibliográfico	35	20	700,00
Aprendizaje Entorno RoboDK	35	35	1.225,00
Diseño Célula Robotizada	35	10	350,00
Diseño CAD Componentes	35	15	525,00
Programación Célula RoboDK	35	45	1.575,00
Programación Algoritmo de Visión	35	25	875,00
Programación Comunicaciones	35	20	700,00
Programación Robots Laboratorio	35	30	1.050,00
Impresión 3D y Ensamblaje	35	10	350,00
Ensayos	35	10	350,00
Redacción Informe	35	80	2.800,00
	TOTAL	**300**	**10.500,00**

Tabla 12. Costes Ingeniero Junior.

Además, también se tendrán en cuenta el trabajo realizado por el tutor de proyecto, al que se le considerará Ingeniero Senior, con una tarifa de 65€/h según establece el convenio.

Tarea	Tarifa (€/H)	Horas	Importe (€)
Tutorización Y Supervisión	65	20	1.300,00
Supervisión Ensayo Laboratorio	65	10	650,00
	TOTAL	**30**	**1.950,00**

Tabla 13. Costes Ingeniero Senior.

Los costes totales de los recursos humanos empleados para la realización del trabajo se muestran en la siguiente tabla.

Categoría	Horas	Importe (€)
Ingeniero Junior	300	10.500,00
Ingeniero Senior	30	1.950,00
	TOTAL	**12.450,00**

Tabla 14. Costes Recursos Humanos.

2.4. Presupuesto Total

Una vez calculados todos los costes directos del proyecto, se podrá calcular el presupuesto total de lo que supone la realización de este. La suma de los costes anteriores se denomina presupuesto de ejecución material.

A esta cantidad, hay que sumarle un porcentaje en concepto de gastos generales (13%) y un porcentaje del beneficio industrial (6%), obteniendo de esta forma el presupuesto de ejecución por contrata.

Finalmente, se suma a esta cantidad el IVA (21%) para obtener el presupuesto total de la realización del proyecto.

Categoría	**Coste (€)**
Costes Amortización	2.533,54 €
Costes Materiales	51,00 €
Costes Recursos Humanos	12.450,00 €
Presupuesto de ejecución material (€)	**15.034,54 €**
Gastos generales (13%) (€)	1.954,49 €
Beneficio industrial (6%) (€)	902,07 €
Presupuesto de ejecución por contrata (€)	**17.891,10 €**
IVA (21%)	3.757,13 €
TOTAL	**21.648,24 €**

Tabla 15. Presupuesto total proyecto.

El presupuesto total del proyecto asciendo a la cantidad de: **VEINTIÚN MIL SEISCIENTOS CUARENTA Y OCHO CON VEINTICUATRO EUROS**

Parte III

ANEXOS

ANEXO A: Ejecución de simulación HTML de RoboDK

En este anexo se va a explicar como poder ejecutar una simulación HTML creada a partir de RoboDK. El proceso es bastante sencillo y no requiere de la instalación de ningún programa extra para su ejecución más que nuestro navegador web.

En primer lugar, se descarga el archivo de Drive enlazado mediante el QR y se guarda en el equipo en una carpeta a su elección.

Figura 74. Ejemplo de ubicación archivo simulación.

A continuación, se selecciona el archivo con el botón derecho y se pulsa sobre Abrir con y se selecciona eñ navegador predeterminado (en este caso Google Chrome). Hecho esto nos deberá aparecer esta pantalla en nuestro navegador.

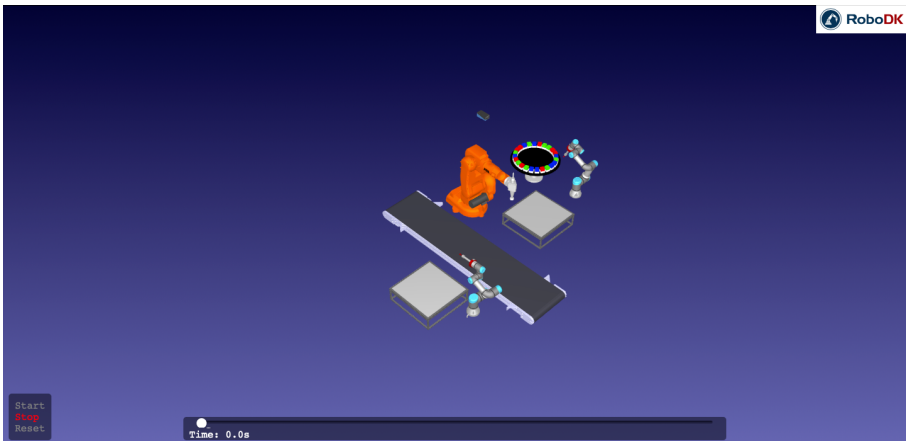

Figura 75. Pantalla de visualización previa a la simulación.

Tardará unos segundos en cargar la simulación y una vez cargada se podrá iniciar la simulación, pararla, reiniciarla y viajar al momento que se desee mediante la barra de tiempo.

Además, si se pulsa la rueda del ratón se podrá desplazarnos por la célula y si se pulsa el botón izquierdo se podrá cambiar la orientación de la vista de la célula.

ANEXO B: Manual Utilización Célula RoboDK

En este Anexo se va a explicar como hay que proceder para ejecutar la célula de simulación desarrollada en el software de RoboDK.

En primer lugar, es necesario la instalación de RoboDK. Aunque el software es de pago, se puede descargar una licencia básica gratis la cual ofrece una prueba de un mes con todas las funciones del programa.

En segundo lugar, es necesario tener instalado Python 3 en el ordenador en el que se vaya a ejecutar el programa. Además de ello, se deberá tener una serie de librerías de Python instaladas para que los programas funcionen correctamente. Estas librerías se muestran a continuación.

- OpenCV
- Numpy
- Socket
- Math
- Time
- Datetime
- Tempfile

Para comenzar el programa, se deberá pulsar dos veces sobre el programa *MainProgram* y comenzará la ejecución de este. No hay que preocuparse por recolocar los robots ni los objetos ya que el programa ejecuta en primer lugar el programa *Reemplazar Objetos*, que coloca todos los elementos en su posición.

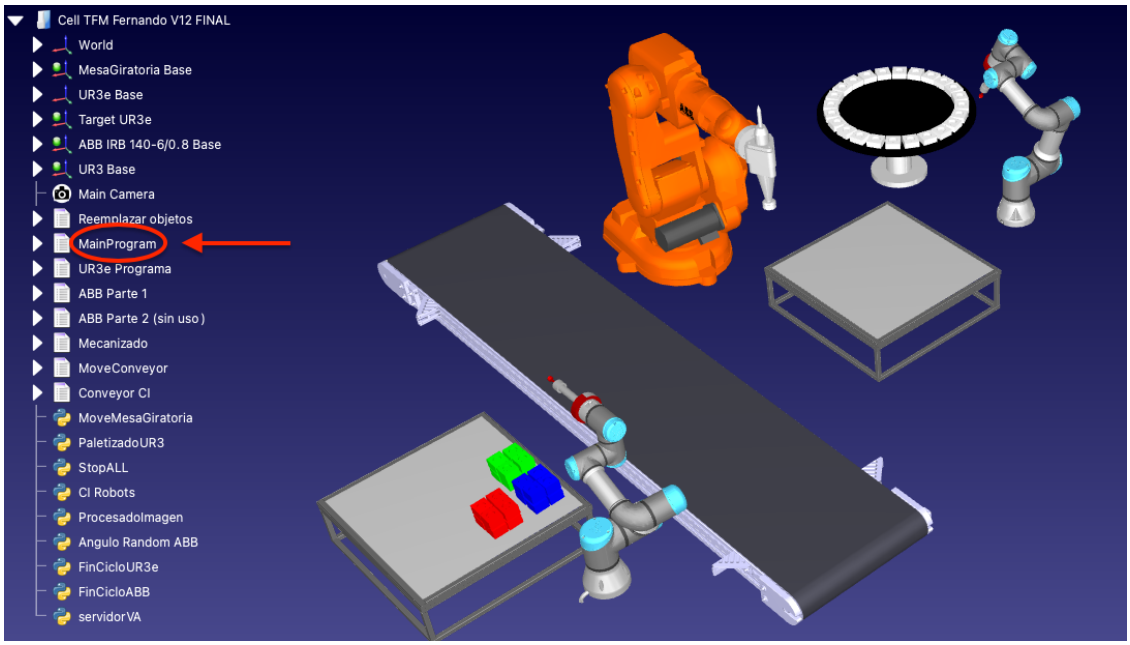

Figura 76. Indicación de la situación de MainProgram en la célula de RoboDK.

Una vez en ejecución si se desea parar la simulación, se deberá pulsar dos veces sobre el programa *StopALL*. Después de ello, si se quiere volver a lanzar la simulación, se pulsará de nuevo sobre *MainProgram*.

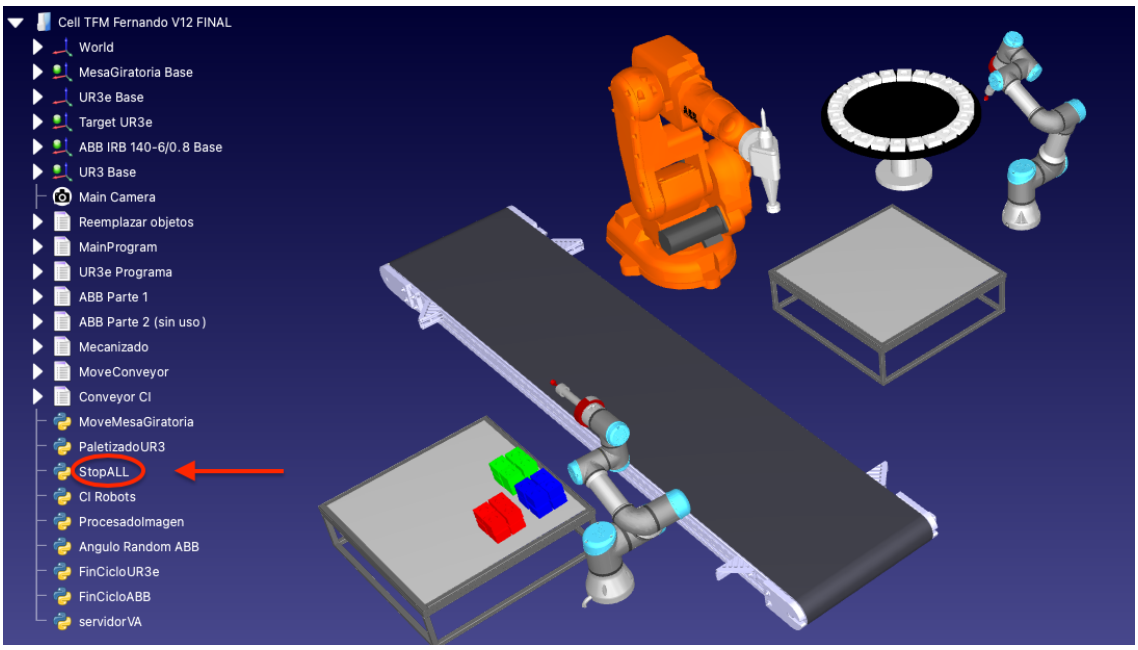

Figura 77. Indicación de la situación de StopALL en la célula de RoboDK.

ANEXO C: Manual Utilización Célula Laboratorio

En este Anexo se va a explicar como hay que proceder para ejecutar correctamente la célula robotizada en el laboratorio.

Antes de empezar, es necesario tener instalado en un PC el software RobotStudio, para ejecutar los programas en el ABB IRB140 del laboratorio y el software PyCharm, para ejecutar el servidor que procesa y envía las imágenes de la cámara.

En primer lugar, se deberá enchufar todos tres controladores de los robots, así como la alimentación de la mesa giratoria y la cinta transportadora que se encuentra en la parte inferior de la mesa del laboratorio. También hay que asegurarse de que el sistema de aire comprimido tenga la válvula abierta para alimentar a los robots.

A continuación, se deberá encender el PC que hay en frente de la célula robotizada e iniciar en la partición de Windows el programa PyCharm. En este software, se deberá cargar *VisionArtificialTFM* donde se encuentra el código para ejecutar el servidor y para realizar el procesamiento de imagen. Además, se deberá asegurarse de que el USB de la cámara de la célula esta enchufada al PC.

El siguiente paso será cargar todos los programas en los robots. Para ello se deberá hacer uso de un USB en el caso de los robots de Universal Robots, y para el ABB se deberá ejecutar RobotStudio y transferir el programa al controlador.

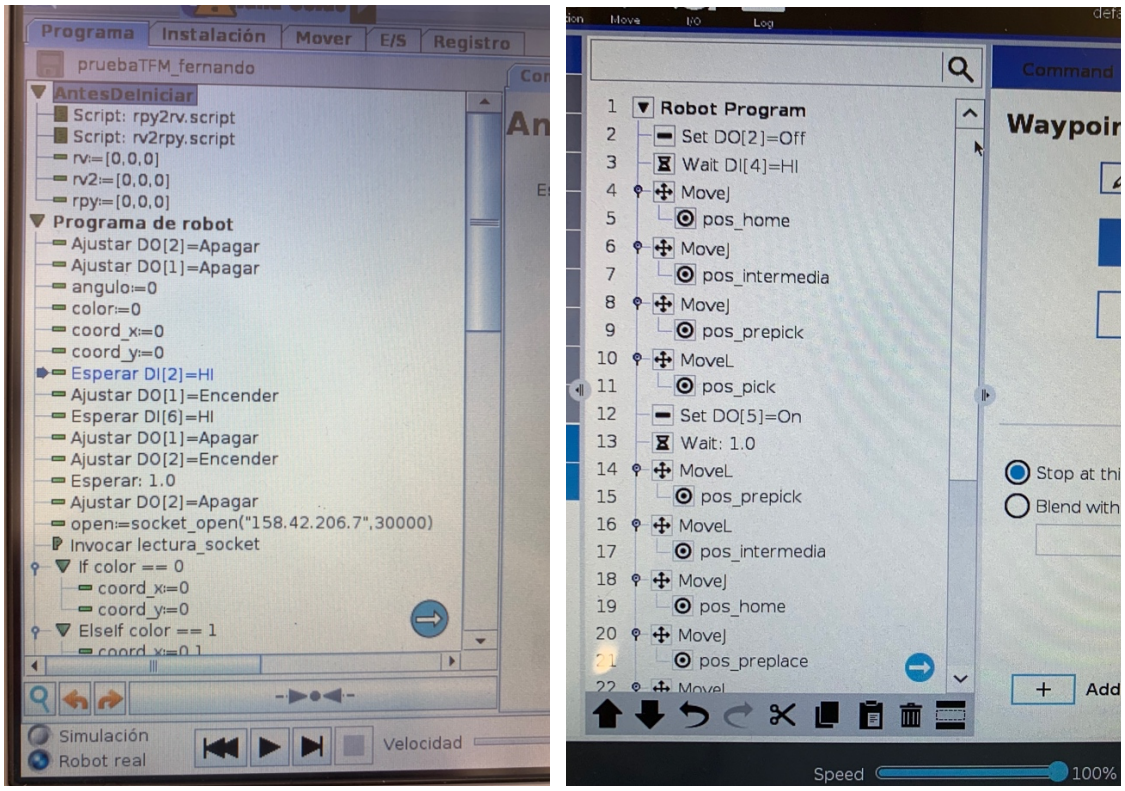

Figura 78. Programas cargados en la pantalla táctil del UR3 y UR3e respectivamente.

Después de ello, se deberá ejecutar el programa GUI_Servidor_Captura en PyCharm. Este último mostrará una interfaz gráfica, en la cual se debe pulsar el botón de iniciar para iniciar el servidor.

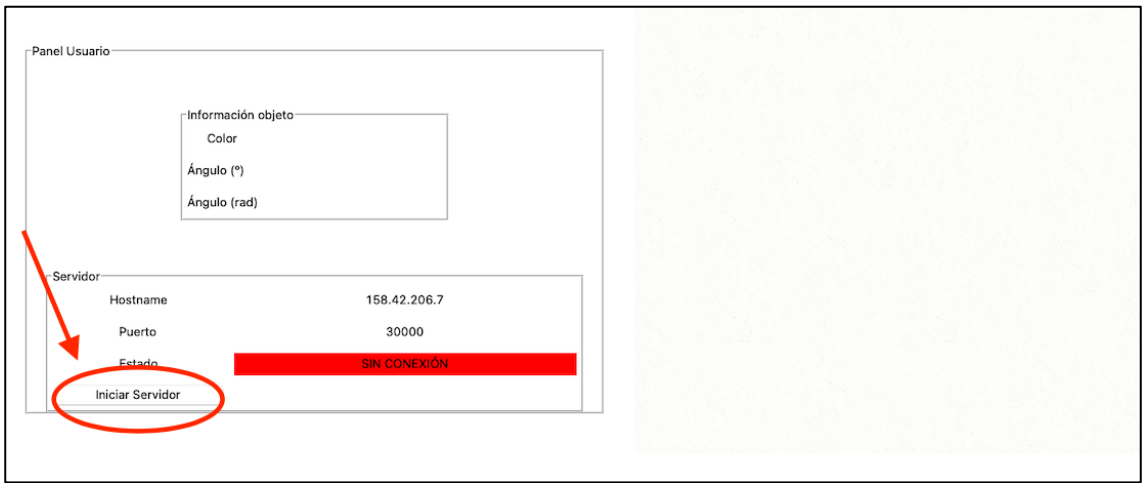

Figura 79. Indicación de la situación del botón para iniciar el servidor en PyCharm.

Antes de ejecutar los programas, hay que asegurarse de que no hay ningún elemento en la célula que pueda colisionar con los robots y hay que colocar las piezas manipuladas en la mesa giratoria.

Una vez hecho esto puede comenzar la ejecución de la célula robotizada. El primer programa que hay que ejecutar es el del UR3, ya que reiniciará las señales digitales correspondientes para que no haya colisiones entre el UR3e y el ABB. A continuación, se pone en marcha el programa del UR3e, el esperará a que se active la señal digital correspondiente y por último se ejecutará el programa del ABB, que dará comienzo con el giro de la mesa giratoria.

Si durante la ejecución de los programas hubiese algún tipo de fallo o posible colisión, se deberá pulsar la seta de emergencia correspondiente al robot que esta fallando para evitar accidente. Dada la complejidad de controlar los tres robots simultáneamente, es recomendable que el ensayo se haga con al menos dos personas en el laboratorio.

ANEXO D: Código de los programas utilizados

En este anexo se va a incluir el código programado en todo el proyecto, y para ello se va a diferenciar tres partes principales: código Python RoboDK, código en PyCharm para el algoritmo de Visión Artificial y código de los robots reales.

D.1. Código Programas RoboDK

En este primer apartado se van a presentar el código de Python de todos los programas empleados para configurar la célula de RoboDK.

MoveMesaGiratoria

```
from robolink import *    # RoboDK API
from robodk import *      # Robot toolbox
RDK = Robolink()
# importamos el objeto de la mesa
mesa_giratoria=RDK.Item('MesaGiratoria')

# obtenemos lo posición de las juntas y las modificamos
joints=mesa_giratoria.Joints().list()
joints[0]=joints[0]+15

# realizamos el movimiento de 15º
mesa_giratoria.MoveJ(joints)
```

CI Robots

```
from robolink import *    # RoboDK API
from robodk import *      # Robot toolbox
RDK = Robolink()

# importación robots
robot = RDK.Item('UR3')
ABB = RDK.Item('ABB IRB 140-6/0.8')
UR3e = RDK.Item('UR3e')

# declaración de sus posiciones home
home=RDK.Item('Home UR3')
home_ABB=RDK.Item('Home ABB')
home_UR3e=RDK.Item('Home')

# movimiento articular a las posiciones de home
robot.MoveJ(home)
ABB.MoveJ(home_ABB)
UR3e.MoveJ(home_UR3e)
```

Angulo Random ABB

```
from robolink import *    # RoboDK API
from robodk import *      # Robot toolbox
RDK = Robolink()

import random

## ROBOT Y HERRAMIENTA
ABB = RDK.Item('ABB IRB 140-6/0.8')
doble_herramientaABB=RDK.Item('Ensamblaje Herramienta Doble v3')

## POSICIONES
home=RDK.Item('Home ABB')
preplace=RDK.Item('Preplace Conveyor')
place=RDK.Item('Place Conveyor')
frame_conveyor=RDK.Item('Frame Conveyor')

## PROGRAMAS
moveConveyor=RDK.Item('MoveConveyor')

# generación ángulo aleatorio y lo aplicamos a la posicion de place
angulo=random.randint(0,90)
place=place.Pose()*rotx(-angulo*pi/180)

# ejecutamos el proceso de place el robot ABB
ABB.MoveJ(home)
ABB.MoveJ(preplace)
ABB.MoveL(place)

doble_herramientaABB.DetachAll(frame_conveyor)

ABB.MoveL(preplace)
ABB.MoveJ(home)

# movemos la cinta
moveConveyor.RunProgram()
```

StopALL

```
from robolink import *    # RoboDK API
from robodk import *      # Robot toolbox
RDK = Robolink()

# Programas
UR3ePrograma=RDK.Item('UR3e Programa')
ABBParte1=RDK.Item('ABB Parte 1')
ABBParte2=RDK.Item('ABB Parte 2')
main=RDK.Item('MainProgram')
paletizadoUR3=RDK.Item('PaletizadoUR3')
server=RDK.Item('servidorVA')

# Paro de programas
UR3ePrograma.Stop()
ABBParte1.Stop()
ABBParte2.Stop()
main.Stop()
paletizadoUR3.Stop()
server.Stop()
```

FinCicloUR3e

```
from robolink import *    # RoboDK API
from robodk import *      # Robot toolbox
RDK = Robolink()

UR3e = RDK.Item('UR3e')
UR3e.setDO("Fin UR3e",1)
```

FinCicloABB

```
from robolink import *    # RoboDK API
from robodk import *      # Robot toolbox
RDK = Robolink()

UR3e = RDK.Item('UR3e')
UR3e.setDO("Fin UR3e",1)
```

ServidorVA

```
from robolink import *    # RoboDK API
from robodk import *      # Robot toolbox
RDK = Robolink()

import cv2
import datetime
import time
import tempfile
import numpy as np
import socket

## FUNCIONES
def procesar_imagen():

    # definimos la ruta en la que se guardará la imagen
    date_str = datetime.datetime.now().strftime("%Y-%m-%d-%H-%M-%S")
    out_path = RDK.getParam('PATH_DESKTOP')+'\TFM FCC\ScreenShots   Virtual
Camera'+'\Image_'+date_str+'.png'

    # tomamos una captura de la imagen
    RDK.Cam2D_Snapshot(out_path)

    # importamos la imagen mediante opencv y hacemos una copia
    imagen=cv2.imread(out_path)
    imagen_angulo=imagen.copy()

    # convertimos la imagen a la escala de color HSV
    imagenHSV=cv2.cvtColor(imagen,cv2.COLOR_BGR2HSV)

    # DEFINICIÓN FILTROS COLORES
    verdeBajo=(30,100,100)
    verdeAlto=(80,255,255)
    mask_verde=cv2.inRange(imagenHSV,verdeBajo,verdeAlto)

    azulBajo=(90,100,100)
    azulAlto=(120,255,255)
    mask_azul=cv2.inRange(imagenHSV,azulBajo,azulAlto)

    rojoBajo=(0,100,100)
    rojoAlto=(20,255,255)
    mask_rojo=cv2.inRange(imagenHSV,rojoBajo,rojoAlto)

    mask=[mask_verde,mask_azul,mask_rojo]

    color_obj=""
```

```
    # mediante la búsqueda de contorno en cada máscara, encontramos el color
     for idx,item in enumerate(mask):
         (contornos,
jerarquia)=cv2.findContours(item,cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)
         img_cont = cv2.drawContours(imagen,contornos,-1,(0,0,0),5)
         if idx == 0 and len(contornos)==1:
             color_obj="verde"
             contornos_buenos=contornos
         if idx == 1 and len(contornos)==1:
             color_obj="azul"
             contornos_buenos=contornos
         if idx == 2 and len(contornos)==1:
             color_obj="rojo"
             contornos_buenos=contornos

     ## DETECCIÓN ÁNGULO

     # calculamos el ángulo a partir de las características extraídas en el
contorno
     rotatedRect = cv2.minAreaRect(cnt[0])
     (cx, cy), (width, height), angle = rotatedRect
     angle=90-angle

     # enviamos informacion en el formato de lectura del cliente
     if color_obj != "" and 0 <= angle <= 90:
         infoUR3 = '('+color_obj + ',' + str(round(angle,3) )+ ')'
         return infoUR3
     else:
         return "error"

 ## ROBOTS Y MECANISMOS
 conveyor= RDK.Item('Conveyor Belt')

 # Posiciones Conveyor
 pos_ini_conv=RDK.Item('Inicio Conveyor')
 pos_camara_conv=RDK.Item('Camara Processing')
 pos_fin_conv=RDK.Item('Final Conveyor')

 ## DEFINICIÓN SERVIDOR
 host = socket.gethostname() # nos da el nombre de la máquina
 port = 12345
 serv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
 serv.bind((host, port))
 serv.listen(5)
```

```
# Bucle para enviar la información
while True:
    conn, addr = serv.accept()
    from_client = ''
    while True:
        data = conn.recv(4096)
        if not data: break
        from_client += data.decode('utf-8')
        print(from_client)
        conveyorMat=Pose_2_TxyzRxyz(conveyor.Pose())
        if conveyorMat[0] == 650.0:
            info = procesar_imagen()
            conn.send(info.encode('utf-8'))

    conn.close()
    print('client disconnected')
```

PaletizadoUR3

```
from robolink import *    # RoboDK API
from robodk import *      # Robot toolbox
RDK = Robolink()

import numpy as np
from math import atan2, cos, sin, sqrt, pi
import socket

## FUNCIONES

def conveyor_2_mesa():

    robot.setSpeed(50)
    robot.MoveJ(prepick)
    robot.MoveL(pick)
    ventosa_UR3.AttachClosest()
    conveyor_CI.RunProgram()
    robot.MoveL(prepick)
    robot.MoveJ(preplace)

def conexion_servidor():

    host = socket.gethostname() # nos da el nombre de la máquina
    port = 12345
```

```
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((host, port))
    client.send("I am CLIENT".encode('utf-8'))
    from_server = client.recv(4096)
    client.close()
    msg=from_server.decode('utf-8')
    return msg

# Programas
conveyor_CI=RDK.Item('Conveyor CI')
stop_all=RDK.Item('StopALL')
ci_robots=RDK.Item('CI Robots')
procesar_img=RDK.Item("ProcesadoImagen")
fin_cicloUR3e=RDK.Item("FinCicloUR3e")
fin_cicloABB=RDK.Item("FinCicloABB")

# Robots y mecanismos
robot = RDK.Item('UR3')
ventosa_UR3 = RDK.Item('Ventosa UR3')
conveyor= RDK.Item('Conveyor Belt')
camera = RDK.Item('Main Camera')
mesa_giratoria=RDK.Item('MesaGiratoria')

# Objetos
mesa1=RDK.Item('Mesa 1')
mesa=RDK.Item('Mesa 2')

# Posiciones Conveyor
pos_ini_conv=RDK.Item('Inicio Conveyor')
pos_camara_conv=RDK.Item('Camara Processing')
pos_fin_conv=RDK.Item('Final Conveyor')

# Posiciones Robot
home=RDK.Item('Home UR3')
prepick=RDK.Item('Prepick Pieza UR3')
pick=RDK.Item('Pick Pieza UR3')
preplace=RDK.Item('Preplace Pieza UR3')
ref_paletizado=RDK.Item('Referencia Paletizado')

## VARIABLES
ancho_pieza=43
dist_seguridad=5

objetos=0
objetos_totales=24
```

```
 pieza_roja=0
 pieza_verde=0
 pieza_azul=0

 flag=True
 flag2=True

 # BUCLE PRINCIPAL PALETIZADO UR3

 while objetos<objetos_totales:

     # verificamos las condiciones del fin del ciclo de los otros robots
     joints=mesa_giratoria.Joints().list()
     mesa1_childs=mesa1.Childs()

     if joints[0] == objetos_totales*15 and flag==True:
         fin_cicloUR3e.RunProgram()
         flag=False

     if joints[0] == objetos_totales*15 and flag2==True and
len(mesa1_childs)==0:
         fin_cicloABB.RunProgram()
         flag2=False
 # verificamos si el objeto se encuentra en la posicion de procesamiento de
imágenes
     conveyorMat=Pose_2_TxyzRxyz(conveyor.Pose())
     x=0
     y=0
     z=0
     if conveyorMat[0] == 650.0:

 # iniciamos conexion al servidor y verificamos que la información es válida

         infoVA = conexion_servidor()

         if infoVA != "" and infoVA != "error":

 # se extrae de la string recibida del servidor los parámetros del color y
el ángulo
             s=infoVA.split('(')[1]
             s=s.split(",")
             color_obj=s[0]
             s=s[1].split(')')
             angulo=float(s[0])
```

```
 #Movemos la cinta a la posición pick del UR3 una vez la imagen ha sido
procesada
            conveyor.MoveJ(pos_fin_conv)

 # realizamos el pick y continuamos hasta la posicion de preplace
            conveyor_2_mesa()

 # en funcion del color y el número de pieza, seleccionamos unas
coordenadas de place correspondientes

            if color_obj=='azul':
                offset_x=0
                offset_y=0
                if pieza_azul == 3 or pieza_azul == 7:
                    x=0
                    y=0
                    z=0
                if pieza_azul == 1 or pieza_azul == 5:
                    x=-ancho_pieza-dist_seguridad
                if pieza_azul == 2 or pieza_azul == 6:
                    y=-ancho_pieza-dist_seguridad
                if pieza_azul == 0 or pieza_azul == 4:
                    x=-ancho_pieza-dist_seguridad
                    y=-ancho_pieza-dist_seguridad
                if pieza_azul > 3:
                    z = 35 #altura objeto - altura muesca en la pieza
                pieza_azul+=1

            if color_obj=='rojo':
                offset_x=130
                offset_y=0
                if pieza_roja == 3 or pieza_roja == 7:
                    x=0
                    y=0
                    z=0
                if pieza_roja == 1 or pieza_roja == 5:
                    x=-ancho_pieza-dist_seguridad
                if pieza_roja == 2 or pieza_roja == 6:
                    y=-ancho_pieza-dist_seguridad
                if pieza_roja == 0 or pieza_roja == 4:
                    x=-ancho_pieza-dist_seguridad
                    y=-ancho_pieza-dist_seguridad
                if pieza_roja > 3:
                    z = 35
                pieza_roja+=1
```

```
            if color_obj=='verde':
                offset_x=0
                offset_y=125
                if pieza_verde == 3 or pieza_verde == 7:
                    x=0
                    y=0
                    z=0
                if pieza_verde == 1 or pieza_verde == 5:
                    x=-ancho_pieza-dist_seguridad
                if pieza_verde == 2 or pieza_verde == 6:
                    y=-ancho_pieza-dist_seguridad
                if pieza_verde == 0 or pieza_verde == 4:
                    x=-ancho_pieza-dist_seguridad
                    y=-ancho_pieza-dist_seguridad
                if pieza_verde > 3:
                    z = 35
                pieza_verde+=1

 # con las coordenadas obtenidas, realizamos el pick sobre las mismas,
corrigiendo el ángulo de desviación
            target=ref_paletizado.Pose()*transl(x-offset_x,y-offset_y,-
z)*rotz((angulo)*pi/180)
            robot.MoveL(target)
            ventosa_UR3.DetachAll(mesa)

 # volvemos a la posicion de espera hasta que haya otra pieza disponible
            robot.MoveL(preplace)
            robot.MoveJ(prepick)
            objetos+=1

 # una vez terminado el paletizado se finaliza de ejecucion de los
programas activos y se lleva a los robots a condiciones iniciales
 stop_all.RunProgram()
 ci_robots.RunProgram()
```

D.2. Código PyCharm

En este apartado se va a adjuntar el código necesario para ejecutar el procesamiento de imagen de la cámara del laboratorio, el servidor y la interfaz gráfica creada. Este código cuenta con dos scripts principales.

GUI_Servidor_Captura

```
import math
from tkinter import *
from PIL import Image
from PIL import ImageTk
import cv2
import imutils
import socket

from procesamiento_imagen import procesamiento_img

def numToColor(num):
    if num == 0:
        return "azul"
    elif num == 1:
        return "rojo"
    elif num == 2:
        return "verde"
    else:
        return "error"

def server():
    global entry_estado, lblImgProcesada
    host = '158.42.206.7'  # Esta función nos da el nombre de la máquina
    port = 30000
    print(host)
    serv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    serv.bind((host, port))
    serv.listen(5)
    entry_estado.config(text="SERVIDOR ACTIVO")
    entry_estado.config(bg="green")
    root.update()

    while True:

        conn, addr = serv.accept()
        from_client = ''
        while True:
            data = conn.recv(4096)
            if not data: break
```

```
            from_client += data.decode('utf-8')
            print(from_client)
            if from_client == "Ready":
                print("procesando imagen")

                entry_estado.config(text="PROCESANDO IMAGEN")
                entry_estado.config(bg="yellow")
                root.update()

                info = procesamiento_img()
                print(info[0])

                aux = cv2.cvtColor(info[3], cv2.COLOR_BGR2RGB)
                img = Image.fromarray(aux)
                img = img.resize((640, 480), Image.ANTIALIAS)
                my_img = ImageTk.PhotoImage(img)

                lblImgProcesada.config(image=my_img)
                lblImgProcesada.image = my_img
                root.update()


                conn.send(info[0].encode('utf-8'))

                entry_color.config(text=info[1])
                entry_angulo.config(text=str(round(info[2], 3)))

entry_angulo_rad.config(text=str(round(info[2]*math.pi/180, 3)))

                entry_estado.config(text="EJECUTANDO PROGRAMA EN UR3")
                entry_estado.config(bg="orange")
                root.update()
                from_client = ""
        conn.close()
        print('client disconnected')
        break

## VARIABLES

global entry_estado, entry_angulo, entry_angulo_rad, entry_color,
lblImgProcesada
```

```
## INTERFAZ GRÁFICA
root = Tk()

img = Image.open("img_blanco.jpg")
img = img.resize((640,480),Image.ANTIALIAS)
my_img = ImageTk.PhotoImage(img)

lb_panel_usuario = LabelFrame(root, text="Panel Usuario")
lb_panel_usuario.grid(column=0, row=0, padx=30, pady=0)

# INFORMACION DEL OBJETO

lb_frame_datos = LabelFrame(lb_panel_usuario, text="Información objeto")
lb_frame_datos.grid(column=0, row=2, padx=0, pady=50)

lb_width = 20

lb_color = Label(lb_frame_datos, text="Color")
lb_color.grid(column=0, row=0, padx=0, pady=0)
entry_color = Label(lb_frame_datos, bg="white")
entry_color.grid(column=1, row=0, padx=5, pady=5, sticky=E)
entry_color.config(width=lb_width)

lb_angulo = Label(lb_frame_datos, text="Ángulo (º)")
lb_angulo.grid(column=0, row=1, padx=0, pady=0,sticky=W)
entry_angulo = Label(lb_frame_datos, bg="white")
entry_angulo.grid(column=1, row=1, padx=5, pady=5)
entry_angulo.config(width=lb_width)

lb_angulo_rad = Label(lb_frame_datos, text="Ángulo (rad)")
lb_angulo_rad.grid(column=0, row=2, padx=0, pady=0)
entry_angulo_rad = Label(lb_frame_datos, bg="white")
entry_angulo_rad.grid(column=1, row=2, padx=5, pady=5, sticky=W)
entry_angulo_rad.config(width=lb_width)

# SERVIDOR

lb_width2 = 40

lb_frame_server = LabelFrame(lb_panel_usuario, text="Servidor")
lb_frame_server.grid(column=0, row=3, padx=20, pady=0)

lb_host = Label(lb_frame_server, text="Hostname")
lb_host.grid(column=0, row=0, padx=0, pady=0)
entry_host = Label(lb_frame_server, bg="white", text="158.42.206.7")
entry_host.grid(column=1, row=0, padx=5, pady=5, sticky=W)
entry_host.config(width=lb_width2)
```

```
lb_puerto = Label(lb_frame_server, text="Puerto")
lb_puerto.grid(column=0, row=1, padx=0, pady=0)
entry_puerto = Label(lb_frame_server, bg="white",text="30000")
entry_puerto.grid(column=1, row=1, padx=5, pady=5, sticky=W)
entry_puerto.config(width=lb_width2)

lb_estado = Label(lb_frame_server, text="Estado")
lb_estado.grid(column=0, row=2, padx=0, pady=0)
entry_estado = Label(lb_frame_server, bg="white", text="SIN CONEXIÓN")
entry_estado.grid(column=1, row=2, padx=5, pady=5, sticky=W)
entry_estado.config(width=lb_width2)
entry_estado.config(bg="red")

btnIniciarServer = Button(lb_frame_server, text="Iniciar Servidor",
width=20, command=lambda: server())
btnIniciarServer.grid(column=0, row=3, padx=5, pady=5)

lblImgProcesada = Label(root,image=my_img)
lblImgProcesada.grid(column=1, row=0, columnspan=2)

root.mainloop()
```

Procesamiento_imagen

```
import datetime
import cv2
import numpy as np
import time

def save_img(img):
    date_str = datetime.datetime.now().strftime("%Y-%m-%d-%H-%M-%S")
    out_path = '/Users/fernandocuellocalvo/Desktop/TFM FCC/ScreenShots
Real Camera' '\Image_' + date_str + '.png'
    cv2.imwrite(out_path, img)

def procesamiento_img():

    video = cv2.VideoCapture(1)
    check, img = video.read()
    img_recortada = img[5:100, 268:370]

    # convertimos la imagen a la escala de color HSV
    imagenHSV = cv2.cvtColor(img_recortada, cv2.COLOR_BGR2HSV)
    imagenRGB = cv2.cvtColor(img_recortada, cv2.COLOR_BGR2RGB)
```

```
    # DEFINICIÓN FILTROS COLORES
    verdeBajo = (30, 100, 0)
    verdeAlto = (80, 255, 255)
    mask_verde = cv2.inRange(imagenHSV, verdeBajo, verdeAlto)

    azulBajo = (90, 100, 0)
    azulAlto = (120, 255, 255)
    mask_azul = cv2.inRange(imagenHSV, azulBajo, azulAlto)

    rojoBajo = (150, 0, 0)
    rojoAlto = (255, 255, 255)
    mask_rojoHSV_alto = cv2.inRange(imagenHSV, rojoBajo, rojoAlto)

    rojoBajo = (0, 150, 0)
    rojoAlto = (50, 255, 255)
    mask_rojoHSV_bajo = cv2.inRange(imagenHSV, rojoBajo, rojoAlto)

    mask_rojoHSV = cv2.add(mask_rojoHSV_bajo, mask_rojoHSV_alto)

    mask = [mask_verde, mask_azul, mask_rojoHSV]
    color_obj = ""
    num_contornos = np.zeros(3)
    contornos_buenos = []

    # mediante la búsqueda de contorno en cada máscara, encontramos el
color
    for idx, item in enumerate(mask):
        (contornos, jerarquia) = cv2.findContours(item, cv2.RETR_TREE,
cv2.CHAIN_APPROX_NONE)
        # img_cont = cv2.drawContours(img_recortada, contornos, -1, (0,
0, 0), 5)
        num_contornos[idx] = len(contornos)
        contornos_buenos.append(contornos)
    print(num_contornos)

    # buscar que color tiene el mayor número de contornos
    max_value = None
    max_idx = None
    for idx, num in enumerate(num_contornos):
        if (max_value is None or num > max_value):
            max_value = num
            max_idx = idx
```

```
    if max_idx == 0:
        color_obj = "verde"
    if max_idx == 1:
        color_obj = "azul"
    if max_idx == 2:
        color_obj = "rojo"

    print(color_obj)

    if color_obj == "":
        print("No se ha podido detectar el color")

    # almacenamos los contornos del color detectado
    contornos_buenos = contornos_buenos[max_idx]

    # DETECCIÓN ÁNGULO

    # filtramos los contornos pequeños
    for idx, contour in enumerate(contornos_buenos):
        area = cv2.contourArea(contour)
        # print(area)
        if area > 20:
            cnt = contornos_buenos[idx]
            id_contornos = idx;

    img_cont = cv2.drawContours(img_recortada, contornos_buenos,
id_contornos, (0, 0, 0), 5)
    cv2.imshow("contornos", img_recortada)
    cv2.waitKey(0)

    # calculamos coordenadas xy, altura, anchura y ángulo

    rotatedRect = cv2.minAreaRect(cnt)
    (cx, cy), (width, height), angle = rotatedRect
    angle = 90 - angle

    if color_obj != "" and 0 <= angle <= 90:
        infoUR3 = color_obj + '_' + str(round(angle,3))
        # print(infoUR3)
        return infoUR3
    else:
        return "error"
```

D.3. Código Robots Reales

Por ultimo, se va a adjuntar el código de los programas de los robots reales utilizados en el laboratorio. El código de los URs se mostrará en su versión en texto y no en su versión gráfica y el código del ABB se mostrará en su versión en RAPID.

ABB IRB140

```
     PROC main()

         setdo do_To_UR_1,0;
         SetDO TABLE_FWD, 0;
         SetDO conveyor_FWD, 0;
         SetDO conveyor_BWD, 0;
         setdo GRIPPER_CLOSE,0;
         movej pos_segura, v50, fine, doble_ploter\WObj:=wobj0;

         WHILE true DO
          SetDO TABLE_FWD, 1; ! Iniciamos el giro de la mesa
          WaitDI TABLE_OBJ_SEN,1; ! Giramos hasta que la mesa alcanza una
casilla nueva
          SetDO TABLE_FWD, 0;
            !MoveAbsJ jphome\NoEOffs, v1000, fine, tool0;

         setdo do_To_UR_1,1; !inicio ciclo UR3e

         WAITDI di_From_UR_1,1; !espera a que UR3e haya acabado
         setdo do_To_UR_1,0;
         movej pos_segura, v50, fine, doble_ploter\WObj:=wobj0;
         movej pos_ploter1, v50, fine, doble_ploter\WObj:=wobj0;
         movej pos_ploter2, v50, fine, doble_ploter\WObj:=wobj0;
         movej pos_ploter3, v50, fine, doble_ploter\WObj:=wobj0;
         movej pos_ploter4, v50, fine, doble_ploter\WObj:=wobj0;
         movej pos_ploter1, v50, fine, doble_ploter\WObj:=wobj0;

         movej pos_segura, v50, fine, doble_ploter\WObj:=wobj0;

         movej pos_ori_ventosa, v50, fine, doble_ventosa\WObj:=wobj0;
         movej pos_pre_pick, v50, fine, doble_ventosa\WObj:=wobj0;
         moveL pos_pick, v50, fine, doble_ventosa\WObj:=wobj0;
         setdo GRIPPER_CLOSE, 1;
         waittime 1;
         moveL pos_pre_pick, v50, fine, doble_ventosa\WObj:=wobj0;
         movej pos_pick_alto, v20, fine, doble_ventosa\WObj:=wobj0;
         movej pos_cinta, v50, fine, doble_ventosa\WObj:=wobj0;
         movej pos_pre_place, v50, fine, doble_ventosa\WObj:=wobj0;
         movej pos_place, v50, fine, doble_ventosa\WObj:=wobj0;
```

```
        setdo GRIPPER_CLOSE, 0;
        waittime 1;
        movej pos_pre_place, v50, fine, doble_ventosa\WObj:=wobj0;

        movej pos_cinta, v50, fine, doble_ventosa\WObj:=wobj0;
        movej pos_segura, v50, fine, doble_ploter\WObj:=wobj0;

        SetDO conveyor_BWD, 1; ! Iniciamos movimiento cinta
        WaitDI di_From_UR_2,1; ! Movemos cinta hasta que avise el UR3
        SetDO conveyor_BWD, 0;

      ENDWHILE
    ENDPROC
ENDMODULE
```

UR3

```
 Programa
    AntesDeIniciar
      Script: rpy2rv.script
      Script: rv2rpy.script
      rv:=[0,0,0]
      rv2:=[0,0,0]
      rpy:=[0,0,0]
    Programa de robot
      Ajustar DO[2]=Apagar
      Ajustar DO[1]=Apagar
      angulo:=0
      color:=0
      coord_x:=0
      coord_y:=0
      Esperar DI[2]=HI
      Ajustar DO[1]=Encender
      Esperar DI[6]=HI
      Ajustar DO[1]=Apagar
      Ajustar DO[2]=Encender
      Esperar: 1.0
      Ajustar DO[2]=Apagar
      open:=socket_open("158.42.206.7",30000)
      Invocar lectura_socket
      If color == 0
        coord_x:=0
        coord_y:=0

      ElseIf color == 1
        coord_x:=0.1
        coord_y:=0
      ElseIf color == 2
        coord_x:=0
        coord_y:=0.1
      place_variable:=p[-0.400+coord_x,-0.120+coord_y,-
0.160,0.559,0.132,-3.175]
      rv:=[place_variable[3],place_variable[4],place_variable[5]]
      rpy:=rv2rpy(rv[0],rv[1],rv[2])
      rv2:=rpy2rv(rpy[0],rpy[1],rpy[2]-d2r(angulo))
 place_variable:=p[place_variable[0],place_variable[1],place_variable[2],
rv2[0],rv2[1],rv2[2]]
      MoverJ
        pos_espera
      Ajustar DO[5]=Apagar
      MoverJ
        pos_prepick
```

```
      MoverL
        pos_pick
      Ajustar DO[5]=Encender
      Esperar: 1.0
      MoverL
        pos_prepick
      MoverJ
        pos_preplace
      MoverL
        place_variable
      Ajustar DO[5]=Apagar
      Esperar: 1.0
      MoverL
        pos_preplace
    lectura_socket
      socket_send_string("Ready")
      receiveFromServ:=socket_read_ascii_float(2)
      Bucle receiveFromServ[0]≠2
        Esperar: 0.1
        receiveFromServ:=socket_read_ascii_float(2)
      color:=receiveFromServ[1]
      angulo:=receiveFromServ[2]
```

UR3e

```
Program
   Robot Program
     Set DO[2]=Off
     Wait DI[4]=HI
     MoveJ
       pos_home
     MoveJ
       pos_intermedia
     MoveJ
       pos_prepick
     MoveL
       pos_pick
     Set DO[5]=On
     Wait: 1.0
     MoveL
       pos_prepick
     MoveL
       pos_intermedia
     MoveJ
       pos_home
     MoveJ
       pos_preplace
     MoveL
       pos_place
     Set DO[5]=Off
     Wait: 1.0
     MoveL
       pos_preplace
     MoveJ
       pos_espera
     Wait: 20
     Set DO[2]=Off
     Wait DI[4]=HI
```