Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

ÁREA DE FABRICACIÓN

DESARROLLO DE PROGRAMAS PARA UN ROBOT DESTINADO A LA PALETIZACIÓN DE CAJAS MEDIANTE VISIÓN ARTIFICIAL

AUTOR: D. MANUEL GONZÁLEZ RÍO

TUTORES: D. GONZALO VALIÑO RIESTRA

D. TOMÁS CASTRO RIERA

FECHA: JULIO 2021

DOCUMENTO 1:

MEMORIA

Manuel González Río

INDICE

INDICE DE FIGURAS

1. Introducción y objetivos.

1.1 ANTECEDENTES

En los últimos años la empresa Normagrup ha apostado fuertemente por la automatización de todos los procesos que se realizan en las diferentes naves situadas en el polígono industrial de Llanera. En el marco de esta propuesta, en el año 2020, se comenzó con la remodelación de la zona de almacenaje situada en la nave principal de la empresa, en la cual se sitúan las cadenas de montaje.

En este almacén se depositan diferentes tipos de productos, desde la materia prima necesaria para el funcionamiento de las cadenas de montaje, que se deposita en las partes 1,2 y 3 de los almacenes, a los productos ya terminados que se depositan en la parte 4 de estos almacenes.

Estos productos finales, se almacenan en cajas, y estas cajas a su vez se almacenan en pallets. El proceso de paletización se realiza actualmente de forma manual con el consiguiente gasto en empleados y espacio en la nave. Por ello, en el marco de esta apuesta por la automatización se ha decido realizar un proyecto para que este proceso de paletización se realice también de forma automática.

Esta automatización permitirá conseguir un ahorro en costes al poder reducir el personal dedicado a este proceso, en tiempos, ya que un robot no necesita tiempo de reposo entre operaciones y en espacio al solo necesitar una célula para situar el robot en vez de una zona completa para el paletizado como se tiene actualmente.

Además, un factor muy importante a tener en cuenta a favor de la automatización de este proceso es la flexibilidad que ofrece para la empresa pudiendo cambiar rápidamente la configuración de cada pallet dependiendo de la caja que se esté almacenando. Esto no es posible en la actualidad, paletizándose cada caja durante un día y pasando al siguiente tipo al día siguiente.

Por lo tanto, se ha decidido realizar este proceso con un robot, acompañado de un sistema de visión artificial que, cada vez que llegue una caja a la zona de paletizado, detecte su tipo y dimensiones para seleccionar el tipo de paletizado que le corresponda.

Con esto en mente, la empresa Normagrup ha decidido seguir adelante con el proyecto y ofertarlo como Trabajo Fin de Máster a los alumnos de la Escuela Politécnica de Ingeniería de Gijón y, tras un proceso de selección, me fue asignado y lo realicé como detallaré a continuación, en las siguientes secciones.

1.2 **OBJETIVOS**

Para la realización del proyecto se decidieron una serie de objetivos a cumplir que se van a detallar a continuación:

- Definir las tareas a realizar por el robot y la tecnología a emplear: En este primer paso, se han de delimitar cuales van a ser las tareas que va a realizar el robot para poder seleccionar de entre todos los tipos de robots que hay en el mercado, cuál se adapta más a las necesidades del proyecto.

- Determinar el sistema de visión adecuado para la ubicación de cajas en 3D. En este paso, se determinará cómo se va a realizar la identificación de las cajas y sus dimensiones, se realizará un estudio sobre las diferentes opciones disponibles y se seleccionará la más adecuada para el proyecto.

- Determinar el robot industrial a utilizar, apto para las tareas propuestas. Una vez recopilada la información sobre los robots disponibles en el mercado se seleccionará el más adecuado.

- Realizar la programación del sistema de visión que permita la identificación de las cajas. El sistema de visión seleccionado requerirá una programación para su funcionamiento dividida en varios módulos.

- Desarrollar los programas del robot, en consonancia con el sistema de visión, para la confección de pedidos del cliente

- Realizar pruebas de funcionamiento. Se realizarán ensayos con los programas ya implantados, para comprobar el correcto funcionamiento de estos. Los ensayos se dividirán en dos partes, realizándose la primera en una zona delimitada de la nave sólo para pruebas y, la segunda parte, ya en la zona de almacenaje donde verdaderamente se va a situar el robot.

2. Estado de la técnica

En este apartado del documento se va a realizar un estudio sobre el estado al inicio del proyecto de cada uno de los componentes que serán necesarios para la realización del mismo.

2.1 **ROBOT INDUSTRIAL**

El principal componente de este proyecto será el robot industrial, el cuál será el encargado de mover las cajas entre los diferentes puntos de la célula.

Los robots industriales se definen como “los robots que se crean y se utilizan en el sector industrial con el objetivo de automatizar un proceso, ya sea dentro de un ambiente colaborativo con humanos o dentro de un vallado de seguridad” (Robots, 2021).

El primer robot industrial fue creado en el año 1937 por el estudiante británico Bill Taylor, quien utilizó los componentes de los juguetes Meccano para crear un robot con forma de grúa al que llamó Gargantua. Este robot (que se puede ver en la Figura 2.1) se usó para mover y colocar objetos (pick and place) (EDS, 2021)

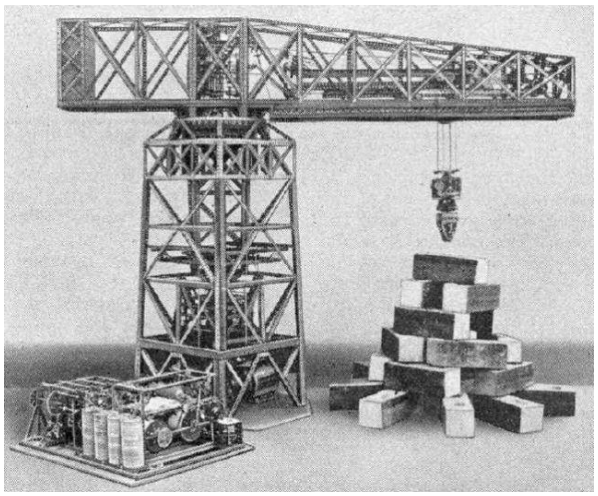


Figura 2.1. Robot Gargantua

En el año 1938 la compañía Devilviss construye el primer brazo robótico que se utilizaría para pintar con pintura en espray. Con estos avances se llega a la feria de Nueva York de 1939, la cual se considera como el inicio de la industria del robot.

La industria sufrió un paro debido a la Segunda Guerra Mundial hasta la década de los 60, en la que se apostó por los robots de gran tamaño. En 1961 la empresa Unimation (Figura 2.2) propiedad de Joseph Engelberger y George Davol, dirigió el desarrollo del primer robot de transferencia programable (EDS, 2021).

Figura 2.2. Logo de la empresa Unimation

La licencia de este robot fue adquirida por la empresa General Motors para introducir un robot en su cadena de montaje que levantara piezas grandes de metal en caliente, uno de los procesos más peligrosos en ese momento para el ser humano.

En 1968 la empresa Kawasaki adquiere los derechos de las licencias de Unimation para su uso en Japón. Además, en 1969 Víctor Scheinman desarrolla el *Standford Arm*, el primer brazo robótico ligero.

En Europa la empresa Nokia lideró el proceso de automatización robótica con el desarrollo de robots de soldadura capaces de soldar 110 automóviles por hora.

En 1975, nació en Europa el robot ASEA IRB, este robot era totalmente eléctrico y utilizaba la tecnología Intel. Por último, en el año 1978, Unimotion y Vicarm, la empresa propiedad de Víctor Scheirman (creador del Standford Arm), lanzaron al mercado el robot PUMA, utilizado en las líneas de ensamblaje de General Motors.

Los años 80 supusieron una explosión en el desarrollo de la robótica y son considerados como el inicio de la Era Robótica, ya que su fabricación y venta aumentaron en un 80%.

Los años 90 y 2000 han llevado a que la robótica de la industria siga en la misma dirección y creciendo a pasos agigantados. La globalización ha contribuido a que se demanden y produzcan muchos más bienes de consumo que hace unas décadas.

El mercado de los robots industriales ha evolucionado hasta situarse a fecha de 2017 como se ve en la Figura 2.3 (A.Diaz, 2020).

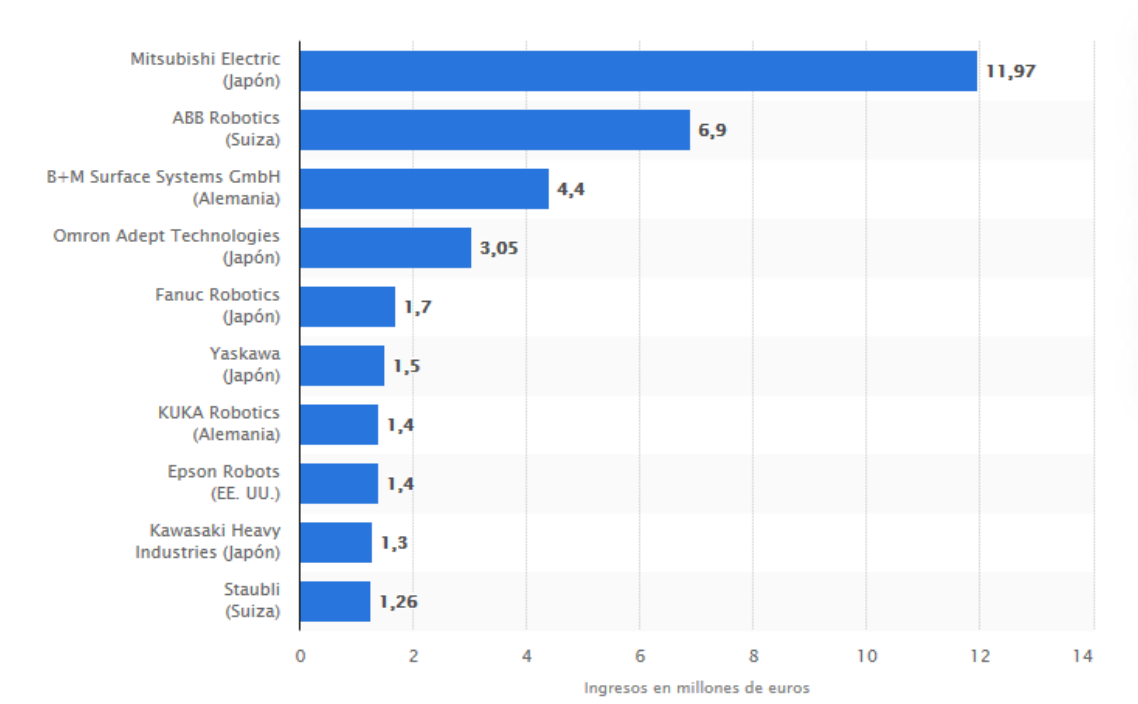


Figura 2.3 Estado del mercado de robots industriales

En esta memoria se va a centrar la mirada en la marca ABB, segunda líder del mercado de la robótica industrial, ya que es la marca con la que trabaja la empresa y a la cual se va a adquirir en robot.

ABB (Figura 2.4) es el acrónimo de Asea Brown Boveri. Esta empresa es el resultado de la fusión de la empresa sueca ASEA y del grupo suizo 'Brown, Boveri & Cie (BBC) en 1967 (Wikipedia, 2021).

Figura 2.4 Logotipo de ABB Robotics

En palabras de la propia empresa ABB mantiene más de 160.000 robots operativos en todo el mundo siendo esta la base instalada de mayor tamaño en el mundo. Entre las grandes empresas que operan con ABB se pueden citar Nestlé y Cadbury en la rama alimenticia, IKEA en el mercado del mobiliario, Apple y Dell en el mercado informático y farmacéuticas muy conocidas estos días como AstraZeneca o Johnson & Johnson. (ABB, 2021)

Dentro del catálogo de ABB los robots se pueden dividir en 5 tipos:

- Robots Articulados
- Robots colaborativos
- Robots de pintura
- Robots delta
- Robots SCARA

Dentro de estos tipos, el necesario para el proyecto será un robot articulado, dentro de los robots articulados la elección se hace en base a la carga que llevará y al alcance necesario. La elección del robot se explicará en el apartado 3 de esta memoria.

2.2 SISTEMAS DE VISIÓN ARTIFICAL

Para enviar la información al robot será necesario poder detectarla, para ello se va a utilizar un sistema de visión artificial. La visión artificial es “una disciplina científica que incluye métodos para adquirir, procesar y analizar imágenes del mundo real con el fin de producir información que pueda ser tratada por una máquina” (Contaval, 2016).

Los sistemas de visión artificial surgen en la década de los 60 con la idea de conectar las cámaras de video a los ordenadores. Esto incluía no solo capturar las imágenes, sino, interpretarlas. El gran pionero de esta visión fue Larry Roberts (Figura 2.5) creador de ARPAnet (red de comunicación entre los diferentes entes del ministerio de defensa de los EE. UU.). Larry creó en 1961 creó el programa “mundo de micro bloques” en el que programa veía una estructura de bloques la analizaba y la reproducía demostrando que la había procesado. (Shun, s.f.)

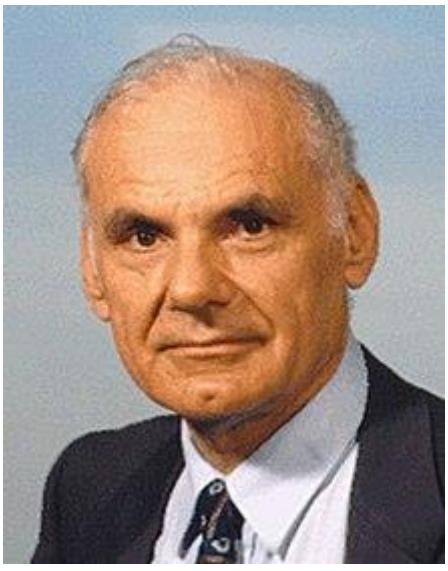


Figura 2.5 Larry Roberts

En la década de los 80, con el desarrollo de los procesadores, se consiguió captar imágenes y procesarlas cuando éstas eran tomadas de forma remota. Además, se consiguió una mayor precisión gracias al avance de las cámaras, lo que permitió detectar contornos y formas. (INFAIMON, 2020).

Actualmente, las principales aplicaciones de la visión artificial son las siguientes (INFAIMON, 2020):

- Realizar el seguimiento de un objeto mediante la secuencia de imágenes.
- Localizar y reconocer ciertos objetos entre varios por sus características físicas.
- Obtener modelos tridimensionales a través de los datos que genera el análisis de una imagen.
- Estimar y comparar las medidas de varios objetos entre sí.
- Crear modelos y patrones a través de archivos de imágenes digitales para realizar respuestas automatizadas.

Para este proyecto la fundamental será la segunda, ya que se buscará detectar el sistema de codificación en la imagen.

2.3 SISTEMAS DE CODIFICACIÓN

La información que será detectada por el visor debe ser adherida a las cajas mediante un sistema de codificación, en el caso de este proyecto se va a diseñar un sistema de codificación QR como se explicará más adelante en este documento. Los códigos QR pueden ser considerados como una evolución de los códigos de barras por lo que se comenzará este apartado con ellos.

Un código de barras se define en el diccionario como “Combinación de líneas y números que se imprime en las envolturas de los productos de consumo para que pueda ser leído

y descifrado por un lector óptico que transmite los datos a una máquina o una computadora; suele contener informaciones como la fecha de envasado, el número de lote, la procedencia y otros datos". (Languages, 2021)

El código de barras fue inventado el 20 de octubre de 1949 por Norman Woodland y Bernard Silver (Figura 2.6) e identificado con la patente US2612994. (Curiosfera, 2020)

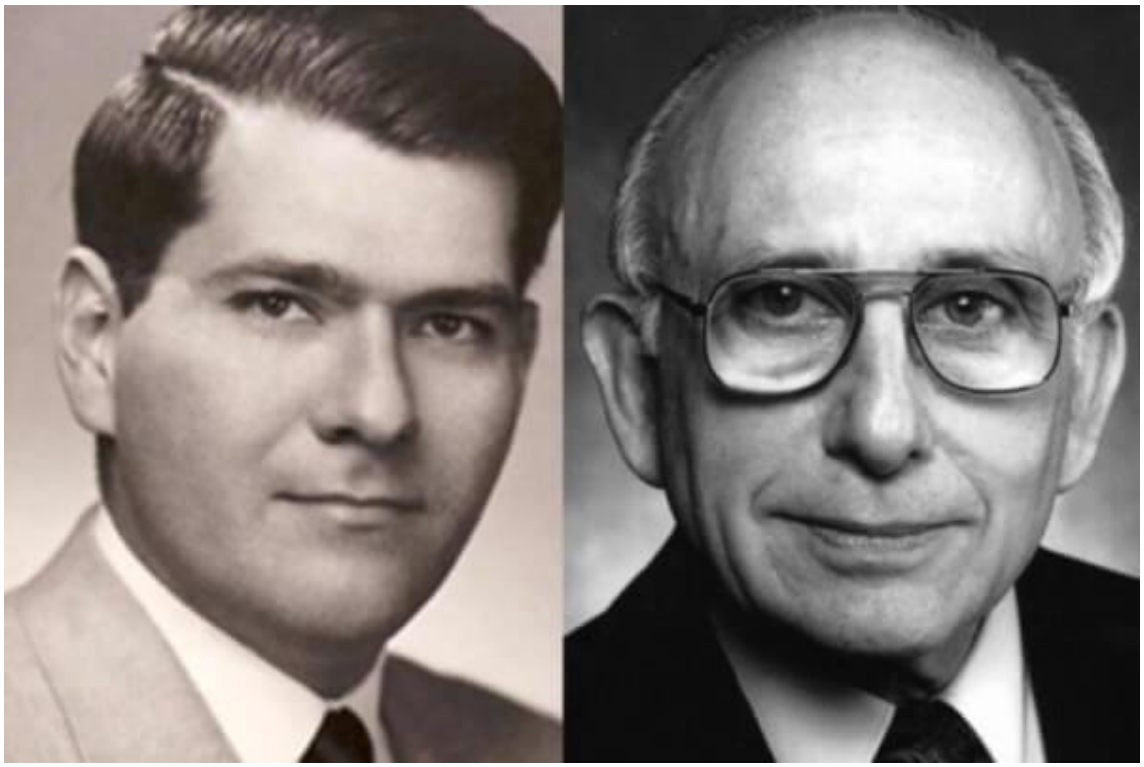


Figura 2.6 Bernard Silver y Norman Woodland

Los primeros códigos de barras tenían forma de diana- No fue hasta la década de los 60 cuando se adoptaron las líneas paralelas y hasta 1973 cuando se diseñó el primer código tal y como lo conocemos con los dígitos.

Ya en el 26 de junio de 1974 se escaneó en Troy, Ohio, el primer producto en un supermercado: un paquete de chicles.

El código QR, por su parte, fue creado en el 1994 por la empresa japonesa Denso-Wave filial de Toyota. (Unidad de Informática, s.f.). Este se creó con el objetivo de condesar una mayor cantidad de información que la que se podía almacenar en los códigos de barras.

QR son las siglas de la frase inglesa “Quick Response” (respuesta rápida en español). Los códigos QR se caracterizan principalmente por la existencia de los tres cuadrados en las esquinas del QR que permiten conocer la posición y la orientación del código (Figura 2.7).

Figura 2.7 Cuadrados del QR

La gran explosión de los códigos QR se dio en el año 2002 cuando se comenzaron a incluir los lectores de QR en los smartphones, esto hizo que se convirtieran en algo común para la gente de a pie, hasta la actualidad en la que ha sustituido al papel en muchos establecimientos como restaurantes o cafeterías en los cuales se puede consultar la carta gracias a un QR. (Unidad de Informática, s.f.)

3. Desarrollo de programas para el robot.

En este apartado del documento se van a especificar cuáles son los equipos necesarios para el funcionamiento del robot, se detallará cuál ha sido el elegido en cada caso y se citarán las características por las cuales se ha realizado esta elección. Por último, se detallarán cada uno de los módulos de programación implantados en el robot y cuáles son las características de su funcionamiento.

3.1 SISTEMAS INTEGRANTES DE LA CELULA ROBOTIZADA

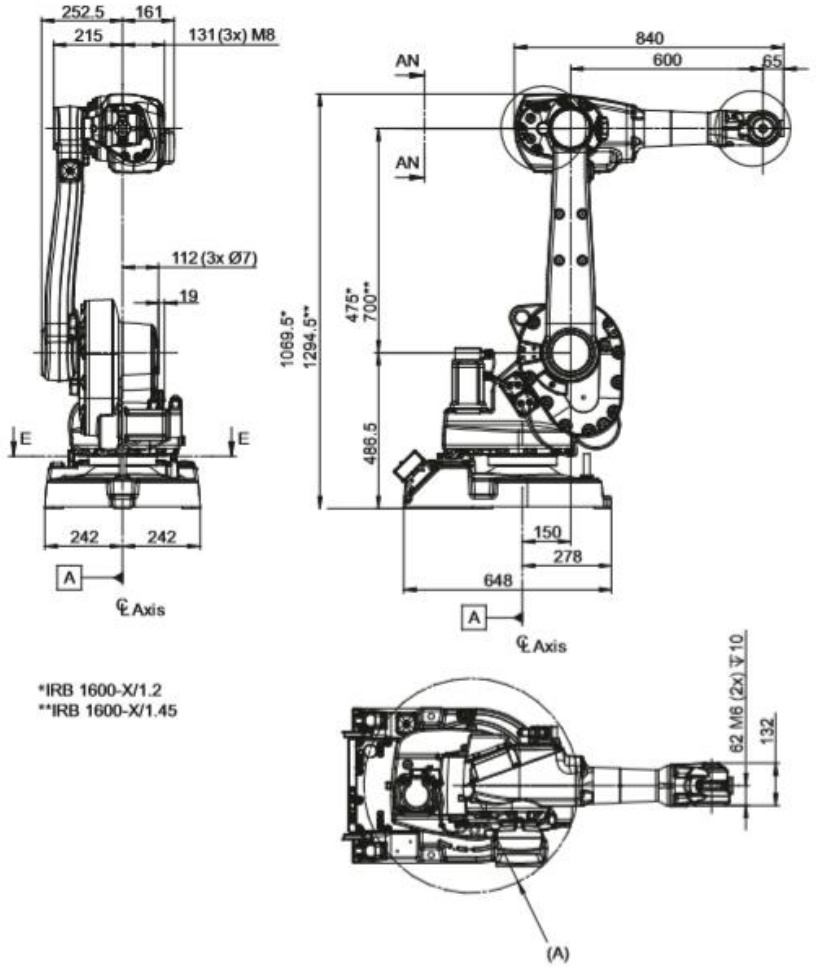
Se comenzará por los sistemas que integran el robot para su correcto funcionamiento. Estos constan del propio robot, la cámara de visión artificial, el sensor de posición, la pinza de recogida, la estación de giro, el control del robot y el puesto de control total.

El primer integrante a tratar será el propio robot. El robot ABB elegido ha sido el IRB 1600 que se puede ver en la Figura 3.1.

Figura 3.1 Robot ABB, modelo IRB 1600

Las dimensiones principales de este robot y el rango de trabajo se pueden ver en las Figura 3.2 a y b.

Dimensions IRB 1600-X/1.2 (1.45)



xx1000000972

A	R= 335 mm minimum turning radius

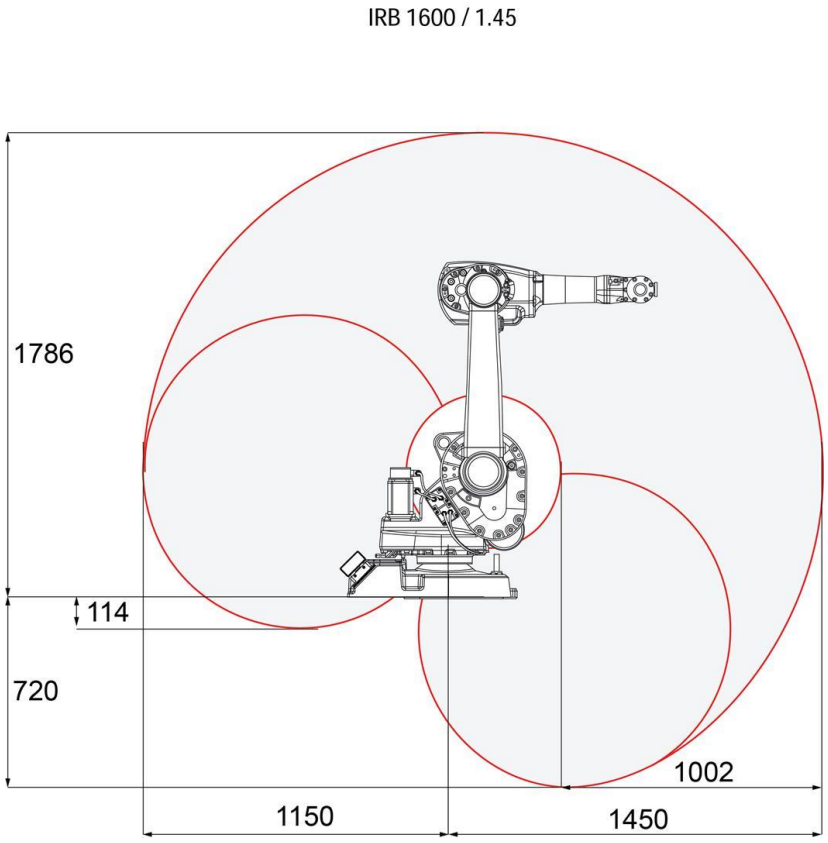
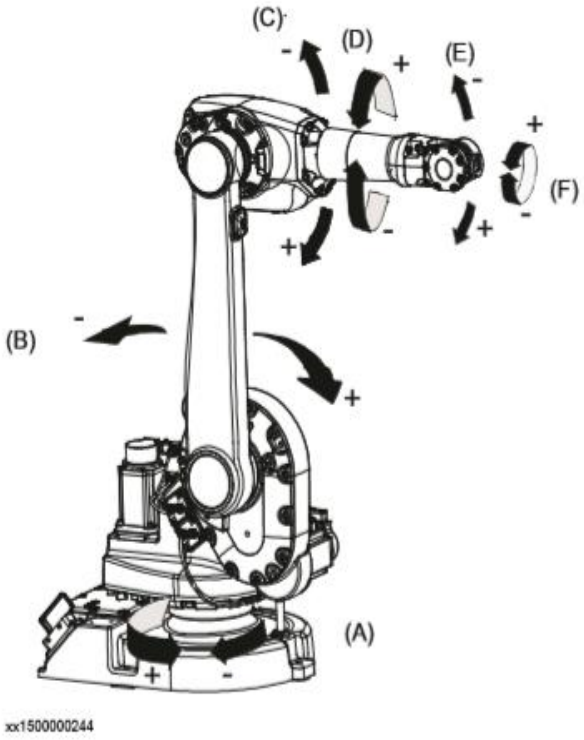


Figura 3.2 Dimensiones (a) y rango de trabajo del robot (b).

Una de las características fundamentales para la elección de este robot son sus seis ejes de giro que permiten alcanzar posiciones mucho más complicadas que serán necesarias para poder realizar las tareas de la célula.

Como se puede ver en la Figura 3.3, tres de los ejes (A, B y C) son ejes de movimiento, mientras que los ejes D, E y F son ejes de rotación.

Ejes del manipulador



xx1500000244

Pos	Descripción	Pos	Descripción
A	Eje 1	B	Eje 2
C	Eje 3	D	Eje 4
E	Eje 5	F	Eje 6

Figura 3.3 Ejes de movimiento del IRB 1600

El resto de las características importantes para la elección del robot son (ABB, 2021):

- **Alta velocidad de producción:** El IRB 1600 ha demostrado en que sus ciclos de trabajo son un 50% más rápidos que los de sus competidores. Esta característica será fundamental para la célula que deberá realizar una gran cantidad de ciclos cada día y, por lo tanto, el tiempo ahorrado gracias a la velocidad de este robot será fundamental.

- **Calidad constante:** Este robot tiene otra característica fundamental con respecto a sus competidores que es la capacidad de no variar la calidad de sus movimientos independientemente de la velocidad de desplazamiento, esta característica también será muy importante para la célula ya que permitirá realizar los movimientos mucho más rápido y, por tanto, aumentar la productividad de la célula.
- **Alta fiabilidad :** La fiabilidad que ofrece el robot IRB 1600 también es una característica importante para el proyecto permitiendo que funcione durante más ciclos sin producirse fallos y por lo tanto al igual que en los casos anteriores, aumentando su productividad.
- **Fácil de integrar:** El IRB 1600 tiene un fácil montaje, lo que de cara al proyecto permitirá desplazarlo con facilidad a diferentes posiciones de la nave en las cuales se podrán realizar diferentes pruebas de funcionamiento antes de su instalación.
- **Sostenible y ecológico:** Por último, el robot IRB 1600 tiene un bajo consumo a altas velocidades, lo que permitirá ahorrar en costes a la empresa y además apostar por la sostenibilidad del medio ambiente.

El robot, tiene incorporado un controlador IRC5 y un flex pendant para el control manual del movimiento de los ejes. El flex pendant (Figura 3.4) se define como “una unidad de operador de mano que se usa para realizar muchas de las tareas implicadas en el manejo de un sistema de robot: ejecutar programas, mover el manipulador, modificar programas del robot, etc.” (Robotics, 2019)

Además, el flex pendant forma parte del controlador IRC5 del robot y está unido a este mediante un cable y un conector. El flex pendant se puede ver en la Figura 3.4.

Figura 3.4 Flex pendant

Este flex pendant incorpora toda la programación del robot, diseñada en el entorno de programación RobotStudio, el cual tiene la gran ventaja de poder conectarse en línea al flex pendant y, por lo tanto, editar la programación del robot desde una estación de control sin tener que desplazarse hasta la zona del robot.

Otro elemento a destacar del robot es la pinza, la cual estará compuesta por una serie de ventosas conectadas a un conducto de vacío que proporcionará la fuerza de succión necesaria para levantar las cajas sin peligro de caídas. Se analizaron varios diseños hasta encontrar uno que satisficiera las necesidades del proyecto (Figura 3.5). Esta pinza fue se adquirió a la empresa OnRobot.

Figura 3.5 Pinza del robot

Con esta pinza se terminan los elementos propios del robot y se comenzará ahora con la explicación de las piezas que no están integradas directamente en el propio robot.

El primer integrante exterior al robot a comentar será el sistema de visión artificial, este sistema estará compuesto de una videocámara situada en un punto fijo que estará enfocando en todo momento hacia la zona donde pueden aparecer cajas a trasportar. La cámara estará conectada al PC del puesto de control total de la estación, este puesto dispondrá de un ordenador y en el cual, la programación que se explicará más adelante en el documento obtendrá las dimensiones de la caja y se las enviará al robot mediante comunicación socket (como se explicará más adelante en este documento).

La cámara seleccionada para hacer la visión artificial es del modelo Teledyne Dalsa Genie HM y se puede ver en la Figura 3.6.

Figura 3.6 Cámara Final

Las principales características de la cámara son las siguientes (Risoul, 2016):

- Sensor CCD de Alta resolución 1/3"
- Resolución de 1024 x 768 píxels
- Tamaño píxel 4,65um
- Alta sensibilidad : 1 Lux con óptica f1.4

- Rango dinámico mejorado y relación señal/ ruido superior a 50dB
- Funciona en formato progresivo de 20 img/seg.
- Salida Digital 8 o 10bit Giga Ethernet (GigE) 10/100/1000Mbps
- Shutter Electrónico de (1:60 a 1 :64.000) y posibilidad de Reset Asíncrono.
- Dispone de señal de píxel clock para aplicaciones donde se necesite precisión subpíxel. -Salida Digital GigE
- I/O: 2 Inputs opto aislados, 2 Outputs
- Montura C
- Dimensiones: 29 x 44 x 67mm
- Peso: 125 gr

Esta cámara ofrece dos grandes ventajas con respecto al resto de las opciones estudiadas para el proyecto que serán muy útiles para este.

La primera de ellas es la captura de imágenes a gran velocidad, esto permitirá detectar los códigos QR de una manera mucho más rápida que en el caso de otras de las cámaras estudiadas como posibilidad. Esto, permitirá acelerar el proceso de la célula.

La segunda gran ventaja es la captura de 20 imágenes por segundo, como se explicará más adelante en este documento, el proceso de detección de QR contará con dos sistemas de mejora de detección (alisado y cuenta de detección), estos dos sistemas mejoran cuanto mayor sea la capacidad de detección de la cámara, ya que cuantas más imágenes se detecten por segundo, más veces se podrá realizar la comprobación.

Además de las ventajas citadas, la empresa cuenta con una amplia experiencia con la marca Teledyne y, por lo tanto, esto fue un gran punto a favor de la elección de esta cámara en vez del resto de opciones estudiadas.

Otro sistema que complementa el sistema de visión artificial es el sensor de posición, este sensor, estará situado apuntando a la zona a la que llegan las cajas para su recogida con el robot y tendrá la función de realizar una comprobación de seguridad antes de lanzar el programa de recogida.

Este sensor comprobará que efectivamente hay una caja que recoger en la zona indicada. La ausencia de caja podría deberse a que, en el tiempo que transcurre desde que la información es recogida en la estación de control y se envía al robot, ésta podría haberse movido accidentalmente y, por lo tanto, no estaría en la zona en la que se debe recoger. Si esto pasara, el programa volvería al estado de búsqueda de cajas hasta detectar una nueva.

El sensor elegido es del tipo fotocélula de la marca SICK modelo P4211 y se puede ver en la Figura 3.7.

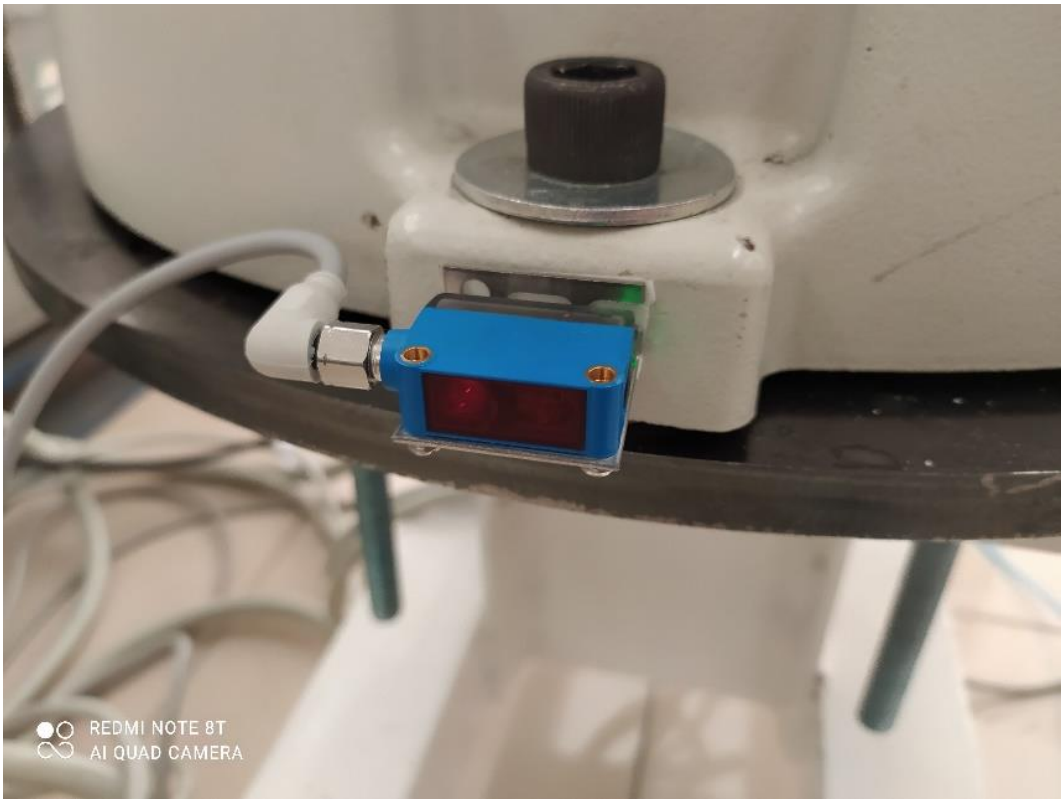


Figura 3.7 Sensor de posición

Las características principales del sensor según la página web del fabricante son (SICK, 2020):

Principio funcional	Fotocélula de detección sobre objeto
Detalle del principio de funcionamiento	Supresión de fondo
Alcance de detección máx.	5 mm … 250 mm
Distancia de conmutación	35 mm … 140 mm
Filtros de polarización	No
Fuente de luz	LED de localización
Tipo de luz	Luz roja visible
Tamaño del spot (distancia)	Ø 6 mm (100 mm)
Datos característicos del LED Longitud de onda	 650 nm
Ajuste	Ajustador mecánico, 5 revoluciones

Figura 3.8 Características del sensor.

Una de las piezas más importantes para el funcionamiento del proyecto es la estación de giro. Hay que tener en cuenta que las cajas recogidas por el robot no siempre van a estar orientadas de forma que la altura sea la dimensión más pequeña de entre altura, ancho y largo, a pesar de que esta disposición es la más indicada para que la paletización sea estable.

Por lo tanto, se instalará una estación intermedia en la que el robot depositará la caja y la recogerá de nuevo con la orientación más estable .A continuación, seguirá con su trayectoria para depositarla en el pallet.

Para esta estación se probaron diferentes diseños siendo uno de ellos el mostrado en la Figura 3.9.

Figura 3.9 Estación de giro

Finalmente, este diseño se descartó porque no ofrecía seguridad al 100% de que la caja se quedara en la posición deseada, debido a la poca altura de las barreras y se acabó seleccionando la estación que se ve en la Figura 3.10.

Figura 3.10 Estación de giro final.

Por último, se debe mencionar la estación de control en la cual se situará el PC en el que se realizarán los procesos de análisis de la información recogida por el sistema de visión y envía esta información al robot. Además también tendrá una pantalla con la cual se podrá controlar el proceso de forma fácil e intuitiva para que los operarios no necesiten un amplio conocimiento del funcionamiento del robot (Esta pantalla se explicará más adelante en este documento).

3.2 ESPECIFICACIONES INICIALES

Una vez analizados los elementos que van a formar parte de la célula es el momento de analizar cuáles van a ser los objetivos de ésta. Como se ha especificado anteriormente, el objetivo principal es conseguir automatizar el proceso de paletización de la empresa.

Para ello, se debe delimitar cuales van a ser las tareas que va a realizar cada una de las partes involucradas en la célula. La primera de ellas es el sistema de visión artificial cuya tarea será realizar un seguimiento de la zona de recogida en tiempo real detectando cada caja que llegue a la zona y hallando sus dimensiones, también en tiempo real, para estar preparado para el momento en el que se le solicite.

Para la detección de las cajas se diseñará un sistema de códigos QR. Dentro de cada uno de éstos estará contenida la información relativa a la dimensión del propio código, la posición de éste y de las dimensiones de la propia caja. El sistema de códigos QR se explicará en detalle en el apartado 3.3 de este documento.

El segundo elemento principal es el robot. La tarea asignada a éste es sencilla: recoger las cajas en la zona de llegada y transportarlas hasta colocarlas en el palé con la orientación óptima, garantizando en todo momento la seguridad de la caja y de su contenido.

Todos estos sistemas deben estar conectados entre sí en todo momento y sincronizados para una correcta ejecución de las tareas encomendadas.

3.3 ESTRUCTURA DE DESARROLLO DE PROGRAMAS

En este apartado del documento se va a dar una breve explicación de cómo se diseñó el proyecto para que cumpla los requisitos de funcionamiento establecidos en el apartado anterior.

Para poder cumplir estos requisitos se diseñaron una gran cantidad de módulos de programación que interrelacionándose entre sí, permiten realizar los procesos correspondientes.

Estos módulos se pueden agrupar principalmente en 3 partes:

- La primera de ellas será la que controle el sistema de visión artificial
- La segunda, controlará el movimiento del robot
- La tercera, controlará el intercambio de información entre los anteriores.

Se comenzará explicando el funcionamiento del visor. Éste está funcionando en todo momento en un estado de espera hasta que llegue una caja a su campo de visión.

Cada caja tendrá un código QR impreso en su superficie. Este código tendrá unos valores en su interior que, al ser decodificados, proporcionará la información necesaria al visor para conocer la posición y las dimensiones de la caja (Figura 3.11).

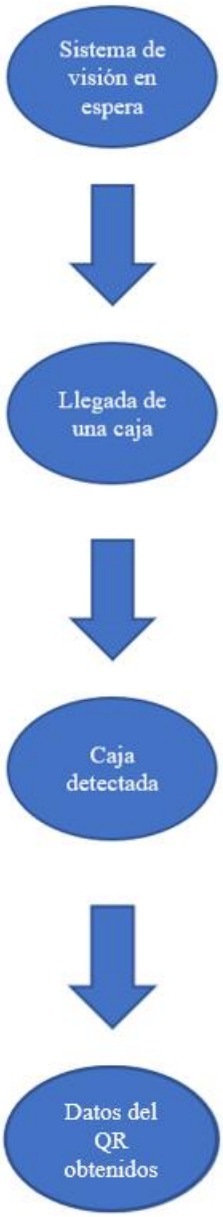


Figura 3.11 Funcionamiento del visor

Este proceso de visión está funcionando ininterrumpidamente, pero no está enviando los datos en todo momento. Paralelamente al funcionamiento del visor, un servidor Socket está en funcionamiento a la espera de que un cliente (que en este caso será el robot) se conecte. Una vez que el robot se conecta, el servidor recibe la información del visor, lanza un módulo que define la posición de la caja en el pallet y envía toda esa información al robot para que se ponga a funcionar (Figura 3.12).

Figura 3.12 Funcionamiento del servidor

La otra gran parte principal del funcionamiento de la célula es la del funcionamiento del robot. Como se especificó anteriormente el robot se conecta al sistema de visión mediante conexión Socket. En este caso el robot actúa como un cliente que se conecta al servidor creado por el sistema de visión.

Una vez que se inicia el programa, éste se conecta al sistema de visión y espera a recibir los datos. Una vez recibidos estos datos, los descomprime ya que no pueden ser enviados directamente. Una vez descomprimidos los datos, se sitúa encima de la caja y, con el

sensor de posición anteriormente mencionado, comprueba si la caja sigue en su posición. Si está en ella, el robot baja a coger la caja.

Una vez que el robot tiene la caja, se abren dos vías: si la caja está en buena posición, la lleva directamente al pallet; si no lo estuviera, lo traslada a la zona de giro para llevarla a una buena posición y, finalmente, ubicarla en el pallet (Figura 3.13).

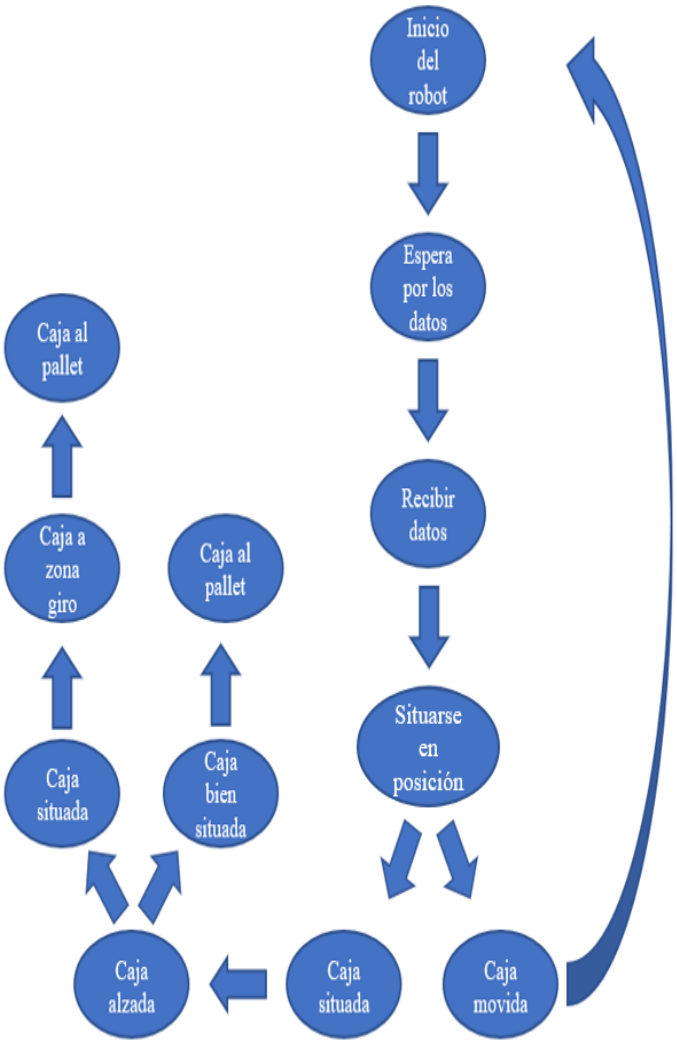


Figura 3.13 Funcionamiento del Robot

Estos son los principales procesos que realiza la célula para funcionar. A ellos hay que añadir algún pequeño proceso que será explicado en los próximos apartados de la memoria.

3.4 DESARROLLO DEL MODULO "WEB CAM CALIBRATION"

Se procederá ahora a explicar uno a uno los módulos más importantes del proyecto. El primero del que se va a hablar es el llamado "Web Cam Calibration". Este módulo, como dice su nombre traducido, tiene como objetivo calibrar la cámara que se usará para el sistema de visión artificial con el objetivo de que el resultado obtenido sea lo más preciso posible.

Para poder realizar esta calibración se necesita imprimir un patrón similar a un tablero de ajedrez para que la cámara detecte los cuadros. Este patrón se ve en la Figura 3.14.

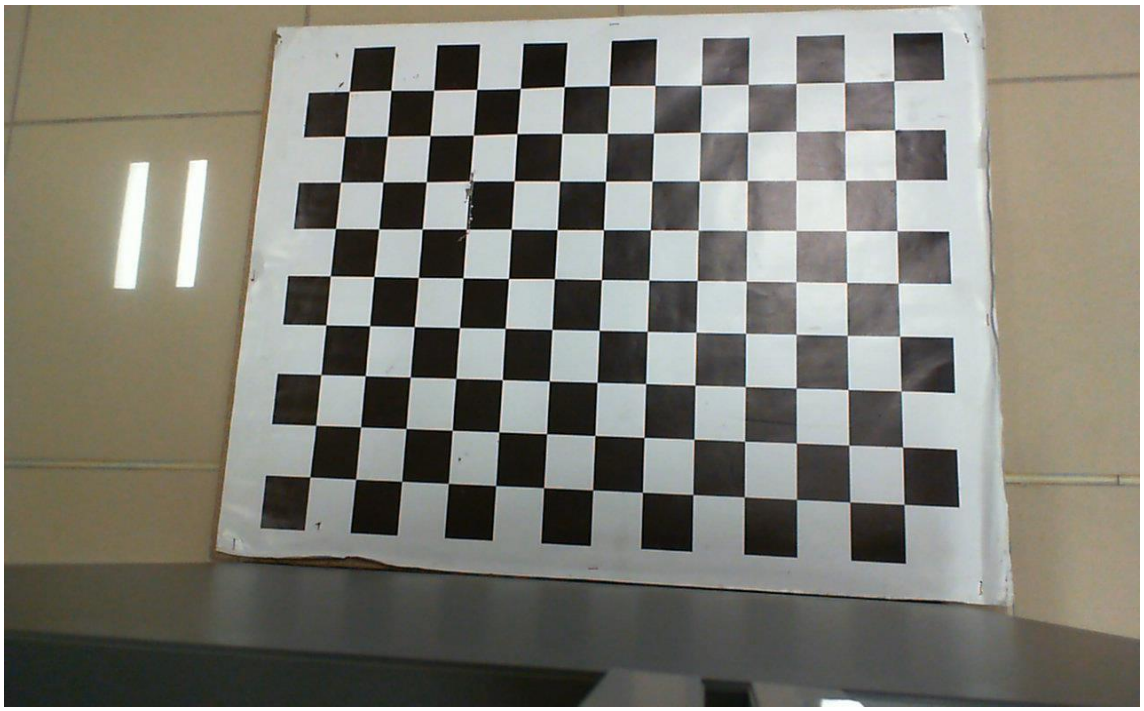


Figura 3.14 Tablero de Calibración

Los parámetros que definen este tablero se deben introducir manualmente en el programa, tal como se muestra en la Figura 3.15.

```
Plantilla_SquareSize = 53
Plantilla_NumCuadros =  (13, 9)
w0, h0 = (1280, 720)
w0, h0 = (1920, 1080)
```

Figura 3.15 Parámetros necesarios para la calibración

Una vez introducidos estos datos se deben realizar fotografías del "tablero" en diferentes posiciones y orientaciones para que el programa las detecte.

Con todo esto preparado, se puede proceder a la calibración de la cámara. Cuando el programa es lanzado, éste realiza una serie de cálculos para obtener las matrices correspondientes a la cámara y con ella crea un archivo ". Json" que servirá como input para el resto de los procesos. La salida del programa tiene la forma que se ve en la Figura 3.16, siendo *Camera Matrix* la matriz general de la cámara, *Dist Coeficients* los coeficientes de distorsión, *rvecs* la matriz de rotación y *tvecs* la matriz de traslación.

```
{
    "CameraMatrix":[
        [
            1392.3371242579865,
            0.0,
            644.3724799313798
        ],
        [
            0.0,
            1391.3437490279052,
            332.6079310970339
        ],
        [
            0.0,
            0.0,
            1.0
        ]
    ],
    "DistCoeficients":[
        [
            -0.026171053668980256,
            1.004091821053159,
            -0.005262419034366928,
            0.003660472388639756,
            -3.2090448977217245
        ]
    ],
    "rvecs":[
        [
            -0.24085464895721279
        ],
        [
            -0.39126471368204124
        ],
        [
            -3.0728027445680883
        ]
    ],
    "tvecs":[
        [
            331.7424716608269
        ],
        [
            160.33496277243356
        ],
        [
            1271.928178235437
        ]
    ]
}
```


Figura 3.16 Resultados de la calibración de la cámara

A modo de resumen se puede utilizar el flujograma de la Figura 3.17.

Figura 3.17 Resumen del módulo calibración

3.5 DESARROLLO DEL SISTEMA DE CODIFICACION QR

Para poder darle a los módulos la información sobre las cajas que necesita para funcionar se diseñó un sistema de códigos QR. Estos códigos contienen la información encriptada en su interior.

Una vez la caja llega al visor, éste detecta el código y lo desencripta mediante el módulo QR que se explicará más adelante en este documento. La información dentro del QR es una cadena de caracteres de la forma ,a,b,c,d,e,f, siendo los valores de estas letras los que se indican en la Figura 3.18 (en ella el cuadrado pequeño corresponde al QR y el rectángulo más grande a la caja). Las comas se utilizarán para separar los valores en el módulo QR como se explicará más adelante.

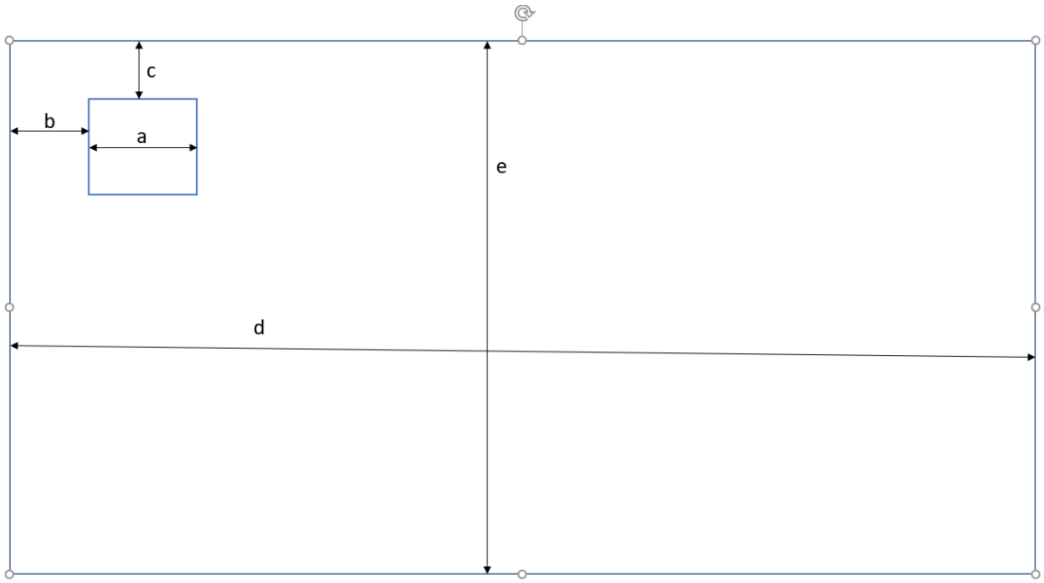


Figura 3.18 Dimensiones contenidas en el QR

A estas dimensiones hay que añadirle el valor de f que corresponde a la altura de la caja. Para cada caja se diseñarán 3 códigos distintos uno para cada cara (las caras son iguales 2 a dos).

Para ilustrar esta codificación con un ejemplo se van a proporcionar los códigos QR correspondientes a una caja con las dimensiones mostradas en la Figura 3.19.

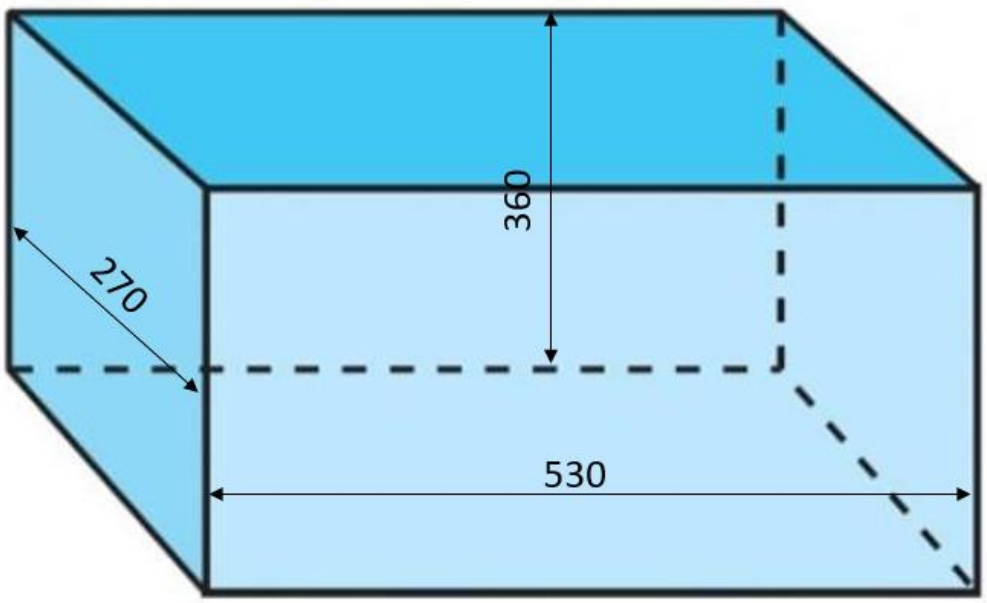


Figura 3.19 Esquema de la caja de ejemplo

Los QR estarán situados en la esquina superior izquierda de la cara en la que esté (solo en este ejemplo), por lo tanto, los valores de b y c serán de 0. Por último, el QR tendrá una dimensión de 80 mm por lo tanto la a valdría 80.

Con estos se pueden proporcionar 3 códigos QR (Figura 3.20) que corresponden a la caja de ejemplo.

Figura 3.20 Códigos QR de ejemplo

3.6 DESARROLLO DEL MODULO QR

Una vez explicado cómo se va a realizar la codificación QR, se pasará ahora a explicar cómo se diseñó el módulo llamado QR. Este módulo es el que recibe la información del QR desde el visor y la desencripta.

Una vez desencriptada mediante unos cálculos matemáticos y unas transformaciones de perspectiva permite detectar los bordes de la caja.

Una vez obtenidos estos bordes se realizarán varias transformaciones de valores y varios cálculos matemáticos para finalmente acabar obteniendo la posición del centro y prepararla para su envío al robot, ya que este punto se utilizará como el punto de recogida de la caja.

Una vez explicado el funcionamiento del módulo se pasa a una explicación más detallada de alguno de los aspectos fundamentales de este.

El módulo QR toma como entrada la imagen del visor, esta imagen se toma mediante la extensión de Python llamada OpenCV. Open CV (Open Source Computer Vision) (Figura 3.21) es la biblioteca dedicada a visión artificial más grande de internet ahora mismo. Comenzó en 2006 como un proyecto de innovación de Intel y ha ido extendiéndose hasta la actualidad, en la que se ha convertido en una biblioteca gratuita que soporta la mayoría de los lenguajes de programación más comunes como C++, Java y Python (Marin, 2020).

Figura 3.21 Logotipo de OpenCV

Con esta biblioteca se tomará la imagen de la cámara gracias a las órdenes de programación de la Figura 3.20, dándole las dimensiones adecuadas a la imagen.

```
self.cap = cv2.VideoCapture(0)
self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1920)
self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)
```

Figura 3.22 Órdenes para capturar la imagen

El siguiente paso será decodificar los códigos QR que aparezcan en cámara para esto se utilizará la biblioteca de Python llamada Pyzbar. Pyzbar es una librería de Python dedicada a la localización y decodificación tanto de códigos QR como de códigos de barras. Esta librería fue creada en 2016 por Lawrence Hudson (Hudson, 2020).

Con la orden *decode* de la biblioteca se decodifican los QR que aparezcan en la imagen (Figura 3.23 a) y se obtienen los puntos que corresponden a los vértices del QR, su orientación y el ángulo que forma con el origen de la imagen.(Figura 3.23 b).

```
#Sacar los datos de QR
decodedObjects = pyzbar.decode(img)
```

```
points=decodedObjects[i].polygon
#Orientación del QR
orientation=str(decodedObjects[i].orientation)
angulorad = math.atan2(points[1].y - points[0].y, points[1].x - points[0].x)
angulo=angulorad*180/(math.pi)
```

Figura 3.23 Órdenes decodificar los QR (a) y hallar sus características (b).

Los puntos servirán para localizar el QR en la imagen, la orientación se obtiene gracias a los tres cuadrados exteriores del QR, dependiendo de la posición de estos, se pueden obtener 4 orientaciones distintas que junto al ángulo que divide las opciones entre mayor de 45 grados y menor de este valor ofrece 8 distintas posibilidades de colocación del QR en la imagen. Cada una de estas posibilidades requiere un tratamiento distinto de la imagen, aunque en este documento solo se va a explicar uno de ellos al no existir cambios significativos entre ellos.

Cabe destacar que el módulo de calibración (explicado en el apartado 3.4 de este documento) asigna a un punto el origen de coordenadas y unos ejes (que más tarde se verán en las imágenes) a partir de los cuales se toman las coordenadas de los puntos del QR.

Una vez determinado cual de tratamientos de QR se ha de seguir se extrae la información que contiene el QR, que se ha explicado en el apartado anterior como viene codificada, con las ordenes que se ven en la Figura 3.24.

```
byte=decodedObjects[i].data
string=str(byte)
vector=string.split(',')
a=float(vector[1])
b=float(vector[2])
c=float(vector[3])
d=float(vector[4])
e=float(vector[5])
f=float(vector[6])
```

Figura 3.24 Extracción de los datos del QR

Como se puede ver las comas se utilizan como elemento separador de los diferentes valores.

Una vez obtenidos todos los datos del QR se procede al tratamiento de la imagen. Para esto se va a utilizar la biblioteca OpenCV de nuevo que incluye órdenes para el tratamiento de imágenes.

Se comenzará situando los puntos del QR en la imagen (Figura 3.25 a) y especificando la medida del QR en la realidad (Figura 3.25 b).

```
#Puntos del QR en la foto
pst1 = [[point.x, point.y] for point in  points]
pts1 = np.float32(pst1)
```

```
#Medidas del QR
pts2 = np.float32([[0,0],[0,a],[a,a],[a,0]])
```

Figura 3.25 Puntos del QR en la imagen(a) y sus dimensiones en la realidad(b)

Con esto hallado se puede obtener la matriz de transformación, esta matriz permite conocer como está distorsionada la imagen con respecto a la realidad. La orden para crear la matriz se ve en la Figura 3.26.

```
#Calculamos la matriz de transformación
matrix, mask = cv2.findHomography(pts2, pts1, cv2.RANSAC, 5.0)
```

Figura 3.26 Creación de la matriz de transformación

El siguiente paso consiste en darle al programa la referencia de donde están las esquinas de la caja con respecto al punto de referencia del QR. Esta es la orden que cambia dependiendo de la orientación y el ángulo, ya que cambia el punto de referencia (Figura 3.27 a). Una vez dados estos puntos se transforman a la imagen mediante la matriz de transformación (Figura 3.27 b).

```
#vamos a dibujar en la posicion del QR el cuadrado
pts3 = np.float32([[-b,-c],[-b,e-c],[d-b,e-c],[d-b,-c]]).reshape(-1, 1, 2)
```

```
#Calcualmos los puntos del cuadrado con la perspectiva del QR
pts4 = cv2.perspectiveTransform(pts3, matrix)
```

Figura 3.27 Vértices de la caja en la realidad (a) y en la imagen (b)

Con esto ya se puede lanzar el programa de localización de los QR, en él se dibujará un rectángulo alrededor del QR y otro alrededor de la caja (Figura 3.28).

```
img = draw(points, img, decodedObjects[i].type, decodedObjects[i].data)
img = cv2.polylines(img, [np.int32(pts4m)], True, (255, 255, 0), 3)
```

Figura 3.28 Órdenes para dibujar

El resultado de estas órdenes se ve en la Figura 3.29.

Figura 3.29 Resultado del dibujo

Como se ve en la figura, el programa funciona y detecta las dimensiones de la caja, pero no lo hace de una forma 100% exacta. Para mejorar la detección se utilizaron dos métodos.

El primero consiste en la utilización de un alisado, la cámara no consigue una medición 100% exacta, por lo tanto, el rectángulo se estaría moviendo en todo momento, para evitarlo, se toman los últimos 15 frames y se hace la media lo que hace la medición mucho más estable. Este alisado se puede ver en la Figura 3.30.

```
if len(lista_punto1_img)<15:
    lista_punto1_img.append(pts4[0])
    lista_punto2_img.append(pts4[1])
    lista_punto3_img.append(pts4[2])
    lista_punto4_img.append(pts4[3])

else:
    lista_punto1_img.append(pts4[0])
    lista_punto2_img.append(pts4[1])
    lista_punto3_img.append(pts4[2])
    lista_punto4_img.append(pts4[3])

    lista_punto1_img.pop(0)
    lista_punto2_img.pop(0)
    lista_punto3_img.pop(0)
    lista_punto4_img.pop(0)
```

Figura 3.30 Alisado

La segunda mejora se dedica también a paliar posibles inexactitudes de la cámara, existe la posibilidad que en algunos frames un QR no sea detectado, para solucionar este problema se crea una lista que vaya aumentando cada vez que un QR no se detecte y se

ponga a 0 si se vuelve a detectar. En el caso de que no se detecte durante 60 frames seguidos se eliminará el QR. Esta programación se ve en la Figura 3.31.

```
if q==0:
#Sacar los datos de QR
    prim = pyzbar.decode(img)
    if len(prim)!=len(p_base):
        decodedObjects=prim
        p_base1=[]
        p_base=[]
        for i in range (len(decodedObjects)):
            p_base1.append(decodedObjects[i].polygon)
            p_base.append(p_base1[i][0][0])
        s2=np.zeros(len(decodedObjects))
        s=list(s2)
elif prim==[]:
    decodedObjects=prim
else:
    p=[]
    comp=pyzbar.decode(img)
    for i in range (len(comp)):
        p1.append(comp[i].polygon)
        p.append(p1[i][0][0])
    t=(10)

    for i in range (len(decodedObjects)):
        c=check_x_axis_list(p,p_base[i],t)
        if c==[]:
            s[i]=s[i]+1
        else:
            s[i]=0

        c=[]
    for i in range (len(decodedObjects)):
        if s[i]==59:
            decodedObjects.pop(i)
            s[i]=0
```

Figura 3.31 Programa para la eliminación de QR

Con estas mejoras la imagen queda mucho más estable como se puede ver en la Figura 3.32.

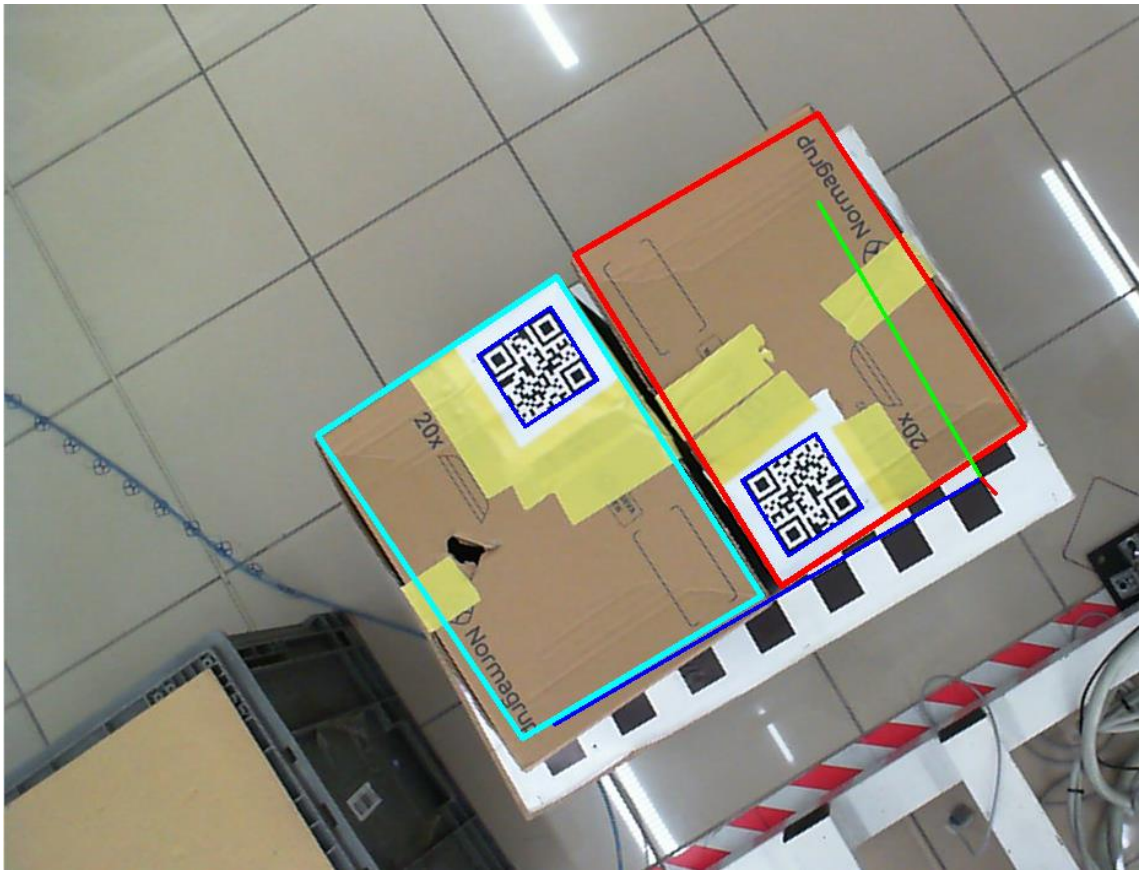


Figura 3.32 Detección de QR mejorada

Con todo esto realizado solo queda comentar un apartado del módulo. Este, está destinado a hallar la orientación de la caja, para determinar si está en la posición más estable de cara a paletizar o se debe pasar por la zona destinada al giro para orientarla de forma estable.

Para determinar si está bien orientada se comprueba si la altura es la más pequeña de las tres dimensiones principales (alto, ancho y largo) si no lo es, hay que girar la caja. Si hay que girarla se comprueba la orientación del QR para saber cuánto hay que girar la caja. Esta información al igual que la posición del centro se enviará al robot para que siga la

trayectoria necesaria. El código para hallar la orientación de la caja se muestra en la Figura 3.33.

```
if f>d and f>e:
    g="1"
elif f>d or f>e:
    g="2"

if orientation=="ZBarOrientation.DOWN":
    z="2"
elif orientation=="ZBarOrientation.LEFT":
    z="1"
elif orientation=="ZBarOrientation.UP":
    z="3"
elif orientation=="ZBarOrientation.RIGHT":
    z="4"
if g=="1" and z=="1":
    ori="1"
elif g=="1" and z=="2":
    ori="2"
elif g=="1" and z=="3":
    ori="3"
elif g=="1" and z=="4":
    ori="4"
elif g=="2" and z=="2":
    ori="5"
elif g=="2" and z=="1":
    ori="6"
elif g=="2" and z=="3":
    ori="7"
elif g=="2" and z=="4":
    ori="8"
else:
    ori="0"
```

Figura 3.33 Obtención de la orientación de la caja

Con esto se puede proceder a enviar los datos del centro y de la orientación al robot mediante el módulo de comunicación Socket que se explicará más adelante, para separar

los diferentes valores a enviar se utilizarán comas que en la parte del socket se explicará su utilidad.

Como resumen del módulo se tiene el flujograma de la Figura 3.34.

Figura 3.34 Resumen del módulo QR

3.7 DESARROLLO DEL MODULO PALLET

Este módulo tiene como cometido definir la distribución de las cajas en el pallet, creando una lista con cada uno de los puntos y enviándolos de uno en uno al robot para que sea el punto final de la trayectoria de este.

Este módulo necesita 6 datos como entrada para funcionar los tres primeros se llamarán d, e y f y corresponden a las dimensiones del pallet. Estas serán introducidas por pantalla por el usuario gracias al módulo de la GUI que se explicará más adelante.

Las otras tres entradas se llaman g, h e i y corresponden a las dimensiones de la caja, estas dimensiones se han obtenido en el módulo anterior.

Pero estas entradas no se pueden introducir al programa sin más. Como se explicó anteriormente las cajas se deben colocar en el pallet de la forma más estable y para ello se deben apoyar sobre la dimensión más pequeña de las principales. Para cumplir con esto se utiliza el código que se muestra en la Figura 3.35.

```
lados=(d,e,f)
Lcaja=max(lados)
Acaja=secondMax(lados)
zcaja=min(lados)
```

Figura 3.35 Asignación de los lados de la caja

Una vez asignadas las dimensiones se calcula el área del pallet y de la caja y con ella el número de cajas por piso y los pisos que tendrá el pallet como se ve en la Figura 3.36.

```
Areapale=Lpale*Apale
Areacaja=Lcaja*Acaja

Ncajas=int(Areapale/Areacaja)
Pisos=int(Zpale/zcaja)
```

Figura 3.36 Obtención del nº de cajas por piso y del número de pisos

Para el correcto funcionamiento de este módulo se crearán tres variables de estado llamadas *ncaja*, *nlinea* y *zbase*, Estas variables indicarán la caja que se está moviendo en ese momento y servirán como límite (por ejemplo, una vez que *zbase* supere el número de plantas calculado el proceso de paletización terminará).

El siguiente paso es la obtención de número de cajas que se van a colocar en cada línea. Para poder obtener este valor, primero se ha de conocer cómo se van a colocar las cajas sobre el pallet. Para conocer esto se crea un algoritmo que divide cada una de las dimensiones principales del pallet entre cada una de las dimensiones de la caja.

Si el resultado es exacto las cajas se distribuirán de esa forma, si, en cambio, ninguno de los resultados es exacto, se calcula el número cajas que entran colocándolas con su longitud paralela a la longitud del pallet o con su ancho paralelo a la longitud del pallet. Una vez obtenidas las cajas que van en cada línea se calculan el número de líneas por piso dividiendo el número de cajas entre las cajas por línea. Este algoritmo se puede ver en la Figura 3.37.

```
if Lpale%Lcaja==0:
    cajas_linea=int(Lpale/Lcaja)
    z="exacto"
elif Lpale%Acaja==0:
    cajas_linea=int(Lpale/Acaja)
    z="exacto"
elif Apale%Acaja==0:
    cajas_linea=int(Lpale/Lcaja)
    z="exacto"
elif Apale%Lcaja==0:
    cajas_linea=int(Apale/Lcaja)
    z="exacto"
else:
    cajas_lineaL=int(Lpale/Lcaja)
    cajas_lineaA=int(Lpale/Acaja)
    z="noexacto"
if z=="exacto":
    lineas=int(Ncajas/cajas_linea)
else:
    lineasL=int(Apale/Acaja)
    lineasA=int(Apale/Lcaja)
    cajas1=cajas_lineaL*lineasL
    cajas2=cajas_lineaA*lineasA
```

Figura 3.37 Algoritmo para la obtención del número de cajas por línea

Con esto se puede proceder a realizar la paletización, se distinguirá primero entre el resultado exacto y el no exacto como se ha explicado anteriormente. Dentro de cada uno de los casos también se realizan una serie de subdivisiones.

Dentro de los exactos se definen cuatro tipos distintos dependiendo de cuál es la división que resulta exacta. En esta memoria solo se va a explicar uno de los casos al ser los cuatro muy similares entre sí.

Se tomará el caso en el cual la longitud del pallet entre la longitud de la caja de un resultado exacto. Para el correcto funcionamiento del módulo se deben crear una serie de matrices que contengan las dimensiones de la caja. La primer de ellas corresponde al punto en el que se depositará la primera caja, hay que recordar que la caja viene en el robot sujetada por su punto medio así que el punto de destino ha de ser el medio del hueco asignado para la caja. La altura será la de una caja ya que el punto para soltarla el superior (si no, la caja se aplastaría). El punto inicial se ve en la Figura 3.38.

```
base=[Lcaja/2,Acaja/2,0]
```

Figura 3.38 Coordenadas del primer punto del pallet.

Las siguientes matrices a crear son las que marcan el avance en el pallet, estas matrices son las que cambian dependiendo de cuál sea el caso de exacto en el que se haya entrado. La matriz a corresponderá a la dimensión de la caja que irá en paralela al ancho del pallet y la matriz b a la dimensión paralela a la longitud. Estas matrices se multiplicarán por la n para que avance con cada caja y por la n de línea cuando se cambie de línea. Cuando se cambie de piso aumentará la Z. Estas matrices se ven en la figura 3.39.

```
a=[0,Acaja,0]
b=[Lcaja,0,0]
c=[numero*nlinea for numero in a]
d=[numero*n for numero in b]
z=[0,0,Z]
```

Figura 3.39 Matrices para el avance del pallet

El siguiente paso es sumar estos valores para hallar las coordenadas de cada una de las cajas. Una vez obtenidas las coordenadas de cada una de las cajas se van añadiendo a una

lista que se utilizará para enviar los datos al robot. Al finalizar este proceso se añade 1 a la n para pasar a la siguiente caja. Esto se puede ver en la Figura 3.40.

```
coord_caja1=list(map(lambda x,y:x+y,c,d))
coord_caja2=list(map(lambda x,y:x+y,coord_caja1,base))
coord_caja=list(map(lambda x,y:x+y,coord_caja2,z))
coord=str(coord_caja)
lista_coord.append(coord)
n=n+1
```

Figura 3.40 Obtención de las coordenadas de cada caja

Para finalizar con esta parte del módulo se añade un algoritmo que comprueba cuando la n iguala al valor anteriormente calculado para cajas_linea, si esto ocurre significa que se ha completado la línea y se procede a comenzar con la siguiente poniendo a 0 el valor de n y añadiendo 1 al valor de nlinea. Cuando este valor de nlinea alcance el valor de líneas se pondrá a 0 y se añadirá uno a zbase cambiando de piso. Este proceso se repetirá hasta que se alcance el número de pisos previamente calculado y se ve en la Figura 3.41.

```
if lineas==1:
    if n==cajas_linea:
        nlinea=nlinea+1
        n=0
else:
    for i in range (1,lineas):
        if n==i*cajas_linea:
            nlinea=nlinea+1
            n=0
```

Figura 3.41 Algoritmo para pasar de línea

Con esto se termina con la parte “sencilla” del algoritmo. La que corresponde a cajas que son divisores exactos del pallet. Se comenzará ahora a explicar el caso de que no se pueda encontrar un divisor exacto.

Este caso conlleva más complicaciones ya que se debe comprobar que distribución es más favorable en cuanto a número de cajas por piso, si la de colocar las cajas con su longitud paralela a la longitud del pallet o la de colocar las cajas con su ancho paralelo a la longitud del pallet.

Además, al no ser exacto cuando se seleccione la distribución pueden quedar huecos en el pallet que se pueden aprovechar para colocar cajas en la orientación no elegida para aprovechar espacio.

El primer paso será, como se ha explicado, será comparar la cantidad de cajas que cabrían por piso si se sitúan con su longitud paralela a la longitud del pallet o con su ancho paralelo al ancho del pallet (Figura 3.42).

```
if (lineasL*cajas_lineaL)>(lineasA*cajas_lineaA):
```

Figura 3.42 Comparación entre distribuciones.

Dependiendo de cuál de estos resultados sea mayor, se seleccionarán las matrices de avance tal y como se hacía en el caso exacto que se explicó anteriormente. Tras esto se seguirá el proceso anterior con una salvedad, al acabar cada una de las filas se calculará el hueco restante en cada dirección con las siguientes ordenes de programación (el valor llamado j corresponde al hueco restante a la longitud) (Figuras 3.43 a y b).

```
nuevo_ancho=Apale-nlinea*Acaja
```

```
p=coord_caja[0]-Lcaja/2
k=Lcaja
j=p+k
```

Figura 3.43 Cálculo del ancho (a) y de la longitud restante (b).

Al acabar cada una de las filas, estos valores calculados se compararán con las dimensiones de la caja como se hacía al comienzo del módulo para ver si una de las dimensiones da un resultado exacto. Si se diera esta circunstancia se pasaría al modo exacto explicado anteriormente y ya se procedería hasta el final.

En el caso de no llegar al modo exacto se procederá a colocar la siguiente línea de cajas y se continuará comprobando al final de cada una.

Por último, cuando se acaben todas las líneas se comprobarán los posibles huecos que queden en el pallet para ver si existe la posibilidad de introducir alguna caja más.

Además de todo esto, en la empresa ya se paletizan las cajas de forma manual y existen ya distribuciones hechas, se han diseñado listas con estas distribuciones para su fácil acceso en el caso de que las cajas ya existieran en la empresa.

En el caso de que se comiencen a fabricar nuevos tipos de productos que requieran cajas se podrá pasar por todo el proceso de paletización para crear una solución óptima y estable.

Para el envío de coordenadas se utilizará una variable auxiliar que irá aumentando de uno en uno cada vez que se le piden datos. Esta variable servirá para que se envíe el valor de la lista de coordenadas en la posición de la variable.

Como resumen del módulo se tiene el flujograma de la Figura 3.44.

Figura 3.44 Resumen del módulo pallet

3.8 DESARROLLO DEL MODULO DE CONEXIÓN SOCKET

El siguiente modulo a tratar es el de la conexión Socket, este tiene como objetivo realizar la comunicación entre el robot y el programa de visión. Como se ha explicado anteriormente el programa de visión enviará unas coordenadas de posición de la caja al robot, por otro lado, el módulo pallet también envía las coordenadas de la posición de la caja en el pallet.

Como este módulo involucra dos tipos de programaciones distintas (la del visor en Python y la del robot en RobotStudio) se dividirá en dos partes.

La primera de ellas será la que actúe como servidor en la conexión. Este servidor estará todo el rato en espera hasta que reciba la conexión de un cliente que será el robot.

El primer paso para poder realizar la conexión será la creación del servidor, para poder crearlo, se debe crear el ADDR (acrónimo de inglés *addres)* el ADDR se compone de dos partes la IP del servidor y el puerto de conexión.

La IP del servidor será la dirección IP asociada a ese ordenador, esta se podrá obtener de dos formas, escribiéndola de forma manual u obteniéndola mediante una orden de programación de Python como la que se ve en la Figura 3.45.

```
SERVER=socket.gethostbyname(socket.gethostname())
```

Figura 3.45 Obtención de la dirección IP

En el caso del puerto se puede seleccionar cualquier puerto del ordenador que no esté ocupado. Una vez seleccionadas la dirección IP y el puerto se combinarán mediante la siguiente orden de programación (Figura 3.46).

```
server.bind(ADDR)
```

Figura 3.46 Obtención del ADDR

La parte de la comunicación Socket en Python se completa con dos funciones, la primera de ellas es la que sirve para manejar las conexiones del cliente, esta recibe la información del visor, lanza el programa pallet y prepara el mensaje que se va a enviar. La función se puede ver en la Figura 3.47.

```
def handle_client(conn, addr,x,y,z):
    global conx
    print(f"[Nueva conexión:] {addr} se ha conectado.")
    pale(d,e,f,x,y,z)
    conn.send(centrop+ab)
    conn.close()
```

Figura 3.47 Función para el manejo de los clientes

La segunda función gestiona el inicio del servidor, esta funciona con un *"threading"*, un *threading* es un módulo de Python que sirve para mantener una orden en ejecución en segundo plano mientras que se realizan otras. Esto permitirá que el servidor esté funcionando en todo momento, aunque es ese momento se esté trabajando con la cámara o con otro de los módulos del programa. La función para el inicio del servidor se puede ver en la Figura 3.48.

```
#Función para lanzar el servidor
def start(x,y,z):
    server.listen()
    print(f"Servidor recibiendo en {SERVER}")
    while True:
        conn, addr = server.accept()
        thread = threading.Thread(target=handle_client, args=(conn, addr,x,y,z))
        thread.start()
        print(f"[Conexiones activas] {threading.activeCount() - 1}")
```

Figura 3.48 función de inicio del servidor

Con esto se finaliza con la explicación de la parte de Python, la otra parte del módulo socket, que funciona como cliente, se gestiona en el programa RobotStudio.

Esta parte es la encargada de recibir la información enviada desde Python y convertirla en una más manejable para el robot. La primera parte de esta consiste en la creación del cliente y su conexión al servidor, al igual que en el caso de Python esta necesita del ADDR para funcionar, pero la IP no se puede obtener de forma automática y se escribirá manualmente como se ve en la Figura 3.49.

```
SocketCreate client;
SocketConnect client, "1.1.13.97", 5050;
SocketReceive client,\Str:= inf_recibida;
```

Figura 3.49 Creación y conexión del cliente

Una vez recibida la información, esta se procesa hasta convertirla en variables para ser utilizadas en la programación del robot.

Para esta tarea son muy importantes las comas que se mencionaron anteriormente. Mediante la orden StrFind se puede obtener la posición de cada una de estas comas y utilizar estas posiciones para la orden StrPart, esta orden se usa para obtener cada uno de los valores. Una vez obtenidos se convierten a variables con la orden Strtoval. Esto se hará tanto para las coordenadas de recogida de la caja (Figura 3.50).

```
spl1:=StrFind(inf_recibida,1,",");
spl2:=StrFind(inf_recibida,spl1+1,",");
spl3:=StrFind(inf_recibida,spl2+1,",");
spl4:=StrFind(inf_recibida,spl3+1,",");
centro1:=StrPart(inf_recibida,1,spl1-1);
centro2:=StrPart(inf_recibida,spl1+1,spl2-spl1-1);
centro3:=StrPart(inf_recibida,spl2+1,spl3-spl2-1);
ori:=StrPart(inf_recibida,spl3+1,spl4-spl3-1);
okx:=StrtoVal (centro1,centrox);
oky:=StrtoVal (centro2,centroy);
okz:=StrtoVal (centro3,centroz);
```

Figura 3.50 Obtención de las coordenadas de recogida de la caja

Como para la obtención de las coordenadas donde se deposita en el pallet (Figura 3.51).

```
spl12:=StrFind(inf_recibida,spl4,"[");
spl22:=StrFind(inf_recibida,spl12+1,",");
spl32:=StrFind(inf_recibida,spl22+1," ");
spl42:=StrFind(inf_recibida,spl32+1,",");
spl52:=StrFind(inf_recibida,spl42+1," ");
spl62:=StrFind(inf_recibida,spl52+1,"]");
centro12:=StrPart(inf_recibida,spl12+1,spl22-spl12-1);
centro22:=StrPart(inf_recibida,spl32+1,spl42-spl32-1);
centro32:=StrPart(inf_recibida,spl52+1,spl62-spl52-1);
okx2:=StrtoVal (centro12,centrox2);
oky2:=StrtoVal (centro22,centroy2);
okz2:=StrtoVal (centro32,centroz2);
```

Figura 3.51 Obtención de las coordenadas de destino

Tras esto se puede cerrar la conexión Socket con la orden SocketClose, este proceso se repetirá al inicio de cada iteración del robot, una vez realizado el robot realizará todos sus movimientos (que se explicarán en el próximo módulo) y al acabar volverá a una posición de reposo en la cual pedirá de nuevo las coordenadas por Socket.

Como resumen del módulo se utiliza el flujograma de la Figura 3.52.

Figura 3.52 Resumen del módulo Socket

3.9 DESARROLLO DEL MODULO PARA EL MOVIMIENTO DEL ROBOT

Este módulo tiene como objetivo gestionar los movimientos del robot. Se divide en tres partes, las dos primeras que son la comunicación socket y la gestión de los datos recibidos se explicaron en el módulo anterior.

La tercera es la llamada "movimientos" y, como su nombre indica gestiona los movimientos del robot. Esta, también se dividirá en tres partes, la primera serán los movimientos para la recogida de la caja. La segunda gestiona el giro de la caja si fuera necesario y, por último, la tercera gestiona el depósito de la caja en el pallet.

Un aspecto importante a comentar es el uso de los WorkObjects u Objetos de Trabajo, estos se utilizan para situar el eje de coordenadas en cada una de las partes ya que las coordenadas se dan con respecto al origen, por ejemplo, en el pallet para una caja de 200x100x100 las coordenadas iniciales serían (100,50,100) con respecto a una de las esquinas del pallet.

Estos WorkObjects se situarán in situ con el robot gracias a un método de definición de objetos de trabajo que se basa situar tres puntos, el origen de coordenadas, un punto en el eje x y un punto en el eje y. Con esto, el Flex Pendant creará automáticamente el objeto de trabajo.

La primera parte consistirá en situarse 100 mm por encima de las coordenadas de recogida, una vez aquí se comprueba mediante el sensor de posición si la caja sigue allí, si está se bajará lentamente hasta recoger la caja y se activará el sistema para recogerla con vacío. Una vez hecho esto se elevará la caja. La programación para esto se ve en la Figura 3.53, los valores que se ven en la parte final de las posiciones corresponden a la orientación de la herramienta y a la configuración del robot.

Todas estas órdenes se ven en la Figura 3.53.

```
!Moverse a la posición de la caja
WoActual:=WoCaja;
pos1:=[[centrox,centroy,centroz]+[0,0,-100],[0.04910,0.01451,0.01902,-0.99851],[-1,-2,1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
SETDO abregarra,1;
MOVEJ pos1, v200, z50, pinza\WObj:=WoActual;
!Comprobar pieza en el sitio
IF DI1=1 THEN
    !Bajar a por la pieza
    pos2:=[[centrox,centroy,centroz+20],[0.04910,0.01451,0.01902,-0.99851],[-1,-2,1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    MOVEJ pos2, v50, fine, pinza\WObj:=WoActual;
    !Coger la pieza
    SETDO vacio_garra, 1;
    !Esperar 1 segundo a que se cierre
    WAITTIME 1;
    MOVEJ pos1, v200, z50, pinza\WObj:=WoActual;
```

Figura 3.53 Movimientos para la recogida de la caja.

La segunda parte corresponde al giro de la caja, a esta parte solo se llegará cuando el valor de ori sea distinto de 0 (la obtención de este valor se explicó en el módulo QR).

Si es necesario pasar por este proceso, el robot ira pasando por diferentes posiciones para ir cambiando la orientación hasta llegar a soltar la caja en la mesa para el giro.

Como se explicó anteriormente esta mesa está inclinada y hará que la caja se sitúe en una posición óptima para su recogida por la parte superior. Una vez recogida se puede elevar para pasar al último proceso. Las ordenes que componen este proceso se pueden ver en la Figura 3.54.

```
IF ori="2" THEN
    !Ir a posición intermedia
     WoActual:=WoParo;
    MOVEJ pos6, v200, z50, pinza\WObj:=WoActual;
    MoveJ pos16, v100, z50, pinza\WObj:=WoActual;
    !Bajar la caja
     MOVEJ pos8, v20, z50, pinza\WObj:=WoActual;
    SETDO vacio_garra, 0;
    !Esperar 1 segundo
    WAITTIME 1;
    !Ir a posición intermedia
    MOVEJ pos14, v100, z50, pinza\WObj:=WoActual;
    !Coger la caja en buena posición
    MoveJ pos24, v100, z50, pinza\WObj:=WoActual;
    MOVEJ pos15, v100, fine, pinza\WObj:=WoActual;
    MoveJ pos5, v100, fine, pinza\WObj:=WoActual;
    MoveJ pos25, v20, fine, pinza\WObj:=WoActual;
    MoveJ pos35, v20, fine, pinza\WObj:=WoActual;
    !Coger la pieza
    SETDO vacio_garra, 1;
    !Esperar 1 segundo
    WAITTIME 1;
    MOVEJ pos26, v200, z50, pinza\WObj:=WoActual;
    MoveJ pos36, v200, z50, pinza\WObj:=WoActual;
ENDIF
```

Figura 3.54 Movimientos para el giro de la caja.

En la figura se muestra como ejemplo la orientación 2, el resto son muy similares, salvo que las posiciones para el giro son diferentes.

La última parte es el depósito en el pallet, esta funciona al igual que la primera, el robot se sitúa encima del punto de depósito y baja lentamente hasta depositarla en buena posición. Estas órdenes se ven en la Figura 3.55.

```
WoActual:=WoPale;
pos3:=[[centrox2,centroy2,-centroz2]+[0,0,-200],[0.04910,0.01451,0.01902,-0.99851],[-1,-2,1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
MOVEJ pos3, v200, z50, pinza\WObj:=WoActual;
!Acercarse a la posicion para dejar la caja
pos4:=[[centrox2,centroy2,-centroz2],[0,1,0,0],[0.04910,0.01451,0.01902,-0.99851],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
MOVEJ pos4, v100, fine, pinza\WObj:=WoActual;
!Soltar la pieza
SETDO vacio_garra, 0;
!Esperar 1 segundo
WAITTIME 1;
!Subir
MOVEJ pos3, v200, z50, pinza\WObj:=WoActual;
!Volver a la posición de inicio
MOVEJ pos0, v200, z50, pinza;
```

Figura 3.55 Movimientos para el depósito de la caja.

Como resumen del módulo se utiliza el flujograma de la Figura 3.56.

Figura 3.56 Resumen del módulo Robot

3.10 DESARROLLO DE LA GUI

El último módulo del que se va a hablar en esta memoria es la GUI. La GUI es una interfaz que se diseña para que se muestre en la pantalla de la unidad de control al ser utilizada la célula y hace que sea muy fácil e intuitivo para un usuario sin conocimiento del proyecto su uso.

Par diseñar esta interfaz se utiliza el programa Qt Designer (Figura 3.57) que permite diseñarla de forma visual sin entrar en programación y una vez diseña mediante la extensión pyuic5 del Python se puede convertir a un archivo .py y añadirle ordenes de programación para darles funciones.

Figura 3.57 Logo del Qt Designer

La interfaz estará diseñada con botones para que tenga un fácil manejo, también incluirá un espacio para que se muestre la imagen de la cámara y así se pueda controlar el proceso y pararlos si existieran errores. La interfaz tendrá el aspecto que se ve en la Figura 3.58.

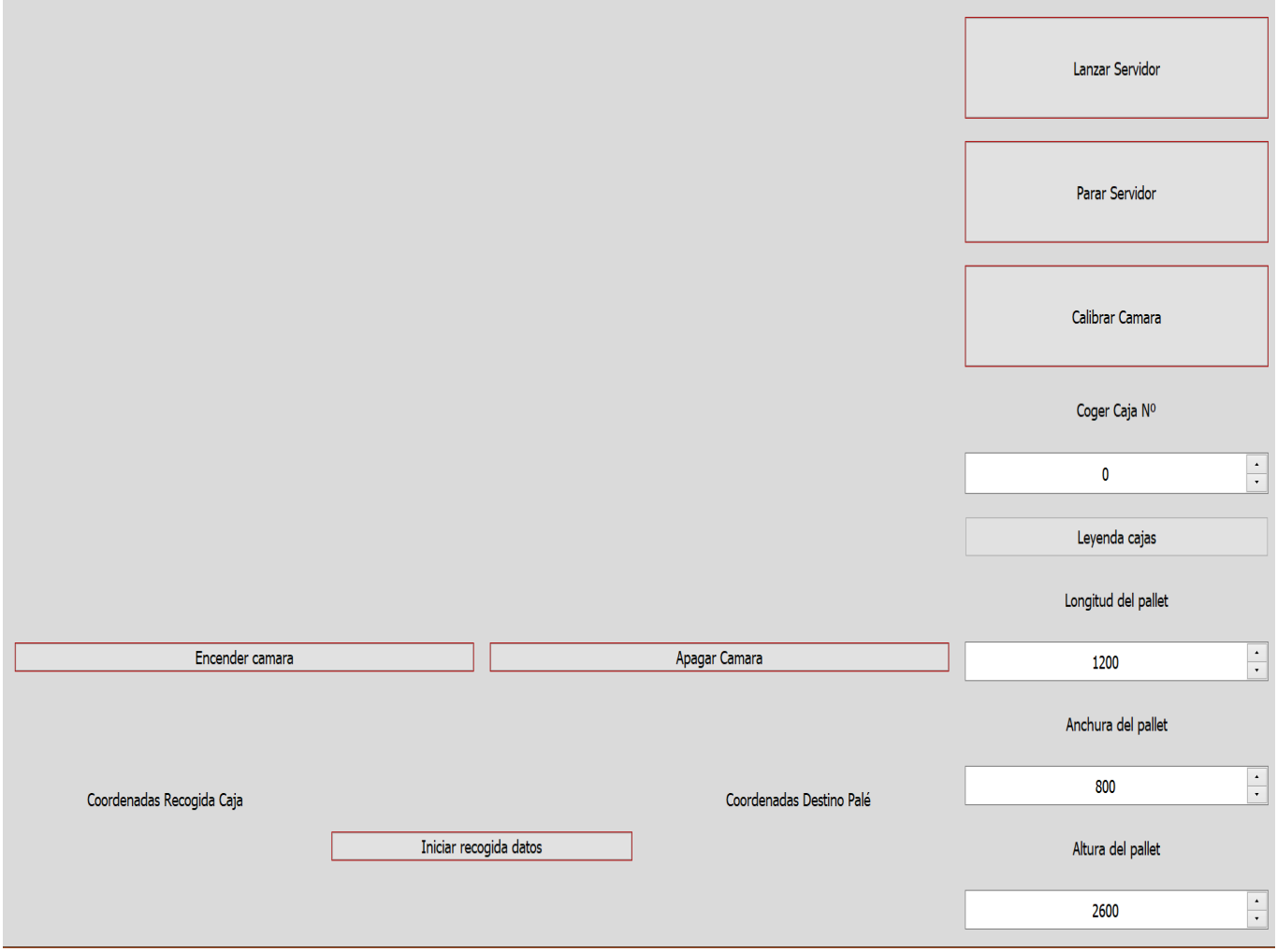


Figura 3.58 Interfaz

Cada uno de los botones que se puede ver en la imagen tiene una función asignada que se lanza automáticamente al pulsarlos. Las entradas de valores en cambio sirven para introducir de forma fácil por pantalla algunos de los valores que se van a utilizar en el resto de los módulos. Las órdenes para configurar los botones se ven en la Figura 3.59.

```
self.pushButton_5.clicked.connect(self.controlTimer)
self.pushButton_4.clicked.connect(self.cerrar_camara)
self.pushButton.clicked.connect(self.lanzar)
self.pushButton_2.clicked.connect(self.parar)
self.pushButton_3.clicked.connect(self.calibra)
self.pushButton_6.clicked.connect(self.leyenda)
self.Actualizar.clicked.connect(self.worker3)
```

Figura 3.59 Órdenes para configurar los botones.

Se comenzará por el botón Lanzar Servidor este al ejecutarlo crea el servidor Socket, revisa el valor de las longitudes del pallet que se han introducido y los toma como entradas del módulo con el mismo nombre.

Una vez que se tienen los datos se realizará la segunda comprobación, se comprobará el valor de una variable vcamara, si este es verdadero significa que se ha activado la cámara, si es falso que la cámara no se ha activado. Para el correcto funcionamiento del proyecto es necesario que se encienda la cámara antes de lanzar el servidor, así que, si la cámara no está encendida se mostrará un mensaje de error como se ve en la Figura 3.60.

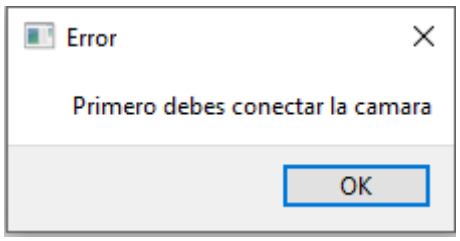


Figura 3.60 Error por lanzar el servidor antes de encender la cámara

Si la cámara esta encendida se podrá lanzar el servidor y se mostrará un mensaje como el siguiente (Figura 3.61).

```
Servidor recibiendo en 1.1.14.186
```

Figura 3.61 Servidor lanzado

La programación del lanzado del servidor se ve en la Figura 3.62.

```
def lanzar(self):
        self.xyz()
        if vcamara==True:
            self.worker=WorkerThread()
            self.worker.start()
        if vcamara==False:
            msg = QMessageBox()
            msg.setWindowTitle("Error")
            msg.setText("Primero debes conectar la camara")
            x = msg.exec_()
```

Figura 3.62 Programación del botón lanzar

Los otros dos botones de la parte superior izquierda son más sencillos, El botón de Parar Servidor para los procesos en ejecución, mientras que el botón Calibrar lanza el módulo Web Cam Calibration explicado anteriormente. La programación de estos botones se ve en la Figura 3.63.

```
def parar(self):
    QR1.exit()

def calibra(self):
    calibrar.calibracion()
```

Figura 3.63 Botón Parar y Botón Calibra

Siguiendo por la parte izquierda de la pantalla, en la zona inferior se sitúan tres SpinBox en las que el usuario puede introducir las dimensiones del Pallet, normalmente no se va a

variar estas dimensiones y se utilizará el pallet europeo (1200x800), pero esta opción se deja abierta por la posibilidad de tener que usar un pallet especial. Estas dimensiones se obtienen de las SpinBoxes mediante las siguientes órdenes (Figura 3.64).

```
x=self.spinBox_4.value()
y=self.spinBox_5.value()
z=self.spinBox_6.value()
c=self.spinBox.value()
```

Figura 3.64 Obtención de los valores del pallet

Además de los tres valores, en la imagen se puede ver otro valor llamado c. este valor corresponde a la SpinBox situada en la cona central izquierda, esta se utiliza para seleccionar la caja a tratar. El proyecto está diseñado para poder detectar hasta 10 cajas que aparezcan en la zona del visor, una vez detectadas con la citada Spin Box el usuario puede seleccionar cual es la que quiere que el robot coja.

Para una fácil identificación de la caja se ha diseñado un sistema que da un color distinto a las líneas que se dibujan para marcar el contorno de cada caja. El usuario puede conocer este código de colores en el botón Leyenda Cajas y con ello elegir la caja que quiera. El código de colores se muestra en la Figura 3.65.

Caja 0:Borde Azul Claro
Caja 1:Borde Rojo
Caja 2:Borde Amarillo
Caja 3:Borde Verde
Caja 4:Borde Negro
Caja 5:Borde Blanco
Caja 6:Borde Gris
Caja 7:Borde Rosa
Caja 8:Borde Marrón
Caja 9:Borde Azul Oscuro

Figura 3.65 Código de colores para la selección de la caja

La siguiente parte a explicar es la zona inferior, esta parte funciona como una barra de estado. En el momento en el que se haga clic en el botón de iniciar recogida de datos se inicia un proceso en el cual se toman y actualizan los datos de las coordenadas de posición de recogida y de depósito a tiempo real como se ve en la Figura 3.66.

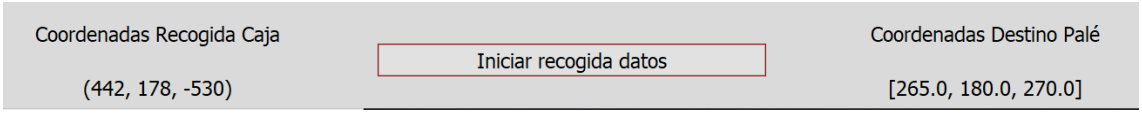


Figura 3.66 Barra de estado

Para poder realizar este trabajo se utiliza un WorkerThread, un WorkerThread es muy similar al anteriormente citado Threading, pero adaptado y con una compatibilidad mejorada con la extensión pyuic5.

Este WorkerThread mira el valor de c y lo almacena en otra variable llamada f, con esta variable entra el módulo QR correspondiente a la caja seleccionada e importa las coordenadas de recogida y de depósito (llamadas asa y asas en la figura). Una vez recogidas las emite con una señal y las proyecta en pantalla. La programación del WorkerThread se ve en la Figura 3.67.

```
class WorkerThread2(QThread):

        sig1=pyqtSignal()

        def __init__(self, parent=None):
            QThread.__init__(self,None)

        def run(self):
            global asa
            global asas
            while True:
                if f==0:
                    from QR1 import asas
                    from QR1 import asa
                elif f==1:
                    from QR2 import asas
                    from QR2 import asa
                elif f==2:
                    from QR3 import asas
                    from QR3 import asa
                elif f==3:
                    from QR4 import asas
                    from QR4 import asa
                elif f==4:
                    from QR5 import asas
                    from QR5 import asa
                elif f==5:
                    from QR6 import asas
                    from QR6 import asa
                elif f==6:
                    from QR7 import asas
                    from QR7 import asa
                elif f==7:
                    from QR8 import asas
                    from QR8 import asa
                elif f==8:
                    from QR9 import asas
                    from QR9 import asa
                else:
                    from QR10 import asas
                    from QR10 import asa
                self.sig1.emit()
                time.sleep(1)
```

Figura 3.67 Programación del WorkerThread

Como se ve en la figura este WorkerThread es llamado como WorkerThread2. El primero de ellos se utiliza para la gestión de la cámara que es el único aspecto que queda por tratar de la GUI.

La cámara se gestiona mediante los dos botones que tiene situados debajo. Como su nombre indica el botón de encender encenderá la cámara y el de apagar la apagará.

Para que la cámara funcione a tiempo real, se utilizará un Timer como se ve en la Figura 3.68, que cada cierto tiempo vuelva a activar la cámara.

```
def controlTimer(self):
    # if timer is stopped
    global vcamara
    vcamara=True
    if not self.timer.isActive():
        # create video capture
        self.cap = cv2.VideoCapture(0)
        self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1920)
        self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)
        # start timer
        self.timer.start(20)

    # if timer is started
    else:
        # stop timer
        self.timer.stop()
        # release video capture
        self.cap.release()
```

Figura 3.68 Configuración del Timer

Una vez que se está tomando la imagen esta sufre una serie de transformaciones con la extensión OpenCV para adaptarla a la GUI, en cuanto a tamaño y resolución. Estas conversiones se ven en la Figura 3.69.

```
# convert image to RGB format
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
# get image infos
height, width, channel = image.shape
step = channel * width
# create QImage from image
qImg = QImage(image.data, width, height, step, QImage.Format_RGB888)
# show image in img_label
Pix=QPixmap.fromImage(qImg)
Pix.scaledToWidth(self.label_7.width())
Pix.scaledToHeight(self.label_7.height())
self.label_7.setPixmap(Pix)
self.label_7.setAlignment(QtCore.Qt.AlignHCenter | QtCore.Qt.AlignVCenter)
```

Figura 3.69 Conversiones de la imagen

Con esto se termina con la configuración de la GUI, cabe destacar que esta interfaz se puede utilizar con cualquier ordenador, ya que la interfaz se adapta a cualquier tamaño de pantalla o resolución.

3.11 OTROS MÓDULOS

Por último, se van a mencionar una serie de pequeños módulos que no son tan importantes como los anteriores pero que son necesarios para el funcionamiento del proyecto. Estos se dividen en dos grupos de funciones.

Las dos primeras son las funciones que se utilizan para cargar y guardar los datos de la calibración de la cámara, cuando se realiza una nueva calibración sobrescribe los datos y cuando se utiliza la cámara carga los datos de la última calibración realizada (Figura 3.70).

```python
#Funcion para guardar los datos de la calibración de la camara
def guardaDatos():
    global Ruta_fichero_calibracion, CameraMatrix, DistCoeficients, rvecs, tvecs, rMat

    SaveDict = {}
    SaveDict["CameraMatrix"] = CameraMatrix.tolist()
    SaveDict["DistCoeficients"] = DistCoeficients.tolist()
    SaveDict["rvecs"] = rvecs.tolist()
    SaveDict["tvecs"] = tvecs.tolist()

    json.dump(SaveDict, codecs.open(Ruta_fichero_calibracion, 'w', encoding='utf-8'), separators=(',', ':'),
              sort_keys=True, indent=4)


#Función para cargar datos de un fichero
def cargaDatos():
    global Ruta_fichero_calibracion, CameraMatrix, DistCoeficients, rvecs, tvecs, rMat

    obj_text = codecs.open(Ruta_fichero_calibracion, 'r', encoding='utf-8').read()
    Datos = json.loads(obj_text)

    CameraMatrix = np.array(Datos["CameraMatrix"])
    DistCoeficients = np.array(Datos["DistCoeficients"])
    rvecs = np.array(Datos["rvecs"])
    tvecs = np.array(Datos["tvecs"])
    rMat = cv2.Rodrigues(rvecs)[0]
```

Figura 3.70 Funciones para guardar y cargar

El otro grupo de funciones es el que se utiliza para convertir los datos del mundo real a coordenadas de la imagen y viceversa. Estas se utilizan para convertir los puntos que se sacan de la imagen (los vértices del QR) a coordenadas del mundo real para pasárselas al robot. Estas funciones se ven en la Figura 3.71.

```python
#Función para pasar puntos de la imagen al mundo real
def ImagenTo3D(pto, z):
    pto1 = [[float(pto[0])], [float(pto[1])], [1.0]]

    tempMat = np.dot(np.dot(np.linalg.inv(rMat), np.linalg.inv(CameraMatrix)), pto1)
    tempMat2 = np.dot(np.linalg.inv(rMat), tvecs)

    s = float(z) + tempMat2[2, 0]
    s /= tempMat[2, 0]

    P = np.dot(np.linalg.inv(rMat), (s * np.dot(np.linalg.inv(CameraMatrix), pto1) - tvecs))

    return map(int, P)


#Función para pasar de puntos del entorno a puntos en la imagen
def D3toImagen(pto):
    res = cv2.projectPoints(np.array([pto]).astype(np.float32), rvecs, tvecs, CameraMatrix, DistCoeficients)
    return res[0][0][0]
```

Figura 3.71 Funciones para convertir entre imagen y mundo real

4. Pruebas de funcionamiento.

En este apartado se explicará todo el proceso de pruebas que se siguió hasta poner la célula en funcionamiento, siguiendo un orden cronológico. Al ser una célula destinada a trabajar diariamente en el almacén se realizaron multitud de pruebas para comprobar la no existencia de errores de funcionamiento.

4.1 PRUEBAS DE FUNCIONAMIENTO DEL VISOR

Se comenzará explicando el proceso de probatura de funcionamiento del sistema de visión artificial. Este sistema es que más probaturas ha necesitado para verificar su funcionamiento.

En el proceso se utilizaron diferentes cámaras hasta llegar a la utilizada para el proyecto. La primera de ellas fue la Logitech C310 HD (Figura 4.1), esta se comenzó utilizando debido a su disponibilidad en los almacenes de la empresa mientras llegaba la primera cámara pedida específicamente para el proyecto.

Figura 4.1 Cámara Logitech C310

Se comenzó probando la detección de las cajas con diferentes métodos. La detección de los vértices con el módulo de Python Canny Edge Detector o la comparación con un "esqueleto" con el módulo Human Pose Estimation. La utilización de estos módulos permite la detección de los vértices de las cajas, pero los resultados son inexactos así que se decidió por el módulo Pyzbar cuyos resultados eran mucho más exactos.

Una vez decido el módulo a utilizar, se comenzó a probar la detección de códigos QR comenzando por códigos situados en imágenes. Como se ven en la Figura 4.2 se consiguió delimitar la posición del QR y extraer su contenido.

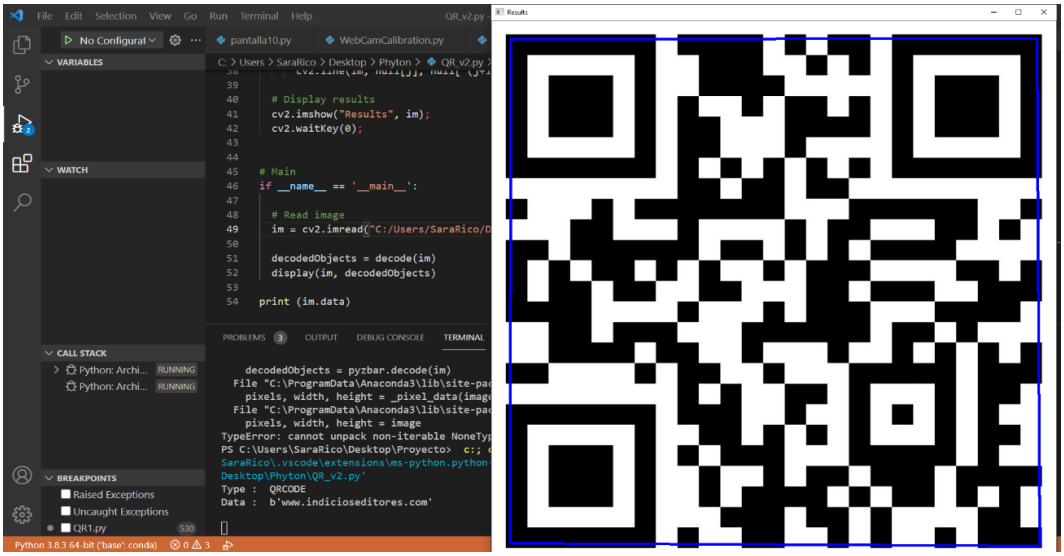


Figura 4.2 Prueba de detección de QR en imágenes estáticas

Una vez que se consiguió detectar QR en imágenes, se procedió a comenzar las pruebas para detectar los QR en la imagen a tiempo real tomada mediante la videocámara. Como se ve en la Figura 4.3 las primeras pruebas no daban resultados correctos, porque pese a detectar el código QR la detección de la caja no estaba bien orientada.

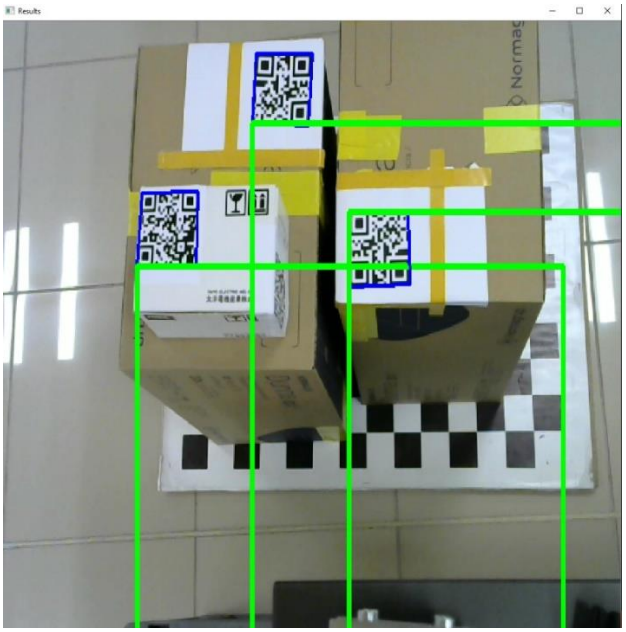


Figura 4.3 primeras pruebas de detección de QR por videocámara

Este problema se solucionó con la inclusión de las extensiones del Pyzbar que permitían comprobar la orientación, gracias a ellas se puede comprobar la orientación del QR en la imagen y adaptar el rectángulo que se usa para la detección de los vértices a esa detección permitiendo unos resultados mucho mejores como se ve en la Figura 4.4.

Figura 4.4 Detección de los QR con la orientación

Aun así, como se puede ver, la detección seguía siendo algo inexacta sobre todo en los vértices más lejanos al QR, para solucionar esto se optó por cambiar la cámara a una con más definición seleccionando la Microsoft Cinema HD que ofrecía una resolución mucho mayor como se puede ver en la Figura 4.5 tomada a través de la interfaz.

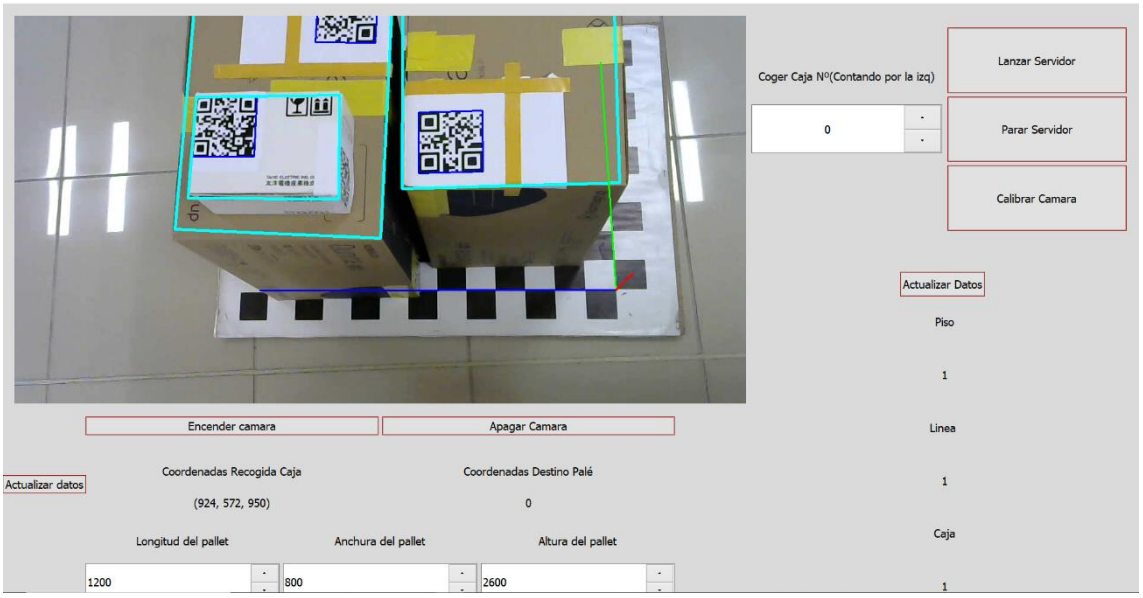


Figura 4.5 Imagen con la segunda cámara

Esta mayor resolución permitía una detección mucho mejor de los bordes del QR lo que, como se ve en la imagen anterior, permitía situar los bordes de la caja con una precisión enorme. Con esto se terminaron las pruebas de la cámara del visor. La cámara se sustituyó por la definitiva (que se explicó anteriormente en este documento) debido a que se prefería utilizar una cámara más adaptada a trabajar como visor que una Webcam.

Otro de los aspectos del visor que recibió varias probaturas fue la de la colocación del QR en las cajas. Como se explicó en la parte de la creación de la codificación QR dos de los datos que se introducen en el QR son la distancia del primer vértice

Debe existir un equilibrio entre dos de los componentes principales para la toma de esta decisión, el primero es la facilidad de detección del QR, en las primeras pruebas se probó a colocar el QR justo en la esquina, pero esto era perjudicial para la detección, por lo tanto, la existencia de un espacio con respecto a la esquina es bueno para el visor. Pero el segundo aspecto a tener en cuenta es el diseño de las cajas, las cajas que se van a paletizar son las mismas que más tarde van a llegar al cliente, por lo tanto, no se puede colocar el QR en medio de la caja, aunque sería lo más óptimo para la detección.

Tras diversas pruebas se decidió que la manera más óptima de colocar el QR será a una distancia de entre 2 y 3 centímetros del borde de la caja.

4.2 PRUEBAS DE FUNCIONAMIENTO DEL ROBOT

Las siguientes pruebas se realizaron para comprobar el funcionamiento óptimo de la programación del robot, para ello se utilizó el complemento de RobotStudio que permite simular el programa.

Con esta simulación se podía comprobar el buen funcionamiento de todos los elementos involucrados en el proceso, el visor detectaría las cajas y se podría comprobar a través de la interfaz que todos los elementos de la programación trabajaban como debían

Para la realización de estas pruebas se debían situar virtualmente los espacios de trabajo caja y pallet, esta situación ficticia podía ayudar luego a la colocación de estos en el mundo real. La colocación de estos se puede ver en la Figura 4.6 que contiene la interfaz del apartado de simulación de Robot Studio.

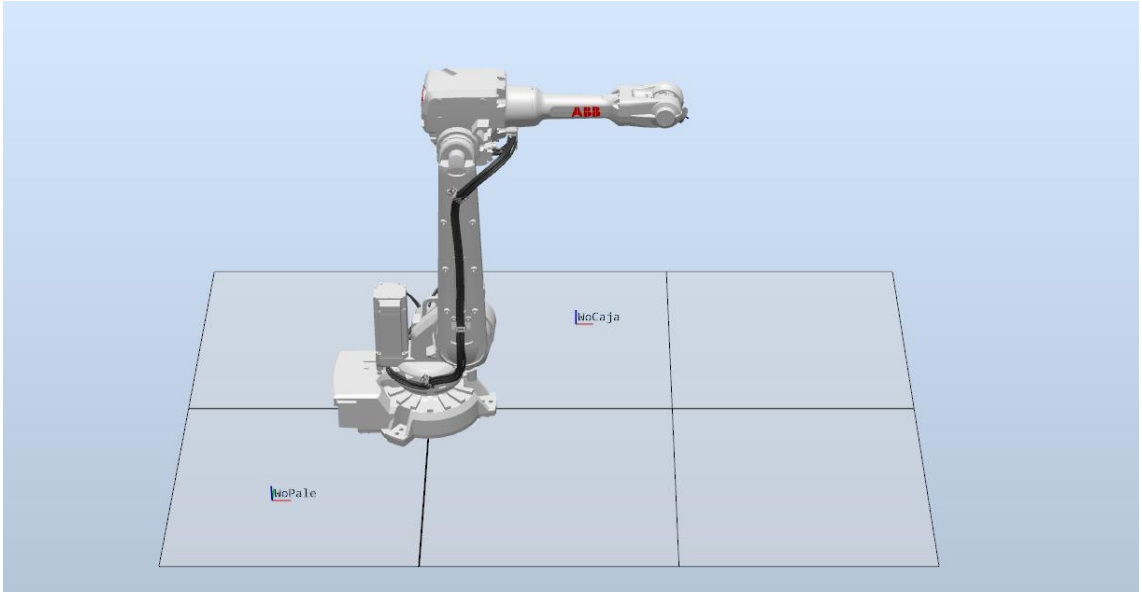


Figura 4.6 Interfaz de la simulación en Robot Studio

Uno de los principales objetivos de esta simulación era el de situar los puntos de referencia del robot. Además de los puntos de recogida y depósito que recibía del visor,

el robot debía tener varios puntos de aproximación a estos, además de uno de reposo y los puntos de giro.

Estos puntos debían situarse con precisión ya que, si no, podría aparecer un error de posición por el cual el robot no pudiera alcanzar la posición deseada.

Tras varias pruebas se situaron los puntos en los datos que se ven en la Figura 4.7 y con ello el robot pudo funcionar sin problemas y se pudo pasar al último apartado de las pruebas.

```
CONST robtarget pos0:=[[0,0,1000],[0,1,0,0],[-1,-2,1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget pos5:=[[100,100,1000],[0,0,1,0],[-1,-2,1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget pos6:=[[100,100,1300],[0,1,0,0],[-1,-2,1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

Figura 4.7 Puntos de referencia del Robot Studio

4.3 PRUEBAS DE FUNCIONAMIENTO REALES

Una vez realizadas todas las pruebas del visor y del robot por separado, se pudo pasar a la realización de pruebas del conjunto de la célula, para ello se habilitó un espacio en la nave 1 de la empresa para la realización de las pruebas pertinentes. La zona de pruebas se puede ver en la Figura 4.8.

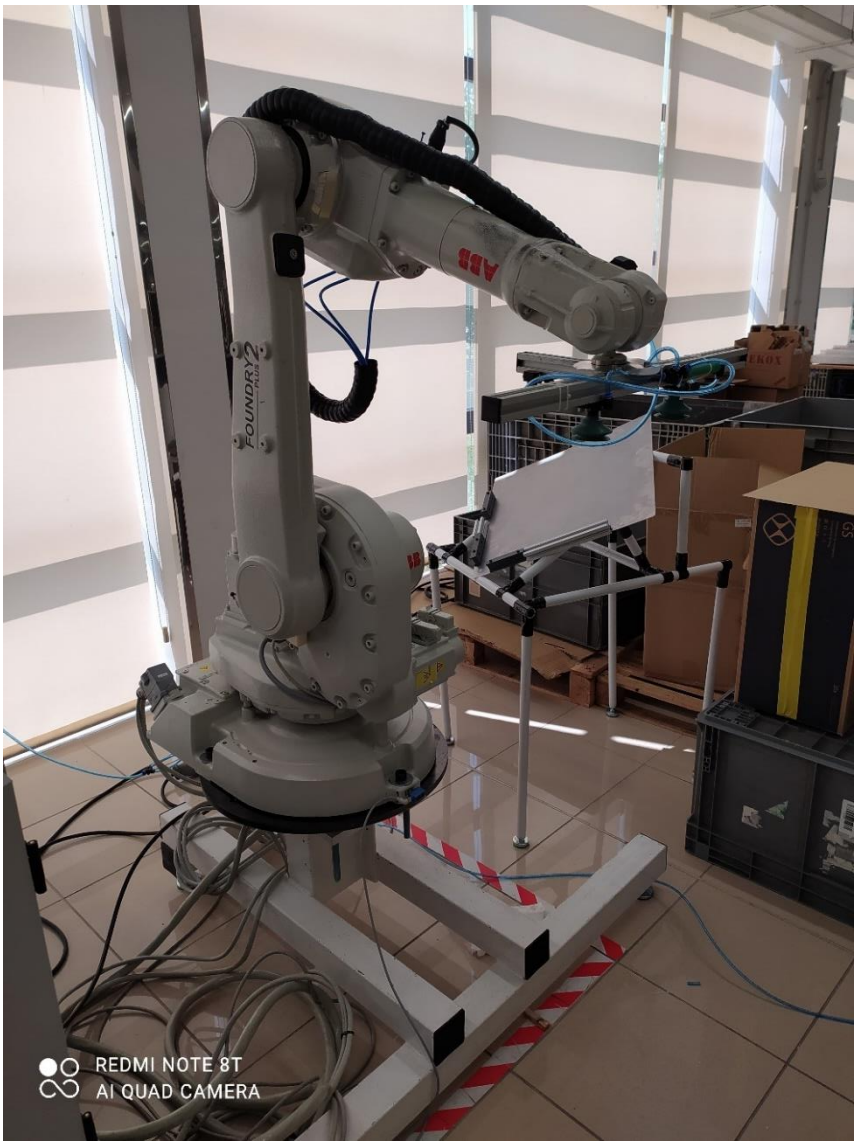


Figura 4.8 Zona de pruebas

Como se ve en la imagen se situaron en la zona los principales elementos que más adelante van a formar parte de la célula y que fueron explicados en el apartado 3 de este documento. Una vez situados todos estos elementos se procedió a comenzar con las pruebas.

Para la realización de estas se debían situar en el plano del trabajo del robot los espacios de trabajo, para simular el espacio de trabajo de la caja se utilizó un contenedor sobre el que se colocó el tablero de calibración como se ve en la Figura 4.9.

Figura 4.9 Contenedor a modo de zona de recogida

Con este tablero se realiza la calibración mediante el módulo explicado anteriormente en el documento. Con esto aparecerán unos ejes en el tablero. Estos ejes se utilizarán como los ejes del espacio de trabajo mediante la herramienta definir del flex pendant, que permite definir los ejes con tres puntos (Figura 4.10).

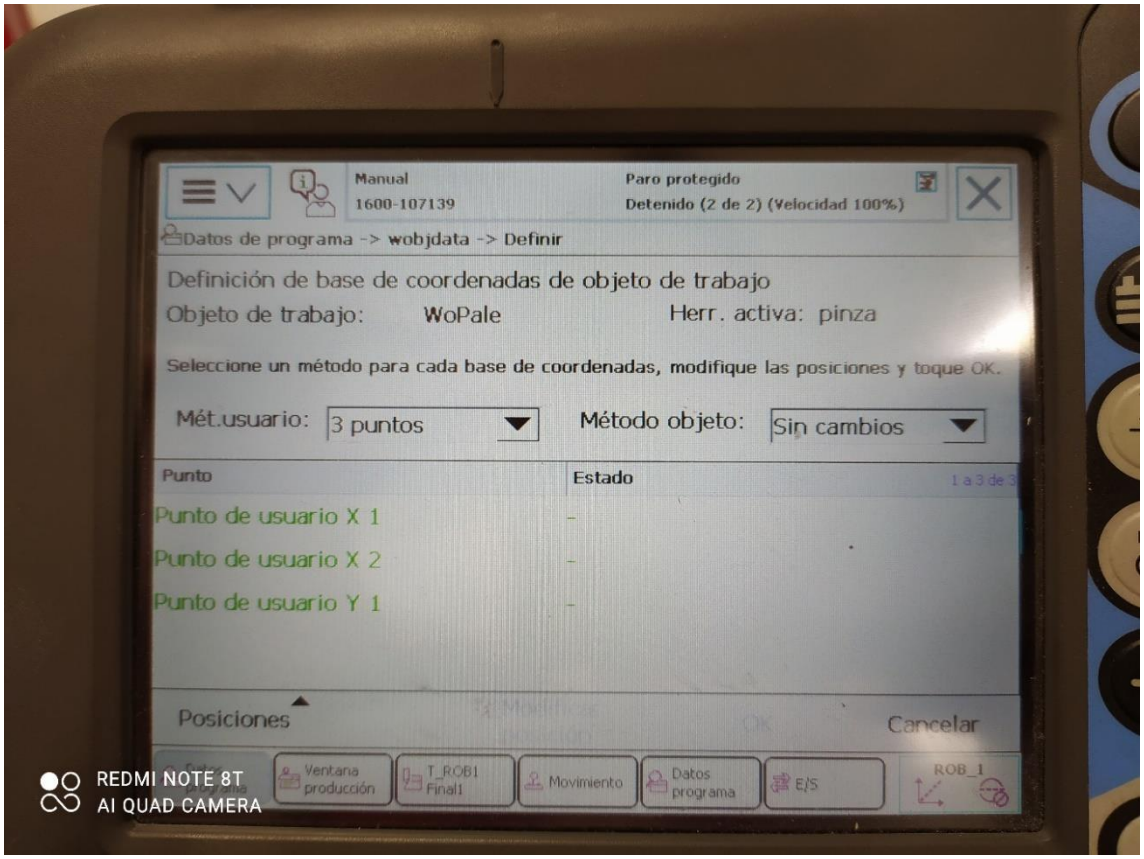


Figura 4.10 Definición de WorkObject por 3 puntos.

El segundo WorkObject corresponde al pallet, para simularlo se utilizó otro contenedor en el cual se depositarían las cajas (Figura 4.11 b). Para que la simulación funcionara correctamente se debía medir este contenedor e introducir estas medidas en la GUI para que sirvan como entrada en el módulo pallet (Figura 4.11 a). Para la colocación de los ejes se utilizaron en este caso los vértices del contenedor como los tres puntos para definir.

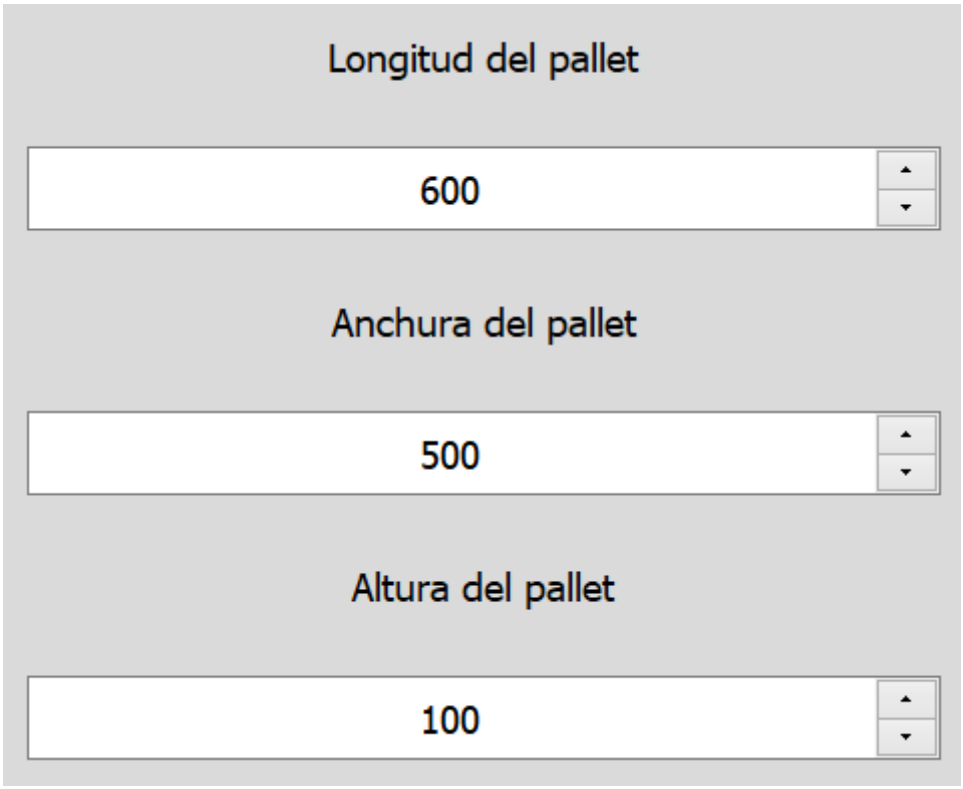


Figura 4.11 Medidas del pallet en la GUI (a) y contenedor a modo de pallet (b).

El último de los elementos fundamentales es la estación de giro, esta fue diseñada por el equipo mecánico de la empresa, el primer diseño de esta estación se ven en la Figura 4.12.

Figura 4.12 Primer prototipo de estación de giro

Tras la realización de las pruebas se consideró que era necesario subir las barreras debido a que se podría dar el caso de que la caja montara por encima de las barreras y esto provocará una inexactitud en el depósito en el pallet por lo que se procedió a dejar la estación de giro como se mostró en el apartado 3 del documento.

Para la colocación de la cámara se utilizó un poste metálico (Figura 4.13), este permitía ver toda la zona de recogida sin interferir con el movimiento del robot, aunque esta

disposición se demostró como algo ineficaz debido a que no permitía detectar los QR con la precisión necesaria y se decidió que a la hora de la verdad la cámara se situaría para realizar una vista cenital de la zona de recogida.

Figura 4.13 Poste para la cámara

Para la colocación de la cámara en vista cenital el equipo mecánico de la empresa diseñó un soporte con elementos plásticos, este soporte mejoró mucho la detección de los elementos y sus dimensiones, el soporte quedó de la forma que se ve en la Figura 4.14.

Figura 4.14 Poste final de la cámara

Con este poste se mejoró mucho la detección de los QR como se ve en la Figura 4.15.

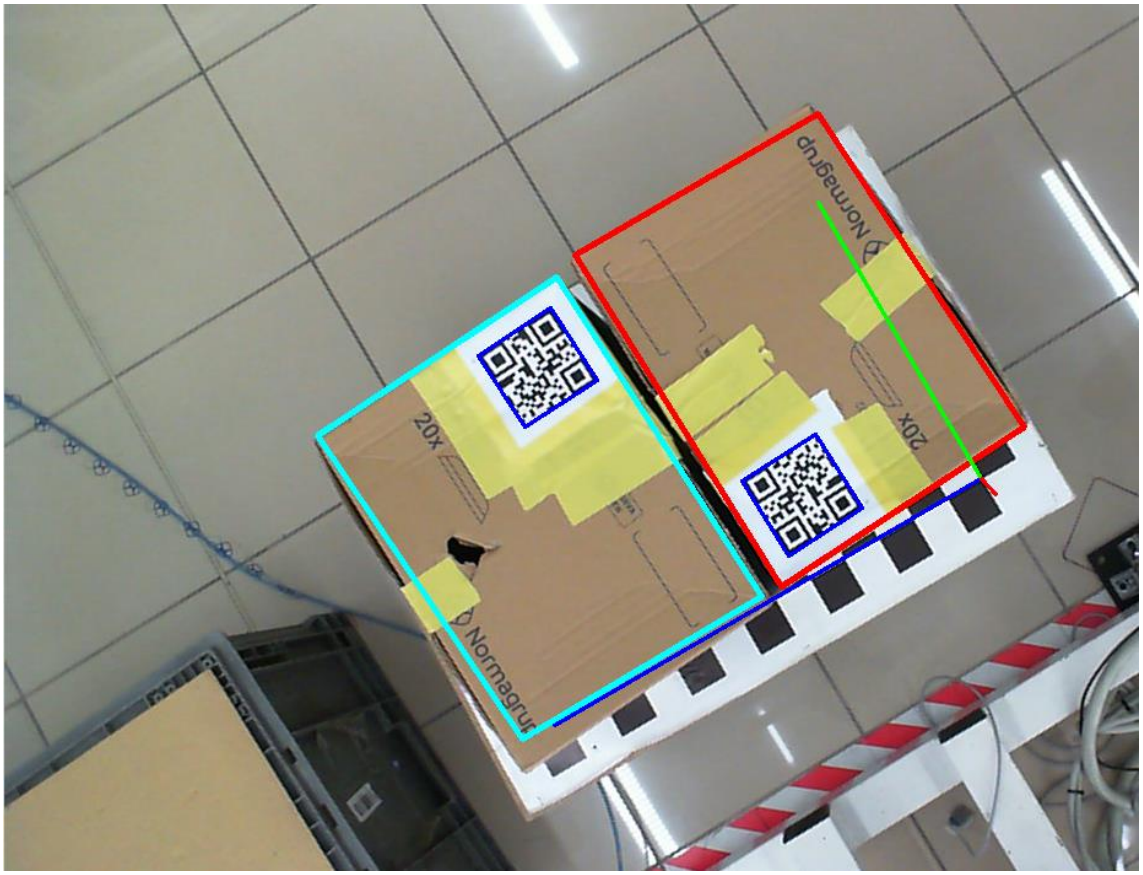


Figura 4.15 Detección con el poste final

Para la realización de las pruebas se diseñó una pinza preliminar con el equipo mecánico de la empresa, esta pinza trabajaba con vacío y permitía una buena sujeción de las cajas, se trabajó con ella durante el periodo de pruebas mientras se esperaba por la pinza definitiva (Figura 4.16).

Figura 4.16 Pinza usada en las pruebas

Por último, se preparó una unidad de control que contenía el controlador, el flex pendant y un portátil que ejercía como el monitor el que se situaría la GUI, la distribución de estos quedaría como se ve en la Figura 4.17.

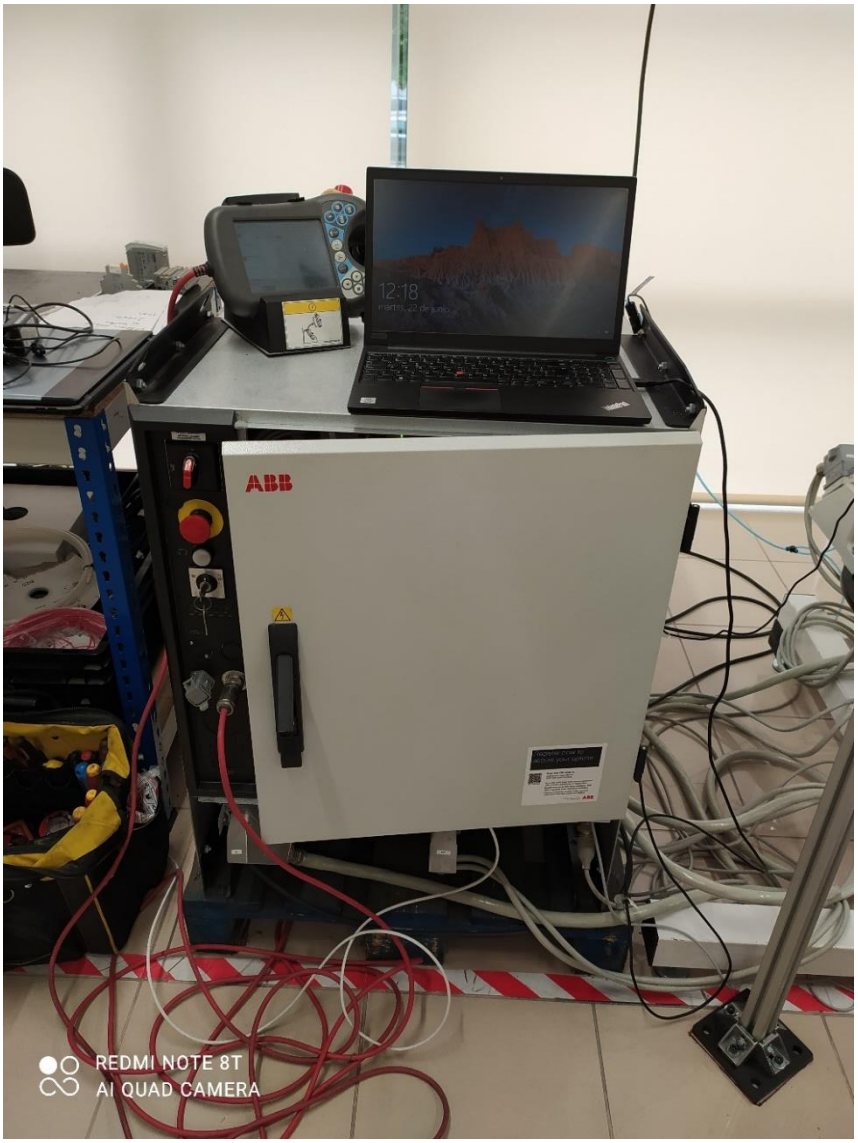


Figura 4.17 Estación de control de la zona de pruebas

Con esto, se tiene un resumen de cuales fueron los elementos que se utilizaron para simular los de la célula, con ello explicado se procederá a explicar cada uno de los experimentos que se realizaron para comprobar el funcionamiento de la célula.

Se comenzó con la prueba más sencilla una caja perfectamente orientada que solo tenía que desplazarse de la zona de recogida a la zona de depósito en la posición correcta. Para el experimento se prepararon 2 cajas del tipo Dunna, una de las cajas que se utilizan en la empresa. Se diseñaron los 3 QR correspondientes y se colocaron en su posición como se ve en la Figura 4.18.

Figura 4.18 Caja con los QR pegados usados para las pruebas

El objetivo de esta prueba era comprobar que se podía crear un pallet con una buena precisión, para ello uno de los aspectos fundamentales era la recogida de la caja, como se puede ver en la Figura 4.19 el robot se dirigía a la posición objetivo de una forma correcta, aunque se debía corregir un poco la inclinación de la pinza

Figura 4.19 Momento de la recogida de la caja en la prueba

Tras esto la siguiente parte importante era la elevación de la caja, esta debía hacerse de una forma que no fuera peligrosa para el contenido de la caja, como se puede ver en la Figura 4.20, se eleva de una forma correcta, aunque se debe solucionar el problema de orientación de la pinza que comparte con la recogida.

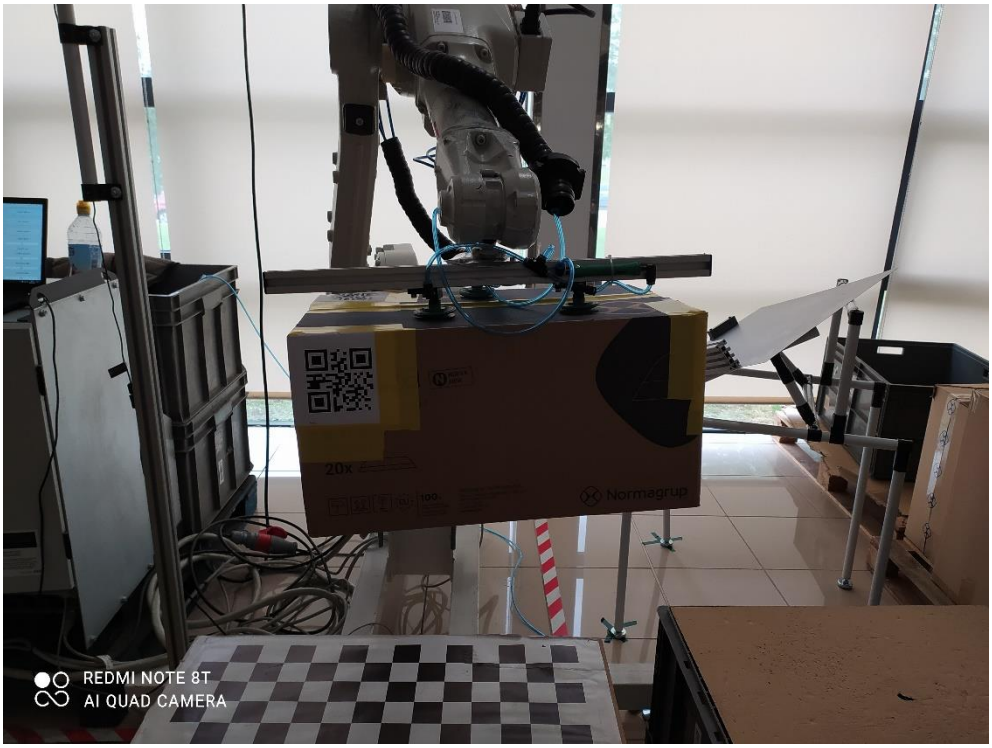


Figura 4.20 Elevación de la caja en la prueba

Por último, se debía comprobar el depósito de las cajas, esta parte se mostró como la más exacta como se puede ver en la Figura 4.21 creando una columna casi perfecta

Figura 4.21 Depósito de las cajas

Con esto se finalizó la prueba con cajas bien orientadas, se pasó entonces a probar con cajas que debían pasar por la zona de giro antes de ser depositadas.

El objetivo de esta prueba era comprobar que se podía girar la caja sin poner en peligro su contenido y manteniendo una buena precisión. Al igual que en el caso de varios módulos anteriormente explicados se tienen diferentes casos dependiendo de la orientación y la posición de la caja, pero solo se explicará uno al ser muy similares entre ellos.

El primer paso al igual que en el caso “normal” es sostener la caja, esto se hace con buena precisión como se ve en la Figura 4.22.

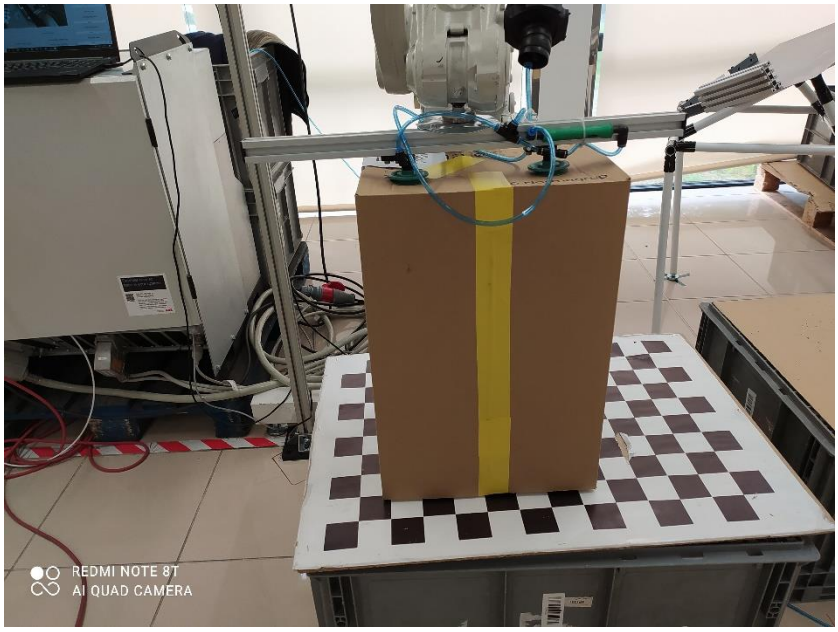


Figura 4.22 Sujeción de la caja

Una vez elevada la caja, esta se debe orientar de forma correcta para realizar su giro como se ve en la Figura 4.23.

Figura 4.23 Orientación de la caja

Una vez orientada la caja, esta se dirige a la mesa de giro, la aproximación a esta se debe hacer a velocidad reducida. La aproximación se puede ver en la Figura 4.24.

Figura 4.24 Llegada a la mesa de giro

Una vez bien orientado en la mesa de giro, se suelta la caja, la pendiente de esta mesa, como se ha explicado anteriormente, hará que se sitúe siempre en el mismo punto como se ve en la Figura 4.25.

Figura 4.25 Caja en la mesa de giro

Una vez situada la caja, el robot se orienta en la parte superior de esta y baja a cogerla al igual que se hacía para recogerla la primera vez (Figura 4.26).

Figura 4.26 Recogida de la caja en la mesa de giro

Una vez recogida la caja se saca de la mesa de giro y se sitúa encima de la zona de depósito como se ve en la Figura 4.27.

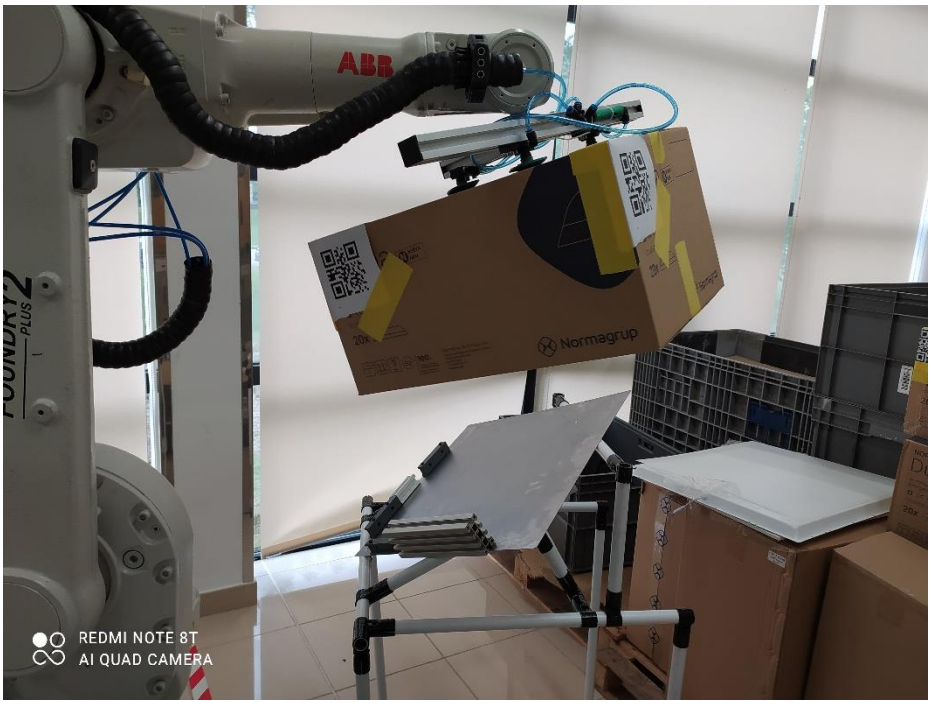


Figura 4.27 Caja recogida de la zona de giro

Por último, la caja se deposita en la zona de depósito como se ve en la Figura 4.28.

Figura 4.28 Momento del depósito de la caja

Al igual que en caso anterior se probó con varias cajas para formar una columna y los resultados se ven en la Figura 4.29.

Figura 4.29 Columna formada

Como resumen de los ensayos, se probaron las diferentes funciones de la célula, lo que llevo a descubrir algunos errores que existían en los módulos o algunas funciones que se debían incorporar, pero una vez solucionados se comprobó que la célula cumplía con lo pedido con una buena precisión y la célula se podía instalar.

5. Puesta en operación y manejo.

Este aparado del documento va a servir como un "manual de instrucciones" destinado a los operarios que van a utilizar el robot, detallando todos los pasos para ponerlo en funcionamiento y conseguir que no se produzcan errores durante este.

5.1 PREPARACIÓN INICIAL

El primer paso para poner el sistema en funcionamiento será encender la unidad de control y el controlador IRC del robot. Una vez encendidos ambos se debe lanzar el programa GUI (llamado GUI.py en el ordenador) en la unidad de control. Se mostrará la siguiente pantalla (Figura 5.1).

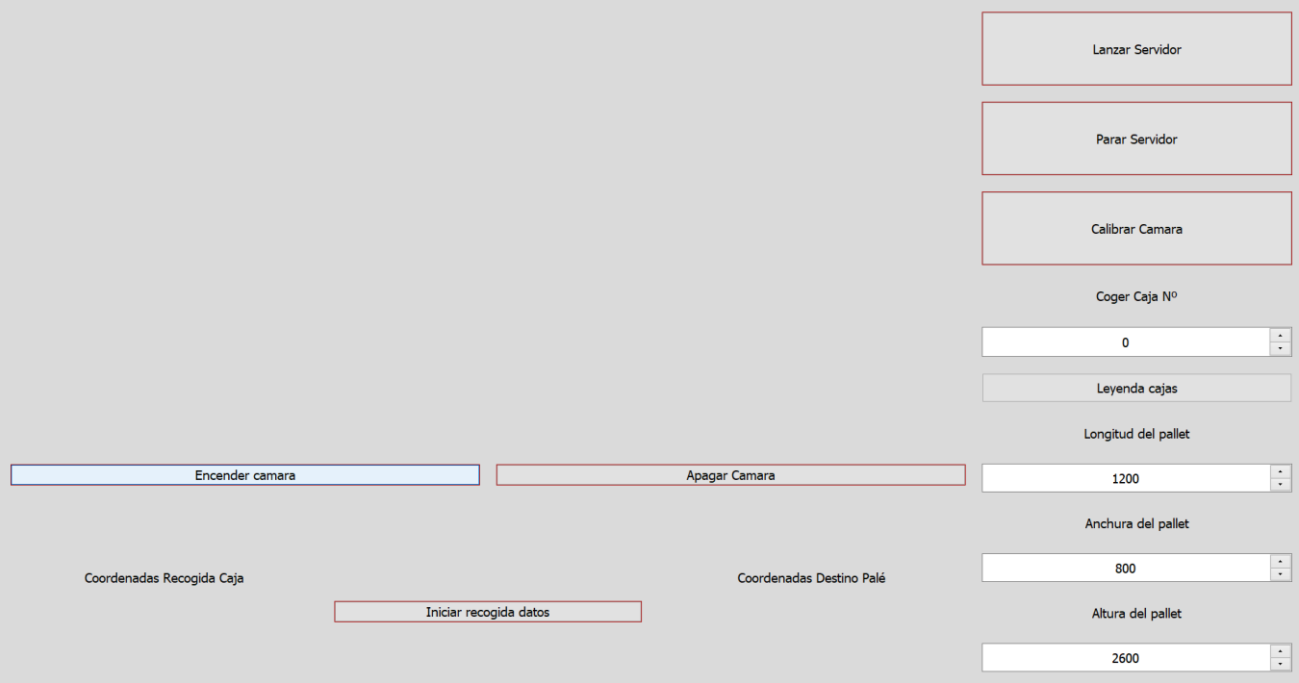


Figura 5.1 Pantalla GUI

El siguiente paso será realizar la calibración, este paso sólo se debe realizar la primera vez que se inicie el funcionamiento del robot, y sólo se debe repetir en el caso de que se detecte que por alguna razón se haya movido la cámara. Para realizar la calibración se utiliza el botón calibrar de la GUI resaltado en la Figura 5.2.

Figura 5.2 Botón para calibrar la cámara

El siguiente paso es la asignación de los WorkObjects con el robot, este paso, al igual que el anterior, sólo se debe realizar la primera vez o en el caso de que alguno de los elementos cuya posición está fijada por un WorkObject se haya movido (zona de recogida, zona de giro y zona de depósito). La asignación de los WorkObjects se hará de la forma explicada en el capítulo anterior del documento.

Con esto se finalizan las preparaciones iniciales para poner el sistema en funcionamiento.

5.2 INICIO DEL SISTEMA

Este es el proceso que se realizará normalmente para iniciar el proceso de trabajo con el robot en las tareas de paletización, salvo que se dé uno de los casos anteriormente explicados que requieran la calibración de la cámara o la redefinición de los WorkObjects . Para realizar este inicio, el primer paso será encender la cámara con el botón correspondiente que se ve en la Figura 5.3.

Figura 5.3 Botón para iniciar la cámara.

Una vez iniciada la cámara, el siguiente paso es iniciar la recogida de datos en tiempo real con el botón que lleva el mismo nombre (esta parte no es necesaria para que el proyecto funcione, pero es buena para ver cómo está funcionando y detectar posibles errores). El botón a pulsar se ve en la Figura 5.4.

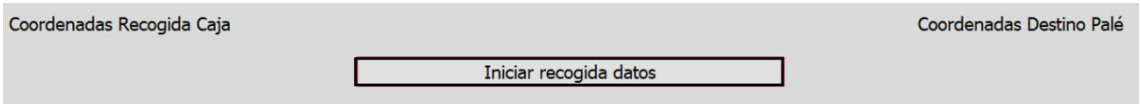
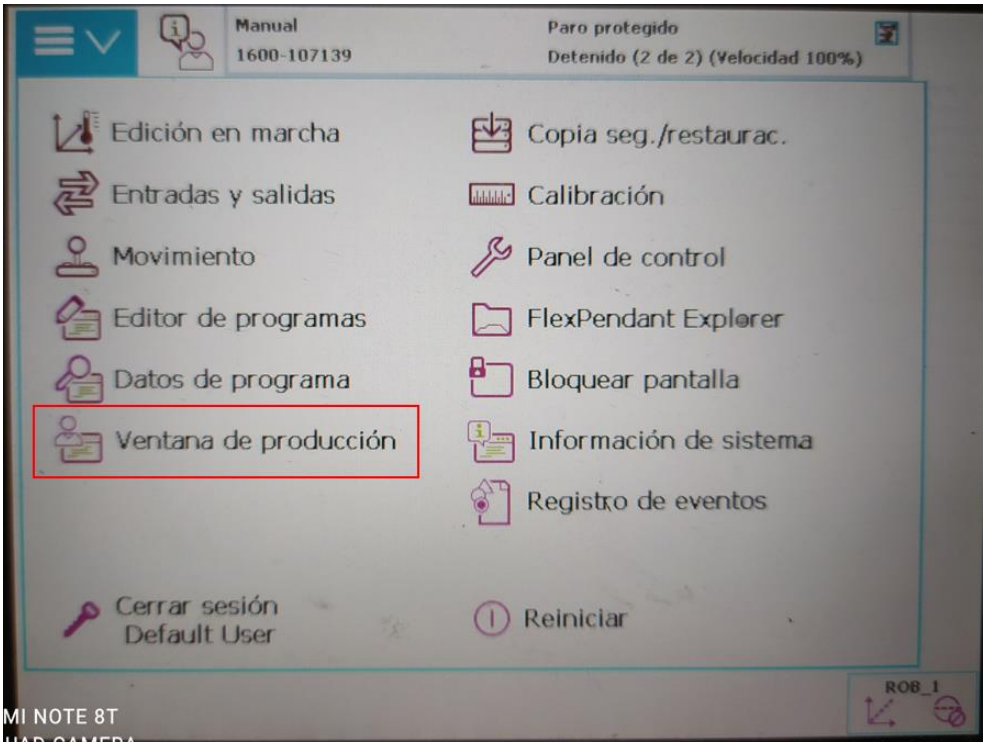


Figura 5.4 Botón para iniciar la recogida de datos.

Una vez hecho todo esto, se procede a lanzar el servidor con el botón que lleva el mismo nombre, con esto se acabaría la manipulación de la GUI y solo se usaría para comprobar el funcionamiento de la célula. El botón de lanzar servidor se ve en la Figura 5.5.

Figura 5.5 Botón para lanzar el servidor.

El siguiente paso es el inicio en el robot, que engloba dos partes fundamentales: la primera de ellas será la ventana de producción, a la que se accede mediante el *flex pendant* del robot (Figura 5.6 a) y con ella se puede colocar el puntero de programa en el inicio de éste mediante la orden *PP a main* (Figura 5.6 b).



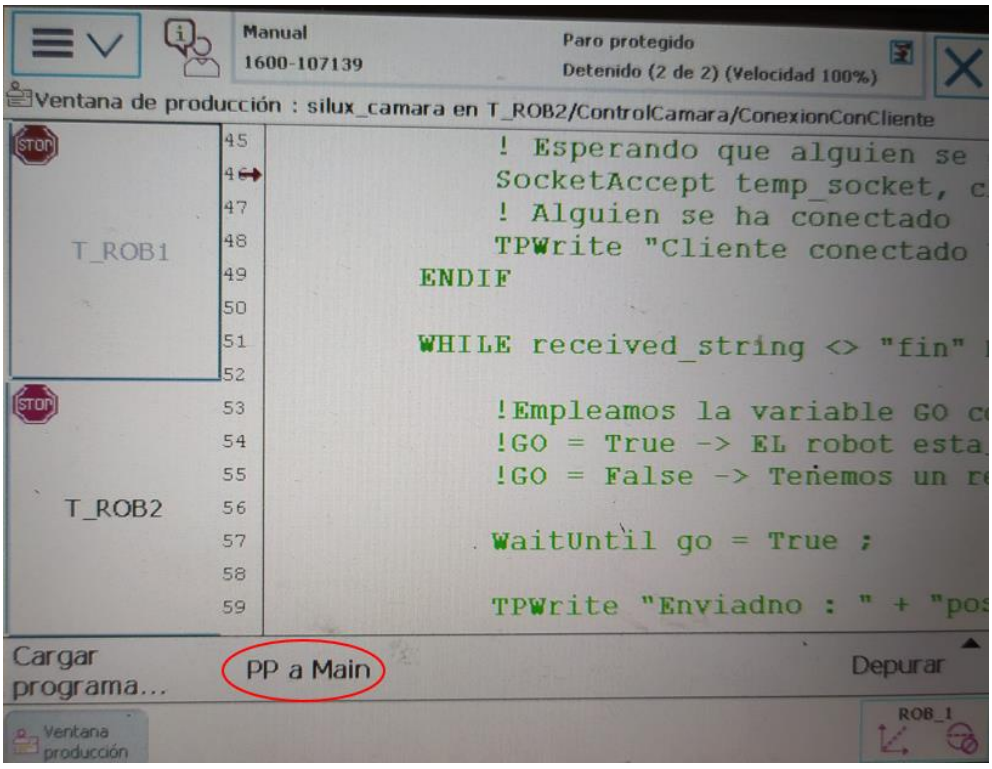


Figura 5.6 Situación de la ventana de producción (a) y de la orden PP a main en ella (b)

Una vez en la ventana de producción, se debe activar el robot pulsando el botón trasero del *flex pendant* y pulsar el botón *play* para comenzar la ejecución, una vez comenzada solo se deberá dejar pulsado el botón trasero y la ejecución seguirá su curso (Figura 5.7).

Figura 5.7 Botón play y activación del botón trasero para iniciar el robot.

5.3 SISTEMA INICIADO

Por último, una vez que el sistema se ha iniciado, quedan por hacer un par de distinciones. La primera de ellas es la existencia del botón para parar el sistema el cual sólo se debe pulsar en el caso de que se detecte un error grave en el funcionamiento del sistema. Su situación en la GUI se ve en la Figura 5.8.

Figura 5.8 Situación del botón para parar en la GUI

Lo último a destacar es que se debe estar atento a la GUI, ya que, al finalizar cada pallet, sacará un aviso por pantalla avisando para que sea retirado (Figura 5.9) y se debe volver a lanzar el servidor tras cambiar el pallet.

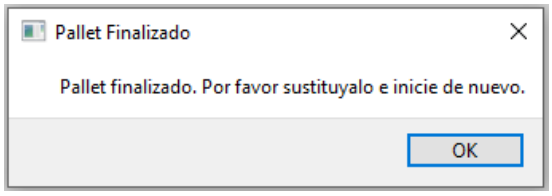


Figura 5.9 Aviso de que se ha terminado el Pallet.

Con esto, se finaliza este apartado que sirve como un “manual de instrucciones” para los operarios.

6. Conclusiones.

Durante el tiempo de desarrollo de este proyecto se ha realizado un gran esfuerzo estudiando y diseñando diferentes opciones para poder cumplir los objetivos que se habían marcado para la célula de producción. Se cree que se ha alcanzado el objetivo de una forma satisfactoria ya que permite cumplir con lo que se deseaba.

Se ha conseguido programar un sistema de visión artificial que trabaja sin apenas errores y en tiempo real, cosa que inicialmente parecía un objetivo demasiado ambicioso y que al final se tendría que desestimar.

Se ha conseguido que el robot se mueva de forma fluida, las cajas se sostengan sin apenas movimiento y se depositen con precisión. Además, se ha conseguido poder trabajar con cajas que estén colocadas de cualquier forma gracias a la zona de giro.

Se ha diseñado una interfaz da fácil uso y que se adapta automáticamente a cualquier resolución y tamaño de pantalla.

Por último, se ha conseguido interrelacionar todos estos elementos gracias a la conexión Socket de forma que trabajen en conjunto y se envíen información en tiempo real y sin fallos.

Pero si hay un elemento a destacar especialmente en el proyecto es el módulo *pallet*. Muchos de los elementos de programación de este proyecto se pudieron aprovechar de otros proyectos existentes ya sea en la empresa o en la red con un trabajo de adaptación para la tarea encomendada. Pero en el caso del módulo *pallet* no se encontró nada en la red ni en la empresa y se diseñó completamente de cero teniendo que idear todo el sistema de variables que utiliza. Por lo tanto, se trata de la mayor contribución al proyecto.

Como aspectos de mejora en el futuro, se cree que se pueden añadir nuevas funcionalidades a la célula: Aunque se cree que ésta se puede adaptar a casi cualquier situación que se dé, en el futuro se podría necesitar mayor velocidad de paletizado y por ello se podría introducir un segundo robot trabajando en paralelo.

Otra posible mejora es la introducción de filtros en el sistema de visión que permitan mejorar la capción de los códigos. Hablando de éstos, en el futuro podrían aparecer nuevos sistemas de codificación más eficientes y podría ser necesario realizar cambios para adaptarse a los mismos.

En definitiva, se considera que el trabajo realizado es muy valioso y se cree que la célula podrá trabajar sin problemas durante los próximos años. Además, el sistema empleado puede ser adaptado fácilmente para mejorar la célula en el futuro.

7. Referencias

A.Diaz, 2020. *Statista.* [En línea]
Available at: https://es.statista.com/estadisticas/601564/principales-empresas-del-sector-de-robotica-industrial-por-ingresos/
[Último acceso: 29 Junio 2021].

ABB, 2018. *ABB Robotics.* [En línea]
Available at: https://search.abb.com/library/Download.aspx?DocumentID=PR10282EN_R8&LanguageCode=en&DocumentPartId=&Action=Launch
[Último acceso: 2 Julio 2021].

ABB, 2021. *ABB Robotics.* [En línea]
Available at: https://new.abb.com/products/robotics/es/robots-industriales/irb-1600
[Último acceso: 14 Junio 2021].

ABB, 2021. *ABB Robotics.* [En línea]
Available at: https://new.abb.com/central-america-caribbean/sobre-nosotros/tecnologia/robots-industriales
[Último acceso: 30 Junio 2021].

Contaval, 2016. *Detection Blog.* [En línea]
Available at: https://www.contaval.es/que-es-la-vision-artificial-y-para-que-sirve/
[Último acceso: 29 Junio 2021].

Curiosfera, 2020. *Curiosfera.* [En línea]
Available at: https://curiosfera-historia.com/quien-invento-el-codigo-de-barras-historia/
[Último acceso: 30 Junio 2021].

EDS, 2021. *EDS Robotics.* [En línea]
Available at: https://www.edsrobotics.com/blog/evolucion-robotica-industrial/
[Último acceso: 13 Junio 2021].

Hudson, L., 2020. *Python Proyects.* [En línea]
Available at: https://pypi.org/project/pyzbar/#description
[Último acceso: 16 Junio 2021].

INFAIMON, 2020. *INFAIMON.* [En línea]
Available at: https://blog.infaimon.com/historia-vision-artificial/
[Último acceso: 30 Junio 2021].

Instituto Nacional de Estadistica, 2021. *Instituto Nacional de Estadistica.* [En línea]
Available at: https://www.ine.es/jaxiT3/Datos.htm?t=6062
[Último acceso: 1 Julio 2021].

Languages, O., 2021. *Diccionario.* [En línea]
Available at: https://languages.oup.com/google-dictionary-es/
[Último acceso: 30 Junio 2021].

Marin, R., 2020. *Revista digital Inesem.* [En línea]
Available at: https://revistadigital.inesem.es/informatica-y-tics/opencv/
[Último acceso: 16 Junio 2021].

Risoul, 2016. *Risoul.* [En línea]
Available at: https://www.risoul.com.mx/blog/sistema-de-inspeccion-industrial-geva-y-camaras-genie-de-teledyne-dalsa
[Último acceso: 12 7 2021].

Robotics, A., 2019. *Manual del operador del ABB IRC5 FlexPendant* , s.l.: s.n.

Robots, R. d., 2021. *Revista de Robots.* [En línea]
Available at: https://revistaderobots.com/robots-y-robotica/robots-industriales-y-robotica-industrial/
[Último acceso: 13 Junio 2021].

Shun, R., s.f. *Sistemas de Visión Artificial, historia, componentes principales y procesamiento digital de imágenes..* [En línea]
Available at: https://es.scribd.com/doc/259914658/Sistemas-de-Vision-Artificial-Historia-Componentes-y-Procesamiento-de-Imagenes
[Último acceso: 30 Junio 2021].

SICK, 2020. *SICK.* [En línea]
Available at: https://www.sick.com/es/es/fotocelulas/fotocelulas/g6/gtb6-p4211/p/p246949
[Último acceso: 2 Julio 2021].

Unidad de Informática, s.f. *Escuela Nacional de Ciencias Bilógicas.* [En línea]
Available at: https://www.encb.ipn.mx/alumnos/udi/uditips/uditips08.pdf
[Último acceso: 30 Junio 2021].

Wikipedia, 2021. *Wikipedia.* [En línea]
Available at: https://es.wikipedia.org/wiki/ABB
[Último acceso: 30 Junio 2021].

DOCUMENTO 2:

PLANIFICACIÓN Y PRESUPUESTO

Manuel González Río

INDICE

INDICE DE FIGURAS

1. Planificación

Las prácticas se dividieron principalmente en dos partes, las prácticas curriculares que comenzaron el 5 de mayo de 2021 y se extendieron hasta el 17 de mayo de 2021 y las extracurriculares que se extendieron desde el 17 de mayo de 2021 al 31 de agosto de 2021.

Para la planificación se tuvieron en cuenta las dos prácticas como si fueran una sola independientemente de qué tipo fueran. Cabe destacar que a redacción de este documento no se pudieron cumplir con las fechas planificadas para las pruebas en el entorno real debido a retrasos en la reforma de los almacenes.

Las diferentes tareas se programaron de la forma que se ve en la Figura 1.1.

Actividad	Fecha de Inicio	Duración (Horas)	Fecha de fin
Repaso conceptos básicos Python	05/05/2021	16	07/05/2021
Aprendizaje con Pyzbar y OpenCV	07/05/2021	40	12/05/2021
Desarrollo del módulo QR	12/05/2021	160	01/06/2021
Programación del robot	19/05/2021	160	08/06/2021
Desarrollo del módulo de calibración	26/05/2021	56	02/06/2021
Desarrollo del módulo Socket	02/06/2021	56	09/06/2021
Desarrollo del módulo pallet	02/06/2021	120	17/06/2021
Realización de pruebas	15/06/2021	160	05/07/2021
Realización de pruebas en el entorno real	05/07/2021	240	04/08/2021

Figura 1.1 Fechas de inicio y fin de las tareas.

Como se ve en la tabla las tareas principales fueron los desarrollos de los módulos principales y el resto de pequeñas tareas se fueron solapando con ellas. Una vez que se terminaron con los módulos se comenzaron con las pruebas y se centró todo el esfuerzo en ellas.

Con esta tabla se diseña el diagrama de Gantt correspondiente (Figura 1.2):

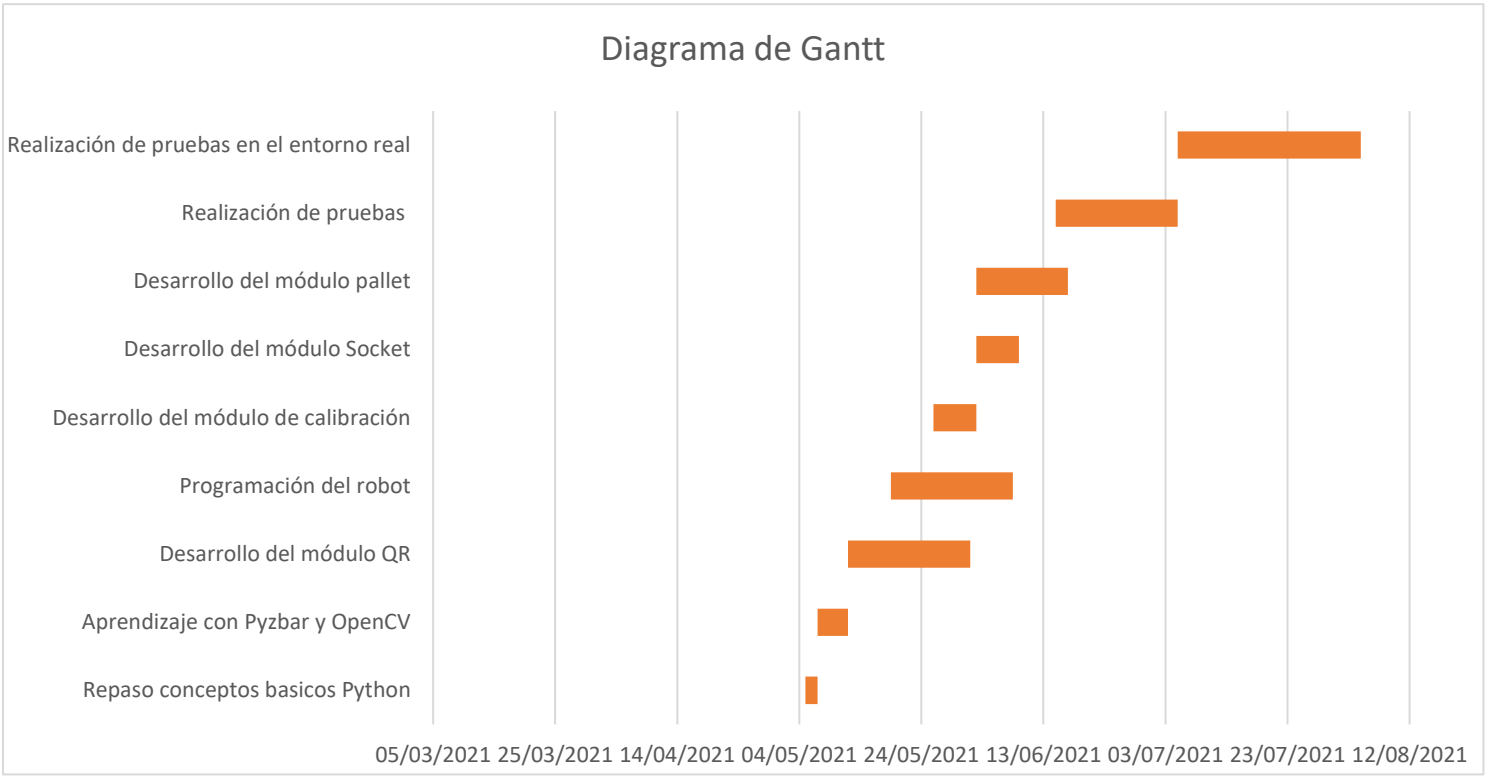


Figura 1.2 Diagrama de Gantt del proyecto

En este diagrama se plasma la planificación del proyecto y se pasará ahora a hablar del presupuesto del mismo.

2. Presupuesto

Para el cálculo del presupuesto del proyecto se tendrán en cuenta los diferentes costes que éste conlleva. El primero de ellos corresponde al coste de ingeniería, estimado en 21,88 €/h en Asturias. (Instituto Nacional de Estadistica, 2021).

Para calcular el coste total de ingeniería se deben calcular las horas trabajadas. La jornada laboral era de 8 h/día , por lo tanto, se va a indicar el número de días trabajados por mes y a partir de ello calcular el total (Figura 2.1).

Abril	20 días
Mayo	21 días
Junio	23 días
Julio	22 días
Agosto	22 días
Total	108 días

Figura 2.1 Días trabajados cada mes

Una vez que se tiene los días totales trabajados se multiplican por 8 para sacar las horas y por el precio de la hora de ingeniería para obtener el coste total en ingeniería (Figura 2.2).

Horas trabajadas	864
Coste de un ingeniero	21,88 €
Total	**18.904,32 €**

Figura 2.2 Cálculo del coste total en ingeniería

A este coste de ingeniería se le debe añadir el coste derivado de las personas que han ayudado en este proyecto, ya sea supervisándolo o realizando tareas mecánicas. El coste

de supervisión se supone en 50 €/h, al ser el tutor en la empresa el jefe de ingenieros de la misma, se asigna una hora diaria de supervisión. En el caso mecánico se supone un coste de 20 €/h y se le asignan 80 horas de trabajo al proyecto. Con estos se calculan los totales que se ven en la Figura 2.3.

Horas supervisión	108
Horas mecánico	80
Coste supervisor	50,00 €
Coste mecánico	20,00 €
Total supervisión	5.400,00 €
Total mecánico	1.600,00 €
Total	**25.904,32 €**

Figura 2.3 Costes de personal

El siguiente aspecto a tener en cuenta es el de los costes de los equipos empleados, estos se adquirieron para el proyecto y el total se ven en la Figura 2.4:

Robot	23.000,00 €
Pinza	3.700,00 €
Cámara 1	60,00 €
Cámara 2	90,00 €
Cámara 3	1.500,00 €
Sensor	100,00 €
Mesa de giro	500,00 €
Poste para la cámara	350,00 €
Portátil	316,67 €
Gastos para pruebas	30,00 €
Pantalla	125,00 €
Total	**29.771,67 €**

Figura 2.4 Costes de equipos empleados

Cabe destacar que, los precios que aparecen en esta figura están sacados de la página web del fabricante, o en su defecto de páginas web especializadas.

Los costes de la mesa de giro y del poste para la cámara han sido estimados por el equipo mecánico de la empresa en base al coste de los materiales utilizados. Lo mismo se hace en el caso de los gastos para las pruebas.

Por último, el coste del portátil corresponde al coste total del portátil (950 euros) escalado a las horas que se ha utilizado en el proyecto con respecto a las horas totales de uso.

Con esto se puede proceder a calcular el coste total del proyecto, primero de todo, se suma el coste de la mano de obra al coste de los equipos. Con esto se halla el primer total que es de 55.675,99 euros.

A estos costes se le han de añadir dos más. El primero de ellos son los gastos generales que corresponden al 6% del total anteriormente calculado. El segundo de ellos es el beneficio industrial que corresponde al 10% del total. Una vez obtenidos ambos se suman al total anterior obteniendo un presupuesto total de 64.584,14 euros.

Por último, a estos costes se le añade el IVA del 21% dando resultado al total final de los costes del proyecto, como resumen de estos costes se tiene la Figura 2.5.

Mano de obra	25.904,32 €
Coste de equipos	29.771,67 €
Total	**55.675,99 €**
Gastos generales (6%)	3.340,56 €
Beneficio Industrial (10%)	5.567,60 €
Presupuesto Total	**64.584,14 €**
IVA (21%)	13.562,67 €
Total Final	**78.146,81 €**

Figura 2.5 Tabla resumen de los costes

Por lo tanto, el presupuesto final de este proyecto es de 78.146,81 euros.