



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Ingeniería de Sistemas y Automática

Manipulación de objetos en aplicaciones de logística
industrial

Trabajo Fin de Máster

Máster Universitario en Automática e Informática Industrial

AUTOR/A: Medina Gamero, Sergio

Tutor/a: Blanes Noguera, Juan Francisco

CURSO ACADÉMICO: 2022/2023

Agradecimientos

Me gustaría dar las gracias, en primer lugar, a Juan Francisco Blanes Noguera por darme la oportunidad de realizar este proyecto tan interesante, también, a los trabajadores del AI2 por la ayuda que me dedicaron en todo momento y, por último, a mi familia y amigos por la energía y la fuerza que me han transmitido para no rendirme en ningún momento y ayudarme en todo lo posible.

Resumen

Una de las tareas más utilizadas en cualquier industria es el *pick y place* de objetos. Este proceso tan repetitivo y pesado para un operario cambió con la aparición de la automatización y la robótica en las fábricas, pasando a ser realizado por robots. Actualmente, la llegada de los sistemas de visión a las industrias ha permitido que este proceso sea más eficaz, pudiendo coger objetos diversos y en posiciones diferentes del área de trabajo, y, además, permitiendo realizar una selección o detección de fallos a la vez. El objeto de este trabajo será utilizar esta nueva técnica para realizar el *pick y place* de cajas de diferentes dimensiones y colores. Estas cajas estarán colocadas en diferentes posiciones y orientaciones en el área de trabajo. También, se deberá conocer el tipo de caja que se va a coger en cada momento, para ello, se utilizará una base de datos que tenga las dimensiones de cada una de ellas. De esta manera, mediante el uso de un robot colaborativo *UR5* y una cámara de visión *Intel RealSense D435*, conseguir un código en el lenguaje de programación *Python* que permite realizar esas tareas de manera robusta y, así, poderlo implementar en la industria en un futuro.

Abstract

One of the most used tasks in any industry is the pick and place of objects. This repetitive and weary process for an operator changed with the appearance of automation and robotics in factories, becoming carried out by robots. Currently, the arrival of vision systems in industries has allowed this process to be more efficient, being able to pick up various objects and in different positions of the work area, and, allowing selection or fault detection at the same time. The aim of this project will be to use this new technic to carry out the pick and place of boxes of different dimensions and colours. These boxes will be placed in different positions and orientations in the work area. Also, the type of box that is going to be picked up at each moment must be known, for this reason, a database will be used that has the dimensions of each one of them. In this way, by using a collaborative robot *UR5* and a vision camera *Intel RealSense D435*, we will get a code in the programming language *Python* that allows to perform these tasks in a robust way and, thus, implement it in the industry in the future.

Índice general

1. Introducción	1
1.1. Objetivos, motivación y antecedentes	2
1.1.1. Objetivos	2
1.1.2. Motivación	2
1.1.3. Antecedentes	2
1.2. Estado del arte	3
1.2.1. Historia de la Visión por Computador	3
1.2.2. Usos actuales de la Visión por Computador	5
1.2.3. Pick and Place con Visión por Computador	6
1.3. Estructura de la memoria	7
2. Dispositivos hardware utilizados	9
2.1. Robot de <i>Universal Robot UR5</i>	10
2.2. Cámara de visión con profundidad <i>Intel RealSense D435</i>	10
3. Explicación y análisis del software utilizado	13
3.1. Objetos	14
3.1.1. Objeto <i>ur_hadle</i>	14
3.1.2. Objeto <i>Database</i>	14
3.2. Código principal	15
3.3. Funciones creadas	17
3.3.1. Posición inicial del robot	17
3.3.2. Inicialización de la cámara y obtención de sus datos	17
3.3.3. Segmentación de la imagen	17
3.3.4. Comprobar si la caja se encuentra en los bordes de la imagen y, si es así, ajustar la posición de la cámara para visualizarla entera	18
3.3.5. Encontrar el ángulo con el que está desplazada la caja respecto la cámara	18
3.3.6. Posición x, y, z del centroide	18
3.3.7. Acercamiento de la cámara al objeto	18
3.3.8. Conocer el tipo de caja	19
3.3.9. Transformaciones de vector de rotación a Roll-Yaw-Pitch y viceversa para conseguir un giro de la última articulación solo en Yaw	19

3.3.10. Giro de la última articulación del robot en Z	19
3.3.11. Pick de la caja	19
3.3.12. Place de la caja	19
4. Resultados experimentales del proyecto	21
4.1. Problemas acaecidos durante las pruebas	22
4.2. Imágenes y comentarios de los resultados de la prueba	23
4.2.1. Imágenes de la cámara obtenidas en la posición inicial	23
4.2.2. Segmentación de las imágenes	24
4.2.3. Objetos en bordes de la imagen	27
4.2.4. Ángulo de la caja respecto a la cámara	30
4.2.5. Acercamiento, centrado y reconocimiento del tipo de caja	31
4.2.6. Pick y Place de las cajas	33
5. Conclusiones y trabajos futuros	37
5.1. Conclusiones	38
5.2. Trabajos futuros	38
A. Hojas de características	39
A.1. Robot de <i>Universal Robot UR5</i>	40
A.2. Cámara de visión con profundidad <i>Intel RealSense D435</i>	42
B. Código completo del software	45
B.1. Objetos	46
B.1.1. Objeto <i>ur_hadle</i>	46
B.1.2. Objeto <i>Database</i>	46
B.2. Código principal	47
B.2.1. Librerías y objetos	47
B.2.2. Funciones creadas	47
B.2.3. Main	58
C. Código adicional	61
C.1. Objeto creado para trabajar con la Base de Datos	62
Bibliografía	65

Índice de tablas

3.1. Base de datos Cajas	14
4.1. Tabla comparativa detección tipos de cajas	32

Índice de figuras

1.1. Organización jerárquica de la IA	4
1.2. Cámara Kinect	5
1.3. Aplicaciones de la Visión por Computador	6
2.1. Vistas frontal y perfil del área de trabajo del robot y la cámara	11
3.1. Árbol de decisión del programa	16
4.1. Imágenes de las 25 posiciones iniciales	23
4.2. 25 transformaciones HSV	24
4.3. 25 imágenes de las máscaras	25
4.4. 25 imágenes de los centroides y los contornos	26
4.5. Imágenes de la colocación de la caja 2 en los límites de la cámara	27
4.6. Imágenes de la colocación de la caja 4 en los límites de la cámara	27
4.7. Imágenes de la colocación de la caja 7 en los límites de la cámara	28
4.8. Imágenes de la colocación de la caja 12 en los límites de la cámara	28
4.9. Imágenes de la colocación de la caja 14 en los límites de la cámara	28
4.10. Imágenes de la colocación de la caja 16 en los límites de la cámara	28
4.11. Imágenes de la colocación de la caja 23 en los límites de la cámara	28
4.12. Imágenes de la colocación de la caja 25 en los límites de la cámara	29
4.13. 25 imágenes de los ángulos de las cajas	30
4.14. 25 imágenes de las cajas acercadas y centradas en el centroide	31
4.15. Fotos de los 25 pick y place (1)	33
4.16. Fotos de los 25 pick y place (2)	34
4.17. Fotos de los 25 pick y place (3)	35

Capítulo 1

Introducción

En este primer capítulo, se describen los objetivos del proyecto, la motivación que ha llevado a la elaboración del trabajo y los antecedentes del mismo.

Por último, se hará un breve desarrollo de como ha evolucionado la visión artificial desde sus inicios hasta hoy, en el apartado *Estado del arte*.

1.1. Objetivos, motivación y antecedentes

1.1.1. Objetivos

A continuación se desglosan los objetivos a alcanzar en la realización del Trabajo de Fin de Máster:

- Recoger imágenes, a través de una cámara de *Intel Real Sense D435*, para poder posteriormente analizarlas mediante visión artificial con el software de programación *Python*¹ y la librería de visión *OpenCV*².
- Localizar en la imagen el objeto deseado y su forma independientemente del color y dimensiones, en el caso del estudio este objeto serán cajas.
- Poder conocer en el mundo real donde se encuentra el objeto respecto de la cámara y, así también, respecto del brazo del robot.
- Tener una base de datos, compatible en *Python*, con la geometría de cada caja para conocer cual es la caja con la que se está trabajando en cada momento.
- Realizar un código robusto que permita hacer el *picking and place* de diferentes cajas de manera eficaz con el robot de *Universal Robot UR5* [8].

1.1.2. Motivación

La finalidad de este Trabajo de Fin de Máster es la realización de un código robusto en el lenguaje de programación *Python* que permita el *picking and place* de cajas de diferentes dimensiones y colores mediante visión artificial con una cámara *Intel Real Sense D435* y el robot de *Universal Robot UR5*. Las ubicaciones y orientaciones de dichas cajas podrá ser diversas siempre y cuando se encuentren en el espacio de trabajo de la cámara del robot.

El pick and place de objetos es una tarea tan común e importante como repetitivo y tediosos en el sector industrial. El desarrollo de nuevas técnicas de visión artificial pretenden solucionar este problema, evitando que los operarios tengan que realizar estas labores y, además, mejorando la producción. Este TFM pretende incidir en este asunto realizando un software que permita realizar el pick and place de cajas de diferentes dimensiones, orientaciones y colores.

1.1.3. Antecedentes

El tema de utilizar la visión artificial para conocer el lugar donde se encuentran los objetos y poder agarrarlos ha sido tratado anteriormente en dos estudios realizados en el Instituto de Automática e Informática Industrial (AI2) de la Universidad Politécnica de Valencia (UPV).

El primero de ellos [3] trataba el uso de la visión artificial para el control de calidad en objetos defectuosos. Para ello, usaron una cámara *Intel Real Sense D415* y el hardware preprogramado para la realización de redes neuronales convolucionales *Intel Neural Compute Stick* 2.

En el segundo [2], en cambio, se utilizaban esos sistemas de visión para la localización del punto de agarra de un objeto mediante el uso de redes neuronales convolucionales, y su comparativa con otras técnicas de visión como las técnicas clásicas o la segmentación utilizando

¹<https://www.python.org/>

²<https://opencv.org/>

una nube de puntos. Para la experimentación, además de utilizarse la cámara y el software para el uso de redes neuronales del estudio anterior, se usó el robot de *Universal Robot UR3e*.

Además de esos estudios citados, con el robot *UR5*, que se utilizará en este trabajo, se realizó un prototipo de software para el *picking and place* con el uso de códigos *Aruco*. Estos códigos estaban ubicados tanto en el objeto a coger como en el lugar de recepción y así la cámara guiándose solo por dichos *Aruco*s podía coger el objeto y dejarlo en el lugar deseado.

1.2. Estado del arte

La visión humana es capaz de detectar e identificar diferentes objetos y su posición a través de la información que le llega de la vista al cerebro. La visión por computador trata de imitar ese comportamiento y que el ordenador puede identificar objetos a través de los datos que le llegan de la cámara.

La visión por computador ha ido evolucionando durante los años, según se ha ido desarrollando la tecnología, a nivel de aumento de la potencia de cálculo de los ordenadores y mejora de las cámaras. A continuación, se hará un breve resumen de como ha ido avanzando este campo.

1.2.1. Historia de la Visión por Computador

Los primeros estudios se sitúan en la década de los 60, donde investigadores como Roberts (1963) [10] o Wichman (1967) [11] realizaron estudios para detectar posiciones de los objetos en imágenes, incluso en tiempo real como en el estudio de Wichman. Estos trabajos tuvieron resultados positivos aunque estaban fundamentados en objetos sencillos y tenían fuertes restricciones. En esta década, también, se implementaron algoritmos que actualmente se siguen utilizando como los detectores de bordes de Roberts (1965), Sobels (1970) y Prewitt (1970) [5].

La motivación inicial en este campo con estos primeros avances se convirtió pronto en desánimo debido a la poca utilidad práctica de estos primeros desarrollos, que tenían demasiadas restricciones, y al hecho de que no fuera tan trivial el reconocimiento de objetos de manera artificial como se pensaba en un principio. Esto provocó que en la década posterior hubiera un pequeño abandono en este campo de investigación.

Al comprobar la dificultad del campo, a partir de los años 80 se empezó a utilizar y a implementar métodos para la extracción de características como medio para la detección de objetos. Así aparecen estudios para la detección de texturas (Haralik, 1979), movimiento (Horn, 1981) o esquinas (Hitchen y Rosendfeld, 1982). Además, es en esta década donde aparece uno de los libros más importantes en el mundo de la visión por computador: "A Computational Investigation into the Human Representation and Processing of Visual Information" (David Marr) [9]. En él, por primera vez, se crea una metodología completa para el análisis de imágenes por ordenador. También, comienza a ser una línea de investigación recurrente y se empieza a estudiar en universidades. Asimismo, la aparición de ordenadores cada vez más potentes permite que los resultados de estos estudios sean cada vez más prometedores y se pueda avanzar en el campo.

En los años 90, se seguirá con el avance en el campo, principalmente en la reconstrucción proyectiva en 3D, llevando a una mejora en los algoritmos de calibración de las cámaras y de reconstrucción de escenas en 3D a partir de múltiples imágenes a través de técnicas estéreo.

Pero es a partir del nuevo siglo cuando vuelve a haber un auge en el campo con el desarrollo

de la inteligencia artificial. Las nuevas técnicas de Machine y Deep Learning (la [Figura 1.1](#)³ muestra un esquema de como se organizaría la IA por subgrupos) permitían el reconocimiento de más objetos y de manera más fiable en las imágenes lo que produjo un gran avance.

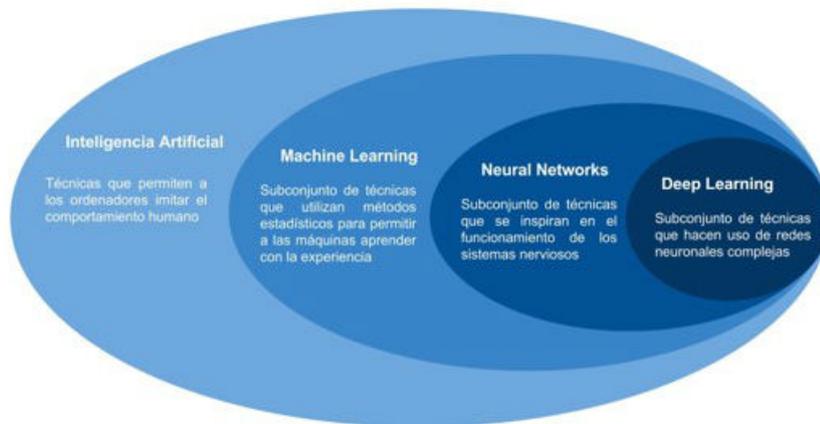


Figura 1.1: Organización jerárquica de la IA

La técnica de Machine Learning trata de utilizar los datos de los que se dispone para que la máquina o el ordenador puedan aprender de manera autónoma. Para conseguir ese aprendizaje se pueden utilizar Algoritmos de Regresión (lineal o polinómica), Algoritmos basados en Instancias, Árboles de Decisión, Algoritmos Bayesianos, Algoritmos de Clustering o Agrupación (como K-Means o K-Medians) o Redes Neuronales [7]. Este último caso, se utiliza para las tareas más complejas y trata de imitar el comportamiento que realizan las neuronas humanas, tomando decisiones en función de las entradas, los pesos y la cantidad y distribución de las neuronas en la red. Por tanto, esta técnica funcionará mejor cuanto más datos (o imágenes en el caso de la visión por computador) se tengan y más fiables sean y, en el caso de las redes neuronales, cuanto mejor se ajuste dicha red a las necesidades de la tarea a analizar, teniendo en cuenta los problemas que pueden surgir de Underfitting o Overfitting.

El Deep Learning es una técnica avanzada de Machine Learning que permite un aprendizaje más en profundidad, con el uso de redes neuronales más complejas. Esta nueva técnica que surgió en 2006 con la creación de la Deep Belief Network [4] y con la aparición de CPUs potentes que permitían realizar ese procesamiento en la red, provocó un nuevo salto en el mundo de la Inteligencia Artificial.

Dentro del Deep Learning, cabe destacar las Redes Neuronales Convolucionales (CNN) [6], que son las redes utilizadas para la clasificación de imágenes. Éstas, inspiradas en el funcionamiento del cortex visual de los animales, funcionan, de manera muy resumida, de la siguiente manera: la red se divide en 3 capas, una primera, llamada capa convolucional, realizará la extracción de las características principales de los píxeles de la imagen, mediante el uso de filtros; la segunda, llamada pooling, será la encargada de reducir las dimensiones de las características extraídas de la capa anterior, manteniendo la información más importante; por último, la tercera capa realizará la clasificación de la imagen. Esta arquitectura permite que esta red sea la más eficiente para la clasificación de imágenes.

Este nuevo siglo también trajo el desarrollo de cámaras cada vez más potentes, lo que ayudó al avance en el campo. Los nuevos modelos de cámaras con más resolución, menos pesadas y más baratas permitían mejores resultados. Además, la aparición, a partir de finales de la primera década del nuevo siglo, de cámaras de visión con profundidad (las llamadas RGB-D) provocaron un nuevo avance. Estas cámaras, no solo permitían realizar fotos y vídeos en

³<https://www.profesionalreview.com/wp-content/uploads/2019/08/Deep-Learning-1.jpeg>

color de buena resolución, sino que podían detectar la profundidad a la que se encontraban los objetos de la imagen. Un ejemplo de estas nuevas cámaras es la que creó Microsoft en 2010 llamada Kinect, creada para jugar a videojuegos en la consola Xbox-360. En la [Figura 1.2](#)⁴ se puede ver dicha cámara.



Figura 1.2: Cámara Kinect

1.2.2. Usos actuales de la Visión por Computador

La visión por computador es una disciplina en auge y que cada vez se está utilizando más en diferentes campos. Aquí se explicarán algunos de los más importantes.

Militar:

La aparición de la tecnología en el área militar es un hecho y los sistemas de visión no son una excepción. La detección y seguimiento de objetivos, el análisis del terreno o la creación de armas inteligentes basadas en estos sistemas son ejemplos de estas aplicaciones.

Robótica:

La implementación tanto de robots fijos como móviles con esta tecnología en un gran avance en el campo, utilizándose para la detección de posiciones y orientaciones de objetos para diferentes aplicaciones (como es el caso del objetivo de este proyecto) o la navegación autónoma en el caso de robots móviles ([Figura 1.3 \(a\)](#)⁵).

Control de calidad:

Posiblemente el campo en que se utilice más. El control de calidad por medio de visión por computador permite un análisis más exhaustivo, preciso y rápido, y se puede utilizar para prácticamente cualquier aplicación de inspección. Algunos ejemplos serían: verificación de etiquetas, control de calidad de comida, inspección de cristales, soldadura, circuitos impresos, maderas, telas... ([Figura 1.3 \(b\)](#)⁶)

Agricultura:

En el mundo de la agricultura también se está progresando. Mediante el uso de imágenes tomadas por satélite se pueden analizar plantaciones, ver su crecimiento, posibles enfermedades... ([Figura 1.3 \(c\)](#)⁷)

Seguridad:

La vigilancia de edificios, la detección de explosivos por rayos X, la identificación de huellas dactilares o el reconocimiento de caras son algunos de los ejemplos en los que la visión por computador está presente en el ámbito de la seguridad.

⁴<https://es.wikipedia.org/wiki/Kinect>

⁵<https://idtxs3.imgix.net/si/40000/93/03.png?w=200&h=150&fit=fill&bg=ffffff&border=0&q=50>

⁶<https://cadecobots.com/wp-content/uploads/2020/04/vision-artificial-trazabilidad-industria-4-0-700x451.jpg>

⁷<https://agriculturers.com/wp-content/uploads/2019/01/Agricultura-e-inteligencia-artificial-de-la-mano-1000x500.jpg>

Control del tráfico:

Para la identificación de matrículas de vehículos o el control del tráfico diario. (Figura 1.3 (d) ⁸)

Biomedicina:

En el mundo de la medicina también está teniendo un papel importante. La transformación de imágenes tomadas por rayos X o ultrasonidos o el análisis de sangre o ADN son claros ejemplos de ello. (Figura 1.3 (e) ⁹)



Figura 1.3: Aplicaciones de la Visión por Computador

1.2.3. Pick and Place con Visión por Computador

El pick and place es una técnica muy utilizada en la industria que consiste en la selección y recogida de objetos para una recolocación, ordenación, empaquetado o paletización entre otras labores. Este proceso tedioso y a menudo poco ergonómico siempre a sido llevado a cabo por uno o varios operarios hasta la llegada de la automatización a las fábricas. La aparición de los robots en las empresas permitía reemplazar a esos operarios en esas tareas repetitivas, siendo éstos más rápidos y precisos y ahorrando costes de mano de obra. Aún así, estos robots no permitían mucha flexibilidad en las tareas, debían ser tareas muy sencillas en cuanto a los objetos en cuestión como al posicionamiento de los mismos.

Para solucionar este problema de rigidez, subieron los sistemas VGR (Vision Guided Robotic Systems), que permitían utilizar la visión artificial para el guiado de los robots. Estos sistemas permitía reconocer objetos, sus formas, su ubicación en el área de trabajo o, incluso, realizar inspecciones de objetos defectuosos, dando una nueva dimensión al mundo del pick and place. En este proyecto se trabajará sobre este punto, aunque cabe destacar que estos sistemas tienen el principal inconveniente de que las piezas a reconocer no pueden estar unas encima de otras, ni solapadas.

⁸https://d7lju56vlbdri.cloudfront.net/var/ezwebin_site/storage/images/_aliases/img_1col/noticias/emplean-la-vision-artificial-en-el-seguimiento-de-matriculas/862937-1-esl-MX/Emplean-la-vision-artificial-en-el-seguimiento-de-matriculas.jpg

⁹https://www.upv.es/noticias-upv/imagenes/int_senolitico.jpg

La aparición del Bin Picking [1] trata de solucionar el problema anterior. Este nuevo avance permite el reconocimiento de objetos dentro de un contenedor lleno de múltiples objetos iguales o diferentes que pueden estar solapados o superpuestos entre sí. Para realizar ese reconocimiento se utilizan las técnicas de inteligencia artificial, sobre todo, el Deep Learning.

1.3. Estructura de la memoria

Este TFM se estructura en cuatro capítulos, a parte de la introducción:

Capítulo 2

Dispositivos hardware utilizados: en él se explicarán los principales componentes hardware utilizados, además del porqué de su uso y las principales características de cada uno de ellos.

Capítulo 3

Explicación y análisis del software utilizado: aquí se analizará el código software creado, explicando paso a paso las diferentes funciones elaboradas y su utilidad para el objetivo del proyecto.

Capítulo 4

Resultados experimentales del proyecto: en este capítulo se mostrarán diferentes imágenes y datos que se van obteniendo de cada una de las partes del código, comentando los resultados obtenidos en cada paso y como afectan a los pasos siguientes y al picking y place final de las piezas.

Capítulo 5

Conclusiones y trabajos futuros: por último, se expondrán las conclusiones obtenidas y algunos trabajos a realizar en el futuro relacionados con este trabajo.

Capítulo 2

Dispositivos hardware utilizados

Durante este capítulo se comentarán las principales funcionalidades de los dispositivos hardware que se utilizarán durante el trabajo. Estos dispositivos serán principalmente 2: el robot de *Universal Robot UR5* y la cámara de visión con profundidad *Intel Real Sense D435*. A continuación, se explicarán cada uno de las principales características de cada uno de ellos. También, en los anexos del documento se encuentran sus datasheet.

2.1. Robot de *Universal Robot UR5*

El robot UR5 fue utilizado para el pick and place de las cajas. Este robot tiene la gran ventaja de ser un robot colaborativo, lo que quiere decir que permite la realización de sus funciones al lado de un operario sin necesidad de jaula o elemento de seguridad entre ellos. Estructuralmente, se trata de un robot relativamente pequeño, con un radio de alcance de 850 mm y un peso de 18.4 kg, formado por 6 ejes y 6 grados de libertad y con rango de movimiento de 360 grados. Además, para el control del robot se dispone de una caja de control y una pantalla táctil, que permite su movimiento a diferentes lugares de manera manual. También, permite la comunicación de manera remota por medio del protocolo de comunicación *TCP/IP* a través de *sockets*. Para nuestro proyecto, realizado en *Python*, se utilizará dicha librería de *sockets* para la comunicación y el desplazamiento del robot de manera remota. En el extremo del robot se encuentra la garra, formada por un grupo de ventosas que se utilizarán para aspirar las cajas y así poder cogerlas. El control de dichas ventosas se puede realizar tanto, a través del monitor táctil que tiene el robot, como de forma remota como en el caso del desplazamiento del robot, añadiéndole la librería *rtde.io*. En el anexo [Sección A.1](#) se pueden ver todas las especificaciones técnicas del robot.

2.2. Cámara de visión con profundidad *Intel RealSense D435*

Esta cámara fue escogida por su sistema de detección de profundidad, lo que la hacía ideal para el objetivo del proyecto. Este sistema estéreo de detección de profundidad permitía una resolución de hasta 1280x720 y hasta 90 fps (fotogramas por segundo). La distancia mínima para una detección adecuada de la profundidad es de 0.105 m. Además, las pequeñas dimensiones de la cámara, 90mm x 25mm x 25mm, y su conexión por USB 3 Type-C, la hacían adecuada para su colocación en el brazo del robot. Hay que decir que, que su comunicación con el sistema de control, en este caso el ordenador portátil, y la cámara sea a través de un cable USB es una desventaja en el sentido de que ambos tienen que estar a una distancia más o menos cercana, cosa que no ocurre con el robot, aún así para este proyecto no fue un inconveniente. También, como aspectos más destacados de esta cámara se encuentran su amplio campo de visión, su sensibilidad, aún con poca luz, y su resolución RGB de hasta 1920x1080. En cuanto a la comunicación con el software, dicha cámara dispone de una librería específica en *Python* (*pyrealsense2*) para la utilización de todas las funciones de las que dispone. Para más información, en el anexo [Sección A.2](#) se encuentra su hoja de características.



Figura 2.1: Vistas frontal y perfil del área de trabajo del robot y la cámara

Capítulo 3

Explicación y análisis del software utilizado

En este capítulo se analizará y explicará como se ha organizado el software, cada función y objeto creado y para qué, además de cada función de las diferentes librerías que se han utilizado.

El programa se estructuró de la siguiente manera, un código principal dividido en diferentes funciones que servirán para organizar cada uno de los pasos a seguir por el software, y, dos objetos creados uno para la comunicación con el robot y su garra y, otro, para las consultas con la base de datos, elaborado en el servidor *PostgreSQL*.

La estructuración del capítulo empezará comentando los dos objetos elaborados, como se realiza esa comunicación por código del robot y de la base de datos, posteriormente, la estructuración del programa principal en el main, los pasos a los que se va llamando a cada función y el porqué de ese orden, y, por último, la explicación de cada una de las funciones que se han ido elaborando en el programa.

Todo el código se muestra en el [Apéndice B](#).

3.1. Objetos

3.1.1. Objeto *ur.hadle*

El objeto creado para la comunicación con el robot se nombró como *ur.hadle* (Subsección B.1.1). Como se explicó en el apartado anterior, el robot de UR permite la comunicación de manera remota por el protocolo *TCP/IP*, esto quiere decir que se realiza a través de *sockets*, por tanto, lo primero que se hizo fue importar dicha librería. Además, para la utilización de la ventosa, se importó la librería *utde.io*. Respecto a la clase del objeto, se crearon dos constructores, uno para el *Host* y otro para la comunicación con la garra, también mediante el *Host* apoyándose de una función interna de la propia librería. Se elaboraron 3 funciones dentro de la clase, dos para la garra y otra para el movimiento del robot. En las dos funciones de la garra permiten introducir la salida digital de las ventosas a 0 o a 1 para realizar la absorción o no. La función de movimientos, permite desplazar el robot a la posición en *x*, *y* y *z* y las orientaciones en los 3 ejes del extremo del robot, con la velocidad y aceleración que se quiera. Para realizar ese proceso, se debe inicializar el *socket*, conectar con el *Host* del robot, enviar la posición del robot y orientaciones además de su velocidad y aceleración en forma de array y, por último, cerrar la comunicación. Toda esta secuencia se realiza con la ayuda de las funciones internas de la propia librería de *socket*.

3.1.2. Objeto *Database*

Como se ha explicado anteriormente, la base de datos que se utilizó se realizó con el servidor *PostgreSQL*, la comunicación con dicho servidor se realiza mediante el protocolo *TCP/IP*, como en el caso del robot. Esta comunicación se puede realizar de dos maneras: estableciendo la comunicación y cerrándola en cada consulta a la base de datos, o, realizando una "piscina" de conexiones que permite no tener que iniciar y cerrar la conexión siempre que se quiera consultar algo, lo que es poco eficiente. Para realizarla de la segunda de las maneras se debe importar la librería *pool* desde *psycopg2*. Se elaboraron 2 clases (Subsección B.1.2), una primera con los métodos que permitían manejar las conexiones y, otra, que permitía la creación y funcionamiento de la piscina de conexiones. La primera de las clases (*Database*) tenía como métodos la inicialización de la conexión, conseguir la conexión, devolver la conexión y cerrar todas las conexiones. Para ello, se utilizaron las funciones internas de la librería importada. La segunda de las clases (*CursorFromConnectionFromPool*), permite establecer esa conexión en cada llamada al objeto, para ello, se utilizan los métodos de la clase anterior.

Base de datos

El objeto anterior permite la comunicación y las consultas con la base de datos, la que se muestra a continuación. Esta tabla contiene los valores geométricos de las diferentes cajas que se utilizaron en el proyecto.

Id	Nombre	Largo (mm)	Ancho (mm)	Altura (mm)
1	Caja 1	220	140	70
2	Caja 2	240	160	80
3	Caja 3	285	190	102
4	Caja 4	325	220	115
5	Caja 5	365	250	130

Tabla 3.1: Base de datos Cajas

3.2. Código principal

El código principal se organizará en funciones que se irán llamando paso a paso en el programa. A continuación, se dará una breve explicación de cada una de las funciones que se van llamando durante la ejecución del software para que se entienda la estructura seguida, posteriormente, en los apartados siguientes se explicarán más en detalle cada una de las funciones realizadas.

Al inicializarse la aplicación, lo primero que se hace es realizar las conexiones con el robot y con la base de datos para poder utilizarse durante el resto del programa. A continuación, dentro de un bucle infinito, ya que la aplicación debe estar ejecutándose todo el tiempo, se van realizando las llamadas a las diferentes funciones. Primero, se llama a *go.home*, esta función permitirá al robot colocarse en la posición inicial de partida dónde se tiene una buena ubicación para que la cámara tome la primera foto. Posteriormente, con el robot colocado en la posición inicial, la cámara toma la primera foto y obtiene los valores de color y profundidad de cada uno de los píxeles, las funciones *Start.Camara* y *Obtain.Data* realizan esa labor. A continuación, conociendo ya los valores de color de cada píxel, se segmenta la imagen para eliminar el fondo y ya conocer la ubicación de la caja en la imagen. El siguiente paso, si la caja se encontrase en un borde de la imagen y no se pudiera ver al completo, se indicará al robot que se desplace ligeramente hacia un lado o otro para ubicarla perfectamente en la imagen, esto se realizará con la función *Obtener.Obj.Borde* y las funciones de desplazamiento del robot *Move.Left*, *Move.Down*, *Move.Up* y *Move.Right*. Con el objeto ya ubicado y segmentado, lo siguiente será conocer la orientación de la caja con respecto a la cámara, que será la misma que con respecto al robot. Para ello, basándose en las esquinas inferior y derecha de la caja y mediante trigonometría, se puede conocer dicha orientación, la función *Encontrar.Angulo* se encargará de obtener dicho ángulo. También, ya sabiendo la posición en píxeles del centroide de la caja, con la función *Segmentacion*, y las correspondientes posiciones en x , y y z de cada píxel, con la función *Obtain.Data*, se conocerá la posición en coordenadas reales del centroide con respecto a la cámara, para ello se utilizó la función *Centroides.XYZ*. Después de conocer esos valores, se moverá el robot para centrar y aproximar la cámara al centro de la caja, para un segundo cálculo de la profundidad de la caja con respecto a la cámara y así evitar equivocaciones en esa medida. Es aquí donde se utilizan las funciones *Pick*, las dos del inicio para obtener los valores de la cámara y la del cálculo de la posición XYZ del centroide. Conocida esa medida de la profundidad, se puede conocer que caja se va a coger, ya que, cada caja tiene una altura diferente. Es en este paso dónde se hará una consulta a la base de datos, dentro de la función *Conocer.Tipo.Caja*. Ahora que se tienen ya todos los datos interesantes de la cámara, el siguiente paso será coger la caja con el robot y la garra. Para ello, lo primero que se hará será girar en z el extremo del robot para orientarse a la posición de la caja, con las funciones *VRotate.To.RPY*, *RPY.To.VRotate* y *Giro.en.Z*, y, posteriormente, bajar la garra hasta una altura marcada por la profundidad medida por la cámara y coger la caja con la ayuda de las ventosas, con la función *Pick1*. Con la caja agarrada por el robot, se vuelve a la posición inicial del robot y se deposita el objeto en una posición que dependerá del tipo de caja que se haya cogido, función *Place*.

Todo lo explicado anteriormente se representa de manera visual en la [Figura 3.1](#).

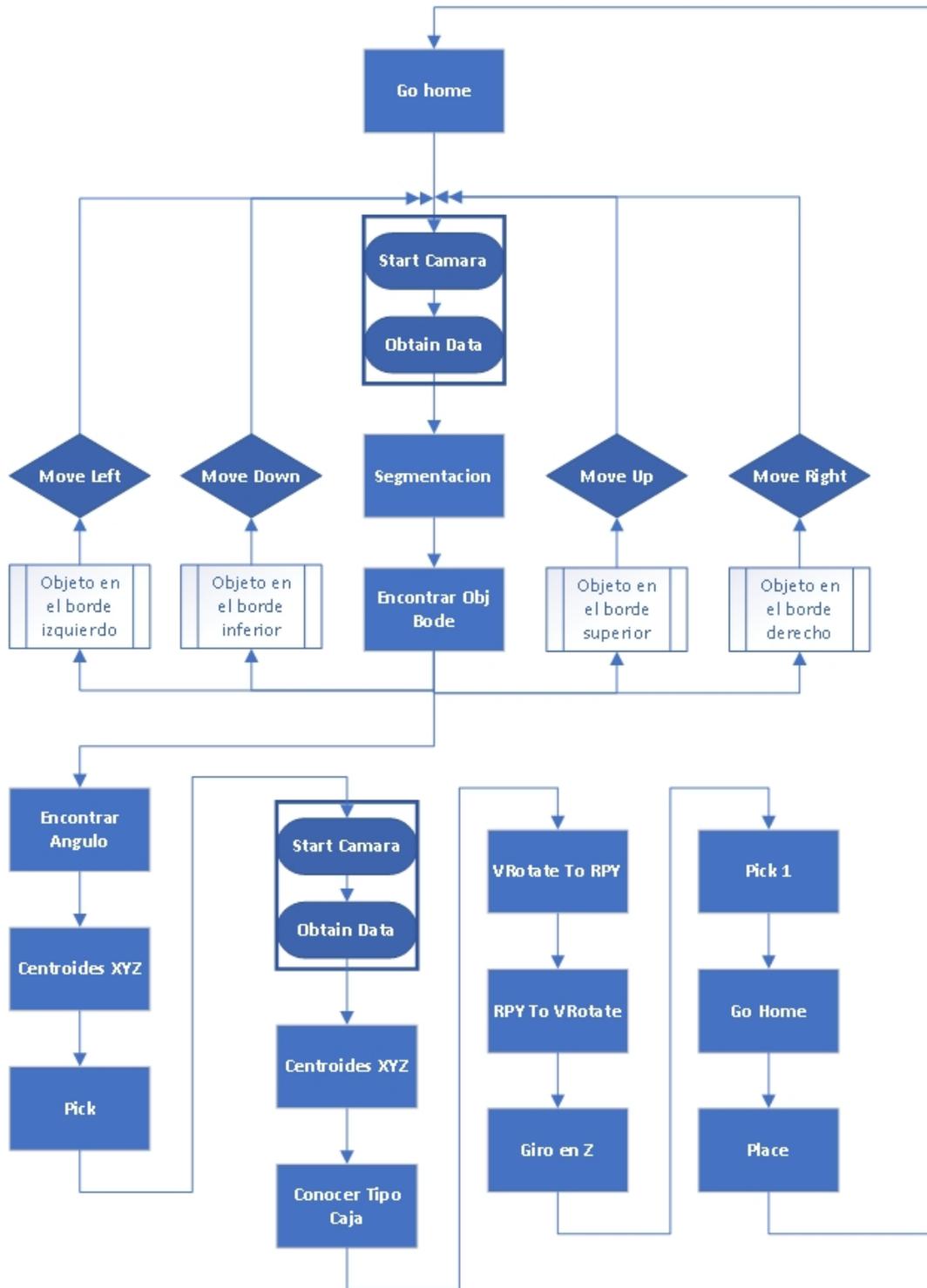


Figura 3.1: Árbol de decisión del programa

3.3. Funciones creadas

3.3.1. Posición inicial del robot

Al iniciar la aplicación lo primero que se debe hacer es colocar al robot en la posición inicial, por tanto, llamará la función *go_home* (Sección B.2.2) que realiza esa tarea. Esta función permite, utilizando la función *socket.moveL* del objeto *ur_hadle*, enviarle al robot las coordenadas de la posición inicial a la que se debe desplazar con la velocidad y aceleración que se quiera, con un cierto delay para que se no ejecute otra función antes de que se haya terminado el desplazamiento del robot.

3.3.2. Inicialización de la cámara y obtención de sus datos

Para la toma de imágenes por parte de la cámara se utilizaron 2 funciones: *start.camara* y *obtain.data* (Sección B.2.2) que se basan en la utilización de funciones internas de la librería de la cámara *pyrealsense2*.

La primera de ellas, permite la configuración de los parámetros de la cámara, haciéndose una nube de puntos, que permita calcular la posición en coordenadas en el espacio de cada uno de los píxeles, o introduciéndose los valores de resolución de la imagen a color y la de profundidad.

Con las variables de configuración obtenidas de la primera de las funciones, se obtiene, en un primer momento, los datos de la imagen a color y de la profundidad de los objetos respecto de la cámara, pudiéndose mostrar esa imagen con las 3 líneas que están comentadas en la función. Después, en las siguientes líneas de código se obtienen las posiciones x , y , z de cada píxel, que se almacenan en el vector `'frame.verts'`. Por último, se para la comunicación con la cámara.

3.3.3. Segmentación de la imagen

La función *segmentacion* (Sección B.2.2) permite segmentar los objetos de la imagen con respecto al fondo. Para ello, lo primero que se hace es pasar la imagen BGR obtenida de la cámara a una imagen HSV, con la que será más fácil realizar la segmentación, con el comando `"cv.cvtColor"`. Para realizar la segmentación se utilizará el color que tiene el fondo del objeto, en este caso, se trabajó sobre una mesa que tenía un tono azulado, por tanto, se usará una máscara que separe los píxeles entre los que tengan ese rango de azul, comprendido entre los vectores `"lower.azul"` y `"upper.azul"`, y el resto de píxeles, para ello se utilizará la función de *OpenCV* `"cv.inRange"`. A la máscara se le realizan algunas transformaciones para dejarla más uniforme y sin aristas, con las funciones `"cv.dilate"` y `"cv.erode"`. A continuación, se obtiene la máscara inversa, con `"cv.bitwise.not"` (la máscara de los objetos detectados) y, se le aplica la función `"cv.connectedComponentsWithStats"`. Esta herramienta interna de *OpenCV* permite obtener características geométricas de cada objeto en la imagen (en este caso 2, el fondo y el objeto), como son las posiciones iniciales de los píxeles en x y en y , su ancho y altura, su área y la posición en píxeles de sus centroides, todos estos parámetros nos servirán de ayuda para nuestro estudio. Posteriormente, con la función `"cv.findContours"`, se tienen los píxeles de los contornos de cada objeto. Las líneas comentadas en el código permiten visualizar el centroide de la caja, el rectángulo donde se encontraría comprendida y el contorno de la caja. Por último, se guardan todos esos valores obtenidos para utilizarlos posteriormente, al igual que la imagen del objeto.

3.3.4. Comprobar si la caja se encuentra en los bordes de la imagen y, si es así, ajustar la posición de la cámara para visualizarla entera

Para reconocer que la cámara ha obtenido la caja al completo y no una parte solo, se utilizará la función *encontrar_obj_borde* y las funciones de desplazamiento del robot *move.left*, *move.right*, *move.down* y *move.up* (Sección B.2.2). Con la función de *OpenCV* “*cv.connectedComponentsWithStats*” utilizada anteriormente, se puede saber si el objeto se encuentra en los límites de la imagen y si es así, poder mover el robot y tomar una nueva imagen hasta conseguir centrar el objeto en la imagen al completo.

3.3.5. Encontrar el ángulo con el que está desplazada la caja respecto la cámara

La función *encontrar_angulo* (Sección B.2.2) permite obtener el ángulo que tiene la caja con respecto a la cámara. Para ello, se utilizó el contorno del objeto y las posiciones máximas de los píxeles en x e y . Lo que se quiere conseguir es obtener las esquinas de la caja y, así, saber en que píxeles se encuentran y, por tanto, conseguir el ángulo de la caja. El primer paso fue crear unas variables locales que vayan modificándose dependiendo las condiciones del bucle. Después, se introdujeron 2 bucles, uno que recorriese los diferentes contornos (en principio, suele haber solo uno, pero puede haber más), y otro que recorriese las posiciones del contorno. A continuación, como se quieren obtener las esquinas del objeto, se compararán los píxeles del contorno respecto a los límites que se han obtenido del objeto. Como puede haber más píxeles que se correspondan con esos límites, se calcularán, de ese rango de valores, tanto el valor menor como el mayor correspondientes al otro píxel, por eso, se obtienen 2 valores para cada esquina, almacenándose en una variable la posición del vector de contornos. Sabiendo esas posiciones se pueden dibujar esos valores obtenidos para poder visualizarlo mejor (es lo que se haría descomentando las líneas comentadas en el código). Conociendo ya los píxeles correspondientes a cada esquina, se puede obtener el ángulo a través de transformaciones trigonométricas. Antes de ello, se evaluará si el objeto está lo suficientemente centrado como para no girarlo, para ello, se observa si el rango de los píxeles del contorno que coinciden con los límites del objeto es mayor de 50 en al menos 2 de los lados, y si es así, el ángulo de giro será 0. Si ese ángulo no es 0, se obtiene el ángulo a partir del seno y el coseno que se calculan de las esquinas inferior y derecha.

3.3.6. Posición x, y, z del centroide

Con los valores obtenidos de los píxeles del centroide y de los datos de posiciones de la cámara, se consigue la posición real del centroide en el mundo, la función *centroide_xyz* (Sección B.2.2) se encargará de calcular esos valores. Además, en esta parte del código se evalúa si existe un error en la toma de datos de la cámara, y si es así, se toman unos nuevos datos y se evalúa de nuevo esa posición.

3.3.7. Acercamiento de la cámara al objeto

La posición de la cámara respecto al objeto está algo lejos, esto puede provocar que la profundidad medida por ella sea algo errónea (unos milímetros), por tanto, lo que se hace con la función *pick* (Sección B.2.2) es centrar la cámara al centroide y acercarla para volver a medir esa profundidad. Debido a que las cajas utilizadas en el proyecto tienen alturas parecidas, un fallo de unos milímetros en la detección de la profundidad podría provocar que el tipo de caja que se detectase fuera erróneo, por tanto, es importante volver a realizar esta medida.

3.3.8. Conocer el tipo de caja

La función *conocer.tipo-caja* (Sección B.2.2) permitirá conocer qué tipo de caja es la que se ha reconocido en la imagen. Para ello, conociendo la profundidad medida por la cámara y la altura a la que se encuentra la cámara del fondo, se obtiene la altura de la caja (todo esto en milímetros) y, recorriendo los datos geométricos de las cajas que se tienen de la base de datos (en este caso la altura de las cajas), se puede conocer cuál es la altura medida que más se asemeja a las alturas conocidas y, así, saber qué tipo de caja es. En esta función se utilizarán los objetos creados para la comunicación y las consultas a la base de datos.

3.3.9. Transformaciones de vector de rotación a Roll-Yaw-Pitch y viceversa para conseguir un giro de la última articulación solo en Yaw

Para conseguir que la ventosa del robot solo gire respecto al eje "z" en función del ángulo obtenido, se debe utilizar el sistema de coordenadas Roll-Yaw-Pitch. Por tanto, lo primero que se hará, será conocer qué ángulo tiene la ventosa en la posición inicial en coordenadas RPY, ya que, sí se conoce en coordenadas de vector de rotación, esta transformación se realiza con la función *VRotateToRPY* (Sección B.2.2). A continuación, el objetivo será conocer las coordenadas del vector de rotación después de sumar el ángulo de la caja (que se obtiene de la función *encontrar.angulo*) en Yaw a la posición inicial obtenida anteriormente en coordenadas RPY. Para la transformación de un ángulo de coordenadas RPY a vector de rotación se utilizará la función *rpmToVRotate* (Sección B.2.2). Las transformaciones realizadas en las dos funciones se pueden obtener fácilmente a través de Internet, como, por ejemplo, en esta página web del pie de página¹.

3.3.10. Giro de la última articulación del robot en Z

Como se ha obtenido de las funciones anteriores lo que debe girar la ventosa en el eje z según como esté girada la caja, la función *giro.en.z* (Sección B.2.2) permitirá que el robot haga ese giro.

3.3.11. Pick de la caja

Después de haberse realizado el giro, el siguiente paso es coger la caja por parte del robot, la función *pick1* (Sección B.2.2) se encarga de realizar ese proceso. Este código desplazará el objeto en x, y y z según las distancias medidas de la cámara respecto a la ventosa del robot y, además, una distancia en z correspondiente a la profundidad medida por la cámara en la segunda de las medidas (después de haberse centrado la cámara en el centroide del objeto y haberse acercado en el eje z). También, en las últimas líneas de la función se realiza la absorción de la caja con la llamada a la función "grip" del objeto creado para la comunicación con el robot.

3.3.12. Place de la caja

Después de que el robot coja la caja y la mueva a la posición inicial, la función *place* (Sección B.2.2) permite dejarla en la misma posición x e y, y a una distancia en z dependiendo el tipo de caja que sea. Esto permitirá dejar la caja con suavidad independientemente la altura que tenga la caja. Al final del código se llama a la función *release* del objeto del robot para que la ventosa deje de absorber.

¹<https://www.zacobria.com/universal-robots-knowledge-base-tech-support-forum-hints-tips-cb2-cb3/index.php/python-code-example-of-converting-rpyeuler-angles-to-rotation-vectorangle-axis-for-universal-robots/>

Capítulo 4

Resultados experimentales del proyecto

En este capítulo, se muestran los resultados experimentales del proyecto. Como se explicó en los apartados anteriores, el objetivo principal del estudio era realizar un código robusto que permitiera el *pick and place* correcto de cajas fuera cual fuera sus dimensiones o colores, por tanto, se realizaron muchas pruebas, aunque en este capítulo solo se mostrarán 25. Además del objetivo principal, existían varios desafíos secundarios a dar solución como son el problema de la luz ambiental y los reflejos en la segmentación por color de los objetos, cómo conocer la ubicación de las esquinas del objeto en cada momento, siendo además, los lugares donde la segmentación podría no ser perfecta, o, cómo resolver el problema de que la cámara no detectara la profundidad y la posición en x e y del objeto correctamente. Todos estos puntos se muestran en el siguiente apartado, dónde se explican alguno de los problemas que fueron surgiendo durante las pruebas y como se fueron solventando.

Posteriormente, se muestran y analizan las 25 muestras de los resultados finales de las pruebas, realizadas con diferentes cajas y en distintas orientaciones y posiciones.

4.1. Problemas acaecidos durante las pruebas

El código explicado en el capítulo anterior y que se muestra en el anexo [Apéndice B](#) es la versión final del proyecto, pero para llegar hasta ese punto se tuvieron que ir resolviendo algunos conceptos clave sin los cuales no se hubiera podido llegar hasta el resultado final que se mostrará en el siguiente apartado.

En primer lugar, conocer como funcionaba la cámara y cómo extraer datos de la misma era uno de los puntos clave del proyecto. Mediante la búsqueda de información del fabricante, de la librería y de foros externos se pudo llegar al código que se ha explicado anteriormente ([Subsección 3.3.2](#)) y que se muestra en el anexo [Sección B.2.2](#), aún así, hay un aspecto importante que no se ha podido solucionar y que es debido al procesamiento del ordenador con el que se realizó el proyecto, que es la resolución de las imágenes. Como se muestra en el código dicha resolución es de 640x480 y, como se puede ver el datasheet [Sección A.2](#), la cámara permite tomar imágenes a una resolución mayor, aun así, el ordenador del proyecto no podía almacenar tal cantidad de datos, por tanto, se tuvo que trabajar con esa baja resolución. Este problema se resolvería fácilmente con un ordenador de mayor procesamiento.

El siguiente aspecto a tratar fue la segmentación de los objetos, con una luz no controlada que podía dar lugar a sombras, y un fondo no del todo uniforme. Para trabajar con ese problema lo primero que se pensó fue intentar realizar la segmentación con una imagen del fondo sin objeto y otro con el objeto e intentando extraer la parte no común de ambas imágenes. Esta solución no dio grandes resultados, ya que, los objetos con un color más oscuro sí que los podía segmentar bien, por el hecho de que el fondo era bastante claro, pero las cajas de colores claros no se detectaban correctamente. Para solucionar este problema se optó por la propuesta que se muestra en el código final [Sección B.2.2](#), en la que se utiliza una segmentación de todo lo que no es un color parecido al del fondo, realizando un cambio de la escala de colores de RGB a HSV.

Otro problema importante, sería el conocer dónde se encuentran las esquinas del objeto, para poder guiar el ángulo de giro del robot. Se planteó comparando los píxeles que se obtenían del contorno con los máximos del mismo (se explica detalladamente en el apartado anterior [Subsección 3.3.5](#) y en el anexo [Sección B.2.2](#)). Utilizar esta manera de encontrar las esquinas, provocaba que tuviera buenos resultados en el que caso que la caja tuviera una cierta inclinación, pero en el caso de cajas sin mucha inclinación con respecto a la cámara podría provocar problemas, es por ello que se concluyó el poner ese ángulo como 0 en ese tipo de casos. Algunos ejemplos de esos casos se muestran en las imágenes posteriores.

Conociendo que el robot debía de girar un cierto ángulo en la mayoría de los casos, el siguiente reto era cómo poder realizar ese giro en el último eje del robot, sin alterar las demás articulaciones. Se resolvió, de manera satisfactoria, realizando cambios de coordenadas en los ejes del robot, de vector de rotación a Roll-Yaw-Pitch y viceversa ([Subsección 3.3.10](#)).

Por último, se comprobó que había casos en el que la cámara no podía tomar correctamente los datos de las coordenadas en x , y y z de los píxeles o que esos datos eran algo incorrectos. Es por ello que se decidió hacer una segunda medida, más cercana, de la profundidad calculada por la cámara con respecto al objeto, ya que, era la medida más crítica para los pasos siguientes, y se introdujo, también, un bucle de comprobación de que la cámara había tomado correctamente las medidas en el píxel a evaluar, volviendo a tomar la imagen en el caso en el que no fuera así.

La resolución de estos problemas permitió que los objetivos que se tenían para el proyecto se cumplieran mayormente y que, como se verá en el siguiente apartado, el *pick* y *place* de los objetos se realizara de manera satisfactoria.

4.2. Imágenes y comentarios de los resultados de la prueba

4.2.1. Imágenes de la cámara obtenidas en la posición inicial

Las 25 imágenes obtenidas en el primer paso son las siguientes y, como se puede observar, se utilizan las 5 cajas que se tenían para el proyecto y las posiciones y orientaciones de cada caja son diferentes en cada una de las imágenes.

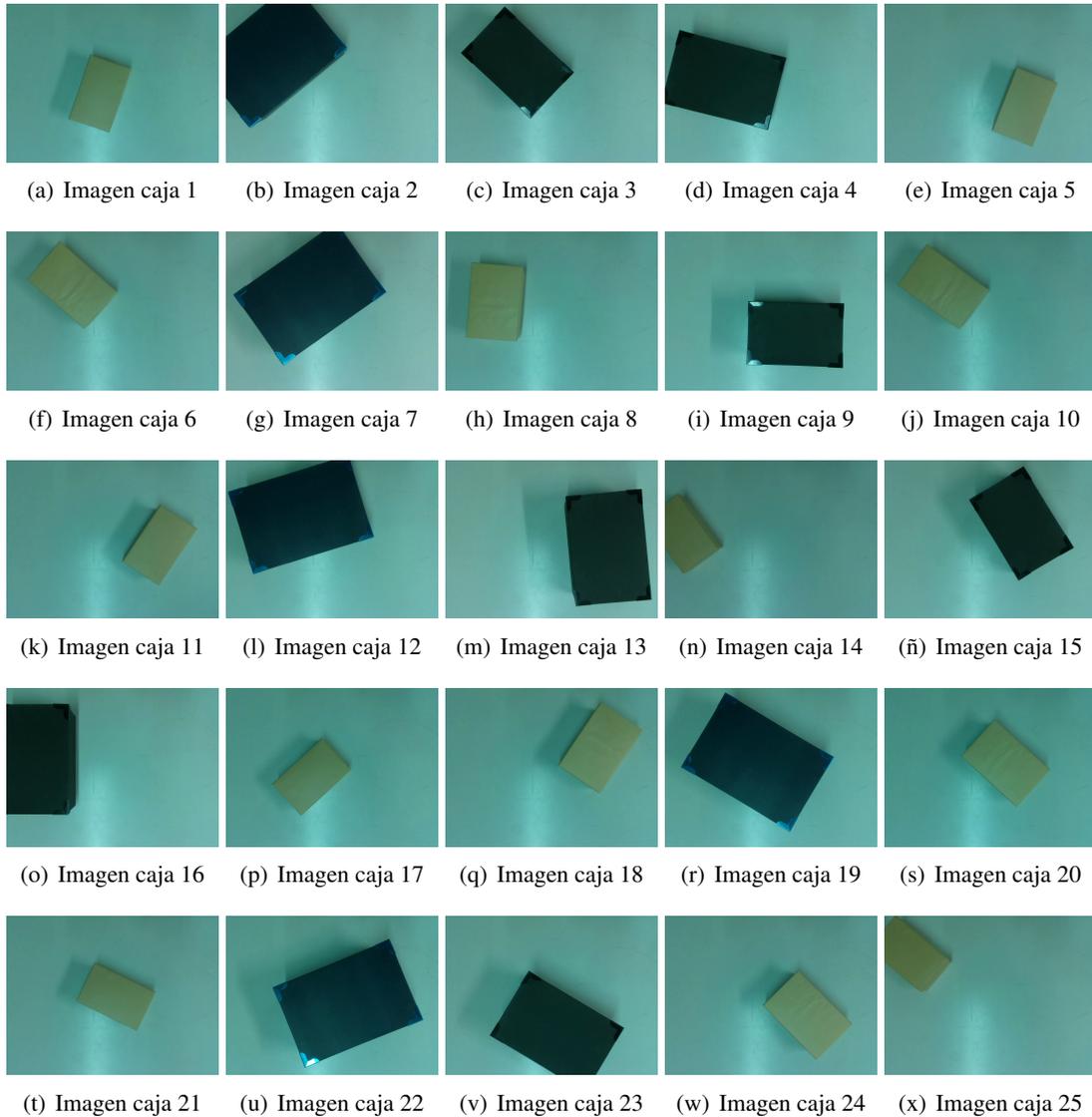


Figura 4.1: Imágenes de las 25 posiciones iniciales

4.2.2. Segmentación de las imágenes

Como se explicó en el capítulo anterior, se hizo una segmentación por color, utilizando el color del fondo y extrayendo todo lo que no tenía ese color. Para hacer este tipo de segmentación lo más interesante es utilizar un mapa de color HSV, que da mejores resultados. Por tanto, el primer paso de esta segmentación es transformar la imagen BGR a HSV. En las [Figura 4.2](#) se muestran esas imágenes en HSV.

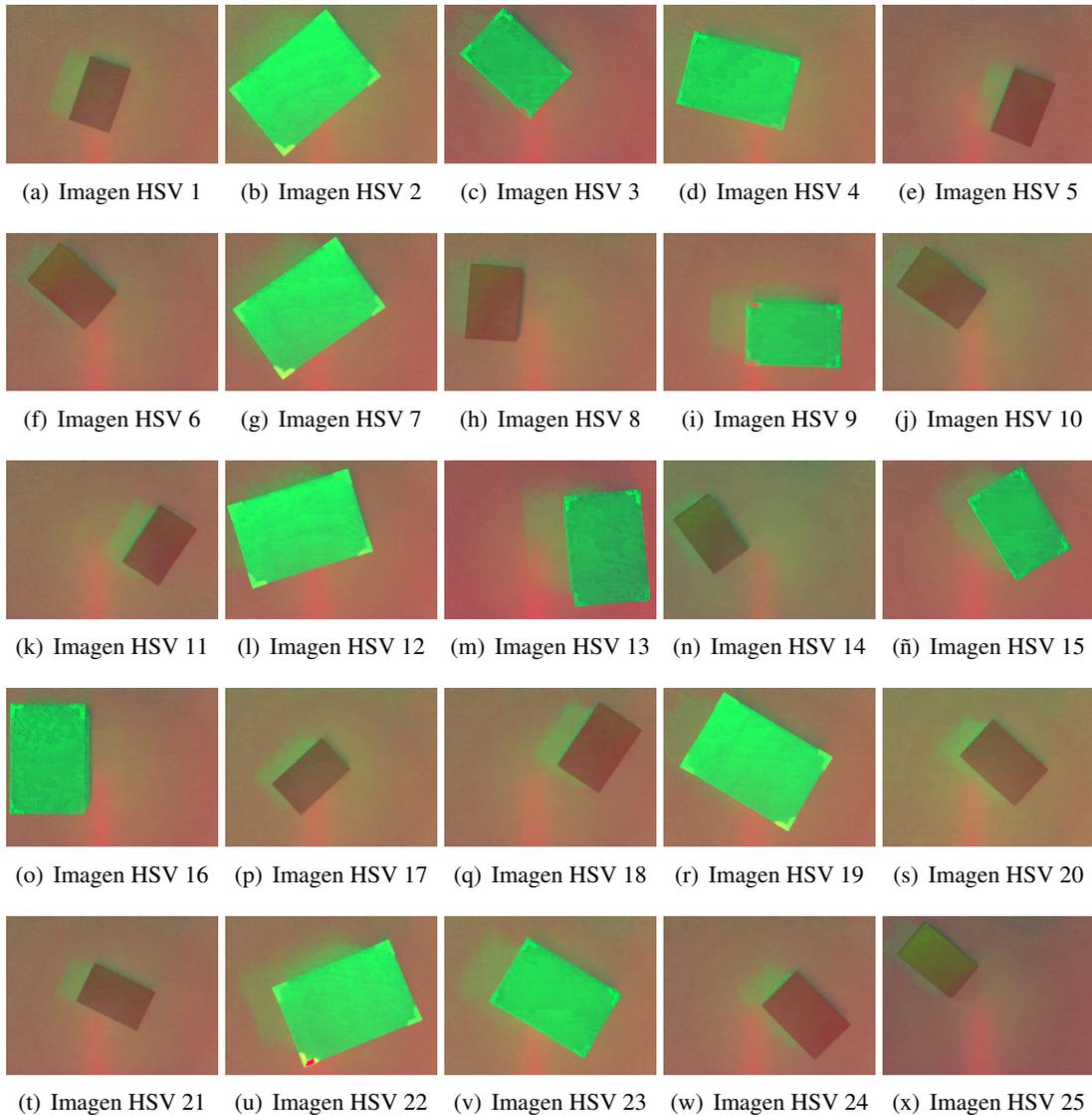


Figura 4.2: 25 transformaciones HSV

Después, se va a obtener la máscara del fondo, para poder así separar el fondo del objeto. En la [Figura 4.3](#) se pueden ver esas máscaras resultantes.

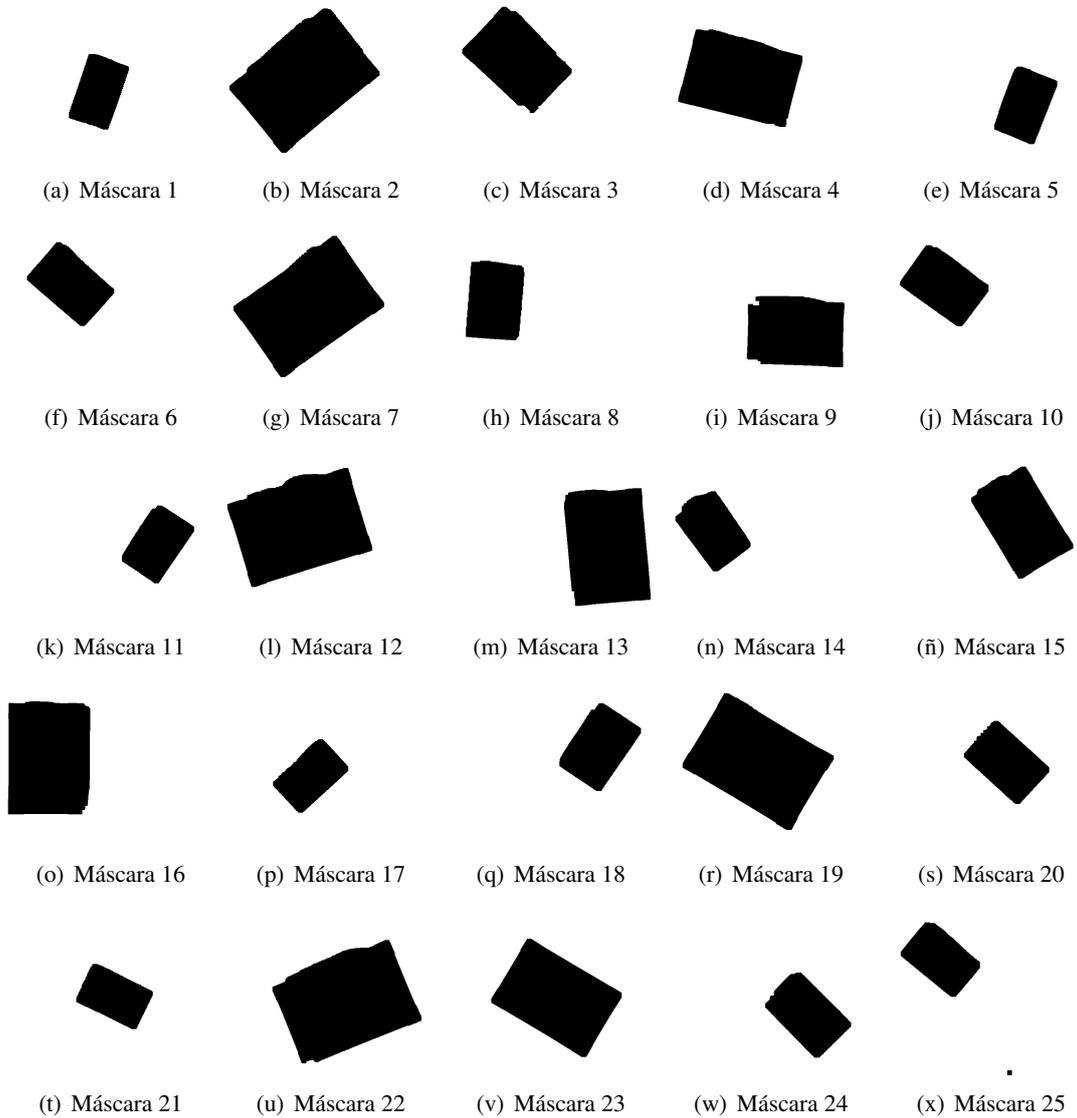


Figura 4.3: 25 imágenes de las máscaras

Por último, con las máscaras invertidas, es decir, de los objetos, se pueden conocer los píxeles más destacados (centroides, máximos y mínimos del objeto o contornos). Por tanto, en la **Figura 4.4** se muestran dibujados todos esos parámetros que son útiles para las transformaciones posteriores.

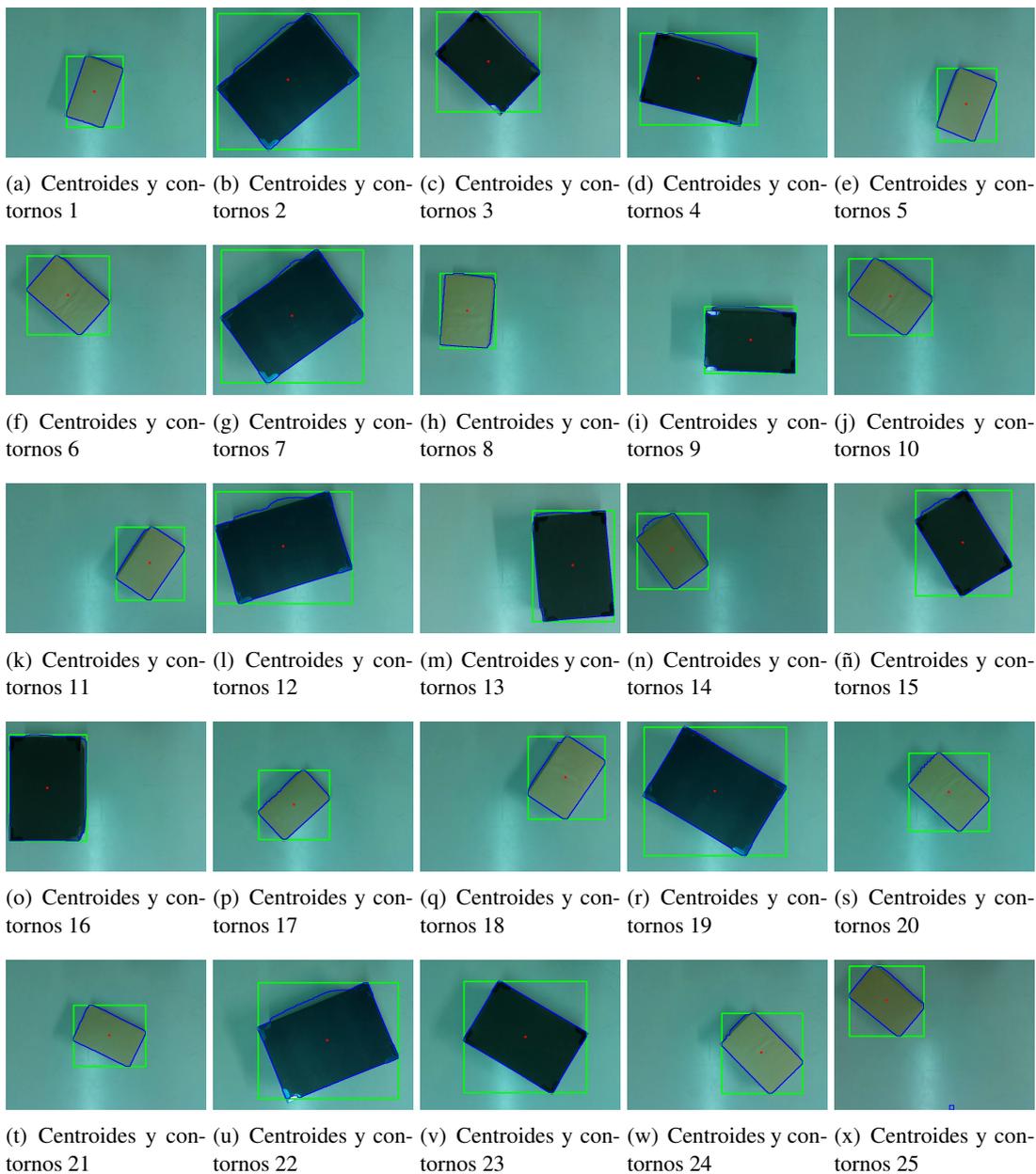


Figura 4.4: 25 imágenes de los centroides y los contornos

Como se puede observar en estas imágenes segmentadas, a veces la segmentación no es del todo perfecta por culpa de las sombras que tienen las cajas, ya que, el área de trabajo no es del todo perfecta en cuanto a la iluminación. Además, en algunas de las cajas más oscuras, sobre todo la caja tipo 5, se produce reflejo en alguna de las esquinas, esto provocará, como se verá después, que el ángulo tomado sea algo incorrecto. Por último, también se puede ver que, aunque la segmentación no sea perfecta en algunos casos, el centroide que se obtiene es bastante exacto en todas las cajas.

4.2.3. Objetos en bordes de la imagen

Se puede observar que en las imágenes de las cajas 2, 4, 7, 12, 14, 16, 23 y 25, no se puede ver la caja al completo por estar en un borde. Como se explicó en el capítulo anterior, la función “encontrar_obj_borde” permite solucionar ese problema y ajustar la posición del robot para que se pueda visualizar la caja al completo. Aquí se puede visualizar las imágenes obtenidas según se va moviendo el robot y la segmentación resultante cuando la caja se muestra al completo.

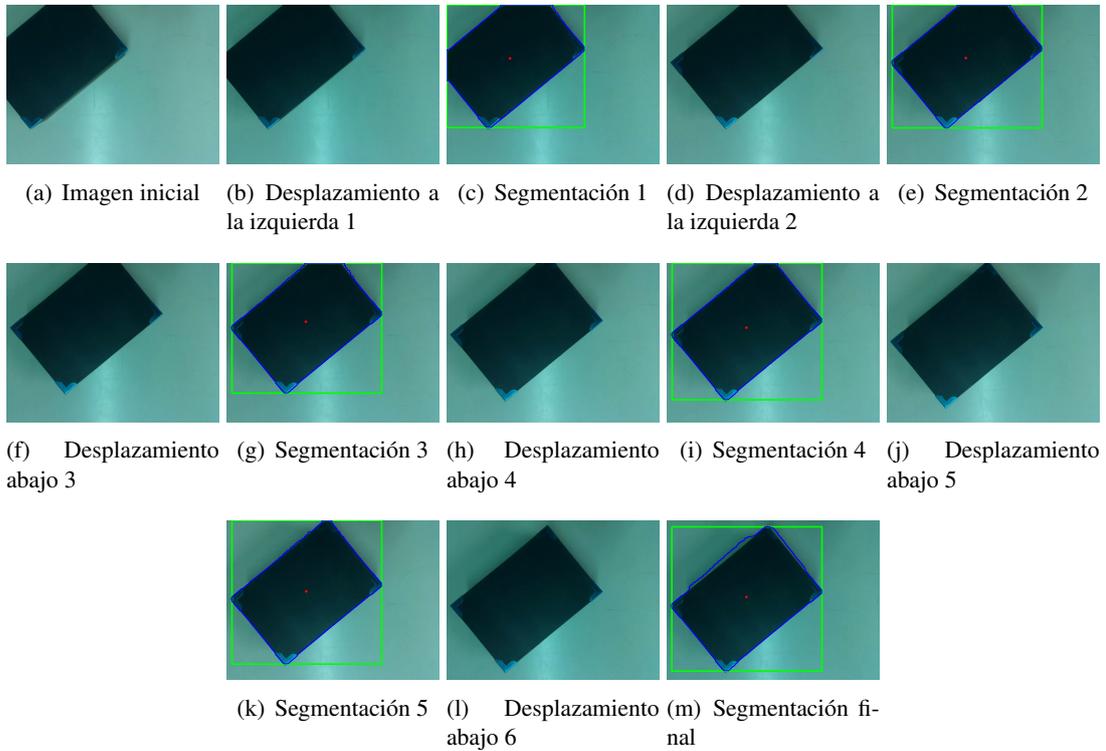


Figura 4.5: Imágenes de la colocación de la caja 2 en los límites de la cámara



Figura 4.6: Imágenes de la colocación de la caja 4 en los límites de la cámara

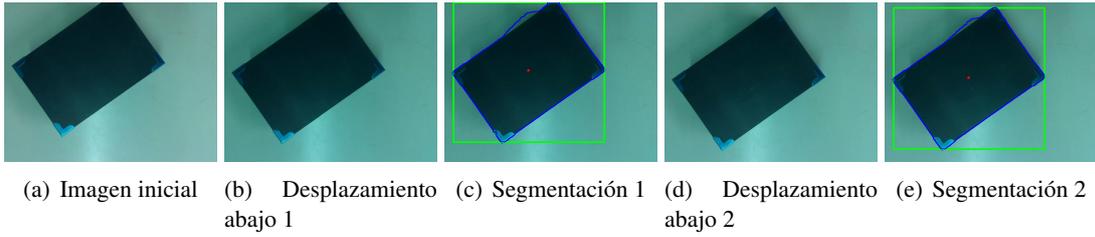


Figura 4.7: Imágenes de la colocación de la caja 7 en los límites de la cámara

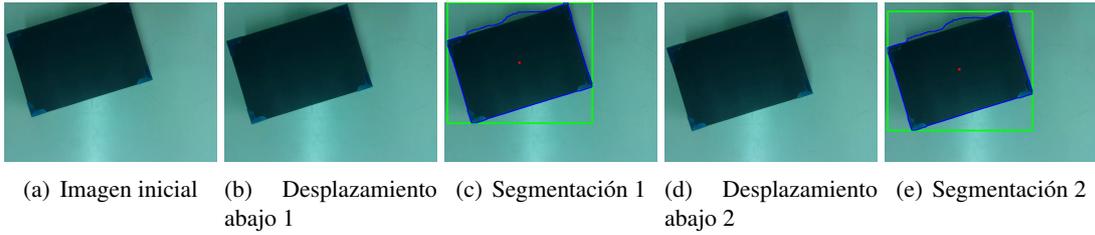


Figura 4.8: Imágenes de la colocación de la caja 12 en los límites de la cámara

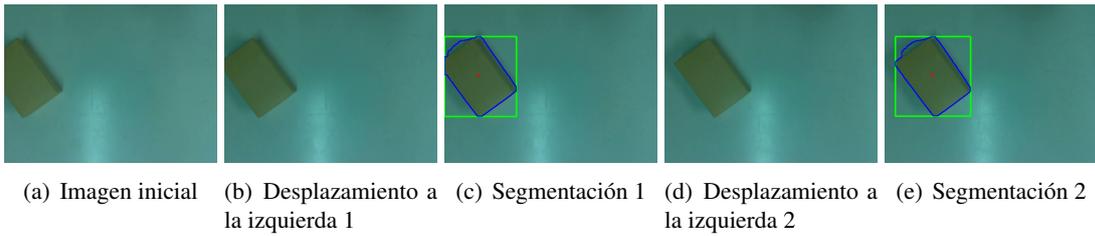


Figura 4.9: Imágenes de la colocación de la caja 14 en los límites de la cámara



Figura 4.10: Imágenes de la colocación de la caja 16 en los límites de la cámara

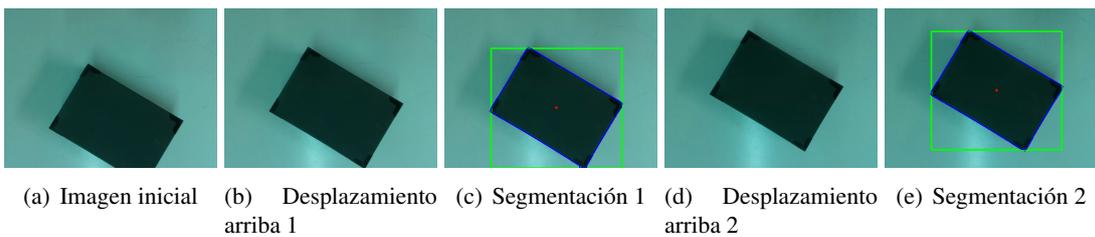


Figura 4.11: Imágenes de la colocación de la caja 23 en los límites de la cámara

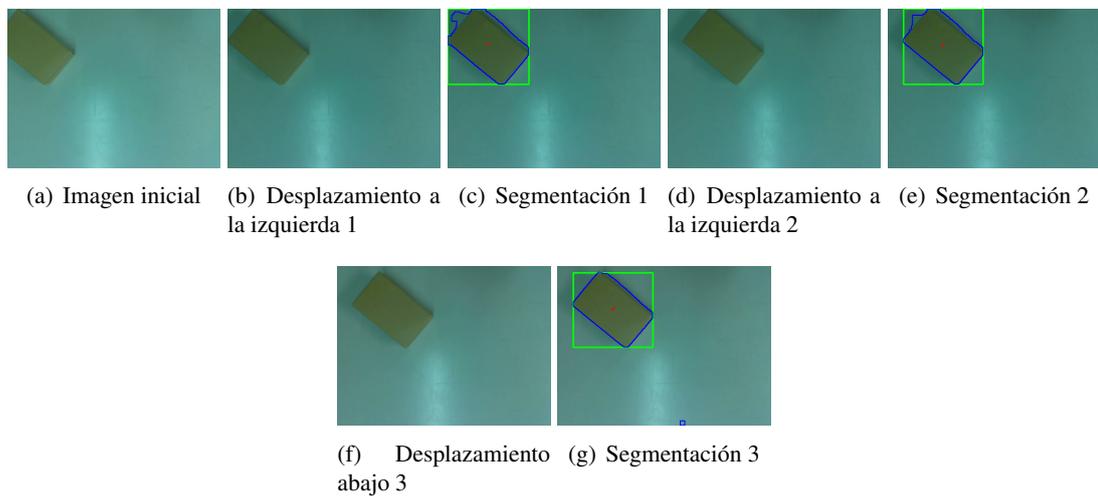


Figura 4.12: Imágenes de la colocación de la caja 25 en los límites de la cámara

4.2.4. Ángulo de la caja respecto a la cámara

Como se expresó en el capítulo anterior, conociendo la posición de los máximos y mínimos del objeto y su contorno se puede calcular la posición aproximada de las esquinas de la caja. Las [Figura 4.13](#) muestran las posiciones de esas esquinas y el coseno y la tangente del ángulo que se va a coger. Con esta visualización se puede observar si ese ángulo será correcto y no.

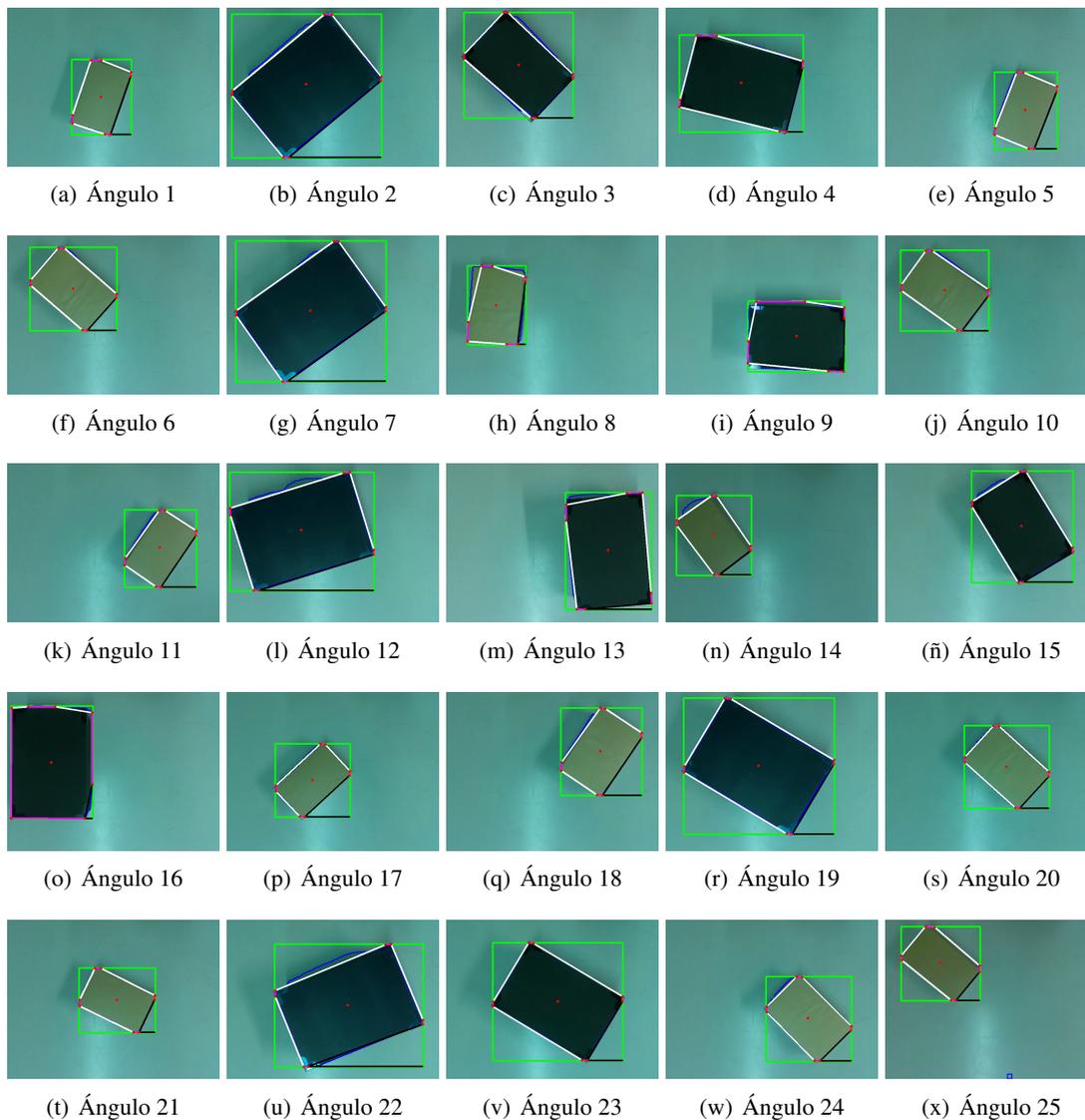


Figura 4.13: 25 imágenes de los ángulos de las cajas

Como se puede observar en las imágenes, se eligió el ángulo de la esquina inferior para encontrar el ángulo porque era el que daba mejores problemas en cuanto a las sombras. Aún así, como se comentó anteriormente, esa esquina provocaba algunos problemas de reflejos en algunas de las cajas más gruesas y oscuras. Pese a esto, era el ángulo que mejores resultados tenía y, como se puede ver en las figuras, casi siempre se obtenía un ángulo correcto en todas las posiciones. Otro aspecto a destacar en estas imágenes, y que más problemas podía dar en cuanto a la obtención del ángulo, eran las cajas que estaban bien alienadas con respecto a la cámara y que daban como resultado ángulo 0. En las 25 figuras se pueden ver dos de estos casos, en la imagen 'i' y 'o'.

4.2.5. Acercamiento, centrado y reconocimiento del tipo de caja

El tomar una nueva imagen de la caja centrada en el centroide y acercada, se realizó para tener una mayor seguridad de la altura que medía la cámara, ya que, como se puede ver en la [Tabla 3.1](#), las cajas tienen alturas muy parecidas, de pocos milímetros de diferencia, por tanto, un fallo de milímetros en esa altura tomada haría que la caja que detecte la función `conocer_tipo_caja` no fuera la que realmente es. En las [Figura 4.14](#) se mostrarán esas nuevas imágenes, comprobándose que están centradas en el centroide, y en la [Tabla 4.1](#) se muestran los resultados obtenidos de la altura medida por la cámara y se comprueban si esos resultados son correctos y la caja detectada es la que realmente es.

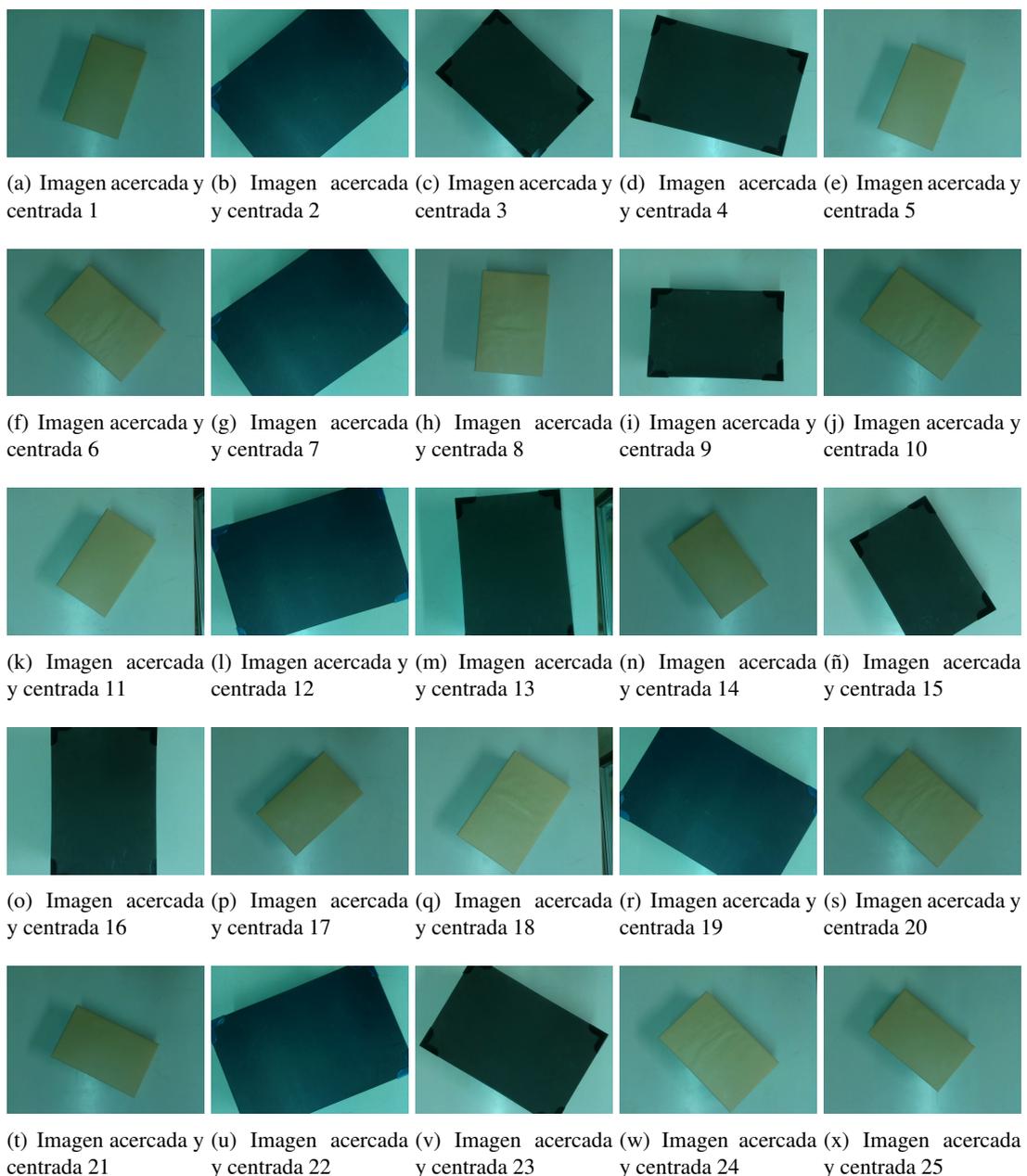


Figura 4.14: 25 imágenes de las cajas acercadas y centradas en el centroide

Profundidad medida (mm)	Altura caja medida (mm)	Altura más parecida BD (mm)	Tipo caja detectada	Tipo caja real
441	71	70	Caja 1	Caja 1
381	131	132	Caja 5	Caja 5
406	106	105	Caja 3	Caja 3
394	118	120	Caja 4	Caja 4
441	71	70	Caja 1	Caja 1
432	80	80	Caja 2	Caja 2
380	132	132	Caja 5	Caja 5
433	79	80	Caja 2	Caja 2
405	107	105	Caja 3	Caja 3
433	79	80	Caja 2	Caja 2
440	72	70	Caja 1	Caja 1
380	132	132	Caja 3	Caja 3
393	119	120	Caja 4	Caja 4
444	68	70	Caja 1	Caja 1
406	106	105	Caja 3	Caja 3
394	118	120	Caja 4	Caja 4
442	70	70	Caja 1	Caja 1
431	81	80	Caja 2	Caja 2
379	133	132	Caja 5	Caja 5
432	80	80	Caja 2	Caja 2
443	69	70	Caja 1	Caja 1
381	131	132	Caja 5	Caja 5
394	118	120	Caja 4	Caja 4
432	80	80	Caja 2	Caja 2
442	70	70	Caja 1	Caja 1

Tabla 4.1: Tabla comparativa detección tipos de cajas

En la tabla anterior se mostraron todas las profundidades detectadas por la cámara después del acercamiento y centrado. Como se observa, las profundidades medidas por la cámara D435 tienen bastante exactitud con las que deberían obtenerse, variando algunos milímetros arriba o abajo, por tanto, la caja detectada en todos los casos es la correcta, permitiendo así que el place final sea siempre el correcto.

4.2.6. Pick y Place de las cajas

Después de los pasos anteriores, lo siguiente sería realizar el pick y place de las cajas. En las siguientes figuras se muestran las fotos de cada uno de los agarres de la ventosa del robot a las diferentes cajas (Imágenes 1 y 2). En estas fotos se puede observar si el centroide y el ángulo calculados son correctos y si la caja se va a coger bien o no. Como se observa, el pick tanto por el centroide como por el ángulo prácticamente en todos los casos es el correcto, y la caja se absorbería bien por la ventosa.

En la tercera de las imágenes, se muestran las fotos de la posición en la que se sueltan las diferentes cajas. Si el pick ha sido correcto y la caja ha sido reconocida bien la posición de recepción será la correcta, sino puede ocurrir que la caja se aplaste ligeramente en el momento de recepción o que se deje demasiado lejos del suelo.

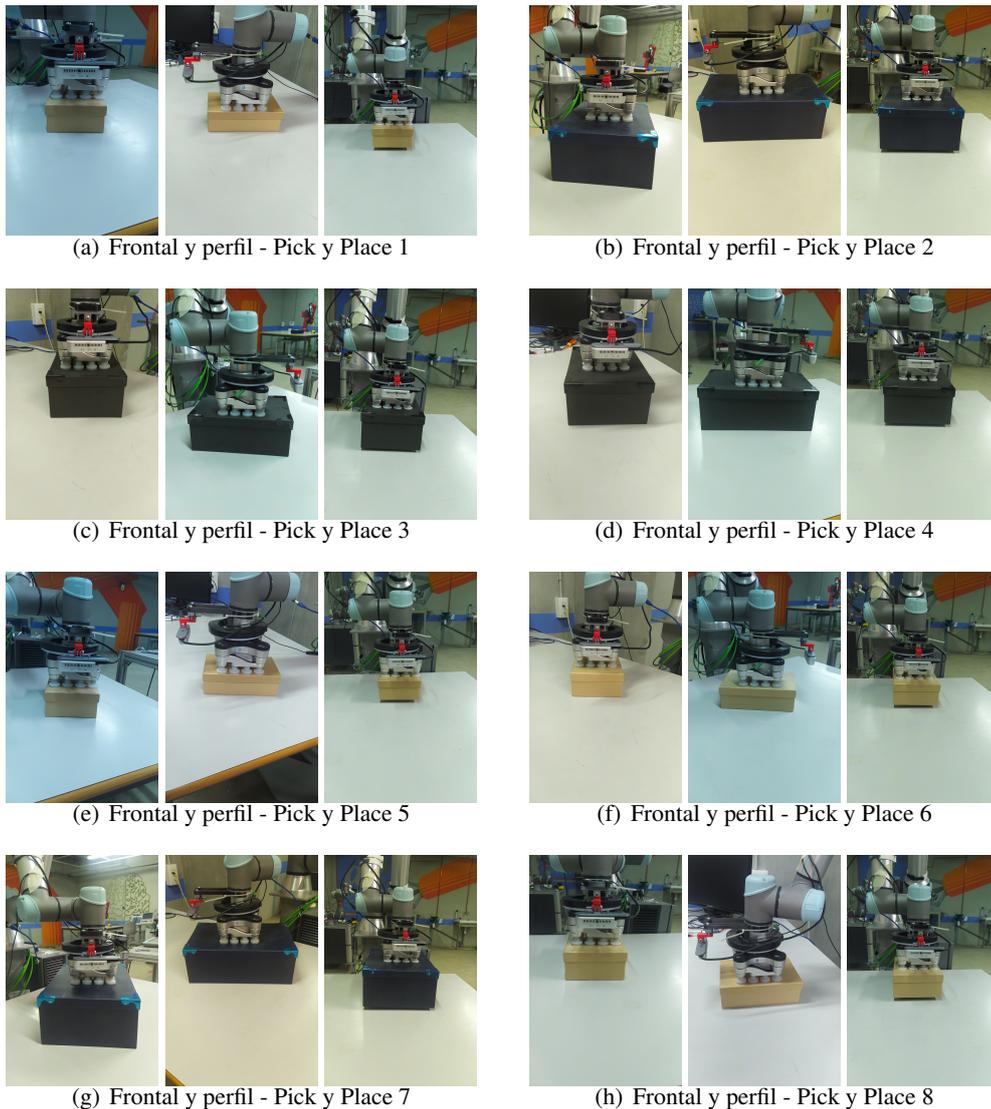
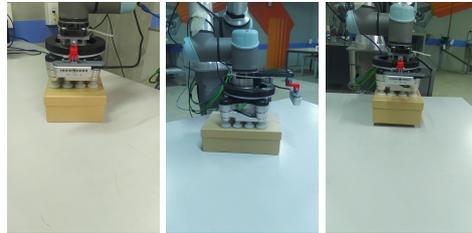


Figura 4.15: Fotos de los 25 pick y place (1)



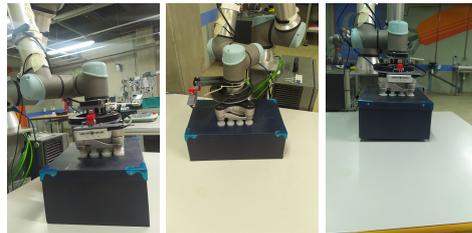
(a) Frontal y perfil - Pick y Place 9



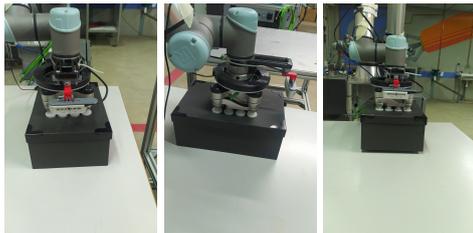
(b) Frontal y perfil - Pick y Place 10



(c) Frontal y perfil - Pick y Place 11



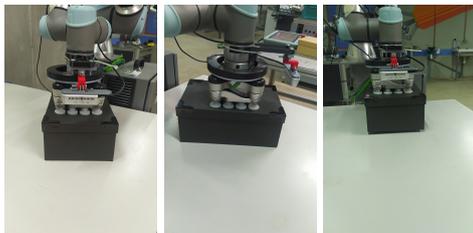
(d) Frontal y perfil - Pick y Place 12



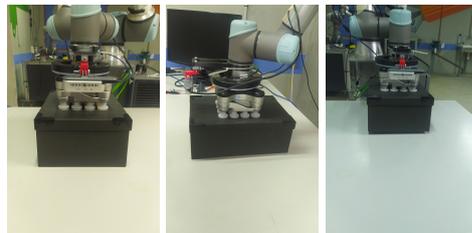
(e) Frontal y perfil - Pick y Place 13



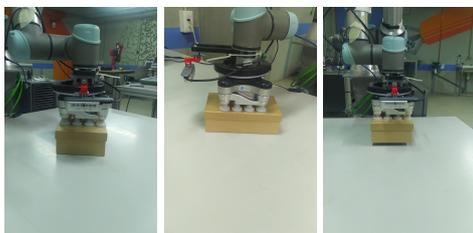
(f) Frontal y perfil - Pick y Place 14



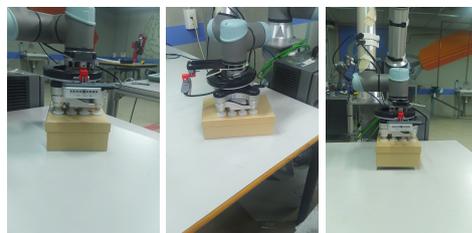
(g) Frontal y perfil - Pick y Place 15



(h) Frontal y perfil - Pick y Place 16



(i) Frontal y perfil - Pick y Place 17



(j) Frontal y perfil - Pick y Place 18

Figura 4.16: Fotos de los 25 pick y place (2)

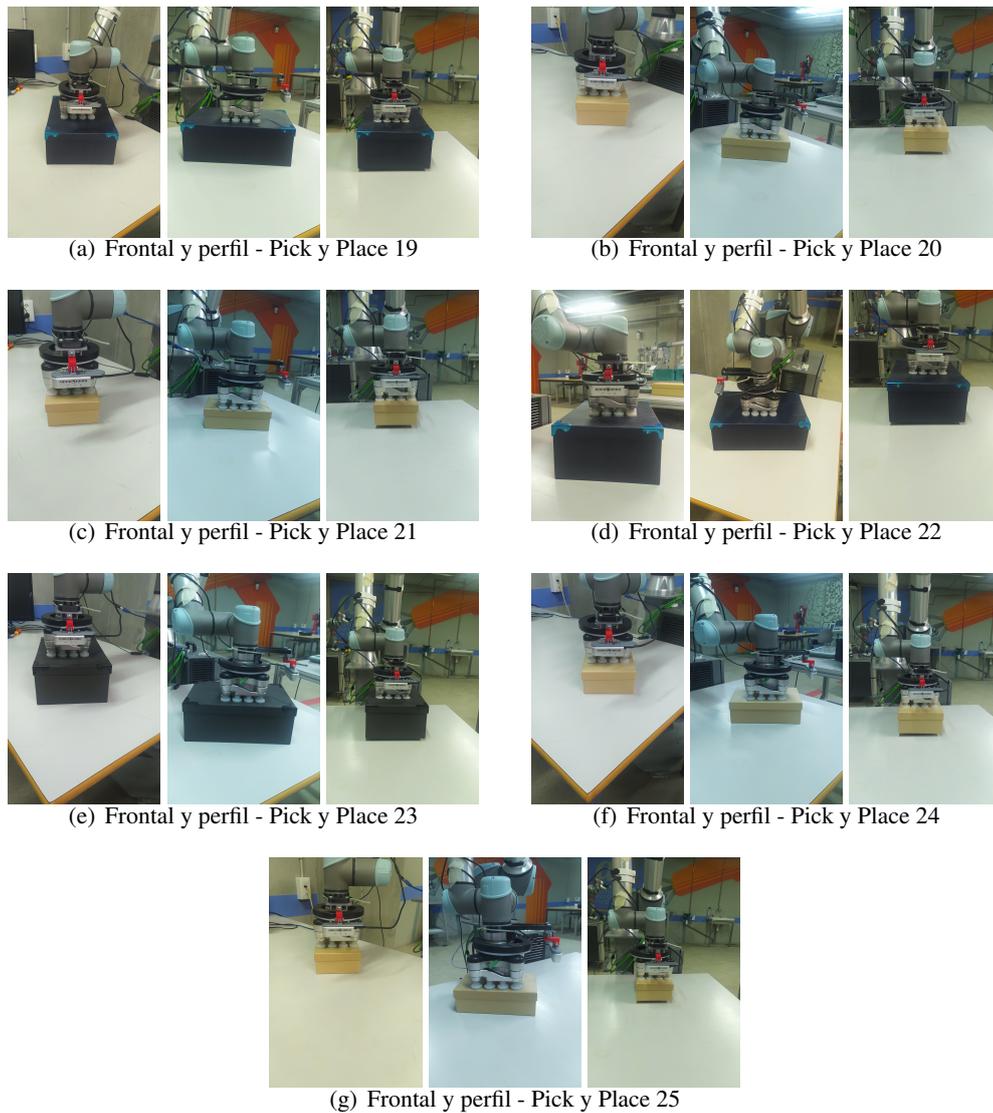


Figura 4.17: Fotos de los 25 pick y place (3)

Capítulo 5

Conclusiones y trabajos futuros

En este capítulo se muestran las conclusiones obtenidas de este Trabajo de Fin de Máster y los posibles trabajos futuros a realizar, relacionados con nuestro estudio.

5.1. Conclusiones

En este Trabajo de Fin de Máster se han logrado muchos de los objetivos que se habían fijado en la elaboración del mismo.

Primero, se ha conseguido probar y elaborar un software para la segmentación de objetos por color que da grandes resultados y funciona bastante bien para cajas de diferentes colores y tamaños, por tanto, se puede deducir que este código para la segmentación daría buenos resultados para otro tipo de cajas o objetos diferentes a los utilizados en el proyecto.

También, se ha podido conocer y experimentar como funcionan las cámaras RGB-D, más concretamente, la cámara *Intel RealSense D435* utilizado en el trabajo, pudiendo comprobar como le afectan las sombras, los reflejos, los diferentes colores y alturas en los datos recibidos y como poder compensar esos problemas mediante el software, utilizando filtros, y mediante el hardware, ajustando el lugar de la cámara.

Además, la elaboración de un código que podía conocer de manera aproximada la posición de las esquinas de las cajas, para así saber la orientación de la misma, era un aspecto conflictivo y problemático en este proyecto, por lo explicado en los capítulos anteriores (sombras, reflejos...), aún así, se podría decir que los resultados experimentales son bastante correctos en general.

Otro aspecto a destacar, es el reconocimiento del tipo de caja con los valores de la base de datos, que dio buenos resultados utilizando solo el dato de la medida de las alturas. También, como los códigos realizados para trabajar con una base de datos de *PostgreSQL* en *Python* se podrían utilizar para proyectos futuros que necesiten de un mayor uso del conocimiento de los parámetros reales del objeto.

Por último, se podría concluir que el objeto principal de este proyecto, de conseguir un software robusto que permita realizar el picking and place de cajas, ha sido cubierto de manera correcta como demuestran los resultados obtenidos.

5.2. Trabajos futuros

En este proyecto se ha tratado de conseguir un software que realice el picking and place de un conjunto de cajas. El mundo del picking and place utilizando visión artificial es un sector emergente y con avances continuos, por tanto son bastantes las líneas de investigación que se pueden seguir. Aquí he representado algunas que se podrían realizar en el futuro y que están muy relacionadas con este proyecto.

- Utilizar la base de datos obtenida para el picking de las cajas y no solo para el place como se hace en este trabajo. Una manera para realizar esto podría ser marcar las esquinas con otro color para realizar solo el reconocimiento de las esquinas y no del objeto entero. Esto permitiría el reconocimiento de las cajas a través de su largo y ancho, además de por su altura como se hace en este proyecto, y así tener un reconocimiento más robusto y, también, poder reconocer cajas de la misma altura pero de diferentes dimensiones. También, daría una mayor robustez en la obtención del ángulo, que como se ha comprobado puede tener fallos de medida por culpa de reflejos o sombras.
- Realizar el picking de dos cajas a la vez, pudiendo realizar una selección tanto manual de la caja que se quiera, como automática dando prioridad a las cajas más grandes o más pequeñas.
- Trabajar sobre este proyecto añadiéndole más cajas de diferentes dimensiones o colores o, incluso, diferentes formas en su base (cajas circulares, triangulares...)

Apéndice A

Hojas de características

A.1. Robot de *Universal Robot UR5*

Hoja de características del robot de *Universal Robot UR5*.



UR5 Technical specifications

Item no. 110105

6-axis robot arm with a working radius of 850 mm / 33.5 in

Weight:	18.4 kg / 40.6 lbs		
Payload:	5 kg / 11 lbs		
Reach:	850 mm / 33.5 in		
Joint ranges:	+/- 360°		
Speed:	All joints: 180°/s. Tool: Typical 1 m/s. / 39.4 in/s.		
Repeatability:	+/- 0.1 mm / +/- 0.0039 in (4 mils)		
Footprint:	Ø149 mm / 5.9 in		
Degrees of freedom:	6 rotating joints		
Control box size (WxHxD):	475 mm x 423 mm x 268 mm / 18.7 x 16.7 x 10.6 in		
I/O ports:		Controlbox	Tool conn.
	Digital in	16	2
	Digital out	16	2
	Analog in	2	2
	Analog out	2	-
I/O power supply:	24 V 2A in control box and 12 V/24 V 600 mA in tool		
Communication:	TCP/IP 100 Mbit: IEEE 802.3u, 100BASE-TX Ethernet socket & Modbus TCP		
Programming:	Polyscope graphical user interface on 12 inch touchscreen with mounting		
Noise:	Comparatively noiseless		
IP classification:	IP54		
ISO Class Cleanroom robot arm:	5		
ISO Class Cleanroom control box:	6		
Power consumption:	Approx. 200 watts using a typical program		
Collaboration operation:	15 Advanced Safety Functions Tested in accordance with: EN ISO 13849:2008 PL d EN ISO 10218-1:2011, Clause 5.4.3		
Materials:	Aluminum, PP plastic		
Temperature:	The robot can work in a temperature range of 0-50°C		
Power supply:	100-240 VAC, 50-60 Hz		
Cabling:	Cable between robot and control box (6 m / 236 in) Cable between touchscreen and control box (4.5 m / 177 in)		

Universal Robots A/S
Energivej 25
DK-5260 Odense S
Denmark
+45 89 93 89 89

www.universal-robots.com
sales@universal-robots.com



A.2. Cámara de visión con profundidad *Intel RealSense D435*

Hoja de características de la cámara de visión con profundidad *Intel RealSense D435*.



PRODUCT BRIEF

INTEL® REALSENSE™ DEPTH CAMERA D435

Powerful, Full-featured Depth Camera

Combined Solution for Development & Production

By introducing the Intel® RealSense™ Depth Camera D435 into the Intel® RealSense™ product lineup, Intel continues our commitment to developing cutting-edge new vision sensing products. Placing an Intel module and vision processor into a small form factor results in a combined solution ideal for development or productization. Lightweight, powerful, and low-cost, this complete package pairs with customizable software to enable the development of next-generation sensing solutions and devices that can understand and interact with their surroundings.

Ideal for Low Light and Wide Field of View

The D435 as a wide field of view solution using global shutter sensors. The combination of a wide field of view and global shutter sensor on the D435 make it the preferred solution for applications such as robotic navigation and object recognition. The wider field of view allows a single camera to cover more area resulting in less "blind spots". The global shutter sensors provide great low-light sensitivity allowing robots to navigate spaces with the lights off.

Complete Suite for Simple Integration

The Intel® RealSense™ Camera D435 is part of the Intel® RealSense™ 400 Series of cameras, a lineup that takes Intel's latest depth-sensing hardware and software offerings and puts them into easy-to-integrate, packaged products. Perfect for developers, makers, and innovators looking to bring depth-sensing vision to devices, Intel® RealSense™ 400 Series Cameras offer simple out-of-the-box integration and enable a whole new generation of intelligent vision-equipped devices.

D435



FEATURES

Use Environment: Indoor/Outdoor

Depth Technology: Active IR Stereo

Image Sensor Technology: Global Shutter; 3µm x 3µm pixel size

Depth Field of View (FOV)—(Horizontal x Vertical) for HD 16:9: 85.2° x 58° (+/- 3°)

Depth Output Resolution & Frame Rate: Up to 1280 x 720 active stereo depth resolution. Up to 90fps

Minimum Depth Distance (Min-Z): 0.105m

Maximum Range: 10m+. Varies depending on performance accuracy, scene and light conditions

RGB Resolution: Up to 1920 x 1080 resolution

RGB FOV (H x V x D): 69.4 x 42.5 x 77 (+/- 3°)

MAJOR COMPONENTS

Camera Module: Intel® RealSense™ Module D430 + RGB Camera

Vision Processor Board: Intel® RealSense™ Vision Processor D4

Copyright © 2017 Intel Corporation. All rights reserved. Intel, the Intel logo and Intel RealSense are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

PHYSICAL

Form Factor: Camera Peripheral

Connectors: USB 3 Type-C

Length x Depth x Height: 90mm x 25mm x 25mm

Apéndice B

Código completo del software

En este anexo se encuentra todo el código del proyecto.

B.1. Objetos

B.1.1. Objeto *ur_hadle*

```

1  import rtde_io
2  import socket
3
4  class ur_handle:
5      def __init__(self, HOST):
6          self.host = HOST
7          self.rtde_io_ = rtde_io.RTDEIOInterface(HOST)
8
9      def grip(self):
10         self.rtde_io_.setToolDigitalOut(0, True)
11         self.rtde_io_.setToolDigitalOut(1, True)
12
13     def release(self):
14         self.rtde_io_.setToolDigitalOut(0, False)
15         self.rtde_io_.setToolDigitalOut(1, False)
16
17     def socket_moveL(self, target_tcp, tool_vel, tool_acc):
18         tcp_command = "movel(p[%f,%f,%f,%f,%f,%f],a=%f,v=%f,t=0,r=0)\n" %
19             ↪ (target_tcp[0], target_tcp[1], target_tcp[2], target_tcp[3],
20             ↪ target_tcp[4], target_tcp[5], tool_acc, tool_vel)
21         tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
22         tcp_socket.connect((self.host, 30003))
23         tcp_socket.send(str.encode(tcp_command))
24         tcp_socket.close()

```

B.1.2. Objeto *Database*

```

1  from psycopg2 import pool
2
3  class Database:
4      __connection_pool = None
5
6      @classmethod
7      def initialise(cls, **datos_postgreSQL):
8          cls.__connection_pool = pool.SimpleConnectionPool(1, 1,
9             ↪ **datos_postgreSQL)
10
11     @classmethod
12     def get_connection(cls):
13         return cls.__connection_pool.getconn()
14
15     @classmethod
16     def return_connection(cls, connection):
17         cls.__connection_pool.putconn(connection)

```

```

17
18     @classmethod
19     def close_all_connection(cls):
20         cls.__connection_pool.closeall()
21
22     class CursorFromConnectionFromPool:
23         def __init__(self):
24             self.connection = None
25             self.cursor = None
26
27         def __enter__(self):
28             self.connection = Database.get_connection()
29             self.cursor = self.connection.cursor()
30             return self.cursor
31
32         def __exit__(self, exception_type, exception_value, exception_traceback):
33             if exception_value is not None:
34                 self.connection.rollback()
35             else:
36                 self.cursor.close()
37                 self.connection.commit()
38             Database.return_connection(self.connection)

```

B.2. Código principal

B.2.1. Librerías y objetos

```

1  import os
2  import time
3  import pyrealsense2 as rs
4  import cv2 as cv
5  import numpy as np
6  from ur5e import ur_handle
7  import math
8  from database import CursorFromConnectionFromPool, Database

```

B.2.2. Funciones creadas

go.home

```

1  def go_home():
2      TCP_pose = [0.12, -0.38, 0.45, 3.139, -0.016, -0.004]
3      ur.socket_moveL(TCP_pose, 0.15, 0.2)
4      time.sleep(4)
5      return TCP_pose

```

start_camara y obtain_data

```

1  def start_camara():
2      results = []
3
4      pc = rs.pointcloud()
5      points = rs.points()
6      pipeline = rs.pipeline()
7      results.append(pc)
8      results.append(points)
9      results.append(pipeline)
10
11     config = rs.config()
12     config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)
13     config.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)
14
15     pipeline.start(config)
16
17     align_to = rs.stream.color
18     align = rs.align(align_to)
19     results.append(align)
20     return results

```

```

1  def obtain_data(pipeline, align, pc):
2      frame_verts = []
3      for x in range(10):
4          pipeline.wait_for_frames()
5          frames = pipeline.wait_for_frames()
6          frame_data = frames.get_color_frame()
7          frame = np.asanyarray(frame_data.get_data())
8          frame_verts.append(frame)
9
10         aligned_frames = align.process(frames)
11         aligned_color_frame = aligned_frames.get_color_frame()
12         aligned_depth_frame = aligned_frames.get_depth_frame()
13
14         imagen_objeto = np.asanyarray(aligned_color_frame.get_data())
15
16         #cv.imshow('Imagen Objeto', imagen_objeto)
17         #cv.waitKey(0)
18         #cv.destroyAllWindows()
19
20         pc.map_to(aligned_color_frame)
21         points = pc.calculate(aligned_depth_frame)
22         w = rs.video_frame(aligned_depth_frame).width
23         h = rs.video_frame(aligned_depth_frame).height
24         verts = np.asanyarray(points.get_vertices()).view(np.float32).reshape(h, w,
↪ 3)
25         frame_verts.append(verts)
26

```

```

27     pipeline.stop()
28
29     return frame_verts, imagen_objeto

```

segmentacion

```

1  def segmentacion(imagen_objeto):
2      hsv = cv.cvtColor(imagen_objeto, cv.COLOR_BGR2HSV)
3      #cv.imshow('hsv', hsv)
4      #cv.waitKey(0)
5      #cv.destroyAllWindows()
6
7      lower_azul = np.array([75, 60, 100])
8      upper_azul = np.array([100, 150, 255])
9
10     mask = cv.inRange(hsv, lower_azul, upper_azul)
11     kernel = np.ones((5, 5), np.uint8)
12     niters = 3
13     mask = cv.dilate(mask, kernel, iterations=niters)
14     mask = cv.erode(mask, kernel, iterations=niters * 2)
15     mask = cv.dilate(mask, kernel, iterations=niters)
16
17     #cv.imshow('Mascara', mask)
18     #cv.waitKey(0)
19     #cv.destroyAllWindows()
20
21     mask1 = cv.bitwise_not(mask)
22
23     numLabels, labels, stats, centroids = cv.connectedComponentsWithStats(mask1)
24
25     #cv.circle(imagen_objeto, (int(centroids[1][0]), int(centroids[1][1])), 4,
26     ↪ # (0, 0, 255), -1)
27     #cv.rectangle(imagen_objeto, (stats[1][0], stats[1][1]), (stats[1][0] +
28     ↪ #stats[1][2], stats[1][1] + stats[1][3]), (0, 255, 0), 3)
29
30     contornos, herencia = cv.findContours(mask1, cv.RETR_EXTERNAL,
31     ↪ cv.CHAIN_APPROX_SIMPLE)
32
33     #cv.drawContours(imagen_objeto, contornos, -1, (255, 0, 0), 2)
34
35     #cv.imshow('Centroides y Contornos', imagen_objeto)
36     #cv.waitKey(0)
37     #cv.destroyAllWindows()
38
39     vector_stats = [stats[1][0], stats[1][1], stats[1][0] + stats[1][2],
40     ↪ stats[1][1] + stats[1][3]]
41     vector_centroides = [int(centroids[1][0]), int(centroids[1][1])]

```

```
39     return imagen_objeto, vector_stats, vector_centroides, contornos
```

encontrar_obj.borde, move.left, move.right, move.down y move.up

```
1  def encontrar_obj_borde(imagen_objeto, frame_verts, vector_stats,
   ↪ vector_centroides, contornos, TCP_pose):
2
3     while ((vector_stats[0] == 0) or (vector_stats[2] == 640) or
4            (vector_stats[1] == 0) or (vector_stats[3] == 480)):
5
6         if vector_stats[0] == 0:
7             print('Objeto a la izquierda')
8             TCP_pose = move_left(TCP_pose)
9
10        elif vector_stats[2] == 640:
11            print('Objeto a la derecha')
12            TCP_pose = move_right(TCP_pose)
13
14        elif vector_stats[1] == 0:
15            print('Objeto abajo')
16            TCP_pose = move_down(TCP_pose)
17
18        elif vector_stats[3] == 480:
19            print('Objeto arriba')
20            TCP_pose = move_up(TCP_pose)
21
22        results = start_camera()
23        frame_verts, imagen_objeto = obtain_data(results[2], results[3],
   ↪ results[0])
24
25        imagen_objeto, vector_stats, vector_centroides, contornos =
   ↪ segmentacion(imagen_objeto)
26
27    return imagen_objeto, frame_verts, vector_stats, vector_centroides,
   ↪ contornos, TCP_pose
```

```
1  def move_left(TCP_pose):
2
3     TCP_pose[0] -= 0.05
4     ur.socket_moveL(TCP_pose, 0.15, 0.2)
5     time.sleep(3)
6     return TCP_pose
7
8  def move_right(TCP_pose):
9
10    TCP_pose[0] += 0.02
11    ur.socket_moveL(TCP_pose, 0.15, 0.2)
```

```

12     time.sleep(3)
13     return TCP_pose
14
15     def move_down(TCP_pose):
16
17         TCP_pose[1] += 0.02
18         ur.socket_moveL(TCP_pose, 0.15, 0.2)
19         time.sleep(3)
20         return TCP_pose
21
22     def move_up(TCP_pose):
23
24         TCP_pose[1] -= 0.05
25         ur.socket_moveL(TCP_pose, 0.15, 0.2)
26         time.sleep(3)
27         return TCP_pose

```

encontrar_angulo

```

1     def encontrar_angulo(imagen_objeto, vector_stats, contornos):
2         x_mayor = 0
3         x_mayor1 = 0
4         y_mayor = 0
5         y_mayor1 = 0
6         x_menor = 640
7         x_menor1 = 640
8         y_menor = 480
9         y_menor1 = 480
10        posicion_y_min = 0
11        posicion_y_min1 = 0
12        posicion_x_max = 0
13        posicion_x_max1 = 0
14        posicion_x_min = 0
15        posicion_x_min1 = 0
16        posicion_y_max = 0
17        posicion_y_max1 = 0
18
19        for j in contornos:
20            for k in range(len(j)):
21
22                if j[k][0][1] == vector_stats[3] or j[k][0][1] == vector_stats[3] -
23                    ↪ 1 or j[k][0][1] == vector_stats[3] + 1 or j[k][0][1] ==
24                    ↪ vector_stats[3] - 2 or j[k][0][1] == vector_stats[3] + 2:
25                    if j[k][0][0] > x_mayor:
26                        x_mayor = j[k][0][0]
27                        posicion_y_min = k
28
29                    if j[k][0][0] < x_menor:

```

```

28         x_menor = j[k][0][0]
29         posicion_y_min1 = k
30
31         if j[k][0][0] == vector_stats[2] or j[k][0][0] == vector_stats[2] -
↪ 1 or j[k][0][0] == vector_stats[2] + 1 or j[k][0][0] ==
↪ vector_stats[2] - 2 or j[k][0][0] == vector_stats[2] + 2:
32
33             if j[k][0][1] > y_mayor:
34                 y_mayor = j[k][0][1]
35                 posicion_x_max = k
36
37             if j[k][0][1] < y_menor1:
38                 y_menor1 = j[k][0][1]
39                 posicion_x_max1 = k
40
41         if j[k][0][0] == vector_stats[0] or j[k][0][0] == vector_stats[0] -
↪ 1 or j[k][0][0] == vector_stats[0] + 1 or j[k][0][0] ==
↪ vector_stats[0] - 2 or j[k][0][0] == vector_stats[0] + 2:
42
43             if j[k][0][1] > y_mayor1:
44                 y_mayor1 = j[k][0][1]
45                 posicion_x_min = k
46
47             if j[k][0][1] < y_menor:
48                 y_menor = j[k][0][1]
49                 posicion_x_min1 = k
50
51         if j[k][0][1] == vector_stats[1] or j[k][0][1] == vector_stats[1] -
↪ 1 or j[k][0][1] == vector_stats[1] + 1 or j[k][0][1] ==
↪ vector_stats[1] - 2 or j[k][0][1] == vector_stats[1] + 2:
52
53             if j[k][0][0] > x_mayor1:
54                 x_mayor1 = j[k][0][0]
55                 posicion_y_max = k
56
57             if j[k][0][0] < x_menor1:
58                 x_menor1 = j[k][0][0]
59                 posicion_y_max1 = k
60
61
62         #cv.line(imagen_objeto, (j[posicion_y_min][0][0], #j[posicion_y_min][0][1]),
↪ (j[posicion_x_max][0][0], j[posicion_x_max][0][1]), #(255, 255, 255), 3)
63         #cv.line(imagen_objeto, (j[posicion_y_min][0][0], #j[posicion_y_min][0][1]),
↪ (j[posicion_x_max][0][0], j[posicion_y_min][0][1]), # (0, 0, 0), 3)
64
65         #cv.line(imagen_objeto, (j[posicion_x_max][0][0], #j[posicion_x_max][0][1]),
↪ (j[posicion_x_max1][0][0], #j[posicion_x_max1][0][1]), (255, 0, 255), 3)
66
67         #cv.line(imagen_objeto, (j[posicion_y_min1][0][0],
↪ #j[posicion_y_min1][0][1]),(j[posicion_x_min][0][0],
↪ j[posicion_x_min][0][1]), # (255, 255, 255), 3)

```

```

68
69     #cv.line(imagen_objeto, (j[posicion_y_min1][0][0],
    ↪ #j[posicion_y_min1][0][1]),(j[posicion_y_min][0][0],
    ↪ j[posicion_y_min][0][1]), #(255, 0, 255), 3)
70
71     #cv.line(imagen_objeto, (j[posicion_y_max1][0][0],
    ↪ #j[posicion_y_max1][0][1]),(j[posicion_x_min1][0][0],
    ↪ #j[posicion_x_min1][0][1]), (255, 255, 255), 3)
72
73     #cv.line(imagen_objeto, (j[posicion_x_min][0][0], #j[posicion_x_min][0][1]),
    ↪ (j[posicion_x_min1][0][0], #j[posicion_x_min1][0][1]), (255, 0, 255), 3)
74
75     #cv.line(imagen_objeto, (j[posicion_y_max][0][0], #j[posicion_y_max][0][1]),
    ↪ (j[posicion_x_max1][0][0], #j[posicion_x_max1][0][1]), (255, 255, 255),
    ↪ 3)
76
77     #cv.line(imagen_objeto, (j[posicion_y_max][0][0], #j[posicion_y_max][0][1]),
    ↪ (j[posicion_y_max1][0][0], #j[posicion_y_max1][0][1]), (255, 0, 255), 3)
78
79     #cv.circle(imagen_objeto, (j[posicion_y_min][0][0],
    ↪ #j[posicion_y_min][0][1]), 4, (0, 0, 255), -1)
80
81     #cv.circle(imagen_objeto, (j[posicion_x_max][0][0],
    ↪ #j[posicion_x_max][0][1]), 4, (0, 0, 255), -1)
82
83     #cv.circle(imagen_objeto, (j[posicion_y_min1][0][0],
    ↪ #j[posicion_y_min1][0][1]), 4, (0, 0, 255), -1)
84
85     #cv.circle(imagen_objeto, (j[posicion_x_min][0][0],
    ↪ #j[posicion_x_min][0][1]), 4, (0, 0, 255), -1)
86
87     #cv.circle(imagen_objeto, (j[posicion_x_min1][0][0],
    ↪ #j[posicion_x_min1][0][1]), 4, (0, 0, 255), -1)
88
89     #cv.circle(imagen_objeto, (j[posicion_y_max1][0][0],
    ↪ #j[posicion_y_max1][0][1]), 4, (0, 0, 255), -1)
90
91     #cv.circle(imagen_objeto, (j[posicion_y_max][0][0],
    ↪ #j[posicion_y_max][0][1]), 4, (0, 0, 255), -1)
92
93     #cv.circle(imagen_objeto, (j[posicion_x_max1][0][0],
    ↪ #j[posicion_x_max1][0][1]), 4, (0, 0, 255), -1)
94
95     #cv.imshow('Angulo', imagen_objeto)
96     #cv.waitKey(0)
97     #cv.destroyAllWindows()
98
99     long_lado_xmax = abs(j[posicion_x_max][0][1] - j[posicion_x_max1][0][1])
100     print('Lado xmax: \n', long_lado_xmax)
101     long_lado_ymin = abs(j[posicion_y_min][0][0] - j[posicion_y_min1][0][0])
102     print('Lado ymin: \n', long_lado_ymin)

```

```

103     long_lado_xmin = abs(j[posicion_x_min][0][1] - j[posicion_x_min1][0][1])
104     print('Lado xmin: \n', long_lado_xmin)
105     long_lado_ymax = abs(j[posicion_y_max][0][0] - j[posicion_y_max1][0][0])
106     print('Lado ymax: \n', long_lado_ymax)
107
108     l_cos = j[posicion_x_max][0][0] - j[posicion_y_min][0][0]
109     l_sin = j[posicion_y_min][0][1] - j[posicion_x_max][0][1]
110     lado1 = math.sqrt(l_sin * l_sin + l_cos * l_cos)
111     l_cos1 = j[posicion_y_min][0][0] - j[posicion_x_min][0][0]
112     l_sin1 = j[posicion_y_min][0][1] - j[posicion_x_min][0][1]
113     lado2 = math.sqrt(l_sin1 * l_sin1 + l_cos1 * l_cos1)
114
115     print('Coseno: \n', l_cos)
116     print('Seno: \n', l_sin)
117
118     if ((long_lado_xmax > 50) and (long_lado_ymin > 50)) or \
119         ((long_lado_xmax > 50) and (long_lado_xmin > 50)) or \
120         ((long_lado_xmax > 50) and (long_lado_ymax > 50)) or \
121         ((long_lado_ymin > 50) and (long_lado_xmin > 50)) or \
122         ((long_lado_ymin > 50) and (long_lado_ymax > 50)) or \
123         ((long_lado_xmin > 50) and (long_lado_ymax > 50)):
124         angulo = 0
125         print('Angulo: \n', angulo)
126
127     elif lado1 >= lado2:
128         angulo = np.arctan(l_cos / (-l_sin))
129         print(angulo)
130
131     else:
132         angulo = np.arctan(l_sin / l_cos)
133         print(angulo)
134
135     return angulo

```

centroide.xyz

```

1     def centroide_xyz(vector_centroides, frame_verts):
2
3         centroide_x = vector_centroides[0]
4         centroide_y = vector_centroides[1]
5
6         frame = frame_verts[0]
7         verts = frame_verts[1]
8
9         xyz = verts[centroide_y, centroide_x, :]
10
11     while xyz[0] == 0 and xyz[1] == 0 and xyz[2] == 0:
12         print('No encuentra el centroide')

```

```

13     results = start_camera()
14     frame_verts = obtain_data(results[2], results[3], results[0])
15
16     frame = frame_verts[0]
17     verts = frame_verts[1]
18
19     xyz = verts[centroide_y, centroide_x, :]
20
21     translation_vector = [xyz[0], xyz[1], xyz[2]]
22     print('Vector de translación: \n', translation_vector)
23
24     return translation_vector

```

pick

```

1  def pick(TCP_pose, translation_vector):
2      TCP_pose[0] += (translation_vector[0])
3
4      TCP_pose[1] -= (translation_vector[1])
5
6      TCP_pose[2] -= 0.3
7
8      ur.socket_moveL(TCP_pose, 0.15, 0.2)
9      time.sleep(5)
10     print('TCP_pose: \n', TCP_pose)
11     return TCP_pose

```

conocer.tipo.caja

```

1  def conocer_tipo_caja(altura_medida):
2      altura_medida = altura_medida*1000
3      altura_camara = 0.412 * 1000
4      altura_min = altura_camara
5      altura_caja = altura_camara - altura_medida
6      print('Altura caja: \n', altura_caja)
7
8      with CursorFromConnectionFromPool() as cursor:
9          for i in range(1, 6):
10             cursor.execute('SELECT * FROM cajas WHERE id=%s', (i,))
11             row = cursor.fetchone()
12             distancia_altura = abs(row[4] - altura_caja)
13             if distancia_altura < altura_min:
14                 altura_min = distancia_altura
15                 tipo_caja = row[1]
16                 altura_caja_mm = row[4]
17

```

```

18     print('Caja: \n', tipo_caja)
19     return tipo_caja, altura_caja_mm

```

VRotateToRPY

```

1  def VRotateToRPY(rx, ry, rz):
2      theta = math.sqrt((rx * rx) + (ry * ry) + (rz * rz))
3      kx = rx / theta
4      ky = ry / theta
5      kz = rz / theta
6      cth = math.cos(theta)
7      sth = math.sin(theta)
8      vth = 1 - math.cos(theta)
9
10     r11 = kx * kx * vth + cth
11     r12 = kx * ky * vth - kz * sth
12     r13 = kx * kz * vth + ky * sth
13     r21 = kx * ky * vth + kz * sth
14     r22 = ky * ky * vth + cth
15     r23 = ky * kz * vth - kx * sth
16     r31 = kx * kz * vth - ky * sth
17     r32 = ky * kz * vth + kx * sth
18     r33 = kz * kz * vth + cth
19
20     beta = math.atan2(-r31, math.sqrt(r11 * r11 + r21 * r21))
21
22     if beta > (math.pi / 2):
23         beta = math.pi / 2
24         alpha = 0
25         gamma = math.atan2(r12, r22)
26     else:
27         if beta < (-math.pi / 2):
28             beta = -math.pi / 2
29             alpha = 0
30             gamma = -math.atan2(r12, r22)
31         else:
32             cb = math.cos(beta)
33             alpha = math.atan2(r21 / cb, r11 / cb)
34             gamma = math.atan2(r32 / cb, r33 / cb)
35     rpy = [[gamma], [beta], [alpha]]
36     return rpy

```

rpyToVRotate

```

1  def rpyToVRotate(R, P, Y):
2      rx = np.matrix([[1, 0, 0],

```

```

3             [0, math.cos(R), -math.sin(R)],
4             [0, math.sin(R), math.cos(R)]]
5     ry = np.matrix([[math.cos(P), 0, math.sin(P)],
6                    [0, 1, 0],
7                    [-math.sin(P), 0, math.cos(P)]]
8     rz = np.matrix([[math.cos(Y), -math.sin(Y), 0],
9                    [math.sin(Y), math.cos(Y), 0],
10                   [0, 0, 1]])
11
12     RotM = rz * ry * rx
13
14     theta = math.acos((RotM[0, 0] + RotM[1, 1] + RotM[2, 2] - 1)/2)
15     multi = 1 / (2*math.sin(theta))
16
17     u = [[multi * (RotM[2, 1] - RotM[1, 2]) * theta],
18          [multi * (RotM[0, 2] - RotM[2, 0]) * theta],
19          [multi * (RotM[1, 0] - RotM[0, 1]) * theta]]
20     print('Vector de rotación para el giro: \n', u)
21     return u

```

giro.en.z

```

1     def giro_en_z(TCP_pose, vx, vy, vz):
2         TCP_pose[3] = vx[0]
3         TCP_pose[4] = vy[0]
4         TCP_pose[5] = vz[0]
5         ur.socket_moveL(TCP_pose, 0.15, 0.2)
6         time.sleep(3)
7
8         return TCP_pose

```

pick1

```

1     def pick1(TCP_pose, translation_vector):
2         TCP_pose[0] -= 0.04
3
4         TCP_pose[1] -= 0.165
5
6         TCP_pose[2] -= translation_vector[2] - 0.073
7
8         ur.socket_moveL(TCP_pose, 0.15, 0.2)
9         time.sleep(5)
10        print('TCP_pose: \n', TCP_pose)
11
12        # GRIP

```

```
13     ur.grip()
14     time.sleep(2)
```

place

```
1  def place(TCP_pose, tipo_caja, altura):
2      if tipo_caja == 'Caja 1':
3
4          TCP_pose[2] -= 0.63 - altura/1000
5          ur.socket_moveL(TCP_pose, 0.15, 0.2)
6          time.sleep(5)
7
8      elif tipo_caja == 'Caja 2':
9
10         TCP_pose[2] -= 0.63 - altura/1000
11         ur.socket_moveL(TCP_pose, 0.15, 0.2)
12         time.sleep(5)
13
14     elif tipo_caja == 'Caja 3':
15
16         TCP_pose[2] -= 0.63 - altura/1000
17         ur.socket_moveL(TCP_pose, 0.15, 0.2)
18         time.sleep(5)
19
20     elif tipo_caja == 'Caja 4':
21
22         TCP_pose[2] -= 0.63 - altura/1000
23         ur.socket_moveL(TCP_pose, 0.15, 0.2)
24         time.sleep(5)
25
26     elif tipo_caja == 'Caja 5':
27
28         TCP_pose[2] -= 0.63 - altura/1000
29         ur.socket_moveL(TCP_pose, 0.15, 0.2)
30         time.sleep(5)
31
32     # RELEASE
33     ur.release()
34     time.sleep(2)
```

B.2.3. Main

```
1  if __name__ == '__main__':
2
3      if os.getenv('ROBOT_ID') == None:
4          os.environ['ROBOT_ID'] = 'ur5'
```

```
5
6     ur = ur_handle("158.42.126.107")
7
8     Database.initialise(user='postgres', password='1234', database='TFM',
9         ↪ host='localhost')
10
11     while True:
12         TCP_pose = go_home()
13         results = start_camera()
14         frame_verts, imagen_objeto = obtain_data(results[2], results[3],
15             ↪ results[0])
16         imagen_objeto, vector_stats, vector_centroides, contornos =
17             ↪ segmentacion(imagen_objeto)
18         imagen_objeto, frame_verts, vector_stats, vector_centroides, contornos,
19             ↪ TCP_pose1 = encontrar_obj_borde(imagen_objeto, frame_verts,
20             ↪ vector_stats, vector_centroides, contornos, TCP_pose)
21         angulo = encontrar_angulo(imagen_objeto, vector_stats, contornos)
22         translation_vector_centroide = centroide_xyz(vector_centroides,
23             ↪ frame_verts)
24         TCP_pose2 = pick(TCP_pose1, translation_vector_centroide)
25         results = start_camera()
26         frame_verts, imagen_objeto = obtain_data(results[2], results[3],
27             ↪ results[0])
28         translation_vector_centroide = centroide_xyz([320, 240], frame_verts)
29         tipo_caja, altura_caja_mm =
30             ↪ conocer_tipo_caja(translation_vector_centroide[2])
31         v = VRotateToRPY(TCP_pose2[3], TCP_pose2[4], TCP_pose2[5])
32         u = rpyToVRotate(v[0][0], v[1][0], v[2][0]+angulo)
33         TCP_pose3 = giro_en_z(TCP_pose2, u[0], u[1], u[2])
34         pick1(TCP_pose3, translation_vector_centroide)
35         TCP_pose = go_home()
36         place(TCP_pose, tipo_caja, altura_caja_mm)
```


Apéndice C

Código adicional

Este anexo muestra el objeto adicional creado para trabajar con la Base de Datos de *PostgreSQL* desde *Python*.

C.1. Objeto creado para trabajar con la Base de Datos

```

1  from Database import CursorFromConnectionFromPool
2
3  class Cajas:
4      def __init__(self, id, nombre, largo, ancho, altura):
5          self.id = id
6          self.nombre = nombre
7          self.largo = largo
8          self.ancho = ancho
9          self.altura = altura
10
11     def __repr__(self):
12         return '<Nombre = {}, Largo = {}, Ancho = {}, Altura =
13         ↪ {}>'.format(self.nombre, self.largo, self.ancho, self.altura)
14
15     def save_to_db(self):
16         with CursorFromConnectionFromPool() as cursor:
17             cursor.execute('INSERT INTO cajas (id, nombre, largo, ancho, altura)
18             ↪ VALUES (%s, %s, %s, %s, %s)',
19                             (self.id, self.nombre, self.largo, self.ancho,
20                             ↪ self.altura))
21
22     def update_row_nombre(self):
23         with CursorFromConnectionFromPool() as cursor:
24             cursor.execute('UPDATE cajas SET largo = %s, ancho = %s, altura = %s
25             ↪ WHERE nombre = %s',
26                             (self.largo, self.ancho, self.altura, self.nombre))
27
28     @classmethod
29     def delete_row(cls, nombre):
30         with CursorFromConnectionFromPool() as cursor:
31             cursor.execute('DELETE FROM cajas WHERE nombre = %s', (nombre,))
32
33     @classmethod
34     def load_data_row(cls, nombre):
35         with CursorFromConnectionFromPool() as cursor:
36             cursor.execute('SELECT * FROM cajas WHERE nombre = %s', (nombre,))
37             cajas_data = cursor.fetchone()
38             return cls(id=cajas_data[0], nombre=cajas_data[1],
39                       ↪ largo=cajas_data[2], ancho=cajas_data[3],
40                       ↪ altura=cajas_data[4])

```

En el código lo primero que hay que tener en cuenta es importar la función *CursorFromConnectionFromPool* del objeto *Database* para realizar la conexión con la Base de Datos. Dentro del objeto creado se puede observar que se utilizan 5 constructores, que corresponden a las 5 columnas de la tabla, 2 funciones y 2 métodos de clase. Las dos funciones se utilizan crear nuevas filas ("save.to.db") o modificar valores de una fila ya existente (update.row.nombre"). Se debe tener en cuenta que la segunda de las funciones sirve para modificar los parámetros de las medidas de las cajas, no para cambiar el nombre ni el id. En cuanto a los métodos, el prime-

ro se puede utilizar para eliminar una fila de la tabla, pasándole como parámetro el nombre de la caja a eliminar, y el segundo permite cargar y mostrar los parámetros de la fila que se quiera, pasándole como parámetro el nombre al igual que en el caso anterior.

Bibliografía

- [1] A. Cordeiro, L. F. Rocha, C. Costa, P. Costa y S. M. S., «Bin Picking Approaches Based on Deep Learning Techniques: A State-of-the-Art Survey,» *IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2022.
- [2] «Detección de agarre robótico usando redes neuronales convolucionales,» *Instituto de Automática e Informática Industrial (AI2)*, jul. de 2020.
- [3] «Detección de objetos y clasificación de imágenes usando redes neuronales convolucionales,» *Instituto de Automática e Informática Industrial (AI2)*, nov. de 2020.
- [4] G. E. Hinton, «Deep belief networks,» *University of Toronto, Canada*, 2009.
- [5] S. Lakshmi y V. Sankaranarayanan, «A study of Edge Detection Techniques for Segmentation Computing Approaches,» *IJCA Special Issue on 'Computer Aided Soft Computing Techniques for Imaging and Biomedical Applications'*, CASCT, 2010.
- [6] Z. Li, F. Liu, W. Yang, S. Peng y J. Zhou, «A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects,» *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, págs. 6999-7019, dic. de 2022.
- [7] B. Maches, «Machine Learning Algorithms - A Review,» *International Journal of Science and Research (IJSR)*, 2020.
- [8] «Manual de Usuario UR5/CB3,» *Universal Robots*, 2017. dirección: https://www.cfzcobots.com/wp-content/uploads/2017/03/ur5_user_manual_es_global.pdf.
- [9] D. Marr, *A Computational Investigation into the Human Representation and Processing of Visual Information*. W. H. Freeman y Company, San Francisco, 1982.
- [10] L. G. Roberts, «Machine perception of three-dimensional solids,» *Thesis, Massachusetts Institute of Technology, Cambridge*, 1963.
- [11] W. M. Wichman, «Use of Optical Feedback in the Computer Control of an Arm,» *Department of Computer Science, Stanford University, California*, 1967.

