ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

“DISEÑO Y SIMULACIÓN DE UN SISTEMA AUTOMATIZADO BASADO EN VISIÓN POR COMPUTADOR PARA UNA ESTACIÓN DE PALETIZADO USANDO ROBOTS COLABORATIVOS”

PROYECTO INTEGRADOR

Previo la obtención del Título de:

Magister en Automatización y Control Industrial

Presentado por:

Benjamin Omar Holmes Cabezas

Tonny Wesley Toscano Quiroga

GUAYAQUIL - ECUADOR

Año: 2024

DEDICATORIA

El presente proyecto va dedicado a mi padre Benjamin, madre Angelica y mis hermanos Nicole, Nicolas y Carla, por su amor incondicional, su comprensión en los momentos difíciles y por estar siempre a mi lado, brindándome su apoyo inquebrantable a lo largo de mi crecimiento personal, espiritual y profesional.

Benjamin Holmes

AGRADECIMIENTOS

Mi más sincero agradecimiento a nuestro tutor el PhD Ángel Sappa y al profesor MSc. Carlos Salazar, por su guía paciencia y apoyo a lo largo de este proceso, su experiencia y conocimiento han sido claves para la culminación de este trabajo.

Finalmente, a todos aquellos que de alguna manera contribuyeron a la realización de esta tesis, les agradezco de todo corazón.

Benjamin Holmes

DEDICATORIA

El presente proyecto se lo dedico a mi bebé Gael, a mi esposa Andrea, a mi mamá Lida, a mis hermanos Viviana y Paúl, que han sido un apoyo incondicional para este gran logro, siendo pilares importantes en mi vida y en mi desarrollo tanto personal como profesional, y al cielo a mi papá que de donde sea que esté sé que me observa.

Tonny Toscano

AGRADECIMIENTOS

Mi más sincero agradecimiento a mi tutor el PhD Ángel Sappa, gran motivador, gran ser humano, excelente tutor y profesor que he tenido el agrado de conocer, y a todos los profesores en mi formación académica y profesional.

Agradecimiento también a las personas que han apoyado de una u otra forma para alcanzar este maravilloso logro.

Tonny Toscano

DECLARACIÓN EXPRESA

“Los derechos de titularidad y explotación, nos corresponde conforme al reglamento de propiedad intelectual de la institución; *Benjamin Omar Holmes Cabezas y Tonny Wesley Toscano Quiroga* y damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Firmado electrónicamente por:
BENJAMIN OMAR
HOLMES CABEZAS

Benjamin Omar Holmes Cabezas

Firmado electrónicamente por:
TONNY WESLEY
TOSCANO QUIROGA

Tonny Wesley Toscano Quiroga

EVALUADORES

Firmado electrónicamente por:
CARLOS ALBERTO SALAZAR LOPEZ

Ph.D Carlos Salazar

PROFESOR DE LA MATERIA

Firmado electrónicamente por:
ANGEL DOMINGO SAPPA .

Ph.D. Ángel Sappa

PROFESOR TUTOR

RESUMEN

En el presente proyecto, se desarrolla y simula una estación de paletizado automatizada utilizando RoboDK para optimizar el reconocimiento, clasificación, almacenamiento de datos y manipulación de productos catalogados previamente mediante códigos QR. Esta solución automatizada busca mejorar la eficiencia y precisión en el proceso de paletizado, respondiendo a la necesidad industrial de organizar, registrar y preparar productos para su almacenamiento. La implementación de esta tecnología se considera esencial para reducir el tiempo de procesamiento y minimizar errores humanos, lo que resulta crucial en un entorno industrial altamente competitivo.

Para el desarrollo del proyecto se utilizan dos cámaras para la adquisición de imágenes, estas imágenes son procesadas mediante programación en Python. Usando la librería OpenCV, se realizan los cálculos necesarios para que, con la primera cámara, se detecte la posición y orientación de cada caja en la banda trasportadora, esta información es enviada al primer robot para que pueda recogerla al momento de llegar a un punto medio de la banda transportadora, este robot cuenta con un sensor para la identificación de dicho objeto. Una vez que el primer robot recoge la caja, se procede a acercar la caja a la segunda cámara, la cual, con programación en Python, se procesa para decodificar el código QR que indica el tipo de producto que ha llegado, para posteriormente ser paletizado en función del tipo de producto.

El primer robot se encarga de girar la caja en las primeras cuatro caras accesibles hacia la cámara. Si en ninguna de estas caras se encuentra el código QR, se envía la señal al segundo robot, el cual se acerca a recoger la caja y gira de tal manera que permita que la segunda cámara acceda a las otras dos caras restantes. Una vez que se encuentra el código QR, se envía la señal al primer robot para que vuelva a recogerla y la coloque sobre la banda transportadora, que lleva la caja al final de esta. Mientras tanto, también se envía la información al tercer robot, encargado de recoger la caja y colocarla en el pallet correspondiente al tipo de producto detectado, acorde al código QR.

Con la primera cámara se detectan los bordes del objeto, con los cuales se procede a encontrar el centro de este, y finalmente, considerando como referencia el eje X, se

calcula la orientación, con esta orientación se hacen los cálculos correspondientes para la orientación de la mano del robot y este pueda así recoger el objeto correctamente.

Palabras Clave: Visión por computadora, Robots, RoboDK, Python, OpenCV,

ABSTRACT

In this project, an automated palletizing station is developed and simulated using RoboDK to optimize the recognition, classification, data storage and handling of products previously cataloged by QR codes. This automated solution seeks to improve efficiency and accuracy in the palletizing process, responding to the industrial need to organize, register and prepare products for storage. The implementation of this technology is considered essential to reduce processing time and minimize human error, which is crucial in a highly competitive industrial environment.

For the development of the project, two cameras are used for image acquisition, these images are processed through Python programming. Using the OpenCV library, the necessary calculations are made so that, with the first camera, the position and orientation of each box on the conveyor belt is detected, this information is sent to the first robot so that it can pick it up when it reaches a midpoint of the conveyor belt, this robot has a sensor for the identification of the object. Once the first robot picks up the box, it proceeds to approach the box to the second camera, which, with Python programming, is processed to decode the QR code that indicates the type of product that has arrived, to be subsequently palletized according to the type of product.

The first robot rotates the box on the first four faces accessible to the camera. If the QR code is not found on any of these faces, the signal is sent to the second robot, which approaches to pick up the box and rotates in such a way as to allow the second camera to access the other two remaining faces. Once the QR code is found, the signal is sent to the first robot to pick it up again and place it on the conveyor belt, which carries the box to the end of the conveyor belt. In the meantime, the information is also sent to the third robot, in charge of picking up the box and placing it on the pallet corresponding to the type of product detected, according to the QR code.

With the first camera the edges of the object are detected, with which the center of the object is found, and finally, considering the X axis as a reference, the orientation is calculated, with this orientation the corresponding calculations are made for the orientation of the robot's hand so that it can pick up the object correctly.

Keywords: Computer Vision, Robots, RoboDK, Python, OpenCV,

ÍNDICE GENERAL

ABREVIATURAS

ESPOL	Escuela Superior Politécnica del Litoral
API	Application Programming Interface
RDK	RoboDK
CAD	Computer Aided Design
QRCode	Quick Response Code
HTTP	Hypertext Transfer Protocol

SIMBOLOGÍA

cm	centímetros

ÍNDICE DE FIGURAS

CAPÍTULO 1

1. INTRODUCCIÓN

En muchas empresas, el paletizado es un proceso crítico en la cadena de suministro, donde se ven involucradas tareas como la preparación de productos para su almacenamiento y transporte. En dicho proceso, es habitual que el operario encargado de la paletización también realice otras tareas, como el llenado de cajas, el encintado o el etiquetado. Por ello este proceso repetitivo y físicamente exigente se adapta bien a la automatización.

Al automatizar este proceso, las empresas pueden mejorar instantáneamente la eficiencia, reducir el riesgo de lesiones de los empleados y liberar a los operarios para que se ocupen de tareas más complejas. Esto puede dar como resultado una mayor productividad y calidad. La demanda de mayor velocidad, precisión y eficiencia ha impulsado a las industrias a adoptar soluciones automatizadas. Los sistemas de paletizado automatizados no solo mejoran la productividad, sino que también reducen los errores y optimizan el espacio de almacenamiento. (Sam, 2023)

En este contexto, los robots se utilizan principalmente como herramientas fundamentales en la automatización de tareas repetitivas y de precisión. La mayoría de estos robots, son manipuladores industriales, de hecho “según la definición del “Robot Institute of America”, un robot industrial es un manipulador programable multifuncional diseñado para mover materiales, piezas, herramientas o dispositivos especiales, mediante movimientos variados, programados para la ejecución de distintas tareas” (Baturone, 2005, pág. 5).

No obstante, un robot por sí solo tiene una utilidad limitada. La ingeniería robótica moderna se centra en el desarrollo de sistemas robóticos integrados, donde el robot no es un elemento aislado, sino una parte fundamental de un conjunto de máquinas y herramientas que trabajan de manera sincronizada para llevar a cabo tareas complejas. En estos sistemas, el robot debe cooperar con otros dispositivos para realizar trabajos de manera eficiente. (Sánchez, 2002, pág. 26)

De igual forma, la visión por computadora ha emergido como una tecnología clave para la automatización de dicho proceso. Esta disciplina permite a los sistemas, "ver" y entender su entorno, posibilitando la detección de defectos, la identificación de productos y la correcta disposición de los mismos en un palet. La visión por computadora se dedica a la deducción automática de la estructura y propiedades de un escenario tridimensional a partir de imágenes, lo que es esencial en un entorno dinámico y cambiante como una línea de producción industrial (Baturone, 2005, págs. 30-31)

Además de su capacidad para mejorar la precisión en el paletizado, la visión por computadora ofrece beneficios adicionales como la vigilancia en tiempo real de los procesos y la gestión eficiente del inventario. Estas capacidades son cruciales en un entorno industrial que exige cada vez más eficiencia y flexibilidad. La integración de robots colaborativos en estos sistemas de paletizado automatizados permite una mayor adaptabilidad y seguridad en las operaciones, ya que estos robots están diseñados para trabajar en estrecha colaboración con los seres humanos, combinando la fuerza y precisión de la automatización con la inteligencia y adaptabilidad humanas.

En definitiva, la evolución del paletizado desde un proceso manual hacia uno altamente automatizado, apoyado por tecnologías avanzadas como la visión por computadora y los robots colaborativos, representa un avance significativo en la optimización de la cadena de suministro. Este proyecto de tesis se centra en el diseño y simulación de un sistema automatizado para una estación de paletizado que aprovecha estas tecnologías, buscando no solo mejorar la eficiencia operativa, sino también explorar nuevas posibilidades en la interacción entre humanos y máquinas en entornos industriales.

1.1 Descripción del problema

La clasificación manual de productos representa una tarea compleja debido a la notable variabilidad de los productos en términos de características físicas, como forma, tamaño, color, textura y otras propiedades. Además, en entornos industriales

y de almacenamiento, la cantidad de productos a clasificar puede ser significativamente alta, lo que convierte la tarea manual en monótona y repetitiva, aumentando así la probabilidad de errores a medida que los operadores se fatigan. La visión por computadora y la robótica ofrecen una solución prometedora para mejorar estos procesos al permitir la identificación, manipulación y clasificación de productos de manera automática.

No obstante, el desafío principal al que nos enfrentamos se encuentra en entornos complejos y dinámicos donde los sistemas tradicionales de visión para reconocimiento a menudo son insuficientes, ya que generalmente solo pueden acceder a la información visible en una de las caras de un objeto. Si el objeto con características tridimensionales está etiquetado o identificado en una de sus caras laterales o la parte inferior, el sistema de reconocimiento no podrá llevar a cabo una identificación efectiva.

Esto es lo que ocurre en una empresa de Plástico reconocida del País, donde en su mayoría las maquinarias poseen robots que permiten al operador retirar el articulo sin que este ingrese al interior de la máquina. Si bien es cierto, durante toda su trayectoria laboral lograron agilizar su productividad mediante el uso de tecnología avanzada en los procesos de fabricación y producción, actualmente se enfrentan a un desafío importante a la hora de identificar, clasificar y paletizar los productos.

Como se puede apreciar en la Figura 1.1 una vez que el artículo en inyectado en el molde y posteriormente retirado con la ayuda un brazo robótico instalado en la máquina, este producto es apilado de manera manual y embalado en cajas de cartón que, en función del producto son catalogados mediante códigos QR, para posteriormente, ser enviados al área de bodega, mediante el uso de bandas transportadoras, donde personal operativo, con la ayuda de scanner manuales, identifican y paletizan los productos.

Cabe recalcar, que tal como se muestra en la Figura 1.2, en esta línea de producción, denominada por esta empresa como “Línea de consumo masivo” existen 4 máquinas, en las que se puede montar alrededor de 10 moldes, mismas que podrán ser instaladas según las necesidades de los clientes. Por tanto, identificar y clasificar estos productos resulta una tarea complicada del proceso productivo de la empresa, que sebe ser automatizada para eliminar errores.

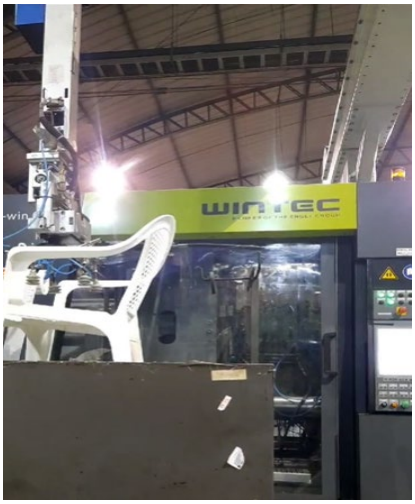

Figura 1.1 Brazo de robot en maquina plastificadora.

Figura 1.2 Brazo de robot en maquina plastificador.

Por lo tanto, el problema que abordaremos en esta tesis se centra en desarrollar un sistema de reconocimiento que sea capaz de acceder a la información independientemente a la cara del objeto donde esta e encuentre ubicada, cabe mencionar que los objetos se encuentran embalados en cajas rectangulares (paralepípedos rectangulares).

1.2 Justificación del problema

La justificación de este proyecto radica en la necesidad imperante de abordar los entornos donde los objetos posean diferentes características físicas y que además se ubiquen en diferentes lugares o posiciones, dando como resultado una tarea compleja a la hora de manipular, identificar y clasificar productos. Por lo que en estos casos en particular se recurre a la identificación manual, lo que puede conducir a errores y pérdidas de productividad.

En este escenario, los sistemas automáticos de identificación de productos, a través de visión por computadora, pueden ayudar a resolver estos problemas. Estos sistemas pueden identificar los productos de forma rápida y precisa, independientemente de su ubicación o posición, mejorando así la eficiencia y productividad de los procesos industriales.

1.3 Objetivos

1.3.1 Objetivo general

Planificar trayectorias de manipuladores industriales en entornos no estructurados.

1.3.2 Objetivos específicos

1. Identificar el entorno donde se realizarán las pruebas
2. Programar las trayectorias de los robots de manera independiente.
3. Establecer la comunicación entre robots, con la planificación establecida
4. Desarrollar sistema de visión para el reconocimiento de objetos.
5. Comprobar la planificación de trayectorias para la identificación de productos

1.4 Marco teórico

1.4.1 Robots manipuladores industrial.

Los robots manipuladores de uso industriales son, “esencialmente, brazos articulados. De forma más precisa, un manipulador industrial convencionalmente es una cadena cinemática abierta formada por un conjunto de eslabones o elementos de la cadena interrelacionados mediante articulaciones o pares cinemáticos” (Baturone, 2005, pág. 16).

Las empresas obtienen grandes beneficios de los brazos robóticos industriales, especialmente en términos de seguridad y productividad. Estos robots mejoran significativamente la seguridad en el entorno laboral porque pueden realizar tareas peligrosas que ponen en riesgo a los trabajadores. Además, su capacidad para funcionar sin interrupciones las 24 horas del día permite a las empresas mantener un alto nivel de producción y realizar inspecciones constantes, lo que resulta en una mayor eficiencia.

La flexibilidad y precisión de los brazos robóticos también son cruciales. En tareas que requieren un alto grado de precisión, su funcionamiento es consistentemente más preciso que el de los humanos. Además, estos robots son extremadamente adaptables, lo que significa que pueden ser reprogramados para realizar nuevas tareas.

Figura 1.3 Brazo robótico industrial *(Intel, 2020)*.

Un brazo robótico posee varias juntas que funcionan como ejes y permiten una variedad de movimientos. Un brazo robótico tiene más libertad de movimiento cuando tiene más juntas rotativas. En su mayoría los brazos robóticos tienen entre cuatro y seis juntas, lo que significa que cuentan con la misma cantidad de ejes de rotación para facilita su movimiento. Dentro de los compontes que lo conforman encontramos el controlador del robot, una herramienta en el extremo del brazo, los accionadores, los sensores, los sistemas de visión, los sistemas de alimentación y los componentes de software, además de las juntas rotativas, tal como se aprecia a continuación. (Intel, 2020)

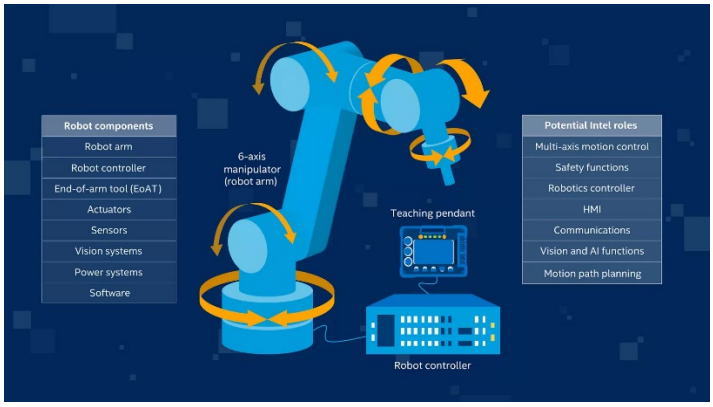

Figura 1.4 Componentes de un brazo robótico industrial *(Intel, 2020)*.

1.4.2 Visión por computadora

La visión por computadora abarca toda una disciplina de estudio en robótica e informática, los avances en este campo han permitido el diseño de soluciones automatizadas inteligentes, dinámicas y versátiles que están ayudando a las marcas de vanguardia a alcanzar objetivos cada vez más ambiciosos. La visión computarizada, también conocida como visión computarizada, se refiere a un conjunto de herramientas y tecnologías que permiten a los equipos capturar imágenes del mundo real, procesarlas y generar información a través de ellas

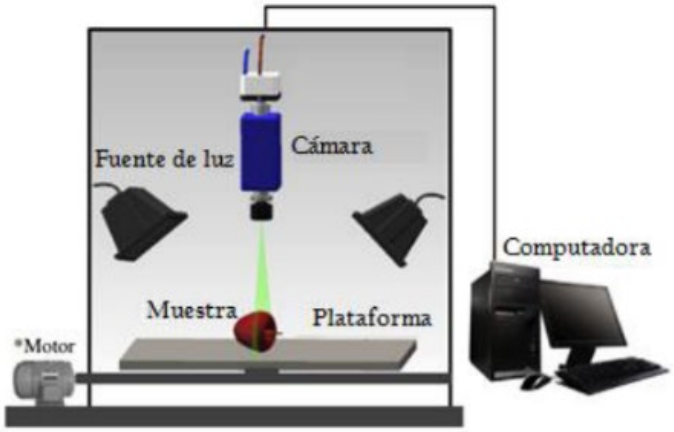

Figura 1.5 Esquema de un sistema de V.C *(Sandoval-Gonzalez, 2016)*.

1.4.3 Open CV

Open CV es una biblioteca código abierto cuyas aplicaciones van desde la visión artificial como detección y reconocimiento de objetos, seguimiento de objetos en movimiento, reconocimiento facial, calibración de cámaras, análisis de imágenes médicas y mucho más se realizan con esta biblioteca. La biblioteca está diseñada en C++ y tiene interfaces para múltiples lenguajes de programación, como Python y Java, lo que la hace más compatible con varios entornos de desarrollo. Por otro lado, debido a su solidez tecnológica, agilidad y disponibilidad de código abierto, OpenCV es ampliamente utilizado por los profesionales de la visión artificial. (Nikitins, 2023)

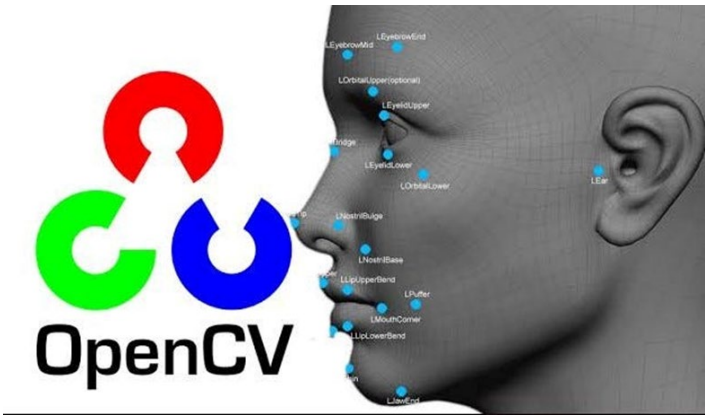

Figura 1.6 Librería OpenCV *(Nikitins, 2023)*.

1.4.4 Ubidots IoT

Incluyendo la robótica industrial, los métodos de programación y la plataforma en línea para gestionar la información, en particular se revisará Ubidots IoT. A continuación, ubidots proporciona a los responsables de la toma de decisiones la información exacta que necesitan para tomar decisiones críticas en tiempo real, independientemente de dónde se encuentren los usuarios. Ya sea en la oficina, en la planta de producción, en casa o de viaje. Ubidots mantiene a los usuarios informados y capacitados para tomar constantemente decisiones basadas en datos para mejorar la producción. (Ubidots, 2024).

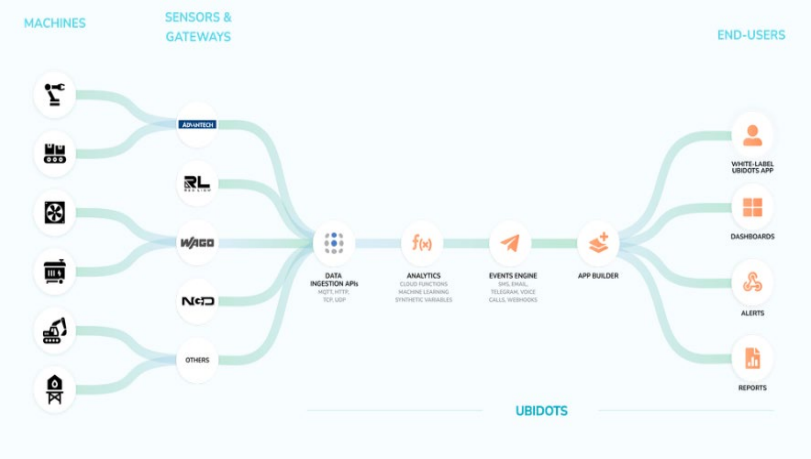

Figura 1.7 Estructura de Ubidots *(Ubidots, 2024)*.

1.4.5 RoboDK

RoboDK es un simulador que se enfoca en aplicaciones industriales de robótica. Esto significa que los programas de robot pueden ser creados y simulados fuera de línea para un controlador de robot y un brazo robot específicos. En otras palabras, RoboDK es un programa de programación fuera de línea.

Para crear programas de robot, es necesario seleccionar un robot, cargar sus herramientas y usar una o más funciones de CAD para crear programas agregando objetivos o usando herramientas específicas. Está disponible una biblioteca extensa de robots industriales. Usando controladores específicos del

proveedor, como los límites de los ejes y el sentido, RoboDK modela los robots industriales de la misma manera que se comportan. (RoboDK, 2024)

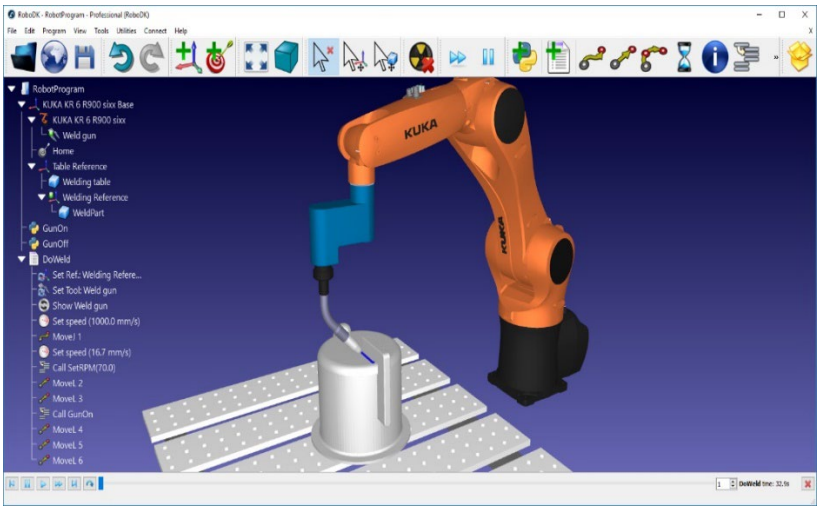

Figura 1.8 Software de simulación para robots industriales *(RoboDK, 2024)*.

1.4.6 Métodos de programación de robot industrial.

Entre las metodologías de programación de robots industriales tenemos, la programación fuera de línea. “Esto significa que los programas de robot pueden ser creados, simulados y generados fuera de línea para un brazo robot específico y un controlador de robot” (RoboDK, 2024).

Una principal ventaja de la programación fuera de línea es permitir estudiar y analizar múltiples eventos o escenarios que pueden suceder en la vida real, de tal forma que se pueden predecir posibles errores durante la ejecución de su programación final ya en la línea de trabajo, además si en un futuro se busca implementar algo nuevo a esa línea de trabajo en el mismo entorno simulado se lo puede agregar para las pruebas previo a la compra y a la instalación.

Otra metodología utilizada para la programación de robots es la programación online, la cual permite conectarse directamente con el robot en tiempo real desde un software, por ejemplo, el software RoboDk, y ejecutar exactamente los mismos comandos que en el simulador, y a su vez ejecutar una programación en Python

que se tenga ya probado en dicho simulador, este método es ideal si se tiene la disposición del robot, para que no interrumpa ninguna línea de trabajo y evitar parar la producción.

Parte de esta programación online está la programación punto a punto, la cual se la realiza con comandos desde un software que permita una conexión directa con el robot industrial como por ejemplo RoboDk, Termius, Kr-term, Putty, MobaXterm, KCWINTCP entre otros, y dependiendo del robot se lo puede realizar mediante protocolo ssh, telnet entre otros, ésta metodología permite mover al robot a posiciones específicas y guardar en su memoria dichas posiciones, de tal forma que se establece al robot todos los puntos en el espacio a los que se desea mover, se los guarda en un archivo con extensión acorde al robot, por ejemplo para los robots de la marca Kawasaki es .as, y después se lo pone a ejecutar cíclicamente, según lo necesitado.

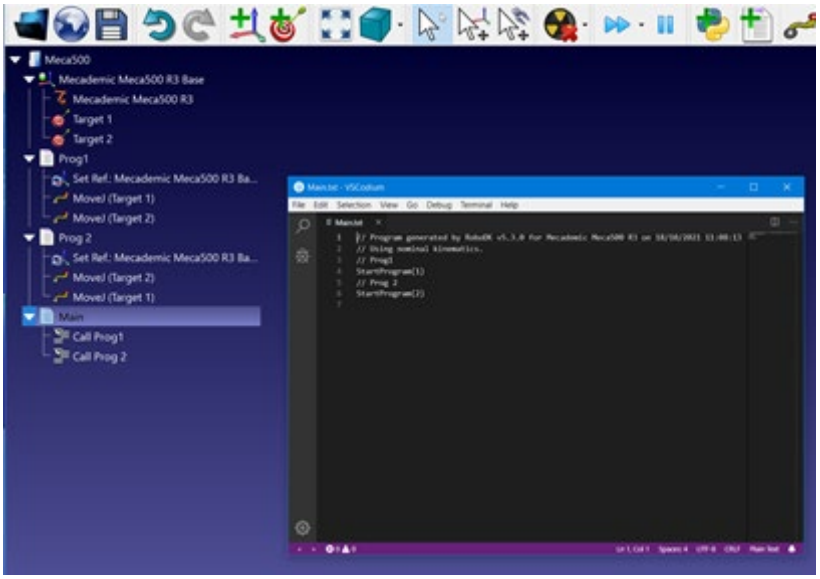

Figura 1.9 Programación fuera de línea RoboDK *(RoboDK, 2024)*.

CAPÍTULO 2

2. METODOLOGÍA

En el presente trabajo, para la programación de los manipuladores o brazos robóticos, utilizamos el software de RoboDk. Esta plataforma de programación nos permite diseñar sistemas automatizados en un entorno virtual, en este sentido, dicho sofware nos ofrece las siguientes alternativas de programación:

1. Programación mediante *arrastrar y soltar* (programación visual): RoboDK proporciona una interfaz de programación visual fácil de usar que permite crear programas de robot arrastrando y soltando iconos que representan diversas acciones y movimientos. Este método es ideal para principiantes y para aquellos que prefieren un enfoque visual de la programación.

2. Programación offline con *Python Scripting*: Para los usuarios más avanzados y aquellos familiarizados con Python, RoboDK ofrece capacidades de programación fuera de línea utilizando scripts de Python. Este método le da más flexibilidad y control sobre el proceso de programación, lo que le permite crear programas de robots complejos o personalizados para distintos tipos de robots.

3. Programación *en línea* con comunicación de socket: RoboDK también es compatible con la programación en línea a través de la comunicación socket. Este método permite la interacción en tiempo real entre el robot y el entorno de programación, lo que le permite controlar dinámicamente los movimientos y acciones del robot.

4. Generación de programas a partir de archivos CAD: RoboDK puede generar programas de robot directamente desde archivos CAD (.dxf, .stp, etc.). Esta función es especialmente útil para crear programas que impliquen movimientos precisos e interacciones con objetos o entornos diseñados en CAD.

5. Utilización de bibliotecas externas: RoboDK se integra perfectamente con varias bibliotecas externas, como OpenCV y NumPy, lo que le permite incorporar funcionalidades avanzadas como el procesamiento de imágenes, el análisis de datos y el aprendizaje automático en sus programas de robot.

De las alternativas antes citadas, utilizaremos la programación fuera de línea dado que, como se mencionó anteriormente, es un método flexible en cuanto a control sobre el proceso.

2.1 Diagrama de flujo de los procesos

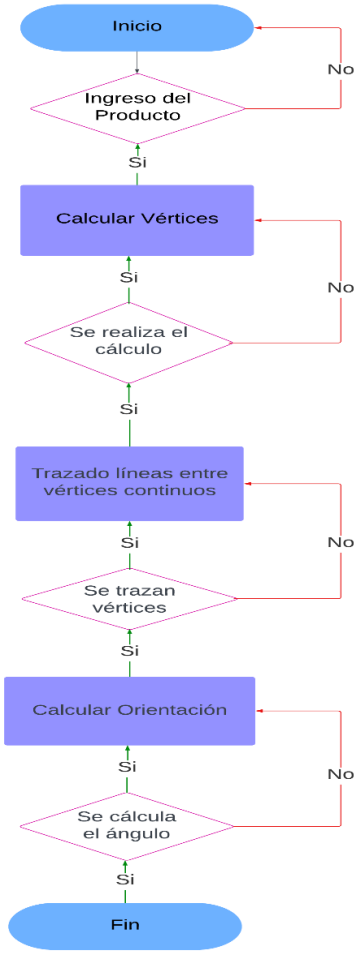

Figura 2.1 Diagrama de flujo para el cálculo de la orientación del producto.

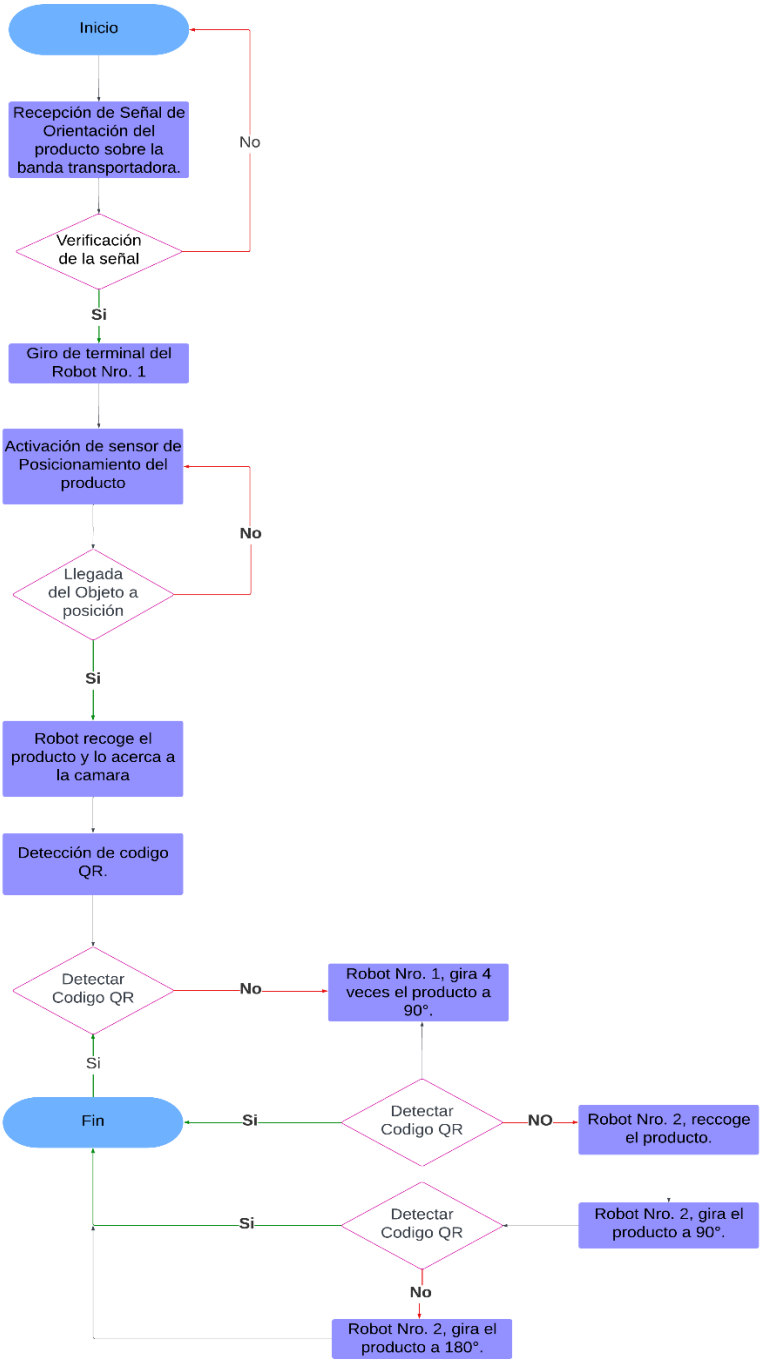

Figura 2.2 Diagrama de flujo para la identificación del producto.

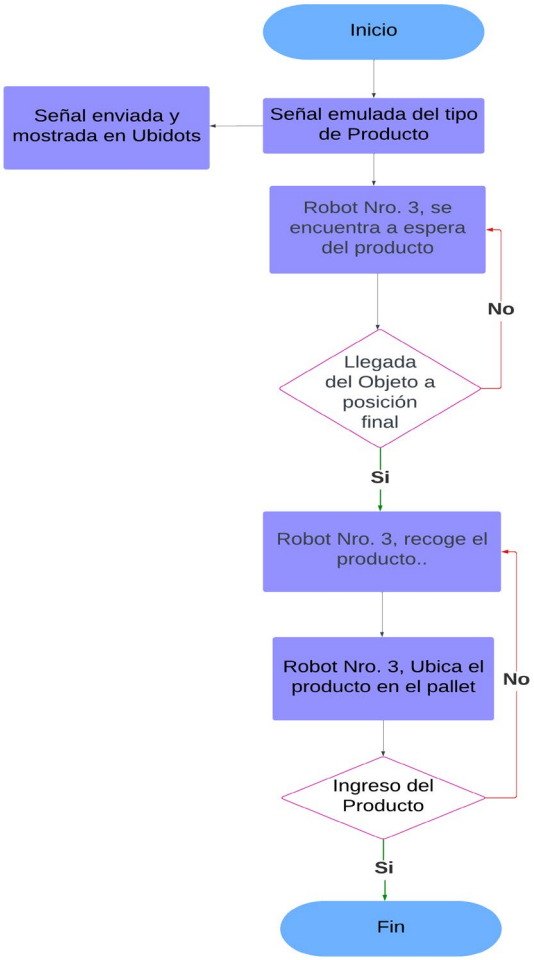

Figura 2.3 Diagrama de flujo para la clasificación y ubicación del producto.

2.2 Definición de entorno de programación.

Uno de los principales retos que debemos afrontar radicaba en la identificación de los productos una vez que estos eran empacados en cajas, más adelante estos productos son enviados mediante bandas transportadoras a un operador humano quienes con un scanner manual identifican el contenido para luego organizarlo en función del tipo de producto almacenado en dichas cajas.

En este sentido definimos los siguientes elementos mínimos necesarios que debe llevar nuestro entorno de programación:

- Cajas para empacar productos de 50cmx50cm

- Cámara para identificación de productos
- Banda transportadora
- Robots manipuladores con pinzas de agarre.

Para el proceso de reconocimiento y manipulación de objetos desarrollado en un entorno simulado, utilizamos el software de simulación y programación de robots industriales RoboDK, a través de la programación fuera de línea con scripts de Python, este tipo de programación nos permite trabajar con robots complejos en ambientes personalizados.

Tomando como base los elementos necesarios de nuestro sistema, en la Figura 2.4 se puede apreciar el entorno propuesto para la implementación de nuestro sistema.

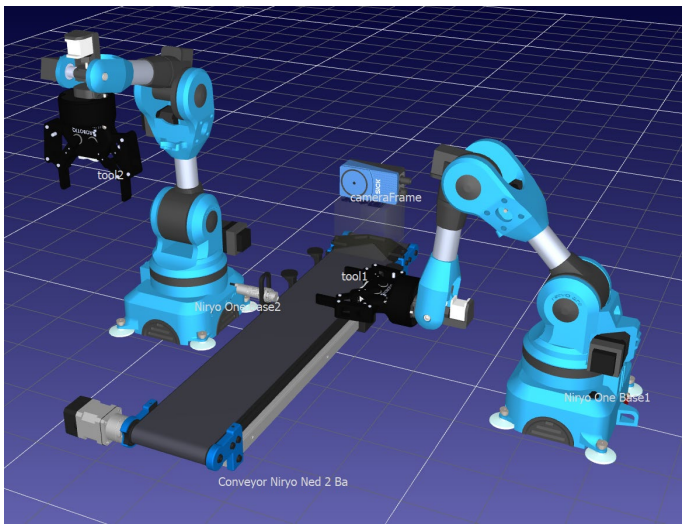

Figura 2.4 Entorno de programación del sistema.

En la Figura 2.5 podemos observar los elementos necesarios que se requirieren para la simulación de nuestra línea de paletizado, descritos de la siguiente forma:

- **Niryo One Base 1:** Es uno de nuestros robots a utilizar, cuenta con una herramienta denominada Tool1 que nos permitirá agarrar los objetos.
- **Niryo One Base 2:** Es nuestro segundo robot para utilizar, al igual que nuestro robot anterior, cuenta con una herramienta denominada Tool2 que nos permitirá agarrar los objetos.

- **Niryo One Base 3:** Es nuestro tercer robot y será utilizado para colocar los productos en los palets.
- **Converyor Niryo Ned 2:** Esta es nuestra banda trasportadora, misma que cuenta con un motor eléctrico para generar movimientos y un sensor de detección de objetos.
- **Cámara 1:** Esta cámara nos permite visualizar la posición del producto que será ubicado en la banda transportadora.
- **Cámara 2:** Esta cámara nos permitirá realizar la identificación del objeto, mediante la visión por computadora.
- **Movimientos:** Contine el código para la manipulación e identificación de objetos.
- **Productbox:** Nombre de los objetos que contienen el código QR con la descripción del producto.

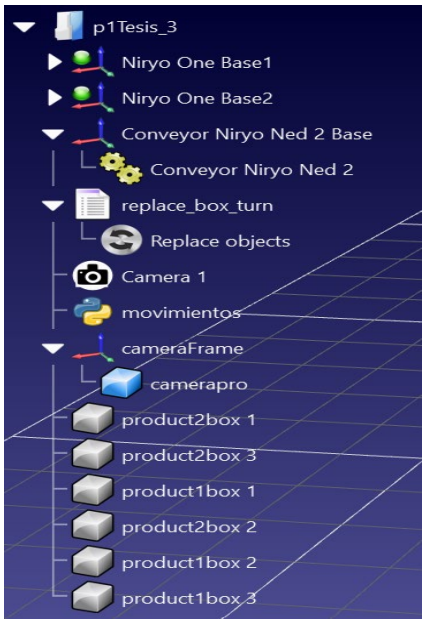

Figura 2.5 Árbol de elementos y acciones de RoboDK.

2.3 Algoritmos para la colaboración e interacción de los manipuladores

En la primera etapa del sistema automatizado definimos el código que permitirá a los robots colaborar para identificar y clasificar los productos, tal como se detalla en la Figura 2.6.

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

Figura 2.6 Parte#1 del código Python.

En esta parte importamos los módulos necesarios de la API de RoboDK y otras librerías como “time” que nos proporcionan funciones para trabajar con tiempos, inducir retrasos o suspender una parte del código para posteriormente continuar con la ejecución. También se creó un objeto denominado “Robolink”, que será la interfaz de comunicación entre Python y RoboDK.

En la Figura 2.7 se presenta la definición de las diferentes cajas utilizando el método Item de la interfaz RDK; estas cajas simularan los productos colocados en la banda transportadora. Como se puede observar estas cajas están siendo referenciadas con nombres específicos ('product1box 1', 'product1box 2', etc.), y luego se almacenan en una lista llamada boxes.

Una vez realizada la definición del entorno de trabajo, ejecutamos un programa llamado "replace_box_turn" en el entorno de RoboDK y definimos diferentes objetos relacionados con el robot Niryo One 1, como la base (refNiryo1), el punto de control de herramientas (tcp1), puntos de recogida y puntos de aproximación, así como puntos específicos para giros (niryo1giro1, niryo1giro2, etc.); realizamos este mismo proceso para el robot Niryo One 2.

```
# Program example:
box1 = RDK.Item('product1box 1')
box2 = RDK.Item('product1box 2')
box3 = RDK.Item('product1box 3')
box4 = RDK.Item('product2box 1')
box5 = RDK.Item('product2box 2')
box6 = RDK.Item('product2box 3')
boxes = [box4,box6,box1,box5,box2,box3]
RDK.RunProgram("replace_box_turn")

#importar todo sobre niryo 1
niryo1 = RDK.Item("Niryo One1")
refNiryo1 = RDK.Item("Niryo One Base1")
tcp1 = RDK.Item("tool1")
pick1 = RDK.Item("pick_box")
aproxpick1 = RDK.Item("retract_p1_pick")
aproxgiro1_n1 = RDK.Item("retract_p1_1")
niryo1giro1 = RDK.Item("p1_1")
niryo1giro2 = RDK.Item("giro1_niryo1")
niryo1giro3 = RDK.Item("giro2_niryo1")
niryo1giro4 = RDK.Item("giro3_niryo1")
girosnyrio1 = [niryo1giro1,niryo1giro2,niryo1giro3,niryo1giro4]
retractgiro4n1 = RDK.Item("retract_grio3_niryo1")

#importar todo sobre niryo 2
niryo2 = RDK.Item("Niryo One2")
refNiryo2 = RDK.Item("Niryo One Base2")
tcp2 = RDK.Item("tool2")
poswait = RDK.Item("wait_niryo2")
aproxgiro1_n2 = RDK.Item("retract_p1_2")
niryo2giro1 = RDK.Item("p1_2")
niryo2giro2 = RDK.Item("giro_1_niryo2")
aproxgiro2_n2 = RDK.Item("retract_p2_2")
niryo3giro2 = RDK.Item("giro2_niryo2")
girosnyrio2 = [niryo2giro1,niryo2giro2,niryo3giro2]
#movimiento robots a posicion de espera
niryo1.MoveJ(aproxpick1)
niryo2.MoveJ(poswait)
```

Figura 2.7 Parte#2 del código de Python.

2.4 Algoritmo para la identificación de objetos

En esta sección se describe el algoritmo desarrollado para identificar el tipo de producto que viene previamente almacenado e identificado en las cajas mediante códigos QR.

En la Figura 2.8, se presenta la función qrcodefoto(). Esta función permite obtener la posición y orientación de un objeto (sistema cartesiano solidario al objeto) respecto del sistema de referencia de la cámara. También permite encontrar la relación entre el sistema de referencia de la cámara y la escena definida en RoboDK. Esta función importa las bibliotecas necesarias para trabajar con códigos QR y manejar imágenes; establece dos variables en RoboDK llamados "producto1" y "producto2" inicializados en “cero”; captura una imagen de la cámara especificada y la guarda en un archivo llamado "image.png" en el directorio especificado de PATH_OPENSTATION.

La función permite: i) procesar la imagen; ii) decodificar el código QR, leyendo así los datos codificados; iii) iterar sobre los datos decodificados y guardar como una cadena de texto; iv) verificar si se detectó algún código QR y, en caso afirmativo, comprobar si es producto uno o producto dos. Luego, permite actualizar los parámetros correspondientes y mostrar un mensaje en RoboDK. Si no se detecta ningún producto, se imprime un mensaje indicándolo.

```python
# camara
def qrcodefoto():
    camref = RDK.Item("cameraFrame")
    camId = RDK.Item("Camera 1")

    import qrcode   #pip install qrcode
    import cv2      #pip install opencv-python
    from pyzbar.pyzbar import decode    #pip install pyzbar
    from PIL import Image


    RDK.setParam("producto1",0)
    RDK.setParam("producto2",0)
    #RDK.ShowMessage("product1=0 \n product2=0")
    RDK.Cam2D_Snapshot(RDK.getParam('PATH_OPENSTATION') + "/image.png", camId)
    imagen = Image.open(RDK.getParam('PATH_OPENSTATION') + "/image.png") # Carga la imagen del código QR
    datos = decode(imagen) # Decodifica el código QR
    print(datos)
    for dato in datos:
        tipo = dato.data.decode('utf-8') # Imprime los datos decodificados

    if (datos):
        if tipo == "PRODUCTO1":
            RDK.setParam("producto1",1)
            RDK.ShowMessage("producto 1 detectado",False)
        elif tipo == "PRODUCTO2":
            RDK.setParam("producto2",1)
            RDK.ShowMessage("producto 2 detectado",False)
        print(tipo)
    else:
        print("no product detect")
#camara
```

Figura 2.8 Parte#3 del código Python.

Luego se genera una función denominada movebox(bx) la cual permite mover una caja (bx) en incrementos verticales. Utiliza un bucle FOR para ajustar la posición de la caja en la dirección Y (vertical) en 23 pasos, cada uno con una pausa de 0.1 segundos, ver Figura 2.8 adjunta a continuación.

```python
#movimiento caja
def movebox(bx):
    for i in range(23):
        bx.setPose(bx.Pose()*transl(0,i))
        sleep(0.1)
```

Figura 2.9 Parte#4 del código Python.

Seguidamente se inicializan las variables necesarias para el funcionamiento del script. Se establecen los contadores “pros1” y “pros2” en 0, y se establecen los parámetros "producto1" y "producto2" en 0 utilizando RDK.setParam(). Además, se desactiva el renderizado RDK.Render(False), ver Figura 2.7.

```
pros1= 0
pros2= 0
RDK.setParam("producto1",0)
RDK.setParam("producto2",0)
RDK.Render(False)
```

Figura 2.10 Parte#5 del código Python.

Tras ocultar todas las cajas (boxes), se vuelven a hacer visibles, moviéndolas con la función movebox(b) definida anteriormente. En esta etapa, se utiliza un bucle FOR para que cada caja sea manipulada por los robots de acuerdo con un conjunto de condiciones. Las acciones incluyen el movimiento de los robots para agarrar y soltar las cajas, así como la detección de productos mediante el escaneo de códigos QR (qrcodefoto()).

Al finalizar el bucle, se muestra un mensaje emergente en RoboDK con el total de productos de tipo 1 y tipo 2 detectados durante el proceso de manipulación de las cajas, tal como se detalla en la Figura 2.11.

```
RDK.ShowMessage("Total p1 = " + str(pros1) + "\n"
                "Total p2 = " + str(pros2 ))
```

Figura 2.11 Parte#6 del código de Python.

2.5 Algoritmo para la clasificación de Objetos

La Figura 2.12 presenta el código usado para inicializar la manipulación del robot. El procesamiento de imágenes es crucial en esta configuración, ya que permite al sistema identificar las cajas que serán manipuladas por el robot. Esto se logra a través de la biblioteca **cv2**, que se utiliza para analizar las imágenes capturadas,

detectar características relevantes, y realizar las tareas necesarias para el manejo preciso de los objetos.

RoboDK es una poderosa herramienta que facilita la programación y simulación de robots. En este contexto, permite crear y controlar el entorno virtual donde se simulan las operaciones robóticas, como el movimiento y la interacción con las cajas en la banda transportadora. Por otro lado, **Robolink** se utiliza para establecer y gestionar la conexión entre el código de control (escrito en Python) y la interfaz de RoboDK. A través de la instancia RDK = Robolink(), se habilita la comunicación directa, permitiendo que las instrucciones programadas en el código se ejecuten en el entorno de RoboDK, controlando así los movimientos del robot y otras acciones relacionadas.

En conjunto, estas herramientas y bibliotecas trabajan armoniosamente para automatizar y optimizar el proceso de manipulación robótica, desde el procesamiento de imágenes para la identificación de objetos, hasta la ejecución precisa de movimientos y operaciones dentro del entorno simulado.

```python
from robodk import*    # RoboDK API
from robolink import*   # Robot toolbox
from time import sleep

import numpy as np
import cv2
import random
import math
import os

RDK = Robolink()
RDK.Render(False)

conveyor = RDK.Item("Conveyor Niryo Ned 2")

conveyor.setJoints([-20000])

RDK.RunProgram("newreplaceobjects")
RDK.RunProgram("newreplaceobjects")
RDK.RunProgram("newreplaceobjects")
```

Figura 2.12 Parte#7 del código Python.

En la Figura 2.13, la función move2conveyor recibe un parámetro movcuan, que indica la distancia que debe moverse la banda transportadora. Esta función utiliza la API de RoboDK para trasladar la banda a una nueva posición multiplicando la pose actual de la banda por una transformación de traslación.

```python
def move2conveyor(movcuan):
    conveyor.MoveJ(conveyor.Pose()*transl(movcuan),True)
```

Figura 2.13 Parte#8 del código Python.

En la Figura 2.14 se presenta la función camas. Esta tiene como objetivo determinar la orientación de la caja. Primero, se configura la cámara y se toma una foto de la caja. Luego, se convierte la imagen a escala de grises y se aplica un umbral para binarizarla.

```python
def camas(bx):
    camId = RDK.Item("Camera 2")
    box=bx
    gir = random.randint(0,90)
    radianss = math.radians(gir)
    box.setPose(box.Pose()*rotz(radianss))
    #toma foto y guarda en la dirección establecida
    ruta_actual = os.path.abspath(__file__)
    ruta_actual = os.path.dirname(ruta_actual)
    RDK.Cam2D_Snapshot(str(ruta_actual)+"/boxnews.png", camId)

    # read the image
    img  = cv2.imread(str(ruta_actual)+"/boxnews.png",0)
    img2 = cv2.imread(str(ruta_actual)+"/boxnews.png")
    # Binarize the image
    ret,thresh = cv2.threshold(img,127,255,0)
    area = 0
    while area < 3000:
        RDK.Cam2D_Snapshot(str(ruta_actual)+"/boxnews.png", camId)
        # read the image
        img  = cv2.imread(str(ruta_actual)+"/boxnews.png",0)
        ret,thresh = cv2.threshold(img,127,255,0)
        # Find the contours
        contours,hierarchy = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE) #1, 2
        cnt = contours[0]
        # Calculate the moments
        M = cv2.moments(cnt)
        # Calculate area
        area = M['m00']
    # Calculate centroid
    cx = int(M['m10']/M['m00'])
    cy = int(M['m01']/M['m00'])

    cv2.drawMarker(img2, [cx,cy], color=(255, 25, 55), markerType=cv2.MARKER_SQUARE, markerSize=7, thickness=4)
    rect = cv2.minAreaRect(cnt)
    box = cv2.boxPoints(rect)
```

Figura 2.14 Parte#9 del código de Python.

Luego se buscan contornos en la imagen binarizada, y se calcula el área del contorno más grande para asegurarse de que sea válido. Se calcula el centroide y los vértices del contorno, y se dibujan en la imagen. El ángulo de la caja se calcula a partir de la diferencia de coordenadas entre dos puntos consecutivos del contorno y se convierte a grados. Este ángulo se ajusta y se visualiza en la imagen antes de ser devuelto por la función como se puede observar en la Figura 2.15 y 2.16.

```
box = np.intp(box)
# Dibujar el rectángulo sobre la imagen original
cv2.drawContours(img2, [box], 0, (255,0,0), 5)
for punto in box:
    cv2.drawMarker(img2, punto, color=(0, 255, 0), markerType=cv2.MARKER_SQUARE, markerSize=7, thickness=4)
    #MARKER_CROSS     #MARKER_DIAMOND   #MARKER_SQUARE

cv2.line(img2, box[2],box[3],  color=(200, 0, 255), thickness=5)
# Calcula el ángulo de la línea entre dos puntos adyacentes del cuadrado
dx = box[1][0] - box[0][0]
dy = box[1][1] - box[0][1]
angulo_cuadrado = np.arctan2(dy, dx)

# Convertir de radianes a grados
angulo_cuadrado_grados = np.degrees(angulo_cuadrado)

# Si la línea horizontal tiene un ángulo de 0 grados, entonces el ángulo relativo
# entre el cuadrado y la línea horizontal es simplemente el ángulo del cuadrado
angulo_relativo = angulo_cuadrado_grados

# Ajustar el rango del ángulo si es necesario
if angulo_relativo < 0:
    angulo_relativo += 180
    angulo_relativo = 180 - round(angulo_relativo)
    #print(f"El ángulo entre el cuadrado y la línea horizontal es: {angulo_relativo} grados")
    cv2.putText(img2,'ang : '+str(angulo_relativo), (10,30),2,1,(0,105,255),4)
elif angulo_relativo == 0:
    cv2.putText(img2,'ang : '+str(round(angulo_relativo)), (10,30),1,1,(0,5,255),4)
    #print(f"El ángulo entre el cuadrado y la línea horizontal es: {round(angulo_relativo)} grados")
```

Figura 2.15 Parte#10 del código de Python.

```
# Mostrar la imagen con el rectángulo dibujado
cv2.imshow('Image with line square and points', img2)
return angulo_relativo
```

Figura 2.16 Parte#11 del código de Python.

Se inicializan variables globales y se configuran las cámaras. La conexión con RoboDK se establece nuevamente para asegurar que todos los componentes

estén listos. Además, se inicializan parámetros para los sensores y contadores de productos como se puede observar en la Figura 2.17.

```
globarles1 = RDK.getParam("prductostotales1")
globarles2 = RDK.getParam("prductostotales1")

camId = RDK.Item("Camera 1")
# Link to RoboDK
RDK = Robolink()
sensor = RDK.getParam("sensor")
global p1,p2
p1=0
p2=0
RDK.setParam("pro1",p1)
RDK.setParam("pro2",p2)
```

Figura 2.17 Parte#11 del código de Python.

En la Figura 2.18 se define la función qrcodefoto() que captura una imagen del código QR en la caja y lo decodifica para identificar el tipo de producto. Se toma una foto con la cámara configurada y se guarda en una ruta específica. Luego, se abre la imagen y se decodifica usando la librería pyzbar. Si se detecta un código QR, se extrae el tipo de producto y se actualizan los parámetros globales. Se muestran mensajes para confirmar la detección del producto.

```python
def qrcodefoto():
    camref = RDK.Item("cameraFrame")
    camId = RDK.Item("Camera 1")
    global p1,p2
    p1=0
    p2=0
    RDK.RunProgram("readycero")
    RDK.setParam("pro1",p1)
    RDK.setParam("pro2",p2)
    #RDK.ShowMessage("product1=0 \n product2=0")
    RDK.Cam2D_Snapshot(RDK.getParam('PATH_OPENSTATION') + "/image.png", camId)
    imagen = Image.open(RDK.getParam('PATH_OPENSTATION') + "/image.png") # Carga la imagen
    datos = decode(imagen) # Decodifica el código QR
    print(datos)
    for dato in datos:
        tipo = dato.data.decode('utf-8') # Imprime los datos decodificados

    if (datos):
        if tipo == "PRODUCTO1":
            p1=1
            pro1 = "pro1"
            RDK.setParam("pro1",p1)
            RDK.ShowMessage("producto 1 detectado",False)
        elif tipo == "PRODUCTO2":
            p2=1
            pro2 = "pro2"
            RDK.setParam("pro2",p2)
            RDK.ShowMessage("producto 2 detectado",False)
        print(tipo)
    else:
        print("no product detect")
```

Figura 2.18 Parte#12 del código de Python.

El bloque de código representada en la Figura 2.19 configura los robots Niryo y las herramientas que utilizarán para manipular las cajas. Cada robot y herramienta se asigna a una variable correspondiente. Se definen los puntos de referencia y las posiciones de recogida y colocación de los objetos que se puede observar en la Figura 2.20. Además, se crea una lista de cajas que serán manipuladas durante el proceso.

```python
#importar todo sobre niryo 1
niryo1 = RDK.Item("Niryo One1")
refNiryo1 = RDK.Item("Niryo One Base1")
tcp1 = RDK.Item("tool1")
lRot")
ed1")
pick1 = RDK.Item("pick_box")
aproxpick1 = RDK.Item("retract_p1_pick")
aproxgiro1_n1 = RDK.Item("retract_p1_1")
niryo1giro1 = RDK.Item("p1_1")
niryo1giro2 = RDK.Item("giro1_niryo1")
niryo1giro3 = RDK.Item("giro2_niryo1")
niryo1giro4 = RDK.Item("giro3_niryo1")
girosnyrio1 = [niryo1giro1,niryo1giro2,niryo1giro3,niryo1giro4]
retractgiro4n1 = RDK.Item("retract_grio3_niryo1")

#importar todo sobre niryo 2
niryo2 = RDK.Item("Niryo One2")
refNiryo2 = RDK.Item("Niryo One Base2")
tcp2 = RDK.Item("tool2")
tcp2_2 = RDK.Item("closed2")
poswait = RDK.Item("wait_niryo2")
aproxgiro1_n2 = RDK.Item("retract_p1_2")
niryo2giro1 = RDK.Item("p1_2")
niryo2giro2 = RDK.Item("giro_1_niryo2")
aproxgiro2_n2 = RDK.Item("retract_p2_2")
niryo3giro2 = RDK.Item("giro2_niryo2")
girosnyrio2 = [niryo2giro1,niryo2giro2,niryo3giro2]
#movimiento robots a posicion de espera
niryo1.MoveJ(aproxpick1)
niryo2.MoveJ(poswait)
```

(variable) pick1: Any

Figura 2.19 Parte#13 del código de Python.

.

```python
for bx in boxes:
    bx.setParentStatic(conveyor.Parent())

def cl(tp,tp2):
    tp.setVisible(True)
    tp2.setVisible(False)

pros1= 0
pros2= 0
RDK.setParam("pros1",pros1)
RDK.setParam("pros2",pros2)

RDK.Render(False)
for b in boxes:
    b.setVisible(False)
RDK.Render(True)
RDK.RunProgram("readycero")
RDK.RunProgram("robotPickandPlace")
print("ready is",RDK.getParam("listo"))
for b in boxes:
    #mostrar la caja a escanear
    b.setVisible(True)
    #se establece la referencia de la caja con respecto al frame sobre la banda transportadora
    b.setParentStatic(boxes_frame)
    #se mueve el robot uno a su posición de espera
    niryo1.MoveJ(aproxpick1)
    #movemos la barra para dar paso a la caja
    RDK.RunProgram("MoveBarra")
    sleep(1)
    #llamar a la función que detecta la orientación de la caja que llega
    angulo = camas(b)   #devuelve el ángulo detectado de la orientacion de la caja
    if angulo > 45: #si esq el angulo es mayor a 45 utilizamos el complemento para hacer mover hacia el otro lado al robot
        angulo =angulo-90
    #se activa el sensor para detectar cuando llegue la caja al punto establecido
    #para que el robot 1 pueda recogerlo
    sensor = RDK.getParam("sensor")
    RDK.ShowMessage("esperando sensor",False)
```

Figura 2.20 Parte#14 del código de Python.

```python
move2conveyor(mov1)
#cambio de terminales para observar la apertura de las pinzas
cl(tcp1,tcp1_1)
cl(tcp2,tcp2_2)
RDK.RunProgram("sensor")
#mientras no se detecte el objeto se continua esperando hasta que llegue
while (not sensor):
    sensor = RDK.getParam("sensor")
#agarre de caja por robot1
RDK.Render(True)
#se establece el tool1
niryo1.setTool(tcp1)
niryo1.MoveJ(pick1.Pose()*transl(0,50))
#se establece el tooldereferencia para el giro
niryo1.setTool(tcp1rot)
#se acerca el robot a la posición de pick
niryo1.MoveJ(niryo1.Pose())
#el robot gira el angulo en el cual fue detectada la caja
niryo1.MoveJ(niryo1.Pose()*roty(math.radians(angulo)))
#se establece el tool1
niryo1.setTool(tcp1)
niryo1.MoveL(niryo1.Pose()*transl(0,-50))
niryo1.setParent(refNiryo1)
cl(tcp1_1,tcp1)
tcp1_1.AttachClosest(tolerance_mm=100)
#movimiento de robot1 con caja a posicion de escaneo
i = 0
p1=0
p2=0
RDK.setParam("pro1",p1)
RDK.setParam("pro2",p2)
```

Figura 2.21 Parte#15 del código de Python.

Finalmente, una vez descritos cada uno de los componentes que integran la programación de todo el proceso se describe a continuación el bucle donde se integran esos componentes.

1. La caja se hace visible y se asigna a un marco estático.
2. El robot Niryo1 se mueve a la posición de recogida.
3. La banda transportadora se mueve y se calcula el ángulo de orientación de la caja.
4. Dependiendo del ángulo, se ajusta la orientación de la caja.
5. Los robots se preparan para recoger la caja.
6. Se toma una foto del código QR de la caja y se decodifica para identificar el tipo de producto.
7. Basado en el código QR, se clasifica la caja y se actualizan los contadores de productos.
8. Los robots mueven la caja a la posición correspondiente y se preparan para la siguiente iteración.

2.6 Algoritmo para el almacenamiento de objetos en la nube

```python
def build_payload(value_1, value_2):
    # Crea valores aleatorios para enviar datos
    #value_1 = 1#random.randint(-10, 50)
    #value_2 = 1#random.randint(0, 85)

    payload = {
        VARIABLE_LABEL_1: value_1,
        VARIABLE_LABEL_2: value_2
    }
    return payload
```

Figura 2.22 Parte#16 del código de Python.

En la Figura 2.22 se define la función build_payload() que toma los valores de los productos tipo 1 y tipo 2 y los organiza en un diccionario con las etiquetas correspondientes.

```python
def post_request(payload):
    url = "http://industrial.api.ubidots.com/api/v1.6/devices/{}".format(DEVICE_LABEL)
    headers = {
        "X-Auth-Token": TOKEN,
        "Content-Type": "application/json"
    }

    response = requests.post(url, json=payload, headers=headers)
    if response.status_code == 200:
        print("Datos enviados correctamente a Ubidots")
    else:
        print("Error al enviar datos a Ubidots")

def sendubidots(val1,val2):
    data_payload = build_payload(val1 , val2)
    post_request(data_payload)
```

Figura 2.23 Parte#11 del código de Python.

En la figura 2.23, se utiliza para el envío del payload, la librería requests para realizar una solicitud HTTP POST a los servidores de Ubidots. La función post_request toma el payload y lo envía a la URL de Ubidots especificada. Se añaden los headers necesarios, incluyendo el token de autenticación y el tipo de contenido. La respuesta del servidor se verifica para confirmar si los datos fueron enviados correctamente.

CAPÍTULO 3

3. RESULTADOS Y ANÁLISIS

En esta sección se analizan los resultados de las etapas definidas en la metodología, tal como se detalla a continuación:

3.1 POSICIONAMIENTO INICIAL DE LAS CAJAS

Durante la etapa inicial del proceso, es necesario encontrar la ubicación exacta del objeto, para ello se utilizó técnicas de visión por computador, como se puede observar en la Figura 3.1, en el escenario ideal la caja llega perfectamente alineada con la banda transportadora, lo cual facilita su análisis, ya que la información que es enviada al primer robot, le indica al mismo que no debe girar su terminal y tan solo debe recoger el producto en un ángulo recto.

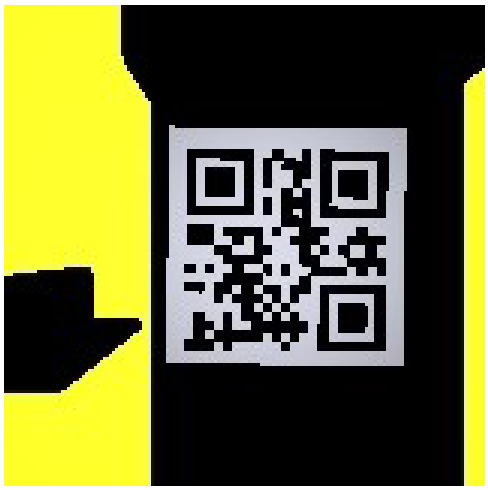

Figura 3.1 Parte#1 del resultado y análisis.

Sin embargo, en el mundo real, esto no sucede siempre, ya que estos productos son colocados en ángulos y posiciones totalmente diferentes, por ello se planteó una segunda opción de ingreso de los objetos la banda transportadora de la estación de paletizado, tal como se muestra en la Figura 3.2, el producto ingresa con cierto ángulo de inclinación.

Figura 3.2 Parte#2 del resultado y análisis.

Uno de los resultados obtenido a través del algoritmo de reconocimiento de objetos, el cálculo de la posición y el ángulo de orientación del producto mediante visión por computador. Para la estimación del ángulo, se puede observar que en efecto la figura 3.3, traza una línea horizontal de referencia dentro de la banda transportadora sobre la cual se están colocando las cajas. Esto con la finalidad de calcular su orientación.

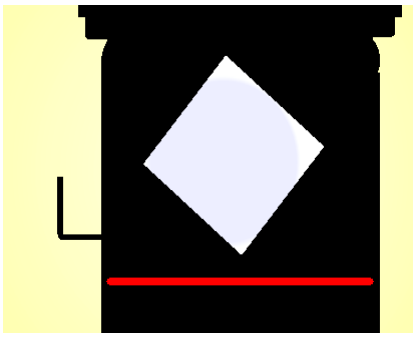

Figura 3.3 Parte#3 del resultado y análisis.

La figura 3.4, muestra cómo se resaltan los vértices del objeto a detectar con ayuda de la librería opencv de Python, luego se dibuja un cuadrado encerrando la figura detectada, tal como se muestra en la Figura 3.5.

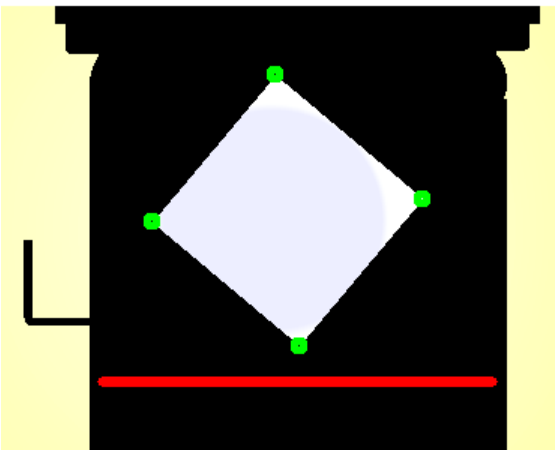

Figura 3.4 Parte#4 del resultado y análisis.

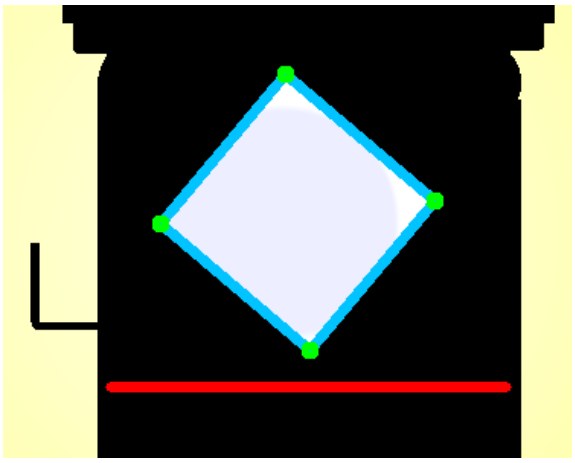

Figura 3.5 Parte#5 del resultado y análisis.

En la siguiente sección del código, se realiza el cálculo del ángulo de inclinación, trazando una línea recta entre dos vértices consecutivos que se encuentren más cercanos a la línea roja horizontal trazado anteriormente, tal como se muestra en la Figura 3.6

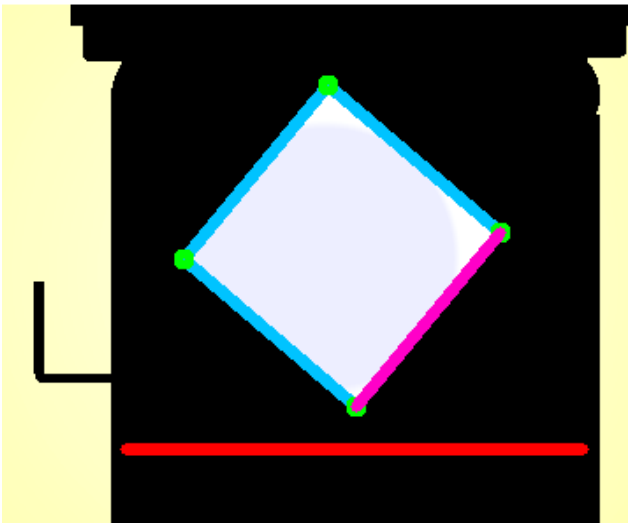

Figura 3.6 Parte#6 del resultado y análisis.

Finalmente, lo que se muestra en la Figura 3.7 es el ángulo calculado entre la línea morada y la línea roja, para enviar el ángulo respectivo al robot para que gire su terminal y recoja correctamente la caja observada.

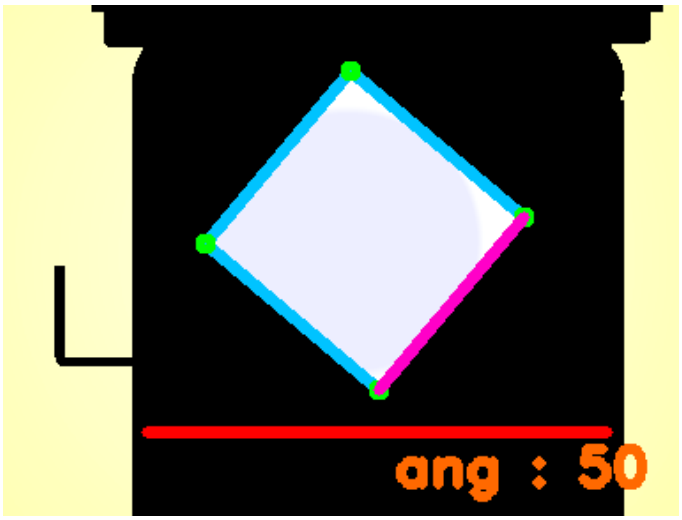

Figura 3.7 Parte#7 del resultado y análisis.

3.2 IDENTIFICACION DE PRODUCTOS Y COLABORACION DE ROBOTS

El proceso inicia con la captura de imágenes de los productos en la línea de producción mediante una cámara, que son luego procesadas para identificar la posición y orientación del objeto utilizando la función camas(bx). Los códigos QR de los productos se escanean y decodifican con la función qrcodefoto(), empleando la biblioteca pyzbar para determinar el tipo de producto (PRODUCTO1 o PRODUCTO2).

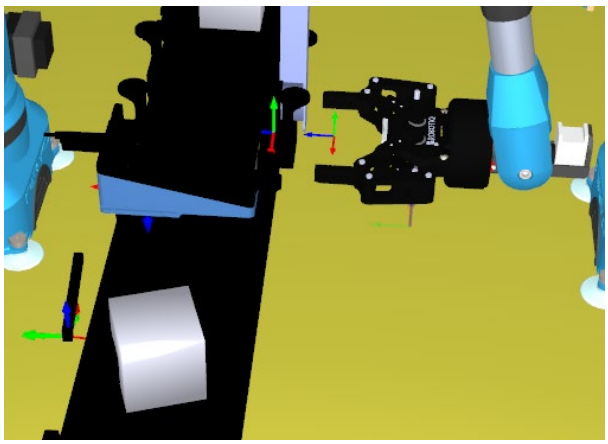

Figura 3.8 Parte#8 del resultado y análisis.

En la Figura 3.8 se muestra cómo se ubica inicialmente el robot1, y como llega una de las cajas, y la cámara sobre dicha caja está calculando la orientación

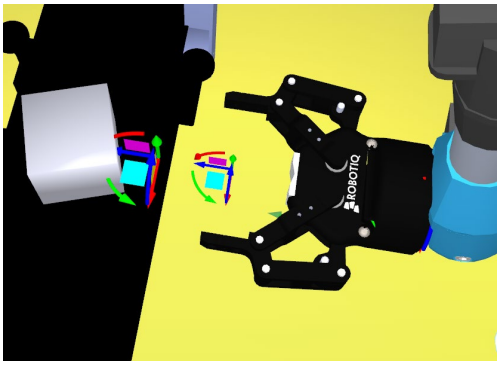

Figura 3.9 Parte#9 del resultado y análisis.

En la Figura 3.9 se aprecia que la caja llega a la posición donde es detectada por un láser y se envía la señal al robot1 para que gire en orientación de la caja, según lo detectado por la cámara 1, al inicio de la banda.

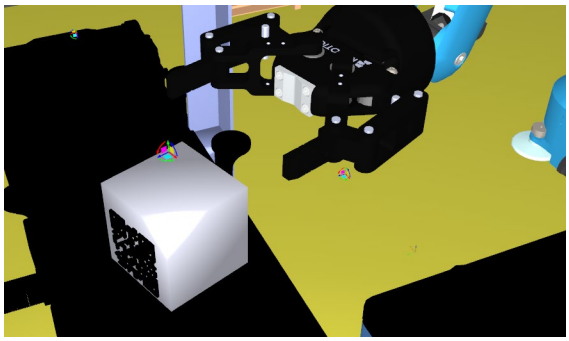

Figura 3.10 Parte#10 del resultado y análisis.

En figura 3.10, se observa que el robot ya ha girado acorde a la orientación de la caja para poder acercarse y recogerla correctamente.

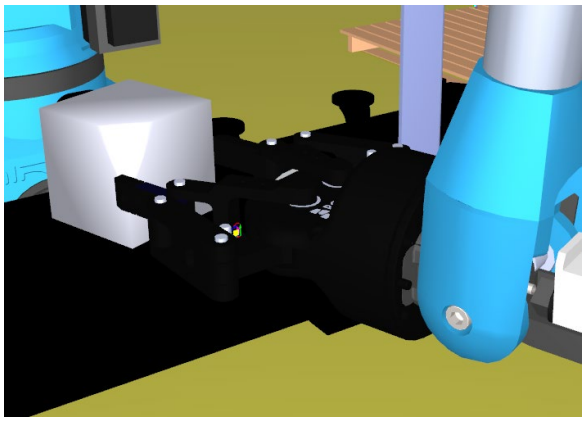

Figura 3.11 Parte#11 del resultado y análisis.

Finalmente se observa en la Figura 3.11 que el robot a alcanzado correctamente al objeto para poder acercarlo a la cámara de detección de qr.

3.3 Desarrollar sistema de visión para el reconocimiento de objetos.

La colaboración entre los robots Niryo One se evidencia cuando el primer robot (Niryo1) recoge y orienta el producto para el escaneo, y si no se puede identificar, lo transfiere al segundo robot (Niryo2) para una segunda oportunidad de escaneo. Los robots trabajan de manera coordinada, moviéndose en función de la identificación del producto.

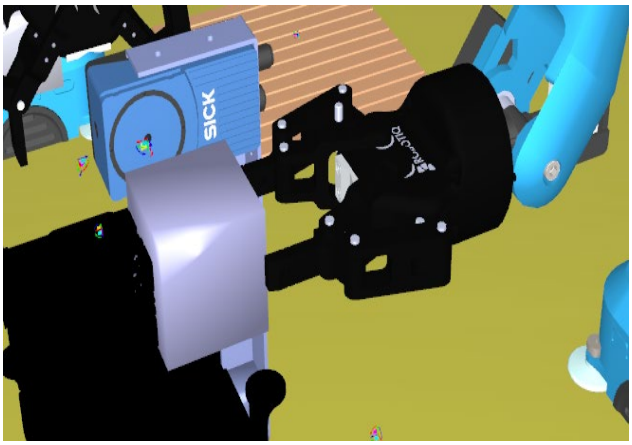

Figura 3.12 Parte#12 del resultado y análisis.

A continuación, en la figura 3.12, se aprecia que el robot1 se mueve la caja a la cámara2 para la detección del código Qr.

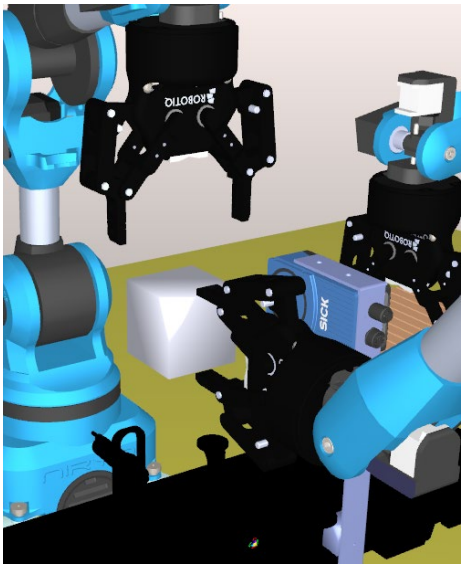

Figura 3.13 Parte#13 del resultado y análisis.

Se observa al segundo robot llegando a recoger la caja, ya que no se ha encontrado el QR en ninguna de las primeras 4 caras de la caja.

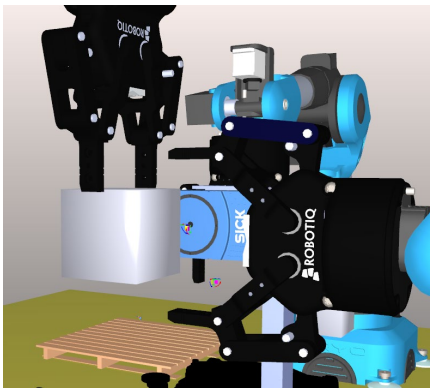

Figura 3.14 Parte#14 del resultado y análisis.

En la figura 3.14, se observa al primer robot alejándose de la caja, y el segundo robot ya sujetando la caja para girar y poder encontrar el código QR. Posteriormente, en la figura 3.15, se muestra como el segundo robot gira la caja de manera correcta para que la cámara pueda detectar el Qr.

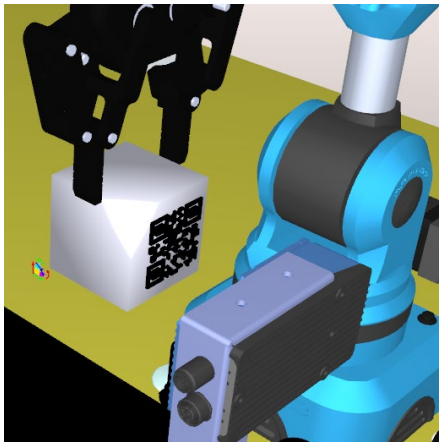

Figura 3.15 Parte#15 del resultado y análisis.

Una vez que el sistema valida el producto, esta información enviada a la nube utilizando el servicio de Ubidots, permitiendo un seguimiento centralizado de la producción, Este proceso asegura la correcta identificación y clasificación de los productos, facilitando la gestión eficiente del inventario y la colaboración efectiva entre los robots.

En la presente imagen se muestra el historial de conteo de cada tipo de producto de la simulación realizada, datos que son observable de cualquier parte del mundo a través del siguiente enlace: https://acortar.link/N3wLdF

Values Table

producto1 (Last value)	producto2 (Last value)
22.00	20.00
21.00	20.00
20.00	20.00
20.00	19.00
19.00	19.00
19.00	18.00
18.00	18.00
17.00	18.00

Figura 3.16 Parte#16 del resultado y análisis.

En la Figura 3.16, Figura 3.18 y Figura 3.19, se puede evidenciar el paletizado final de los productos acorde fueron identificados por el sistema, En esta imagen el robot ya ha recibido la señal del sistema indicando que tipo de producto es, evidenciando en la siguiente imagen que lo va a colocar en el pallet correspondiente según el tipo de producto.

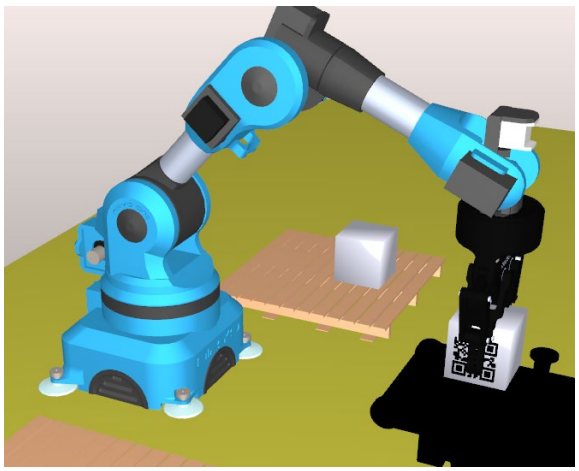

Figura 3.17 Parte#17 del resultado y análisis.

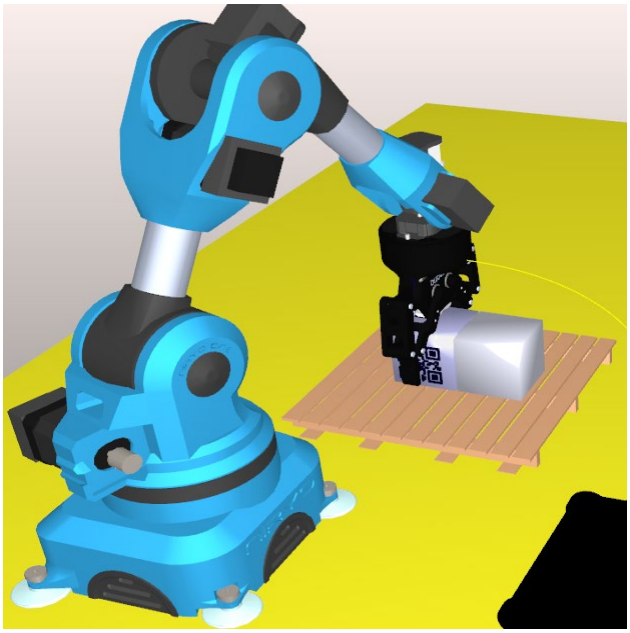

Figura 3.18 Parte#18 del resultado y análisis.

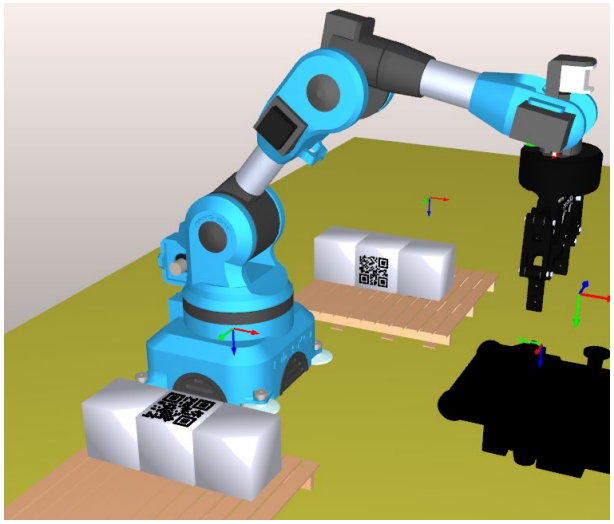

Figura 3.19 Parte#19 del resultado y análisis.

CAPÍTULO 4

4. CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

Para concluir, en este trabajo de tesis se desarrolló un algoritmo con un enfoque específico en un producto previamente definido, considerando sus dimensiones particulares de alto y ancho según las necesidades operativas de la planta. Este enfoque centrado en el producto fue fundamental para garantizar que el algoritmo pudiera ofrecer una precisión y eficiencia óptimas en el procesamiento y reconocimiento de imágenes dentro del entorno industrial específico para el cual fue diseñado.

Desde la perspectiva del procesamiento de imágenes, se implementaron varios pasos meticulosos para asegurar la eficacia del sistema. Uno de los pasos cruciales fue el registro de dos productos en la base de datos. Esta base de datos sirve como referencia fundamental para el sistema, permitiéndole realizar un reconocimiento preciso y eficiente de los productos. La selección y el registro de estos productos se llevaron a cabo con un rigor metodológico, asegurando que los ejemplos almacenados fueran representativos y adecuados para las necesidades de la planta.

El reconocimiento de productos mediante procesamiento de imágenes no solo implica el almacenamiento de datos visuales, sino también la creación de algoritmos robustos que puedan interpretar y comparar estos datos en tiempo real. La capacidad del algoritmo para reconocer los productos registrados en diversas condiciones de iluminación, orientación y posición es una demostración de su robustez y adaptabilidad. Este nivel de precisión y adaptabilidad es esencial para el funcionamiento continuo y sin interrupciones de la planta, ya que reduce significativamente los errores y aumenta la eficiencia operativa.

El diseño del algoritmo también incorporó consideraciones críticas de la planta, como la velocidad de procesamiento y la capacidad para integrarse sin problemas

con los sistemas existentes. La implementación exitosa del algoritmo en la planta demuestra que se lograron estos objetivos, ya que el sistema no solo reconoce los productos con precisión, sino que tampoco interfiere con la producción continua.

Además, la base de datos de productos juega un papel crucial en el aprendizaje continuo del sistema. Con la capacidad de actualizar y expandir esta base de datos, el sistema está diseñado para adaptarse a futuras necesidades de la planta, incluyendo la incorporación de nuevos productos o la modificación de los existentes. Este enfoque garantiza que el sistema siga siendo relevante y útil a medida que las necesidades de la planta evolucionan con el tiempo.

La simulación de este algoritmo ha demostrado ser un paso significativo hacia la automatización avanzada en la planta. No solo mejorará la precisión en el reconocimiento de productos, sino que también permitirá liberar recursos humanos que estaban dedicadas a tareas repetitivas de inspección visual. Estos recursos podrían ser reubicados en áreas que requieren más intervención humana, mejorando así la eficiencia general y la productividad de la planta.

En resumen, el algoritmo diseñado ha cumplido y superado las expectativas iniciales. Su enfoque en un producto previamente definido con dimensiones específicas ha permitido una personalización y una precisión sin precedentes en el reconocimiento de productos mediante procesamiento de imágenes. La inclusión de dos productos en la base de datos como referencia ha sido crucial para el éxito del sistema, proporcionando una base sólida para el reconocimiento preciso. La integración efectiva del sistema en la planta y su capacidad de adaptarse a futuras necesidades será directa y no requerirá mayor trabajo. El sistema propuesto será una herramienta valiosa para la planta en los años venideros.

4.2 RECOMENDACIONES

Para mantener y optimizar el funcionamiento del sistema de reconocimiento de productos mediante procesamiento de imágenes, es fundamental seguir ciertas

recomendaciones en el caso de que se desee agregar productos con tamaños diferentes o nuevos productos. A continuación, se detallan recomendaciones específicas que aseguran que el sistema siga siendo preciso y eficiente:

4.2.1 Adaptación del algoritmo a nuevos tamaños

Cuando se quiera agregar un producto cuyo tamaño difiere de los productos previamente definidos, es necesario modificar el código del algoritmo en Python. Esto se debe a que el algoritmo está optimizado para reconocer productos con dimensiones específicas. Al introducir un producto de tamaño distinto, se requiere ajustar los parámetros del procesamiento de imágenes para mantener la precisión en el reconocimiento.

Identificación de parámetros clave: identificar los parámetros críticos del algoritmo que están relacionados con las dimensiones del producto, como la resolución de la imagen, la escala de análisis y los umbrales de detección.

Pruebas y validación: realizar pruebas exhaustivas con las nuevas dimensiones para asegurar que el algoritmo sigue funcionando correctamente. Esto incluye la captura de imágenes de muestra del nuevo producto y la evaluación del rendimiento del sistema.

4.2.2 Ajuste de la segmentación y detección de bordes

Los métodos de segmentación y detección de bordes pueden necesitar ajustes para manejar el nuevo tamaño del producto. Estos métodos son fundamentales para identificar correctamente las características del producto en la imagen.

Segmentación: modificar los parámetros de segmentación para asegurar que el nuevo producto se aísle correctamente del fondo.
Detección de bordes: ajustar los filtros de detección de bordes para que sean sensibles a las nuevas dimensiones del producto, evitando tanto falsas detecciones como omisiones.

4.2.3 Actualización de la interfaz de usuario

Si el sistema incluye una interfaz de usuario para la visualización o el control de la clasificación de productos, también será necesario actualizarla para reflejar las nuevas dimensiones del producto. Asegurarse de que las representaciones visuales y los controles sean coherentes con el nuevo tamaño del producto.

4.2.4 Inclusión de datos representativos

Cuando se agregue un nuevo producto, es esencial registrarlo en la base de datos con imágenes y características representativas. Este registro permite al sistema reconocer y clasificar el nuevo producto adecuadamente.

Captura de imágenes de alta calidad: asegurar que las imágenes capturadas del nuevo producto sean de alta calidad y en diversas condiciones de iluminación y ángulos.

Atributos y etiquetas: registrar todos los atributos relevantes del producto, como dimensiones, colores y texturas, y etiquetar adecuadamente las imágenes para facilitar el proceso de entrenamiento del algoritmo.

4.2.5 Actualización continua de la base de datos

Mantener la base de datos actualizada con nuevas imágenes y datos es crucial para la evolución del sistema. Esto permite al sistema adaptarse a variaciones en la apariencia de los productos y mantener su precisión a lo largo del tiempo.

Recolección Continua de Datos: Establecer un proceso continuo para recolectar y registrar nuevas imágenes y datos de los productos.
Monitoreo y evaluación: implementar un sistema de monitoreo y evaluación que revise periódicamente el rendimiento del algoritmo y actualice la base de datos según sea necesario.

BIBLIOGRAFÍA

Baturone, A. O. (2005). *ROBÓTICA Manipuladores y robots móviles.* Barcelona: MARCOMBO, S.A.

Intel. (Marzo de 2020). *Intel.la*. Obtenido de https://www.intel.la/content/www/xl/es/robotics/robotic-arm.html#:~:text=Los%20brazos%20rob%C3%B3ticos%20industriales%20ayudan%20a%20las%20empresas%20a%20alcanzar,analicen%20y%20comprendan%20sus%20entornos.

Nikitins, N. (9 de abril de 2023). *Medium*. Obtenido de https://levelup.gitconnected.com/mastering-opencv-with-python-a-comprehensive-guide-for-image-processing-and-computer-vision-732d457ed38

RoboDK. (14 de agosto de 2024). *RoboDK.com*. Recuperado el 2024, de https://robodk.com/doc/es/Robot-Programs.html#:~:text=La%20programaci%C3%B3n%20fuera%20de%20l%C3%ADnea,utilizando%20el%20dispositivo%20de%20programaci%C3%B3n).

Sam, R. (31 de Agosto de 2023). *Ev Tesla*. Obtenido de https://evtesla.tech/overcoming-boundaries-to-palletizing-automation/

Sánchez, J. A. (2002). *AVANCE EN ROBÓTICA Y VISIÓN POR COMPUTADORA* (Vol. 38). Ediciones de la Universidad de Castilla-La Mancha Cuenca,2002.

Sandoval-Gonzalez, O. O. (octubre de 2016). *ResearchGate*. Obtenido de https://www.researchgate.net/publication/321886350_Diseno_de_un_sistema_clasificador_de_objetos_en_movimiento

Ubidots. (2024). *Ubidots*. Obtenido de Ubidots: https://es.ubidots.com/platform