Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

DESARROLLO DE UNA APLICACIÓN ROBÓTICA BASADA EN VISIÓN POR COMPUTADOR PARA GESTIÓN DE TAREAS "PICK AND PLACE"

Alumno: Pablo Martínez de la Rosa

Tutor: Prof. D. ª Silvia Satorres Martínez

D. Ramón Mancebo Garrido

Dpto: Dpto. Ingeniería Electrónica y Automática

Julio, 2020

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

Doña Silvia María Satorres Martínez, tutor del Proyecto Fin de Carrera titulado: Desarrollo de una aplicación robótica basada en visión por computador para gestión de tareas “pick and place”, que presenta Pablo Martínez de la Rosa, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Julio de 2020

El alumno: Los tutores:

PABLO MARTÍNEZ DE LA ROSA SILVIA M. SATORRES MARTÍNEZ

MARTINEZ DE LA ROSA PABLO - 77379139D

Digitally signed by MARTINEZ DE LA ROSA PABLO - 77379139D
DN: C=ES, SERIALNUMBER=IDCES-77379139D, G=PABLO, SN=MARTINEZ DE LA ROSA, CN=MARTINEZ DE LA ROSA PABLO - 77379139D
Reason: I am the author of this document
Location: your signing location here
Date: 2020-07-31 10:36:06
Foxit Reader Version: 9.6.0

SATORRES MARTINEZ SILVIA - 26741651B

Firmado digitalmente por SATORRES MARTINEZ SILVIA - 26741651B
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-26741651B, givenName=SILVIA, sn=SATORRES MARTINEZ, cn=SATORRES MARTINEZ SILVIA - 26741651B
Fecha: 2020.07.31 10:52:21 +02'00'

Índice

RESUMEN x

PALABRAS CLAVE x

ABSTRACT x

KEYWORDS x

Capítulo 1. Introducción 1

1.1 Motivación 1

1.2 Objetivos y alcance 1

1.3 Metodología 2

1.4 Estructura de la memoria 3

Capítulo 2. Estudio previo 4

2.1 Marco teórico 4

2.1.1 Conceptos básicos 4

2.1.2 Calibración 5

2.1.3 Transformación 8

2.1.4 Cálculo de la transformación rígida 9

2.1.5 Relación entre los distintos sistemas de coordenadas 9

2.1.6 Obtención de la coordenada en Z 10

2.1.7 Tecnología Intel® RealSense™ 11

2.2 Estado del arte 14

2.2.1 Sensores 14

2.2.2 Tecnología para la obtención de nubes de puntos 14

2.2.3 Software y aplicaciones industriales 15

Capítulo 3. Desarrollo del proyecto 17

3.1 Aspectos generales 17

3.2 Herramientas de hardware y software del entorno de trabajo 17

3.3 Estudio de la solución para la aplicación de *pick and place* 18

3.3.1 Modos de obtención de imagen 18

3.3.2 Programa principal **principal.py** 18

3.3.3 ProcesarFrame() 19

3.3.4 CambioDeBase() 21

3.3.5 CargarModuloABB() 22

3.4 Estudio de la solución para la aplicación de *picking* 23

3.4.1 Calibración 23
3.4.2 Generación de la nube de puntos 26
3.4.3 Obtención de coordenadas del centroide del objeto 27
3.4.4 Consecución del *Workobject* de la pieza y cálculos para obtener la rotación de la pieza 27
3.4.5 Código RAPID 28
3.5 Integración del sistema 28
3.5.1 RobotStudio 28
3.5.2 Comunicación 30
3.5.3 Simulación en RobotStudio 30
3.5.4 FlexPendant 33
Capítulo 4. Experimentos y resultados obtenidos 34
Capítulo 5. Conclusiones y trabajo futuro 38
5.1 Conclusiones del proyecto 38
5.2 Trabajo futuro 38
Capítulo 6. Bibliografía 40
ANEXO I: DIAGRAMAS DE FLUJO 42
1. Diagramas de flujo visión 2D 42
2. Diagramas de flujo visión 3D 51
ANEXO II: CARACTERÍSTICAS DE LA CÁMARA INTEL REALSENSE D415 60
ANEXO III: VISIÓN 3D, ASPECTOS MAS IMPORTANTES 62
1. Tipos de datos en 3D 62
2. Proceso de reconstrucción 3D y alineamiento 62
3. Segmentación 66
4. Reconocimiento de objetos en 3D 67
ANEXO IV: PRESUPUESTO 70

Índice de figuras

Figura 1: Modelo "pinhole" simplificado 4
Figura 2: Relación punto real y proyectado 5
Figura 3: Distorsión radial 6
Figura 4: Distorsión tangencial 6
Figura 5: Patrones conocidos: tablero de ajedrez, puntos y marcadores ArUco (de izquierda a derecha) 7
Figura 6: Posiciones de la cámara respecto a una escena 7
Figura 7: Rotación de un objeto respecto al plano X e Y de la imagen 9
Figura 8: Relación entre transformaciones 10
Figura 9: Muestra de la captura L y R de las cámaras estéreo de la Intel RealSense D415 10
Figura 10: Figura profundidad 11
Figura 11: Diferencia entre imagen a color y mapa de profundidad 11
Figura 12: Herramienta Intel Realsense Viewer. Panel para modificar los parámetros de cada sensor: módulo estéreo y cámara RGB 12
Figura 13: Visualizador en 2D de la adquisición de los distintos sensores 12
Figura 14: Etapas sistema de visión estéreo 13
Figura 15: Emisión en el espectro infrarrojo, perceptible para otras cámaras, pero no para el ojo humano 13
Figura 16: Técnicas 3D 14
Figura 17: Sistema iRVision de Fanuc 15
Figura 18: Intel RealSense D415 17
Figura 19: Montaje sistema de visión 17
Figura 20: Muestra de color en el espacio de trabajo 18
Figura 21: Posición de la muestra con una circunferencia y sistema de coordenadas del plano de trabajo en verde 19
Figura 22: Imagen HSV segmentada y contorno encontrado 20
Figura 23: Pieza posición favorable vs. pieza posición desfavorable 23
Figura 24: Conjunto de puntos 24
Figura 25: Conjunto de puntos emparejados tras aplicar traslación 24
Figura 26: Resultado de aplicar algoritmo de Kabsch. (a) representa dos vectores en sus posiciones originales, en (b) los vectores se trasladan y se centran uno respecto al otro y en (c) los vectores son rotados. 25
Figura 27: Nube de puntos del objeto. (a) Vista con perspectiva. (b) Vista cenital 27
Figura 28: Nube de puntos reducida. (a) Vista con perspectiva. (b) Vista cenital 27
Figura 29: Etapas de detección de agarre robótico 28
Figura 30: Rango de trabajo IRB120 29
Figura 31: Estación 29
Figura 32: Intento de conexión con el controlador del robot físico 30
Figura 33: Excel con la IP del robot 30
Figura 34: Diferencia entre las tres formas de desplazamiento 31
Figura 35: Captura del diseño de lógica de estación 32
Figura 36: Captura de señales y conexiones de lógica de estación 32
Figura 37: FlexPendant con las instrucciones en RAPID del movimiento del robot 33
Figura 38: Simulación de prueba con y sin éxito en el posicionamiento de la pieza en el contenedor de dejada 34

Figura 39: Contornos bien definidos gracias a la segmentación en HSV 34
Figura 40: Error en la detección del contorno debido a la falta de iluminación 35
Figura 41: Orientación de la pieza paralela con la horizontal 35
Figura 42: Orientación de la pieza 30º con respecto a la horizontal 36
Figura 43: Orientación de la pieza -30º con respecto a la horizontal 36
Figura 44: Orientación de la pieza en vertical rotado 30º con respecto a la horizontal 37
Figura 45: Pick and place, sistema 2D 37
Figura 46: D415 vs. D435 61
Figura 47: Tipos de estructuras de datos en 3D (I) 62
Figura 48: Tipos de estructuras de datos en 3D (II) 62
Figura 49: Sistema de binpicking 63
Figura 50: Secuencia de imágenes RGB y profundidad 63
Figura 51: Creación de fragmentos a partir de secuencias 64
Figura 52: A partir de capturas en RGBD se obtiene el 3D de la escena en forma de nube de puntos. El trazo azul de la izquierda es el recorrido en forma de barrido que ha hecho la cámara sobre la escena 64
Figura 53: Puntos comunes en cada scan de la escena 65
Figura 54: Scan Matching 65
Figura 55: Árbol Kd 66
Figura 56: RANSAC 67
Figura 57: Transformada de Hough 67
Figura 58: Reconocimiento de objetos en tiempo real 68
Figura 59: Extracción de características 69
Figura 60: Computa descriptores 69
Figura 61: Cálculo de emparejamientos 69

Índice de tablas

Tabla 1: Transformaciones 8
Tabla 2: Comparación entre varios modelos 60
Tabla 3: Presupuesto hardware 70

Índice de algoritmos

Algoritmo 1: Cambio de base 21
Algoritmo 2: Signo 22

Índice de flujogramas

Flujograma 1: Programa principal 42
Flujograma 2: ProcesarFrame 43
Flujograma 3: ContornosConMuestra 44
Flujograma 4: Segmentacion 45
Flujograma 5: CallBackFunc 46
Flujograma 6: AbrirCamaraIntel 47
Flujograma 7: CapturaCamaraIntel 48
Flujograma 8: CambioDeBase 49
Flujograma 9: CodigoRAPIDCentroides 50
Flujograma 10: Signo 50
Flujograma 11: principal 51
Flujograma 12: continuación principal 52
Flujograma 13: PoseEstimation 53
Flujograma 14: obtiene puntos en 3D del patrón de ajedrez 54
Flujograma 15: calcula transformacion Kabsch 54
Flujograma 16: convierte coordenadas de píxeles a valor métrico 54
Flujograma 17: calcula nube de puntos acumulada 55
Flujograma 18: procesa frame de profundidad 55
Flujograma 19: convierte frame de profundidad a nube de puntos 56
Flujograma 20: obtiene nube de puntos recortada 56
Flujograma 21: calcula los puntos del bounding box 57
Flujograma 22: centroide 58
Flujograma 23: obtiene Workobject objeto 59

RESUMEN

Este proyecto aborda la tarea de manipulación a través de la robótica guiada por visión (VGR) dentro del entorno industrial. Se va a desarrollar en torno a dos conceptos principales, los cuales denotaremos como aplicación de *pick and place* la cual necesita de información en dos dimensiones, para ello se aplicarán técnicas de visión 2D. Y una aplicación de *picking,* que a través de la visión 3D obtiene información tridimensional de la escena para así poder localizar y obtener la posición de objetos. La visión 2D es fácilmente abordable a través de una captura del mundo real (3D) y su representación en una escena bidimensional, en la que hay una correspondencia entre puntos y píxeles en la escena. Para ello se emplea una cámara matricial, que registra los píxeles de la escena y, mediante el procesado de esta información, determina el centroide de los objetos presentes. En la parte que corresponde a visión 3D, se necesita información sobre la profundidad del mundo real. Aquí entra en juego otro sensor llamado de profundidad o *depth,* y al conjunto de información RGB y de profundidad adquiere el nombre de *RGB-depth* o *RGB-d.*

Se pueden diferenciar tres tareas principales en la detección del agarre de un robot basado en visión, que son localización del objeto, estimación de la posición de ese objeto y estimación del agarre. En el presente proyecto se ha empleado métodos tradicionales de visión por computador, pero también están los métodos basados en *Deep Learning*. Todo esto será abordado con mayor profundidad en el estado del arte.

PALABRAS CLAVE

Visión por computador, *pick and place*, *picking*, 2D, 3D, Opencv, Realsense, nube de puntos, ABB, robótica, RobotStudio.

ABSTRACT

This project approaches the task of manipulation through vision-guided robotics (VGR) within the industrial environment. It will be developed around two main concepts, which we will denote as a pick and place application which needs information in two dimensions, for which 2D vision techniques will be applied. And a picking application, which through the 3D vision obtains three-dimensional information of the scene in order to locate and obtain the position of objects. The 2D vision is easily approached through a snapshot of the real world (3D) and its representation in a two-dimensional scene, in which there is a correspondence between points and pixels in the scene. This is done using a matrix camera, which records the pixels of the scene and, by processing this information, determines the centroid of the objects. In 3D vision task, depth information of the real world is required in addition to that described above. Another sensor called depth comes into play here, and RGB and depth information set is called RGB-depth or RGB-d.

Three main tasks can be differentiated in vision-based robot grip detection, which are object location, object pose estimation and grasp estimation. In the present project, traditional computer vision methods have been used, but there are also methods based on Deep Learning. All of this will be tackled more deeply in the state of the art.

KEYWORDS

Computer vision, pick and place, picking, 2D, 3D, Opencv, Realsense, point cloud, ABB, robotics, RobotStudio.

Capítulo 1. Introducción

La robótica es uno de los campos de la industria 4.0 que está en continua evolución y auge. Por ello cada vez más se requieren de robots autónomos, capaces de desarrollar tareas complejas o difíciles de llevar a cabo por humanos, como pueden ser procesos de soldadura de elementos estructurales, manipulación de reactivos peligrosos en la industria química, etc. Por ello se requiere de otros campos de la industria para prestarle de esa autonomía. La disposición en el robot de elementos sensoriales, provoca la realimentación del proceso y con ello se puede obtener información del medio en el cual está trabajando ese robot. Los *cobots* o robots colaborativos, incorporan sensores de fuerza para detectar cualquier obstrucción o intrusión, provocando la detención automática.

Otra forma de reconocer el entorno y las partes del mismo, llegando incluso a detectarlo y clasificarlo, es mediante la integración de sensores de visión en los procesos productivos.

> La visión por computador o visión artificial es la disciplina que trata la adquisición, procesamiento y análisis de imágenes, de las cuales obtiene información para posteriormente almacenarla y procesarla. En sus inicios, la tecnología 2D era la más extendida, pero en la actualidad se han mejorado los sensores y algoritmos para trabajar con tecnología 3D, la cual permite obtener información de la escena como la localización de los objetos, texturas y altura o profundidad. Esto es gracias a la obtención de datos en 3D, como por ejemplo la nube de puntos, con sensores como Realsense [1], Ensenso [2], Photoneo [3], y más.

El desarrollo de estos sistemas permite la realización de tareas de mayor complejidad. Así se consigue analizar la superficie de trabajo del robot, detectar los objetos e identificar su posición. Esto a priori es una tarea que puede desarrollar una persona de forma trivial, pero para un sistema robótico puede llegar a suponer un gran desafío, debido a la aleatoriedad y el desorden con el que vienen las piezas en un entorno industrial, por lo que un sistema de visión es crucial para que el brazo o brazos robóticos dispongan cada vez, de las nuevas coordenadas y posiciones de agarre.

1.1 Motivación

Hoy en día, los sistemas están muy avanzados y existen numerosos algoritmos de visión, testeados y preparados para funcionar. Hay gran cantidad de sensores comerciales, disponibles según la aplicación a desarrollar, tales como cámaras lineales y matriciales, 3D y perfilómetros, infrarrojas/térmicas, espectrales, de alta velocidad e incluso cámaras inteligentes. Además, existen ya soluciones de robots con sistema de visión incorporado como el TM5M-700 de TM-robot.

La principal motivación de este trabajo fin de grado, es la programación industrial, tanto de los algoritmos de visión para la localización de los objetos como la programación de los movimientos del robot.

1.2 Objetivos y alcance

Se va a desarrollar a continuación los objetivos de éste TFG. La idea es obtener al final una integración optima entre los diferentes sistemas y adquirir un conocimiento y desarrollar un software apropiado para ponerlo en práctica en un sistema real.

- Estudio de documentación relacionada:

  - Librería de código abierto *Opencv* con sus funciones y clases [4]. Y programación en lenguaje *Python*
  - Librería *librealsense* de la cámara *Intel Realsense D415* [1] (funciones y clases), con su código libre del repositorio de *Github* [5].
  - Artículos de investigación relacionados con la robótica basada en visión.
  - Artículos de investigación sobre calibración, segmentación, detección, localización, estimación de la posición y agarre.
  - Algoritmos de visión 2D y 3D.
- Diseño e implementación de un sistema 2D de detección de objetos y sus centroides para el guiado robótico.
- Diseño de un entorno virtual de controlador más robot para la tarea de *pick and place.*
- Programación de módulo en lenguaje *RAPID* para las tareas de movimiento del robot.
- Simulación en software para una visualización más o menos real y fidedigna con respecto a una "puesta en marcha" de ambos sistemas integrados.
- Calibración intrínseca y extrínseca de la cámara, para poder pasar de un sistema de coordenadas mundo al sistema de coordenadas de la cámara y al sistema de coordenadas en píxeles.
- Diseño e implementación de un sistema 3D, obteniendo datos como las nubes de puntos y conocer los algoritmos para obtener la estimación de la 'pose' del objeto.
- Evaluación del sistema y búsqueda de las mejoras de la aplicación.

1.3 Metodología

Para llevar a cabo éste TFG de investigación, se hará una revisión completa de la documentación publicada en repositorios de *GitHub,* así como *papers* de robótica basado en visión. También se realizará una revisión bibliográfica de manuales de visión por computador tales como aprendizaje de OpenCV [4]. Para ver la viabilidad del proyecto habrá que comparar el sistema propuesto con otras metodologías de la literatura relacionada, así se obtendrán unas conclusiones y un punto de partida del que empezar a diseñar y desarrollar la idea.

Se consultará webs de venta de sistemas de visión y aplicaciones tales como Keyence [6], Infaimon [7], Cognex [8], bcnvision [9], además se asistirá de forma online a *webinars* realizados por algunas de éstas empresas para profundizar en conceptos de visión 3D.

Tanto la programación del sistema basado en visión 2D como para el sistema 3D se usarán métodos tradicionales de visión, ya que el aprendizaje profundo requiere de gran poder de computación debido los niveles jerárquicos de las redes neuronales, que hace que sea un proceso lento y que haya que adquirir un gran volumen de imágenes para poder entrenar a la red y posteriormente, con otro banco de imágenes, usarlas de muestra para hacer pruebas de reconocimiento de objetos, por ejemplo.

Este proyecto está enmarcado en una situación de pandemia global debido al coronavirus COVID-19, es por ello que en un inicio estaba planteado como un trabajo de programación industrial, tanto la visión por computador y la robótica, con "puesta en marcha" para comprobar la bondad de los resultados obtenidos en la fase de programación. Tuvo que modificarse la parte de "puesta en marcha" con un robot y un sistema de visión real por la simulación de los distintos procesos, lo cual hace que etapas como la estimación del agarre robótico no pueda llevarse a cabo, debido a que es necesario para poder configurar la herramienta, creación de espacios de trabajo y calibración del TCP un robot físico. También existen limitaciones, ya que en el laboratorio se puede dejar configurada la cámara en un

utillaje y no es necesario la referencia de coordenadas cada vez que se quiere probar, sin embargo, se va a tener que contemplar y suplir en el código, las coordenadas en las cuales se va a encontrar la zona de trabajo cada vez que se quiera probar, ya que se empleará un trípode para la sujeción de la cámara.

Al estar frente a un proyecto más teórico, se hará uso del método experimental para la obtención de resultados acordes al tema elegido y obtener resultados que den solución a éste TFG que lleva por título desarrollo de una aplicación robótica basada en visión por computador para gestión de tareas "pick and place".

1.4 Estructura de la memoria

Esta memoria está estructurada de la siguiente forma:

- Capítulo 2. Estudio previo: se hará una revisión de conceptos básicos de visión y se introducirá la tecnología de la cámara Intel.
  El estado del arte trata en que punto está la tecnología actual y a nivel comercial qué empresas existen en el mercado y qué ofrecen. Además, se tratan los sensores que existen y los sistemas de visión más utilizados en la industria junto a las diferentes técnicas de visión 3D.
- Capítulo 3. Desarrollo del proyecto: aquí se empieza a tratar la metodología empleada para la programación de los sistemas de *pick and place* y *picking.*
- Capítulo 4. Experimentos y resultados obtenidos: en este capítulo se desarrolla toda la parte experimental y la realización de ensayos en simulación para comprobar como interactúan los sistemas de visión y robótica.
- Capítulo 5. Conclusiones y trabajo futuro: se presentan una serie de conclusiones obtenidas en el transcurso del proyecto y se trata del trabajo futuro para mejorar la aplicación e implementar nuevas características.

Capítulo 2. Estudio previo

Este proyecto engloba dos disciplinas dentro de la industria 4.0, la visión por computador y la robótica industrial. Por su carácter multidisciplinar, es necesario definir una serie de conceptos básicos, útiles para seguir el flujo de los programas, así como para tener una visión general de las distintas etapas de la programación de un sistema de visión junto a sus algoritmos.

2.1 Marco teórico

Existen varias metodologías a seguir a la hora de localizar un objeto en una escena, vamos a analizar las diferentes etapas las cuales se han recogido en la etapa de estudio de la documentación.

2.1.1 Conceptos básicos

Las coordenadas en una imagen se definen en píxeles. El píxel es la menor unidad homogénea que forma parte de una imagen digital [10].

La distancia focal, denotada por f, es un parámetro que relaciona el tamaño de la imagen con el objeto distante.

El eje óptico es un eje imaginario que define la ruta que seguirá la luz cuando se propague a través de la lente de la cámara.

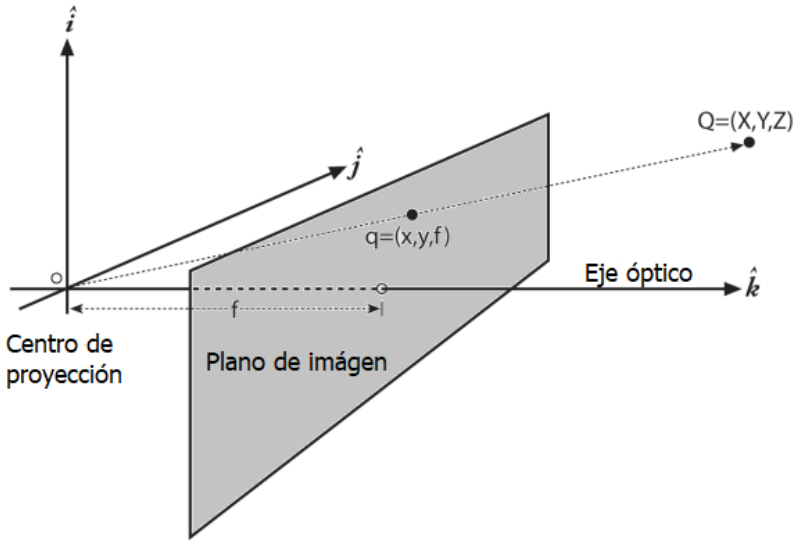


Figura 1: Modelo "pinhole" simplificado

El eje óptico(vector unitario $\vec{K}$ de la Figura 1) no tiene por qué coincidir con el centro del chip de la cámara, por ello surgen dos nuevos parámetros que modelan el desplazamiento del eje óptico: c_x y c_y o también se puede encontrar en la bibliografía como ppx y ppy.

Más arriba se ha definido la distancia focal, pero realmente no existe una sola distancia focal, ya que en realidad los píxeles no son cuadrados, sino rectangulares: fx y fy

$$f_x = f\, s_x \qquad (1)$$

$$f_y = f\, s_y \quad (2)$$

Siendo f la distancia focal física de la lente medida en mm, y $s_{x,y}$ el tamaño de los elementos individuales de la cámara medido en pixel/mm. Tanto f como $s_{x,y}$ no se pueden obtener directamente a través del proceso de calibración de la cámara, pero si se obtienen f_x y f_y.

2.1.2 Calibración

El proceso de calibración, cuando vamos a trabajar con un sistema de visión, es fundamental, ya que entran en juego una serie de parámetros intrínsecos y extrínsecos de la cámara. Los parámetros internos o intrínsecos nos proporcionan un modelo de la geometría de la cámara, así como un modelo de distorsión de la lente. La matriz intrínseca de la cámara es un matriz 3x3 (ecuación 3), la cual está compuesta por los parámetros definidos anteriormente:

$$M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (3)$$

La proyección de los puntos del mundo físico en la cámara se puede expresar como:

$$\vec{p} = M \cdot \vec{P} \quad (4)$$

Donde $\vec{p} = \begin{bmatrix} x \\ y \end{bmatrix}$ y $\vec{P} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$

A esta relación entre un punto $\vec{P}_i$ del mundo físico y su proyección en la pantalla, se ilustra en la Figura 2, es denominado transformación proyectiva y es conveniente utilizar coordenadas homogéneas para trabajar con estas transformaciones.

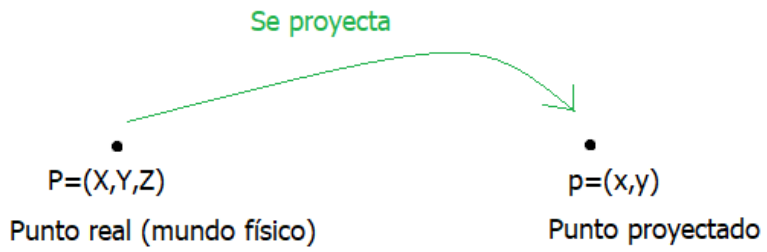


Figura 2: Relación punto real y proyectado

En la práctica es muy complejo fabricar una lente parabólica para evitar la distorsión, por lo que se fabrican con forma esférica. También es difícil alinear lente y cámara. Por lo que produce distorsión en la imagen de dos tipos:

- Radial: debido a la forma de la lente.

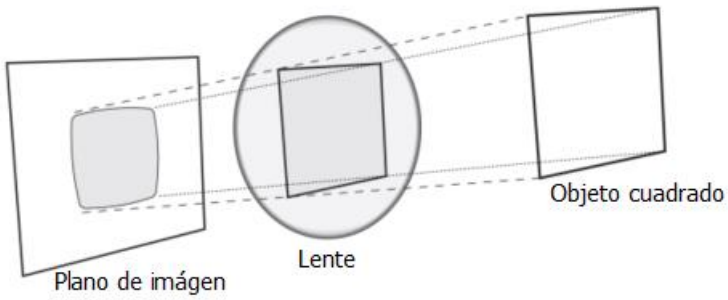


Figura 3: Distorsión radial

- Tangencial: debido al proceso de montaje de la cámara.

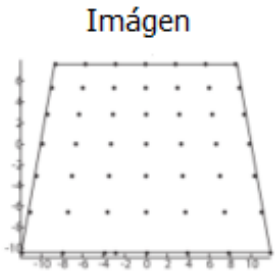


Figura 4: Distorsión tangencial

El vector que modela la distorsión se define como:

$$Vector\ de\ distorsión = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3) \qquad (5)$$

Para obtener los parámetros internos y de distorsión se suele emplear patrones conocidos como un tablero de ajedrez, una disposición de puntos, marcadores ArUco [11] o una mezcla de varias, como se muestra en la Figura 5.

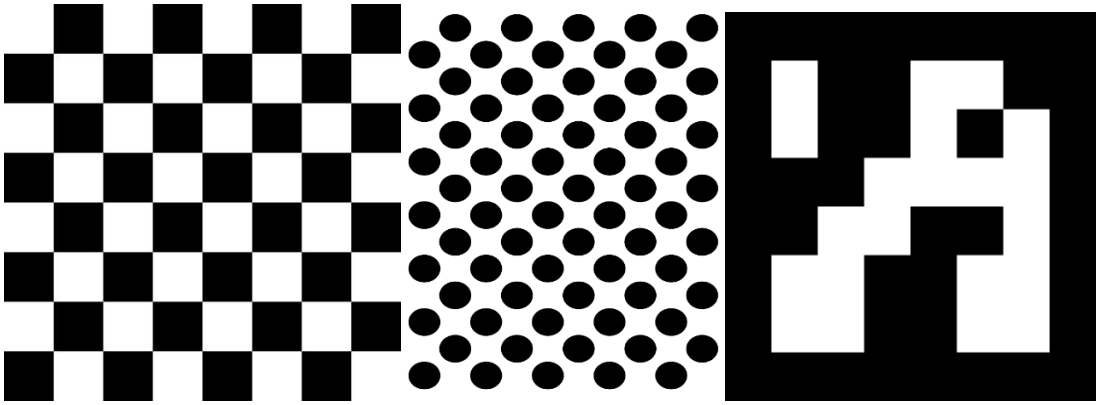


Figura 5: Patrones conocidos: tablero de ajedrez, puntos y marcadores ArUco (de izquierda a derecha)

Usando las funciones de OpenCV ***findChessboardCorners(), calibrateCamera()*** y el patrón de ajedrez se obtienen los parámetros deseados.

Los parámetros extrínsecos o externos de la cámara nos proporcionan la posición de la cámara respecto a la escena que se está adquiriendo, a cada toma desde un punto de vista distinto se le denomina 'pose' de la cámara (Figura 6).

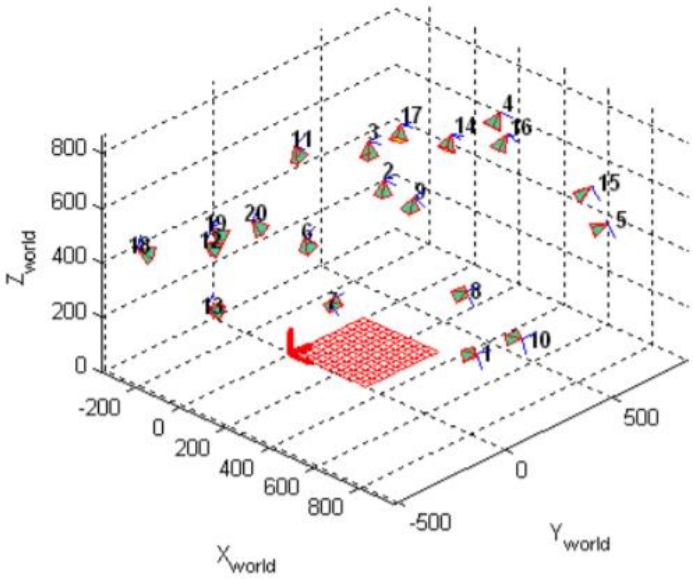


Figura 6: Posiciones de la cámara respecto a una escena

2.1.3 Transformación

Es un proceso a través del cual se obtiene la posición de los puntos de una escena en el mundo físico y se transforman al sistema de coordenadas de la cámara. A continuación, se muestra en la Tabla 1 las distintas transformaciones que existen:

TRANSFORMACIÓN	MATRIZ	DoF	CONSERVA	GRAFISMO
Translación	$[I/t]_{3x4}$	3	Orientación	
Rígida (Euclidiana)	$[R/t]_{3x4}$	6	Longitudes	
Similitud	$[sR/t]_{3x4}$	7	Ángulos	
Afín	$[A]_{3x4}$	12	Paralelismos	
Proyectiva	$\left[\tilde{H}\right]_{3x4}$	15	Líneas rectas	

Tabla 1: Transformaciones

La transformación rígida o Euclidiana, implica una traslación más una rotación. Se definen a través de la matriz de rotación y el vector de translación. Las ecuaciones 6, 7, 8 y 9 hacen referencia a la rotación, mientras que la ecuación 10 es la fórmula que define la translación:

$$R = Rx\psi \cdot Ry\varphi \cdot Rz\theta \quad (6)$$

Siendo:

$$R_x(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & \sin\psi \\ 0 & -\sin\psi & \cos\psi \end{bmatrix} \quad (7)$$

$$R_y(\varphi) = \begin{bmatrix} \cos\varphi & 0 & -\sin\varphi \\ 0 & 1 & 0 \\ \sin\varphi & -\sin\psi & \cos\varphi \end{bmatrix} \quad (8)$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (9)$$

$$T = Coordenadas\ origen_{objeto} - Coordenadas\ origen_{cámara} \quad (10)$$

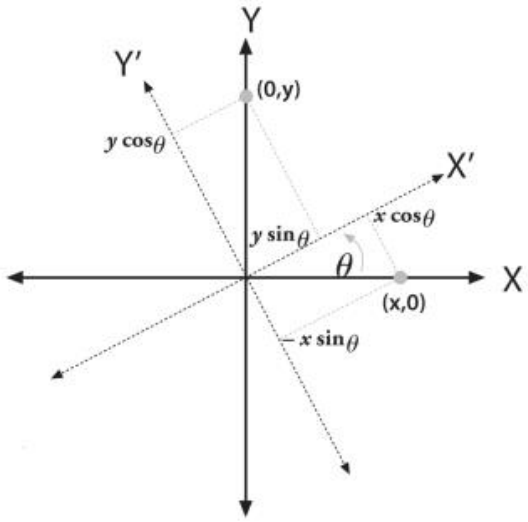


Figura 7: Rotación de un objeto respecto al plano X e Y de la imagen

La Figura 7 muestra una rotación antihoraria de los ejes de coordenadas X e Y, con un ángulo theta positivo.

2.1.4 Cálculo de la transformación rígida

Existen dos opciones a la hora de implementarlo:

- Solución cerrada: para cuando se conoce la correspondencia entre los puntos.
    a. Usando Descomposición en Valores Singulares de una matriz (SVD)
    b. Usando matrices ortonormales
    c. Usando cuaternios
- Solución iterativa: para cuando se desconoce la correspondencia entre puntos.
    a. Algoritmo *Iterative Closest Points* (ICP), está basado en la correspondencia entre puntos *point to point.*

Los cuaternios es un par ordenado, construido a partir de ocho parámetros reales. Debido a que las transformaciones rígidas tienen seis grados reales de libertad, los cuaterniones dobles incluyen dos restricciones algebraicas [12]. Aunque se ha optado por la solución de descomposición de valor singular usando el algoritmo de Kabsch, por la simplicidad del método (explicado en el apartado 3.4.1)

2.1.5 Relación entre los distintos sistemas de coordenadas

Se ha mencionado anteriormente todas las transformaciones que existen. Como aclaración se presenta en la Figura 8 las dos transformaciones que se van a llevar a cabo en la ejecución del programa de visión:

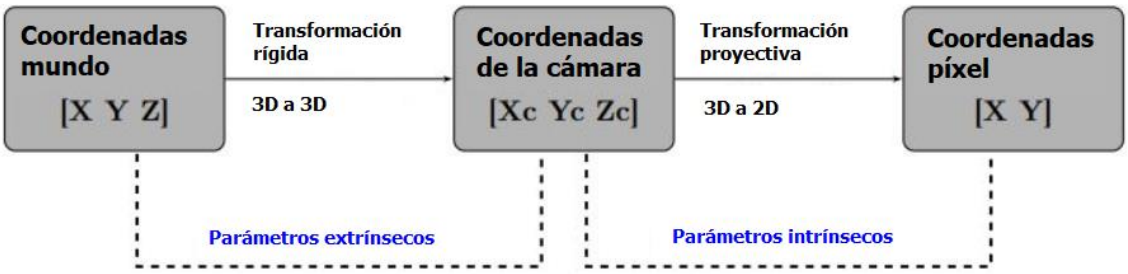


Figura 8: Relación entre transformaciones

2.1.6 Obtención de la coordenada en Z

A cada pixel de una imagen RGB, le corresponde un valor en la matriz de la imagen referenciada a la fila y columna y un valor de intensidad en cada matriz de color. Pero, ¿cómo podemos obtener el valor en Z de cada píxel de la imagen? La tecnología 3D se ocupa de esto, gracias a que, a través de diferentes técnicas, se verán a continuación en el estado del arte, se puede obtener la distancia a la que se encuentra un punto de un objeto de la escena a la cámara. La técnica de obtención de imágenes en 3D, estéreo activo, se compone de dos cámaras y un proyector infrarrojo, las imágenes que se obtienen son las representadas en la Figura 9.

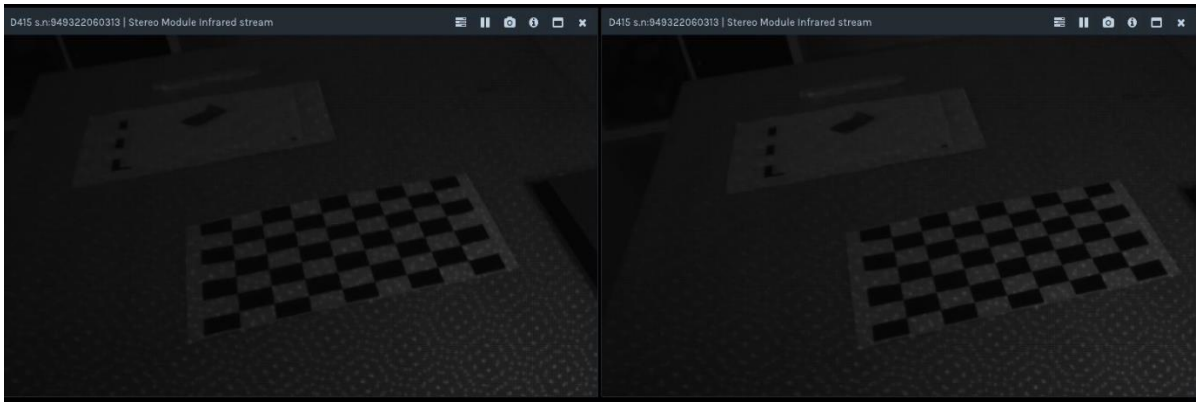


Figura 9: Muestra de la captura L y R de las cámaras estéreo de la Intel RealSense D415

El patrón de puntos blancos que se aprecian en las imágenes es el resultado del proyector infrarrojo sobre la escena. Las dos imágenes pertenecen a la captura de los sensores de la cámara situados a la izquierda y derecha. Para obtener la profundidad o *Depth,* los puntos de la izquierda se correlacionan con los de la derecha y calcula el valor de profundidad para cada pixel.

$$Profundidad = \frac{Distancia\ entre\ los\ sensores\ \cdot Distancia\ focal}{Disparidad} \qquad (11)$$

La disparidad se refiere a la distancia entre un punto de la imagen izquierda y su correspondiente de la imagen derecha de un par estéreo. Posteriormente se triangula y se obtiene la coordenada Z.

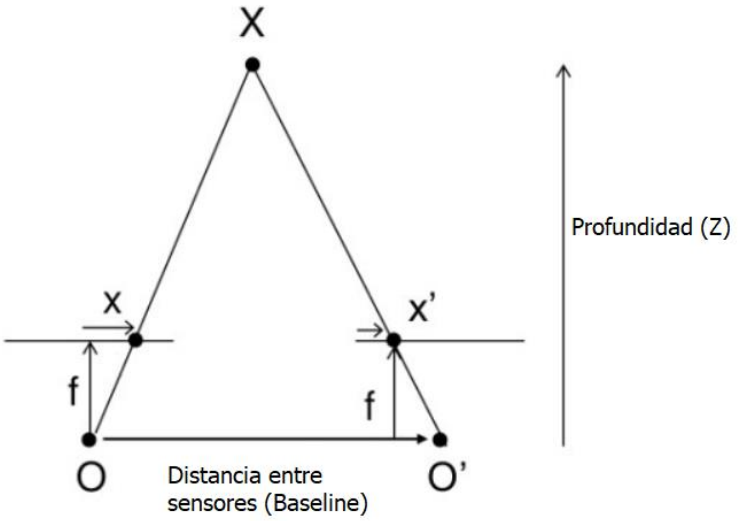


Figura 10: Figura profundidad

2.1.7 Tecnología Intel® RealSense™

Estas cámaras desarrolladas por Intel, con la cual se va a experimentar en este TFG, realizan todo el proceso descrito anteriormente sin tener que hacer cálculos de parámetros intrínsecos, distancia entre sensores, etc. Ya que vienen con estos parámetros definidos según la construcción de la cámara y del modelo. Por lo que se va a hacer una revisión de las etapas que realiza la cámara para entregar el resultado final, que es un tipo de imagen llamado mapa de profundidad en el cual a través de distintos colores muestra la profundidad en metros de los objetos en una escena. se muestra un ejemplo en la Figura 11. Las zonas coloreadas de rojo están a una distancia de un metro, mientras que las zonas amarillas están a medio metro con respecto a la cámara. Cada pixel en el mapa corresponde al mismo pixel en la imagen, pero el cambio de color no está asociado a la percepción de color de la imagen, sino que corresponde a la profundidad en ese punto.

Intel posee una interfaz para cambiar los parámetros de la cámara a nuestra elección, así como visualizar los cambios que hemos realizado en tiempo real de cada sensor, ver Figura 12 y 13.

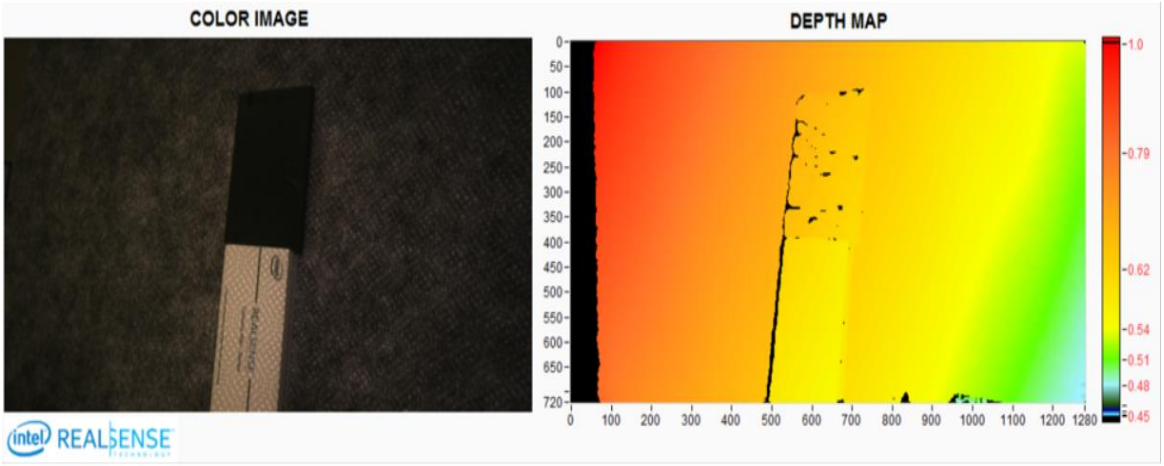


Figura 11: Diferencia entre imagen a color y mapa de profundidad

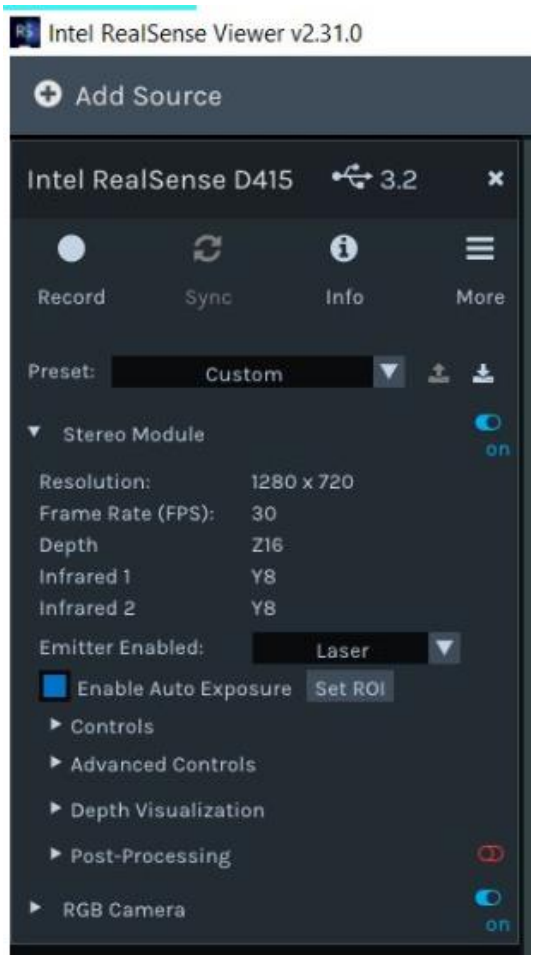


Figura 12: Herramienta Intel Realsense Viewer. Panel para modificar los parámetros de cada sensor: módulo estéreo y cámara RGB

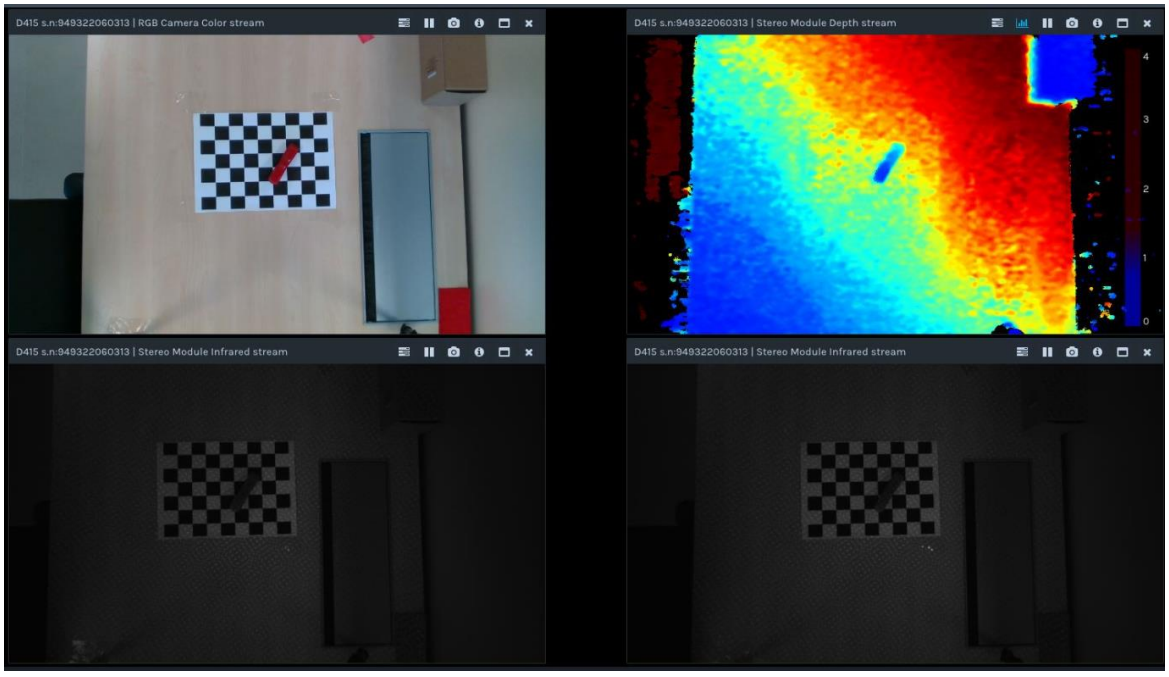


Figura 13: Visualizador en 2D de la adquisición de los distintos sensores

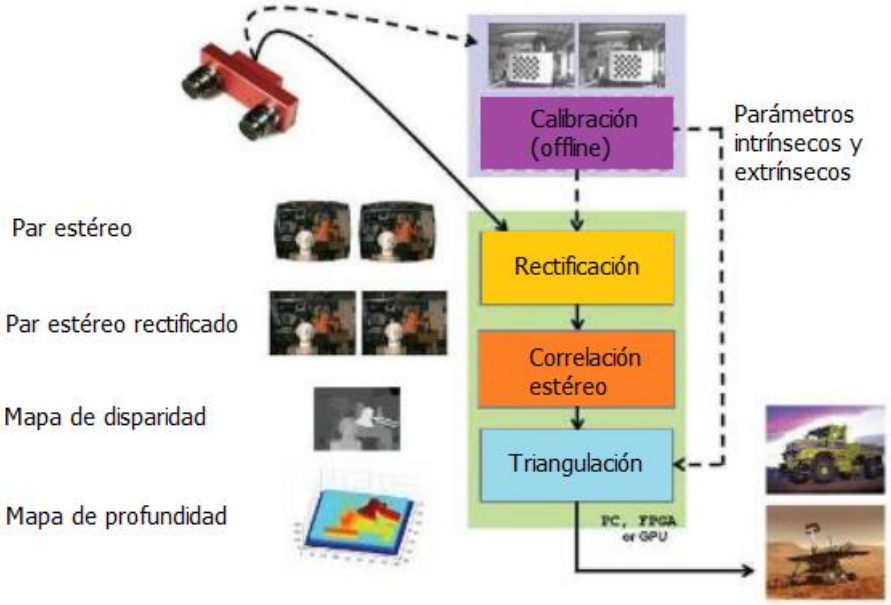


Figura 14: Etapas sistema de visión estéreo

La cámara realiza varias etapas para conseguir el mapa de profundidad, como se muestra en la Figura 14. El emisor infrarrojo proyecta un patrón conocido sobre la superficie de trabajo (ver Figura 15) y los dos sensores estéreo distanciados una longitud conocida llamado *baseline* capturan las imágenes y miden la deformación de ese patrón. Con la técnica estéreo lo que hace es emular la visión binocular humana, se producen dos imágenes, desde dos puntos de vista distintos, que posteriormente se hace una correlación de estéreo para encontrar un mismo punto en ambas imágenes y se triangula para obtener la profundidad a la que se encuentra cada plano de la escena.

Figura 15: Emisión en el espectro infrarrojo, perceptible para otras cámaras, pero no para el ojo humano

2.2 Estado del arte

La robótica industrial es definida por la norma ISO 8373 como "manipulador multifuncional, controlado automáticamente, reprogramable en tres o más ejes, que puede estar fijo o móvil para uso en aplicaciones de automatización industrial" [13]. Hoy en día, existe una estrecha relación entre la industria con sus procesos de manufactura y la robótica. Además, la integración de visión por computador en estos procesos permite controlar y aportar información del entorno [7]. Esta simbiosis entre robot y visión facilita tareas industriales como las que competen a este trabajo, tales como *pick and place* y *picking* (*bin-picking,* si las piezas vienen en un contenedor).

2.2.1 Sensores

Existen una gran variedad de sensores en el mercado para la obtención de información de una escena. Estos sensores se pueden clasificar en:

- Sensores de visión
- Perfilómetros
- Lectores de códigos de barras

Las cámaras de visión industrial, que son los tipos de sensores con los que se va a realizar este proyecto, se pueden clasificar en:

- Cámaras matriciales: el sensor toma una imagen de un área, formando una matriz de píxeles.
- Cámaras lineales: obtienen una imagen de una sola línea de pixeles, es necesario un barrido lineal para la obtención de una imagen como las que proporciona una matricial.
- Cámaras 3D: obtiene una imagen de la escena en 3D, para ello existen diferentes tecnologías.
- Cámaras multiespectrales: las imágenes que se generan son con multitud de longitudes de onda.
- Cámaras de alta velocidad: para aplicaciones en las que la velocidad de análisis es determinante.
- Cámaras inteligentes: son cámaras integradas, las cuales incluyen el procesamiento, la memoria y sistema de comunicación, además de los componentes convencionales.
- Cámaras infrarrojas: trabajan en el espectro infrarrojo.

La elección de uno de ellos se realizará según el tipo de aplicación que vayamos a desarrollar y para qué tipo de industria. Por ejemplo, es muy común el uso de sistemas multiespectrales para la inspección de productos o materias prima en la industria alimentaria o sistemas 3D para la detección de desperfectos en piezas mecánicas en el sector automovilístico.

2.2.2 Tecnología para la obtención de nubes de puntos

Existen distintas técnicas para la obtención de imágenes en 3D de una escena [14]:

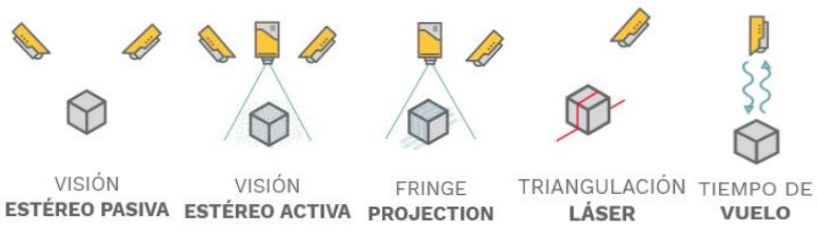


Figura 16: Técnicas 3D

I. Estéreo pasivo: se emplea para aplicaciones donde la escena es estática. Se colocan dos cámaras a una distancia conocida entre ellas y se consigue obtener un mapa 3D parcial o total del objeto con una iluminación controlada.

II. Estéreo activo: la escena puede estar en movimiento. Se obtienen nubes de puntos densas gracias a que además de las dos cámaras se emplea un proyector de luz con un patrón específico.

III. Proyección *fringe*: se pueden emplear patrones diferentes, requiriendo un proyector y una sola cámara. El 3D es calculado a través de la deformación de dicho patrón.

IV. Triangulación láser: se emplea un láser de línea como proyector y una cámara monocroma. Se obtiene una imagen 3D a través de perfiles del contorno de la pieza, los cuáles se generan gracias al desplazamiento de la cámara sobre la pieza o viceversa. A través de la triangulación trigonométrica se obtiene la distancia entre láser-sensor y sensor-pieza.

V. Tiempo de vuelo (ToF): un transmisor emite una luz modulada, se reflecta y llega al sensor que determina el tiempo, así se puede obtener la distancia, conocida la velocidad de la luz, entre objeto y pixel.

2.2.3 Software y aplicaciones industriales

En el mercado de la visión y robótica existen sistemas o aplicaciones preparados para funcionar según la aplicación que se requiera, como el ya mencionado TM5M-700. Otro ejemplo, el fabricante Fanuc ofrece su sistema iRVision, que es una herramienta *plug and play* de detección visual, Figura 17. Localiza y reconoce piezas independientemente de su tamaño, forma o posición.

Figura 17: Sistema iRVision de Fanuc

Otro desarrollador de soluciones de visión aplicadas a la industria 4.0 es Infaimon, el cual ha desarrollado su aplicación para sistemas VGR llamado InPicker. Es una herramienta que han estandarizado para cualquier modelo y marca de cámara. Trabaja con cualquier técnica de obtención de imágenes 3D, y el proceso lo dividen en seis pasos: captura de imagen, creación de mapa 3D de la escena, detección y localización de piezas, posición de agarre del robot, prevención de colisiones y envió de coordenadas [15].

Todo esto en cuanto a aplicaciones comerciales listas para ser integradas en un proceso de producción, control de calidad, etc. Pero si nuestra finalidad es programar un sistema concreto, podemos obtener una licencia de software estándar integral para visión con un entorno de desarrollo integrado, como HALCON de MVTec [16]. O su "hermano pequeño" MERLIC, software todo en uno para programar rápidamente aplicaciones de visión sin necesidad de programación [17].

Capítulo 3. Desarrollo del proyecto

3.1 Aspectos generales

Tras esta introducción a las distintas tecnologías y técnicas empleadas en la industria recogidas en el estado del arte, podemos exponer a continuación las decisiones tomadas en este proyecto para la obtención de la solución. Se empleará un sistema fijo de trípode más cámara, para el cual el objetivo de la cámara se configurará la posición paralela a la superficie de trabajo.

3.2 Herramientas de hardware y software del entorno de trabajo

El hardware utilizado para la realización de este proyecto son las siguientes:

- Cámara o sensor Intel® RealSense™ D415: cámara con tecnología de profundidad estéreo IR activo. Es un dispositivo con un sensor RGB estándar y dos sensores estéreo con un proyector infrarrojo. Esta cámara emplea una técnica de visión estéreo activa con un proyector de patrón específico de luz infrarroja.

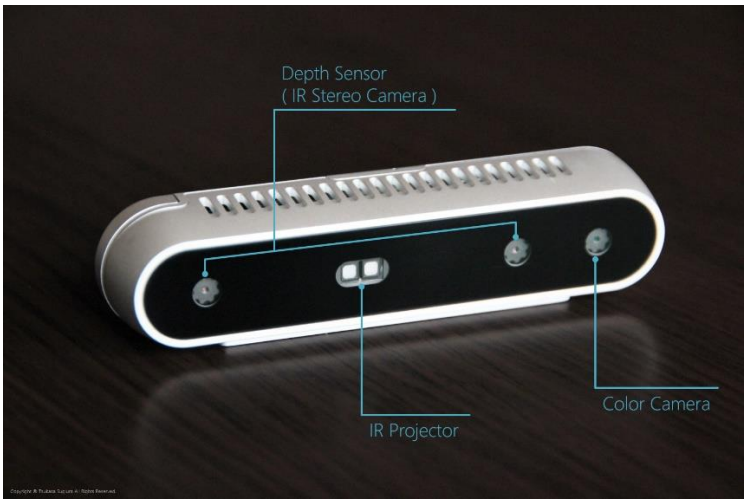


Figura 18: Intel RealSense D415

- Trípode para anclar la cámara para la estabilidad de la misma sobre una superficie.

Figura 19: Montaje sistema de visión

- Ordenador portátil MSI GP63: para la programación del software y simulación.

El software empleado se detalla a continuación:

- Sistema operativo Windows 10.
- Visual Studio Code: editor de código fuente desarrollado por Microsoft, a través del cual se ha configurado para trabajar con el lenguaje Python de programación.
- Intel Realsense Viewer: aplicación de escritorio para acceder rápidamente a la cámara de profundidad para ver el flujo de profundidad, visualizar nubes de puntos, grabar y reproducir flujos, configurar la cámara, etc.
- Robotstudio: herramienta *offline* de la marca ABB para programar y simular de forma óptima.

3.3 Estudio de la solución para la aplicación de *pick and place*

En esta parte de la memoria se detallará la solución por la que se ha optado para dar solución a la aplicación mencionada:

3.3.1 Modos de obtención de imagen

Se han creado 4 modos diferentes dependiendo de la procedencia de la captura:

- Modo 1: imagen guardada. Obtiene la imagen de la memoria del ordenador, cargándola desde la ruta indicada.
- Modo 2: video guardado. Obtiene la captura de video de la ruta especificada.
- Modo 3: Modo de captura de imagen desde la cámara Intel.
- Modo 4: Modo captura de video desde la cámara Intel.

3.3.2 Programa principal principal.py

El programa principal se puede dividir en varias etapas:

i. Definición de la posición de la muestra: se indica al programa donde se encuentra la muestra de color en píxeles, para la posterior segmentación.

Figura 20: Muestra de color en el espacio de trabajo

ii. Ajuste de la cámara: se emplea la función **CallBackFunc()** para obtener los valores en píxeles del objeto de trabajo o sistema de coordenadas con dos puntos en x y un punto en y, para así formar el plano de trabajo respecto al cual estarán referenciadas las

coordenadas de los objetos que encuentre el sistema. Se define también la longitud en mm de la tabla de trabajo, se tomará el patrón de ajedrez como tabla de trabajo el cual tendrá unas medidas estándar de un folio din A4 (297x210). Por último, calculará el valor de la variable Ratio_mm_pixel, que relaciona la distancia que mide un lado de un pixel en mm.

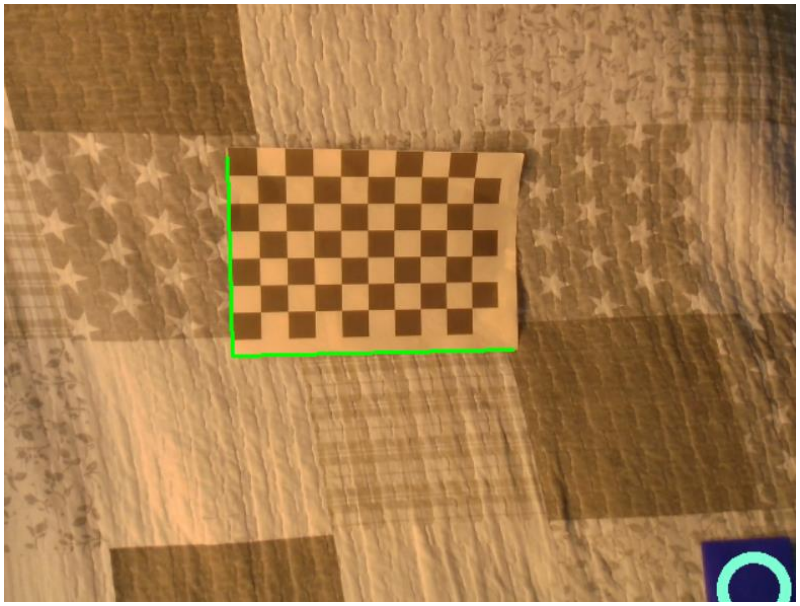


Figura 21: Posición de la muestra con una circunferencia y sistema de coordenadas del plano de trabajo en verde

iii. Funciones para la gestión de la cámara: funciones propias **AbrirCamaraIntel()** y **CapturaCamaraIntel()** para la configuración de los sensores y la adquisición de los *frames* o capturas.

iv. Procesa *frame*: Tras la configuración y ajuste de la cámara se pasa a ejecutar la función **ProcesarFrame().**

3.3.3 ProcesarFrame()

Segmentacion.py, segmenta en el espacio de color HSV la muestra (Figura 20). y se obtienen los contornos de la imagen **cv.findContours()** (Figura 39) y su jerarquía, eliminando los que están fuera de la tabla de trabajo y los contornos que no superan un valor de área. Con la jerarquía se gestiona si unos contornos están dentro de otros. Se utiliza **cv.moments** para obtener los centroides de las piezas.

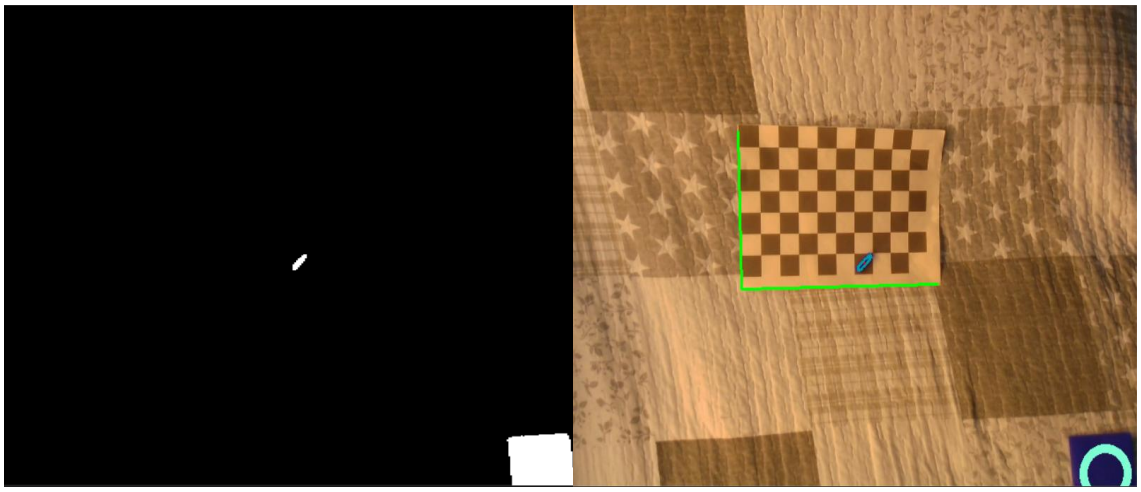


Figura 22: Imagen HSV segmentada y contorno encontrado

Con **CambiodeBase()** se hace una transformación de las coordenadas de los centroides de la imagen a coordenadas mundo, donde se encontrará centrado la base del robot y se multiplica las coordenadas obtenidas por la variable Ratio_mm_pixel para pasar de valor de píxel a mm. A continuación, se presenta el Algoritmo 1 en pseudocódigo para entender los distintos pasos del mismo. El Flujograma 8 también muestra las etapas a través de un diagrama de flujo (Anexo 1).

$$Coordenadas\ robot\ en\ X = coordenada\ x\ \cdot Ratio\_mm\_pixel \quad (12)$$

$$Coordenadas\ robot\ en\ Y = coordenada\ y\ \cdot Ratio\_mm\_pixel \quad (13)$$

3.3.4 CambioDeBase()

A continuación, se presenta el pseudocódigo del algoritmo para hacer un cambio de base:

Procedimiento CambioDeBase(Coordenadas_Imagen, BasePuntoX1, BasePuntoX2, BasePuntoY1)
u← BasePuntoX2 - BasePuntoX1
v← (-u.y, u.x)
Q0.x ← (BasePuntoX2.y*u.x*v.x - (BasePuntoY1.y*u.x + BasePuntoX2.x*u.y)*v.x + BasePuntoY1.x*u.x*v.y) / (u.x*v.y - u.y*v.x)
Q0.y ← (BasePuntoY1.y*u.y*v.x - BasePuntoX2.y*u.x*v.y + BasePuntoX2.x*u.y*v.y – BasePuntoY1.x*u.y*v.y) / (u.y*v.x - u.x*v.y)
Para i← 0 **Hasta** tamanio(Coordenadas_Imagen) **Hacer**:
Si Coordenadas_Imagen.x = 0.0 **Y** Coordenadas_Imagen.y = 0.0 **Entonces**
$\text{Coordenadas\_BaseRobot}_i$.x ← 0.0
$\text{Coordenadas\_BaseRobot}_i$.y ← 0.0
Si no Entonces
Q1.x ← ($\text{Coordenadas\_Imagen}_i$.x*u.x*v.x - BasePuntoX2.y*u.x*v.x + BasePuntoX2.x*u.y*v.x - $\text{Coordenadas\_Imagen}_i$.y*u.x*v.y) / (u.y*v.x – u.x*v.y)
Q1.y ← ($\text{Coordenadas\_Imagen}_i$.x*u.y*v.x - BasePuntoX2.y*u.x*v.y + BasePuntoX2.x*u.y*v.y - $\text{Coordenadas\_Imagen}_i$.y*u.y*v.y) / (u.y*v.x – u.x*v.y)
Q2.x ← ($\text{Coordenadas\_Imagen}_i$.x*u.x*v.x - BasePuntoY1.y*u.x*v.x + asePuntoY1.x*u.x*v.y - $\text{Coordenadas\_Imagen}_i$.y*u.y*v.x) / (-u.y*v.x + u.x*v.y)
Q2.y ← (-$\text{Coordenadas\_Imagen}_i$.x*u.x*v.y + BasePuntoY1.y*u.y*v.x – BasePuntoY1.x*u.y*v.y + $\text{Coordenadas\_Imagen}_i$.y*u.y*v.y) / (u.y*v.x – u.x*v.y)
$\text{Coordenadas\_BaseRobot}_i$.x ← Signo(BasePuntoX2.y – Q0.y)*Signo(Q1.y – Q0.y)*$\sqrt{(Q0.x - Q1.x)^2 + (Q0.y - Q1.y)^2}$
$\text{Coordenadas\_BaseRobot}_i$.y ← Signo(BasePuntoY1.y – Q0.y)*Signo(Q2.y – Q0.y)*$\sqrt{(Q0.x - Q2.x)^2 + (Q0.y - Q2.y)^2}$
Fin Si
Fin Para
Retorna Coordenadas_BaseRobot
Fin Procedimiento

Algoritmo 1: Cambio de base

```
Procedimiento Signo(valor)
    Si valor>=0 Entonces
        Retorna 1.0
    Si valor<0 Entonces
        Retorna -1.0
    Fin Si
Fin Procedimiento
```

Algoritmo 2: Signo

Por aclarar el seudocódigo se va a desarrollar el proceso de la función CambioDeBase() por etapas:

I. Se crean las variables u, v, Q0, Q1 y Q2.
II. Calcula los vectores directores.
III. A continuación, calcula la intersección de los ejes de coordenadas.
IV. Calcula las coordenadas x e y en el objeto de trabajo (*workobject*) del robot:
    i. Calcula la proyección del punto de la imagen sobre el eje x del nuevo sistema de coordenadas.
    ii. Calcula la proyección del punto de la imagen sobre el eje y del nuevo sistema de coordenadas.
    iii. Calcula la diferencia con el punto Q0 para obtener las coordenadas en la nueva base.

3.3.5 CargarModuloABB()

Una vez se obtienen las coordenadas referenciadas con respecto a la base del robot, se envían estas coordenadas a una función la cual se encargar de generar el código en lenguaje *RAPID* para la ejecución del movimiento del robot en el software de ABB, RobotStudio. La función genera un archivo llamado **Contornos.mod,** que es la extensión que define RobotStudio para cargar módulos. Esta parte será tratada con más profundidad en los apartados 5.4.5 y 5.5.2.

3.4 Estudio de la solución para la aplicación de *picking*

Para la aplicación de *picking* se requiere de información en tres dimensiones para la localización del objeto. Mientras que en el programa de visión 2D solo requería la posición del centroide de las piezas para que el robot, con una garra como herramienta, tuviera las coordenadas donde dirigirse; el programa que gestiona la visión 3D requiere obtener la altura a la que se encuentran las piezas, así como si estas piezas no están colocadas de forma favorable, es decir, están rotadas un ángulo con respecto a la horizontal. En esta parte la coordenada Z no es constante como pasaba en el caso anterior, por lo que el programa determinará esta coordenada, para que la garra tome la pieza por el centroide según la posición de la misma.

Figura 23: Pieza posición favorable vs. pieza posición desfavorable

3.4.1 Calibración

El primer paso que realiza el programa, tras definir todas las variables iniciales, es realizar un proceso de calibración, obteniendo los parámetros intrínsecos y la posición del tablero de ajedrez, ya que se empleará este elemento como patrón de calibración junto a la librería opencv, pyrealsense2 y funciones propias para determinar los parámetros mencionados. El algoritmo empleado para calcular la transformación rígida o Euclidiana, explicado en el apartado 4.1.3, es el algoritmo de Kabsch. Este método calcula la matriz de rotación minimizando la RMSD entre dos conjuntos emparejados de puntos (ver Figura 21 y 22) [18].

Cada conjunto de puntos P y Q, pueden representarse como una matriz Nx3:

$$\begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_N & y_N & z_N \end{pmatrix} \quad (14)$$

Siendo cada fila las coordenadas de cada punto.

El algoritmo consta de tres pasos:

I. Translación (Figura 23 b)
II. Cálculo de la matriz de covarianza
III. Cálculo de la matriz de rotación optima (Figura 23 c)

La traslación se consigue haciendo coincidir el centro de gravedad con el origen del sistema de coordenadas

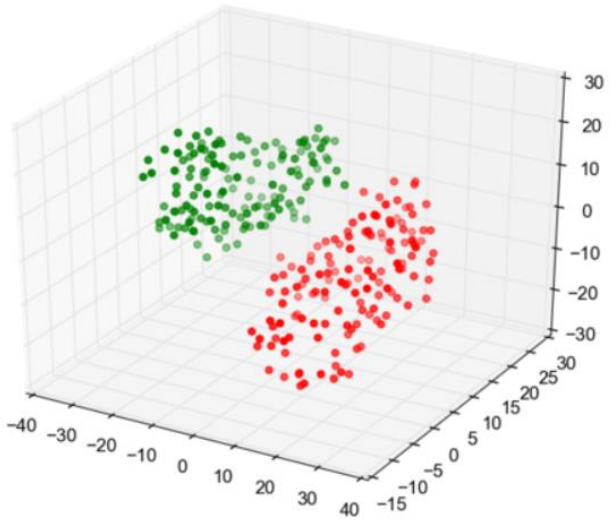


Figura 24: Conjunto de puntos

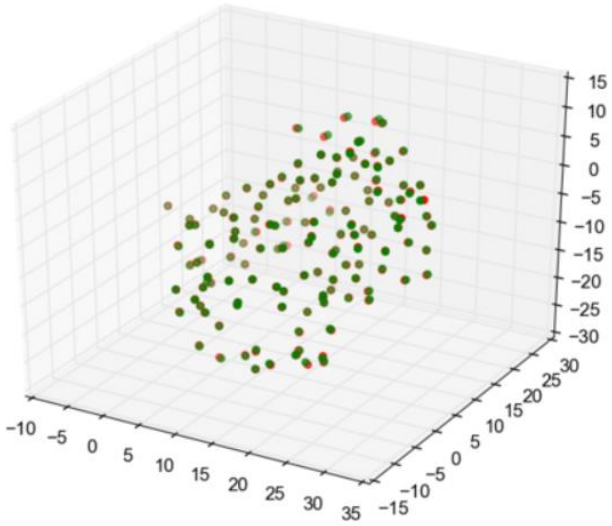


Figura 25: Conjunto de puntos emparejados tras aplicar traslación

Para el cálculo de la matriz de covarianza H, se usa:

$$H = P^T \cdot Q \qquad (15)$$

$$H_{ij} = \sum_{k=1}^{N} P_{ki} \cdot Q_{kj} \qquad (16)$$

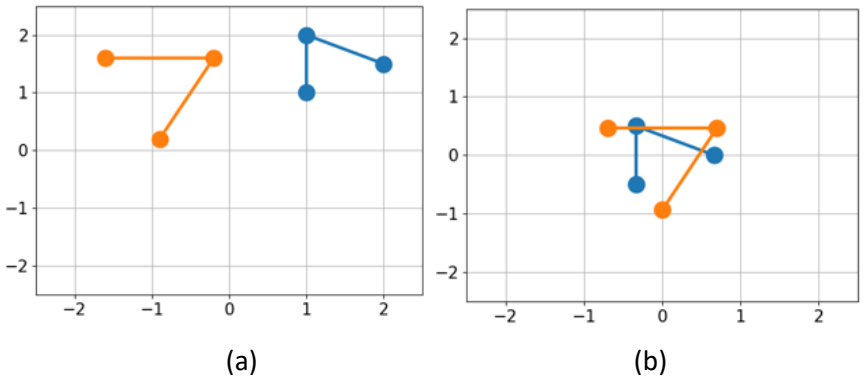
La matriz de rotación óptima se calcula:

$$R = (H^T \cdot H)^{\frac{1}{2}} \cdot H^{-1} \quad (17)$$

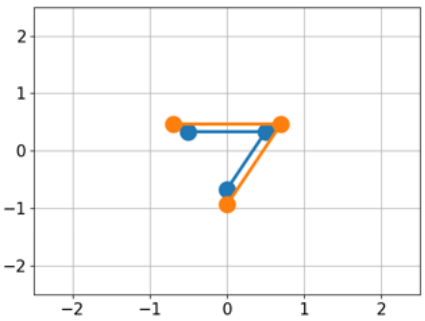
$$H = U \cdot S \cdot V^T \quad (18)$$

$$d = \det(V \cdot U^T) \quad (19)$$

$$R = V \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & d \end{pmatrix} \cdot U^T \quad (20)$$



(a) (b)



(c)

Figura 26: Resultado de aplicar algoritmo de Kabsch. (a) representa dos vectores en sus posiciones originales, en (b) los vectores se trasladan y se centran uno respecto al otro y en (c) los vectores son rotados.

Se ha tomado un repositorio de *Github* para las funciones creadas por el autor Charnley, que calcula la raíz de la desviación cuadrático media (RMSD) entre dos moléculas usando la rotación [19], que, aunque no es nuestro caso, el algoritmo es idéntico para lo requerido en éste TFG.

La desviación cuadrática media es una medida de la diferencia entre los valores de muestra (población) y los valores observados, pronosticado por un estimador. El RMSD es una medida de precisión, para comparar errores de pronóstico de diferentes modelos para un conjunto de datos en particular y no entre conjuntos de datos, ya que depende de la escala [20]. Ésta es la definición estadística, pero quizás la que mejor se adapta al marco de éste TFG es que el RMSD mide la distancia promedio entre dos conjuntos de vectores.

$$RMSD(v,w) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left((v_{ix}-w_{ix})^2 + \left(v_{iy}-w_{iy}\right)^2 + (v_{iz}-w_{iz})^2\right)} \quad (21)$$

Dado dos conjuntos de n puntos v y w. Su unidad es el m.

Tras obtener la posición del tablero de ajedrez en la escena, obteniendo las coordenadas de las esquinas del tablero en 3D y posteriormente usando el algoritmo descrito para encontrar la transformación rígida entre los dos espacios de coordenadas, se aplica dicha transformación a los puntos del tablero en 3D para pasar del espacio de coordenadas mundo al espacio de coordenadas de la cámara (ilustración 8).

3.4.2 Generación de la nube de puntos

Tras la calibración, se obtiene la matriz de posición de la cámara (pose_mat), los parámetros intrínsecos de un sensor infrarrojo, del sensor de color y del sensor de profundidad y los parámetros extrínsecos de la cámara (rotación y traslación).

Se posiciona en la escena el objeto del que se quiere obtener las coordenadas de su centroide. Aquí es donde entra en juego la nube de puntos del objeto, la cual se calcula a través de la función **calculate_cumulative_pointcloud().** Necesita de una región (ROI) para calcular la nube de puntos del objeto dentro del espacio del patrón de ajedrez, así se simplifica la cantidad de datos a procesar, ya que uno de los grandes problemas en la visión 3D es la potencia de computación requerida para el procesamiento de estos tipos de datos. La nube de puntos se genera con una imagen de profundidad. Las fórmulas que se emplean son las siguientes:

$$X = \frac{pixel_x - ppx}{f_x} \cdot profundidad \quad (22)$$

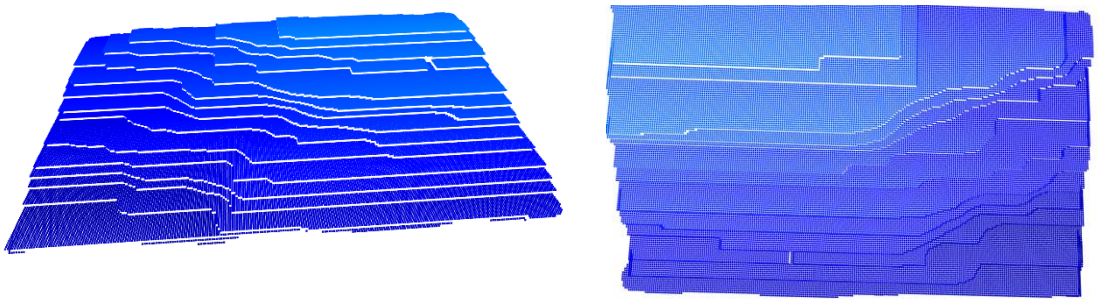
$$Y = \frac{pixel_y - ppy}{y} \cdot profundidad \quad (23)$$

$$Z = profundidad \quad (24)$$

Siendo ppx y ppy los puntos proyectados en cada eje, fx y fy las distancias focales y el valor de profundidad en m.

Posteriormente, se pasa a obtener las medidas del objeto: longitud, anchura y altura, para posteriormente poder obtener el *workobject* del objeto.

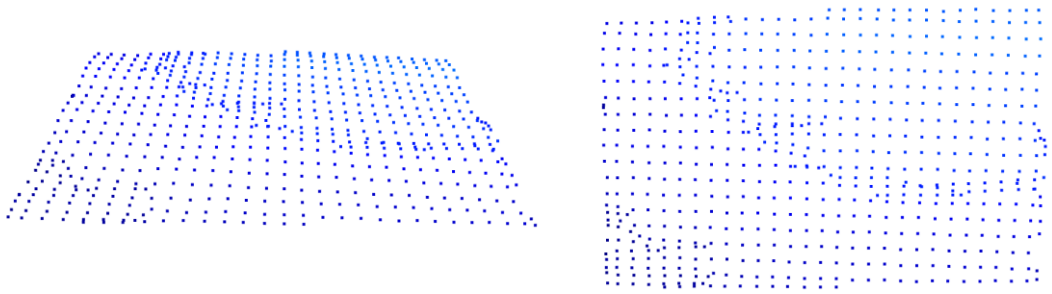
El resultado de la nube de puntos que se obtiene de la superficie de la pieza es:



(a) (b)

Figura 27: Nube de puntos del objeto. (a) Vista con perspectiva. (b) Vista cenital

Pero para disminuir el tiempo de computación, se realiza una reducción de puntos, es decir, se reduce el número de muestras.



(a) (b)

Figura 28: Nube de puntos reducida. (a) Vista con perspectiva. (b) Vista cenital

3.4.3 Obtención de coordenadas del centroide del objeto

Para obtener el centroide, se procede con la imagen capturado con el sensor RGB de la cámara. Y se sigue el mismo flujo que en el programa de visión 2D. la función **centroide()** necesita el *frame* a color, la variable Ratio_mm_pixel que relaciona la medida de un pixel, el *workobject* del plano de trabajo(coordenadas del origen del patrón de ajedrez) y la longitud del plano de trabajo en x e y. Ésta devuelve las coordenadas del centroide y el ángulo de rotación de la pieza si ésta lo estuviese. El programa da elegir al usuario si quiere obtener las coordenadas de ese *frame* o la opción de refrescar para obtener otro.

3.4.4 Consecución del *Workobject* de la pieza y cálculos para obtener la rotación de la pieza

Con la función **obtiene_Workobjet_objeto()** se consigue centrar el objeto de trabajo en el centroide del objeto, por lo que el robot tendrá que ir siempre a la coordenadas (0, 0, 0). Para crear un *workobject* del objeto es necesario dos puntos en x y un punto en y. La creación del mismo facilita la orientación de la herramienta, ya que ésta se posicionará con respecto al objeto

y lo agarrará por la parte más estrecha. La detección del agarre es un uno de los objetivos estudiados en la robótica guiada por visión [21].

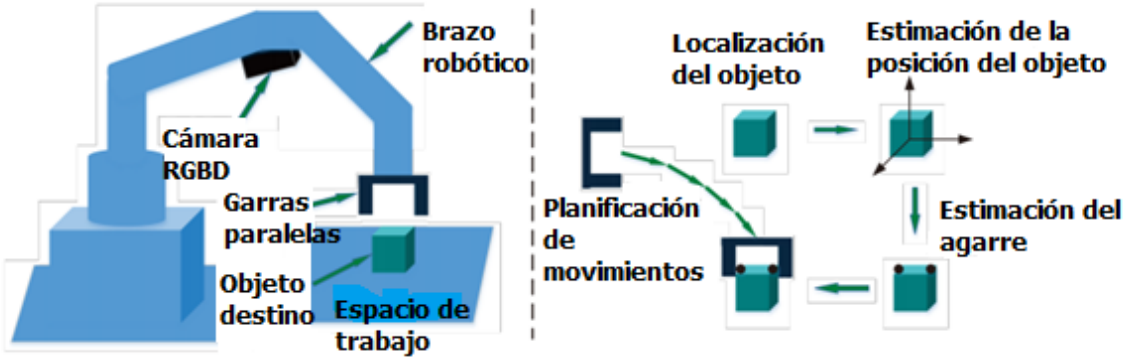


Figura 29: Etapas de detección de agarre robótico

Si la pieza se coloca en la escena, paralela al espacio de trabajo el ángulo de rotación alfa vale 0 grados. Sin embargo, la pieza puede ser colocada en cualquier posición y ángulo, por lo que es necesario obtener el workobject rotado, ver Figura 7.

Si la pieza esta girada antihoraria:

$$X1 = (fila, columna) \quad (25)$$

$$X2 = (fila - |d_x \cdot \sin\alpha|, columna + |d_x \cdot \cos\alpha|) \quad (26)$$

$$Y1 = (fila - |d_y \cdot \sin(90+\alpha)|, columna + |d_y \cdot \cos(90+\alpha)|) \quad (27)$$

Si la pieza esta girada horaria:

$$X1 = (fila, columna) \quad (28)$$

$$X2 = (fila + |d_x \cdot \sin\alpha|, columna + |d_x \cdot \cos\alpha|) \quad (29)$$

$$Y1 = (fila - |d_y \cdot \cos\alpha|, columna + |d_y \cdot \sin\alpha|) \quad (30)$$

3.4.5 Código RAPID

El programa finalmente llama a la función **CodigoRAPIDCentroides()** para escribir en un bloc de notas el código del movimiento del robot en formato .mod.

3.5 Integración del sistema

3.5.1 RobotStudio

Por el potencial y la cantidad de información en foros que existe sobre la herramienta de software para sistemas robóticos de ABB, se ha escogido para el desarrollo de la parte robótica, RobotStudio [22]. Para éste TFG se ha creado una estación en este software, que

comparte el mismo robot, controlador y modelado tanto para el programa de visión 2D como para el de 3D. El robot escogido es el IRB120 con su controlador IRC5 con FlexPendant. El controlador IRC5 contiene los elementos electrónicos necesarios para controlar el manipulador, los ejes adicionales y equipos periféricos [23]. Se ha escogido este robot, el más pequeño de ABB, ya que es de pequeñas dimensiones, apto para el rango de trabajo que se requiere y con capacidad de soportar cargas de hasta 4kg [24].

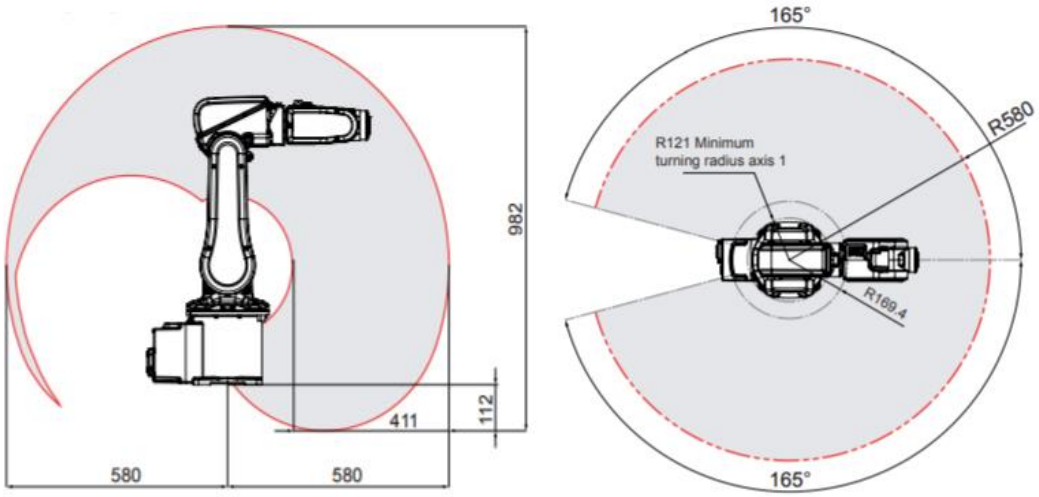


Figura 30: Rango de trabajo IRB120

3.5.1.1 Presentación de la estación

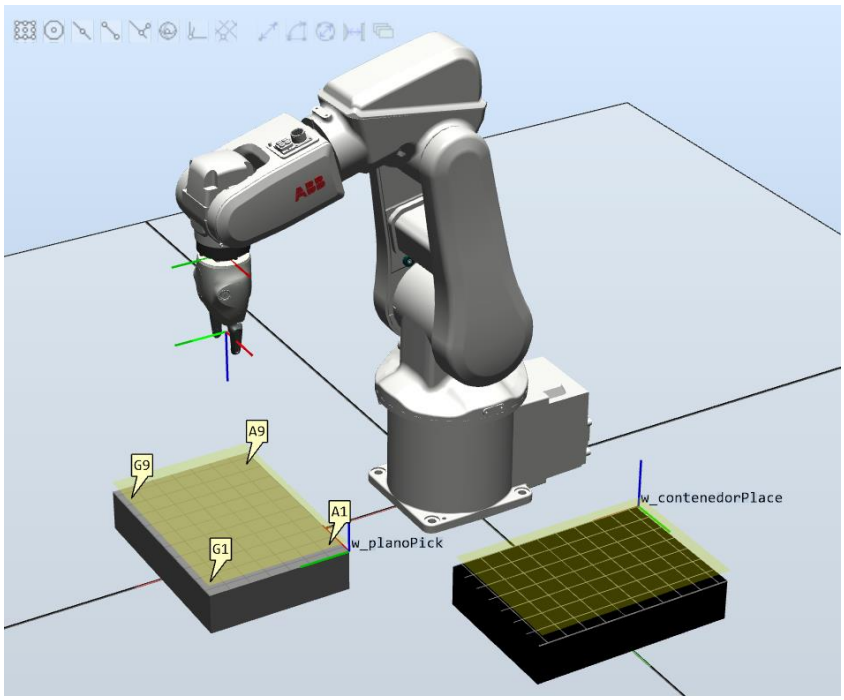


Figura 31: Estación

La estación está compuesta por el robot, y dos contenedores. El contenedor de donde el robot cogería las piezas y el contenedor donde el robot dejaría las piezas. Tienen una cuadrícula, ya que para la visión 2D se ha empleado como guía el tablero de ajedrez y se han

numerado las casillas de tal forma que las filas van desde la A hasta la G y las columnas van desde la 1 hasta la 9. Así es más sencillo a la hora de simular y hacer pruebas obtener las coordenadas del centroide de las piezas y el correspondiente posicionamiento de esas piezas en simulación. Como piezas se han empleado cubos de 30mm de cara y para el programa de visión 3D se ha empleado un paralelepípedo rectangular de dimensiones 140x45x50 mm.

3.5.2 Comunicación

Al no haberse podido implementar el sistema de forma real, no se ha desarrollado, aunque sí estudiado, una parte de comunicación como tal. Pero, se ha destinado un pequeño módulo de comunicación en caso de querer transferir y ejecutar el programa en el controlador físico del robot. **CargarModuloABB()** es una función que genera un documento XML y un script de comandos de *Windows* llamado CargarModuloCmd, para ejecutar la comunicación con el robot, habiendo creado una hoja de cálculo de *Microsoft Excel* con la IP del robot, previamente. Esta función genera unos comandos con unos parámetros y unas acciones para ser ejecutado a través de **runjob.exe,** un archivo interno del sistema.

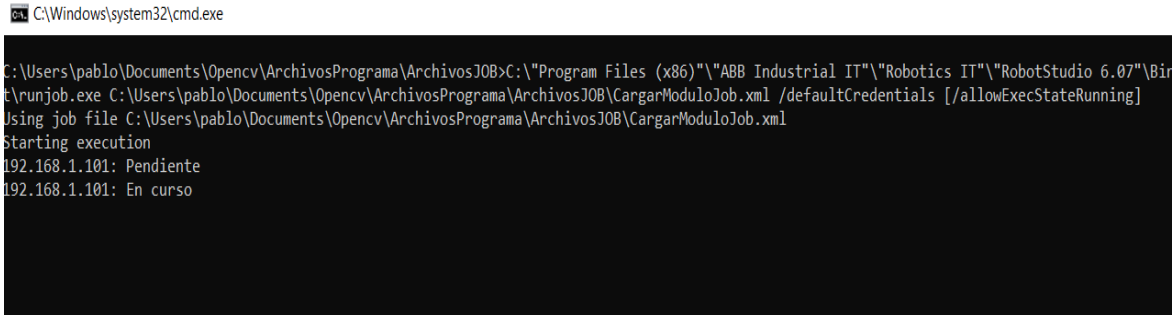


Figura 32: Intento de conexión con el controlador del robot físico

	A	B	C	D	E
1	**Network Address**	**Controller Name**	**System Name**	**Group**	**SubGroup**
2	192.168.1.101	ROBOT	120-101534		

Figura 33: Excel con la IP del robot

3.5.3 Simulación en RobotStudio

Con lo mencionado anteriormente, se ha optado por la simulación de todo el proceso, tanto la colocación de la pieza en las coordenadas del contenedor de recogida, la simulación de las físicas de las piezas, el movimiento del robot, colocación de sensores en la estación para la realimentación, creación de señales E/S y propiedades y conexionado de la lógica de estación.

3.5.3.1 WorkObject y Tool

Es necesario definir tres parámetros en RobotStudio, la tarea que se está ejecutando, el objeto de trabajo y la herramienta. En este caso la tarea es por defecto llamada **T_ROB1,** pero también existen tareas de más tipos, por ejemplo, la llamada tarea de *background* o de fondo, la cual se ejecuta de manera continua, aunque no se esté ejecutando la tarea principal, esta tarea es útil para crear seguridades, por ejemplo, que el robot pare su movimiento si se accede a la zona del mismo.

El *WorkObject* u objeto de trabajo en el caso del 2D son el **w_planoPick,** de donde se cogen y referencian los objetos y el **w_contenedorPlace,** donde se depositan las piezas. En la simulación 3D también está **w_objeto** que es el objeto de trabajo de la pieza centrado en su centroide.

Y la herramienta utilizada se denomina **t_pinza.**

3.5.3.2 RAPID

En Python se genera directamente el programa en RAPID, por lo que solo habrá que cargar el módulo cada vez que se haga una nueva adquisición con la cámara.

El módulo en RAPID se compone de unas declaraciones de variables de velocidades y Z de trabajo, en el caso del programa 2D, funciones para ir a la posición *HOME,* para generar las piezas y posicionarlas en la simulación y el programa **CoordenadasCamara()** que contiene las instrucciones de movimiento del robot. Hay definidas dos tipos de instrucciones:

- *MOVEJ:* se usa para mover el robot rápidamente de un punto a otro.
- *MOVEL:* se utiliza para trasladar el punto central de la herramienta (TCP) en sentido lineal hacia un punto de destino.

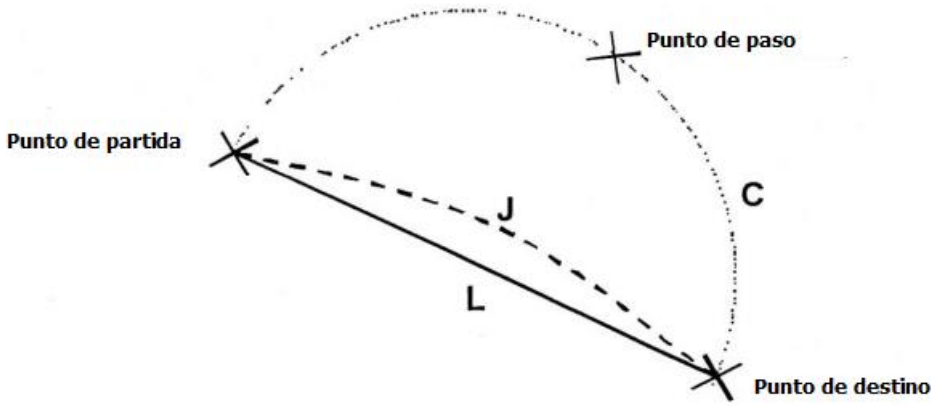


Figura 34: Diferencia entre las tres formas de desplazamiento

3.5.3.3 Componentes inteligentes

Los componentes inteligentes se dividen en categorías según su función y sirven para simular los distintos procesos dentro de RobotStudio. Se han empleado para la realización de esta solución los siguientes:

- *PoseMover:* componente perteneciente a la categoría manipuladores que mueve los ejes de un mecanismo hasta una pose definida. Se han empleado dos, uno para cerrar y otro para abrir la pinza de la herramienta.
- *Source:* Crea una copia de un componente gráfico en las coordenadas especificadas. Esto es para posicionar las distintas piezas en el contenedor de cogida.
- *Sink:* contrario al anterior, elimina la copia creada.
- *PhysicsControl:* para darle física a las piezas, en este caso se han escogida piezas de material plástico que son menos pesadas y se le proporciona un comportamiento dinámico, esto hace por ejemplo que al caer reboten o que se deformen si la pinza cierra con fuerza.
- *PlaneSensor:* se han usado sensores de plano, para cuando se simule el programa ejecute el movimiento del robot si las piezas han sido posicionadas y que quede listo para otro ciclo cuando todas las piezas hayan sido depositadas en el contenedor de dejada.
- *LogicGate (AND):* una puerta con lógica AND para cuando se den dos sucesos se obtenga una salida a '1'. En este caso cuando detecta que todas las piezas se han colocado en el

contenedor de dejada y se ha activado una señal de simulación terminada, la simulación pasa a eliminar las piezas que se han colocado en el contenedor, para un nuevo ciclo.

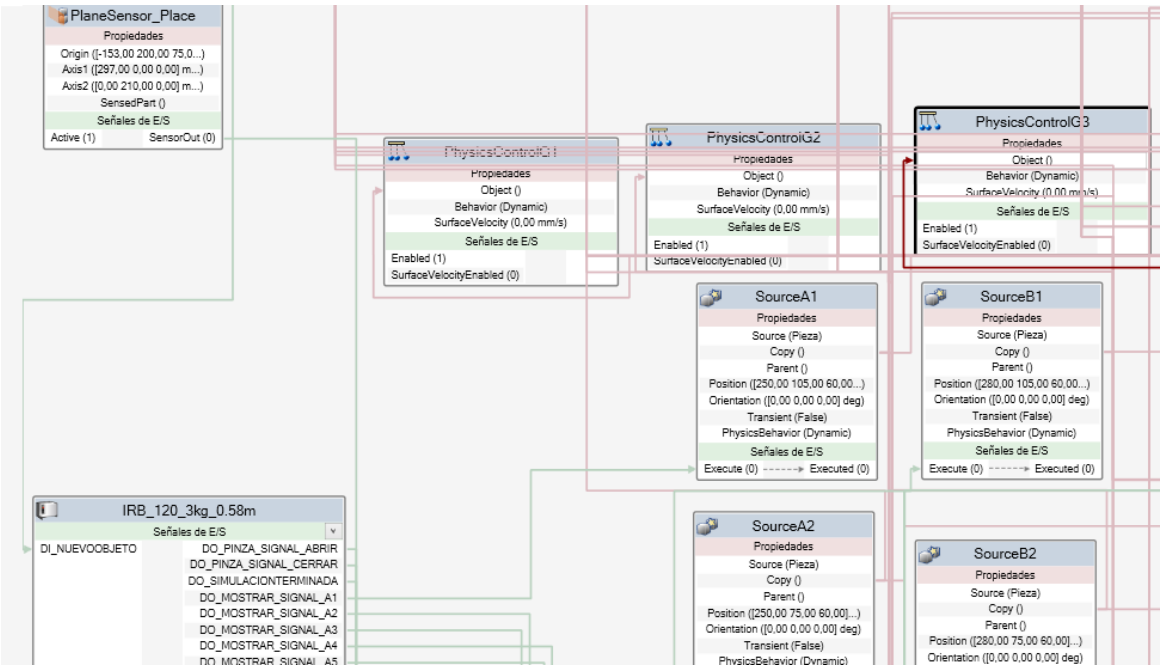


Figura 35: Captura del diseño de lógica de estación

Conexiones de E/S

Objeto de origen	Señal de origen	Objeto de destino	Señal o propiedad de destino
IRB_120_3kg_0.58m	DO_PINZA_SIGNAL_ABRIR	PoseMover_Abrir [Pinza_Abierta]	Execute
IRB_120_3kg_0.58m	DO_PINZA_SIGNAL_CERRAR	PoseMover_Cerrar [Pinza_Cerrada]	Execute
PlaneSensor_Pick	SensorOut	IRB_120_3kg_0.58m	DI_NUEVOOBJETO
PlaneSensor_Place	SensorOut	LogicGate [AND]	InputA
IRB_120_3kg_0.58m	DO_SIMULACIONTERMINADA	LogicGate [AND]	InputB
IRB_120_3kg_0.58m	DO_MOSTRAR_SIGNAL_A1	SourceA1	Execute
IRB_120_3kg_0.58m	DO_MOSTRAR_SIGNAL_A2	SourceA2	Execute
IRB_120_3kg_0.58m	DO_MOSTRAR_SIGNAL_A3	SourceA3	Execute
IRB_120_3kg_0.58m	DO_MOSTRAR_SIGNAL_A4	SourceA4	Execute
IRB_120_3kg_0.58m	DO_MOSTRAR_SIGNAL_A5	SourceA5	Execute
IRB_120_3kg_0.58m	DO_MOSTRAR_SIGNAL_A6	SourceA6	Execute
IRB_120_3kg_0.58m	DO_MOSTRAR_SIGNAL_A7	SourceA7	Execute
IRB_120_3kg_0.58m	DO_MOSTRAR_SIGNAL_A8	SourceA8	Execute
IRB_120_3kg_0.58m	DO_MOSTRAR_SIGNAL_A9	SourceA9	Execute
IRB_120_3kg_0.58m	DO_MOSTRAR_SIGNAL_B1	SourceB1	Execute
IRB_120_3kg_0.58m	DO_MOSTRAR_SIGNAL_B2	SourceB2	Execute
IRB_120_3kg_0.58m	DO_MOSTRAR_SIGNAL_B3	SourceB3	Execute
IRB_120_3kg_0.58m	DO_MOSTRAR_SIGNAL_B4	SourceB4	Execute
IRB_120_3kg_0.58m	DO_MOSTRAR_SIGNAL_B5	SourceB5	Execute
IRB_120_3kg_0.58m	DO_MOSTRAR_SIGNAL_B6	SourceB6	Execute
IRB_120_3kg_0.58m	DO_MOSTRAR_SIGNAL_B7	SourceB7	Execute
IRB_120_3kg_0.58m	DO_MOSTRAR_SIGNAL_G7	SourceG7	Execute
IRB_120_3kg_0.58m	DO_MOSTRAR_SIGNAL_B8	SourceB8	Execute

Figura 36: Captura de señales y conexiones de lógica de estación

3.5.4 FlexPendant

La FlexPendant virtual o consola virtual es una representación exacta de la consola real junto al controlador. En este proyecto se ha usado para tener referencia de las coordenadas en las que se encuentra el robot X, Y, Z junto a las orientaciones en cuaternios.

También durante la simulación se producen errores de movimiento como: el brazo no puede alcanzar una posición o que para alcanzarla colisiona con algún elemento de la estación o con la propia base del robot. Incluso errores en puntos por estar muy próximos entre sí. Con esta herramienta podemos visualizar de forma completa el origen del error y su posible solución.

Además, se ha empleado la consola para obtener coordenadas por ejemplo de la posición *HOME.*

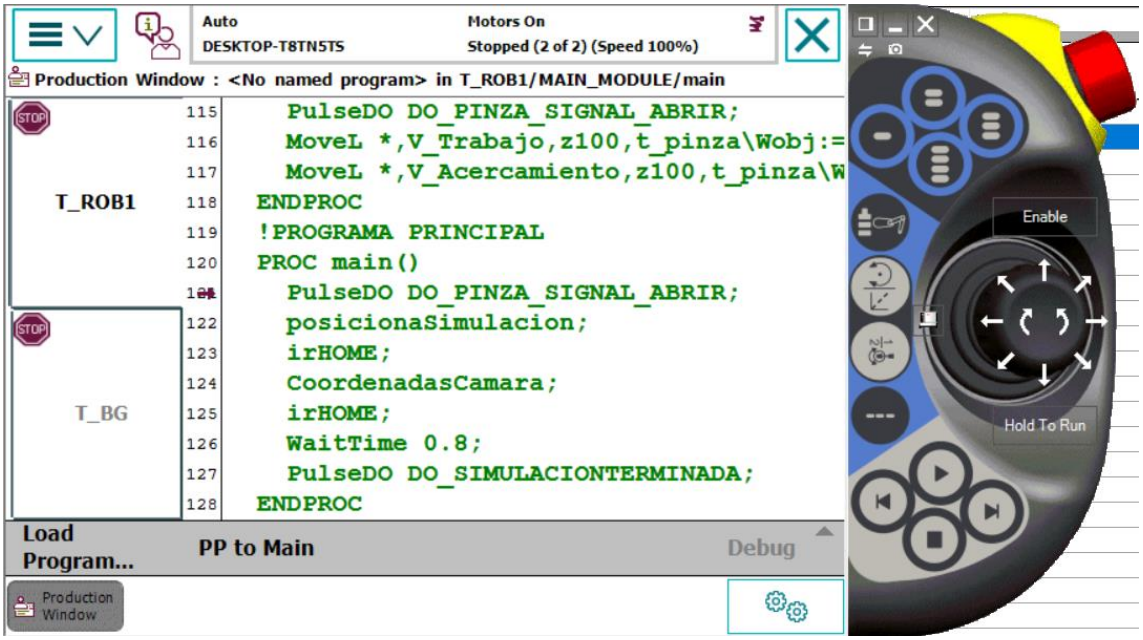


Figura 37: FlexPendant con las instrucciones en RAPID del movimiento del robot

Capítulo 4. Experimentos y resultados obtenidos

Con el fin de probar y demostrar el alcance que se ha obtenido durante el desarrollo de este proyecto, se han realizado numerosos experimentos para valorar la versatilidad de la solución adoptada. Existen ciclos en el que la cogida por parte de la herramienta del objeto resulta ser ineficiente, ya que, durante el movimiento del robot para depositar la pieza correspondiente en el contenedor de dejada, ésta cae durante el proceso.

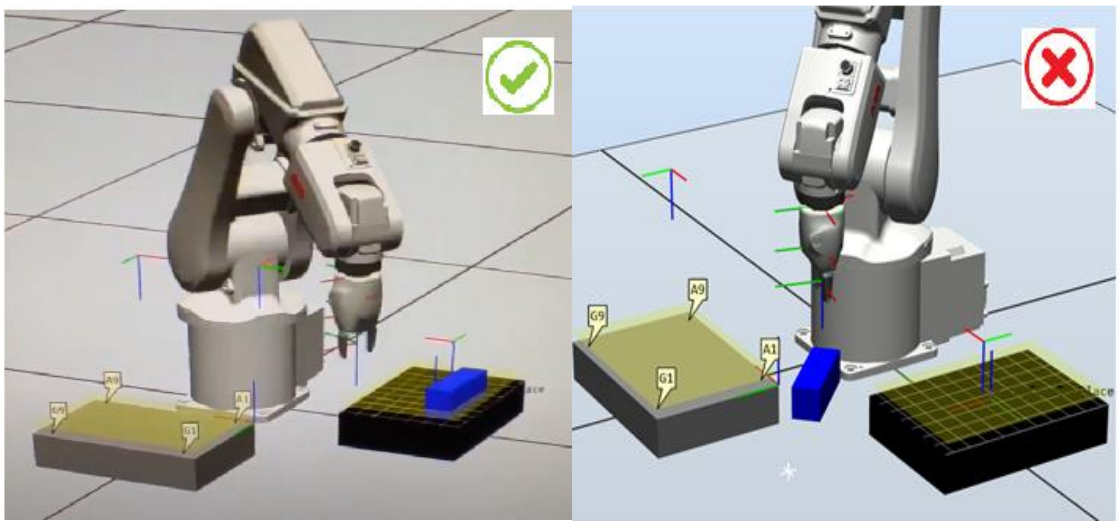


Figura 38: Simulación de prueba con y sin éxito en el posicionamiento de la pieza en el contenedor de dejada

Otro error típico es en la obtención del contorno en el programada 3D, ya que es más dependiente de un sistema de iluminación radial por no aplicar segmentación por color como ocurre con el programa 2D, ya que en el 3D se puede posicionar cualquier pieza que sea un paralelepípedo rectangular independientemente del color de éste. Por lo que se obtiene una pequeña sombra en la imagen que la detecta como contorno. Lo cual hace que se cometa un pequeño error de desviación del centroide de su posición, solucionable con el mencionado sistema de iluminación.

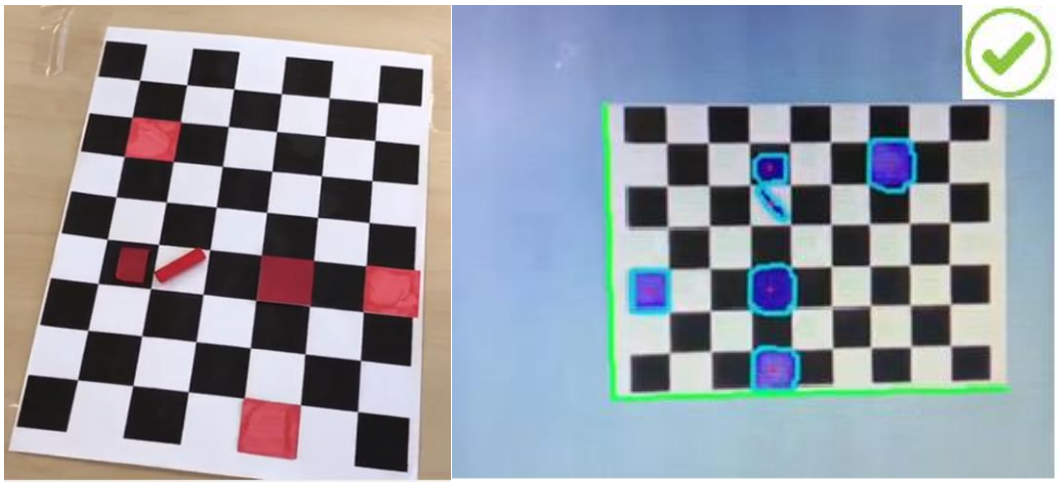


Figura 39: Contornos bien definidos gracias a la segmentación en HSV

Figura 40: Error en la detección del contorno debido a la falta de iluminación

Con la posición de la pieza que más problemas da, es cuando ésta está en vertical, ya que el cierre de la garra cambia con respecto a las demás orientaciones de la pieza y en un 90% de los casos la pieza no llega a su posición de destino.

Figura 41: Orientación de la pieza paralela con la horizontal

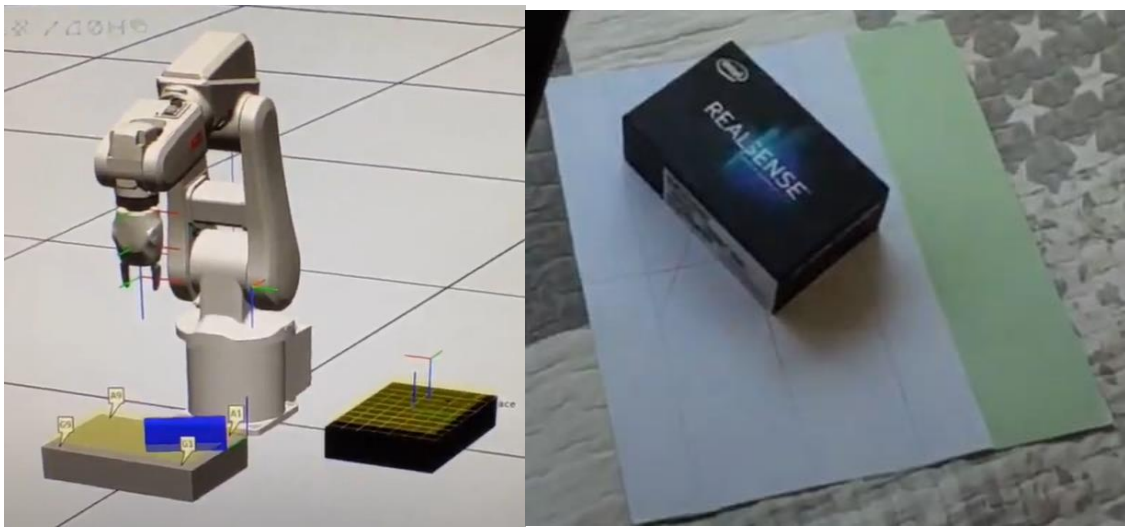


Figura 42: Orientación de la pieza 30º con respecto a la horizontal

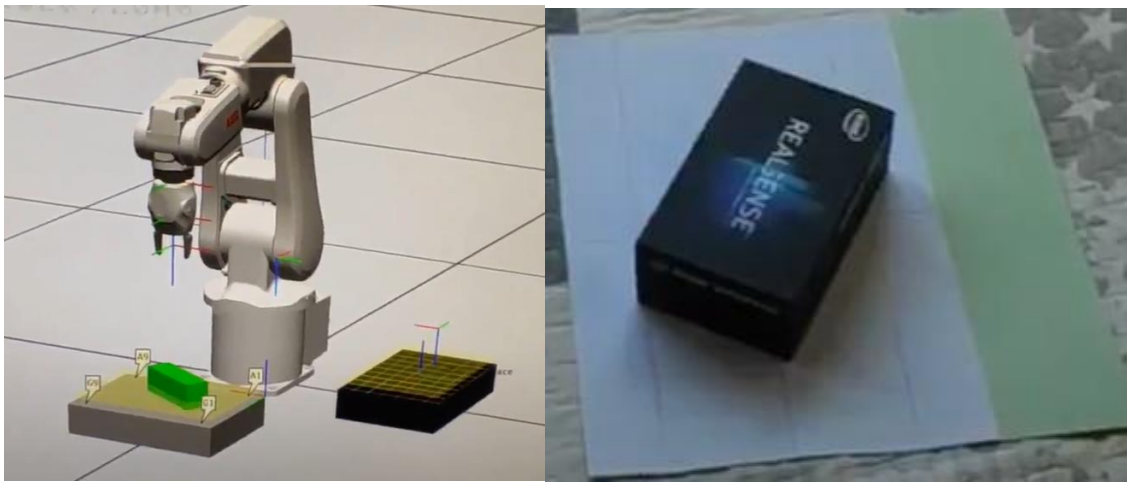


Figura 43: Orientación de la pieza -30º con respecto a la horizontal

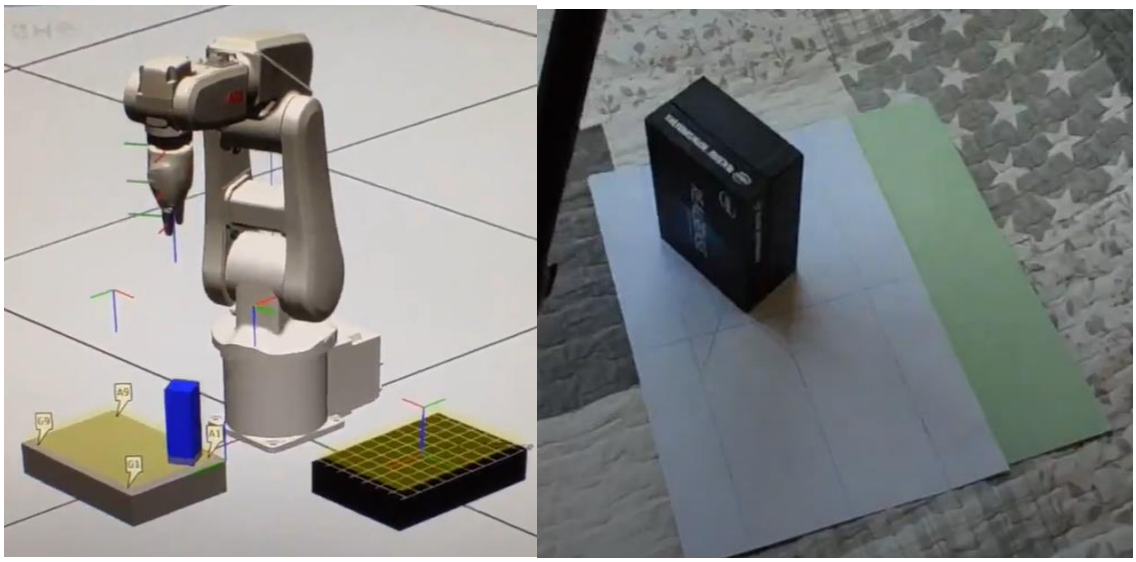


Figura 44: Orientación de la pieza en vertical rotado 30º con respecto a la horizontal

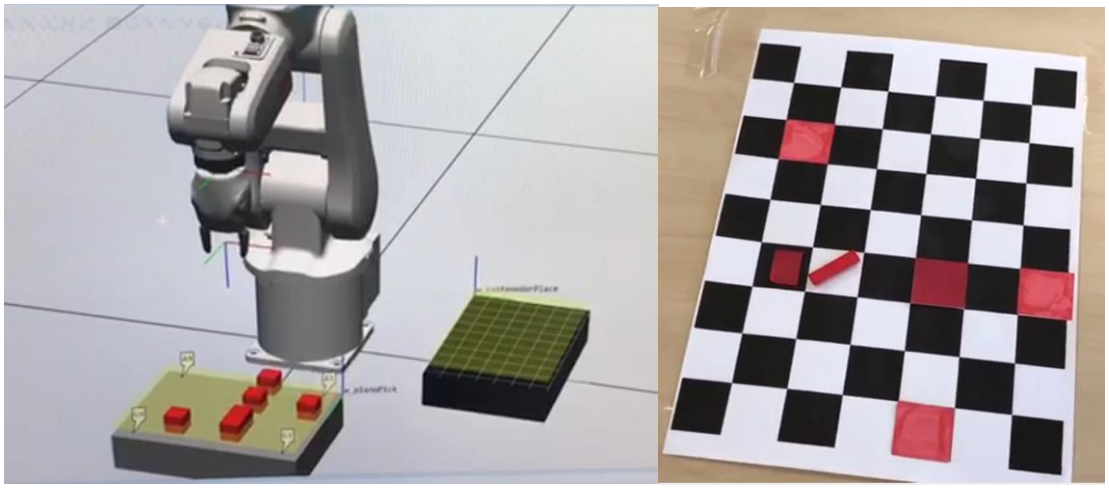


Figura 45: Pick and place, sistema 2D

Capítulo 5. Conclusiones y trabajo futuro

5.1 Conclusiones del proyecto

En este proyecto se ha presentado una solución para un sistema VGR con el fin de desarrollar una aplicación de *pick and place* y otra de *picking* empleando visión en dos y en tres dimensiones, obteniendo las siguientes conclusiones:

La industria necesita de la visión por computador para mejorar y optimizar los procesos de producción, es por ello que las empresas necesitan cada vez más invertir en estas tecnologías.

La visión en 2D no presenta excesiva dificultad, por estar basada en métodos de visión tradicionales, pero cuando se quiere trabajar en 3D con diferentes tipos de datos y métodos basados en *Deep Learning,* aumenta la complejidad. En este proyecto se ha hecho un acercamiento al sistema operativo Linux, ya que la mayoría de los ingenieros que programan sistemas de visión son ingenieros informáticos y usan este sistema para desarrollar sus programas. Pero es complejo la integración de las distintas dependencias en este sistema operativo, por lo que se optó por seguir en *Windows,* con el lenguaje Python en un editor de código.

Cuando se trabaja con estos tipos de sistemas es muy común hacer el montaje de la cámara en la propia garra del robot, es decir, en el *end effector.* Así se combina una serie de barridos de movimiento para proporcionar una vista completa del contenido del contenedor evitando oclusiones de piezas en el caso del *binpicking.* Posteriormente se puede aplicar una técnica de SfM (Structure from Motion), para obtener los distintos puntos de vista y formar el 3D del contenedor, así aplicando *3D Matching,* se consigue teniendo un modelo de referencia, compararlo con los objetos dentro del contenedor y encontrar el mejor candidato para el agarre del robot, generalmente, el que tenga con menos oclusiones, es decir, el objeto más próximo a la superficie apilada de piezas.

La cámara *Intel Realsense D415* [1] es una buena opción debido a su bajo coste para proyectos de este tipo, ya que es *plug and play* y posee sus herramientas de visualización y configuración directamente sin necesidad de programación. Si el sistema se implementase en la industria habría que elegir otra cámara más robusta como, por ejemplo, las Ensenso [2].

Para finalizar esta conclusión comentar que la gran limitación que ha tenido este proyecto ha sido, debido a las circunstancias de pandemia generada por el coronavirus COVID-19, tener una idea inicial de la programación de un sistema más su posterior "puesta en marcha", y no haber podido trabajar con un sistema robótico real lo que hizo tener que recurrir a herramientas de simulación. Se deja contemplado en el trabajo futuro, la comunicación, integración y "puesta en marcha" de este sistema VGR.

5.2 Trabajo futuro

Por último, tras todo lo contemplado en los apartados de este TFG, se presentan una serie de elementos para ayudar a la mejora y el desarrollo de este proyecto:

- Testear y mejorar la simulación para contemplar más posiciones aleatorias de las piezas.
- Dotar al sistema de visión de mayor capacidad a la hora de reconocer diferentes piezas y clasificarlas según el tipo, para así poder obtener una clasificación de piezas en distintos contenedores según la forma, color, tipo, etc.
- Emplear *Deep learning* o aprendizaje profundo para todo el sistema, tanto la localización del objeto, como la estimación de la 'pose' del mismo y estimación del agarre del robot.

Usar también algoritmos más extendidos en la detección del agarre como *RANSAC* y en la estimación de la 'pose' como el algoritmo *ICP* (iteración del punto más cercano).

- Usar otro sistema operativo como *Linux* para realizar un sistema más complejo, ya que existen librerías completas como *PCL* para visión que tienen gran cantidad de algoritmos ya implementados y *Kinetic Kame* de *ROS* para robótica por ser una plataforma junto a la herramienta *rviz* para la simulación y visualización de robots de varios fabricantes.
- Desarrollar la aplicación completa de *binpicking,* en la que exista un contenedor con piezas dispuestas de forma totalmente aleatoria y en el que incluso haya oclusiones entre piezas y no se pueda obtener en cada caso la nube de puntos completa con respecto al modelo aislado.
- Creación de un servidor IoT para adquirir y procesar imágenes tomadas por cámaras remotas, para pasar de sistemas de visión individualizados a sistemas distribuidos y así minimizar los tiempos muertos.
- Probar con un robot y controlador real la bondad de la solución obtenida, así como experimentar con el agarre robótico de la herramienta y estimar los casos en el que la orientación y el agarre son válidos.

Capítulo 6. Bibliografía

[1] Intel, «RealSense,» [En línea]. Available: https://www.intel.es/content/www/es/es/architecture-and-technology/realsense-overview.html.

[2] «Ensenso,» [En línea]. Available: https://www.ensenso.com/.

[3] «Photoneo,» [En línea]. Available: https://www.photoneo.com/.

[4] A. K. &. G. Bradsky, Learning OpenCV 3, O'Reilly Media, Inc., 2016.

[5] Intel, «Intel® RealSense™ SDK,» [En línea]. Available: https://github.com/IntelRealSense/librealsense.

[6] Keyence, «Keyence,» [En línea]. Available: https://www.keyence.com/.

[7] Infaimon, «Infaimon,» [En línea]. Available: https://blog.infaimon.com/la-vision-artificial-industria-4-0/.

[8] Cognex, «Cognex,» [En línea]. Available: https://www.cognex.com/es-es/what-is/machine-vision/what-is-machine-vision.

[9] bcnvision, «bcnvision,» [En línea]. Available: https://www.bcnvision.es/.

[10] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/P%C3%ADxel#:~:text=Un%20p%C3%ADxel%20o%20pixel%2C%E2%80%8B,parte%20de%20una%20imagen%20digital..

[11] Uco, «ArUco: librería para aplicaciones de realidad aumentada basada en OpenCV de la Universidad de Córdoba,» [En línea]. Available: https://www.uco.es/investiga/grupos/ava/node/26.

[12] Wikipedia, «Cuatrnion dual,» [En línea]. Available: https://es.wikipedia.org/wiki/Cuaterni%C3%B3n_dual.

[13] ISO, «Robots y dispositivos robóticos,» [En línea]. Available: https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-2:v1:en.

[14] «Infaimon,» [En línea]. Available: https://www.infaimon.com/.

[15] Infaimon, «InPicker,» [En línea]. Available: https://inpicker.com/es/.

[16] MVTec, «HALCON,» [En línea]. Available: https://www.mvtec.com/products/halcon/.

[17] MVTec, «MERLIC,» [En línea]. Available: https://www.mvtec.com/products/merlic/.

[18] Wikipedia, «Algoritmo de Kabsch,» [En línea]. Available: https://es.qwe.wiki/wiki/Kabsch_algorithm.

[19] Charnley, «rmsd,» [En línea]. Available: https://github.com/charnley/rmsd.

[20] A. B. R. J.Hyndman, «"Another look at measures of forecast accuracy",» 2006. [En línea]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0169207006000239?via%3Dihub..

[21] K. W. ,. S. L. ,. K. Z. Guoguang Du, «Vision-based Robotic Grasp Detection From Object Localization, Object Pose,» *Cloudminds Technologies.*

[22] ABB, «RobotStudio,» [En línea]. Available: https://new.abb.com/products/robotics/es/robotstudio.

[23] ABB, «Manual del controlador IRC5».

[24] ABB, «Datasheet IRB120».

[25] Xataka, «Intel lanza dos cámaras RealSense con sensor de profundidad: visión en 3D para cualquier dispositivo,» [En línea]. Available: https://www.xataka.com/realidad-virtual-aumentada/intel-lanza-dos-camaras-realsense-con-sensor-de-profundidad-vision-en-3d-para-cualquier-dispositivo.

[26] Microsoft, «Kinect,» [En línea]. Available: https://developer.microsoft.com/es-es/windows/kinect/.

[27] Q.-Y. Z. a. J. P. y. V. Koltun, «Open3D,» [En línea]. Available: http://www.open3d.org/.

[28] Wikipedia, «Arbol Kd,» [En línea]. Available: https://es.wikipedia.org/wiki/%C3%81rbol_kd.

[29] F. H. Z. S. S. y. G. C. N. Sayantan Chatterjee, «Real-Time Object Detection and Recognition on Low-Compute,» *Robotics and Artificial Intelligence Laboratory at the Indian Institute of Information Technology, Allahabad, Prayagraj (Uttar Pradesh), India.,* 2020.

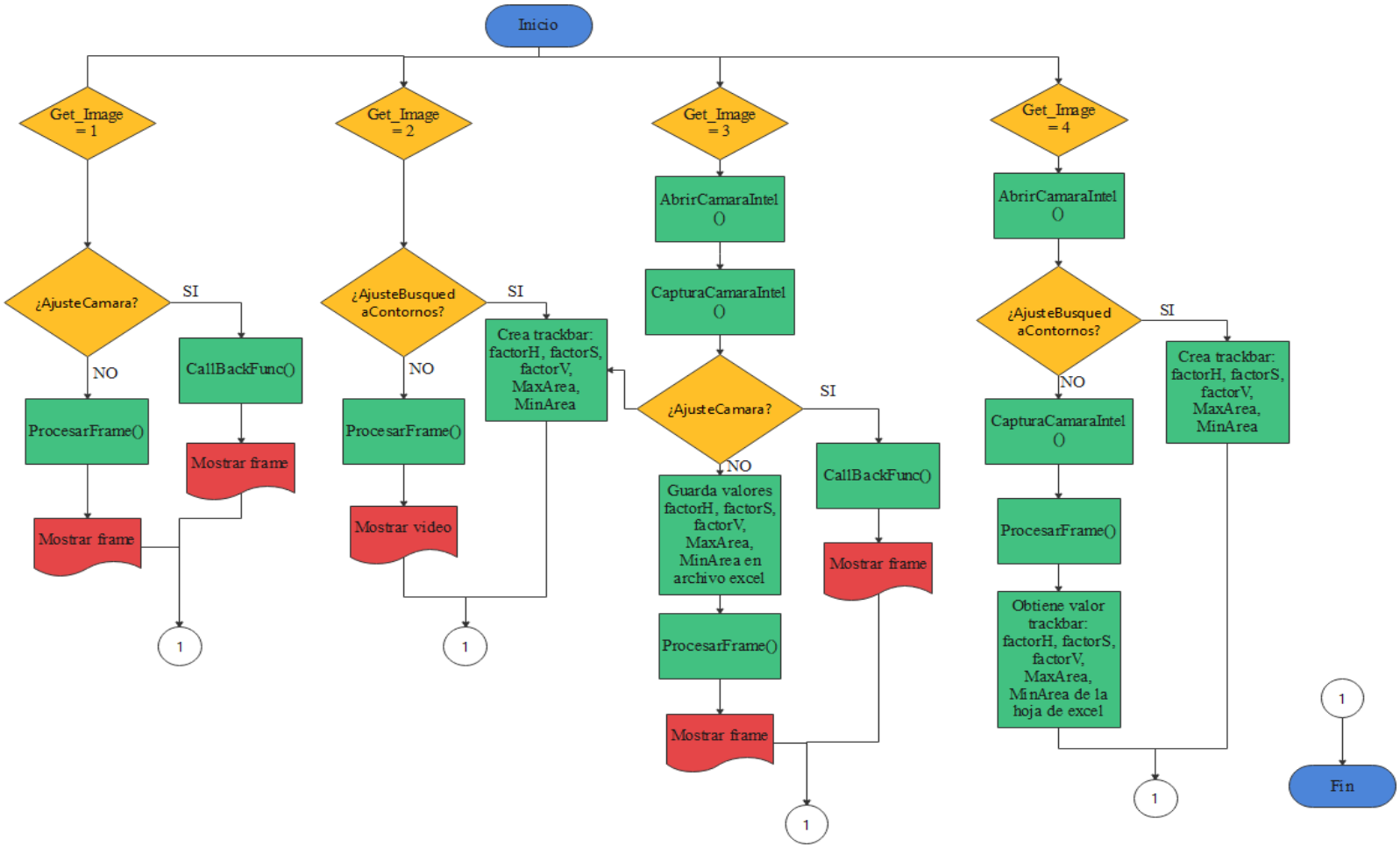
[30] C.-Y. W. y. H.-Y. M. L. Alexey Bochkovskiy, «YOLOv4: Optimal Speed and Accuracy of Object Detection,» 2020.

[31] A. Garcia-Garcia, F. Gomez-Donoso, J. Garcia-Rodriguez, S. Orts-Escolano, M. Cazorla y J. Azorin-Lopez, «PointNet: A 3D Convolutional Neural Network for real-time object class recognition,» *IEEE,* 2016.

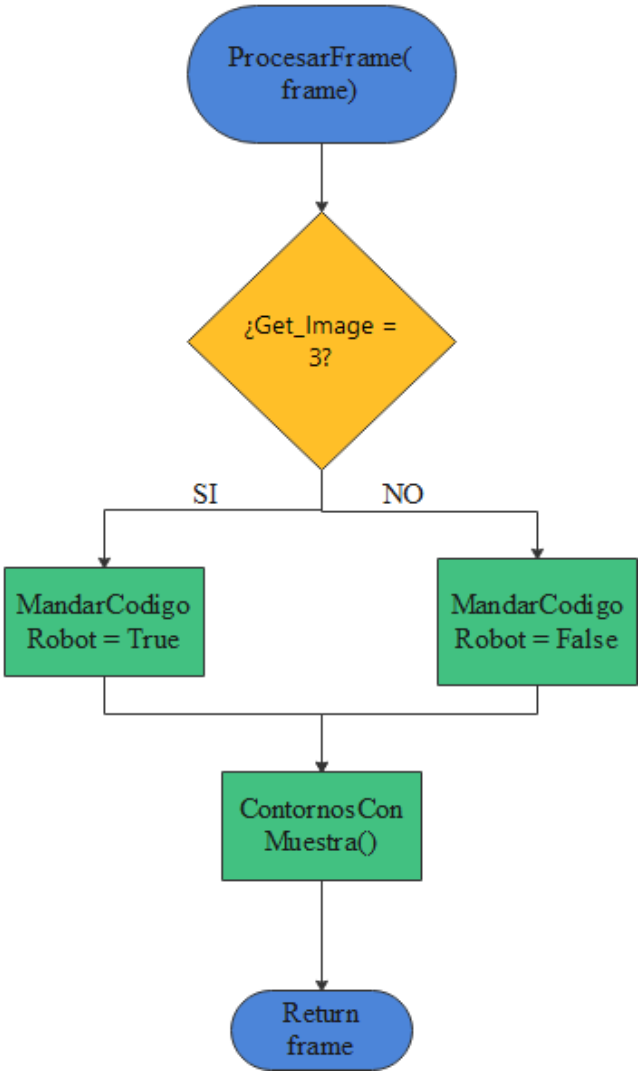
[32] Asus, «Xtion,» [En línea]. Available: https://www.asus.com/es/3D-Sensor/Xtion_PRO/.

ANEXO I: DIAGRAMAS DE FLUJO

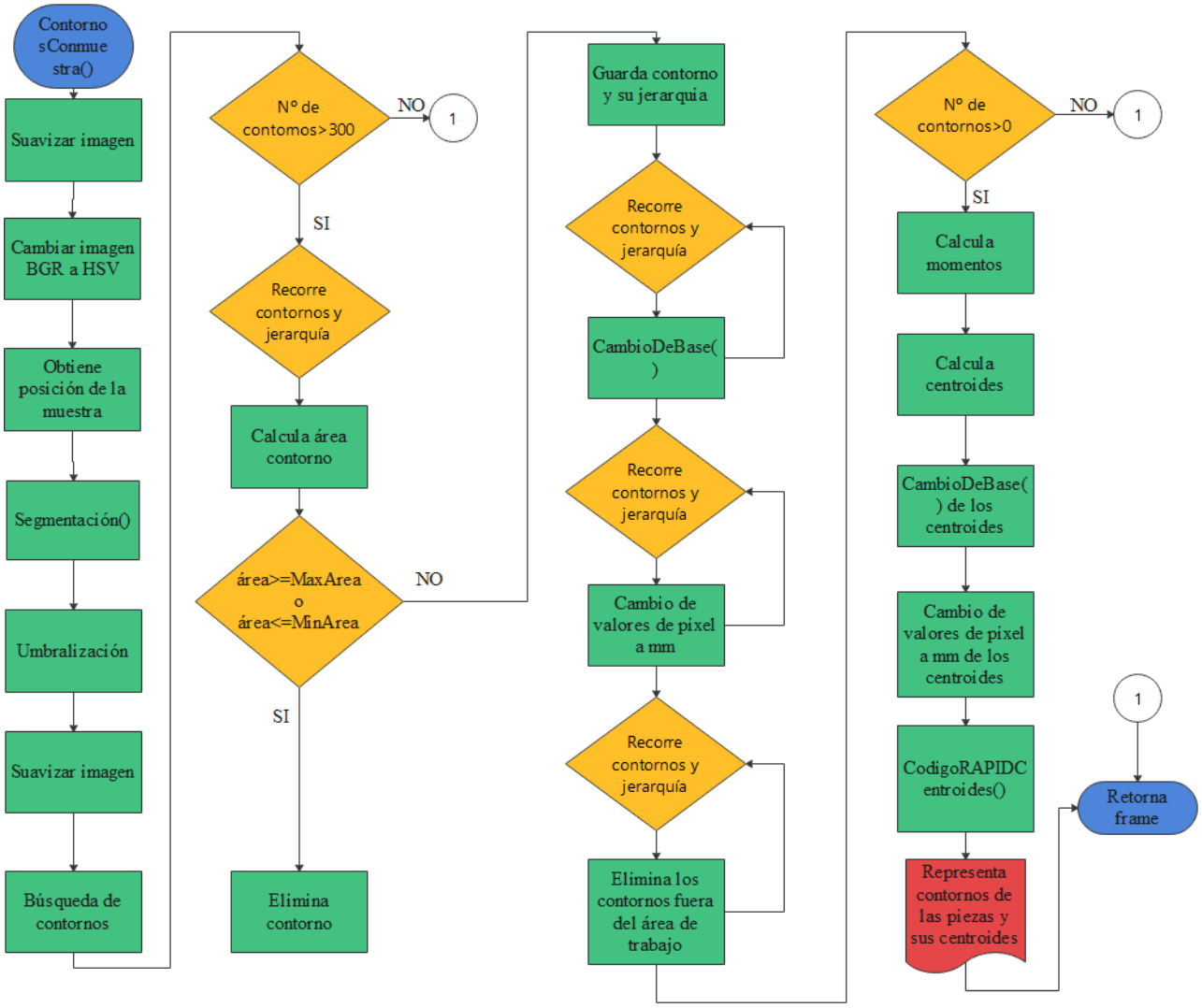
1. Diagramas de flujo visión 2D



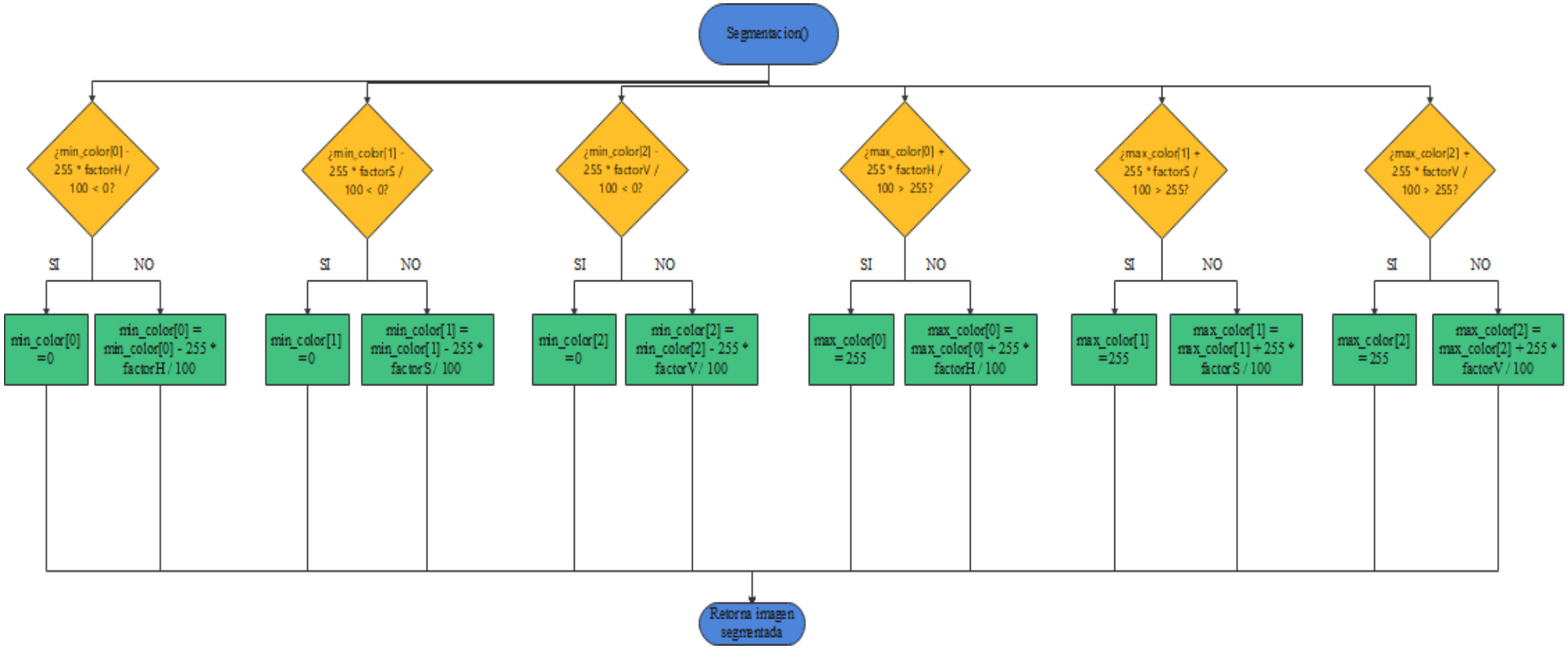
Flujograma 1: Programa principal



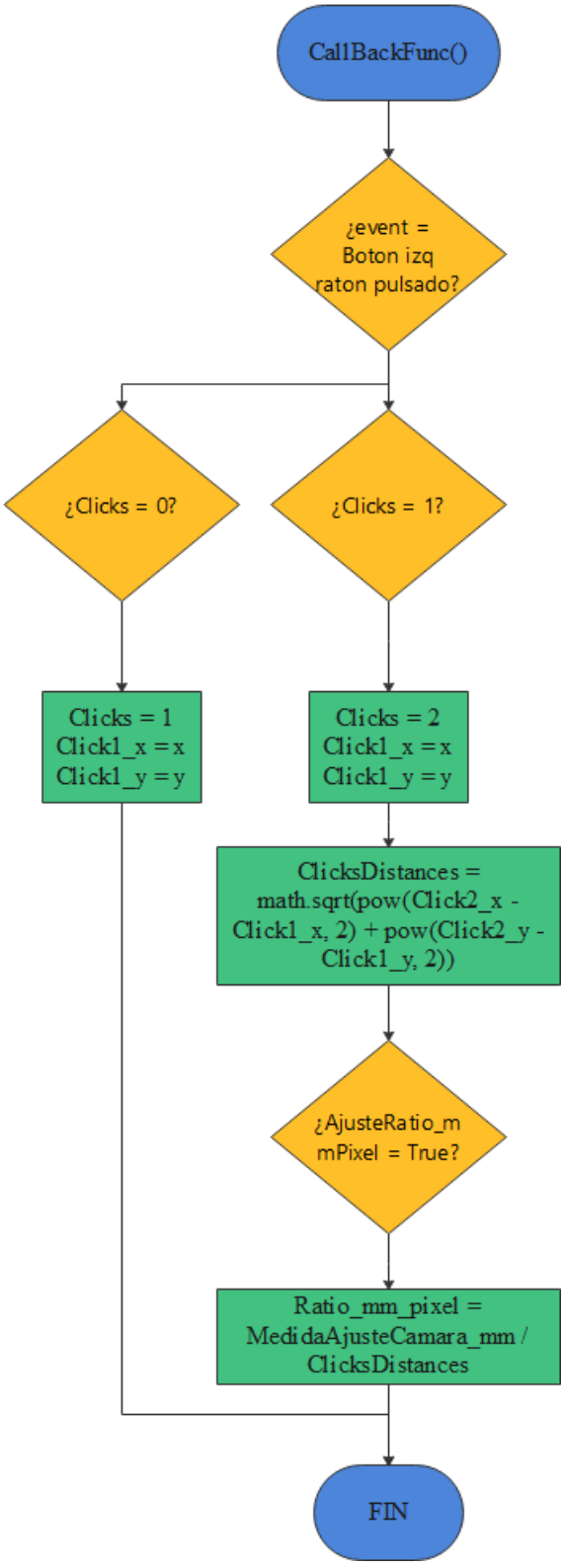
Flujograma 2: ProcesarFrame



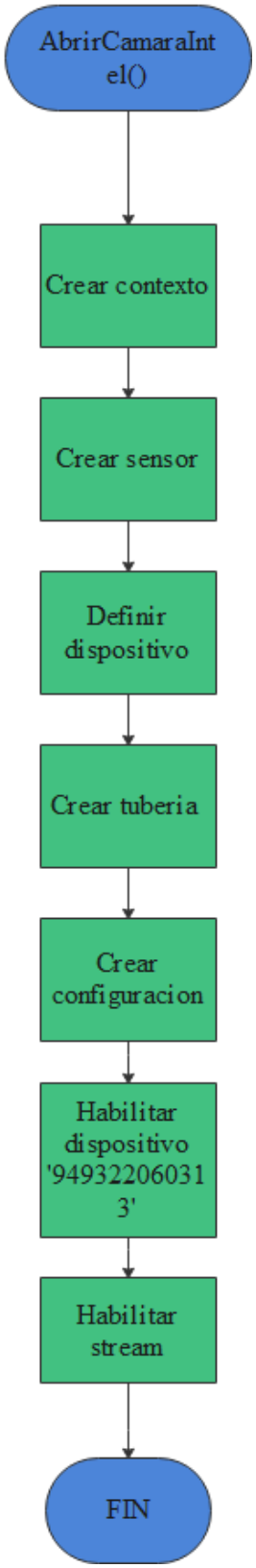
Flujograma 3: ContornosConMuestra



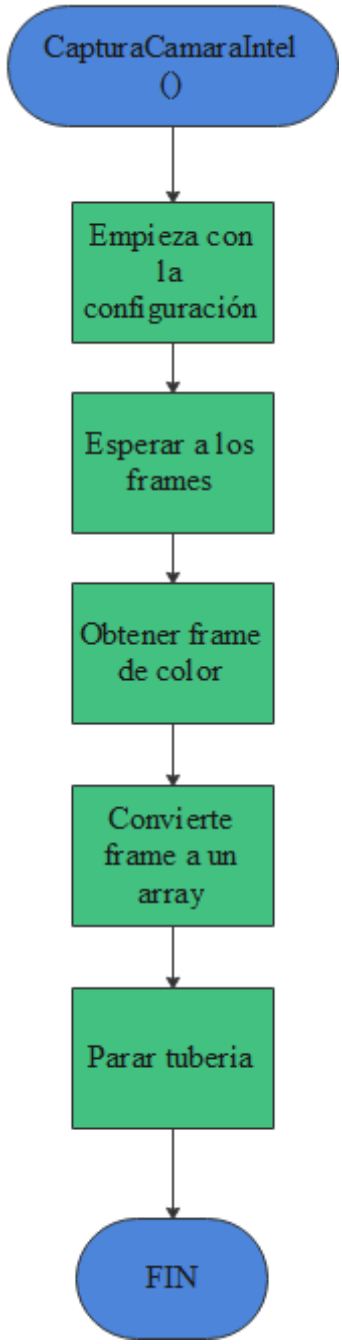
Flujograma 4: Segmentacion



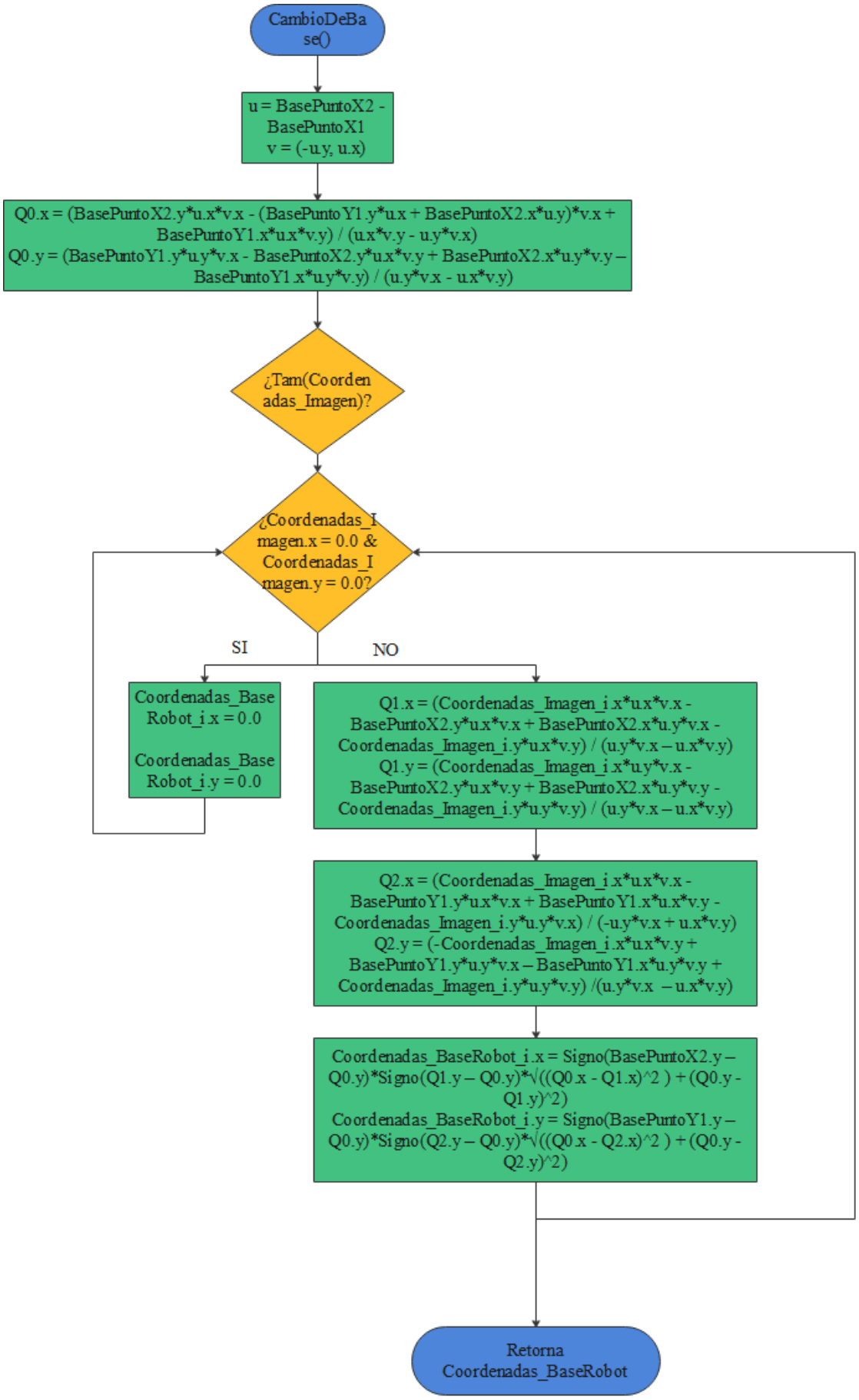
Flujograma 5: CallBackFunc



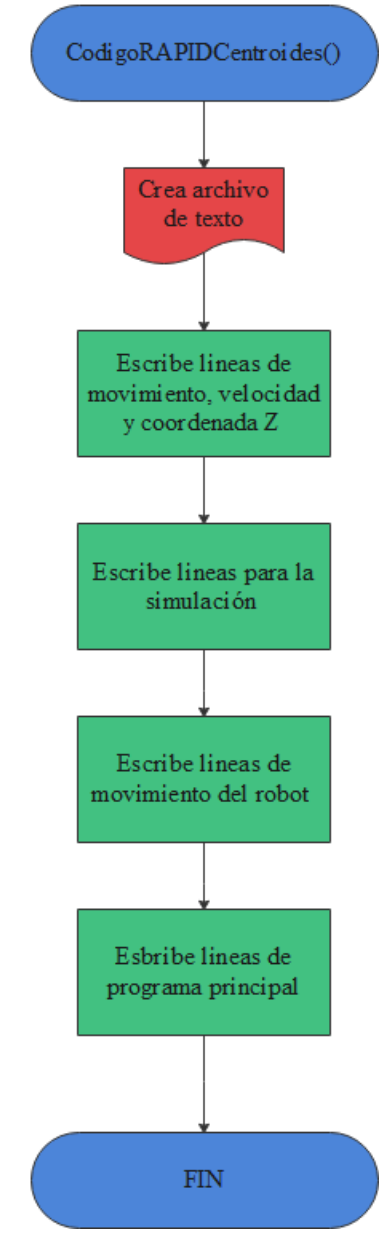
Flujograma 6: AbrirCamaraIntel



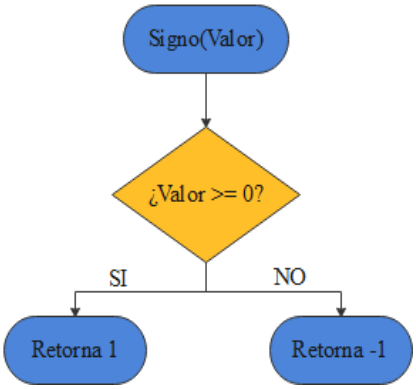
Flujograma 7: CapturaCamaraIntel



Flujograma 8: CambioDeBase

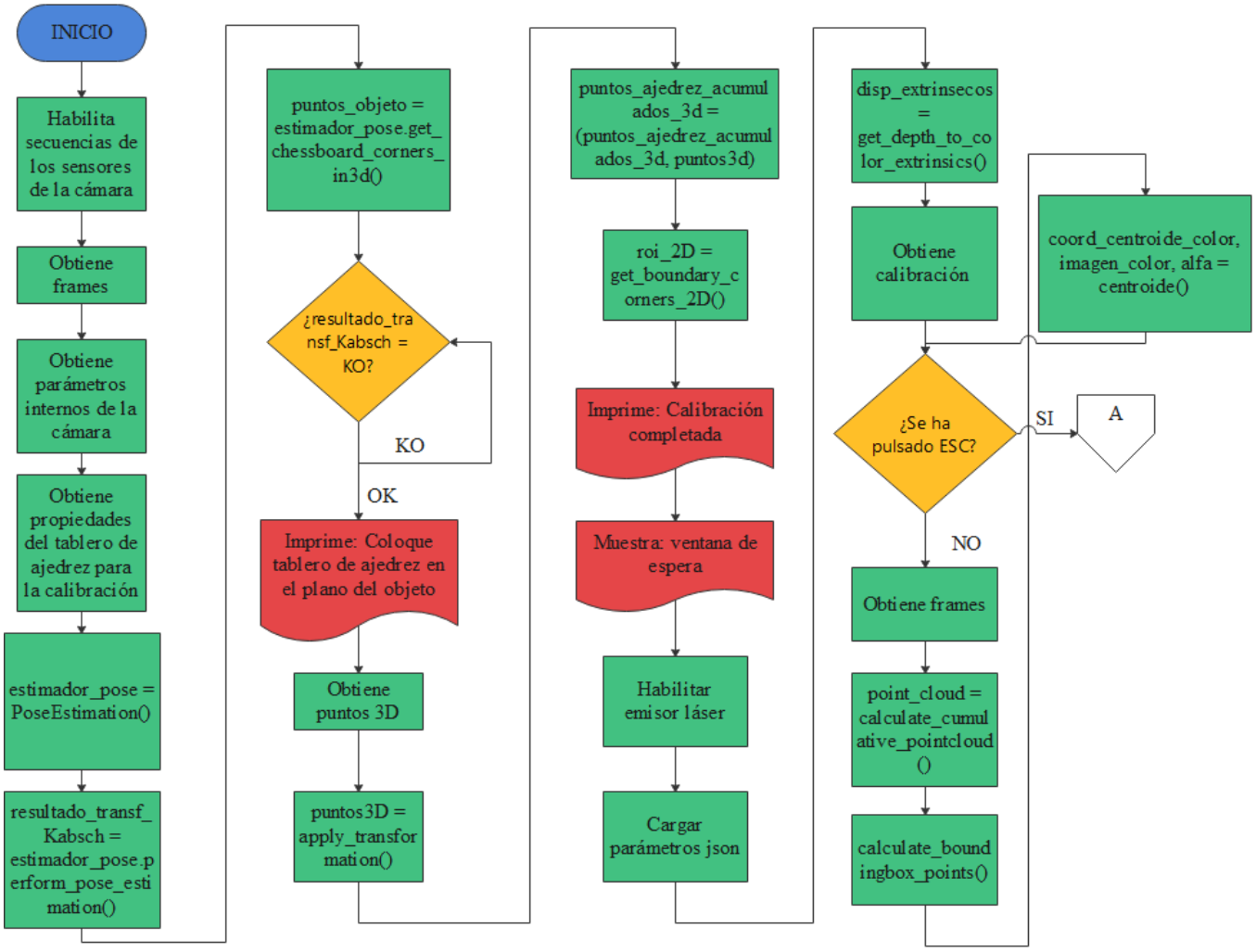


Flujograma 9: CodigoRAPIDCentroides

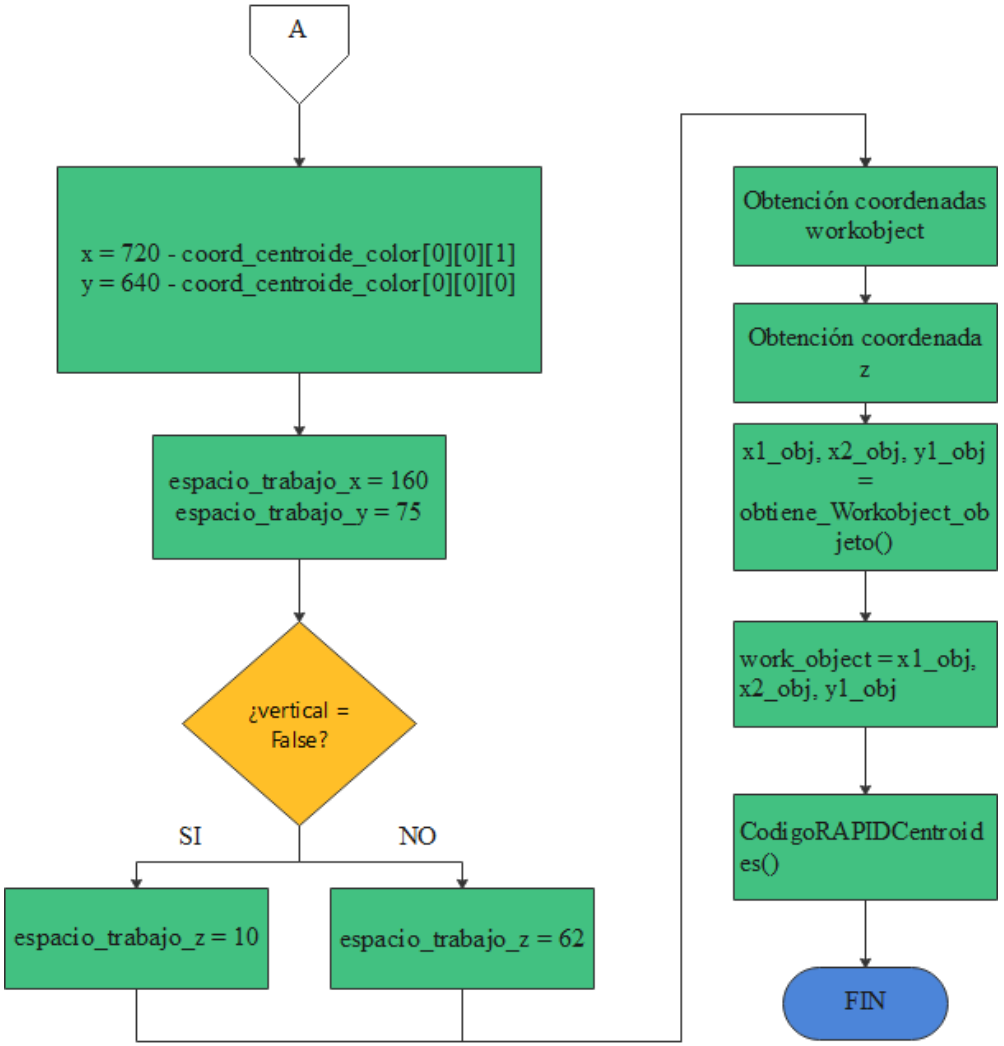


Flujograma 10: Signo

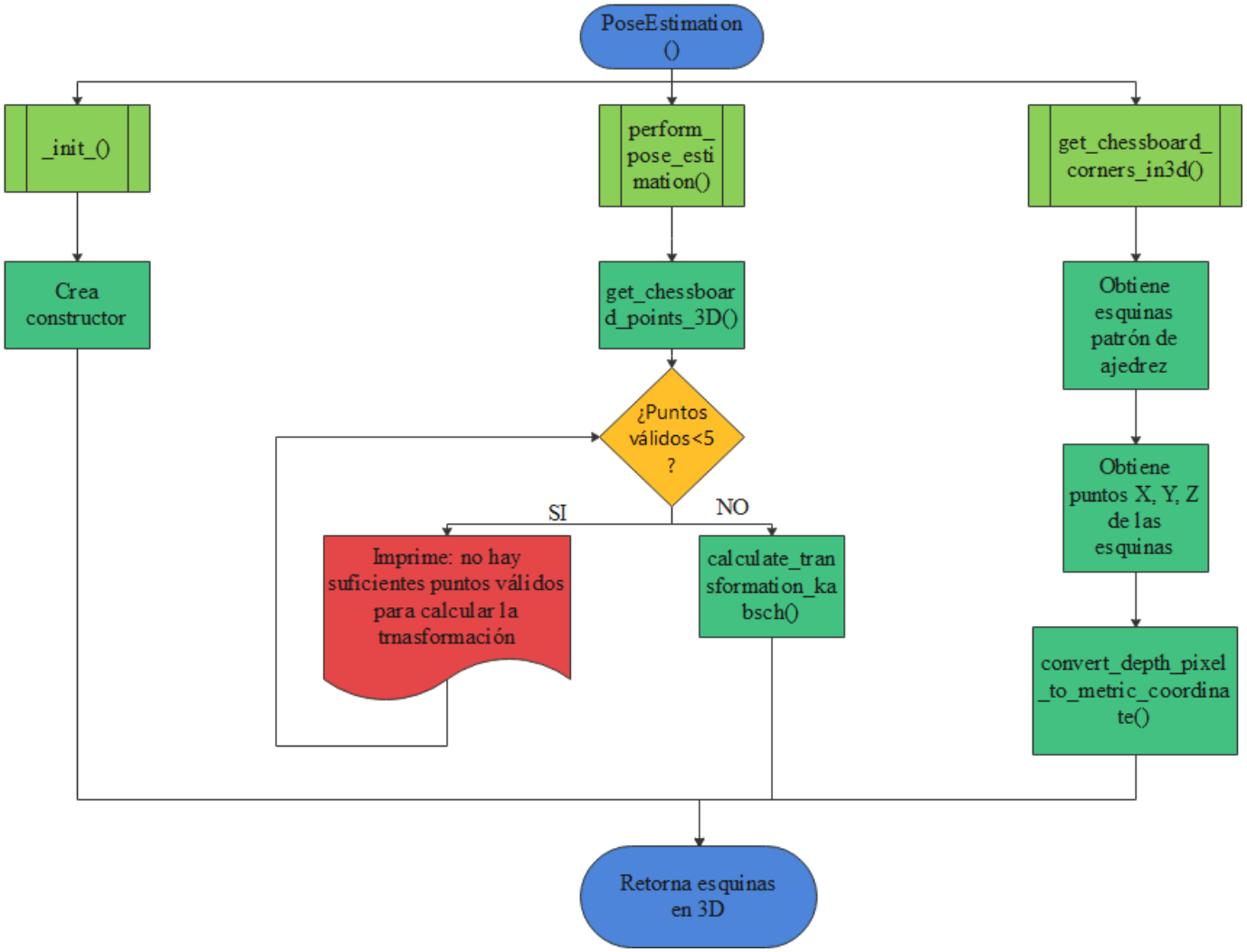
2. Diagramas de flujo visión 3D



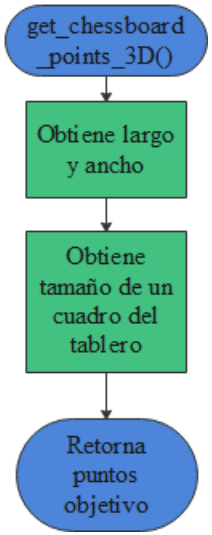
Flujograma 11: principal



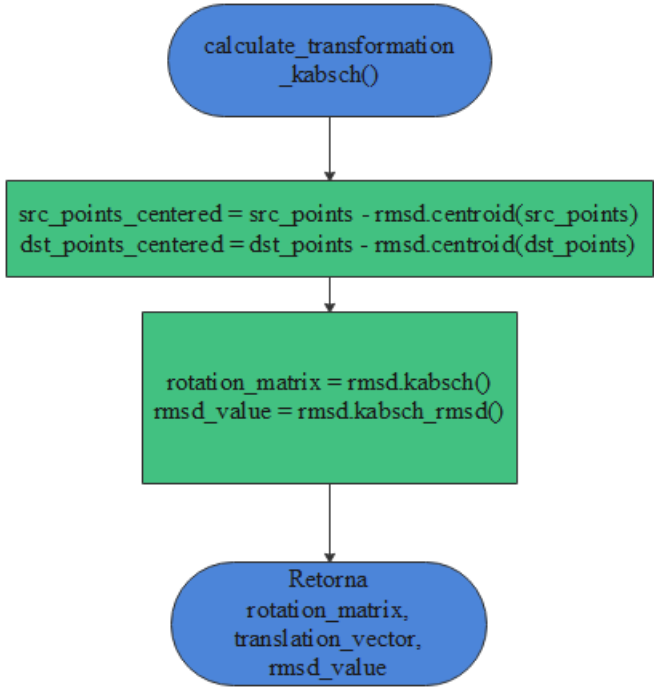
Flujograma 12: continuación principal



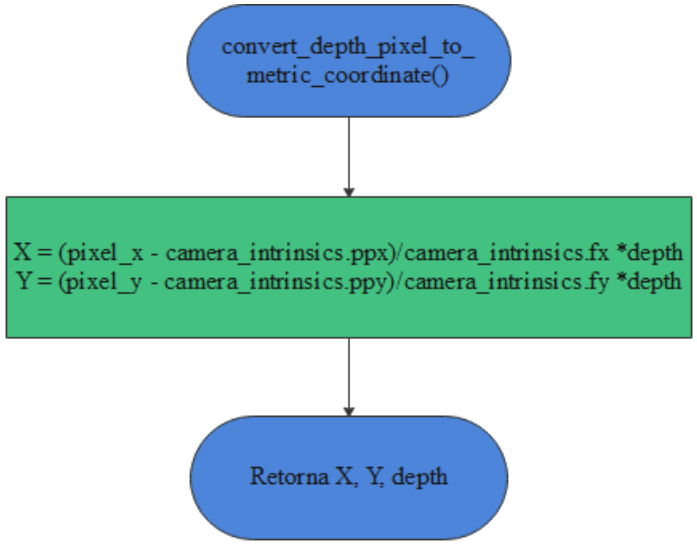
Flujograma 13: PoseEstimation



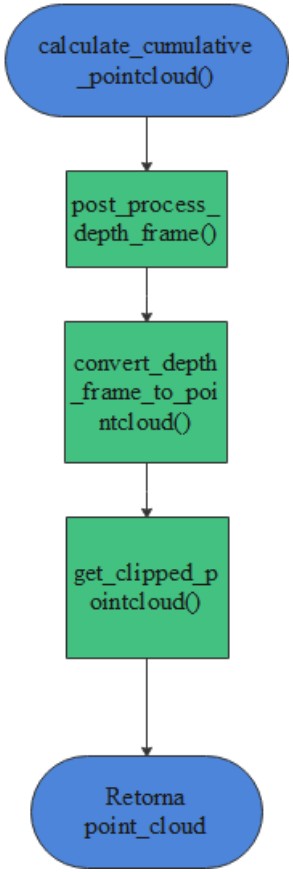
Flujograma 14: obtiene puntos en 3D del patrón de ajedrez



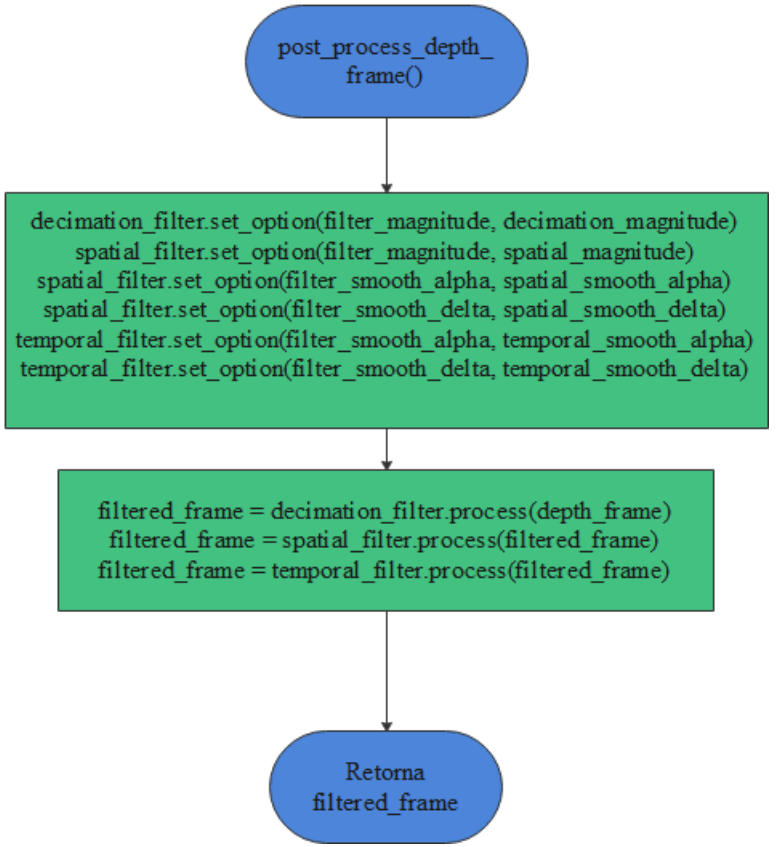
Flujograma 15: calcula transformacion Kabsch



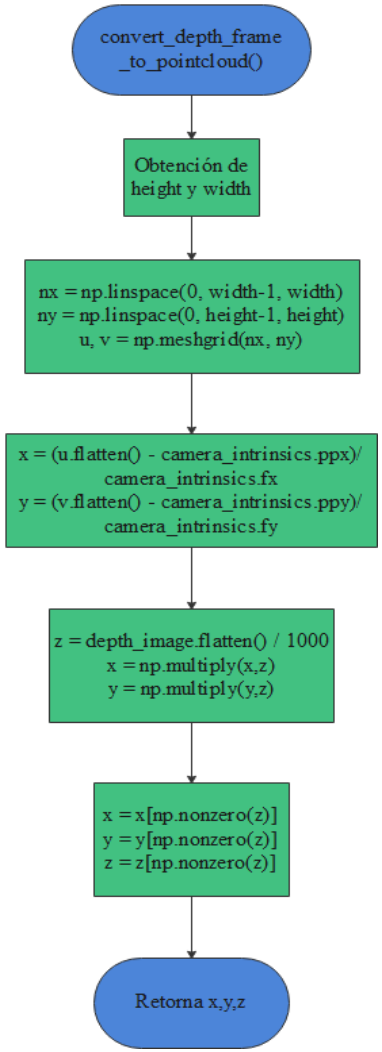
Flujograma 16: convierte coordenadas de píxeles a valor métrico



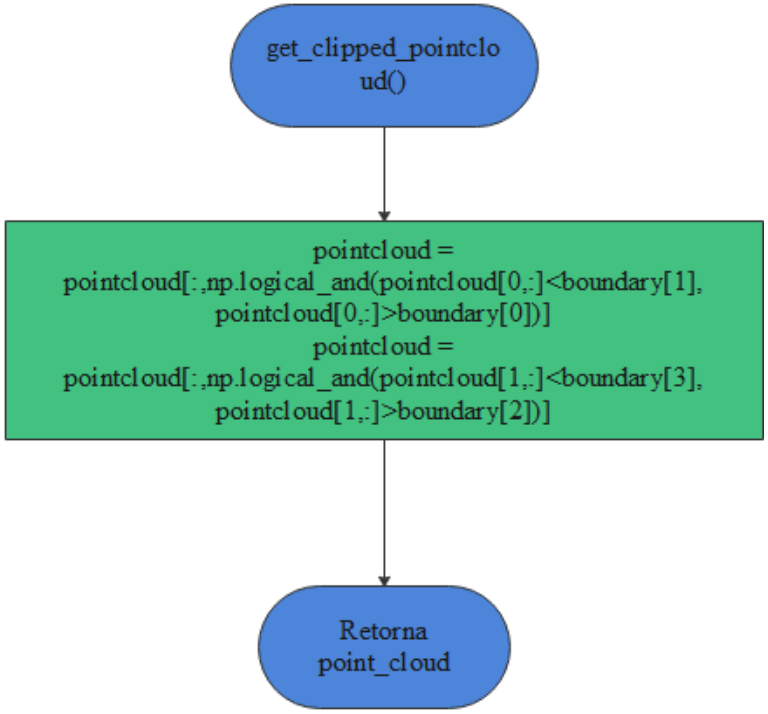
Flujograma 17: calcula nube de puntos acumulada



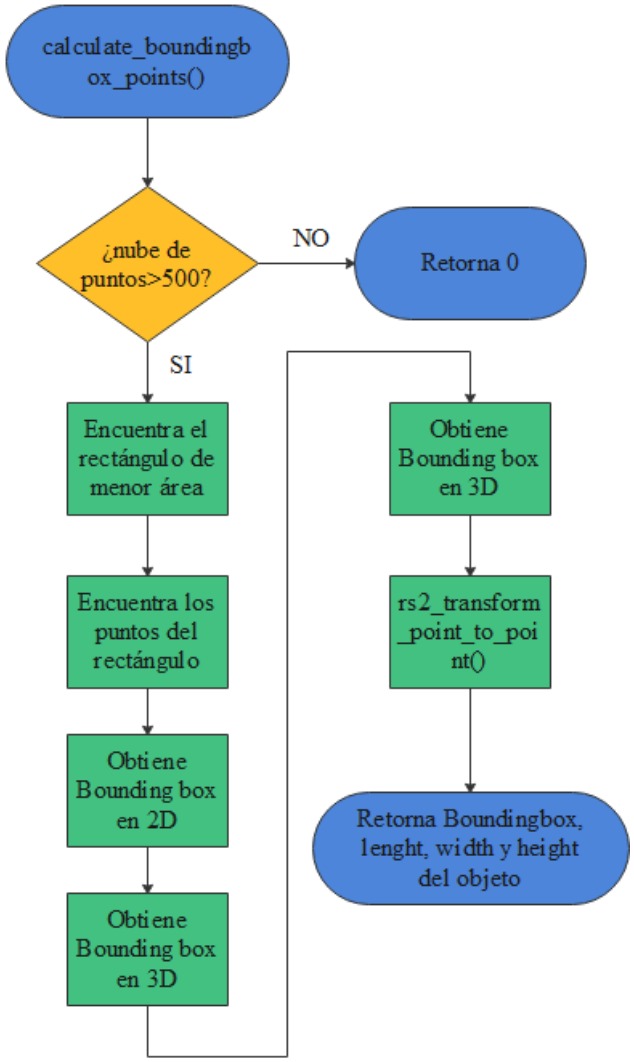
Flujograma 18: procesa frame de profundidad



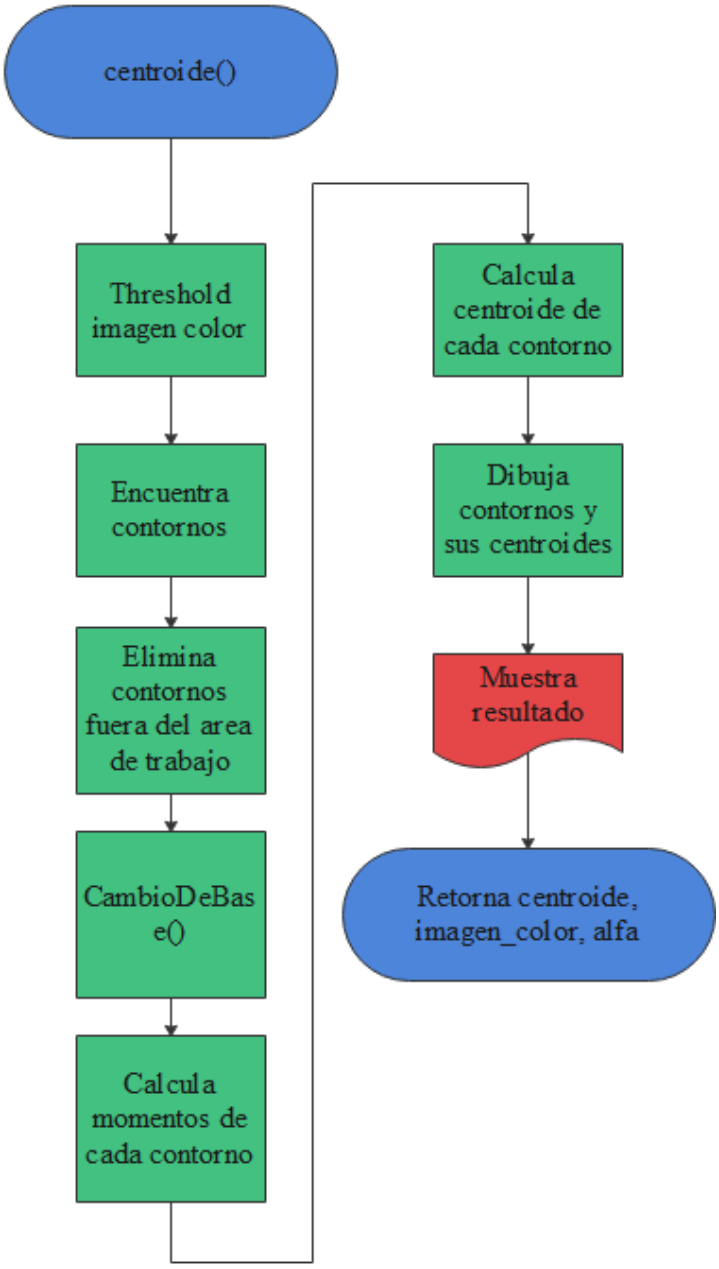
Flujograma 19: convierte frame de profundidad a nube de puntos



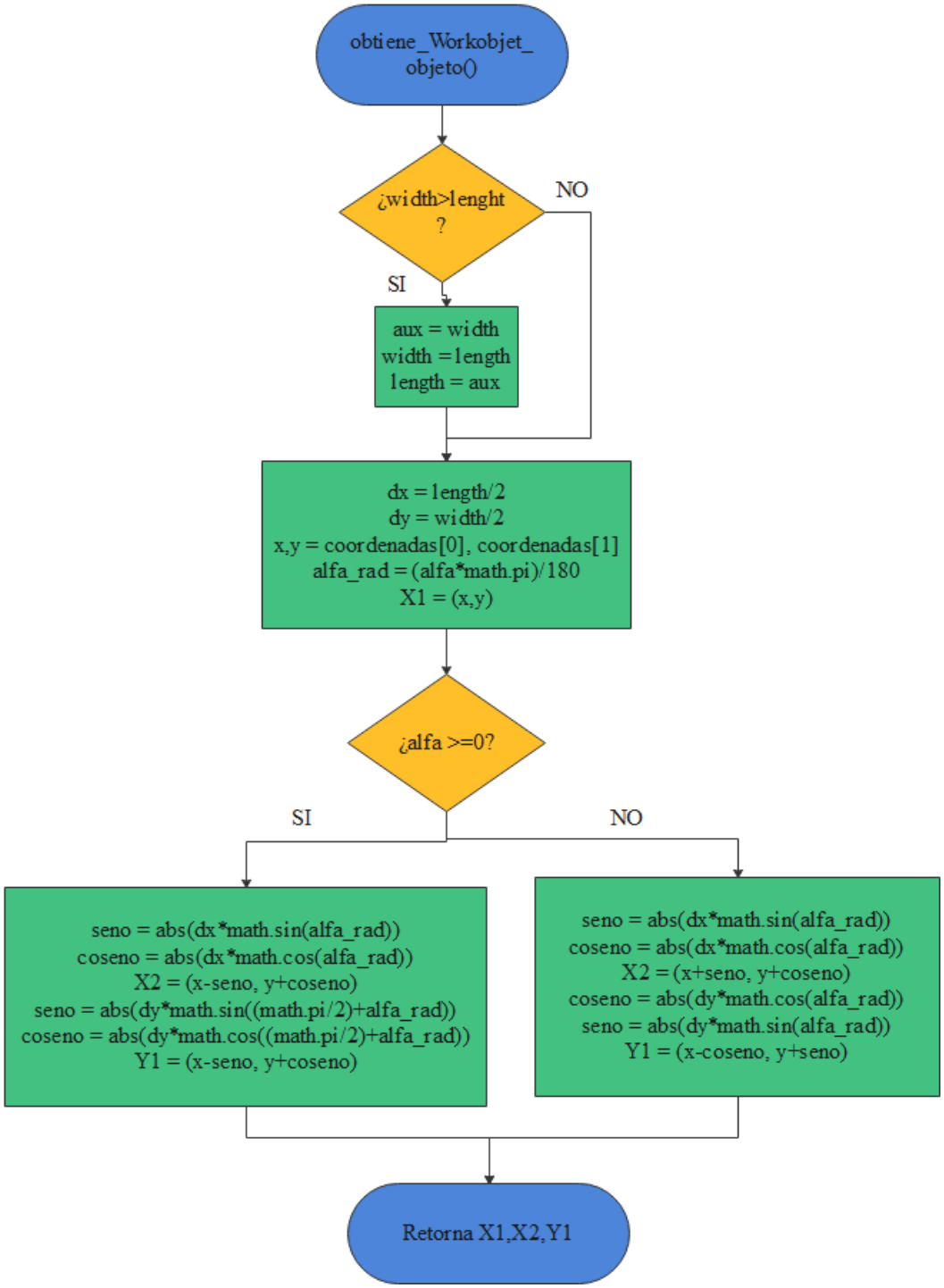
Flujograma 20: obtiene nube de puntos recortada



Flujograma 21: calcula los puntos del bounding box



Flujograma 22: centroide



Flujograma 23: obtiene Workobject objeto

ANEXO II: CARACTERÍSTICAS DE LA CÁMARA INTEL REALSENSE D415

La tecnología *deep sense* o detección de profundidad permite a este modelo obtener imágenes en 3D en forma de mapa de profundidad. Esta cámara tiene un par de sensores de profundidad, sensor RGB y proyector de infrarrojos, que, con una serie de algoritmos, se pueden obtener datos en forma de nubes de puntos.

Sus dos sensores de imagen permiten que consigan una resolución HD de hasta 1280 x 720, con 90 fps y un rango máximo de aproximadamente 10 metros. En su interior nos encontramos con un procesador D4 de visión potente de 28 nanómetros, potenciado por un algoritmo de profundidad estéreo para mejorar la percepción recogida por los sensores [25].

A continuación, se muestra una tabla comparativa con las características de las distintas cámaras de Intel:

	D400	**D410**	**D415**	**D420**	**D430**
Tecnología de profundidad	Estéreo pasivo	Estéreo activo IR	Estéreo activo IR	Estéreo pasivo	Estéreo activo IR
FOV profundidad	69.4° x 42.5° x 77°			91.2° x 62.5° x 100.6°	
Obturador	Persiana			Global	
Sensor RGB	NO		1920x1080 hasta 60 fps	NO	

Tabla 2: Comparación entre varios modelos

Ahora se va a comparar el modelo D415, con un modelo muy similar en prestaciones, la D435:

PARÁMETRO	**D415**	**D435**
Sensor	OV2740	OV9282
Píxeles	1920 x 1080	1280 x 800
Relación de aspecto	16:9	8:5
Formato	10-bit RAW	10-bit RAW
Número F	f/2.0	f/2.0
Distancia focal	1.88mm	1.93mm
Tipo de filtro	IR	Ninguno
Foco	Fijo	Fijo
Tipo de obturador	Persiana	Global
Interfaz de señal	MIPI CSI-2, 2X Lanes	MIPI CSI-2, 2X Lanes
FOV horizontal	69.4	91.2
FOV vertical	42.5	65.5
FOV diagonal	77	100.6
Distorsión	<=1.5%	<=1.5%

Figura 46: D415 vs. D435

Es necesario hacer una comparación de características con otro adversario muy empleado en este tipo de proyectos por otros estudiantes, ya que mientras se estudiaba la documentación se han encontrado muchas referencias, librerías y proyectos hechos con esta cámara. Se trata de la Kinect de Microsoft [26]:

PARÁMETRO	**D415**	**Kinect**
Sensores	RGB, profundidad	RGB, profundidad
Píxeles	1920 x 1080	640 x 480
FOV horizontal	69.4	57
FOV vertical	42.5	43
Rango de profundidad	10 m máx.	1.2-3.5 m
Otros	-	Micrófono multiarray

ANEXO III: VISIÓN 3D, ASPECTOS MAS IMPORTANTES

1. Tipos de datos en 3D

A partir de imágenes RGBD, es decir, RGB más imagen de profundidad, se puede obtener una nube de puntos que a su vez si ésta se procesa se pueden obtener otras estructuras de datos en 3D tales como la malla de triángulos y *lineset* (Figura 47), también se puede obtener una cuadrícula de *voxel (volume pixel)* o un octree(Figura 48).

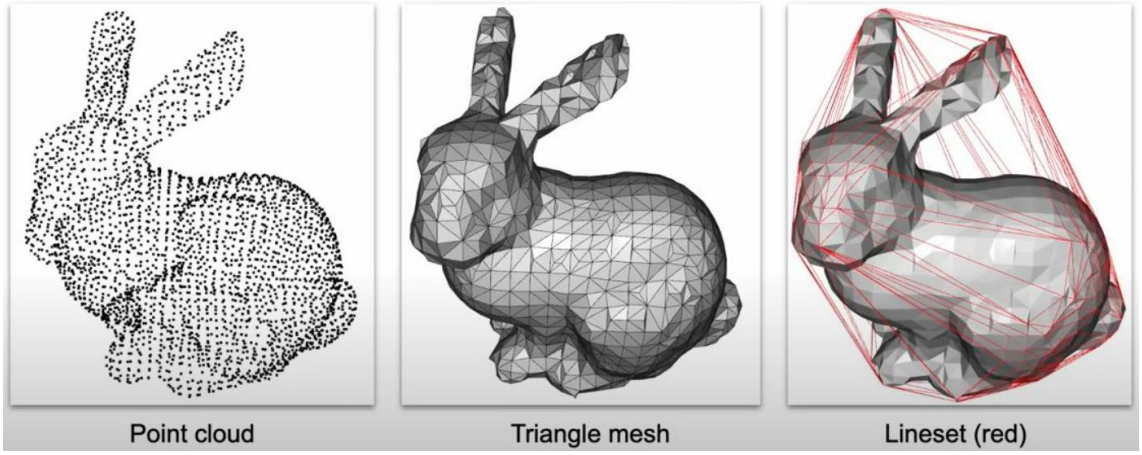


Figura 47: Tipos de estructuras de datos en 3D (I)

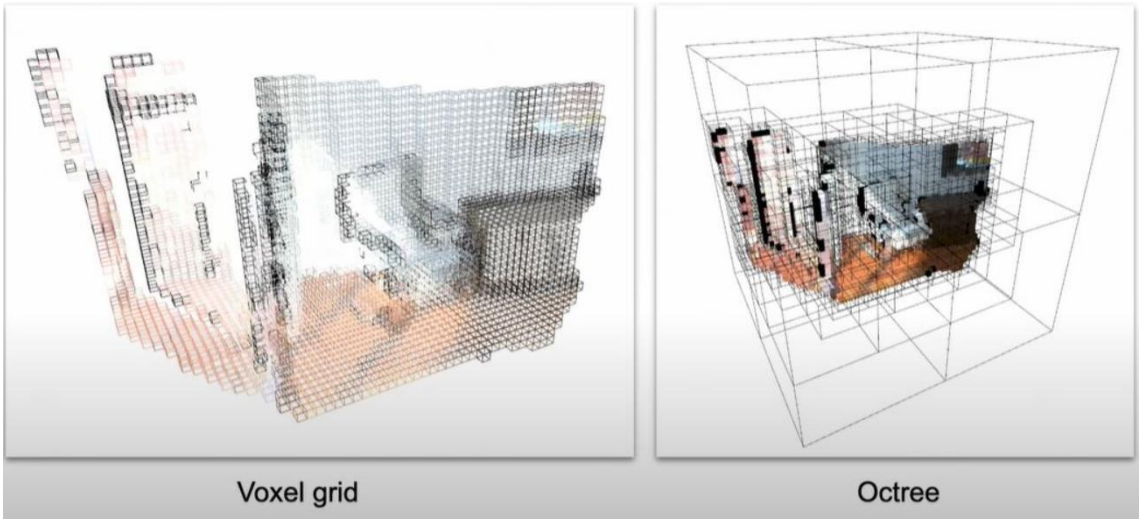


Figura 48: Tipos de estructuras de datos en 3D (II)

El voxel es la mínima unidad cubica que compone un objeto en 3D y un árbol octal o *octree* es una estructura de datos en la cual cada nodo posee 8 octantes.

2. Proceso de reconstrucción 3D y alineamiento

Cuando se requieren tareas de *binpicking*, donde las piezas se encuentran apiladas de forma aleatoria en un contenedor, es necesario un proceso de reconstrucción de la escena, es decir, conseguir digitalmente una escena en 3D sin oclusiones. Esto se realiza desde diferentes puntos de vista o *scan* del contenedor con las piezas, lo que se obtiene una representación parcial de la escena. Para obtener la imagen final se deben alinear todos los *scans* en un único sistema de coordenadas.

La técnica más común empleada para realizar la reconstrucción 3D más el alineamiento de los *scans* es SfM o estructuración a partir del movimiento. Esto es, a partir de tener varias secuencias, secuencias de imágenes en RGB y de profundidad (Figura 48), se forman fragmentos parciales, se muestra en la Figura 49, de la escena.

Figura 49: Sistema de binpicking

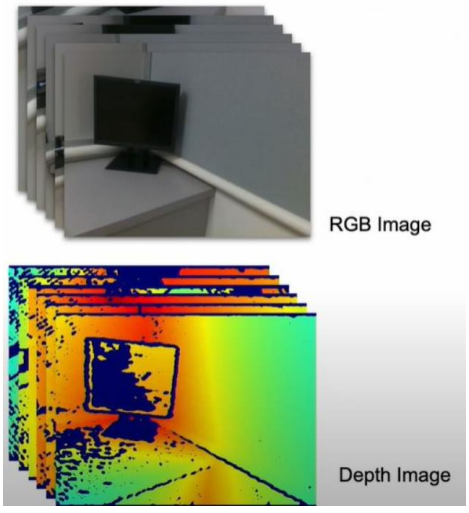


Figura 50: Secuencia de imágenes RGB y profundidad

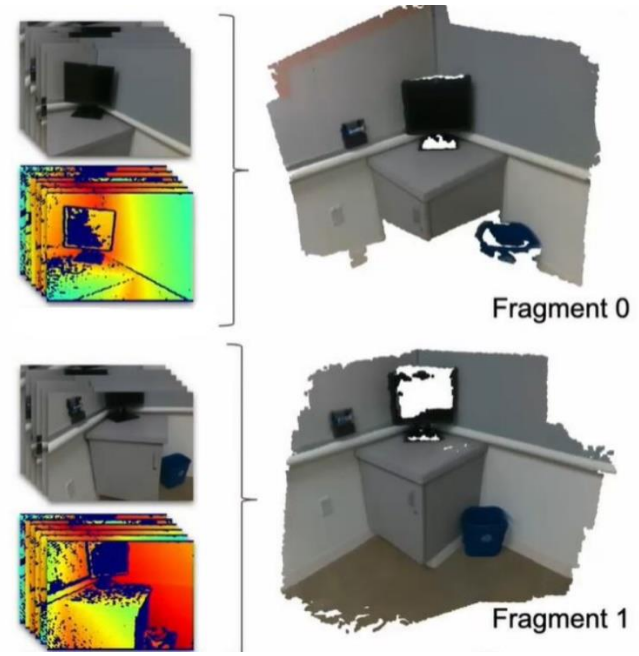


Figura 51: Creación de fragmentos a partir de secuencias

Por último, se registra los fragmentos en forma de nube de puntos y se obtiene la transformación entre los fragmentos, esto es, se encuentran los puntos comunes en cada escena y se alinean en una única imagen, ver Figura 50 [27].

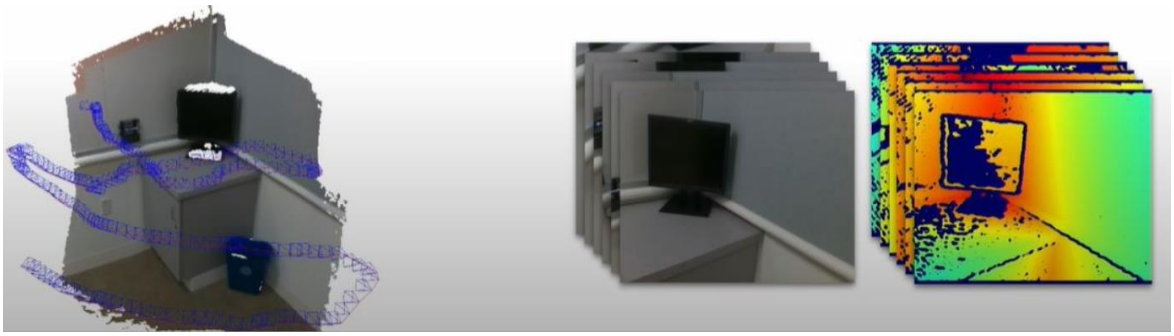


Figura 52: A partir de capturas en RGBD se obtiene el 3D de la escena en forma de nube de puntos. El trazo azul de la izquierda es el recorrido en forma de barrido que ha hecho la cámara sobre la escena

Para realizar lo descrito anteriormente, la transformación que se emplea es la ya comentada en el apartado 2.1.4 que es la rígida o Euclidiana entre los dos sistemas de referencia, alineando los puntos correspondientes a las dos nubes de puntos (Figura 50), esto se denomina *scan matching (Figura 51).*

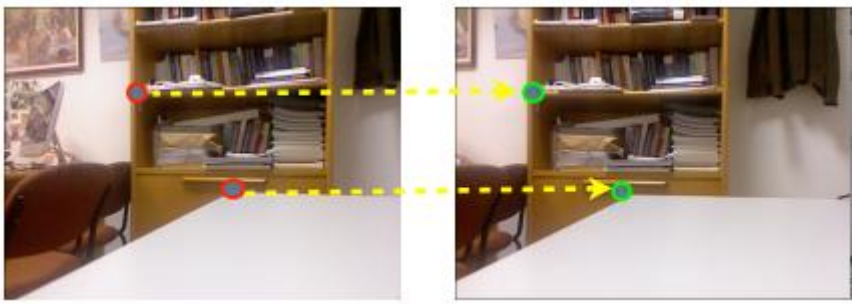


Figura 53: Puntos comunes en cada scan de la escena

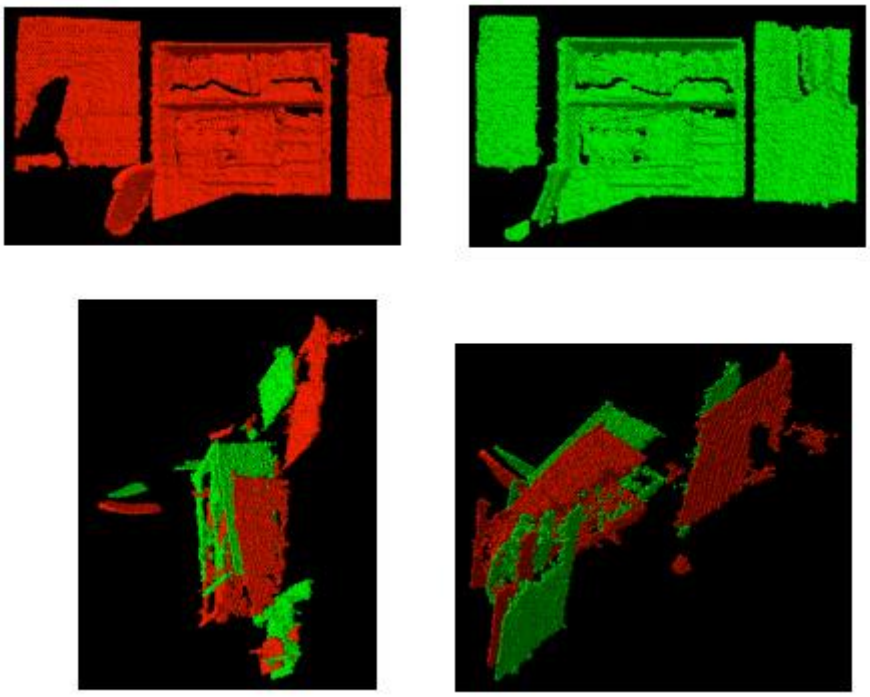


Figura 54: Scan Matching

El *matching* tiene un gran inconveniente de optimización, por lo que se suele emplear una función de coste que mide la calidad de la alineación entre puntos obtenida.

Por lo general, en este tipo de procesos, se suele emplear el algoritmo ICP para obtener la transformación rígida (R,t). Este es un algoritmo iterativo basado en la correspondencia punto a punto, que, a diferencia del utilizado en este proyecto para la obtención de la transformación, en este se desconoce la correspondencia entre puntos.

Para conseguir tiempos de emparejamiento de puntos más cortos, se emplea un *Kd-tree,* es una estructura de datos de particionado del espacio que organiza los puntos en un Espacio euclídeo de k dimensiones [28].

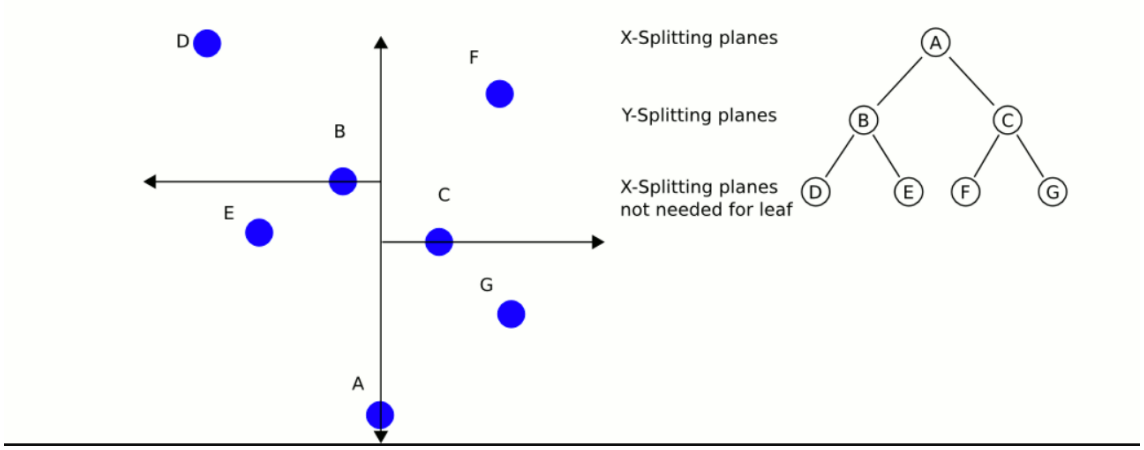


Figura 55: Árbol Kd

Posteriormente se realiza una ponderación y se aplican pesos a cada punto. A continuación, se eliminan las correspondencias, también llamado *outliers,* con distancia entre sus puntos mayores a un umbral fijado, así se eliminan los emparejamientos que no se desean, incluso se pueden eliminar también emparejamientos vecinos, los cuales se pueden considerar inconsistentes.

Para medir el error de emparejamiento se usan dos técnicas:

- Punto a punto: considerando la suma de las distancias al cuadrado entre puntos.
- Punto a plano: considerando la suma de las distancias al cuadrado entre un punto y su plano más cercano.

Por último, para conseguir acelerar la convergencia del algoritmo, se minimiza de forma iterativa el error.

Se ha explicado el método basado en puntos, ya que posiblemente sea el más utilizado, pero también existen otros dos métodos:

- Basado en características: se extraen las características de la imagen.
- Punto a característica: combina características y puntos de las nubes.

3. Segmentación

Los más comunes son los métodos basados en bordes, regiones y atributos. Pero en visión 3D y concretamente en tareas de *binpiking* es muy común emplea métodos basados en modelo, esto es que usa primitivas geométricas, tales como la esfera, el cilindro, cono y plano para agrupar los distintos puntos. Se agruparán aquellos puntos que tengan la misma representación matemática. Para solucionar esto el método de referencia es emplear el algoritmo de RANSAC, que es un método iterativo, el cual, a más iteraciones, mayor probabilidad hay de obtener un resultado razonable (Figura 56). También es típico usar la transformada de Hough (Figura 57) para líneas y círculos, basado en un sistema de votación, cada punto de la nube de puntos se le da un valor y solo pasaran los puntos que superen un cierto valor umbral.

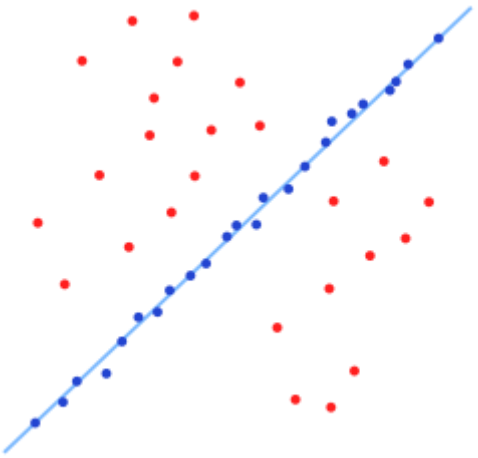


Figura 56: RANSAC

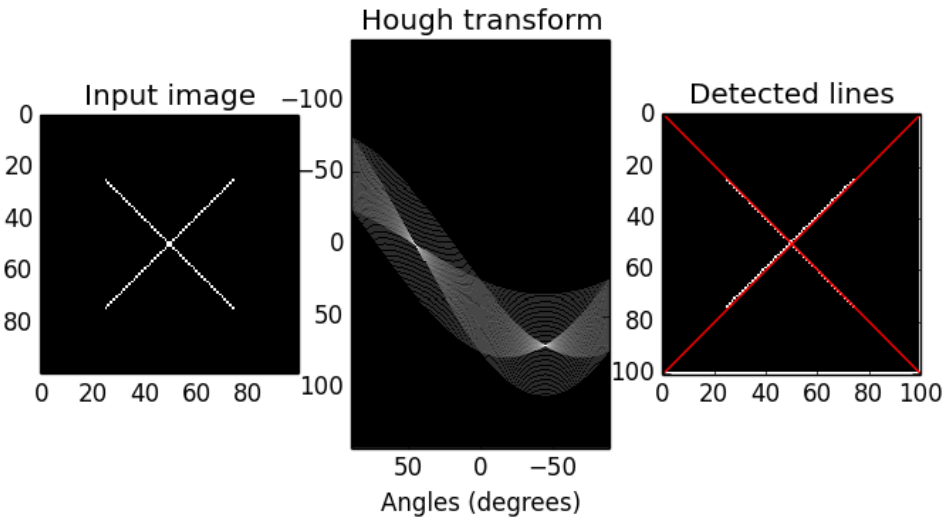


Figura 57: Transformada de Hough

4. Reconocimiento de objetos en 3D

En esta tarea es muy común ver soluciones que emplean Deep Learning como apunta el *paper* basado en detección y reconocimiento de objetos en tiempo real usando Deep Learning [29], ver Figura 58. Este emplea una modificación de YOLO, que es otra herramienta de detección rápida de objetos [30] .

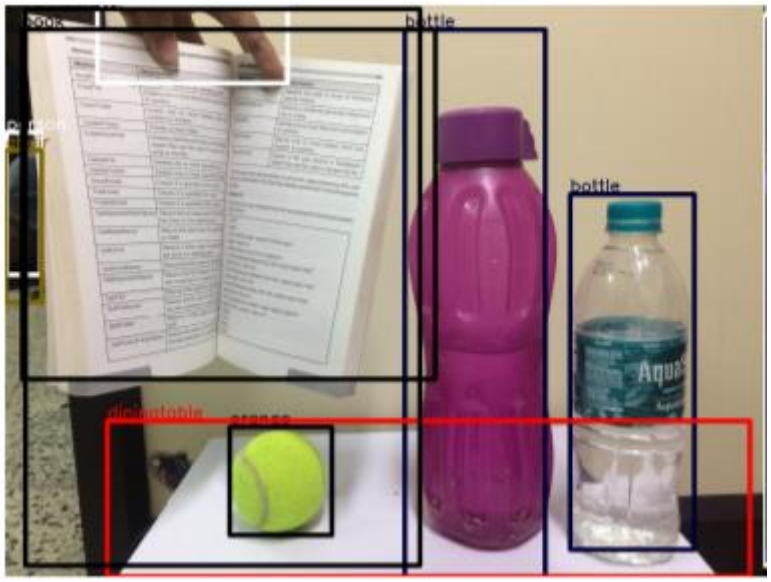


Figura 58: Reconocimiento de objetos en tiempo real

Por exponer otro ejemplo, ya que son muy numerosas las publicaciones sobre este tema, PonitNet [31], organiza la nube de puntos en la estructura vista anteriormente llamada cuadrícula de *voxel.* Esta herramienta aprovecha la arquitectura de convolución.

Un flujo tradicional para estas tareas podría ser:

i. Se tiene una imagen de la escena y del objeto a encontrar en ella en forma de nubes de puntos.
ii. Se hace una extracción de características(SIFT, Harris), Figura 59 y se obtienen los puntos clave.
iii. Se computan los descriptores, Figura 60 (SHOT, RIFT, FPFH) y se calculan los emparejamientos (KdTree), Figura 61.
iv. Se aplica el algoritmo RANSAC.
v. Se calcula la transformación rígida empleando el algoritmo descrito anteriormente, ICP (método iterativo del punto más cercano).

Figura 59: Extracción de características

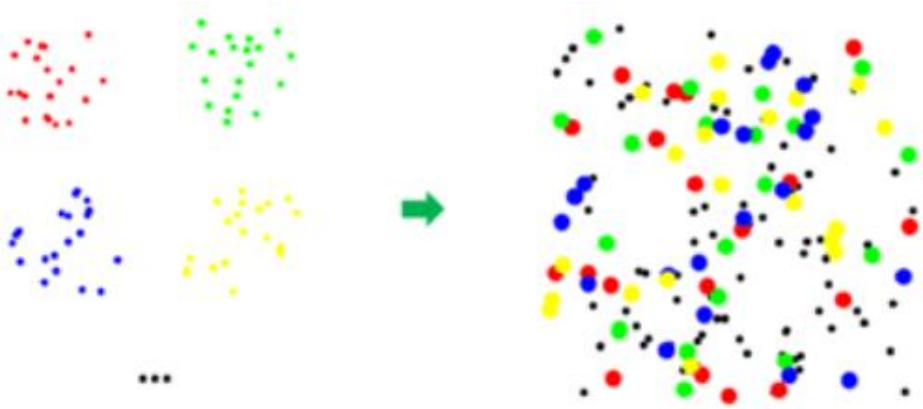


Figura 60: Computa descriptores

Figura 61: Cálculo de emparejamientos

ANEXO IV: PRESUPUESTO

Hardware				
Descripción	Componente	Precio unitario	Ud	Total
Ordenador portátil MSI GP63	Portátil	1000	1	1000
Cámara Intel RealSense D415	Cámara	341,55	1	341,55
Trípode	Trípode	33,99	1	33,99
Total				1375,54

Tabla 3: Presupuesto hardware