UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA
INGENIERÍA ELÉCTRICA – SISTEMAS ELECTRÓNICOS

SISTEMA DE VISIÓN COMPUTACIONAL PARA EL AGARRE DE PIEZAS DE MANUFACTURA IMPLEMENTADO EN SISTEMAS EMBEBIDOS

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN INGENIERÍA

PRESENTA:
FÍS. VALENTE VÁZQUEZ VELÁZQUEZ

TUTOR:
DR. JUAN MARIO PEÑA CABRERA,
INSTITUTO DE INVESTIGACIONES EN MATEMÁTICAS
APLICADAS Y EN SISTEMAS

CIUDAD UNIVERSITARIA, CD. MX., FEBRERO 2022

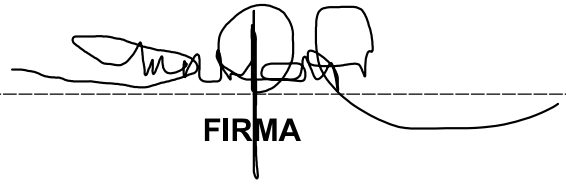
JURADO ASIGNADO:

Presidente:	Dr. Escalante Ramírez Boris
Secretario:	Dr. Pérez Alcázar Pablo Roberto
1 er. Vocal:	Dr. Peña Cabrera Juan Mario
2 do. Vocal:	Dra. Navarrete Montesinos Margarita
3 er. Vocal:	Dr. De La Rosa Nieves Saúl

Lugar o lugares donde se realizó la tesis: Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, U.N.A.M.

TUTOR DE TESIS:

Dr. Juan Mario Peña Cabrera



FIRMA

(Segunda hoja)

Este escrito va dedicado a ti
que estás leyendo esto.

"Aquellos que se enamoran solo de la práctica,
sin cuidar de la exactitud, o por mejor decir, de la ciencia,
son como el piloto que se embarca sin timón ni aguja;
y así nunca sabrá dónde va a parar".

- Leonardo da Vinci

Agradecimientos

En primer lugar, debo agradecer por todo el apoyo, la paciencia y la comprensión a las cuatro personas más importantes en mi vida: Lucina, Arturo, Teresa y Julieta, mi familia. Gracias por absolutamente todo.

A mi tutor: Dr. Juan Mario Peña Cabrera; le agradezco sus palabras, consejos, y sobretodo, su orientación para la realización de este trabajo de tesis.

A mis sinodales: Dr. Escalante Ramírez Boris, Dr. Pérez Alcázar Pablo Roberto, Dra. Navarrete Montesinos Margarita y Dr. De La Rosa Nieves Saúl; muchas gracias por tomarse el tiempo de revisar este texto y aportar sus comentarios pertinentes para su debida conclusión.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por la beca otorgada durante los dos años de estudios.

Un enorme agradecimiento al Laboratorio de Electrónica y Automatización para la Industria 4.0, ubicado dentro del IIMAS, por el espacio y material brindado para la elaboración y conclusión de este trabajo de tesis.

A mis compañeros de momentos: Miguel, Magdiel, Dulce, Anamallery, Borre, Paco y Luis; gracias por seguir haciendo placentero el camino, amigos. Además, un especial agradecimiento a las familias Guillén Ángeles y Ramos Gayol.

A la familia Acevedo Molina; muchas gracias por todo su apoyo tanto a mí como a mi familia.

Por último, un agradecimiento muy especial al Dr. Fernando Angeles Uribe, a la Dra. Donají Xochitl Cruz López y al Lic. en C. C. Luis Alberto Ramirez Bermudez; gracias por su amistad, su apoyo, sus regaños, sus consejos, la guía, las clases, las ñoñeces, las comidas, las sobremesas, las risas interminables, los viajes... y demás; sin lugar a dudas, lo aprendido con ustedes sigue siendo inmensurable.

Índice general

Acrónimos

Índice de figuras

Índice de tablas

Resumen

Actualmente vivimos en la cuarta revolución industrial (también llamada Industria 4.0), en donde todo dispositivo electrónico deberá estar conectado a internet (Drath and Horch, 2014). En esta Industria 4.0, existe una competencia sobre obtener la mayor eficiencia en líneas de producción en empresas alrededor del mundo utilizando nuevas tecnologías, como el Internet de las Cosas, la Inteligencia Artificial y la Ciencia de Datos, por mencionar algunas, esto debido a que la industria que obtenga los mejores resultados utilizándolas, serán las que obtengan mayores ganancias económicas (CIC, 2019).

Este trabajo surge de la necesidad de contar con una Celda de Manufactura Flexible Inteligente dentro del Laboratorio de Electrónica y Automatización para la Industria 4.0 ubicado en el Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas de la Universidad Nacional Autónoma de México, con el fin de aportar el desarrollo académico, científico y tecnológico que actualmente se necesita en la industria del país. Por tanto, se diseña e implementa un Sistema de Agarre de Piezas de Manufactura, específicamente, un sistema autónomo basado en algoritmos de Visión Artificial entre un brazo robótico y una banda transportadora, creando el entorno necesario para sujetar al vuelo diferentes tipos de objetos de geometría simple de una banda con velocidad y dirección variables, utilizando sistemas embebidos y protocolos del ámbito del Internet de las Cosas, esto con el fin de integrar este sistema como un elemento dentro de la celda de manufactura mencionada anteriormente, la cual está compuesta por varios elementos, donde una banda transportadora y un brazo robótico Kuka KR 5-2 arc HW son los elementos principales para la integración dinámica en el transporte de piezas dentro de ella.

1

Introducción

"Solo se progresa cuando se piensa que se puede hacer algo más".

– Guillermo Marconi

1.1. Motivación

Actualmente, las necesidades que se tienen dentro de la industria han aprovechado los avances tecnológicos con el único fin de optimizar los tiempos de producción. Estos avances son específicamente en las áreas de la electrónica y computación ya que cada vez es más fácil y económico obtener estos recursos. Estas necesidades que surgen en un Laboratorio de Robótica, pueden ser muy básicas o muy complejas, todo depende de los requerimientos y especificaciones técnicas: desde utilizar microcontroladores sencillos, hardware reconfigurable, sistemas embebidos, computadoras avanzadas o hasta necesitar de estaciones de trabajo enteras y equipadas, este es el caso del Laboratorio de Electrónica y Automatización para la Industria 4.0 (LEAI 4.0) que se encuentra dentro del Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas (IIMAS) de la Universidad Nacional Autónoma de México (UNAM).

Es por lo anterior que, en este laboratorio, surge la necesidad de realizar investigación y desarrollo tecnológico en temas relacionados a la robótica, sistemas de visión artificales

e Internet de las Cosas (IoT) para aplicarlos dentro la industria, es decir, implementar nuevas técnicas dentro de la Industria 4.0, y para llevar a bien este fin es necesario contar con equipo y cómputo especializado, por ejemplo, el equipo y los elementos que conforman una Celda de Manufactura (CM). El laboratorio cuenta con un Brazo Robótico (BR) KR 5-2 arc HW, una Banda Transportadora (BT) y demás equipo especializado, por tanto, es necesario poner en marcha una integración física, es decir, una Celda de Manufactura Flexible Inteligente (CMFI), esto con el fin de estudiar, aprender, resolver y optimizar los problemas que se vayan presentando dentro de la Industria 4.0 desde un enfoque académico, ingenieril y científico.

1.2. Objetivos

El objetivo principal de este trabajo de tesis de maestría, es diseñar e implementar un Sistema de Agarre de Piezas de Manufactura (SAPM), específicamente, un sistema autónomo basado en algoritmo de visión artificial entre un BR y una BT, creando el entorno necesario para sujetar al vuelo diferentes tipos de objetos de geometría simple de una banda transportadora con velocidad y dirección variables, utilizando sistemas embebidos y protocolos del ámbito del IoT, esto con el fin de en un futuro próximo, integrar este sistema como un elemento dentro de una CMFI en el LEAI 4.0 del IIMAS.

Los objetivos específicos, son:

- Diseño e implementación de un sistema de control de velocidad y dirección de la banda transportadora
- Diseño e implementación de un sistema de comunicación dentro de la CMFI
- Diseño e implementación de un Sistema de Visión Artificial (SVA) basado en flujo óptico
- Integrar el SAPM en un simulador
- Migrar el SAPM del ambiente virtual al físico utilizando la BT y el BR del LEAI 4.0

1.3. Planteamiento del problema

Para este trabajo, el objetivo es tomar al vuelo objetos de geometría simple desde una BT utilizando un BR, todo esto con el apoyo de tecnologías actuales del ámbito del IoT, sin embargo, el principal reto a superar es la automatización del sistema, ya que será necesario aprender e implementar métodos estadísticos y computacionales para que el sistema por sí mismo sea capaz de decidir cuándo, cómo y dónde tomar con el BR una pieza de manufactura que viaje sobre la BT con el menor error posible.

Con esto me refiero a que no solo bastará con programar los algoritmos de reconocimiento de patrones y/o formas, sino que se debe realizar una caracterización de ellos, modelar matemáticamente su funcionamiento y realizar pruebas, además de implementar la electrónica y protocolos necesarios para llevar a buen término este trabajo.

1.4. Metodología y requerimientos

A continuación, presento una metodología y los requerimientos para lograr los objetivos de este trabajo de tesis:

1. *Revisión del Estado del Arte.* Realizar una investigación documental que permite el estudio del conocimiento acumulado dentro del área de la robótica industrial.

2. *Planteamiento del problema y familiarización con las herramientas de trabajo e infraestructura del LEAI 4.0.* Plantear el problema, conjuntamente se aprenderá a identificar y operar el equipo que se tiene en el LEAI 4.0, ubicado en la ala sur del tercer piso del IIMAS. Este trabajo tiene como requerimiento principal utilizar la BT y el BR con el que este cuenta.

    - Brazo Robótico Kuka KR 5-2 arc HW. Realizar seminarios con estudiantes y profesores del Laboratorio de Robótica para enseñar y aprender, según sea

el caso, el funcionamiento del brazo robótico, conocer sus usos, limitantes, características y ventajas.

- Banda Transportadora. Realizar seminarios con profesores que conocen el funcionamiento de la banda transportadora, para aprender sus funciones y características, esto con el fin de actualizarla con un sistema de control de velocidad y dirección y otras cosas según lo que parezca conveniente.
- Cámara Industrial (CI) y SVA. Realizar seminarios con estudiantes más avanzados sobre los sistemas de visión computacional que han implementado dentro del laboratorio, esto con el objetivo de conocer los equipos de cámaras industriales que han utilizado y así tener una idea clara del equipo a utilizar en este trabajo.
- Equipo de cómputo y sistemas embebidos. Conocer las capacidades y limitaciones del equipo de cómputo y electrónico con el que cuenta el LEAI 4.0.

3. *Diseñar e implementar un sistema de comunicación dentro de la CMFI.* Este sistema administra la comunicación entre todos los los componentes principales de la CMFI. Los requerimientos de este sistema son:

   - Para cumplir con la palabra *flexible*, entonces la comunicación debe ser de manera inalámbrica
   - Por seguridad, la red debe ser local, sin conexión a Internet
   - Utilizar protocolos de comunicación y sistemas embebidos actuales dentro del ámbito del IoT

4. *Diseñar e implementar un sistema de control de velocidad a la BT.* Este sistema controla la velocidad y dirección del motor que mueve la BT con la que cuenta el

LEAI 4.0. Los requerimientos de este sistema son:

- Comunicación inalámbrica con los demás elementos de la CMFI
- Los datos de velocidad y sentido deberán ser transmitidos en todo momento mientras se encuentre en funcionamiento

5. *Diseñar e implementar un SVA.* Este sistema obtiene variables físicas de un objeto de geometría simple que atraviese su campo de visión y seguirá su movimiento. Los requerimientos son:

   - Utilizar el vector descriptivo BOF para obtener variables físicas de una pieza
   - Utilizar flujo óptico en el seguimiento de los objetos sobre la BT
   - Énfasis en el reconocimiento de los bordes de las piezas y no en detalles sobre la superficie de estos

6. *Cámara.* Estudio y selección de la cámara tipo RGB-D que mejor se adapte a los requerimientos del sistema de visión a implementar, por tanto se recomienda que la cámara cuente con las siguientes características:

   - Resolución mínima VGA
   - Latencia máxima 1 frame
   - Profundidad de al menos 0.5 m

7. *Integrar el SAPM dentro del simulador CoppeliaSim.* Llevar a cabo la implementación del sistema de agarre de piezas de geometría simple al vuelo por parte del BT en

función de la información procesada por el SVA y el movimiento actual de la banda dentro del simulador CoppeliaSim con la finalidad de observar su funcionamiento y detectar errores de ejecución.

8. *Migrar el SAPM del ambiente virtual al físico utilizando la BT y el BR del LEAI 4.0.* Llevar a cabo la implementación del sistema de agarre de piezas de geometría simple al vuelo por parte del BR en función de la información procesada por el SVA y el movimiento actual de la BT dentro de las instalaciones del LEAI 4.0

9. *Resultados finales.* Realizar las pruebas pertinentes del sistema de agarre para obtener los resultados finales de este trabajo; se realizarán los cambios o las optimizaciones pertinentes al sistema para un mejor funcionamiento y resultados.

A continuación presento el diagrama de la metodología descrita anteriormente junto con un cronograma para el periodo 2020-2021:

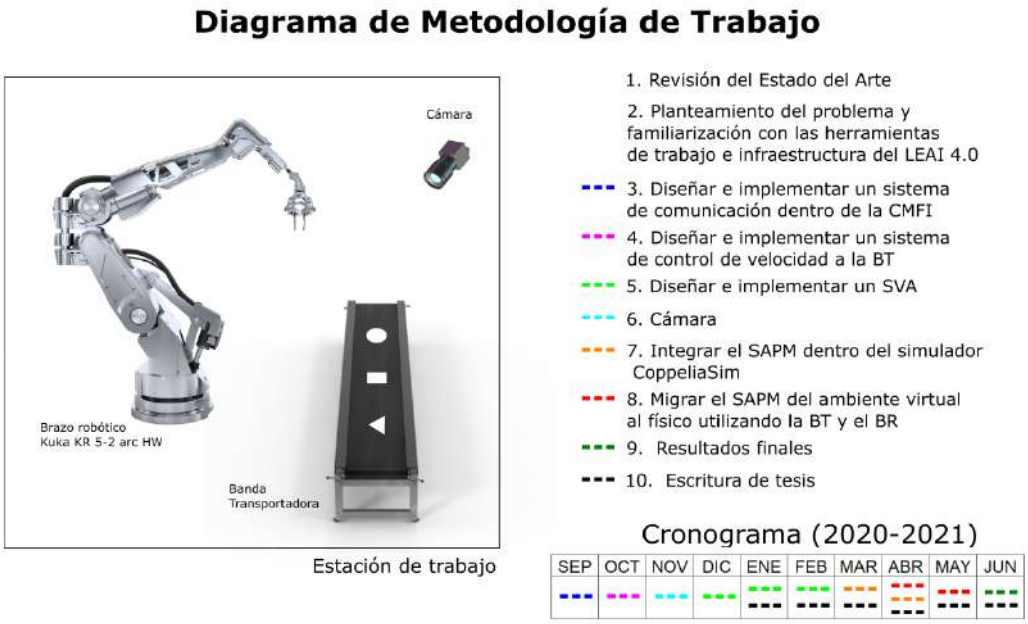


Figura 1.1: Cronograma de trabajo en función de la metodología propuesta.

2

Antecedentes

"Yo no pienso; investigo".

– Wilhelm Röntgen

2.1. Antecedentes

Desde tiempos remotos, el ser humano siempre ha tenido la necesidad de realizar diferentes tipos de procesos lo más rápido y eficiente posible, esto ha dado como resultado la invención de distintas máquinas, primero totalmente mecánicas las cuales ayudaban a aminorar el esfuerzo y el tiempo empleado por el ser humano; después surgieron máquinas electromecánicas, con estas, la energía eléctrica se convirtió en energía mecánica, resultando con el nacimiento de las primeras máquinas autómatas, capaces de realizar toda una tarea específica sin ayuda del ser humano. La aparición de una máquina en específico, llamada motor, desató toda una nueva era en el proceso de crear y armar objetos y/o servicios, esta era se le conoce como Revolución Industrial, donde no solo este proceso fue afectado, sino también el comercio y por ende la economía. Por ejemplo, en 1771 se inventó el primer molino de hilado totalmente automatizado, impulsado por energía hidráulica. En 1785 se desarrolló un molino de harina automático, convirtiéndose en el primer proceso industrial completamente automatizado. Particularmente, para el proceso

de manufactura, en 1913, Ford Motor Company introdujo una línea de montaje de producción de automóviles que se considera uno de los pioneros en la automatización de la industria manufacturera; ahora es impensable considerar una empresa de este estilo que no tenga líneas de producción semiautónomas o autónomas completamente (Corvo, 2019).

Ahora, ¿cómo solucionó el ser humano el problema de mover grandes cantidades de material de un lugar a otro sin usar gran esfuerzo físico? La respuesta es sencilla, usando lo que ahora se conoce como bandas transportadoras. Las bandas transportadoras se definen como una plataforma fija con un movimiento controlado de velocidad. Aparecieron a finales del siglo XVIII en la industria minera, para mover herramientas y carbón principalmente; no fue hasta 1913 donde la empresa Ford utilizó estas bandas en la industria para la producción de sus automóviles (TAPYC, 2019). A partir de ese momento se han realizado mejoras, no solo a los materiales de las bandas, sino incluyendo agentes externos, tales como cámaras de vídeo para usar sistemas de visión y herramienta especializada, como lo son los brazos robóticos, claramente todo esto depende de la tecnología disponible en su momento (véase Fig. 2.1).

Figura 2.1: Fotografía de una celda de manufactura: brazo robótico y bandas de transporte (ITAM, 2019).

Uno de los más grandes beneficios que se ha logrado dentro de la industria manufactu-

rera en bandas transportadoras fue el ahorro de tiempo, lo cual implica mayor producción, aquí la clave fue tener capacidad de agarre con alguna maquinaria objetos dentro de la banda transportadora mientras esta se encuentra en movimiento. Puede decirse que esta implementación está resuelta desde hace algunas décadas, sin embargo, nunca se ha dejado de optimizar el tiempo y los recursos tecnológicos para obtener una industria manufacturera más eficiente; un ejemplo de lo anterior nos los dan Khan et al. (2005) en su artículo llamado: *Automatic Inspection System Using Machine Vision*.

Así que, el problema a resolver fue como obtener dicha eficiencia, en primer lugar se necesitan bases de dónde partir, por ejemplo Lin et al. (1989) nos dicen que para trabajar en este tema, sobre el agarre de objetos en movimiento, se necesitan cuatro requerimientos mínimos: mínimo tiempo de operación, condiciones de agarre, restricciones de los actuadores y prevención de colisiones, deduzco que no es necesario explicar cada una, se entiende perfectamente cuál es el papel juega cada requerimiento.

Se ha intentado jugar con la cantidad de bandas transportadoras y maquinaria especial para aumentar la productividad, lo cual conlleva al término: *celda de manufactura*; según Liker (2004), una CM consiste en una disposición cercana de personas, máquinas y/o estaciones de trabajo en un cierto proceso, son creadas para facilitar el flujo de una pieza de un producto o servicio a través de varias operaciones, por ejemplo: soldadura, ensamblaje y empaque; son operadas a un ritmo determinado por las necesidades del cliente y con la menor cantidad de retrasos posibles, es decir un proceso eficiente; por ejemplo, Pardo-Castellote et al. (1995) estudiaron particularmente casos donde en una misma banda transportadora existen más de un brazo electromecánico con el objetivo de ensamblar piezas, en resumen, el primer brazo prende una primer pieza al vuelo y la guarda mientras el segundo brazo prende una segunda pieza y se la entrega al primer brazo para que este la ensamble con la pieza anterior (Véase Fig. 2.2). Claramente solo un brazo podría hacer este trabajo, sin embargo, el uso de dos brazos aminora el tiempo de producción. Otro objetivo donde se debe usar más de un brazo es cuando la masa de los objetos a mover es grande, del orden de centenas de kilos, es preferible usar dos o más

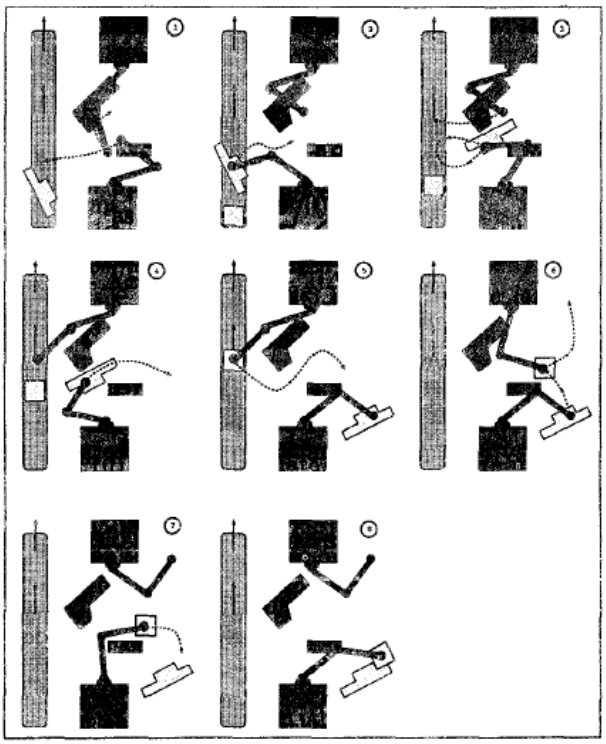


Figura 2.2: Diagrama físico de operación de una celda de manufactura utilizando dos brazos robóticos (Pardo-Castellote et al., 1995).

brazos para repartir la carga entre ellos (Mirrazavi Salehian et al., 2017).

Gracias a que los avances tecnológicos han hecho más accesibles los recursos, se ha desatado una nueva revolución: la Industria 4.0. Desde el 2010, se le ha dado este nombre a la digitalización de los procesos industriales por medio de la interacción de la inteligencia artificial con las máquinas y la optimización de recursos, es decir, la sinergia entre los sistemas embebidos y los recursos informáticos (Logicbus, 2019). Es por esto que, en este trabajo se aborda el término: Celda de Manufactura Flexible Inteligente (CMFI).

Es importante definir que un sistema embebido es un sistema computacional dedicado a realizar una tarea específica gracias a que su desarrollo es configurable, aumentando así su eficiencia computacional (Valvano, 2013). Gracias a esto, estos sistemas son especialmente útil para el proceso de imágenes, de hecho, lo que hace diferente de una Cámara Industrial a una cámara comercial, es esto, su sistema embebido, las cámaras industriales además

de capturar imágenes también las procesan con un fin específico, muchas de ellas deben procesar video en tiempo real.

Aquí es donde entran los Sistemas de Visión Artificial, estos se definen como un campo de la Inteligencia Artificial que, mediante la utilización de las técnicas adecuadas, permite la obtención, procesamiento y análisis de información obtenida a través de imágenes digitales, es un campo multidisciplinario ya que abarca la informática, óptica, mecánica e ingenierías. Ahora, específicamente la Visión Artificial Industrial se utiliza como un gran apoyo tecnológico para la selección, armado e inspección visual de piezas de manufactura utilizando nuevas tecnologías (Gobierno de España, 2012).

2.2. Trabajo Previo y Estado del Arte

A partir de la primer década del siglo actual, la gente empezó a enfocar sus esfuerzos en crear algoritmos para seguir optimizando la producción. ¿Por qué enfocarse en algoritmos? porque antes no se contaba con los recursos computacionales para efectuar algoritmos complejos de inteligencia artificial, por decir un ejemplo, es por esto que, al utilizar las nuevas tecnologías, como los sistemas de visión, los sistemas embebidos y aplicando conocimientos específicos, es posible obtener nuevas maneras para aumentar y optimizar la tasa de producción.

De hecho, es muy común observar que cada vez más las CM carecen de recurso humano, es decir, ya no es necesario contar con personas dentro de una celda de manufactura, estas ya son completamente autónomas y flexibles, de hecho, existen estudios donde se realizan cálculos para determinar el número mínimo de empleados que pueden estar en las fábricas donde se encuentren celdas de manufactura, por ejemplo Bozejko et al. (2018) nos dicen que basándose en un método de nombre *Branch and Bound*, se puede determinar el número mínimo de miembros capaces de operar celdas de manufactura, haciéndola más eficiente. Por ejemplo, se han hecho actualizaciones a celdas de manufactura que aún cuentan con personal humano, tal es el caso de Araujo et al. (2016), nos explican que se

desarrolló sobre la base de una solicitud industrial, mejoras a una celda de manufactura semiautomática dedicada a la fabricación de alfombrillas de suspensión de asientos de vehículos; se desarrollaron nuevos conceptos de alimentación y manipulación de alambre para permitir un mejor flujo de material a lo largo de la celda, esta nueva celda fue diseñada y construida con éxito, permitiendo obtener un sistema totalmente automatizado, lo que conduce a una mejor productividad y confiabilidad del proceso de fabricación.

Lo que cambia una Celda de Manufactura Flexible (CMF) de una fija, es que sus componentes son modulares, capaces de realizar distintas tareas o no participar en ellas dependiendo del objetivo final, generalmente estas partes están conectadas a un punto común, una computadora, la cual es la encargada de computar el trabajo y decidir qué parte hará cada módulo; en la actualidad dicha comunicación ya es inalámbrica. Aguirre et al. (2013) nos muestran en la Fig. 2.3 un diagrama de ejemplo de una CMF, en él se pueden observar distintos módulos como dos brazo robóticos, una CNC, y una banda transportadora.

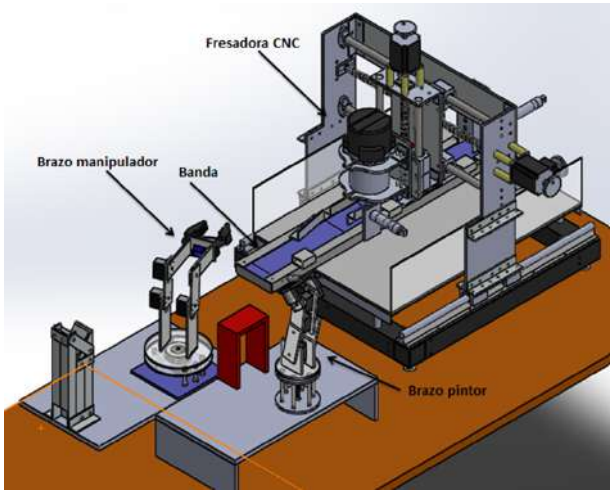


Figura 2.3: Diagrama celda de manufactura flexible (Aguirre et al., 2013).

Actualmente las celdas de manufactura son utilizadas en muchos campos dentro de la industria, por ejemplo (Irani, 1999):

- Industria de maquinaria y herramientas

- Equipos de construcción y agricultura
- Equipo médico
- Productos de defensa
- Automóviles y motores
- Piezas, partes y componentes
- Productos electrónicos
- Equipamiento químico
- Industrias de embalaje

Un ejemplo de celdas de manufactura nos lo da Barghash et al. (2017), ellos explican que este tipo de tecnología dentro de la rama farmacéutica es de suma importancia, ya que el resultado de implementarla conduce a una mayor productividad; utilizaron AHP (proceso analítico jerárquico) para la toma autónoma de decisiones multiobjetivo, esto con el fin de evaluar el mejor escenario entre los generados, modelando y evaluando cada uno de ellos, logrando excelentes resultados para cada una de las pruebas realizadas.

Se dice que la frase "divide y vencerás"se aplica en muchos conceptos, y por supuesto que dentro de la industria no se queda atrás, muchos algoritmos publicados *dividen* los objetivos para lograr una mayor efectividad; existen artículos donde, para prender un objeto al vuelo se divide en dos: una planificación visual instantánea del brazo robótico una vez que el objeto sobre la banda transportadora aparece y una replanificación visual del robot mientras el objeto se mueve (Cubero, 2007). Otra forma es aplicando algoritmos específicos para reconocer y estimar el objeto en movimiento en tiempo real, identificando características robustas aceleradas (SURF) y la búsqueda del vecino más cercano (FLANN), esto con el fin de ubicar centroides de ciertos objetos o figuras (Wang et al., 2015).

Por otro lado, la industria manufacturera debe darle agradecimientos a la ciencia e ingeniería por hacer posible que la electrónica sea más pequeña, rápida y económica, ya

que gracias a esto, los sensores utilizados en los sistemas de visión artificial han tenido una actualización importante, sobre todo en la última década. Por ejemplo existen nuevos sensores de imagen llamados Tiempo de Vuelo (TOF, por sus siglas en inglés), son sistema de lectura de profundidad basado en haces de luz infrarroja. Concretamente, haces de luz de 20 MHz para que la cámara no tenga problemas en distinguirlos de la propia luz ambiental, de forma que se puedan tomar las lecturas sin interferencias a la hora de poder modelar la escena en tres dimensiones. Con este sistema infrarrojo se lanzan varios haces hacia delante y la cámara se encarga de medir el tiempo que tardan entre su emisión y su recepción. El equivalente a disparar hacia una pared y comprobar el tiempo que tarda en impactar la bala. Existen artículos donde utilizan este tipo de sensor para el análisis de profundidad de diferentes objetos (Arif et al., 2010), por ejemplo, en la Fig. 2.4 se puede observar el proceso de analizar el volumen de un objeto usando sensores TOF.

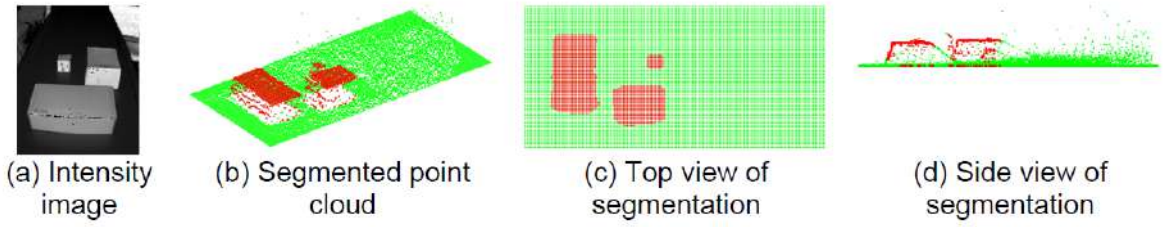


Figura 2.4: Proceso de la obtención de una nube de puntos de un objeto usando sensores TOF (Arif et al., 2010).

Otro sensor utilizado en la actualidad son los RGB-D, estos también pueden medir la profundidad de un objeto; dependiendo del tipo de cámara, algunas también se pueden considerar sensores ToF, SV o SL. Existen artículos donde utilizan estos sensores para obtener medidas con gran precisión de los objetos que pasen por una banda transportadora, por ejemplo Park et al. (2018), explican que ellos implementaron un algoritmo para medir las cajas de transporte de una manera eficiente y rápida sobre una banda transportadora en movimiento, llegaron a obtener mediciones menores al 4 % de error.

Falta mencionar un parámetro muy importante cuando se trata de estudiar y/o desarrollar celdas de manufactura: la velocidad de operación. Partiendo de este último artículo,

Park et al. (2018) mencionan que su algoritmo tiene los mejores resultados cuando los objetos escaneados viajan a través de la BT a una velocidad de 3.4 km/h (0.94 m/s).

Ahora bien, la velocidad máxima de operación es una combinación lineal de factores, tales como la velocidad y calidad de las bandas transportadoras, el número de brazos robóticos que operen sobre las BT, el sistema de visión artificial utilizado en cuestión, y por último, el hardware y software utilizados. Es por esto que dependiendo del tipo de trabajo y autores, la velocidad óptima y/o máxima de operación puede estar en función de algunos de los parámetros mencionados anteriormente. Dependiendo los autores y el tipo de trabajo, es el reporte de resultados, por ejemplo, Allaoui et al. (2017) nos dicen que su implementación de flujo óptico dentro de un FPGA tiene como resultado una velocidad de escaneo de 3.74 ns/pixel y de un trabajo similar al anterior, Gultekin and Saranli (2013) nos mencionan que de su implementación obtienen de resultados escaneos de hasta 257 fps. Otro ejemplo nos lo muestran Park et al. (2020), ellos desarrollaron un sistema de medición de paquetes que viajan sobre bandas transportadoras, enfocados a negocios tipo Amazon, su velocidad máxima de funcionamiento es de 0.5 m/s.

¿Qué tipo de problemas pueden ser solucionados con este tipo tecnología? Depende la necesidades. Para la selección de productos o piezas dañadas en una línea de producción, se han escrito artículos donde usando sistemas de visión, pueden seleccionar productos alimenticios, como galletas, basándose en su color se decide si la galleta tiene poca cocción o se encuentra quemado el producto (Nashat et al., 2011).

Una aplicación bastante interesante, es la de seleccionar la basura. Se sabe que la basura que generan las ciudades se concentran en puntos específicos donde alguna de estas se seleccionan para su correcto reciclado. ¿Cómo hacer este proceso más rápido?, sí, usando sistemas de visión y robótica de manufactura; es simple, la basura pasa sobre una banda, como si fueran piezas de producción, una cámara puede clasificar en tiempo real el tipo de basura que se desee, por ejemplo las botellas PET (véase la Fig. 2.5), un brazo pueda tomar dichos elementos y apartarlos de la banda (Zhihong et al., 2017). Es claro observar que la tecnología puede usarse para cosas que beneficien al planeta.

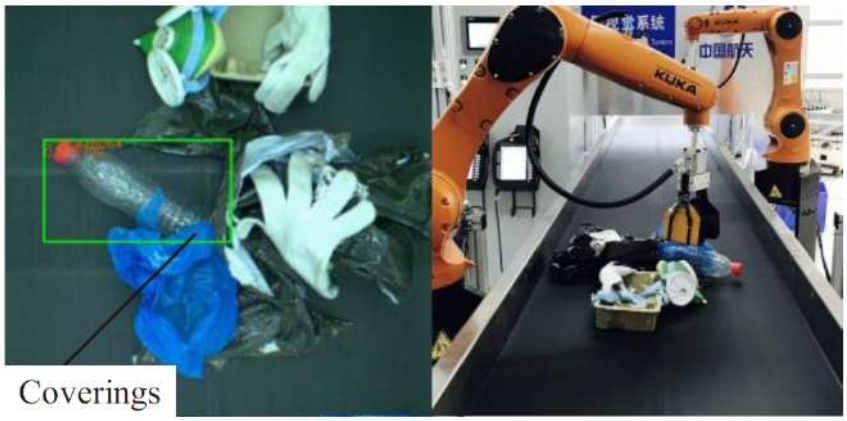


Figura 2.5: Sistemas de Visión aplicado a la selección de basura (Zhihong et al., 2017).

Los métodos que existen a la fecha para el correcto agarre de una pieza que pasa sobre una BT son variados, aunque llevan una misma metodología: detectar por medio de sensores ópticos la pieza en cuestión, calcular su posición y/o velocidad y darle al robot la posición física de sujeción de la pieza. Antes de utilizar cámaras, era común utilizar sensores infrarrojos, estos detectaban cuando un objeto atravesaba su campo de detección y en función de la información recaba los robots procedían a la correcta toma de la pieza, un ejemplo es el trabajo de Konukseven and Kaftanoglu (2000), nos explican que pudieron reducir el error de sujeción al utilizar sensores infrarrojos sobre la pinza de agarre con un sistema de control fuzzy. Por otro lado, un trabajo más reciente es el de Huang et al. (2020), en él nos explican a detalle que ellos realizaron un algoritmo para el agarre de piezas sobre una banda transportadora utilizando una cámara RGB-D y un brazo con 5 ejes de libertad; la cámara detecta al objeto, enseguida halla sus bordes y toma como puntos característicos las esquinas, el punto medio entre estas será el punto de agarre.

Al comparar las metodologías de los dos artículos mencionados en el párrafo anterior, puedo hacer hincapié en qué la tecnología ha sido un hito para la implementación de nuevas técnicas, y por supuesto, mejores resultados.

La electrónica que hace posible todos estos desarrollos descritos anteriormente, actualmente es implementada utilizando sistemas embebidos. Es muy común usar microcontroladores, FPGAs y demás hardware que haya sido concebido para realizar una tarea

específica. Por ejemplo, Chen et al. (2017) nos explican como se puede resolver el problema de los cierres inesperados de los controladores de robots en general, utilizando sistemas embebidos como lo son los FPGAs.

Actualmente, es común que los SVA sean implementados en hardware y arquitecturas específicas, otorgando considerables ventajas en tiempo de cómputo y por ende energía eléctrica utilizada. Lómas (2016) nos da un ejemplo en su tesis doctoral, él implementa redes neuronales artificiales para el reconocimiento y la clasificación de objetos, basándose en la Teoría de Resonancia Adaptativa, particularmente es una red FUZZY ART MAP. Por otro lado, un ejemplo de que no siempre se utilizan arquitecturas específicas la muestra la industria aeroespacial, Ruiz et al. (2020) nos detallan el desarrollo de un sistema de visión que detecta y clasifica elementos de fijación en un entorno real no controlado utilizando técnicas de aprendizaje basadas en redes neuronales implementadas con una tarjeta gráfica GTX1070Ti, utilizando Teras y TensoFlow con Python, logrando una precisión del 98.3 % en un tiempo de procesamiento de 0.8 ms por imagen procesada.

2.3. Cámaras RGB-D

Las cámaras RGB son un tipo específico de dispositivos de detección de profundidad que funcionan en asociación con una cámara RGB, es decir, se complementa la imagen convencional con información de profundidad (relacionada con la distancia al sensor) por píxel (Global, 2020).

Existen tres principales tipos de tecnologías que utilizan las cámaras RGB-D (Brading, 2020):

- Visión Estéreo (SV): Se combinan dos sensores 2D separadas una de otra, así, utilizando las distancias entre ambos sensores, se obtiene una distancia al objeto enfocado. Se dice que es un sensor pasivo.
- Luz Estructurada (SL): Aquí se proyecta un patrón infrarrojo sobre el objeto de interés, la deformación de este patrón da como resultado la distancia hacía el objeto

enfocado. Se dice que es un sensor activo.

- Tiempo de Vuelo (ToF): Este sensor inunda toda la escena con luz, principalmente infrarroja, y se toma el tiempo que el fotón tarda regresar al sensor, para así saber la distancia recorrida, que será la distancia hacía el objeto enfocado entre dos. Se dice que es un sensor activo.

Una comparación sencilla entre los tres tipos de tecnologías descritos anteriormente está descrito en la siguiente tabla:

		SL		
	SV	Patrón Fijo	Patrón Programable	**ToF**
Precisión de profundidad	mm - cm	mm - cm	μm - mm	mm - cm
Velocidad de escaneo	Medio	Rápido	Rápido - Medio	Rápido
Rango de distancia	Medio	Corto - Medio	Corto - Medio	Corto - Largo
Rendimiento baja luz	Débil	Bueno	Bueno	Bueno
Rendimiento exteriores	Bueno	Débil - Suficiente	Débil - Suficiente	Suficiente
Complejidad de software	Alto	Bajo - Medio	Medio - Alto	Bajo
Costo	Bajo	Medio	Medio - Alto	Medio

Tabla 2.1: Comparación de tecnologías utilizadas en cámaras RGB-D (Brading, 2020).

En la siguiente tabla se muestra una comparación de diferentes cámaras RGB-D que actualmente existen en el mercado:

	Nombre	**Tipo**	**Profundidad [m]**	**Resolución (3D/RGB) [PX]**	**Latencia**	**Precio Aprox** [1]
1	Microsoft Kinect	ToF	0.5 - 4.5	512 × 424/1920 × 1080	20 ms	70 USD
2	ASUS XtionPro Live	SL	0.8 - 3.5	640 × 480/1280 × 1024	∼1.5 frame	245 USD
3	Stereolabs ZED	SV	1.5 - 20	2208 × 1242	1 frame	349 USD
4	C.R. Multisense S7	SV	0.5 - ∞	2048 × 1088	1 frame	Registro
5	Intel RealSense D415	SV	0.3 - 10	1280 × 720/1920 × 1080	ND	130 USD
6	Intel RealSense D435	SV	0.105 - 10	1280 × 720/1920 × 1080	ND	224 USD
7	Orbbec Astra Mini	SL	0.6 - 5.0	640 × 480	1 frame	160 USD
8	roboception rc_visard	SV	0.5 - 3.0	640 × 480/1280 × 960	1 frame	Registro
9	duo3d DUO MC	SV	0.23 - 2.5	752 × 480	1 frame	Registro
10	Omega Arcure	SV	0.3 - 50	1280 × 1024	6600 LSB10/(Lux.s)	Registro
11	Basler ToF	ToF	0.5 - 0.8	640 × 480	1 frame	Registro
12	MYNT EYE	SV	0.5 - 18	752 × 480	1 frame	250 USD

Tabla 2.2: Algunas cámaras RGB-D actualmente en el mercado (ROS-Industrial, 2020).

Todas las cámaras mencionadas anteriormente tienen ya desarrollado un nodo para su uso en el sistema operativo ROS. Además, las cámaras de la 7 a la 12 son hechas

específicamente para tareas dentro de celdas de manufactura, ya desde un punto fijo o en movimiento.

2.4. IoT dentro de la Industria

El auge de la micro electrónica y la interconexión entre dispositivos cada vez más versátil utilizando nuevas tecnologías, como lo es la red 5G, trajo una consecuencia principal: La Cuarta Revolución Industrial o también llamada Industria 4.0.

Esta nueva revolución consiste en digitalizar los procesos industriales a través de interacciones tecnológicas y optimización de recursos; esto da como resultado una nueva manera de organizar los medios de producción, basándose en una más eficiente adaptación a las necesidades y a los requerimientos propios de los procesos de producción ((IPN, 2013)).

Según el IPN (2013), las principales características de la Industria 4.0 se pueden resumir en:

- *Conexión vertical en forma de red*: Los Sistemas Ciberfisicos están interconectados entre ellos y con trabajadores, directivos, desarrolladores, proveedores, clientes y hasta con el propio producto una vez vendido, gracias al Internet de las Cosas y al Internet de los Servicios (Cloud Computing).
- *Virtualización*: El mundo real de la planta es capturado por sensores, creando una imagen virtual de la misma, que está a su vez está conectada a Modelos de Simulación, Aplicaciones de Análisis Predictivos y Software para la ayuda de toma de decisiones. Todo ello ayudado por el Big Data.
- *Descentralización*: La toma de decisiones es ejecutada por los Sistemas Ciberfísicos, ayudada por Modelos Predictivos y Aplicaciones para la Toma de Decisiones.
- *Reacción en tiempo real*: La captura de la información, su procesado y las decisiones tomadas al respecto se realizan en tiempo real.

- *Orientación al cliente*: La arquitectura de la Industria 4.0 está diseñada para establecer un feedback directo entre el usuario, el producto y el diseñador del mismo.
- *Modularidad*: En un mercado tan cambiando, una Fábrica Inteligente debe adaptarse a los cambios que se producen en el mercado de forma rápida y eficiente, mientras que hacer un estudio del mercado y un cambio de producción puede llevar mínimo una semana las fabricas inteligentes estan preparadas para adaptarse al cambio de forma rápida y seguir las tendencias del mercado.
- *Analítica avanzada*: Sin duda es una de las partes más valiosas de esta industria 4.0, la capacidad para mejorar y optimizar los programas y procesos de producción es una parte vital dentro de cualquier empresa que quiera mantener un nivel alto de productividad y eficiencia. Los análisis avanzados para tomar decisiones sobre la planificación son de vital importancia en estas fábricas 4.0, consiguiendo una mayor agilidad en la cadena de producción y evitando de esta forma los cuellos de botella.

Con la lista anterior, es claro el papel que juega IoT dentro de la industria, es por esto que es importante describir su ecosistema. Microsoft (2021) nos explica que cada componente o nivel del ecosistema se define como:

- *Nivel de dispositivos*: Combinación de sensores, actuadores, hardware, software, conectividad y puertas de enlace que constituyen un dispositivo que se conecta e interactúa con una red.
- *Nivel de datos*: Datos que se recopilan, procesan, envían, almacenan, analizan, presentan y usan en contextos empresariales.
- *Nivel de negocio*: Funciones empresariales de las tecnologías de IoT, incluida la administración de la facturación y los marketplace de datos.
- *Nivel de usuario*: Personas que interactúan con dispositivos y tecnologías de IoT.

En este trabajo se abordan los primeros dos niveles.

2.4.1. Protocolos y Conectividad IoT

Al realizar la conectividad entre dos o más dispositivos, estos deben cumplir con cierto sistema de reglas, como lo debe ser la sintaxis y semántica, a esto se le conoce como protocolo de comunicación y tanto hardware como software deben cumplir con las reglas dictadas.

La arquitectura de la comunicación del Internet de las Cosas, puede observarse en la Fig. 2.6, en ella existen cinco capas (Sharma and Gondhi, 2018):

- *Física*: Medio por donde la información será transmitida, es decir, si los datos serán transmitidos alámbrica o inalámbricamente.
- *Enlace de datos*: Direcciones físicas para acceder a los dispositivos que componen la red.
- *Red*: Revisa que el envío y recepción de información sean llevados correctamente.
- *Transporte*: El modo de transporte de datos, es decir, por dónde y cómo viajará la información a través de la red.
- *Aplicaciones*: La posibilidad de acceder a los servicios de las capas anteriores, esta define los protocolos a usar para el intercambio de la información dentro de una red.

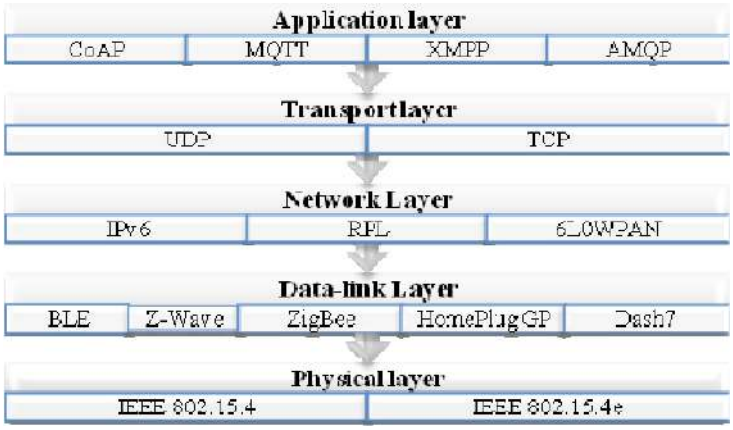


Figura 2.6: Arquitectura de comunicación del IoT (Sharma and Gondhi, 2018).

Protocolos IoT

Respecto a la capa de aplicaciones de la Fig. 2.6, muestro la tabla 2.3 en donde comparo los protocolos más comunes utilizados en IoT: HTTP[2], MQTT[3], AMQP[4] , CoAP[5] y WebSocket[6]; en ella se puede observar que si la comunicación debe ser sincrónica, entonces HTTP es la opción, de lo contrario, otros factores como el modo de conexión, el tipo de protocolo o el tamaño del mensaje pueden llegar a ser clave en la elección del protocolo a utilizar.

Nombre	HTTP	MQTT	AMQP	CoAP	WebSocket
Tipo	Solicitar/Responder	Publicar/Suscribir	Publicar/Suscribir	Publicar/Suscribir, Solicitar/Responder	Solicitar/Responder
Carga útil	Grande	Pequeña	Grande	Pequeña	Pequeña
Encabezado	Grande	2 bytes	8 bytes	4 bytes	2 bytes
Conexión	Uno a uno	Uno a uno, uno a muchos	Uno a uno uno a muchos	Uno a uno	Uno a uno
Cifrado de datos	SSL[7]/TLS[8]	SSL/TLS	SSL/TLS	DTLS[9]	SSL/TLS
Aplicaciones	D-N[10], N-N[11]	D-D[12], D-N, N-N	D-D, D-N, N-N	D-D	D-N, N-N
Protocolo transporte	TCP[13]	TCP	TCP	UDP[14]	TCP
QoS[15]	1 nivel	3 niveles	2 nivel	2 nivel	1 nivel
Comunicación	Sincrónico	Asincrónico	Asincrónico	Asincrónico	Asincrónico

Tabla 2.3: Comparativa protocolos de comunicación IoT.

[2] https://tools.ietf.org/html/rfc2616
[3] https://mqtt.org/
[4] https://www.amqp.org/
[5] https://coap.technology/
[6] https://tools.ietf.org/html/rfc6455
[7] Secure Sockets Layer
[8] Transport Layer Security
[9] Datagram TLS
[10] Dispositivo-Nube
[11] Nube-Nube
[12] Dispositivo-Dispositivo
[13] Transmission Control Protocol
[14] User Datagram Protocol
[15] Quality of Service

Dispositivos IoT

Existe una gran gama de dispositivos que diariamente se utilizan dentro del desarrollo de aplicaciones en el ámbito del Internet de las Cosas, estos son (Microsoft, 2021):

- Sensores
- Transductores
- Microcontroladores
- Microprocesadores
- Sistemas embebidos

Es bastante común que sean utilizados juntos más de uno de los elementos anteriores en un solo dispositivo listo para su uso, es decir, un sistema embebido. En internet, pueden encontrarse catálogos con una lista completa de este tipo de dispositivos, por ejemplo la que nos presenta Microsoft Azure en el siguiente enlace: https://devicecatalog.azure.com/.

¿Cómo elegir el dispositivo correcto? La elección depende de dos factores: requerimientos y costo. Según los requerimientos señalados en el capítulo anterior, en este trabajo deben utilizarse sistemas embebidos que cuenten con redes de corto alcance para ejecutar en ellos protocolos de comunicaciones inalámbricas, preferentemente con bajo consumo de operación, de un precio bajo y que en uno de ellos sea posible implementar algoritmos de visión computacional.

3

Marco Teórico

"Es por la lógica que demostramos pero por la intuición que descubrimos".

– Henri Poincaré

En este capítulo describo los conceptos utilizados para el desarrollo de este trabajo de tesis.

3.1. Comunicación dentro de la CMFI

La comunicación dentro de la CMFI es una parte fundamental de este escrito ya que los elementos de dicha celda deben estar comunicados entre sí en todo momento mientras esta se encuentre en funcionamiento.

Dada la tabla 2.3, he elegido utilizar el protocolo Message Queuing Telemetry Transport (MQTT) porque es un protocolo seguro, tiene una comunicación uno a muchos y uno a uno, se aplica de dispositivo a dispositivo y de dispositivo a la nube y tiene un nivel más de QoS en comparación a AMQP y CoAP; además, es de código abierto, su implementación es sencilla en sistemas embebidos y puede ser montado dentro de una red local sin necesidad de una conexión a internet.

3.1.1. Protocolo MQTT

MQTT es un protocolo de mensajería estándar de OASIS[1] para IoT. Está diseñado como un transporte de mensajería de publicación/suscripción a través de TCP/IP extremadamente liviano que es ideal para conectar dispositivos remotos con una huella de código pequeña y un ancho de banda de red mínimo. Hoy en día, MQTT se utiliza en una amplia variedad de industrias, como la automotriz, la fabricación, las telecomunicaciones, el petróleo y el gas, etc (MQTT, 2021).

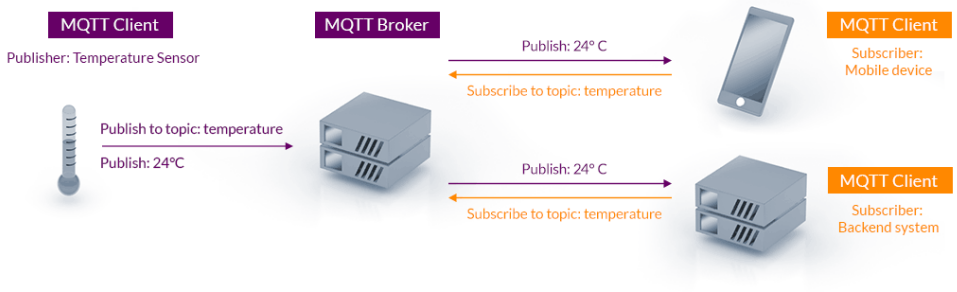


Figura 3.1: Arquitectura de publicación/suscripción del protocolo MQTT (MQTT, 2021).

La arquitectura de publicación/suscripción se presenta en la Fig. 3.1, en ella se puede observar un servidor o *broker* el cual se encarga de habilitar el canal de comunicación, existen diversos brókers en la web tanto libres como de pago.

La información se concentra en nodos llamados temas, tópicos o *topics*, los clientes publican (envían información) o se suscriben (leen información) en estos, los cuales se representan de la siguiente forma:

- tema/subtema/subsubtema/...

por ejemplo, si lo que se busca es monitorear la temperatura y la humedad de la cocina dentro de un hotel, entonces los tópicos podrían ser de la siguiente manera:

[1]OASIS Open, es uno de los organismos de normalización sin fines de lucro más respetados del mundo, ofrece proyectos, incluidos proyectos de código abierto, para promover proyectos de ciberseguridad, blockchain, IoT, gestión de emergencias, computación en la nube, intercambio de datos legales y mucho más. https://www.oasis-open.org/org/

- Hotel/Monitoreo/Cocina/Temperatura
- Hotel/Monitoreo/Cocina/Humedad

3.2. Algoritmos de Visión Computacional

Dentro de la automatización industrial, es indispensable utilizar algoritmos dentro del ámbito de Visión Computacional con el fin de obtener numéricamente características físicas de una pieza, objeto o figura en movimiento que atraviese el campo de visión de una cámara.

Las características físicas mínimas necesarias son:

- Tamaño
- Orientación
- Forma
- Posición
- Velocidad

Es por esto que, para este trabajo de tesis utilizo dos algoritmos: BOF y Flujo Óptico por el método de Lucas-Kanade.

3.2.1. Vector Descriptivo BOF

Boundary Object Function (BOF) es un algoritmo descriptor único que calcula las distancias euclidianas entre los puntos frontera de una figura y su centroide (Peña-Cabrera, 2005); se habla de una figura en tres dimensiones proyectada a un plano en dos dimensiones, en palabras mas sosas, es una fotografía de la pieza tomada desde su cenit. Este algoritmo da como resultado un vector con las distancias calculadas, donde su tamaño estará en función del número de distancias calculadas.

Las ventajas de este algoritmo son:

- Invariante en escala
- Invariante de rotación
- Invariante en traslación
- Se pueden obtener otras características a través de él

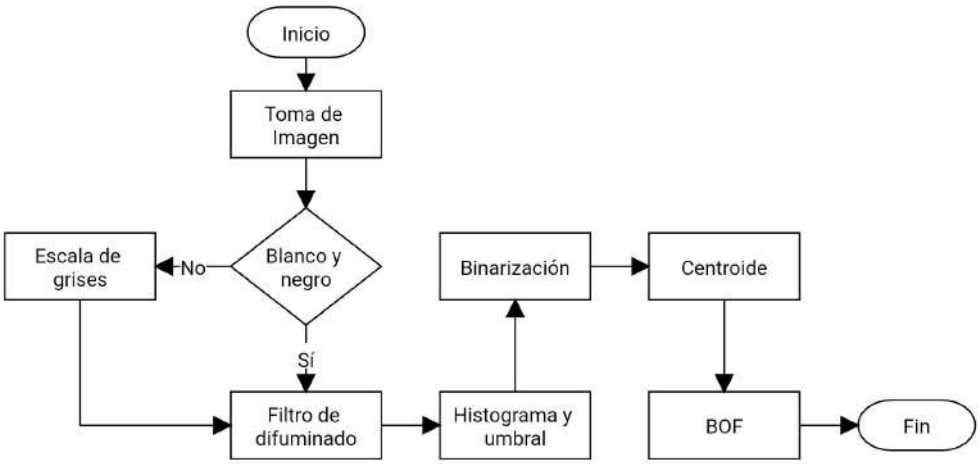


Figura 3.2: Diagrama de flujo del algoritmo BOF.

El algoritmo del descriptor BOF (véase Fig. 3.2), lo describo a continuación:

Toma de imagen de pieza

Se obtiene una fotografía de la pieza desde su cenit preferentemente en escala de grises.

Escala de grises

Si la fotografía fue tomada a color, entonces se convierte a escala de grises, para esto, se obtiene una media de la luminancia de la imagen en cada componente RGB para cada píxel. La expresión matemática que describe las palabras anteriores (OpenCV, 2021a), es la siguiente:

$$I = 0.3R + 0.59G + 0.11B \tag{3.1}$$

en esta ecuación puede observarse que los factores multiplicativos no son los mismos para cada componente o canal de luz, esto es consecuencia de la sensibilidad del ojo humano a las frecuencias del espectro cercanas al rojo, verde y azul; por ejemplo, para un píxel con valor RGB de (144, 255, 50) el valor en escalara de grises queda como 199.

Difuminado

- *Filtros.* Dentro del área de Proceso Digital de Imágenes, existe una gran cantidad de técnicas que, a partir de una fotografía o imagen, se puede obtener otra con características similares, resaltadas o eliminadas según sea el objetivo. Giménez Palomares et al. (2016) explican que existen una serie de diversos filtros básicos como lo son: filtro pasa-bajas, pasa-altas, detección de bordes, de direcciones y demás.

Figura 3.3: Ejemplos de filtros (Giménez Palomares et al., 2016).

- *Convolución.* Para implementar un filtro, este debe estar en forma matricial, llamados núcleos o kernels. El tamaño de estas matrices generalmente son pequeños, del orden de 3×3 o de 5×5. Por ejemplo, para realizar un filtro de desenfoque (pasa-bajas) se utiliza un kernel como el que se muestra en la Ec. 3.2

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3.2}$$

La forma de operar estos núcleos con una imagen, se llama convolución. Se denomina convolución a una función, que de forma lineal y continua, transforma una señal de

entrada en una nueva señal de salida. La función de convolución se expresa por el símbolo $*$.

En un sistema unidimensional, se dice que $g(x)$ convoluciona $f(x)$ cuando:

$$f(x) * g(x) = \int_{-\infty}^{+\infty} f\left(x'\right) g\left(\left(x - x'\right) dx' \tag{3.3}$$

donde x' es una variable de integración.

El resultado de $g(x)$ depende únicamente del valor de $f(x)$ en el punto x, pero no de la posición de x. Es la propiedad que se denomina invariante respecto la posición y es condición necesaria en la definición de las integrales de convolución. En el caso de una función continua, bidimensional, como es el caso de una imagen monocroma, la convolución de $f(x, y)$ por $g(x, y)$ será:

$$f * g = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f\left(x', y'\right) g\left(x - x', y - y'\right) dx' dy' \tag{3.4}$$

$g(x, y)$ debe cumplir el requisito de no variar según la posición x e y.

Lo anterior cumple para sistemas continuos, sin embargo, las imágenes son sistemas discretos, En un sistema discreto, como el de las imágenes digitalizadas, la convolución de la función $f(x, y)$ por $g(x, y)$, en la que $g(x, y)$ es una matriz de M filas por N columnas, es:

$$I(x, y) = f(x, y) * g(x, y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m, n) g(x - m, y - n) \tag{3.5}$$

donde $x = 0, 1, 2, ..., M$ y $y = 0, 1, 2, ..., N$ (Gonzalez and Woods, 2018).

Para nuestro caso, $f(x, y)$ es la imagen de entrada, $g(x, y)$ el núcleo del filtro a operar y $I(x, y)$ la imagen resultante. Un ejemplo de implementar una convolución de una imagen con un núcleo de la forma vista en la Ec. 3.2, se puede observar en la Fig. 3.4.

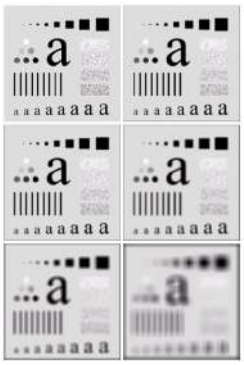


Figura 3.4: Ejemplos de desenfoque (Gonzalez and Woods, 2018).

Histograma

Para los siguientes pasos, es imperante calcular un umbral, con el propósito de que a partir de este número sea posible obtener una binarización de la imagen resultante al aplicar un filtro de desenfoque. Para esto es importante extraer el histograma de la imagen en primer lugar.

- *Histograma.* El histograma de una imagen consiste en una gráfica donde se muestra el número de píxeles que contienen un cierto nivel de gris, es decir, la frecuencia de la intensidad. No solo aplica a imágenes en blanco y negro, sino también a cada canal en imágenes a color (Gonzalez and Woods, 2018). A través de esta gráfica es posible obtener información sobre exposición, contraste, umbrales, y demás (véase Fig. 3.5).

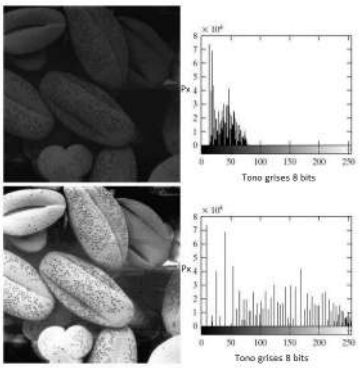


Figura 3.5: Ejemplos de histograma (Gonzalez and Woods, 2018).

El pseudocódigo para obtener el histograma nos lo presenta Kaeli (2015):

```
int histogram[]
main( ) {
  for (each input value) {
    histogram[value]++
  }
}
```

- *Umbral*. Existen diversos algoritmos para obtener un número óptimo donde se separan los dos máximos de un histograma bimodal. Entre los métodos más utilizados está el método de Otsu (Otsu, 1979). Básicamente, calcula el valor umbral de forma que la varianza dentro de cada segmento sea lo más pequeña posible, pero al mismo tiempo la varianza sea lo más alta posible entre las clases o los máximos diferentes.

Ya obtenido el histograma de la figura en cuestión y calculado el valor umbral, entonces es posible realizar una segmentación[2] de dicha figura. A partir de este número se decidirán qué píxeles se dibujarán como blanco o negro. Un ejemplo gráfico nos lo muestra Atienza Vanacloig (2011) (ver Fig. 3.6), en él puede observarse la segmentación (c)) de una herramienta (a)) respecto al fondo, nótese en b) el histograma bimodal.

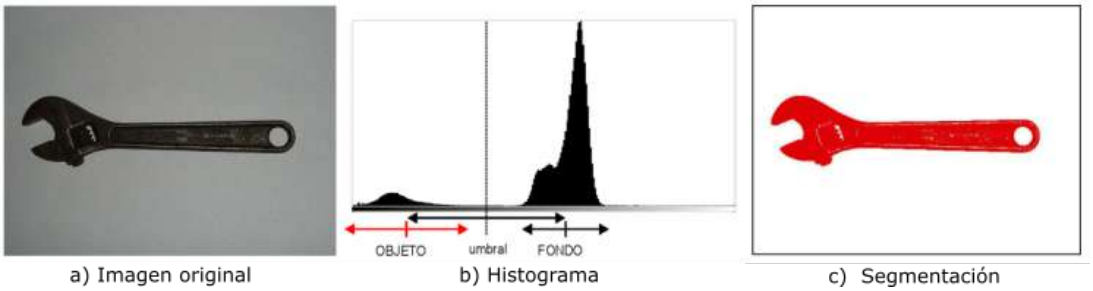


Figura 3.6: Ejemplos de histograma bimodal para segmentación (Atienza Vanacloig, 2011).

[2]Una segmentación es dividir una imagen digital en grupos de píxeles con las mismas características.

Binarización

Al tipo de segmentación mostrada en la figura anterior se le conoce como Binarización. Una vez obtenido el valor umbral, se crea una nueva imagen donde todos los valores de la imagen de entrada que sean menores al valor umbral se pintan como negro y el resto de blanco, esto lo describe la Ec. 3.6:

$$I_{i,j} = \begin{cases} 0 \rightarrow I_{i,j} < \text{ umbral} \\ 1 \rightarrow I_{i,j} \geq \text{ umbral} \end{cases} \tag{3.6}$$

Centroide

Para obtener el centroide de una figura de geometría simple en una imagen binaria, este se calcula con los siguientes parámetros de nombre momentos:

- M_{00}: Todos los píxeles blancos de la imagen (área de imagen)
- M_{10}: Los píxeles blancos sobre el eje X
- M_{01}: Los píxeles blancos sobre el eje Y

Al final, el centroide con coordenadas (x_c, y_c) se obtiene a través de las siguientes expresiones matemáticas:

$$x_c = \frac{M_{10}}{M_{00}} = \frac{\sum_x \sum_y xI(x,y)}{\sum_x \sum_y I(x,y)} \tag{3.7}$$

$$y_c = \frac{M_{01}}{M_{00}} = \frac{\sum_x \sum_y yI(x,y)}{\sum_x \sum_y I(x,y)} \tag{3.8}$$

BOF

Por último, ya que se cuenta con la posición del centroide de un objeto (x_c, y_c), se calculan las distancias euclidianas (Ec. 3.9) entre este y puntos del borde de dicho objeto $(x_b,\ y_b)$ (ver Fig. 3.7); para esto, utilizando coordenadas polares (r, θ), se inspecciona la imagen desde el centroide cada cierto grado θ hasta que cambie de color, con esto se

obtiene r, el cual representa la distancia. Al final el vector resultante es normalizado para cumplir con la invarianza al escalamiento.

$$d = \sqrt{(x_b - x_c)^2 + (y_b - y_c)^2} \tag{3.9}$$

Presento el siguiente pseudo código:

```
vector_BOF = []
BOF(){
  partiendo del centroide
  para cada grado
    contar cada pixel hasta el cambio de color
    agregar total de pixeles a vector_BOF
}
```

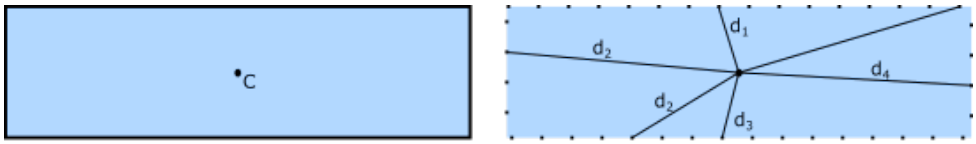


Figura 3.7: Diagrama BOF.

Por tanto, el vector resultado debería ser de la forma:

$$\text{BOF} = [d_1, d_2, d_3, ..., d_n] \tag{3.10}$$

Suele presentarse gráficamente este resultado, por ejemplo, la gráfica del vector BOF para el rectángulo de la Fig. 3.7, es de la forma:

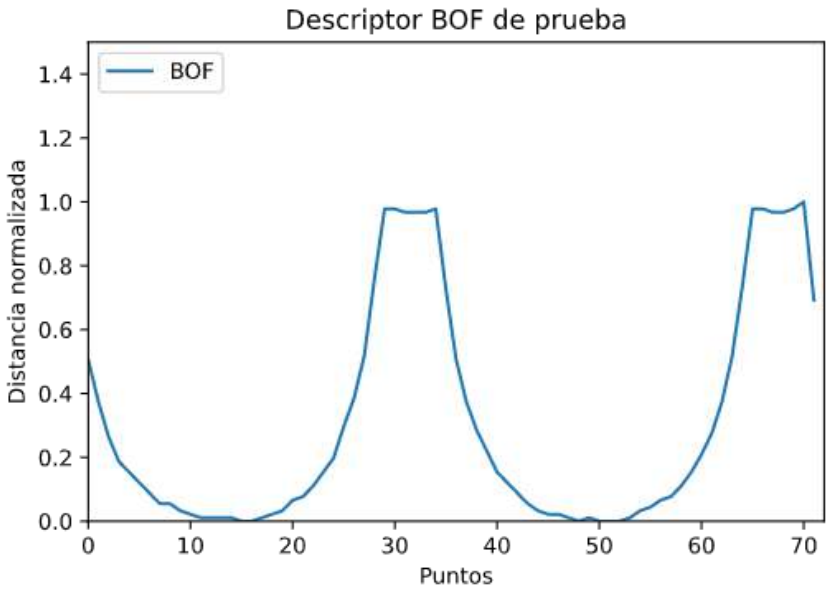


Figura 3.8: Gráfica del vector BOF de la 3.7.

Características extras

A partir del vector BOF se pueden obtener otras características, como lo son:

- *Orientación*. Para obtener la orientación respecto a un eje, simplemente se busca la posición del valor mínimo del vector característico y a este se le multiplica por la frecuencia de cálculo de distancias (cada cuantos grados se obtiene r). Según convenga, se debe restar 180° al valor obtenido anteriormente. La expresión matemática que describe lo anterior queda como:

$$\text{Orientación} = \text{mín}\,(\text{BOF}) * \nu \tag{3.11}$$

- *Longitud*. Este valor es más simple, ya que solo se toma ahora el valor máximo del vector característico (Ec. 3.12). Este valor me asegura que longitud más grande de una pieza.

$$\text{Longitud} = 2\,\text{máx}\,(\text{BOF}) \tag{3.12}$$

Estos dos anteriores parámetros sirven para enviar al robot las indicaciones de cómo sujetar la pieza.

3.2.2. Flujo Óptico

El flujo óptico es un patrón de movimiento aparente de los objetos posible de calcular mediante dos fotogramas consecutivos, este movimiento puede ser propio del objeto o de la cámara. En la Fig. 3.9 puede verse un objeto esférico que se mueve en cinco fotogramas consecutivos; este algoritmo es muy útil cuando se trata de estabilizar y comprimir vídeos, y además, seguir el movimiento de una pieza que se mueve sobre una BT.

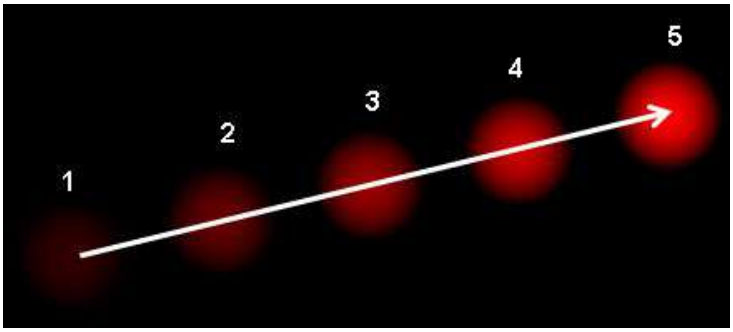


Figura 3.9: Ejemplo de flujo óptico (OpenCV, 2021b).

El flujo óptico funciona con varios supuestos:

- Las intensidades de píxeles de un objeto no cambian entre fotogramas consecutivos
- Los píxeles vecinos tienen un movimiento similar.

Entonces, considerando un píxel $I(x, y, t)$ del primer cuadro de un vídeo y este se mueve una distancia (dx, dy) al siguiente cuadro dt, dados los supuestos anteriores, se tiene:

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{3.13}$$

de donde se obtiene:

$$f_x u + f_y v + f_t = 0 \tag{3.14}$$

donde:

$$\begin{aligned} f_x = \frac{\partial f}{\partial x}; f_y = \frac{\partial f}{\partial y} \\ u = \frac{dx}{dt}; v = \frac{dy}{dt} \end{aligned} \tag{3.15}$$

Lucas-Kanade

La ecuación 3.14 es la llamada ecuación de flujo óptico, se puede observar el gradiente, sin embargo u y v son desconocidos. Una manera de encontrar este valor la da el método de Lucas-Kanade (OpenCV, 2021b), (Lucas and Kanade, 1981).

Se toma un parche de 3×3 alrededor de punto de interés, así la ecuación 3.14 pasa a tener 9 ecuaciones y dos incógnitas (véase la Fig. 3.10). Utilizando mínimos cuadrados se llega a:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i f_{x_i}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i f_{y_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_{x_i} f_{t_i} \\ -\sum_i f_{y_i} f_{t_i} \end{bmatrix} \tag{3.16}$$

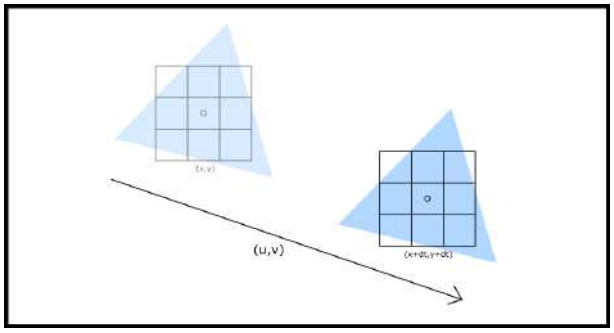


Figura 3.10: Diagrama del flujo óptico.

3.3. Análisis de movimiento

En este trabajo se plantea que los objetos que viajen sobre la BT, cuenten con un Movimiento Rectilíneo Uniforme (MRU). A dicho movimiento se le debe aplicar una extrapolación con el fin de predecir su posición tiempo después de haberle sido aplicados los algoritmos de visión descritos en la sección anterior. Para aplicar esta técnica, se debe hacer un análisis estadístico de las posiciones obtenidas a través del flujo óptico, para este caso, mínimos cuadrados.

3.3.1. Movimiento Rectilíneo Uniforme

En primer lugar, Alonso and Finn (1970) nos explican que el MRU es cuando un cuerpo sigue una trayectoria recta en su movimiento constante.

En la Ec. 3.17 se observa la descripción matemática que describe el cambio de posición respecto al tiempo transcurrido en dicho cambio.

$$v = \frac{dx}{dt} \tag{3.17}$$

Dado que lo que se necesita es obtener una posición en el espacio, entonces se debe integrar la Ec. 3.17, quedando:

$$\int_{x_0}^{x} dx = \int_{t_0}^{t} v dt \tag{3.18}$$

donde x_0 es la posición inicial en $t = 0$.

El lado izquierdo de la Ec. 3.18 queda como $x - x_0$, luego entonces, se obtiene la posición x en cualquier tiempo t yendo a cierta velocidad v:

$$x(t) = x_0 + v(t - t_0) \tag{3.19}$$

3.3.2. Mínimos Cuadrados

Dado que se obtendrán una serie de puntos que describen el movimiento lineal del objeto en cuestión, se puede aplicar el método de mínimos cuadrados para encontrar una función que describa el movimiento mencionado, para así, calcular una posición futura del objeto sobre la BT.

D. C. Baird (1991) explica que este método estadístico ajusta una función que mejor describe una nube de puntos (véase Fig. 3.11), puede ser una función lineal, cuadrática, exponencial o cualquier otra que se requiera. El problema es encontrar la función

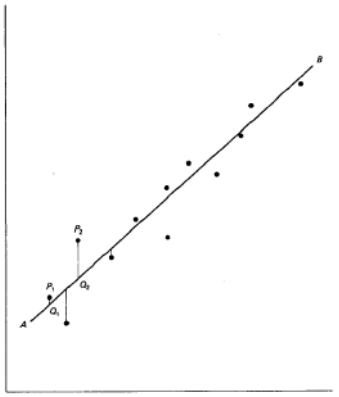


Figura 3.11: Ajuste de una línea recta a un conjunto de puntos por el método de mínimos cuadrados (Baird, 1991).

que mejor lo haga, y para esto se miden las diferencias cuadradas entre los puntos y la función eligiendo las más pequeñas, de aquí el nombre.

Para una función lineal, como en este caso (véase Ec. 3.19), se debe ajustar una función del tipo:

$$y = mx + b \tag{3.20}$$

de aplicar el método de mínimos cuadrados se obtienen una pendiente (m) y una ordenada al origen (b) que mejor describen el conjunto de puntos (x, y) tal como se aprecian a continuación:

$$m = \frac{N\sum(x_i y_i) - \sum x_i \sum y_i}{N\sum x_i^2 - \left(\sum x_i\right)^2} \tag{3.21}$$

$$b = \frac{\sum x_i^2 \sum y_i - \sum x_i \Sigma\,(x_i y_i)}{N\sum x_i^2 - \left(\sum x_i\right)^2} \tag{3.22}$$

donde N es el número de pares de puntos; y por último, es posible calcular sus incertidumbres gracias a las siguientes ecuaciones:

$$\Delta m = \sqrt{\frac{N}{N\sum x^2 - (\Sigma x)^2} \cdot \frac{\phi^2(b,m)}{N-2}} \tag{3.23}$$

$$\Delta b = \sqrt{\frac{\sum x^2}{N\sum x^2 - (\Sigma x)^2} \cdot \frac{\phi^2(b,m)}{N-2}} \tag{3.24}$$

donde $\phi^2(b,m)$ es la diferencia de los puntos x_i y y_i con la recta.

Resumiendo, en este trabajo el procedimiento será encontrar las m y b que mejor describan la nube de puntos de las posiciones que se obtengan a partir del seguimiento por flujo óptico, esto con la finalidad de calcular por el método de mínimos cuadrados la velocidad actual de la pieza de manufactura, y posteriormente, calcular una posición futura a un determinado tiempo t para sujetar la pieza al vuelo con el BR. En el capítulo siguiente ejemplifico lo descrito anteriormente.

3.4. Simulador CoppeliaSim

El simulador de robot CoppeliaSim (antes V-REP), con entorno de desarrollo integrado, se basa en una arquitectura de control distribuido: cada objeto o modelo se puede controlar individualmente mediante un script integrado, un plugin, un nodo de ROS o BlueZero, un cliente API remoto o una solución personalizada. Esto hace que CoppeliaSim sea muy versátil e ideal para aplicaciones de múltiples robots. Los controladores se pueden escribir en C/C++, Python, Java, Matlab, Octave o por defecto, Lua.

CoppeliaSim se utiliza para el desarrollo rápido de algoritmos, simulaciones de automatización de fábricas, creación rápida de prototipos y verificación, educación relacionada con la robótica, monitoreo remoto, doble verificación de seguridad, como gemelo digital y mucho más (Coppelia-Robotics, 2021a).

Este simulador es de pago, sin embargo, es posible obtener una versión de estudiante, la cual cuenta con todas las características de la edición profesional presentando una marca de agua dentro del espacio de trabajo.

Se puede poner en marcha una CMF dentro de este simulador ya que cuenta con las herramientas y equipo necesarios como lo son:

- Brazos robóticos de marcas profesionales como Kuka o ABB
- Bandas de transporte de diferentes tamaños, capacidades y formas
- Diferentes tipos de sensores: de visión, proximidad y fuerza.
- Y diferente tipos de mobiliario útil, como paredes, mesas, mamparas, y la posibilidad de crear objetos de diferentes formas geométricas.

Figura 3.12: CoppeliaSim (Coppelia-Robotics, 2021a).

3.5. Brazos Robóticos

Un brazo robótico industrial es un dispositivo electromecánico autónomo, diseñado y construido con la intención de imitar y amplificar las funcionalidades del brazo humano. Están fabricados con el objetivo de realizar trabajos repetitivos durante largas jordanas de funcionamiento.

Las principales ventajas de utilizar este tipo de instrumentos son:

- Optimización de recursos y procesos
- Aumento en la producción
- Precisión en la realización de tareas repetitivas
- Objetos colaborativos entre sí

A nivel mundial existen diversas compañías dedicadas a comercializar este tipos de robots, por mencionar a algunas:

- KUKA (https://www.kuka.com/es-mx)
- ABB (https://www.abbrobotics.com.mx/)
- OMRON (https://automation.omron.com/es/mx/)
- KAWASAKI (https://robotics.kawasaki.com/)
- FANUC (https://www.fanuc.eu/es/es/robots)

3.5.1. Movimiento de Brazos Robóticos

Mover un BR depende de sus características físicas, tales como el número de grados de libertad, el tipo de actuadores que mueven las partes del brazo, y el método fisico-matemático hacerlo. Para tal propósito, el brazo se modela como una cadena articulada en modo de lazo abierto, donde un extremo se ancla a una base y el otro queda en libertad, por tanto, el objetivo es mover el extremo libre.

Es aquí donde una rama de la física entra en acción: la cinemática. Esta rama, aplicada a los brazos robóticos, trata con el estudio analítico de la geometría que describe el movimiento de estos en función de un marco de referencia, el cual es, por convención y practicidad, la misma base del robot. En la figura 3.13 muestro 5 diferentes tipos de BR según la geometría de su movimiento.

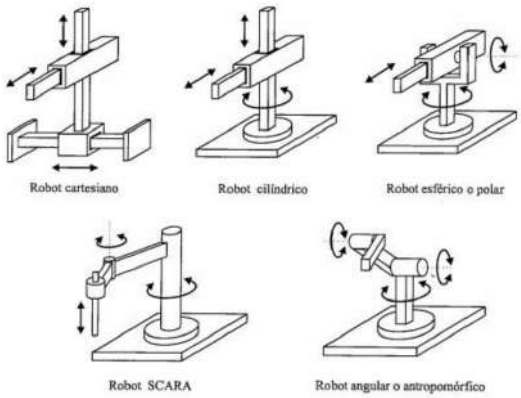


Figura 3.13: Tipos de BR según su geometría de movimiento (Granja, 2014).

Ahora, el problema es cómo mover el brazo robótico como si se tratara de mi mismo brazo, para esto se tienen dos soluciones: la cinemática directa y la cinemática inversa. La primera nos dice que para un BR determinado, se tiene un vector de ángulos de cada articulación y este vector nos dará la posición final de la punta del brazo; la segunda, tal como su nombre lo indica, el proceso se hace de forma inversa, es decir, a partir de las coordenadas de la posición física de la punta del brazo, se obtiene un vector con las posiciones de las articulaciones que cumplen con dicha coordenada (véase Fig. 3.14).

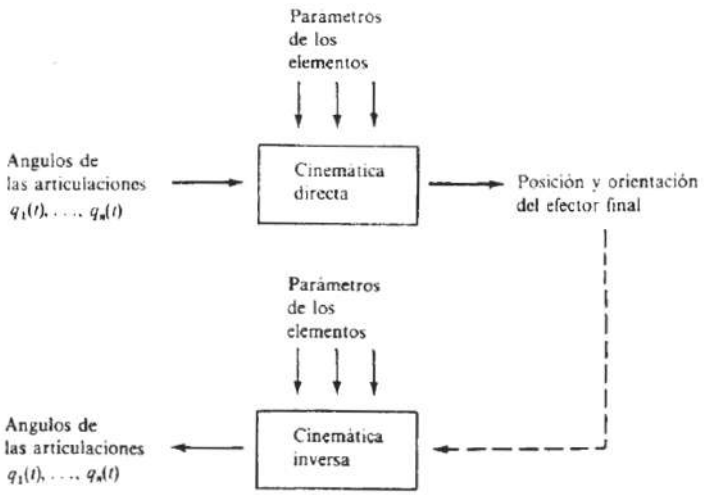


Figura 3.14: Diagrama de la cinemática directa e inversa (Fu et al., 1990).

- *Cinemática Directa.* La cinemática directa esencialmente es obtener una matriz de transformación que relacione el sistema de coordenadas del marco de referencia al del cuerpo (Fu et al., 1990). Se utilizan diversas matrices de rotación, por ejemplo las que muestro a continuación:

$$\mathbf{R}_{x,\alpha} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\operatorname{sen}\alpha \\ 0 & \operatorname{sen}\alpha & \cos\alpha \end{bmatrix} \tag{3.25}$$

$$\mathbf{R}_{y,\phi} = \begin{bmatrix} \cos\phi & 0 & \operatorname{sen}\phi \\ 0 & 1 & 0 \\ -\operatorname{sen}\phi & 0 & \cos\phi \end{bmatrix} \tag{3.26}$$

$$\mathbf{R}_{z,\theta} = \begin{bmatrix} \cos\theta & -\operatorname{sen}\theta & 0 \\ \operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.27}$$

  Lo importante de estas matrices es su interpretación geométrica, *grosso modo*, las rotaciones del eje Z nos dará el giro del brazo (específicamente de la unión en cuestión), del eje Y la elevación y del eje X la desviación.

- *Cinemática Inversa.* Para la tareas de pick and place, es más conveniente utilizar el método de cinemática inversa, por el hecho de que solo se conoce un punto, que es donde el efector (herramienta en la punta del brazo) actuará.

Existe una gran cantidad de métodos para obtener el movimiento de cada grado de libertad en función de las coordenadas del efector, como lo son el método geométrico o el método de ángulos de Euler, ambos son bastante precisos.

El entorno de trabajo con el robot Kuka, ya cuenta con los recursos de software necesarios para aplicar cualquiera de estos tipos de movimiento, por lo que no es necesario desarrollar la construcción matemática, aunado a esto, el simulador CoppeliaSim, cuenta también con soluciones para que la implementación del movimiento de un BR sea por cinemática directa o inversa (Coppelia-Robotics, 2021b).

3.5.2. Brazo robótico KUKA

El LEAI 4.0 cuenta con un brazo robótico de la marca Kuka que cuenta con seis grados de libertad, modelo KR 5-2 arc HW (sus características técnicas las enuncio en el apéndice 6), este equipo es controlado por su propio sistema operativo KUKA.SystemSoftware, en él es posible configurar el modo de uso y movimiento, además cuenta con la posibilidad de programar rutinas directamente en él.

Sistemas de Coordenadas

El BR Kuka cuanta con cuatro sistemas de coordenadas para trabajar con él: Robroot, Mundo, Base y Herramienta; el primero es un sistema de coordenadas cartesianas donde el origen se encuentra en la base del robot, el segundo es igual pero con el origen desplazado frente al brazo, el tercero el origen se encuentra en la plataforma de trabajo, y por último, en el cuarto el origen es el efector final del brazo (ver Fig. 3.15).

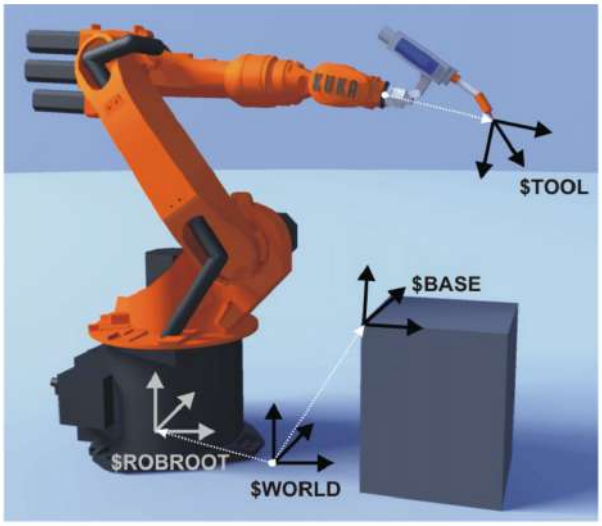


Figura 3.15: Sistemas coordenados del BR Kuka KR 5-2 arc HW (KUKA, 2015b).

Es sencillo intuir que dependiendo la aplicación es el sistema coordenado a utilizar, por ejemplo, si se necesita pulir una pieza o escribir en cierto material el sistema coordenado sería Herramienta, sin embargo, si se desea tomar y/o soltar piezas desde una superficie, cualquier sistema de los primeros sería idóneo, para este trabajo de tesis el elegí el sistema Mundo (KUKA, 2015b).

Calibración

Según el manual del robot, existe un proceso de calibración para los sistemas coordenados base y herramienta, en el primero se obtiene el área de trabajo de la base y el segundo el tamaño físico del efector final. Para este trabajo no es necesario realizar alguno de estos tipos de calibraciones, esto es derivado a que utilizo el modo Mundo para el movimiento del robot.

Tipos de movimiento

Los brazos robóticos Kuka cuentan con tres principales tipos de movimientos que pueden ser programados: Point-to-point (PTP), Linear Motion (LIN) y Circular Motion (CIRC). A continuación los describo:

- *PTP*: El robot se mueve a lo largo de la ruta más rápida, no necesariamente en línea recta pues no se puede predecir la trayectoria exacta. La velocidad de movimiento es programada en porcentaje de las velocidades articulares.

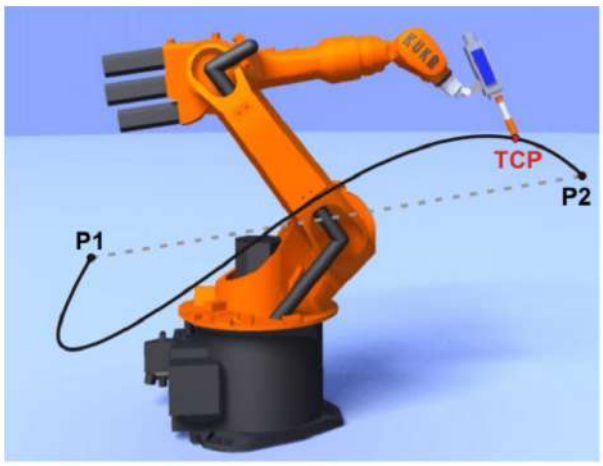


Figura 3.16: Movimiento del BR tipo PTP (KUKA, 2015b).

- *LIN*: El robot se desplaza de un punto inicial a uno final en línea recta a una velocidad constante.

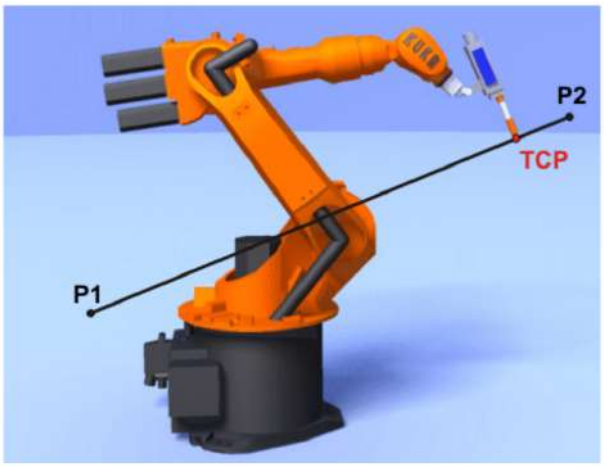


Figura 3.17: Movimiento del BR tipo LIN (KUKA, 2015b).

- *CIRC*: El robot se desplaza a velocidad constante a lo largo de una trayectoria circular. La trayectoria circular está definida por un punto inicial, un auxiliar y un

punto final.

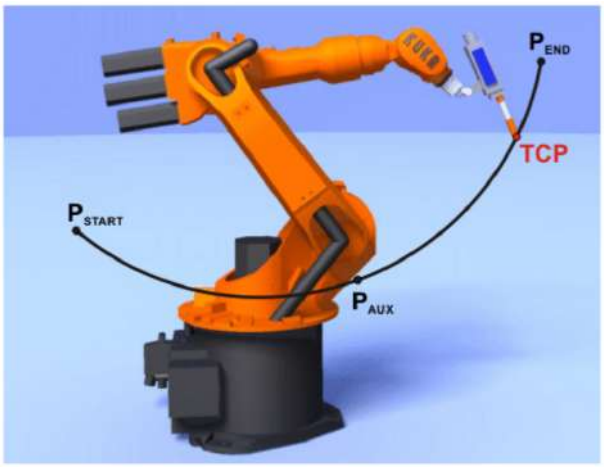


Figura 3.18: Movimiento del BR tipo CIRC (KUKA, 2015b).

4

Desarrollo

"En ciencia uno intenta decir a la gente, en una manera en que todos lo puedan entender, algo que nunca nadie supo antes. La poesía es exactamente lo contrario".

– Paul Dirac

Este capítulo tiene como objetivo describir la implementación para poner en marcha el SAPM. Este sistema será probado en una CMF dentro del simulador CoppeliaSim para su posterior migración a un sistema Brazo-Banda con la que cuenta el LEAI 4.0 del IIMAS. Todo esto con base en los requerimientos mostrados en el capítulo uno (véase Metodología y requerimientos).

4.1. Arquitectura General

La arquitectura general del sistema a implementar consta de tres capas: Dispositivos, Comunicación y Aplicaciones (véase la Fig. 4.1). En la primera capa se cuentan con tres dispositivos fundamentales, una mini computadora Raspberry Pi y dos módulos Node-MCU ESP32s; en el primero implemento el SVA junto con el servidor principal de la comunicación entre los dispositivos, el segundo es el encargado de recibir, transmitir y procesar la información desde-hacía la BT y el tercero lo utilizo para transmitir hacía el

efector final del robot la información de cuándo abrir y cerrar la pinza, principalmente. La comunicación es de manera inalámbrica (WiFi) a través del protocolo MQTT. Por último, la capa de aplicaciones representa a los tres componentes principales de este trabajo tesis: el BR, la BT y el SVA, los cuales describo más adelante.

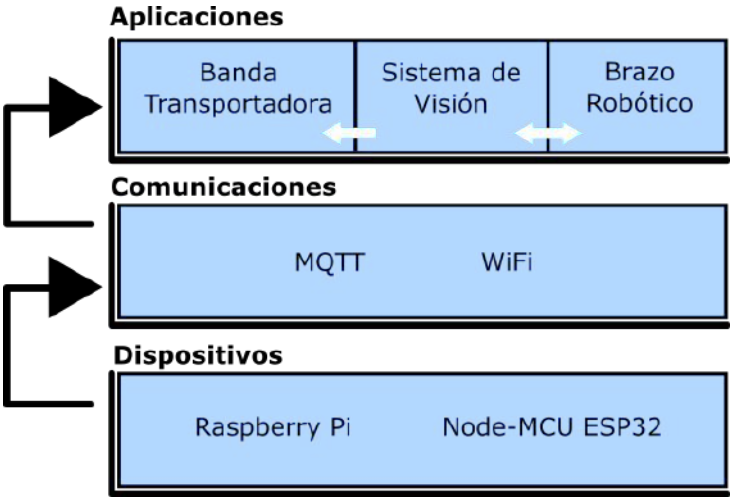


Figura 4.1: Arquitectura General de la CMFI.

4.2. Comunicación General

Uno de los objetivos a corto plazo del LEAI 4.0 es implementar una CMFI, dado que uno de los principales requerimientos es contar con una conexión inalámbrica entre los componentes del sistema, utilicé tecnologías recientes del ámbito IoT, como protocolos de comunicación y algunos sistemas embebidos que describo más adelante, esto con el fin de poner en marcha la comunicación interna entre los componentes de la celda.

4.2.1. Arquitectura de Comunicación General

El diseño de la arquitectura de comunicación general de la celda a implementar se muestra en el diagrama de bloques de la Fig. 4.2, en ella puede observarse que cada componente debe estar conectado inalámbricamente mediante WiFi a un bróker o *broker*. Para este trabajo, en una Raspberry centralizo la comunicación de la CMF.

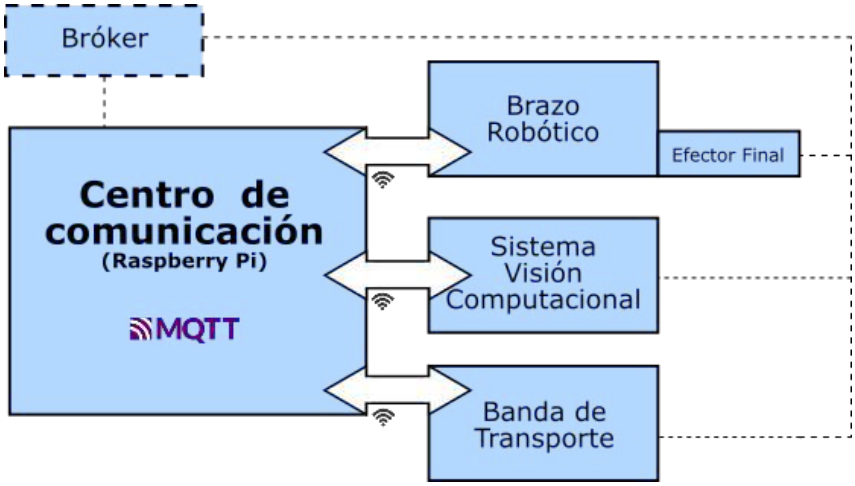


Figura 4.2: Arquitectura general de comunicación de la CMFI.

4.2.2. Implementación de la Comunicación General

La comunicación general la baso en el protocolo MQTT, utilicé el bróker Mosquitto (2021), este agente es perfecto ya que es liviano, seguro, adecuado para su uso en toda una gama de dispositivos y es de código abierto. Según su página oficial, este puede ser instalado en Windows, Linux, Mac y en dispositivos de menor consumo como lo son las micro computadoras Raspberry, por tanto, utilicé un dispositivo de este tipo para tal efecto, debido a su potencial uso y bajo costo, específicamente una Raspberry Pi 4 Model B, en el apéndice Raspberry Pi 4 muestro las características técnicas de este dispositivo.

Configurar un bróker Mosquitto dentro de una Raspberry es sencillo, solo se necesitan los siguientes comandos en la terminal para aplicarlo:

```
pi@raspberry:~ $ sudo apt update
pi@raspberry:~ $ sudo apt install -y mosquitto mosquitto-clients
pi@raspberry:~ $ sudo systemctl enable mosquitto.service
```

Respecto a los tópicos, estos dependerán de cada módulo de la CMFI, se tienen planeados dentro del diseño los mostrados en la siguiente tabla:

Número	Módulo CMFI	Tópico	Descripción
1	BT	LEAI40/CMFI/BT/VIN	Velocidad de entrada
2	BT	LEAI40/CMFI/BT/DIN	Dirección de entrada
3	BT	LEAI40/CMFI/BT/VOUT	Velocidad de salida
4	BT	LEAI40/CMFI/BT/DOUT	Dirección de salida
5	SVA	LEAI40/CMFI/SVA/POS	Posición de agarre
6	SVA	LEAI40/CMFI/SVA/ANG	Ángulo de agarre
7	SVA	LEAI40/CMFI/SVA/PINZA	Apertura de la pinza
8	BR	LEAI40/CMFI/BR/OBJ	Si se agarró o no al objeto

Tabla 4.1: Tópicos dentro del protocolo MQTT.

4.3. Banda Transportadora

La BT que se encuentra dentro del LEAI 4.0 cuenta con una área efectiva de 0.4 m^2, la cual es impulsada por un motor de corriente alterna monofásico con una potencia de 120 W con un rango de velocidad angular que va de 90 hasta 1600 RPM (ver Fig. 4.3).

Figura 4.3: Banda Transportadora con la que cuenta el LEAI 4.0.

En trabajos previos, ya se ha comenzado a actualizar el controlador de la banda transportadora, Gómez (2019) explica la caracterización física del motor y el funcionamiento de una interfaz realizada para el control de movimiento manual de la banda utilizando tecnologías como AJAX, sin embargo, es necesario realizar una actualización más para que su velocidad y dirección sean controlados de forma automática en función del SVA y del BR; además de llevar a cabo la actualización del control de la banda, debo integrarla como componente a la CMFI, para tal efecto, es necesario utilizar sistemas embebidos y/o en chip que puedan cumplir ambas funciones.

4.3.1. Arquitectura de Comunicación de la Banda Transportadora

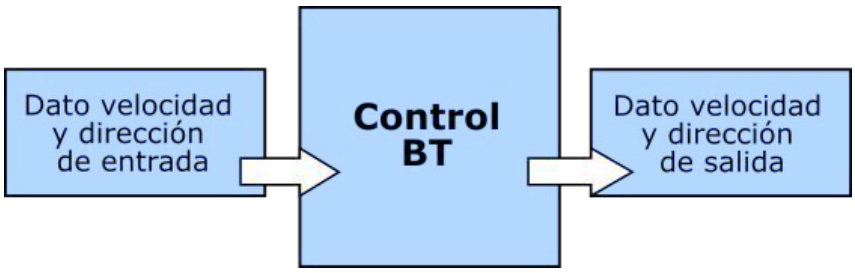


Figura 4.4: Arquitectura de comunicación de la Banda Transportadora.

La arquitectura de la BT consta de tres partes: envío y recepción de los datos de velocidad y dirección, y el control de esta en función de la información recabada.

Según la Fig. 4.4:

- Envío y recepción de dato de velocidad: el envío y recepción de la información está a cargo del módulo NodeMCU ESP32 a través del protocolo MQTT.
- Control de velocidad: este control está en función de la información de entrada, emitiendo señales de modulación por ancho de pulso (PWM por sus siglas en inglés) para el movimiento del motor.

4.3.2. Control de Velocidad de la BT

La velocidad y dirección de la BT están establecidas por medio de un motor de corriente alterna, sin embargo, debido a temas sanitarios, implementé un sistema prototipo con un motor de corriente continua, haciendo énfasis al controlador lógico.

Existe una gran y compleja gama de dispositivos llamados System on Chip (SoC) disponibles en el mercado que pueden llevar a cabo el envío de señales Pulse-Width Modulation (PWM), sin embargo, esta señal depende de la información que circula dentro de la CMFI por lo que el dispositivo debe contar con conectividad inalámbrica del tipo WiFi. Afortunadamente, este tipo de dispositivos son comunes dentro del ámbito IoT, de manera que busqué sistemas embebidos que cumplan las dos funciones mencionadas anteriormente. Presento en la tabla 4.2 una comparativa entre algunos SoC más comunes de uso, haciendo énfasis en si cuentan con PWM y con convertidores digital-analógico (DAC), esto último para prevenir si el controlador del motor AC no sea compatible con PWM.

Nombre	Fabricante	Procesador	Reloj	PWM	DAC	Precio	Información
ESP8266	Espressif	Xtensa L106	>80 MHz	X		5 USD	Enlace
ESP32	Espressif	Xtensa LX6	240 MHz	X	X	9 USD	Enlace
CC32XX	Texas Instruments	ARM Cortex-M4	80 MHz	X		40 USD	Enlace
RTL8710	Realtek	ARM Cortex-M3	166 MHz	X		10 USD	Enlace
MT768X	MediaTek	ARM Cortex-M4	>80 MHz	X		20 USD	Enlace

Tabla 4.2: Comparativa de SoC en IoT.

Es por lo anterior que, decidí utilizar el módulo NodeMCU ESP32 en su variante ESP-WROOM-32 para el segmento lógico del circuito electrónico presentado en la Fig. 4.5, en el apéndice NodeMCU ESP32s presento las características técnicas de este dispositivo.

Este circuito lo diseñé con la finalidad de controlar el giro de un motor DC, este tiene dos segmentos principales: la lógica a través de un módulo NodeMCU ESP32s y la potencia por medio de un puente H L293D[1]. Respecto al software, decidí hacer un sistema

[1]La hoja de datos técnicos del componente L293D, puede verse en: https://www.ti.com/lit/ds/symlink/l293.pdf.

de lazo abierto utilizando PWM y Look Up Tables (LUTs), con la finalidad de calibrar la velocidad de motor respecto a los pulsos enviados desde el microcontrolador, logrando de esta manera, un cómputo más eficiente.

Este sistema es un prototipo, deberá ser actualizado más adelante para su uso con un motor AC, cabe recalcar que el segmento de lógica lo diseñé pensando en mínimas modificaciones.

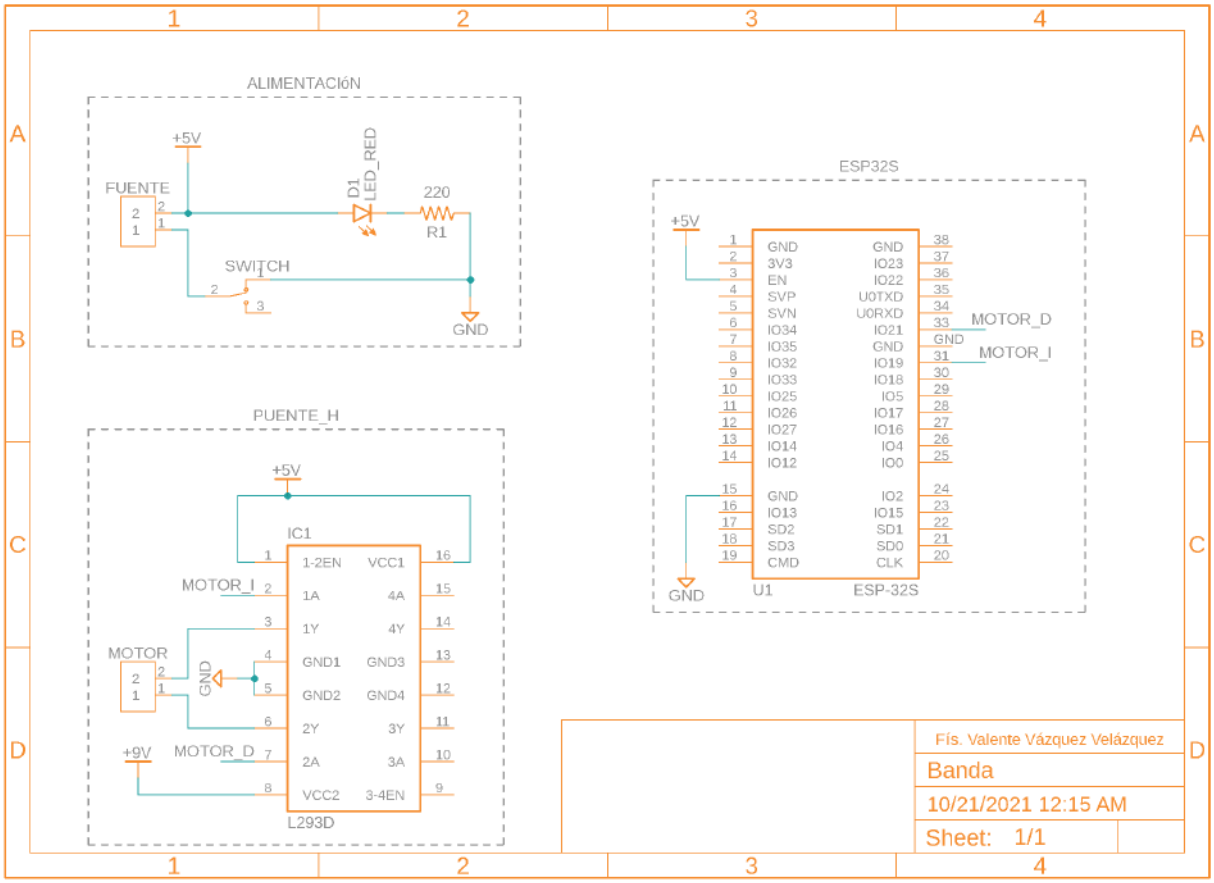


Figura 4.5: Circuito del sistema prototipo de control de la Banda Transportadora.

4.3.3. Comunicación de la BT

El módulo ESP32s fue programado con el entorno Arduino, se utilizó la biblioteca del protocolo MQTT de nombre *EspMQTTClient.h* (Lapointe, 2021) para poder publicar y recibir la información requerida. Para este fin se dieron de alta los primeros cuatro tópicos mostrados en la tabla 4.1. El funcionamiento del sistema de comunicación y control de la BT corresponde al diagrama de flujo mostrado en la Fig. 4.6.

Dentro del programa, los datos técnicos utilizados en la implementación de este tipo de señales, fueron: Frecuencia = 10 KHz y Bits de resolución = 8; cabe aclarar que estos datos son dependientes del tipo de motor con el que se trabaje. El código completo puede observarse en el anexo Códigos en C.

Figura 4.6: Diagrama de flujo del funcionamiento sistema de comunicación y control de la BT

4.4. Sistema de Visión Artificial

El SVA contempla adquirir imágenes por medio de una cámara, estas deberán ser procesadas digitalmente en un sistema embebido y/o computadora para aplicar métodos y algoritmos de visión computacional con el objetivo de sustraer características físicas de los objetos que sean transportados por la BT.

Este sistema lo implementé con una cámara de 5 MP conectada a una Raspberry Pi 4 Model B (la información técnica se encuentra en el apéndice Cámara Raspberry Pi 4) en vez de algunas de las cámaras comparadas en la Tabla 2.2 del capítulo 2 (debido a temas sanitarios), esto con la finalidad de probar físicamente el sistema ejecutándose dentro de

este sistema embebido.

Para este trabajo, diseñé un algoritmo de visión capaz de obtener un vector característico de una pieza o figura de geometría simple, con el fin de obtener su ángulo de sujeción óptimo, su tamaño de la pieza para saber que tanto abrir la pinza de agarre y un vector de posiciones respecto al tiempo. Este algoritmo consta de dos partes fundamentales: obtener la orientación y el tamaño de la pieza al calcular el vector característico BOF y el seguimiento de la pieza en tiempo real por medio de flujo óptico.

4.4.1. Arquitectura del Sistema de Visión Artificial

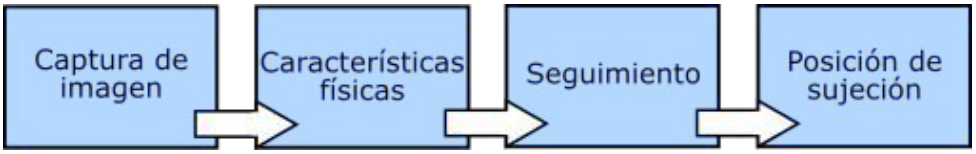


Figura 4.7: Arquitectura del SVA.

La arquitectura del SVA consta de cuatro elementos principales (véase Fig. 4.7): la cámara captura una imagen con una figura de geometría simple dentro de su campo de visión, se procesa dicha imagen para obtener sus características físicas de la figura junto al vector descriptivo, se da seguimiento en tiempo real de la pieza por medio de flujo óptico, y por último, se analiza la trayectoria por el método de mínimos cuadrados para así calcular la posición de agarre y enviarla al BR.

4.4.2. BOF

Para obtener las características físicas como lo son el tamaño de las piezas y su orientación, es necesario obtener un vector descriptivo llamado BOF, por lo que realicé una serie de pasos en procesamiento de imágenes, tales como los describí en el capítulo anterior.

Con el fin de corroborar que este proceso funcione dentro de la Raspberry, procedí a documentar el proceso para serlo completamente inteligible utilizando una figura com-

puesta de piezas lego, tal como los muestro a continuación (el algoritmo de este proceso se muestra en el diagrama de flujo de la Fig. 3.2):

- *Captura de imagen.* Una vez que se tiene una figura de geometría simple dentro del campo de visión de la cámara, se toma una imagen.

Figura 4.8: Imagen prueba para el SVA.

- *Escala de grises.* La imagen mostrada anteriormente, es pasada al espacio de color monocromático con 8 bits de resolución.

Figura 4.9: Fig. 4.8 en escala de grises.

- *Filtro pasa-bajas.* Ahora, a la imagen de la Fig. 4.9 se le aplica una convolución con un kernel gaussiano de 5×5, con el objetivo de implementar un filtro pasa-bajas y

obtener una imagen difuminada.

Figura 4.10: Fig. 4.9 con filtro gaussiano.

- *Histograma*. Enseguida, se obtiene un histograma de la nueva figura, esto con la intención de calcular un umbral para de separar con un punto medio los tonos bajos (píxeles correspondientes a la banda) y los altos (correspondientes a la pieza).

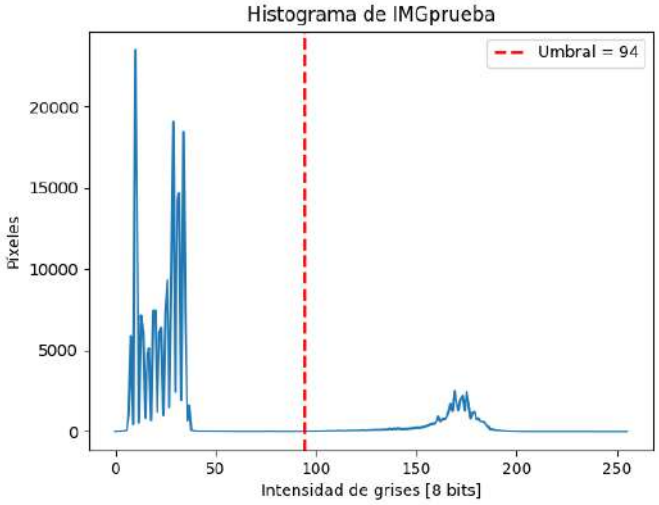


Figura 4.11: Histograma de la Fig. 4.10.

- *Binarización*. Con ayuda del umbral obtenido, se procede a realizar una binarización de la imagen anterior; el objetivo es crear una nueva imagen donde los pixeles

correspondientes a la banda sean negros (0) y los correspondientes a la figura o pieza, sean blancos (255).

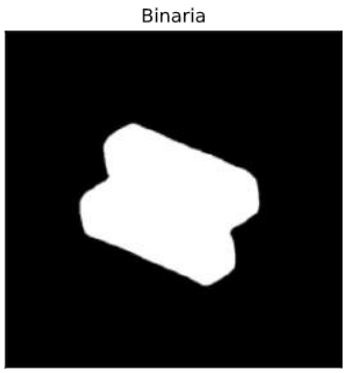


Figura 4.12: Fig. 4.10 binarizada.

- *Centroide*. Se obtiene el único punto característico: el centroide de la figura. Este dato es fundamental para calcular el vector BOF y para su posterior seguimiento por medio de flujo óptico.

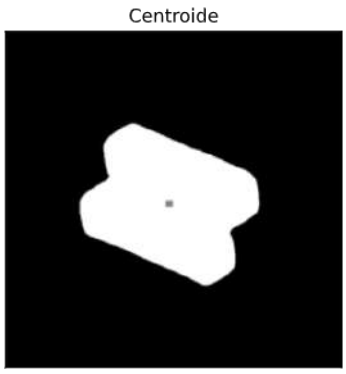


Figura 4.13: Centroide de la Fig. 4.12 señalado.

- *BOF*. Por último, a partir de la Fig. 4.13 se obtiene el vector BOF, en la Fig. 4.14 se muestra una gráfica del vector BOF de la pieza mostrada en la Fig. 4.8. Nótese que la distancia se encuentra normalizada. Así mismo, en la Fig. 4.15, se muestran los resultados numéricos obtenidos en este ejemplo.

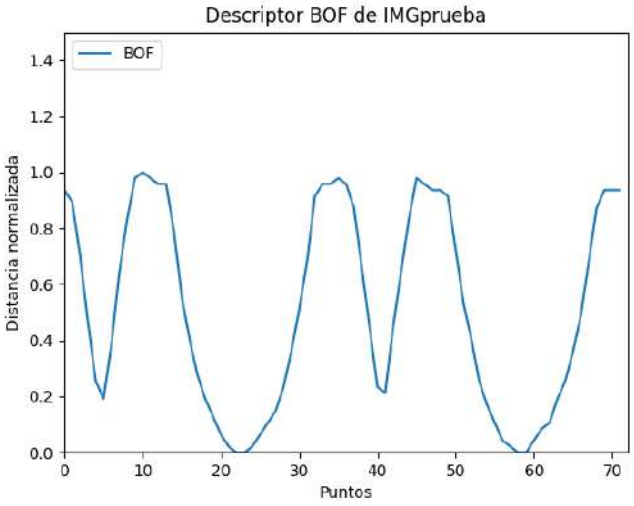


Figura 4.14: Vector BOF de la pieza de la Fig. 4.8.

```
Vector BOF: [0.9361702127659575, 0.8936170212765957, 0.7021276595744681, 0.4680851063
8297873, 0.2553191489361702, 0.19148936170212766, 0.3829787234042553, 0.6382978723404
256, 0.8297872340425532, 0.9787234042553191, 1.0, 0.9787234042553191, 0.9574468085106
383, 0.9574468085106383, 0.7872340425531915, 0.5531914893617021, 0.40425531914893614,
 0.2765957446808511, 0.19148936170212766, 0.1276595744680851, 0.06382978723404255, 0.
02127659574468085, 0.0, 0.0, 0.02127659574468085, 0.06382978723404255, 0.106382978723
40426, 0.14893617021276595, 0.23404255319148937, 0.3617021276595745, 0.51063829787234
04, 0.6808510638297872, 0.9148936170212766, 0.9574468085106383, 0.9574468085106383, 0
.9787234042553191, 0.9574468085106383, 0.8723404255319149, 0.6595744680851063, 0.4468
0851063829785, 0.23404255319148937, 0.2127659574468085, 0.44680851063829785, 0.638297
8723404256, 0.8297872340425532, 0.9787234042553191, 0.9574468085106383, 0.93617021276
59575, 0.9361702127659575, 0.9148936170212766, 0.723404255319149, 0.5319148936170213,
 0.40425531914893614, 0.2553191489361702, 0.1702127659574468, 0.10638297872340426, 0.
0425531914893617, 0.02127659574468085, 0.0, 0.0, 0.0425531914893617, 0.08510638297872
34, 0.10638297872340426, 0.19148936170212766, 0.2553191489361702, 0.3617021276595745,
 0.48936170212765956, 0.6808510638297872, 0.8723404255319149, 0.9361702127659575, 0.9
361702127659575, 0.9361702127659575, 0.9361702127659575]
Orientación: 70 grados
Tamaño del eje más largo: 137 píxeles
Centroide: [242, 256]
```

Figura 4.15: Resultados numéricos al obtener el vector BOF de la Fig. 4.8.

Puede notarse en la imagen anterior que, con la información obtenida también fue posible obtener las características mencionadas al principio de esta subsección además del centroide: orientación o ángulo de sujeción y tamaño de la pieza.

4.4.3. Flujo Óptico

El algoritmo de Lucas-Kanade trabaja con puntos característicos dentro de las imágenes (o *frames*) que analiza, en este trabajo solamente existe un punto de interés útil para realizar el seguimiento en tiempo real por parte del SVA, el cual es el centroide de la figura.

Ejemplificando, en la Fig. 4.16 muestro cuatro cuadros del seguimiento obtenido utilizando flujo óptico a la pieza de la Fig. 4.8 mientras es desplazada horizontalmente, en estas imágenes pueden observarse la posición actual del centroide y la rapidez, principalmente. Por cada cuadro que captura la cámara, este proceso calcula y guarda en una lista la nueva posición de dónde se encuentra el centroide, esto con el fin de calcular la velocidad media de la pieza haciendo un análisis por mínimos cuadrados al utilizar la ecuación 3.19, e intrínsecamente también la de la BT.

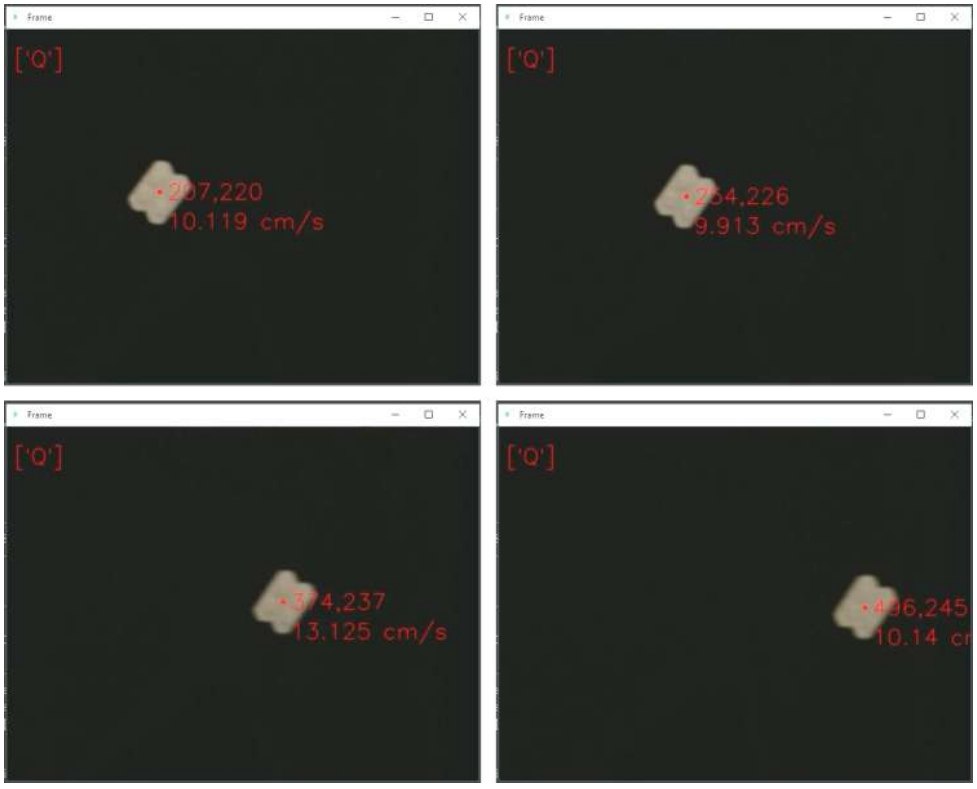


Figura 4.16: Fotogramas del seguimiento por flujo óptico de la pieza de la Fig. 4.8.

Para este ejemplo, la lista de las posiciones en el eje x, queda como:

$$
\begin{aligned}
P_x = \; & [201, \; 205, \; 207, \; 211, \; 213, \; 216, \; 220, \; 223, \; 226, \; 229, \\
& 232, \; 234, \; 237, \; 241, \; 244, \; 246, \; 250, \; 253, \; 254, \; 259, \\
& 262, \; 265, \; 267, \; 270, \; 274, \; 277, \; 279, \; 282, \; 286, \; 288, \\
& 291, \; 294, \; 298, \; 301, \; 304, \; 306, \; 310, \; 312, \; 316, \; 319, \\
& 322, \; 325, \; 327, \; 330, \; 333, \; 336, \; 340, \; 342, \; 346, \; 349, \\
& 352, \; 354, \; 357, \; 361, \; 364, \; 367, \; 369, \; 373, \; 374, \; 379, \\
& 382, \; 385, \; 387, \; 390, \; 394, \; 397, \; 400, \; 402, \; 405, \; 408, \\
& 411, \; 414, \; 417, \; 421, \; 424, \; 426, \; 430, \; 432, \; 436, \; 438, \\
& 441, \; 444, \; 448, \; 451, \; 454, \; 456, \; 460, \; 463, \; 466, \; 468, \\
& 472, \; 474, \; 477, \; 481, \; 483, \; 486, \; 489, \; 492, \; 496, \; 499, \\
& 501, \; 504, \; 508]
\end{aligned}
$$

de donde, al realizar un análisis de la información por mínimos cuadrados con la ecuación 3.19 se obtiene una velocidad media de:

$$v = 10.8 \pm 0.1 \text{ cm/s}$$

Este método nos arroja dos variables importantes: la incertidumbre de la velocidad y el coeficiente de determinación (r^2); estos datos son fundamentales en la decisión de si es posible o no tomar el objeto que atraviesa el campo de visión de la cámara. Una vez puesto en marcha el sistema, se calibrará qué porcentaje de error será el máximo permitido al dato de la velocidad y el número mínimo permitido que tendrá el coeficiente de determinación. El programa completo del sistema puede observarse en el anexo Códigos en Python, junto con las bibliotecas propias realizadas, como la comunicación vía MQTT, el análisis por mínimos cuadrados y el algoritmo del vector BOF.

4.4.4. Comunicación del SVA

Dado que la información será enviada de manera inalámbrica a través de MQTT, entonces los temas para estos resultados, son:

- T5: LEAI40/CMFI/SVA/POS
- T6: LEAI40/CMFI/SVA/ANG

- T7: LEAI40/CMFI/SVA/PINZA

en T5 se envía la posición de agarre del objeto, en T6 la orientación de agarre y en T7 el tamaño de la pieza. Cabe resaltar que los tópicos 6 y 7 dependen del tipo de efector final que se utilice en el brazo robótico, si es una pinza estas variables son muy importantes, en cambio, si se utiliza una ventosa de vacío estos datos no serán necesarios.

4.5. Brazo Robótico

El BR marca KUKA con el que cuenta el LEAI 4.0 es el modelo KR 5-2 arc HW, en el apéndice Brazo Robótico muestro sus características técnicas más importantes.

Figura 4.17: Brazo Robótico Kuka con el que cuenta el LEAI 4.0

4.5.1. Inserción del Brazo Robótico

Una vez que el SVA calcula y publica la posición de agarre de la pieza, el brazo robótico comienza a actuar, además es preferible saber si el robot tomó o no la pieza en cuestión, esto es posible si se utilizan sensores de proximidad en el efector final del BR, donde esta información sería publicable en el tópico 8 mostrado en la tabla 4.1.

Ahora, para comunicar el SVA con el BR se deben utilizar otros diferentes medios de

los que he descrito anteriormente. Para este fin, y dado que las capacidades del software del robot lo permiten, se utiliza el protocolo TCP/IP, a través de este es posible enviar y/o recibir información hacía/desde el robot, de hecho, existe un servidor TCP/IP que permite realizar todo lo anterior de nombre KukavarProxy (Sanfilippo et al., 2015), este ya ha sido ejecutado dentro del LEAI 4.0 con resultados satisfactorios.

Envío de información

El robot industrial Kuka se encuentra integrado en una red LAN, esto con el fin de integran varios módulos para enlazar una comunicación y poder realizar diferentes procesos con un mismo robot. La Raspberry Pi donde se ejecuta el SVA debe estar conectada en la misma red donde se encuentra el BR mediante el protocolo TCP/IP, tomando un rango de nuestra IP entre: 192.168.0.0 a 192.168.0.255, 172.16.0 a 172.16.255.255, 172.31.0 a 172.31.255.255, esto de acuerdo al manual del robot (KUKA, 2015b).

Para lograr la comunicación del robot con la Raspberry Pi será necesario abrir archivos dentro del software del robot, siendo más preciso se modificará el programa *config.dat* que se encuentra disponible al iniciar como usuario experto. Este programa contiene variables globales de la programación del robot (ver Fig. 4.18).

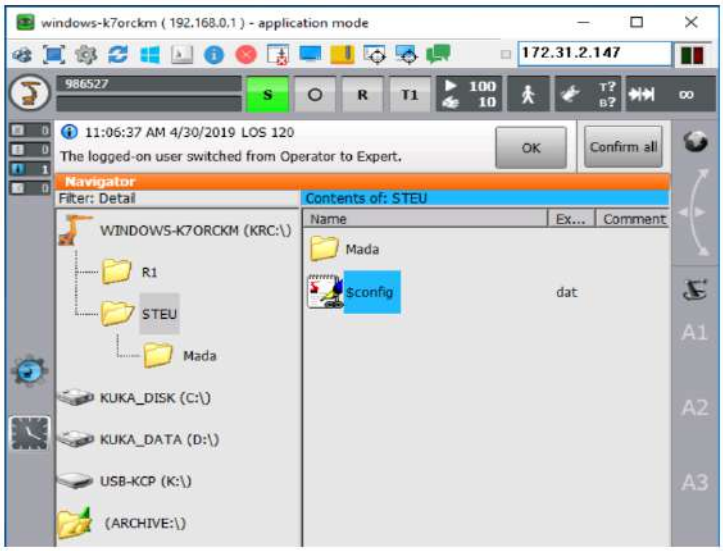


Figura 4.18: Captura de pantalla del Pad del BR mostrando el archivo config.dat.

En este archivo es necesario escribir una variable global de enlace para que a través de la comunicación entre la Raspberry Pi y el brazo industrial se puedan modificar ciertos parámetros, principalmente las coordenadas X, Y, Z, α, β y γ, las primeras tres son del sistema de coordenadas Mundo y las últimas tres los ángulos del último grado de libertad (donde se ubica el efector final) respecto a los primeros tres ejes. Por tanto, declaro la variable UBICACION, donde los parámetros que deberá leer esta variable son los ángulos mencionados anteriormente provenientes del SVA (ver Fig. 4.19).

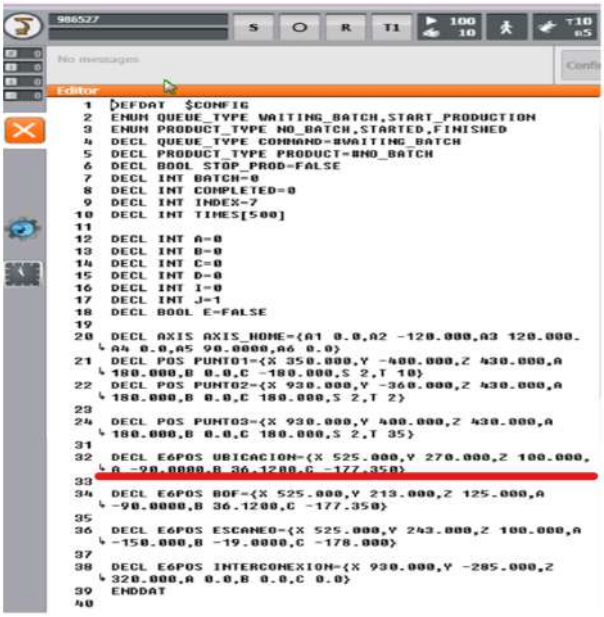


Figura 4.19: Captura de pantalla del Pad del BR mostrando la variable UBICACION declarada el archivo config.dat.

Una vez definidas las variables de enlace, se utilizará el modo de trabajo automático del brazo industrial, en el cual el sistema del brazo libera todos sus modos de seguridad, esto con la finalidad de que una vez iniciado el programa el robot puede trabajar sin la necesidad de un operador. Para lograr que el robot lea las variables de enlace que manda la Raspberry Pi, se debe escribir un programa en el cual se hagan referencia a dichas variables creadas en el fichero *config.dat*, ahí mismo es posible escribir los movimientos autónomos del robot, como a dónde mover la pieza que sostiene (ver Fig. 4.20).

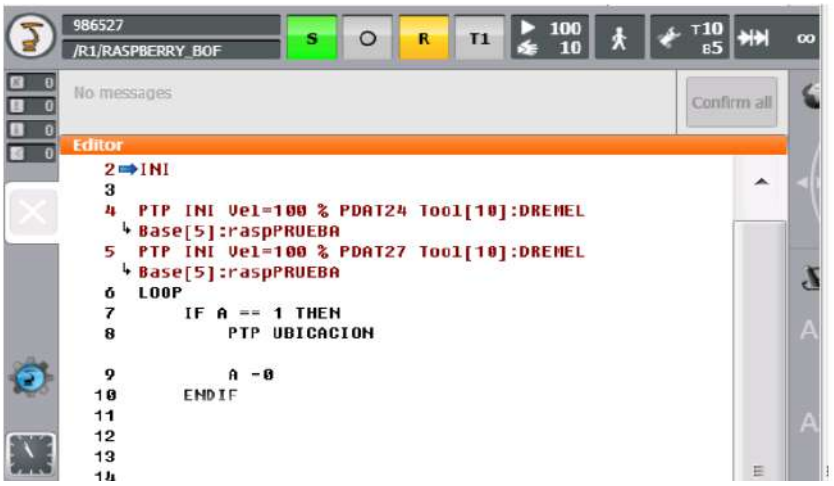


Figura 4.20: Captura de pantalla del Pad del BR mostrando un programa simple utilizando la variable UBICACION.

Por último, simplemente en nuestro programa del SVA tendremos que pasar cada uno de los parámetros que se definieron como las variables de enlace, que serían las posiciones en X, Y y Z, así como el ángulo de inclinación α, β y γ de la pinza de agarre. Es necesario que en este programa se encuentre inicializada la biblioteca kukavarproxy. En el siguiente código muestro una sintaxis en Python:

```
from kukavarproxy import *
from time import sleep

robot = KUKA ("172.31.2.147")
sleep(5)

robot.write("UBICACION", "{E6POS: X 500, Y 250, Z 700, A -90.000, B 0, C
 180}")
```

4.6. Sistema de Agarre de Piezas de Manufactura

Una vez concluidas las secciones anteriores, procedo a unificarlas para así obtener el sistema objetivo de este trabajo de tesis.

4.6.1. Arquitectura del SAPM

En primer lugar, la arquitectura general del SAPM consta de cuatro unidades o módulos(véase la Fig. 4.21):

- *Banda Transportadora*. La BT tendrá un movimiento según lo indique el SVA y/o el BR.
- *Sistema de Visión Artificial*. Este sistema es el encargado de recabar información gráfica de piezas sobre la banda, a partir de esta se envía la información correspondiente al SAPM.
- *Brazo Robótico*. El BR recibirá las coordenadas a la cual debe moverse para sujetar la pieza registrada por el SVA y notificar si el efector final realizó una correcta sujeción o no.
- *Sistema de Agarre de Piezas de Manufactura*. Esta parte es la encargada de administrar la información entrante y/o saliente de las tres unidades anteriores, el objetivo principal de este trabajo de tesis.

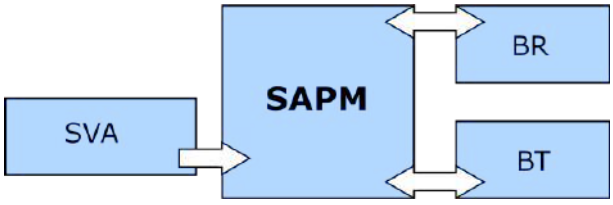


Figura 4.21: Arquitectura general de la CMF.

4.6.2. Algoritmo del SAPM

La información circulante dentro de la CMFI es administrada por el SAPM, dicha administración debe ser jerárquica, el movimiento de la BT está función del SVA y el BR, en ese orden (véase la Fig. 4.22), es decir, una vez que el SVA puede realizar un correcto seguimiento de la banda, ahora el control de la velocidad de esta lo toma el brazo revisando si el agarre de la pieza resultó efectivo o no.

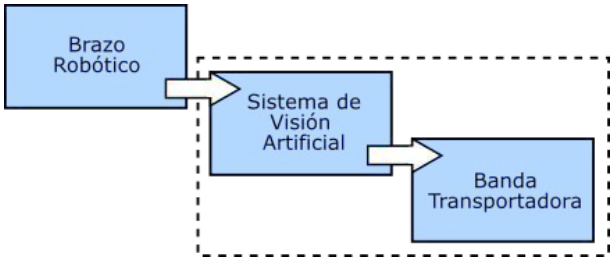


Figura 4.22: Control jerárquico de la BT.

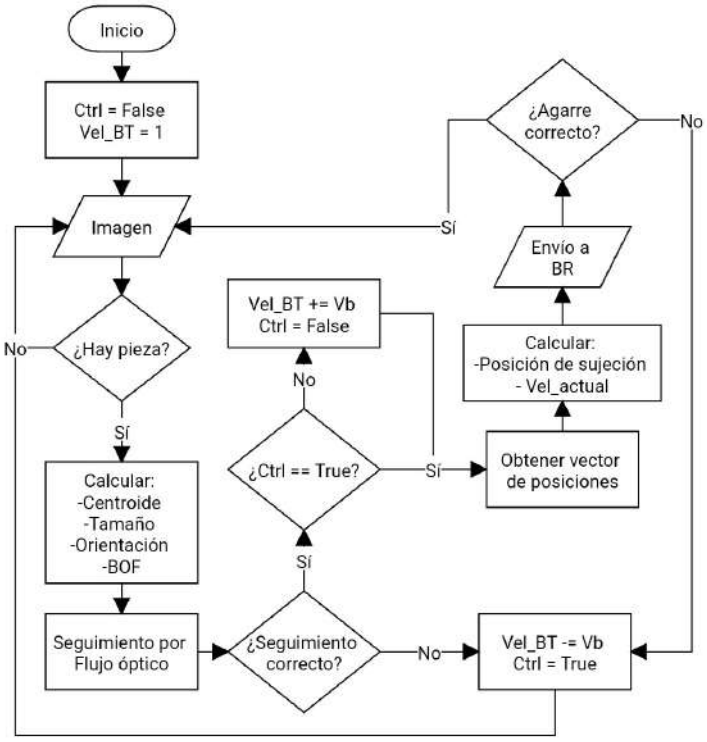


Figura 4.23: Diagrama de flujo del algoritmo implementado en el SAPM.

Una vez que se acciona el movimiento de la BT también se ejecuta el SVA; su funcionamiento lo describo a continuación: cada vez que exista una pieza dentro del campo de visión de la cámara, el SVA calculará el vector BOF y dará seguimiento al trayecto, si este seguimiento fue correcto, entonces se envía la información al BR sobre las coordenadas de sujeción del objeto y se aumenta un valor v_b a la velocidad de la BT, así hasta que el seguimiento de un objeto sea incorrecto; la penúltima velocidad será la óptima para el funcionamiento del sistema de visión. Por otra parte, el BR confirma si el agarre de la pieza fue o no satisfactorio, si sí lo fue la velocidad se mantiene, de lo contrario disminuye (véase Fig. 4.23).

4.7. Implementación virtual del SAPM

Antes de realizar el montaje en físico, es recomendable simular el proceso, esto con el objetivo de identificar fallas sin poner el riesgo al equipo o al personal. Es por lo anterior que formé una CMF dentro del simulador CoppeliaSim (véase Fig. 4.24).

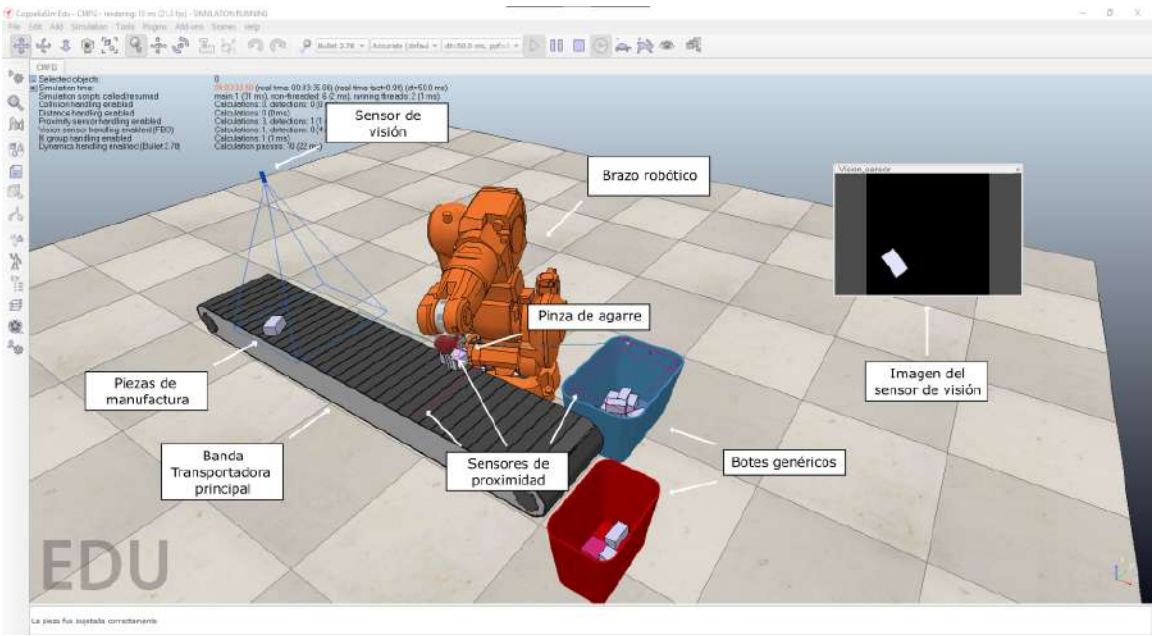


Figura 4.24: CMF puesta en marcha dentro del simulador CoppeliaSim.

4.7.1. Comunicación Python-CoppeliaSim

La forma común de programar las acciones que debe realizar cada elemento componente de la CMF es a través de *scripts* escritos en lenguaje Lua, sin embargo, dado que los algoritmos del SVA los programé en Python, decidí utilizar ambos lenguajes, Lua será para las tareas específicas dentro del simulador y Python para ejecutar el SVA principalmente, ambos estarán interconectados, esto es posible gracias a la API de Pyhton del simulador, el cual hace la conexión remota entre ambos a través de funciones ya establecidas (Coppelia-Robotics, 2020).

El primer paso es crear una conexión entre Python y CoppeliaSim, para esto, con Python es necesario tener instalada la biblioteca *sim*, de donde la función *simxStart* es la encargada de enlazar ambas plataformas, solo es necesario darle los parámetros de host y puerto, principalmente. En el siguiente código es posible observar su implementación:

```
import sim


sim.simxFinish(-1)
clientID = sim.simxStart('127.0.0.1', 19999, True, True, 5000, 5)
```

Ahora, dentro del simulador, se crea un *Non Threaded child script* para algún elemento de la celda, como puede ser el sensor de visión o la banda transportadora, y se escribe la siguiente función:

```
function sysCall_init()
simRemoteApi.start(19999)
end
```

Con esto es suficiente para comunicar el simulador CoppeliaSim con Python, por lo que si todo marcha correctamente, al ejecutar la simulación y enseguida el programa de Python, no debería existir ningún error en ambos.

4.7.2. Configuración de los elementos componentes de la CMF

A continuación enlisto los elementos de la CMF mostrado en la figura 4.24, mencionando sus principales características y funciones:

- *Banda transportadora*. La BT tiene como objetivo transportar de un punto a otro las piezas de manufactura sobre ella. Dentro del simulador coloqué en este objeto un *Non Threaded child script*, en el cual realizo la conexión entre Python y el simulador, programo las rutas por las que se moverá el robot y su velocidad de movimiento.

- *Sistema de visión*. El sensor de visión fue uno de tipo perspectivo, lo configuré a una resolución de 256×256 px, esto debido a que el simulador envía la imagen al SVA en forma de lista, por lo que es necesario hacer una remodelación de la información y trasladar de una dimensión a dos y esto consume más recursos computacionales.

- *Piezas de manufactura*. Una gran ventaja de utilizar este simulador es crear cada cierto tiempo objetos de geometría simple, los cuales serán utilizados como piezas de manufactura para nuestro caso. Programé un *Threaded child script* donde cada 15 segundos aparece una nueva pieza sobre la banda transportadora. Para este trabajo utilicé dos tipos de objetos un prisma rectangular de tamaño $4 \times 7 \times 3.5$ cm y cubos de tamaño $5 \times 5 \times 5$ cm, repartidas en posiciones y orientaciones aleatorias.

- *Brazo robótico*. El BR que utilicé para la implementación dentro del simulador es el ABB IRB 140 que cuenta con 6 grados de libertad, implementé su movimiento a través de cinemática inversa utilizando el tutorial oficial de Coppelia-Robotics (2021b). La elección de este brazo la basé en que es el robot con las características más parecidas al brazo físico que se encuentra dentro del LEAI 4.0. Le programé un *Threaded child script* donde especifico cuándo y hacía dónde se mueve el BT utilizando las rutas programadas en el *script* de la banda. En la Fig. 4.25 se muestra un diagrama de flujo del proceso.

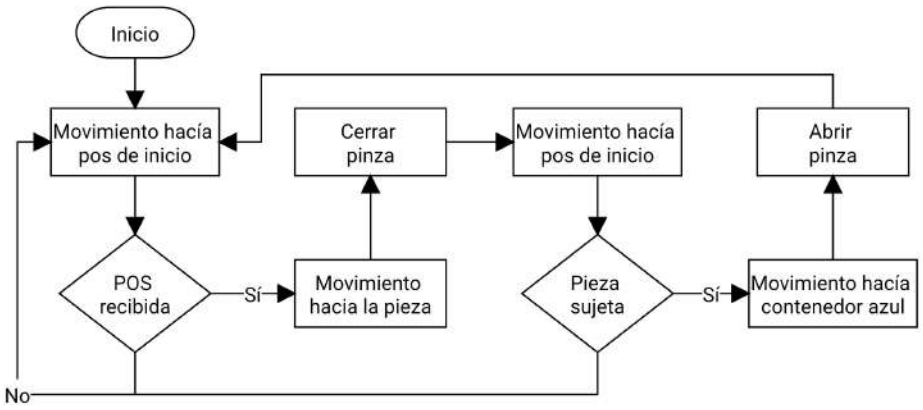


Figura 4.25: Diagrama del funcionamiento del BR.

- *Pinza de agarre*. Este efector final es de tipo *Baxter Gripper*, lo seleccioné por tener las paletas de agarre más anchas, dando una ventaja al momento de tomar los objetos.
- *Contenedores*. Estos contenedores genéricos son utilizados para depositar las piezas de manufactura, en el azul se depositan las piezas sujetadas correctamente y en el rojo las piezas perdidas.
- *Sensores de proximidad*. Existen tres sensores de proximidad, en la pinza, en la banda y en el contenedor azul, el primero notifica al SVA si la pieza fue tomada o no, mientras que los últimos dos solo existen para tener un registro de cuántas piezas entran al contenedor azul y cuántas al rojo, con la intención de tener un registro numérico de esa información.

En el anexo Códigos en Lua, ubicado al final de este texto, se pueden observar los diferentes *scripts* realizados dentro del simulador CoppeliaSim.

Esta CMF funciona de la siguiente manera: una vez recibida la posición de agarre y el ángulo de sujeción, el robot va a la posición, sujeta la pieza y la deposita dentro el contenedor azul; en caso de que exista un error y la pieza no haya sido tomada, entonces esta pieza termina al fondo del contenedor rojo.

4.7.3. Puesta en marcha del SAPM

Finalmente, una vez que cada componente ya se encuentra configurado para su correcto funcionamiento, se ejecuta la simulación en CoppeliaSim seguido del algoritmo mostrado en el diagrama de flujo de la Fig. 4.23 escrito en Python. El código final se muestra en el anexo Códigos en Python al final de este trabajo.

Para obtener una métrica del funcionamiento del SAPM, programé un par de funciones que tienen como propósito graficar el análisis por mínimos cuadrados de cada una de las piezas que el SVA analiza y al final, otra gráfica donde se muestran las velocidades de cada objeto que pasó por el campo visual del sensor de visión.

4.8. Implementación física del SAPM

Para la migración del sistema virtual al físico, es importante tener los siguientes puntos resueltos:

- *Base del sensor de visión.*
- *El sistema de coordenadas.*
- *Las piezas de manufactura reales.*
- *El efector final.*

4.8.1. Base del sensor de visión

Este sistema fue creado con la intención de colocar una cámara en posición cenit respecto a la BT, esto con el objetivo de digitalizar las piezas de manufactura que vayan atravesando su campo de visión, por lo que es necesario diseñar y montar una estructura que soporte la cámara.

El diseño de la estructura debe ser en forma de un prisma rectangular para que esta cuente con puntos de estabilidad, tal como se muestra en la Fig. 4.26. Como se observa

en esta figura, sobre la cara superior del prisma se encuentra la base donde se colocará el sensor de visión. El material propuesto para armar la estructura prototipo es PVC.

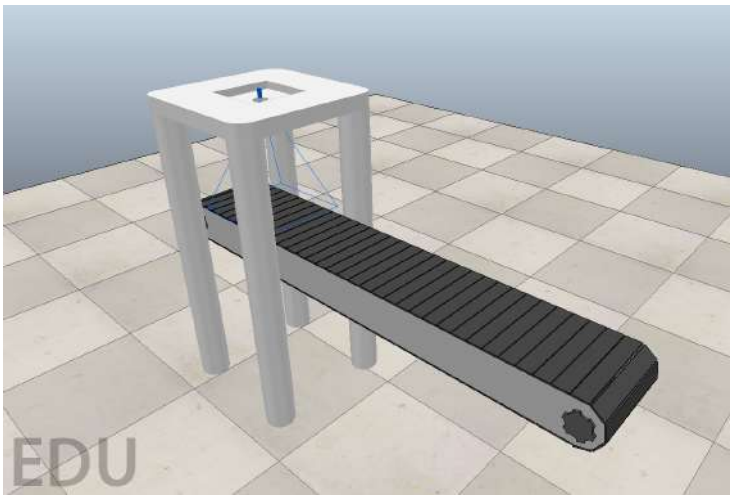


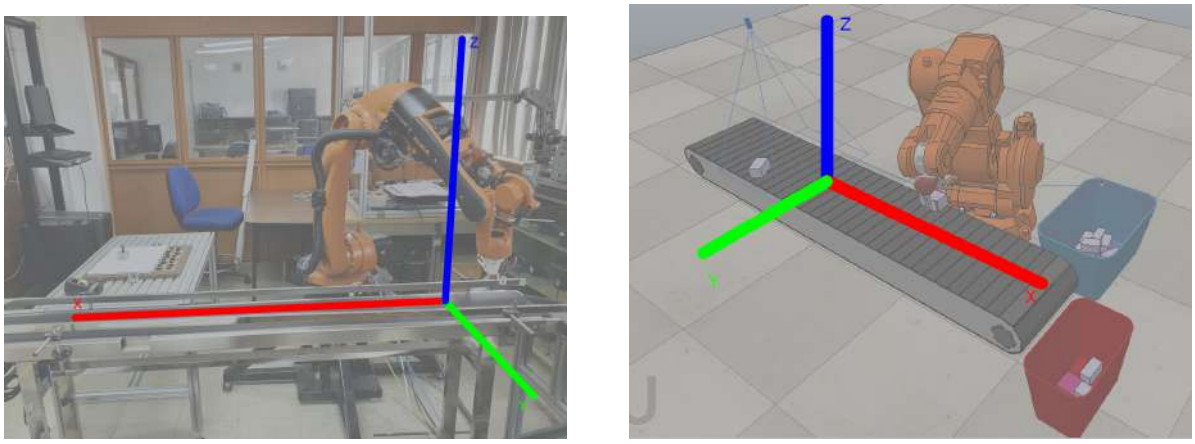
Figura 4.26: Base propuesta para el sensor de visión del SVA.

4.8.2. El sistema de coordenadas

El sistema de coordenadas utilizado en el BR Kuka es Mundo, tal como lo mencioné anteriormente. Ahora, es importante verificar que este sistema de coordenadas $(X_k, Y_k, Z_k, \alpha_k, \beta_k, \gamma_k)$ coincida con el sistema de coordenadas del simulador CoppeliaSim $(X_c, Y_c, Z_c, \alpha_c, \beta_c, \gamma_c)$, es decir:

$$(X_k, Y_k, Z_k, \alpha_k, \beta_k, \gamma_k) = (X_c, Y_c, Z_c, \alpha_c, \beta_c, \gamma_c) \tag{4.1}$$

Para esto, solo basta revisar y comparar los ángulos del sistema real (ver Fig. 4.27(a)) con el virtual (ver Fig. 4.27(b)).



(a) Ejes cartesianos de la CMF del LEAI 4.0

(b) Ejes cartesianos de la CMF de CoppeliaSim

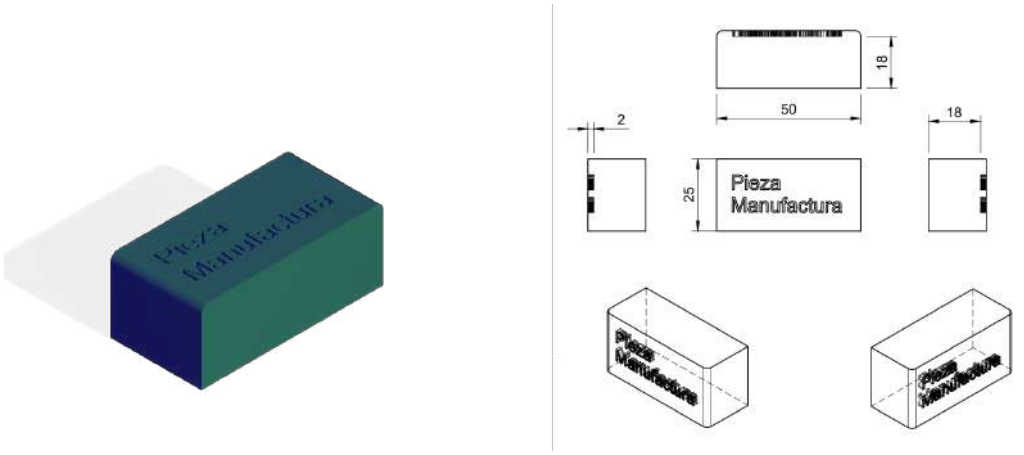
Figura 4.27: Ejes cartesianos de las CMF física y virtual.

En las imágenes anteriores es posible observar que el eje X de la CMF está en modo espejo respecto a la del simulador, sin embargo, esto no es problema al momento de enviar y/o recibir las coordenadas de sujeción si se realiza la calibración correspondiente, ya que el movimiento del robot depende de su propio sistema de referencia. Respecto a los ángulos de giro α, β y γ, la única diferencia es que el eje de giro sobre el eje Z en el sistema físico es la coordenada α, mientras que en el sistema virtual es la coordenada γ. Por tanto la ecuación 4.1 quedaría como:

$$(X_k, Y_k, Z_k, \alpha_k, \beta_k, \gamma_k) = (X_c, Y_c, Z_c, \gamma_c, \beta_c, \alpha_c) \tag{4.2}$$

4.8.3. Las piezas de manufactura reales

La solución más rápida y factible de obtener los objetos de geometría simple, que en este trabajo he llamado piezas de manufactura, es imprimirlas en plástico ABS, en 3D. Lo ideal, es diseñar prismas rectangulares, esto es debido a que es una pieza polivalente, con ella se nota el giro en el eje de la pinza para su sujeción, la pieza propuesta es posible observarla en la Fig. 4.28.



(a) Imagen 3D de la pieza de manufactura. (b) Diagrama de la pieza de manufactura (mm).

Figura 4.28: Pieza de manufactura.

4.8.4. El efector final

En general existen dos tipos de efectores para sujetar piezas: pinzas y de vacío. En el primero es importante el ángulo de agarre mientras que en el segundo no, por lo que el primer dispositivo es el idóneo para este trabajo. Lo ideal es de nuevo imprimir en 3D una pinza de sujeción, así se controla tanto el tamaño como la forma de la pinza. Un ejemplo de pinzas de sujeción la da Atthariq (2021) en la Fig. 4.29.

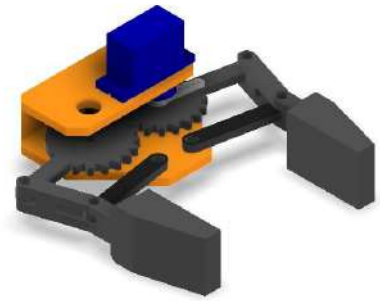


Figura 4.29: Modelo de pinza como efector final en 3D (Atthariq, 2021).

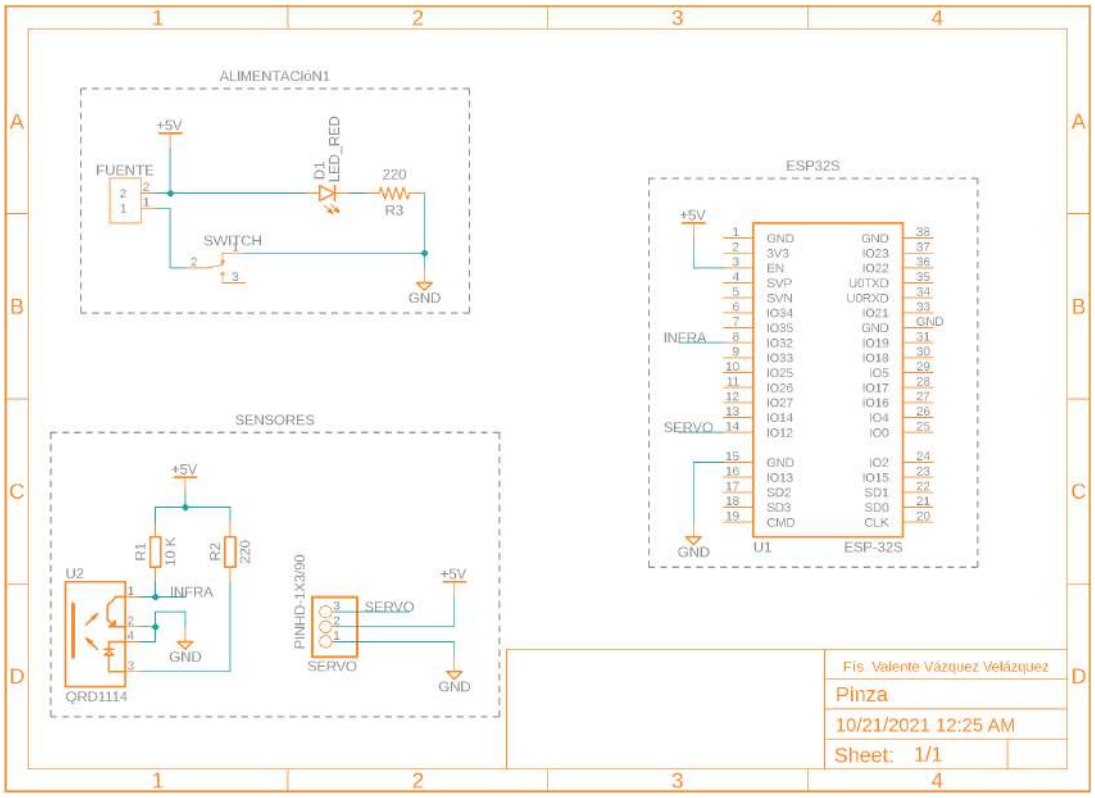


Figura 4.30: Circuito para la apertura de la pinza prototipo.

El movimiento de estos dispositivos está en función de algún actuador, como motores, servomotores o motores a pasos. En el caso particular de la pinza de la figura 4.29, su movimiento de cierre y apertura es a través de un servo motor SG90[2], por lo que diseñé un circuito (ver Fig. 4.30) donde la pinza se cierra en función de una variable recibida por medio del protocolo MQTT, además cuenta con un sensor infrarrojo modelo QRD1114[3] que tiene como función notificar al sistema si se tomó correctamente la pieza de manufactura o no.

[2]La hoja de datos técnicos del componente Servo motor SG90, puede verse en: http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf.

[3]La hoja de datos técnicos del componente QRD1114, puede verse en: https://www.onsemi.com/pdf/datasheet/qrd1114-d.pdf.

5

Resultados

"El futuro mostrará los resultados y juzgará a cada uno de acuerdo a sus logros".

– Nikola Tesla

En este capítulo presento los resultados de la puesta en marcha del SAPM, así como también un análisis general de su funcionamiento basándome en métricas aplicadas lo largo de sus múltiples pruebas.

5.1. Comunicación del SAPM

La comunicación general resultó exitosa dentro de la Raspberry Pi 4 Model B mediante el protocolo de mensajería MQTT, en la Fig. 5.1 se muestra un diagrama de bloques con los tópicos y los principales componentes integrantes de la celda. La telemetría fue ejecutada con aplicaciones externas como *MQTT Explorer* (Nordquist, 2021), de hecho, en la figura 5.2 muestro una captura de pantalla de esta aplicación mientra el SVA está en funcionamiento, pueden observarse los tópicos asociados a la BT, los primeros cuatro de la tabla 4.1.

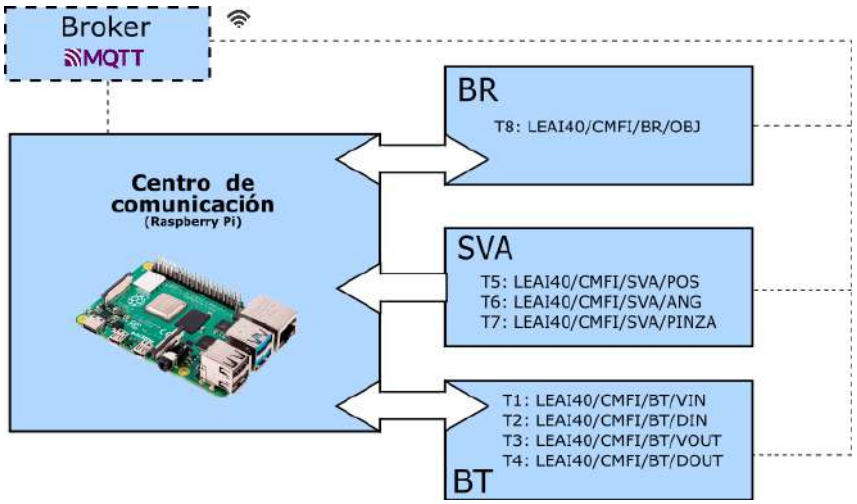


Figura 5.1: Diagrama de bloques de la comunicación dentro de la CMFI.

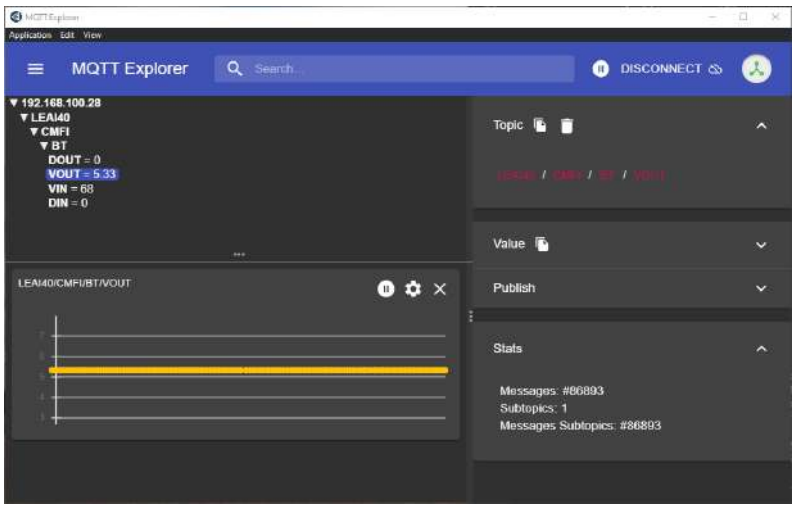


Figura 5.2: Captura de pantalla de la aplicación *MQTT Explorer* mientras está en ejecución el SVA.

5.2. Banda Transportadora

El sistema prototipo del control de velocidad de un motor DC fue ejecutado correctamente, la velocidad y la dirección de movimiento se controla de manera inalámbrica gracias al protocolo de mensajería MQTT, según se requiera a través del módulo ESP32s (véase la Fig. 5.3)

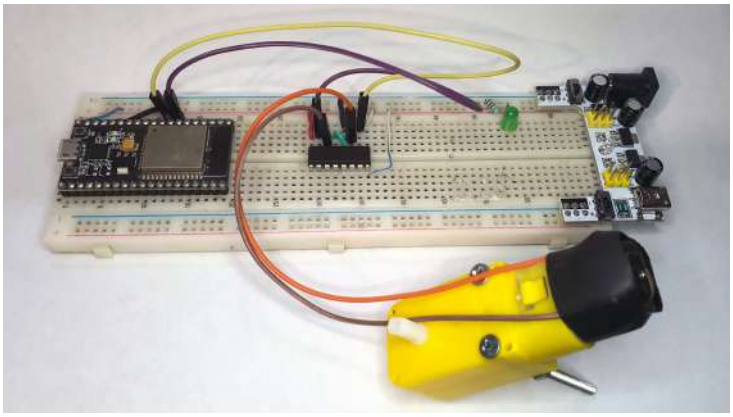


Figura 5.3: Circuito del sistema prototipo de control (4.5) de un motor de prueba conectado en una protoboard.

No presento un circuito montado en una placa final ya que este diseño solo fue un prototipo para observar, principalmente, el funcionamiento del sistema y no el de la BT en particular.

5.3. Sistema de Visión Artificial

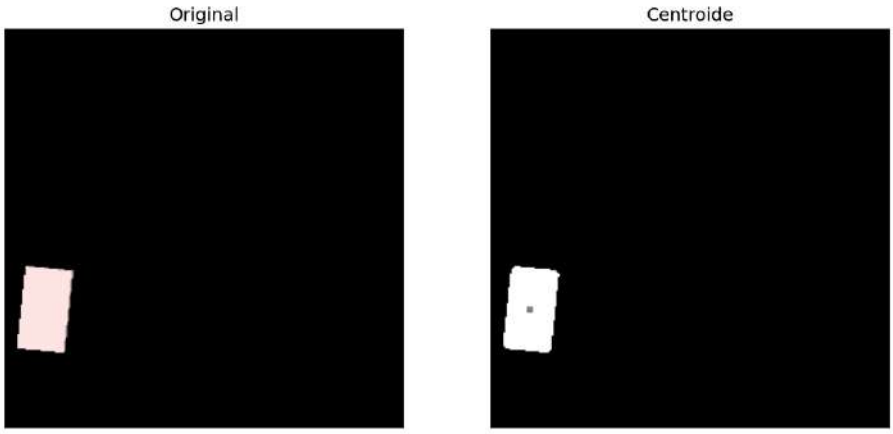
Al momento de ejecutar el SVA, se obtuvieron dos resultados, el vector característico BOF y del seguimiento por flujo óptico.

5.3.1. BOF

En el capítulo anterior, ejemplifiqué con imágenes la metodología utilizada para la obtención del vector característico BOF, este proceso lo realicé dentro de una Raspberry Pi 4 Model B (el mismo dispositivo donde implementé la comunicación de la CMF), por tanto, este sistema realiza su función sin problemas detectados; las imágenes del resultado de este proceso se muestran desde la Fig. 4.8 hasta la Fig. 4.15 del capítulo anterior.

Por otro lado, dado que utilicé sensores de visión dentro del simulador CoppeliaSim, muestro a continuación los resultados obtenidos al calcular el vector BOF. En primer lugar obtuve el único punto característico de interés en este trabajo de tesis, el centroide

la figura; en la figura 5.4(a) presento la imagen original de la pieza y en la figura 5.4(b) el centroide señalado. Por último, obtuve la gráfica del vector BOF (ver Fig. 5.5). Las demás imágenes resultantes las presento en el anexo Imágenes obtenidas de BOF.



(a) Imagen original de una pieza de manufactura del simulador.

(b) Centroide de una pieza de manufactura del simulador.

Figura 5.4: Centroide de una pieza de manufactura del simulador.

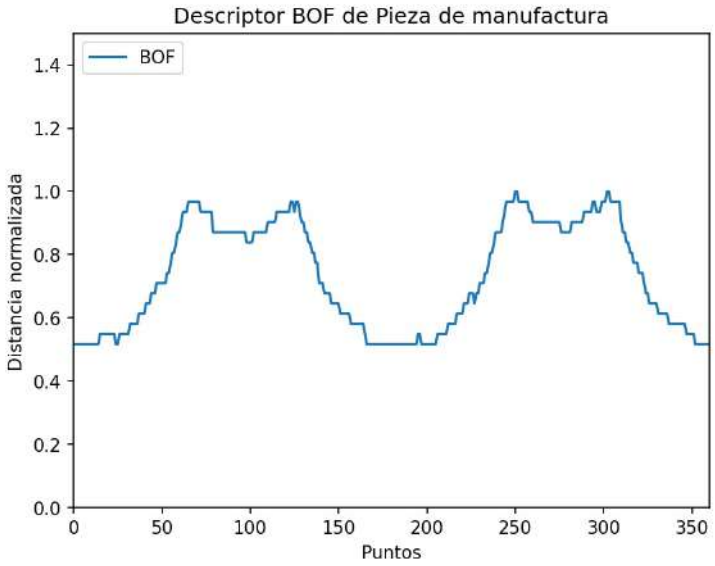


Figura 5.5: BOF resultante de la pieza de manufactura de la Fig. 5.4(a) del simulador.

Entre el sistema real (Raspberry Pi) y el virtual (CoppeliaSim) no noté diferencias significativas al obtener el vector característico BOF.

5.3.2. Flujo Óptico

De la misma forma que en el punto anterior, este procedimiento lo implementé de nueva cuenta en la Raspberry Pi, y como el seguimiento de la pieza mediante flujo óptico lo ejemplifiqué con imágenes reales, este objetivo fue llevado a cabo satisfactoriamente. En la figura 4.16 puede observarse lo comentado aquí.

Respecto al simulador, las imágenes del seguimiento de la pieza fueron enviadas en formato de lista a Python, una vez convertidas de nuevo a imágenes en formato RGB, procedí a aplicar el algoritmo de flujo óptico. Gráficamente el centroide lo represento durante el seguimiento de la pieza con un punto rojo (ver Fig. 5.6).

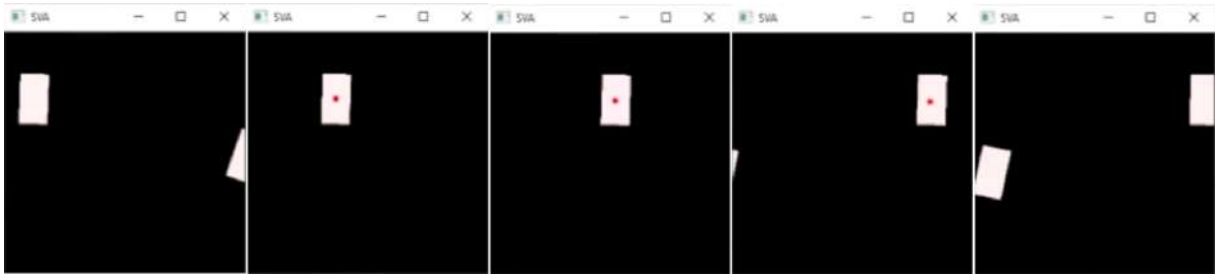


Figura 5.6: Secuencia del seguimiento por flujo óptico a una pieza de manufactura del simulador.

Una vez concluido el seguimiento, uno de los primeros resultados que entrega el SVA es la velocidad media de cada una de las piezas analizadas, dicho análisis lo represento con gráficas como las que muestro en la Fig. 5.7, los puntos azules son las posiciones obtenidas a través del flujo óptico, mientras que la línea negra representa la recta del ajuste lineal, por tanto, al obtener la pendiente de dicha recta, se obtiene la velocidad media del objeto. Nótese bien el error de la pendiente y el coeficiente de determinación r^2

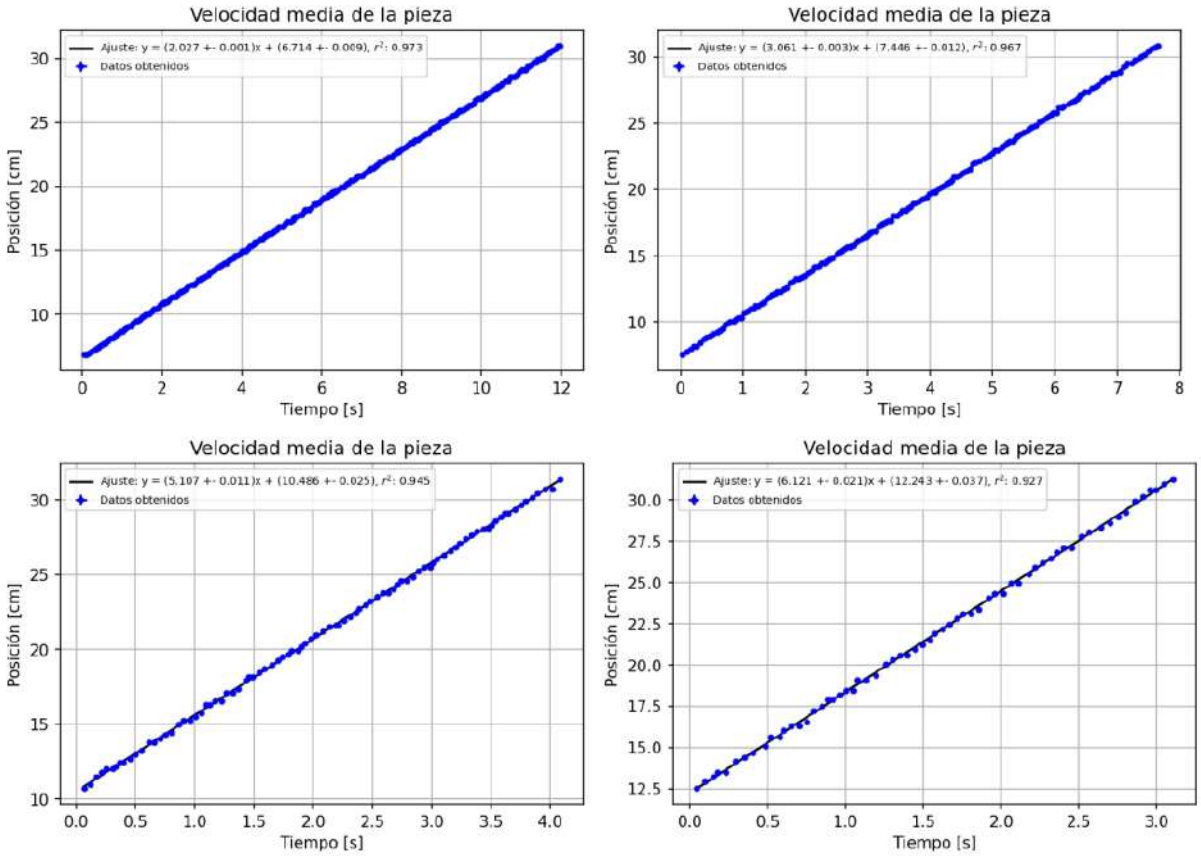


Figura 5.7: Ajuste lineal por mínimos cuadrados de la posición de una serie de piezas de manufactura.

En estas gráficas es fácil notar los siguientes puntos:

- En el eje de las abscisas se muestra el tiempo que le lleva realizar el seguimiento teniendo resultados exitosos al momento de calcular la posición de agarre, es por esto que, para una partida de 50 piezas de manufactura, muestro el movimiento de las piezas 1, 3, 6 y 25, alcanzando una velocidad máxima de funcionamiento, en este caso, de aproximadamente 6 cm/s.
- El análisis por mínimos cuadrados no solo arroja la velocidad media de cada pieza, también da dos parámetros auxiliares con la finalidad de cuantificar dicho análisis, esto para decidir sobre si el proceso fue ejecutado correctamente o no, y como consecuencia, saber si la posición de agarre calculada es precisa o no; estos datos son el error o la incertidumbre de la velocidad y el coeficiente de determinación r^2

- El coeficiente de determinación, como su nombre lo indica, determina la calidad del modelo para replicar resultados, esto lo hace un excelente parámetro para predecir futuros resultados como las futuras posiciones de la pieza de manufactura fuera del campo de visión de la cámara. Para este trabajo en particular, si su valor es superior a 0.9 el proceso será exitoso. Por otro lado, es fácil observar que este parámetro es inversamente proporcional a la velocidad media de la pieza, luego entonces, entre más rápido se mueva la BT menor es el coeficiente de correlación, haciendo que este dato también funja como un indicador de la velocidad máxima de operación de la banda.
- La incertidumbre de la velocidad describe si la precisión de la velocidad calculada es alta o no, mientras se encuentre por debajo del 0.5 %, el cálculo se toma como exitoso.

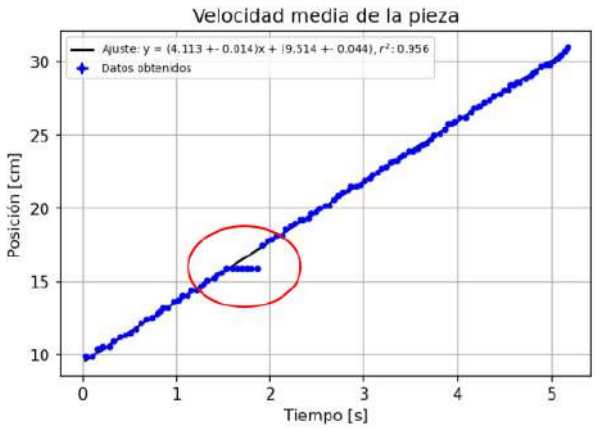


Figura 5.8: Ajuste lineal por mínimos cuadrados de la posición de una pieza de manufactura con un error en el seguimiento del objeto señalado en rojo.

Este método de mínimos cuadrados es robusto ante fallas en el seguimiento de las piezas de manufactura, una forma de comprobar esto es analizando los resultados arrojados, por ejemplo, en la Fig. 5.8 muestro uno de los tantos análisis de seguimiento por mínimos cuadrados efectuados, con la peculiaridad de que el seguimiento tuvo un error durante un pequeño lapso; como consecuencia puedo mencionar que aún con este tipo de errores

por parte del SVA al momento del escaneo de la pieza, el método de mínimos cuadrados se efectúa correctamente calculando la recta que mejor se ajusta a la nube de datos obteniendo la velocidad media de la pieza junto con su incertidumbre y el coeficiente de determinación.

Ahora bien, es importante decir que este tipo de fallas solo se presentaron al ejecutar el SVA con las imágenes provenientes del simulador, por lo que deduzco que este problema sea consecuencia del cómputo y no un error en la implementación.

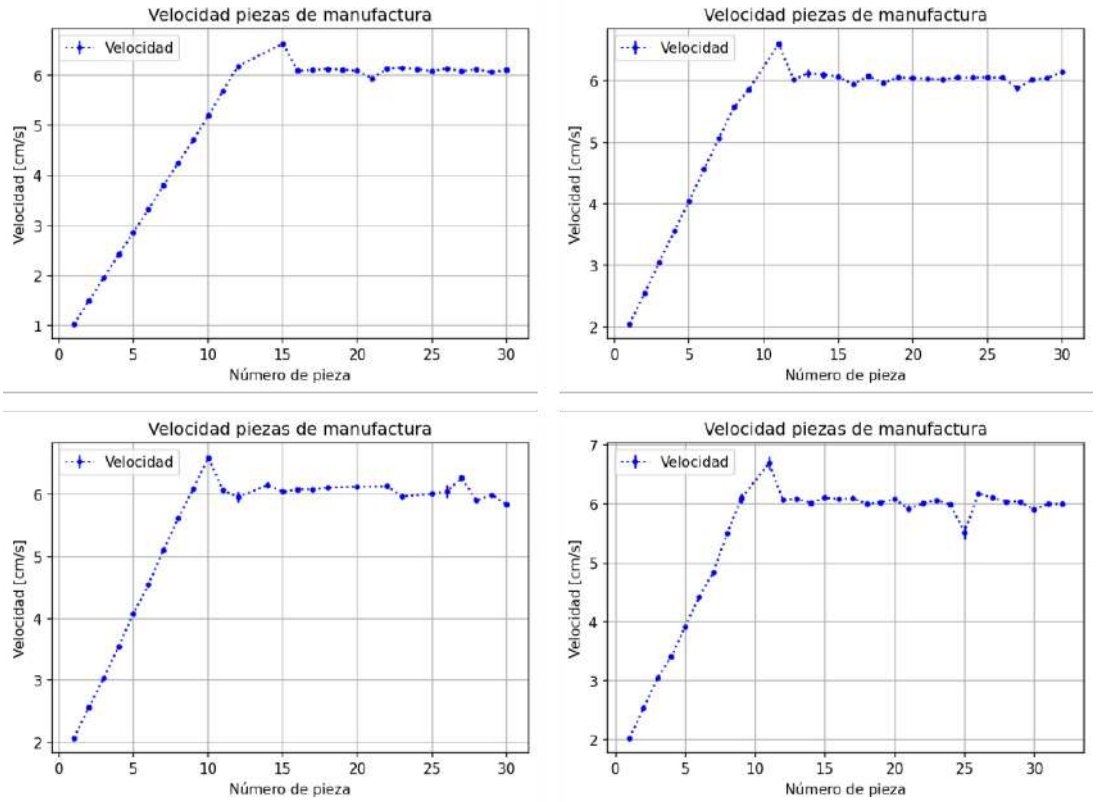


Figura 5.9: Gráficas de las velocidades de las piezas de manufactura en cuatro ejecuciones del SAPM con partidas de 50 objetos c/u.

Al ejecutar la parte correspondiente al SVA del algoritmo de la Fig. 4.23 se obtienen gráficas como las mostradas en la Fig. 5.9, en estas se muestran cuatro ejecuciones de dicho algoritmo con partidas de 30 objetos cada una, comenzando de una velocidad inicial de 2 cm/s para la primer pieza y terminando en una velocidad constante de 6 cm/s para al

menos la última mitad de las piezas. Al inicio la velocidad va aumentando una unidad hasta que los parámetros de incertidumbre de la velocidad y el coeficiente de determinación de cada pieza en cuestión indiquen al sistema si el seguimiento del objeto fue llevado a cabo correctamente o no, por lo que cuando una pieza no fue correctamente escaneada, el sistema se queda con la penúltima velocidad registrada ya de manera constante.

5.4. Implementación virtual del SAPM

A continuación, presento algunas capturas de pantalla en donde es posible observar el funcionamiento final del SAPM. El sensor de visión obtiene el centroide de la pieza junto con el vector descriptivo BOF, enseguida se realiza un seguimiento de la pieza por medio de flujo óptico y se guardan las posiciones recorridas para un posterior análisis estadístico, todo esto con la finalidad de obtener la velocidad media de la pieza y una posición en el espacio donde el brazo robótico la sujetará para llevarla y depositarla en el contenedor azul, todo con rutas en modo de líneas rectas (LIN); en caso de piezas perdidas estas se depositarán en el contenedor rojo (ver de la Fig. 5.10 hasta la Fig. 5.16). La implementación completa del SAPM dentro del simulador CoppeliaSim la presento en este vídeo: `https://youtu.be/-rzgCwq1pLQ`.

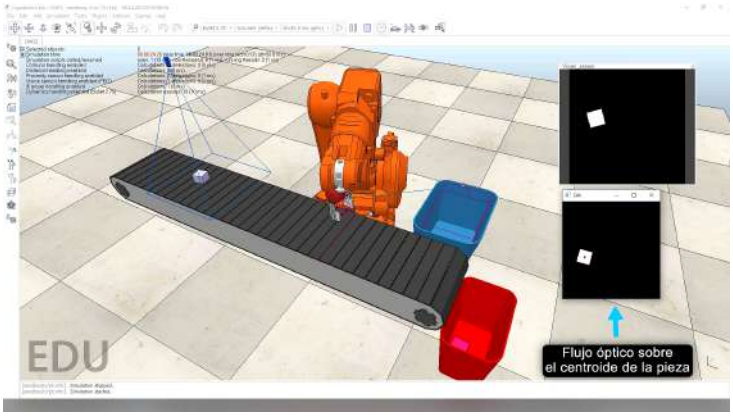


Figura 5.10: Seguimiento de la primera pieza de manufactura por parte del SVA.

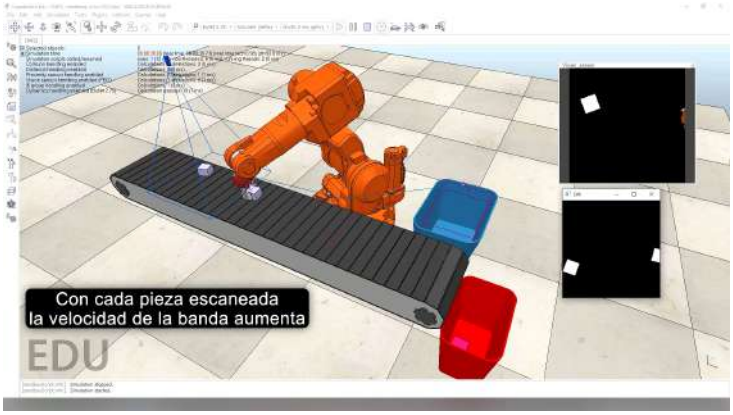


Figura 5.11: Toma de la pieza por parte del BR.

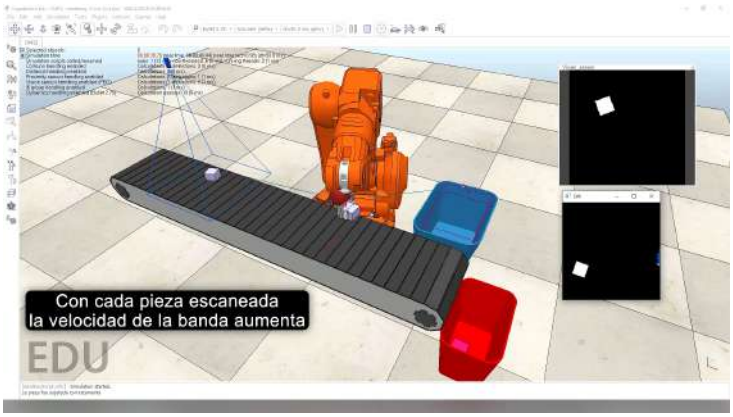


Figura 5.12: Movimiento del BR hacía el contenedor azul con la pieza sujetada.

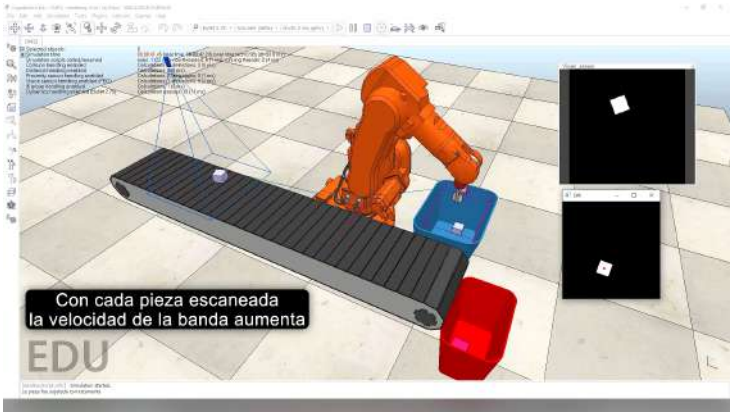


Figura 5.13: El BR deposita de la pieza en el contenedor azul.

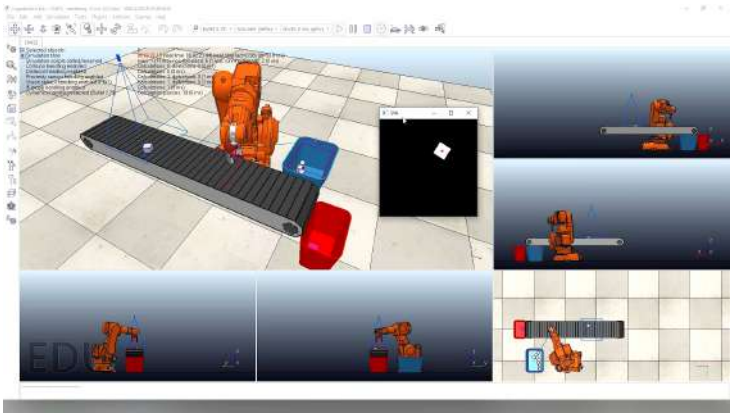


Figura 5.14: Se muestra el sistema en diferentes vistas.

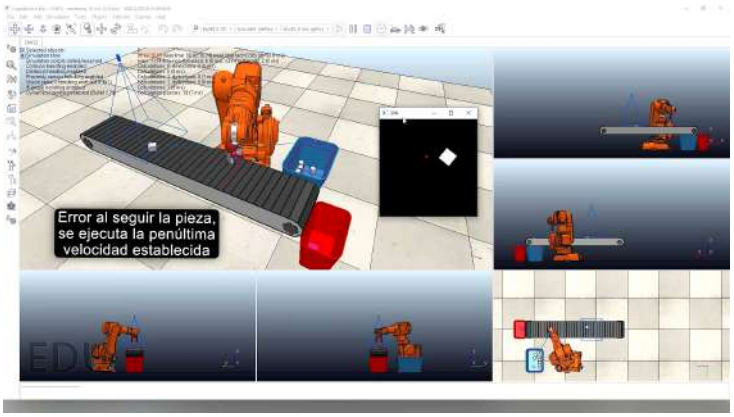


Figura 5.15: Error en el seguimiento de la pieza de manufactura.

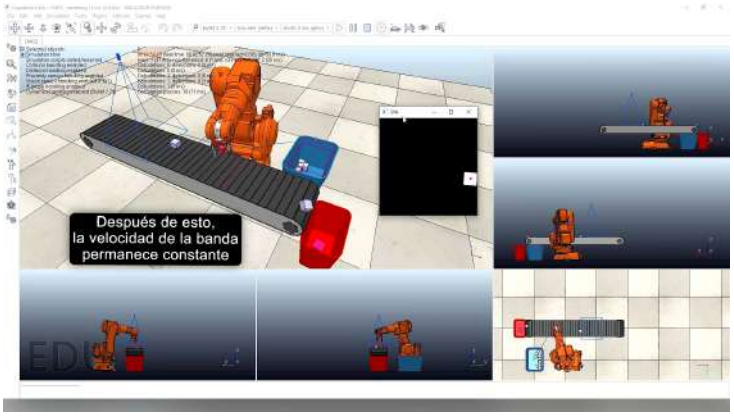


Figura 5.16: Depósito de la pieza erróneamente seguida en el contenedor rojo.

Para que el proceso fuera exitoso y hasta cierto punto cómodo de trabajar, fue necesario realizar una calibración de las coordenadas del robot, que para este caso lo realicé con un sistemas de coordenadas auxiliar como el mostrado en la Fig. 5.17, en ella se puede observar que el origen del sistema de referencia queda justo al final del campo de visión de la cámara sobre la BT, tal como la muestro a continuación:

$$\begin{aligned} x &= 0.2 + (x/100) \\ y &= -0.671 + (y/100) \end{aligned}$$

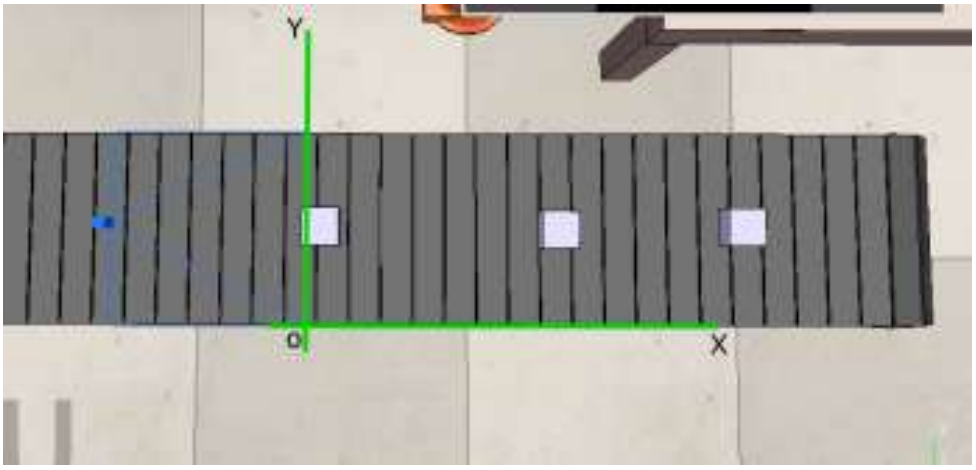


Figura 5.17: Calibración entre las coordenadas del BR y el SVA.

Respecto a las velocidades máximas a las que el SAPM opera, puedo hacer notar lo siguiente:

- Los procesos abiertos de la computadora donde se está ejecutando el simulador.
  Dependiendo de los procesos que la computadora estuviera efectuando al momento de lanzar la simulación, afectaban en el rendimiento de esta. El hecho más notorio era que cuando grababa las puestas en marcha del sistema, su rendimiento no era el mismo comparado a cuando no se grababa la pantalla.
- La vecindad del análisis del flujo óptico.
  El algoritmo Lucas-Kanade revisa en una vecindad de los puntos de interés los cambios que han ocurrido de un cuadro del video respecto a su predecesor, el tamaño

de esta vecindad es inversamente proporcional al rendimiento de la velocidad máxima de ejecución del SAPM. Esto se notó en el tipo de pieza que circulaba sobre la banda, entre más grande o irregular fuera, la velocidad máxima de funcionamiento disminuía, es por lo anterior que los mejores resultados del sistema se obtuvieron mientras las piezas de manufactura fueran cubos de no más de 6 cm de lado; la desventaja de utilizar solo cubos es que no hay una observación clara si la pinza que toma estos objetos gira de acuerdo al ángulo calculado. Mientras se utilizaron cubos, el sistema trabajo con velocidades máximas de funcionamiento de hasta 12 cm/s, mientras que con prismas rectangulares no sobrepasó los 7 cm/s (ver Fig. 5.18).

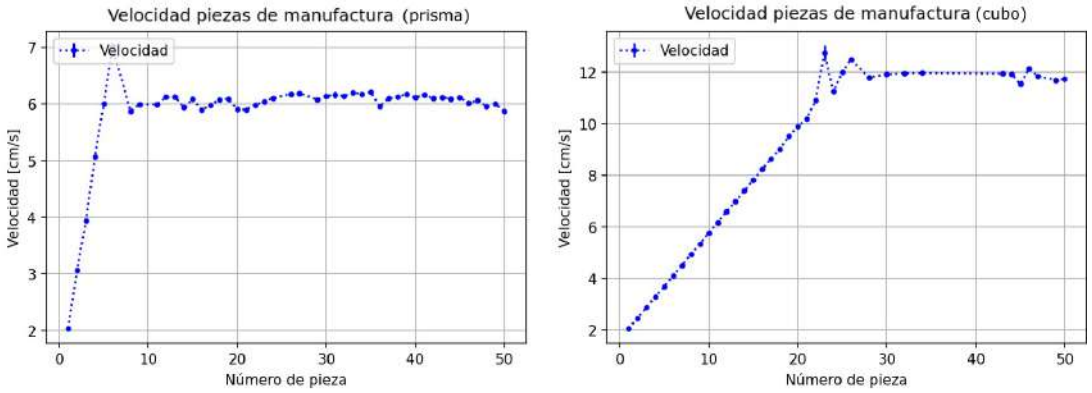


Figura 5.18: Gráficas de la velocidad máxima de funcionamiento del sistema.

5.5. Implementación física del SAPM

La migración del sistema virtual al físico no pude concluirla al 100 % debido a las restricciones sanitarias que siguen existiendo alrededor del mundo a causa de la pandemia provocada por el virus SARS-CoV-2. Durante gran parte del 2020 y 2021 el IIMAS cerró a todo público, sin embargo, en cuanto se pudo volver a trabajar el esquema fue por citas dando prioridad a las investigaciones y tesis doctorales, por lo que tuve en este periodo pandémico acceso al LEAI 4.0 solamente en tres ocasiones.

En estas visitas identifiqué el alcance que podía tener el trabajo dentro del laboratorio, el sistema completo tal como se muestra en el simulador era prácticamente imposible de lograrlo, principalmente por las siguientes situaciones:

- No se realizó la compra de algunas de las cámaras que se anunciaron en el capítulo 2, por lo que hacer una estructura para colocarla cenit a la banda era un caso inviable.
- La actualización del controlador de la banda llevaría más tiempo del que se contaba.
- Y por último, no se contaba con un efector final para sujetar piezas, sin embargo, lo que sí realicé fue el circuito necesario para su posterior integración a la pinza de agarre, (ver Fig. 5.19); las pruebas realizadas en él fueron exitosas, por medio del protocolo MQTT fue posible mover el servo motor y recibir información del sensor infrarrojo.

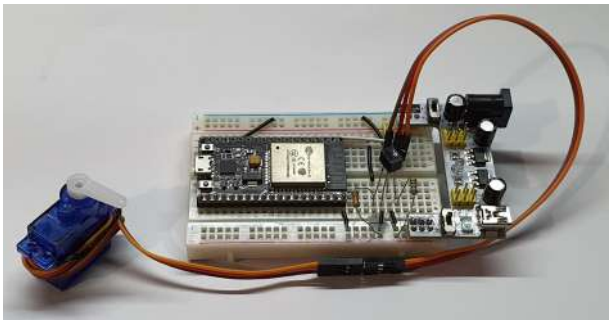


Figura 5.19: Circuito prototipo para el uso y comunicación del efector final mostrado en el diagrama de la Fig. 4.30.

La solución más certera fue combinar el sistema virtual con el físico, es decir, poner en marcha el SVA dentro del simulador y enviar las posiciones de agarre tanto al BR Kuka como al BR virtual; este sistema emergente no sujetaría la pieza, pero sí podrían ambos brazos recorrer las mismas rutas programadas.

El primer paso fue realizar otra simulación partiendo de la primera, donde los dos cambios significativos fueron: quitar el contenedor azul y crear rutas que el robot debía recorrer para efectuar un seguimiento de la pieza junto con su movimiento al punto de inicio (ver Fig. 5.20).

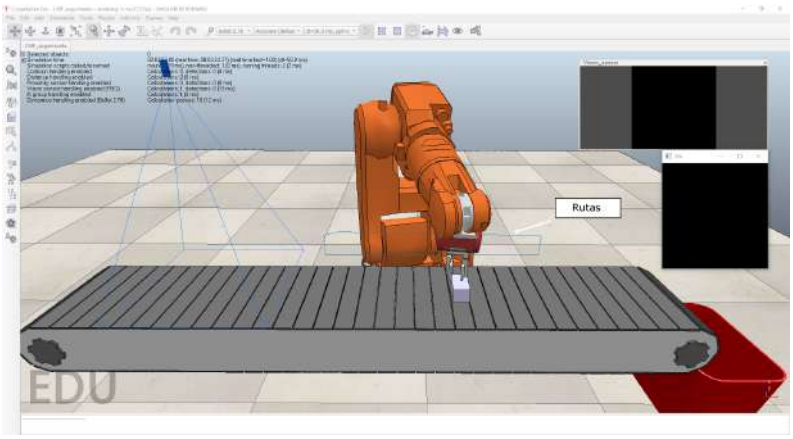


Figura 5.20: Seguimiento de una pieza de manufactura por parte del BR en el simulador.

Asimismo, programé desde el pad del brazo Kuka una rutina igual, donde recorrerá la banda señalando la pieza durante su trayecto y de regreso a su punto de origen, todo esto a partir del envío de la posición de agarre por parte del SVA ejecutado en el simulador; el código se muestra en el anexo Códigos en el Pad Kuka. Por otro lado, los cambios dentro del programa principal hecho en Python (ver Códigos en Python) fueron principalmente en la función moveArm(), quedando de la siguiente manera:

```
from kukavarproxy import *

def moveArm(pos):
  posSIM = []
  posKUKA = []

  #Ajuste al sistemas de coordenadas de la BT del simulador
  posSIM[0] = 0.25 + (pos[0]/100)
  posSIM[1] = -0.671 + (pos[1]/100)
  posSIM[5] = pos[5]*pi/180
  p = str(posSIM[0])+" "+str(posSIM[1])+" "+str(pos[2])+" "+str(pos[3])+" "
   +str(pos[4])+" "+str(posSIM[5])

  #Envio de coordenadas de garre al BR del simulador
  sim.simxCallScriptFunction(clientID,
```

```
                "ConveyorBelt",
                sim.sim_scripttype_childscript,
                "posTarget",[],[],[p],"",
                sim.simx_opmode_blocking)

  #Ajuste al sistemas de coordenadas de la BT fisica
  posKUKA[0] = 975 - (pos[0]/1000)
  posKUKA[1] = 571 + (pos[1]/1000)
  posKUKA[2] = 700
  posKUKA[5] = pos[5]

  #Envio de coordenadas de agarre al BR KUKA
  robot.write("UBICACION.X", posKUKA[0])
  robot.write("UBICACION.Y", posKUKA[1])
  robot.write("UBICACION.Z", posKUKA[2])
  robot.write("UBICACION.A", pos[5])
  robot.write("UBICACION.B", pos[4])
  robot.write("UBICACION.C", pos[3])
  robot.write("STATE", 1)
  time.sleep(1)
  robot.write("STATE", 0)

robot = KUKA('172.31.2.147')
time.sleep(5)

##Ejecucion de BOF

##Ejecucion del flujo optico

##Obtencion de la posicion de agarre: pos

moverArm(pos)
```

El resultado de esta implementación híbrida se muestra en las capturas de pantalla mostradas de la Fig. 5.21 a la Fig. 5.25. En estas imágenes se puede notar que ambos brazos se mueven en coordinación y a la misma velocidad. Con esto puedo mencionar que la comunicación entre el SVA ejecutado en mi computadora personal y el sistema del robot Kuka fue llevada de manera satisfactoria. Esta implementación híbrida la presento en este vídeo: https://youtu.be/FEXknOHatPc.

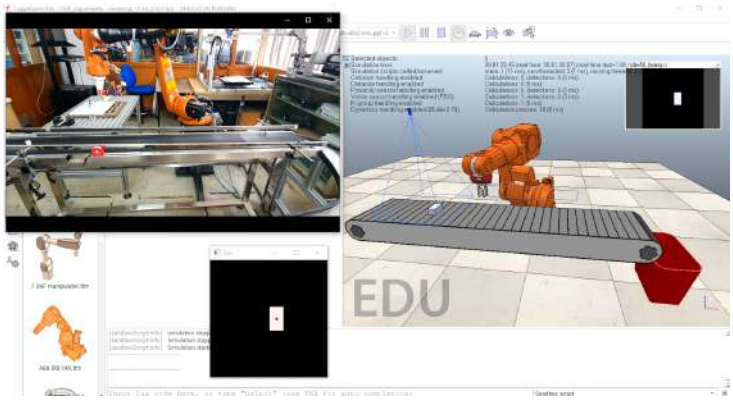


Figura 5.21: Seguimiento de la primera pieza de manufactura por parte del SVA.

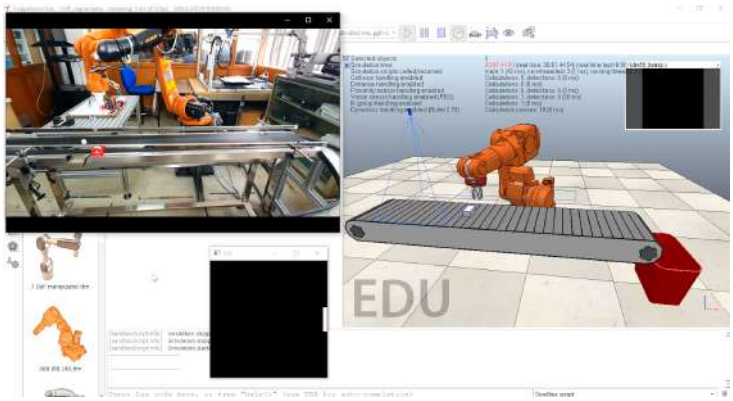


Figura 5.22: Comienzo del seguimiento de la pieza de manufactura por parte del BR.

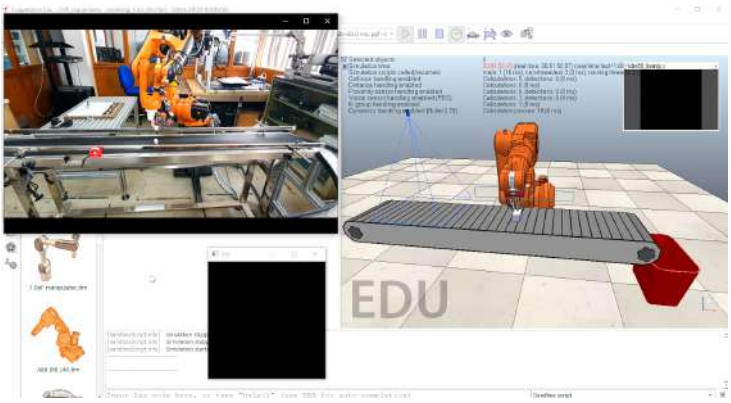


Figura 5.23: Seguimiento de la pieza de manufactura por parte del BR.

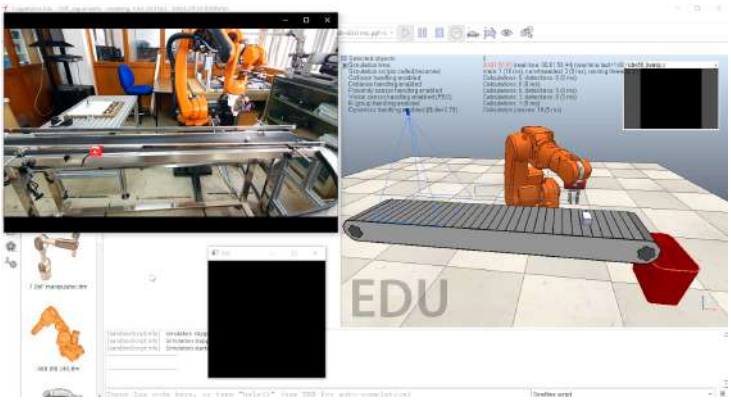


Figura 5.24: Fin del seguimiento de la pieza de manufactura por parte del BR.

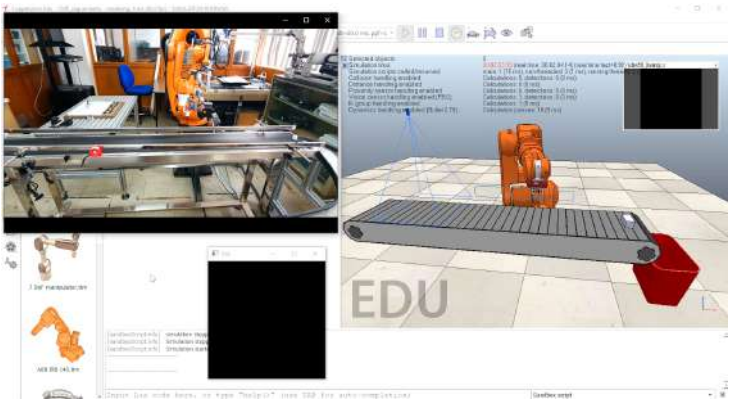


Figura 5.25: El brazo regresando a la posición de origen.

6

Conclusiones

"No basta tener un buen ingenio, lo principal es aplicarlo bien".

– René Descartes

En este trabajo de tesis implementé de manera satisfactoria un sistema encargado de analizar objetos de geometría simple, a los que llamé piezas de manufactura, que acarrea una banda transportadora con el fin de identificar su centroide, obtener su velocidad media y por último, sujetar con un brazo robótico para depositar la pieza en cuestión dentro de un contenedor. Este trabajo surgió como una necesidad de implementar una celda manufactura flexible inteligente dentro del Laboratorio de Electrónica y Automatización para la Industria 4.0 que se encuentra en el Instituto de Investigaciones en Matemáticas Aplicadas y Sistemas de la Universidad Nacional Autónoma de México.

La comunicación interna del sistema, basada en el protocolo MQTT, demostró ser robusta, funcional y sencilla de implementar dentro de los dispositivos utilizados en este trabajo. Resultó en una buena práctica implementar el bróker de este protocolo dentro del mismo dispositivo donde se realizó físicamente el SVA.

Hasta ahora, durante las pruebas realizadas del SVA, puedo decir que este sistema se comporta de manera robusta ante fallas ya que no se han presentado inconvenientes, ni cuando ocurren las siguientes circunstancias:

- Al no lograr un correcto seguimiento de la pieza de manufactura
- Al tener piezas que no están totalmente apoyadas sobre la banda
- En cualquier orientación de la pieza
- Al presentarse más de una pieza sobre el campo de visión de la cámara, siempre y cuando no se esté llevando a cabo el proceso para obtener el vector BOF.

De manera general, el SAPM desarrollado en este trabajo se basa en algoritmos de visión computacional fuertemente optimizados, funcionales y robustos: BOF y flujo óptico por Lucas-Kanade, esto hace que el sistema herede las mismas características, siendo así una excelente solución, y de contribución original, para los sistemas *pick and place* dentro de Industria 4.0.

El principal trabajo a futuro definitivamente es finalizar la implementación física dentro del LEAI 4.0 una vez que las condiciones sanitarias lo permitan, comenzado así con el desarrollo de una CMFI dentro del LEAI 4.0. Una opción más sería volcar el algoritmo principal a una arquitectura electrónica específica, como los es un FPGA, esto daría ventajas al momento de computar haciendo que la velocidad de operación del SAPM crezca un porcentaje significativo, ya que pasaría de un lenguaje interpretado a un lenguaje máquina. Por último, una aplicación posible es identificar con ayuda de diversos métodos de inteligencia artificial, las piezas que van circulando por la BT, y es aquí donde el vector característico BOF toma relevancia, ya que este elemento matemático sería indispensable para las etapas de entrenamiento y ejecución de cual sea el método computacional que se utilice.

Sin duda, el ser humano atraviesa una nueva etapa en su paso por el planeta, una etapa llena de máquinas, circuitos, computadoras que toman decisiones por sí solas y con una enorme red al alcance de las manos con los conocimientos que nuestros antepasados nunca se imaginaron. Este es el inicio de una nueva era, un era donde ya será inexplicable no convivir con elementos electromecánicos que tienen como objetivo entretenernos y hacernos la vida mucho más fácil, y no solo como individuos sino como pueblo, e incluso,

como especie.

Bibliografía

Aguirre, A., Suarez, F., Bernal, G., and Bustamante, C. (2013). Celda de manufactura flexible. In *4ta Feria Mecastrónica, Instituto Tecnológico de Culiacán.* (Citado en página 12.)

Allaoui, R., Mouane, H. H., Asrih, Z., Mars, S., El Hajjouji, I., and El mourabit, A. (2017). Fpga-based implementation of optical flow algorithm. *2017 International Conference on Electrical and Information Technologies (ICEIT).* (Citado en página 15.)

Alonso, M. and Finn, E. J. (1970). *Física, Vol. I Mecánica.* Fondo Educativo Interamericano S.A. (Citado en página 38.)

AranaCorp (2020). *Crear una interfaz web para controlar su ESP32 NodeMCU.* Obtenido de: https://www.aranacorp.com/es/crear-una-interfaz-web-para-controlar-su-esp32-nodemcu/. (Citado en página 151.)

Araujo, W. F. S., Silva, F. J. G., Campilho, R. D. S. G., and Matos, J. A. (2016). Manufacturing cushions and suspension mats for vehicle seats: a novel cell concept. *The International Journal of Advanced Manufacturing Technology*, 90(5-8):1539–1545. (Citado en página 11.)

Arif, O., Marshall, M., Daley, W., Vela, P., Teizer, J., Ray, S., and Stewart, J. (2010).

Tracking and classifying objects on a conveyor belt using time-of-flight camera. *2010 - 27th International Symposium on Automation and Robotics in Construction, ISARC 2010*. (Citado en página 14.)

Atienza Vanacloig, V. L. (2011). El histograma de una imagen digital. Obtenido de: https://riunet.upv.es/handle/10251/12711. (Citado en página 32.)

Atthariq, L. (2021). *Gripper servo sg90*. Obtenido de: https://grabcad.com/library/gripper-servo-sg90-1. (Citado en página 77.)

Baird, D. C. (1991). *Experimentación. Una introducción a la teoría de mediciones y al diseño de experimentos*. Prentice Hall Hispanoamericana S.A. (Citado en página 38.)

Barghash, M. A., Al-Mandahawi, N., AbuJbara, N., Al-Abbadi, R., and Hussein, S. (2017). Design of manufacturing cells pharmaceutical factory. *Journal of Applied Research on Industrial Engineering*, 4(3):158–173. (Citado en página 13.)

Bozejko, W., Pempera, J., and Wodecki, M. (2018). Minimization of the number of employees in manufacturing cells. *Advances in Intelligent Systems and Computing*, pages 241–248. (Citado en página 11.)

Brading, M. (2020). *3-D Sensors Bring Depth Discernment to Embedded Vision Designs*. Obtenido de: https://www.edge-ai-vision.com/2013/07/3-d-sensors-bring-depth-discernment-to-embedded-vision-designs/. (Citado en páginas 17 y 18.)

Chen, L., Wu, Y., Du, Z., Tao, T., and Zhao, F. (2017). Development of an industrial robot controller with open architecture. In *2017 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, pages 754–757. (Citado en página 17.)

CIC, C. I. (2019). *Industria 4.0, la cuarta revolución industrial y la inteligencia operacio-*

nala. Obtenido de: https://www.cic.es/industria-40-revolucion-industrial/. (Citado en página xxv.)

Coppelia-Robotics (2020). *Remote API funvtions (Python)*. Obtenido de: https://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsPython.htm. (Citado en página 71.)

Coppelia-Robotics (2021a). Coppeliasim. Obtenido de: https://www.coppeliarobotics.com/. (Citado en páginas 40 y 41.)

Coppelia-Robotics (2021b). Inverse kinematics tutorial. Obtenido de: https://www.coppeliarobotics.com/helpFiles/en/inverseKinematicsTutorial.htm. (Citado en páginas 44 y 72.)

Corvo, T. S. (2019). *Automatización industrial: historia, características y tipos*. Obtenido de: https://www.lifeder.com/automatizacion-industrial/. (Citado en página 8.)

Cubero, S. (2007). *Industrial robotics: theory, modelling and control*. Pro-Literatur-Verl. (Citado en página 13.)

Drath, R. and Horch, A. (2014). Industrie 4.0: Hit or hype? [industry forum]. *IEEE Industrial Electronics Magazine*, 8(2):56–58. (Citado en página xxv.)

ElLinux.org (2021). *Rpi Camera Module*. Obtenido de: https://elinux.org/Rpi_Camera_Module. (Citado en páginas 153 y 154.)

Espressif (2021). *ESP32 Series Datasheet*. Obtenido de: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. (Citado en páginas 149 y 151.)

Fu, K., González, R. C., and Lee, C. S. G. (1990). *Robótica, control, detección, visión e inteligencia*. McGraw-Hill. (Citado en página 43.)

Giménez Palomares, F., Monsoriu Serrá, J. A., and Alemany Martínez, E. (2016). Aplicación de la convolución de matrices al filtrado de imágenes. *Modelling in Science Education and Learning*, 9(1):97. (Citado en página 29.)

Global, I. (2020). *What is RGB-D Camera (Depth Sensor)*. Obtenido de: https://www.igi-global.com/dictionary/human-motion-analysis-and-simulation-tools/50427. (Citado en página 17.)

Gobierno de España, M. d. E. (2012). *Aplicación práctica de la visión artificial en el control de procesos industriales*. Obtenido de: http://visionartificial.fpcat.cat/wp-content/uploads/UD_1_didac_Conceptos_previos.pdf. (Citado en página 11.)

Gonzalez, R. C. and Woods, R. E. (2018). *Digital image processing*. Pearson. (Citado en páginas 30 y 31.)

Granja, M. (2014). *Modelación y Análisis de la Cinemática Directa e Inversa del Manipulador Stanford de Seis Grados de Libertad.* Escuela Politécnica Nacional, Facultad de Ingeniería Mecánica. (Citado en página 42.)

Gultekin, G. K. and Saranli, A. (2013). An fpga based high performance optical flow hardware design for computer vision applications. *Microprocessors and Microsystems*, 37(3):270–286. (Citado en página 15.)

Gómez, H. (2019). Manejador del control de velocidad de banda transportadora en una celda de manufactura en el contexto de la "Internet Industrial de las cosas". In *SOMI, XXXIV Congreso de Instrumentación.* (Citado en página 53.)

Huang, J., Zhang, F., Dong, X., Yang, R., Xie, J., and Shang, W. (2020). Vision-guided dynamic object grasping of robotic manipulators. In *2020 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 460–465. (Citado en página 16.)

IPN (2013). *Industria 4.0*. Obtenido de: https://e4-0.ipn.mx/industria-4-0/. (Citado en página 19.)

Irani, S. A. (1999). *Handbook of cellular manufacturing systems.* John Wiley And Sons, 1 edition. (Citado en página 12.)

ITAM (2019). *Laboratorio de Celdas de Manufactura Robotizada.* Obtenido de: http://industrialyoperaciones.itam.mx/es/1/paginas/laboratorio-de-celdas-de-manufactura-robotizada. (Citado en página 8.)

Kaeli, D. R. (2015). *Heterogeneous computing with OpenCL 2.0.* Elsevier, Morgan Kaufmann. (Citado en página 32.)

Khan, U. S., Iqbal, J., and Khan, M. A. (2005). Automatic inspection system using machine vision. In *34th Applied Imagery and Pattern Recognition Workshop (AIPR'05)*, pages 6 pp.–217. (Citado en página 9.)

Konukseven, E. and Kaftanoglu, B. (2000). Robot end-effector based sensor integration for tracking moving parts. In *KES'2000. Fourth International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies. Proceedings (Cat. No.00TH8516)*, volume 2, pages 628–634 vol.2. (Citado en página 16.)

KUKA (2015a). Kr 5 arc h. Obtenido de: https://www.kuka.com/-/media/kuka-downloads/imported/6b77eecacfe542d3b736af377562ecaa/pf0012_kr_5_arc_hw_en.pdf. (Citado en páginas 154, 155 y 156.)

KUKA (2015b). Kuka system software 8.3. operating and programming instructions for system integrators. Obtenido de: https://www.kuka.com/. (Citado en páginas 45, 46, 47 y 65.)

Lapointe, P. (2021). *MQTT and Wifi handling for ESP8266 and ESP32.* Obtenido de: https://github.com/plapointe6/EspMQTTClient. (Citado en página 56.)

Liker, J. K. (2004). *The Toyota way.* McGraw-Hill. (Citado en página 9.)

Lin, Z., Zeman, V., and Patel, R. V. (1989). On-line robot trajectory planning for catching a moving object. In *Proceedings, 1989 International Conference on Robotics and Automation*, pages 1726–1731 vol.3. (Citado en página 9.)

Logicbus (2019). *La Industria 4.0 a través de sus características y cómo funciona*. Obtenido de: https://www.logicbus.com.mx/que-es-la-industria-4-0.php. (Citado en página 10.)

Lucas, B. D. and Kanade, T. (1981). Iterative image registration technique with an application to stereo vision. In *Conference: Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*, volume 2, pages 674–679. cited By 6530. (Citado en página 37.)

Lómas, V. (2016). *Tesis Doctoral: Aprendizaje y reconocimiento invariante de objetos en ensamble con robots empleando redes neuronales artificiales implementadas con FPGA*. Universidad Nacional Autónoma de México. (Citado en página 17.)

Microsoft (2021). *Protocolos y Tecnologías de IoT*. Obtenido de: https://azure.microsoft.com/es-mx/overview/internet-of-things-iot/iot-technology-protocols/. (Citado en páginas 20 y 23.)

Mirrazavi Salehian, S. S., Figueroa Fernandez, N. B., and Billard, A. (2017). Dynamical system-based motion planning for multi-arm systems: Reaching for moving objects. *International Joint Conference on Artificial Intelligence, Melbourne, Australia,*. (Citado en página 10.)

Mosquitto (2021). *Eclipse Mosquitto*. Obtenido de: https://mosquitto.org/. (Citado en página 51.)

MQTT (2021). Mqtt. Obtenido de: https://mqtt.org/. (Citado en página 26.)

Nashat, S., Abdullah, A., Aramvith, S., and Abdullah, M. (2011). Support vector machine

approach to real-time inspection of biscuits on moving conveyor belt. *Computers and Electronics in Agriculture*, 75(1):147 – 158. (Citado en página 15.)

Nordquist, T. (2021). *MQTT Explorer*. Obtenido de: http://mqtt-explorer.com/. (Citado en página 79.)

OpenCV (2021a). Color conversions. Obtenido de: https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html. (Citado en página 28.)

OpenCV (2021b). Optical flow. Obtenido de: https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html. (Citado en páginas 36 y 37.)

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66. (Citado en página 32.)

Pardo-Castellote, G., Schneider, S. A., and Cannon, R. H. (1995). System design and interfaces for intelligent manufacturing workcell. In *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, volume 1, pages 1105–1112 vol.1. (Citado en páginas 9 y 10.)

Park, H.-M., Van Messem, A., and De Neve, W. (2018). Box-scan: An efficient and effective algorithm for box dimension measurement in conveyor systems using a single rgb-d camera. *The 7th IIAE International Conference on Industrial Application EngineeringAt: Kitakyushu, Japan.* (Citado en páginas 14 y 15.)

Park, H.-m., Van Messem, A., and De Neve, W. (2020). Item measurement for logistics-oriented belt conveyor systems using a scenario-driven approach and automata-based control design. *2020 IEEE 7th International Conference on Industrial Engineering and Applications (ICIEA).* (Citado en página 15.)

Peña-Cabrera, M. (2005). *Aprendizaje y reconocimiento invariante de objetos en ensamble con robots empleado redes neuronales artificiales.* PhD thesis, Centro de Ingeniería y Desarrollo Industrial. (Citado en página 27.)

RaspberryPi (2020). *Raspberry Pi 4 Tech Specs*. Obtenido de: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/. (Citado en páginas 152 y 153.)

ROS-Industrial (2020). *A Comprehensive List of 3D Sensors Commonly Leveraged in ROS Development*. Obtenido de: https://rosindustrial.org/3d-camera-survey. (Citado en página 18.)

Ruiz, L., Torres, M., Gómez, A., Díaz, S., González, J. M., and Cavas, F. (2020). Detection and classification of aircraft fixation elements during manufacturing processes using a convolutional neural network. *Applied Sciences (2076-3417)*, 10(19):6856. (Citado en página 17.)

Sanfilippo, F., Hatledal, L. I., Zhang, H., Fago, M., and Pettersen, K. Y. (2015). Controlling kuka industrial robots: Flexible communication interface jopenshowvar. *IEEE Robotics Automation Magazine*, 22(4):96–109. (Citado en página 65.)

Sharma, C. and Gondhi, D. N. K. (2018). Communication protocol stack for constrained iot systems. *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*, pages 1–6. (Citado en página 21.)

TAPYC, C. T. (2019). *La historia de las cintas transportadoras*. Obtenido de: http://www.cintastransportadorastapyc.com/la-historia-de-las-cintas-transportadoras/. (Citado en página 8.)

Valvano, J. W. (2013). *Embedded Systems*. Jonathan W. Valvano. (Citado en página 10.)

Wang, L., Wang, Z., Yu, H., Ha, H., Kim, Y., and Lee, J. (2015). Vision based robot for recognizing and grasping fast moving conveyor products. In *2015 15th International Conference on Control, Automation and Systems (ICCAS)*, pages 1211–1215. (Citado en página 13.)

Zhihong, C., Hebin, Z., Yanbo, W., Binyan, L., and Yu, L. (2017). A vision-based robotic grasping system using deep learning for garbage sorting. In *2017 36th Chinese Control Conference (CCC)*, pages 11223–11226. (Citado en páginas 15 y 16.)

Anexos

Imágenes obtenidas de BOF

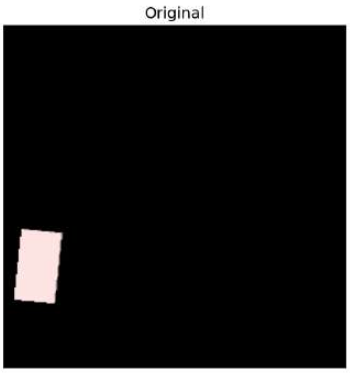


Figura A1: Imagen original de una pieza de manufactura del simulador.

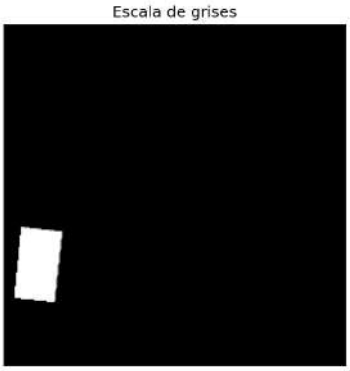


Figura A2: Imagen en escala de grises de la Fig. A1.

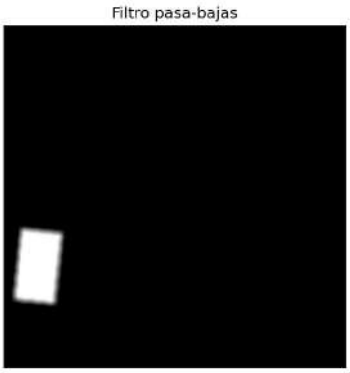


Figura A3: Imagen con filtro pasa bajas de la Fig. A2.

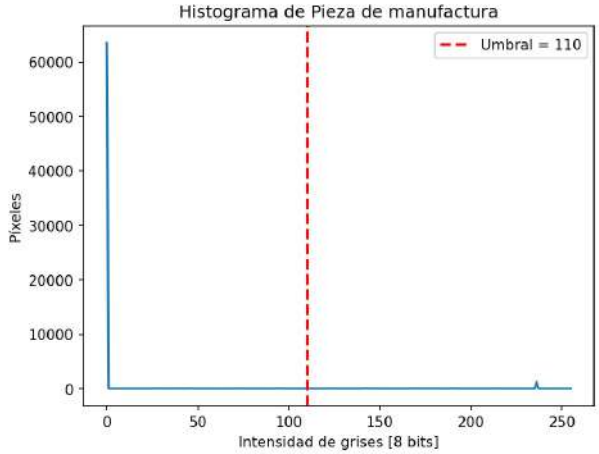


Figura A4: Histograma de la Fig. A3.

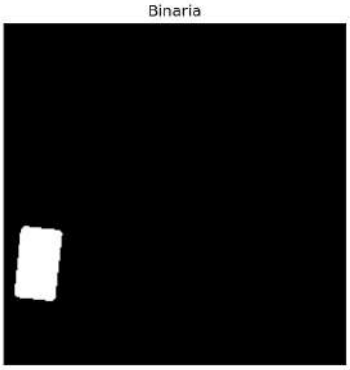


Figura A5: Imagen binaria de la Fig. A3.

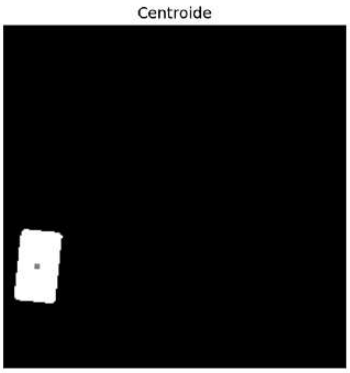


Figura A6: Centroide señalado de la pieza de la Fig. A5.

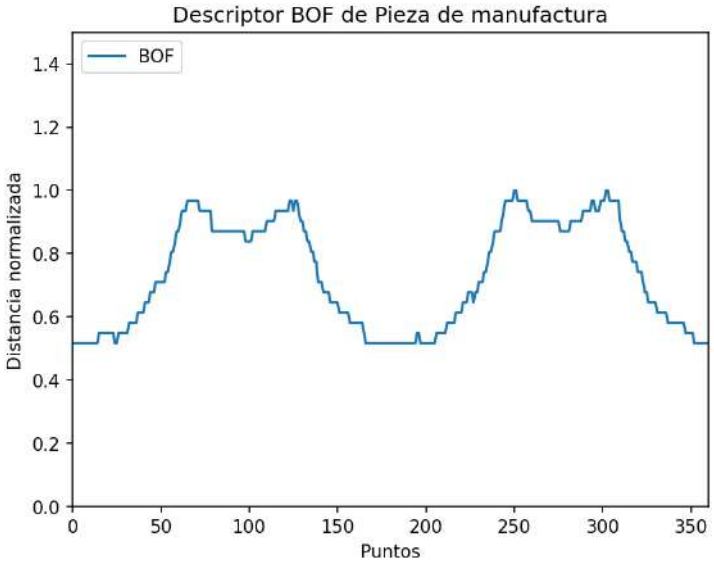


Figura A7: BOF resultante de la pieza de manufactura de la Fig. A1 del simulador.

Códigos en Lua

Banda transportadora:

```
--Non-threaded child script

function sysCall_init()
--Inicio comunicacion con Python
```

```
simRemoteApi.start(19999)

--Velocidad por defecto de la banda en m/s
beltVelocity = 0.15

--Manejador de la Banda
BeltpathHandle = sim.getObjectHandle("ConveyorBeltPath")

--Manejador del dummy target
targetDummy = sim.getObjectHandle('target')
idlePos = sim.getObjectPosition(targetDummy,-1)
idleOrient = sim.getObjectOrientation(targetDummy,-1)

--Manejador del dummy TakenPos
takenPosHandle = sim.getObjectHandle("TakenPos")
takenPos = sim.getObjectPosition(takenPosHandle,-1)
takenOrient = sim.getObjectOrientation(takenPosHandle,-1)

--Manejador del dummy ReleasePos
releasePosHandle = sim.getObjectHandle("ReleasePos")
releasePos = sim.getObjectPosition(releasePosHandle,-1)
releaseOrient = sim.getObjectOrientation(releasePosHandle,-1)

--Manejador de la rutas principales del robot
releasePath = createPath("releasePath",idlePos,idleOrient,releasePos,
    releaseOrient)
takenPath = createPath("takenPath",idlePos,idleOrient,takenPos,takenOrient)

-- Manejador del script del robot
robotScriptHandle = sim.getScriptHandle("IRB140")
sim.setScriptVariable("releasePath",robotScriptHandle,releasePath)
sim.setScriptVariable("takenPath",robotScriptHandle,takenPath)

--Creacion del los dummy de las rutas (se eliminaran mas adelante)
```

```
path = sim.createPath(1);
sim.setObjectName(path,"pickupPath")
path2 = sim.createPath(1);
sim.setObjectName(path2,"pickupPath2")
end

function sysCall_actuation()
--Movimiento de la banda
moveBelt(beltVelocity)
end

function createPath(name,startPoint,startOrient,endPoint,endOrient)
--Crear objeto de ruta
local path = sim.createPath(1)

-- Crear variables buffer
local buffer = {startPoint[1],startPoint[2],startPoint[3],startOrient[1],
    startOrient[2],startOrient[3], 1,0,0,0,0,
  endPoint[1],endPoint[2],endPoint[3],endOrient[1],endOrient[2],endOrient
    [3],              1,0,0,0,0}

--Puntos de inicio y final
sim.insertPathCtrlPoints(path,0,0,2,buffer)

--Renombre de objeto
sim.setObjectName(path,name)

return path
end

function updatePickupPath(posT)
--Manejadores de rutas
local path = sim.getObjectHandle("pickupPath")
local path2 = sim.getObjectHandle("pickupPath2")
```

```
--Remover ambas rutas
sim.removeObject(path)
sim.removeObject(path2)

--Crear dummies de rutas
local dummyPos = {posT[1], posT[2], posT[3]}
local dummyPos2 = {posT[1], posT[2], posT[3]+0.05}
local dummyOrient = {posT[4], posT[5], posT[6]}
--[[
print("La posocion de agarre es:")
print(dummyPos)
print("La orientacion de agarre es:")
print(dummyOrient)
]]--

--Crear rutas
local pickPath = createPath("pickupPath",dummyPos2,dummyOrient,dummyPos,
    dummyOrient)
local pickPath2 = createPath("pickupPath2",idlePos,idleOrient,dummyPos2,
    dummyOrient)
sim.setScriptVariable("pickupDummy",robotScriptHandle,pickPath)
sim.setScriptVariable("pickupDummy2",robotScriptHandle,pickPath2)

--Senial de activacion del movimiento del robot
sim.setIntegerSignal("objectAvailable",1)
end

function posTarget(inInts,inFloats,inStrings,inBuffer)
-- Recibir las coordenadas de la pieza
local pos = {}
local count = 1
for i in string.gmatch(inStrings[1], "%S+") do
pos[count] = tonumber(i)
```

```
count = count + 1
end
--print(pos)
updatePickupPath(pos)
end

function setVelocity(inInts,inFloats,inStrings,inBuffer)
beltVelocity=inFloats[1]
return {},{},{},''
end

function moveBelt(beltVelocity)
local dt=sim.getSimulationTimeStep()
local posb=sim.getPathPosition(BeltpathHandle)
posb=posb+beltVelocity*dt
sim.setPathPosition(BeltpathHandle,posb)
end
```

Cuboid (creación de objetos utilizados como piezas de manufactura):

```
--Threaded child script

function sysCall_threadmain()
  local PieceTime = 15
  local VISIBLE_EDGES = 2
  local RESPONDABLE_SHAPE = 8
  local cuboSize = {0.04, 0.07, 0.035}
  local number = 0
  sim.wait(PieceTime)

  while true do
    number = number + 1
    local cubo = sim.createPureShape(0,VISIBLE_EDGES+RESPONDABLE_SHAPE,
```

```
    cuboSize,0.025,NULL)
     local numberPos = {-0.2, math.random(410, 590)*(-0.001), 0.50}
     local numberOri = {0, 0, math.random(0,45)*math.pi/180}
     sim.setObjectPosition(cubo,-1,numberPos)
     sim.setObjectOrientation(cubo,-1,numberOri)
     sim.setObjectSpecialProperty(cubo,(
    sim.objectspecialproperty_detectable_all +
    sim.objectspecialproperty_renderable))
     sim.wait(PieceTime)
  end
end
```

IRB140:

```
--Threaded child script

--Velocidad y acelracion del robot
nominalVel = 0.5
nominalAcc = 0.75

--Manejadores
target = sim.getObjectHandle("target")
connector = sim.getObjectHandle("Connector")
belt1_script = sim.getScriptHandle("ConveyorBelt")
Proximity = sim.getObjectHandle("Proximity_sensor")

--Variables
takenPosHandle = -1
releasePath = -1
pickupDummy = -1
pickupDummy2 = -1

--Pinza
```

```
connect = sim.getObjectHandle("IRB140_connection")
gripper = sim.getObjectChild(connect, 0)
gripperName = "BaxterGripper"

function sysCall_threadmain()
  while sim.getSimulationState()~=sim.simulation_advancing_abouttostop do
    --En espera de la senial de activacion "objectAvailable"
    sim.waitForSignal("objectAvailable")
    --Manejadores de las rutas fijas
    path = sim.getObjectHandle("pickupPath")
    path2 = sim.getObjectHandle("pickupPath2")
    --Recorrido de ruta hacia posicion cenit de la pieza
    sim.followPath(target,path2,3,0,nominalVel,nominalAcc)
    --Recorrido de ruta hacia la pieza
    sim.followPath(target,path,3,0,nominalVel+0.1,nominalAcc+0.1)
    --Cierre de pinza
    sim.setIntegerSignal(gripperName.."_close", 1)
    sim.wait(0.2)
    --Recorrido de ruta hacia posicion cenit de la pieza
    sim.followPath(target,path,3,1,-nominalVel-0.1,-nominalAcc-0.1)
    --Recorrido de ruta hacia la posicion de inicio
    sim.followPath(target,path2,3,1,-nominalVel,-nominalAcc)
    --Revisar si fue sujetada la pieza correctamente
    local sensorGripper = sim.readProximitySensor(Proximity)
    --print(sensorGripper)

    if (sensorGripper == 1) then
      print("La pieza fue sujetada correctamente")
      --Recorrido de ruta hacia posicion para depositar la pieza al
   contenedor azul
      sim.followPath(target,releasePath,3,0,nominalVel,nominalAcc)
      -- Tiempo de espera
      sim.wait(0.25)
      --Apertura de pinza
```

```
        sim.clearIntegerSignal(gripperName..”_close”)
        --Regreso a la posicion de incio
        sim.followPath(target,releasePath,3,1,-nominalVel,-nominalAcc)
        else
        sim.clearIntegerSignal(gripperName..”_close”)
        print(”Error al sujetar la pieza”)
      end
      --Limpia la variable de activacion del robot
      sim.clearIntegerSignal(”objectAvailable”)
      print(”-----------------------------------\n”)
    end
end
```

Bote0 (contar piezas no sujetadas):

```
--Non-threaded child script

function sysCall_init()
  sensorB0 = sim.getObjectHandle(”Proximity_sensor_B0”)
  countB0 = 0
end

function sysCall_actuation()
  if val_sensorB0 == 1 then
    countB0 = countB0 + 1
    --print(countB0)
  end
end

function sysCall_sensing()
  val_sensorB0 = sim.readProximitySensor(sensorB0)
  sim.setIntegerSignal(”Val_Prox_B0”, countB0)
end
```

Bote1 (contar piezas sí sujetadas):

```
--Non-threaded child script

function sysCall_init()
  sensorB1 = sim.getObjectHandle("Proximity_sensor_B1")
  countB1 = 0
end

function sysCall_actuation()
  if val_sensorB1 == 1 then
    countB1 = countB1 + 1
    --sim.setIntegerSignal("Val_Prox_B1", countB1)
    --print(countB1)
  end
end

function sysCall_sensing()
  val_sensorB1 = sim.readProximitySensor(sensorB1)
  sim.setIntegerSignal("Val_Prox_B1", countB1)
end
```

BaxterGripper:

```
--Non-threaded child script

function sysCall_init()
  gripperHandle=sim.getObjectHandle('BaxterGripper')
  objectName=sim.getObjectName(gripperHandle)

  motorHandle=sim.getObjectHandle('BaxterGripper_closeJoint')
```

```
  openedGap=sim.getScriptSimulationParameter(sim.handle_self,'openedGap')
  closedGap=sim.getScriptSimulationParameter(sim.handle_self,'closedGap')
  interval={0-openedGap,openedGap-closedGap}
  sim.setJointInterval(motorHandle,false,interval)

  gripperName = sim.getObjectName(sim.getObjectAssociatedWithScript(
    sim.handle_self))
end

function sysCall_actuation()
  close=sim.getIntegerSignal(gripperName..'_close')

  if (close==1) then
    sim.setJointTargetVelocity(motorHandle,0.1)
  else
    sim.setJointTargetVelocity(motorHandle,-0.1)
  end
end
```

Códigos en Python

BOF:

```
#Bibliotecas-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np
import math
import cv2

def BOF(img, center, div, name, prt):
  BOF_vector = []
  polares = []
```

```
  gr = math.pi/180
  for grados in range(0, 360, div):
    polares = [0, center[0], center[1]] #R, Rx, Ry
    while img[polares[2]][polares[1]] > 128:
      polares[0] += 1
      polares[1] = round(polares[0]*(math.cos(grados*gr)))+center[0]
      polares[2] = round(polares[0]*(math.sin(grados*gr)))+center[1]
    BOF_vector.append(polares[0])
  maximo = max(BOF_vector)
  minimo = min(BOF_vector)
  BOF_norma = [(i/maximo) for i in BOF_vector]
  orientacion = (BOF_vector.index(minimo)*div)
  tamano = 2*maximo

  if prt == True:
    plt.plot(BOF_norma, label="BOF")
    plt.xlim(0, 360/div), plt.ylim(0, max(BOF_norma)*1.5)
    plt.legend(loc='upper left'), plt.title("Descriptor BOF de "+name)
    plt.xlabel("Puntos"), plt.ylabel("Distancia normalizada")
    plt.savefig("Resultados/BOF.jpg", bbox_inches='tight', dpi=150)
    plt.show()

  return BOF_norma, orientacion, tamano


def centroide(img, prt):
  mom = [0,0,0] #M00, M10, M01
  for i in range(img.shape[0]):
    for j in range(img.shape[1]):
      d = img[i][j]/255
      mom[0] += d
      mom[1] += j*d
      mom[2] += i*d
```

```
  if mom[0] > 0: #Caso donde SI hay objeto
    centro = [round(mom[1]/mom[0]), round(mom[2]/mom[0])] #Cx, Cy
    st = True
  else:
    centro = [-1,-1] #Cx, Cy
    st = False

  if (prt == True) and (st == True):
    imagen_centroide = img.copy()
    n = 2
    for i in range(centro[1]-n, centro[1]+n): #dibujar el centroide
    for j in range(centro[0]-n, centro[0]+n):
    imagen_centroide[i][j] = 128
    plt.imshow(imagen_centroide, cmap="gray"), plt.xticks([]), plt.yticks
    ([])
    plt.title("Centroide")
    plt.savefig("Resultados/Centroide.jpg", bbox_inches='tight', dpi=150)
    plt.show()

  return centro, st

def binaria(img, umbral, prt):
  imagen_bin = np.zeros_like(img)
  for i in range(img.shape[0]):
    for j in range(img.shape[1]):
      if img[i][j] > umbral:
        imagen_bin[i][j] = 255

  if prt == True:
    plt.imshow(imagen_bin, cmap="gray"), plt.xticks([]), plt.yticks([])
    plt.title("Binaria")
    plt.savefig("Resultados/Binaria.jpg", bbox_inches='tight', dpi=150)
    plt.show()
```

```
  return imagen_bin


def histograma(img, name, prt):
  hist = cv2.calcHist([img], [0], None, [256], [0, 256])
  thresh,_ = cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
  if thresh == 0:
    thresh = 1

  if prt == True:
    plt.plot(hist)
    plt.xlabel('Intensidad de grises [8 bits]'), plt.ylabel('Pixeles')
    plt.title("Histograma de "+name)
    plt.axvline(thresh, color = 'r', linestyle = 'dashed', linewidth = 2,
   label="Umbral = "+str(int(thresh)))
    plt.legend()
    plt.savefig("Resultados/Histograma.jpg", bbox_inches='tight', dpi=150)
    plt.show()

  return thresh

def filtro_pb(img, ker, prt):
  conv = np.zeros_like(img)
  kernel = np.ones((ker, ker), np.float32)/(ker**2)
  conv = cv2.filter2D(img, -1, kernel)

  if prt == True:
    plt.imshow(conv, cmap="gray"), plt.xticks([]), plt.yticks([])
    plt.title("Filtro pasa-bajas")
    plt.savefig("Resultados/PasaBajas.jpg", bbox_inches='tight', dpi=150)
    plt.show()

  return conv
```

```
def gris(img, prt):
  img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  # img_gray = np.zeros((img.shape[0], img.shape[1]))
  # for i in range(img.shape[0]):
  #     for j in range(img.shape[1]):
  #         img_gray[i][j] = round(0.11*(img[i][j][0])+0.59*(img[i][j][1])
    +0.30*(img[i][j][2]))
  # img_gray = img_gray.astype(np.uint8)

  if prt == True:
    plt.imshow(img_gray, cmap="gray"), plt.xticks([]), plt.yticks([])
    plt.title("Escala de grises")
    plt.savefig("Resultados/Grises.jpg", bbox_inches='tight', dpi=150)
    plt.show()

  return img_gray

def algoritmo(img, kernel, name, albof, grd, prt):
  if prt == True:
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)), plt.xticks([]), plt.
    yticks([])
    plt.title("Original")
    plt.savefig("Resultados/Original.jpg", bbox_inches='tight', dpi=150)
    plt.show()

  #Imagen escala de grises:
  imagen_gris = gris(img, prt)
  #Filtro pasa bajas
  imagen_filtro = filtro_pb(imagen_gris, kernel, prt)
  #Histograma
  Umbral = histograma(imagen_filtro, name, prt)
  #Binarizacion
  imagen_binaria = binaria(imagen_filtro, Umbral, prt)
  #Centroide
```

```
  imagen_centroide, state_centroide = centroide(imagen_binaria, prt)
  #BOF
  if (albof == True) and (state_centroide == True):
    vector_bof, direccion_bof, radio_bof = BOF(imagen_binaria,
   imagen_centroide, grd, name, prt)
  else:
    vector_bof = 0
    direccion_bof = 0
    radio_bof = 0

  return vector_bof, direccion_bof, radio_bof, imagen_centroide,
   state_centroide
```

Ajuste lineal por Mínimos Cuadrados:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np



def lineal(vector1, vector2, ordenada, tit, titX, titY, num, grf):
  n = len(vector1) #Numero de pares de datos

  #----Calculo por minimo cuadrados----
  sumaX = 0
  sumaY = 0
  sumaX2 = 0
  sumaY2 = 0
  sumaXY = 0
  phi = 0

  for i in range(0, n): #Sumas----
    sumaX += vector1[i]
```

```
    sumaY += vector2[i]
    sumaX2 += vector1[i]**2
    sumaY2 += vector2[i]**2
    sumaXY += (vector1[i])*(vector2[i])

if ordenada == False:
    m = sumaXY/sumaX2 #pendiente
    b = 0 #ordenada
    for i in range(0, n):
    phi += (vector2[i]-m*vector1[i]-b)**2
    cons = (1/(n*sumaX2-sumaX**2))*(phi/(n-2))
    dm = np.sqrt(n*cons) #Error pendiente
    db = 0 #Error ordenada
else:
    m = (n*sumaXY-sumaX*sumaY)/(n*sumaX2-sumaX**2) #pendiente
    b = (sumaY-m*sumaX)/n #ordenada
    for i in range(0,n):
    phi += (vector2[i]-m*vector1[i]-b)**2
    cons = (1/(n*sumaX2-sumaX**2))*(phi/(n-2))
    dm = np.sqrt(n*cons) #Error pendiente
    db = np.sqrt(sumaX2*cons) #Error ordenada

r = sumaXY/np.sqrt(sumaX2*sumaY2) #Coeficiente de correlacion
r2 = r**2 #Coeficiente de determinacion
#----Impresion de datos----
rd = 3 #Redondeo de numeros
pendiente = str(round(m,rd))+" +- "+str(round(dm,rd))
ordenada = str(round(b,rd))+" +- "+str(round(db,rd))
coef_r = str(round(r,rd))
coef_r2 = str(round(r2,rd))
recta = "y = ("+pendiente+")x"+" + ("+ordenada+")"
# print("Pendiente: ", pendiente)
# print("Ordenada al origen: ", ordenada)
# print("Coeficiente de correlacion: ", coef_r)
```

```
  # print("Coeficiente de determinacion: ", coef_r2)
  # print("Ecuacion de la recta: ", recta)

  #----Graficar----
  if grf == True:
    postext = 2 #Posicion del texto dentro de la grafica
    y = []
    for i in range(0, n):
      y.append(m*vector1[i]+b) #Recta ajuste

    plt.plot(vector1, y, "k") #graficar ajuste lineal
    plt.errorbar(vector1, vector2, yerr=0, xerr=0, fmt="b.") #Graficar
    puntos

    if m <= 0: #Posicion texto recta
      postext = 1
    plt.legend(["Ajuste: "+recta+", $r^2$: "+coef_r2,"Datos obtenidos"],
    fontsize='x-small',loc=postext) #Etiquetas
    plt.grid()
    plt.title(tit), plt.xlabel(titX), plt.ylabel(titY)
    plt.savefig("Resultados/Grafica_"+str(num)+".jpg", bbox_inches='tight',
    dpi=150)
    plt.show()

  return m, dm, b, db, r2
```

Publicaciones por MQTT:

```
import paho.mqtt.publish as publish

# broker = "test.mosquitto.org"
broker = "192.168.100.28"
topic_vel = "LEAI40/CMFI/BT/VIN"
```

```
topic_dir = "LEAI40/CMFI/BT/DIN"

def publicar(vel, dir):
  publish.single(topic_vel, str(vel), hostname=broker)
  publish.single(topic_dir, str(dir), hostname=broker)
```

Archivo con parámetros y variables:

```
import numpy as np

# Fuente y color del texto en pantalla
fontScale = 0.4
thickness = 1
color = (0,0,255)
posText = (10, 10)

# Banderas
stop = False
flag_velocity = True
flag_BOF = False
flag_OF = False

# Variables globales
velocidad = []
velocidad_error = []
objetos = []
obj_trak = 0
obj = 0
obj_lost = 0
obj_taken = 0

# Valor de cada pixel en centimetros
pixel = 0.1336 #cm
```

```
# Parametros
vel_banda = 26 #Busqueda en la tabla
paso_banda = 0.5
tiempo_pieza = 0.03
valor_tiempo_pieza = 0.03
obj_num = 50

# Tabla velocidades
tabla_banda = np.array([
[ 0 , 0.00 ],[ 1 , 0.08 ],[ 2 , 0.16 ],[ 3 , 0.24 ],
[ 4 , 0.31 ],[ 5 , 0.39 ],[ 6 , 0.47 ],[ 7 , 0.55 ],
[ 8 , 0.63 ],[ 9 , 0.71 ],[ 10 , 0.78 ],[ 11 , 0.86 ],
[ 12 , 0.94 ],[ 13 , 1.02 ],[ 14 , 1.10 ],[ 15 , 1.18 ],
[ 16 , 1.25 ],[ 17 , 1.33 ],[ 18 , 1.41 ],[ 19 , 1.49 ],
[ 20 , 1.57 ],[ 21 , 1.65 ],[ 22 , 1.73 ],[ 23 , 1.80 ],
[ 24 , 1.88 ],[ 25 , 1.96 ],[ 26 , 2.04 ],[ 27 , 2.12 ],
[ 28 , 2.20 ],[ 29 , 2.27 ],[ 30 , 2.35 ],[ 31 , 2.43 ],
[ 32 , 2.51 ],[ 33 , 2.59 ],[ 34 , 2.67 ],[ 35 , 2.75 ],
[ 36 , 2.82 ],[ 37 , 2.90 ],[ 38 , 2.98 ],[ 39 , 3.06 ],
[ 40 , 3.14 ],[ 41 , 3.22 ],[ 42 , 3.29 ],[ 43 , 3.37 ],
[ 44 , 3.45 ],[ 45 , 3.53 ],[ 46 , 3.61 ],[ 47 , 3.69 ],
[ 48 , 3.76 ],[ 49 , 3.84 ],[ 50 , 3.92 ],[ 51 , 4.00 ],
[ 52 , 4.08 ],[ 53 , 4.16 ],[ 54 , 4.24 ],[ 55 , 4.31 ],
[ 56 , 4.39 ],[ 57 , 4.47 ],[ 58 , 4.55 ],[ 59 , 4.63 ],
[ 60 , 4.71 ],[ 61 , 4.78 ],[ 62 , 4.86 ],[ 63 , 4.94 ],
[ 64 , 5.02 ],[ 65 , 5.10 ],[ 66 , 5.18 ],[ 67 , 5.25 ],
[ 68 , 5.33 ],[ 69 , 5.41 ],[ 70 , 5.49 ],[ 71 , 5.57 ],
[ 72 , 5.65 ],[ 73 , 5.73 ],[ 74 , 5.80 ],[ 75 , 5.88 ],
[ 76 , 5.96 ],[ 77 , 6.04 ],[ 78 , 6.12 ],[ 79 , 6.20 ],
[ 80 , 6.27 ],[ 81 , 6.35 ],[ 82 , 6.43 ],[ 83 , 6.51 ],
[ 84 , 6.59 ],[ 85 , 6.67 ],[ 86 , 6.75 ],[ 87 , 6.82 ],
[ 88 , 6.90 ],[ 89 , 6.98 ],[ 90 , 7.06 ],[ 91 , 7.14 ],
[ 92 , 7.22 ],[ 93 , 7.29 ],[ 94 , 7.37 ],[ 95 , 7.45 ],
```

```
[ 96  , 7.53  ],[ 97  , 7.61  ],[ 98  , 7.69  ],[ 99  , 7.76  ],
[ 100 , 7.84  ],[ 101 , 7.92  ],[ 102 , 8.00  ],[ 103 , 8.08  ],
[ 104 , 8.16  ],[ 105 , 8.24  ],[ 106 , 8.31  ],[ 107 , 8.39  ],
[ 108 , 8.47  ],[ 109 , 8.55  ],[ 110 , 8.63  ],[ 111 , 8.71  ],
[ 112 , 8.78  ],[ 113 , 8.86  ],[ 114 , 8.94  ],[ 115 , 9.02  ],
[ 116 , 9.10  ],[ 117 , 9.18  ],[ 118 , 9.25  ],[ 119 , 9.33  ],
[ 120 , 9.41  ],[ 121 , 9.49  ],[ 122 , 9.57  ],[ 123 , 9.65  ],
[ 124 , 9.73  ],[ 125 , 9.80  ],[ 126 , 9.88  ],[ 127 , 9.96  ],
[ 128 , 10.04 ],[ 129 , 10.12 ],[ 130 , 10.20 ],[ 131 , 10.27 ],
[ 132 , 10.35 ],[ 133 , 10.43 ],[ 134 , 10.51 ],[ 135 , 10.59 ],
[ 136 , 10.67 ],[ 137 , 10.75 ],[ 138 , 10.82 ],[ 139 , 10.90 ],
[ 140 , 10.98 ],[ 141 , 11.06 ],[ 142 , 11.14 ],[ 143 , 11.22 ],
[ 144 , 11.29 ],[ 145 , 11.37 ],[ 146 , 11.45 ],[ 147 , 11.53 ],
[ 148 , 11.61 ],[ 149 , 11.69 ],[ 150 , 11.76 ],[ 151 , 11.84 ],
[ 152 , 11.92 ],[ 153 , 12.00 ],[ 154 , 12.08 ],[ 155 , 12.16 ],
[ 156 , 12.24 ],[ 157 , 12.31 ],[ 158 , 12.39 ],[ 159 , 12.47 ],
[ 160 , 12.55 ],[ 161 , 12.63 ],[ 162 , 12.71 ],[ 163 , 12.78 ],
[ 164 , 12.86 ],[ 165 , 12.94 ],[ 166 , 13.02 ],[ 167 , 13.10 ],
[ 168 , 13.18 ],[ 169 , 13.25 ],[ 170 , 13.33 ],[ 171 , 13.41 ],
[ 172 , 13.49 ],[ 173 , 13.57 ],[ 174 , 13.65 ],[ 175 , 13.73 ],
[ 176 , 13.80 ],[ 177 , 13.88 ],[ 178 , 13.96 ],[ 179 , 14.04 ],
[ 180 , 14.12 ],[ 181 , 14.20 ],[ 182 , 14.27 ],[ 183 , 14.35 ],
[ 184 , 14.43 ],[ 185 , 14.51 ],[ 186 , 14.59 ],[ 187 , 14.67 ],
[ 188 , 14.75 ],[ 189 , 14.82 ],[ 190 , 14.90 ],[ 191 , 14.98 ],
[ 192 , 15.06 ],[ 193 , 15.14 ],[ 194 , 15.22 ],[ 195 , 15.29 ],
[ 196 , 15.37 ],[ 197 , 15.45 ],[ 198 , 15.53 ],[ 199 , 15.61 ],
[ 200 , 15.69 ],[ 201 , 15.76 ],[ 202 , 15.84 ],[ 203 , 15.92 ],
[ 204 , 16.00 ],[ 205 , 16.08 ],[ 206 , 16.16 ],[ 207 , 16.24 ],
[ 208 , 16.31 ],[ 209 , 16.39 ],[ 210 , 16.47 ],[ 211 , 16.55 ],
[ 212 , 16.63 ],[ 213 , 16.71 ],[ 214 , 16.78 ],[ 215 , 16.86 ],
[ 216 , 16.94 ],[ 217 , 17.02 ],[ 218 , 17.10 ],[ 219 , 17.18 ],
[ 220 , 17.25 ],[ 221 , 17.33 ],[ 222 , 17.41 ],[ 223 , 17.49 ],
[ 224 , 17.57 ],[ 225 , 17.65 ],[ 226 , 17.73 ],[ 227 , 17.80 ],
[ 228 , 17.88 ],[ 229 , 17.96 ],[ 230 , 18.04 ],[ 231 , 18.12 ],
```

```
[ 232 , 18.20 ],[ 233 , 18.27 ],[ 234 , 18.35 ],[ 235 , 18.43 ],
[ 236 , 18.51 ],[ 237 , 18.59 ],[ 238 , 18.67 ],[ 239 , 18.75 ],
[ 240 , 18.82 ],[ 241 , 18.90 ],[ 242 , 18.98 ],[ 243 , 19.06 ],
[ 244 , 19.14 ],[ 245 , 19.22 ],[ 246 , 19.29 ],[ 247 , 19.37 ],
[ 248 , 19.45 ],[ 249 , 19.53 ],[ 250 , 19.61 ],[ 251 , 19.69 ],
[ 252 , 19.76 ],[ 253 , 19.84 ],[ 254 , 19.92 ],[ 255 , 20.00 ],

], dtype=float)
```

Programa principal:

```
#--Bibliotecas de terceros--
import matplotlib.pyplot as plt
from time import time
import numpy as np
from math import pi
import sim
import cv2

#--Bibliotecas propias--
import MQTT
import Settings as s
import BOF #Algoritmo BOF
import AjusteLineal as aj #Ajuste lineal por Minimos Cuadrados

#--Funciones--
#--Movimiento Brazo--
def moveArm(pos):
  # Ajuste al sistemas de coordenadas de la BT del simulador
  pos[0] = 0.2 + (pos[0]/100)
  pos[1] = -0.671 + (pos[1]/100)
  pos[5] = pos[5]*pi/180
  p = str(pos[0])+" "+str(pos[1])+" "+str(pos[2])+" "+str(pos[3])+" "+str(
```

```
    pos[4])+" "+str(pos[5])

  # Envio de coordenadas de garre al BR del simulador
  sim.simxCallScriptFunction(clientID,
                  "ConveyorBelt",
                  sim.sim_scripttype_childscript,
                  "posTarget",[],[],[p],"",
                  sim.simx_opmode_blocking)

#--Velocidad de la banda--
def moveBelt(V):
  sim.simxCallScriptFunction(clientID,
                  "ConveyorBelt",
                  sim.sim_scripttype_childscript,
                  "setVelocity",[],[V/100],[],"",
                  sim.simx_opmode_blocking)

#--Bordes dde la imagen para el sensor de vision--
def edgeIN(entrada):
  flag = False
  for i in range(entrada.shape[1]):
    if entrada[i][0][0] > 5:
      flag = True
      break
  return flag

def edgeOUT(entrada):
  flag = False
  for i in range(entrada.shape[1]):
    if entrada[i][entrada.shape[1]-1][0] > 5:
      flag = True
      break
  return flag
```

```
#---Ajuste lineal por Minimos Cuadrados para obtener
#   la velocidad media de cada pieza de manufactura---
def velocity(tiempo, pos, pix, n):
  titulo = "Velocidad media de la pieza"
  ejeX = "Tiempo [s]"
  ejeY = "Posicion [cm]"

  for i in range(0, len(pos)):
    pos[i] = pos[i]*pix #Cambio de pixeles a cm

  v, dv, y, dy, rr = aj.lineal(tiempo, pos, True, titulo, ejeX, ejeY, n,
   False) #False para obligar a b = 0, True para calcular b

  return v, dv, y, dy, rr

#---Graficar objetos vs sus velocidades---
def graficar(piezas, vel, vel_error):
  plt.errorbar(piezas, vel, yerr=vel_error, fmt="b.:")
  plt.legend(["Velocidad"],loc='upper left')
  plt.grid()
  plt.title("Velocidad pieza de manufactura")
  plt.xlabel("Numero de pieza")
  plt.ylabel("Velocidad [cm/s]")
  plt.savefig("Resultados/Grafica_Velocidades.jpg", bbox_inches='tight',
   dpi=150)
  plt.show()

  print("Piezas totales: ", s.obj-1)
  print("Piezas escaneadas: ", s.obj_trak)
  print("Piezas tomadas: ", s.obj_taken)
  print("Piezas perdidas: ", (s.obj-1)-(s.obj_taken))
  print("-------------------------------------------------\n")
  moveBelt(10)
```

```
#--Conexionn al simulador--
sim.simxFinish(-1)
clientID = sim.simxStart('127.0.0.1', 19999, True, True, 5000, 5)

#--Componentes a enlazar--
res, v1 = sim.simxGetObjectHandle(clientID, 'Vision_sensor', sim.
    simx_opmode_oneshot_wait)

#--Primer movimiento de la banda--
moveBelt(s.tabla_banda[s.vel_banda][1])


#--Principal--
if clientID != -1:
  print("Conexion remota establecida")

  while s.stop == False:
    if (cv2.waitKey(1) & 0xFF == ord('q')) or (s.obj >= s.obj_num+1):
      graficar(s.objetos, s.velocidad, s.velocidad_error)
      s.stop = True
      break

    tiempo = []
    posicion = []

    #--BOF--
    err, resolution, image = sim.simxGetVisionSensorImage(clientID, v1, 0,
    sim.simx_opmode_streaming)
    print("...Esperando pieza...")
    while (s.flag_BOF == False):
      err, resolution, image = sim.simxGetVisionSensorImage(clientID, v1,
    0, sim.simx_opmode_buffer)
      if err == sim.simx_return_ok:
```

```
        img = np.array(image, dtype=np.uint8)
        img.resize([resolution[1], resolution[0], 3]) #frame
        BOF_vector, BOF_direccion, BOF_radio, Img_centroide, ST_centroide =
    BOF.algoritmo(img, 5, "Figura", False, 18, False)
        cv2.imshow('SVA',img)
        if (ST_centroide == True) and (edgeIN(img) == False) and (edgeOUT(
    img) == False):
            print("Obteniendo BOF...")
            frame_anterior = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            BOF_vector, BOF_direccion, BOF_radio, Img_centroide, ST_centroide
    = BOF.algoritmo(img, 5, "Pieza de manufactura", True, 5, False)
            punto_elegido = np.array([[[Img_centroide[0], Img_centroide
    [1]]]],np.float32)
            print("------ BOF obtenido")
            s.flag_BOF = True
            s.flag_OF = True
            break
        if cv2.waitKey(1) & 0xFF == ord('q'):
            s.stop = True
            moveBelt(10)
            break
    elif err == sim.simx_return_novalue_flag:
        if cv2.waitKey(1) & 0xFF == ord('q'):
            s.stop = True
            moveBelt(1)
            break
    else:
        print("err", err)


#---Flujo optico---
print("Obteniendo flujo optico...")
tiempo_inicial = time()
while (s.flag_OF == True):
```

```
    err, resolution, image = sim.simxGetVisionSensorImage(clientID, v1,
0, sim.simx_opmode_buffer)
    if err == sim.simx_return_ok:
      img = np.array(image,dtype=np.uint8)
      img.resize([resolution[1],resolution[0],3]) #frame
      frame_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
      punto_elegido, _, _ = cv2.calcOpticalFlowPyrLK(frame_anterior,
frame_gray,punto_elegido,None,winSize=(20,20), maxLevel=3)
      tiempo.append(time()-tiempo_inicial)
      px = round(punto_elegido[0][0][0])
      py = round(punto_elegido[0][0][1])
      posicion.append(px)
      cv2.circle(img, (px, py), 3, s.color, -1)
      cv2.imshow("SVA", img)
      frame_anterior = frame_gray
      # print("Pos: ", px, py)

      #Cuando la pieza llegue al final del campo de vision de la imagen
      if (img[py][img.shape[1]-1][1] > 5) and (py > 0):
        # print("Pos: ", px, py)
        s.obj += 1
        # Se obtiene el analisis por minimos cuadrados
        v, dv, posxi, dposxi, r2 = velocity(tiempo, posicion, s.pixel, s.
obj)

        # Se comparan los errores y la poscicion del centroide respecto a
 la figura
        if (r2 > 0.90) and (px > img.shape[1]-(BOF_radio)):
          print("------ Flujo optico llevado acabo correctamente")
          s.obj_trak += 1
          s.objetos.append(s.obj)
          s.velocidad.append(v)
          s.velocidad_error.append(dv)
          print("Pieza num:", s.obj)
```

```
            print("Velocidad:", round(v, 3), "+-", round(dv, 4), "cm/s,")
            print("r2:", round(r2, 3))

            if s.flag_velocity == True:
              s.tiempo_pieza += s.valor_tiempo_pieza
              s.vel_banda += 2

            moveBelt(s.tabla_banda[s.vel_banda][1])
            posRobot= [s.tabla_banda[s.vel_banda][1]*s.tiempo_pieza + posxi
, py*s.pixel, 0.44, 0, 0, BOF_direccion-90]
            moveArm(posRobot)
            # print("Coordenadas Robot: ", posRobot)
            print("-----------------------------------------------------\n")

          else:
            if s.flag_velocity == True:
              s.vel_banda -= 4
              s.tiempo_pieza -= (s.valor_tiempo_pieza*2)
              s.flag_velocity = False
              moveBelt(s.tabla_banda[s.vel_banda][1])
            # print("-----------------------------------------------------")
            print("Error al momento de seguir la pieza")
            # print("No se calculo posicion de agarre")
            print("-----------------------------------------------------\n")

          MQTT.publicar(s.vel_banda, 0) #Envio de la velocidad por MQTT
          print(s.tabla_banda[s.vel_banda][1])
          s.flag_BOF = False
          s.flag_OF = False

          break


        if (cv2.waitKey(1) & 0xFF == ord('q')):
```

```
            graficar(s.objetos, s.velocidad, s.velocidad_error)
            s.stop = True
            break

        elif err == sim.simx_return_novalue_flag:
          if cv2.waitKey(1) & 0xFF == ord('q'):
            s.stop = True
            moveBelt(10)
            break
        else:
          print("err", err)

    _, s.obj_lost = sim.simxGetIntegerSignal(clientID, "Val_Prox_B0", sim.
    simx_opmode_streaming)
    _, s.obj_taken = sim.simxGetIntegerSignal(clientID, "Val_Prox_B1", sim.
    simx_opmode_streaming)

  print("Sistema detenido!")
  sim.simxFinish(clientID)
  cv2.destroyAllWindows()

else:
  print("Conexion remota no establecida")
```

Códigos en C

Módulo ESP32s para el control de la banda:

```
#include "EspMQTTClient.h"
#include "BANDA.h"

#define PinPWM_1 32
```

```
#define PinPWM_2 33

const int freq = 10000;
const int ChanelPWM_1 = 0;
const int ChanelPWM_2 = 1;
const int resolution = 8;

int Vel_IN = 0;
int Dir_IN = 0;

EspMQTTClient client(
"SSID",
"PASSWORD",
"192.168.100.28",   // MQTT Broker server ip local (Raspberry)
//"test.mosquitto.org", // MQTT Broker Mosquitto
"Banda",      // Client name that uniquely identify your device
1883          // The MQTT port, default to 1883. this line can be omitted
);

void setup() {
  Serial.begin(115200);

  client.enableDebuggingMessages();
  client.enableHTTPWebUpdater();

  pinMode(PinPWM_1, OUTPUT);
  pinMode(PinPWM_2, OUTPUT);

  ledcSetup(ChanelPWM_1, freq, resolution);
  ledcAttachPin(PinPWM_1, ChanelPWM_1);
  ledcSetup(ChanelPWM_2, freq, resolution);
  ledcAttachPin(PinPWM_2, ChanelPWM_2);
}
```

```
void onConnectionEstablished(){
  client.subscribe("LEAI40/CMFI/BT/VIN", [](const String & Vin) {
    Vel_IN = Vin.toInt();
    Serial.print("La velocidad de entrada es: ");
    Serial.println(Vel_IN);
  });
  client.subscribe("LEAI40/CMFI/BT/DIN", [](const String & Din) {
    Dir_IN = Din.toInt();
    Serial.print("El sentido de entrada es: ");
    Serial.println(Dir_IN);
  });
}

void loop() {
  client.loop();
  movimiento(Vel_IN, Dir_IN);
  delay(10);
}

void movimiento(int v, int d){
  if(d == 0){
    ledcWrite(ChanelPWM_1, v);
    ledcWrite(ChanelPWM_2, d);
  }
  else if(d == 1){
    ledcWrite(ChanelPWM_1, (d-1));
    ledcWrite(ChanelPWM_2, v);
  }
  client.publish("LEAI40/CMFI/BT/VOUT", String(velocidades[v][1]));
  client.publish("LEAI40/CMFI/BT/DOUT", String(d));
}
```

Tabla de búsqueda para la velocidades de la banda (archivo .h):

```
float velocidades[256][2] = {
  { 0 , 0.00 }, { 1 , 0.08 }, { 2 , 0.16 }, { 3 , 0.24 },
  { 4 , 0.31 }, { 5 , 0.39 }, { 6 , 0.47 }, { 7 , 0.55 },
  { 8 , 0.63 }, { 9 , 0.71 }, { 10 , 0.78 }, { 11 , 0.86 },
  { 12 , 0.94 }, { 13 , 1.02 }, { 14 , 1.10 }, { 15 , 1.18 },
  { 16 , 1.25 }, { 17 , 1.33 }, { 18 , 1.41 }, { 19 , 1.49 },
  { 20 , 1.57 }, { 21 , 1.65 }, { 22 , 1.73 }, { 23 , 1.80 },
  { 24 , 1.88 }, { 25 , 1.96 }, { 26 , 2.04 }, { 27 , 2.12 },
  { 28 , 2.20 }, { 29 , 2.27 }, { 30 , 2.35 }, { 31 , 2.43 },
  { 32 , 2.51 }, { 33 , 2.59 }, { 34 , 2.67 }, { 35 , 2.75 },
  { 36 , 2.82 }, { 37 , 2.90 }, { 38 , 2.98 }, { 39 , 3.06 },
  { 40 , 3.14 }, { 41 , 3.22 }, { 42 , 3.29 }, { 43 , 3.37 },
  { 44 , 3.45 }, { 45 , 3.53 }, { 46 , 3.61 }, { 47 , 3.69 },
  { 48 , 3.76 }, { 49 , 3.84 }, { 50 , 3.92 }, { 51 , 4.00 },
  { 52 , 4.08 }, { 53 , 4.16 }, { 54 , 4.24 }, { 55 , 4.31 },
  { 56 , 4.39 }, { 57 , 4.47 }, { 58 , 4.55 }, { 59 , 4.63 },
  { 60 , 4.71 }, { 61 , 4.78 }, { 62 , 4.86 }, { 63 , 4.94 },
  { 64 , 5.02 }, { 65 , 5.10 }, { 66 , 5.18 }, { 67 , 5.25 },
  { 68 , 5.33 }, { 69 , 5.41 }, { 70 , 5.49 }, { 71 , 5.57 },
  { 72 , 5.65 }, { 73 , 5.73 }, { 74 , 5.80 }, { 75 , 5.88 },
  { 76 , 5.96 }, { 77 , 6.04 }, { 78 , 6.12 }, { 79 , 6.20 },
  { 80 , 6.27 }, { 81 , 6.35 }, { 82 , 6.43 }, { 83 , 6.51 },
  { 84 , 6.59 }, { 85 , 6.67 }, { 86 , 6.75 }, { 87 , 6.82 },
  { 88 , 6.90 }, { 89 , 6.98 }, { 90 , 7.06 }, { 91 , 7.14 },
  { 92 , 7.22 }, { 93 , 7.29 }, { 94 , 7.37 }, { 95 , 7.45 },
  { 96 , 7.53 }, { 97 , 7.61 }, { 98 , 7.69 }, { 99 , 7.76 },
  { 100 , 7.84 }, { 101 , 7.92 }, { 102 , 8.00 }, { 103 , 8.08 },
  { 104 , 8.16 }, { 105 , 8.24 }, { 106 , 8.31 }, { 107 , 8.39 },
  { 108 , 8.47 }, { 109 , 8.55 }, { 110 , 8.63 }, { 111 , 8.71 },
  { 112 , 8.78 }, { 113 , 8.86 }, { 114 , 8.94 }, { 115 , 9.02 },
  { 116 , 9.10 }, { 117 , 9.18 }, { 118 , 9.25 }, { 119 , 9.33 },
  { 120 , 9.41 }, { 121 , 9.49 }, { 122 , 9.57 }, { 123 , 9.65 },
  { 124 , 9.73 }, { 125 , 9.80 }, { 126 , 9.88 }, { 127 , 9.96 },
```

```
  { 128 , 10.04 }, {  129 , 10.12 }, {  130 , 10.20 }, {  131 , 10.27 },
  { 132 , 10.35 }, {  133 , 10.43 }, {  134 , 10.51 }, {  135 , 10.59 },
  { 136 , 10.67 }, {  137 , 10.75 }, {  138 , 10.82 }, {  139 , 10.90 },
  { 140 , 10.98 }, {  141 , 11.06 }, {  142 , 11.14 }, {  143 , 11.22 },
  { 144 , 11.29 }, {  145 , 11.37 }, {  146 , 11.45 }, {  147 , 11.53 },
  { 148 , 11.61 }, {  149 , 11.69 }, {  150 , 11.76 }, {  151 , 11.84 },
  { 152 , 11.92 }, {  153 , 12.00 }, {  154 , 12.08 }, {  155 , 12.16 },
  { 156 , 12.24 }, {  157 , 12.31 }, {  158 , 12.39 }, {  159 , 12.47 },
  { 160 , 12.55 }, {  161 , 12.63 }, {  162 , 12.71 }, {  163 , 12.78 },
  { 164 , 12.86 }, {  165 , 12.94 }, {  166 , 13.02 }, {  167 , 13.10 },
  { 168 , 13.18 }, {  169 , 13.25 }, {  170 , 13.33 }, {  171 , 13.41 },
  { 172 , 13.49 }, {  173 , 13.57 }, {  174 , 13.65 }, {  175 , 13.73 },
  { 176 , 13.80 }, {  177 , 13.88 }, {  178 , 13.96 }, {  179 , 14.04 },
  { 180 , 14.12 }, {  181 , 14.20 }, {  182 , 14.27 }, {  183 , 14.35 },
  { 184 , 14.43 }, {  185 , 14.51 }, {  186 , 14.59 }, {  187 , 14.67 },
  { 188 , 14.75 }, {  189 , 14.82 }, {  190 , 14.90 }, {  191 , 14.98 },
  { 192 , 15.06 }, {  193 , 15.14 }, {  194 , 15.22 }, {  195 , 15.29 },
  { 196 , 15.37 }, {  197 , 15.45 }, {  198 , 15.53 }, {  199 , 15.61 },
  { 200 , 15.69 }, {  201 , 15.76 }, {  202 , 15.84 }, {  203 , 15.92 },
  { 204 , 16.00 }, {  205 , 16.08 }, {  206 , 16.16 }, {  207 , 16.24 },
  { 208 , 16.31 }, {  209 , 16.39 }, {  210 , 16.47 }, {  211 , 16.55 },
  { 212 , 16.63 }, {  213 , 16.71 }, {  214 , 16.78 }, {  215 , 16.86 },
  { 216 , 16.94 }, {  217 , 17.02 }, {  218 , 17.10 }, {  219 , 17.18 },
  { 220 , 17.25 }, {  221 , 17.33 }, {  222 , 17.41 }, {  223 , 17.49 },
  { 224 , 17.57 }, {  225 , 17.65 }, {  226 , 17.73 }, {  227 , 17.80 },
  { 228 , 17.88 }, {  229 , 17.96 }, {  230 , 18.04 }, {  231 , 18.12 },
  { 232 , 18.20 }, {  233 , 18.27 }, {  234 , 18.35 }, {  235 , 18.43 },
  { 236 , 18.51 }, {  237 , 18.59 }, {  238 , 18.67 }, {  239 , 18.75 },
  { 240 , 18.82 }, {  241 , 18.90 }, {  242 , 18.98 }, {  243 , 19.06 },
  { 244 , 19.14 }, {  245 , 19.22 }, {  246 , 19.29 }, {  247 , 19.37 },
  { 248 , 19.45 }, {  249 , 19.53 }, {  250 , 19.61 }, {  251 , 19.69 },
  { 252 , 19.76 }, {  253 , 19.84 }, {  254 , 19.92 }, {  255 , 20.00 }
};
```

Módulo ESP32s para el control de la pinza de agarre:

```
#include "EspMQTTClient.h"
#include <ESP32Servo.h>

#define pinServo 12
#define pinSensor 32

int pos = 0;
bool state = false;

Servo myservo;
EspMQTTClient client(
"SSID",
"PASSWORD",
"192.168.100.28",
//"test.mosquitto.org",
"ESP32_MQTT",
1883
);

void setup() {
  Serial.begin(115200);
  pinMode(pinSensor, INPUT_PULLUP);

  ESP32PWM::allocateTimer(0);
  ESP32PWM::allocateTimer(1);
  ESP32PWM::allocateTimer(2);
  ESP32PWM::allocateTimer(3);
  myservo.setPeriodHertz(50);    // standard 50 hz servo
  myservo.attach(pinServo, 500, 2000);
}
```

```
void onConnectionEstablished(){
  client.subscribe("LEAI40/CMFI/SVA/PINZA", [](const String & payload) {
    if (payload == "1"){
      myservo.write(180);
      Serial.println("Pinza cerrrada");
      state = true;
    }
    else if (payload == "0"){
      myservo.write(0);
      Serial.println("Pinza abierta");
      state = false;
    }
    else{
      Serial.println("Dato no valido");
      state = false;
    }
    delay(1000);
  });
}
void loop() {
  client.loop();
  if ((digitalRead(pinSensor) == 1)&&(state == true)){
    delay(1000);
    client.publish("LEAI40/CMFI/SVA/OBJ", "1");
    Serial.println("Pieza no tomada");
  }
  else if ((digitalRead(pinSensor) == 0)&&(state == true)){
    delay(1000);
    client.publish("LEAI40/CMFI/SVA/OBJ", "0");
    Serial.println("Pieza si tomada");
  }
  else{
    Serial.println("Pinza en espera");
```

```
    }
}
```

Códigos en el Pad Kuka

Programa en el PAD del sistema KUKA:

```
DEF ROBSEGUIDOR

INT STATE = 0
PTP HOME

LOOP
   IF STATE == 1 THEN
      PTP UBICACION
      LIN P1 CONT VEL=0.04m/s CPDAT1
      LIN P2 CONT VEL=0.04m/s CPDAT1
      LIN P3 CONT VEL=0.04m/s CPDAT1
      LIN HOME VEL=0.04m/s CPDAT1
      STATE = 0
   ENDIF
ENDLOOP

END
```

Apéndices

NodeMCU ESP32s

El núcleo de este módulo es el chip ESP32, que es escalable y adaptable y es posible controlar individualmente dos núcleos de CPU. La frecuencia del reloj se puede ajustar de 80 a 240 MHz.

El módulo integra bluetooth tradicional, bluetooth de bajo consumo y WiFi. El WiFi admite una amplia gama de conexiones de comunicación, así como conexión directa a Internet a través de un enrutador; el bluetooth permite a los usuarios conectarse a un teléfono móvil o transmitir señales para la detección e identificación del equipo. El módulo admite velocidades de datos de hasta 150 Mbps y una potencia de salida de antena de 20 dBm para una comunicación inalámbrica máxima.

Como resultado, este módulo tiene especificaciones líderes en la industria y funciona bien en términos de alta integración, distancia de transmisión inalámbrica, consumo de energía y conectividad de red.

Según su hoja de datos, cuenta con las siguientes características y puertos (Espressif, 2021):

- 802.11 b/g/n, 802.11 n (2.4 GHz), con velocidades hasta de 150 Mbps
- Potencia de transmisión de +12 dBm
- Microprocesador Xtensa single-/dual-core 32-bit LX6
- 448 KB ROM y 520 KB SRAM

- 8 MHz Oscilador interno con calibración
- RTC watchdog
- 34 × GPIOs programables
- 12-bit SAR ADC hasta 18 canales
- 2 × 8-bit DAC
- 10 × sensores touch
- 4 × SPI
- 2 × I^2S
- 2 × I^2C
- 3 × UART
- 1 host (SD/eMMC/SDIO)
- 1 slave (SDIO/SPI)
- Interfaz Ethernet MAC con DMA dedicado y compatibilidad con IEEE 1588
- Interfaz automotriz de dos cables (TWAIN®, compatible con ISO 11898-1)
- IR (TX/RX)
- Motor PWM
- LED PWM hasta 16 canales
- Sensor Hall

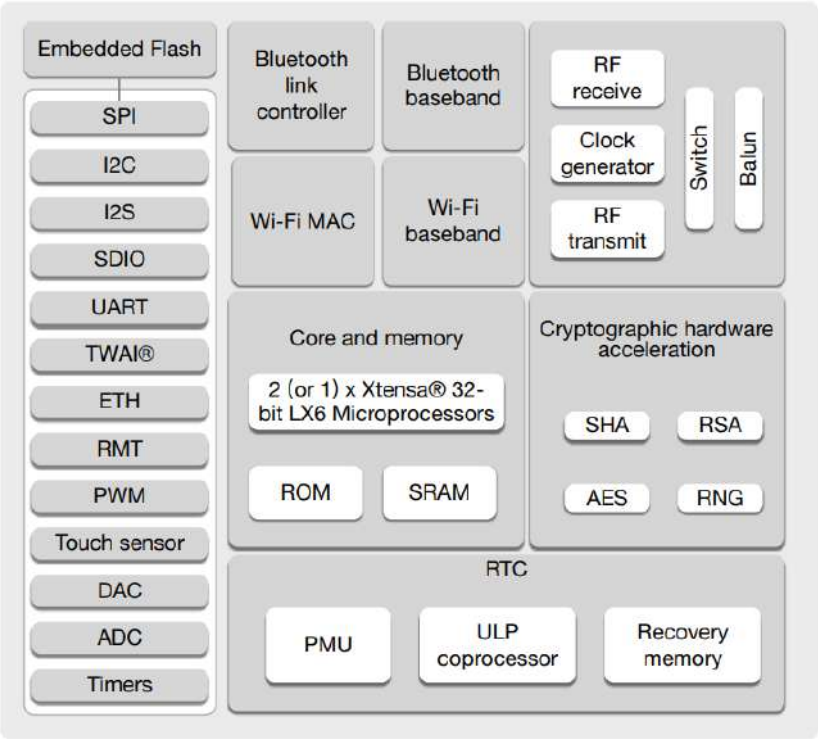


Figura A8: Diagrama de bloques del módulo NodeMCU ESP32s (Espressif, 2021).

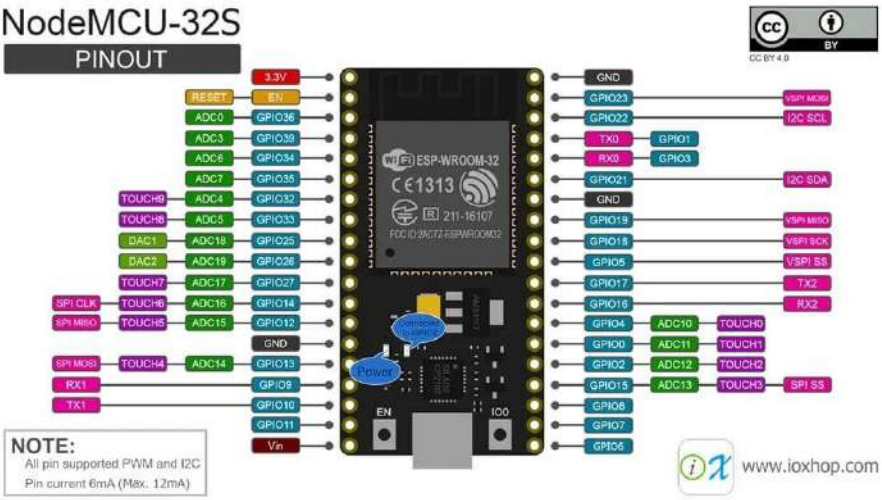


Figura A9: Pinout del módulo NodeMCU ESP32s (AranaCorp, 2020).

Raspberry Pi 4

Las Raspberry son ordenadores de placa reducida de bajo costo y para propósito general, desarrolladas en el Reino Unido por la Raspberry Pi Foundation.

Según su página de internet oficial, cuenta con las siguientes características y puertos (RaspberryPi, 2020):

- Broadcom BCM2711, SoC de 64 bits Cortex-A72 (ARM v8) de cuatro núcleos a 1,5 GHz
- SDRAM LPDDR4-3200 de 2GB, 4GB u 8GB (según el modelo)
- 2.4 GHz y 5.0 GHz IEEE 802.11ac inalámbrica, Bluetooth 5.0, BLE
- Gigabit Ethernet
- 2 puertos USB 3.0; 2 puertos USB 2.0.
- Cabecera GPIO estándar Raspberry Pi de 40 pines (totalmente compatible con las placas anteriores)
- 2 × puertos micro-HDMI (hasta 4kp60 compatible)
- Puerto de pantalla MIPI DSI de 2 carriles
- Puerto de cámara MIPI CSI de 2 carriles
- Puerto de video compuesto y audio estéreo de 4 polos
- H.265 (decodificación 4kp60), H264 (decodificación 1080p60, codificación 1080p30)
- OpenGL ES 3.1, Vulkan 1.0
- Ranura para tarjeta micro-SD para cargar el sistema operativo y el almacenamiento de datos
- 5 V CC a través del conector USB-C (mínimo 3 A)
- 5 V CC a través del encabezado GPIO (mínimo 3 A)

- Alimentación a través de Ethernet (PoE) habilitada (requiere un SOMBRERO PoE separado)
- Temperatura de funcionamiento: 0 - 50 grados C ambiente

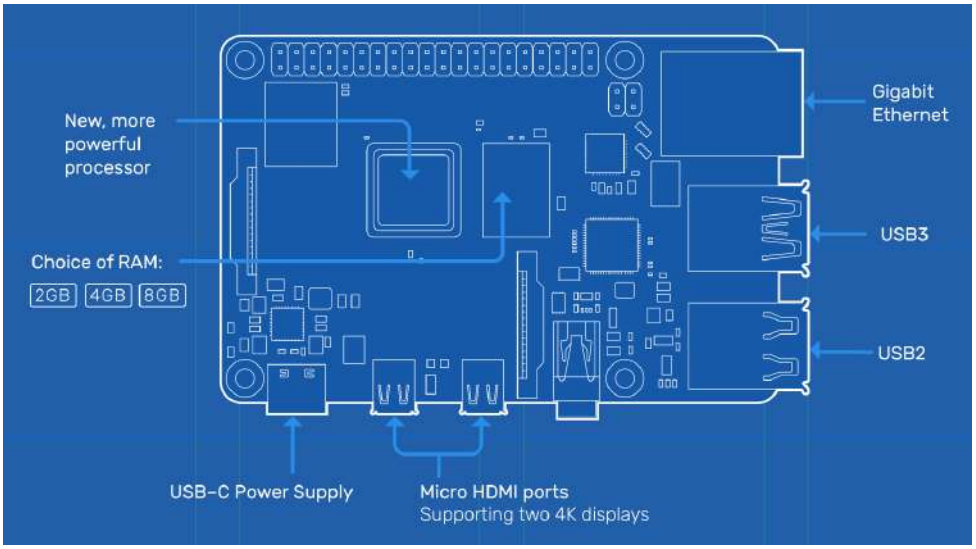


Figura A10: Diagrama de periféricos de la Raspberry Pi 4 Model B (RaspberryPi, 2020).

Cámara Raspberry Pi 4

El sensor de visión utilizado en las pruebas del SVA implementado dentro de la Raspberry Pi 4 model B, fue el modelo: Omnivision 5647 CMOS, a continuación presento las características más importantes (ElLinux.org, 2021):

- Tamaño del sensor: 3,67 x 2,74 mm (formato de 1/4 ")
- Número de píxeles: 2592 x 1944
- Tamaño de píxel: 1,4 x 1,4 um
- Lente: f = 3,6 mm, f / 2,9
- Ángulo de visión: 54 x 41 grados
- Campo de visión: 2,0 x 1,33 ma 2 m

- Lente SLR de fotograma completo equivalente: 35 mm
- Enfoque fijo: 1 m hasta el infinito
- Video: 1080p a 30 fps con códec H.264 (AVC)
- Hasta 90 fps de video en VGA
- Tamaño de la placa: 25 x 24 mm (sin incluir el cable flexible)

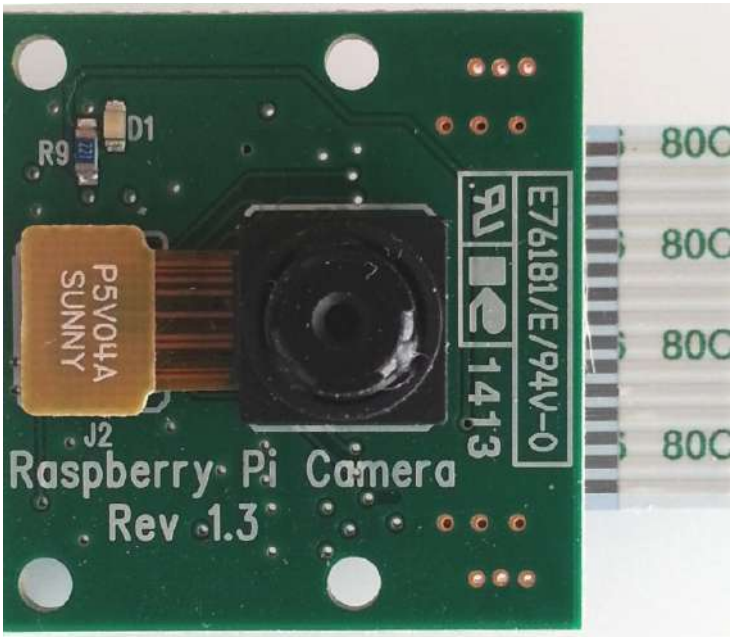


Figura A11: Cámara utilizada en la implementación del SVA dentro de la Raspberry Pi (ElLinux.org, 2021).

Brazo Robótico

El brazo robótico KUKA utilizado en este trabajo es el modelo modelo KR 5-2 arc HW. A continuación presento sus características más importantes (KUKA, 2015a).

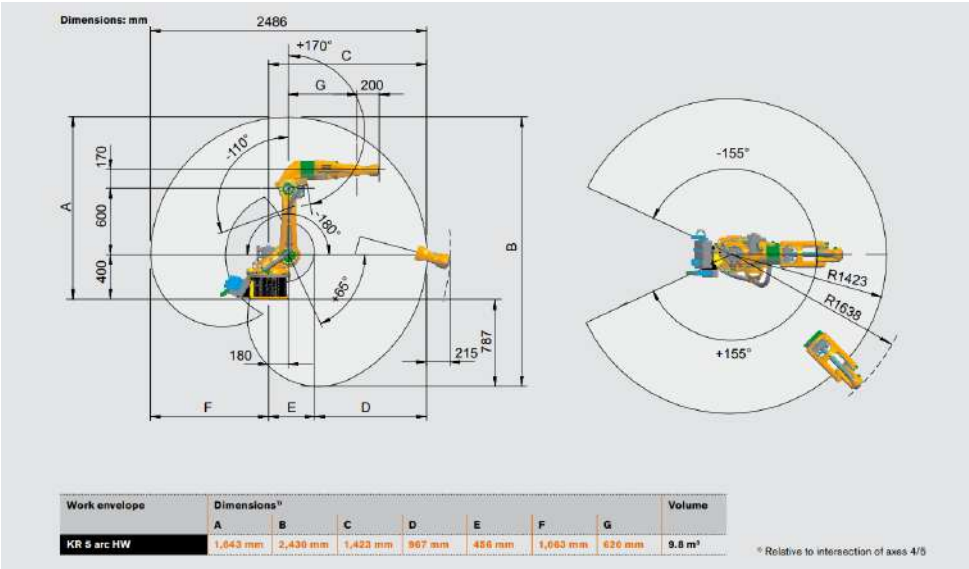
- Alcance máximo: 1.423 mm
- Carga útil nominal: 5 kg
- Carga total máxima: 37 kg

- Grados de libertad: 6
- Posición de montaje: suelo, techo
- Repetibilidad de posicionamiento: ± 0.04 mm
- Controlador: KR C2 edición 2005
- Peso (excluyendo el controlador), aprox: 126 Kg
- Temperatura durante el funcionamiento: de +10 a +55 grados celcius
- Clase de protección: IP 54
- Huella del robot 324 mm x 324 mm
- Conexión: 7.3 kVA
- Nivel de ruido: $<$ 75 dB

Respecto a sus ejes, el número de grados a los que es posible moverlo viene dado por la tabla A1, además en la figura XX pueden observarse las dimensiones de sus ejes.

Eje	Rango	Velocidad con carga de 5 Kg
A1	$\pm 155°$	$\pm 156°/s$
A2	$+155°/-180°$	$\pm 156°/s$
A3	$+170°/-110°$	$\pm 227°/s$
A4	$\pm 165°$	$\pm 390°/s$
A5	$\pm 140°$	$\pm 390°/s$
A6	∞	$\pm 858°/s$

Tabla A1: Información sobre los ejes del BR KUKA modelo KR 5-2 arc HW (KUKA, 2015a).



Work envelope	Dimensions*							Volume
	A	B	C	D	E	F	G	
KR 5 arc HW	1,643 mm	2,430 mm	1,423 mm	967 mm	456 mm	1,065 mm	620 mm	9.8 m³

* Relative to intersection of axes 4/5

Figura A12: Dimensiones de los ejes del BR KUKA modelo KR 5-2 arc HW (KUKA, 2015a).