

UNIVERSIDAD NACIONAL DE CÓRDOBA  
FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES

INGENIERÍA EN COMPUTACIÓN



Universidad  
Nacional  
de Córdoba

---

## Sistema inteligente de relevamiento de stock

---

*Autor*  
Gonzalo Alecha  
Matrícula: 37825549

*Autor*  
Martín Ferreiro  
Matrícula: 37736743

*Director*  
PhD. Orlando Micolini

*Codirector*  
Ing. Luis Ventre

Noviembre de 2019



# Resumen

El presente Proyecto Integrador tiene como objetivo el desarrollo de un sistema de relevamiento de *stock* en una tienda, haciendo uso de un robot comercial y de técnicas de aprendizaje automático junto a distintos métodos heurísticos. Por un lado, se lleva a cabo el desarrollo de mecanismos de desplazamiento y orientación del robot, a partir de un previo análisis de alternativas disponibles. Además se realiza la prueba de distintas cámaras compatibles con el robot elegido, para lograr tomar fotografías en movimiento. Por otro lado, se realiza una investigación de antecedentes en el reconocimiento de productos en estanterías. Luego de esto, se desarrolla un proceso de reconocimiento de productos en varias etapas, a través de distintas técnicas de procesamiento de imagen.

El proceso de diseño se lleva a cabo utilizando metodologías ágiles. Se utiliza un método iterativo incremental donde el proyecto se planifica en distintas iteraciones. En cada una de ellas se elige un conjunto de requerimientos asociados al proyecto y se trabaja sobre los mismos, logrando entregar al final de cada etapa una versión evolucionada del producto.

Se demuestra que utilizando la metodología planteada es posible identificar productos en una estantería a partir de imágenes obtenidas por un robot. También se comprueba que las técnicas utilizadas permiten realizar el proceso de reconocimiento con pocas imágenes de referencia por producto.



*Dedicamos este trabajo a nuestras familias y amigos por su apoyo incondicional a lo largo de todos estos años de estudio.*

*Además, queremos agradecer al director del proyecto Dr. Ing. Orlando Micolini y al codirector Ing. Luis Ventre por la predisposición, la continua interacción y más importante aún, las enseñanzas y los consejos brindados.*

*Por ultimo agradecer a la Universidad Nacional de Córdoba, publica y gratuita.*



# Índice

<b>1. Introducción</b>	<b>1</b>
1.1. Estado del arte . . . . .	1
1.1.1. StockBot . . . . .	1
1.1.2. BossaNova Robots . . . . .	1
1.1.3. Tally . . . . .	2
1.1.4. TagSurveyor . . . . .	3
1.1.5. AdvanRobot . . . . .	4
1.2. Motivación . . . . .	5
1.3. Objetivos . . . . .	6
1.3.1. Objetivo General . . . . .	6
1.3.2. Objetivos Secundarios . . . . .	6
1.4. Requerimientos . . . . .	6
1.4.1. Requerimientos Funcionales . . . . .	6
1.4.2. Requerimientos No Funcionales . . . . .	7
1.5. Análisis de Riesgos . . . . .	7
1.5.1. Listado de riesgos . . . . .	7
1.5.2. Estimación de probabilidad . . . . .	9
1.5.3. Exposición al riesgo . . . . .	10
<b>2. Marco Teórico</b>	<b>12</b>
2.1. Metodología de desarrollo . . . . .	12
2.1.1. Metodologías Tradicionales . . . . .	12
2.1.1.1. Método de la Cascada . . . . .	12
2.1.1.2. Método en Espiral . . . . .	12
2.1.2. Metodologías Ágiles . . . . .	13
2.1.2.1. Metodología Incremental . . . . .	13
2.2. Robot AlphaBot 2 Pi . . . . .	13
2.2.1. Descripción General . . . . .	13
2.2.2. Componentes del robot . . . . .	14
2.2.3. Raspberry Pi . . . . .	16
2.2.3.1. Raspberry Pi 3 modelo B+ . . . . .	16
2.3. Software utilizado . . . . .	17
2.3.1. Sistemas Operativos para Raspberry Pi . . . . .	17
2.3.1.1. Raspbian . . . . .	17
2.3.1.2. Windows 10 IoT Core . . . . .	17
2.3.1.3. Ubuntu Core . . . . .	18
2.3.2. Secure Shell . . . . .	18
2.3.3. Python . . . . .	18
2.4. Componentes de Hardware . . . . .	18
2.4.1. Baterías Li-Ion . . . . .	18
2.4.2. Modulo cargador TP4056 . . . . .	19
2.4.3. Sensores ultrasónicos . . . . .	19
2.4.3.1. Sensor HC-SR04 . . . . .	20
2.4.4. Sensores fotoeléctricos . . . . .	21
2.4.4.1. Sensor fotoeléctrico ITR20001/T . . . . .	22
2.5. Cámaras compatibles con Raspberry Pi 3 B+ . . . . .	22
2.5.1. Cámara Raspberry Pi (B) . . . . .	23
2.5.2. Raspberry Pi Camera Module v2 . . . . .	23
2.5.3. Cámara Raspberry Pi NoIR v2 . . . . .	24

2.5.4. Cámara Kuman para Raspberry PI . . . . .	24
2.5.5. Cámara Raspberry Pi, con lente ojo de pez . . . . .	24
2.6. Parámetros en fotografía . . . . .	25
2.6.1. Sensibilidad ISO . . . . .	25
2.6.2. Velocidad de obturación . . . . .	25
2.7. Generación de imágenes panorámicas . . . . .	26
2.7.1. Implementación de image stitching . . . . .	27
2.8. Redes Neuronales . . . . .	28
2.8.1. Redes Neuronales Convolucionales . . . . .	29
2.8.1.1. Faster R-CNN . . . . .	29
2.8.1.2. Mask R-CNN . . . . .	29
2.8.1.3. VGG16 . . . . .	30
2.9. Aprendizaje por transferencia . . . . .	30
2.10. Reconocimiento de objetos . . . . .	31
2.11. Segmentación de imagen . . . . .	31
2.12. Algoritmo K-NN . . . . .	32
2.13. Trabajos relacionados al reconocimiento de objetos en estanterías. . . . .	32
2.13.1. Product Recognition in Store Shelves as a Sub-Graph Isomorphism Problem . . . . .	32
2.13.2. A deep learning pipeline for product recognition in store shelves . . . . .	33
2.13.3. Retail Shelf Analytics Through Image Processing and Deep Learning . . . . .	34
<b>3. Desarrollo</b>	<b>36</b>
3.1. Iteración 0: Decisiones preliminares . . . . .	36
3.1.1. Introducción . . . . .	36
3.1.2. Requerimientos . . . . .	36
3.1.3. Desarrollo . . . . .	36
3.1.3.1. Arquitectura preliminar de alto nivel del sistema . . . . .	36
3.1.3.2. Elección del Sistema Operativo . . . . .	37
3.1.3.3. Elección del lenguaje de programación . . . . .	37
3.1.3.4. Conexión entre robot y computadora remota . . . . .	37
3.1.3.5. Creación de repositorio asociado al proyecto . . . . .	37
3.1.3.6. Elección de la metodología de desarrollo . . . . .	37
3.1.3.7. Planeamiento de Iteraciones . . . . .	38
3.1.3.8. Matriz de trazabilidad de iteraciones . . . . .	38
3.1.4. Resultados . . . . .	38
3.1.5. Riesgos superados . . . . .	38
3.1.6. Conclusiones . . . . .	39
3.2. Iteración 1: Desplazamiento de robot . . . . .	40
3.2.1. Introducción . . . . .	40
3.2.2. Requerimientos . . . . .	40
3.2.3. Desarrollo . . . . .	40
3.2.3.1. Investigación y familiarización con el robot a utilizar . . . . .	40
3.2.3.2. Análisis de alternativas para la trayectoria recta del robot . . . . .	40
3.2.3.3. Elección de alternativa . . . . .	43
3.2.3.4. Determinación de espacio de pruebas . . . . .	44
3.2.3.5. Implementación de la alternativa . . . . .	44
3.2.3.6. Seguidor de línea . . . . .	45
3.2.3.7. Sensores de distancia . . . . .	47
3.2.4. Resultados . . . . .	49
3.2.5. Riesgos superados . . . . .	50
3.2.6. Conclusiones . . . . .	50
3.3. Iteración 2: Optimización de mecanismo de reorientación . . . . .	51

3.3.1.	Introducción . . . . .	51
3.3.2.	Requerimientos . . . . .	51
3.3.3.	Desarrollo . . . . .	51
3.3.3.1.	Búsqueda de alternativa a batería actual . . . . .	51
3.3.3.2.	Refactorización de sensores de distancia . . . . .	51
3.3.3.3.	Integración de sensores de distancia con seguidor de línea . . . . .	53
3.3.3.4.	Descripción de algoritmos utilizados . . . . .	55
3.3.3.5.	Rutina de sensores de distancia . . . . .	56
3.3.3.6.	Extracción de buzzer de placa principal . . . . .	57
3.3.4.	Riesgos superados . . . . .	57
3.3.5.	Resultados . . . . .	58
3.3.6.	Conclusiones . . . . .	58
3.4.	Iteración 3: Obtención y transmisión de imágenes . . . . .	59
3.4.1.	Introducción . . . . .	59
3.4.2.	Requerimientos . . . . .	59
3.4.3.	Desarrollo . . . . .	59
3.4.3.1.	Análisis de cámaras compatibles con el robot . . . . .	59
3.4.3.2.	Pruebas de cámaras seleccionadas . . . . .	59
3.4.3.3.	Elección de la cámara . . . . .	63
3.4.3.4.	Elección de método de transmisión de imágenes . . . . .	63
3.4.3.5.	Generación de imagen de la góndola . . . . .	64
3.4.4.	Riesgos superados . . . . .	66
3.4.5.	Resultados . . . . .	67
3.4.6.	Conclusiones . . . . .	67
3.5.	Iteración 4: Reconocimiento de productos . . . . .	68
3.5.1.	Introducción . . . . .	68
3.5.2.	Requerimientos . . . . .	68
3.5.3.	Desarrollo . . . . .	68
3.5.3.1.	Primera implementación de reconocimiento de objetos . . . . .	68
3.5.3.2.	Ánalisis de trabajos relacionados a reconocimiento de productos en tiendas . . . . .	70
3.5.3.3.	Implementación de la alternativa elegida . . . . .	72
3.5.3.4.	Mejoras realizadas a la alternativa implementada . . . . .	80
3.5.3.5.	Diagrama de sistema completo . . . . .	82
3.5.3.6.	Rendimiento obtenido . . . . .	84
3.5.4.	Riesgos superados . . . . .	86
3.5.5.	Resultados . . . . .	87
3.5.6.	Conclusiones . . . . .	87
<b>4.</b>	<b>Resultados</b>	<b>88</b>
<b>5.</b>	<b>Conclusiones</b>	<b>93</b>
5.1.	Trabajos futuros . . . . .	93
<b>Bibliografía</b>		<b>95</b>
<b>A.</b>	<b>Anexos</b>	<b>98</b>
A.1.	Detecciones obtenidas en distintas imágenes . . . . .	98
A.1.1.	Prueba 1 . . . . .	98
A.1.2.	Prueba 2 . . . . .	101
A.1.3.	Prueba 3 . . . . .	103
A.1.4.	Prueba 4 . . . . .	105

A.1.5. Prueba 5 . . . . .	107
---------------------------	-----

# Índice de figuras

1.1.	Robot Stockbot de PAL Robotics . . . . .	1
1.2.	Robot de la empresa Bossa Nova en WalMart. . . . .	2
1.3.	Robot Tally. . . . .	3
1.4.	Robot TagSurveyor. . . . .	4
1.5.	Grado de cobertura de TagSurveyor. . . . .	4
1.6.	Robot AdvanBot. . . . .	4
2.1.	Estructura del modelo en cascada. . . . .	12
2.2.	Ciclo de vida del modelo Iterativo. . . . .	13
2.3.	Vista frontal y trasera de Robot Alphabot 2 Pi. . . . .	14
2.4.	Conexión de placa superior e inferior de AlphaBot 2 Pi. . . . .	14
2.5.	Vista placa inferior Alphabot 2. . . . .	15
2.6.	Vista placa superior Alphabot 2. . . . .	16
2.7.	Bateria 3.7v 4000mAh. . . . .	19
2.8.	Modulo TP4056. . . . .	19
2.9.	Diagrama de funcionamiento de sensor de distancia. . . . .	20
2.10.	Sensor de distancia HC-SR04. . . . .	20
2.11.	HC-SR04: Rangos de medición efectiva. . . . .	20
2.12.	Diagrama de funcionamiento del sensor fotoeléctrico reflectivo. . . . .	21
2.13.	Factor de reflexión en distintos colores de superficie. . . . .	21
2.14.	Diagrama de funcionamiento del sensor fotoeléctrico de barrera. . . . .	22
2.15.	Diagrama de funcionamiento del sensor fotoeléctrico retroreflectivo. . . . .	22
2.16.	Sensor ITR20001/T. . . . .	22
2.17.	Cámara Pi (B) Rev 2.0. . . . .	23
2.18.	Vista frontal de la cámara. . . . .	23
2.19.	Cámara Kuman. . . . .	24
2.20.	Cámara con lente ojo de pez. . . . .	25
2.21.	Velocidad de Obturación. . . . .	25
2.22.	Imagen 1 a unir. . . . .	26
2.23.	Imagen 2 a unir. . . . .	26
2.24.	Puntos en común entre las imágenes de entrada. . . . .	26
2.25.	Imagen panorámica generada a partir de dos imágenes de entrada. . . . .	27
2.26.	Ejemplo de una red neuronal totalmente conectada. . . . .	28
2.27.	Distintos estados de una red en base al entrenamiento. . . . .	29
2.28.	Mask R-CNN. . . . .	29
2.29.	Arquitectura VGG16. . . . .	30
2.30.	Aprendizaje por transferencia. . . . .	30
2.31.	Reconocimiento de objetos. . . . .	31
2.32.	Segmentación de imagen. . . . .	31
2.33.	Algoritmo K-NN con k=3 y k=7. . . . .	32
2.34.	Sistema de reconocimiento a partir de planogramas. . . . .	33
2.35.	Pipeline de reconocimiento de productos de A. Tonioni. . . . .	34
2.36.	Pipeline de reconocimiento de productos de A. De Biasio. . . . .	35
3.1.	Diagrama de arquitectura preliminar. . . . .	36
3.2.	Espacio de trabajo elegido. . . . .	44
3.3.	Implementación de la alternativa elegida. . . . .	45
3.4.	Sensores de seguidor de línea y estimaciones asociadas. . . . .	46
3.5.	Funcionamiento de sensor HC-SR04. . . . .	47
3.6.	Circuito asociado a cada sensor HC-SR04. . . . .	47
3.7.	Errores porcentuales obtenidos de sensor 1. . . . .	48

3.8.	Casos de acción de sensor HC-SR04.	49
3.9.	Círcuito implementado para regular voltaje de batería.	51
3.10.	Error en medición de distancia.	52
3.11.	Casos de análisis general.	52
3.12.	Caso 1: Robot sobre la línea.	54
3.13.	Caso 2: Robot entre la góndola y la línea.	54
3.14.	Caso 3: Robot más allá de la línea.	55
3.15.	Caso 4: Robot girado completamente.	55
3.16.	Diagrama de actividad general.	56
3.17.	Diagrama de actividad de rutina de sensores ultrasonido.	57
3.18.	T exp. 1/410 - ISO 100.	60
3.19.	T exp. 1/101 - ISO 100.	60
3.20.	T exp. 1/50 - ISO 100.	60
3.21.	T exp. 1/33 - ISO 100.	60
3.22.	T exp. 1/410 - ISO 300.	60
3.23.	T exp. 1/101 - ISO 300.	60
3.24.	T exp. 1/50 - ISO 300.	61
3.25.	T exp. 1/33 - ISO 300.	61
3.26.	T exp. 1/410 - ISO 500.	61
3.27.	T exp. 1/101 - ISO 500.	61
3.28.	T exp. 1/50 - ISO 500.	61
3.29.	T exp. 1/33 - ISO 500.	61
3.30.	T exp. 1/410 - ISO 100.	62
3.31.	T exp. 1/101 - ISO 100.	62
3.32.	T exp. 1/50 - ISO 100.	62
3.33.	T exp. 1/33 - ISO 100.	62
3.34.	T exp. 1/410 - ISO 300.	62
3.35.	T exp. 1/101 - ISO 300.	62
3.36.	T exp. 1/50 - ISO 300.	62
3.37.	T exp. 1/33 - ISO 300.	62
3.38.	T exp. 1/410 - ISO 500.	63
3.39.	T exp. 1/101 - ISO 500.	63
3.40.	T exp. 1/50 - ISO 500.	63
3.41.	T exp. 1/33 - ISO 500.	63
3.42.	Imagen sin comprimir.	64
3.43.	Imagen comprimida.	64
3.44.	Imagen en movimiento 1.	65
3.45.	Imagen en movimiento 2.	65
3.46.	Imagen en movimiento 3.	65
3.47.	Imagen en movimiento 4.	65
3.48.	Imagen obtenida con el algoritmo de stitching.	66
3.49.	Etiquetado de imagen de entrenamiento con LabelImg	69
3.50.	Ejemplo 1 de detección en Faster R-CNN	69
3.51.	Ejemplo 2 de detección en Faster R-CNN	70
3.52.	Ejemplo 3 de detección en Faster R-CNN	70
3.53.	Ejemplo de góndola de supermercado.	72
3.54.	Funcionamiento esperado del detector.	73
3.55.	Detección de botellas a través de una red preentrada con COCO Dataset.	74
3.56.	Etiquetado de objetos con VIA tool.	75
3.57.	Detección de productos con Mask R-CNN luego de 30 épocas.	77
3.58.	Imagen obtenida con script de generación de cuadros.	77
3.59.	Imágenes obtenidas a partir de las regiones detectadas.	78

3.60. Obtención de imagen de referencia. . . . .	78
3.61. Extracción de features con VGG16. . . . .	79
3.62. Extracción y posterior comparación de features. . . . .	79
3.63. Error en la detección de cajas contiguas. . . . .	81
3.64. Advertencia generada ante una posible detección errónea. . . . .	82
3.65. Diagrama completo de la implementación realizada. . . . .	83
3.66. Variación de la precisión con el entrenamiento del detector. . . . .	84
3.67. Variación de la exhaustividad con el entrenamiento del detector. . . . .	85
3.68. Cambios en la precisión a partir de la variación de imágenes de referencia. . . . .	85
3.69. Cambios en la exhaustividad a partir de la variación de imágenes de referencia. . . . .	86
4.1. Robot realizando recorrido alrededor de la estantería. . . . .	88
4.2. Imagen de góndola 1. . . . .	89
4.3. Imagen de góndola 2. . . . .	89
4.4. Imagen de góndola 3. . . . .	89
4.5. Imagen de góndola 4. . . . .	89
4.6. Imagen generada a partir de las imágenes recibidas de la góndola. . . . .	89
4.7. Imagen generada a partir de las imágenes recibidas de la góndola. . . . .	90
4.8. Recortes generados a partir de detecciones. . . . .	90
4.9. Recortes generados a partir de detecciones. . . . .	91
4.10. Comparación y elección de imagen de referencia de mayor similitud. . . . .	91
4.11. Salida de consola de imagen ingresada. . . . .	92
A.1. Imagen de entrada de prueba 1 . . . . .	98
A.2. Formas detectadas en prueba 1 . . . . .	99
A.3. Salida de consola en prueba 1 . . . . .	100
A.4. Imagen de entrada de prueba 2 . . . . .	101
A.5. Formas detectadas en prueba 2 . . . . .	101
A.6. Salida de consola en prueba 2 . . . . .	102
A.7. Imagen de entrada de prueba 3 . . . . .	103
A.8. Formas detectadas en prueba 3 . . . . .	103
A.9. Salida de consola en prueba 3 . . . . .	104
A.10.Imagen de entrada de prueba 4 . . . . .	105
A.11.Formas detectadas en prueba 4 . . . . .	105
A.12.Salida de consola en prueba 4 . . . . .	106
A.13.Imagen de entrada de prueba 5 . . . . .	107
A.14.Formas detectadas en prueba 5 . . . . .	107
A.15.Salida de consola en prueba 5 . . . . .	108

# Índice de Tablas

1.1. Requerimientos funcionales. . . . .	6
1.2. Requerimientos no funcionales. . . . .	7
1.3. Riesgo Sistema operativo incorrecto del robot. . . . .	7
1.4. Riesgo Incompatibilidad o avería de componentes. . . . .	8
1.5. Riesgo Elección incorrecta de escenario de prueba. . . . .	8
1.6. Riesgo Modificación de los requerimientos del proyecto. . . . .	8
1.7. Riesgo Dificultad en conseguir determinados componentes. . . . .	8
1.8. Riesgo Excesivo tiempo para cumplir los objetivos del proyecto. . . . .	9
1.9. Riesgo Reducción de la fuerza de trabajo. . . . .	9
1.10. Riesgo Pérdida de información relacionada al proyecto. . . . .	9
1.11. Criterios de ocurrencia de riesgos. . . . .	9
1.12. Riesgos y probabilidades asociadas. . . . .	10
1.13. Criterios de cuantificación de impacto. . . . .	10
1.14. Riesgos e impactos asociados. . . . .	10
1.15. Riesgos y exposición. . . . .	11
2.1. Especificaciones Raspberry PI 3 B+. . . . .	17
3.1. Plan de acciones asociado a cada iteración. . . . .	38
3.2. Plan de acciones asociado a cada iteración. . . . .	38
3.3. Riesgos mitigados en iteración 0. . . . .	39
3.4. Alternativa RFID. . . . .	40
3.5. Alternativa Brújula electrónica. . . . .	41
3.6. Alternativa Kinect. . . . .	41
3.7. Alternativa Seguidor de línea. . . . .	42
3.8. Alternativa Sensores de distancia. . . . .	42
3.9. Alternativa encoder. . . . .	43
3.10. Alternativa Giroscopio. . . . .	43
3.11. Funcionalidades asignadas a pines GPIO. . . . .	48
3.12. Riesgos mitigados en iteración 1. . . . .	50
3.13. Tabla de pines. . . . .	53
3.14. Riesgos mitigados en iteración 2. . . . .	58
3.15. Comparación de cámaras elegidas. . . . .	59
3.16. Comparación de transferencia de imágenes con y sin compresión. . . . .	64
3.17. Riesgos mitigados en iteración 3. . . . .	67
3.18. Alternativa de identificador de objetos con isomorfismo de grafos. . . . .	71
3.19. Alternativa de identificador de objetos con pipeline basado en Deep Learning. . . . .	71
3.20. Alternativa de identificador de objetos con isomorfismo de grafos. . . . .	71
3.21. Comparación de distintos métodos de segmentación de imagen. . . . .	73
3.22. Párametros de Mask RCNN. . . . .	76
3.23. Recursos de hardware utilizados. . . . .	76
3.24. Riesgos mitigados en iteración 4. . . . .	87

# 1. Introducción

## 1.1. Estado del arte

En la vorágine de la vida moderna, las personas buscan realizar sus tareas cotidianas en el menor tiempo posible. En particular, al realizar las compras en tiendas se pueden encontrar espacios vacíos en un estante y asumir que los productos se han vendido, sin tomarse el tiempo para consultar la disponibilidad. Por ello, es importante para los comerciantes vigilar los niveles de inventario y mantener los estantes lo más completos posible.

En la actualidad, uno de los objetivos de la industria consiste en utilizar la tecnología disponible para lograr la automatización de tareas que son repetibles, previsibles y manuales, como el escaneo de estantes en busca de artículos agotados, precios incorrectos y etiquetas incorrectas o faltantes.

Los robots que realizan un seguimiento del inventario pueden indicar a los empleados que los artículos tienen una demanda tan alta que se están agotando más rápido de lo esperado, y las personas que trabajan en el piso de ventas necesitan reponer el suministro.

Poder automatizar la detección de elementos faltantes en góndolas permite aprovechar de manera más eficiente el tiempo, ya que se puede realizar este tipo de análisis mientras la tienda se encuentra cerrada. El personal anteriormente asignado a estas tareas puede ser reubicado en la empresa, permitiendo aumentar la capacidad productiva.

A continuación, se enumeran algunos prototipos desarrollados hasta el momento.

### 1.1.1. StockBot

El *StockBot* (Figura 1.1) es un robot creado por la empresa catalana *PAL Robotics*; consiste en un sistema autónomo que permite realizar inventario diario en tiendas minoristas y almacenes.

*StockBot* solo requiere una primera configuración, la cual se realiza a través de un joystick, moviendo el robot por el área de trabajo para que se genere un mapa. Luego de esto, el robot está en condiciones de comenzar a detectar elementos a través de la tecnología *RFID*, haciendo uso de 8 antenas (4 a cada lado).

Cabe destacar que posee una autonomía de 8 horas de uso continuo, tomando 4 horas para una recarga completa de sus baterías.



Figura 1.1: Robot Stockbot de PAL Robotics

### 1.1.2. BossaNova Robots

La empresa *BossaNova Robots* ha desarrollado un sistema de automatización de control de inventario para la cadena de supermercados *WalMart*. El mismo permite detectar la baja de disponibilidad de un producto, además de analizar precios incorrectos o etiquetas faltantes.

El robot (*Figura 1.2*) utiliza escáneres láser que le permite construir una imagen de lo que le rodea, y evitar fácilmente obstáculos de cualquier tipo (tecnología *LiDAR*). Este es un sistema del cual no se cuenta con suficiente información por parte de la empresa, ya que se encuentra en etapa de prueba.



Figura 1.2: Robot de la empresa Bossa Nova en WalMart.

### 1.1.3. Tally

El robot *Tally* (*Figura 1.3*) de la empresa *Simbe Robotics* utiliza *RFID* y tecnología de *computer vision* para capturar, informar y analizar la cantidad y ubicación del inventario de la tienda [1]. Desde su lanzamiento en 2015, *Tally* ha sido implementado por 11 minoristas internacionales líderes. *Simbe Robotics* comenta que su tecnología proporciona los siguientes servicios:

- Auditorías precisas de inventario en toda la tienda y recuentos de ciclos regulares.
- Alertas para artículos agotados o inventario bajo.
- Ubicación e identificación de productos extraviados con precisión inferior al metro.
- Auditorías visuales de mercancías y optimización de diseño de inventario.

Al igual que con el robot *Bossa Nova*, la información encontrada es bastante reducida.

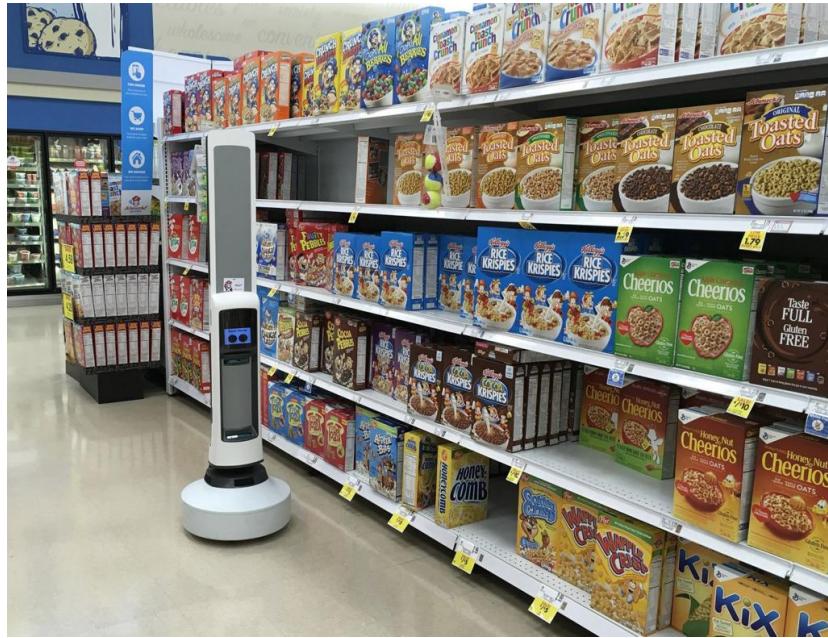


Figura 1.3: Robot Tally.

#### 1.1.4. TagSurveyor

El *TagSurveyor* (*Figura 1.4*) de la compañía *Fetch Robotics* es un robot que posee tres antenas *RFID*, las cuales permiten una cobertura de detección de elementos con etiquetas *RFID* de aproximadamente 7,6 metros [2].

Posee un sistema de reconocimiento del ambiente de trabajo, el cual es ejecutado por primera vez para generar un mapa. Luego de esto, el robot está en condiciones de comenzar a realizar la detección. Para desplazarse hace uso de un sistema de posicionamiento que le permite recorrer toda la superficie con gran precisión.

Un aspecto a destacar es el hecho de que las tres antenas *RFID* están posicionadas de una manera que permiten una cobertura de 82 grados, logrando así la lectura de productos a gran altura, como se muestra en la *Figura 1.5*.



Figura 1.4: Robot TagSurveyor.

Figura 1.5: Grado de cobertura de TagSurveyor.

### 1.1.5. AdvanRobot

*AdvanRobot* (Figura 1.6) es un sistema *RFID* móvil y autónomo, desarrollado por un grupo de trabajo de la *Universitat Pompeu Fabra*, que realiza automáticamente el inventario de un espacio determinado, por ejemplo, una tienda minorista o un almacén de techo bajo, proporcionando una mayor precisión de inventario *RFID* que los lectores portátiles [3].

Para la tarea de reconocimiento del entorno hace uso de una cámara *RGBD* ubicada en la parte inferior del robot, que se utiliza para detección de obstáculos en 3 dimensiones. También utiliza un localizador de rango láser, conocido como *SLAM*[4], y varias funcionalidades del sistema operativo *ROS* [5].

Además, utiliza 12 antenas *RFID*, 6 a cada lado, lo que le da la capacidad de tener un gran radio de cobertura y poder detectar productos posicionados en la parte superior de las estanterías.



Figura 1.6: Robot AdvanBot.

## 1.2. Motivación

La robótica está teniendo un gran crecimiento, tanto en las industrias como en la sociedad en general. En la actualidad, está surgiendo una nueva generación de robots que pueden entender y asistir a los seres humanos. Se predice que los mismos van a transformar en buena medida la vida de las personas en un futuro próximo, ya que realizarán tareas en beneficio de la humanidad. Cabe destacar que la robótica se ocupa del diseño, manufactura y programación de robots; combinando diversas disciplinas como mecánica, electrónica, informática, inteligencia artificial e ingeniería de control. Lo anteriormente descripto forma parte de los conocimientos adquiridos en diversas asignaturas de la carrera de Ingeniería en Computación, por lo cual, la utilización de un robot posibilita el desarrollo de un proyecto pertinente.

En diversos laboratorios alrededor del mundo se están haciendo desarrollos en robótica, tendiendo a evolucionar, con una marcada aceleración en los últimos años. Siguiendo esta tendencia, se concretan trabajos y desarrollos en el Laboratorio de Arquitectura de Computadoras (LAC) de la Facultad de Ciencias Exactas Físicas y Naturales de la UNC. Dentro de los trabajos realizados, se encuentra el robot *Hermes I* orientado al ámbito educacional, el *Hermes II* que posee un sistema de movilidad omnidireccional, y el *Hermes III* que a diferencia de su predecesor permite generar dinámicamente un mapa de la habitación donde se encuentra.

Junto al director del proyecto se detectó la necesidad de resolver la problemática asociada al relevamiento de *stock* en tiendas, donde los trabajadores realizan tareas repetitivas y tediosas. A partir de ello, el equipo de trabajo propuso llevar adelante el desarrollo de un sistema de relevamiento de *stock* en tiendas a baja escala, el cual permita realizar las tareas repetitivas asociadas de manera autónoma. Se plantea llevar a cabo la implementación haciendo uso del robot *Alphabot 2 Pi*, el cual fue puesto a punto por uno de los integrantes de este grupo de trabajo en su Práctica Profesional Supervisada. En esta primera implementación se desea que el sistema pueda reconocer dos tipos de formas de productos, y al menos dos productos por forma. De esta manera se sientan las bases para una futura implementación a gran escala, que permita reconocer una amplia variedad de productos.

Teniendo en cuenta que los avances tecnológicos marcan una fuerte tendencia hacia la *industria 4.0*, se propone el desarrollo del prototipo de un sistema que sea capaz de recolectar un gran conjunto de datos utilizando sensores, poder almacenarlos y procesarlos a partir de algoritmos de inteligencia artificial, posibilitando la toma de decisiones en tiempo real. De esta manera se puede obtener información sobre el *stock* en góndola, lo que permite predecir qué cantidad de productos se requiere en un próximo pedido, como así también cuando realizarlo. En definitiva, se logra la optimización del proceso logístico sin la intervención de un humano.

En resumen, las motivaciones para el desarrollo de este proyecto pueden separarse en dos grandes grupos. Por un lado aquellas relacionadas al producto en sí, entre las cuales pueden destacarse la actualidad del tema planteado, la capacidad del mismo de facilitar la vida de los empleados de tiendas, como también la posibilidad optimizar los procesos de logística a partir de la información obtenida.

Por otro lado, existen sin dudas ciertas motivaciones de naturaleza académica. Entre ellas podemos mencionar la integración de los conocimientos adquiridos a lo largo de nuestros estudios, como así también la posibilidad de incorporar conocimientos en áreas con un gran auge, como es el reconocimiento de objetos en imágenes y el desarrollo de robots autónomos.

### 1.3. Objetivos

#### 1.3.1. Objetivo General

- Implementar un sistema inteligente para llevar a cabo el relevamiento automático de stock en tiendas comerciales.

#### 1.3.2. Objetivos Secundarios

- Reutilizar el robot *AlphaBot 2 Pi*, armado en la Práctica Supervisada "Armado y puesta en funcionamiento de robot *AlphaBot 2 Pi* y *RB4*" [6].
- Implementar en el robot un Sistema de Control de trayectoria.
- Reutilizar los resultados de la investigación "Selección de herramientas de Machine Learning aplicado a problemas de Ingeniería" [7].
- Implementar un sistema que sea capaz de reconocer productos de manera autónoma y mostrar los resultados obtenidos.
- Considerar la variabilidad de productos de una gondola tipo para determinar la cantidad de clases y productos a detectar.

### 1.4. Requerimientos

Los requerimientos funcionales, enunciados en la *Tabla 1.1*, son aquellas declaraciones de los servicios que debe proporcionar el sistema, cómo debe reaccionar ante determinados eventos y cómo debe comportarse en situaciones particulares.

Los requerimientos no funcionales, descriptos en la *Tabla 1.2*, son aquellos que no se refieren directamente a la funcionalidad del sistema sino a aquellas propiedades que emergen de ella como la fiabilidad, la capacidad de almacenamiento, disponibilidad, etc. También, definen las restricciones del sistema como los dispositivos de entrada/salida y las interfaces de comunicación del sistema [8].

A partir del análisis de documentos de trabajos previos y de reuniones compartidas con los docentes a cargo del proyecto, se determinaron los requerimientos enunciados a continuación.

#### 1.4.1. Requerimientos Funcionales

ID	Descripción
RF1	El robot debe poder seguir una trayectoria definida.
RF2	En caso de perder la trayectoria definida, debe poder encontrarla nuevamente.
RF3	En caso de encontrar obstáculos sobre su trayectoria, el robot debe detener su normal funcionamiento y dar aviso.
RF4	El robot debe saber en qué lugar se encuentra en un determinado momento.
RF5	Se debe permitir establecer una conexión remota inalámbrica entre el robot y la computadora de análisis.
RF6	El robot debe obtener imágenes sin detener su desplazamiento.
RF7	El sistema de obtención de imagen debe permitir independizarse de la iluminación del ambiente.
RF8	A partir de una imagen, el sistema debe reconocer al menos dos clases y dos tipos de productos de cada clase.

Tabla 1.1: Requerimientos funcionales.

#### 1.4.2. Requerimientos No Funcionales

ID	Descripción
RNF1	El robot debe tener una autonomía superior a 2 horas.
RNF2	El robot no debe generar contaminación acústica, ni debe representar un riesgo para las personas que se encuentren en el ambiente de trabajo del mismo.
RNF3	El espacio de trabajo del robot debe ser un ambiente cerrado, con determinadas condiciones que permitan el normal funcionamiento.
RNF4	La transmisión de datos entre el robot y la computadora de análisis de imagen se debe de manera cifrada.
RNF5	El robot debe tomar fotos cada un tiempo $t$ preestablecido.
RNF6	Cada foto capturada por el robot debe ser transmitida instantáneamente hacia la computadora de análisis.
RNF7	En la placa base del robot se debe utilizar un Sistema Operativo <i>Open Source</i> .
RNF8	Para el reconocimiento de objetos se deben utilizar como máximo 20 imágenes por producto.

Tabla 1.2: Requerimientos no funcionales.

#### 1.5. Análisis de Riesgos

El riesgo en un proyecto es un evento incierto o condición incierta qué, si ocurre, tiene un efecto positivo o negativo sobre el proyecto.

Para el análisis de los riesgos del proyecto, se decidió utilizar la estrategia "*RMMM, Risk Mitigation, Monitoring and Management*" (mitigación, monitoreo y gestión de los riesgos). El objetivo principal de la misma, es marcar las estrategias y formas de actuar del equipo de trabajo frente a los riesgos. Las etapas de esta estrategia son las siguientes:

- Prevención (o mitigación): define las estrategias necesarias para evitar el riesgo.
- Monitorización: busca detectar tempranamente las situaciones que suelen desencadenar la materialización de un riesgo en particular, para poder evitar que se concrete el riesgo.
- Gestión y plan de contingencia: se definen las estrategias y acciones a tomar para evitar que los efectos se minimice, en el caso que la prevención y monitorización falle.

##### 1.5.1. Listado de riesgos

En las *Tablas 1.3-1.10*, se presentan los distintos riesgos encontrados y su correspondiente análisis.

<b>RI-01 Sistema operativo incorrecto del robot.</b>
Condición: Elección incorrecta del sistema operativo a utilizar en la placa base del robot.
Consecuencias: Interrupciones en el funcionamiento del robot, comportamiento errático, pérdida de enlaces de comunicación, reinicios inesperados, entre otros.
Efecto: No se logrará cumplir con las expectativas mínimas del robot.

Tabla 1.3: Riesgo Sistema operativo incorrecto del robot.

<b>RI-02 Incompatibilidad o avería de componentes.</b>
<u>Condición:</u> Utilización de componentes electrónicos defectuosos o que al funcionar afectan negativamente a los demás componentes. También, puede darse incompatibilidad de tecnologías, haciendo que directamente no pueda ponerse en funcionamiento un determinado componente.
<u>Consecuencias:</u> Componentes inservibles, mediciones erróneas de sensores, daños parciales o totales de componentes.
<u>Efecto:</u> Incremento del costo económico por tener que reemplazar los componentes incompatibles o dañados.

Tabla 1.4: Riesgo Incompatibilidad o avería de componentes.

<b>RI-03 Elección incorrecta de escenario de prueba.</b>
<u>Condición:</u> Utilización de un escenario de prueba muy diferente al ambiente de trabajo final del robot.
<u>Consecuencias:</u> Al momento de probar el robot en el ambiente de trabajo, el comportamiento no será el esperado.
<u>Efecto:</u> Incremento tanto en costos como en tiempo para lograr el correcto desempeño del robot en el ambiente de trabajo.

Tabla 1.5: Riesgo Elección incorrecta de escenario de prueba.

<b>RI-04 Modificación de los requerimientos del proyecto.</b>
<u>Condición:</u> Definición de nuevos requisitos o modificación de los existentes durante el desarrollo del proyecto.
<u>Consecuencias:</u> Replanificación de las tareas a realizar. Desarrollos en etapas avanzadas en los que se invirtió tiempo y esfuerzo pueden interrumpirse y quedar inservibles para los fines del proyecto.
<u>Efecto:</u> Retrasos en la finalización del producto final del proyecto.

Tabla 1.6: Riesgo Modificación de los requerimientos del proyecto.

<b>RI-05 Dificultad en conseguir determinados componentes.</b>
<u>Condición:</u> Componentes electrónicos que deben ser importados y provenientes de empresas que no trabajan con envíos al exterior. Altos costos de importación y/o envío. Grandes tiempos de demora en los envíos desde el exterior.
<u>Consecuencias:</u> Análisis de componentes alternativos que puedan ser utilizados para el mismo fin que el componente que no puede conseguirse. En última instancia podrían adaptarse los requerimientos del proyecto.
<u>Efecto:</u> Retrasos en la finalización del producto final del proyecto. Degradación de las capacidades del robot.

Tabla 1.7: Riesgo Dificultad en conseguir determinados componentes.

<b>RI-06 Excesivo tiempo para cumplir los objetivos del proyecto.</b>
<u>Condición:</u> Cualquier dificultad técnica que retrase significativamente el progreso del proyecto. Escasez de tiempo disponible de los integrantes del equipo de desarrollo.
<u>Consecuencias:</u> Robot con capacidades reducidas, menores a las esperadas.
<u>Efecto:</u> Directores del proyecto podrán readaptar los objetivos a alcanzar dando una extensión de tiempo más allá de lo planificado desde un comienzo.

Tabla 1.8: Riesgo Excesivo tiempo para cumplir los objetivos del proyecto.

<b>RI-07 Reducción de la fuerza de trabajo.</b>
<u>Condición:</u> Miembros del equipo que abandonen el proyecto.
<u>Consecuencias:</u> Las tareas a realizar llevarán más tiempo concretarlas
<u>Efecto:</u> Retraso en la entrega final del proyecto.

Tabla 1.9: Riesgo Reducción de la fuerza de trabajo.

<b>RI-08 Pérdida de información relacionada al proyecto.</b>
<u>Condición:</u> Pérdida de código desarrollado documentación del proyecto.
<u>Consecuencias:</u> Reelaboración de la información perdida.
<u>Efecto:</u> Retraso en la entrega final del proyecto.

Tabla 1.10: Riesgo Pérdida de información relacionada al proyecto.

### 1.5.2. Estimación de probabilidad

En la *Tabla 1.11* se exponen los criterios en base a los cuales se cuantificará la probabilidad de ocurrencia de los riesgos anteriormente mencionados.

Rango de Probabilidad	Promedio para el Cálculo	Expresión en Lenguaje Natural	Valor Numérico	Código de Color
1 % a 20 %	10 %	Muy Baja Probabilidad	1	
21 % a 40 %	30 %	Baja Probabilidad	2	
41 % a 60 %	50 %	Mediana Probabilidad	3	
61 % a 80 %	70 %	Alta Probabilidad	4	
81 % a 99 %	90 %	Muy Alta Probabilidad	5	

Tabla 1.11: Criterios de ocurrencia de riesgos.

A continuación, en la *Tabla 1.12*, se expresan los riesgos identificados para el proyecto con las probabilidades estimadas subjetivamente para cada uno de ellos.

ID	Riesgo	Probabilidad
RI-01	Sistema operativo incorrecto del robot	Baja
RI-02	Incompatibilidad o avería de componentes	Muy alta
RI-03	Elección incorrecta de escenario de prueba	Alta
RI-04	Modificación de los requerimientos del proyecto	Mediana
RI-05	Dificultad en conseguir determinados componentes	Mediana
RI-06	Excesivo tiempo para cumplir los objetivos del proyecto	Alta
RI-07	Reducción de la fuerza de trabajo	Muy baja
RI-08	Pérdida de información relacionada al proyecto	Muy baja

Tabla 1.12: Riesgos y probabilidades asociadas.

En la *Tabla 1.13* se exponen los criterios en base a los cuales se cuantificará el impacto de los riesgos.

Criterio	Retraso en la Planificación	Valor Numérico	Código de Color
Insignificante	1 semana	1	Verde
Moderado	2 a 3 semanas	2	Azul
Medio	4 a 5 semanas	3	Verde
Crítico	6 a 8 semanas	4	Naranja
Catastrófico	Más de 8 semanas	5	Rojo

Tabla 1.13: Criterios de cuantificación de impacto.

Por otra parte, en la *Tabla 1.14* se expresa el impacto estimado en cuanto al máximo de tiempo que puede provocar la efectivización de un riesgo en el peor de los casos.

ID	Riesgo	Impacto
RI-01	Sistema operativo incorrecto del robot	Medio
RI-02	Incompatibilidad o avería de componentes	Moderado
RI-03	Elección incorrecta de escenario de prueba	Medio
RI-04	Modificación de los requerimientos del proyecto	Medio
RI-05	Dificultad en conseguir determinados componentes	Insignificante
RI-06	Excesivo tiempo para cumplir los objetivos del proyecto	Moderado
RI-07	Reducción de la fuerza de trabajo	Crítico
RI-08	Pérdida de información relacionada al proyecto	Catastrófica

Tabla 1.14: Riesgos e impactos asociados.

### 1.5.3. Exposición al riesgo

Se calcula la ponderación de riesgos en base a la siguiente ecuación:

$$\text{Estimacion de la probabilidad} * \text{Estimacion del impacto} = \text{Exposicion al riesgo} \quad (1)$$

Se puede observar la exposición de cada riesgo en la *Tabla 1.15*.

ID	Riesgo	Probabilidad	Impacto	Exposición
RI-01	Sistema operativo incorrecto del robot	30 %	3	0.9
RI-02	Incompatibilidad o avería de componentes	90 %	2	1.8
RI-03	Elección incorrecta de escenario de prueba	70 %	3	2.1
RI-04	Modificación de los requerimientos del proyecto	50 %	3	1.5
RI-05	Dificultad en conseguir determinados componentes	50 %	1	0.5
RI-06	Excesivo tiempo para cumplir los objetivos del proyecto	70 %	2	1.4
RI-07	Reducción de la fuerza de trabajo	10 %	4	0.4
RI-08	Pérdida de información relacionada al proyecto	10 %	5	0.5

Tabla 1.15: Riesgos y exposición.



## 2. Marco Teórico

### 2.1. Metodología de desarrollo

Existen dos tipos principales de metodologías, las Ágiles (o ligeras) y las Tradicionales (o Pesadas). Las primeras son metodologías extremadamente prácticas que generalmente obvian gran parte de la documentación y están orientadas a utilizarse en proyectos cuyos requisitos cambiarán constantemente durante todo el proceso.

Las segundas son metodologías donde todo está mucho más controlado y se genera muchísima documentación antes de proceder a implementar el proyecto, con mucho mayor peso del análisis y el diseño sobre el proyecto. Estas últimas son indicadas para proyectos grandes o cuyo rendimiento y nivel de calidad son críticos para el éxito de éste [9].

#### 2.1.1. Metodologías Tradicionales

##### 2.1.1.1 Método de la Cascada

Es un método que ordena rigurosamente las etapas del proceso para el desarrollo, de tal manera que el inicio de cada etapa debe esperar a la finalización de la inmediatamente anterior. La aplicación del modelo en cascada es adecuado para los proyectos en los cuales los requisitos se encuentran definidos claramente y no requieren futuras modificaciones [10]. En la *Figura 2.1* se puede ver la estructura de dicho modelo.

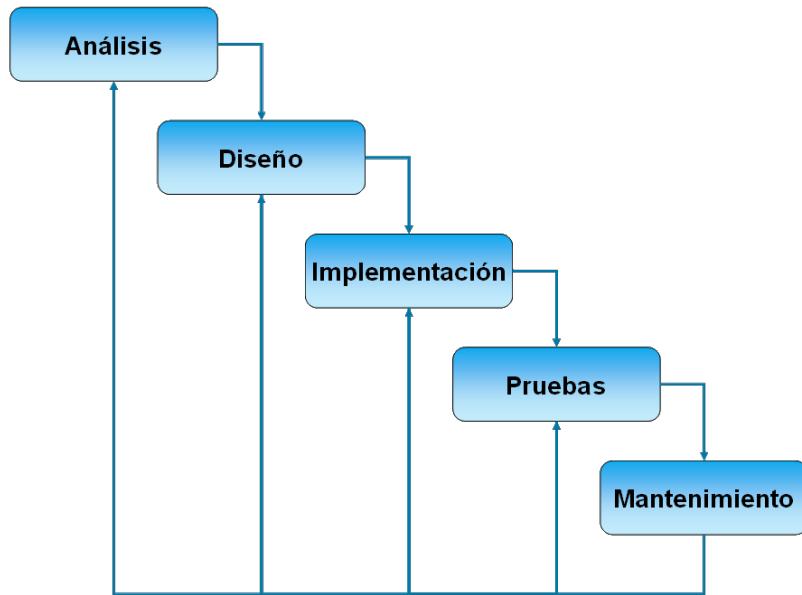


Figura 2.1: Estructura del modelo en cascada.

##### 2.1.1.2 Método en Espiral

Las actividades de este modelo se conforman en una espiral, en la que cada bucle o iteración representa un conjunto de actividades. Las actividades no están fijadas a ninguna prioridad, sino que las siguientes se eligen en función del análisis de riesgo, comenzando por el bucle interior [11].

El movimiento de la espiral, incrementando con cada iteración su amplitud radial, indica que cada vez se van construyendo versiones sucesivas del software cada vez más completas.

El modelo en espiral se divide en un número de actividades de marco de trabajo, también llamadas regiones de tareas. Cada una de las regiones están compuestas por un conjunto de tareas que se adaptan a las características del proyecto que va a emprenderse. En todos los casos se aplican actividades de protección.

### 2.1.2. Metodologías Ágiles

#### 2.1.2.1 Metodología Incremental

En un desarrollo iterativo e incremental, el proyecto se planifica en diversos bloques temporales llamados iteraciones, las cuales se pueden entender como pequeños proyectos. En todas ellas se repite un proceso de trabajo similar para proporcionar un resultado completo sobre el producto final, de manera que el cliente pueda obtener los beneficios del proyecto de forma incremental. Para ello, cada requerimiento se debe completar en una única iteración. De esta manera, no se deja para el final del proyecto ninguna actividad arriesgada relacionada con la entrega de requerimientos. En la *Figura 2.2* se presenta el ciclo de vida del modelo.

En cada iteración, el equipo evoluciona el producto (hace una entrega incremental) a partir de los resultados completados en las iteraciones anteriores, añadiendo nuevos objetivos/requerimientos o mejorando los que ya fueron realizados. Un aspecto fundamental para guiar el desarrollo iterativo e incremental, es la priorización de los objetivos/requerimientos en función del valor que aportan al cliente [12].

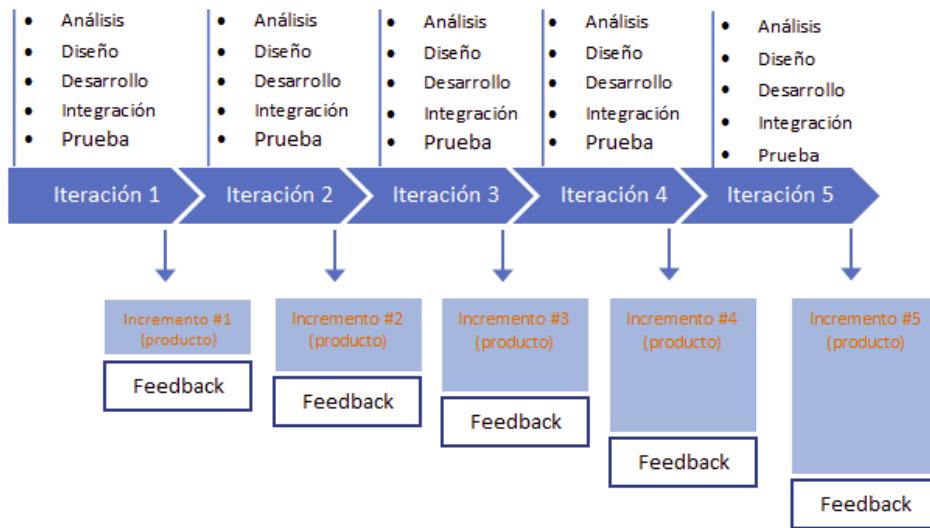


Figura 2.2: Ciclo de vida del modelo Iterativo.

## 2.2. Robot AlphaBot 2 Pi

### 2.2.1. Descripción General

*Alphabot 2* es un pequeño robot compacto, con un sistema de locomoción diferencial. Su armado no requiere soldadura ni cableado adicional, lo único que se debe hacer es unir sus conectores en el lugar correcto. Cuenta con funciones como seguimiento de líneas, detección de obstáculos, control remoto (*Bluetooth, infrarrojo, WiFi*), monitoreo de vídeo, etc.

El robot es compatible con las placas *Arduino*, *Raspberry Pi 3 Model B*, y con *Raspberry Pi Zero W*. Para cada una de las placas hay un modelo de robot distinto, donde lo único que cambia es la placa

superior (placa adaptadora). En este caso se utilizó la placa *Raspberry Pi 3 Modelo B+*, por lo que el robot es el *Alphabot 2 pi*.

En la *Figura 2.3* se puede observar que el robot está formado principalmente por dos placas, una inferior (o base), donde se encuentran las ruedas y gran parte de los sensores; y por otro lado, la placa superior (o adaptadora) donde se fija la cámara y la *Raspberry Pi*.



Figura 2.3: Vista frontal y trasera de Robot Alphabot 2 Pi.

En la *Figura 2.4*, se puede observar la conexión de ambas placas.



Figura 2.4: Conexión de placa superior e inferior de AlphaBot 2 Pi.

### 2.2.2. Componentes del robot

A continuación se presentan las placas que conforman al robot junto con los componentes integrados y una breve descripción de cada uno. Esta información fue extraída de la *wiki* brindada por la empresa fabricante *Waveshare* [13].

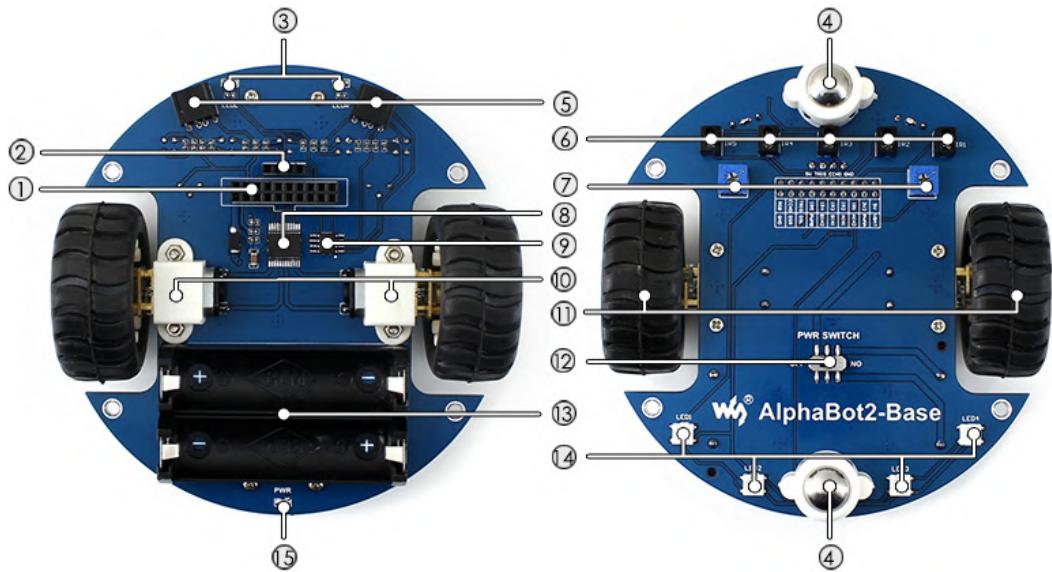


Figura 2.5: Vista placa inferior Alphabot 2.

A continuación se detalla cada uno de los componentes enumerados en la *Figura 2.5*.

1. Interfaz de control *AlphaBot2*: es utilizada para conectar dicha placa con la placa superior.
2. Interfaz de módulo ultrasónico: sirve para conectar el sensor ultrasónico, pero debido al diseño de la estructura, este módulo no puede ser usado en esta versión.
3. Indicadores utilizados para evitar obstáculos.
4. Puntos de apoyo (*ball caster*).
5. ST188: sensor fotoeléctrico infrarrojo reflectante para evitar obstáculos.
6. ITR20001/T: sensor fotoeléctrico infrarrojo reflectivo para seguimiento de línea.
7. Potenciómetro para ajustar la distancia admitida en la función de evitar obstáculos.
8. TB6612FNG: es un controlador que permite manejar dos motores de corriente continua, variando tanto la velocidad como el sentido de giro. Es una versión mejorada del *L298N*, y al igual que este, está formado por dos puentes-H.
9. Comparador de voltaje LM393.
10. Reductor del micro motor de engranajes, con tasa 1:30, 6V / 600RPM.
11. Ruedas de goma de diámetro 42 mm, y ancho 19 mm.
12. Interruptor de encendido.
13. Soporte de batería: utiliza pilas 14500.
14. WS2812B: LEDs RGB.
15. Indicador de encendido.

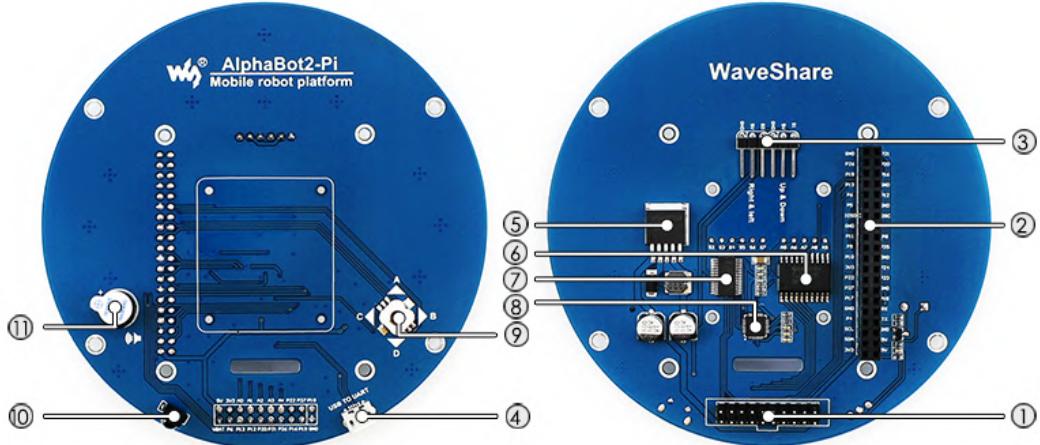


Figura 2.6: Vista placa superior Alphabot 2.

A continuación se detalla cada uno de los componentes enumerados en la *Figura 2.6*.

1. Interfaz de control *AlphaBot2*: para conectar dicha placa con la placa base.
2. Interfaz *Raspberry Pi*: para conectar la *Raspberry Pi 3 Modelo B* con el robot.
3. Interfaz donde se conectan los servomotores utilizados para direccionar la cámara.
4. Conversor *USB a UART*: para controlar la *RaspberryPi* a través de *UART*.
5. *LM2596*: regulador de voltaje 5V.
6. *TLC1543*: chip de adquisición AD de 10 bits, el cual permite a la *Raspberry Pi* usar sensores analógicos.
7. *PCA9685*: permite que el movimiento de la cámara sea más suave.
8. *CP2102*: conversor de *USB a UART*.
9. Joystick.
10. Receptor de infrarrojos.
11. *Buzzer*.

### 2.2.3. Raspberry Pi

*Raspberry Pi* es un ordenador de placa reducida, ordenador de placa única u ordenador de placa simple (*SBC*) de bajo costo desarrollado en el Reino Unido por la Fundación *Raspberry Pi*, con el objetivo de estimular la enseñanza de informática en las escuelas [14].

Desde sus comienzos hasta la actualidad se desarrolló una amplia gama de versiones. Actualmente se comercializa la versión *Raspberry Pi 4*.

#### 2.2.3.1 Raspberry Pi 3 modelo B+

El modelo B+ de la *Raspberry Pi 3* apareció en marzo del 2018 para actualizar el modelo anterior, la *Raspberry Pi 3 Model B*. Entre sus mejoras, cuenta con un nuevo procesador y mejor conectividad, pasando de tener 1.2 Ghz a tener 1.4 Ghz. En cuanto a la conectividad inalámbrica, incorpora doble banda a 2,4 GHz y 5 GHz, y su nuevo puerto Ethernet se triplica, pasa de 100 Mbits/s (en el modelo

anterior) a 300 Mbits/s en el nuevo modelo. Además cuenta con *Bluetooth 4.2 (Low Energy)* [14]. Las características destacadas de la placa se ven reflejadas en la *Tabla 2.1*.

Procesador	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC.
Frecuencia de reloj	1,4 GHz.
Memoria	1 GB SDRAM.
Conectividad Inalámbrica	- 2.4 GHz / 5 GHz IEEE 802.11.b/g/n/ac - Bluetooth 4.2, BLE
Conectividad de red	Gigabit Ethernet over USB 2.0 (300 Mbps de máximo teórico)
Puertos	GPIO 40 pines HDMI 4 x USB 2.0 CSI (cámara Raspberry Pi) DSI (pantalla táctil) Toma auriculares / vídeo compuesto Micro SD Micro USB (alimentación) Power-over-Ethernet (PoE)

Tabla 2.1: Especificaciones Raspberry PI 3 B+.

## 2.3. Software utilizado

### 2.3.1. Sistemas Operativos para Raspberry Pi

#### 2.3.1.1 Raspbian

*Raspbian* [15] es una distribución del sistema operativo *GNU/Linux*, y por lo tanto libre, basado en *Debian* para la placa (*SBC*) *Raspberry Pi*, lanzado inicialmente en junio de 2012.

Técnicamente, el sistema operativo es un *port* (adaptación) no oficial de *Debian armhf* para el procesador (CPU) de *Raspberry Pi*. El *port* fue necesario al no haber versión *Debian armhf* para la CPU *ARMv6* que contiene el *Raspberry Pi*.

Destaca también el menú *raspi-config* que permite configurar el sistema operativo sin tener que modificar archivos de configuración manualmente. Entre sus funciones, permite expandir la partición *root* para que ocupe toda la tarjeta de memoria, configurar el teclado, aplicar *overclock*, etc.

Al ser una distribución de *GNU/Linux* las posibilidades son infinitas. Todo software de código abierto puede ser recompilado en la propia *Raspberry Pi* (con arquitectura *armhf*), para permitir ser utilizado en el dispositivo en caso de que el desarrollador no proporcione una versión ya compilada para esta arquitectura. Además, esta distribución, como la mayoría, contiene repositorios donde el usuario puede descargar multitud de programas como si se tratase de una distribución de *GNU/Linux* para equipos de escritorio.

#### 2.3.1.2 Windows 10 IoT Core

*Microsoft* lanzó al mercado en 2015 su propio sistema operativo para dispositivos de la Internet de las cosas (*Internet of Things, IoT*) como el *Raspberry Pi* (2 o 3). La aplicación se orienta principalmente a desarrolladores y aficionados a la programación que quieren conectar los aparatos cotidianos con Internet o crear elementos interconectados. Si bien, tanto la descarga como la utilización del software para *Raspberry Pi* son gratuitas, no pueden realizarse cambios en el núcleo del sistema, lo que representa una gran desventaja [16].

### 2.3.1.3 Ubuntu Core

*Ubuntu* es una de las distribuciones de *Linux* más populares. El software, basado en *Debian* y desarrollado desde 2004 por *Canonical*, destaca en primer lugar por su elevada adaptabilidad y usabilidad. Bajo el nombre *Ubuntu Core*, los desarrolladores publicaron una versión en 2014 que representa una edición minimalista de la edición de servidor y puede utilizarse como sistema operativo para *Raspberry Pi*. Las principales diferencias de *Ubuntu Core* con otros sistemas operativos para *Raspberry Pi* son que cada paquete de software representa una unidad individual (“*snap*”), incluso el núcleo de *Linux*. [16]

### 2.3.2. Secure Shell

*Secure Shell (SSH)* es un protocolo que facilita las comunicaciones seguras entre dos sistemas usando una arquitectura cliente/servidor, que permite a los usuarios conectarse a un *host* remotamente. A diferencia de otros protocolos de comunicación remota tales como *FTP* o *Telnet*, *SSH* encripta la sesión de conexión.

*SSH* está diseñado para reemplazar los métodos más viejos y menos seguros para registrarse remotamente en otro sistema a través de la *shell* de comando, tales como *telnet* o *rsh*. Un programa relacionado, *scp*, reemplaza otros programas diseñados para copiar archivos entre *hosts* como *rcp*. El uso de métodos seguros para registrarse remotamente a otros sistemas reduce los riesgos de seguridad tanto para el sistema cliente como para el sistema remoto [17].

### 2.3.3. Python

*Python* es un lenguaje de programación multiparadigma ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Es administrado por la *Python Software Foundation*. Posee una licencia de código abierto, denominada *Python Software Foundation License*, que es compatible con la Licencia pública general de *GNU* a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores [18].

## 2.4. Componentes de Hardware

### 2.4.1. Baterías Li-Ion

La batería de iones de litio (*Figura 2.7*) utiliza como electrolito una sal de Litio contenido en un solvente orgánico (líquido), que proporciona los iones necesarios que circularán desde el cátodo (negativo) hasta el ánodo (positivo) durante la descarga, permitiendo que los electrones puedan moverse por el circuito y proporcionar la energía. Las principales propiedades de las baterías de *Li-ion* son la ligereza de sus componentes, su elevada capacidad energética y la resistencia a la descarga, poco efecto memoria y capacidad para funcionar con un elevado número de ciclos de regeneración. Estas características han permitido diseñar baterías ligeras, de pequeño tamaño y variadas formas. Este tipo de baterías tiene técnicas de fabricación muy pulidas y avanzadas dada su madurez en el mercado habiendo pasado muchos años desde sus primeras implementaciones (*Sony* en 1991). Al día de hoy siguen siendo muy utilizadas en la electrónica móvil de consumo masivo como los *Smartphones*, aunque la tendencia marca una migración progresiva hacia las baterías de *LiPo*.



Figura 2.7: Bateria 3.7v 4000mAh.

#### 2.4.2. Modulo cargador TP4056

El módulo *TP4056* (*Figura 2.8*) posibilita la carga de baterías *LiPo* o *Li-ion* de una sola celda de 3.7V 1Ah o superior. Basado en el chip *TP4056* y el chip de protección de batería *DW01*, este módulo ofrece una corriente de carga de 1A y luego corta cuando finaliza la carga.

Además, cuando el voltaje de la batería cae por debajo de 2,4 V, el chip de protección desconecta la carga para proteger la celda de funcionar a una tensión demasiado baja y también protege contra la conexión de sobretensión y polaridad inversa [19].



Figura 2.8: Modulo TP4056.

#### 2.4.3. Sensores ultrasónicos

Los sensores ultrasónicos miden la distancia mediante el uso de ondas ultrasónicas. El cabezal emite una onda ultrasónica y recibe la onda reflejada que retorna desde el objeto. Los sensores ultrasónicos miden la distancia al objeto contando el tiempo entre la emisión y la recepción.

La distancia se puede calcular con la siguiente fórmula:

$$L = \frac{1}{2} T C \quad (2)$$

donde L es la distancia, T es el tiempo entre la emisión y la recepción, y C es la velocidad del sonido. (El valor se multiplica por 1/2 ya que T es el tiempo de recorrido de ida y vuelta) [20]. En la *Figura 2.9* se puede observar el funcionamiento del sensor.

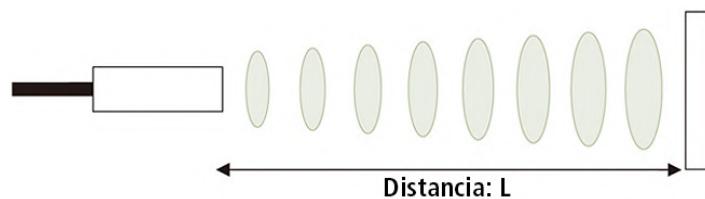


Figura 2.9: Diagrama de funcionamiento de sensor de distancia.

#### 2.4.3.1 Sensor HC-SR04

El funcionamiento del sensor es simple: se genera una onda ultrasónica en el emisor mediante un pulso en el pin "Trig" (*trigger*), la cual al encontrarse con algún obstáculo rebotará, volviendo al sensor y siendo registrada por el receptor, traduciéndose esta en un pulso en el pin "Echo".

Como se puede ver en la *Figura 2.10*, este sensor está conformado por dos cilindros puestos uno al lado del otro, uno de ellos es quien emite la señal ultrasónica mientras que el otro es quien la recibe.



Figura 2.10: Sensor de distancia HC-SR04.

El rango de medida del sensor va de 2cm a 400cm, con una resolución de 0.3cm y su ángulo de detección es de 30°, aunque para medidas más eficaces se recomienda estar dentro de los 15° [21]. Esto se puede ver ilustrado en la *Figura 2.11*.

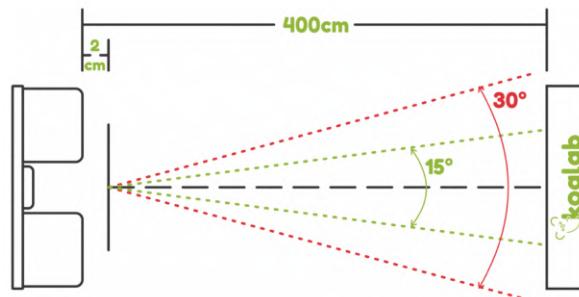


Figura 2.11: HC-SR04: Rangos de medición efectiva.

#### 2.4.4. Sensores fotoeléctricos

Un sensor fotoeléctrico es un dispositivo que detecta la presencia o alguna característica en particular de un objeto mediante luz (visible o no visible). Se pueden aplicar para detectar presencia, tamaño, color, brillo de objetos [22].

Los sensores de este tipo pueden clasificarse dentro de tres categorías dependiendo de la forma en la que detectan los objetos [23].

- Reflectivo

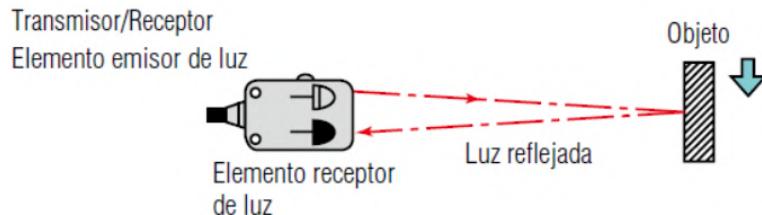


Figura 2.12: Diagrama de funcionamiento del sensor fotoeléctrico reflectivo.

Como se puede ver en la *Figura 2.12*, tanto el emisor de luz como los elementos receptores están contenidos en una sola carcasa. El sensor recibe la luz reflejada desde el objeto. La distancia de detección de este tipo de sensores va a depender directamente del color del objeto a detectar, debido a que cada color tiene un factor de reflexión de la luz diferente [24]. En la *Figura 2.13* se ilustra como varia la reflexión de la luz, cuando la misma incide sobre superficies de distinto color.

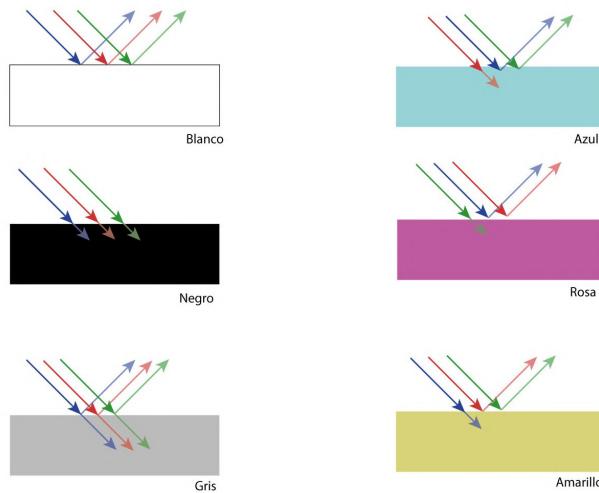


Figura 2.13: Factor de reflexión en distintos colores de superficie.

- De barrera

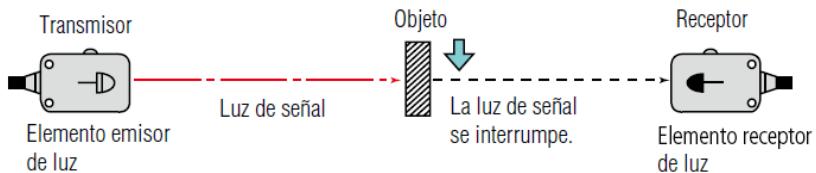


Figura 2.14: Diagrama de funcionamiento del sensor fotoeléctrico de barrera.

En la *Figura 2.14* se puede ver que en este tipo de sensor, el transmisor y el receptor están separados. Cuando el objeto se encuentra entre el transmisor y el receptor, se interrumpe la luz.

- Retroreflectivo

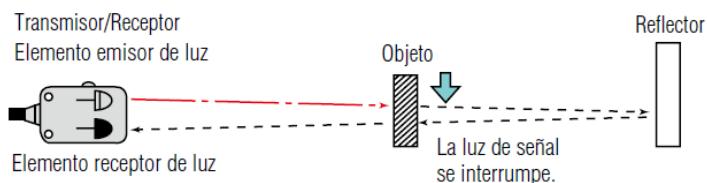


Figura 2.15: Diagrama de funcionamiento del sensor fotoeléctrico retroreflexivo.

Tanto el emisor de luz como los elementos receptores están contenidas en un mismo recinto. La luz del elemento emisor incide en el reflector y regresa al elemento receptor de luz. Cuando hay un objeto presente, se interrumpe la luz (*Figura 2.15*).

#### 2.4.4.1 Sensor fotoeléctrico ITR20001/T

El *ITR20001/T* (*Figura 2.16*) es un sensor fotoeléctrico de tipo reflectivo. Consta de un diodo emisor de infrarrojos (IR) y un fototransistor de silicio NPN (PN) encajados lado a lado en una carcasa termoplástica negra. En la situación normal, el fototransistor recibe radiación del IR solamente. Cuando un objeto reflectante se ubica cercano al sensor, el fototransistor recibe la radiación reflejada [25].

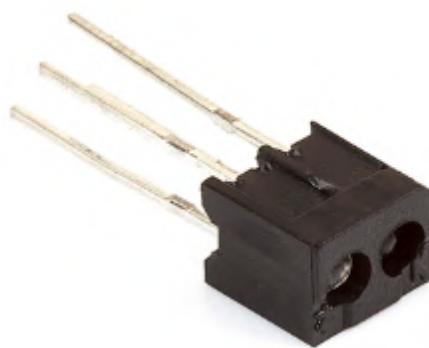


Figura 2.16: Sensor ITR20001/T.

### 2.5. Cámaras compatibles con Raspberry Pi 3 B+

Se realizó una investigación de las distintas alternativas de cámaras compatibles con la placa utilizada. A continuación se detallan las principales características de las alternativas encontradas.

### 2.5.1. Cámara Raspberry Pi (B)

El robot *AlphaBot 2 Pi* cuenta con la cámara *Raspberry Pi (B) Rev 2.0* (*Figura 2.17 y 2.18*), que posee 5 megapíxeles, un sensor OV5647 y foco de distancia ajustable.

Algunas de sus características son:

- Tamaño de sensor CCD : 1/4 pulgada.
- Apertura (F): 2.0.
- Longitud Focal: 6mm.
- Ángulo de visión (Diagonal): 60.6 grados.
- Mejor resolución de sensor: 1080p.



Figura 2.17: Cámara Pi (B) Rev 2.0.



Figura 2.18: Vista frontal de la cámara.

### 2.5.2. Raspberry Pi Camera Module v2

Posee un sensor *Sony IMX219* de 8 megapíxeles y sensores retroiluminados *Exmor R*. Se puede utilizar para tomar vídeo de alta definición, así como fotografías. Hay muchos ejemplos disponibles en los que se utiliza para capturar imágenes en distintos modos como cámara rápida, cámara lenta y otras técnicas de vídeo.

El sensor *IMX219* y la arquitectura de sensores retroiluminados *Exmor R* no solo permite mejoras respecto a la resolución, sino que es un gran avance en calidad de imagen, fidelidad de color y capturas en entornos con poca iluminación [26].

A continuación se presentan las especificaciones de la cámara [27]:

- Resolución fija: 8 megapíxeles.
- Modos de vídeo: 1080p30, 720p60 y 640 × 480p60 / 90.
- Sensor: *Sony IMX219*.
- Resolución del sensor: 3280 × 2464 píxeles.
- Área de imagen del sensor: 3,68 × 2,76 mm (4,6 mm diagonal).
- Tamaño óptico: 1/4".
- Longitud focal: 3.04 mm.
- Campo de visión horizontal: 62.2 grados.
- Campo de visión vertical: 48.8 grados.

### 2.5.3. Cámara Raspberry Pi NoIR v2

El *Pi NoIR* ofrece todo lo que ofrece el módulo de cámara normal, con una diferencia: no emplea un filtro de infrarrojos (*NoIR* = *No Infra Red.*) Esto posibilita ver en la oscuridad con luz infrarroja [28].

### 2.5.4. Cámara Kuman para Raspberry PI

Cámara de visión nocturna *Raspberry Pi* de foco ajustable (*Figura 2.19*), compatible con todas las versiones de *Raspberry*. Tiene un Sensor OV5647 de 5 megapíxeles. Incluye 2pcs luz LED infrarroja. La cámara es capaz de capturar imágenes estáticas de 2592 x 1944 píxeles, y también es compatible con 1080 p @ 30 fps, 720 p @ 60 fps y 640 x480 p 60/90 de grabación de vídeo.

Especificaciones de la cámara [29]:

- Lente : 1/4 5M.
- Apertura (F): 2.9.
- Distancia focal: 3.29M.
- Diagonal: 72.4 grados.
- La mejor resolución del sensor: 1080p (2592 × 1944 píxeles).
- Dimensión: 25 mm x 24 mm x 6 mm.

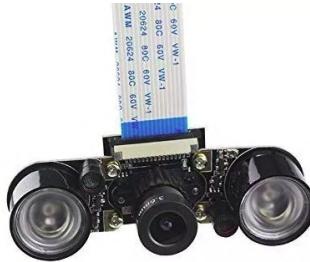


Figura 2.19: Cámara Kuman.

### 2.5.5. Cámara Raspberry Pi, con lente ojo de pez

Esta cámara contiene un lente ojo de pez, el cual ofrece un campo de visión más amplio *Figura 2.20*. El sensor es el OV5647 de 5 megapíxeles, y la distancia de enfoque es ajustable [30].

Especificaciones de la cámara:

- Tamaño CCD: 1/4 pulgadas.
- Longitud focal: 0.87mm.
- Ángulo de visión (diagonal): 200 grados (mientras que otras cámaras normales suelen ser de 72 grados).
- Sensor de mejor resolución: 1080p.
- Resolución de imagen fija 2592 × 1944.
- Admite grabación de vídeo 1080p30, 720p60 y 640x480p60.
- Dimensión: 25mm x 24mm.



Figura 2.20: Cámara con lente ojo de pez.

## 2.6. Parámetros en fotografía

### 2.6.1. Sensibilidad ISO

La sensibilidad ISO es la capacidad que tiene el sensor de la cámara para captar luz. A valores bajos de la sensibilidad ISO, el sensor es menos sensible a la luz, y su capacidad de captarla es pequeña (Valores de ISO 100 o 200, por ejemplo). A valores altos de la sensibilidad ISO el sensor es más sensible a la luz, siendo mas alta su capacidad de captarla (Valores de ISO 1600 o 3200, por ejemplo). **Variar la sensibilidad ISO tiene efectos sobre la exposición y el ruido de una imagen [31].**

### **2.6.2. Velocidad de obturación**

En fotografía, velocidad de obturación o velocidad de disparo, corresponde al inverso del tiempo de exposición y hace referencia al periodo durante el cual está abierto el obturador de una cámara fotográfica. [32]



Figura 2.21: Velocidad de Obturación.

Viendo la *Figura 2.21*, se comprueba que trabajando con velocidades de obturación rápidas (tiempos de exposición cortos), la cámara abrirá y cerrará el obturador muy rápido. Esto implica que la cantidad de luz que entra en el sensor lo hará durante un breve período de tiempo, por lo tanto, para lograr una buena exposición deberemos compensarlo iluminando bien la escena a retratar, y/o jugando con los otros dos parámetros del triángulo de la exposición: la apertura de diafragma y la sensibilidad ISO.

En cuanto a una velocidad de obturación lenta (tiempo de exposición largo), se tendrá el obturador de la cámara abierto durante más tiempo, lo cual implica que se deja pasar más luz al sensor de la

cámara. En estos casos no hará falta subir tanto la ISO ni abrir tanto el diafragma para conseguir una exposición adecuada [33].

## 2.7. Generación de imágenes panorámicas

Existen casos donde se desea capturar una gran escena pero se cuenta con una cámara que proporciona resoluciones específicas como 640x480, imposibilitando la obtención de imágenes de gran anchura. Para lograrlo se deben capturar múltiples fotografías de la escena y luego deben unirse con un algoritmo. Este proceso se denomina unión de imágenes (*image stitching*), el cual consiste en obtener los puntos en común de distintas imágenes de una misma escena, y a partir de la superposición de estos, generar una nueva imagen de mayor resolución.

En primer lugar se deben introducir imágenes de una escena, es decir, que contengan puntos en común como las dadas en la *Figura 2.22* y *2.23*.



Figura 2.22: Imagen 1 a unir.

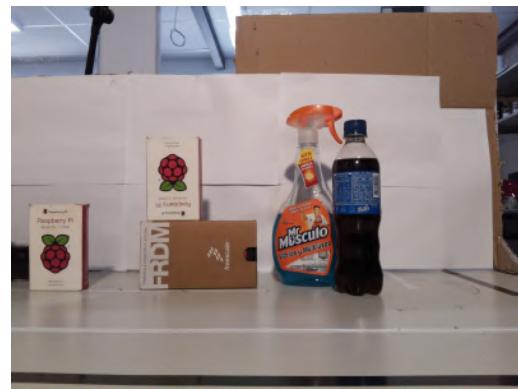


Figura 2.23: Imagen 2 a unir.

Se puede observar a simple vista que las imágenes forman parte de una misma escena y que contienen puntos en común, los cuales se observan en la *Figura 2.24*.

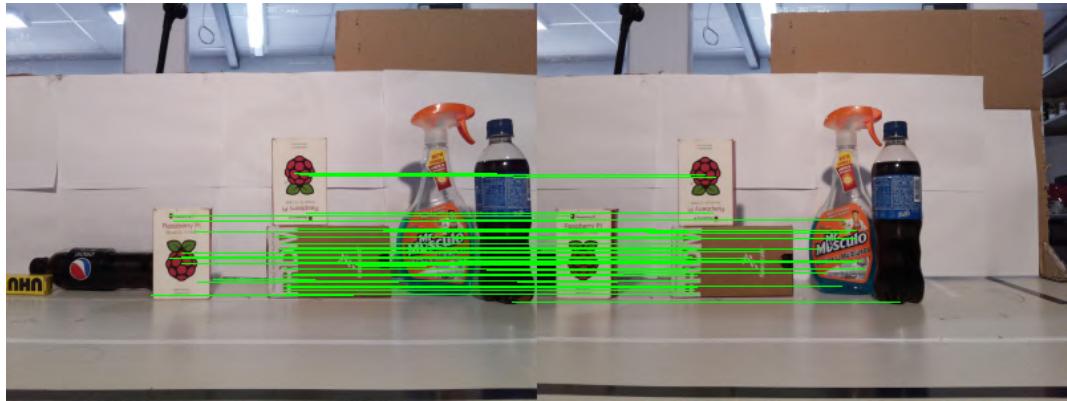


Figura 2.24: Puntos en común entre las imágenes de entrada.

Una vez que se obtienen los puntos en común de ambas imágenes se procede a formar una nueva a partir de la correspondencia de estos. La imagen resultante se ve en la *Figura 2.25*.



Figura 2.25: Imagen panorámica generada a partir de dos imágenes de entrada.

En resumen, el proceso completo contiene 3 pasos:

1. Se detectan puntos claves y se extraen descriptores invariantes locales de las dos imágenes de entrada.
2. Se realiza una comparación entre los descriptores de ambas imágenes.
3. Se estima una matriz homográfica usando las características (*features*) presentes en cada imagen, para generar una nueva.

#### 2.7.1. Implementación de image stitching

En el trabajo realizado por Matthew Brown y David G. Lowe [34] se implementa un algoritmo de unión de imágenes a partir de características invariantes en las mismas. Esta implementación tiene la ventaja de lograr una correspondencia confiable de secuencias de imágenes panorámicas a pesar de la rotación, el zoom y el cambio de iluminación en las imágenes de entrada. Además, al ver la unión de imágenes como un problema de coincidencia de múltiples imágenes, se puede descubrir automáticamente las relaciones de coincidencia entre las imágenes y reconocer panoramas en conjuntos de datos desordenados. Los pasos del algoritmo son los enumerados a continuación.

1. Extraer las características (*features*) SIFT de todas las imágenes. Dado que las mismas son invariables bajo rotación y cambios de escala, el sistema puede manejar imágenes con orientación y zoom variables, siendo esto una ventaja con respecto a implementaciones antiguas que usaban *Harris corners* para correlación de imágenes.
2. Luego de que se extraen las *features*, debe buscarse la similitud entre las mismas. Para lograr esto se utiliza un algoritmo llamado RANSAC (*random sample consensus*) junto a un modelo probabilístico para verificar la coincidencia encontrada.
3. Existe un tercer paso donde se hace un ajuste y corrección de las coincidencias para minimizar errores dados por la cámara que obtiene las fotos. Las imágenes se agregan al ajustador una por una, con la mejor imagen coincidente (número máximo de coincidencias consistentes), y la nueva imagen se inicializa con la misma rotación y longitud focal que la imagen con la que mejor se adapta.

## 2.8. Redes Neuronales

Una red neuronal es un sistema de computación compuesto por un gran número de elementos simples altamente interconectados, los cuales procesan información por medio de su estado dinámico como respuesta a entradas externas. Los datos ingresan por medio de la “capa de entrada”, pasan a través de la “capa oculta” y salen por la “capa de salida”. Cabe mencionar que la capa oculta puede estar constituida por varias capas (*Figura 2.26*).

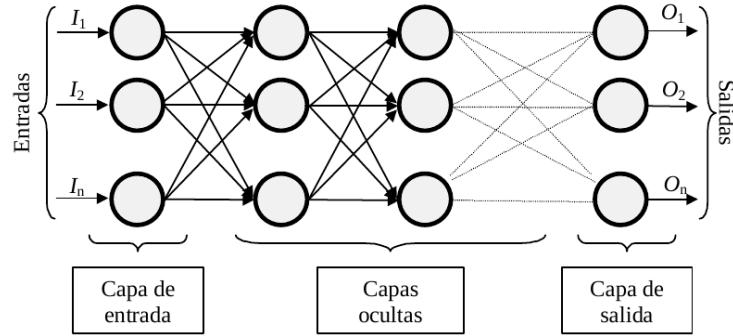


Figura 2.26: Ejemplo de una red neuronal totalmente conectada.

Las redes neuronales se entrena de una forma similar a los demás algoritmos de aprendizaje automático, brindando un *dataset* con ciertas características de entrada (conectadas a la capa de entrada) y las etiquetas correspondientes (conectadas a la capa de salida). Todos los parámetros internos de la red se inicializan aleatoriamente o en cero, y en la etapa de entrenamiento se van ajustando para que la salida coincida con la etiqueta indicada (esto se realiza en la etapa denominada "*backpropagation*", llevando al mínimo una función de costo) [35].

Una época (*epoch* en inglés) se da cuando un conjunto de datos entero se pasa hacia adelante y hacia atrás a través de la red neuronal, en los algoritmos de *forward propagation* y *backpropagation*. Como este proceso requiere un alto costo computacional, se divide en trabajos más pequeños denominados *batches*. El hecho de llevar a cabo muchas épocas se debe a que la implementación de los métodos de aprendizaje automático, utiliza un algoritmo de gradiente descendente, el cual es un proceso iterativo y una sola época sería insuficiente.

El numero de épocas óptimo está asociado a muchos factores como el tamaño del *dataset*, a la variación del mismo, entre otros. Existen tres "estados" (*Figura 2.27*) que una red puede tomar:

- Luego de una época, la red se encuentra subentrenada (o *underfitting* en inglés). En este punto la red no tiene la capacidad de predecir resultados para lo cual fue entrenada.
- Al pasar por una cantidad determinada de épocas, se alcanza un estado de entrenamiento óptimo, donde será capaz de predecir con éxito a partir de los datos dados en el entrenamiento.
- El tercer estado se da cuando la red es sobreentrenada (o *overfitting*, en inglés) y el algoritmo de aprendizaje puede quedar ajustado a características muy específicas del conjunto de datos de entrenamiento, y no sea capaz de predecir con éxito nuevas entradas.

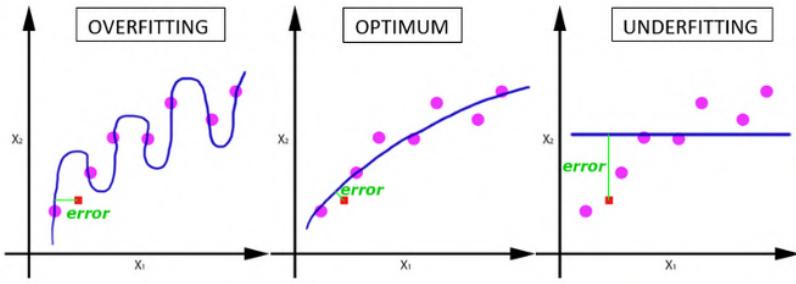


Figura 2.27: Distintos estados de una red en base al entrenamiento.

### 2.8.1. Redes Neuronales Convolucionales

Las redes neuronales convolucionales (CNNs, por sus siglas en inglés) se aplican principalmente para desarrollar modelos supervisados y no supervisados cuando los datos de entrada son imágenes. En general, las convoluciones bi-dimensionales (2D) son aplicadas a las imágenes, mientras que las convoluciones uni-dimensionales (1D) pueden ser usadas en una entrada secuencial para capturar dependencias temporales.

#### 2.8.1.1 Faster R-CNN

*Faster R-CNN* es un tipo de red *R-CNN* (basada en regiones), que consta de dos etapas. La primera, llamada Red de Propuesta de Región (*RPN*), propone cuadros de límite de objetos candidatos. La segunda etapa, que es en esencia *Fast R-CNN*, extrae las características de cada cuadro candidato usando una técnica denominada *RoIPool*, y luego realiza la clasificación y la regresión del cuadro de límite. Esta red es mucho más rápida que sus predecesoras (*Fast R-CNN*, *SPP-Net*, *R-CNN*).

#### 2.8.1.2 Mask R-CNN

Por otra parte, *Mask R-CNN* adopta el mismo procedimiento de dos etapas. La primera es idéntica; en la segunda etapa, en paralelo a la predicción de la clase y el desplazamiento de la caja, *Mask R-CNN* también genera una máscara binaria para cada Región de interés (*RoI*). Es decir que *Mask R-CNN* extiende *Faster R-CNN* al agregar una rama para predecir las máscaras de segmentación en cada región de interés, en paralelo con la rama existente para la clasificación y el delimitador de regresión de cuadros, como se puede ver en la Figura 2.28. La rama de la máscara es una pequeña *FCN* aplicada a cada *RoI*, que predice una máscara de segmentación de píxel a píxel.

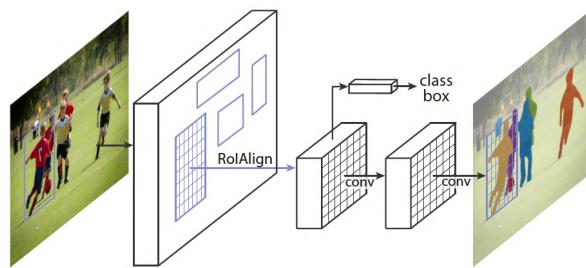


Figura 2.28: Mask R-CNN.

### 2.8.1.3 VGG16

VGG es un modelo de red neuronal convolucional para el reconocimiento de imágenes propuesto por *Visual Geometry Group* en la Universidad de Oxford, donde *VGG16* se refiere a un modelo con 16 capas de peso. La Figura 2.29 ilustra la arquitectura de *VGG16*: la capa de entrada toma una imagen del tamaño de  $(224 \times 224 \times 3)$ , y la capa de salida es una predicción en 1000 clases llamada *softmax*. Desde la capa de entrada hasta la última capa de agrupación máxima (etiquetada por  $7 \times 7 \times 512$ ) se considera la parte de extracción de características del modelo, mientras que el resto de la red se considera parte de la clasificación del modelo.

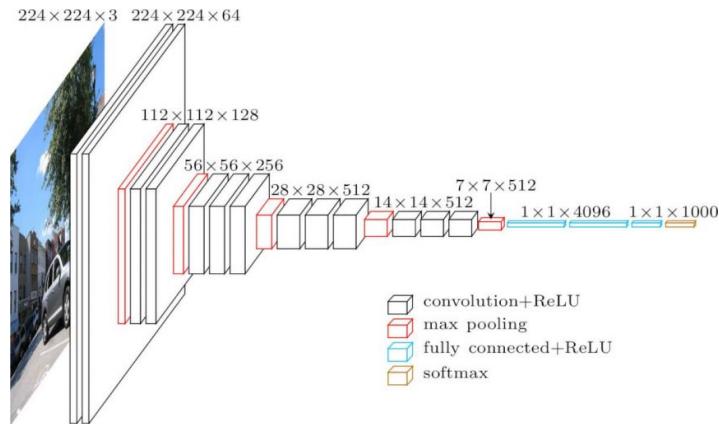


Figura 2.29: Arquitectura VGG16.

## 2.9. Aprendizaje por transferencia

El aprendizaje por transferencia (*Figura 2.30*) es un método de aprendizaje automático en el que un modelo desarrollado para una tarea se reutiliza como punto de partida para un modelo en una segunda tarea. Esto permite independizarse de los enormes recursos de hardware necesarios y los grandes conjuntos de datos para entrenar modelos de *Deep Learning*.

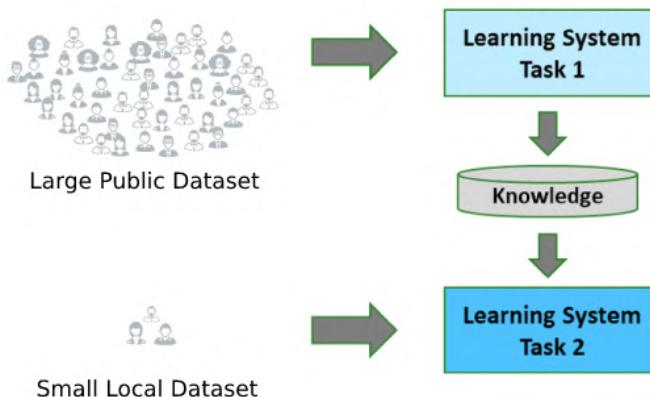


Figura 2.30: Aprendizaje por transferencia.

## 2.10. Reconocimiento de objetos

Reconocimiento de objetos - en *computer vision* - es una tarea que consiste en encontrar e identificar objetos en una imagen o secuencia de vídeo (*Figura 2.31*). Los humanos reconocen una multitud de objetos en imágenes con poco esfuerzo, a pesar del hecho que la imagen del objeto puede variar un poco en diferentes puntos de vista, en diferentes tamaños o escala e incluso cuando están trasladados o rotados. El objetivo es enseñar a una computadora a hacer lo que es natural para los humanos: obtener un nivel de comprensión de lo que contiene una imagen. Esto es un desafío para los sistemas de *computer vision*.



Figura 2.31: Reconocimiento de objetos.

## 2.11. Segmentación de imagen

La segmentación en el campo de la visión artificial es el proceso de dividir una imagen digital en varias partes (grupos de píxeles) u objetos. El objetivo de la segmentación es simplificar y/o cambiar la representación de una imagen en otra más significativa y más fácil de analizar. La segmentación se usa tanto para localizar objetos como para encontrar los límites de estos dentro de una imagen. Más precisamente, la segmentación de la imagen es el proceso de asignación de una etiqueta a cada píxel de la imagen de forma que los píxeles que comparten la misma etiqueta también tendrán ciertas características visuales similares (*Figura 2.32*) [36].



Figura 2.32: Segmentación de imagen.

Esta técnica da una comprensión más granular de los objetos en la imagen.

## 2.12. Algoritmo K-NN

El algoritmo clasifica cada dato nuevo en el grupo que corresponda, según tenga  $k$  vecinos más cerca de un grupo o de otro. Es decir, calcula la distancia del elemento nuevo a cada uno de los existentes y ordena dichas distancias de menor a mayor para ir seleccionando el grupo al que pertenece. Este grupo será, por tanto, el de mayor frecuencia con menores distancias.

El K-NN es un algoritmo de aprendizaje supervisado, es decir, que a partir de un juego de datos inicial su objetivo será el de clasificar correctamente todas las instancias nuevas.

En la *Figura 2.33* se puede ver que se intenta clasificar un elemento (cuadrado amarillo), en alguna de las dos clases, y que el resultado obtenido varía de acuerdo al valor asignado a  $k$ .

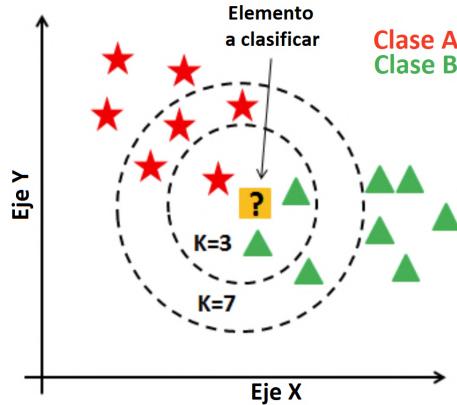


Figura 2.33: Algoritmo K-NN con  $k=3$  y  $k=7$ .

## 2.13. Trabajos relacionados al reconocimiento de objetos en estanterías.

### 2.13.1. Product Recognition in Store Shelves as a Sub-Graph Isomorphism Problem

El proyecto [37] plantea llevar a cabo el reconocimiento de objetos en góndolas a partir de planogramas. Un planograma es la representación gráfica del acomodo de mercancías o productos en un área específica de un establecimiento comercial, que puede ser una góndola, un expositor o un espacio seleccionado.

El autor plantea un *pipeline* (*Figura 2.34*) basado en *computer vision* que, dado el planograma y una imagen de los estantes observados, puede localizar correctamente cada producto, verificar si el posicionamiento de los productos cumple con el planificado y detectar elementos faltantes o extraviados.

En una primer etapa, se realiza una detección de objetos a partir de una imagen de una estantería, en la cual pueden existir errores que posteriormente serán corregidos. A partir de la detección realizada, se genera un grafo con los productos detectados llamado planograma observado. Luego de esto, se obtiene un subgrafo dado por el isomorfismo entre el planograma de referencia y el planograma observado. Esto permite encontrar nodos inconsistentes correspondientes a falsas detecciones de la primera etapa.



Figura 2.34: Sistema de reconocimiento a partir de planogramas.

Detectada la inconsistencia, pueden presentarse dos situaciones:

- Se generó una falsa detección, es decir, que en la región existe un producto diferente al establecido en el planograma observado.
- El producto realmente no se encuentra en stock.

Como conclusión, se establece que esta solución se puede aplicar en situaciones donde se tiene productos con superficies planas no reflectantes. Además, para cada estantería se debe generar un planograma de referencia, el problema de ellos es que los supermercados continuamente cambian la distribución de sus productos.

### 2.13.2. A deep learning pipeline for product recognition in store shelves

Este trabajo [38] realiza el reconocimiento de objetos en las góndolas de un supermercado, teniendo en cuenta una de las principales problemáticas del entorno: la gran cantidad de tipos de productos que existen en un supermercado, y el cambio constante del *packaging* de los mismos.

Lo que propone Alessio Tonioni es un *pipeline* (Figura 2.35) de varias etapas. En la primera de ellas, se encuentra una red CNN (detector), que dada una imagen de consulta de una góndola, pueda obtener un conjunto de cuadros delimitadores que se utilizarán como propuestas regionales en la siguiente etapa de reconocimiento. Idealmente, cada cuadro delimitador debe contener exactamente un producto, ajustarse perfectamente a la forma del mismo y proporcionar un puntaje que determine el nivel de confianza de la detección. El detector no determina un producto específico si no que establece una región donde puede existir un producto. De esta manera, se puede utilizar en tiendas que ofrecen distintos productos.

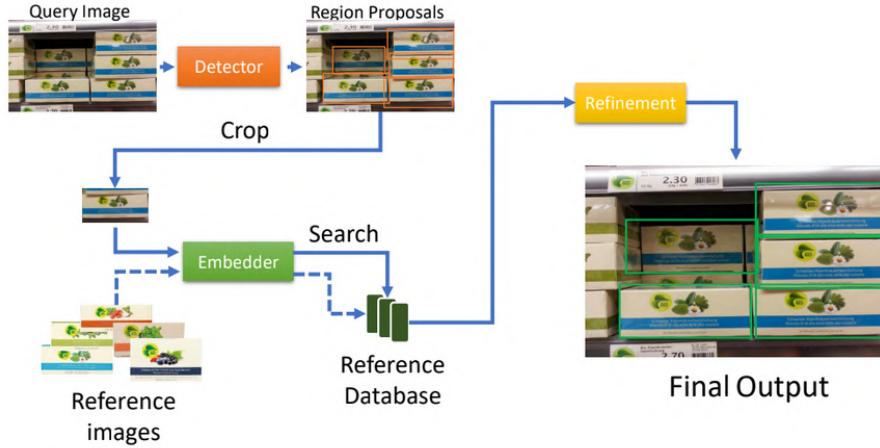


Figura 2.35: Pipeline de reconocimiento de productos de A. Tonioni.

En primer lugar, para realizar el reconocimiento la red *Embedder* caracteriza cada imagen de referencia disponible a través de un descriptor global (*features*) y así crea la base de datos de referencia de descriptores, asociados con los productos a reconocer. Luego, cuando se procesa una imagen de consulta, se calculan las *features* en cada una de las regiones candidatas recortadas de la imagen de consulta. Finalmente, para cada región candidata se calcula la distancia entre el descriptor de la región y cada descriptor de referencia, a través de un algoritmo *K-NN*. La red *Embedder* se implementa utilizando una red *VGG\_16* preentrenada sobre *Imagenet*.

En la última etapa, se combinan diferentes estrategias para realizar un paso de refinamiento final que ayuda a eliminar las detecciones falsas y evitar ambigüedades entre productos similares.

Por un lado, realiza un reacomodo del algoritmo *K-NN*. Para esto, se obtiene un peso a través de una fórmula que relaciona la distancia espacial de las *features* locales extraídas de la imagen con respecto al centro de la misma, con la distancia en el espacio de descriptores de las imágenes. Este peso será mayor cuando las *features* comparadas compartan la misma posición relativa con respecto al centro de la imagen, y tengan descriptores cercanos en el espacio de *features*. Finalmente, el primer *K-NN* es reacomodado a partir de las sumas de los pesos de la imagen.

Un segundo paso de refinamiento se da a través del criterio de relación de distancia, donde se calcula la relación entre las distancias en el espacio de *features*, del descriptor de la imagen de consulta con su 1-*NN* y 2-*NN*. Si el radio está por encima de un umbral determinado, el reconocimiento es considerado ambiguo y es descartado.

Por último, se clasifican los productos en subcategorías a partir de características en común, mejorando el porcentaje de acierto en la detección. Dadas las regiones candidatas extraídas de la imagen de consulta y el correspondiente conjunto de *K-NN*, se considera las 1-*NN* de la región con una alta confianza ( $> 0.1$ ) por parte del Detector, para encontrar la categoría principal de la imagen. En caso de que la mayoría de las detecciones voten por la misma categoría, es seguro asumir que el estante en la imagen contiene casi exclusivamente elementos de esa categoría, por lo tanto filtra el *K-NN* para todas las regiones candidatas en consecuencia.

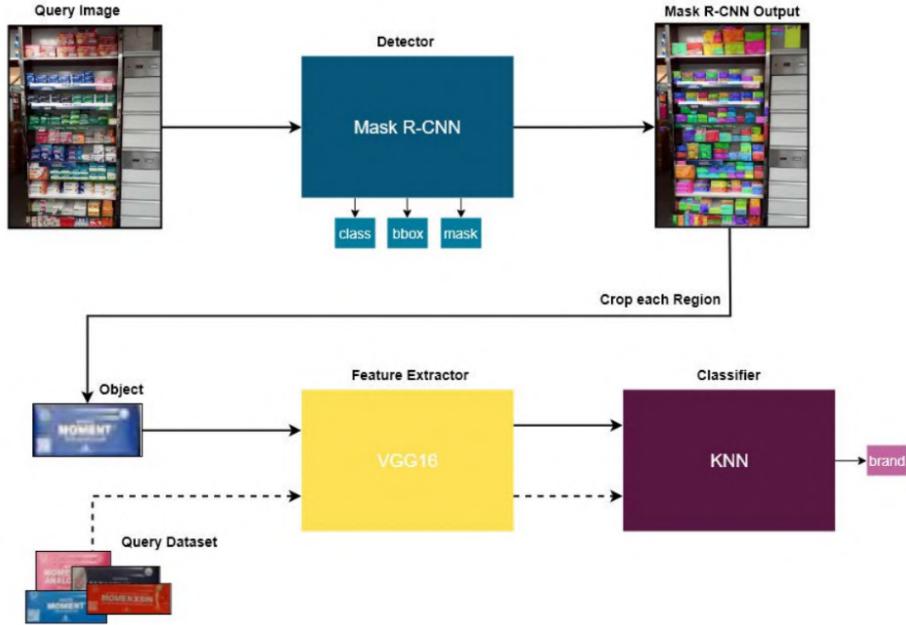
### 2.13.3. Retail Shelf Analytics Through Image Processing and Deep Learning

La investigación [39] se basa en los trabajos anteriormente nombrados de Alessio Tonioni.

En primer lugar, destaca que actualmente no hay un conjunto de datos que pueda adoptarse para el reconocimiento de productos en los estantes de las tiendas con técnicas de segmentación de instancias.

En este trabajo se lleva a cabo una implementación similar a la de A. Tonioni haciendo uso de una *Mask R-CNN*, una red *VGG16* y un algoritmo *K-NN* (*Figura 2.36*).

La implementación algorítmica de *Mask R-CNN* se basa en *TensorFlow* y, en particular, *Keras* para el desarrollo de alto nivel de la red neuronal.



*Figura 2.36: Pipeline de reconocimiento de productos de A. De Biasio.*

En particular, el algoritmo *Mask R-CNN* ha sido elegido por la posibilidad de calcular con mayor precisión la superficie expuesta para los diversos productos en las imágenes. Esto se debe en particular a las características del algoritmo para hacer predicciones en píxeles para cada objeto identificado en las imágenes.

La configuración final de la red incluye una red *backbone* convolucional *ResNet-50* con una topología FPN. Muchos de los parámetros de configuración RPN, se han determinado mediante la selección del modelo o análisis visual para maximizar la precisión.

Además de *Mask R-CNN*, se ha implementado un detector de las marcas de los productos, similar al de Tonioni. Esto explota el algoritmo *K-NN* para clasificar los objetos identificados por *Mask R-CNN* a nivel de marcas a través de un extracto de características basado en *VGG16*. En particular, las regiones extraídas por *Mask R-CNN* se envían a *VGG16* que extrae las características. Estos se comparan con las características de los productos previamente extraídos por *VGG16*, en un conjunto de datos de consulta a través del algoritmo *K-NN*.

El *dataset* utilizado consiste en 410 imágenes segmentadas en píxeles de alta resolución para un total de 8 categorías. El conjunto de datos se ha dividido en un conjunto de entrenamiento de 300 imágenes y uno de prueba con 110 imágenes.

El detector permite abstraer el contexto minorista y clasificar diferentes productos sin tener que poseer un conjunto de datos especializado con las anotaciones de las diferentes clases. Dado el número reducido de imágenes en el conjunto de datos, este enfoque es actualmente el único que puede clasificar objetos a nivel de marca en el dominio comercial.

### 3. Desarrollo

#### 3.1. Iteración 0: Decisiones preliminares

##### 3.1.1. Introducción

En primer lugar, en la siguiente iteración se presenta un diagrama preliminar del sistema a implementar. Luego de esto, se elige el sistema operativo a utilizar en la placa del robot y el lenguaje de programación de las rutinas del mismo.

Por otro lado, se elige un protocolo de transferencia entre el robot y la computadora de control.

Por último, se presenta el planeamiento de las actividades a realizar en las distintas iteraciones del proyecto, previamente habiendo optado por la metodología de trabajo iterativa.

##### 3.1.2. Requerimientos

Los requerimientos asociados a la presente iteración son los siguientes:

- **RF5:** Se debe permitir establecer una conexión remota inalámbrica entre el robot y la computadora de análisis.
- **RNF4:** La transmisión de datos entre el robot y la computadora de análisis de imagen se debe de manera cifrada. un medio cifrado.
- **RNF7:** En la placa base del robot se debe utilizar un Sistema Operativo *Open Source*.

##### 3.1.3. Desarrollo

###### 3.1.3.1 Arquitectura preliminar de alto nivel del sistema

A continuación se bosqueja el diagrama de arquitectura preliminar.

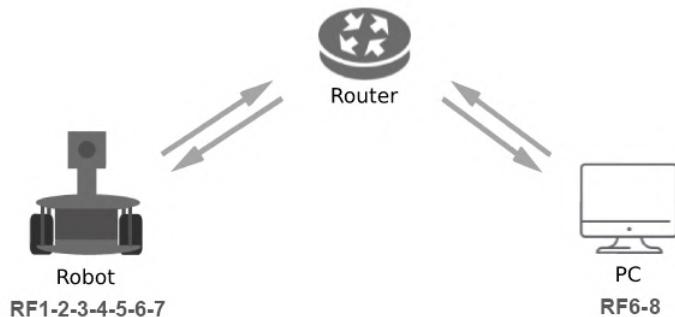


Figura 3.1: Diagrama de arquitectura preliminar.

En la *Figura 3.1* se diferencian los dos elementos principales del sistema: el Robot y la computadora de monitoreo, control y análisis de datos. A su vez, estos dos están compuestos por distintos elementos que interactúan entre sí, con el fin de satisfacer los requerimientos planteados anteriormente. También, se visualiza un *router* que permite la conexión de los dos elementos principales del sistema.

### 3.1.3.2 Elección del Sistema Operativo

Analizando el sistema operativo *Linux Core*, uno de los inconvenientes detectados fue que los *snaps* requieren más espacio que los paquetes de software clásicos debido a que deben guardarse varias bibliotecas en reiteradas ocasiones, y la memoria es un recurso muy limitado en la placa elegida. Otro de los motivos por lo cual no se optó por dicha alternativa fue que este sistema no se encuentra estable para su utilización [40].

Por otra parte, el principal motivo por el cual se descartó *Windows 10 IoT Core* fue porque no permite realizar cambios en el núcleo del mismo, siendo un limitante si eventualmente se requiere modificarlo.

**El sistema operativo elegido para utilizar en la placa es *Raspbian*.** Las ventajas son que es el SO oficial de *Raspberry*, es fácil de utilizar y además, el más utilizado, lo cual implica que se puede encontrar soporte rápidamente.

### 3.1.3.3 Elección del lenguaje de programación

El lenguaje elegido para trabajar es *Python*, debido a los motivos listados a continuación.

- Todos los códigos fuentes proporcionados por la empresa desarrolladora del robot se encuentran en dicho lenguaje.
- Es un lenguaje muy utilizado, por lo que existe una gran cantidad de librerías disponibles y mucha información de soporte.

### 3.1.3.4 Conexión entre robot y computadora remota

Al utilizar una placa *Raspberry Pi* con el sistema operativo *Raspbian*, se puede hacer uso de las aplicaciones comunes de sistemas *Linux*. Una de ellas es *ssh* (basada en el protocolo *SSH*), la cual brinda un acceso remoto a un servidor por medio de un canal seguro en el que toda la información está cifrada. Para realizar una conexión entre la computadora y el robot se ejecuta

```
1 | $ ssh pi@server
```

donde *pi* hace referencia al usuario a utilizar en la conexión, y *server* hace referencia a la dirección de destino (generalmente es una dirección *IP*).

Luego de realizar el *login* se puede ejecutar cualquier comando en el sistema operativo del robot. De esta manera está en condiciones de correr los *scripts* en *Python* que permiten hacer uso de todas las funcionalidades del robot.

### 3.1.3.5 Creación de repositorio asociado al proyecto

Para poder mantener el código asociado al proyecto en un sitio que asegure integridad, se creó un repositorio [41] en *Github*.

### 3.1.3.6 Elección de la metodología de desarrollo

Luego de haber analizado las distintas alternativas se optó por utilizar el Modelo Ágil Incremental. Los siguientes ítems son los que se tuvieron en cuenta a la hora de la toma de decisión.

- En este proyecto se tiene un contacto directo con el cliente, lo cual permite gestionar las expectativas del cliente de manera regular.
- Al ser un equipo de dos personas hay muy buena comunicación, lo que favorece a una toma de decisiones rápida.
- Se desea tener versiones "presentables" al finalizar cada iteración.

### 3.1.3.7 Planeamiento de Iteraciones

En la *Tabla 3.1* se muestra el planeamiento de las iteraciones, es decir, qué se debe realizar en cada una de ellas.

Iteración	Acciones
1	Elegir e implementar una alternativa para que el robot siga una trayectoria rectilínea.
2	Seleccionar e implementar una alternativa que le permita al robot retomar la trayectoria rectilínea definida en caso de perderla.
3	Configurar la cámara. Lograr que el robot tome fotografías mientras sigue una trayectoria establecida. Procesar las fotografías para lograr una única imagen de la góndola.
4	Realizar un <i>dataset</i> de productos a reconocer por el robot. Análisis e implementación de una red neuronal que tenga la capacidad de reconocer distintos tipos de productos. Análisis de resultados asociados al reconocimiento de los productos por parte de la red neuronal implementada.

Tabla 3.1: Plan de acciones asociado a cada iteración.

### 3.1.3.8 Matriz de trazabilidad de iteraciones

En la *Tabla 3.2* se da una vista general de cómo se deben cumplir los requerimientos funcionales a medida que se avanza en el proyecto, acorde a las iteraciones planificadas en la *Tabla 3.1*.

	RF1	RF2	RF3	RF4	RF5	RF6	RF7	RF8
It. 0								
It. 1								
It. 2								
It. 3								
It. 4								

Tabla 3.2: Plan de acciones asociado a cada iteración.

El color **rojo** indica que el requerimiento no estará cumplido, el **amarillo** que el requerimiento estará parcialmente cumplido, y el **verde** asegura que el requerimiento estará totalmente cumplido.

### 3.1.4. Resultados

#### 3.1.5. Riesgos superados

El análisis de mitigación de riesgos al finalizar la presente iteración es el mostrado en la *Tabla 3.3*.

ID	Riesgo	Probabilidad	Impacto	Exposición Inicio Iteración	Exposición Final Iteración
RI-01	<b>Sistema operativo incorrecto</b>	10 % ↓	3	0.9	0.3
RI-02	Incompatibilidad o avería de componentes	90 %	2	1.8	1.8
RI-03	<b>Elección incorrecta de escenario de prueba</b>	50 % ↓	3	2.1	1.5
RI-04	Modificación de los requerimientos del proyecto	50 %	3	1.5	1.5
RI-05	Dificultad en conseguir determinados componentes	50 %	1	0.5	0.5
RI-06	<b>Excesivo tiempo para cumplir los objetivos del proyecto</b>	55 % ↓	2	1.4	1.10
RI-07	Reducción de la fuerza de trabajo	10 %	4	0.4	0.4
RI-08	<b>Pérdida de información relacionada al proyecto</b>	5 % ↓	5	0.5	0.25

Tabla 3.3: Riesgos mitigados en iteración 0.

### 3.1.6. Conclusiones

A lo largo de esta iteración se lograron cumplir requerimientos elegidos para el proyecto, teniendo en cuenta los riesgos identificados. A partir de esto se pudo mitigar algunos de ellos:

- **RI-01:** La elección del sistema operativo *Raspbian*, permitió reducir la probabilidad de cometer un error en la elección de estos componentes. Elegir un SO popular con un gran contenido asociado disponible en la web, con un gran trabajo de testing sobre el mismo, permite asegurar un comportamiento estable. Además, es una distribución de *GNU/Linux* en la cual los integrantes del proyecto están familiarizados con su utilización.
- **RI-03:** Desarrollar un escenario de prueba acorde al espacio de trabajo final del robot permite reducir errores de implementación.
- **RI-06:** Haber definido el sistema operativo, la placa a utilizar, la metodología de desarrollo y el espacio de trabajo, permite establecer tiempos de trabajo y poder respetar los mismos. De esta manera se evita que el tiempo establecido se exceda.
- **RI-08:** Utilizar un repositorio remoto brinda la seguridad de que el proyecto se mantiene vigente a pesar de cualquier inconveniente en las computadoras de trabajo.

## 3.2. Iteración 1: Desplazamiento de robot

### 3.2.1. Introducción

En la presente iteración se hace foco en poder lograr que el robot se mantenga en una trayectoria recta. Se buscan distintas alternativas implementadas en la actualidad para el objetivo propuesto, analizando sus pros y contras.

### 3.2.2. Requerimientos

En esta iteración se atiende fundamentalmente los siguientes requerimientos.

- **RF1:** El robot debe poder seguir una trayectoria definida.
- **RF2:** En caso de perder la trayectoria definida, debe poder encontrarla nuevamente.
- **RNF3:** El espacio de trabajo del robot debe ser un ambiente cerrado, con determinadas condiciones que permita el normal funcionamiento.

### 3.2.3. Desarrollo

#### 3.2.3.1 Investigación y familiarización con el robot a utilizar

El robot a utilizar es un *Alphabot 2 Pi* de la empresa *WaveShare*, el cual fue armado y puesto a punto por uno de los integrantes del grupo en la Práctica Profesional de la carrera[6]. Debido a esto, el proceso de familiarización con el robot fue relativamente sencillo. Se probaron funcionalidades básicas, y además, se analizaron algunos códigos de ejemplo para tratar de comprender la forma en la cual trabajan los motores del robot.

#### 3.2.3.2 Análisis de alternativas para la trayectoria recta del robot

Uno de los objetivos fijados consistió en lograr implementar una alternativa para que el robot pueda avanzar de manera autónoma, siguiendo una trayectoria recta. Para esto se realizó una investigación de distintas alternativas que permitió realizar las siguientes tablas.

Alternativa	RFID
Descripción	Se deben colocar <i>tags RFID</i> en el suelo, generando una trayectoria que el robot debe seguir. Para poder reorientarse en caso de perder la trayectoria, deben colocarse al menos tres filas de <i>tags</i> , para saber hacia donde se desvió.
Ventaja	Utilizando esta técnica el entorno no afecta el funcionamiento como en el seguidor de línea (el cual por una mancha en el piso puede dejar de funcionar).
Desventaja	Para que sea efectivo los <i>tags</i> deben estar colocados muy próximos entre sí.
Elementos requeridos	Lectores <i>RFID</i> : 1 o 3 dependiendo la implementación. <i>Tags RFID</i> : se pueden utilizar <i>tags</i> programables (se le puede asignar un valor) o preprogramados (tienen un valor ya asignado que no puede modificarse).
Costo	Tanto el precio de los lectores como de <i>tags</i> depende si son programables o no (\$75 o \$30). Lector = \$530.

Tabla 3.4: Alternativa RFID.

Alternativa	Brújula electrónica
Descripción	Colocar brújula fija en el robot. Cada vez que se inicie el relevamiento de una góndola se debe colocar el robot en paralelo a la góndola, respetando el sentido. Una vez que el robot se encuentra en esta posición, comienza a desplazarse verificando mediante software que mantenga la dirección correcta.
Ventaja	Su utilización es fácil; no se requiere de muchos elementos para implementarla.
Desventaja	Si la góndola a relevar presenta alguna curvatura produciría un error. Cada vez que arranque el análisis de una góndola (recta) debe calibrar la brújula para establecer el grado al cual se desea mantener fijo. En un principio se debe colocar el robot paralelo a la góndola.
Elementos requeridos	Brújula electrónica <i>HMC5883</i> .
Costo	Se cuenta con stock en el laboratorio.

Tabla 3.5: Alternativa Brújula electrónica.

Alternativa	Mapeo de espacio utilizando Kinect
Descripción	Utilizar un sensor Kinect permite generar un mapa del espacio de trabajo del robot. Posee un sensor de profundidad que posibilita generar la habitación en 2D en cualquier condición de luz ambiental. El sensor se coloca delante del robot y permite identificar la distancia hacia una determinada góndola.
Ventaja	No es un método invasivo que requiere modificar el ambiente de trabajo para poder ser utilizado. Permite tener conocimiento con precisión del entorno de trabajo del robot.
Desventaja	Con este método, el robot no puede saber cuál es su posición actual en el mapa generado, por lo que se requiere de un método complementario de localización. Se requiere un procesamiento computacional importante para realizar el análisis y la generación del mapa. Requiere tiempo su puesta a punto, lo que puede generar retrasos en los plazos del proyecto.
Elementos requeridos	Sensor <i>kinect de XBox 360</i> .
Costo	Se cuenta con stock en el laboratorio.

Tabla 3.6: Alternativa Kinect.

Alternativa	Seguidor de línea
Descripción	Se debe colocar en paralelo a las góndolas una línea de color (negro preferiblemente). Solo se debería colocar el robot al comienzo de la góndola, y el mismo se movería siguiendo dicha línea.
Ventaja	El robot ya dispone de un seguidor de línea. Utilizando esta técnica, el robot puede pasar de una góndola a otro sin mayor complicación. Solo debe haber una continuación de la línea. Se podría hacer un circuito de tal forma que el robot circule por todas las góndolas, sin la intervención del humano. No se requiere realizar ninguna modificación al robot.
Desventaja	Debe haber un contraste significativo entre el color suelo y la línea colocada. En un supermercado (ambiente en que se utilizará el robot) es muy posible que el suelo se manche. Esto produce que el funcionamiento no sea como el esperado. Si el robot pierde la línea, la convergencia hacia la misma es casi imposible.
Elementos requeridos	Solo se debe colocar una línea en el supermercado en paralelo a las góndolas.
Costo	Se cuenta con sensores seguidores de línea.

Tabla 3.7: Alternativa Seguidor de línea.

Alternativa	Sensores de distancia
Descripción	Colocar uno o dos sensores de distancia a los costados del robot. Inicialmente se debe colocar al robot de tal manera que los dos sensores obtengan un valor similar. Realizado esto, se puede comenzar con el movimiento del mismo verificando constantemente la distancia entre el robot y la góndola.
Ventaja	La implementación no es difícil. Ya se tiene experiencia previa con este tipo de sensores, lo cual no representa un gran desafío.
Desventaja	Siempre a la altura en la que estén los sensores debe haber una parte lisa en la góndola, que permita la correcta medición. Se debe verificar que los sensores realizan bien su tarea cuando el robot se encuentra en movimiento.
Elementos requeridos	Uno o dos sensores de distancia (ultrasónico).
Costo	Se cuenta con stock en el laboratorio.

Tabla 3.8: Alternativa Sensores de distancia.

Alternativa	Encoder
Descripción	Consiste en realizar mediciones de la rotación de cada motor para calcular desviaciones en trayectorias.
Ventaja	Posibilita tener conocimiento con precisión del giro de cada rueda con lo cual, se puede realizar cálculos y corregir errores para que el robot se mueva en línea recta.
Desventaja	La gran desventaja de este método es que no es posible verificar cuando una rueda queda patinando, lo cual generaría un error que sería muy difícil de corregir. Por otra parte, el robot del cual se dispone no cuenta con encoders en los motores.
Elementos requeridos	Encoder Rotativo <i>Ky-040</i> .
Costo	\$ 78.

Tabla 3.9: Alternativa encoder.

Alternativa	Giroscopio
Descripción	Este dispositivo brinda la posibilidad de detectar variaciones de velocidades angulares y de rotación. Un giroscopio se coloca paralelo a la dirección de movimiento del robot para poder detectar variaciones en ángulos de giro, y luego corregir la trayectoria.
Ventaja	Permite saber con gran precisión el ángulo de giro del robot en cualquier instante, y corregir este ángulo para que continúe en línea recta.
Desventaja	El eje del giroscopio tiende a ir corriéndose con el tiempo lo cual termina introduciendo errores.
Elementos requeridos	Modulo Acelerómetro Giróscopo <i>Mpu6050</i> 9 Ejes.
Costo	\$119.

Tabla 3.10: Alternativa Giroscopio.

### 3.2.3.3 Elección de alternativa

En primer lugar, el método de guiado a través de *RFID* (*Tabla 3.4*) fue descartado ya que el alcance de las antenas convencionales es de muy corta longitud. Adquirir antenas de largo alcance generaría un costo adicional muy difícil de afrontar en el proyecto.

Además, se debe tener en cuenta que habría que etiquetar la trayectoria del robot, colocando etiquetas *RFID* cada una cierta distancia. Esto sería un trabajo extra y otro costo a afrontar.

No se tuvo en cuenta la brújula electrónica (*Tabla 3.5*) ya que solo permite generar trayectorias rectas, en cuanto el robot ingrese en una curva, su orientación comenzaría a cambiar y sería muy difícil de recalcular la trayectoria.

El sensor *Kinect* de la *XBox 360* (*Tabla 3.6*) es un dispositivo muy útil, de hecho fue utilizado por estudiantes de la carrera para la realización del robot *Hermes III* [42]. El inconveniente que surge al utilizar este método es que requiere de un gran poder de computó que supera al brindado por la placa utilizada. Además, necesita de una fuente de energía de mayor tamaño, que el robot elegido no puede transportar.

La utilización de *encoders* (*Tabla 3.9*) no es posible de implementar en nuestro proyecto, ya que el robot elegido no cuenta con motores que posean chip *encoders* para medir el ángulo de giro de la rueda. Además, en caso de realizar modificaciones para agregarlos a la estructura del robot, los *encoders* poseen la desventaja de que si las ruedas quedan patinando se cometería un error en la

medición, ya que hace creer que están avanzando, cuando en realidad se mantienen en el mismo lugar.

El giroscopio electrónico (*Tabla 3.10*) posee la desventaja de ir introduciendo de a poco errores en sus mediciones. Además, al igual que la brújula, cuando el robot deba recorrer zonas con curvaturas no es útil.

Luego de analizar cada una de las alternativas, comparar las ventajas, desventajas, elementos requeridos y costos, se llegó a la conclusión de que era necesario realizar una combinación de dos métodos para así lograr mejores resultados.

Por un lado, se dedujo de que el seguidor de línea (*Tabla 3.7*) es una muy buena opción, debido a que el robot cuenta con 5 sensores de este tipo, y a que hay implementado un algoritmo para ellos.

Por otra parte, se decidió utilizar dos sensores de distancia HC-SR04 (*Tabla 3.8*), los cuales miden el tiempo que tarda en viajar (ida y vuelta) una onda. El robot adquiere la capacidad de seguir trayectorias predefinidas con anterioridad, permitiendo hacer un recorrido por las góndolas sin inconvenientes. Además, los dos sensores permiten mantenerse siempre a una distancia correcta de la góndola de interés, lo cual es algo fundamental para la cámara que debe ir capturando imágenes.

### 3.2.3.4 Determinación de espacio de pruebas

Teniendo en cuenta los requerimientos del proyecto, se decidió diseñar un espacio de pruebas que permite simular el recorrido del robot a través de la estantería de un supermercado.

En la *Figura 3.2* se puede observar el espacio de pruebas elegido. Se realizan algunas simplificaciones, por ejemplo suponer una estantería recta donde el robot no deba realizar curvas. Además, se observa que el robot se desplaza a una distancia cercana a los productos.

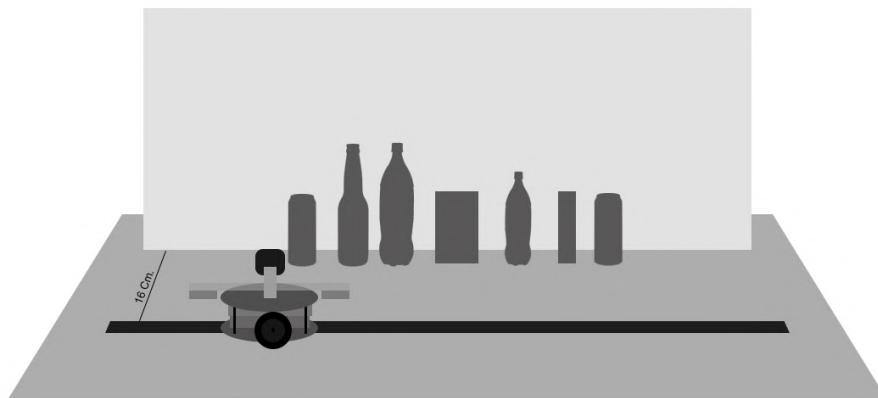


Figura 3.2: Espacio de trabajo elegido.

### 3.2.3.5 Implementación de la alternativa

Teniendo en cuenta lo comentado, se procedió a realizar la implementación de la alternativa elegida.

Se hace uso de los 5 sensores fotoeléctricos de reflexión infrarroja ITR20001/T que posee integrados el robot, los cuales permiten detectar líneas en superficies de alto contraste (por ejemplo, suelo blanco y línea negra).

Por otro lado, se utilizan dos sensores de distancia HC-SR04. Tener dos sensores permite obtener la distancia del robot a la góndola, tanto desde la parte de adelante como desde la parte de atrás. Si

estas medidas coinciden, el robot se mantiene en trayectoria recta. En caso de una variación de estas medidas, se conoce hacia donde se ha desviado el robot para luego poder corregirlo.

De esta manera, el robot posee dos datos de referencia con respecto a la distancia a la góndola, y también, tiene su trayectoria definida a través de la línea negra.

Se representa en la *Figura 3.3* el resultado de la alternativa elegida.

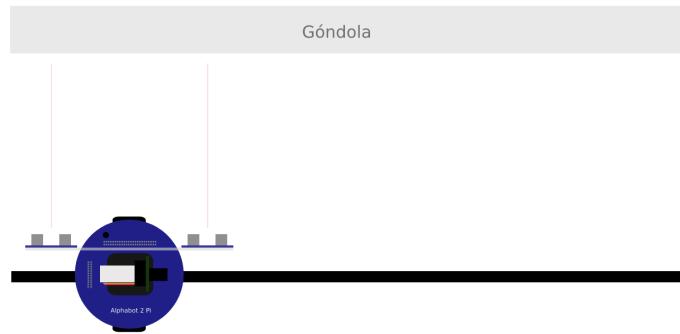


Figura 3.3: Implementación de la alternativa elegida.

### 3.2.3.6 Seguidor de línea

El robot *Alphabot 2 Pi* cuenta con 5 sensores fotoeléctricos de reflexión infrarroja *ITR20001/T*, los cuales permiten detectar una línea negra sobre fondo blanco o viceversa.

Para la implementación del seguidor de línea, se utilizó una librería realizada por la empresa creadora del robot, *WaveShare*, que consta de 2 archivos en *Python*: *TRSsensor.py* y *LineFollower.py*.

El archivo *TRSsensor.py* contiene la lógica de funcionamiento del seguidor de línea. En él se encuentran 4 métodos fundamentales:

- **AnalogRead()**, obtiene los valores de los sensores. Los valores devueltos son una medida de la reflectancia en unidades abstractas, con valores más altos correspondientes a una reflectancia más baja (por ejemplo, una superficie negra).
- **Calibrate()**, lee los sensores 10 veces y usa los resultados para la calibración. Los valores máximos y mínimos encontrados se almacenan internamente y se utilizan para el método siguiente.
- **readCalibrated()**, devuelve los valores leídos por cada sensor en un rango entre 0 y 1000, donde 0 hace referencia al menor valor de calibración, y 1000 al máximo.
- **readLine()**, además de devolver los valores obtenidos por **readCalibrate()**, devuelve una posición estimada del robot con respecto a la línea.

Como se comentó anteriormente, el método **readLine()** devuelve una posición estimada del robot con respecto a la línea, lo cual es un dato fundamental ya que a partir de éste, se puede ir modificando la potencia en los motores para que siempre se mantenga sobre la línea.

La estimación se realiza utilizando un promedio ponderado de los índices del sensor multiplicado por 1000, de modo que un valor de retorno de 0 indica que la línea está directamente debajo del sensor 0, un valor de retorno de 1000 indica que la línea está directamente debajo del sensor 1, 2000 indica que está debajo del sensor 2, etc. Por otro lado, los valores intermedios indican que la línea está entre dos sensores. Lo anteriormente descripto se ve reflejado en la *Figura 3.4*.

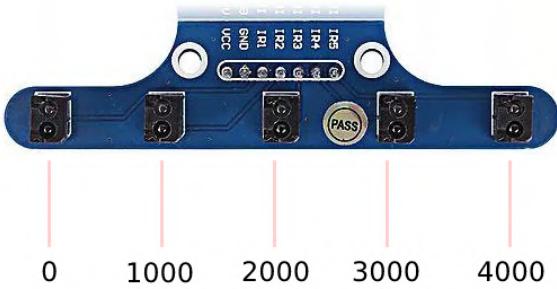


Figura 3.4: Sensores de seguidor de línea y estimaciones asociadas.

Por otra parte, el archivo `LineFollower.py` permite crear una instancia de `TRSsensor`, y utilizar los métodos anteriormente descriptos. Su secuencia de ejecución es la siguiente:

1. Rutina de calibración, donde se hace girar el robot 180° para obtener muestras de la superficie de trabajo y poder obtener valores máximos y mínimos de cada sensor. Se hace uso del método `Calibrate()` de `TRSsensor`.
2. Una vez calibrado, comienza la rutina de detección y corrección de posición, donde el robot utiliza el método `readLine()` para obtener posición estimada con respecto a la línea y valor de los sensores.

En caso de que alguno de los sensores este exactamente sobre la línea (valor medido mayor a 900), no se modifica la potencia de los motores, es decir, el robot no realiza un giro.

En caso de que el robot no tenga ningún sensor sobre la línea, lo que se busca es posicionar al sensor 2 (el del medio) encima de la línea, para lo cual se implementa un controlador PID [43] que consta de tres parámetros:

- *Proporcional* depende del error actual. Si la línea se desvía a la derecha, se gira a la derecha y al revés. Esto se hace de forma proporcional: cuanto mayor es el desvío más se debe girar, es decir, a mayor error, mayor corrección.
- *Integral* hace referencia a que en algunas ocasiones el control proporcional no es capaz de hacer una corrección suficiente para llevar el valor de error a cero, por ejemplo, en una curva que se va cerrando poco a poco. Lo que se hace es tratar de averiguar si ese error se está acumulando sin ser corregido. La medida de ese error acumulado se obtiene de la suma de los errores en cada paso del bucle. Es una medida del error del pasado.
- *derivador* surge de comparar el error en cada ciclo del bucle con el anterior, para ver si está subiendo o está bajando, y de ese modo tener información sobre el error del futuro. Con esa diferencia se puede hacer una tercera corrección, la cual ayuda a responder a variaciones muy rápidas del error.

Se utiliza la suma de estas tres acciones para ajustar al proceso por medio de un elemento de control, que en este caso, es la posición actual del robot.

El algoritmo busca centrar al robot siempre bajo el sensor 2 (el del medio) encima de la línea, ya que esto permite mantenerlo perfectamente sobre la trayectoria deseada.

La rutina comparará la posición estimada (rango de 0-4000) con la deseada (2000), y en base a esta diferencia, calculará el proporcional.

$$proporcional = posicion - 2000 \quad (3)$$

Se obtiene el derivativo:

$$derivativo = proporcional - lastproporcional \quad (4)$$

Por último, se obtiene el integrador:

$$integral+ = proporcional \quad (5)$$

Con estos valores definidos, se procede a obtener la potencia a aplicar a los motores con la fórmula:

$$powerDifference = proporcional / 20 + integral / 10000 + derivativo * 6 \quad (6)$$

Cabe destacar que los valores asociadas a la fórmula se pueden variar para mejorar la performance del seguidor de línea. En este caso, se obtuvieron mejores resultados con los valores 20, 1000 y 6.

Si *powerDifference* es positivo, el robot debe girar hacia la derecha, en caso contrario debe hacerlo hacia la izquierda. La magnitud definirá el ángulo del giro.

### 3.2.3.7 Sensores de distancia

Los sensores de distancia son implementados con dos sensores HC-SR04, los cuales poseen un emisor de onda (*Trigger*) y un receptor (*Echo*). Se envía un pulso, y se cuenta el tiempo en que el *Echo* detecta el rebote de ese pulso sobre la superficie a medir (*Figura 3.5*).

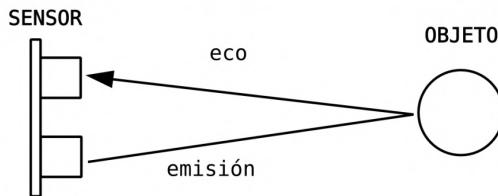


Figura 3.5: Funcionamiento de sensor HC-SR04.

Un detalle a tener en cuenta es que las entradas y salidas de la placa *Raspberry Pi* trabajan con tensiones de 3,3 V, mientras que el sensor *HC-SR04* con 5V. Para poder utilizarlos, se hace uso de un divisor resistivo. El circuito para los sensores se ve en la *Figura 3.6*.

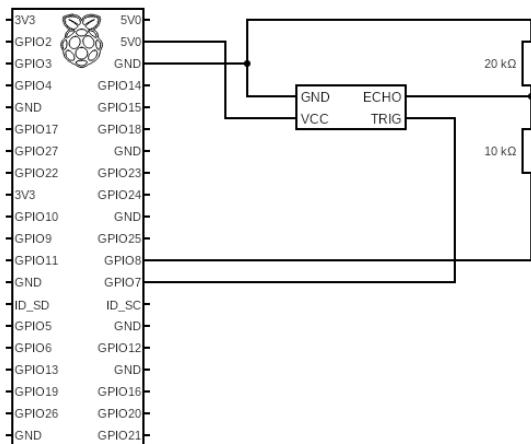


Figura 3.6: Circuito asociado a cada sensor HC-SR04.

Un problema que se encontró en esta etapa fue que todos los pines de la placa estaban utilizados por Hardware del robot. Esto implicó dejar de utilizar algún dispositivo Hardware, para poder realizar la conexión de los sensores de distancia. Se elaboró una tabla en la cual se indica qué funcionalidad utiliza cada pin GPIO. Con esta se determinó dejar de utilizar el "Joystick", ya que no es indispensable para el sistema y permite liberar cinco pines de los cuales se ocupan cuatro para los sensores y uno quedará disponible para futuras necesidades. En la *Tabla 3.11* se presentan las funcionalidades asignadas a los pines, ya con los cambios realizados para conectar los sensores de distancia.

GPIO	Funcionalidad	GPIO	Funcionalidad	GPIO	Funcionalidad
GPIO 2	Motor	GPIO 11	Libre	GPIO 20	Motor
GPIO 3	Motor	GPIO 12	Motor	GPIO 21	Motor
GPIO 4	Buzzer	GPIO 13	Motor	GPIO 22	Motor
GPIO 5	ADC	GPIO 14	UART	GPIO 23	ADC
GPIO 6	Motor	GPIO 15	UART	GPIO 24	ADC
GPIO 7	Sensor de distancia	GPIO 16	Motor	GPIO 25	ADC
GPIO 8	Sensor de distancia	GPIO 17	Motor	GPIO 26	Motor
GPIO 9	Sensor de distancia	GPIO 18	Led RGB	GPIO 27	Motor
GPIO 10	Sensor de distancia	GPIO 19	Motor		

Tabla 3.11: Funcionalidades asignadas a pines GPIO.

Otro detalle importante es que los sensores tienen un error en la medición, y este error es particular de cada sensor, por lo que se debe realizar una estimación del mismo para corregir las mediciones.

Para corregir el error se realizaron mediciones de 0 a 50 centímetros y se observaron los valores reales y los valores leídos por cada sensor. Colocando las medidas tomadas por el sensor y las medidas reales en un sistema cartesiano, se ve que la relación entre ambas en una función lineal dada por:

$$\text{medidaObtenida} = \text{medidaReal} * x + b \quad (7)$$

El valor de  $x$  y  $b$  se obtiene realizando regresión lineal entre los pares de datos. Para el sensor 1 se calculó el error porcentual de cada medición, mostrado en la *Figura 3.7*.

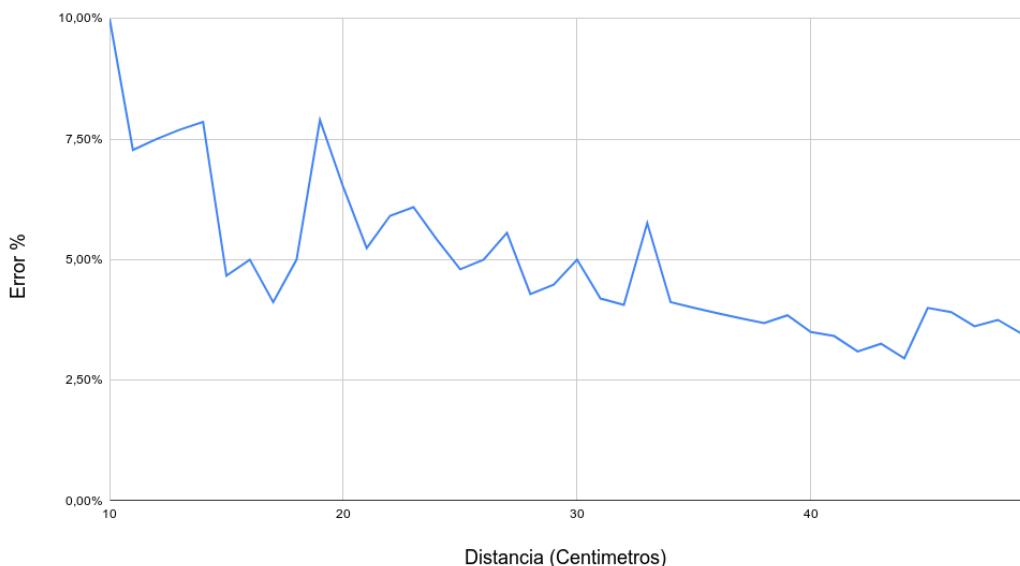


Figura 3.7: Errores porcentuales obtenidos de sensor 1.

Se observa que a medida que la distancia crece, el error se reduce.

Luego de realizar este proceso para ambos sensores, se obtiene que la pendiente para el sensor 1 es 0.9840 y su ordenada es -0.864. Por otra parte, para el sensor 2 la pendiente es 0.9810, y su ordenada es -0.477.

Cabe destacar que por cuestión de presentación no se muestra el análisis de corrección para el otro sensor.

Una vez que se obtienen los datos se procede implementar el algoritmo asociado a estos sensores. Saber la distancia, tanto en la parte delantera como trasera del robot hacia la góndola, permite tomar una decisión con respecto al giro que deben realizar los motores.

En la *Figura 3.8* se visualizan los dos casos posibles que se pueden presentar.

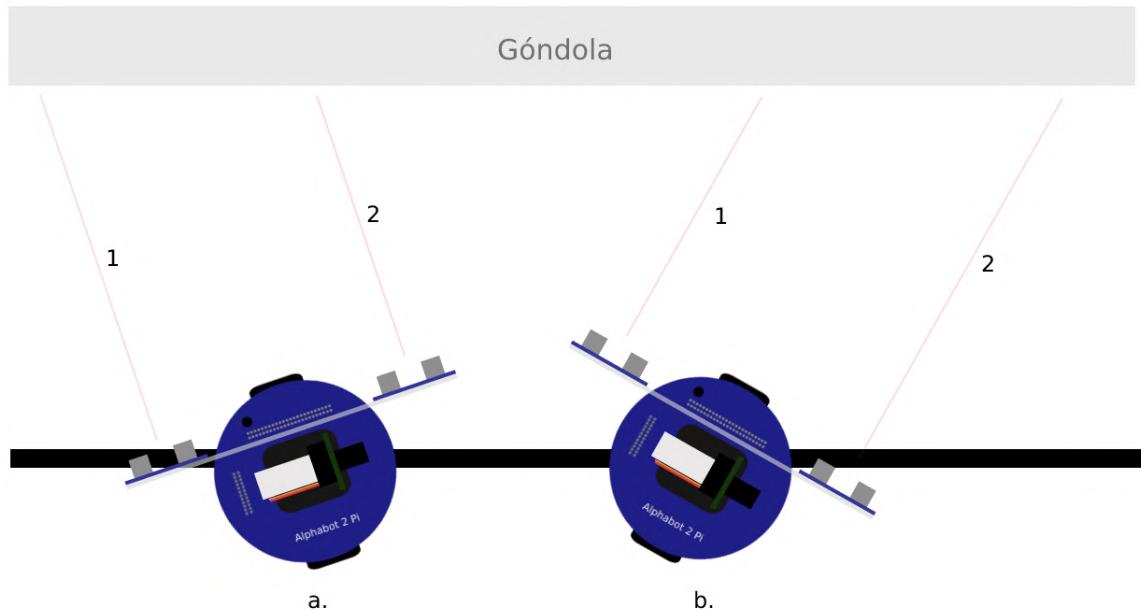


Figura 3.8: Casos de acción de sensor HC-SR04.

En el caso *a* se deduce que el sensor 1 mide una mayor distancia hacia la góndola que el sensor 2, por lo que se determina que el robot debe realizar un giro hacia la derecha para posicionarse paralelo a la góndola.

Por el contrario, en el caso *b*, el sensor 2 es el que mide una mayor distancia y se debe realizar un giro hacia la izquierda para orientarse correctamente.

Esta alternativa requiere que la línea a seguir siempre esté en paralelo a la góndola. Poder saber la distancia permite mantenerse siempre en paralelo, aún en caso de que la línea por algún motivo no pueda ser detectada, lo cual aumenta la fiabilidad del robot.

### 3.2.4. Resultados

Se ha definido al seguidor de línea como nuestra alternativa principal para generar una trayectoria rectilínea del robot. En diversas pruebas realizadas se logró que el robot avance sin problemas sobre una línea negra de 1,5 centímetros de ancho.

Además, se eligió un método complementario de guiado a través de sensores ultrasónicos. Luego de colocar dos sensores, se obtuvieron mediciones con respecto a la distancia a la góndola para poder calibrarlos por software. Cabe destacar que si entre la superficie a calcular la distancia y el sensor se

genera un ángulo mayor a la mitad del medición efectiva (30), el valor obtenido será erróneo. A partir de esto, se plantea la necesidad de agregar más sensores con distinto ángulo.

Una vez logrado esto, se definieron los casos posibles de acción de este sistema de guiado a implementarse en la siguiente iteración.

### 3.2.5. Riesgos superados

El análisis de mitigación de riesgos al finalizar la presente iteración es el presentado en la *Tabla 3.12*.

ID	Riesgo	Probabilidad	Impacto	Exposición Inicio Iteración	Exposición Final Iteración
RI-01	Sistema operativo incorrecto del robot	10 %	3	0.3	0.3
RI-02	<b>Incompatibilidad o avería de componentes</b>	70 % ↓	2	1.8	1.4
RI-03	<b>Elección incorrecta de escenario de prueba</b>	40 % ↓	3	1.5	1.2
RI-04	Modificación de los requerimientos del proyecto	50 %	3	1.5	1.5
RI-05	<b>Dificultad en conseguir determinados componentes</b>	40 % ↓	1	0.5	0.4
RI-06	Excesivo tiempo para cumplir los objetivos del proyecto	55 %	2	1.1	1.1
RI-07	Reducción de la fuerza de trabajo	10 %	4	0.4	0.4
RI-08	Pérdida de información relacionada al proyecto	5 %	5	0.25	0.25

Tabla 3.12: Riesgos mitigados en iteración 1.

### 3.2.6. Conclusiones

En esta iteración se mitigaron algunos riesgos:

- **RI-02:** Elegir los sensores de seguidor de línea que posee la placa, testear su funcionamiento y comprobar que su desempeño es correcto, redujo el riesgo de incompatibilidad de componentes.
- **RI-03:** Definir el método seguidor de línea permitió continuar con el diseño y la puesta a punto del escenario de prueba. En este sentido, se implementó la línea y la góndola objetivo con éxito.
- **RI-05:** El riesgo de no poder obtener componentes se vio reducido notablemente ya que por una parte, el robot contaba con sensores seguidores de línea y además, el laboratorio de Arquitectura de Computadoras contaba con los sensores de ultrasonido necesarios.

### 3.3. Iteración 2: Optimización de mecanismo de reorientación

#### 3.3.1. Introducción

En esta iteración se lleva adelante el reemplazo de la batería utilizada en el robot, para asegurar mayor autonomía.

Por otra parte, se agregan y configuran más sensores de distancia en el robot solucionando el inconveniente detectado en la iteración anterior.

#### 3.3.2. Requerimientos

En la presente iteración se atiende los siguientes requerimientos.

- **RF2:** En caso de perder la trayectoria definida, debe poder encontrarla nuevamente.
- **RNF1:** El robot debe tener una autonomía superior a 2 horas.
- **RNF2:** El robot no debe generar contaminación acústica, ni debe representar un riesgo para las personas que se encuentren en el ambiente de trabajo del mismo.

#### 3.3.3. Desarrollo

##### 3.3.3.1 Búsqueda de alternativa a batería actual

El robot cuenta con una batería conformada por dos pilas 14500 3.6V, de 800mAh cada una. Estas pilas se colocan en serie generando un voltaje máximo de 7.2V, el cual luego es regulado con el integrado LM2596 para obtener los 5V necesarios por la *Raspberry Pi*. El circuito implementado en el robot para dicha tarea es el mostrado en la *Figura 3.9*.

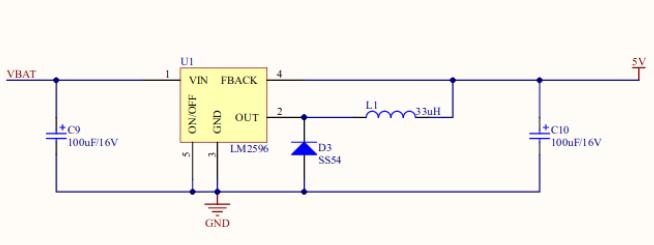


Figura 3.9: Circuito implementado para regular voltaje de batería.

Las pilas poseen la gran desventaja de tener una baja cantidad de mAh, con lo cual el tiempo de trabajo es muy corto (2 hs. máximo), debiendo ser recargadas frecuentemente. Esto vuelve tedioso y lento el proceso de desarrollo y prueba de avances en el robot. Una solución a este inconveniente es utilizar baterías de 3.7V y 4000mAh que permite trabajar durante periodos de tiempo mayores a las pilas que el robot posee.

Colocando dos baterías en serie se obtiene el voltaje requerido para que el robot funcione, cuadruplicando los mAh de las pilas. Luego de analizar la viabilidad de esta alternativa se procedió a la implementación de la misma.

##### 3.3.3.2 Refactorización de sensores de distancia

Un inconveniente que se presenta al utilizar sensores de distancia, se da cuando los valores obtenidos por los sensores no son correctos, ya que el ángulo formado entre ellos y la pared es mayor al ángulo ( $15^\circ$ ) de detección efectiva (*Figura 3.10*).

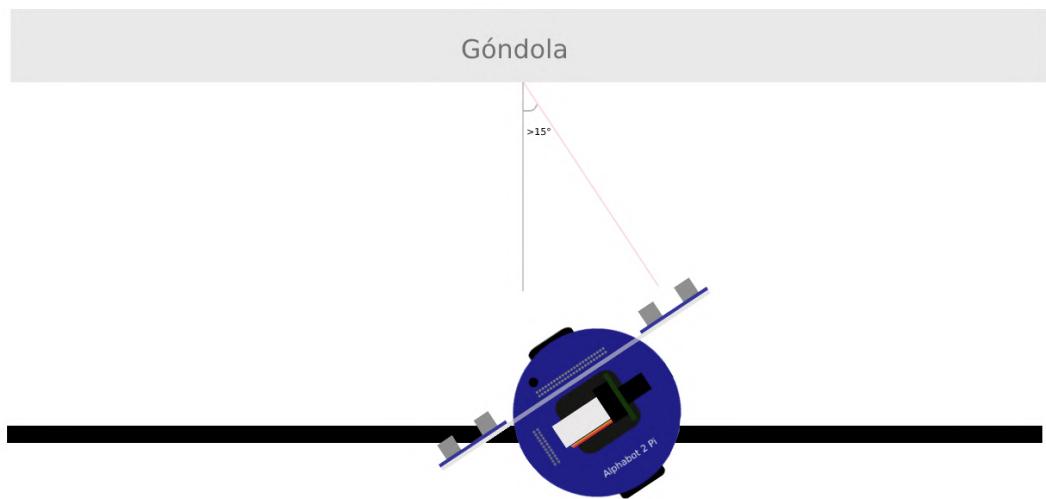


Figura 3.10: Error en medición de distancia.

Esta problemática fue planteada en una reunión con los docentes a cargo del proyecto, en la cual se concluyó que la manera óptima de resolverla era colocando dos sensores más, formando un ángulo de 30° con respecto a los sensores iniciales. Esto se muestra en la *Figura 3.11*, donde también se observa la numeración utilizada para cada uno de los sensores en la presente iteración.

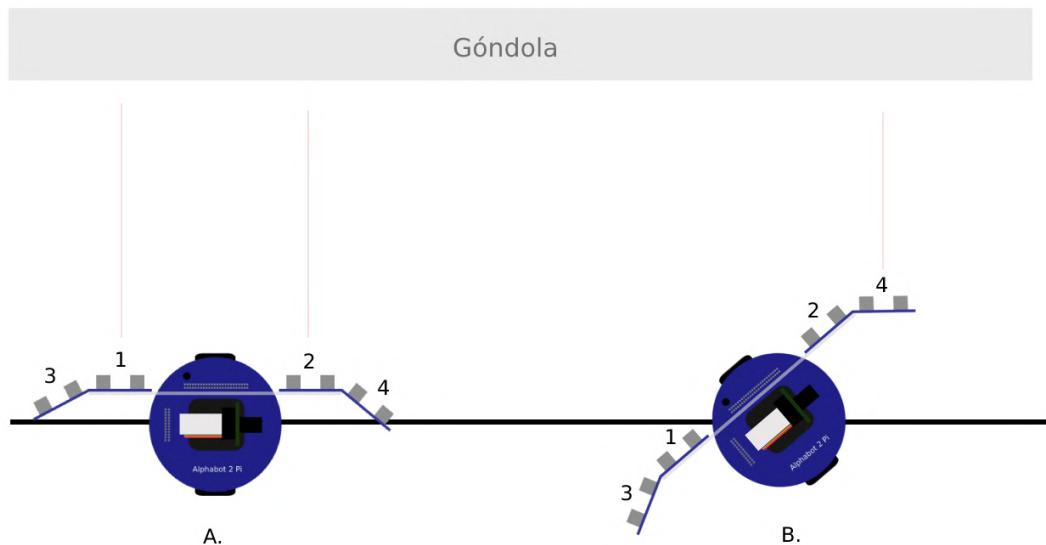


Figura 3.11: Casos de análisis general.

Tal como se explicó en la iteración anterior, uno de los inconvenientes del robot es la falta de pines. Como en este caso se necesita cuatro pines más para poder controlar los nuevos sensores, se decidió eliminar el *buzzer* y utilizar los pines de la *UART*. En la *Tabla 3.13*, se presentan las funciones de los pines actualizadas.

GPIO	Funcionalidad	GPIO	Funcionalidad	GPIO	Funcionalidad
GPIO 2	Motor	GPIO 11	Sensor de distancia	GPIO 20	Motor
GPIO 3	Motor	GPIO 12	Motor	GPIO 21	Motor
GPIO 4	Sensor de distancia	GPIO 13	Motor	GPIO 22	Motor
GPIO 5	ADC	GPIO 14	Sensor de distancia	GPIO 23	ADC
GPIO 6	Motor	GPIO 15	Sensor de distancia	GPIO 24	ADC
GPIO 7	Sensor de distancia	GPIO 16	Motor	GPIO 25	ADC
GPIO 8	Sensor de distancia	GPIO 17	Motor	GPIO 26	Motor
GPIO 9	Sensor de distancia	GPIO 18	Led RGB	GPIO 27	Motor
GPIO 10	Sensor de distancia	GPIO 19	Motor		

Tabla 3.13: Tabla de pines.

Una de las conclusiones a la que se llegó con el grupo de trabajo, es que idealmente el robot debe tener a su alrededor al menos 6 sensores de distancia, cada uno apuntando hacia una dirección, para obtener mayor información del entorno. Una mejora a esta alternativa requiere que cada sensor se encuentre sobre un servo, el cual permite rotar el campo de acción del sensor. Estas alternativas si bien fueron pensadas y analizadas, no se aplicaron para poder continuar con el desarrollo del proyecto en los tiempos pactados.

### 3.3.3.3 Integración de sensores de distancia con seguidor de línea

El objetivo principal de integrar el algoritmo de seguimiento de línea con el algoritmo encargado de los sensores *HC-SR04*, es el de encontrar una solución en casos donde el robot no pueda detectar la línea.

Mientras detecta la línea, el robot realiza su funcionamiento normal, haciendo uso de los 5 sensores seguidores de línea. El problema surge cuando ninguno de estos sensores detecta la línea, y cualquier movimiento que se realice en sentido equivocado lleva al robot a una divergencia.

Se plantearon las siguientes situaciones en las cuales los sensores de distancia actúan:

1. El robot se encuentra sobre la línea pero rotado sobre la misma, en consecuencia ningún sensor puede detectar la línea (*Figura 3.12*).
2. El robot se encuentra fuera de la línea, en una posición ubicada entre la línea y la góndola (*Figura 3.13*).
3. El robot está posicionado entre la línea y el exterior (*Figura 3.14*).
4. El robot está posicionado fuera de la línea, paralelo a la góndola, pero girado unos 180°, de tal forma que los sensores de distancia miden una distancia mayor a un umbral establecido. (*Figura 3.15*).

Para el primer caso, se hacen uso de los sensores colocados en los extremos (3 y 4), los cuales permiten medir correctamente la distancia hacia la góndola. Una vez detectado cual sensor midió la menor distancia, se gira el robot en el sentido necesario. En este punto, los seguidores de línea deben detectarla nuevamente. En el caso que no la detecten, nuevamente se mide y se realiza el giro con el sentido correspondiente.

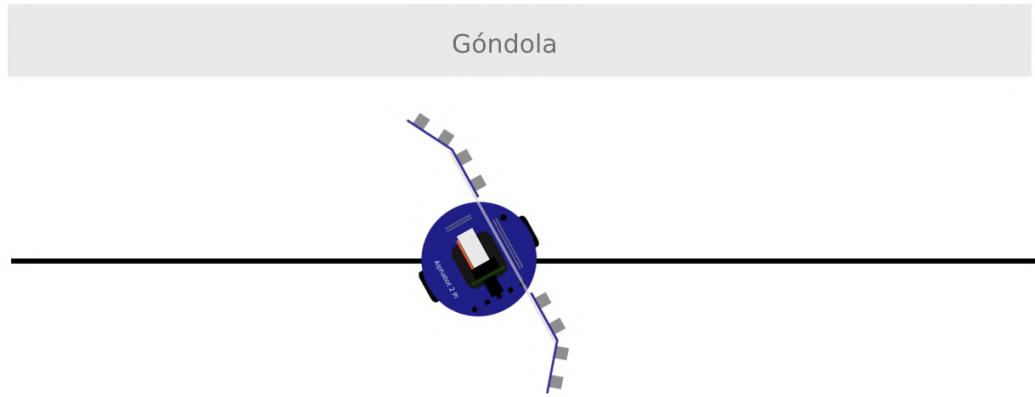


Figura 3.12: Caso 1: Robot sobre la línea.

En el segundo caso, el robot se encuentra entre la góndola y la línea, con lo cual debe girar hacia su derecha hasta encontrarla nuevamente. La distancia entre la línea y la góndola es un valor conocido y establecido en el código, con lo cual, si cambia la distancia debe actualizarse dicho valor. En este caso, se realiza el giro correspondiente y luego se avanza para aproximarse a la línea. En caso que no se logre en un intento, se realiza nuevamente hasta encontrarla.

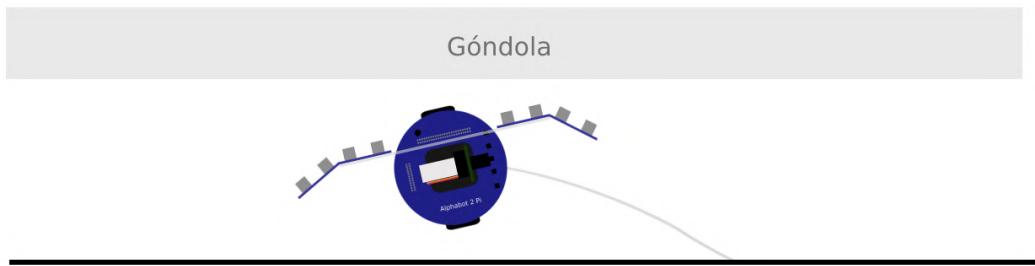


Figura 3.13: Caso 2: Robot entre la góndola y la línea.

El tercer caso es similar al segundo, solo que esta vez el robot gira en el sentido contrario. Cabe destacar que la distancia máxima de medición está definida en 100 centímetros, luego de esto se genera una advertencia, y el robot detiene su movimiento.

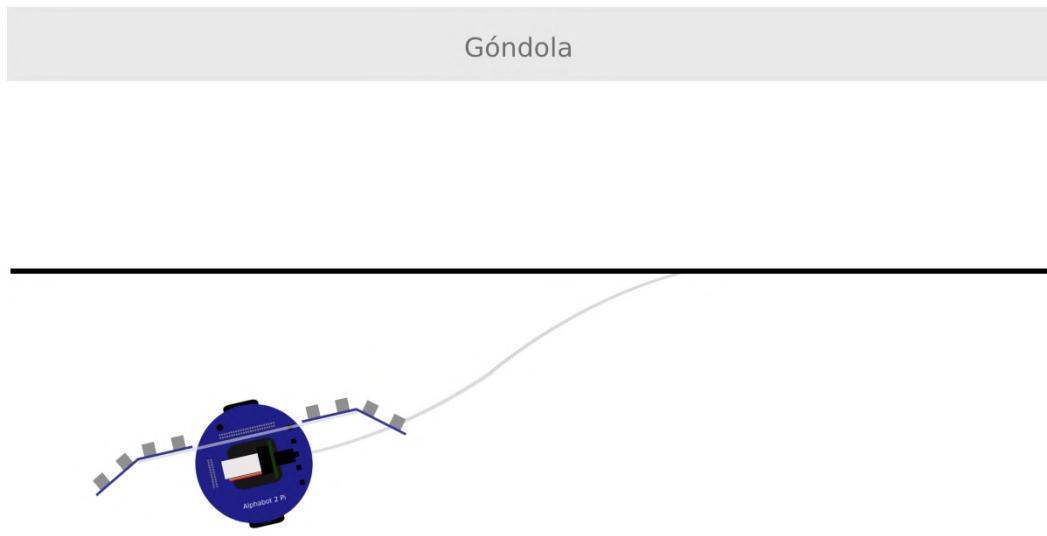


Figura 3.14: Caso 3: Robot más allá de la línea.

Considerando el último caso, este se produce cuando todos los sensores de distancia miden más de 100 centímetros. Cuando eso ocurre, el robot gira 180° posicionando al robot en el sentido correcto, para luego continuar el movimiento de acuerdo al caso 2 o 3, retornando a la línea.

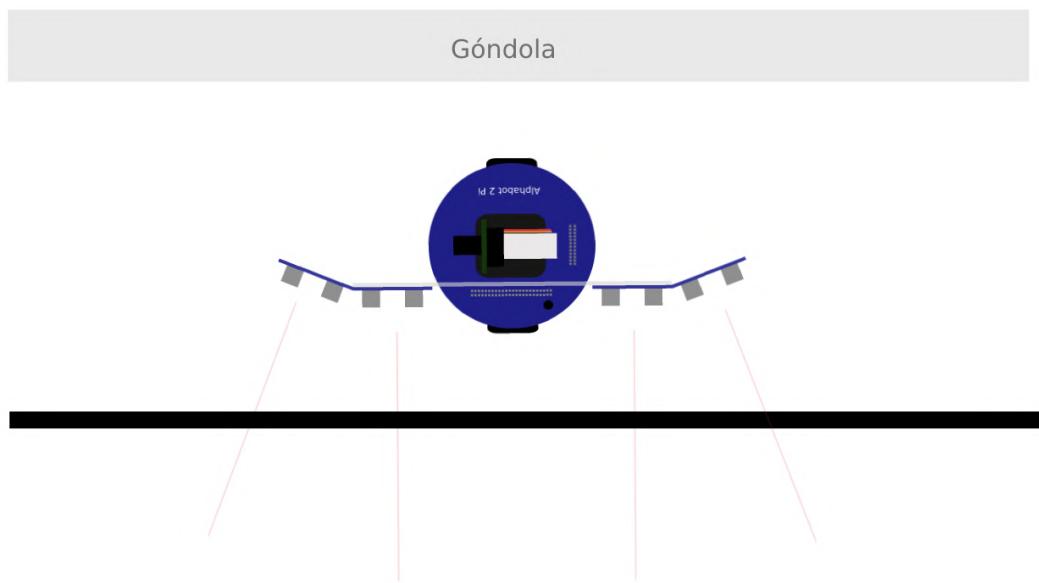
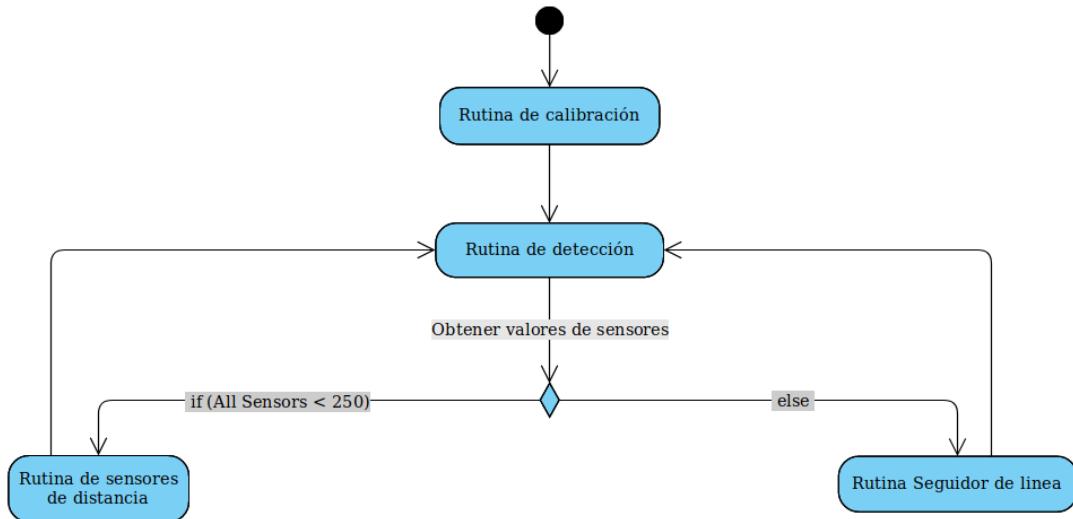


Figura 3.15: Caso 4: Robot girado completamente.

#### 3.3.3.4 Descripción de algoritmos utilizados

El movimiento del robot está determinado por dos algoritmos fundamentales: el algoritmo encargado del seguidor de línea y el de los sensores de distancia.

En la *Figura 3.16* se puede observar como se complementan uno con el otro para poder cubrir todas las eventuales situaciones que el robot puede atravesar.



*Figura 3.16:* Diagrama de actividad general.

En principio, el robot ingresa en una rutina de calibración en la cual realiza un giro de 180° sobre la línea, obteniendo valores que dependen de la reflectancia de la superficie. A partir de estos valores se van a poder calibrar los siguientes y corregir errores en la detección.

Realizada la calibración, el robot comienza a avanzar obteniendo nuevos valores de los sensores de línea. Si todos los valores son menores a 250 (en una escala que va 1 a 1000), significa que el robot está sobre una superficie que no se corresponde con una línea negra. En este caso, el programa ingresa en la rutina correspondiente a los sensores de distancia. En caso contrario, ingresa a la rutina encargada del seguidor de línea.

### 3.3.3.5 Rutina de sensores de distancia

La rutina encargada de los sensores de distancia se visualiza en la *Figura 3.17*. Cabe destacar que algunos valores en el análisis que se presenta, están ligado al espacio de pruebas elegido para el robot. En primer lugar, se obtienen los valores de los cuatro sensores de ultrasonido. Una vez obtenidos se incrementa la variable perdido que indica que el robot ha perdido la línea. Luego, se evalúan las siguientes situaciones:

- Si perdido ha llegado a 3, el robot evalúa si la distancia mínima es mayor o menor a 15; si es menor, el robot se encuentra entre la góndola y la línea y debe girar a la derecha, por el contrario, si es mayor gira a la izquierda ya que se encuentra entre la línea y el exterior. Una vez girado, el robot avanza para acercarse a la línea. Esta subrutina hace referencia a los casos 2 y 3 anteriormente descriptos.
- Si todos los sensores obtienen valores mayores a 100, significa que el robot está totalmente girado con respecto a la góndola (Caso 4), con lo cual tiene que realizar un giro de 180° hacia su izquierda.
- Si el sensor 4 es el que mide la menor distancia, se gira el robot hacia la izquierda para poder orientarlo nuevamente sobre la línea (Caso 1).

- Al igual que el caso anterior, si el sensor 3 obtiene la menor medición, girará hacia la derecha.
- En el caso que el valor mínimo sea obtenido por alguno de los sensores internos (sensor 1 y 2), se obtiene un promedio de ambas mediciones y se evalúa si esta medición es menor a 17 centímetros (distancia entre línea y góndola) o mayor a 18.5 centímetros (distancia entre exterior de línea y góndola). En el primer caso, el robot deberá girar hacia la derecha, y en el segundo hacia la izquierda. Esto hace referencia a lo planteado en el caso 3.

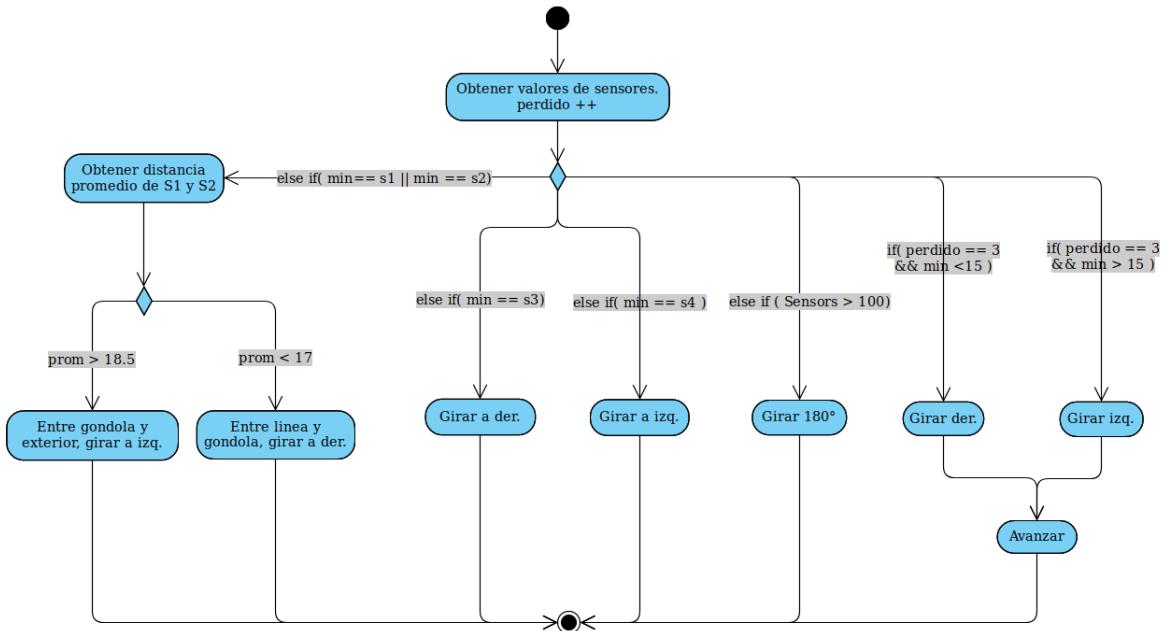


Figura 3.17: Diagrama de actividad de rutina de sensores ultrasonido.

### 3.3.3.6 Extracción de buzzer de placa principal

Uno de los requerimientos no funcionales establece que el robot no debe generar contaminación acústica en el espacio de trabajo. Sin embargo, en la versión actual de la placa del *AlphaBot* se cuenta con un *buzzer* que emite un sonido en una cierta frecuencia que es molesto para el oído humano. Por la configuración establecida en los pines, este *buzzer* se mantiene encendido todo el tiempo que el robot se encuentre activo.

Teniendo en cuenta lo anterior, se procedió a la extracción del *buzzer* de la placa principal, realizando el procedimiento con éxito.

### 3.3.4. Riesgos superados

En la *Tabla 3.14* se muestra el análisis de mitigación de riesgos al finalizar la presente iteración.

ID	Riesgo	Probabilidad	Impacto	Exposición Inicio Iteración	Exposición Final Iteración
RI-01	Sistema operativo incorrecto del robot	10 %	3	0.3	0.3
RI-02	<b>Incompatibilidad o avería de componentes</b>	60 % ↓	2	1.4	1.2
RI-03	Elección incorrecta de escenario de prueba	40 %	3	1.2	1.2
RI-04	Modificación de los requerimientos del proyecto	50 %	3	1.5	1.5
RI-05	<b>Dificultad en conseguir determinados componentes</b>	30 % ↓	1	0.4	0.3
RI-06	Excesivo tiempo para cumplir los objetivos del proyecto	55 %	2	1.1	1.1
RI-07	Reducción de la fuerza de trabajo	10 %	4	0.4	0.4
RI-08	Pérdida de información relacionada al proyecto	5 %	5	0.25	0.25

Tabla 3.14: Riesgos mitigados en iteración 2.

### 3.3.5. Resultados

En esta iteración se logró definir e implementar el sistema de movimiento del robot, agregando nuevos sensores y reescribiendo el algoritmo asociado a los mismos. Por último, se puede destacar que se reemplazaron las pilas tradicionales del robot por baterías, lo cual permite duplicar la autonomía del robot.

### 3.3.6. Conclusiones

En esta iteración se mitigaron los siguientes riesgos:

- **RI-02:** Agregarle dos sensores de ultrasonido y comprobar su correcto funcionamiento permite reducir este riesgo.
- **RI-05:** El riesgo de no poder obtener componentes se vio reducido debido a que se contaba con stock de sensores ultrasónicos en el laboratorio.

### 3.4. Iteración 3: Obtención y transmisión de imágenes

#### 3.4.1. Introducción

En la siguiente iteración se seleccionan dos cámaras para *Raspberry Pi* a partir del análisis previo realizado sobre cámaras disponibles. Luego de esto, se llevan adelante diferentes pruebas para poder determinar con cual de las dos se obtiene un mejor rendimiento.

Posteriormente, se define una configuración de cámara (velocidad de disparo, ISO, iluminación necesaria) para obtener imágenes con la mayor calidad posible, estando el robot en movimiento.

Realizado esto, se implementa un algoritmo que posibilita unir todas las imágenes obtenidas de la góndola, formando una sola.

Por último, se integra el código de captura de imágenes con el algoritmo seguidor de línea y de los sensores de distancia, para que el robot vaya obteniendo imágenes mientras recorre la trayectoria establecida.

#### 3.4.2. Requerimientos

En la presente iteración se atienden los siguientes requerimientos.

- **RF6:** El robot debe obtener imágenes sin detener su desplazamiento.
- **RNF5:** El robot debe tomar fotos cada un tiempo  $t$  preestablecido.

#### 3.4.3. Desarrollo

##### 3.4.3.1 Análisis de cámaras compatibles con el robot

Luego de analizar las distintas alternativas (ver 2.5), las cámaras *Raspberry Pi NoIR v2* y *Kuman* fueron descartadas debido a que su uso está orientado a trabajar en la oscuridad. Por otro lado, la cámara *Raspberry Pi* con lente ojo de pez también fue descartada, ya que al tomar imágenes deforma los objetos, lo cual puede ser un problema para las futuras detecciones de los productos. **En definitiva, se seleccionaron las cámaras *Raspberry Pi (B)*, y *Raspberry Pi Module v2* para realizar diversas pruebas y ver cual es la mejor alternativa a utilizar.** En la *Tabla 3.15* se realiza la comparación de las dos cámaras seleccionadas.

	Raspberry Pi Camera (B) 2.0	Raspberry Pi Camera Module v2
Resolucion fija	5 megapixeles	8 megapixeles
Tamaño optico	1/4"	1/4"
Sensor	OV5647	Sony IMX219
Resolucion del sensor	1920×1080 píxeles (FULL HD)	3200×2464 píxeles (QUXGA)

Tabla 3.15: Comparación de cámaras elegidas.

##### 3.4.3.2 Pruebas de cámaras seleccionadas

Debido a que el código del proyecto se realiza en lenguaje *Python*, se buscó una librería que permita hacer uso de la cámara. La librería utilizada es *Picamera* [44], la cual provee de una interfaz en *Python* muy completa que permite modificar una gran cantidad de parámetros asociados al periférico.

Utilizando el escenario de prueba como referencia, se fue modificando el tiempo de exposición de disparo y el ISO hasta lograr capturar buenas imágenes en movimiento.

Vale aclarar que una cámara para obtener imágenes en movimiento con fidelidad, debe contar con una gran cantidad de luz.

### Raspberry Pi Camera (B) Rev 2.0

Al medir la cantidad de luz del ambiente de prueba que requiere la cámara para obtener imágenes con una calidad aceptable, se obtuvo 800 lúmenes. Sin embargo, lo ideal sería contar con al menos 2000 lúmenes.

Fue variándose el tiempo de exposición desde 1/410 segundos hasta 1/33 segundos, como también la sensibilidad ISO desde 100 hasta 700 para saber en donde se encontraron los mejores resultados. Cabe destacar que la distancia entre el robot y los productos es de aproximadamente 40 centímetros.



Figura 3.18: T exp. 1/410 - ISO 100.

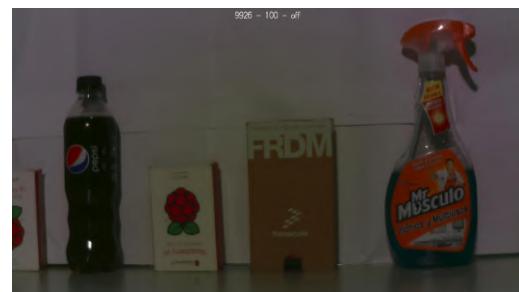


Figura 3.19: T exp. 1/101 - ISO 100.

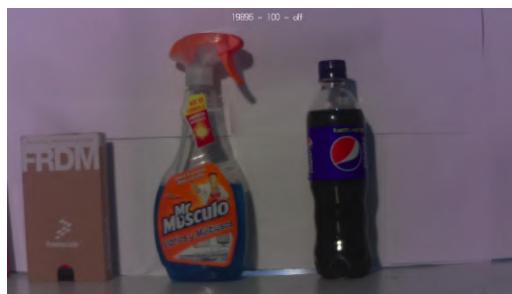


Figura 3.20: T exp. 1/50 - ISO 100.

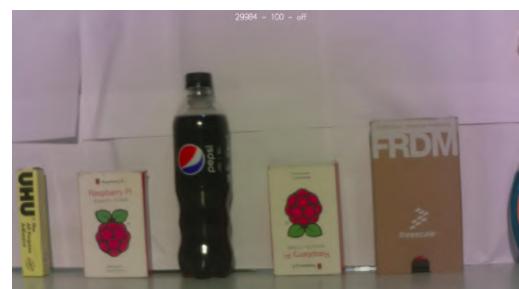


Figura 3.21: T exp. 1/33 - ISO 100.

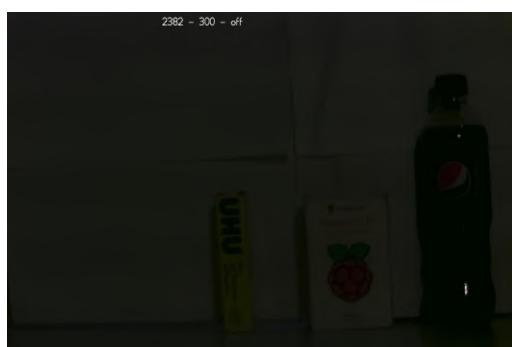


Figura 3.22: T exp. 1/410 - ISO 300.

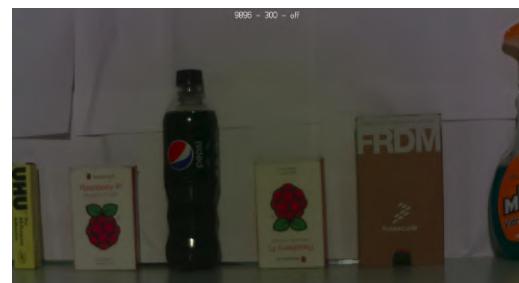


Figura 3.23: T exp. 1/101 - ISO 300.



Figura 3.24: T exp. 1/50 - ISO 300.



Figura 3.25: T exp. 1/33 - ISO 300.

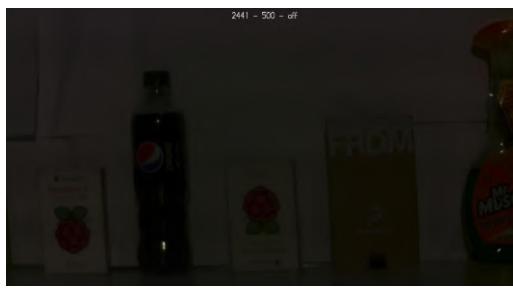


Figura 3.26: T exp. 1/410 - ISO 500.

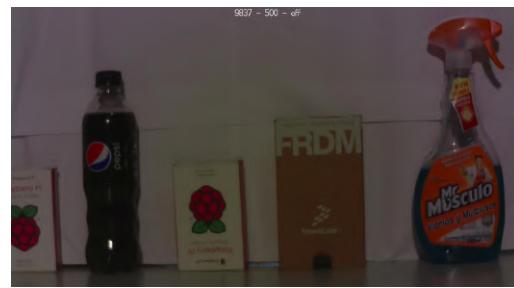


Figura 3.27: T exp. 1/101 - ISO 500.



Figura 3.28: T exp. 1/50 - ISO 500.



Figura 3.29: T exp. 1/33 - ISO 500.

A partir de distintas pruebas realizadas (*Figura 3.18 - 3.29*) queda en evidencia que para cualquier ISO, la mejor imagen capturada con el robot en movimiento se obtiene con un tiempo de exposición entre 1/50 y 1/33 segundos. También se puede observar que aumentar el ISO permite obtener fotos con más iluminación, pero se debe tener en cuenta que también crece el ruido presente en la imagen. Por este motivo, se selecciona 400 como el ISO óptimo.

Es importante destacar que estos parámetros van a depender de la distancia del robot al objetivo, de la cantidad del luz en el ambiente, de la velocidad con la que avanza el robot, entre otros.

### Raspberry Pi Camera Pi Module v2

Por otro lado, se realizaron las mismas pruebas con la *Camera Pi Module v2* como se muestra en la *Figura 3.30 - 3.41*.

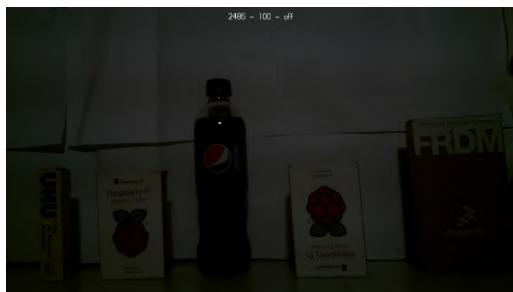


Figura 3.30: T exp. 1/410 - ISO 100.



Figura 3.31: T exp. 1/101 - ISO 100.



Figura 3.32: T exp. 1/50 - ISO 100.



Figura 3.33: T exp. 1/33 - ISO 100.



Figura 3.34: T exp. 1/410 - ISO 300.



Figura 3.35: T exp. 1/101 - ISO 300.



Figura 3.36: T exp. 1/50 - ISO 300.

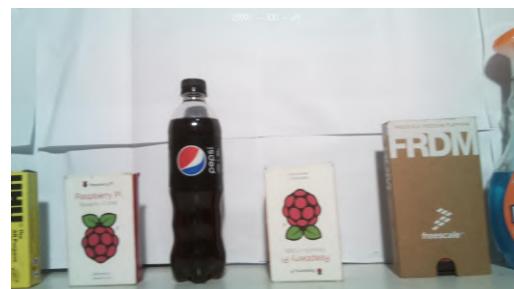


Figura 3.37: T exp. 1/33 - ISO 300.

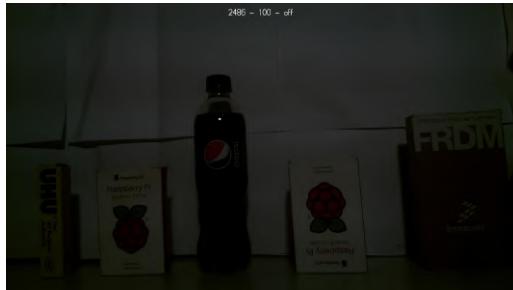


Figura 3.38: T exp. 1/410 - ISO 500.



Figura 3.39: T exp. 1/101 - ISO 500.



Figura 3.40: T exp. 1/50 - ISO 500.



Figura 3.41: T exp. 1/33 - ISO 500.

Se observa que las mejores capturas se logran con un tiempo de exposición cercano a 1/50 segundos al igual que en la cámara anteriormente probada. Además, la sensibilidad ISO produce el mismo efecto permitiendo obtener imágenes con mayor iluminación, considerándose óptimo un valor de 500.

#### 3.4.3.3 Elección de la cámara

Luego de observar las características de las dos cámaras elegidas para *Raspberry* (Tabla 3.15), y teniendo en cuenta los resultados obtenidos en las pruebas realizadas, se tomó la decisión de elegir la cámara *Raspberry Pi Camera Module v2*. Esta cámara permite obtener fotos en movimiento con una calidad mayor, donde los productos se perciben con gran definición.

#### 3.4.3.4 Elección de método de transmisión de imágenes

Se realiza un análisis de distintas alternativas para llevar a cabo la transmisión de las imágenes capturadas por el robot hacia la computadora de procesamiento.

- Transmisión de imagen vía red en el momento que la foto es capturada.
- Transmisión de imágenes vía red luego de capturar una góndola completa, donde cada vez que se recorra una góndola el robot se detiene y envía el conjunto de imágenes asociado a la misma.
- Transmisión de imágenes vía red cada un determinado tiempo, donde el conjunto de imágenes obtenido hasta el momento se envía hacia la computadora mientras el robot continua con su tarea de desplazamiento y captura.
- Almacenamiento de las imágenes en memoria externa a medida que son capturadas. Luego de la adquisición, se extrae la memoria para su posterior transferencia a la computadora de procesamiento.

Inicialmente, se descartó la cuarta alternativa, ya que requiere personal que realice la tarea de extracción del medio de almacenamiento del robot, y su posterior transferencia hacia la computadora. Este proceso se contrapone a la idea de automatización del sistema.

Las restantes alternativas tienen en común que la transferencia se realiza a través de la red. Se llevó a cabo un análisis para determinar si a partir de la compresión de las imágenes, existe una mejora en los tiempos de transferencia. Cabe destacar que se trabajó con una resolución 1920x1080, siendo el tamaño de cada imagen de 1,1 MB aproximadamente, y la transferencia se realizó a través del protocolo *SCP*. Para realizar la compresión se realizó un *script* en *Python* utilizando la librería *Pillow*, donde se reduce la calidad de la imagen un 30% obteniendo una notable disminución en el tamaño del archivo. En la *Tabla 3.16* se explicitan los tiempos que conlleva cada uno de los casos.

	Imagen original	Imagen comprimida
Tamaño de archivo	1.1 MB	53 KB
Tiempo de compresión	-	0.34495 segundos
Tiempo de transferencia	1.5 segundos	0.3 segundos
Tiempo total	1.5 segundos	0.64495 segundos

Tabla 3.16: Comparación de transferencia de imágenes con y sin compresión.

En la *Figura 3.42* y *3.43* se ve que la calidad de la imagen continua siendo aceptable para su posterior análisis luego de realizar la compresión.



Figura 3.42: Imagen sin comprimir.



Figura 3.43: Imagen comprimida.

Del análisis realizado se concluye que se presenta una mejora significativa en los tiempos de transmisión, realizando la compresión de las imágenes. Lo anterior implica que además de requerir un hilo que se encargue de la transmisión de las imágenes, se necesita uno encargado de la previa compresión. Esto llevó a desistir del primer método y del tercero ya que, teniendo en cuenta que se utilizan dos hilos para el desplazamiento del robot y la captura de las imágenes, el hardware disponible no tiene la capacidad de agregar dos nuevos hilos para la compresión y transmisión. Una posible solución a este inconveniente es realizar la implementación en la placa *Raspberry Pi 4*.

La segunda opción es la elegida debido a que no requiere de personal para su funcionamiento, y no genera una sobrecarga de procesamiento en la placa del robot. Si bien se optó por este método, la solución óptima es la primer opción ya que permitiría realizar un análisis en tiempo real.

### 3.4.3.5 Generación de imagen de la góndola

Uno de los problemas a los que se enfrenta es que no se puede tener dos imágenes diferentes donde se capture el mismo producto, ya que al intentar detectarlo automáticamente se reconocerían dos unidades de un producto del cual solo existe una.

Para solucionar este problema, se lleva a cabo un procedimiento denominado *image stitching*, el cual consta de superponer imágenes que poseen puntos en común, logrando una imagen de mayor resolución.

Se realiza una implementación en *Python* y *OpenCV* del modelo descripto en el Capítulo 2.7.1 debido a que el mismo posee la ventaja de poder generar una panorámica aunque exista rotación, zoom o cambio de iluminación entre las imágenes de entrada. Además el algoritmo permite ingresar las fotos de manera desordenada y automáticamente identifica la correspondencia y el orden. Lo anteriormente nombrado es la mayor ventaja con respecto a otros métodos como los que utilizan *curvas de Harris*.

Luego de obtener las fotos, se procede a realizar el mapeo general, es decir, la obtención de una imagen que represente al total de la góndola.

Para realizar lo anteriormente nombrado, primero se obtienen imágenes cada un determinado tiempo  $t$ . Para exemplificar, se obtienen las siguientes 4 imágenes con el robot en movimiento (*Figura 3.44-3.47*).



Figura 3.44: Imagen en movimiento 1.



Figura 3.45: Imagen en movimiento 2.



Figura 3.46: Imagen en movimiento 3.

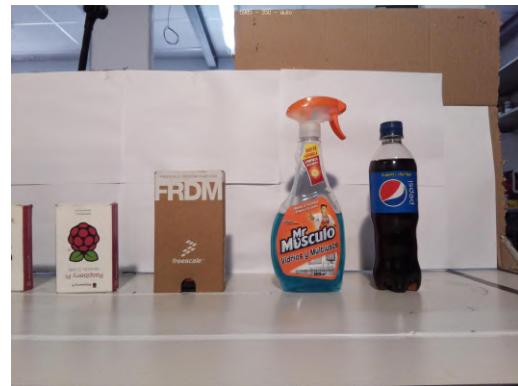


Figura 3.47: Imagen en movimiento 4.

Hecho esto, las capturas se unen logrando una imagen de mayor resolución, y donde los productos no se encuentran repetidos. Aplicando el algoritmo de *stitching* se obtiene la imagen mostrada en la *Figura 3.48*.



Figura 3.48: Imagen obtenida con el algoritmo de stitching.

#### Integración de cámara con el código principal

Obtenidas las fotos con una calidad que permita su futuro análisis, se procedió a integrar el *script* de captura de imágenes al código principal. Para esto se hace uso de *Threading* en *Python*, posibilitando asignar hilos de ejecución a distintas tareas con el objetivo de capturar fotos cada un determinado tiempo, sin que se interrumpa la ejecución de la rutina encargada del movimiento del robot.

Utilizando la librería *threading* [45] se define un *thread* para cada tarea particular que tiene que realizar el robot. Cabe destacar que no existe dependencia entre los resultados de distintas tareas, por lo que se deduce que no se producirán interbloqueos.

- Un hilo se encarga de realizar el seguimiento de la línea, leyendo valores de los sensores y modificando la energía suministrada a los motores para que éstos giren. A su vez, también se encarga de manejar los sensores de distancia en caso que sea necesario su uso.
- Un segundo hilo es el encargado de tomar fotos cada cierto tiempo y almacenarla en una carpeta en particular, para luego ser enviada a la computadora que realiza el procesamiento de las imágenes.

#### 3.4.4. Riesgos superados

En la *Tabla 3.17* se muestra el análisis de mitigación de riesgos al finalizar la presente iteración.

ID	Riesgo	Probabilidad	Impacto	Exposición Inicio Iteración	Exposición Final Iteración
RI-01	Sistema operativo incorrecto del robot	10 %	3	0.3	0.3
RI-02	<b>Incompatibilidad o avería de componentes</b>	50 % ↓	2	1.2	1
RI-03	Elección incorrecta de escenario de prueba	40 %	3	1.2	1.2
RI-04	Modificación de los requerimientos del proyecto	50 %	3	1.5	1.5
RI-05	<b>Dificultad en conseguir determinados componentes</b>	25 % ↓	1	0.3	0.25
RI-06	<b>Excesivo tiempo para cumplir los objetivos del proyecto</b>	45 % ↓	2	1.1	0.9
RI-07	Reducción de la fuerza de trabajo	10 %	4	0.4	0.4
RI-08	Pérdida de información relacionada al proyecto	5 %	5	0.25	0.25

Tabla 3.17: Riesgos mitigados en iteración 3.

#### 3.4.5. Resultados

En esta iteración se analizaron las alternativas existentes en el mercado sobre cámaras para *Raspberry Pi*. Luego de esto, se eligieron dos modelos en particular sobre los cuales se realizaron distintas pruebas para poder determinar cual era el más conveniente.

Después de elegir la cámara, se llevó a cabo un análisis para elegir el método de transferencia de las imágenes, determinando la necesidad de una compresión previa al envío de las mismas.

Además se llevó a cabo la implementación de un algoritmo que permite generar una representación de la góndola, a partir de un conjunto de imágenes con puntos en común.

Por otra parte, se implementó el uso de *threads* en el código principal, lo cual permitió integrar la función de obtener imágenes mientras el robot continua su trayectoria.

#### 3.4.6. Conclusiones

En esta iteración se mitigaron los siguientes riesgos:

- **RI-02:** Seleccionar la cámara oficial de *Raspberry*, permitió tener seguridad en la adaptación de la misma al proyecto. Se comprobó la compatibilidad de la cámara, tomando imágenes con distintas configuraciones, cumpliendo las expectativas.
- **RI-05:** El riesgo de no poder obtener componentes se vio reducido debido a la adquisición de la cámara elegida.
- **RI-06:** El riesgo de que se extiendan los tiempos establecidos para cumplir los objetivos del proyecto, se vio disminuido ya que la obtención de la imagen, y su posterior procesamiento, se logró en un tiempo reducido.

## 3.5. Iteración 4: Reconocimiento de productos

### 3.5.1. Introducción

En esta iteración se lleva a cabo la investigación y estudio de distintos trabajos relacionados al reconocimiento de objetos en góndolas, para su posterior implementación.

Luego de la implementación, se realiza el cálculo de métricas de rendimiento, con el fin de cuantificar el rendimiento del sistema desarrollado.

### 3.5.2. Requerimientos

En la presente iteración se atiende los siguientes requerimientos.

- **RF8:** A partir de una imagen, el sistema debe reconocer al menos dos clases y dos tipos de productos de cada clase.
- **RNF8:** Para el reconocimiento de objetos se deben utilizar como máximo 20 imágenes por producto.

### 3.5.3. Desarrollo

#### 3.5.3.1 Primera implementación de reconocimiento de objetos

En primer lugar, se llevó adelante la implementación de un detector de objetos a partir de una red neuronal convolucional basada en regiones (*R-CNN*). Se eligió *Faster-RCNN* ya que es un tipo de red altamente utilizada, con gran rendimiento y que presenta una ventaja en cuanto a velocidad y precisión sobre sus predecesoras.

Se comenzó a trabajar a partir de una implementación [46] realizada en *Python*, usando la API de detección de objetos en *TensorFlow*. Este tipo de redes requieren disponer de una gran cantidad de imágenes por objeto para su etapa de entrenamiento, como mínimo 300 por cada uno.

Se creó un *dataset* de entrenamiento para dos productos en particular (caja de *Raspberry Pi* y botella de gaseosa *Pepsi Zero*), utilizando alrededor de 800 imágenes por cada uno. Cada uno de los elementos visibles en las imágenes deben ser etiquetados, especificando a qué clase pertenece y la posición dentro de la imagen. El etiquetado de los productos se realizó con la herramienta *LabelImg* [47], como se muestra en la *Figura 3.49*.

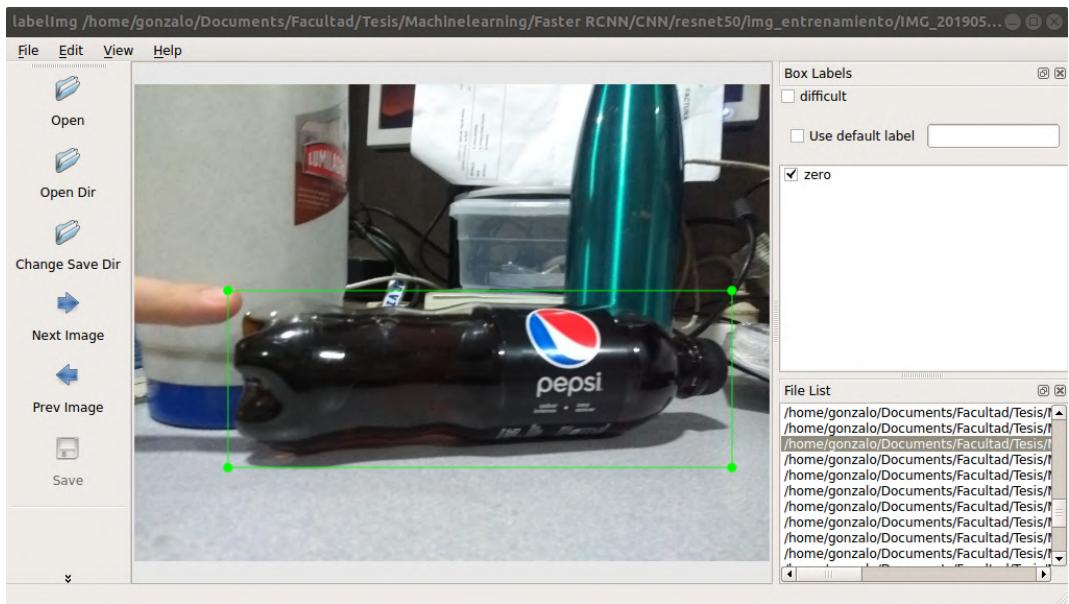


Figura 3.49: Etiquetado de imagen de entrenamiento con LabelImg

Luego de esto, se procedió al entrenamiento de la red a través de 80 epochs. En la *Figura 3.50, 3.51 y 3.52* se muestran algunas detecciones realizadas por la red ya entrenada.

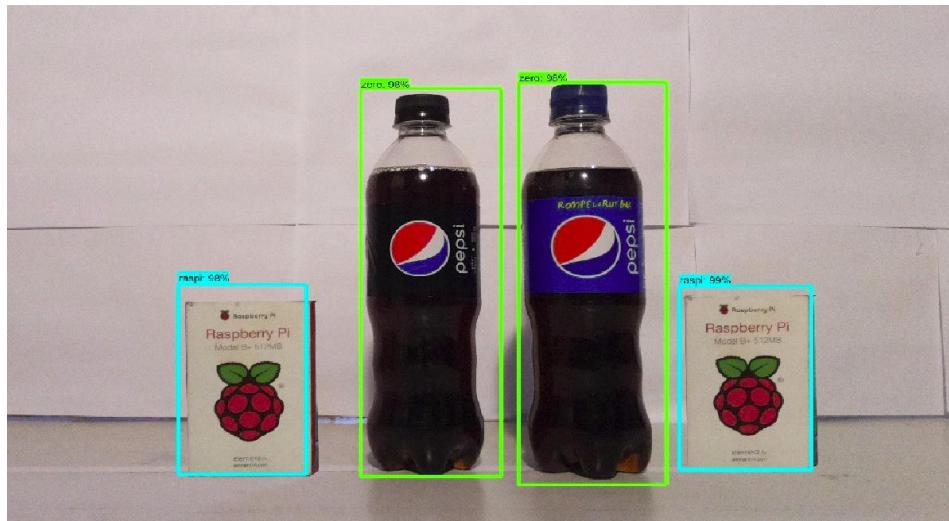


Figura 3.50: Ejemplo 1 de detección en Faster R-CNN

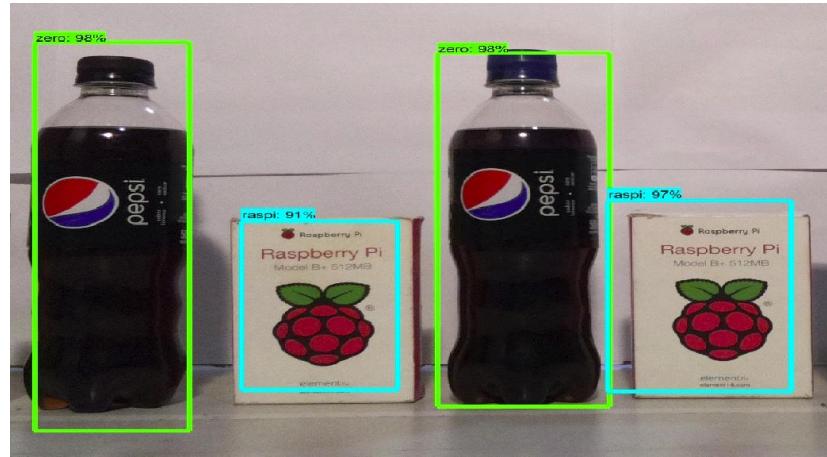


Figura 3.51: Ejemplo 2 de detección en Faster R-CNN

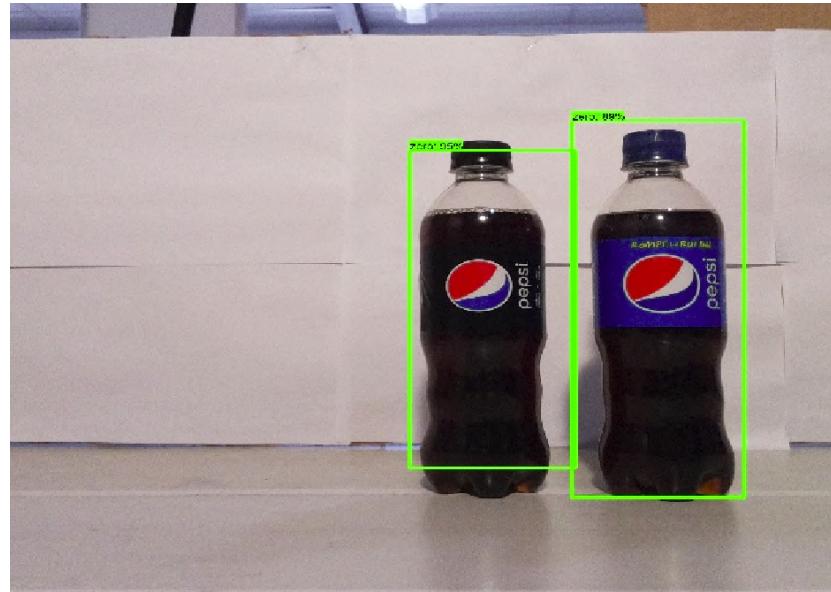


Figura 3.52: Ejemplo 3 de detección en Faster R-CNN

Se puede observar que el detector funciona correctamente. Además, se ve que la probabilidad de pertenecer a cada clase es muy alta (mayor a 90 % en todos los objetos).

A pesar de su fácil implementación y sus buenos resultados, la cantidad de imágenes necesarias es la principal desventaja de este método. Una tienda posee una gran variedad de productos que cambian continuamente sus etiquetas. Por cada uno de ellos se debe obtener y etiquetar más de 300 imágenes, sin tener en cuenta que cada vez que haya un cambio en su envoltorio el proceso debe volver a realizarse.

### 3.5.3.2 Análisis de trabajos relacionados a reconocimiento de productos en tiendas

Teniendo en cuenta lo planteado, se desea encontrar un método que posibilite reconocer productos sin la necesidad de una gran cantidad de imágenes de referencia por cada uno de ellos. Para esto se

analizan los trabajos mencionados en el *Capítulo 2.13*, donde se presentan algunos desafíos:

- La tarea exige el reconocimiento de un número extremadamente alto de artículos diferentes, del orden de varios miles para tiendas medianas y pequeñas, y muchas de ellas presentan una pequeña variabilidad inter e intraclase.
- Las bases de datos de productos disponibles generalmente incluyen solo una o unas pocas imágenes con calidad de estudio por producto, mientras que en el momento de la prueba el reconocimiento se realiza con imágenes que muestran una parte de un estante que contiene varios productos.
- Como los artículos a la venta en una tienda y su apariencia cambian con el tiempo, un sistema de reconocimiento práctico debería manejar productos/paquetes nuevos sin problemas.

Teniendo esto en cuenta, se obtuvo ventajas y desventajas de las implementaciones investigadas, para poder determinar cual es la más conveniente. Cabe destacar que la medida de *Precisión* hace referencia al total de detecciones correctas sobre el total de detecciones, *Exhaustividad* al numero de productos correctamente detectados sobre el numero de productos visibles en la imagen.

Alternativa	<i>Product Recognition in Store Shelves as a Sub-Graph Isomorphism Problem</i>
Ventaja	Permite obtener el posicionamiento exacto de los productos dentro de la estantería. Posee una etapa de verificación para corregir falsas detecciones.
Desventaja	Por cada estantería se debe generar un planograma de referencia, lo cual es tedioso ya que los supermercados cambian el orden de sus productos constantemente.
Dataset	<i>Grocery Products dataset</i> : 70 imágenes con alrededor de 12 productos diferentes por imagen.
Rendimiento	Precisión: 0.9047 - Exhaustividad: 0.9026

Tabla 3.18: Alternativa de identificador de objetos con isomorfismo de grafos.

Alternativa	<i>A deep learning pipeline for product recognition in store shelves</i>
Ventaja	Permite obtener tipo y cantidad de objetos en la estantería, a partir de pocas imágenes de referencia por producto.
Desventaja	Requiere de algoritmos muy complejos en la etapa de refinamiento, para el alcance de nuestro proyecto.
Dataset	1247 imágenes para entrenar <i>Detector</i> . Para entrenar el <i>Embedder</i> se usan 3288 imágenes (una por producto).
Rendimiento	Precisión: 0.91 - Exhaustividad: 0.82

Tabla 3.19: Alternativa de identificador de objetos con pipeline basado en Deep Learning.

Alternativa	<i>Retail Shelf Analytics Through Image Processing and Deep Learning</i>
Ventaja	Al ser una implementación del <i>pipeline</i> de A. Tonioni, posee todas sus ventajas. Además, no utiliza una etapa de refinamiento, lo cual hace más simple al sistema.
Desventaja	Al no realizar refinamiento y corrección de errores, los resultados finales obtenidos no son los mismos que en la alternativa anterior.
Dataset	10 diferentes instancias para cada superclase.
Rendimiento	Precisión: 0.73 - Exhaustividad: 0.65

Tabla 3.20: Alternativa de identificador de objetos con isomorfismo de grafos.

La primer alternativa (*Tabla 3.18*) se descarta debido a que los supermercados cambian continuamente el orden de sus productos en góndola, por lo que generar un planograma de referencia de cada una se vuelve un proceso repetitivo y laborioso.

La segunda opción (*Tabla 3.19*) posee la ventaja de reconocer productos a partir de pocas imágenes de referencia. Sin embargo, la etapa de refinamiento convierte a esta alternativa en compleja, por lo que su implementación demanda mayor tiempo que el planificado.

La tercera alternativa (*Tabla 3.20*) extiende de la anterior, realizando una gran simplificación en su implementación al no utilizar una etapa de refinamiento. Más allá de que los valores de Precisión y Exhaustividad son más bajos, la simpleza de su implementación convierte a esta en la mejor alternativa a ser utilizada en nuestro proyecto.

### 3.5.3.3 Implementación de la alternativa elegida

#### Elección de productos a identificar

Generalmente en una estantería de un supermercado los productos se agrupan por tipo y marca, como se muestra en la *Figura 3.53*.



Figura 3.53: Ejemplo de góndola de supermercado.

Al no existir una gran variabilidad de objetos en cada góndola se decidió trabajar con 6 productos diferentes, siendo esto una limitación que permite ganar tiempo sin dejar de representar la realidad. En este punto se debe destacar que uno de los grandes problemas que se presenta en el proceso de detección de objetos en góndolas, es la forma irregular que presentan determinados productos. Por este motivo se optó por elegir botellas y cajas, las cuales mantienen su forma en cualquier imagen tomada.

Los productos elegidos son los siguientes:

- Botella gaseosa *Pepsi* 500 ml.
- Botella gaseosa *Pepsi Zero* 500 ml.
- Caja de *Raspberry Pi*.
- Caja de *Freescale*.
- Caja de pegamento *UHU*.
- Botella *Mr. Musculo* limpia vidrios

## Detector de formas

Trabajar con una cantidad limitada de productos y que estos contengan formas similares, permite ahorrar tiempo en el proceso de entrenamiento del detector. El detector debe determinar las regiones donde existe un posible producto, y clasificarlo de acuerdo a su forma (*Figura 3.54*).

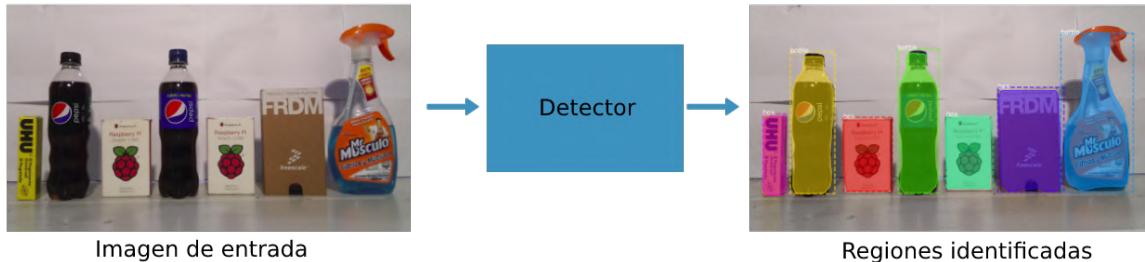


Figura 3.54: Funcionamiento esperado del detector.

Para llevar adelante la implementación del detector, se analizaron las siguientes alternativas para la segmentación de imágenes (*image segmentation*) (*Tabla 3.21*). La segmentación de imagen —más precisamente la segmentación de instancia— crea una máscara de píxel para cada objeto en la imagen, permitiendo obtener el objeto en particular y aislarlo de su entorno [48].

Algoritmo	Descripción	Ventaja	Limitaciones
Segmentación basada en regiones	Separa los objetos en diferentes regiones en función de algunos valores de umbral.	a. Cálculos simples b. Velocidad de operación rápida c. Cuando el objeto y el fondo tienen un alto contraste, este método funciona muy bien	Cuando no hay una diferencia significativa en escala de grises o una superposición de los valores de píxeles en escala de grises, se hace muy difícil obtener segmentos precisos.
Segmentación de detección de bordes	Utiliza características locales discontinuas de una imagen para detectar bordes y, por lo tanto, definir un límite del objeto.	Es bueno para las imágenes que tienen un mejor contraste entre los objetos.	No es adecuado cuando hay demasiados bordes en la imagen y si hay menos contraste entre los objetos.
Segmentación basada en agrupaciones ( <i>Clustering</i> )	Divide los píxeles de la imagen en grupos homogéneos.	Funciona muy bien en pequeños conjuntos de datos y genera excelentes grupos.	El tiempo de cálculo es demasiado grande y costoso.
Mask R-CNN	Da tres salidas para cada objeto en la imagen: su clase, coordenadas del cuadro delimitador y máscara de objeto	a. Enfoque simple, flexible y general b. Es la alternativa más usada actualmente.	Alto tiempo de entrenamiento

Tabla 3.21: Comparación de distintos métodos de segmentación de imagen.

Se eligió el método de *Mask R-CNN* debido a que posee la capacidad de calcular con mayor precisión la superficie expuesta para los diversos productos en las imágenes y además es la alternativa

que mejor se adapta a la problemática planteada. Esta ventaja esta dada por las características del algoritmo para hacer predicciones en píxeles para cada objeto identificado en las imágenes. El alto tiempo de entrenamiento es una desventaja de este método, pero en la implementación a realizar la red será entrenada en escasas ocasiones.

Luego de la elección se llevó a cabo la implementación basándose en un trabajo previo [49] bajo *Python*, *Keras* [50], y *TensorFlow* [51]. Además se utilizó una red preentrenada sobre el *dataset* de *COCO* [52], el cual posee 80 clases diferentes. Al estar la red entrenada para identificar un amplio conjunto de clases —como personas, autos, alimentos, entre otros— permite realizar un proceso de aprendizaje por transferencia (*transfer learning*). Esto posibilita agregar nuevas clases sin la necesidad de utilizar un gran conjunto de datos, ya que la red contiene muchas imágenes (~120K) y los pesos entrenados ya han aprendido muchas de las características comunes en las imágenes naturales.

Una de las clases de *COCO* permite reconocer botellas de una manera muy precisa (*Figura 3.55*). Se observa en la imagen que todas las botellas (incluso envases similares como sprays) son reconocidos bajo la clase *Bottle*, con un gran porcentaje de acierto. Sin embargo se observa que no se detectan las cajas visibles en la imagen, por lo que se debe realizar el entrenamiento de la red para que incorpore la detección de esta forma de productos.



Figura 3.55: Detección de botellas a través de una red preentrenada con COCO Dataset.

Se procedió a entrenar la red con imágenes de distintas cajas, creando la clase *Box*. Para esto, primero se realizó un etiquetado de 200 imágenes, con el programa *VIA Tool* [53], como se muestra en la *Figura 3.56*. Cabe destacar que este procedimiento será necesario para cada forma nueva que se desee incorporar.

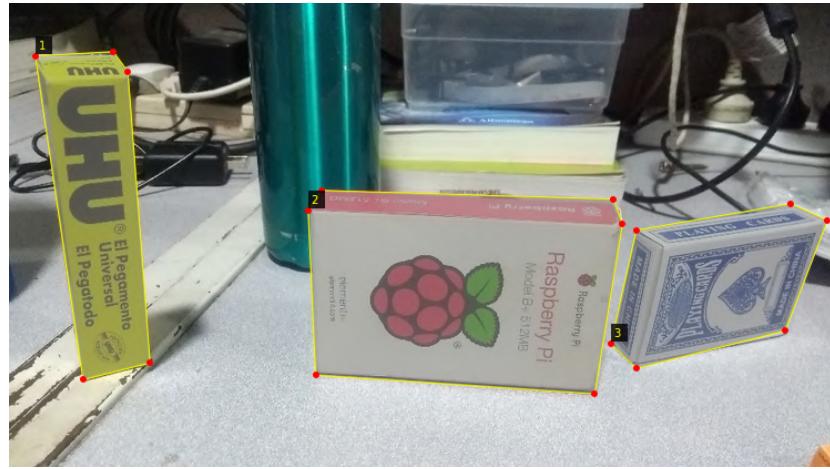


Figura 3.56: Etiquetado de objetos con VIA tool.

Dentro de la arquitectura de la red utilizada, existe un conjunto de parámetros que permiten optimizar el entrenamiento de la misma y otros asociados a la etapa de detección. En la Tabla 3.22 se listan algunos de ellos junto a una breve descripción y al valor utilizado en la implementación elegida.

Cada vez que se agrega una nueva clase se debe modificar el parámetro *NUM CLASSES*, el cual contendrá el numero de clases a identificar más la clase *background*. Por otro lado se debe modificar *RPN ANCHOR SCALES*, el cual hace referencia a los tamaños que tomará la ventana deslizante de la red RPN. Cuando la nueva forma a identificar surge de objetos muy pequeños, se deben utilizar valores como (8, 16, 32, 64, 128). En este caso, como las cajas siempre contendrán un tamaño relativamente grande (con respecto al tamaño de la imagen) se elige (32, 64, 128, 256, 512). Un parámetro asociado a esto es *RPN ANCHOR RATIOS*, que determinará la relación de aspecto de la ventana deslizante. Si se desea entrenar la red para detectar elementos verticales y delgados (como un poste de luz) se debe utilizar configuraciones como (0.1,0.2,0.5). Otro parámetro que se puede editar es el tamaño de imagen de entrada a la red, el cual está determinado por *IMAGE MIN DIM* y *IMAGE MAX DIM*. Si se establece un tamaño de imagen pequeño se obtendrá un entrenamiento más rápido, siendo conveniente en casos donde las imágenes de entrada sean pequeñas. Reducir el tamaño cuando se cuenta con imágenes muy grandes puede generar perdida de información. Para la implementación se utilizó un tamaño de 1024x1024 ya que se busca tener una mayor precisión, a pesar de un mayor tiempo de entrenamiento.

Parámetro	Descripción	Valor utilizado
IMAGES PER GPU	Número de imágenes para entrenar en cada GPU. Se debe usar el número más alto que la GPU pueda manejar para obtener el mejor rendimiento.	1
VALIDATION STEPS	Número de pasos de validación para ejecutar al final de cada época de entrenamiento. Un número mayor mejora la precisión de las estadísticas de validación, pero ralentiza el entrenamiento.	50
BACKBONE	<i>Backbone network architecture:</i> Los valores admitidos son: <i>resnet50</i> , <i>resnet101</i> .	resnet101
NUM CLASSES	Número de clases de clasificación (incluido background).	1 + 2
RPN ANCHOR SCALES	Tamaño del lado de la ventana en píxeles.	32, 64, 128, 256, 512
USE MINI MASK	Si está habilitado, cambia el tamaño de las máscaras de instancia a un tamaño más pequeño para reducir la carga de memoria. Recomendado cuando se usan imágenes de alta resolución.	True
MINI MASK SHAPE	(alto, ancho) de la mini máscara.	(56, 56)
IMAGE RESIZE MODE	Generalmente se utiliza el modo "square". En este, se cambia el tamaño y rellena con ceros para obtener una imagen cuadrada de tamaño [max_dim, max_dim]. Los otros modos disponibles son "none", "pad64", "pad64" y "crop".	"square"
IMAGE CHANNEL COUNT	Número de canales de color por imagen. RGB = 3, escala de grises = 1, RGB-D = 4.	3
DETECTION MAX INSTANCES	Número máximo de detecciones finales.	100
DETECTION MIN CONFIDENCE	Valor mínimo de probabilidad para aceptar una instancia detectada. Se omiten los ROI por debajo de este umbral	0.75
DETECTION NMS THRESHOLD	Umbral de Non-maximum suppression	0.3
LEARNING RATE	El papel <i>Mask R-CNN</i> usa un valor de 0.02, pero este en <i>TensorFlow</i> causa problemas en el entrenamiento y vuelve inestable la red, especialmente cuando se utiliza un <i>Batch Size</i> pequeño	0.001

Tabla 3.22: Párametros de Mask RCNN.

El entrenamiento fue realizado en un *hardware* (Tabla 3.23) perteneciente al Centro de Computación de Alto Desempeño (CCAD) de la UNC, acelerando en gran medida el proceso. Se entrenó la red durante 60 épocas, en un tiempo aproximado de 6 hs.

CPU	Intel i5-8400
Memoria RAM	8GB
GPU	Nvidia Tesla K40 12GB

Tabla 3.23: Recursos de hardware utilizados.

Luego del entrenamiento con un *dataset* que incluyen cajas, se realizó una detección y se comprobó

que las cajas comienzan a ser detectadas (*Figura 3.57*). Si bien algunos objetos que antes detectaba como una botella (como los spray) ahora son detectados como una caja, no se generan inconvenientes ya que en este punto la idea es identificar objetos, no realizar su categorización.

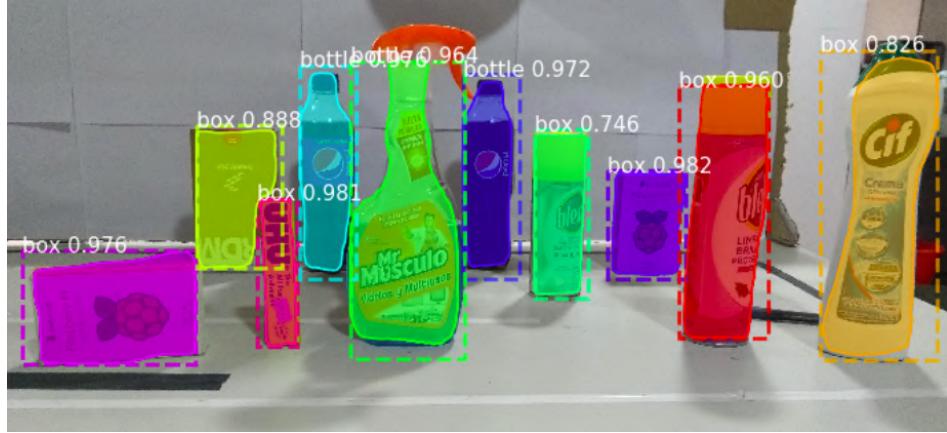


Figura 3.57: Detección de productos con Mask R-CNN luego de 30 épocas.

Luego de realizar la detección, se debe obtener las regiones generadas. Para esto, se generó un *script* en *Python* que obtiene la imagen junto a las regiones, pero colocándole un fondo negro en cada una de ellas (*Figura 3.58*).



Figura 3.58: Imagen obtenida con script de generación de cuadros.

De esta imagen se procede a realizar un recorte de cada cuadro generado. Así, se obtienen tantas imágenes, como elementos detectados (*Figura 3.59*).



Figura 3.59: Imágenes obtenidas a partir de las regiones detectadas.

### Generación de imágenes de referencia

Para proceder en la creación del *pipeline*, se debe generar las imágenes de referencia que representen a cada producto a identificar, las cuales se utilizarán para ser comparadas con las regiones obtenidas en el detector.

Se debe tener en cuenta que estas imágenes no pueden tener una calidad totalmente diferente a las que ingresan al detector, ya que se necesita que exista la mayor similitud posible, para que el clasificador funcione correctamente.

Por cada producto se obtienen aproximadamente 8 imágenes de referencia, eliminando el fondo de la misma como se ve a continuación (*Figura 3.60*).



Figura 3.60: Obtención de imagen de referencia.

### Obtención de features de imagen

Para poder comparar dos imágenes, se necesita caracterizar las mismas de forma matemática. Esto se logra a través de las *features*, las cuales son patrones simples, a partir de los cuales se puede describir lo que se ve en la imagen. El papel principal de las *features* es transformar la información visual en el espacio vectorial. Esto posibilita realizar operaciones matemáticas con estas, por ejemplo, encontrar vectores similares.

Continuando con la arquitectura elegida, el extractor de *features* se implementa a través de una red VGG16 preentrenada con *Imagenet* [54]. Esta red tiene una capa de entrada que toma una imagen de 224x224x3, y una capa de salida que es una predicción *softmax* de 1000 clases. Este proceso se realiza también con un modelo *ResNet 50*, obteniendo resultados similares a VGG16, por lo cual se optó por utilizar este último.

Como no se busca realizar una predicción, se obtiene la salida de la última capa de *max pooling*, la cual contendrá las *features* de la imagen ingresada, con un tamaño de 7x7x512.



Figura 3.61: Extracción de features con VGG16.

Por lo tanto, a la salida de este proceso se obtiene una matriz de 7x7x512 (*Figura 3.61*), que representará las características de cada imagen. Para facilitar el trabajo matemático posterior, esta matriz es convertida en un vector con 25088 elementos.

### Comparador de features

Obtenidas las *features* de las imágenes de referencia, junto con las *features* de las regiones candidatas a contener un producto, se procede a realizar la comparación de las mismas.

Como las *features* serán vectores de 25088 elementos, la comparación se puede realizar a través de la similitud de coseno. Matemáticamente, esta hace referencia a la similitud existente entre dos vectores, en un espacio que posee un producto interior con el que se evalúa el valor del coseno del ángulo comprendido entre ellos. La fórmula que se utiliza es la siguiente:

$$\cos(\Theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}} \quad (8)$$

A medida que los vectores sean más cercanos, el ángulo tenderá a ser 0 y la similitud se maximizará. Por eso, cuando se comparan dos *features* de imágenes del mismo producto, se obtiene un porcentaje mayor, que al comparar imágenes entre distintos productos.

El comparador a implementar tendrá dos entradas:

1. **Detecciones:** La *feature* del recorte de la región que posee un producto candidato.
2. **Imágenes de referencia:** El conjunto de *features* de las imágenes de referencia.

Para cada *feature* de una región se calcula la similitud de coseno con cada uno de las *features* de referencia, como se muestra en la *Figura 3.62*.

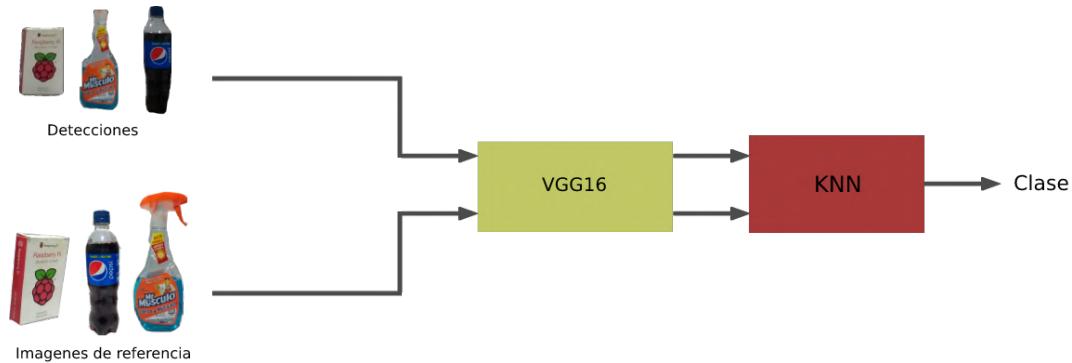


Figura 3.62: Extracción y posterior comparación de features.

Un detalle a tener en cuenta, es que las imágenes de referencia están clasificadas en dos subclases (*Box* o *Bottle*). De esta manera, la comparación se llevará a cabo a partir de la forma reconocida en la *Mask R-CNN*, la cual determinará la clase del producto.

Una vez comparada la *feature* de una región con todas las *features* de referencia, se genera un ranking descendente, que contiene la clase a la cual pertenece la *feature* de referencia, y el valor de similitud (1-distancia calculada).

1. (2)-(0.7842)
2. (2)-(0.6541)
3. (3)-(0.5722)
4. (2)-(0.5160)
5. (1)-(0.3032)
6. (5)-(0.1909)

Luego, se seleccionan los 3 elementos posicionados en la cima del *ranking*, y se evalúa la ocurrencia de la clase. Se puede dar el caso que una clase se repita 2/3 de las veces, y será determinada como la clase del producto. En el caso que las tres primeras posiciones se encuentren 3 clases distintas, se seleccionará la que tenga mayor valor de similitud (posición mas alta en el *ranking*).

El proceso anteriormente nombrado se denomina *K-NN* (*K-Nearest Neighbor*), el cual es un método que posibilita clasificar objetos con un aprendizaje "vago", ya que su entrenamiento solo requiere de ejemplos cercanos en el espacio de los elementos. El valor K determinará con cuantos vecinos será comparado el elemento del cual se quiere obtener la clase. En nuestra implementación se trabajó con un valor de K=3.

Es necesario destacar que el porcentaje de similitud que se obtiene dependerá de que tan similar sea la imagen de referencia con la región detectada. Por ese mismo motivo, las imágenes de referencia deben ser tomadas en condiciones de luz similares, y con una cámara similar a la que luego obtendrá fotos de las góndolas.

#### 3.5.3.4 Mejoras realizadas a la alternativa implementada

Con la alternativa implementada, los valores de métrica obtenidos fueron muy altos, pero se detectaron algunos problemas que ocurrían frecuentemente, los cuales se decidió buscar alguna solución a estos. A continuación se detallan los errores detectados, y la solución implementada.

##### Confusión entre botella de *Pepsi Común* con *Pepsi Zero*

La confusión de una botella de *Pepsi Común* con *Pepsi Zero* fue el error más frecuente. Este error era comprensible, ya que ambos tienen la misma forma, y mismo color (en gran parte de su superficie); el único cambio notable, es la diferencia de color de su etiqueta y la de su tapa.

Para solucionar esto, y evitar cualquiera falsa detección de la misma naturaleza, se planteo una solución basada en la composición de colores *RGB* de los objetos detectados.

Lo que se realizó, fue separar las imágenes de referencia teniendo en cuenta el color *RGB* predominante en cada una. Es decir que la base de datos de referencia quedó dividida en seis grupos: una primera división, en cajas y botellas; y una segunda, en tres subgrupos de acuerdo al color rojo, verde o azul que predomina.

De esta manera, a la hora de realizar las detecciones, se obtiene el color *RGB* predominante de cada uno de los elementos obtenidos de la red MASK R-CNNN. Teniendo en cuenta el tipo de objeto (caja o botella) y del color *RGB*, se realiza la comparación de la *feature* obtenida, con el grupo de referencia correspondiente.

Para la obtención del color *RGB* dominante, se utilizó la librería de *Python Scikit-Image*. [55]

### Detección de cajas unidas

En la *Figura 3.63* se puede ver que la red toma las dos cajas apiladas como si fuera una sola. Este problema se detectó en varias imágenes que presentaban cajas muy próximas entre si. Para este tipo de error no se planteó una solución, pero si la detección de ellos. Para ello se tuvo en cuenta la relación de aspecto de las imágenes de los productos (proporción entre el ancho y la altura).



Figura 3.63: Error en la detección de cajas contiguas.

Lo que se realiza, es la comparación de la relación de aspecto ( $RA$ ) de la detección, con la  $RA$  de la imagen de referencia con mayor similitud (teniendo en cuenta los resultados obtenidos con el algoritmo  $K-NN$ ). En los casos que la diferencia entre los valores de  $RA$  sean mayores a  $0,3$ , se considera que puede haber un error, por lo cual dicha detección debe ser revisada; caso contrario, la detección es correcta.

Cuando se ejecuta el programa, el usuario obtiene una advertencia que le informa que pueden existir más productos en la detección realizada (*Figura 3.64*).

```
Para imagen box-score:0.9992218-20190919T111019-0.612.png:  
0: ['2', 0.7329831, 0.622]  
1: ['2', 0.43493822, 1.695]  
2: ['0', 0.41487247, 0.626]
```

Con k=3 tenemos que el recorte es: **FreeScale**

```
Para imagen box-score:0.9992912-20190919T111019-0.932.png:  
0: ['2', 0.5493651, 1.695]  
1: ['2', 0.42905015, 0.622]  
2: ['2', 0.40724084, 0.622]
```

Con k=3 tenemos que el recorte es: **FreeScale**

**Atención:** Es posible que existan más productos en la detección

La imagen ingresada contiene:  
**FreeScale**  
**FreeScale**

Figura 3.64: Advertencia generada ante una posible detección errónea.

Si bien con esto no se obtienen mejoras en las métricas, pero si brinda información útil al usuario. Un trabajo a futuro, sería solucionar estos errores.

### 3.5.3.5 Diagrama de sistema completo

Luego de incorporar los métodos anteriormente nombrados para mejorar la detección de los productos, el diagrama resultante es el que se muestra en la *Figura 3.65*.

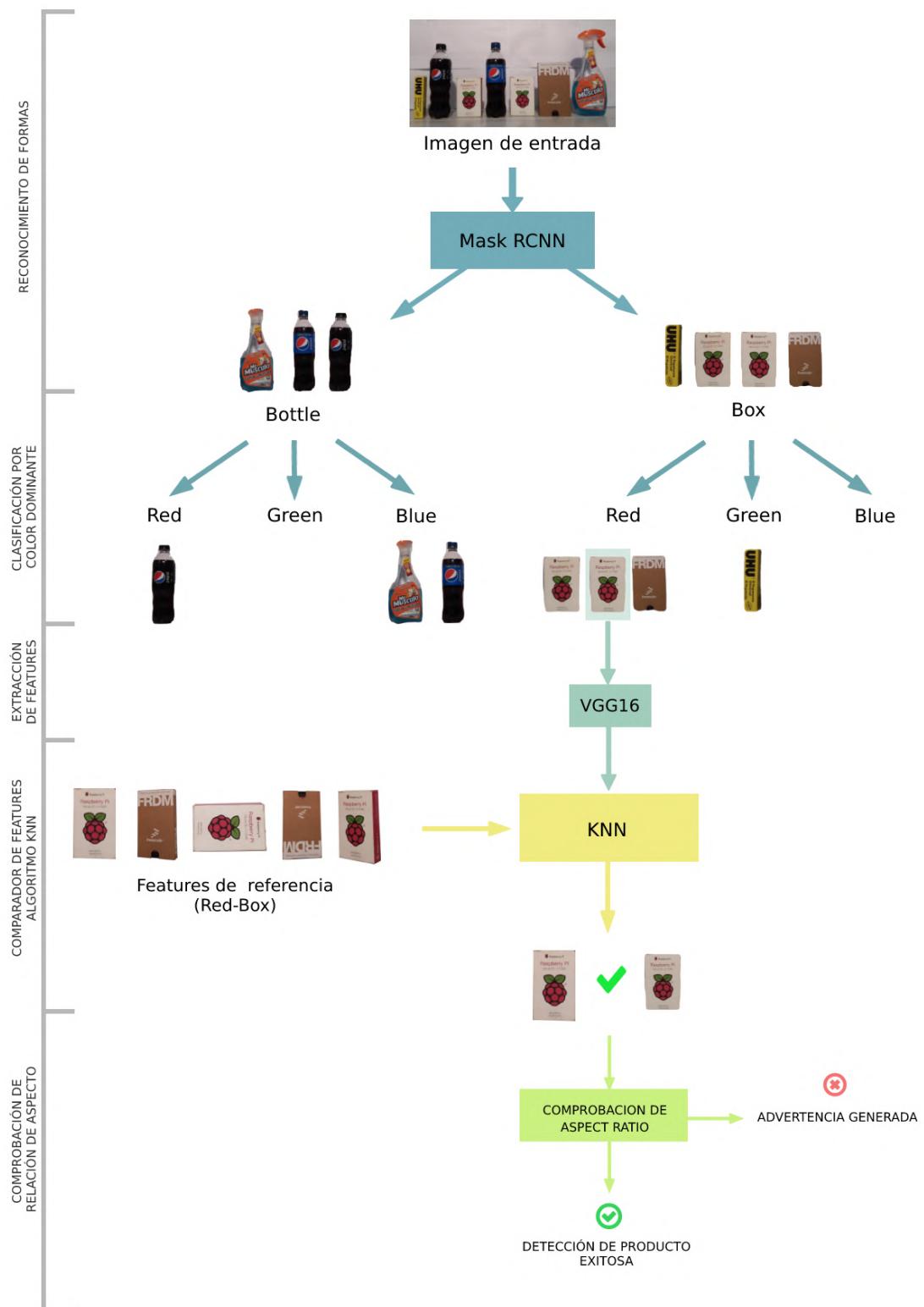


Figura 3.65: Diagrama completo de la implementación realizada.

### 3.5.3.6 Rendimiento obtenido

Para poder analizar el rendimiento del sistema implementado se utilizan dos métricas: *precisión* y *exhaustividad*.

La precisión hace referencia a que porcentaje de predicciones son correctas sobre las detecciones realizadas. En base a esto, se sabe que tan bueno es el sistema para determinar el producto, luego de que su forma ha sido determinada.

Por otra parte, la exhaustividad es una medida que permite determinar las predicciones realizadas correctamente sobre elementos visibles en las imágenes. Esta métrica se ve afectada cuando el detector de formas no detecta un producto (por cercanía con uno similar, por estar tapado, etc). Se puede determinar con la misma que tan bueno es el sistema para clasificar los elementos visibles.

En el siguiente gráfico se observa como van mejorando las métricas a medida que se entrena el detector de formas. Para estas pruebas se utilizaron 20 imágenes de referencia.

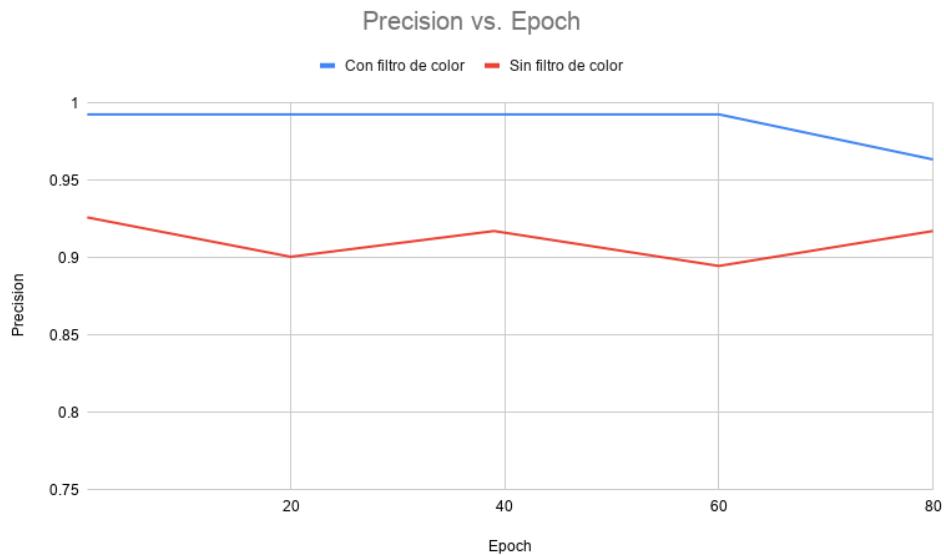


Figura 3.66: Variación de la precisión con el entrenamiento del detector.

Tanto en la *Figura 3.69*, como en la *Figura 3.67*, se ve que el filtro de color aumenta las métricas en gran medida. Además, se ve en la *Figura 3.69*, que luego de la *epoch* 60 comienza a producirse *overfitting*, es decir, que la red se está entrenando de una manera que solo permite obtener buenos resultados con el conjunto de entrenamiento, pero no con futuras muestras, por lo que los pesos de la red en la *epoch* 60 serán los elegidos para nuestro sistema.

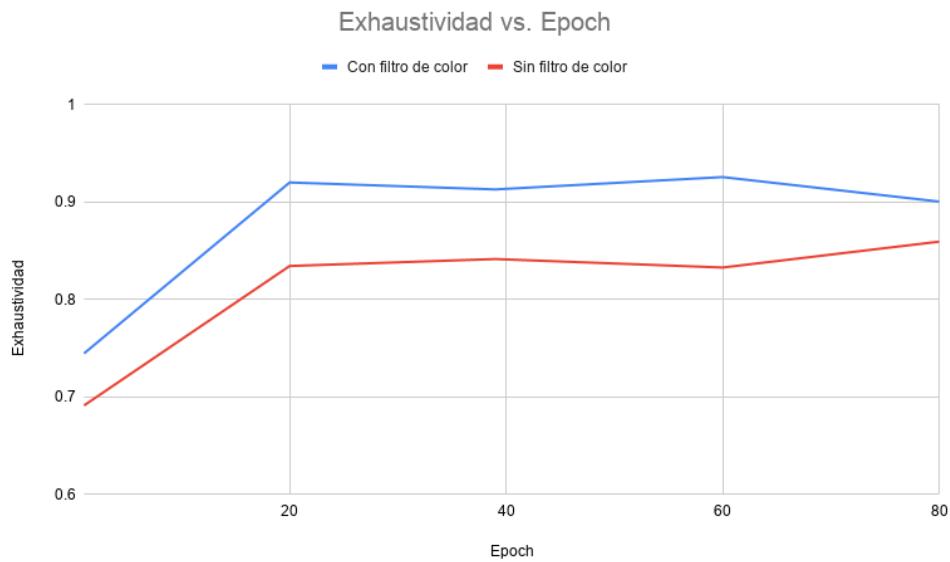


Figura 3.67: Variación de la exhaustividad con el entrenamiento del detector.

Otro parámetro que tiene un efecto sobre nuestro sistema, es la cantidad de imágenes de referencia que utiliza el comparador de la tercera etapa. Cuando se tiene una sola imagen de referencia por producto, se debe utilizar  $K=1$ , es decir, el algoritmo  $K$ -NN solo tendrá en cuenta el vecino más cercano al producto al cual se le quiere determinar su clase. Para mayor cantidad de imágenes de referencia, se elige un  $K=3$ .

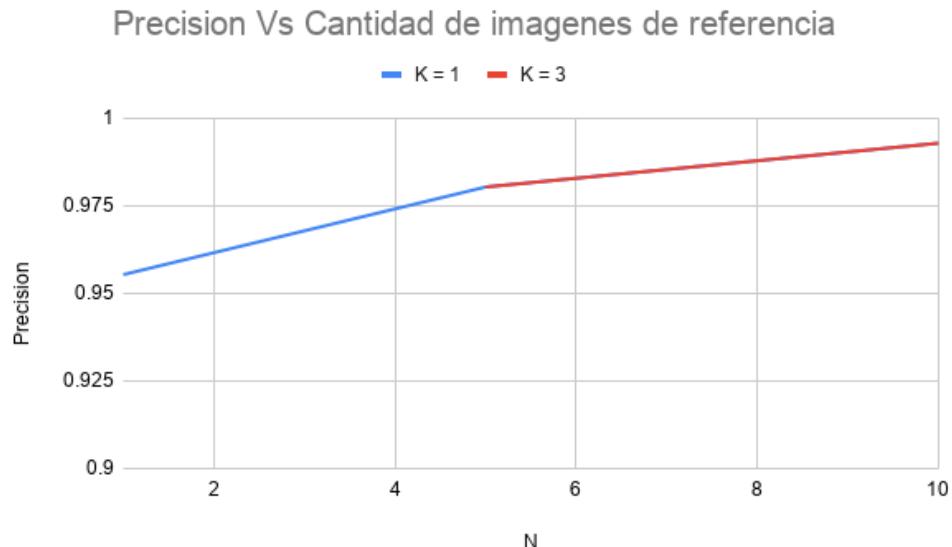


Figura 3.68: Cambios en la precisión a partir de la variación de imágenes de referencia.

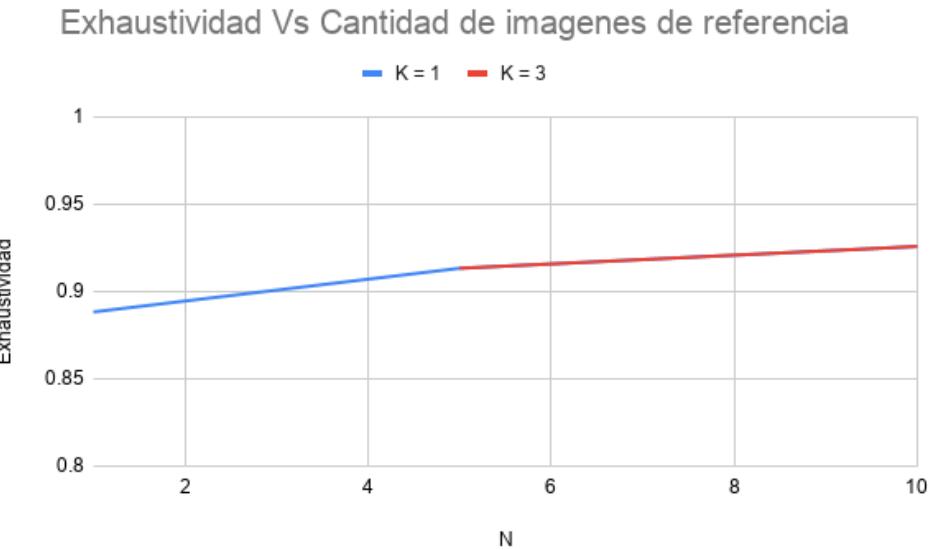


Figura 3.69: Cambios en la exhaustividad a partir de la variación de imágenes de referencia.

Ambos gráficos se obtiene con el filtro de color ya implementado en el sistema. Se ve que no existe gran diferencia trabajando con 5 o 10 imágenes de referencia, sin embargo, se intenta tener una cantidad que pueda representar al objeto en todos sus ángulos de visión posible (aproximadamente 8 a 10 imágenes por producto).

Por otro lado, siempre se intenta trabajar con  $K$  impar, y en la implementación se elige  $K=3$ , ya que si existe un empate entre los dos primeros vecinos, el tercero determinará la clase del producto.

### 3.5.4. Riesgos superados

En la *Tabla 3.24* se muestra el análisis de mitigación de riesgos al finalizar la cuarta iteración.

ID	Riesgo	Probabilidad	Impacto	Exposición Inicio Iteración	Exposición Final Iteración
RI-01	Sistema operativo incorrecto del robot	10 %	3	0.3	0.3
RI-02	Incompatibilidad o avería de componentes	50 %	2	1	1
RI-03	Elección incorrecta de escenario de prueba	40 %	3	1.2	1.2
RI-04	<b>Modificación de los requerimientos del proyecto</b>	30 % ↓	3	1.5	0.9
RI-05	Dificultad en conseguir determinados componentes	25 %	1	0.25	0.25
RI-06	<b>Excesivo tiempo para cumplir los objetivos del proyecto</b>	20 % ↓	2	0.9	1
RI-07	Reducción de la fuerza de trabajo	10 %	4	0.4	0.4
RI-08	Pérdida de información relacionada al proyecto	5 %	5	0.25	0.25

Tabla 3.24: Riesgos mitigados en iteración 4.

### 3.5.5. Resultados

En esta iteración se comenzó con la investigación y estudio de distintos trabajos relacionados al reconocimiento de objetos en góndolas utilizando redes neuronales. Realizado esto, se optó por uno para implementarlo.

Luego de la implementación, se prosiguió al cálculo de las métricas de rendimiento (precisión y exhaustividad), con el fin de cuantificar el rendimiento del sistema desarrollado

### 3.5.6. Conclusiones

En esta iteración se mitigaron los siguientes riesgos:

- **RI-04:** Haber cumplido con éxito los requerimientos RF8 y RNF8 permite reducir la probabilidad de modificación en los requerimientos del proyecto.
- **RI-06:** Luego de esta iteración están cubiertos la mayoría de los requerimientos del proyecto, lo que permite reducir la probabilidad de que se excedan los tiempos establecidos.

## 4. Resultados

En este capítulo se muestran la integración final del sistema completo y se hace la síntesis de los resultados obtenidos en la implementación realizada.

En primer lugar, para lograr que el robot realice el recorrido alrededor de la estantería, se implementó un mecanismo de desplazamiento utilizando los sensores seguidores de línea que posee el robot. Se logró adaptar el algoritmo proporcionado por el fabricante a las necesidades del proyecto. Luego de esto, se llevó a cabo un mecanismo de reorientación, el cual le permite al robot retomar la línea en caso de que por algún motivo pierda la misma. Para esto, se colocaron dos sensores de ultrasonido que permiten calcular la distancia hacia la góndola. En algunas ocasiones, se obtuvieron mediciones incorrectas debido al ángulo generado entre la superficie a calcular la distancia y el sensor, el cual era mayor a la mitad del de medición efectiva (30). Para solucionarlo se agregaron dos sensores más con un ángulo de 30° con respecto a los anteriores.

Con respecto a la cámara, se analizaron distintas alternativas compatibles con la placa utilizada. En primer instancia se seleccionaron las cámaras *Raspberry Pi Camera B*, de la empresa fabricante del robot, y la *Raspberry Pi Camera Module v2*, la cual es la oficial de *Raspberry*. Sobre éstas se realizaron diversas pruebas modificando distintos parámetros como el tiempo de exposición y la sensibilidad ISO. Se buscó una configuración que permita obtener imágenes con el robot en movimiento y en condiciones lumínicas determinadas. Se concluyó que la mejor opción es la cámara *Raspberry Pi Module v2*, que permite obtener imágenes aceptables en un ambiente con 800 lumens. Para la misma se configuró la sensibilidad ISO en 400 y el tiempo de exposición en 1/50 segundos.

En la *Figura 4.1* se puede ver el robot luego de realizar las modificaciones para su funcionamiento.

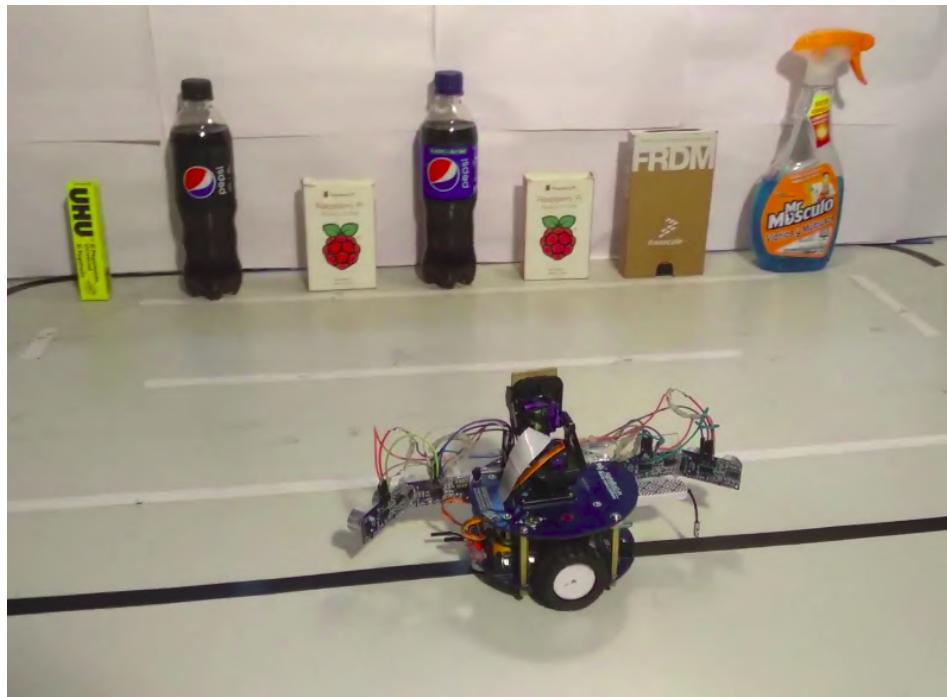


Figura 4.1: Robot realizando recorrido alrededor de la estantería.

Luego de esto se desarrolló la rutina de obtención de imágenes, la cual realiza capturas cada un determinado tiempo. Se muestra un posible conjunto de imágenes de una góndola en la *Figura 4.2*, *4.3*, *4.4* y *4.5*.



Figura 4.2: Imagen de góndola 1.



Figura 4.3: Imagen de góndola 2.



Figura 4.4: Imagen de góndola 3.



Figura 4.5: Imagen de góndola 4.

Se envian las imágenes capturadas a la computadora de análisis, donde se lleva a cabo el proceso de *image stitching* para poder generar una nueva imagen a partir de las recibidas (*Figura 4.6*).

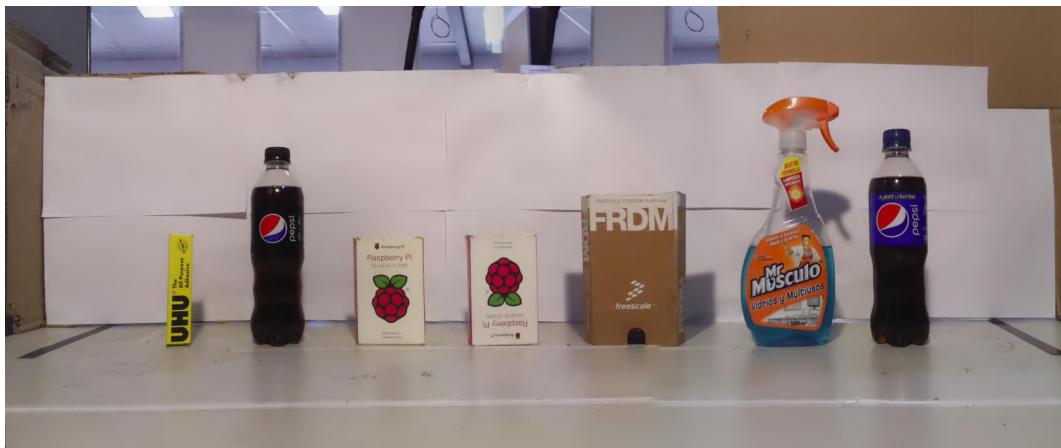


Figura 4.6: Imagen generada a partir de las imágenes recibidas de la góndola.

Se continuó con la etapa de reconocimiento de las formas presentes en la imagen. Inicialmente se utilizó una red *Faster-RCNN*, la cual realiza la detección en una sola etapa a partir de un conjunto de

imágenes de entrenamiento. Si bien los resultados obtenidos fueron aceptables, el principal inconveniente de esta solución es que requería disponer de una gran cantidad de imágenes por objeto.

Luego de realizar una investigación en trabajos relacionados, se concluyó que la solución óptima consiste en una implementación segmentada, separando el proceso de detección en distintas etapas. En la primer etapa se implementó un detector de formas utilizando una red *Mask-RCNN*. Esta red fue entrenada para realizar detecciones de los posibles productos visibles en la imagen, clasificando según la forma correspondiente (*Figura 4.7*).



Figura 4.7: Imagen generada a partir de las imágenes recibidas de la góndola.

Una vez que las formas de los posibles productos fueron detectadas, se llevaron adelante modificaciones en la implementación para realizar recortes automáticos de las detecciones. El recorte permite aislar a cada una de ellas del fondo de la imagen (*Figura 4.8*).



Figura 4.8: Recortes generados a partir de detecciones.

Los recortes de los posibles productos que estaban divididos en dos grupos (cajas y botellas), se subdividieron según su color predominante (Rojo, Verde y Azul) como se muestra en la *Figura 4.9*.

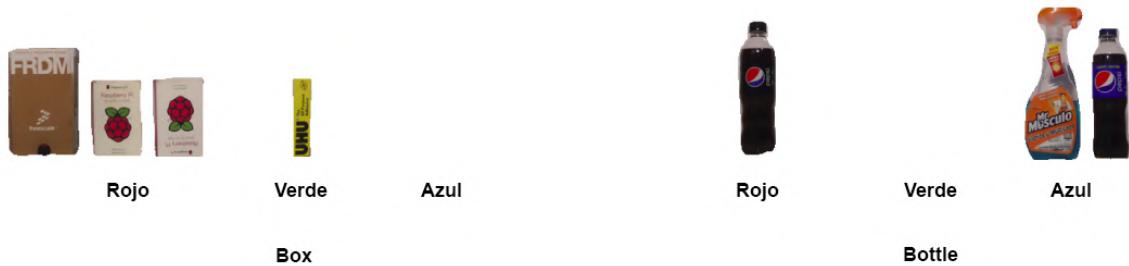


Figura 4.9: Recortes generados a partir de detecciones.

Luego de esto se llevó adelante la implementación de una red *VGG16* que posee la capacidad de expresar de forma matemática las características de la imagen. Esta red extrae las *features* de cada recorte, convirtiéndolas en matrices, posibilitando realizar operaciones matemáticas sobre ellos.

A continuación, se procedió a realizar la comparación entre cada uno de los recortes y las imágenes de referencia asociadas a cada una de las subcategorías (*clase-color*). Este proceso se implementó a través del algoritmo *K-NN*, como se muestra en la *Figura 4.10*.

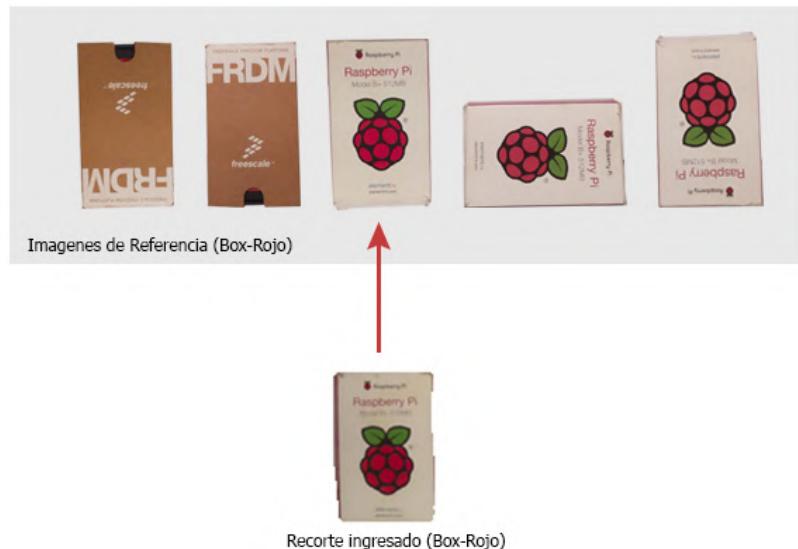


Figura 4.10: Comparación y elección de imagen de referencia de mayor similitud.

El algoritmo desarrollado realiza un lista decreciente de acuerdo a la similitud entre el recorte y las imágenes de referencia asociadas a la clase-color del mismo. Esta similitud es calculada a partir de la distancia cosenoidal de las *features* de entrada. Luego, selecciona la imagen de referencia de mayor similitud con el recorte, y se extrae de ella la clase a la que pertenece (*Raspberry Pi* en este caso). Este proceso se realiza para todos los recortes realizados de la imagen.

A partir de esto, se lograron clasificar todos los productos visibles dentro de la imagen, como se ve en la *Figura 4.11*.

```
La imagen ingresada contiene:  
UHU  
Mr Musculo  
FreeScale  
Pepsi Zero  
Raspberry  
Pepsi Comun  
Raspberry
```

Figura 4.11: Salida de consola de imagen ingresada.

Se presentaron ocasiones donde las detecciones contenían algunos errores. Cuando dos cajas se encontraban muy próximas entre sí, la red detectora de formas determinaba que existía una sola. Tal como se mostró en el *Capítulo 3.5.3.4*, se identificó este tipo de errores generando una advertencia al usuario de que la detección era anómala.

En el *Anexo A* se muestran pruebas realizadas sobre el sistema final, comprobando el correcto funcionamiento para distintas situaciones.



## 5. Conclusiones

Durante el desarrollo de este trabajo se cubrieron las principales áreas de la carrera de Ingeniería en Computación, y se incorporaron conocimientos sobre desarrollo de sistemas embebidos y de software, redes neuronales y sistemas de control.

A través de un proceso de diseño de ingeniería se logró implementar un robot autónomo a partir del robot comercial *AlphaBot 2 Pi*. El mismo es capaz de desplazarse alrededor de una góndola, utilizando un sistema de control de trayectoria y reorientación, mientras captura imágenes, posibilitando realizar el reconocimiento de los productos visibles. Éstas capacidades sientan las bases para el desarrollo a futuro de un robot que pueda navegar por una tienda, sin la necesidad de tener una referencia como una línea, y que pueda reconocer no solo productos, si no también precios de los mismos.

Para la implementación del sistema se ha aplicado un método de desarrollo iterativo que ha permitido avanzar en un problema de esta complejidad sin mayores contratiempos, satisfaciendo los principales requerimientos propuestos al principio del proyecto. Aún así, no significa que las decisiones que se tomaron en determinadas ocasiones hayan sido las mejores. La oportunidad de perfeccionamiento de las soluciones aplicadas existe, y plantea un futuro prometedor para aquellas personas que deseen continuar con el proyecto.

Del análisis de trabajos relacionados, se determinó utilizar una combinación de tecnologías actuales y distintos métodos heurísticos para reconocimiento de productos. Esto permitió llegar a una solución simple y escalable, que posibilita la detección de nuevas formas a partir de 200 imágenes por cada una, y el reconocimiento de cada producto utilizando aproximadamente ocho imágenes. La solución implementada posibilitó la correcta detección de dos formas de productos diferentes (cajas y botellas), y tres tipos de productos por cada una, lo que permite el cumplimiento de los objetivos planteados al principio del proyecto. Esto abre camino hacia una futura implementación donde, basándose en el prototipo realizado, pueda reconocerse un conjunto variado de formas y productos.

El prototipo realizado es compatible con los estándares propuestos en la creciente *industria 4.0*, siendo capaz de recolectar datos a través de sensores y procesarlos utilizando métodos de aprendizaje automático. A partir de los resultados obtenidos se vislumbra un gran potencial en la solución desarrollada. En un futuro es posible incorporar un nuevo grado de automatización al proceso, donde se generen pedidos teniendo en cuenta el faltante de *stock* detectado por el sistema.

A lo largo de este trabajo se descubrieron nuevos desafíos a resolver en un futuro. La forma irregular de determinados artículos dificulta su correcta identificación. Además, las superficies reflectantes (como los envases metálicos) impiden que las capturas se realicen correctamente debido al reflejo generado por la luz. Por último, aún se presentan dificultades al determinar la cantidad de productos cuando están posicionados uno detrás de otro.

En resumen, este trabajo propone un sistema novedoso y simple basándose en la autonomía de un robot y en la capacidad de reconocer patrones en imágenes, abriendo camino a la resolución de problemáticas en distintos ámbitos.

### 5.1. Trabajos futuros

En esta sección se detallan las mejoras o extensiones posibles al proyecto que se han visualizado a lo largo de su desarrollo.

- Duplicar la cantidad de sensores de distancia utilizados, formando un hexágono. De esta manera el robot tendrá conocimiento en 360° de la distancia de objetos a su alrededor. Otra alternativa a analizar es utilizar un servo que permita girar el sensor, haciendo un barrido de los objetos alrededor del robot.
- Reemplazar al seguidor de línea por un sistema de mapeo como el utilizado en el *Hermes III* [42], que permita desprenderse de la necesidad de tener que modificar el entorno.

- Llevar a cabo un sistema de localización en el robot que le permita saber en qué lugar se encuentra en un momento determinado, a partir de referencias en el espacio de trabajo como *Tags RFID* bajo el suelo.
- Utilizar un hardware con una capacidad de procesamiento que permita realizar las rutinas de desplazamiento, captura, transferencia y compresión de imágenes en un sistema operativo de tiempo real, aportando determinismo al sistema.
- Incorporar flash automático a la cámara de al menos 1200 lúmenes.

## Referencias

- [1] "Tally - Symbe Robotics." <https://roboticsandautomationnews.com/2018/12/11/simbe-robotics-installs-inventory-robot-tally-in-decathlon-store/20243/>. Accedido: 2019-05-15.
- [2] "Robot TagSurveyor." <https://fetchrobotics.com/products-technology/datasurvey/tagsurveyor/>. Accedido: 2019-04-26.
- [3] "Rotbot AdvanRobot." [https://wwwresearchgatenet/publication/317751756\\_Development\\_of\\_an\\_RFID\\_inventory\\_robot\\_AdvanRobot](https://wwwresearchgatenet/publication/317751756_Development_of_an_RFID_inventory_robot_AdvanRobot). Accedido: 2019-04-26.
- [4] D.-W. H and B. T, "Simultaneous localization and mapping: part I," *IEEE Robotics Automation Magazine*, vol. 13, p. 99–110, Jun 2006.
- [5] "ROS- Robot Operating System." <http://wikiros.org/navigation>. Accedido: 2019-04-29.
- [6] M. Ferreiro, *Armado y puesta en funcionamiento de robot AlphaBot 2 Pi y RB4*. 2018.
- [7] G. Ortmann, Nestor Trejo, "Selección de herramientas de Machine Learning aplicado a problemas de ingeniería," 2019.
- [8] I. Sommerville, *Ingeniería de software*. Pearson Educación, 2011.
- [9] "Metodología Desarrollo." <http://imaudgedu/~sellares/EINF-ES2/Present1011/MetodoPesadesDocumentaciopdf>. Accedido: 2019-04-29.
- [10] "Metodo de la Cascada." <https://esslidesharenet/cristhianandresaguilar/medolos-tradicionales-de-desarrollo-de-software-cascada-espiral>. Accedido: 2019-04-29.
- [11] "Wiki Espiral." [https://es.wikipedia.org/wiki/Desarrollo\\_en\\_espiral](https://es.wikipedia.org/wiki/Desarrollo_en_espiral). Accedido: 2019-04-29.
- [12] "Metodo Iterativo." <https://proyectosagilesorg/desarrollo-iterativo-incremental/>. Accedido: 2019-04-29.
- [13] "Alphabot 2 Pi - wiki." <https://wwwwavesharecom/wiki/AlphaBot2-Pi>. Accedido: 2019-04-29.
- [14] "Wikipedia - Raspberry Pi." [https://es.wikipedia.org/wiki/Raspberry\\_Pi](https://es.wikipedia.org/wiki/Raspberry_Pi). Accedido: 2019-05-02.
- [15] "Raspbian." <https://wwwraspbianorg/>. Accedido: 2019-04-30.
- [16] "10 SO para Raspberry Pi." <https://wwwionoses/digitalguide/servidores/know-how-sistemas-operativos-para-raspberry-pi/>. Accedido: 2019-05-03.
- [17] "Protocolo ssh." <https://webmitedu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-sshhtml>. Accedido: 2019-05-02.
- [18] "Wikipedia - Python." <https://es.wikipedia.org/wiki/Python>. Accedido: 2019-05-03.
- [19] "Modulo de carga TP4056."
- [20] "Fundamentos del Sensor Ultrasonico." <https://wwwkeyencecommx/ss/products/sensor/sensorbasics/ultrasonic/info/>. Accedido: 2019-04-30.
- [21] "HC-SR04: rangos de medicion." <https://koalabtech/aprende/componentes/sensor-ultrasonico-hc-sr04/>. Accedido: 2019-09-22.

- [22] "Sensores Fotoelectricos - Autonics." <http://dominioncommx/descargas/sensores-fotoelectricospdf>. Accedido: 2019-04-30.
- [23] "Clasificacion Sensores Fotoelectricos." <https://wwwkeyencecommx/ss/products/sensor/sensorbasics/photoelectric/info/>. Accedido: 2019-04-30.
- [24] "Factor de reflexión según color de superficie." <https://wwwcontavales/que-tipos-de-sensores-fotoelectricos-existen/>. Accedido: 2019-04-30.
- [25] "Datasheet itr20001-t." <http://wwweverlightcom/file/ProductFile/ITR20001-Tpdf>. Accedido: 2019-04-30.
- [26] "Raspberry Camera v2." <https://wwwraspberrypiorg/products/camera-module-v2/>. Accedido: 2019-05-14.
- [27] "Especificaciones Camera Raspi v2." <https://wwwraspberrypiorg/documentation/hardware/camera/>. Accedido: 2019-05-14.
- [28] "Pi NoIR Camera V2." <https://wwwraspberrypiorg/products/pi-noir-camera-v2/>. Accedido: 2019-05-14.
- [29] "Camera Kuman." [http://wwwkumantechcom/kuman-5mp-1080p-hd-camera-module-for-raspberry-pi-for-raspberry-pi-3-model-b-b-a-rpi-2-1-sc15\\_p0063html](http://wwwkumantechcom/kuman-5mp-1080p-hd-camera-module-for-raspberry-pi-for-raspberry-pi-3-model-b-b-a-rpi-2-1-sc15_p0063html). Accedido: 2019-05-14.
- [30] "Camera Fisheye Lens." <https://wwwwavesharecom/product/mini-pc/raspberry-pi/cameras/rpi-camera-mhtm>. Accedido: 2019-05-15.
- [31] "ISO de camara." <https://wwwdzoomorges/sabes-para-que-sirve-la-sensibilidad-iso-de-tu-camara/>. Accedido: 2019-05-15.
- [32] "Wikipedia - Velocidad de obturación." [https://es.wikipedia.org/wiki/Velocidad\\_de\\_obturaci%C3%B3n](https://es.wikipedia.org/wiki/Velocidad_de_obturaci%C3%B3n). Accedido: 2019-05-15.
- [33] "Velocidad de obturación." <https://wwwdzoomorges/para-que-sirve-la-velocidad-de-obturacion/>. Accedido: 2019-05-15.
- [34] M. Brown and D. G. Lowe, "Automatic panoramic image stitching using invariant features," 2007.
- [35] D. J. Matich, "Redes neuronales: Conceptos básicos y aplicaciones," 2001.
- [36] "Segmentación de imágenes - Wikipedia." [https://es.wikipedia.org/wiki/Segmentaci%C3%B3n\\_\(procesamiento\\_de\\_im%C3%A1genes\)](https://es.wikipedia.org/wiki/Segmentaci%C3%B3n_(procesamiento_de_im%C3%A1genes)). Accedido: 2019-10-21.
- [37] A. Tonioni and L. Di Stefano, "Product recognition in store shelves as a sub-graph isomorphism problem," 2017.
- [38] A. Tonioni and L. Di Stefano, "A deep learning pipeline for product recognition on store shelves," 2018.
- [39] A. D. Biasio, "Retail shelf analytics through image processing and deep learning," 2019.
- [40] "Elección de SO Raspi." <https://wwwatareaoes/tutorial/raspberry-pi-primeros-pasos/sistema-operativo-raspbian/>. Accedido: 2019-05-03.
- [41] "Repositorio asociado al proyecto "Sistema inteligente de relevamiento de stock"." <https://githubcom/alechagonzalo/mlpipeline/>. Accedido: 2019-07-08.

- [42] H. M. Juan Agustín Coutinho, *Hermes III, Robot Omnidireccional con Capacidad de SLAM*. 2018.
- [43] “Controlador PID.” [https://es.wikipedia.org/wiki/Controlador\\_PID](https://es.wikipedia.org/wiki/Controlador_PID). Accedido: 2019-04-15.
- [44] “Picamera Python Library.” <https://picamerareadthedocsio>. Accedido: 2019-04-09.
- [45] “Threading Python Library.” <https://docspythonorg/2/library/threadinghtml>. Accedido: 2019-04-09.
- [46] “Repositorio detección de objetos con tensorflow.” [https://githubcom/puigalex/deteccion\\_objetos](https://githubcom/puigalex/deteccion_objetos). Accedido: 2019-07-15.
- [47] “Repositorio LabelImg.” <https://githubcom/tzutalin/labelImgs>. Accedido: 2019-07-15.
- [48] “Introduction to Image Segmentation Techniques.” [https://wwwanalyticsvidhyacom/blog/2019/04/introduction-image-segmentation-techniques-python/?utm\\_source=blog&utm\\_medium=computer-vision-implementing-mask-r-cnn-image-segmentation](https://wwwanalyticsvidhyacom/blog/2019/04/introduction-image-segmentation-techniques-python/?utm_source=blog&utm_medium=computer-vision-implementing-mask-r-cnn-image-segmentation). Accedido: 2019-10-21.
- [49] W. Abdulla, “Mask r-cnn for object detection and instance segmentation on keras and tensorflow.” [https://githubcom/matterport/Mask\\_RCNN](https://githubcom/matterport/Mask_RCNN), 2017.
- [50] “Keras: The Python Deep Learning library.” <https://kerasio/>. Accedido: 2019-08-05.
- [51] “TensorFlow.” <https://wwwtensorfloworg/>. Accedido: 2019-07-15.
- [52] “Keras: COCO Dataset.” <http://cocodatasetorg/>. Accedido: 2019-08-05.
- [53] A. Dutta, A. Gupta, and A. Zissermann, “VGG image annotator (VIA).” <http://www.robots.ox.ac.uk/vgg/software/via/>, 2016. Accedido: 2019-08-08.
- [54] “Repositorio de modelos entrenados con ImageNet.” <https://githubcom/tensorflow/models/tree/mas-ter/research/slim>. Accedido: 2019-08-10.
- [55] “Scikit Learn.” <https://scikit-learnorg/stable/>. Accedido: 2019-09-06.



## A. Anexos

### A.1. Detecciones obtenidas en distintas imágenes

A continuación se muestran pruebas realizadas con distintas imágenes, y los resultados obtenidos de la detección. Se presenta la imagen de entrada capturada por el robot, las detecciones realizadas por el detector de formas, y por ultimo, la salida en consola con las detecciones de los productos.

#### A.1.1. Prueba 1

En la *Figura A.2* se observa la salida de la red *Mask R-CNN*, tomando como entrada la *Figura A.1*. El resultado final de la detección se observa en la *Figura A.3*.



Figura A.1: Imagen de entrada de prueba 1



Figura A.2: Formas detectadas en prueba 1

```

Clasificando productos
Producto 7/7 |██████████| 100.0% Completo

Para imagen box-score:0.99879056-20190919T142752-0.625.png:
0: ['2', 0.7851265, 0.622]
1: ['2', 0.39980254, 0.622]
2: ['0', 0.37828946, 0.634]

Con k=3 tenemos que el recorte es: FreeScale

Para imagen bottle-score:0.9940221-20190919T142752-0.425.png:
0: ['4', 0.7045164, 0.395]
1: ['4', 0.69287044, 0.372]
2: ['4', 0.6007017, 0.507]

Con k=3 tenemos que el recorte es: Mr Musculo

Para imagen bottle-score:0.99645364-20190919T142752-0.317.png:
0: ['1', 0.69506115, 0.303]
1: ['1', 0.5443145, 0.289]
2: ['1', 0.5311303, 0.281]

Con k=3 tenemos que el recorte es: Pepsi Zero

Para imagen box-score:0.99973327-20190919T142752-0.685.png:
0: ['0', 0.74679583, 0.626]
1: ['0', 0.5408603, 1.515]
2: ['0', 0.52743495, 0.626]

Con k=3 tenemos que el recorte es: Raspberry

Para imagen box-score:0.99968946-20190919T142752-0.641.png:
0: ['0', 0.7399057, 0.626]
1: ['0', 0.5056075, 1.515]
2: ['0', 0.47343487, 0.626]

Con k=3 tenemos que el recorte es: Raspberry

Para imagen bottle-score:0.9977927-20190919T142752-0.284.png:
0: ['1', 0.6542413, 0.303]
1: ['6', 0.59672976, 0.395]
2: ['6', 0.5499325, 0.356]

Con k=3 tenemos que el recorte es: Pepsi Comun

Para imagen box-score:0.99796665-20190919T142752-0.318.png:
0: ['3', 0.48276794, 0.286]
1: ['3', 0.29674724, 0.228]
2: ['0', 0.29487255, 0.634]

Con k=3 tenemos que el recorte es: UHU

```

Figura A.3: Salida de consola en prueba 1

### A.1.2. Prueba 2

En la *Figura A.5* se observa la salida de la red Mask R-CNN, tomando como entrada la *Figura A.4*. El resultado final de la detección se observa en la *Figura A.6*.



Figura A.4: Imagen de entrada de prueba 2



Figura A.5: Formas detectadas en prueba 2

```

Clasificando productos
Producto 6/6 | ██████████ | 100.0% Completo

Para imagen box-score:0.9996486-20190919T150248-1.694.png:
0: ['0', 0.74061763, 1.515]
1: ['0', 0.56697655, 0.626]
2: ['0', 0.46026966, 0.626]

Con k=3 tenemos que el recorte es: Raspberry

Para imagen box-score:0.99906474-20190919T150249-0.229.png:
0: ['3', 0.47103336, 0.286]
1: ['3', 0.33160722, 0.228]
2: ['0', 0.29255414, 0.634]

Con k=3 tenemos que el recorte es: UHU

Para imagen box-score:0.99986315-20190919T150248-0.705.png:
0: ['0', 0.6652544, 0.626]
1: ['0', 0.494748, 1.515]
2: ['0', 0.46067265, 0.626]

Con k=3 tenemos que el recorte es: Raspberry

Para imagen box-score:0.99988294-20190919T150248-0.624.png:
0: ['2', 0.7681284, 0.622]
1: ['2', 0.40201232, 0.622]
2: ['0', 0.37997174, 0.626]

Con k=3 tenemos que el recorte es: FreeScale

Para imagen bottle-score:0.9986123-20190919T150249-0.417.png:
0: ['4', 0.5866334, 0.395]
1: ['4', 0.54964894, 0.372]
2: ['4', 0.48783556, 0.507]

Con k=3 tenemos que el recorte es: Mr Musculo

Para imagen bottle-score:0.99598664-20190919T150249-0.33.png:
0: ['1', 0.6178201, 0.303]
1: ['1', 0.54281944, 0.289]
2: ['1', 0.52302295, 0.281]

Con k=3 tenemos que el recorte es: Pepsi Zero

```

Figura A.6: Salida de consola en prueba 2

### A.1.3. Prueba 3

En la *Figura A.8* se observa la salida de la red Mask R-CNN, tomando como entrada la *Figura A.7*. El resultado final de la detección se observa en la *Figura A.9*.



Figura A.7: Imagen de entrada de prueba 3

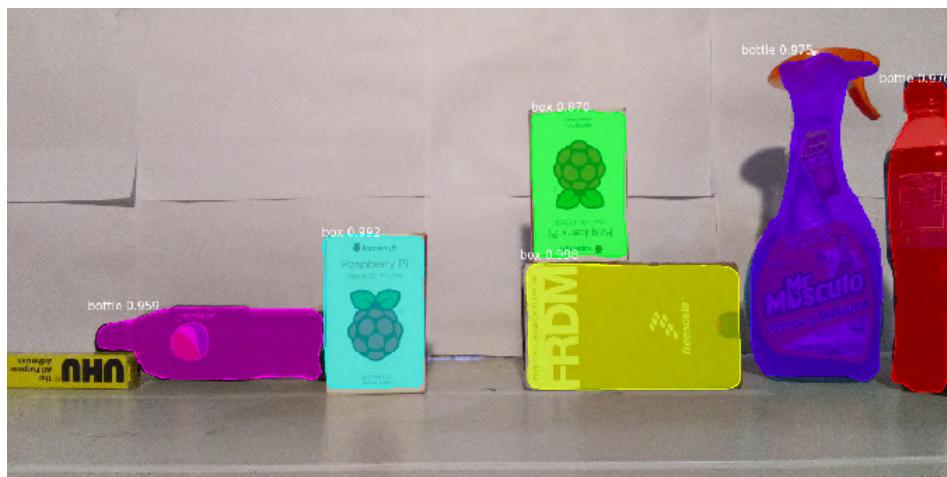


Figura A.8: Formas detectadas en prueba 3

```

Clasificando productos
Producto 6/6 | ██████████ | 100.0% Completo

Para imagen bottle-score:0.98574346-20190919T150919-0.389.png:
0: ['4', 0.67463756, 0.395]
1: ['4', 0.5797271, 0.507]
2: ['4', 0.57553905, 0.372]

Con k=3 tenemos que el recorte es: Mr Musculo

Para imagen box-score:0.99889684-20190919T150919-1.738.png:
0: ['2', 0.743216, 1.695]
1: ['2', 0.6076638, 1.519]
2: ['2', 0.42829734, 5.348]

Con k=3 tenemos que el recorte es: FreeScale

Para imagen box-score:0.9981468-20190919T150919-0.704.png:
0: ['0', 0.7812223, 0.626]
1: ['0', 0.5578233, 1.396]
2: ['0', 0.55494684, 1.515]

Con k=3 tenemos que el recorte es: Raspberry

Para imagen bottle-score:0.99604887-20190919T150919-0.207.png:
0: ['6', 0.32013562, 0.382]
1: ['6', 0.30670017, 0.356]
2: ['6', 0.30035335, 0.404]

Con k=3 tenemos que el recorte es: Pepsi Comun

Para imagen box-score:0.9996779-20190919T150919-0.644.png:
0: ['0', 0.7210068, 0.626]
1: ['0', 0.53175014, 1.501]
2: ['0', 0.521787, 1.515]

Con k=3 tenemos que el recorte es: Raspberry

Para imagen bottle-score:0.986121-20190919T150919-2.919.png:
0: ['1', 0.50030977, 3.46]
1: ['1', 0.46260482, 3.448]
2: ['1', 0.31741905, 3.279]

Con k=3 tenemos que el recorte es: Pepsi Zero

```

Figura A.9: Salida de consola en prueba 3

#### A.1.4. Prueba 4

En la *Figura A.11* se observa la salida de la red *Mask R-CNN*, tomando como entrada la *Figura A.10*. El resultado final de la detección se observa en la *Figura A.12*.



Figura A.10: Imagen de entrada de prueba 4



Figura A.11: Formas detectadas en prueba 4

Clasificando productos  
Producto 4/4 | ██████████ | 100.0% Completo

Para imagen bottle-score:0.99953306-20190919T151450-0.285.png:  
0: ['1', 0.65598184, 0.289]  
1: ['1', 0.5747236, 0.291]  
2: ['1', 0.5546568, 0.282]

Con k=3 tenemos que el recorte es: **Pepsi Zero**

Para imagen bottle-score:0.980959-20190919T151450-0.147.png:  
0: ['1', 0.28974825, 0.282]  
1: ['1', 0.28123072, 0.289]  
2: ['1', 0.27578935, 0.303]

Con k=3 tenemos que el recorte es: **Pepsi Zero**

Para imagen box-score:0.9862331-20190919T151450-2.019.png:  
0: ['2', 0.28763202, 5.348]  
1: ['3', 0.2788678, 4.951]  
2: ['2', 0.26418138, 1.31]

Con k=3 tenemos que el recorte es: **FreeScale**

**Atención:** Es posible que existan más productos en la detección

Para imagen bottle-score:0.9862233-20190919T151450-0.29.png:  
0: ['4', 0.41548085, 0.404]  
1: ['4', 0.39592212, 0.352]  
2: ['4', 0.32470533, 0.363]

Con k=3 tenemos que el recorte es: **Mr Musculo**

Figura A.12: Salida de consola en prueba 4

### A.1.5. Prueba 5

En la *Figura A.14* se observa la salida de la red *Mask R-CNN*, tomando como entrada la *Figura A.13*. El resultado final de la detección se observa en la *Figura A.15*.



Figura A.13: Imagen de entrada de prueba 5



Figura A.14: Formas detectadas en prueba 5

```

Clasificando productos
Producto 7/7 |██████████| 100.0% Completo

Para imagen bottle-score:0.99830073-20190919T152610-0.315.png:
0: ['6', 0.52645546, 0.356]
1: ['6', 0.52559954, 0.395]
2: ['6', 0.47418213, 0.382]

Con k=3 tenemos que el recorte es: Pepsi Comun

Para imagen box-score:0.99947137-20190919T152609-0.416.png:
0: ['2', 0.528466, 0.536]
1: ['2', 0.4527523, 0.622]
2: ['2', 0.4019635, 0.579]

Con k=3 tenemos que el recorte es: FreeScale

Para imagen box-score:0.9992368-20190919T152609-0.315.png:
0: ['3', 0.5358727, 0.296]
1: ['3', 0.45710734, 0.279]
2: ['0', 0.4231122, 0.635]

Con k=3 tenemos que el recorte es: UHU

Para imagen box-score:0.9998605-20190919T152609-0.629.png:
0: ['0', 0.7188585, 0.626]
1: ['0', 0.5882435, 1.515]
2: ['0', 0.5599494, 1.501]

Con k=3 tenemos que el recorte es: Raspberry

Para imagen bottle-score:0.9953328-20190919T152610-0.312.png:
0: ['1', 0.59755427, 0.303]
1: ['1', 0.59441113, 0.289]
2: ['1', 0.5819031, 0.291]

Con k=3 tenemos que el recorte es: Pepsi Zero

Para imagen box-score:0.9982566-20190919T152610-0.704.png:
0: ['0', 0.6440805, 0.626]
1: ['0', 0.51481766, 1.515]
2: ['0', 0.49566785, 1.396]

Con k=3 tenemos que el recorte es: Raspberry

Para imagen bottle-score:0.8402789-20190919T152610-0.304.png:
0: ['4', 0.47124985, 0.395]
1: ['4', 0.4556065, 0.301]
2: ['4', 0.42159656, 0.507]

Con k=3 tenemos que el recorte es: Mr Musculo

```

Figura A.15: Salida de consola en prueba 5