DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN ELÉCTRONICA Y TELECOMUNICACIONES

TRABAJO DE TITULACIÓN, PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

TEMA: DESARROLLO DE UN ALGORITMO DE VISIÓN ARTIFICIAL MEDIANTE EL PROCESAMIENTO DIGITAL DE IMÁGENES CON EL SENSOR KINECT ORIENTADO A LINUX

AUTOR: PÉREZ ÁLVAREZ, PATRICIO JAVIER

DIRECTOR: ING. TORRES TELLO, JULIO WLADIMIR MSc.

SANGOLQUÍ

2017

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNCACIONES

CERTIFICACIÓN

Certifico que el trabajo de titulación, "DESARROLLO DE UN ALGORITMO DE VISIÓN ARTIFICIAL MEDIANTE EL PROCESAMIENTO DIGITAL DE IMÁGENES CON EL SENSOR KINECT ORIENTADO A LINUX" realizado por el señor PATRICIO JAVIER PÉREZ ÁLVAREZ, ha sido revisado en su totalidad y analizado por el software anti-plagio, el mismo cumple con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de Fuerzas Armadas ESPE, por lo tanto me permito acreditarlo y autorizar al señor PATRICIO JAVIER PÉREZ ÁLVAREZ para que lo sustente públicamente.

Sangolquí, 05 de septiembre del 2017

ING. JULIO TORRES TELLO

DIRECTOR

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA
CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNCACIONES

AUTORÍA DE RESPONSABILIDAD

Yo, PATRICIO JAVIER PÉREZ ÁLVAREZ, con cédula de identidad N° 1721411385, declaro que este trabajo de titulación "DESARROLLO DE UN ALGORITMO DE VISIÓN ARTIFICIAL MEDIANTE EL PROCESAMIENTO DIGITAL DE IMÁGENES CON EL SENSOR KINECT ORIENTADO A LINUX" ha sido desarrollado considerando los métodos de investigación existentes, así como también se ha respetado los derechos intelectuales de terceros considerándose en las citas bibliográficas.

Consecuentemente declaro que este trabajo es de mi autoría, en virtud de ello me declaro responsable del contenido, veracidad y alcance de la investigación mencionada.

Sangolquí, 05 de septiembre del 2017

PATRICIO JAVIER PÉREZ ÁLVAREZ

C.C. 1721411385

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA
CARRERA DE INGENIERÍA EN ELECTRÓNICA Y TELECOMUNCACIONES

AUTORIZACIÓN

Yo, PATRICIO JAVIER PÉREZ ÁLVAREZ, autorizo a la Universidad de las Fuerzas Armadas ESPE publicar en la biblioteca Virtual de la institución el presente trabajo de titulación "DESARROLLO DE UN ALGORITMO DE VISIÓN ARTIFICIAL MEDIANTE EL PROCESAMIENTO DIGITAL DE IMÁGENES CON EL SENSOR KINECT ORIENTADO A LINUX" cuyo contenido, ideas y criterios son de mi autoría y responsabilidad.

Sangolquí, 05 de septiembre del 2017

PATRICIO JAVIER PÉREZ ÁLVAREZ

C.C. 1721411385

DEDICATORIA

Dedico este trabajo a Dios por haberme dado la salud, la vida y la capacidad para culminar esta meta.

A mis padres Gina y Xavier que siempre me han apoyado en todo lo que me he propuesto, brindándome sabios consejos y su amor incondicional que supo levantarme cuando sentía que ya no podía más.

A mi novia Estefanía que siempre estuvo conmigo en las buenas y en las malas brindándome su amor incondicional y ayudándome a seguir adelante cuando pensaba que todo iba mal.

A mi familia en general por siempre brindarme su apoyo ya sea estando cerca o a la distancia, nunca dejaron de confiar en mí.

AGRADECIMIENTO

“Apunta al sol y por lo menos llegarás a la luna”

Xavier Pérez

A Dios por poner en mi camino a personas tan maravillosas, por darme la oportunidad de estudiar en mi amada ESPE, y por guiarme a lo largo de toda mi carrera universitaria.

A mis padres que a base de amor supieron inculcar en mí los valores que hoy gobiernan sobre mi vida y que gracias a su apoyo incondicional siempre he seguido adelante.

A mis hermanos, María Belén y Paúl Andrés que gracias a sus logros y enseñanzas me motivan a ser mejor para alcanzarlos.

A Estefanía que es la mujer que ha sido mi compañera de vida, que con su amor y paciencia me supo comprender y ayudarme a ser mejor día a día, siempre seremos el mejor equipo.

A mis abuelitas, Emilia y Margarita por siempre estar pendientes de mi vida y de mis estudios.

A mi madrina Maritza por brindarme consejos y por ser un gran ejemplo de lucha y perseverancia.

A mis amigos que me han demostrado que son como familia para mí, que siempre han estado conmigo en los buenos y malos momentos.

A mis profesores que supieron dar lo mejor de cada uno para transmitir sus conocimientos así como también sus experiencias de vida.

A mi tutor Ing. Julio Torres por asesorarme en este trabajo, por poner luz cuando el camino estaba nubloso y por brindarme su amistad a lo largo de toda la investigación.

ÍNDICE DE CONTENIDOS

CERTIFICACIÓN ii
AUTORÍA DE RESPONSABILIDAD iii
AUTORIZACIÓN iv
DEDICATORIA v
AGRADECIMIENTO vi
ÍNDICE DE CONTENIDOS vii
ÍNDICE DE TABLAS xii
ÍNDICE DE FIGURAS xiii
RESUMEN xvi
ABSTRACT xvii
CAPÍTULO 1 1
INTRODUCCIÓN 1
1.1 Antecedentes 1
1.2 Justificación e Importancia 4
1.3 Alcance del Proyecto 6
1.4 Objetivos 7
1.4.1 General 7
1.4.2 Específicos 7
CAPÍTULO 2 8
VISIÓN ARTIFICIAL 8
2.1 Visión Artificial 8
2.1.1 Definición 8
2.1.2 Elementos de un sistema de visión artificial 8
2.2 Principios de funcionamiento 9

2.2.1 Captación 10
2.2.2 Pre-procesamiento 10
2.2.3 Segmentación 10
2.2.4 Descripción 10
2.2.5 Reconocimiento 11
2.2.6 Interpretación 11
2.3 Visión humana y visión artificial 11
2.3.1 Sistema de Visión Humana 11
2.3.2 Sistema de visión artificial 13
2.3.3 Comparación entre visión humana y visión artificial 14
2.4 Usos y Aplicación de la VA 16
2.4.1 Usos 16
2.4.2 Aplicación 16
2.5 Importancia 16
CAPÍTULO 3 18
IMAGEN, IMAGEN DIGITAL Y PROCESAMIENTO DIGITAL DE IMÁGENES 18
3.1 Imagen 18
3.1.1 Definición 18
3.2 Clasificación 18
3.2.1 Imagen Directa 18
3.2.2 Indicios 19
3.2.3 Ícono 19
3.2.4 Signos convencionales o símbolos 20
3.2.5 Imágenes naturales 20
3.2.6 Imágenes mentales 20

3.2.7 Imágenes creadas 21
3.2.8 Imágenes registradas 21
3.2.9 Función de las imágenes 22
3.3 Imagen digital 22
3.3.1 Definición 22
3.3.2 Imagen estática 22
3.3.3 Imagen dinámica 22
3.3.4 Formatos de imagen 23
3.3.5 Clasificación 23
3.4 Procesamiento Digital de imágenes 24
3.4.1 Definición 24
3.4.2 Etapas de procesamiento digital de imágenes 25
3.4.3 Técnicas del PDI 27
3.4.4 Campos de aplicación del Procesamiento de Imágenes 27
CAPÍTULO 4 28
HARDWARE Y SOFTWARE 28
4.1 Hardware 28
4.1.1 Sensor Kinect 28
4.1.2 Raspberry Pi 3 29
4.2 Software 31
4.2.1 GNU/Linux 32
4.2.2 Ubuntu 32
4.2.3 Raspbian 32
4.2.4 Python 32
4.2.5 OpenKinect 34
4.2.6 OpenCV 35

CAPÍTULO 5 37
DISEÑO DEL ALGORITMO 37
5.1 Dependencias necesarias para el desarrollo del algoritmo 37
5.1.1 Instalación de la librería Libfreenect 37
5.1.2 Instalación de la librería OpenCV 40
5.2 Descripción general del sistema 40
5.3 Algoritmo de detección del sensor Kinect 43
5.3.1 Captura 44
5.3.2 Pre procesamiento 45
5.3.3 Segmentación 46
5.3.4 Extracción 48
5.3.5 Identificación de objetos 53
5.4 Función de la cámara IP 54
5.5 Función de generar el mapa en 2D 62
CAPÍTULO 6 67
PRUEBAS Y RESULTADOS 67
6.1 Protocolo de pruebas 67
6.1.1 Identificación de un objeto visto desde la esquina 67
6.1.2 Validación del porcentaje para identificar el objeto potencialmente peligroso 67
6.1.3 Mapeo del entorno con un objeto no peligroso 67
6.1.4 Mapeo del entorno con tres objetos; uno de ellos es potencialmente peligroso 68
6.1.5 Mapeo del entorno con tres objetos; un objeto potencialmente peligroso parcialmente cubierto 68
6.2 Realización de pruebas y obtención de resultados 68
6.3 Cálculo de errores 84

CAPÍTULO 7 86

CONCLUSIONES Y RECOMENDACIONES 86

7.1 Conclusiones 86

7.2 Recomendaciones 87

Bibliografía 88

ÍNDICE DE TABLAS

Tabla 1 Visión Humana vs Visión Artificial..... 15

ÍNDICE DE FIGURAS

Figura 1. Principios de Funcionamiento de la Visión Artificial 10
Figura 2. Visión humana vs visión artificial 15
Figura 3. Imagen Directa 19
Figura 4. Indicios 19
Figura 5. Ícono 19
Figura 6. Referencias 20
Figura 7. Imagen Natural 20
Figura 8. Imagen Mental 21
Figura 9. Imagen Creada 21
Figura 10. Imagen Registrada 21
Figura 11. Imagen Vectorial y Mapa de Bits 24
Figura 12. Etapas del PDI 26
Figura 13. Partes que componen un Kinect 28
Figura 14. Descripción de una Tarjeta de memoria SD 30
Figura 15. Ejecución de glview 39
Figura 16. Kinect montado en el robot 41
Figura 17. Representación de los planos 41
Figura 18. Cámara IP en el techo 42
Figura 19. Imagen del ambiente controlado visto desde la cámara IP 42
Figura 20. Robot con las placas de color verde 43
Figura 21. PDI, algoritmo elaborado, imagen 44
Figura 22. Imagen de profundidad con la pérdida de 10px 46
Figura 23. Ángulo de visión del sensor Kinect 47
Figura 24. Ejemplo de la toma de mediciones 48
Figura 25. Triangulo para calcular la longitud de visión del Kinect 52
Figura 26. Imagen del área de pruebas desde la cámara IP 56
Figura 27. Máscara de la imagen de la cámara IP 56
Figura 28. Eje de referencia y eje trasladado 57
Figura 29. Ejemplo para el cálculo del ángulo entre los centros de las placas verdes 57

Figura 30. Punto proyectado la distancia de detección 59
Figura 31. Ejemplo de la proyección del punto del Kinect al objeto 59
Figura 32. Distancia entre la placa verde pequeña y el borde del Kinect 60
Figura 33. Representación del plano vertical sobre el plano horizontal 61
Figura 34. Centroides y punto P3 en el plano vertical 61
Figura 35. Plano horizontal con proyección de los centroides del plano vertical 62
Figura 36. Mapa cuadriculado 64
Figura 37. Longitud a sumar desde el centroide del objeto 64
Figura 38. Captura de datos de la cámara IP 65
Figura 39. Datos presentados en el mapa 65
Figura 40. Captura de datos de la cámara IP (peligroso) 66
Figura 41. Datos presentados en el mapa (peligroso) 66
Figura 42. Escenario prueba 1A vista desde la cámara IP 68
Figura 43. Profundidad prueba 1A 69
Figura 44. Máscara prueba 1A 69
Figura 45. Mapa prueba 1A 69
Figura 46. Escenario prueba 1B vista desde la cámara IP 70
Figura 47. Profundidad prueba 1B 70
Figura 48. Máscara prueba 1B 70
Figura 49. Mapa prueba 1B 71
Figura 50. Escenario prueba 2A 71
Figura 51. Máscara prueba 2A 72
Figura 52. Mapa prueba 2A 72
Figura 53. Escenario prueba 2B 72
Figura 54. Máscara prueba 2B 73
Figura 55. Mapa prueba 2B 73
Figura 56. Generación del mapa, prueba 3A 74
Figura 57. Generación del mapa, prueba 3B 74
Figura 58. Generación del mapa, prueba 3C 74
Figura 59. Generación del mapa, prueba 4A 75
Figura 60. Generación del mapa, prueba 4B 75
Figura 61. Generación del mapa, prueba 4C 76

Figura 62. Generación del mapa, prueba 4D 76
Figura 63. Generación del mapa, prueba 4E 76
Figura 64. Generación del mapa, prueba 4F 77
Figura 65. Generación del mapa, prueba 4G 77
Figura 66. Generación del mapa, prueba 5A 78
Figura 67. Generación del mapa, prueba 5B 78
Figura 68. Generación del mapa, prueba 5C 78
Figura 69. Generación del mapa, prueba 5D 79
Figura 70. Generación del mapa, prueba 5E 79
Figura 71. Generación del mapa, prueba 6A 80
Figura 72. Generación del mapa, prueba 6B 80
Figura 73. Generación del mapa, prueba 6C 80
Figura 74. Generación del mapa, prueba 6D 81
Figura 75. Generación del mapa, prueba 6E 81
Figura 76. Generación del mapa, prueba 6F 81
Figura 77. Generación del mapa, prueba 7A 82
Figura 78. Generación del mapa, prueba 7B 82
Figura 79. Generación del mapa, prueba 7C 83
Figura 80. Generación del mapa, prueba 7D 83
Figura 81. Generación del mapa, prueba 7E 83
Figura 82. Generación del mapa, prueba 7F 84
Figura 83. Generación del mapa, prueba 7G 84
Figura 84. Mapa, marcado los errores 85

RESUMEN

El presente trabajo de titulación propone un sistema de visión artificial que le permita a un robot *Dagu Wild Thumper* crear mapas en dos dimensiones de la zona por la que se desplaza (ambiente controlado) y a su vez le permita al mismo detectar objetos potencialmente peligrosos.

Para el desarrollo del algoritmo de visión artificial se usa como *hardware* al sensor *Kinect* y para la programación al *software Python*, cabe recalcar que para la detección de objetos en el ambiente controlado solo se usó el sensor de profundidad del *Kinect*, la cámara RGB del mismo solo se utilizó para ver la imagen por donde se transita.

El procesamiento digital de las imágenes de profundidad se lo hace en tiempo real por medio de una tarjeta *Raspberry Pi 3* con sistema operativo Raspbian, implementada en el robot mencionado.

Para la ubicación del robot en el ambiente controlado, se utilizó una cámara IP colocada en el techo del área de pruebas, de tal forma en que se obtenga una imagen de todo el ambiente controlado, de la misma manera esta imagen obtenida se procesa de manera digital en la *Raspberry Pi 3* para obtener la posición del robot en tiempo real dentro del área de pruebas y así realizar el mapa en 2D, mientras el robot sigue explorando el área.

En la representación del mapa, los objetos no peligrosos (para esta investigación cajas) son marcados en color negro, y los potencialmente peligrosos (para este trabajo una pelota) es marcada en rojo.

PALABRAS CLAVES:

- **SENSOR KINECT**
- **PROCESAMIENTO DIGITAL DE IMÁGENES**
- **VISIÓN ARTIFICIAL (VA)**
- **OPENCV**
- **SOFTWARE LIBRE**

ABSTRACT

The present work proposes an artificial vision system which allows a Dagu Wild Trumper robot to create 2D maps through the area where it moves (a controlled environment) and, in turn, detects potentially danger objects.

For the development of the artificial vision algorithm it is used as the main hardware component the Kinect sensor and the main software tool is the programming language Python. It should be stressed that for the object detection in the controlled environment it was only used the Kinect depth sensor. The RGB camera is only used to visualize the movement of the robot.

The digital processing of the depth images has been done in real time over a Raspberry Pi 3 board with Raspbian operating system, all of this implemented on the mentioned robot.

For the robot location in the controlled environment it has been used an IP camera mounted on the roof of the test area; in such a way to obtain an image of the whole controlled environment. Likewise, this image is digitally processed through the Raspberry Pi 3 board. In this way, the robot location is obtained in real time within the test area and while the robot is exploring it, it is also making the 2D map.

In this map, no dangerous objects (boxes for this project) have been colored in black and the potentially dangerous objects (a ball) has been colored in red.

KEYWORDS:

- **KINECT SENSOR**
- **DIGITAL IMAGE PROCESSING**
- **ARTIFICIAL VISION (VA)**
- **OPENCV**
- **FREE SOFTWARE**

CAPÍTULO 1
INTRODUCCIÓN

1.1 Antecedentes

La robótica desde sus inicios ha pretendido usar controles remotos para el movimiento y manejo de los robots un buen ejemplo es el "Lunojod 1" un robot astromóvil que fue el primero en pisar la luna en 1970 y que era controlado de forma remota desde la Tierra. (Flores, 2010)

En la navegación robótica el principal problema se descompone en la evasión de obstáculos, la construcción de un mapa del entorno, la localización automática del robot en el mismo y, finalmente, la navegación desde un punto inicial hasta uno final, utilizando el mapa construido. (Cristóforis & Nitsche, 2010)

Por otro lado, la visión artificial es el conjunto de técnicas que permiten a un sistema la obtención de información por métodos visuales, tanto de manera automática como asistida por una persona. (Gutiérrez, 2016)

La capacidad de visión de un ser humano es (al menos en la actualidad) irreproducible. Es un prodigio de adaptación y flexibilidad. Las experiencias de sistemas automáticos que ven en entornos no controlados (sistemas de conducción automática de vehículos, guiado de robots) solo se aproximan a nuestra capacidad visual. Sin embargo, con su pobre capacidad, hay una ventaja fundamental en los sistemas automáticos a la hora de hacer controles y es su repetitividad. Un ser humano, con los medios necesarios, siempre hará mejor un número pequeño de controles visuales que un sistema automático. En cambio, si debemos realizar un gran número controles a lo largo de un tiempo mayor, nuestra capacidad de repetición decae rápidamente y se ve afectada por distintos factores como el sueño, enfermedades, etc. Se tiene que conseguir sistemas robustos que reemplacen las inspecciones de bajo valor añadido, tediosas, o de condiciones adversas, pero sin eliminar totalmente una cierta supervisión humana de alto valor añadido. En condiciones "no habituales" las personas somos más flexibles e intuitivas. (Gutiérrez, 2016)

A través del tiempo se han diseñado y desarrollado métodos de captura de movimiento los mismos que han sido aplicados en diferentes áreas; la captura del movimiento de objetos ha sido elemento de estudio durante décadas, un ejemplo claro es la aparición de la cámara fotográfica que sirve para captar imágenes y le dio paso a la cámara de video que sirve para capturar el movimiento de objetos. Esta técnica se ha aplicado en teléfonos celulares, sensores, sistemas infrarrojos, entre otros, es así que se ha globalizado la importancia de su uso.

Kinect es un dispositivo, inicialmente pensado como un simple controlador de juego, que gracias a los componentes que lo integran: sensor de profundidad, cámara RGB, arreglo de micrófonos y sensor de infrarrojos (emisor y receptor), es capaz de capturar el esqueleto humano, reconocerlo y posicionarlo en el plano. Gracias a toda la información que captura este dispositivo, los desarrolladores de *software* pueden hacer uso de él para programar toda una serie de aplicativos cuyo activo principal es la interacción con los elementos virtuales a través de los distintos movimientos del cuerpo humano. (Murillo, 2012)

Para el uso del sensor *Kinect* en Linux se debe tener en cuenta la instalación de las librerías que hacen posible el funcionamiento de todas las herramientas que dispone el sensor ya que sin la instalación completa de las mismas el sensor no va a presentar un trabajo adecuado o no se va a tener acceso a todos los sensores y cámaras que dispone el *Kinect*. (Mejía, 2015)

Se eligió el sensor *Kinect* principalmente por 3 razones; es capaz de detectar la morfología de los objetos, lo que le da una clara ventaja sobre las cámaras IP, puede trabajar también en ambientes con luz limitada es decir en la noche, lo que no limita la hora en la que se puede realizar la toma de datos, y a su precio ya que un sistema de visión artificial resultaría complejo y costoso según National Instruments (2016) costando alrededor de 12000 USD, ya que cada componente supera las 3 cifras de costo, por ejemplo, el NI CVS-1458 cuyo precio es de 3850 dólares cada uno. Es por ello que el sensor *Kinect*, que es actualmente un dispositivo accesible se ha convertido en la base de muchos proyectos de investigación ya que brinda la posibilidad de integrarlo fácilmente. Por ejemplo, “SISTEMA DE EDUCACIÓN PARA NIÑOS DE

3 A 5 AÑOS, MEDIANTE UN ROBOT CONTROLADO POR EL SENSOR *KINECT*". (Ilvay, 2014)

El procesamiento digital de imágenes es el almacenamiento, transmisión y representación de información de imágenes captadas de forma analógica y gestionada de forma digital. (García V. , 2009)

En términos generales los pasos fundamentales del procesamiento digital de imágenes son: adquisición, realce, restauración de la imagen, procesamiento de la imagen en color, ondeletas, y procesamiento de multiresolución, compresión, procesos morfológicos, segmentación, representación y descripción y reconocimiento. (García V. , 2009)

El interés del procesamiento digital de imágenes en este proyecto se basa principalmente en el tratamiento de los datos de una escena para la percepción autónoma de una máquina en este caso una tarjeta *Raspberry Pi 3*. (García V. , 2009)

La visión es el sentido más avanzado, y no es sorpresa que las imágenes jueguen un papel primordial en la percepción humana. A diferencia de los humanos que solo perciben una mínima parte del espectro electromagnético, los equipos de adquisición de imágenes digitales pueden trabajar con todo el rango del espectro. (García V. , 2009)

Los seres vivos poseen patrones de movimiento corporales particulares como caminar, internos como el ritmo cardíaco, voluntarios como hablar e involuntarios como los reflejos; estos patrones permiten al ser humano identificar el movimiento ejecutado a través de los sentidos y órganos involucrados que permiten a las personas la identificación e interpretación de su ambiente. Lo que se buscó con este trabajo es imitar a este tipo de sistemas biológicos.

El presente proyecto realizó el diseño de un algoritmo de visión artificial mediante el uso del procesamiento digital de las imágenes con el sensor *Kinect* para un robot *Dagu Wild Thumper*, que le permita al mismo generar mapas 2D del lugar en el que se desplaza por lo cual se convierte en interés de estudio, para incursionar en ambientes controlados y de aplicación directa en el proyecto de investigación "LOCALIZACIÓN DE TNT Y PÓLVORA DE BASE DOBLE A TRAVÉS DE SENSADO QUÍMICO

EN UN ENTORNO CONTROLADO MEDIANTE ROBÓTICA COOPERATIVA (SmellRobSense)". (Gaibor & Mediavilla, 2016)

El robot mediante este algoritmo detectó los objetos potencialmente peligrosos presentes, y realizó un mapa 2D de su entorno.

1.2 Justificación e Importancia

En los últimos años la robótica ha pasado a formar parte importante en áreas jamás pensadas como la medicina, comercial, doméstica, militar entre otras; con aplicaciones diversas y con desarrollo de varias herramientas necesarias para los seres humanos. La Robótica ha evolucionado hacia los sistemas móviles autónomos, que son aquellos capaces de desenvolverse por sí mismos en entornos desconocidos y parcialmente cambiantes sin necesidad de supervisión. (Jjazdanethe, 2009)

Recientemente, el campo de la robótica, especialmente la robótica móvil se ha identificado como una de las áreas más importantes de la investigación debido a su enorme potencial para las exploraciones autónomas en diferentes dominios peligrosos o tóxicos e inaccesibles. Hay varias aplicaciones de la vida real con sistemas robóticos, como la exploración planetaria, rescate, la limpieza o mapeado de entorno desconocido, pero el problema principal en la exploración y la búsqueda es el alto coste computacional y el tiempo de ejecución debido a la complejidad del entorno para una solución de un solo robot. (Martinez, 2015)

Las técnicas usadas en visión artificial se han desarrollado a gran velocidad en las últimas décadas; los primeros sistemas se basaron en imágenes binarias que se procesaban en bloques, ventanas o píxeles. Gracias al desarrollo de nuevos algoritmos de visión artificial se logró reconocer el contorno de objetos y su posición dentro de una imagen, estos algoritmos tenían la limitante de no poder operar en diferentes tipos de iluminación. Posteriormente se introdujeron los sistemas de intensidad de gris, en estos cada uno de los píxeles de la imagen es representado con un número proporcional a la intensidad de gris de dicho elemento; esta técnica podía operar en diferentes tipos de iluminación, ya que pueden encontrar los bordes de los objetos utilizando cambios en los valores de intensidad de los píxeles. (Shun, 2015)

La visión por computador actualmente comprende tanto la obtención como la caracterización e interpretación de los objetos contenidos en una imagen y es uno de los elementos sensoriales más atractivos para incrementar la autonomía en la robótica ya que provee de información relevante sobre su entorno físico inmediato y son bastante baratos en comparación con otros; la única desventaja es la extracción de la información a partir de una imagen. Es aquí donde aparece la tecnología del sensor *Kinect*. (Shun, 2015)

Mediante el sensor *Kinect* se abre un nuevo panorama en el campo de estudio de la robótica; en el pasado la movilidad de los robots dependía del control del ser humano, en ambientes planos que se alejaban de la realidad; en la actualidad se puede hacer referencias reales y dinámicas de un espacio y los robots pueden movilizarse de forma autónoma y auto-localizarse en diferentes ambientes.

Para el desarrollo de la investigación se planteó el uso del sensor *Kinect* y un sistema robótico los cuales se basan en el mapeo de profundidad para determinar la ubicación de un objeto en un espacio. Similar como un radar en base a sonido (sonar), la unidad mide el tiempo que toma la luz en rebotar en un objeto y regresa al origen. Esto crea una visualización en 2D de un objeto. (Codigobit, 2010)

En temas de investigación relacionados con el sensor *Kinect* se encontró varios trabajos realizados en diferentes áreas como: detección de objetos (Velasco N. , 2013), control de robots mediante la detección de gestos corporales (Chávez & Tatayo, 2015), aplicaciones educativas de apoyo para niños (Ilvay, 2014), adquisición y procesamiento de imágenes (Leoro, 2013), video juegos orientados al aprendizaje de materias escolares. (Calahorrano, 2015), detección de movimientos a través del reconocimiento de las articulaciones empleando esqueletización (Villalba, 2015), reconocimientos de patrones para seguridad inteligente (Ayala & Rosa, 2013). En todos los trabajos mencionados el sensor *Kinect* trabaja un punto fijo desde el cual realiza la toma de datos de los movimientos del entorno, para este proyecto se montó el sensor *Kinect* en el robot, lo que invirtió los papeles ya que el sensor *Kinect* pasó a ser móvil y el ambiente fijo.

De los temas de investigación mencionados se debe recalcar que no se encontró trabajos realizados en *software* libre en lo que a detección de objetos se refiere a nivel local.

El aporte que se realizó con este proyecto de investigación es disponer de un algoritmo de visión artificial para el robot *Dagu Wild Thumper*, en un ambiente controlado mediante el sensor *Kinect* orientado a sistema operativo Linux. Un requerimiento para el robot de este proyecto, fue la posibilidad de generar mapas 2D del entorno en donde se encuentra y diferenciar objetos potencialmente peligrosos.

Para este proyecto un ambiente controlado se refiere a un área de 2,73 x 1,90 metros aproximadamente con obstáculos estáticos (a ser modificados para cada prueba), con iluminación artificial y con acceso a señales de radio frecuencia para el envío de información del mapa generado al usuario.

Para este trabajo de investigación se utilizó 3 cajas de diferentes dimensiones, la primera de (33 x 18,5 x 11,5) cm la segunda de (43 x 30 x 34) cm y la tercera (45 x 23 x 12,5) cm. Para el objeto potencialmente peligroso se utilizó una pelota con diámetro de 40cm.

El robot *Dagu Wild Thumper*, ha sido utilizado, modificado e implementado en tres trabajos de investigación, siendo la última modificación realizada por (Alejandro & Venegas, 2017), del cual se dispone de un sistema de movimiento controlado por las teclas del computador W (adelante), S (atrás), A (izquierda) y D (derecha), para el control del robot, este sistema de movimiento fue el utilizado para esta investigación.

1.3 Alcance del Proyecto

El presente proyecto se desarrolló en base al proyecto de Investigación "LOCALIZACIÓN DE TNT Y PÓLVORA DE BASE DOBLE A TRAVÉS DE SENSADO QUÍMICO EN UN ENTORNO CONTROLADO MEDIANTE ROBÓTICA COOPERATIVA (SmellRobSense)"; donde se diseñó un algoritmo de visión artificial mediante el procesamiento digital de imágenes con el sensor *Kinect* para un robot, que le facilitó al mismo detectar objetos potencialmente peligrosos a su alrededor y generar mapas 2D del lugar en el que se desplaza.

El procesamiento digital de imágenes se lo realizó en una tarjeta *Raspberry Pi 3* que está implementada en el robot.

Para el posicionamiento del robot se utilizó una cámara IP que fue instalada en el techo del área de pruebas, para que mediante un algoritmo similar al usado con el *Kinect*, obtenga ubicación y el ángulo en el que apunta el robot.

1.4 Objetivos

1.4.1 General

Diseñar e implementar un algoritmo de visión artificial mediante el procesamiento digital de imágenes con el sensor *Kinect* en *software* libre.

1.4.2 Específicos

- Analizar los algoritmos de detección de objetos por captura de movimiento.
- Desarrollar un algoritmo de visión artificial que permita realizar un mapeo del ambiente mediante el sensor *Kinect* en *software* libre.
- Realizar el procesamiento digital de las imágenes en tiempo real.
- Detectar objetos potencialmente peligrosos.
- Desarrollar el protocolo de pruebas.
- Evaluar el rendimiento del algoritmo en una tarjeta *Raspberry Pi 3*.

CAPÍTULO 2
VISIÓN ARTIFICIAL

2.1 Visión Artificial

2.1.1 Definición

La visión artificial o también conocida como visión por computador, es un proceso por el cual se obtiene e interpreta imágenes del mundo real, con el fin de obtener información numérica o simbólica que pueda ser procesada por un computador; esta información es necesaria para controlar una máquina, robot o un proceso industrial, etc.

La visión por computador tiene la finalidad de imitar la visión del ser humano; tal y como los seres humanos utilizan la visión para interpretar y comprender el mundo que los rodea, con la visión artificial se pretende tener el mismo efecto para que las computadoras puedan procesar una imagen o secuencia de imágenes y actuar según convenga la situación.

> Hoffman dice que: “tal vez estemos en una época en la que el reconocimiento de objetos sea parte de la Visión Artificial (VA), pero no sería extraño que en el futuro asistiéramos a una separación de esta materia de la VA como ya la hubo del procesamiento de imágenes y el análisis de imágenes. (Visión Artificial e Interacción sin mandos, 2010)

La detección de objetos es una parte de la visión artificial que estudia cómo detectar la presencia de objetos en una imagen; se pueden nombrar dos fases en la detección de objetos: la extracción de información de una imagen o la búsqueda y detección de objetos en dicha imagen.

2.1.2 Elementos de un sistema de visión artificial

Un sistema de visión artificial se compone de los siguientes elementos:

- Fuente de luz

Es un aspecto muy importante, proporciona las condiciones de luz uniformes e independientes del entorno; facilita también la extracción de rasgos de interés.

- Sensor de imagen

Recoge las características del objeto que se encuentra en estudio.

- Tarjeta de captura o adquisición de imágenes

Es la interfaz entre el sensor y el computador que permite al mismo disponer la información capturada por el sensor de imagen.

- Algoritmos de análisis de imágenes

Es el punto inteligente del sistema; extrae la información de las imágenes capturadas, aplica las transformaciones necesarias y obtiene los resultados para los cuales fue diseñado.

- Computador

Sistema que analiza las imágenes captadas por el sensor de imagen; extrae la información de interés y ejecuta los algoritmos diseñados.

- Sistema de respuesta en tiempo real

Con la información extraída, el sistema de visión artificial puede tomar las decisiones necesarias y oportunas.

2.2 Principios de funcionamiento

Cada aplicación de visión artificial puede tener sus características propias, sin embargo, se puede decir que existe un tronco común de principios de funcionamiento, generalmente se divide en seis áreas (Porras, 2012):
(Ver Figura 1, las flechas indican la secuencia a seguir)

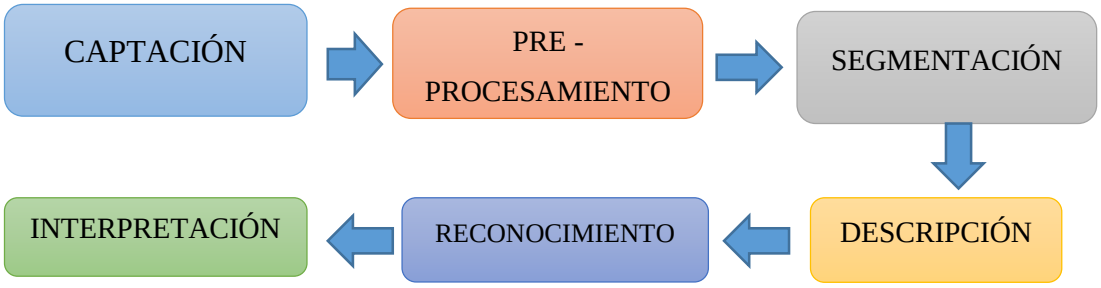

Figura 1. Principios de Funcionamiento de la Visión Artificial

2.2.1 Captación

Proceso en el cual se obtiene la imagen. Tiene tres aspectos importantes para obtener una imagen de calidad, que no altere las condiciones reales y no agregue errores considerables al sistema: la resolución, la profundidad del color y la nitidez.

2.2.2 Pre-procesamiento

Método en el cual se quita o elimina características no deseadas de la imagen como por ejemplo el ruido.

Esta etapa permite el mejoramiento de algunas características importantes en las imágenes originales y así mejorar el proceso de segmentación.

2.2.3 Segmentación

Divide la imagen en objetos de interés; la técnica más usada en la segmentación es la umbralización, este método permite convertir una imagen de manera que los objetos queden separados del fondo.

2.2.4 Descripción

Obtención de todas las características de un objeto como: color, tamaño, forma; y así poder diferenciarlo o distinguirlo de otros objetos.

2.2.5 Reconocimiento

Proceso en el que se diferencia un objeto dentro de una escena; en el área de descripción se obtiene todas las características de un objeto en específico; por el contrario, en el reconocimiento se diferencia un objeto en toda la escena captada.

2.2.6 Interpretación

Después de que la imagen fue procesada y segmentada y se ha extraído las características, se da un significado al conjunto de objetos reconocidos, se hace una serie de pruebas y mediciones a los elementos que aparecen en una escena.

2.3 Visión humana y visión artificial

2.3.1 Sistema de Visión Humana

Según un estudio realizado en Instituto Max Planck de Psicolingüística, la visión humana (Orfila, 2015), es el sentido más importante de los seres humanos, hay tres causas principales que explican por qué la vista es el sentido más importante:

- En primer lugar, se debe a la estructura del cerebro: “Casi el 50% del cerebro se dedica al procesamiento visual”,
- Otra explicación hace referencia a que “vemos constantemente durante el día, aunque parpadeemos, nuestra mente rellena el hueco, de manera que experimentamos una imagen constante.”
  Eso es diferente de saborear, por ejemplo, no estamos comiendo todo el día, ni tampoco oliendo”
- Hay una tercera explicación más social: coordinamos el conjunto de experiencias a través de la vista. De este modo, se puede usar una palabra como "mira" en el sentido literal, pero también se puede usar con sentidos diferentes.

(Orfila, 2015)

En la vista hay más de dos millones de terminaciones nerviosas; el ojo es un órgano que detecta luz y es la base del sentido de la vista.

El ojo humano posee un lente llamado cristalino que se ajusta según la distancia; la pupila cuyo diámetro lo regula el iris y la retina que es sensible a la luz.

El sentido de la vista en los seres humanos es muy complejo y necesita de dos elementos básicos: el ojo y el cerebro; la luz también es un factor fundamental en la visión, ya que sin ella no podríamos ver y es la que se introduce por el ojo captando la imagen que llegará al cerebro.

> El cerebro es capaz, de manera inconsciente, de determinar la distancia a los objetos, de reconocerlos en diferentes posiciones, aunque se encuentren rotados y con información parcialmente oculta. En definitiva, el cerebro presenta una sofisticación en la percepción que ni ahora ni en mucho tiempo habrá posibilidad de implementar artificialmente.
>
> (Visión artificial e interpretación sin mandos, 2010)

Recorrido de la luz:

1) La luz pasa a través de la córnea y llega a la pupila que se contrae o expande según su intensidad. La pupila será más pequeña cuanta más luz haya para evitar deslumbramientos. En habitaciones o lugares en penumbra aumentará de tamaño para dejar entrar más cantidad de luz.
2) El cristalino del ojo será quien proyecte las imágenes enfocadas en la retina. Puede aplanarse o abombarse según lo cerca o lejos que esté el objeto que veamos.
3) La retina recibe la imagen invertida en sus paredes. La luz estimula los conos y los bastones quienes transforman esa información en impulsos nerviosos. Esta electricidad se trasladará al cerebro a través del nervio óptico. El cerebro es quien realmente ve las imágenes. Endereza la imagen invertida de la retina e interpreta la información de color, tamaño, posición, etc.

(FotoNostra, 2008)

La retina está compuesta por dos tipos de fotorreceptores, bastones y conos:

Los conos se encargan de la percepción del color y los detalles finos de una imagen, existen aproximadamente 6 millones y cada cono tiene conexión a varias neuronas, se pueden dividir en conos rojos (64%), conos verdes (32%) y conos azules (2%) los

mismos proveen de sensibilidad de color al ojo. Basándose en la información aportada por los conos, el cerebro construye la sensación de color.

> Estudios fisiológicos han revelado que existen tres tipos de conos, denominados tipos S, M y L. Los S son más sensibles a las ondas cortas (azules– 450nm), los M a las medias (verde – 540 nm) y los L a las de longitudes largas (rojo -650 nm). Este hecho ha dado base a la teoría del triestimulo, de manera que el color se puede representar en una base de tres componentes fundamentales: rojo, verde y azul (RGB – Red, Green, Blue).
>
> (Alejandro, 2009)

Por el contrario, los bastones son más de 100 millones y son capaces de detectar la intensidad lumínica, dan una impresión general del campo de visión y también son los responsables de la sensibilidad a niveles bajos de iluminación; los bastones no son sensibles a los colores. Por ejemplo: Un objeto que a la luz del día tiene colores vivos, observado a la luz de la luna aparece sin colores, esto es debido a que tan solo los bastones están estimulados.

La lente del ojo está formada por capas concéntricas de células fibrosas, compuesto principalmente por agua (entre el 60 y 70%), grasa (6%) y proteínas; aquí se absorbe aproximadamente el 8% del espectro de luz y también una gran proporción de luz infrarroja y ultravioleta.

> El ojo humano no puede tener una resolución exacta porque nuestra visión no está enfrascada en una imagen fija. Al contrario, podemos mover los ojos hacia los lados, hacia arriba y hacia abajo, creando un campo de visión mucho más grande y con distintos grados de inclinación y, además, también podemos mover la cabeza en varias direcciones para capturar más imágenes aún.
>
> (Gonzalez, 2015)

2.3.2 Sistema de visión artificial

Con el avance de la tecnología, la creación de sensores, computadoras y algoritmos de procesamiento potentes han permitido que la visión artificial evolucione de a poco con el pasar de los años; la visión de computador es muy eficaz para tareas de visión

repetitiva, la extracción de información se da de forma automática y sin contacto con el objeto.

Un sistema de visión artificial se compone por varios subsistemas capaces de realizar dos funciones básicas:

- Captar la información de una escena real mediante una imagen.
- Analizar las imágenes y extraer la información que contienen.

> "La visión artificial, es una técnica de captación óptica, limpia, segura y muy versátil. El análisis de las imágenes obtenidas permite detectar en un objeto características físicas invisibles al ojo humano o establecer o comprobar alguna propiedad o medida predeterminada."
>
> (Alejandro, 2009)

A diferencia del sistema de visión humana, el Sensor *Kinect* 1.0, es un dispositivo en forma de barra, usa un proyector infrarrojo y una cámara que escanea la escena y envía esta información a un microchip preparado para capturar el movimiento de objetos y/o personas en 3D.

2.3.3 Comparación entre visión humana y visión artificial

El sentido de la vista proporciona al ser humano el 75% de la información que recibe el cerebro, pero no es infalible; mientras que la visión artificial extrae características visuales relevantes utilizando procedimientos automáticos. (Veciana, 2011) *(Ver* Figura 2*)*

En la siguiente tabla, se realiza una comparación entre Visión Humana y Visión Artificial:

Tabla 1

Visión Humana vs Visión Artificial

VISIÓN HUMANA	VISIÓN ARTIFICIAL
Poca precisión apoyándose en instrumentos de medida	Mejor midiendo magnitudes físicas
Poco consistente por la fatiga	Mejor para la realización de tareas rutinarias
Proyección sobre la retina, sus fotorreceptores son los conos y bastones	Proyección sobre el sensor, sus elementos fotosensibles son los semiconductores
Transmite la información a través del nervio óptico	Transmite la información a través de señal de imágenes y video
La velocidad de respuesta de la visión humana es de 60 milisegundos	La velocidad de respuesta es de 0,01 milisegundos para dispositivos actuales
Mejor en tareas de alto nivel de proceso	Mejor en tareas de bajo nivel de proceso
Mejor adaptación a situaciones imprevistas	Mejor en trabajar en ambientes muy peligrosos

Fuente: *(Dioxido, 2017)*

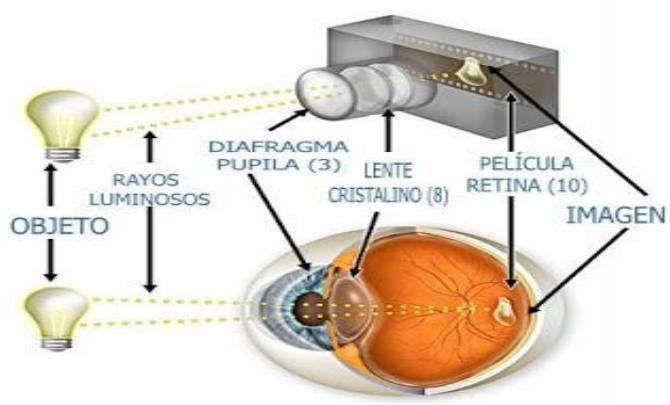

Figura 2. Visión humana vs visión artificial

Fuente: (García R. M., 2008)

2.4 Usos y Aplicación de la VA

2.4.1 Usos

La visión artificial abarca un sinnúmero de usos que se podrían simplificar de la siguiente manera:

- Control de procesos.
- Control de calidad.
- Aplicaciones no industriales.

2.4.2 Aplicación

Una aplicación puede generar una o más funciones de procesamiento de imágenes, que cuando se combinan pueden crear soluciones.

Con la visión artificial se puede:

- Automatizar tareas repetitivas de inspección realizadas por operadores.
- Realizar inspecciones de objetos sin contacto físico, vehículos automatizados, establecimiento de relaciones espaciales entre varios objetos (guiado de robots).
- Identificación e inspección de objetos.
- Determinación de la posición de los objetos en el espacio y las coordenadas importantes del mismo.
- Análisis de imágenes por *software*.

(Visión Artificial, 2014)

2.5 Importancia

La VA se está convirtiendo rápidamente en un factor clave para la calidad dentro de los procesos de automatización industrial; la visión artificial permite el proceso de producción sin fatigas ni distracciones.

La VA junto con la inteligencia artificial crea mecanismos de trabajo porque es capaz de recibir órdenes, captar información y de procesar la misma, tomar decisiones

y ejecutarlas; también puede ser empleada en varias ramas de estudio como la medicina, robótica, etc.

En el presente proyecto la VA fue importante para la detección de objetos por medio del sensor *Kinect*, el mismo que fue montado en el robot, así como también sirvió para la ubicación del robot por medio de una cámara IP instalada en el techo del área de trabajo.

CAPÍTULO 3

IMAGEN, IMAGEN DIGITAL Y PROCESAMIENTO DIGITAL DE IMÁGENES

3.1 Imagen

3.1.1 Definición

Una imagen (del latín imago) es una figura o representación de alguna cosa, escena o suceso.

Una imagen se puede dividir en dos dominios:

- **Dominio inmaterial de las imágenes en nuestra mente:** éstas aparecen como visiones, fantasías, imaginaciones, esquemas o modelos; son el resultado, en la imaginación y en la memoria, de las percepciones externas, subjetivas por el individuo.
- **El segundo es el dominio de las imágenes como representación visual:** diseño, pinturas, fotografías, imágenes cinematográficas, televisivas e infografías. Estas imágenes son las percibidas por los sentidos en el mundo exterior. Son materiales porque existen en el mundo físico de los objetos.

(Definición de Imagen, 2014)

3.2 Clasificación

Las imágenes son representaciones gráficas que comunican o transmiten algo; se pueden clasificar por:

3.2.1 Imagen Directa

Imágenes que se observan de manera natural, tienen una interpretación directa o de reconocimiento. (Ver Figura 3)

a) ***Playa***

b) ***Tigre***

Figura 3. Imagen Directa

3.2.2 Indicios

Son imágenes naturales que mantienen una relación lógica con lo que significan; por ejemplo, si vemos humo significa que hay fuego en aquel lugar. (Ver Figura 4)

a) Incendio

b) Huellas en la arena

Figura 4. Indicios

3.2.3 Ícono

Son las imágenes creadas por el ser humano, que mantienen una relación con lo que se pretende representar. (Ver Figura 5)

Figura 5. Ícono

3.2.4 Signos convencionales o símbolos

Son imágenes que se les asigna un significado general que permiten al receptor interpretar y reconocer el mismo. (Ver Figura 6)

Figura 6. Referencias

3.2.5 Imágenes naturales

Imágenes de percepción diaria, son imágenes reales. (Ver Figura 7)

a) Mariposa

b) Hoja

Figura 7. Imagen Natural

3.2.6 Imágenes mentales

Su principal característica es que son imágenes intangibles; es de naturaleza mental y no es necesario que exista una estimulación visual del exterior, se relaciona con la actividad del cerebro de aquí se fundamenta las imágenes en los sueños y las fantasías. (Ver Figura 8)

a) Manos

b) Reloj

Figura 8. Imagen Mental

3.2.7 Imágenes creadas

Imágenes producidas por el hombre con intención comunicativa. (Ver Figura 9)

a) Crucero

b) Fotografía

Figura 9. Imagen Creada

3.2.8 Imágenes registradas

Imágenes generadas a través de los medios de comunicación, pueden ser imágenes de anuncios, comics, imagen televisiva, etc. (Ver Figura 10)

a) Cómic

b) Anuncio

Figura 10. Imagen Registrada

3.2.9 Función de las imágenes

Se puede identificar diferentes funciones para las imágenes, detalladas en el siguiente listado:

- Informar.
- Distinguir y reconocer.
- Comunicar y vender.
- Aprender y reconocer.
- Entretenimiento.
- Expresión.

3.3 Imagen digital

3.3.1 Definición

Una imagen digital es una representación bidimensional de la realidad de una imagen captada, que utiliza bits (unos y ceros).

> Imagen digital: Es un producto del desarrollo de la informática que tiene como antecesor a la fotografía. La imagen digital toma vida mediante un archivo de diferentes formatos, que puede ser almacenado en un ordenador, enviado por correo electrónico e incluso ser impreso. (EcuRed, Imágen Digital, 2012)

3.3.2 Imagen estática

La imagen estática es el conjunto de elementos visuales (imágenes, signos, iconos, gráficos, etc.) que están presentes en un sistema multimedia y carecen de parámetros de tiempo esto quiere decir que son imágenes fijas o ilustraciones.

3.3.3 Imagen dinámica

La imagen dinámica es aquella que se caracteriza por el movimiento que presenta, crea en el espectador la sensación de movimiento; está formada por: imagen y el área dinámica o interactiva.

> Las imágenes dinámicas aparecen en tiempo de ejecución cuando el usuario desplaza el puntero sobre el área dinámica en tiempo de ejecución. Son de gran utilidad en las diapositivas que están repletas de detalles o en las que contienen botones o barras de herramientas que requieren una explicación. (Jaisingh, 2017)

3.3.4 Formatos de imagen

Hace referencia a los distintos formatos en los cuales se puede guardar una imagen, cada uno corresponde a una extensión específica del archivo que lo contiene.

- JPEG (Joint Photographic Experts Group) es un formato bastante flexible para almacenar imágenes optimizadas del mundo real, usa una distribución de 24 bits/pixel, cada imagen contiene alrededor de 16 777 216 colores, el algoritmo de compresión es de tipo *lossy*. Los archivos de este tipo al ser comprimidos resultan más pequeños que los de tipo GIF. Una fotografía digitalizada en formato JPG no permite al ojo humano notar las diferencias con una foto convencional.
- GIF (Graphic Interchange Format), estándar creado por Compuserve. Se distribuye a 8 bits/pixel, contiene aproximadamente 256 colores, tiene una característica distintiva y es que puede lograr transparencias en el fondo y puede ser animado.
- PNG (Portable Network Graphic) es el nuevo formato de compresión de imágenes para la *Web*. Entre sus más destacadas ventajas están la gran calidad y la alta compresión.

  Conjuga lo mejor de los formatos que habitualmente se han venido utilizado, que son .GIF y .JPG. Es de uso libre, es decir, no es necesario pagar ningún tipo de licencia para usarlo.

  (EcuRed, Imágen Digital, 2012)

3.3.5 Clasificación

Las imágenes digitales estáticas se dividen en dos tipos:

- Imágenes Vectoriales

Son representaciones geométricas de objetos (círculos, polígonos, arcos, rectángulos o segmentos). Están representadas por fórmulas matemáticas (un rectángulo está definido por dos puntos; un círculo, por un centro y un radio; una curva, por varios puntos y una ecuación). El procesador traducirá estas formas en información que la tarjeta gráfica pueda interpretar.

Se puede aplicar fácilmente transformaciones geométricas (ampliación, expansión, posición etc.) sin perder información (no se distorsiona), son representaciones simples de poco peso que se componen de rellenos y contornos perfectamente definidos.

- Mapa de bits

Es una imagen creada sobre una cuadrícula, cada uno de esos cuadros se denomina pixel, el mismo que gurda información del color, entre más pixeles más grande es la imagen; se guarda como fichero.

No pueden ser sometidas a transformaciones sin sufrir pérdida de información (distorsión de la imagen). Ver Figura 11

Figura 11. Imagen Vectorial y Mapa de

3.4 Procesamiento Digital de imágenes

3.4.1 Definición

El Procesamiento Digital de Imágenes (PDI) es el procesamiento, almacenamiento, transmisión y representación de imágenes digitales por medio de una computadora o

por un método óptico; la adquisición de una imagen digital se puede dar por medio de dispositivos (scanner, cámara digital, etc.) o directamente por medio de programas informáticos.

En general el PDI consiste en mejorar ciertos aspectos de las imágenes, extraer ciertas características o propiedades y hacer en ellas más evidentes ciertos detalles que se desean hacer notar.

> El desarrollo de los métodos de procesamiento digital de imágenes tiene su origen en dos áreas principales de aplicación: el mejoramiento de la información pictórica para la interpretación humana, y el procesamiento de datos de la imagen para la percepción de máquina autónoma en el que se incluyen etapas de transmisión y/o almacenamiento de estos datos [de la imagen]. (Geologica, 2013)

Existen técnicas para el procesamiento de imágenes recopiladas en tres grandes grupos:

- Modificación del color.
- Modificación de la imagen.
- Generación de efectos.

El procesamiento de imágenes se puede dar por:

- Procesamiento Óptico.
- Procesamiento Digital.

3.4.2 Etapas de procesamiento digital de imágenes

Las etapas del PDI utilizadas en este proyecto son las siguientes: (*Ver Figura 12, las flechas indican la secuencia a seguir)*:

Figura 12. Etapas del PDI

3.4.2.1 Captura

Esta etapa se enfoca en las propiedades para la captura: tipo de cámara, distancia al objeto, iluminación, pixeles, etc.

3.4.2.2 Pre procesamiento

Se reduce las características que no son de interés de la imagen captada: fondo, ruido, otros objetos, etc.

3.4.2.3 Segmentación

Reconoce, identifica y extrae cada uno de los objetos presentes en la imagen captada; se denomina segmentación porque es el proceso en el cual la imagen digital se divide en varias partes o en los objetos que la conforman.

3.4.2.4 Extracción de características

Selecciona y extrae las características deseadas para la identificación de objetos.

3.4.2.5 Identificación de objetos

Principalmente utiliza un modelo de toma de decisión para clasificar los objetos captados.

3.4.3 Técnicas del PDI

3.4.3.1 Digitalización y cuantificación

La digitalización supone un cambio radical en el tratamiento de la información y la cuantificación es uno de los pasos que se sigue para lograr la digitalización de una señal analógica. Básicamente, la cuantificación lo que hace es convertir una sucesión de muestras de amplitud continua en una sucesión de valores discretos preestablecidos según el código utilizado.

3.4.3.2 Compresión

Consiste en reducir o comprimir la información con el fin de reducir el costo de la transmisión de la imagen.

3.4.3.3 Análisis

Extrae información significativa de la imagen.

3.4.3.4 Reconstrucción

Obtención de información de un objeto en base a sus proyecciones, ejemplo: tomografía, imagen tomada desde un satélite, etc.

3.4.3.5 Transformaciones

Combinación de imágenes de distintos sensores, combinaciones multitemporales, etc.

3.4.4 Campos de aplicación del Procesamiento de Imágenes

- Análisis de imágenes digitales.
- Toma de datos remota.
- Identificación o conteo de objetos.
- Mapeo de un área.
- Reconocer y medir el tamaño, la forma y la posición de determinados objetos.

CAPÍTULO 4

HARDWARE Y SOFTWARE

4.1 Hardware

Hace referencia a todas las partes físicas y tangibles de una computadora; son las partes del computador que se pueden ver y sentir.

4.1.1 Sensor Kinect

4.1.1.1 Definición

Es un dispositivo que gracias a sus componentes permite al usuario obtener las imágenes del entorno sin necesidad del contacto físico.

4.1.1.2 Componentes del Sensor Kinect 1.0

El sensor *Kinect* incluye los siguientes componentes:

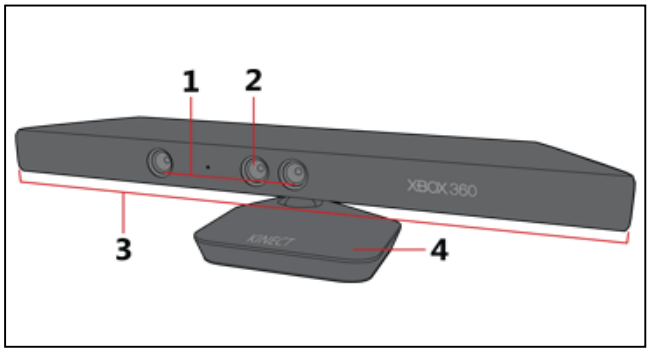

Figura 13. Partes que componen un Kinect

Fuente: *(Componentes del sensor Kinect para Xbox 360, s.f.)*

1. Sensores de profundidad

   Está compuesto por un sensor emisor infrarrojo y un receptor CMOS. (Geek, 2010)

   Los sensores tridimensionales son capaces de hacer un seguimiento de un "objeto" dentro de un área.

2. Cámara RGB

   Una cámara RGB (rojo, azul, verde) ayuda a identificar, captar imágenes o videos.

3. Micrófonos

   Se encarga del reconocimiento de voz.

4. Inclinación motorizada

   Inclina al sensor de manera automática hacia arriba o hacia abajo según sea necesario.

4.1.1.3 Usos para Kinect

El sensor *Kinect* es un dispositivo, inicialmente pensado simplemente como una herramienta de juego, sin embargo y gracias a sus componentes se puede hacer uso del mismo para programar una serie de aplicativos.

El Sensor *Kinect* 1.0, cuenta con una cámara con una resolución VGA capaz de trabajar a 1280x1024 px; sus sensores cuentan con lentes de color y sensación de profundidad, ajustes de sensor con motor de inclinación, campo de visión: horizontal: 57°, vertical: 43°, rango de inclinación física: ± 27°, rango de profundidad del sensor: 1,2 - 3,5 m; identifica hasta 6 personas en una escena.

4.1.2 Raspberry Pi 3

4.1.2.1 Definición

Es un computador en una placa reducida.

> La *Raspberry Pi 3* cuenta con un procesador ARM Cortex A53, un procesador de cuatro núcleos a 1.2GHz de 64 bits y que, según sus datos, tiene un rendimiento 10 veces superior al de la Raspberry Pi original y un 50% más que la Raspberry Pi 2, el modelo anterior. (Moya, 2016)

4.1.2.2 Componentes de la Raspberry

Los componentes principales de una Raspberry son:

- CPU: Quad-Core Cortex A7 a 900MHz.
- GPU: VideoCore IV de doble núcleo.
- RAM: 1GB DDR2.

Puertos:

- 4 USB.
- 1 x 40 GPIO pin.
- 1 X HDMI 1.4.
- 1 x Ethernet.
- 1 x Combo audio/mic.
- 1 x Interfaz de cámara (CSI).
- 1 X Interfaz de Pantalla (DSI).
- 1 x Micro SD.
- 1 x Núcleo Grafico 3D.
- Módulo Bluetooth.
- Módulo de Wi-Fi b/g/n en la banda de 2.4GHz.

4.1.2.3 Accesorios:

- Tarjeta de memoria SD
  - Definición

Una tarjeta de memoria SD es un soporte de almacenamiento externo, que permite guardar información o datos; se usa en diferentes dispositivos como cámaras, celulares, reproductores de música, consolas de videojuegos, entre otros.

  - Características básicas

En una tarjeta de memoria SD se puede diferenciar las siguientes características:

Figura 14. Descripción de una Tarjeta de memoria SD

1. Formato de Capacidad

Hasta ahora podemos encontrar: normal, HC y XC.

- SD -> 16MB, 32 MB, 64MB, 128MB, 256MB, 512MB, 1GB, 2GB, 4GB, 8GB, 16GB, 32GB. No soportan el BUS Ultra Alta Velocidad (UHS siglas en inglés).
- SDHC-> 4GB, 8GB, 16GB, 32GB. Soportan BUS UHS.
- SDXC-> 64GB, 128GB, 200GB, 512GB, 2TB. Soportan BUS UHS.

2. Tipo de BUS UHS

- UHS-I: 104MB/s Velocidad máxima de lectura/escritura.
- UHS-II: 312MB/s Velocidad máxima de lectura/escritura.

3. Clase de BUS UHS

- U1: 10MB/s Velocidad mínima de escritura.
- U3: 30MB/s Velocidad mínima de escritura.

4. Clase

- Clase 2: 2MB/s
- Clase 4: 4MB/s
- Clase 6: 6MB/s
- Clase 10: 10MB/s

5. Capacidad de almacenamiento

Es la cantidad de memoria que puede almacenar esa tarjeta. Para el presente Proyecto se usó la tarjeta de memoria SD de 32 Gb.

4.2 Software

Hace referencia a todos los programas informáticos que hacen posible la ejecución de tareas dentro del computador.

4.2.1 GNU/Linux

4.2.1.1 Definición

Es un sistema operativo que permite la interacción del usuario con el computador y la ejecución de otros programas.

4.2.2 Ubuntu

4.2.2.1 Definición

Ubuntu es una palabra africana que significa 'Humanidad hacia otros', es un sistema operativo basado en GNU/Linux, que contiene todas las aplicaciones necesarias con un enfoque fácil de usar con una licencia libre. (Merino, 2010)

4.2.3 Raspbian

4.2.3.1 Definición

Raspbian es un sistema operativo libre y gratuito basado en Debian y optimizado para el *hardware* de la Raspberry Pi. Es el conjunto de programas básicos y utilitarios (programas de soporte), que permiten que la Raspberry haga algo útil. Sin embargo, Raspbian es algo más que un sistema operativo, pues viene con unos 35 mil paquetes, pre-compilados, de forma tal que sea fácil instalar el que necesitemos en la Raspberry Pi. (Michelone, 2012)

4.2.4 Python

4.2.4.1 Definición

Python es un lenguaje de *scripting* independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones *Windows* a servidores de red o incluso, páginas *web*. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad. (Alvarez, 2003)

4.2.4.2 Principales librerías

Para la ejecución del presente proyecto se utilizaron las siguientes librerías:

- PIL
- freenect
- cv2
- numpy
- math

4.2.4.3 Características del lenguaje *Python*

- Propósito general

Se pueden crear todo tipo de programas. No es un lenguaje creado específicamente para la *web*.

- Multiplataforma

Hay versiones disponibles de *Python* en muchos sistemas informáticos distintos. Originalmente se desarrolló para Unix, aunque cualquier sistema es compatible con el lenguaje siempre y cuando exista un intérprete programado para él.

- Interpretado

Quiere decir que no se debe compilar el código antes de su ejecución. En realidad, sí que se realiza una compilación, pero esta se realiza de manera transparente para el programador. En ciertos casos, cuando se ejecuta por primera vez un código, se producen unos "códigos de byte" que se guardan en el sistema y que sirven para acelerar la compilación implícita que realiza el intérprete cada vez que se ejecuta el mismo código.

- Interactivo

Python dispone de un intérprete por línea de comandos en el que se pueden introducir sentencias. Cada sentencia se ejecuta y produce un resultado visible, que puede ayudarnos a entender mejor el lenguaje y probar los resultados de la ejecución de porciones de código rápidamente.

- Orientado a Objetos

La programación orientada a objetos está soportada en *Python* y ofrece en muchos casos una manera sencilla de crear programas con componentes reutilizables.

- Funciones y librerías

Dispone de muchas funciones incorporadas en el propio lenguaje, para el tratamiento de *strings*, números, archivos, etc. Además, existen muchas librerías que podemos importar en los programas para tratar temas específicos como la programación de ventanas o sistemas en red o cosas tan interesantes como crear archivos comprimidos en *.zip*.

- Sintaxis clara

Por último, destacar que *Python* tiene una sintaxis muy visual, gracias a una notación identada (con márgenes) de obligado cumplimiento. En muchos lenguajes, para separar porciones de código, se utilizan elementos como las llaves o las palabras clave *begin* y *end*. Para separar las porciones de código en *Python* se debe tabular hacia dentro, colocando un margen al código que iría dentro de una función o un bucle. Esto ayuda a que todos los programadores adopten unas mismas notaciones y que los programas de cualquier persona tengan un aspecto muy similar.

(Alvarez, 2003)

4.2.5 OpenKinect

4.2.5.1 Definición

Open*Kinect* es una comunidad abierta de personas interesadas dar uso a la cámara *Kinect* de Xbox con ordenadores y otros dispositivos. Trabaja con bibliotecas libres, de código abierto, que permite a la *Kinect* ser utilizada con *Windows*, Linux y Mac.

La comunidad Open*Kinect* consta de más de 2000 miembros que aportan su tiempo y conocimientos al proyecto. Los miembros se han unido a este proyecto con la misión de crear el mejor conjunto posible de aplicaciones para la *Kinect*.

Su objetivo principal actualmente es el *software libfreenect*. Todo código contribuido a *OpenKinect* se pondrá bajo disposición Apache 2.0 o licencias GPL2 opcionales.

(OpenKinect, 2012)

4.2.5.2 Contenido de Open Kinect

- Project Roadmap - La hoja de ruta actual para el proyecto (*libfreenect*, librerías de análisis y aplicaciones).
- Project Policies - El nombre oficial del proyecto, licencia, política de contribución, coordinación de los desarrolladores y toma de decisiones.
- Installation - Cómo descargar, compilar e instalar en Linux, OS X y *Windows*.
- Contributing Code - Repositorios oficiales, uso, firmados, comunicaciones y evaluación.
- FAQ - Preguntas más frecuentes.
- Project Ideas - Ideas y conceptos para explorar el uso de *OpenKinect*.
- Gallery and websites - Videos y enlaces a cosas que la gente está haciendo con *OpenKinect*.

(OpenKinect, 2012)

4.2.6 OpenCV

OpenCV es una librería de visión por computador de código abierto, disponible en http://sourceforge.net/projects/opencvlibrary/

La librería está escrita en los lenguajes C y C++ y es compatible con Linux, *Windows* y Mac OS X. Cuenta con un desarrollo activo en interfaces para *Python*, Ruby, Matlab y otros lenguajes.

OpenCV ha sido diseñado para ser eficiente en cuanto a gasto de recursos computacionales y con un enfoque hacia las aplicaciones de tiempo real.

OpenCV está escrito y optimizado en C y puede tomar ventaja de los procesadores con múltiples núcleos.

(Bazaga, 2015)

4.2.6.1 Módulos de OpenCV

OpenCV tiene una estructura modular. Los módulos principales de OpenCV son:

- core:

  Este es el módulo básico de OpenCV. Incluye las estructuras de datos básicas y las funciones básicas de procesamiento de imágenes.

- highhui:

  Este módulo provee interfaz de usuario, códigos de imagen y vídeo y capacidad para capturar imágenes y vídeo, además de otras capacidades como la de capturar eventos del ratón, etc.

- imgproc:

  Este módulo incluye algoritmos básicos de procesado de imágenes, incluyendo filtrado de imágenes, transformado de imágenes, etc.

- video:

  Este módulo de análisis de vídeo incluye algoritmos de seguimiento de objetos.

- objdetect:

  Incluye algoritmos de detección y reconocimiento de objetos estándar.

(Bazaga, 2015)

4.2.6.2 Aplicación de Open CV

La librería tiene más de 2500 algoritmos, que incluye algoritmos de *machine learning* y de visión artificial para usar.

Estos algoritmos permiten identificar objetos, caras, clasificar acciones humanas en vídeo, hacer tracking de movimientos de objetos, extraer modelos 3D, encontrar imágenes similares, eliminar ojos rojos, seguir el movimiento de los ojos, reconocer escenarios, etc.

Se usa en aplicaciones como la detección de intrusos en vídeos, monitorización de equipamientos, ayuda a navegación de robots, inspeccionar etiquetas en productos, entre otros.

(Gracia, 2013)

CAPÍTULO 5

DISEÑO DEL ALGORITMO

5.1 Dependencias necesarias para el desarrollo del algoritmo

5.1.1 Instalación de la librería Libfreenect

Para el correcto funcionamiento del sensor *Kinect*, se debe tomar en cuenta que es un *hardware* de Microsoft, por lo tanto la mayor parte de librerías son para el sistema operativo *Windows*, pero grupos de investigación como *OpenKinect* ha elaborado una serie de librerías para que se pueda utilizar el sensor *Kinect* en diferentes sistemas operativos como en este caso se usa para Linux (Ubuntu y Raspbian).

Dicho grupos de investigación, en su página *web*, explica paso a paso como se procede con la instalación de las librerías, así como también se mencionan los paquetes que deben estar instalados o actualizados que son los siguientes:

- Libusb: proporciona un acceso a los dispositivos USB.
- CMake: herramienta para construir, probar y empaquetar *software*.
- *Python*: entorno de programación.

Para actualizar todo lo necesario se debe abrir un Terminal (consola) y ejecutar el comando:

```
1. sudo apt-get upgradesudo apt-get install git-core cmake
   freeglut3-dev pkg-config build-essential libxmu-dev libxi-dev
   libusb-1.0-0-dev
```

Luego de esto, se debe actualizar al sistema con los comandos:

```
1. sudo apt-get update
2. sudo apt-get upgrade
```

Una vez hecho esto se procede con la instalación de *libfreenect*; se detallan los pasos a continuación:

Se clona la carpeta que contiene los instaladores de las librerías en el directorio deseado, es decir en el terminal, se debe estar ubicado en la dirección donde se quiere guardar la carpeta clon.

```
1. git clone git://github.com/OpenKinect/libfreenect.git
```

Con este comando se ingresa a la carpeta *libfreenect*.

```
1. cd libfreenect
```

Se crea una carpeta con el nombre build.

```
1. mkdir build
```

Se ingresa en la carpeta build.

```
1. cd build
```

Se ejecuta este comando para construir los paquetes necesarios, -L es para listar todas las opciones.

```
1. cmake -L ..
```

Genera todas las dependencias necesarias que van a ser instaladas.

```
1. make
```

Comando para instalar *libfreenect* en el sistema.

```
1. sudo make install
```

Para crear el vínculo a la librería en esa ubicación.

```
1. sudo ldconfig /usr/local/lib64/
```

Comandos para añadir el usuario con el que se está trabajando para que pueda acceder a las funciones del *Kinect*, sirve para omitir el comando *sudo* cada vez que se ejecuta algún programa con la librería *libfreenect*.

```
1. sudo adduser $USER video
2. sudo adduser $USER plugdev
```

Abre una hoja de tipo gedit (bloc de notas).

```
1. sudo nano /etc/udev/rules.d/51-Kinect.rules
```

En el gedit se debe agregar el siguiente código:

```
1. # ATTR{product}=="Xbox NUI Motor"
2. SUBSYSTEM=="usb", ATTR{idVendor}=="045e",
   ATTR{idProduct}=="02b0", MODE="0666"
3. # ATTR{product}=="Xbox NUI Audio"
4. SUBSYSTEM=="usb", ATTR{idVendor}=="045e",
   ATTR{idProduct}=="02ad", MODE="0666"
5. # ATTR{product}=="Xbox NUI Camera"
6. SUBSYSTEM=="usb", ATTR{idVendor}=="045e",
   ATTR{idProduct}=="02ae", MODE="0666"
7. # ATTR{product}=="Xbox NUI Motor"
8. SUBSYSTEM=="usb", ATTR{idVendor}=="045e",
   ATTR{idProduct}=="02c2", MODE="0666"
9. # ATTR{product}=="Xbox NUI Motor"
10. SUBSYSTEM=="usb", ATTR{idVendor}=="045e",
    ATTR{idProduct}=="02be", MODE="0666"
11. # ATTR{product}=="Xbox NUI Motor"
12. SUBSYSTEM=="usb", ATTR{idVendor}=="045e",
    ATTR{idProduct}=="02bf", MODE="0666"
```

Todas las líneas descritas sirven para poder establecer un tipo de reglas para el funcionamiento de todas las herramientas que posee el *Kinect*.

Después de todos los pasos descritos ya se puede ejecutar un programa que viene entre los ejemplos de la librería *libfreenct*, estos ejemplos están programados en lenguaje C++ y se pueden ejecutar desde el terminal, en este caso se ha seleccionado el ejemplo *freenect-glview* que muestra una imagen donde presenta la información que obtiene la cámara RGB (imagen a color) y los sensores de profundidad (imagen en escala de grises).

```
1. freenect-glview
```

Al ejecutar este último comando, se presentará una ventana similar a la siguiente:

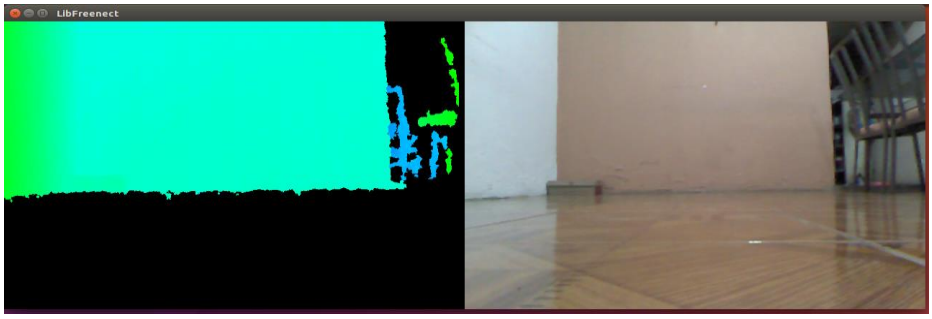

Figura 15. Ejecución de glview

En la parte izquierda se aprecia la imagen de profundidad y en la derecha la imagen RGB ambas proporcionadas por el sensor *Kinect*.

Una vez se ha probado el ejemplo, se procede a instalar complementos de *Python*, para que al momento de programar no presente errores en la ejecución de los programas, se debe instalar lo siguiente:

```
1. sudo apt-get install cython
2. sudo apt-get install Python-dev
3. sudo apt-get install Python-numpy
```

(Naman, 2014)

5.1.2 Instalación de la librería OpenCV

La instalación de OpenCV es similar a la instalación de *libfreenect*, se debe clonar la carpeta, crear un directorio para construir los paquetes necesarios y después instalar a continuación los comandos:

```
1. git clone https://github.com/opencv/opencv.git
2. cd ~/opencv
3. mkdir release
4. cd release
5. cmake
6. make
7. sudo make install
```

Una vez que se ha ejecutado todos los comandos, ya se puede empezar a programar en *Python* la aplicación que se desea.

Nota: los comandos son válidos hasta la versión de Ubuntu 14.04 LTS, y para Raspbian Jessie. (Ferdous, 2015)

5.2 Descripción general del sistema

El diseño del algoritmo tiene como objetivo el mapeo del área por la cual se traslada el robot, reconociendo los obstáculos presentes en dicho recorrido e identificando los que sean potencialmente peligrosos, para lo cual emplea el sensor *Kinect* que está montado sobre el robot (Ver Figura 16), los objetos detectados son plasmados en un mapa en 2D.

Figura 16. Kinect montado en el robot

El sensor *Kinect* obtiene imágenes de profundidad del plano vertical, que pasan a ser procesadas por la tarjeta *Raspberry Pi 3*, en este procesamiento se busca si existen o no objetos en el campo de visión, en el momento en que se ha detectado algo, el algoritmo pide a la tarjeta que se comunique vía WiFi con una cámara IP, la misma que fue instalada en el techo del área de pruebas, para que esta le envíe una imagen vista desde arriba (plano horizontal) de toda el área, con ello se procede a identificar la posición actual del robot en el ambiente controlado y posteriormente ubicar al objeto que fue detectado, una vez se obtiene la ubicación del objeto en cuestión se procede a graficarlo en el mapa que se está creando.

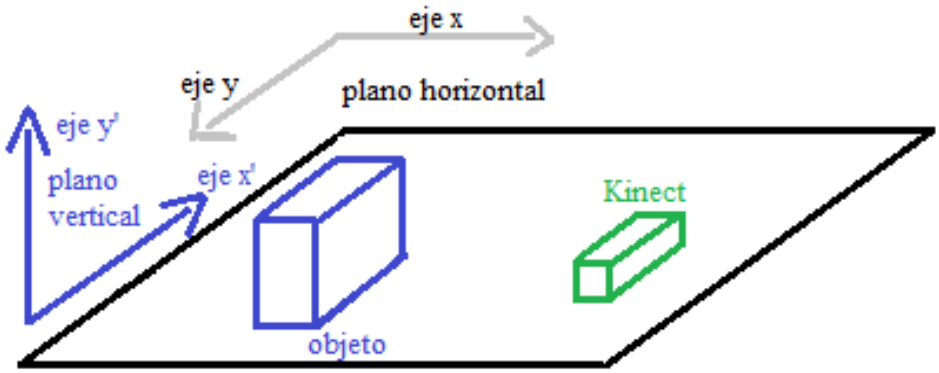

Figura 17. Representación de los planos

Para la ubicación del robot en el espacio se consideraron dos opciones, la primera usar una función de tiempo en la tarjeta “arduino due” (que es la tarjeta que maneja los motores del robot), para así obtener el tiempo en el que los motores se encuentran

en movimiento, realizar un cálculo y obtener la posición aproximada del robot, y la segunda opción ubicar una cámara en el techo para que mediante un procesamiento digital de imágenes obtenga la posición y dirección del robot, para esta investigación se usó la segunda ya que el trabajo se enfoca en el procesamiento digital de imágenes.

Para la ubicación del robot en el ambiente controlado se utilizó una cámara IP que como ya se mencionó fue instalada en el techo del área de pruebas apuntando al suelo (ver Figura 18), la misma que obtiene una imagen total del ambiente controlado. (Ver Figura 19)

Figura 18. Cámara IP en el techo

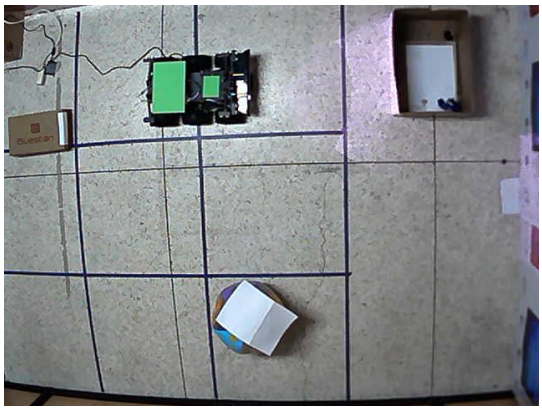

Figura 19. Imagen del ambiente controlado visto desde la cámara IP

Para que la cámara IP pueda detectar al robot, se colocó dos rectángulos de color verde sobre el chasis, uno pequeño de 15x10cm y otro grande del doble de tamaño 30x20cm, esto con el fin de poder diferenciar cual está en la punta del robot y cual está detrás, como se puede ver en la Figura 20.

Figura 20. Robot con las placas de color verde

Como se mencionó anteriormente el sensor *Kinect* obtiene la imagen del plano vertical, por lo tanto a los objetos detectados se los debe pasar al plano horizontal que es el suelo por donde transita el robot. Una vez ubicado el lugar del objeto encontrado, en el plano horizontal, se procede a bosquejarlo en una imagen (mapa en 2D) y se continúa con el recorrido del robot, la imagen que se menciona es una representación del plano horizontal.

Se debe mencionar que para realizar el cambio de coordenadas entre planos vertical y horizontal se ha utilizado coordenadas rectangulares para recorrer el punto en el eje "x" y "y" y en el caso de proyectar el punto (se tiene modulo y ángulo) se usó coordenadas polares para encontrar coordenadas rectangulares con el mismo fin de encontrar la distancia que se debe recorrer en cada eje.

5.3 Algoritmo de detección del sensor Kinect

Se debe mencionar que en este punto solo se identifica el objeto, en el apartado 5.4 se detalla cómo se lo ubica en el espacio.

Para comprender el funcionamiento del algoritmo se ha usado las cinco etapas del PDI, las mismas que se describirán en el siguiente diagrama realizando una comparación con el algoritmo elaborado:

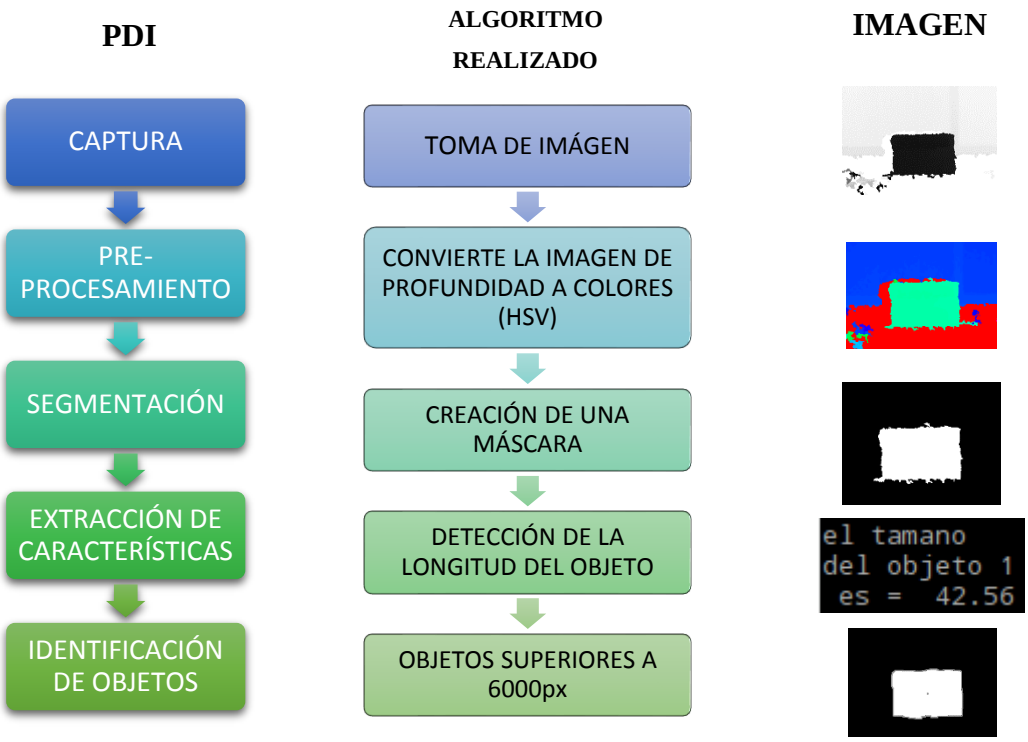

Figura 21. PDI, algoritmo elaborado, imagen

5.3.1 Captura

Primero se crean las funciones de adquisición de datos get_video ( ) toma la información de la cámara RGB del sensor *Kinect*, y get_depth ( ) obtiene la información del sensor de profundidad. (Naman, 2014)

```
1. #function to get RGB image from Kinect
2. def get_video():
3.     array,_ = freenect.sync_get_video()
4.     array = cv2.cvtColor(array,cv2.COLOR_RGB2BGR)
5.        return array
6.
7. #function to get depth image from Kinect
8. def get_depth():
9.     array,_ = freenect.sync_get_depth()
10.          array = array.astype(np.uint16)
11.             array1=gradiente_color_depth (array, 0, 1200, cv2.
   COLORMAP_HSV)
12.             array1=array1[0:479,0:629]
13.             return array1
```

El *Kinect* toma la captura de la profundidad en tiempo real (plano vertical), cuando el algoritmo llama a la función respectiva (get_depth), la imagen obtenida se guarda

en el directorio donde se encuentra el algoritmo, y luego se la abre para guardarla en una matriz (variable imagen) para ser tratada de manera digital.

```
#Captura de una profundidad
    depth1 = get_depth()
    cv2.imwrite('my3.png',depth1)

    #Deteccion del color de profundidad
    imagen = cv2.imread('my3.png')
```

5.3.2 Pre procesamiento

Se convierte la imagen de profundidad, la cual es presentada en escala de grises, a colores, se puede elegir varias gamas de colores, para este caso se usó la escala: matiz, saturación y brillo (HSV siglas en inglés). (EcuRed, Modelo HSV, 2012)

Además en esta etapa se elimina los 10 últimos px de la imagen capturada, esto se debe a que el sensor *Kinect* toma la imagen de 640x480 px, pero los últimos pixeles, es decir entre el 630 y 640, son nulos por ello se procedió a eliminarlos.

```
#funcion que aplica un gradiente de color a la matriz depth para poder diferenciar los valores
def gradiente_color_depth(depth, min, max, mapa_color):
    #trunca los valores al intervalo de min y max, donde lo demas se vuelve 0
    d1 = depth * np.logical_and(depth >= min, depth <= max)

    #genera los valores de 0 a 256 para el intervalo definido
    #concatenate, concatena datos
    d2 = np.concatenate([
        #inicializa con el valor de ceros que tenga min
            np.zeros(min, dtype=np.uint16),
            #linspace, distribuye los valores desde 0 a 255 en el intervalo
            np.linspace(0, 255, max - min).astype(np.uint16),
            np.ones(65535-max, dtype=np.uint16)*255
        ])

        #convierte los valores a uint8
        d3 = d2[d1].astype(np.uint8)

        #aplica un mapa de color a la imagen en gris
        #0  COLORMAP_AUTUMN
        #1  COLORMAP_BONE
        #2  COLORMAP_JET
        #3  COLORMAP_WINTER
        #4  COLORMAP_RAINBOW
        #5  COLORMAP_OCEAN
        #6  COLORMAP_SUMMER
        #7  COLORMAP_SPRING
        #8  COLORMAP_COOL
```

```
29.         #9  COLORMAP_HSV
30.         #10 COLORMAP_PINK
31.         #11 COLORMAP_HOT
32.
33.         d4=cv2.applyColorMap(d3, mapa_color)
34.
35.         return d4
```

El proceso de convertir la profundidad de escala de grises a colores es para poder trabajarla como imagen y utilizar las funciones de OpenCV.

En la Figura 22 se presenta enmarcado en color amarillo los 10 px que son nulos.

Figura 22. Imagen de profundidad con la pérdida de 10px

5.3.3 Segmentación

Se crea una máscara filtrando un rango de color que se desea en función a la distancia de detección (entre 0,49 y 3,50 m); para este caso la distancia establecida para la detección fue de 0,60m, con una tolerancia de ±0,04m, esto se debe a que el receptor de infrarrojos del sensor *Kinect* está ubicado en la mitad, por lo tanto la distancia perpendicular es ligeramente menor a la distancia a los extremos, entonces se tomó dicho margen para detectar los objetos a lo largo de toda la imagen. (Gl4r3, Detección de colores con OpenCV y Python, 2014)

Para establecer la distancia de detección se tomó en cuenta dos aspectos, el primero es que el robot tiene sensores ultrasónicos que detectan obstáculos a 30cm de distancia aproximadamente por lo que se decidió utilizar una mayor distancia para realizar la

detección y el segundo aspecto es que el área de trabajo es relativamente pequeña por lo que no resulta factible tomar una distancia de detección elevada.

```
#Rango de colores detectados:
#Azules:
azul_claro = np.array([180,255,3], dtype=np.uint8)
azul_oscuro = np.array([125, 250, 0], dtype=np.uint8)
mascara_azul = cv2.inRange(imagen, azul_oscuro, azul_claro)
```

A continuación se explica cómo se obtuvo el valor de tolerancia. En la Figura 23 se presenta el ángulo de visión del sensor *Kinect* (57°) en donde la variable "x" representa el espacio en metros desde la mitad de la visión del sensor hasta un extremo, que puede captar el *Kinect* a una distancia perpendicular de 0,6m; se procede a calcular la distancia lineal (d) para obtener la variación respecto a la distancia perpendicular (0,6m).

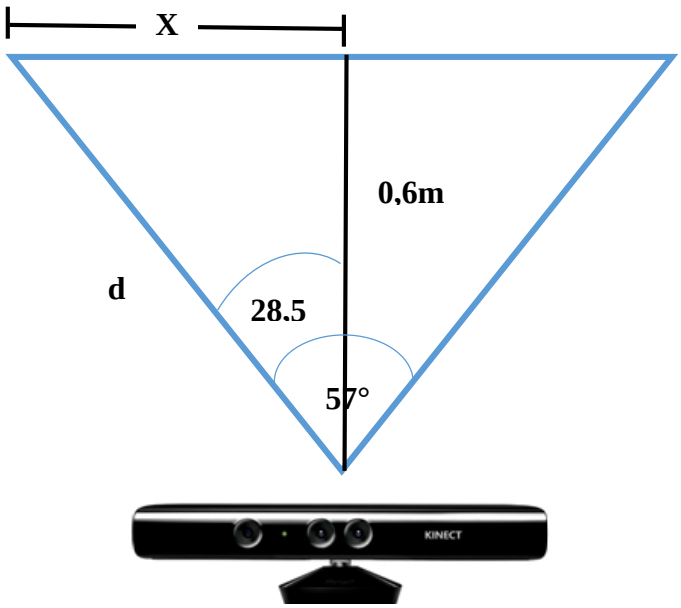

Figura 23. Ángulo de visión del sensor Kinect

$$\tan(28{,}5°) = \frac{x}{0{,}6}$$

$$x = 0{,}326\ m$$

$$d = \sqrt{0{,}6^2 + 0{,}326^2}\ m$$

$$d = 0{,}68m$$

Como resultado se obtuvo una distancia de 0,68m que vendría a ser el punto extremo máximo que capta el sensor por lo cual como se mencionó a la distancia de 0,6m se le agrego un ±0,04m de tolerancia.

Se realizó una tabla con la información RGB de cada imagen, de un experimento con 350 mediciones, se colocó el sensor *Kinect* frente a una pared y se efectuó la toma de imágenes desde 0m hasta 3,5m a pasos de 0,01m, para obtener la variación de color a medida que la distancia aumenta; para la distancia establecida (0,60m) el rango de color fue: margen superior [B,G,R]=[180,255,3] y margen inferior [B,G,R]=[125,250,0]. En el Anexo A se muestra la tabla que contiene la información RGB de cada una de las 350 imágenes realizadas.

Figura 24. Ejemplo de la toma de mediciones

5.3.4 Extracción

Se tomó la máscara generada en la segmentación y se le aplicó un filtro *close/open* en el cual se usa una matriz auxiliar de blancos, para este trabajo fue seleccionada una matriz de 15x15, dicha matriz sirve para descartar el ruido en la imagen tanto dentro como fuera del objeto detectado, tomando como área de comparación la dimensión seleccionada, el filtro *close* desecha las marcas negras (ruido) sobre las áreas blancas de la máscara, y por el contrario el filtro *open* elimina el ruido (marcas blancas) sobre las áreas negras. (Gl4r3, Cómo filtrar el ruido de una máscara con OpenCV, 2015)

```
1. #Filtrar el ruido con un CLOSE/OPEN
2. kernel = np.ones((15,15),np.uint8) #filtro de area 15x15
   pixeles areas inferiores a 15x15 se eliminan
3. mascara_azul = cv2.morphologyEx(mascara_azul, cv2.MORPH_CLOSE,
   kernel)
4. mascara_azul = cv2.morphologyEx(mascara_azul, cv2.MORPH_OPEN,
   kernel)
```

A continuación, con la ayuda de las funciones de OpenCV, se aplica un *GaussianBlur* (desenfoque gaussiano) que sirve para suavizar los contornos para que la imagen no tenga picos muy pronunciados ya que esto dificulta el trazado del contorno que es el siguiente paso.

```
1. #Difuminar la mascara para suavizar los contornos
2. blur = cv2.GaussianBlur(mascara_azul, (5, 5), 0)
```

> El método de desenfoque gaussiano IIR [Respuesta de Impulso Infinita, siglas en inglés] actúa sobre cada píxel de la capa activa o selección, estableciendo su valor como el promedio de todos los valores de los píxeles incluidos en un radio definido en el diálogo. Un valor alto producirá un mayor efecto de desenfoque. (AntI, 2010)

El siguiente paso es aplicar una búsqueda de contornos que en las librerías de OpenCV lleva el nombre de *findContours*, esto sirve para delimitar el contorno de todas las áreas detectadas en la máscara.

Luego se aplica la función de OpenCV *contourArea* que sirve para obtener el área en pixeles de los contornos obtenidos en el paso anterior.

(Gl4r3, Detección de colores con OpenCV y Python, 2014)

```
1. #Detectar y guardar los contornos. Tambien sus areas.
2. _, contours, _= cv2.findContours(blur,cv2.RETR_TREE,cv2.CHAIN_
   APPROX_SIMPLE)
3. areas = [cv2.contourArea(c) for c in contours]
```

Las áreas obtenidas se guardan en un arreglo (variable áreas) ya que depende del número de objetos que sean detectados, cada uno tendrá su propia área.

A continuación se realiza un ciclo de repetición (bucle) que en cada vuelta va a tomar un valor de área encontrado, en el orden de cómo se fue guardando en la variable áreas, que va desde la primera posición hasta la última del arreglo. Dentro de este ciclo

se incluye una condición, que según la experimentación de Gl4r3 (Detección de colores con OpenCV y Python, 2014) el área debe ser superior a 6000px, esto se realiza por cuanto en la toma de la imagen de profundidad a veces se presenta ruido por lo cual las áreas que son menores a 6000px no se toman en cuenta. Una vez se cumpla la condición se procede a calcular el número de vértices que tiene la figura detectada utilizando la función propia de OpenCV *approxPolyDP*, el valor obtenido se guarda en un arreglo ya que será necesario para saber si el objeto es potencialmente peligroso o no, se detalla el uso de este arreglo en el apartado 5.3.5.

Ahora para encontrar la longitud del objeto, dentro del bucle primero se procede a calcular los puntos extremos tanto el izquierdo como el derecho de cada una de las formas contenidas en la máscara y esta información se envía a la función tamaño, la cual realiza el cálculo de la longitud. (Rahman, 2013)

Este valor de longitud se guarda en un arreglo ya que de igual manera se va a utilizar todos los valores de longitud encontrados en la generación del mapa (que se explica en el apartado 5.5), dentro del bucle solo se necesita el valor de longitud actual para establecer una condición de comparación en la cual si el valor de la longitud es diferente de cero entonces se presenta en pantalla el valor de la longitud del objeto y se llama a la función "captura", que es la función de adquisición de datos de la cámara IP que devuelve la posición actual del robot, es decir las coordenadas "x" y "y" así como el ángulo en el que está apuntando el robot (esta función se detalla en el apartado 5.4).

Finalmente en este bucle se traza el contorno de las formas encontradas con la función propia de OpenCV *drawContours*. (Gl4r3, Detección de colores con OpenCV y Python, 2014)

```
1. for extension in areas:
2.    if extension > 6000:
3.       actual = contours[i]
4.             #Aproximar el numero de vertices
5.             approx = cv2.approxPolyDP(actual,0.001*cv2.arcLeng
   th(actual,True),True)
6.             ap[k] = len(approx)
7.       #Obtener los puntos maximos izquierdo y derecho
8.       leftmost = tuple(actual[actual[:,:,0].argmin()][0])
9.       rightmost = tuple(actual[actual[:,:,0].argmax()][0])
10.            longitud = tamano(leftmost, rightmost,1)
```

```
11.             longg[k] = longitud
12.             if (longg[k] != 0):
13.                print 'la longitud aproximada del objeto',k+1,'
   es = ', longg[k]
14.                xx, yy, alpha = captura()
15.                xk,yk = puntoxy(xx,yy,alpha) #proyeccion del
   punto a 60 cm de distancia
16.             else:
17.                print 'ningun objeto en el area de deteccion'
18.
19.                   cv2.drawContours(imagen,[actual],0,(0,0,0),2
   )
20.             cv2.drawContours(mascara_azul,[actual],0,(150),2)
21.             k=k+1
22.
23.          else:
24.             print 'objeto muy pequeno'
25.          i = i+1
```

En la función tamaño mencionada anteriormente, llegan como datos las coordenadas (x, y) de los puntos extremos tanto el izquierdo como el derecho y una variable auxiliar “a”, para obtener la longitud del objeto solo es necesario restar las coordenadas en x de ambos puntos ya que el mapa es en 2D, por lo tanto no se necesita obtener la altura de los objetos, entonces solo se utiliza la posición cero de ambos arreglos, que es la posición que contiene el valor de x, el resultado es la distancia en px que existe entre ambos puntos (variable dist1), para obtener la distancia en unidades de longitud (para este caso cm) se debe hacer una regla de tres.

La variable auxiliar “a” es la que permite reutilizar la función, ya que más adelante se necesita hacer el cálculo de otra distancia entre puntos, pero el valor que se debe obtener debe ser entero sin decimales, por ello se utiliza a la variable auxiliar para que le ayude a decidir a la función cual distancia debe retornar.

```
1. #funcion para obtener la longitud del objeto
2. def tamano(p_izq, p_der,a):
3.     if (a==1):
4.    dist1= int(p_der[0] - p_izq[0])
5.    dist2= round(((dist1*(0.532*0.6*2))/6.3),2)  #el valor es
   630 pero para que resulte en cm le dejamos con 6.3
6.     elif(a==2):
7.         dist1= p_izq - p_der
8.         dist2= int((dist1*(0.532*0.6*2))/6.3)
9.     return dist2
```

Se debe realizar una operación de transformación ya que los puntos que se obtuvieron son en escala de 630x480, por lo tanto se debe transformar a unidades de longitud, por lo que se necesita realizar lo siguiente:

Como la imagen fue recortada en 10px, se debe calcular el ángulo de visión del sensor descontando los nulos, partiendo de que el ángulo de visión del sensor *Kinect* es de 57° entonces la regla de 3 queda de la siguiente manera:

$$57° \rightarrow 640px$$
$$x° \rightarrow 630px$$
$$x = \frac{630 * 57°}{640}$$
$$x = 56{,}11°$$

El nuevo ángulo de visión entonces es 56,11°, entonces para este valor angular se debe calcular la longitud máxima en metros que puede observar a una distancia de 0,6m del sensor, la variable "x" representa la longitud desde el centro a uno de los extremos, se utilizó el siguiente triángulo:

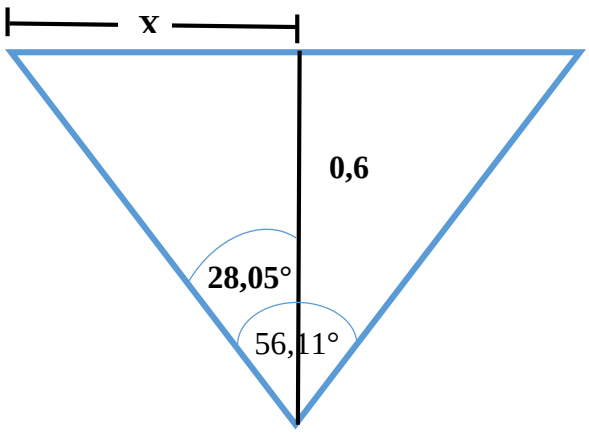

Figura 25. Triangulo para calcular la longitud de visión del Kinect

Para encontrar el valor de "x" se utiliza la función tangente:

$$\tan(28{,}05°) = \frac{x}{0{,}6}$$
$$x = 0{,}32$$

Y a este valor se debe multiplicar por 2 ya que solo se tomó uno de los extremos:

$$x_{TOTAL} = 0{,}32 * 2$$

$$x_{TOTAL} = 0{,}64\ m$$

Entonces en los 630px se tiene 0,64 m y como ya se ha calculado la longitud del objeto en cuestión (variable dist1) entonces se efectúa la siguiente regla de tres:

$$0{,}64\ m \rightarrow 630px$$

$$x \rightarrow dist1$$

$$x = \frac{dist1 * 0{,}64}{630} * 100$$

$$x = \frac{dist1 * 0{,}64}{6{,}3}$$

Donde x viene a ser la variable dist2 en metros, se multiplica por 100 para que el resultado obtenido sea en centímetros, la variable dist2 es la que contiene la longitud final del objeto, por lo tanto es la variable que se retorna.

5.3.5 Identificación de objetos

Como se mencionó anteriormente, el algoritmo puede identificar objetos potencialmente peligrosos, y para ello se ocupa el valor del número de vértices que tiene cada figura, obtenido de la función *approxPolyDP*, que se obtuvo en el bucle de la etapa de extracción (apartado 5.3.4), en base a varios experimentos realizados se llegó al número límite de 80, es decir que un objeto normal, que para este proyecto es una caja, se aproxima a tener entre 40 y 75 vértices, esto se debe a que la imagen de la máscara no resulta un rectángulo perfecto, por otro lado un objeto circular (una pelota) sobrepasa este número variando entre 85 y 110 vértices.

Entonces se realiza una comparación con el número de vértices encontrados, y se asigna un valor a una variable auxiliar que posteriormente ayudará a graficar el objeto en el mapa.

```
1. if(ap[m]>=80):
2.     aux3=2
3. else:
4.     aux3=1
```

5.4 Función de la cámara IP

En este apartado se obtiene la posición del robot sobre el plano horizontal que es el piso del ambiente controlado así como también se obtiene el ángulo en el que está apuntando el sensor *Kinect*, esta función es la que se denomina "captura" que fue mencionada en el punto 5.3.4.

Para que la cámara IP funcione y se conecte con la tarjeta *Raspberry Pi 3*, es necesario incluir el siguiente código:

```
1. class FoscamCamera(object):
2.     UP = 0
3.     STOP_UP = 1
4.     DOWN = 2
5.     STOP_DOWN = 3
6.     LEFT = 4
7.     STOP_LEFT = 5
8.     RIGHT = 6
9.     STOP_RIGHT = 7
10.
11.         def __init__(self, url='192.168.1.10', user='admin',
   pwd='foscam'):  # clase incializar camara IP
12.             super(FoscamCamera, self).__init__()
13.             self._user = user
14.             self._pwd = pwd
15.             self._url = url
16.             self._isPlaying = 0
17.
18.         def isPlaying(self):
19.             return self._isPlaying
20.
21.         def setIsPlaying(self, val):
22.             self._isPlaying = val
23.
24.         def setURL(self, url):
25.             self._url = url
26.
27.         def url(self):
28.             return self._url
29.
30.         def setUser(self, usr):
31.             self._user = usr
32.
33.         def user(self):
34.             return self._user
35.
36.         def setPassword(self, pwd):
37.             self._pwd = pwd
38.
39.         def password(self):
40.             return self._pwd
41.
42.         def setUserAndPassword(self, user, password):
```

```
43.                 self.setUser(user)
44.                 self.setPassword(password)
45.
46.             def move(self, direction):
47.                 cmd = {'command': direction}
48.                 f = self.sendCommand('decoder_control.cgi', cmd)
49.
50.             def snapshot(self):
51.                 f = self.sendCommand('snapshot.cgi', {})
52.                 return f.read()
53.
54.             def sendCommand(self, cgi, parameterDict):
55.                 url = 'http://%s/%s?user=%s&pwd=%s' % (self.url(
    ),
56.                                                        cgi,
57.                                                        self.user
    (),
58.                                                        self.pass
    word())
59.                 for param in parameterDict:
60.                     url = url
    + '&%s=%s' % (param, parameterDict[param])
61.
62.                 return urllib.urlopen(url)
```

Este código permite inicializar a la cámara IP para poder utilizar todas las funciones que dispone. (Ramsey, 2013)

Cabe mencionar que en este código de inicialización se debe llenar la dirección IP el nombre de usuario y la contraseña que posee la cámara, para este trabajo la dirección IP es: **192.168.1.10**, el nombre de usuario: admin y la contraseña es: foscam, los dos últimos vienen por defecto en la cámara.

Como se indicó con anterioridad, la cámara está ubicada en el techo del ambiente controlado y está apuntando al suelo, por lo tanto la cámara IP obtiene la imagen de todo lo que está dentro del área de pruebas (plano horizontal), para la ubicación del robot se realizó un proceso similar al descrito en el apartado 5.3.

Para iniciar, la cámara IP toma una imagen del área de pruebas. (Ver Figura 26)

Figura 26. Imagen del área de pruebas desde la cámara IP

La imagen obtenida (Figura 26), es de 640x480px, esta imagen se envía a la *Raspberry* vía WiFi para que realice el procesamiento digital de la misma, el cual inicia por obtener una máscara de esta imagen, separando el color verde, como se mencionó anteriormente el robot tiene dos placas de color verde en su parte superior, el segundo paso es obtener las coordenadas del centroide de ambas placas de color verde y marcarlas en la máscara el resultado es el siguiente:

Figura 27. Máscara de la imagen de la cámara IP

Antes de realizar el cálculo del ángulo es necesario mencionar el sistema de coordenadas aplicado, donde se tiene un eje de referencia y un eje trasladado:

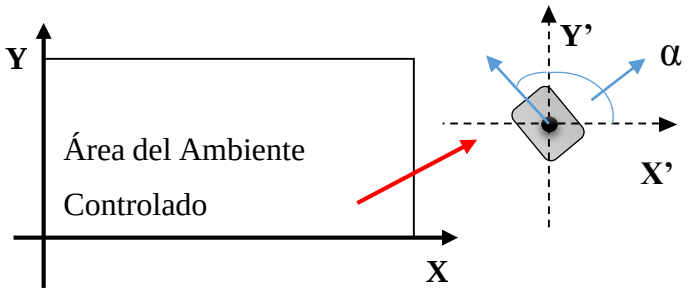

Figura 28. Eje de referencia y eje trasladado

En la Figura 28 se puede observar que el eje de referencia es estático y paralelo a los bordes del área del ambiente controlado mientras que, el eje trasladado tiene su centro en el centroide de la placa verde de mayor tamaño, como se mencionó anteriormente la placa verde de mayor tamaño se ubica detrás de la pequeña. El ángulo α es hacia donde está apuntando el *Kinect*

Para calcular el ángulo que existe entre estos dos puntos se procede a realizar lo siguiente:

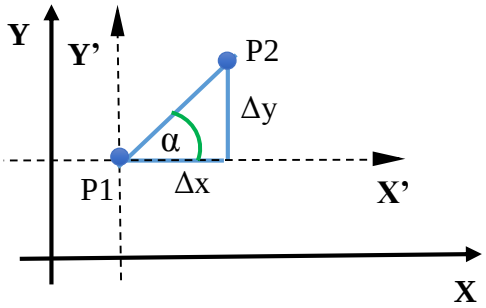

Figura 29. Ejemplo para el cálculo del ángulo entre los centros de las placas verdes

Como se puede observar en la Figura 29, el punto P1 es el centroide de la placa verde de mayor tamaño y el punto P2 es el centroide de la placa verde de menor tamaño. Para obtener el valor de la variación en el eje x (Δx) se necesita realizar una resta entre la coordenada "x" de P2 y la "x" de P1, de igual manera para obtener la variación en el eje y (Δy) se realiza una resta de la coordenada "y" de P2 y la "y" de P1.

Una vez obtenidos ambos valores de variación, por medio de las funciones trigonométricas se calcula el ángulo α.

$$\tan(\alpha) = \frac{\Delta y}{\Delta x}$$

$$\alpha = \arctan(\frac{\Delta y}{\Delta x})$$

Luego de esto se procede a cambiar las coordenadas de los centroides de $px\ a\ cm$, como se mencionó al inicio de este apartado, la imagen que obtiene la cámara IP es de $640x480px$ y esta imagen observa todo el ambiente controlado que mide $273x190cm$ entonces se aplica una regla de tres:

$$273\ cm \rightarrow 640px$$

$$x \rightarrow x2$$

En esta regla de tres x es la coordenada en "x" en centímetros, x2 es el valor de la coordenada en "x" en px del centroide en cuestión, por lo tanto se procede a calcular su valor en centímetros.

$$x = \frac{x2 * 273}{640}$$

Del mismo modo se realiza para la coordenada en "y" del centroide:

$$190\ cm \rightarrow 480px$$

$$y \rightarrow y2$$

De igual manera en esta regla de tres y es la coordenada en "y" en centímetros, y2 es el valor de la coordenada en "y" en px del centroide en cuestión, entonces:

$$y = \frac{y2 * 190}{480}$$

Finalmente se debe recordar que se requiere ubicar el objeto detectado en el apartado 5.3, en el plano horizontal para poder trazarlo en el mapa, como para este punto ya se dispone de las coordenadas del robot que es donde está montado el *Kinect* y estas coordenadas ya están sobre el plano horizontal ahora se debe obtener las coordenadas donde se encuentra el objeto que en un principio son detectadas en el plano vertical, para lo cual se debe proyectar el punto donde se encuentra el robot la distancia de detección que para este caso es 60cm.

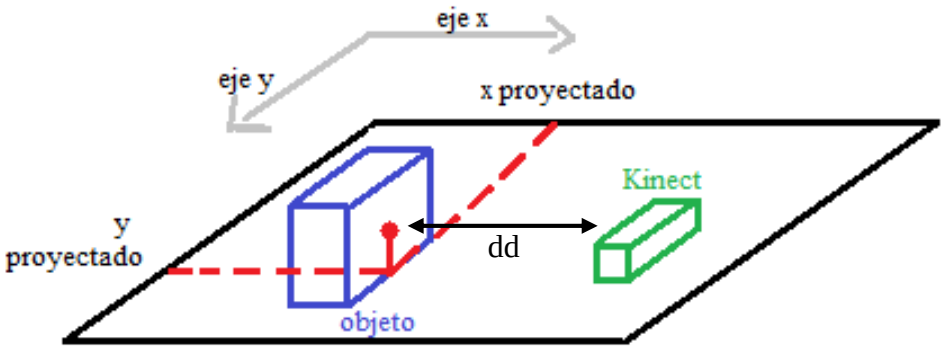

Figura 30. Punto proyectado la distancia de detección

En la Figura 30 se muestra el ejemplo de un objeto detectado así como la posición donde se encuentra el sensor *Kinect* montado en el robot, la distancia de detección (dd en la Figura 30) es lo que se debe recorrer las coordenadas del *Kinect* para llegar al objeto detectado entonces:

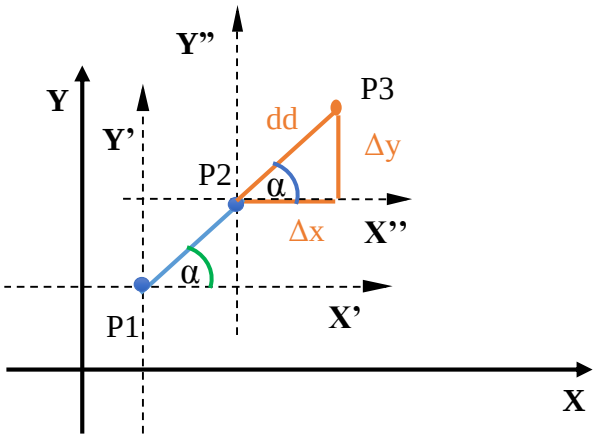

Figura 31. Ejemplo de la proyección del punto del Kinect al objeto

En la Figura 31 se toma la vista del plano horizontal similar a la Figura 29, donde P1 es el centro de la placa verde grande, P2 es el centro de la placa verde pequeña y P3 viene a ser el punto proyectado la distancia de detección, se puede observar también que ahora existe un nuevo eje proyectado ubicado en el centro de la placa verde pequeña (P2) de este punto se realiza la proyección, como se puede observar el ángulo α es el mismo, por lo tanto se tiene coordenadas polares, ya que la distancia de

detección tiene el valor de 76cm (60+16)cm 60cm es la distancia que se fijó desde el principio, y los 16cm es la distancia que existe entre la placa verde pequeña (que ahora es el centro de los ejes proyectados) y el borde del sensor *Kinect*. Se marcó con una flecha blanca la distancia mencionada de 16cm. (Ver Figura 32)

Figura 32. Distancia entre la placa verde pequeña y el borde del Kinect

Retomando el análisis de la Figura 31 para obtener la variación en "x" y "y" se aplican formulas trigonométricas:

$$\Delta x = 76 * \cos(\alpha)$$

$$\Delta y = 76 * \sin(\alpha)$$

Para obtener el punto proyectado se debe sumar los resultados obtenidos al punto P2, con ello se obtiene las coordenadas del punto P3.

$$x3 = x2 + \Delta x$$

$$y3 = y2 + \Delta y$$

Donde x3 y y3 son las coordenadas del punto P3, x2 y y2 son las coordenadas del punto P2.

El punto P3 viene a ser el punto medio de la imagen que captura el *Kinect*.

Finalmente para poder ubicar al objeto detectado por el *Kinect* en el apartado 5.3, se necesita obtener su posición respecto del plano horizontal, como el mapa que se va a generar es en 2D, solo se necesita la ubicación en la coordenada "x". (Ver Figura 33)

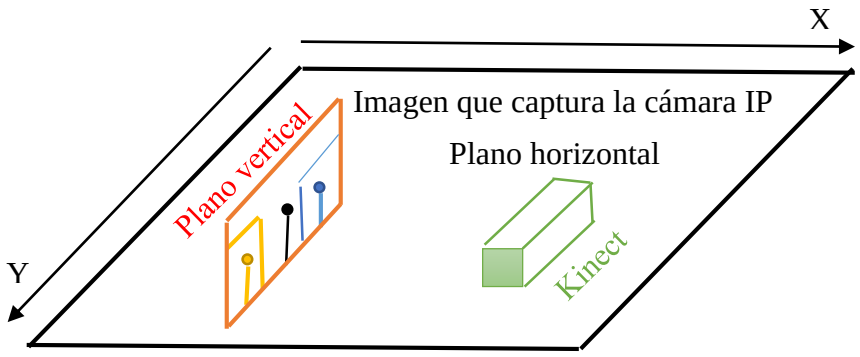

Figura 33. Representación del plano vertical sobre el plano horizontal

Por lo tanto para ubicar los objetos, se necesita tener el centroide de cada objeto detectado y después realizar una resta en la coordenada "x" del centroide de cada uno respecto del punto P3 que ya se obtuvo en los pasos anteriores. Como las figuras detectadas por el sensor *Kinect* se toman de una en una para realizar el análisis, la distancia entre el centro de la imagen (P3) y los centroides (C1 y C2) son calculadas de una en una. Como se mencionó en el apartado 5.3 para este cálculo se vuelve a utilizar la función **tamaño** que es la que obtiene la distancia entre dos puntos.

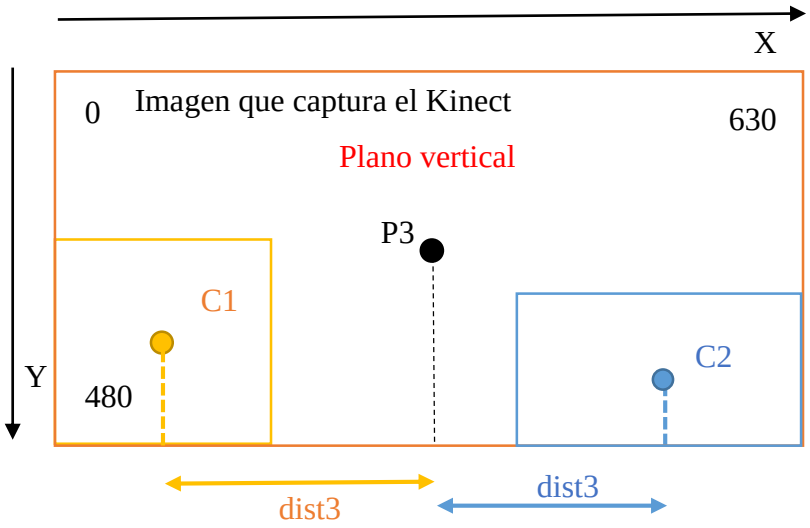

Figura 34. Centroides y punto P3 en el plano vertical

Por último para obtener la ubicación de esos puntos sobre el plano horizontal se utilizó coordenadas polares para obtener las respectivas coordenadas rectangulares, en

este caso se tiene el valor de la distancia (dist3), las coordenadas del punto P3 y el ángulo γ, como se puede observar en la Figura 35.

Para obtener el valor del ángulo γ se hace:

$$\gamma = 90° - \alpha$$

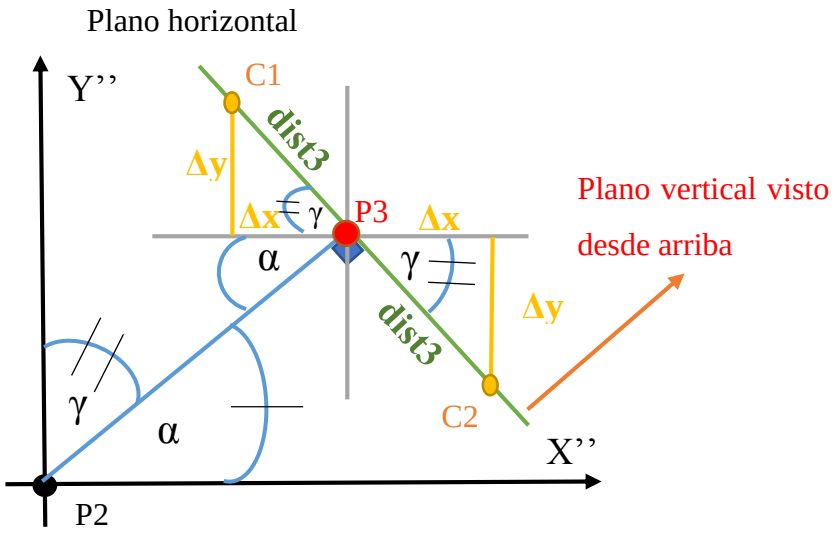

Figura 35. Plano horizontal con la proyección de los centroides del plano vertical

Para obtener los valores de las variaciones en "x" y "y" se realiza:

$$\Delta x = dist3 * \cos(\gamma)$$

$$\Delta y = dist3 * \sin(\gamma)$$

Finalmente esta variación se debe sumar al punto P3 tanto en "x" como en "y".

$$xk = x3 + \Delta x$$

$$yk = y3 + \Delta y$$

Estas coordenadas "xk" y "yk" son los puntos necesarios para trazar el objeto en el mapa, pues estos puntos están en las coordenadas del plano horizontal.

5.5 Función de generar el mapa en 2D

En este apartado, se menciona como ir generando la imagen conforme el robot avanza en el ambiente controlado.

Primero se procede a crear la imagen estática vacía para luego añadir los objetos que se detectan:

Entonces se genera una imagen de las dimensiones del ambiente controlado 273x190px por lo que cada pixel representa 1 centímetro en esta imagen.

Esta imagen se utiliza como una matriz por lo tanto tiene 279 columnas y 190 filas.

```
w, h = 273, 190 #w eje X y h eje Y
imagenF = np.ones((h, w, 3), dtype=np.uint8)*255
imagenF[0:h-1 , 0]=[255, 0, 0]      #borde izquierdo
imagenF[0:h-1 , w-1]=[255, 0, 0]    #borde derecho
imagenF[0, 0:w-1]=[255, 0, 0]       #borde superior
imagenF[h-1, 0:w-1]=[255, 0, 0]     #borde inferior

aux1=int(w/30) #aux1 da el numero de columnas
aux2=int(h/30) #aux2 da el numero de filas
```

A continuación se le agregan bordes superior, inferior, derecho e izquierdo, y se calcula dos valores auxiliares, esto con el fin de trazar una cuadrícula referencial en el mapa para poder verificar la posición de los objetos detectados.

La cuadricula para esta investigación se estableció en cuadrados de 30x30cm cada uno, por lo que los valores auxiliares se obtienen dividendo el número de filas y columnas para 30.

```
for i in range(1,aux1):
    imagenF[0:h-1 , 30*i]=[255,255,0]

for j in range(1,aux2):
    imagenF[30*j , 0:w-1]=[255,255,0]

#cv2.imwrite('map.png',imagenF)
imagenF = cv2.imread('map.png')
```

Se realiza dos ciclos (bucles) uno para las líneas horizontales y otro para las líneas verticales estos ciclos realizan un salto de valor auxiliar en las filas y columnas, que como se mencionó es cada 30cm. (Ver Figura 36)

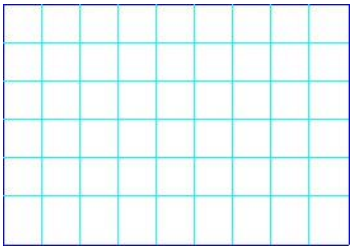

Figura 36. Mapa cuadriculado

Luego de esto se procede a llenar la imagen con los objetos encontrados, para esto, se recibe como datos de entrada el punto centroide y la longitud de la figura detectada, así como también recibe el ángulo en el que está apuntando el sensor *Kinect* y la variable auxiliar que permite detectar si es objeto potencialmente peligroso o no.

Como llega el punto medio (centroide) del objeto detectado, la longitud del mismo debe ser dividida a la mitad ya que habrá que sumar la mitad a cada lado. (Ver Figura 37)

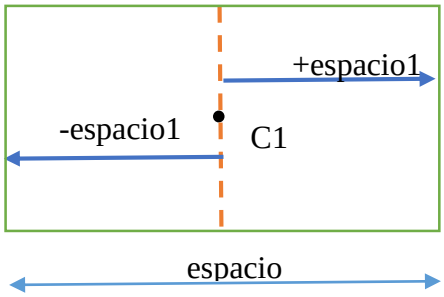

Figura 37. Longitud a sumar desde el centroide del objeto

Como se puede observar en la Figura 37 la variable espacio es la longitud total del objeto y la variable espacio1 es la división de la longitud total para 2.

```
1. def crear(x , y, espacio, angulo, q): #la variable q recibe el
   dato de si es
2.     espacio1=espacio/2           #objeto peligroso o no
3.     if (x<=0):
4.         x=0
5.     if (q==1):
6.             dy1=y-espacio1
7.             dy2=y+espacio1
```

```
8.              imagenF[dy1:dy2, x:x+5]=[0,0,0]
9.              dx1=x-espacio1
10.             dx2=x+espacio1
11.             imagenF[y:y+5, dx1:dx2]=[0,0,0]
12.         elif(q==2):
13.              dy1=y-espacio1
14.              dy2=y+espacio1
15.              imagenF[dy1:dy2, x:x+5]=[0,0,255]
16.              dx1=x-espacio1
17.              dx2=x+espacio1
18.              imagenF[y:y+5, dx1:dx2]=[0,0,255]
19.         cv2.imwrite('map.png',imagenF)
```

Para ir dibujando en la imagen se debe sumar y restar la longitud (espacio1) a las coordenadas del punto que llega como dato (centroide), con esto se obtiene 4 variables nuevas que son las distancias respectivas (dx1, dx2, dy1, dy2) estos valores pasan a ser los limites inicial y final del trazo de la figura del objeto.

La variable auxiliar “q” sirve para diferenciar si el objeto es potencialmente peligroso o no, como se mencionó en el apartado 5.3, esto se realiza comparando el número de vértices que posee cada figura detectada, y dependiendo de esos vértices la variable auxiliar “q” puede tener el valor de 1 o 2, cuando es q=1 se envía la orden de que el grafico sea en color negro [0, 0, 0] y cuando q=2 se envía la orden de que el grafico sea en color rojo [0, 0, 255], ambas escalas están en [B, G, R].

A continuación se presenta el ejemplo de un objeto normal, para esta investigación una caja. (Ver Figura 38 y Figura 39)

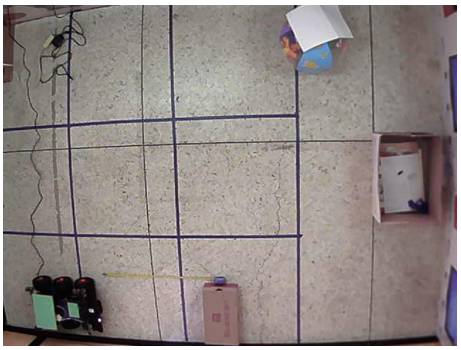

Figura 38. Captura de datos de la cámara IP

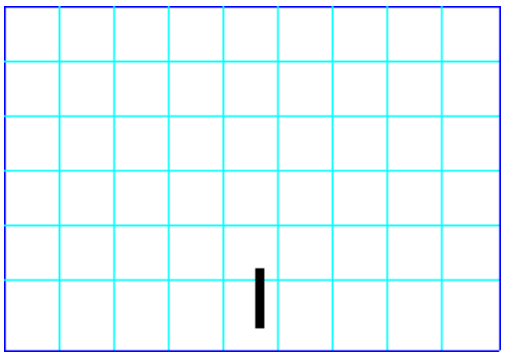

Figura 39. Datos presentados en el mapa

Ahora se presenta un ejemplo de un objeto potencialmente peligroso, para este trabajo es una pelota. (Ver Figura 40 y Figura 41)

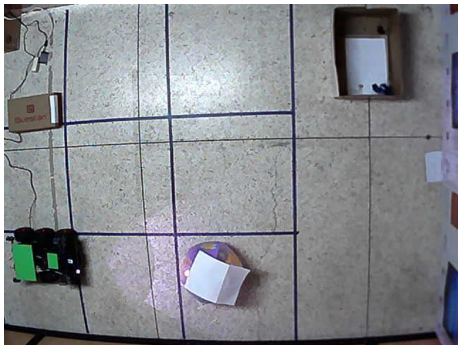

Figura 40. Captura de datos de la cámara IP (peligroso)

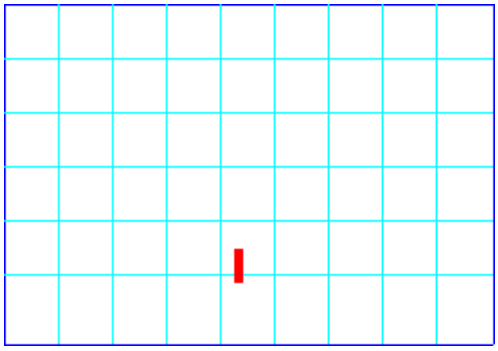

Figura 41. Datos presentados en el mapa (peligroso)

CAPÍTULO 6

PRUEBAS Y RESULTADOS

6.1 Protocolo de pruebas

El área de pruebas consta de 2,73 x 1,90 metros, para este trabajo de investigación se utilizó 3 cajas de dimensiones “caja1” (33 x 18,5 x 11,5) cm “caja2” (43 x 30 x 34) cm y “caja3” (45 x 23 x 12,5) cm para el objeto potencialmente peligroso se utilizó una pelota con diámetro de 40cm.

Para realizar las pruebas del sistema en general, se planteó varios escenarios, los cuales se describen a continuación:

6.1.1 Identificación de un objeto visto desde la esquina

Para este experimento, se coloca la caja1 en el área de pruebas y se ubica al sensor de manera que apunte a la esquina de la caja.

Para una segunda prueba, se mueve el robot y ahora se ubica la caja de forma que la esquina se encuentre de frente al sensor *Kinect*.

6.1.2 Validación del porcentaje para identificar el objeto potencialmente peligroso

Para esta prueba se planteó descubrir desde que porcentaje visible, ya se puede identificar que es un objeto potencialmente peligroso, por lo cual dentro del ambiente controlado se incluyó al objeto potencialmente peligroso parcialmente cubierto por la “caja3” hasta determinar en qué porcentaje de descubierto ya se identifica como objeto peligroso.

6.1.3 Mapeo del entorno con un objeto no peligroso

Para este escenario, se planteó realizar una inspección corta por lo que se utilizó solo un objeto, la “caja1” para diagramar todos los lados posibles.

6.1.4 Mapeo del entorno con tres objetos; uno de ellos es potencialmente peligroso

En este caso se realizaron 3 experimentos, dado a que se tiene 3 objetos, en cada prueba se cambió de lugar los objetos, se utilizó la “caja1” la “caja2” y la pelota.

6.1.5 Mapeo del entorno con tres objetos; un objeto potencialmente peligroso parcialmente cubierto

Para este experimento, se utilizó las 3 cajas y la pelota, siendo la “caja3” la que obstruía la visión directa al objeto potencialmente peligroso (pelota).

6.2 Realización de pruebas y obtención de resultados

Para la experimentación vale recalcar que se necesita iluminación artificial, ya que la cámara IP instalada en el techo presenta errores de lectura de colores cuando existe la presencia de la luz natural.

- **Identificación de un objeto visto desde la esquina.**

El escenario para esta prueba fue el siguiente: (Ver Figura 42)

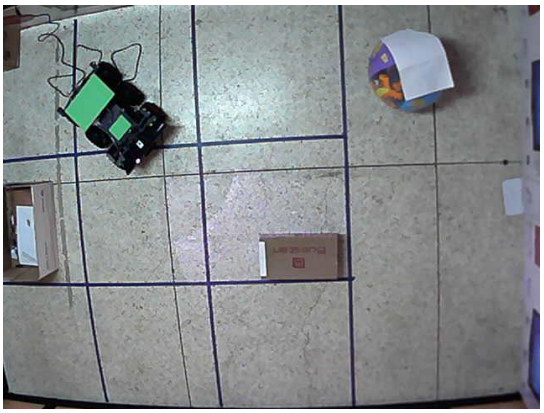

Figura 42. Escenario prueba 1A vista desde la cámara IP

La imagen de profundidad obtenida es la siguiente: (Ver Figura 43)

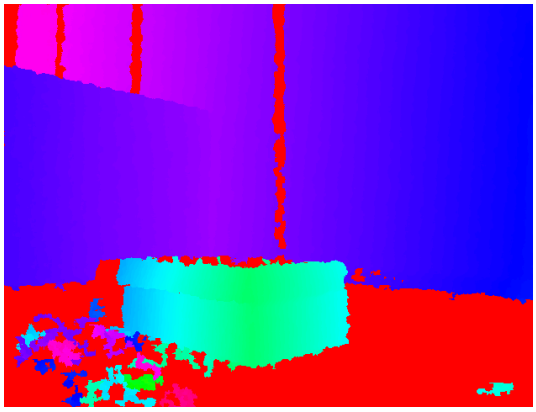

Figura 43. Profundidad prueba 1A

Como se puede observar en la Figura 43, el objeto si es detectado, pero solo una pequeña parte entra en el margen de reconocimiento lo que genera la siguiente máscara.

Figura 44. Máscara prueba 1A

Como se puede observar en la Figura 44 si se capta el objeto, pero no de manera total, el mapa que se generó con este objeto detectado es el siguiente: (Ver Figura 45)

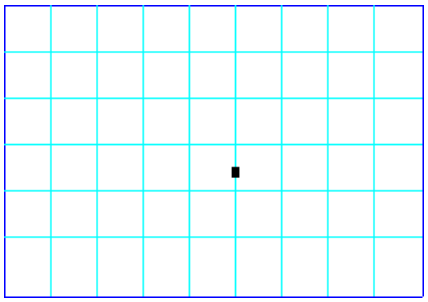

Figura 45. Mapa prueba 1A

En la segunda prueba el escenario es el siguiente: (Ver Figura 46)

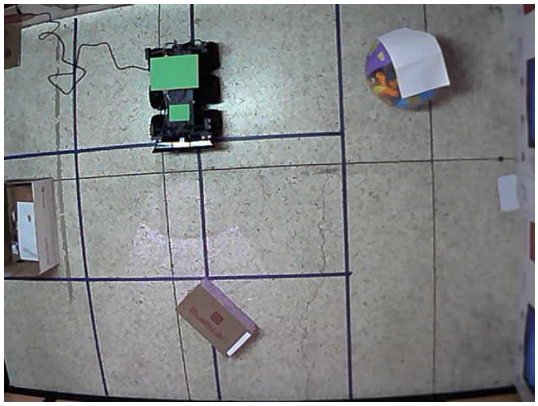

Figura 46. Escenario prueba 1B vista desde la cámara IP

Entonces se obtiene la imagen de profundidad: (Ver Figura 47)

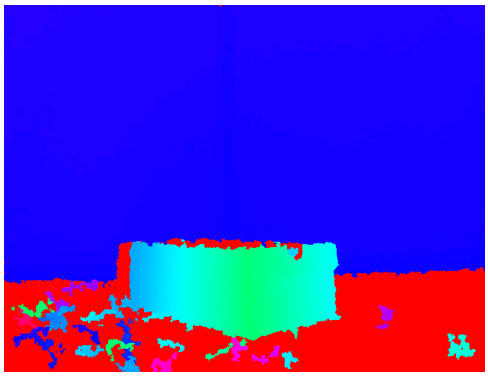

Figura 47. Profundidad prueba 1B

Como se puede observar en la Figura 47, el objeto está dentro del margen de detección lo que genera la siguiente máscara: (Ver Figura 48)

Figura 48. Máscara prueba 1B

Como se puede observar en la máscara, esta vez el objeto ha sido detectado como que fuera plano y no una esquina. Esta máscara generó el siguiente mapa: (Ver Figura 49)

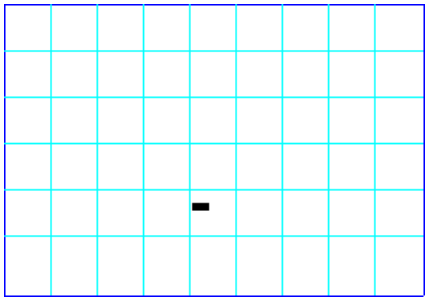

Figura 49. Mapa prueba 1B

- **Validación del porcentaje para identificar el objeto potencialmente peligroso.**

Para esta experimentación como ya se mencionó se utilizó un objeto (caja3) para cubrir parcialmente el objeto potencialmente peligroso, hasta descubrir el porcentaje mínimo para reconocer que es objeto peligroso.

El escenario de la prueba 2A es el siguiente: (Ver Figura 50)

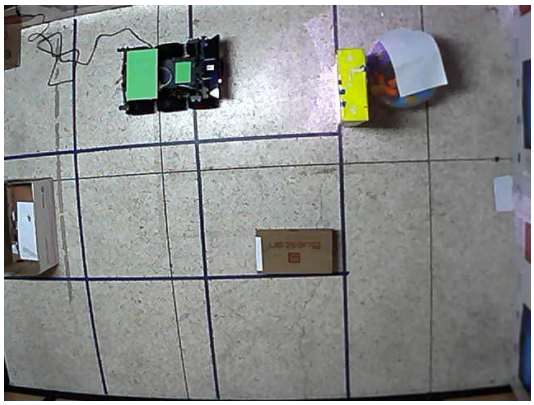

Figura 50. Escenario prueba 2A

La máscara obtenida es: (Ver Figura 51)

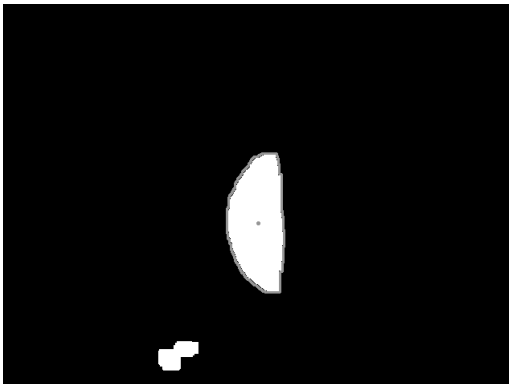

Figura 51. Máscara prueba 2A

Esta máscara genera el siguiente mapa: (Ver Figura 52)

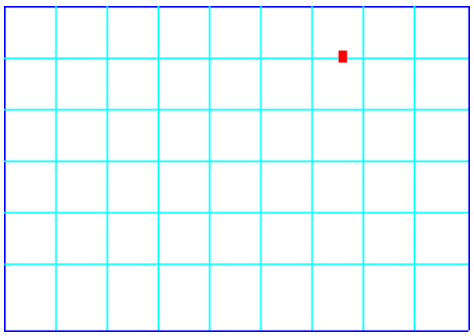

Figura 52. Mapa prueba 2A

En este caso el objeto si fue identificado como potencialmente peligroso.

El escenario 2 es similar al 1, pero se movió 0,5 cm el objeto que obstaculiza. (Ver Figura 53)

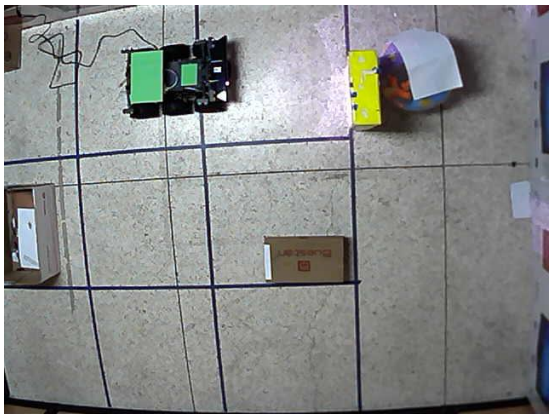

Figura 53. Escenario prueba 2B

Se obtiene la siguiente máscara: (Ver Figura 54)

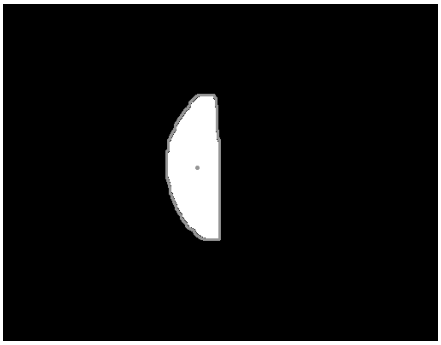

Figura 54. Máscara prueba 2B

Y el mapa presenta lo siguiente: (Ver Figura 55)

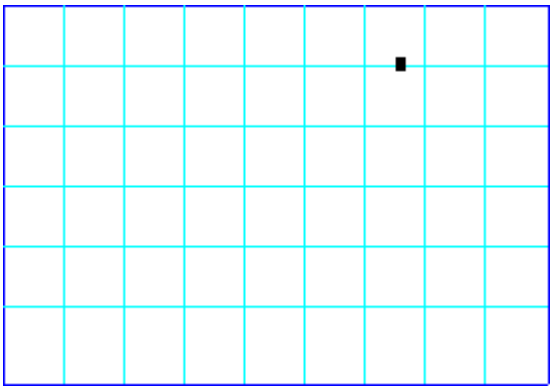

Figura 55. Mapa prueba 2B

Como se puede observar en este caso no detecto como objeto potencialmente peligroso, cuando detecto que si era objeto peligroso el objeto estaba obstruyendo 26cm de la pelota, entonces:

$$40\ cm \rightarrow 100\%$$

$$26\ cm \rightarrow x$$

$$x = \frac{100 * 26}{40} = 65\%$$

El porcentaje obtenido es el porcentaje q estuvo cubierta la pelota, por lo tanto desde que se ve el 35% de la pelota ya le detecta como objeto potencialmente peligroso.

- **Mapeo del entorno con un objeto no peligroso.**

Como ya se mencionó en este experimento se colocó solo la caja1 para realizar el mapa. La generación del mapa por pasos fue el siguiente, en la izquierda se presenta la imagen obtenida por la cámara IP, y en la derecha la generación del mapa:

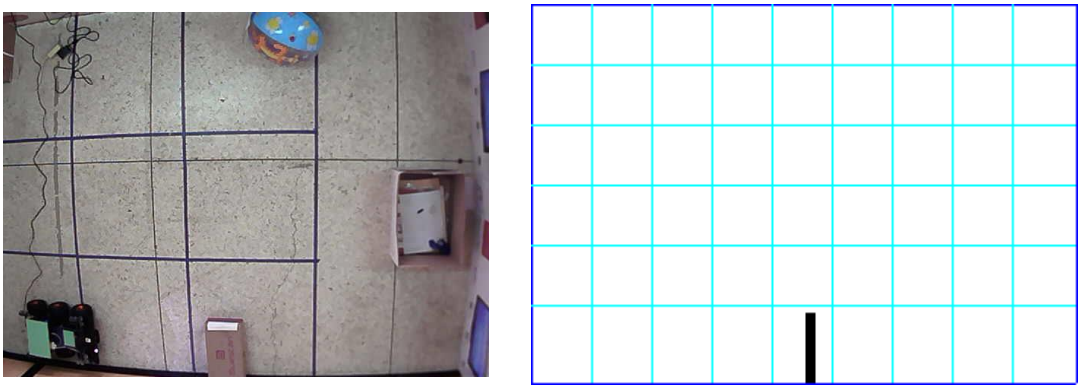

Figura 56. Generación del mapa, prueba 3A

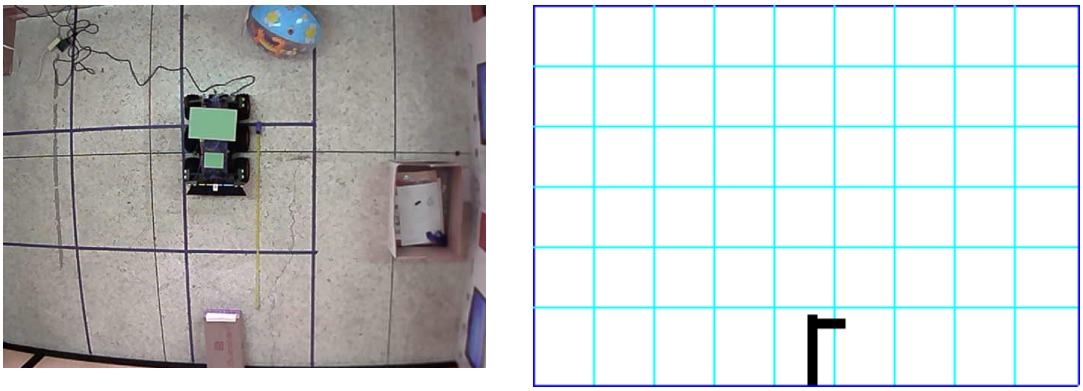

Figura 57. Generación del mapa, prueba 3B

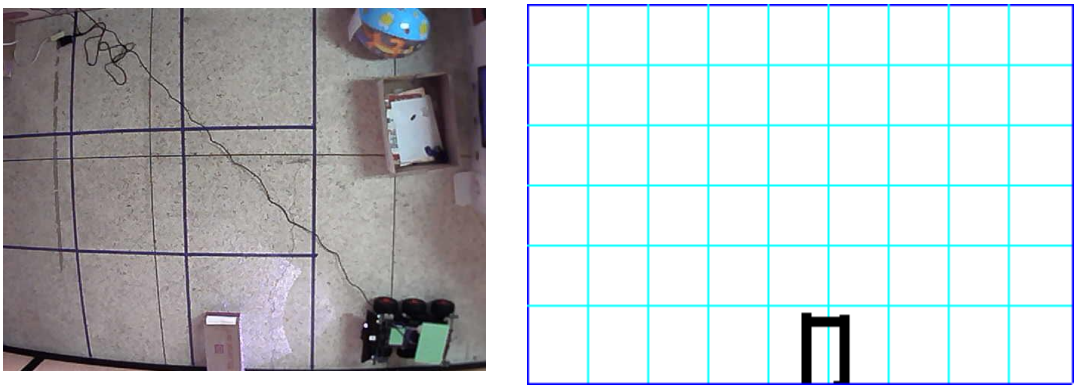

Figura 58. Generación del mapa, prueba 3C

Como se puede observar en la Figura 58, la caja1 se ha plasmado en el mapa 2D, se puede notar que la figura no es un rectángulo perfecto, esto se debe a que existe un error en diferentes lugares del ambiente controlado debido a que la cámara IP que se colocó en el techo tiene lente “ojo de pescado” entonces la imagen que se obtiene del área no es un rectángulo perfecto.

- **Mapeo del entorno con 3 objetos; 1 potencialmente peligroso.**

Para los siguientes experimentos los objetos se movieron de lugar en cada prueba para obtener diferentes mapas, así como también el recorrido inicio desde diferentes puntos. Escenario 1: prueba 4, generación secuencial del mapa:

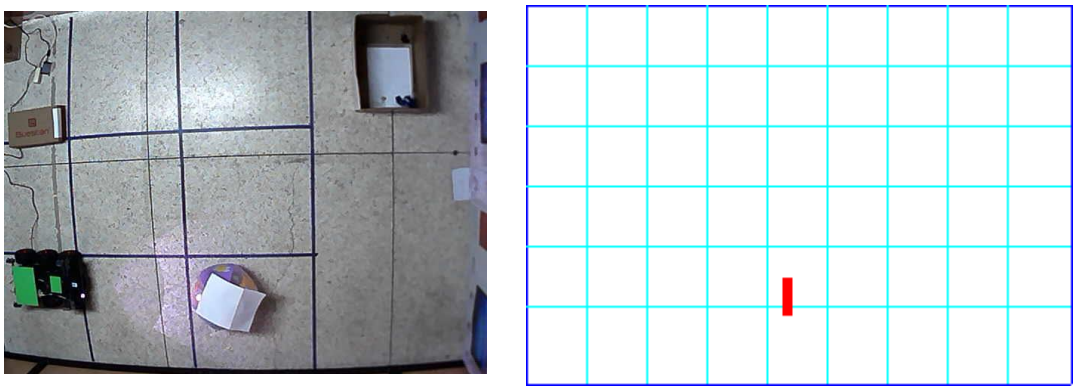

Figura 59. Generación del mapa, prueba 4A

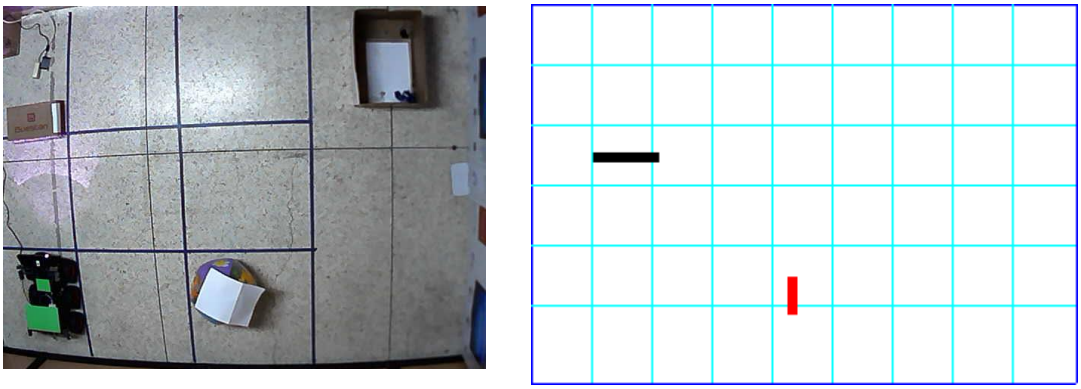

Figura 60. Generación del mapa, prueba 4B

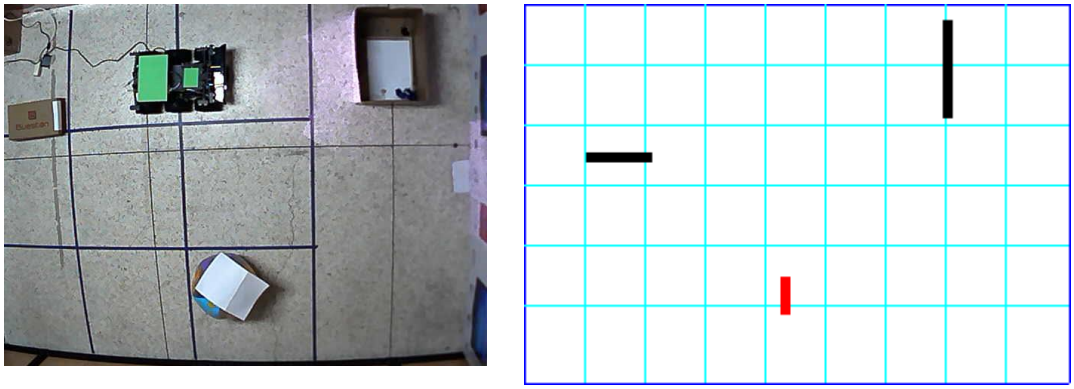

Figura 61. Generación del mapa, prueba 4C

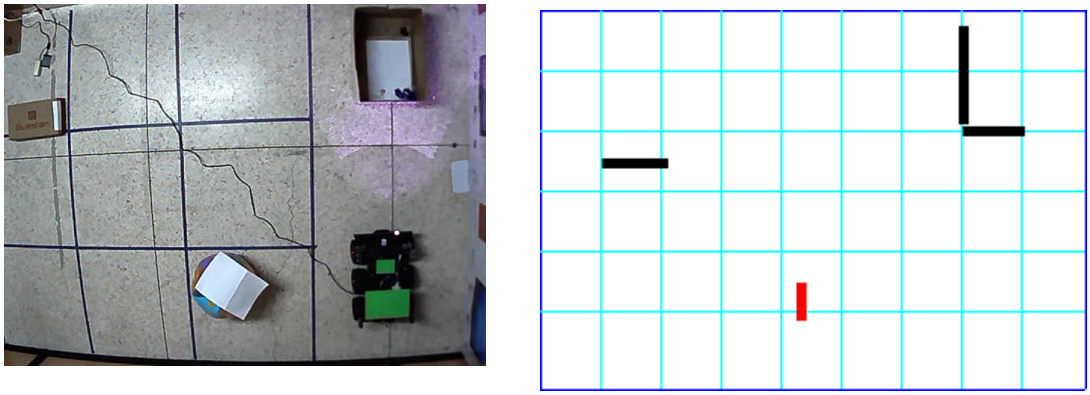

Figura 62. Generación del mapa, prueba 4D

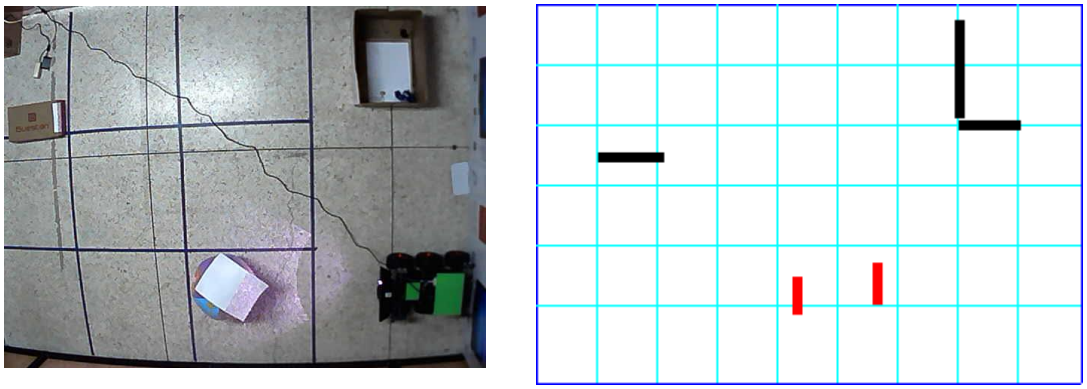

Figura 63. Generación del mapa, prueba 4E

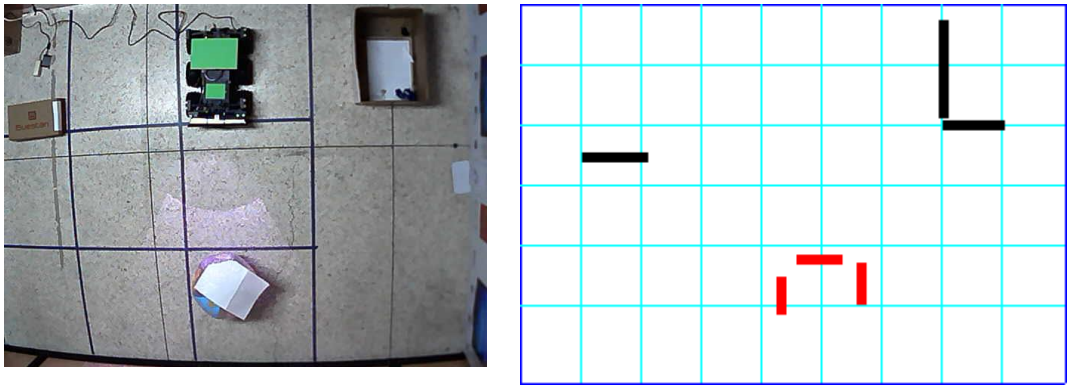

Figura 64. Generación del mapa, prueba 4F

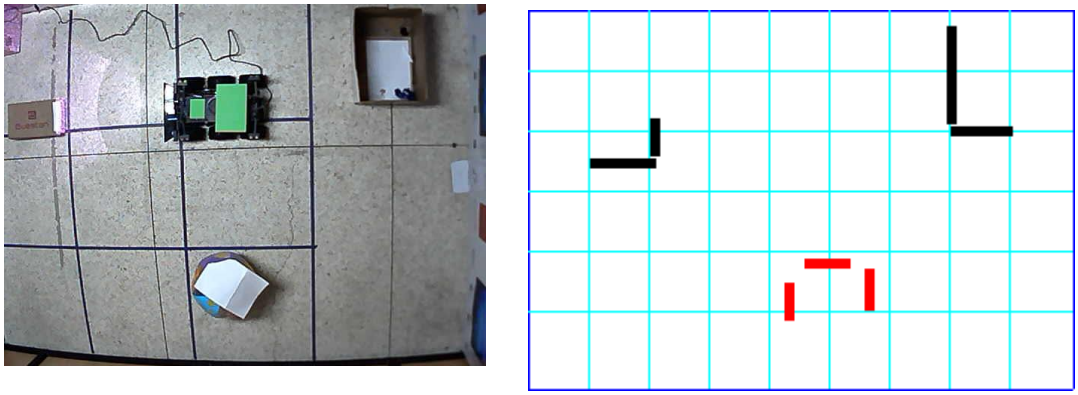

Figura 65. Generación del mapa, prueba 4G

Como se puede observar en la Figura 65, el mapa se obtuvo de manera secuencial, y se puede notar que el objeto potencialmente peligroso no aproxima sus bordes, esto se debe a que el objeto es de forma circular por lo tanto el sensor no logra captarlo a lo largo de todo su diámetro, por lo que no lo puede graficar de forma total.

Escenario 2: prueba 5, se colocó dos cajas relativamente juntas para evaluar la detección de dos objetos al mismo tiempo, generación secuencial:

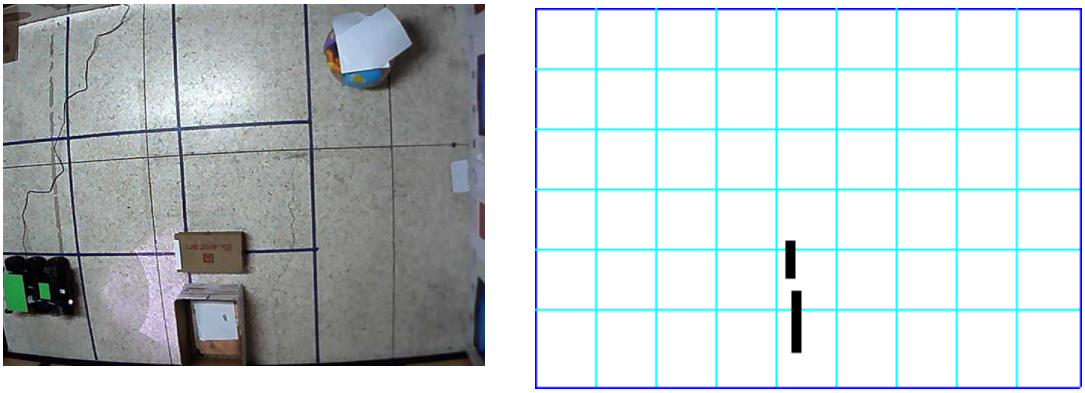

Figura 66. Generación del mapa, prueba 5A

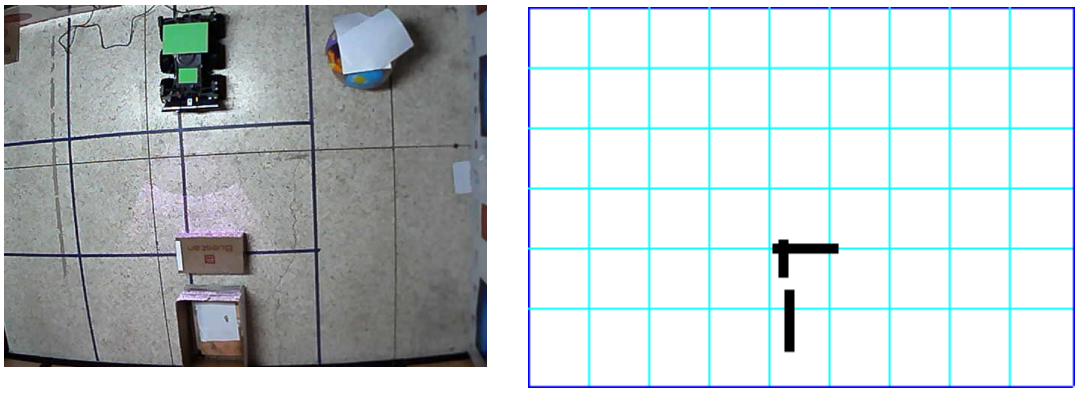

Figura 67. Generación del mapa, prueba 5B

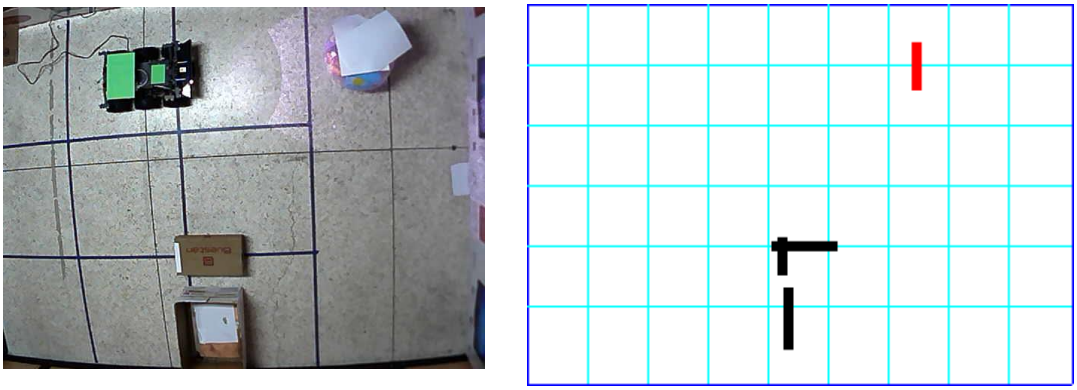

Figura 68. Generación del mapa, prueba 5C

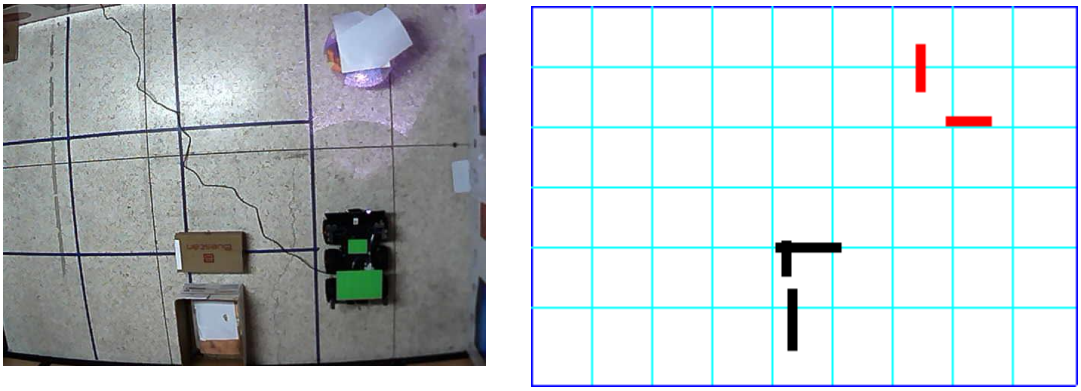

Figura 69. Generación del mapa, prueba 5D

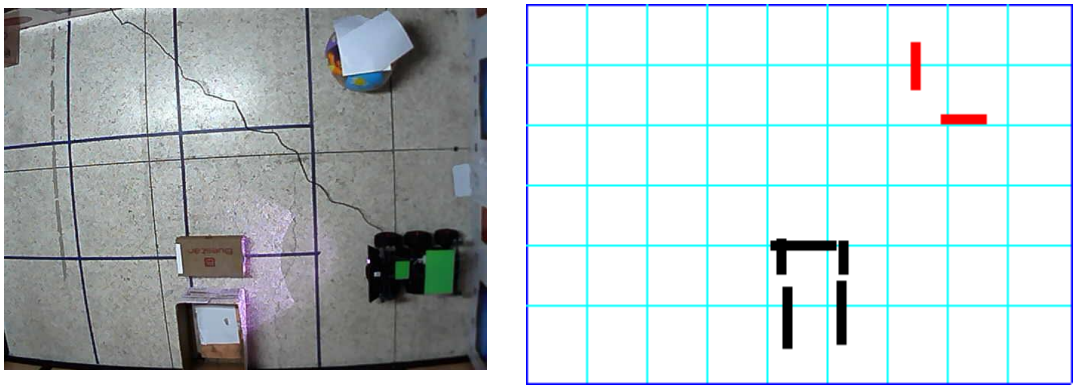

Figura 70. Generación del mapa, prueba 5E

Como se puede observar en la Figura 70, el mapa está completo, en este caso, uno de los objetos detectados en la parte inferior de la Figura 70, no empata en su borde izquierdo, se debe que como ya se mencionó la cámara IP tiene lente “ojo de pescado” por lo que existe un error al ubicar el objeto en el plano, de igual manera el objeto potencialmente peligroso no se puede diagramar de forma completa ya que como se mencionó es de forma circular por lo cual se tiene una aproximación de sus bordes.

Escenario 3: prueba 6, se cambió de lugar todos los objetos, y el recorrido empieza desde otro punto del área, obtención del mapa secuencial:

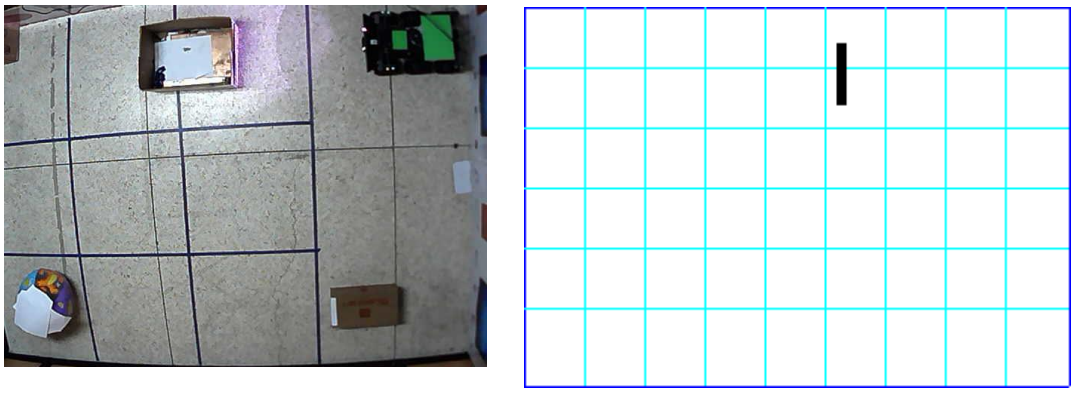

Figura 71. Generación del mapa, prueba 6A

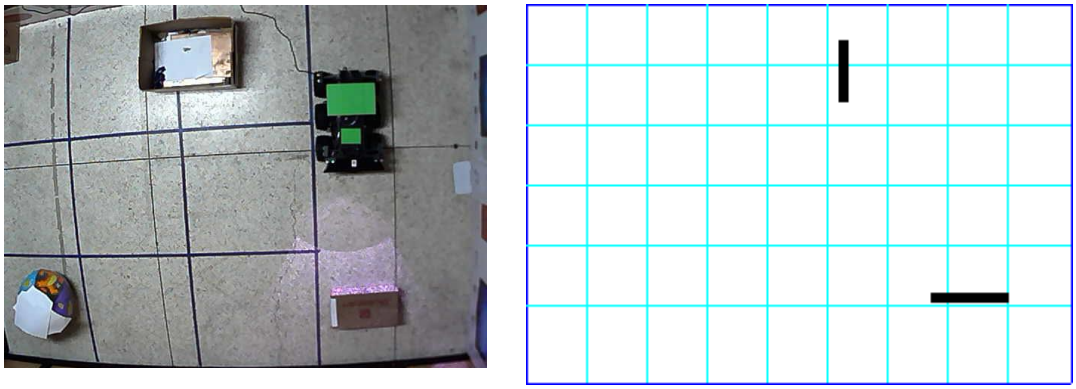

Figura 72. Generación del mapa, prueba 6B

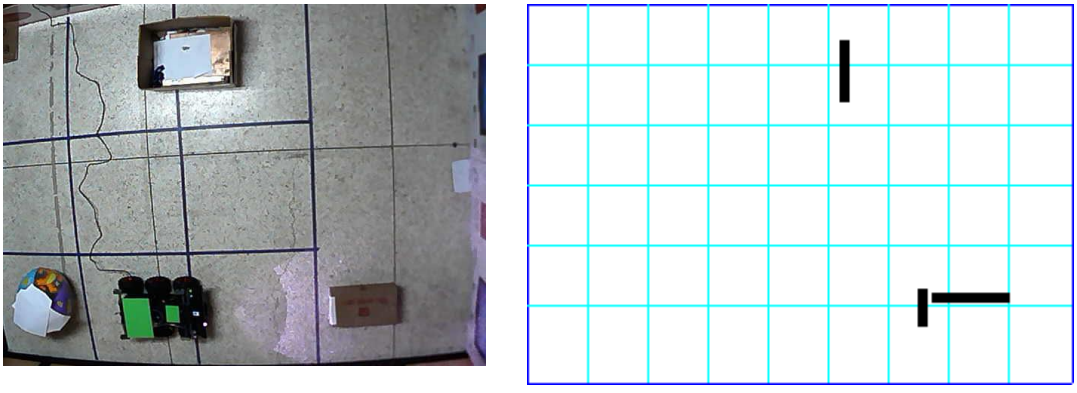

Figura 73. Generación del mapa, prueba 6C

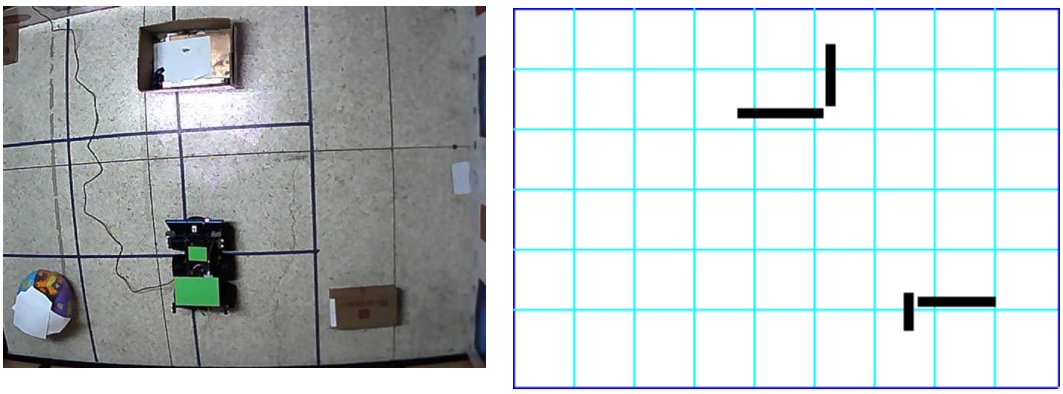

Figura 74. Generación del mapa, prueba 6D

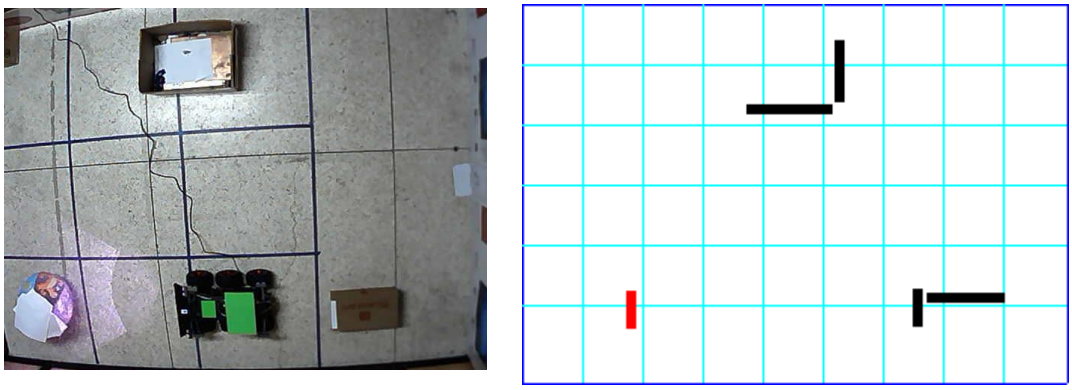

Figura 75. Generación del mapa, prueba 6E

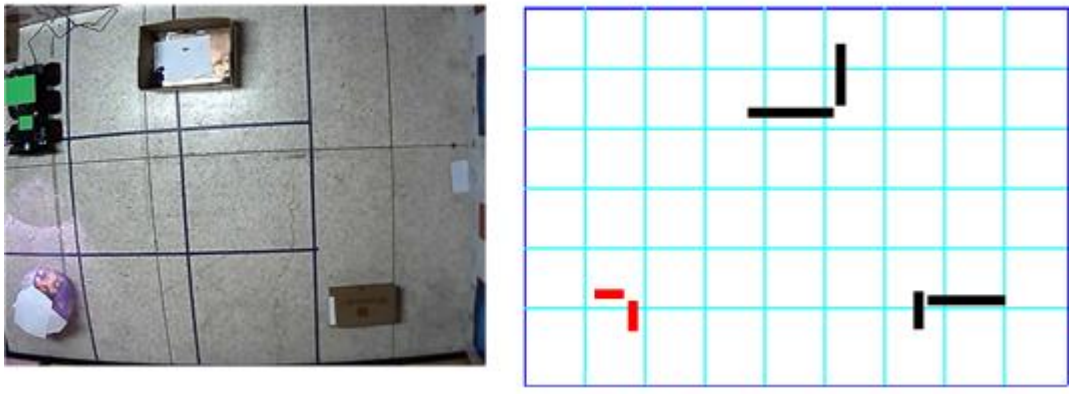

Figura 76. Generación del mapa, prueba 6F

Como se puede observar en la Figura 76, el mapa está construido, al igual que en casos anteriores se puede observar existe un espacio entre los bordes detectados, que como ya se mencionó se debe a la cámara IP.

- **Mapeo del entorno con 3 objetos; objeto potencialmente peligroso parcialmente cubierto.**

Escenario 4: prueba 7, para el último experimento se utilizó un objeto que obstruya la visión al objeto potencialmente peligroso, el mapa se construye de manera secuencial:

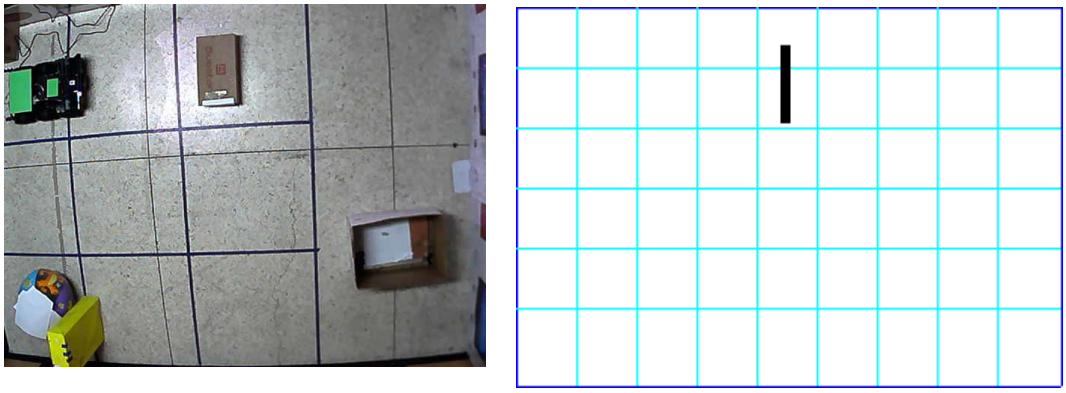

Figura 77. Generación del mapa, prueba 7A

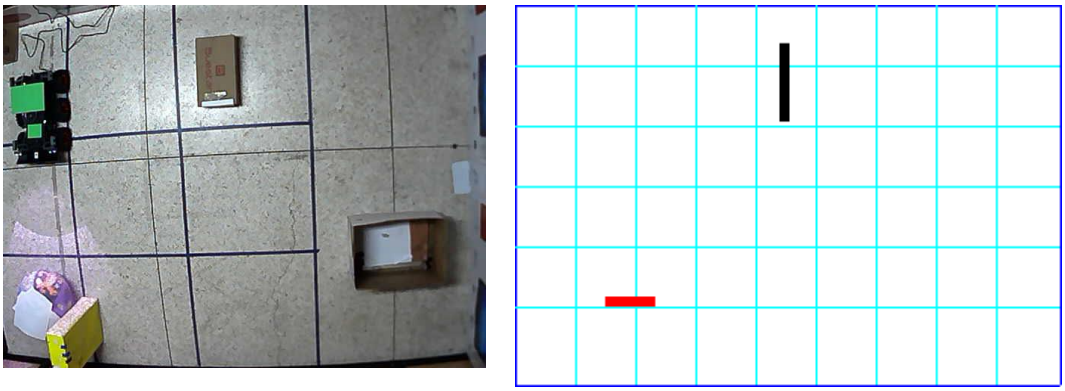

Figura 78. Generación del mapa, prueba 7B

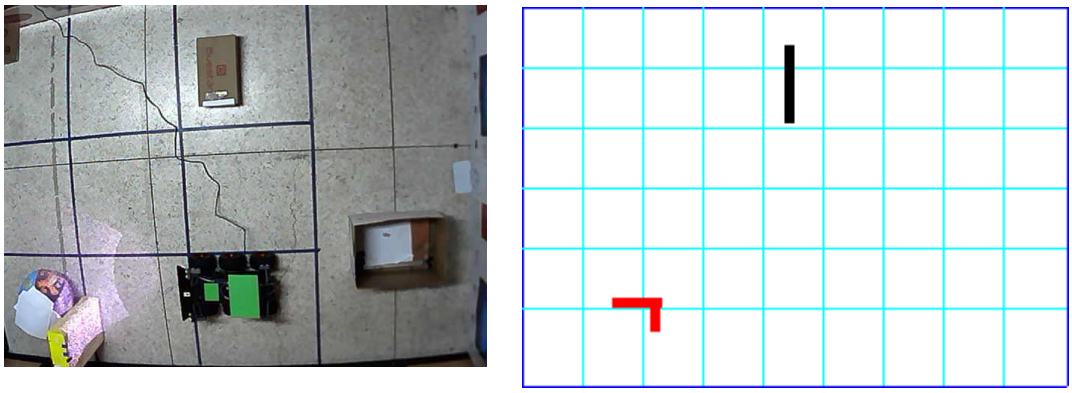

Figura 79. Generación del mapa, prueba 7C

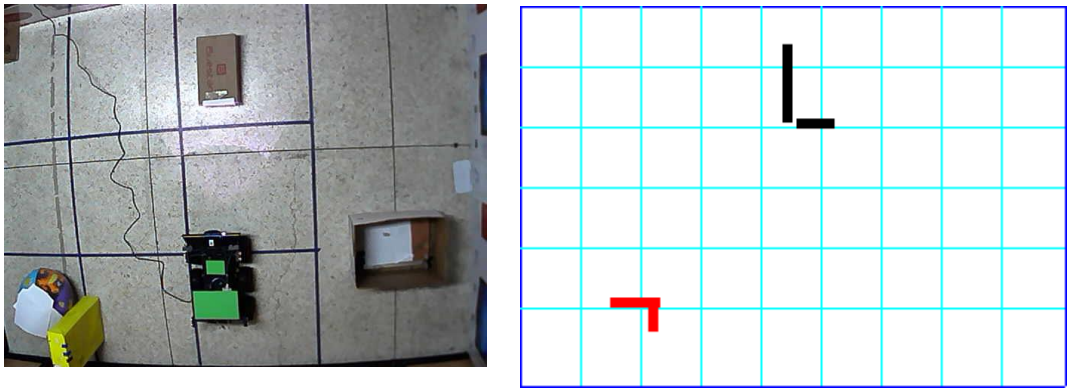

Figura 80. Generación del mapa, prueba 7D

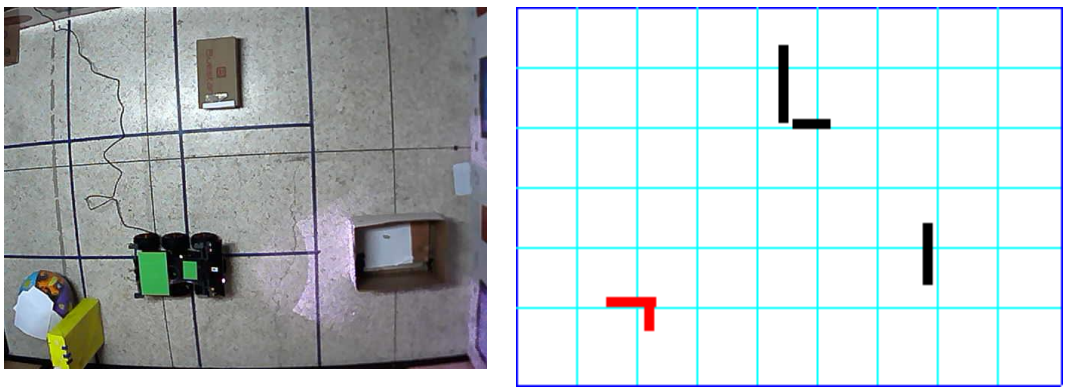

Figura 81. Generación del mapa, prueba 7E

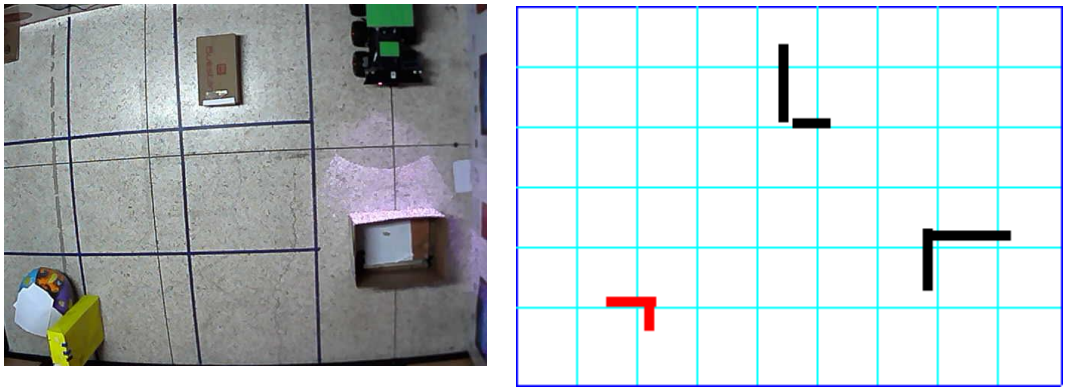

Figura 82. Generación del mapa, prueba 7F

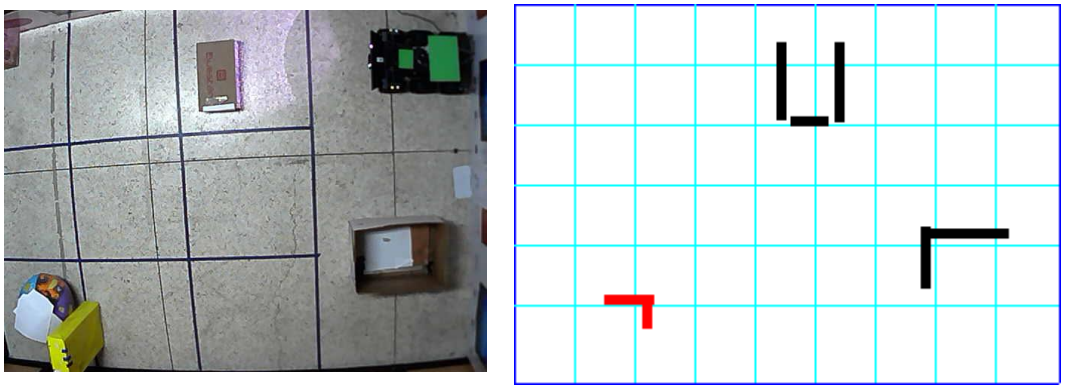

Figura 83. Generación del mapa, prueba 7G

Como se puede apreciar en la Figura 83, el mapa está completo, y su generación ha sido similar a las anteriores, con el error que existe en los bordes de las figuras detectadas debido a la cámara IP.

6.3 Cálculo de errores

En este apartado se procede a realizar el cálculo de los errores relativos porcentuales.

El primer error que se procede a calcular es el error de la longitud de los objetos detectados:

$$Er = \frac{valor\ real - valor\ medido}{valor\ real} * 100$$

Para este caso el valor real del objeto es de 18,5cm

El valor medido es 18,34 entonces:

$$Er = \frac{18{,}5 - 18{,}34}{18{,}5} * 100\% = \ 0{,}86\%$$

Para la longitud de los objetos se tiene un error relativo porcentual de 0,86%.

Ahora el error en la distancia de detección:

La distancia real es: 61,3cm

La distancia medida es: 60cm entonces:

$$Er = \frac{61{,}3 - 60}{61{,}3} * 100\% = 2{,}12\%$$

Para la distancia a la cual se detectan los objetos se tiene un error relativo porcentual de 2,12%.

Por último, el error en las coordenadas finales del robot en el plano horizontal:

Valor del punto real P2 (127,97)

Valor del punto medido P2 (120,92)

$$Er_x = \frac{127 - 120}{127} * 100\% = 5{,}51\%$$

$$Er_y = \frac{97 - 92}{97} * 100\% = 5{,}15\%$$

Como se puede observar los valores del error relativo porcentual en las coordenadas son de aproximadamente el 5%, como ya se mencionó anteriormente, esto se debe a que el lente de la cámara IP es "ojo de pescado" por tal motivo se tiene el error de cálculo en las coordenadas, lo que provoca el error en los bordes de las figuras detectadas mencionadas en las pruebas realizadas como se puede ver en la Figura 84

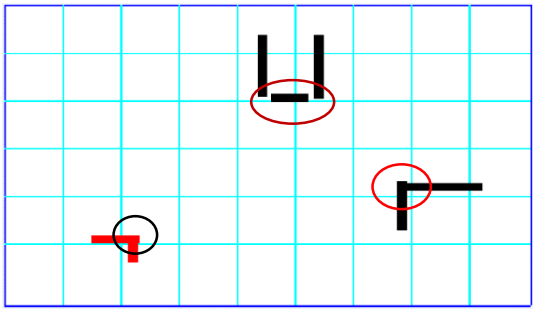

Figura 84. Mapa, marcado los errores

CAPÍTULO 7

CONCLUSIONES Y RECOMENDACIONES

7.1 Conclusiones

1. Se creó el algoritmo de visión artificial en el *software Python* para que mediante el procesamiento digital de imágenes detecte objetos e identifique cuál de ellos es potencialmente peligroso, en el ambiente controlado usando *software* libre para todo.

2. El algoritmo diseñado realizó el mapeo del ambiente controlado, lo cual permitió que se obtenga una imagen final en 2D del área que ha recorrido el robot, con la ubicación de los objetos que fueron detectados.

3. Por medio de la tarjeta *Raspberry Pi 3*, se realizó el procesamiento digital de imágenes en tiempo de real, lo cual permitió que el mapa se vaya generando mientras el robot recorría el ambiente controlado.

4. El algoritmo propuesto, detecta objetos presentes en el ambiente controlado, los cuales son marcados en color negro, y puede identificar cuál de ellos es un objeto potencialmente peligroso el mismo que es marcado de color rojo en el mapa final.

5. Mediante las pruebas realizadas se concluye que el algoritmo diseñado cumple con los objetivos propuestos.

6. La carga computacional del algoritmo en la tarjeta *Raspberry Pi 3*, fue alto, llegando a ocupar entre el 75 y 80% de la memoria que dispone la tarjeta.

7.2 Recomendaciones

1. Al usar una cámara IP estándar del mercado, se recomienda que las pruebas se las realice en un ambiente donde predomine la luz artificial sobre la luz natural, esto con el fin de evitar errores en la detección de colores.

2. Se recomienda trabajar en algoritmos para solucionar el problema que representa la luz natural para esta investigación.

3. Se recomienda que para mejorar la odometría del sistema, se trabaje sobre algoritmos que permitan corregir las fallas que se dan por el lente "ojo de pescado" de la cámara IP.

4. Se recomienda que para usar más tareas sobre la misma tarjeta sea de manera no simultánea o que se incluya otra *Raspberry*, ya que como se mencionó en las conclusiones este algoritmo ocupa cerca del 80% de la capacidad de memoria disponible, incluir más tareas puede provocar un mal funcionamiento en las mismas.

5. Se recomienda que para trabajos futuros se utilice una cámara IP con mejor resolución y un mayor ángulo de apertura, con el fin de aumentar el área del ambiente controlado.

Bibliografía

Alejandro. (Octubre de 2009). *Sistema de visión artificial*. Obtenido de file:///C:/Users/ROBERTO%20RUIZ/Downloads/IDMRM0910_Trabajo_VisionArtificial_v2.pdf

Alejandro, C., & Venegas, C. (2017). *Optimización e integración de una naríz electrónica autónoma enbebida en un sistema robótico para la identificación de sustancias explosivas como TNT y pólvora base doble en ambientes controlados.* Obtenido de http://repositorio.espe.edu.ec/bitstream/21000/13237/1/T-ESPE-057279.pdf

Alvarez, M. A. (19 de 11 de 2003). *Qué es Python.* Recuperado el 02 de 08 de 2017, de https://desarrolloweb.com/articulos/1325.php

AntI, I. (2010). *Programa de manipulación de imágenes de GNU*. Obtenido de https://docs.gimp.org/es/plug-in-gauss.html

Ayala, C., & Rosa, G. (02 de mayo de 2013). *Sistema de seguridad inteligente basado en reconocimiento de patrones mediante tecnología Kinect para restringir el acceso no autorizado a consolas de administración y monitoreo.* Obtenido de http://repositorio.espe.edu.ec/bitstream/21000/6517/1/T-ESPE-047111.pdf

Bazaga, A. R. (18 de agosto de 2015). *OpenCV: Librería de Visión por Computador*. Obtenido de http://osl.ull.es/software-libre/opencv-libreria-vision-computador/

Calahorrano, G. (mayo de 2015). *Desarrollo de un videojuego para complementar la enseñanza de matemáticas elementales en niños de segundo año de educación básica, utilizando el dispositivo de entrada por detección de movimiento "Kinect" de Microsoft.* Obtenido de http://repositorio.espe.edu.ec/bitstream/21000/10822/1/T-ESPE-048991.pdf

Chávez, M., & Tatayo, Í. (agosto de 2015). *Desarrollo e implementación de un sistema para el control de un robot móvil para acceso a lugares remotos utilizando tecnología Kinect.* Obtenido de http://repositorio.espe.edu.ec/xmlui/bitstream/handle/21000/10112/T-ESPEL-ENI-0356.pdf?sequence=1&isAllowed=y

Codigobit. (11 de Abril de 2010). *Cómo funciona el Kinect*. Obtenido de http://www.codigobit.info/2010/11/como-funciona-el-kinect.html

Componentes del sensor Kinect para Xbox 360. (s.f.). Obtenido de https://support.xbox.com/es-MX/xbox-360/accessories/kinect-sensor-components

Cristóforis, P., & Nitsche, M. (agosto de 2010). *Navegación de un robot móvil en ambientes desconocidos utilizando redes neuronales.* Obtenido de http://www-2.dc.uba.ar/materias/lrn/trabajos09/Cristoforis-Nitsche.pdf

Definición de Imagen. (14 de Mayo de 2014). Obtenido de http://conceptodefinicion.de/imagen/

Dioxido. (30 de 01 de 2017). *¿Cuantos FPS puede ver realmente el ojo humano ?* Obtenido de http://www.hd-tecnologia.com/cuantos-fps-puede-ver-realmente-el-ojo-humano/

EcuRed. (7 de octubre de 2012). *Imágen Digital*. Obtenido de https://www.ecured.cu/Imagen_digital

EcuRed. (15 de octubre de 2012). *Modelo HSV*. Obtenido de https://www.ecured.cu/Modelo_HSV

Ferdous, N. (6 de abril de 2015). *install opencv-2.4.11 in ubuntu*. Obtenido de https://gist.github.com/dynamicguy/3d1fce8dae65e765f7c4

Flores, P. (08 de junio de 2010). *Lunokhod 1, el robot lunar que se niega a morir*. Obtenido de https://hipertextual.com/2010/06/lunokhod-1-el-robot-lunar-que-se-niega-a-morir

FotoNostra. (2008). *Funcionamiento del ojo*. Obtenido de http://www.fotonostra.com/digital/ojohumano.htm

Gaibor, D., & Mediavilla, E. (01 de abril de 2016). *Desarrollo e implementación de un sistema robótico para el análisis de acetona en un ambiente controlado.* Obtenido de http://repositorio.espe.edu.ec/bitstream/21000/11834/1/T-ESPE-053235.pdf

García, F. J. (18 de Octubre de 2011). *Desarrollo e implantación de plataforma robótica móvil en entorno distribuido*. Obtenido de http://tierra.aslab.upm.es/documents/controlled/ASLAB-D-2011-019.pdf

García, R. M. (25 de junio de 2008). *explorando el mundo de la visión*. Obtenido de http://rosavision.blogspot.com/2008/

García, V. (22 de abril de 2009). *Introducción al Procesamiento Digital de Imágenes.* Obtenido de http://es.slideshare.net/IDVicMan/introduccion-al-procesamiento-digital-de-imagenes

Geek, I. (10 de noviembre de 2010). *Funcionamiento del sensor de movimiento en Kinect*. Obtenido de http://ideasgeek.net/2010/11/10/funcionamiento-del-sensor-de-movimiento-en-kinect/

Geologica, I. (22 de abril de 2013). *Procesamiento Digital de una Imágen*. Obtenido de https://es.slideshare.net/ingenieriageologica1/capitulo-vi-procesamiento-digital-de-una-imagen

Gl4r3. (2 de julio de 2014). *Detección de colores con OpenCV y Python*. Obtenido de https://robologs.net/2014/07/02/deteccion-de-colores-con-opencv-y-python/

Gl4r3. (26 de 07 de 2015). *Cómo filtrar el ruido de una máscara con OpenCV*. Obtenido de http://robologs.net/2015/07/26/como-filtrar-el-ruido-de-una-mascara-con-opencv/

Gonzalez, G. (16 de Febrero de 2015). *¿Cuál es la resolución del ojo humano?* Obtenido de http://blogthinkbig.com/cual-es-la-resolucion-del-ojo-humano/

Gracia, L. M. (2013). *¿Qué es OpenCV?* Obtenido de https://unpocodejava.wordpress.com/author/lsmgl/

Gutiérrez, J. (25 de febrero de 2016). *10 conceptos clave en visión artificial*. Obtenido de http://blogs.tecnalia.com/inspiring-blog/2016/02/25/vision-artificial/

Ilvay, R. (2014). *Sistema de educación para niños de 3 a 5 años, mediante un robot controlado por el sensor Kinect.* Obtenido de http://dspace.espoch.edu.ec/bitstream/123456789/3276/1/108T0089.pdf

Jaisingh, D. P. (31 de marzo de 2017). *Cómo utilizar imágenes e imágenes dinámicas con Captivate*. Obtenido de https://helpx.adobe.com/es/captivate/using/images-rollover-images.html

Jjazdanethe. (23 de 11 de 2009). *Evolución de la Robótica*. Obtenido de http://jjazdanethe16.blogspot.es/1258950780/

Leoro, J. (22 de mayo de 2013). *Diseño y construcción de un robot cuatricar asistente de simple tracción con seguimiento de esqueleto por medio del dispositivo de adquisición y procesamiento de imágenes Microsoft Kinect.* Obtenido de

http://repositorio.espe.edu.ec/bitstream/21000/6699/1/AC-MECA-ESPE-047175.pdf

Martinez. (Diciembre de 2015). *"Desarrollo del sistema de navegación de un robot móvil para pruebas de exploración en ambientes cerrados usando tecnología de robótica autónoma."*. Obtenido de http://www.inifim.uni.edu.pe/web/wp-content/uploads/2016/07/4.-Informe-final-2015-Proyec.-Robot-Movil-Ing.-Oliden.pdf

Mejía, J. (05 de febrero de 2015). *Kinect, OpenNI, NITE y OpenCV en Ubuntu*. Obtenido de http://entranasderobot.blogspot.com/2015/02/kinect-y-processing-en-ubuntu.html

Merino, J. P. (2010). *Ubuntu*. Obtenido de http://definicion.de/ubuntu/

Michelone, M. L. (02 de 08 de 2012). *Raspbian: Sistema operativo gratuito para la Raspberry Pi*. Obtenido de https://www.unocero.com/noticias/raspbian-sistema-operativo-gratuito-para-la-raspberry-pi/

Moya, P. (29 de febrero de 2016). *Todo sobre la nueva Raspberry Pi 3, más potente y conectada que nunca*. Obtenido de http://omicrono.elespanol.com/2016/02/raspberry-pi-3-model-b/

Murillo, A. (06 de noviembre de 2012). *¿Qué es el dispositivo Kinect?* Obtenido de http://www.kinectfordevelopers.com/es/2012/11/06/que-es-el-dispositivo-kinect/

Naman. (24 de Junio de 2014). *Experimenting with Kinect using opencv, python and open kinect (libfreenect)*. Obtenido de https://naman5.wordpress.com/2014/06/24/experimenting-with-kinect-using-opencv-python-and-open-kinect-libfreenect/

OpenKinect. (02 de agosto de 2012). *Main Page/es*. Obtenido de https://openkinect.org/wiki/Main_Page/es

Orfila, M. (30 de enero de 2015). *La vista el sentido mas importante*. Obtenido de http://www.cromo.com.uy/la-vista-es-el-sentido-mas-importante-n588224

Porras, J. (2012). *Clasification system based on computer vision*. Obtenido de http://www.urp.edu.pe/pdf/ingenieria/electronica/CAP-1_Taller_de_Electronica_IV_b.pdf

Rahman, A. (20 de septiembre de 2013). *Contour Properties*. Obtenido de https://github.com/abidrahmank/OpenCV2-Python-Tutorials/blob/master/source/py_tutorials/py_imgproc/py_contours/py_contour_properties/py_contour_properties.rst

Ramsey, k. (18 de febrero de 2013). *Controlling Foscam MJPEG Cameras in Python*. Obtenido de http://foscam.us/forum/controlling-foscam-mjpeg-cameras-in-python-t4591.html#p21614

Shun, R. (25 de marzo de 2015). *Sistemas de Visión Artificial, historia, componentes principales y procesamiento digital de imágenes.* Obtenido de https://es.scribd.com/doc/259914658/Sistemas-de-Vision-Artificial-Historia-Componentes-y-Procesamiento-de-Imagenes

Veciana, B. (2011). *Agudeza visual: ¿cómo vemos?* Obtenido de http://www.upc.edu/saladepremsa/informacio/monografics/agudeza-visual-bfcomo-vemos?set_language=es

Velasco, N. (abril de 2013). *Desarrollo e implementación de un algoritmo para detección de objetos con tecnología Kinect.* Obtenido de http://repositorio.espe.edu.ec/bitstream/21000/6155/1/T-ESPEL-ENI-0299.pdf

Velasco, R. (2015 de 09 de 2015). *Raspbian, el sistema operativo para Raspberry Pi, se actualiza a Debian 8 "Jessie".* Recuperado el 02 de 08 de 2017, de https://www.redeszone.net/2015/09/30/raspbian-el-sistema-operativo-para-raspberry-pi-se-actualiza-a-debian-8-jessie/

Villalba, F. (18 de agosto de 2015). *Diseño e implementación de un sistema teleoperado para un brazo robótico "Robotic arm edge" con sensor Kinect para Windows a través de una red Wireless.* Obtenido de http://repositorio.espe.edu.ec/bitstream/21000/10908/1/T-ESPE-049231.pdf

Visión Artificial. (2014). Obtenido de http://www.etitudela.com/celula/downloads/visionartificial.pdf

Visión Artificial e Interacción sin mandos. (diciembre de 2010). Obtenido de http://sabia.tic.udc.es/gc/Contenidos%20adicionales/trabajos/3D/VisionArtificial/

Visión artificial e interpretación sin mandos. (12 de 2010). Obtenido de http://sabia.tic.udc.es/gc/Contenidos%20adicionales/trabajos/3D/VisionArtificial/