



**MEDICIÓN DE CALIBRE DE PLANTAS EN VIVEROS  
FRUTALES CON VISIÓN ESTEREOSCÓPICA Y  
PROCESAMIENTO DIGITAL DE IMÁGENES**

**MEMORIA PARA OPTAR AL TÍTULO DE  
INGENIERO CIVIL MECATRÓNICA**

**Profesor Guía:** Prof. M. Sc. Ignacio Torres

**JOSÉ DANIEL URIBE OSORIO**

**Curicó - Chile**

**2021**

## CONSTANCIA

La Dirección del Sistema de Bibliotecas a través de su encargado Biblioteca Campus Curicó certifica que el autor del siguiente trabajo de titulación ha firmado su autorización para la reproducción en forma total o parcial e ilimitada del mismo.



Curicó, 2023

# **Medición de calibre de plantas en viveros frutales con visión estereoscópica y procesamiento digital de imágenes**

**José Daniel Uribe Osorio**

**Profesor Guía**  
Ignacio Torres Herrera  
Universidad de Talca  
Facultad de Ingeniería  
Camino a Los Niches, Km. 1, s/n  
igtorres@utalca.cl

©José Daniel Uribe Osorio

# Resumen

El trabajo de investigación revisado corresponde a la implementación de un sistema de visión estereoscópica para la medición de calibre de plantas en viveros frutales; realizado con procesamiento digital de imágenes en *Python* con ayuda de la librería *OpenCV*. Obteniendo como resultados en esta primera etapa, conocimientos de como calcular profundidad de un objeto dentro de una imagen y simulaciones que aluden a la extracción de características específicamente al diámetro de un objeto dentro de una imagen.

El desarrollo de sistemas de visión estereoscópica se utiliza principalmente para la obtención de información tridimensional a partir de recursos bidimensionales, como lo son las capturas de imagen mediante el uso de elementos fotográficos. Las cámaras web comienzan a ganar terreno dentro del área de proyectos de bajo costo por sus altas prestaciones de capturas en alta calidad por bajos precios de adquisición. Se busca implementar un sistema funcional mediante la compra de dos cámaras web que capturen fotografías de un objetivo y mediante transformaciones digitales de imágenes se logre una correspondencia de características en las fotografías para la obtención de un mapa de profundidades y posterior obtención de distancia del objeto de estudio a las cámaras.

El proyecto se enfoca, aparte de la implementación de una herramienta de medición de calibre con uso de la visión estéreo, en el diseño de un robot móvil con la capacidad de transportar dicha herramienta realizando una ruta de toma de mediciones siguiendo una ruta con sensores reflectantes infrarrojos que a través de un programa el microcontrolador es capaz de dar instrucciones al robot móvil de seguir una línea mientras realiza las mediciones dentro de un laboratorio de ambiente controlado.

También, se exponen los procedimientos realizados para lograr los objetivos propuestos dentro de la etapa de planificación del proyecto, en donde en una primera instancia se encuentra la búsqueda de información e investigación sobre programación y lenguajes de programación compatibles con las herramientas a utilizar. Para luego, pasar a una etapa de prototipado básico para la extracción de información. Y así finalmente, generar un diseño completo de un sistema robótico haciendo uso de la programación, diseño mecánico y electrónico con un enfoque mecatrónico.

Por último, se presentan los resultados obtenidos de la experiencia práctica de toma de imágenes con *webcams* como par estereoscópico; logrando medir en base a imágenes, el diámetro de un objeto determinado de estudio para su clasificación según calibre. Considerando que este objeto representa el tallo de una planta frutal dentro de un vivero.



# Palabras Clave

Visión estereoscópica, robot móvil, profundidad, raspberry pi 4, algoritmos, OpenCV, disparidad, calibre, vivero.

# Índice

|                                                                                        |           |
|----------------------------------------------------------------------------------------|-----------|
| <b>1. Introducción</b>                                                                 | <b>1</b>  |
| 1.1. Introducción general . . . . .                                                    | 1         |
| 1.2. Estado del arte . . . . .                                                         | 2         |
| 1.2.1. Viveros frutales . . . . .                                                      | 2         |
| 1.2.2. Visión estéreo . . . . .                                                        | 2         |
| 1.2.3. Robot móvil . . . . .                                                           | 5         |
| 1.3. Revisión bibliográfica . . . . .                                                  | 7         |
| 1.3.1. Trabajos previos . . . . .                                                      | 7         |
| 1.4. Discusión . . . . .                                                               | 11        |
| 1.5. Hipótesis . . . . .                                                               | 12        |
| 1.6. Objetivos . . . . .                                                               | 12        |
| 1.6.1. Objetivo general . . . . .                                                      | 12        |
| 1.6.2. Objetivos específicos . . . . .                                                 | 12        |
| 1.7. Alcances del proyecto . . . . .                                                   | 13        |
| 1.8. Metodología . . . . .                                                             | 14        |
| <b>2. Aplicación de visión estéreo para medición de calibre</b>                        | <b>15</b> |
| 2.1. Definición de lenguaje de programación a utilizar . . . . .                       | 15        |
| 2.2. Cálculo y obtención del ancho de un objeto a través de una imagen . . . . .       | 16        |
| 2.2.1. Obtención de imagen . . . . .                                                   | 18        |
| 2.2.2. Búsqueda de bordes . . . . .                                                    | 18        |
| 2.2.3. Búsqueda de líneas rectas . . . . .                                             | 21        |
| 2.2.4. Cálculo de ancho o diámetro en píxeles . . . . .                                | 22        |
| 2.2.5. Transformación de píxeles a milímetros . . . . .                                | 23        |
| 2.3. Cálculo de profundidad mediante visión estéreo . . . . .                          | 24        |
| 2.3.1. Geometría del sistema . . . . .                                                 | 24        |
| 2.3.2. Calibración de cámaras individuales . . . . .                                   | 26        |
| 2.3.3. Calibración de las cámaras como par estereoscópico . . . . .                    | 29        |
| 2.3.4. Correspondencia estéreo . . . . .                                               | 31        |
| 2.3.5. Cálculo de distancia mediante uso de visión estéreo . . . . .                   | 33        |
| 2.3.6. Comentarios . . . . .                                                           | 37        |
| 2.4. Implementación de algoritmos en un ordenador de placa reducida . . . . .          | 38        |
| 2.4.1. Justificación . . . . .                                                         | 38        |
| 2.4.2. Depuración de códigos . . . . .                                                 | 39        |
| 2.4.3. Algoritmos depurados aplicados en Raspberry Pi 4 . . . . .                      | 39        |
| 2.5. Especificación de materiales utilizados para pruebas . . . . .                    | 41        |
| 2.5.1. Especificaciones técnicas de la cámara web Logitech C270 HD. . . . .            | 41        |
| 2.5.2. Planos de sistema de sujeción . . . . .                                         | 42        |
| <b>3. Diseño de robot móvil para transporte de instrumento de medición<br/>estéreo</b> | <b>43</b> |
| 3.1. Robot móvil tipo diferencial . . . . .                                            | 43        |
| 3.1.1. Requerimientos . . . . .                                                        | 43        |
| 3.1.2. Selección de componentes . . . . .                                              | 43        |
| 3.2. Diseño 3D del robot móvil . . . . .                                               | 45        |

|                                                                                           |           |
|-------------------------------------------------------------------------------------------|-----------|
| 3.2.1. Análisis de elemento finito . . . . .                                              | 46        |
| <b>4. Diseño electrónico</b>                                                              | <b>49</b> |
| 4.1. Requerimientos del sistema . . . . .                                                 | 49        |
| 4.2. Selección de componentes electrónicos . . . . .                                      | 50        |
| 4.2.1. Selección de método de movimiento autónomo . . . . .                               | 50        |
| 4.2.2. Selección de motores DC . . . . .                                                  | 52        |
| 4.2.3. Controlador de motores - puente H . . . . .                                        | 53        |
| 4.2.4. Seguidor de línea . . . . .                                                        | 53        |
| 4.2.5. Microcontrolador . . . . .                                                         | 53        |
| 4.2.6. Alimentación de componentes . . . . .                                              | 54        |
| 4.2.7. Selección de batería . . . . .                                                     | 54        |
| 4.3. Diagramas de conexión . . . . .                                                      | 55        |
| 4.4. Programa Arduino UNO . . . . .                                                       | 56        |
| 4.5. Clasificación de calibre . . . . .                                                   | 57        |
| 4.5.1. Componentes . . . . .                                                              | 58        |
| 4.5.2. Comunicación I2C . . . . .                                                         | 58        |
| 4.5.3. Diagrama de conexión . . . . .                                                     | 59        |
| 4.5.4. Programación . . . . .                                                             | 60        |
| <b>5. Resultados experimentales</b>                                                       | <b>62</b> |
| 5.1. Primer acercamiento de prueba a la programación con OpenCV y C++ . . . . .           | 62        |
| 5.2. OpenCV y Python . . . . .                                                            | 63        |
| 5.2.1. Obtención de imágenes para la calibración de cámaras . . . . .                     | 63        |
| 5.2.2. Cálculo de distancia . . . . .                                                     | 64        |
| 5.2.3. Cálculo de diámetro . . . . .                                                      | 67        |
| 5.3. Implementación de sistema de prueba para visión estereoscópica . . . . .             | 75        |
| 5.4. Resultados finales aplicados en Raspberry Pi 4 . . . . .                             | 78        |
| 5.5. Aplicación de movimiento a robot móvil . . . . .                                     | 80        |
| <b>6. Conclusiones</b>                                                                    | <b>84</b> |
| 6.1. Sumario . . . . .                                                                    | 84        |
| 6.2. Conclusiones . . . . .                                                               | 86        |
| 6.3. Plan de cierre . . . . .                                                             | 87        |
| 6.4. Trabajos futuros . . . . .                                                           | 87        |
| <b>7. Anexos</b>                                                                          | <b>91</b> |
| A. Especificaciones de ordenador en primera iteración de creación de algoritmos . . . . . | 91        |
| B. Especificaciones técnicas de la Raspberry Pi 4. . . . .                                | 92        |
| C. Planos de robot . . . . .                                                              | 93        |
| D. Planos de sistema de sujeción . . . . .                                                | 99        |
| E. Código depurado para uso en <i>Raspberry Pi 4</i> . . . . .                            | 103       |
| E.a. Código principal . . . . .                                                           | 103       |
| E.b. Calibración de cámaras . . . . .                                                     | 104       |
| E.c. Detección de objetivo . . . . .                                                      | 106       |
| E.d. Cálculo de distancia . . . . .                                                       | 108       |

|      |                                                  |     |
|------|--------------------------------------------------|-----|
| E.e. | Cálculo de diámetro . . . . .                    | 110 |
| E.f. | Programa Arduino UNO seguidor de línea . . . . . | 113 |

## Índice de Figuras

|                                                                                                                                                                                          |    |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 1.1. 1. Córnea, 2. Iris, 3. Cristalino, 4. Cuerpo ciliar, 5. Esclera, 6. Coroides, 7. Retina, 8. Nervio óptico, 9. Canal hialoideo, 10. Humor vítreo, 11. Ora serrata. [4] . . . . .     | 3  |
| 1.2. Robot tipo diferencial diseñado para el proyecto. . . . .                                                                                                                           | 6  |
| 1.3. Resultados experimentales de la implementación de visión estéreo en la navegación autónoma de vehículos tripulados. Experimento de medición de distancia. Extraído de [13]. . . . . | 8  |
| 1.4. Vista ampliada de definición de valores límites en interfaz gráfica para la clasificación según tamaño. [16] . . . . .                                                              | 9  |
| 1.5. Resultado obtenido para la detección y cuantificación de defectos. [17] .                                                                                                           | 10 |
| 2.1. Resultado obtenido de un modelado 3D con Regard 3D. . . . .                                                                                                                         | 15 |
| 2.2. Resultado de la ejecución del Código 5.1. . . . .                                                                                                                                   | 17 |
| 2.3. Diagrama de flujo de los procedimientos realizados para el cálculo de ancho o diámetro de un tronco. . . . .                                                                        | 18 |
| 2.4. Tránsito de un kernel. Extraído de [20]. . . . .                                                                                                                                    | 19 |
| 2.5. Ejemplo imagen con aplicación de algoritmo de Canny. Extraído de [21].                                                                                                              | 21 |
| 2.6. Configuración Canónica de un sistema estéreo [24]. . . . .                                                                                                                          | 24 |
| 2.7. Relación Geométrica de un Sistema de Visión Estereoscópica [26]. . . .                                                                                                              | 25 |
| 2.8. Modelo básico de una cámara. Extraído de [27]. . . . .                                                                                                                              | 27 |
| 2.9. Diagrama de flujo que describe el proceso de obtención de distancia con visión estéreo. . . . .                                                                                     | 34 |
| 2.10. Fotografía izquierda y derecha captadas y rectificadas. Extraído de [30].                                                                                                          | 35 |
| 2.11. Cinco direcciones consideradas en el algoritmo <i>SGBM</i> . . . . .                                                                                                               | 36 |
| 2.12. Ejemplo de aplicación de la eliminación de ruidos según <i>MorphologyEx</i> de clase "close". Extraído de [32]. . . . .                                                            | 37 |
| 2.13. Raspberry Pi 4 en donde se implementa el proyecto de visión estéreo. .                                                                                                             | 38 |
| 2.14. Diagrama de flujo explicativo de algoritmo depurado para implementación en raspberry pi 4. . . . .                                                                                 | 40 |
| 2.15. Sistema par el montaje de las cámaras web ensamblado. . . . .                                                                                                                      | 42 |
| 3.1. Diseño de la rueda. . . . .                                                                                                                                                         | 44 |
| 3.2. Diseño de la rueda loca. . . . .                                                                                                                                                    | 44 |
| 3.3. Modelo 3D del robot móvil. . . . .                                                                                                                                                  | 45 |
| 3.4. Plano y distribución de componentes sobre placa inferior. . . . .                                                                                                                   | 46 |
| 3.5. Plano y distribución de componentes en placa superior del robot. . . .                                                                                                              | 47 |
| 3.6. Modelo 3D del conjunto motor - rueda. . . . .                                                                                                                                       | 47 |
| 3.7. Análisis estático, tensión de Von Misses. . . . .                                                                                                                                   | 48 |
| 4.1. Diagrama de conexión para circuito de alimentación del robot. . . . .                                                                                                               | 56 |
| 4.2. Diagrama de conexión para circuito de control del robot. . . . .                                                                                                                    | 56 |
| 4.3. Montaje de sensores reflectivos infrarrojos. . . . .                                                                                                                                | 57 |
| 4.4. Pantalla LCD LCM-S01602DTR. Extraído de [41]. . . . .                                                                                                                               | 58 |
| 4.5. Módulo interfaz serial I2C. Extraído de [43]. . . . .                                                                                                                               | 59 |
| 4.6. Estructura de un mensaje I2C. Extraído de [44]. . . . .                                                                                                                             | 60 |
| 4.7. Diagrama de conexión para visualizar resultados en pantalla LCD. . . .                                                                                                              | 60 |
| 5.1. Curva de comportamiento y curva aproximada con la función de cuarto orden. . . . .                                                                                                  | 67 |

|                                                                                                   |     |
|---------------------------------------------------------------------------------------------------|-----|
| 5.2. Imágenes de prueba obtenidas. . . . .                                                        | 69  |
| 5.3. Líneas detectadas gracias a las funciones en el Código 5.12. . . . .                         | 70  |
| 5.4. Resultado de la superposición de imágenes. . . . .                                           | 71  |
| 5.5. Gráfico para la aproximación de una curva y su función. . . . .                              | 74  |
| 5.6. Montaje de cámaras en sistema de sujeción. . . . .                                           | 75  |
| 5.7. Imagen para la calibración de las cámaras. . . . .                                           | 76  |
| 5.8. Disposición de materiales en el espacio físico de pruebas. . . . .                           | 76  |
| 5.9. Mapa de profundidad que entrega la distancia de las cámaras al objeto<br>de estudio. . . . . | 77  |
| 5.10. Resultado final, listo para la clasificación. . . . .                                       | 78  |
| 5.11. Situación de estudio. . . . .                                                               | 78  |
| 5.12. Resultado de la detección de objetivo y ubicación aproximada de su centro. . . . .          | 79  |
| 5.13. Espacio de trabajo del sistema estéreo para correctas mediciones. . . . .                   | 79  |
| 5.14. Resultado de aplicación de algoritmos en Raspberry Pi 4. . . . .                            | 80  |
| 5.15. Mensajes finales del resultado de aplicación de algoritmos en Raspberry<br>Pi 4. . . . .    | 80  |
| 5.16. Simulación de comportamiento seguimiento de línea. Recorrido en línea<br>recta. . . . .     | 81  |
| 5.17. Simulación de comportamiento seguimiento de línea. Giro a la izquierda. . . . .             | 82  |
| 5.18. Simulación de comportamiento seguimiento de línea. Giro a la derecha. . . . .               | 83  |
| 7.1. Placa inferior robot móvil. . . . .                                                          | 93  |
| 7.2. Placa superior robot móvil. . . . .                                                          | 94  |
| 7.3. Placa frontal robot móvil. . . . .                                                           | 95  |
| 7.4. Placa lateral robot móvil. . . . .                                                           | 96  |
| 7.5. Distribución de componentes en la placa inferior. . . . .                                    | 97  |
| 7.6. Distribución de componentes en la placa superior. . . . .                                    | 98  |
| 7.7. Pieza de agarre para cámara web. . . . .                                                     | 99  |
| 7.8. Pieza de anclaje de dos cámaras. . . . .                                                     | 100 |
| 7.9. Pieza base, para la estabilización y postura en superficies. . . . .                         | 101 |
| 7.10. Pieza base modificada para montaje en robot móvil. . . . .                                  | 102 |

## Índice de Tablas

|                                                                                      |    |
|--------------------------------------------------------------------------------------|----|
| 1.1. Terminología empleada para el calibre de la planta [3]. . . . .                 | 2  |
| 2.1. Especificaciones técnicas de <i>Webcam Logitech C270 HD</i> . . . . .           | 42 |
| 3.1. Propiedades del acrílico. Información extraída de [33]. . . . .                 | 43 |
| 4.1. Matriz de priorización para elección de control de movimiento de robot. . . . . | 51 |
| 4.2. Parámetros de funcionamiento del robot. . . . .                                 | 52 |
| 4.3. Componentes conectados a microcontrolador. . . . .                              | 54 |
| 4.4. Consumo máximo de componentes conectados a la batería. . . . .                  | 55 |
| 7.1. Especificaciones técnicas computador de pruebas. . . . .                        | 91 |

## Acrónimos

|             |                                                       |
|-------------|-------------------------------------------------------|
| <i>THD</i>  | Distorsión armónica total (Total Armonic Distortion). |
| <i>IDE</i>  | Integrated Development Environment.                   |
| <i>RGB</i>  | Red Green Blue                                        |
| <i>BGR</i>  | Blue Green Red                                        |
| <i>RMF</i>  | Robot Motor Factor                                    |
| <i>PWM</i>  | Pulse Width Modulation                                |
| <i>IOT</i>  | Internet Of Things                                    |
| <i>RAE</i>  | Real Academia Española                                |
| <i>SGBM</i> | Semi-Global Block Matching                            |
| <i>LCD</i>  | Liquid-Crystal Display                                |
| <i>I2C</i>  | Inter Integrated Circuits                             |



# 1. Introducción

## 1.1. Introducción general

La economía y sociedad rural chilena y latinoamericana se han transformado notablemente en las últimas décadas, a consecuencia del creciente desarrollo capitalista de la agricultura y de su mayor integración a la economía mundial. Con la industrialización de la agricultura, el poder de la agroindustria creció nacional e internacionalmente, convirtiéndose en un actor clave en el desarrollo del régimen alimentario mundial [1]. Chile basa gran parte de su economía en la agroindustria y los avances tecnológicos que se han utilizado para mejorar la calidad y cantidad de las producciones van desde los estudios de laboratorio de las plantas y terrenos hasta la aplicación y uso de tecnologías inalámbricas para la recolección de información sobre ciertos parámetros requeridos para la mejora de los productos agrícolas (Implementación de tecnologías IOT) [2].

Según la *RAE*, un vivero se define como un terreno adonde se trasplantan desde la almáciga los árboles pequeños, para trasponerlos, después de recriados, a su lugar definitivo. En Chile, cada vez aumenta la cantidad de viveros frutales, lo cual ha significado una expansión que también trae consigo que una mayor cantidad de terreno y plantas dentro de los viveros sea más y más común. Las plantas dentro de viveros deben ser clasificadas según características físicas y morfológicas que posean, principalmente según su calibre.

El calibre de plantas frutales se mide mediante el diámetro del tallo o tronco a 1 m de altura desde el cuello de su raíz, en este caso la medición se realiza mediante instrumentos manuales como el calibre (pie de metro), cintas de medir y herramientas creadas mediante moldes, las cuales son susceptible a un gran error provocado por la percepción de quienes están a cargo de realizar la tarea. Además, según información recabada en conversaciones con personal de viveros existe una pérdida provocada por la devolución de productos por una incorrecta medición de calibre. Dado el problema anterior; se propone una alternativa de solución basado en el sistema de visión estereoscópica. Este último término es una simulación de la visión humana haciendo uso de dos cámaras separadas a una distancia conocida, con el fin de obtener información tridimensional para conocer las distancias que existen entre las cámaras y los objetos que están dentro de las imágenes captadas.

Las tecnologías actuales permiten aumentar la productividad y disminuir pérdidas debido a procesos manuales que se realizan. Particularmente, la solución propuesta en este proyecto de memoria consta en el diseño de un robot móvil capaz de desplazarse dentro de un laboratorio en condiciones ideales simulando ser un vivero para realizar mediciones de calibre haciendo uso de un par de cámaras de acceso comercial capaces de recabar información tridimensional para la obtención de correctas mediciones.

## 1.2. Estado del arte

### 1.2.1. Viveros frutales

“El término ampliamente utilizado para definir el tamaño de la planta es “**el calibre**”. Éste hace referencia al perímetro del tronco (cm) medido a 1 *m* de altura a partir del cuello de la raíz. No se ‘calibra’ en el cuello de la raíz porque éste es muy variable. Usualmente los viveristas catalogan sus plantas en función de un rango o intervalo. Por ejemplo, plantas con un calibre de 6 – 8 significa que, su perímetro del tronco medido a un metro del cuello de la raíz del árbol, está comprendido entre 6 y 8 *cm*. Existe equivalencia entre el perímetro del tronco y su diámetro” [3], la clasificación realizada es hecha principalmente para establecer los precios de las plantas para su posterior venta, mientras más diámetro o perímetro tenga la planta, más aumenta su valor. Cabe mencionar que siempre que se habla del calibre de una planta se está refiriendo a plantas que se denominan “**arbolado**”, éstas son de tamaño arbóreo con un sólo tronco y miden a lo menos 2 metros de alto. En la Tabla 1.1, se pueden visualizar los diversos calibres y distribución de categorías.

Tabla 1.1: Terminología empleada para el calibre de la planta [3].

| CALIBRE | PERÍMETRO (cm) | DIÁMETRO(mm)  |
|---------|----------------|---------------|
| 6 – 8   | Entre 6 y 8    | Entre 19 y 25 |
| 8 – 10  | Entre 8 y 10   | Entre 25 y 32 |
| 10 – 12 | Entre 10 y 12  | Entre 32 y 38 |
| 12 – 14 | Entre 12 y 14  | Entre 38 y 45 |
| 14 – 16 | Entre 14 y 16  | Entre 45 y 51 |
| 16 – 18 | Entre 16 y 18  | Entre 51 y 57 |
| 18 – 20 | Entre 18 y 20  | Entre 57 y 64 |
| 20 – 25 | Entre 20 y 25  | Entre 64 y 80 |
| 25 – 30 | Entre 25 y 30  | Entre 80 y 95 |

### 1.2.2. Visión estéreo

La visión humana puede definirse como un sistema integrado, de forma genérica, por dos ojos, el nervio óptico y el cerebro. El globo ocular posee tres túnicas, la capa fibrosa, la capa vascular y la capa nerviosa. La capa fibrosa es la capa más superficial del ojo y se compone por la córnea y la esclera, la primera de ellas representa el principal medio refractivo del ojo, es transparente y debe ser lo suficientemente fuerte como para soportar la presión intraocular para proteger las delicadas estructuras internas del ojo, además actúa como barrera contra microorganismos y amenazas externas, este sólo cubre la porción del ojo que posee la pupila, cristalino e iris. La esclera, es una membrana blanca y resistente, esta proporciona rigidez al globo ocular y protege los elementos internos de est, es la parte blanca del ojo como se le conoce y cubre la mayor parte de este. La ubicación de la esclera y la córnea es en la capa más externa del ojo y estas se unen por medio del limbo esclerocorneal. La capa vascular se compone de tres capas: el iris, el cuerpo ciliar y el coroides. El primero de estos es la parte anterior de la capa vascular, se proyecta desde el cuerpo ciliar y es una estructura que puede variar en su pigmentación, caracterizándose por dar el color a los ojos. En su centro se encuentra una apertura central, la pupila. El iris tiene la capacidad de modificar

el diámetro pupilar (dilatación y constricción), controlando la cantidad de luz que se dirige hacia la retina. El cuerpo ciliar es una estructura que hace puente entre las estructuras anteriores y posteriores del ojo y mantiene el cristalino en su posición; es muy pigmentado, se encuentra entre la coroides y el iris. El coroides es una capa pigmentada muy vascularizada que se une con la retina en su parte interna y con la esclera en su parte externa, nutre la parte posterior de la retina. La capa nerviosa se compone de la retina y los medios transparentes del ojo, donde la retina es la capa más interna del globo ocular; es una capa fotosensible que contiene fotorreceptores y los elementos neuronales que inician el procesamiento de la información visual. Su misión es transformar la luz que recibe, en un impulso nervioso que viaja hasta el cerebro a través del nervio óptico, y se convierte en las imágenes que percibimos. Los medios transparentes del ojo se encargan de permitir el paso desde el medio ambiente externo; dentro de los cuales se encuentra el cristalino, que es un lente biconvexo que está suspendido en medio del ojo, se encarga del reflejo de acomodación necesario para la visión de cerca o para enfocar correctamente los objetos a diferentes distancias. [4]

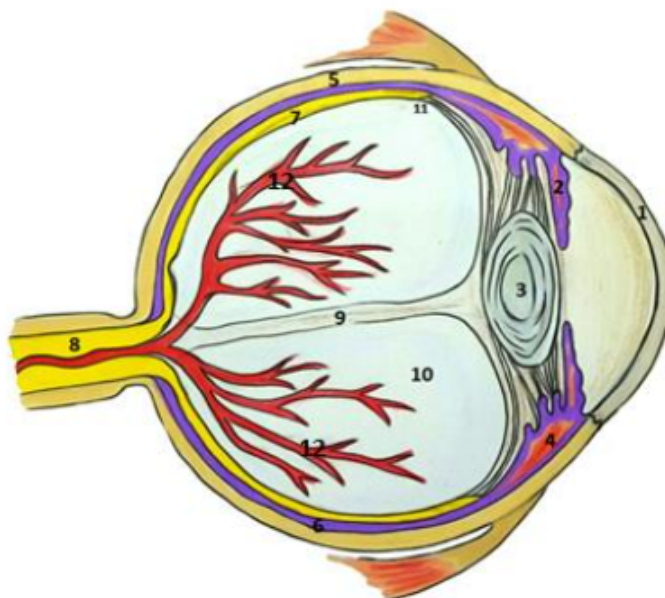


Figura 1.1: 1. Córnea, 2. Iris, 3. Cristalino, 4. Cuerpo ciliar, 5. Esclera, 6. Coroides, 7. Retina, 8. Nervio óptico, 9. Canal hialoideo, 10. Humor vítreo, 11. Ora serrata. [4]

La luz llega a través de la córnea, cruzando la pupila y el cristalino hasta que llega a la retina, desde aquí, a través del nervio óptico, la información es transmitida al cerebro para que se procese y se extraiga la información necesaria [5]. Derivado de lo anterior, es posible obtener la definición de la visión estereoscópica, que es una facultad físico - neurológica que posee el ser humano, que le permite ver en tercera dimensión aquellos objetos que contempla mediante su visión binocular. Sobre cada una de las retinas de ambos ojos, se forma una imagen perspectiva del mismo objeto, las cuales difieren entre sí, debido a la distinta posición de los puntos de vista, produciendo el efecto de relieve. La distancia entre estos dos puntos de vista, o sea, la separación entre los órganos receptores del ser humano, tiene un valor promedio de 65 mm y se la llama “distancia

interpupilar” [6].

Los avances tecnológicos y la necesidad de mejorar la calidad de vida de los seres humanos ha llevado a la industria robótica y las ciencias a investigar y comprender como replicar habilidades humanas con el uso de las diversas tecnologías hoy en día. Es por esto que la creación de las cámaras es un avance tan importante, considerando que emula la visión humana en cierta forma con la capacidad de obtener información bidimensional de un escenario tridimensional. Pero, ¿cómo es posible replicar la visión estereoscópica? Considerando que se necesita de dos órganos oculares para poder recavar información tridimensional como la distancia, es intuitivo pensar que es necesario de dos cámaras para simular esta visión estereoscópica con el uso de máquinas electrónicas. Lo anterior suena sencillo pero actualmente es una materia de estudio que se encuentra vigente y se sigue avanzando buscando mejoras y optimizaciones de los procesos. La tecnología de visión en robots inteligentes es uno de los temas más estudiados en el campo de los robots recientemente, la visión estéreo binocular (visión estereoscópica) tiene un importante valor tanto en la aplicación para robots industriales como para robots móviles. La localización tridimensional es una rama de la visión computacional y su problema principal es cómo reconstruir un escenario 3D a través de imágenes bidimensionales. [7]

Dada la posición de los ojos en los humanos y la forma de moverlos las imágenes que se reciben en cada ojo son prácticamente iguales, con una diferencia en la posición relativa de los objetos. Estas diferencias relativas en la posición en cada imagen (la disparidad), tiene una relación directa con la distancia (profundidad) a la que se encuentran los objetos entre sí, y al observador. El cerebro es capaz de interpretar esa diferencia y reconstruir la estructura de la escena que ve el observador. Existen tres etapas en el proceso de recuperación de la estructura de una escena. Estas son, primero, seleccionar un punto característico de un objeto en una de las imágenes (vistas por cada ojo), segundo, encontrar el mismo punto característico en la otra imagen, y tercero, medir la diferencia relativa (disparidad) entre la posición de estos dos puntos.

El objetivo de un algoritmo estéreo de cálculo de disparidad es obtener la profundidad en todos los píxeles de las imágenes del par estéreo; hallando las parejas de puntos correspondientes que son proyección del mismo punto del espacio, para todos los puntos en cada una de las imágenes. De esta forma se recupera un mapa denso de disparidades. [8]

La disparidad es el parámetro en el cual se ha investigado y creado diversos algoritmos con la finalidad de optimizar este proceso. Las librerías de *OpenCV* incorporan una función para realizarlo y se explica en el desarrollo del proyecto, el algoritmo utilizado es el *SGBM*. El *SGBM* es un algoritmo para la generación de mapas de disparidad densos y suavizados a partir de un par de imágenes rectificadas. Este método requiere una gran cantidad de recursos computacionales y es incluido por la librería de *OpenCV*, debido a la alta calidad en los resultados obtenidos por Heiko Hirschmüller [9]; en donde evidencia la precisión del método llegando a nivel de píxel mediante la optimización de una función de costo global. En donde adicionalmente, se presentan mejoras en el postfiltrado para refinar los resultados obtenidos.

### 1.2.3. Robot móvil

La investigación y desarrollo tecnológico ha ido aumentando de forma exponencial los últimos años buscando mejorar la calidad de vida de las personas y aumentando la productividad de las empresas e industrias. La incorporación de máquinas autónomas, hoy en día, se está masificando; y no es de extrañar que ya realizan labores que antes desempeñaban personas, pero, con más precisión y confiabilidad.

La incorporación de dichas máquinas autónomas se ha creado en su mayoría gracias a la carrera armamentista existente en el mundo, pero esto en la actualidad ya se ha comenzado a expandir a las diversas áreas de trabajo e investigación, como lo es la medicina, ciencias de la computación, construcción, cuidado del medio ambiente, agricultura, entre muchos otros.

Los vehículos con ruedas son la solución más simple y eficiente para conseguir la movilidad en terrenos suficientemente duros y libres de obstáculos, permitiendo conseguir velocidades relativamente altas. [10]

Algunos de los sistemas o configuraciones de robots más comunes dentro de la robótica móvil son los siguientes. [10]

- **Ackerman**

Es el vehículo de cuatro ruedas convencional. La tracción existe en las cuatro ruedas considerando el tipo de terreno o sólo en dos considerando que las ruedas delanteras siempre son las que se encargan de dar dirección al vehículo no tripulado; El mayor problema de la locomoción Ackerman es la limitación en la maniobrabilidad.

- **Triciclo clásico**

Sistema de locomoción de tres ruedas, de las cuales la delantera funciona tanto para la dirección como para la tracción del vehículo, el eje trasero, con dos ruedas laterales, es pasivo y sus ruedas se mueven libremente. La maniobrabilidad aumenta con respecto al Ackerman pero es común que presente problemas de estabilidad. No es apto para terrenos en pendiente ni terrenos complejos. De todas formas, es usado frecuentemente en vehículos robóticos para interiores y exteriores pavimentados.

- **Direccionamiento diferencial**

El direccionamiento viene dado por la diferencia de velocidades de las ruedas laterales. La tracción se consigue también con estas mismas ruedas. Adicionalmente, existen una o más ruedas para dar soporte y estabilidad al robot. Este tipo de robot es escogido por su simplicidad de controlar y buena estabilidad al poseer doble tracción en la zona delantera y un apoyo tipo “rueda loca” que permite seguir el movimiento del eje de tracción sin problemas.

- **Pistas de deslizamiento**

Más conocidos como vehículos tipo oruga en los que tanto la impulsión como el

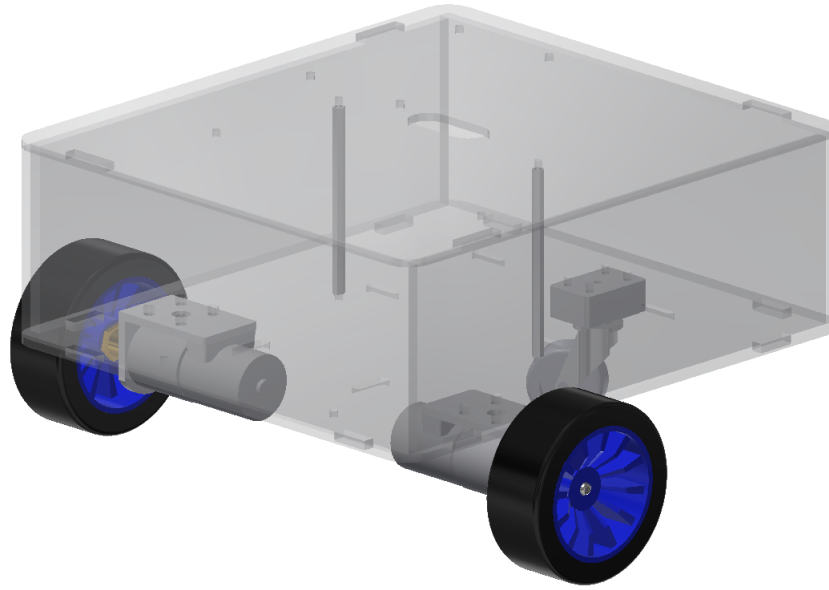


Figura 1.2: Robot tipo diferencial diseñado para el proyecto.

direccionamiento se consiguen mediante pistas de deslizamiento. Es un vehículo en terrenos irregulares.

#### ■ Otras configuraciones

- **Uso de ruedas especiales**  
El uso de ruedas especiales es especialmente para lograr mayor movilidad y maniobrabilidad en configuraciones en donde estas capacidades están limitadas, un ejemplo de esto son las ruedas suecas que son ruedas que consiguen un movimiento omnidireccional del robot.
- **Locomoción mediante patas**  
Permiten aislar el cuerpo del terreno empleando únicamente puntos discretos de soporte.
- **Configuraciones articuladas**  
Especialmente diseñadas para terrenos de difícil desplazamiento, recurriendo a la redundancia para asegurar su funcionamiento en ambientes adversos, añadiendo articulaciones y ruedas al sistema.

## 1.3. Revisión bibliográfica

### 1.3.1. Trabajos previos

En [11] se utiliza la visión por computador como una herramienta muy poderosa utilizada en el proyecto *Mars Rover (Mars Exploration Rover Mission)* con el fin de explorar superficies detectando zonas potencialmente peligrosas antes de transitar el robot móvil por ellas. Esto se realiza mediante un constante muestreo de la zona en donde se encuentra el robot generando un mapa local del terreno usando la visión estereoscópica. Haciendo hincapié en el bajo consumo de energía por parte de la electrónica asociada ala captura de imágenes conciderando los beneficios que entrega, como el conocimiento amplio sobre el entorno.

En [12] se utiliza un sistema de visión estereoscópica para la navegación autónoma de vehículos no tripulados para lo cual se hace uso de MATLAB con ayuda de librerías que permiten implementar algoritmos conocidos de procesamiento digital de imágenes. Además, se implementa inteligencia artificial para la detección y esquivo de obstáculos.

Últimamente, en la industria se ha utilizado la visión por computador para la búsqueda de piezas o artefactos con el fin de mejorar los tiempos de armado y montaje de máquinas y productos que crean o ensamblan varias empresas, como por ejemplo la industria automovilística. La detección de obstáculos es crucial para los asistentes de manejo de los vehículos autónomos existentes hoy en día, evitando así colisiones. Sin embargo, la factibilidad de estos sistemas en automóviles que transportan pasajeros requieren de un preciso y robusto sistema de detección. Es aquí donde entra un sistema de detección de obstáculos que usa la visión estéreo. Es por esto que se desarrolla un sistema de detección de obstáculos basado en la visión estéreo para la circunstancia específica de viaje en carretera. En donde se hizo uso de algoritmos que detectan esquinas y correspondencia estéreo basada en las características, haciendo uso de otro algoritmo de correspondencia llamado Sampson que tiene la particularidad de no necesitar el par correspondiente para comprobar la restricción epipolar. Solucionando los problemas de captura instantánea de fotografías durante el viaje. Llegando incluso a detectar cuando automóviles se cuelan en el carril. Estas pruebas experimentales fueron realizadas en autopistas en el país de Corea. [13]

En [14] se construye sistema de visión estéreo montado sobre una placa *FPGA* utilizando la transformada “census”, la cual provee información de disparidad densa en tiempo real. También se implementa considerando etapas de pre y post procesado de las imágenes. Para lo cual existe una alta optimización de procesos para la recopilación de información tridimensional. Realizando la activación y captura de imágenes simultáneas gracias a la posibilidad de montar el sistema sobre una *FPGA*.

El creciente interés de la investigación por los sistemas mecánicos a pequeña escala ha generado una necesidad de sistemas de medición de deformaciones y tensión para aplicaciones a microescala. Los sistemas de medición óptica, como la correlación digital de imágenes, constituyen una opción obvia debido a su naturaleza sin contacto. Sin embargo, la transferencia de la tecnología de medición desarrollada para aplicaciones a macroescala a la microescala presenta desafíos únicos debido a las diferencias en la

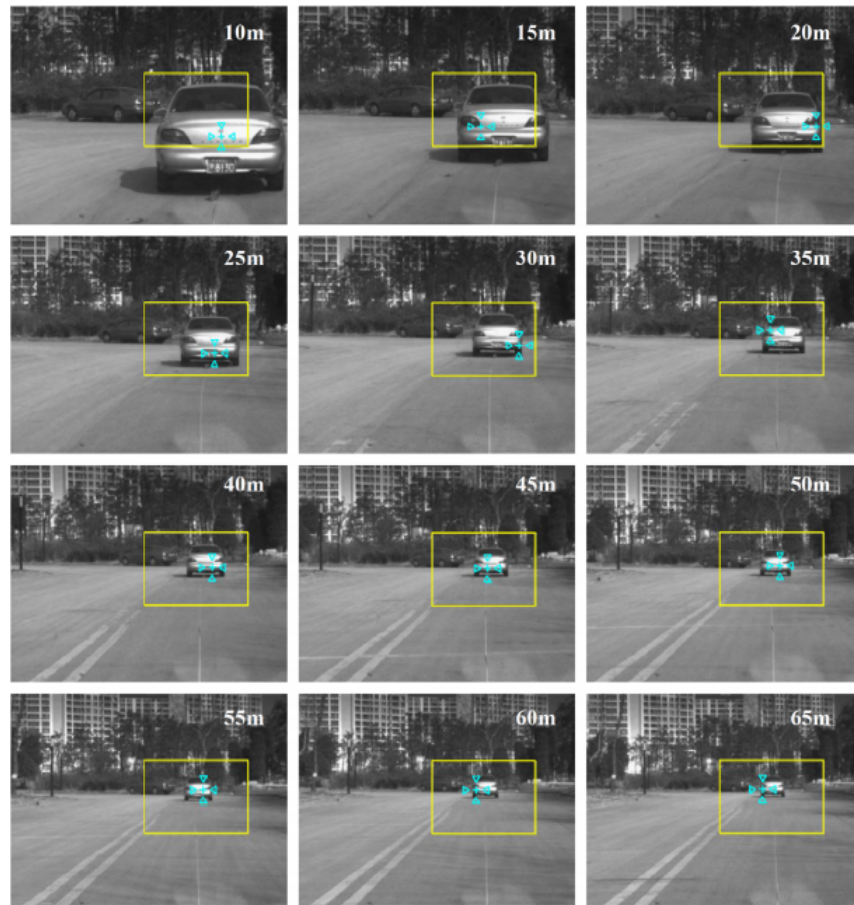


Figura 1.3: Resultados experimentales de la implementación de visión estéreo en la navegación autónoma de vehículos tripulados. Experimento de medición de distancia. Extraído de [13].

óptica de alto aumento requerida. Es por eso que en [15] se implementan técnicas de calibración manual estereoscópica y técnicas para adquirir deformaciones tridimensionales en muestras a pequeña escala. Logrando incluso a comprobar que es posible calibrar sistemas de visión estéreo en la microscopía de forma precisa y fiable usando técnicas de distorsión. Comprobando que la tecnología y la aplicación del par estéreo mejora y se estudia cada día más para la constante actualización de sistemas en variadas temáticas de la investigación pasando de herramientas o métodos manuales a métodos computarizados y automatizados.

En [16] se aborda la implementación de un sistema de visión artificial que sirve de ayuda para la clasificación de granos de cacao según características externas como el tamaño en la fase final del secado. En este caso se hizo uso de dos cámaras, de las cuales una está posicionada en la parte superior captando el largo y ancho del grano de cacao y otra en el costado de este para la detección de achatamiento. Además, se hace uso de las imágenes para detectar si los granos están pegados y tienen buen color. De este documento se pudo llegar a la conclusión de que se ha podido hacer uso de *OpenCV* para la extracción de características físicas trabajando con píxeles y diversas perspectivas de un objeto en un entorno controlado haciendo correcto uso de iluminación según se



requiera. La clasificación de los granos se realiza mediante procesamiento digital de imágenes pero la obtención de distancia para estimar el tamaño se realiza calibrando de forma estática un espacio controlado en donde se ubican los granos para su estudio.

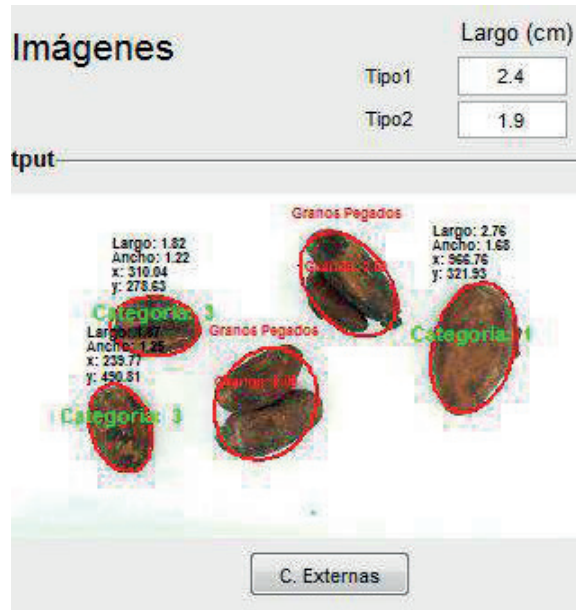


Figura 1.4: Vista ampliada de definición de valores límites en interfaz gráfica para la clasificación según tamaño. [16]

En [17] tratan el procesamiento de imagen en conjunto de una red neuronal para realizar la clasificación de productos frutales en cuanto a calidad de refiere, clasificándolos en Calidad, EXTRA, I, II, III y IV, teniendo como objetivo de estudio una inspección visual que da a conocer mediante imperfecciones y manchas en la fruta el tipo de categoría a la que pertenece siendo EXTRA, la mejor y IV, la peor. El proceso realizado para lograr el objetivo principal es obteniendo una fotografía de la fruta, para luego mejorar la imagen y realizar una segmentación y extracción de características para poder lograr un reconocimiento de patrones que pueden indicar la clase a la que pertenece la fruta. En consecuencia de lo realizado, en este artículo se exponen los resultados obtenidos dando a conocer que se acertó en más del 90 % de las pruebas, no obstante, aunque en una fruta se pueden realizar múltiples mediciones para la determinación de su calidad, “no es posible determinar todos los parámetros por medio de inspección visual, como el aroma, la textura de la pulpa, la dureza y el contenido de defectos internos y de azúcares, por lo que un trabajo como este representa una solución importante pero no representa la solución completa.” De lo anterior es importante recalcar que la inspección visual es la menos invasiva si se considera que sólo se debe transportar para clasificar sin alterar o dañar la fruta, trabajo que cada vez se ve más trabajado para pulir los criterios de detección y mejorar la calidad de los productos.

Una aplicación industrial En la industria alimentaria es el uso de una maquinaria desarrollada por la empresa *Raytec Vision* que se encarga de descartar y apartar tomates que no se encuentran en su fase de madurez completa (“verdes”), esta selección la realiza a una alta velocidad con una captura constante de imágenes mientras los tomates son

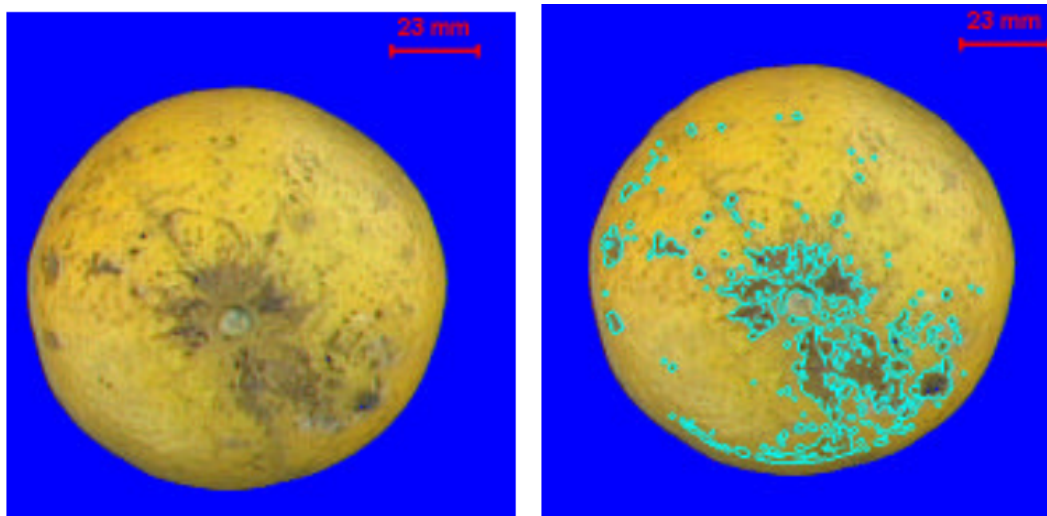


Figura 1.5: Resultado obtenido para la detección y cuantificación de defectos. [17]

transportados luego de su descarga ascendiendo a una plataforma en la cual pronto son arrojados a un riel que mueve con un flujo constante de agua hacia el interior de la planta. Para la detección de color se utiliza una gran cantidad de cámaras que detectan el color y guardan la posición en la que se están desplazando. Luego, mediante técnicas de control se activan unos dedos robóticos utilizando algoritmos de predicción que se encargan de empujar y descartar aquellos tomates que carecen de madurez. Lo anterior es extraído de una experiencia personal realizando la práctica técnico profesional en el año 2015 en la empresa *Agrozzi*, durante esta práctica se entablaron conversaciones con personal técnico de la empresa *Raytec Vision* quienes realizaban las mantenciones de las maquinarias en *Agrozzi*.

## 1.4. Discusión

La robótica a través de los años ha ido avanzando a pasos agigantados y aún así la investigación acerca de la visión estéreo y robots móviles son materia de estudio que sigue vigente y en constante actualización.

Si bien existen diversos métodos para captar la disparidad de entre imágenes, se opta por utilizar la librería *OpenCV*; en donde los usos de esta librería facilitan la comprensión, además de estar bastante documentado y con buenos resultados, utilizando el algoritmo *SGBM*; el uso de dos cámaras asegura la obtención de información tridimensional con el uso de algoritmos antes mencionados, y dependiendo del problema se requiere de una o más cámaras, ya que como queda de manifiesto en [16] se establece que es posible obtener la distancia de un observador al objetivo con sólo una cámara, sí y solo sí, el entorno no varía y se realiza una calibración de la distancia que existe desde el fondo o suelo del lugar de pruebas hacia la cámara que permanece estática. En el presente caso se pretende realizar un robot móvil lo que añade al problema una variable de desplazamiento, por ende, es mucho más simple realizar las mediciones con un par estéreo que con sólo una cámara, porque el ambiente cambiará y no existirán referencias de distancia a las cuales se pueda configurar.

La implementación y desarrollo de procesamiento digital de imágenes es viable y factible a la hora de extraer ciertas características, en el caso de este proyecto se busca sólo buscar objetos específicos (diámetro del tallo de una planta) dentro de una imagen. Lo que sí se hace necesario, es la implementación de un sistema estéreo para poder dar la profundidad de la imagen y determinar la distancia de la medición al objetivo.

El estudio y la toma de imágenes en las investigaciones y trabajos realizados enfocados en la agricultura están principalmente ligados a la mejora de rendimiento, calidad y conservación de frutos o productos finales a partir de una planta a la cual se le aplican diversos tratamientos y cuidados que influyen en el resultado final del fruto. El estudio de un fruto puede dar cuenta de lo bien o mal que se ha implementado un plan de riego o mantenimiento de un vivero, también puede dar entregar información sobre los efectos que tienen los sistemas de conservación sobre el fruto. Además, es posible llegar a conclusiones según características visuales a simple vista como el estado de madurez que va directamente enlazado a la idea de mejora de calidad de un producto. Pero pocas veces se habla sobre el producto que se extrae de un vivero que en este caso son arboles frutales, es por esto que los trabajos previos estudiados son acerca de uso de procesamiento digital de imágenes en la agricultura.

Por otro lado, el movimiento de un robot dentro de un ambiente es material de estudio reciente que busca automatizar y desligar al humano de la conducción de diversas máquinas móviles, ya sean voladores, terrestres o acuáticas. Esto lo realizan considerando diversas técnicas de evasión de obstáculos, pero para este proyecto se estima conveniente proponer una solución más simple en dónde se busca que el robot realice una ruta tomando mediciones de calibre en el trayecto siguiendo una ruta conocida y demarcada por una línea en el suelo para efectos de estudiar la viabilidad dentro de un ambiente de laboratorio.

## 1.5. Hipótesis

El sistema de visión estereoscópica puede ser aplicado para la medición de calibre de objetos bajo condiciones especiales de laboratorio.

## 1.6. Objetivos

### 1.6.1. Objetivo general

- Diseñar un sistema capaz de identificar el calibre de viveros frutales y clasificarlos según diámetro o perímetro, montado en una plataforma de clasificación móvil.

### 1.6.2. Objetivos específicos

- Investigar sobre la viabilidad de la extracción de características a partir de imágenes digitales tomadas con cámaras web.
- Definir *softwares* a utilizar para el desarrollo.
- Calcular el diámetro del tallo de una planta mediante visión estéreo, haciendo uso de *Python* y *OpenCV*.
- Estimar la distancia de una planta a las cámaras mediante visión estereoscópica con *Python* y *OpenCV*.
- Justificar que la realización de este proyecto es viable y ejecutable.
- Diseñar un sistema de clasificación de plantas según diámetro.
- Diseñar sistema de transporte y clasificación.
- Diseñar conexionado eléctrico entre componentes necesarios para el funcionamiento del robot
- Diseñar un sistema unificado de clasificación móvil con un sistema de visión estéreo.

## 1.7. Alcances del proyecto

- El robot tendrá la capacidad de seguir una ruta demarcada en el piso para el desplazamiento dentro de un laboratorio.
- Se estima una autonomía de 30 minutos de funcionamiento del robot.
- El proyecto consta sólo del diseño del sistema presentado, realizando sólo pruebas en el área de la visión estéreo.
- las simulaciones se realizarán en lenguaje *Python* utilizando *Visula Studio Code* para escribir el código y realizar las pruebas de visión estéreo.
- Sólo se utilizan herramientas de visión estéreo y robótica, no se busca probar o explorar nuevas técnicas, sino implementar aquellas que obtengan buenos resultados para el diseño de una solución a la problemática presentada.
- Se consideran condiciones ideales de luminosidad y terreno para pruebas e implementación futura en un ambiente controlado.
- Los resultados obtenidos se presentan gráficamente, pero en la práctica estos resultados no son visibles más allá de la clasificación realizada. Estos resultados gráficos expuestos son netamente para mostrar evidencias dentro del documento.
- Se realiza implementación de algoritmos en un ordenador de placa reducida.
- Se consideran bajas velocidades de desplazamiento para correcta toma imágenes.
- El análisis que se realiza es de un sólo objetivo independiente de la cantidad de objetos presentes en la fotografía.
- Se establece la comunicación serial como técnica de comunicación entre *Raspberry Pi 4* y *Arduino UNO*.

## 1.8. Metodología

Es necesario especificar las habilidades y competencias adicionales requeridas que ayudan para el buen desarrollo de este proyecto. Se requiere dominio básico/intermedio en el área del diseño y uso de creación de objetos 3D en algún *software* como *inventor* o *solidworks*, un dominio en la programación en lenguaje *Python* con librerías de *OpenCV*.

Para el diseño del robot equipado con la visión estereoscópica es necesario realizar una serie de actividades que marcan el desarrollo correcto y sostenido del proyecto. Las actividades necesarias se presentan a continuación.

- Estudio teórico sobre visión estéreo.

Se revisará la bibliografía y diversos métodos para extraer las características a partir de una imagen. Conjuntamente se recopila información sobre uso de visión estéreo y la correspondencia entre imágenes. En este apartado se ha hecho uso de buscadores de información académica como *Google Scholar* y *IEEE-Xplore*.

- Implementación de algoritmos de prueba sobre imágenes.

Se buscarán las mejores alternativas para el uso de un lenguaje de programación y un *IDE* para la implementación de algoritmos que permita una simulación y extracción de información de forma cómoda y eficaz.

- Estudio sobre robots móviles.

En este apartado se hará la recopilación de información sobre distribución de energía en un robot, en conjunto de su dinámica y estrategia de control para su uso ya sea autónomo o guiado.

- Diseño de un robot móvil para transporte de sistema de visión estéreo.

Se diseñará un prototipo capaz de cumplir con los requisitos mínimos para el desarrollo y recopilación de datos experimentales, como también se realizará el estudio de resistencia y selección de componentes mecánicos que sean capaces de cumplir con los requerimientos del proyecto.

- Obtención de resultados experimentales.

Se recopilarán los resultados obtenidos y se dará un veredicto de la viabilidad de implementación de visión estéreo para la realización de las tareas propuestas.

- Evaluar posibles mejoras en el trabajo realizado.

## 2. Aplicación de visión estéreo para medición de calibre

### 2.1. Definición de lenguaje de programación a utilizar

En plano de investigación del proyecto, se llegó a simular y probar como funcionaría la detección del diámetro de diversos objetos como tubos, fundas y lápices en una unidad de píxel, luego de haber realizado investigaciones y distintas pruebas con diversas formas de procesar imágenes; desde el modelado 3D, hasta el estudio de imágenes mediante el procesamiento digital de imágenes.

Al comienzo de toda la investigación, gracias a una reunión con el profesor, académico de la Universidad de Talca de la escuela de Ingeniería Civil en Computación, Matthew Bardeen, se decidió comenzar por la investigación de fotogrametría y modelado 3D, en donde se debió justificar el por qué no se usaría dicha tecnología para este proyecto. La fotogrametría es una técnica de medición de coordenadas 3D, que utiliza fotografías o vídeos mediante una o más cámaras, como medio fundamental para la medición. En esta ocasión se intentó trabajar con modelado 3D para crear un objeto y extraer sus medidas. El problema de esto es que el algoritmo demanda el uso demasiados recursos computacionales, se debían tomar demasiadas fotografías para obtener un modelado preciso, además de que aún no se sabía como determinar la distancia entre la cámara y el objetivo. Es por esto que para poder descartar esta opción se realizaron pruebas de modelado 3D con el *software: Regard 3D*, que a partir de fotografías tomadas desde diversos ángulos, con solo una cámara, al objeto a medir, se les realiza una correspondencia de puntos dentro de cada imagen para generar una vista 3D. El resultado se puede ver en la figura 2.1.

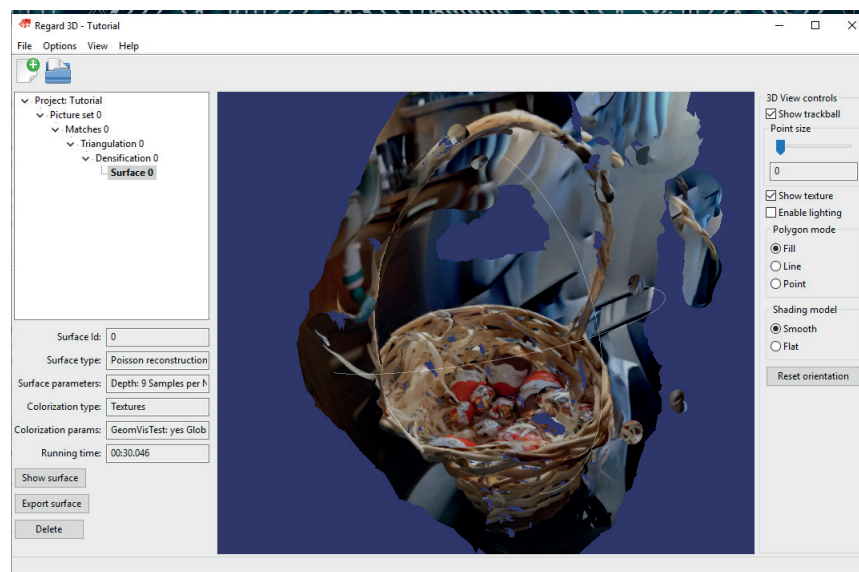


Figura 2.1: Resultado obtenido de un modelado 3D con Regard 3D.

A partir de los resultados obtenidos en la figura 2.1, se ha decidido que es muy difícil obtener los parámetros deseados desde un modelado 3D, es por esto que se ha

optado por la investigación de otros métodos de estudio que solucionen estos problemas. Muchas inquietudes y preguntas surgieron respecto a la complejidad de trabajar con *softwares Open Source* de modelado 3D.

Es llegado a este punto, que se comienza con la investigación un poco más en profundidad sobre la visión estereoscópica y sus aplicaciones. La visión estereoscópica se encarga de simular la visión humana haciendo uso de dos cámaras separadas a una distancia conocida para obtener información tridimensional a través del procesamiento y manipulación digital de imágenes. La información tridimensional que más interesa en el presente proyecto es la distancia de las cámaras al objeto de interés.

Para lo anterior, se realizó la investigación correspondiente llegando a encontrar *OpenCV* que es una librería *open source* para el procesamiento de imagen y visión por computadora, originalmente desarrollado por *Intel*, que luego fue respaldada por *Willow Garage* y ahora es mantenida por *Itseez*. [18]

*OpenCV* en conjunto con C++ ha sido probada para el objetivo planteado por esta investigación. Haciendo uso de tutoriales y documentación se pudieron realizar las primeras manipulaciones de imagen para poder facilitar la extracción de diámetro de un objeto.

En el Código 5.1, que se encuentra en la Sección Simulaciones, se ve cómo se desarrolló una prueba con un programa de prueba originalmente desarrollado por *OpenCV* y modificado para su correcta funcionalidad. Debido a la antigüedad y poca cantidad de documentación, en conjunto de comandos obsoletos o parámetros que ya no se utilizaban, se planteó la siguiente pregunta, ¿se había escogido de forma correcta el lenguaje en el cual se implementaba el proyecto en esta etapa de prototipado e investigación?. Con el anterior código mencionado se pudo establecer una idea de como es que se planeaba seguir con el trabajo. Los resultados de esta experiencia se evidencian en la figura 2.2.

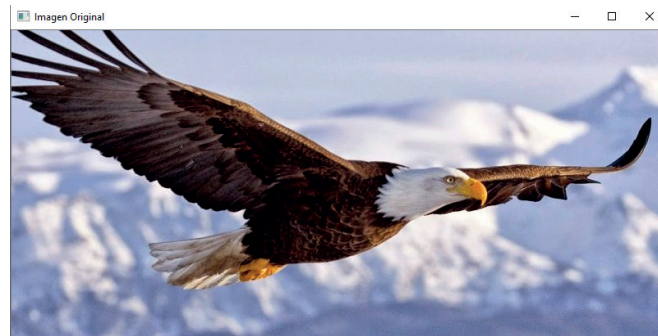
Al seguir desarrollando el Código 5.1 se encontraron bastantes errores de librería y poca documentación al respecto. Por lo que respondiendo a la pregunta planteada en el párrafo anterior, se decidió cambiar el lenguaje de programación por otro que tuviese un poco más de documentación y respaldo con respecto a funciones para el tratamiento de imágenes.

## 2.2. Cálculo y obtención del ancho de un objeto a través de una imagen

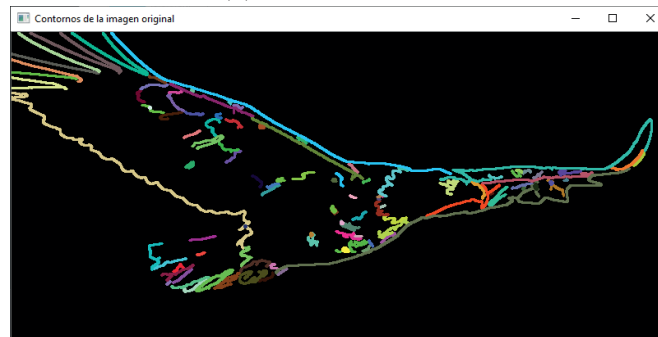
Debido a la cantidad de documentación e información sobre la librería *OpenCV* utilizada en *Python*, se tomó la decisión de utilizar dicho lenguaje de programación para el desarrollo del proyecto.

Para la segunda iteración del proyecto creando algoritmos que reconozcan y midan o extraigan ciertas características de un objetivo, es necesario disponer de un ordenador, cámaras web y un objetivo. Las características del ordenador se pueden revisar en el Anexo A.





(a) Imagen Original.



(b) Bordes encontrados de la imagen original.

Figura 2.2: Resultado de la ejecución del Código 5.1.

Para comenzar con la creación de los algoritmos es necesario disponer de un entorno de desarrollo integrado (o por sus siglas en inglés IDE). Un IDE consiste de un editor de código fuente, un depurador, entre otros componentes. Para este caso se hace uso de *Visual Studio Code* la cual es una herramienta gratuita entregada y desarrollada por *Microsoft*, esta permite instalar diversas extensiones útiles que sirven para el desarrollo de diversos objetivos, como por ejemplo la instalación de Python 3.7.3 que en sus especificaciones indica que contiene *intellisense* para el autocompletado de comandos y revisión de sintaxis, *debugging* para correr y depurar el código línea por línea, entre otros. Por otro lado, para trabajar con la librería *OpenCV* es necesario realizar una instalación previa según se indica en [19].

Una vez se tiene todo preparado se procede con la creación del código que permite calcular el ancho de un objeto dentro de una fotografía, para esto se realiza el Código 5.1. A continuación, en la figura 2.3 se muestra un diagrama que explica brevemente el proceso de la creación del código que realiza el cálculo de ancho o diámetro.

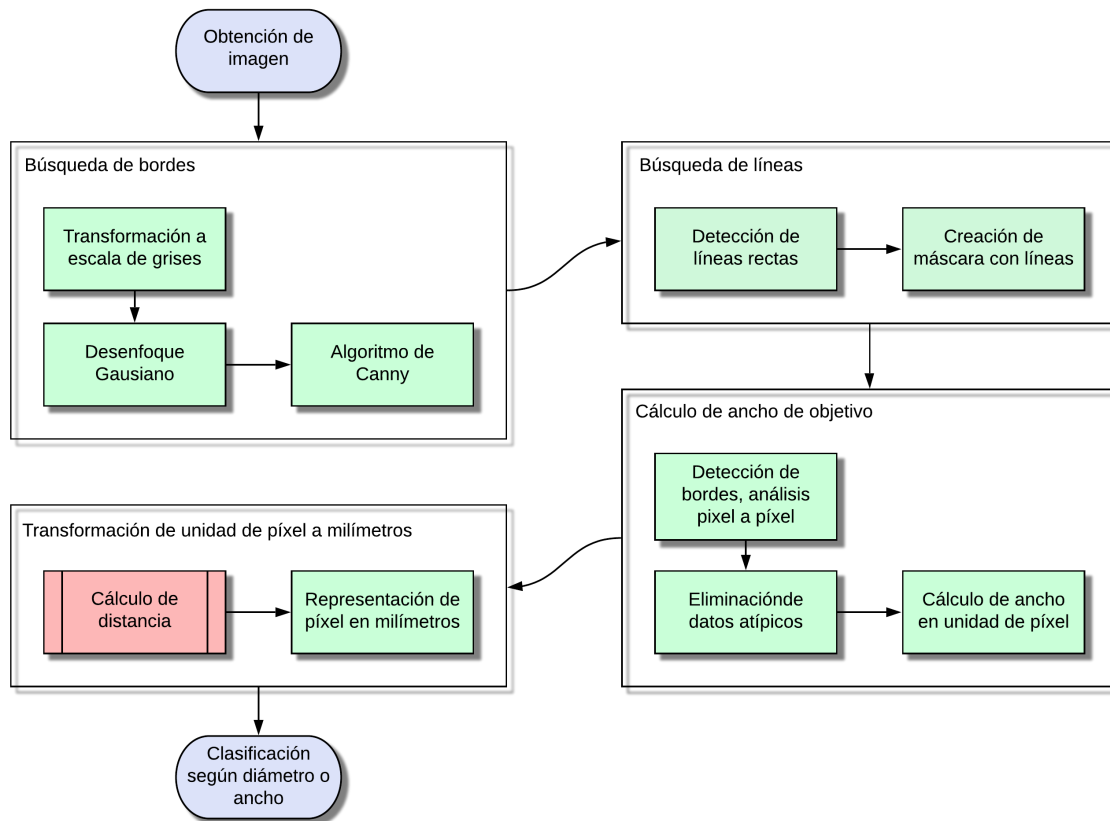


Figura 2.3: Diagrama de flujo de los procedimientos realizados para el cálculo de ancho o diámetro de un tronco.

Describiendo el procedimiento que se sigue según el diagrama de la figura 2.3, se obtiene el resultado del procesamiento digital de una imagen tomada con el objetivo de detectar el ancho o diámetro de un objeto.

### 2.2.1. Obtención de imagen

#### ■ Captura de imagen

Consta de la toma de fotografías de los objetivos de estudio con el uso de *webcams* de acceso comercial.

#### ■ Copia de imagen

Esto se hace para trabajar sobre la copia de imagen en donde se harán modificaciones, para no alterar la original y guardarla en caso de pérdida de datos para un re-estudio.

### 2.2.2. Búsqueda de bordes

#### ■ Transformación a escala de grises

La copia de imagen se modifica de RGB a escala de grises para obtener un mejor contraste entre la diferencia de colores para la correcta detección de bordes.

## ■ Desenfoque Gausiano

Este desenfoque elimina los detalles minúsculos que se encuentran en los bordes de las figuras, con el fin de obtener cambios más uniformes que facilitan la detección de bordes. Se realiza a través de una convolución entre la imagen a tratar y un kernel gaussiano en el cual solo se define el tamaño. Dentro de los tamaños más comunes están  $(3 \times 3)$ ,  $(5 \times 5)$  y  $(7 \times 7)$ . El más utilizado y que obtiene mejores resultados de suavizado sin distorcionar tanto los contornos de una imagen es el kernel de  $(5 \times 5)$ .

Un kernel es una máscara definida por una matriz cuadrada, esta matriz se considera una máscara de  $(n \times n)$  píxeles, donde  $n$  es un número impar (con el fin de tener un píxel central, el cual es analizado) esta máscara se utiliza recorriendo toda la fotografía para difuminarla y así eliminar los ruidos asociados a los componentes electrónicos por falta de luminosidad. Este filtro gaussiano utiliza una máscara modelizando la función gaussiana que se ve en la ecuación 2.1.

$$G(x, y) = \frac{1}{2 \cdot \pi \sigma^2} \cdot e^{-\frac{x^2 + y^2}{2 \cdot \sigma^2}} \quad (2.1)$$

El uso de kernel o núcleos se realiza para el análisis o reconocimiento de patrones y estudio de sus relaciones, esto a través del análisis aumentando las dimensiones de una imagen. La forma más fácil de explicar como funciona un kernel es con un ejemplo sencillo de 2 a 3 dimensiones y con pocos puntos. En la figura 2.4, se puede observar una nube de puntos dibujados en el mantel de una mesa (1), a primera vista es imposible tender una línea para separar ambos colores, pero si se toman los bordes del mantel y se elevan (2), los puntos verdes se separan. Los puntos verdes ya se encuentran en un plano distinto y separados (3). El plano (3) intersecta el espacio de 2 dimensiones de (1) en una curva (4). [20]

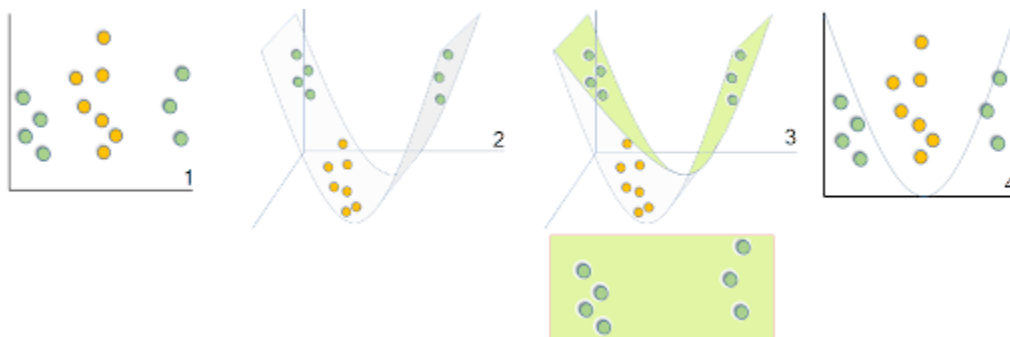


Figura 2.4: Tránsito de un kernel. Extraído de [20].

De esta forma el desenfoque Gausiano logra un efecto de difuminación y elimina puntos o detalles no relevantes en la imagen sin deformarla, gracias a la correcta elección del kernel.

## ■ Algoritmo de Canny para la detección de bordes

Este algoritmo toma en consideración que los bordes se pueden encontrar en cualquier dirección, es por esto, que se usan cuatro filtros que son capaces de detectar

cambios de color en direcciones verticales, horizontales y diagonales en la imagen con el *desenfoque Gaussiano* aplicado.

El procedimiento que rige dicho algoritmo responde a que en una primera etapa se aplica un filtro gaussiano (ya aplicado anteriormente). Posterior, se buscan gradientes de intensidad dentro de la imagen con el objetivo de detectar la continuidad de un borde en distintas direcciones ya mencionadas en el párrafo anterior. que cuenta con un operador de detección de bordes, que incluye variedad de métodos matemáticos que se encargan de identificar los puntos en una imagen digital en donde cambia bruscamente el brillo. Estos cambios bruscos de brillo se denominan bordes. Dicho operador entrega un valor para la primera derivada en la dirección horizontal ( $G_x$ ) y la dirección vertical ( $G_y$ ). A partir de esto, se puede determinar el gradiente y la dirección del borde, a través de la ecuación 2.2 y la ecuación 2.3.

$$G = \sqrt{G_x^2 + G_y^2} \quad (2.2)$$

$$\Theta = \text{atan2}(G_y, G_x) \quad (2.3)$$

En donde  $G$ , puede ser calculado como la hipotenusa de de un triángulo rectángulo de lados  $G_y$  y  $G_x$ . Mientras que el ángulo  $\Theta$  es el resultado de la función  $\text{atan2}$  que es el arco tangente de dos argumentos, este ángulo  $\Theta$ , es redondeado a uno de los cuatro ángulos (vertical, horizontal y diagonales).

Luego, se aplica una supresión no máxima para la eliminación de respuestas falsas en la detección de bordes. Esta es una técnica de reducción de bordes, la cual busca suprimir aquellos píxeles que (según en la fuerza del gradiente) no posean píxeles contiguos con el suficiente brillo o fuerza para ser considerados un borde, por otro lado también es útil para verificar según el gradiente en que dirección se denota el borde. En simples palabras elimina aquellos píxeles que no presentan un valor máximo por zona analizada según el valor de la gradiente.

Dentro del mismo algoritmo *Canny*, existe una etapa que se denomina doble umbral en donde es otra instancia para eliminar aquellos píxeles restantes que son originados por el ruido y las variaciones de color que no pudieron ser excluidos por el desenfoque Gaussiano ni la supresión no máxima. Para esto se seleccionan umbrales, uno bajo y otro alto, entonces si el valor del gradiente del píxel es mayor que el umbral alto, se dice que el píxel es un borde fuerte. Si el gradiente del píxel es menor que el umbral alto pero mayor que el umbral bajo se dice que es un borde débil. Y si el gradiente del píxel es un valor menor al umbral bajo, el píxel en cuestión es suprimido. Estos píxeles son denotados de forma visible dentro de una máscara, que es una imagen en este caso de fondo negro para colocar en ella los bordes encontrados y visualizarlos de mejor manera, posteriormente esta máscara se puede superponer a la imagen original para verificar la calidad de los bordes encontrados.

Finalmente, el algoritmo *Canny* posee una última etapa denominada seguimiento

de bordes por histéresis, la cual hace uso del análisis de blob, el cual consiste en el etiquetado y análisis de componentes conectados. Para este caso, se hace uso para tomar la decisión de qué píxel de borde débil es un aporte para la imagen. Se realiza el siguiente estudio mediante el análisis de vecindarios en los píxeles de borde débil. Si dentro de su vecindario (ocho píxeles vecinos conectados, formando una especie de cuadro de  $3 \times 3$  píxeles) existe un píxel de borde fuerte, este píxel de borde débil es conservado. En el caso contrario de no poseer un píxel vecino fuerte, este es eliminado.

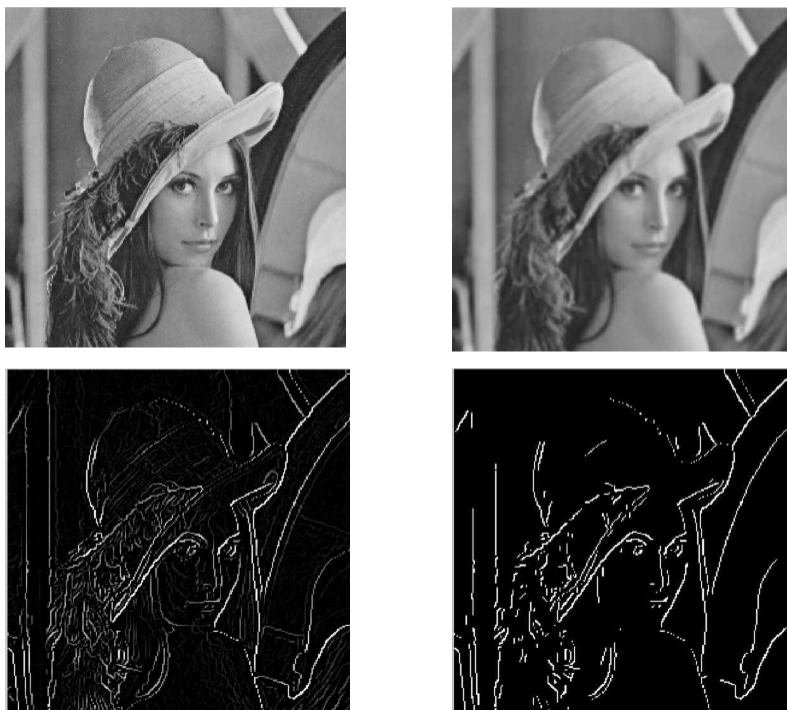


Figura 2.5: Ejemplo imagen con aplicación de algoritmo de Canny. Extraído de [21].

En la figura 2.5, la imagen superior izquierda muestra la imagen original de Lena. La figura superior derecha muestra la imagen de Lena después de haber pasado un filtro gaussiano. La figura inferior izquierda muestra la imagen de Lena después de haber pasado por la supresión de no máximos. La figura inferior derecha muestra la imagen de Lena después del proceso de histéresis, y en los que se usa umbrales de 100 y 130. [21]

Se puede visualizar al final que la función *Canny* recibe como argumentos, la imagen post-desenfoco Gaussiano y los parámetros de umbral alto y bajo para determinar cual es la exactitud que se quiere para la creación de los bordes.

### 2.2.3. Búsqueda de líneas rectas

#### ■ Detección de líneas rectas con *houghlinesP*

Dicha función se encarga de buscar píxeles contiguos que formen líneas, los parámetros se pueden modificar con el fin de obtener resultados personalizados dependiendo de los requerimientos. Se buscan píxeles contiguos que forman mínimo líneas de cierto

número de píxeles, también se puede configurar el ángulo de las líneas que se desean encontrar con respecto al eje horizontal. Además, posee la opción de determinar cuanta separación máxima es la aceptada entre píxeles para considerarlos contiguos. Todos estos datos son guardados en una matriz con la información de las líneas encontradas.

En particular, *HoughlinesP* es la transformada de Hough, una herramienta que permite detectar curvas en una imagen. Es una técnica muy robusta frente al ruido y a la existencia de huecos en la frontera del objeto. A la hora de aplicar la transformada de Hough a una imagen es necesario obtener primero una imagen binaria de los píxeles que forman parte de la frontera del objeto. [22]

Para comprender un poco mejor como trabaja la función se presenta un ejemplo simplificado de aplicación. El objetivo de la transformada de Hough es encontrar puntos alineados que puedan existir dentro de una imagen, es decir, puntos en la imagen que satisfagan la ecuación de la recta, para distintos valores de  $\rho$  y  $\theta$ .

Ecuación de la recta en forma polar:  $\rho = x \cdot \cos\theta + y \cdot \sin\theta$ . Por tanto, hay que realizar una transformación entre el plano imagen (coordenadas  $x$ - $y$ ) y el plano o espacio de parámetros  $(\rho, \theta)$ .

Para aplicar la transformada de Hough es necesario discretizar el espacio de parámetros en una serie de celdas denominadas celdas de acumulación. Esta discretización se realiza sobre los intervalos  $(\rho_{min}, \rho_{max})$  y  $(\theta_{min}, \theta_{max})$ .

El siguiente paso es evaluar la ecuación de la recta para cada punto de la imagen  $(x_k, y_k)$ , si se cumple esta ecuación se incrementa en uno el número de votos de la celda. Un número de votos elevado indica que el punto pertenece a la recta. [22]

Para la función utilizada es necesario entregar como argumento la imagen de estudio (en este caso los arreglos de información que contienen los píxeles encontrados con la función *Canny*),  $\rho$ ,  $\theta$ , un valor mínimo de píxeles para considerar una línea recta y un valor máximo de separación para considerar un siguiente píxel parte de una línea recta.

#### ■ Creación de máscara con las líneas rectas encontradas

Para realizar la máscara se crea una imagen en negro y se insertan los datos de píxeles color blanco. Anteriormente, se habló de una máscara y es en este punto que luego de aplicados todos los filtros para obtener un correcto resultado de líneas rectas que se crea una máscara que contiene la información depurada.

El resultado de la búsqueda de bordes rectos se puede visualizar en la figura 5.3, en donde posterior a una búsqueda de bordes con el algoritmo de Canny se filtra para poder encontrar los bordes rectos de interés para el estudio del objetivo.

#### 2.2.4. Cálculo de ancho o diámetro en píxeles

##### ■ Búsqueda de pares de píxeles blancos en cada fila

Esto hace referencia al análisis de la máscara creada fila por fila, agrupando en pares los píxeles encontrados por fila.

#### ■ Eliminación de datos atípicos y píxeles contiguos

La eliminación de píxeles contiguos se hace dentro del proceso de búsqueda de pares, en donde fila por fila se realiza la resta de los valores de posición en coordenadas de píxel, en donde, si el resultado es distinto de uno se guardan las posiciones de los píxeles como objetos de estudio. Por otro lado, la eliminación de datos atípicos se realiza mediante la aplicación de un filtro de desviación estándar a los arreglos que guardan los datos por separado de la línea recta izquierda que describe al objetivo dentro de la imagen y otro arreglo que guarda las ubicaciones de los píxeles de la línea recta derecha que describe al objetivo.

#### ■ Determinación de ancho en unidad de píxel

En este punto se toman los arreglos que ya han sido procesados por la función que aplica la desviación estándar y se aplica la operación promedio en cada arreglo para realizar la operación de resta de los promedios y se obtiene la medida de ancho o diámetro del objeto de estudio en unidad de píxel.

### 2.2.5. Transformación de píxeles a milímetros

#### ■ Obtención de profundidad

Como se menciona anteriormente se realiza un ingreso de distancia, esta distancia se obtiene mediante el uso de algoritmos estudiados más adelante para la obtención de profundidad mediante visión estereoscópica. Véase en la sección 2.3.5.

#### ■ Obtención de función mediante pruebas con cámara a utilizar

Para esto se toman fotografías de un mismo objeto a diversas distancias conocidas, estas se procesan para encontrar el ancho en cantidad de píxeles, para luego aproximar una función polinómica capaz de transformar esa cantidad de píxeles a milímetros. Para esto se hace necesario dar la distancia que existe desde el objeto de estudio a la cámara.

#### ■ Representación de un píxel en milímetros

Se aproxima otra curva de comportamiento esta vez comparando la disparidad existente de las imágenes obtenido con la función polinómica obtenida en el punto anterior y la distancia al objetivo, conociendo tanto el ancho o diámetro del objetivo y la distancia de medida. Estas funciones son capaces de describir y obtener resultados muy aproximados a los reales, gracias a los métodos de recolección de información. Esta función en particular entrega como resultado la representación de un píxel en milímetros a la distancia requerida.

### ■ Impresión de imagen original y máscara

Para finalizar el proceso, se hace una impresión en pantalla de la imagen original y la máscara creada superpuesta a la original.

## 2.3. Cálculo de profundidad mediante visión estéreo

En este apartado se expone la teoría necesaria para comprender el funcionamiento del algoritmo de trabajo para el cálculo de profundidad, explicando mediante etapas secuenciales el cómo se va aplicando la visión estéreo al proceso de medición de distancia. Comenzando con la calibración de las cámaras como par estéreo y finalizando con la creación y análisis de mapas de disparidad para el cálculo de distancia.

### 2.3.1. Geometría del sistema

En la figura 2.6, el plano formado por la unión del punto  $P$  con  $O_I$  y  $O_D$  se denota como plano epipolar. La proyección de la línea del punto focal  $O_I$  al punto  $P$  sobre el plano de la imagen derecha (línea en rojo) se conoce como línea epipolar. [23]

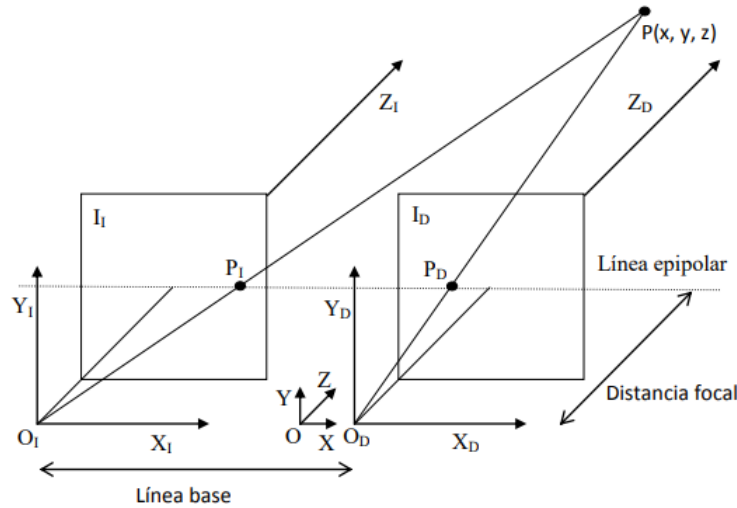


Figura 2.6: Configuración Canónica de un sistema estéreo [24].

Como se menciona en la sección de estado del arte, en lo descrito por [6], la capacidad de poder ver en tercera dimensión, que entrega la visión estereoscópica puede ser útil a la hora de realizar mediciones sobre imágenes o distintas demandas por la industria. Es por esto que Mathew Kokshoorn en [25], trabaja en la búsqueda e implementación de formas matemáticas y geométricas que estiman la distancia de un objeto a una cámara. Habla de la utilización de geometría para la obtención de distancia de un punto arbitrario a la línea base en donde se encuentran ubicadas las cámaras. Teniendo en consideración lo que se puede visualizar en la figura 2.7.



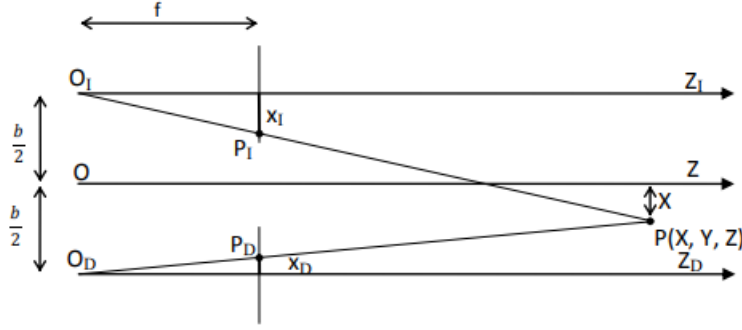


Figura 2.7: Relación Geométrica de un Sistema de Visión Estereoscópica [26].

Donde  $f$  es la distancia focal y  $b$  representa la separación entre las lentes conocida como *línea base*. Además,  $O_I$  y  $O_D$  son puntos principales de las cámaras izquierda y derecha respectivamente y el eje formado por  $O - Z$  se denomina eje óptico del par estéreo.  $P$  el punto de estudio de coordenadas  $(X, Y, Z)$ ,  $P_I$  de coordenadas  $(x_I, y_I)$  y  $P_D$  de coordenadas  $(x_D, y_D)$  corresponden a las proyecciones del punto  $P$  en las imágenes izquierda y derecha respectivamente. Finalmente  $X$  consiste de la distancia en el eje  $x$  del punto  $P$  al eje óptico del par estéreo. De la figura 2.7 se puede deducir la ecuación 2.4 y ecuación 2.5.

$$\text{Imagen izquierda} \rightarrow \frac{b/2 + X}{z} = \frac{x_I}{f} \Rightarrow x_I = \frac{f}{z} \left( x + \frac{b}{2} \right) \quad (2.4)$$

$$\text{Imagen derecha} \rightarrow \frac{b/2 - X}{z} = \frac{x_D}{f} \Rightarrow x_D = \frac{f}{z} \left( x - \frac{b}{2} \right) \quad (2.5)$$

$d$  son los píxeles de separación entre los puntos correspondientes entre dos imágenes, esto se conoce como *disparidad*. Considerando un punto en el espacio 3D ( $P$ ), proyectado en la imagen izquierda y derecha en el espacio 2D como  $P_I$  y  $P_D$  respectivamente, se tiene el cálculo de disparidad según la ecuación 2.6

$$d = x_I - x_D = \frac{f \cdot b}{z} \Rightarrow z = \frac{f \cdot b}{d} \quad (2.6)$$

El proceso de Visión Estéreo requiere de diversas técnicas y procedimientos que conllevan a resolver la Ecuación 2.6, lo cual resuelve el problema de la búsqueda de profundidad de un objeto dentro de una imagen. Luego de esto queda el procesamiento de la imagen en donde se puede modificar crear o ajustar diversas características dependiendo de lo que se quiera resaltar.

El procesamiento digital de imágenes se ha desarrollado para la comodidad de obtener más información con menos recursos o artefactos. Ha surgido como una nueva forma de extraer diversas características provenientes del entorno, aunque se debe ser cuidadoso a la hora de analizar las fotografías o imágenes de estudio debido a que los métodos utilizados son susceptibles a error por no poder separar algunos objetos superpuestos

o diferenciar entre el objetivo y el ambiente que lo rodea debido a las condiciones de luminosidad.

En la investigación realizada sobre la detección de profundidad de un objetivo a partir de una imagen aparecen varios conceptos que se deben tener en consideración y claridad.

Para lo anterior existen dos formas de poder realizar la visión en relieve o visión 3D con un par estereoscópico, estos se distinguen por la disposición de los ejes ópticos de ambas cámaras. La primera configuración consta de líneas de visión convergentes, y la segunda de líneas de visión paralelas. La primera mencionada busca que los ejes ópticos converjan en el punto de interés mientras que la segunda configuración busca que los ejes converjan en el infinito. Para facilitar el proceso de obtención de imágenes se hace uso de la configuración de ejes ópticos paralelos, porque para hacer uso de la otra alternativa se debería utilizar un mecanismo automatizado que mueva cada cámara por separado tratando de enfocar en su eje óptico el objetivo de estudio (similar a la visión humana). Si se dispone de dos ejes paralelos es simplemente buscar dentro de la imagen un objetivo y analizarlo.

### 2.3.2. Calibración de cámaras individuales

Esta etapa se encarga de recopilar los parámetros que se necesitan para establecer una mejora en la aplicación de algoritmos de visión computarizada, como los parámetros intrínsecos (distancia focal, factor de corrección por deformación FOV) y parámetros extrínsecos (posicionamiento y orientación de las cámaras), con la calibración realizada se pueden alinear las imágenes y buscar las líneas epipolares que se forman en las imágenes con mejores resultados y ahorrando coste computacional.

Antes de comenzar con el proceso de calibración se deben conocer ciertos términos que ayudan a la comprensión de parámetros útiles de una cámara. Para una buena comprensión revisar cada parámetros entorno a la figura 2.8.

1. **Parámetros intrínsecos:** Son aquellos que definen la geometría interna y la óptica de la cámara. Éstos determinan cómo la cámara proyecta los puntos del mundo 3D al plano de la imagen en 2D, siendo constantes en tanto no varíen las características y posiciones relativas entre la óptica y el sensor de imagen.
2. **Parámetros extrínsecos:** Éstos relacionan los sistemas de referencia del mundo real y la cámara, describiendo la posición y orientación de la cámara en el sistema de coordenadas del mundo real.
  - a) Vector de traslación  $T$ : Vector que determina la ubicación del centro óptico de la cámara ( $O$ ) con respecto a los ejes del mundo real  $(v, u, w)$ .
  - b) Matriz de rotación  $R$ : Matriz que relaciona la rotación de la posición de la cámara ( $O, X, Y, Z$ ) con respecto a los ejes del mundo real  $(v, u, w)$ .
3. **Punto principal:** El punto principal ( $C$ ) es el punto intersección entre el plano de la imagen y el eje óptico (recta perpendicular al plano de la imagen que pasa por

el centro de cámara  $O$ ). Las coordenadas de este punto vienen dadas en píxeles, y son expresadas respecto al sistema solidario al plano de la imagen  $(O', x', y')$ .

4. **Distancia focal:** Es la distancia existente entre la  $(O)$  y el punto principal  $(C)$ . Las coordenadas de este punto vienen dadas en píxeles horizontales y verticales.

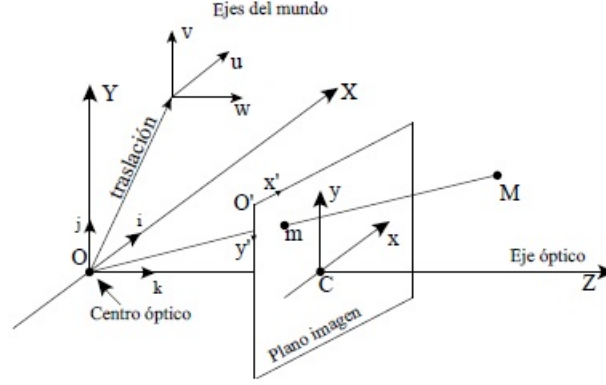


Figura 2.8: Modelo básico de una cámara. Extraído de [27].

#### ■ Resumen general de calibración de una cámara

La calibración de las cámaras tiene como finalidad encontrar la proyección de puntos en un escenario 3D y plasmarlos en uno 2D. Para esto se transforman los puntos de un sistema coordenado global al sistema coordenado de la cámara usando los parámetros extrínsecos (Matrices de rotación  $R$  y traslación  $t$ ). Una vez conocidos todos los parámetros de las cámaras y sus correspondientes coeficientes de distorsión se dice que las cámaras se encuentran calibradas.

La relación entre los puntos 3D en coordenadas globales  $(X_w, Y_w, Z_w)$  con su proyección  $(u, v)$  en las coordenadas de la imagen es la vista en la ecuación 2.7.

$$\begin{bmatrix} u' \\ v' \\ z' \end{bmatrix} = P \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \quad (2.7)$$

$$u = \frac{u'}{w'}$$

$$v = \frac{v'}{w'}$$

Donde,  $P$  es una matriz de proyección de  $3 \times 4$  que consta de dos partes: la matriz intrínseca ( $K$ ) que contiene los parámetros intrínsecos y la matriz extrínseca ( $[R \mid t]$ ) que es una combinación de la matriz de rotación  $R$  de  $3 \times 3$  y un vector de traslación  $t$  de  $3 \times 1$ . Véase en la ecuación 2.8.

$$P = \overbrace{K}^{\text{Matriz intrínseca}} \times \overbrace{[R \mid t]}^{\text{Matriz extrínseca}} \quad (2.8)$$

La matriz intrínseca es triangular superior, como se ve en la siguiente expresión.

$$K = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (2.9)$$

donde,

$f_y, f_x$  son las distancias focales  $x$  e  $y$ .

$c_y, c_x$  son las coordenadas  $x$  e  $y$  del centro óptico en el plano de la imagen.

$\gamma$  es la inclinación entre los ejes. Suele ser 0.

El objetivo de la calibración es entonces encontrar la matriz  $K$ , la matriz de rotación  $R$  y el vector de traslación  $t$ . Utilizando un conjunto de puntos 3D conocidos y sus correspondientes coordenadas en la imagen. Cuando se obtienen los parámetros de intrínsecos y extrínsecos se puede decir que la cámara está calibrada.

#### ■ Modelo lineal de la cámara

El modelo básico de la cámara representa la transformación de las coordenadas de los puntos de la escena en las coordenadas de la imagen. Se muestra en la figura 2.8.

El modelo lineal viene dado por la ecuación 2.10.

$$q_i = \lambda \cdot K \cdot [R \ t] \cdot p = \lambda \cdot K \cdot [r_1 \ r_2 \ r_3 \ t] \cdot p_i \quad (2.10)$$

Donde,  $q_i = (u_i, v_i, h_i)$  que representa las coordenadas homogéneas en la imagen de un punto en la escena  $p_i = (x_i, y_i, z_i, 1)$ . El modelo contiene la orientación de la cámara en la escena y su posición expresadas con una matriz de rotación  $R$  y un vector de traslación  $t = (t_x, t_y, t_z)$  respectivamente, además de las características intrínsecas de la cámara contenidas en la matriz  $K$ .

#### ■ Modelo no lineal de la cámara

Seguido de esto, se encuentra el modelo de distorsión de la cámara que busca modelar las distorsiones dentro de la imagen tanto tangencial (denotada por la letra  $p$ ), radial (denotada por la letra  $k$ ) y prismática (denotada por letra  $s$ ), siendo la distorsión efectiva la suma de todas las anteriormente mencionadas. Obteniendo la siguiente expresión.

$$\begin{aligned} \delta_u(u_p, v_p) &= u_p \cdot k_1 \cdot r^2 + p_1(3u_p^2 + v_p^2) + 2p_2 \cdot u_p \cdot v_p + s_1 \cdot r^2 \\ \delta_v(u_p, v_p) &= v_p \cdot k_1 \cdot r^2 + 2p_1 \cdot u_p \cdot v_p + 2p_2(u_p^2 + 3v_p^2) + s_2 \cdot r^2 \end{aligned} \quad (2.11)$$

Esta última siendo la expresión del modelo no lineal de la cámara.  $r$  es la distancia de un píxel al punto principal ( $C$ ) de la imagen.

### ■ Método de calibración

Existen diversos métodos de calibración. Para tomar la decisión de qué método utilizar es importante destacar que el método debe ser versátil y de fácil aplicación, es por esto que basado en un estudio realizado en [28], se realiza mediante el método de Zhang ya que resulta muy flexible desde el punto de vista de que tanto la cámara como la plantilla (plantilla necesaria para realizar tomas de fotografías de prueba) puede ser movida libremente y además se pueden tomar tantas imágenes como se quieran sin tener que volver a realizar medidas en la plantilla. [28]

Para el cálculo de los valores intrínsecos y extrínsecos se pueden revisar en [29].

### ■ Procedimiento de calibración

1. Diseñar una plantilla tipo tablero de ajedrez de  $9 \times 6$  cuadros.
2. Tomar 30 imágenes de la plantilla desde varias posición y orientaciones.
3. Detectar las esquinas de los cuadrados del tablero en las imágenes.
4. Estimar los cinco parámetros intrínsecos y todos los parámetros extrínsecos usando la solución explícita planteada por Zhang en [29].
5. Estimar los coeficientes de la distorsión radial, resuelto por mínimos cuadrados lineales. Véase más en detalle en [29].
6. Depurar todos los parámetros minimizando un funcional.

Dentro de la librería *OpenCV*, la función encargada de realizar el método propuesto por Zhang es *calibrateCamera*. La cual entrega como resultado la matriz intrínseca de las cámaras, coeficientes de distorsión del lente, el vector de rotación y el vector de traslación.

### 2.3.3. Calibración de las cámaras como par estereoscópico

Para conseguir buenos resultados en las mediciones y estudios de las imágenes obtenidas por un par estereoscópico, es necesario realizar una calibración previa de las cámaras. De tal forma que estas logren reconocer puntos de interés localizados en distintas partes de ambas imágenes capturadas como el mismo punto.

### ■ Calibración estéreo

Se realiza mediante la función *stereoCalibrate* de *OpenCV*, la cual se encarga de estimar la transformación entre dos cámaras formando un par estéreo. Explicado con otras palabras, la calibración estéreo se realiza para establecer la transformación que hay entre los planos de las dos imágenes, es decir la traslación que existe entre el punto principal de una y otra, como también la orientación de una respecto a la otra. Un

ejemplo para entender la teoría detrás de la función se explica en los próximos párrafos. Extraído de [30].

Teniendo una configuración de cámaras como se observa en la sección 2.3.1, en donde la posición relativa y la orientación entre ambas son fijas, se puede calcular la posición de un punto en una imagen captada por la primera cámara con respecto a la otra captada por la segunda con  $(R_1, T_1)$  y  $(R_2, T_2)$ , respectivamente. Lo anterior mencionado quiere decir que, si la posición y orientación relativas  $(R, T)$  de las dos cámaras se conoce, es posible calcular  $(R_2, T_2)$  cuando  $(R_1, T_1)$  es conocido. La ecuación 2.12 describe el comportamiento de lo recién aclarado. Siendo  $R$  la matriz de rotación y  $T$  el vector de traslación de ambas cámaras encontradas en los parámetros extrínsecos obtenidos en la sección 2.3.2.

$$\begin{aligned} R_2 &= RR_1 \\ T_2 &= RT_1 + T \end{aligned} \tag{2.12}$$

Por lo tanto, se puede calcular la representación de coordenadas de un punto 3D para el sistema de coordenadas de la segunda cámara cuando se le da la representación de coordenadas del punto en el sistema de coordenadas de la primera cámara. Ver ecuación 2.13.

$$\begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ 1 \end{bmatrix} \tag{2.13}$$

La función *stereoCalibrate* tiene la capacidad de calcular los parámetros intrínsecos sin la necesidad de utilizar la calibración individual de las cámaras, pero esto no se recomienda debido al error que puede existir debido a todas las estimaciones que debe realizar la función, además consume demasiada memoria. Es por ello que simplemente se recomienda determinar anteriormente dichos parámetros como se hace en la sección 5.2.1.

## ■ Rectificación del proceso de calibración estéreo

El fin de esta rectificación es facilitar el proceso de correspondencia estéreo de imágenes. Esto se consigue conociendo matrices proyección que ajustan las imágenes de tal manera que se alinean de forma paralela las líneas epipolares correspondientes entre las imágenes derecha e izquierda. A su vez, se acomodan de tal forma que dichas líneas epipolares se encuentren paralelas a la línea base de la geometría de las cámaras.

- **Rectificación de distorsión:** La encargada de esta tarea es la función *initUndistortRectifyMap* la cual se aplica tantas veces como cantidad de cámaras conformen el sistema, para el caso de visión estéreo se aplica dos veces, una para cada cámara.

Esta se encarga de crear los mapas de rectificación que son capaces de determinar las coordenadas para cada píxel de la imagen rectificada, las coordenadas correspondientes en la imagen de origen sin rectificar.

Este proceso logra eliminar distorsiones producidos por la forma de los lentes de las cámaras. Un ejemplo de lente que produce una alta distorsión es el conocido lente *fisheye* u “ojo de pez”.

## ■ Filtro WLS

El filtro por mínimos cuadrados ponderados o por sus siglas en inglés *WLS*, es aplicado con la finalidad de obtener un suavizado más uniforme dentro de las imágenes que ya han sido transformadas a un mapa de disparidad, esto para conseguir que cada objeto por separado tenga un color uniforme, la gracia de este filtro es que permite difuminar los objetos pero manteniendo la calidad de los bordes. En simples palabras permite un suavizado de la imagen preservando los bordes involucrados. El objetivo de este proceso es obtener un mapa de disparidad evitando la aparición de píxeles atípicos en cuanto a disparidad.

### 2.3.4. Correspondencia estéreo

Se denomina correspondencia estéreo al emparejamiento de las proyecciones de un punto del espacio 3D en las imágenes capturadas por las cámaras. Para esto es necesario que ambas imágenes contengan la información de ese punto. Por lo que, el área en la que se pueden calcular las correspondencias son aquellas que se encuentran presentes en ambas fotografías.

Una vez conocidas las posiciones relativas de las cámaras, la matriz de rotación y vector de traslación, se puede calcular la disparidad de los puntos con correspondencia, usando la ecuación 2.14.

$$d = x_I - x_R \quad (2.14)$$

Siendo la disparidad  $d$  la diferencia entre las coordenadas  $x_I$ ,  $x_D$ , que corresponden a las coordenadas de los puntos  $P_I$  y  $P_D$  respectivamente que son la proyección 2D dentro de las imágenes de un punto  $P$  en el espacio 3D. La disparidad será mayor mientras

menor sea la distancia entre el objetivo y las cámaras, y la disparidad será menor mientras mayor sea la distancia.

Como resultado del cálculo de correspondencias se obtiene un mapa de disparidad, que consta de una imagen en escala de grises, en la que se representan las disparidades encontradas. La tonalidad de gris del píxel del mapa depende del grado de disparidad existente, y dado que la disparidad existente depende de la distancia a la que se encuentra el punto, se utiliza este mapa para obtener la coordenada  $Z$ , de distancia entre el objetivo y las cámaras.

### ■ Métodos de correspondencia en visión estéreo

La reconstrucción de escenarios 3D mediante el uso de imágenes estéreo es un tema ampliamente estudiado, dentro de lo cual existen dos corrientes que definen en cómo se realiza la correspondencia de las imágenes, por un lado existen aquellos que utilizan métodos basados en las áreas, que es utilizado para la creación de mapas de disparidad densos, ya que la correspondencia se realiza píxel a píxel. Sobre este método se desarrolla este trabajo. Por otro lado, existen los métodos basados en las características, que son elementos como píxeles de los bordes, segmentos lineales o curvas de los bordes, regiones, entre otros que se usan para encontrar las correspondencias entre imágenes.

A su vez, existen otras tres grandes categorías que predominan a la hora de categorizar los métodos de correspondencia, estos son los métodos locales y métodos globales. Los locales consideran la evaluación píxel a píxel de las fotografías, sólo considerando los píxeles vecinos, que llamaremos ventana (píxel de estudio y sus vecinos). Los métodos globales consideran la información disponible dentro de toda la imagen. Los primeros mencionados carecen de capacidades de manejar situaciones como cambios abruptos de variación de profundidad (disparidad), para solucionar estos problemas es preferible utilizar los métodos globales. Por otro lado, existen los métodos semi globales que usan la información proveniente de los vecinos del píxel de estudio en diferentes direcciones, por lo que para evitar un costo computacional elevado se pueden analizar bloques más pequeños dentro de la misma imagen.

El algoritmo elegido es una correspondencia densa híbrida denominado *SGBM* (*Semi-Global Block Matching*), que se caracteriza por ser un punto intermedio entre los algoritmos globales y locales, intentando acercarse a la precisión de los globales sin sacrificar la eficiencia computacional de los locales. En la siguiente sección se explica en qué consiste a grandes rasgos la correspondencia *SGBM*. Para analizar más detalles véase en el documento [24].

### ■ Configuración de parámetros para el método de correspondencia

Este apartado corresponde a la sección de calibración debido a que son tareas que se realizan sólo una vez para configurar y establecer parámetros para el correcto funcionamiento de la función que implementa el método *SGBM*.

Los parámetros importante a seleccionar corresponden al siguiente listado.



- La cantidad mínima del valor de disparidad permitido.
- La cantidad máxima del valor de disparidad permitido.
- Tamaño del bloque emparejado.
- Parámetros para el control de suavizado para la creación del mapa de disparidad.
- Valor de estudio del suavizado aplicado para la eliminación de bloques de bajo interés.

En la sección de 2.3.5 se puede revisar como trabaja la función *StereoSGBM*.

### 2.3.5. Cálculo de distancia mediante uso de visión estéreo

Una vez terminada la calibración y configuración de parámetros se comienza con el tratamiento directo sobre la imagen de estudio para el análisis estéreo. Hasta el momento las imágenes obtenidas solo han sido útiles para la calibración, y sólo deben ser capturadas una vez. Para luego ser utilizadas una y otra vez para realizar todos los procedimientos de la sección 2.3.3. En la figura 2.9, se logra visualizar el algoritmo a través de un diagrama de flujo que explica desde la captura de imágenes hasta la obtención de profundidad.

#### ■ Captura de imágenes

Se realiza la captura de imágenes del par estéreo las cuales son guardadas para su correcto tratamiento. Estas son tomadas a un objetivo de interés.

#### ■ Eliminación de distorsión y ajuste de líneas epipolares

La matriz de rotación y el vector de traslación encontrados en la calibración del par estéreo son utilizados en conjunto de los valores de distorsión para eliminar, justamente eliminar la distorsión creada por las lentes de las cámaras. Se ajusta un cuadro de trabajo en donde se puede realizar la correspondencia de puntos entre las imágenes izquierda y derecha que se pueden observar en la figura 2.10, que muestra de forma gráfica como ajusta las líneas epipolares de las imágenes y corrigiendo la distorsión existente en la imagen delimitado de color verde.

#### ■ Transformación a escala de grises

Para la correcta aplicación de algoritmos de correspondencia es un requisito transformar las imágenes a escalas de grises, para denotar y clarificar los bordes dentro de la imagen.

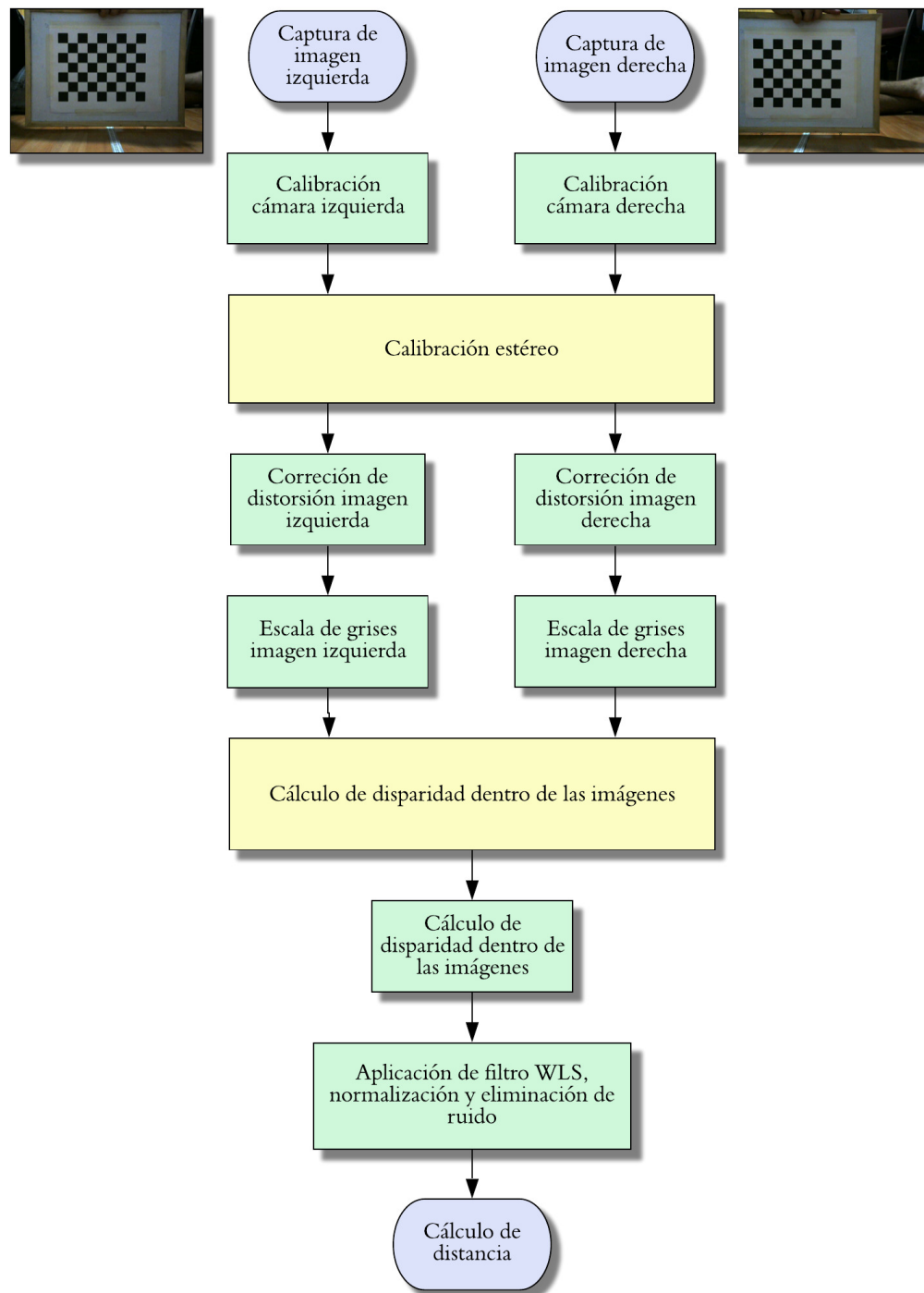


Figura 2.9: Diagrama de flujo que describe el proceso de obtención de distancia con visión estéreo.

### ■ Cálculo de disparidad

El algoritmo utilizado es el *SGBM*, que es una técnica de correspondencia densa híbrida que analiza las imágenes a través de sub-partes o ventanas, pero no para calcular

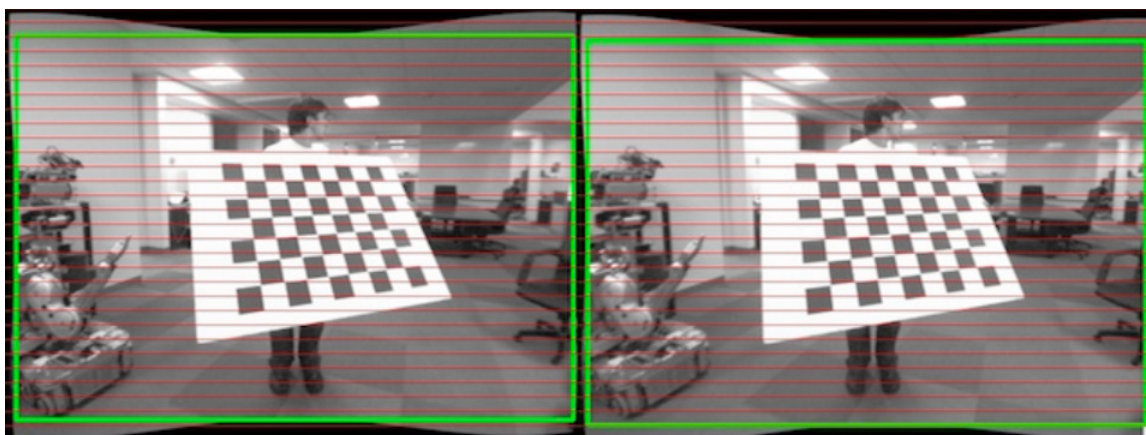


Figura 2.10: Fotografía izquierda y derecha captadas y rectificadas. Extraído de [30].

emparejamientos o “matching” no buscan similitudes individuales entre píxeles como los métodos globales, sino que más bien intenta minimizar una función de energía a nivel de ventana.

Para explicar el algoritmo detrás de la función *SGBM*, el cual para utilizarlo se debe tener la imagen rectificada seguido de “.compute”, para calcular las disparidades existentes. Es necesario conocer dos conceptos útiles que sirven para esclarecer el funcionamiento de este método de correspondencia.

- **Niveles de disparidad:** Son un parámetro que se utiliza para definir el espacio de búsqueda para la coincidencia, sabiendo que hay píxeles que no se pueden estudiar porque no existe ese punto en la otra imagen debido al desplazamiento o ángulo de captura. Para expresar de mejor forma se expone un ejemplo. Un punto  $P$  en el espacio 3D representado por las proyecciones a través de puntos en 2 imágenes izquierda y derecha ( $P_I$  y  $P_D$  respectivamente) de coordenadas  $(x_I, y)$  y  $(x_D + d, y)$ . Asumiendo que  $y$  al estar las imágenes calibradas y rectificadas es el mismo valor en ambas.  $d$  es la disparidad existente entre ambas imágenes o entre los puntos  $P_I$  y  $P_D$ . Al existir esta disparidad las  $d$  primeras columnas de píxeles de la imagen izquierda no se pueden utilizar por no tener su correspondiente en la imagen derecha. Por su parte, en la imagen derecha las últimas  $d$  columnas no tienen su similar en la imagen izquierda. Es a esto lo que se llaman niveles de disparidad.
- **Número de direcciones:** En el algoritmo *SGBM*, para optimizar la función de energía, la imagen de entrada es considerada de distintas direcciones como se visualiza en la figura 2.11.

El algoritmo *SGBM* posee tres módulos principales de trabajo, enumerados a continuación con su respectiva definición.

- a) **Cálculo de costos de coincidencia:** El *SGBM* para calcular el costo de coincidencias utiliza dos pasos.
  - 1) Center Symmetric Census Transform (CSCT): El cual consiste en una ventana deslizante de  $9 \times 7$  píxeles, que recorre ambas imágenes capturadas. El CSCT para el píxel central se estima comparando el valor de

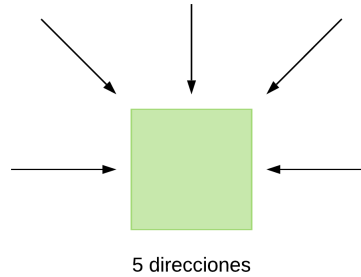


Figura 2.11: Cinco direcciones consideradas en el algoritmo *SGBM*.

cada píxel con su correspondiente contraparte simétrica, fijando el punto centro como eje de simetría. Si el valor del píxel es mayor su correspondiente simétrico, el resultado es 1, de lo contrario, es 0. Generando de esta forma 31 resultados provenientes de las comparaciones los cuales en su conjunto forman un número de 31 bits. Y así, este número de 31 bits es la salida CSCT para cada píxel de ambas imágenes.

- 2) **Cálculo de distancia de Hamming:** Consiste en realizar la operación *XOR* por píxeles de las salidas del *CSCT* lo cual busca las diferencias entre una imagen y otra a través de los números de 31 bits encontrados en ambas imágenes para cada píxel, dando así una distancia o diferencia entre los puntos de interés.
- b) **Cálculo de costos direccionales:** Se realiza debido al ruido asociado dentro de las imágenes el resultado del costo de coincidencia podría ser incierto y tener un menor que lo esperado considerando que deben ser costes mayores que las mediciones correctas. Por lo tanto, se aplican restricciones adicionales para aumentar la uniformidad al penalizar los cambios de disparidades vecinas abordando cada píxel de 5 distintas direcciones (figura 2.11). Para más detalles al respecto revisar [31].
- c) **Posprocesado:** Este módulo posee tres pasos de trabajo los cuales consisten en metodologías matemáticas para el cálculo de costos mínimos para píxeles, interpolación y funciones de unicidad para asegurar confiabilidad de las disparidades mínimas calculadas mediante estos tres pasos.

## ■ Aplicación de filtros WLS

Esta etapa no es más que aplicar el filtro configurado en la sección de calibración, que como se explica es para suavizar las imágenes pero sin perder la calidad de los bordes que denoten las figuras existentes en las imágenes.

## ■ Normalización de imagen

Es el proceso en el que se cambian los rangos de valores de la intensidad de los píxeles para una mejor visualización si se quieren mostrar los resultados obtenidos por

el mapa de disparidad filtrado.

#### ■ Eliminación de ruidos

Para esto se utiliza la herramienta *morphologyEx*, el cual consta de eliminación de ruido dentro del mapa de disparidad. El tipo de corrección es de clase "closez" trata de cerrar los vacíos generados por el ruido dentro de una figura. Ver figura 2.12.



Figura 2.12: Ejemplo de aplicación de la eliminación de ruidos según *MorphologyEx* de clase "close". Extraído de [32].

#### ■ Obtención de profundidad

Mediante la selección de un punto  $P$  de coordenadas  $(x, y)$  dentro de la imagen de disparidad se evalúa una función obtenida mediante el cálculo práctico, a través de múltiples pruebas de detección de objetivos a una distancia conocida para establecer un rango de disparidad de trabajo. Esta función polinómica de grado cuatro que relaciona disparidad vs distancia del objeto de interés a las cámaras.

$$z = 33,766d^4 - 117,2d^3 + 165,7d^2 - 137,87d + 85,475 \quad (2.15)$$

Donde,  $z$  representa la distancia del objetivo a las cámaras en milímetros y  $d$  representa la disparidad existente en el punto de interés seleccionado.

#### 2.3.6. Comentarios

Es importante mencionar que todos los procesos descritos en este apartado poseen una lógica de funcionamiento independiente el uno de otro. Es decir, la calibración de cámaras funciona solamente para el programa de cálculo de distancia, la distancia resultante de este mismo mencionado debe ser ingresado manualmente al programa de cálculo de diámetro. Por otro lado, como el cálculo de diámetro y distancia funcionan por separado es que tienen diversos métodos de captura de imagen para funcionar de manera correcta.

El cálculo de diámetro, requiere de una toma de fotografía por parte de la cámara izquierda, esta fotografía es analizada y en su resultado final es posible visualizar los resultados finales del algoritmo, que en este caso es la obtención de diámetro del objetivo.

Para el cálculo de distancia, se necesita una toma constante de imágenes, formando en tiempo real un análisis de un mapa de disparidad que es similar a un vídeo con una

baja tasa de refresco debido al alto trabajo de computar información que demanda el analizar, modificar, calibrar y mostrar en pantalla los resultados del mapa de disparidad.

Es por esto que es necesario modificar los comportamientos de estos algoritmos con la finalidad de unificarlos y automatizar el proceso de traspaso de información.

## 2.4. Implementación de algoritmos en un ordenador de placa reducida

En este capítulo se responde la pregunta ¿por qué es necesario el uso de un ordenador de placa reducida?, para luego exponer cómo fue implementado el algoritmo en dicha plataforma. La aplicación de algoritmos se abordará de una manera más compacta que en los capítulos anteriores considerando que las librerías y lenguaje de programación son los mismos que fueron usados en el ordenador de escritorio en el cual se realizaron las pruebas de funcionamiento.

### 2.4.1. Justificación

Un ordenador de placa reducida se define como un computador completo implementado en solo una placa de tamaño pequeño en relación a los ordenadores de escritorio, sacrificando rendimiento en general a cambio de portabilidad y versatilidad.

El uso de este ordenador de placa reducida es se justifica debido a la necesidad de reducir la masa total de un sistema de transporte que carga con el sistema de visión estéreo. Necesitando así una unidad de procesamiento portátil, lo que significa que se necesita reducir el tamaño y masa de la unidad encargada de ejecutar los algoritmos que se explicaban en la sección anterior.

Para la selección de este componente es necesario que sea fácil implementar y que pueda existir una traducción de los algoritmos desarrollados sin necesidad de modificar demasiado los códigos. Es por esta razón que se trabaja sobre la plataforma *Raspberry Pi 4*, las especificaciones de esta se pueden ver en el anexo B.



Figura 2.13: Raspberry Pi 4 en donde se implementa el proyecto de visión estéreo.

Las herramientas digitales necesarias para utilizar la *Raspberry Pi 4* una previa

instalación de un sistema operativo, en este caso se utiliza *Raspbian*. Se instalan los paquetes de *Python* y la librería de *OpenCV*.

### 2.4.2. Depuración de códigos

El poder de procesamiento de una *raspberry* es muy inferior al de un ordenador de sobremesa con las características que se mencionan en el anexo B, por esta razón es necesario modificar los códigos para lo que se utiliza el algoritmo de la figura 2.14 como base de cambio.

### 2.4.3. Algoritmos depurados aplicados en Raspberry Pi 4

El trabajo que se describe en la sección 2.2 y sección 2.3 se consideran como base de trabajo para el algoritmo final depurado que se rige según el diagrama de flujo presente en la figura 2.14.

Para lograr esto, se realiza a través de creación de funciones que ajusten y mejoren el funcionamiento actual de los algoritmos creados.

El objetivo principal de esta adaptación de algoritmos es que se realice una unificación de tareas, con esto tener un código que realice de forma secuencial sin intervención humana la captura, detección de objetivo, calibración, cálculo de ancho y distancia. Para esto se crean cuatro funciones que se describen a continuación.

#### ■ Algoritmo de calibración

Es lo mismo descrito en la sección 2.3.2 y sección 2.3.3. Con la diferencia de haber sido separados del algoritmo de cálculo de distancia. La calibración de cámaras es una tarea que se debe realizar solo una vez, es por esto que para ejecutar en menor tiempo todos estos algoritmos es que la calibración fue separada de la detección de distancia, ya que la distancia es algo que se va requiriendo reiteradas veces a lo largo del funcionamiento del código depurado, como se visualiza en la figura 2.14.

#### ■ Algoritmo de detección de objetivos

Este algoritmo se encarga de realizar la captura de un objeto, en este caso considera la la figura de un tronco o tallo que se encuentre frente a un fondo o telón de color blanco para generar el alto contraste que es necesario para la detección de este objetivo.

En primer lugar, es importante considerar que la *raspberry pi 4* debido a su capacidad de procesamiento no es idóneo para el análisis y trabajo sobre imágenes de alta resolución. Entonces, el primer paso que se debe hacer es configurar la captura de vídeo de las cámaras a una resolución menor, para lo cual se determinó mediante prueba y error que la resolución máxima permitida es  $544p \times 288p$ . Esta resolución permite trabajar con un buen tiempo de desarrollo de los códigos sin sacrificar calidad de los resultados obtenidos.

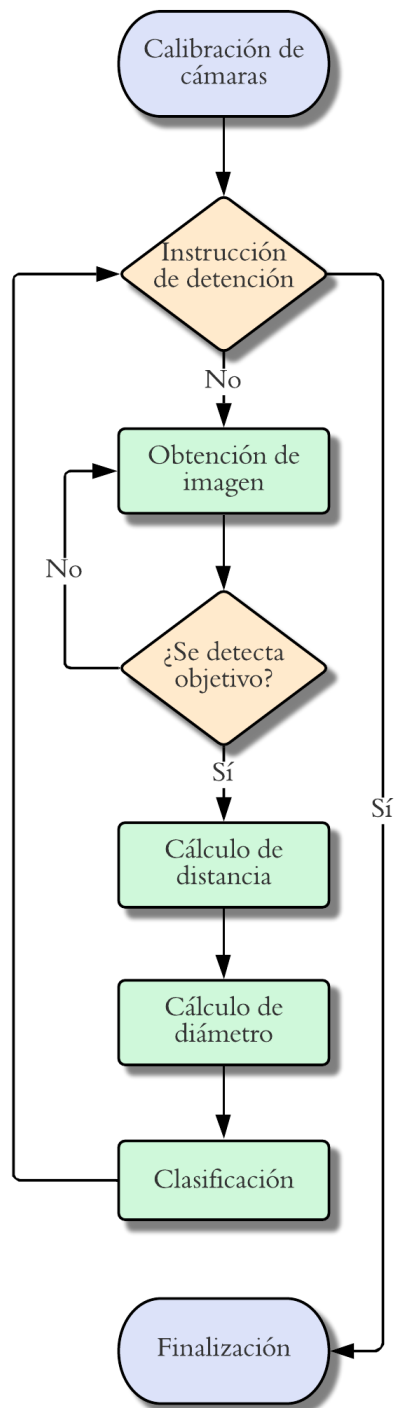


Figura 2.14: Diagrama de flujo explicativo de algoritmo depurado para implementación en raspberry pi 4.



Segundo, se utilizan funciones de transformación y análisis de imagen píxel a píxel según su color en formato RGB (red, green, blue). Seguido de trabajar la imagen, se crea una máscara en donde el resultado del análisis de imagen para detectar una figura por color, es plasmado en forma de rectángulo describiendo solo el borde de todo el rectángulo. Lo importante de esta figura es que se puede utilizar el punto centro de esta figura en coordenadas de píxel con el fin de que este punto obtenido sea el cual se analice para la obtención de profundidad dentro del mapa de disparidad en la función de cálculo de distancia.

Finalmente, la forma de decidir cuando detener esta función de detección es si el objetivo (rectángulo formado por lo descrito en el párrafo anterior) se mantiene por 50 ciclos de trabajo del algoritmo, si este se interrumpe comienza la cuenta nuevamente y sigue buscando. Una vez detecta durante 50 ciclos de trabajo un objetivo, se busca enviar una señal de detención para lo cual se mantiene la detención del sistema móvil durante 50 ciclos más de detección. Finalmente, se envían las fotografías analizadas y centro de objetivo como resultado de la función *detect()*.

#### ■ Cálculo de distancia (depurado)

El código en general cumple las mismas funciones, pero con diferentes prestaciones. Esto quiere decir que, en primer lugar se declara este código como una función que puede ser llamada desde otro código. Por otro lado, se modifica el método de ingreso de datos, al ser una función requiere de ciertos parámetros para funcionar de manera correcta. También, se suprime la opción de poder visualizar los resultados del mapa de disparidad para ahorrar coste computacional, por lo que el único resultado visible es una variable que contiene el valor de distancia obtenido. Debido a lo anterior ya no se puede visualizar un vídeo en tiempo real de lo que se ve a través del par estereoscópico, y a su vez este algoritmo solo se ejecuta una vez para la obtención del resultado final (distancia), considerando además que como argumento de esta función se encuentran las imágenes captadas en el apartado de detección, sin cotar todos los parámetros de calibración necesarios para trabajar con la visión estéreo.

#### ■ Cálculo de diámetro (depurado)

No existen grandes variaciones en este apartado, más allá de transformar este código en una función que recibe como argumento la distancia obtenida y la imagen captada por la cámara izquierda, y entrega como resultado el diámetro del objeto estudiado.

## 2.5. Especificación de materiales utilizados para pruebas

### 2.5.1. Especificaciones técnicas de la cámara web Logitech C270 HD.

Se requiere de una cámara web de acceso comercial que posea compatibilidad segura con la *Raspberry Pi 4*, es por esta razón que Logitech C270 HD es la mejor opción debido a evidencias a través de papers e investigaciones en donde trabajan con webcams logitech en conjunto de ordenadores *Raspberry* sin dificultades. Las especificaciones

técnicas de la *Raspberry Pi 4* y el ordenador para realizar las primeras pruebas se encuentran en los anexos B y A respectivamente.

Tabla 2.1: Especificaciones técnicas de *Webcam Logitech C270 HD* .

| Especificaciones Técnicas      |                           |
|--------------------------------|---------------------------|
| Tipo de conexión               | USB con cable             |
| Tipo de USB                    | USB 2.0 de alta velocidad |
| USB VID_PID                    | VID_046D&PID_081A         |
| Objetivo y tipo de sensor      | Plástico                  |
| Tipo de enfoque                | Fijo                      |
| Campo visual                   | 60°                       |
| Longitud focal                 | 4 mm                      |
| Resolución óptica              | 1280 × 960 1,2 MP         |
| Captura de imágenes (W 16 : 9) | 320p, 480p, 720p          |
| Captura de vídeo (W 16 : 9)    | 320p, 480p, 720p          |
| Frecuencia de cuadro (máx.)    | 30 fps: 640 × 480         |
| Longitud del cable             | 1,5 metros                |

### 2.5.2. Planos de sistema de sujeción

En este apartado se muestra el resultado final de la creación de un sistema de sujeción de dos cámaras web. Este sistema fue realizado a la medida y para los requerimientos predefinidos, como la distancia entre ejes focales de 10[cm] y que la distancia de dichos ejes a la superficie de apoyo sea de 13[cm]. En la figura 2.15. Adicionalmente, se menciona que se han utilizado pernos M3 con sus respectivas tuercas y golillas, para realizar la unión de las piezas. Los planos de las piezas por separado se pueden visualizar en la Sección D de los Anexos. Ver las figuras 7.7, 7.8 y 7.9.

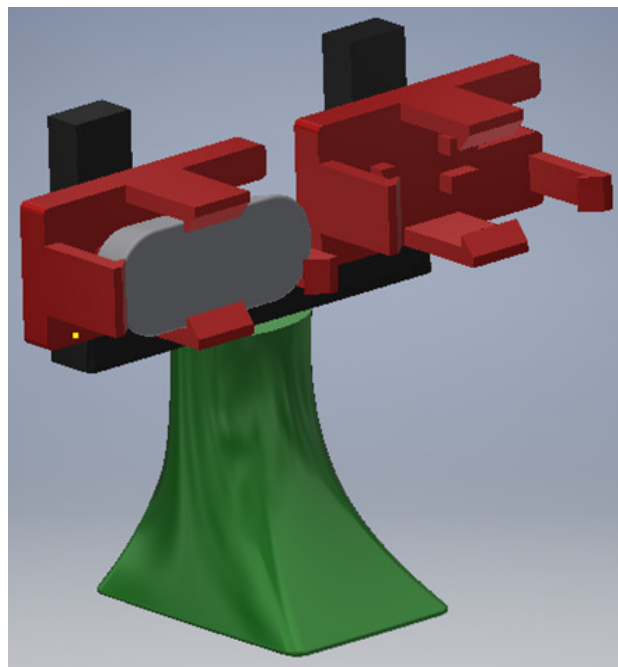


Figura 2.15: Sistema par el montaje de las cámaras web ensamblado.

### 3. Diseño de robot móvil para transporte de instrumento de medición estéreo

#### 3.1. Robot móvil tipo diferencial

El robot móvil tipo tracción diferencial se caracteriza porque el movimiento se consigue con dos ruedas, cada una colocada junto a su respectivo motor, teniendo comúnmente 3 ruedas de apoyo. El robot de esta forma puede moverse de distintas formas; en línea recta, si los motores se fijan a una misma velocidad en magnitud y sentido; girar en una dirección, si las velocidades de ambos motores difieren en magnitud pero mantienen su mismo sentido; sobre su propio eje, cuando las ruedas se ajustan a velocidades iguales en magnitud pero en sentidos opuestos. Es por esta simplicidad al controlar que se escoge por sobre otras alternativas existentes con más ruedas de tracción.

##### 3.1.1. Requerimientos

El robot tipo diferencial diseñado para el presente proyecto debe ser capaz de soportar cargas menores, no superiores a 3 *Kg*, adicionales a su propio peso. Considerando que debe cargar con el hardware de control y alimentación de todo el sistema.

La velocidad a la cual el robot debe moverse para obtener una correcta detección de objetivos y lograr detenerse a tiempo es de 0,18 *km/h*, además como se considera un ambiente controlado para el desarrollo del proyecto se realizan cálculos para superficies planas.

En cuanto a dimensiones se requiere que sea lo suficientemente grande como para albergar todos los componentes mencionados al comienzo sin que generen conflictos entre sí, ya sea por efectos adversos en las señales como temperaturas.

##### 3.1.2. Selección de componentes

A partir de los requerimientos, considerando la velocidad de 5 *cm/s* y una estructura capaz de soportar 5 *kg* adicionales a su propia masa, esperando que la masa del carro sea de máximo 5 *kg*.

Se considera que para efectos de funcionalidad y visual de componentes interiores se utilicen placas de acrílico de 3 *mm* de espesor para la base inferior y superior de la caja del robot. El acrílico ha sido escogido por su estética y buenas características mecánicas como de resistencia para los usos que se le busca dar.

Tabla 3.1: Propiedades del acrílico. Información extraída de [33].

|                               |          |
|-------------------------------|----------|
| Módulo de Young               | 2,7 GPa  |
| Coefficiente de Poisson       | 0,35     |
| Límite de elasticidad         | 48,9 MPa |
| Resistencia máxima a tracción | 79,8 MPa |

Por otro lado, las ruedas diseñadas se basan en ruedas de caucho antideslizantes comerciales que se acoplan a los motores seleccionados para proyectos de robótica móvil.

En la figura 3.1 se visualiza el diseño de esta junto a su respectiva llanta.

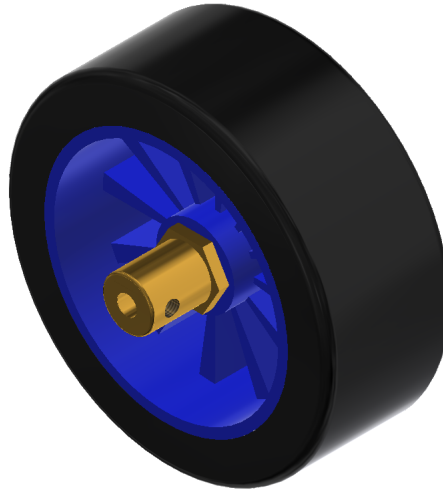


Figura 3.1: Diseño de la rueda.

Finalmente, la rueda sin tracción aplicada es diseñada en base a modelos existentes en el mercado denominadas “rueda loca” la cual consiste en una base giratoria que sujeta una rueda de forma similar a la anterior vista pero de menor tamaño. La incorporación de una rueda de ese tipo; además de ser la tercera pieza de apoyo del robot; facilita la movilidad del vehículo permitiendo seguir la dirección dada por las ruedas de tracción sin oponer mayor resistencia. En la figura 3.2 se ve el diseño de la rueda en cuestión.

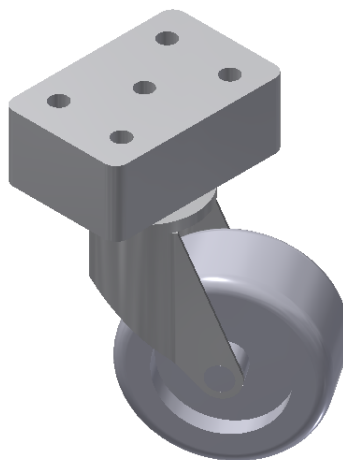


Figura 3.2: Diseño de la rueda loca.

### 3.2. Diseño 3D del robot móvil

El diseño 3D del robot móvil es realizado en *AutoCAD Inventor*, para su creación se consideraron los requerimientos anteriores mencionados, tomando en cuenta que es un robot tipo tracción diferencial; este cuenta con 3 apoyos: 2 ruedas laterales y 1 rueda loca que se ubica en la zona posterior del robot de transporte que permite el desplazamiento y seguimiento del movimiento impulsado por los motores.

Para resguardar la integridad de los componentes electrónicos utilizados, se diseña una carrocería tipo caja de tamaño  $220mm \times 200mm$  para albergar todo de forma correcta y ordenada teniendo en cuenta que la base de la caja sirve para el montaje de circuitos electrónicos de alimentación y la parte interior de la tapa superior para soportar el microcontrolador y el ordenador de placa reducida, además la parte superior de la tapa superior de la caja debe llevar montado el sistema de visión estéreo.

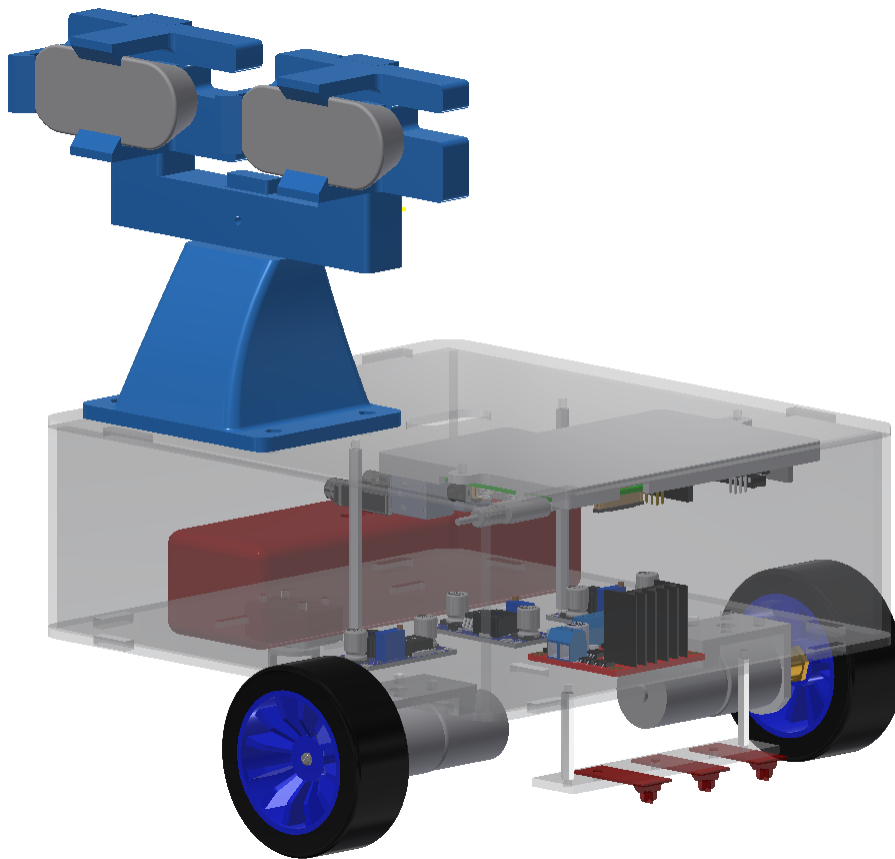


Figura 3.3: Modelo 3D del robot móvil.

La distribución de los componentes electrónicos dentro del robot se encuentran en la base y tapa del cofre del robot; en la base inferior se encuentran los elementos de alimentación y distribución de energía para motores y microcontroladores, como las baterías LiPo y las placas reguladoras de voltaje LM2596 más la placa controladora de motores L298n. Cabe destacar que en la parte inferior de la base del robot se montan los soportes de sensores seguidores de línea, los soportes de motores y el soporte de la rueda loca.

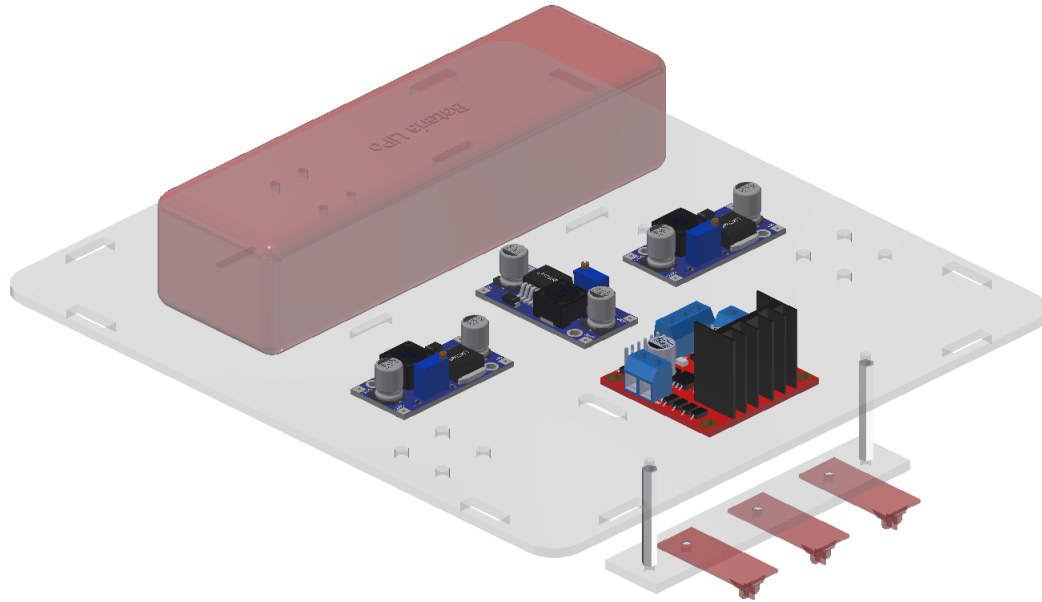


Figura 3.4: Plano y distribución de componentes sobre placa inferior.

Sobre la placa superior se encuentra el par estereoscópico y su soporte; y en la parte inferior de la tapa se encuentran anclados el arduino y la Raspberry junto a su base de montaje.

Entre la base superior e inferior se encuentran separadores de placa para asegurar estabilidad de la estructura, considerando además, que el espacio a disposición dentro del robot es suficiente para poder gestionar correctamente la conexión de cables para alimentar y controlar todo el sistema. El cofre del robot o caja está hecho de acrílico de 3 mm para la base y tapa y 4 mm de espesor para las paredes de la caja que permiten generar un ambiente cerrado y así evitar en la medida de lo posible la entrada de agentes nocivos del exterior como el polvo y objetos contundentes que dañen los componentes.

Para la sujeción de motores se usa un soporte impreso en ABS a medida, en la figura 3.6 se visualiza el modelado 3D del conjunto motor, soporte, rueda y acople.

### 3.2.1. Análisis de elemento finito

Un material sometido a cargas ya sea constantes o esporádicas sufre deformaciones; es por esto que para asegurar el correcto funcionamiento del robot se realiza un análisis de esfuerzos. El objetivo de este es asegurar que la pieza o conjunto de estudio será capaz de soportar las cargas o componentes del robot. Para esto se hace uso del software *Autodesk Inventor* que dentro de sus herramientas contiene la posibilidad de realizar análisis estático, y dentro de los resultados que entrega este análisis se encuentra la tensión de Von Misses.

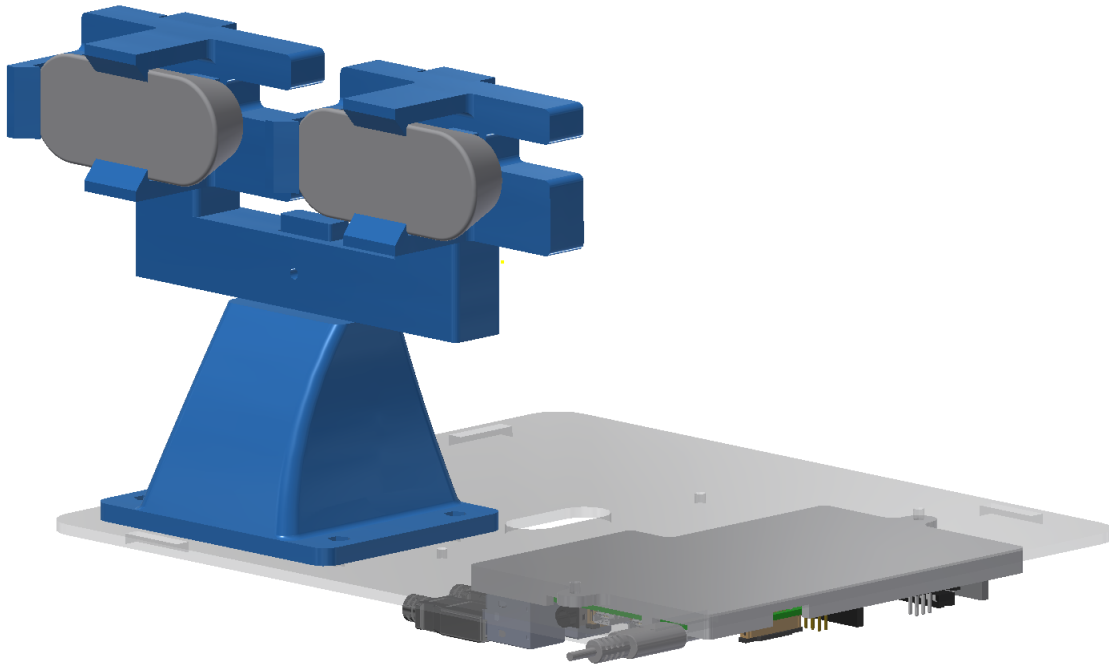


Figura 3.5: Plano y distribución de componentes en placa superior del robot.

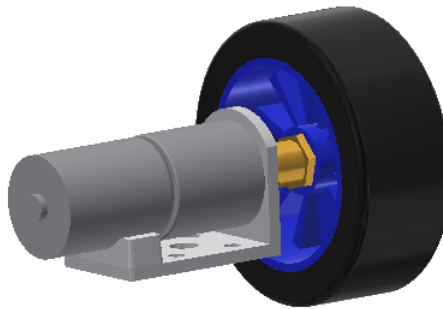


Figura 3.6: Modelo 3D del conjunto motor - rueda.

Para el estudio se considero el peso de los componentes que se montan sobre y dentro la caja del robot que se transformaron en cargas puntuales en diversas zonas del robot. Con una carga total aproximada de  $3\text{ kg}$ , distribuida sobre y dentro el robot; cada una de estas cargas ejerce una fuerza la cual está representada con fuerzas puntuales para cada componente involucrado. La fuerza total aplicada sobre el robot es de  $30\text{ N}$ . Y las restricciones utilizadas para el modelos consta de fijar las ruedas y restringir la unión de componentes de dentro del ensamble mediante el uso de restricción del tipo pasador, indicando que se encuentran unidas por un perno y tuerca o solo por tornillos. El resul-

tado que se esperaba era obtener una tensión máxima en los ejes de los motores, pero para sorpresa al encontrarse sometido a diversas cargas actuando en diversos puntos del robot, esta tensión máxima se ocasionó en la zona inferior de los separadores de de placas de apoyo, con una tensión máxima de  $3,39 \text{ MPa}$  que se encuentra muy por debajo de los límites de elasticidad de los materiales de los componente implicados: el aluminio y el acrílico, que cuentan con un límite de elasticidad de 68 y 49 MPa respectivamente. En la figura 3.7 se logra visualizar el análisis utilizando el criterio de Von Misses, indicando la magnitud y locación de la tensión máxima existente en el robot.

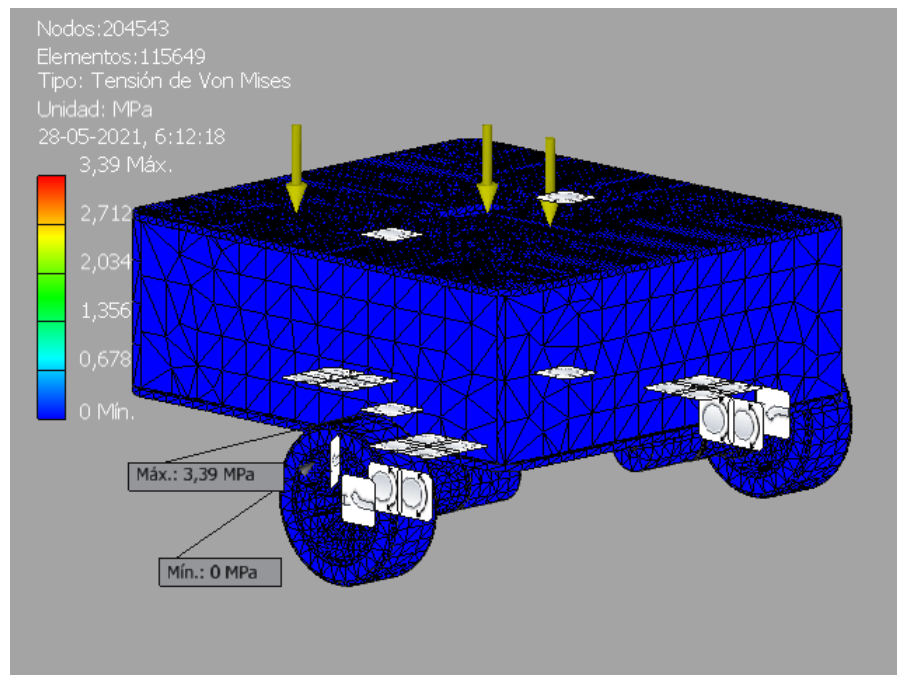


Figura 3.7: Análisis estático, tensión de Von Misses.

Gracias al análisis realizado anteriormente se puede concluir que la estructura logrará soportar las cargas para las cuales ha sido diseñado sin sufrir deformaciones en las partes que lo componen, ya sea una pieza de acrílico, aluminio o plástico ABS.



## 4. Diseño electrónico

El diseño electrónico implica la propuesta de un circuito de control y alimentación, seleccionando los componentes adecuados para un robot de transporte que sea capaz de moverse en un entorno controlado con ayuda de una guía.

Este apartado se divide en cuatro partes esenciales:

- **Requerimientos del sistema:**  
Corresponde al apartado en donde se establecen los lineamientos del sistema para el diseño a la medida de los circuitos. En este punto es importante establecer correctamente parámetros del robot considerando una propuesta de bajo costo para un primer prototipo.
- **Selección de componentes:**  
Según los requerimientos existentes se eligen componentes específicos para obtener un funcionamiento en condiciones de laboratorio. La selección indaga en alternativas dentro de las temáticas de autonomía, microcontroladores, sensores y motores.
- **Diagramas de conexión:**  
Una vez seleccionados los componentes se procede con los diagramas de conexión que buscan instruir a la hora de que sea necesario implementar el proyecto en cuestión. Se utilizan diagramas de bloques para facilitar la comprensión del funcionamiento de los circuitos y además se hace uso de *softwares* de simulación de circuitos electrónicos para corroborar el funcionamiento y obtener una visión temprana del comportamiento de los sistemas.
- **Escritura de programa de control de movimiento:**  
Para finalizar este apartado se explica y analiza el programa creado para realizar las labores de transporte del par estereoscópico.
- **Clasificación de calibre:**  
Para hacer visible los resultados obtenidos por los programas de detección de calibre se propone el diseño de un sistema de visualización mediante el uso de una pantalla LCD conectada a la *Raspberry Pi 4*.

### 4.1. Requerimientos del sistema

En primer lugar, se realiza un levantamiento de información el cual define las características principales o requisitos del sistema de medición móvil. Para esto, se ponen en la palestra las condiciones bajo las cuales estará sometido el robot.

Se espera que el robot se pueda movilizar a ras de piso dentro de un ambiente controlado sin irregularidades ni pendientes que dificulten el desplazamiento. Por otro lado, las condiciones de luminosidad en el ambiente se suponen ideales y de buena calidad e intensidad.

En otro ámbito, los requisitos del robot se centran en características del funcionamiento, entre estos se encuentran:

- La autonomía que se requiere es de 30 minutos, porque se estima realizar tres mediciones por minuto, logrando así un total de 90 mediciones aproximadamente.
- Se busca simplicidad y funcionalidad en el apartado de navegación autónoma; en el presente trabajo no se están realizando labores de investigación sobre nuevas formas de mejorar o implementar sistemas de navegación autónoma, sino más bien se está buscando diseñar un sistema funcional y simple que permita realizar las primeras pruebas de funcionamiento de los algoritmos de medición.
- El robot no superará los 5 *kg* de peso considerando el esqueleto o carcasa y componentes. El robot se encarga de transportar el par estereoscópico, por lo que su carga primordial es ésta; con esto se busca explicar que al utilizar sistemas simples de mover y controlar el carro móvil no se le deberían añadir componentes de gran tamaño y peso.
- Para una buena toma de mediciones se considera una baja velocidad de desplazamiento para detectar oportunamente los objetivos a estudiar.

## 4.2. Selección de componentes electrónicos

### 4.2.1. Selección de método de movimiento autónomo

Los estudios en los diversos tipos de robots utilizando las tecnologías disponibles siempre buscan indagar en nuevos sistemas que desplazan tecnologías usadas anteriormente incluso llegando a dejar de lado el control enfocado en un sólo objeto sino más bien enfocando en estudios de control de sistemas robóticos de agentes múltiples (MARS). [34] Lo que da para pensar que no es necesario indagar en nuevas tecnologías que cubran la necesidad de este proyecto, sino más bien se pueden utilizar tecnologías ya existentes, que cumplen con las necesidades básicas de este proyecto. La tónica que se ve en las diversas formas de controlar el movimiento dentro de un entorno controlado es la infaltable presencia de sensores que captan información del entorno.

La planificación de movimientos puede ser dividida de acuerdo con la información que el robot posee de su entorno. La planificación de movimientos con información local, o navegación reactiva, se destaca por la suposición de que la información sobre el espacio de trabajo es incompleta, es decir, la localización de los obstáculos se desconoce. El robot necesita contar con sensores (contacto, ultrasónico, láser, etc.) que le permitan la detección de los obstáculos y de esta forma evitar su colisión mientras avanza hacia el objetivo. En las primeras propuestas de este método [35], se considera al robot como un punto que se mueve en una escena bidimensional, en la cual no existen obstáculos que se interpongan en la trayectoria. Es por esto que ya se puede descartar el uso de varios sensores de proximidad, tacto y otros con la finalidad de detectar obstáculos. Las alternativas que se abordan a continuación son las alternativas disponibles para tratar y desarrollar durante este proyecto.

■ **Planificación de movimientos basado en algoritmos:**

El problema principal que resuelve dicho método es mover al robot desde un punto inicial a un punto final evitando colisiones con otros objetos dentro del propio espacio de trabajo; ya sean obstáculos de los cuales se conoce su posición como otros de los cuales se desconoce. Esto mediante la implementación de algoritmos complejos que aún se buscan optimizar y que dependen también mucho de la configuración o tipo de robot que se dispone. El grado de complejidad radica en la implementación de un programa de planificación de movimientos basado en algoritmos, que requiere un conocimiento mayor acerca de las diversas técnicas de control de movimiento como se mencionan en [36], dentro del apartado de taxonomía de los algoritmos de planificación; los algoritmos son en general computacionalmente complejos, logrando que se busque acelerar los procesos de búsqueda de caminos.

■ **Seguimiento de ruta previamente especificada:**

Dentro de este apartado existe el seguimiento de una ruta especificada previamente de forma física y tangible, mediante el uso de sensores que permiten el reconocimiento de este camino sin importar lo que exista en frente como obstáculo. Este método es ampliamente utilizado en competencias de robótica, específicamente en los eventos de competencias de robótica: seguidor de línea velocista. En donde existe un ambiente totalmente controlado sin perturbaciones en donde pueden desempeñarse los robots, tal como se realiza en el presente trabajo. La complejidad a la hora de realizar la programación es bastante baja, no requiere gran cantidad de poder de procesamiento y es de fácil y rápida implementación.

Tabla 4.1: Matriz de priorización para elección de control de movimiento de robot.

| Criterio                     | Basado en algoritmos | Seguimiento de línea |
|------------------------------|----------------------|----------------------|
| Dificultad de implementación | 4                    | 1                    |
| Versatilidad                 | 5                    | 2                    |
| Costo/beneficio              | 3                    | 4                    |
| Tiempo de ejecución          | 5                    | 1                    |
| TOTAL                        | 17                   | 8                    |

En la tabla 4.2 se utilizan criterios para definir qué tecnología es más recomendable abordar y utilizar en el diseño del robot móvil. Para esto se aplicaron cuatro criterios principales: dificultad de implementación, versatilidad, costo/beneficio y tiempo de ejecución de la estrategia. Para este caso en particular es recomendado utilizar el que menos puntaje total obtenga; esto se explica debido a lo que se ha mencionado durante todo el trabajo, y es que se consta de un ambiente totalmente controlado sin obstáculos y se necesita implementar rápidamente teniendo en consideración que el foco de investigación de este trabajo es la visión estereoscópica y la medición de calibre. De acuerdo a lo obtenido en la tabla 4.2 se concluye que la mejor alternativa es el uso de un seguidor de línea, teniendo presente que se debe intervenir el espacio en condiciones ideales adhiriendo a la superficie una cinta de color que sea de alto contraste en comparación al suelo para que el robot pueda reconocer la ruta que se debe seguir.

Tabla 4.2: Parámetros de funcionamiento del robot.  
Parámetros

|                      |                      |
|----------------------|----------------------|
| Velocidad del robot  | 5 cm/s               |
| Aceleración de robot | 10 cm/s <sup>2</sup> |
| Masa total del robot | 5 kg                 |
| Cantidad de motores  | 2                    |
| Radio de las ruedas  | 3,35 cm              |

Antes de comenzar con la selección de componentes del seguidor de línea es importante realizar selección de componentes de los motores a controlar y el controlador de motores.

#### 4.2.2. Selección de motores DC

Existen diversos tipos de motores pero dentro de la literatura de robótica móvil de tamaño medio y pequeños, se hace uso de motores DC (motorreductor). Para elegir este componente se utilizan parámetros y requerimientos del sistema robóticos que se presentan a continuación.

Debido a las bajas velocidades necesarias se escoge un motor con reductor asociado; los motorreductores comerciales existentes en el mercado se escogen por la velocidad y torque principalmente. El cálculo de velocidad asociado al motor necesario se rige según el siguiente ejercicio.

$$\begin{aligned}
 V &= 2 \cdot r_{rueda} \cdot \pi \cdot RPS \\
 RPS &= \frac{V}{2 \cdot r_{rueda} \cdot \pi} \\
 RPS &= 0,2375 \\
 RPM &= 14,26
 \end{aligned} \tag{4.1}$$

Donde  $V$  es la velocidad en cm/s,  $r_{rueda}$  es el radio de las ruedas y  $RPS$  son las revoluciones por segundo.

Al ser un motor tan lento se asegura el torque necesario para mover una masa de 5 kg.

En *Society of Robots* existe un parámetro que se denomina  $RMF$ , una forma simple para resumir las ecuaciones que desembocan en la elección de un buen motor para cada situación [37]. Asegurando una correcta elección de motor. Donde  $T$  representa el torque del motor,  $RPS$  las revoluciones por segundo a la cual giran las ruedas,  $M$  la masa total del vehículo,  $A$  y  $V$  la aceleración y velocidad del robot respectivamente. En resumen, El  $RMF$  resultante de las especificaciones del motor debe ser siempre mayor que el resultado  $RMF$  calculado con los requerimientos del robot.

$$\begin{aligned}
 \text{Motor RMF} &\geq \text{Resultado RMF} \\
 T \cdot RPS &\geq M \cdot A \cdot V \cdot \frac{1/\eta}{2 \cdot \pi} \\
 0,0225 &\geq 0,00725
 \end{aligned} \tag{4.2}$$

Por ende, el motor seleccionado es un motorreductor DC de 12V y 26 RPM que provee de un torque de  $9 \text{ kg} \cdot \text{cm}$ .

#### 4.2.3. Controlador de motores - puente H

Un microcontrolador es capaz de controlar la velocidad y sentido de giro de un motor, pero no se recomienda hacer por los bajos límites de corriente que soportan las salidas de estos. Por otro lado, una fuente de alimentación con mayor capacidad de entregar corriente es capaz de suplir la falencia que existe en los microcontroladores, pero estos no pueden controlar velocidad ni sentido de giro; entonces, ¿cómo se controla la velocidad y sentido de giro de un motor DC mientras recibe alimentación externa? Los controladores de motores tipo puente H, consisten en un arreglo de transistores que reciben señales de control que provienen de un microcontrolador y este arreglo de transistores da paso dependiendo de dichas señales a la alimentación de los motores. El puente H se encarga de alternar el sentido de giro y también de regular la velocidad de los motores DC conectados mediante polarización de alimentación y señales PWM respectivamente.

Existen alternativas comerciales, y para este caso es necesario basarse en la selección de los motores, utilizando principalmente los datos de consumo de corriente y voltaje de alimentación de los motores. Por esto se selecciona la placa *L298n* que posee doble puente h para controlar dos motores DC de entre 12 y 35 V a la vez con un consumo máximo de 2 A por motor. Considerando que según datasheet con carga se estima un consumo de 0,5 A y un consumo de 1,3 A en bloqueo.

#### 4.2.4. Seguidor de línea

Lo necesario para diseñar el seguidor de línea es la elección de un microcontrolador y sensores.

Al escoger el método por seguimiento de línea para el desplazamiento del robot se opta por sensores de bajo costo y existe la alternativa de sensores reflectantes infrarrojos. En este caso comercialmente existen circuitos que operan con el sensor *TCRT5000* [38], que es un sensor que consta de un emisor infrarrojo y un fototransistor.

El funcionamiento es bastante simple, el emisor infrarrojo emite una luz infrarroja que se refleja en la superficie la cantidad reflejada depende netamente del color al cual se está apuntando; si el color en donde está siendo reflejada la luz es más oscuro, la cantidad que recibe el fototransistor es baja. Por el contrario, si el color donde se refleja la luz emitida por el infrarrojo es más claro el fototransistor recibe una mayor cantidad de luz reflejada. Es este el principio de funcionamiento de los seguidores de línea.

#### 4.2.5. Microcontrolador

Para el caso de seleccionar un microcontrolador se debe tener en consideración las siguientes especificaciones. En la tabla 4.3 se muestran las entradas, salidas y puertos necesarios para lograr realizar las tareas de seguimiento de línea y comunicación con la *Raspberry Pi 4*.

Tabla 4.3: Componentes conectados a microcontrolador.

| Componente     | $V_{cc}$ | $IN_{dig}$ | $OUT_{dig}$ | $COM$ |
|----------------|----------|------------|-------------|-------|
| Encoder        | 3,3V     | 2          | 0           | -     |
| L298n          | 12       | 0          | 6 (2 PWM)   | -     |
| Sensor línea   | 3,3      | 3          | 0           | -     |
| Raspberry Pi 4 | 5        | -          | -           | USB   |

Considerando lo anterior, existen diversas alternativas que se ajusta a los requerimientos, pero al no necesitar un gran poder de procesamiento ni demasiadas entradas y salidas se opta por utilizar una placa *ARDUINO UNO* que funciona con una alimentación de 5 V y posee: 6 pines analógicos y 14 pines digitales de los cuales 6 soportan señales PWM.

#### 4.2.6. Alimentación de componentes

El robot necesita de diversos componentes electrónicos para poder cumplir su cometido, estos se clasifican en dos grandes secciones; control y alimentación. Dentro de cada circuito es necesario alimentar cada elemento que hay conectado.

Las cámaras web se alimentan al conectarse a la *Raspberry Pi 4*, los sensores infrarrojos se conectan al *Arduino UNO*, por lo tanto se alimentan desde este mismo microcontrolador, los motores se conectan directamente al controlador de motores L298n del cual se alimentan.

Por otro lado, la *Raspberry Pi 4*, el *Arduino UNO* y el controlador de motores necesitan ser alimentados por otra fuente, en este caso una batería, los voltajes de alimentación de cada componente son: 12 V para el controlador de motores, 7 V para el *Arduino UNO* y 5 V para la *Raspberry Pi 4*, proviene de una batería. Esta batería es capaz de entregar un voltaje pero para derivar de un voltaje a 3, que se utilizan tres reguladores de voltaje *LM2596*, configurados a la tensión necesaria de cada componente.

#### 4.2.7. Selección de batería

En comparación con las baterías de ácido de plomo, NiMh (níquel-metalhidruro) y NiCd (níquel-cadmio), la batería LiPo (polímero de litio) tiene muchas ventajas; como alta densidad de energía, baja tasa de autodescarga, alto voltaje de celda y larga vida útil. Estas ventajas hacen que las baterías LiPo se utilicen ampliamente en vehículos, bicicletas eléctricas, ordenadores portátiles y otros productos eléctricos. Baterías utilizadas en la mayoría de estos las aplicaciones mencionadas anteriormente se componen de largas cadenas de celdas de batería para lograr los voltajes operativos apropiados. [39]. Debido a estas grandes ventajas existentes es que se selecciona una batería LiPo.

Las especificaciones de esta batería son consecuencia de los componentes utilizados y que necesitan alimentación directa por parte de la batería. En este caso los componentes son: motores, *Raspberry Pi 4* y *Arduino UNO*. Siendo el voltaje más alto el que determina la cantidad de celdas a utilizar. Los motores DC funcionan a 12  $V_{DC}$  es por esto que la batería a utilizar debe ser de 4 celdas. La conexión de las celdas es en serie por lo que cada celda aporta 3,17 V teniendo un total suministrado por la batería de 14,8 V.

El consumo total que recaerá en la batería se visualiza en la tabla 4.4.

Tabla 4.4: Consumo máximo de componentes conectados a la batería.

|                     | Voltaje | Corriente | Potencia |
|---------------------|---------|-----------|----------|
| Arduino UNO         | 7 V     | 1 A       | 7 W      |
| Raspberry Pi 4      | 5 V     | 3 A       | 15,3 W   |
| Motor DC $\times$ 2 | 12 V    | 2,6 A     | 31,2 W   |
| Total               | -       | 6,6 A     | 53,5 W   |

Por lo tanto, según requerimientos de autonomía del robot se indica una durabilidad de 30 minutos, con el tiempo y el consumo se puede calcular la capacidad mínima requerida para la batería.

$$\begin{aligned} 6600 \text{ mA} \cdot 0,5 \text{ h} &= 3300 \text{ mAh} \\ 3300 \text{ mAh} \cdot 1,3 &= 4290 \text{ mAh} \end{aligned} \quad (4.3)$$

La ecuación 4.3 indica que la capacidad mínima de la batería a escoger debe ser de 3300 mAh asegurando los 30 minutos de autonomía. Por otro lado, una recomendación es que se reserve el 30 % de capacidad para el cuidado y buena mantención de la batería, por lo que se añade un factor de seguridad de 1,3, resultando finalmente una batería de 4290 mAh de capacidad.

Por último pero no menos importante, se debe seleccionar el parámetro **C** que mide la intensidad en función de la capacidad de la batería; la cual indica la máxima intensidad mantenida y segura de carga y descarga de la batería. Este valor C multiplica la capacidad de la batería para saber cual es la intensidad máxima mantenida que es segura proveer. Para este caso es necesario indicar que la corriente máxima mantenida es de 6600 mAh, por ende el parámetro que se busca debe ser mínimo de 2 esto asegurando que la capacidad de descarga sostenida logre ser mayor a 6600 mAh.

Con toda esta información y datos recaudados sobre la batería LiPo que se necesita, se concluye que la batería ideal es la *Turnigy 5000mAh 4S 25C*, que si bien se escapa un poco de los cálculos obtenidos, cumple con ellos y daría aproximadamente 7 minutos adicionales.

### 4.3. Diagramas de conexión

Para clarificar y dejar documentado cómo se realizaría la conexión y montaje de los componentes electrónicos se diseñan diagramas de conexión para las 2 distintas secciones en el área del diseño electrónico; circuito de alimentación y circuito de control. En la figura 4.1 se encuentra el conexionado de la batería a los reguladores de tensión que alimentan al *Arduino UNO*, la *Raspberry Pi 4* y el controlador de motores.

Por otro lado, en la figura 4.2 se ven las conexiones de control por entre el *Arduino UNO*, los sensores y el controlador de motores. Además, se establece la comunicación serial entre la *Raspberry Pi 4* y el *Arduino UNO* mediante una conexión USB.

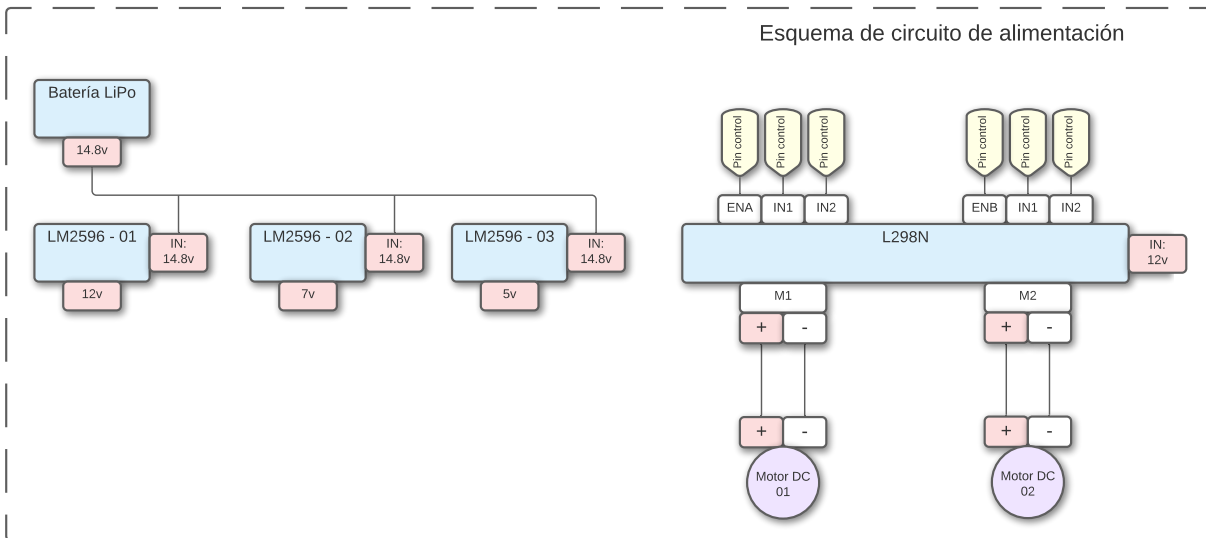


Figura 4.1: Diagrama de conexión para circuito de alimentación del robot.

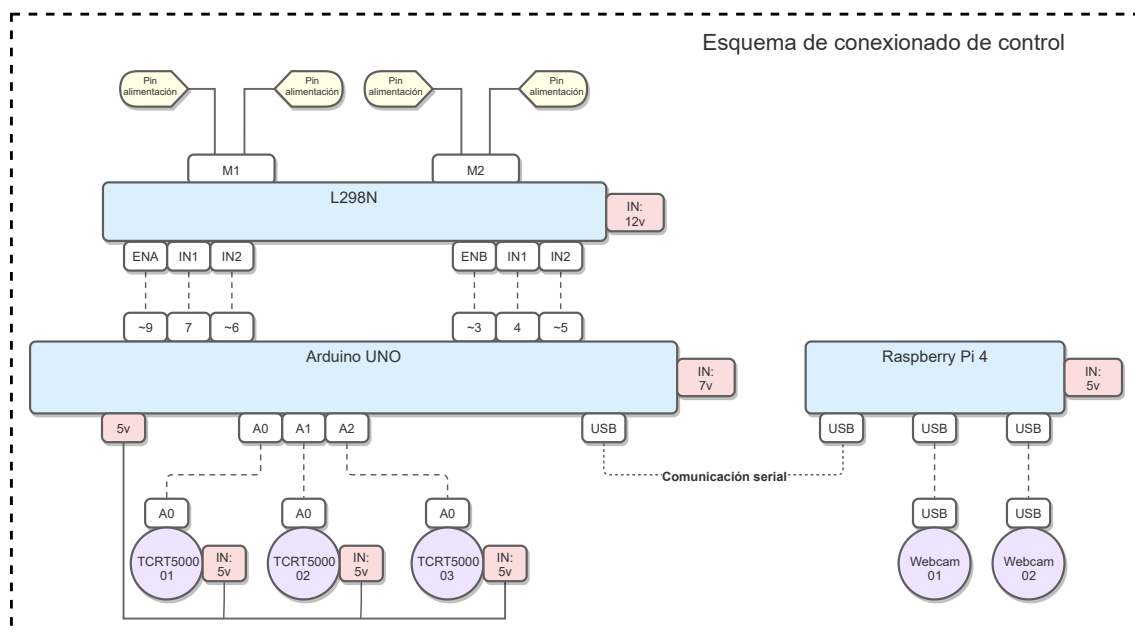


Figura 4.2: Diagrama de conexión para circuito de control del robot.

#### 4.4. Programa Arduino UNO

Para ejecutar las acciones de control del robot mediante el seguimiento de línea es necesario crear un programa para la placa *Arduino UNO* mediante el uso del *software Arduino IDE*. El programa completo se puede ver en el anexo 7.6, el objetivo de este programa es establecer la comunicación entre la *Raspberry Pi 4* y el *Arduino UNO*, esta comunicación es serial mediante USB en donde se envía una orden desde el programa de medición de calibre que indica si debe detenerse al detectar un objetivo de estudio y otra instrucción de proseguir en caso de terminar el análisis de calibre.



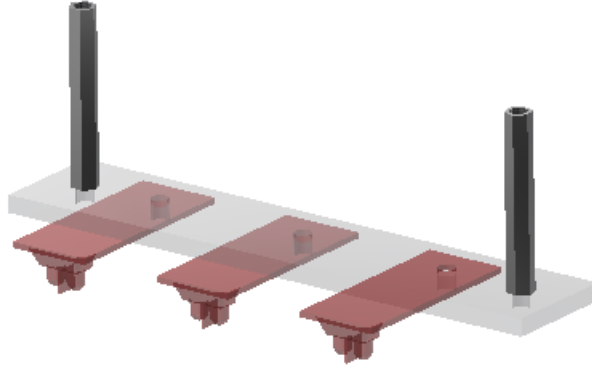


Figura 4.3: Montaje de sensores reflectivos infrarrojos.

Dentro del programa, como se visualiza en 4.1, existe en primer lugar la comunicación con la *Raspberry Pi 4* que entrega la instrucción de detención y avance; donde el avance se desarrolla siguiendo una línea o guía de alto contraste adherido al suelo.

```
1 void loop() {
2
3
4     if (Serial.available()) { //Si está disponible
5         char c = Serial.read(); //Guardamos la lectura en una variable char
6         if (c == 'H') { //Si es una 'H', comenzamos a seguir la linea
7             si=lectura_a(sensorI);
8             sc=lectura_a(sensorC);
9             sd=lectura_a(sensorD);
10
11             seguidor(si,sc,sd);
12         } else if (c == 'L') { //Si es una 'L', dejamos de seguir la linea
13             detencion();
14         }
15     }
16 }
```

Código 4.1: Seguidor de línea.

## 4.5. Clasificación de calibre

Para mostrar los resultados de la clasificación de calibre se opta por diseñar un circuito que incluye una pantalla LCD que se conecta a la *Raspberry Pi 4*. El montaje de este se realiza en la parte superior del robot anclado con separadores de placa al acrílico.

#### 4.5.1. Componentes

- **Pantalla LCD:**

La pantalla debe mostrar los resultados obtenidos por los algoritmos de obtención de diámetro del objetivo. Es por esto que no es necesario una pantalla con alta definición. Se propone el uso de la pantalla LCD LCM-S01602DTR [40] que se puede visualizar en la figura 4.5; esta permite la escritura de 32 caracteres, suficientes considerando que se mostrará el diámetro y su clasificación.

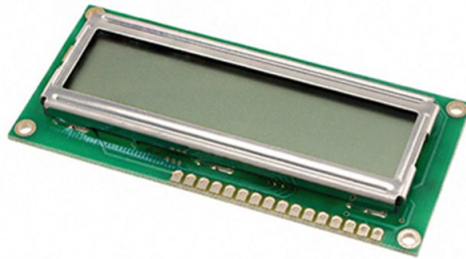


Figura 4.4: Pantalla LCD LCM-S01602DTR. Extraído de [41].

- **Módulo interfaz serial I2C** Con el fin de simplificar la conexión entre la *Raspberry Pi 4* y la Pantalla LCD se propone el uso de una placa que incorpora el circuito PCF8574A [42] que es un expansor de pines de entrada y salida de 8 bits para el bus I2C de dos líneas. Este módulo tiene como finalidad convertir la información de un protocolo de comunicación serial a información entregada en paralelo para la pantalla LCD, además de reducir la cantidad de cables que van desde la pantalla hacia el ordenador de placa reducida dejando disponibles los pines de salida digital de la *Raspberry Pi 4* disponibles.

#### 4.5.2. Comunicación I2C

La comunicación I2C, desarrollado por Phillips Semiconductors, es un protocolo de comunicación serial *half-duplex* sincrónica. I2C tiene la particularidad de que puede existir varios maestros controlando uno o varios esclavos. Esto esperando que el canal se encuentre disponible para transmitir información. La velocidad de transferencia de información es de 100 Kbits/s.

Las señales que componen este protocolo de comunicación son:

- **SCL:** *System Clock* línea que transmite los pulsos de reloj que sincronizan el sistema.

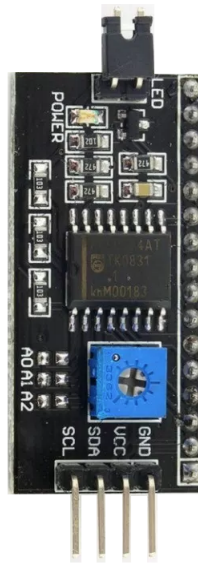


Figura 4.5: Módulo interfaz serial I2C. Extraído de [43].

- **SDA:** *System Data* línea de transmisión de datos entre maestros y esclavos.
- **GND:** común de la interconexión entre todos los dispositivos.

El SDA y el SCL son líneas de drenaje abierto, es decir, se deben polarizar en estado alto mediante conexión de resistencias “pull-up”.

El protocolo de comunicación indica que para establecer la comunicación entre los maestros y los esclavos se requiere primero que todo tener el bus libre para el envío de los datos. Pero, ¿cómo se reconoce hacia que esclavo está enviado el mensaje? Luego de iniciar la comunicación el bus se muestra ocupado al darse la condición de inicio mediante la secuencia de inicio. La comunicación serial comienza una vez enviado el primer *byte* que indica la dirección del esclavo al que va dirigido el mensaje y si la acción a realizar es de lectura o escritura; si la comunicación se establece el esclavo responde con un bit en estado bajo. Una vez realizada la confirmación comienza la transferencia de información. Esta información se detiene cuando es detectada la secuencia de parada o condición de paro. La trama de comunicación se muestra en la figura 4.6.

#### 4.5.3. Diagrama de conexión

Se presenta a continuación un diagrama que plasma el conexionado existente entre la *Raspberry Pi 4* y el módulo interfaz serial I2C, que a su vez se acopla directamente con sus pines a la pantalla LCD.

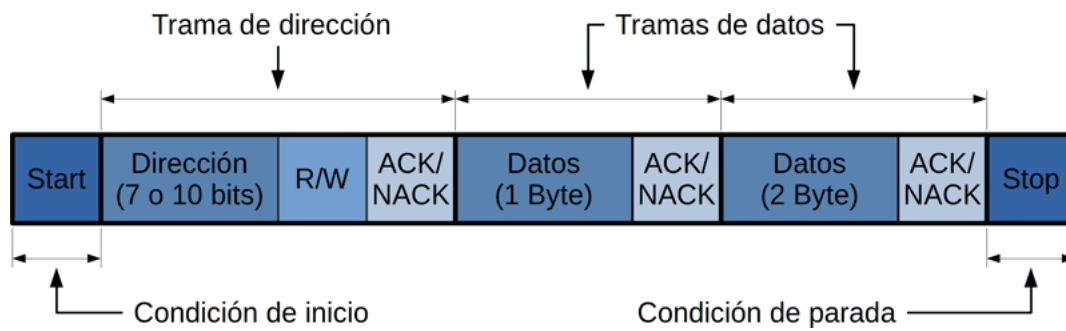


Figura 4.6: Estructura de un mensaje I2C. Extraído de [44].

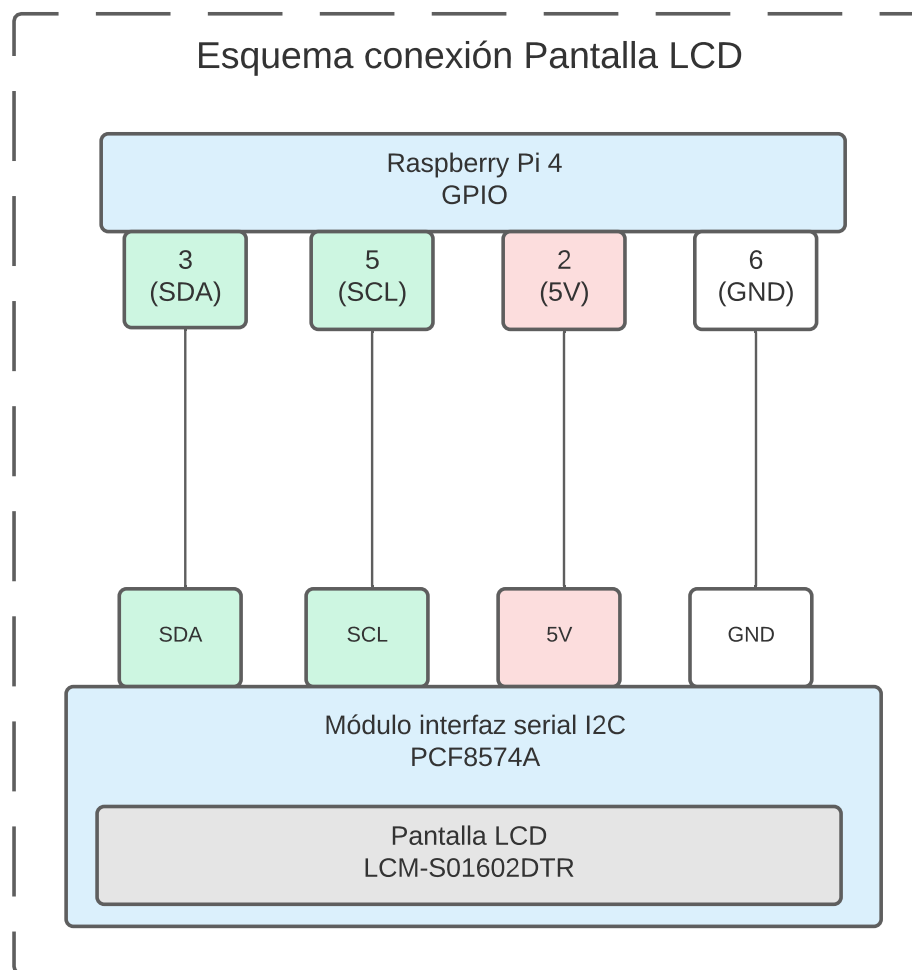


Figura 4.7: Diagrama de conexión para visualizar resultados en pantalla LCD.

#### 4.5.4. Programación

En primer lugar es importante realizar una activación del protocolo I2C dentro de la *Raspberry Pi 4* dentro de sus configuraciones de comunicación. Luego, se debe reconocer la dirección registrada por el esclavo una vez conectado y posteriormente, se procede a instalar la librería correspondiente llamada *I2C\_LCD\_driver*. Esta contiene

todas las funciones necesarias para entablar comunicación entre los dispositivos.

Finalmente, la inclusión de este apartado en los programas ya realizados de medición de calibre es simple, se añade la librería anteriormente instalada y luego se crea la función que se visualiza en el código 4.2; la función se encarga de clasificar el diámetro del tronco medido en las categorías visualizadas en la tabla 1.1, para luego enviar esta información en conjunto del diámetro al módulo interfaz serial I2C y este la decodifique para ser entregada a la pantalla LCD.

```

1  import I2C_LCD_driver
2
3
4  def mostrar_pantalla(diametro):
5      LCD=I2C_LCD_driver.lcd()
6      mensaje_diametro="Diámetro: %.2f" % diametro
7      str calibre
8      if diametro>=19 & diametro<=25:
9          calibre="6 - 8"
10     elif diametro>25 & diametro<=32:
11         calibre="8 - 10"
12     elif diametro>32 & diametro<=38:
13         calibre="10 - 12"
14     elif diametro>38 & diametro<=45:
15         calibre="12 - 14"
16     elif diametro>45 & diametro<=51:
17         calibre="14 - 16"
18     elif diametro>51 & diametro<=57:
19         calibre="16 - 18"
20     elif diametro>57 & diametro<=64:
21         calibre="18 - 20"
22     elif diametro>64 & diametro<=80:
23         calibre="20 - 25"
24     elif diametro>80 & diametro<=95:
25         calibre="25 - 30"
26     else:
27         calibre="error"
28
29     mensaje_calibre="Calibre: "
30
31     LCD.lcd_clear()
32     LCD.lcd_display_string(mensaje_diametro + "mm",1)
33     sleep(1)
34     LCD.lcd_display_string(mensaje_calibre + calibre,2)
35     sleep(1)

```

Código 4.2: Envío de información a pantalla LCD.

## 5. Resultados experimentales

### 5.1. Primer acercamiento de prueba a la programación con OpenCV y C++

En el Código 5.1 se visualiza cómo se implementó en primera instancia las funciones de *OpenCV* en *C++*.

```

1  #include "opencv2/highgui/highgui.hpp"
2  #include "opencv2/imgproc/imgproc.hpp"
3  #include <iostream>
4  #include <stdio.h>
5  #include <stdlib.h>
6
7
8  using namespace cv;
9  using namespace std;
10
11 Mat src; Mat src_gray;
12 int thresh = 100;
13 int max_thresh = 255;
14 RNG rng(12345);
15
16 /// Funcion principal
17 void thresh_callback(int, void*);
18
19 int main(int argc, char** argv)
20 {
21     src = imread("D:/OneDrive/OneDrive - Universidad de Talca/U/9no/Integra
22                 2/OpenCVRoot3/aguila.jpg");
23
24     cvtColor(src, src_gray, CV_BGR2GRAY);
25     blur(src_gray, src_gray, Size(3,3));
26
27     String name = "Imagen Original";
28     namedWindow(name, CV_WINDOW_AUTOSIZE);
29     imshow(name, src);
30
31     createTrackbar(" Canny thresh:", "Source", &thresh, max_thresh,
32                  thresh_callback);
33     thresh_callback(0, 0);
34
35     waitKey(0);
36     return(0);
37 }
38
39 void thresh_callback(int, void*)
40 {
41     Mat canny_output;
42     vector<vector<Point>> contours;
43     vector<Vec4i> hierarchy;
44
45     Canny(src_gray, canny_output, thresh, thresh * 2, 3);

```

```

45 findContours(canny_output, contours, hierarchy, CV_RETR_TREE,
46             CV_CHAIN_APPROX_SIMPLE, Point(0, 0));
47
48 Mat drawing = Mat::zeros(canny_output.size(), CV_8UC3);
49 for (int i = 0; i < contours.size(); i++)
50 {
51     Scalar color = Scalar(rng.uniform(0, 255), rng.uniform(0, 255), rng.
52     uniform(0, 255));
53     drawContours(drawing, contours, i, color, 2, 8, hierarchy, 0, Point()
54 );
55 }
56 namedWindow("Contornos de la imagen original", CV_WINDOW_AUTOSIZE);
57 imshow("Contornos de la imagen original", drawing);
58 }

```

Código 5.1: Código prueba en C++.

## 5.2. OpenCV y Python

### 5.2.1. Obtención de imágenes para la calibración de cámaras

Para poder realizar la medición estereoscópica y tener buenos resultados, es imprescindible realizar una calibración para la extracción y corrección de valores en las matrices intrínsecas de las cámaras web. Es por esto que a continuación se explicará el funcionamiento del código desarrollado para la obtención de imágenes para la calibración, para esto es necesario tener un tablero de ajedrez impreso que sea de  $9 \times 7$  cuadros.

En el código 5.2, se realiza la toma de fotografía del tablero, para este caso se toman 30 muestras para tener una buena calibración (esto se decide mientras se ejecuta el código), en las cuales se buscan las intersecciones del tablero o esquinas de cada cuadro.

```

1 retR, frameR= CamR.read()
2
3
4 retL, frameL= CamL.read()
5 grayR= cv2.cvtColor(frameR, cv2.COLOR_BGR2GRAY)
6 grayL= cv2.cvtColor(frameL, cv2.COLOR_BGR2GRAY)
7
8
9 #busca esquinas
10 retR, esquinaR=cv2.findChessboardCorners(grayR,(8,6),None)
11 retL, esquinaL=cv2.findChessboardCorners(grayL,(8,6),None)

```

Código 5.2: Búsqueda de esquinas en imágenes para la calibración.

En el código 5.3 se realiza la acción de determinar si la imagen está en buena calidad y contiene la información necesaria del tablero correcta, es decir, que las esquinas estén bien reconocidas. En caso de estar correcto se presiona la tecla *s* y para cancelar esa imagen se presiona la tecla *c*, esta imagen se guarda con un nombre el cual se va modificando con la toma secuencial de fotografías y el lugar de almacenamiento es la

misma carpeta donde se encuentra el código.

```

1
2 if (retR == True) & (retL == True):
3     esquina2R = cv2.cornerSubPix(grayR, esquinaR, (11,11), (-1,-1),
4         criteria)
5     esquina2L = cv2.cornerSubPix(grayL, esquinaL, (11,11), (-1,-1),
6         criteria)
7
8     #Dibujo sobre tablero
9     cv2.drawChessboardCorners(grayR, (8,6), esquina2R, retR)
10    cv2.drawChessboardCorners(grayL, (8,6), esquina2L, retL)
11    cv2.imshow('VideoR', grayR)
12    cv2.imshow('VideoL', grayL)
13
14    if cv2.waitKey(0) & 0xFF == ord('s'): # Presiona "s" para
15    guardar la imagen y "c" si no quieres guardar la imagen
16        str_nombre_img = str(nombre_img)
17        print('Images' + str_nombre_img + 'saved for right and left
18        cameras')
19        cv2.imwrite('chessboard-R'+str_nombre_img+'.png', frameR) #
20        Guardar la imagen donde el programa esta ubicado
21        cv2.imwrite('chessboard-L'+str_nombre_img+'.png', frameL)
22        nombre_img = nombre_img + 1
23    else:
24        print('Images not saved')
25
26    if cv2.waitKey(1) & 0xFF == ord(' '):
27        break

```

Código 5.3: Toma de imágenes en ventana emergente.

### 5.2.2. Cálculo de distancia

En el código 5.4, se realizan las manipulaciones necesarias en la imagen para encontrar los patrones que describen el tablero de ajedrez.

```

1
2 for i in range(0,30):
3     t = str(i)
4     ChessImaR = cv2.imread('Calibracion\chessboard-R'+t+'.png', 0)
5     ChessImaL = cv2.imread('Calibracion\chessboard-L'+t+'.png', 0)
6     retR, cornersR = cv2.findChessboardCorners(ChessImaR, (8,6), None)
7     retL, cornersL = cv2.findChessboardCorners(ChessImaL, (8,6), None)
8     if (True == retR) & (True == retL):
9         objpoints.append(objp)
10        cv2.cornerSubPix(ChessImaR, cornersR, (11,11), (-1,-1), criteria)
11        cv2.cornerSubPix(ChessImaL, cornersL, (11,11), (-1,-1), criteria)
12        imgpointsR.append(cornersR)
13        imgpointsL.append(cornersL)

```

Código 5.4: Calibración de las cámaras.



En el código 5.5 se obtienen los valores de las matrices intrínsecas, extrínsecas y valores de distorsión que ayudan para rectificar las imágenes y obtener mejores resultados utilizando visión estéreo.

```

1
2 retR , mtxR , distR , rvecsR , tvecsR = cv2.calibrateCamera(objpoints ,
   imgpointsR , ChessImaR . shape [ : : - 1 ] , None , None)
3 hR , wR = ChessImaR . shape [ : 2 ]
4
5 retL , mtxL , distL , rvecsL , tvecsL = cv2.calibrateCamera(objpoints ,
   imgpointsL , ChessImaL . shape [ : : - 1 ] , None , None)
6 hL , wL = ChessImaL . shape [ : 2 ]

```

Código 5.5: Determinación de los nuevos valores para los diferentes parámetros.

En el Código 5.6, se configuran los parámetros para la creación de un mapa de disparidad con el uso de la función *StereoSGBM*, que incluye un algoritmo para la detección de disparidad teniendo a disposición un par de imágenes estéreo rectificadas.

```

1
2 window_size = 3
3 min_disp = 2
4 num_disp = 130 - min_disp
5 stereo = cv2.StereoSGBM_create(minDisparity = min_disp ,
6     numDisparities = num_disp ,
7     blockSize = window_size ,
8     uniquenessRatio = 10 ,
9     speckleWindowSize = 100 ,
10    speckleRange = 32 ,
11    disp12MaxDiff = 5 ,
12    P1 = 8 * 3 * window_size ** 2 ,
13    P2 = 32 * 3 * window_size ** 2)

```

Código 5.6: Determinación de los nuevos valores para los diferentes parámetros.

La detección de profundidad se realiza mediante un doble click en la zona de interés dentro de una transmisión que se realiza en tiempo real, como consecuencia se indica la distancia que se encuentran las cámaras al objeto de estudio. Para realizar la transmisión en tiempo real se debe realizar en todo momento una rectificación de las imágenes obtenidas por las cámaras web, con los parámetros antes obtenidos gracias a la calibración realizada. Luego, se computa la información de profundidad para la imagen de profundidad que se verá como resultado final de este apartado. Después, se aplican filtros para eliminar ruidos dentro de la imagen final, se cambia el color para distinguir las profundidades, en donde se usan tonalidades azuladas. Finalmente, se muestra la imagen en donde se ve un mapa de profundidades en tiempo real.

```

1
2 while True:
3     retR , frameR = CamR.read()
4
5     retL , frameL = CamL.read()
6

```

```

7   Left_nice= cv2.remap(frameL, Left_Stereo_Map[0], Left_Stereo_Map[1],
cv2.INTER_LANCZOS4, cv2.BORDER_CONSTANT, 0)
8   Right_nice= cv2.remap(frameR, Right_Stereo_Map[0], Right_Stereo_Map[1],
cv2.INTER_LANCZOS4, cv2.BORDER_CONSTANT, 0)
9
10  grayR= cv2.cvtColor(Right_nice, cv2.COLOR_BGR2GRAY)
11  grayL= cv2.cvtColor(Left_nice, cv2.COLOR_BGR2GRAY)
12
13  disp= stereo.compute(grayL, grayR)
14  dispL= disp
15  dispR= stereoR.compute(grayR, grayL)
16  dispL= np.int16(dispL)
17  dispR= np.int16(dispR)
18
19  filteredImg= wls_filter.filter(dispL, grayL, None, dispR)
20  filteredImg = cv2.normalize(src=filteredImg, dst=filteredImg, beta=0,
alpha=255, norm_type=cv2.NORM_MINMAX)
21  filteredImg = np.uint8(filteredImg)
22  disp= ((disp.astype(np.float32)/ 16)-min_disp)/num_disp
23
24  closing= cv2.morphologyEx(disp, cv2.MORPH_CLOSE, kernel)
25
26  dispC= (closing-closing.min())*255
27  dispC= dispC.astype(np.uint8)
28  disp_Color= cv2.applyColorMap(dispC, cv2.COLORMAP_OCEAN)
29  filt_Color= cv2.applyColorMap(filteredImg, cv2.COLORMAP_OCEAN)
30
31  cv2.imshow('Filtered Color Depth', filt_Color)
32
33  cv2.setMouseCallback("Filtered Color Depth", coords_mouse_disp,
filt_Color)

```

Código 5.7: Visualización de mapa de disparidad para la detección de profundidad dentro de una imagen.

En el cálculo de distancia es necesario calcular de forma práctica mediante una serie de pruebas que se hacen del valor de la disparidad vs la distancia que se encuentra el objeto de estudio. Por esto, se crea una función a partir de las pruebas mencionadas, la evidencia de las pruebas y la función extraída se visualiza en la figura 5.1. En el Código 5.8, se ejecuta la acción de determinar la distancia mediante el uso de la función de cuarto orden encontrada. El comportamiento de la curva descrita por la función polinómica de cuarto grado dice que mientras menor es la distancia entre las cámaras y el objetivo la disparidad aumenta, lo que tiene sentido debido a que mientras más cerca un objeto a las cámaras, más píxeles son necesarios para representar a este dentro de una imagen, sin contar que se llega a un límite de acercamiento en donde el objeto deja de visualizarse en ambas imágenes. Este algoritmo trabaja de forma correcta entre los límites de 30 cm y 75 cm de distancia, considerando que fuera de estos límites no se asegura la obtención de resultados coherentes, considerando que el valor máximo de disparidad normalizado es de 1.

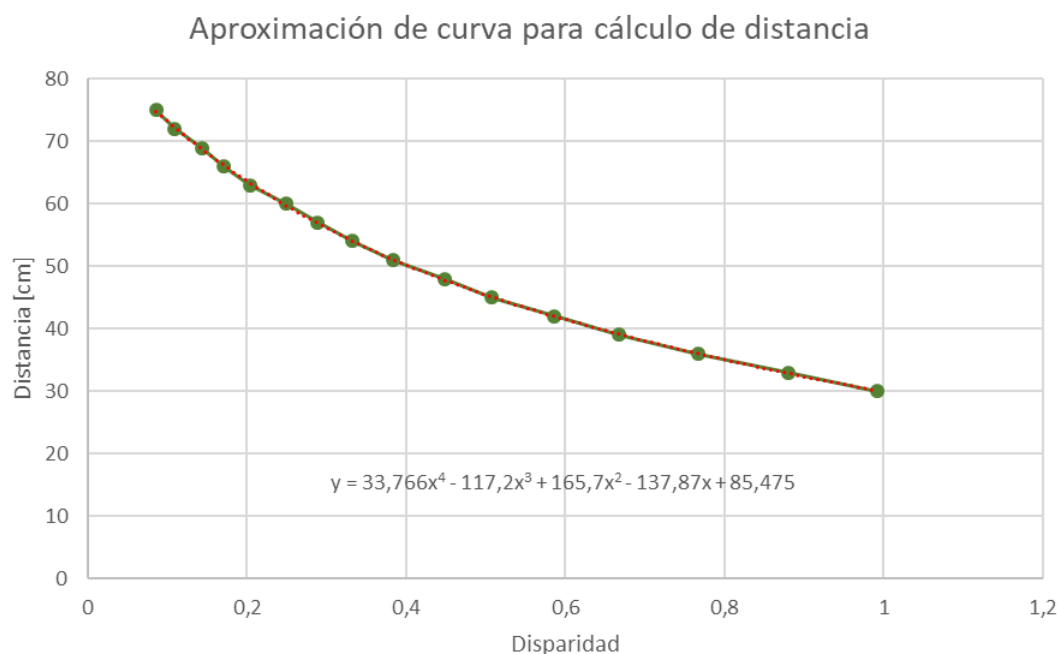


Figura 5.1: Curva de comportamiento y curva aproximada con la función de cuarto orden.

```

1
2 def coords_mouse_disp(event,x,y,flags,param):
3     if event == cv2.EVENT_LBUTTONDOWN:
4
5         average=0
6         for u in range (-1,2):
7             for v in range (-1,2):
8                 average += disp[y+u,x+v]
9         average=average/9
10        Distance= 33.766*average**(4) - 117.2*average**(3) + 165.7*
11        average**(2) - 137.87*average + 85.475
12        Distance= np.around(Distance,decimals=1)
13        print('Distance: ' + str(Distance)+' cm')

```

Código 5.8: Manipulación de información de disparidad y evaluación de esta en la función de cuarto grado para la obtención de distancia.

### 5.2.3. Cálculo de diámetro

En la siguiente secuencia se han descrito y explicado los códigos principales utilizados para la extracción del ancho o diámetro de un objeto.

```

1 imagen=cv2.imread("flauta3.jpg")
2 copia=np.copy(imagen)

```

Código 5.9: Código para seleccionar imagen a analizar.

En el Código 5.9, se ven los comandos para poder cargar una imagen cualquiera y una copia de esta para su posterior estudio.

```
1 canny_imagen=canny(copia)
```

Código 5.10: Comando *canny*.

La función que se ve en el Código 5.10 necesita como argumento la imagen cargada, en este caso la copia, para asegurarnos de no perder o modificar la imagen original en caso de cualquier falla. Esta función modifica la imagen que recibe para poder aplicar el algoritmo de Canny que se encarga de encontrar bordes. Esta función ha sido creada y las operaciones que realiza se pueden visualizar en el Código 5.11.

```
1 def canny(imagen):
2     gris=cv2.cvtColor(imagen, cv2.COLOR_RGB2GRAY)
3     blur=cv2.GaussianBlur(gris, (5,5), 0)
4     canny=cv2.Canny(blur, 50, 150)
5     return canny
```

Código 5.11: Función *Canny*.

En la línea 2 del Código 5.11 se puede ver que para trabajar esta imagen para encontrar bordes que establezcan el ancho del objeto de estudio, se le debe transformar a escala de grises. Luego, en la línea 3 se aplica un desenfoque Gaussiano que se encarga de suavizar la imagen y los bordes para hacer un poco más efectiva la detección de bordes hecha por la función *cv2.Canny* aplicada en la línea 4. Finalmente, se retorna *canny* que es una imagen que contiene los bordes encontrados de la imagen original.

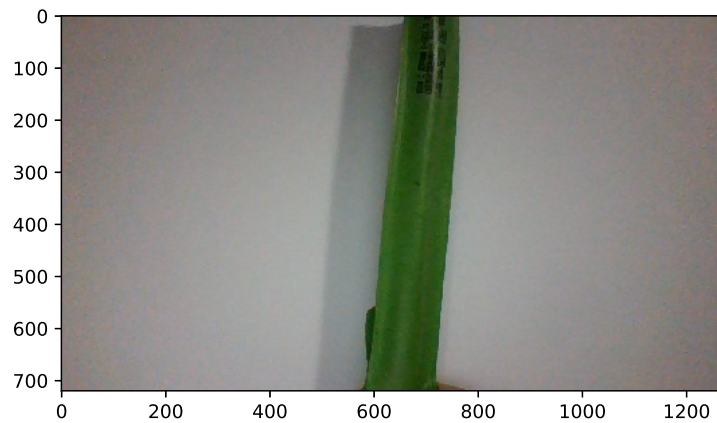
Al tener los bordes detectados se debe empezar con los filtros para que las líneas generadas estén más marcadas y solo queden las líneas de la figura de estudio. Para esto se usa la función *HoughLinesP* que se puede visualizar en el Código 5.12, en el cual se realiza un barrido píxel a píxel en busca de líneas de un largo determinado y separadas por cierta cantidad máxima de píxeles determinada por los dos últimos argumentos de esta función. Luego, al obtener estas líneas se crea una máscara con las líneas encontradas, la función *mostrar\_lineas* se puede visualizar en el Código 5.13.

```
1 lineas=cv2.HoughLinesP(canny_imagen, 1, np.pi/180, 115, np.array([]),
2   minLineLength=100, maxLineGap=10)
3 lineas_imagen=mostrar_lineas(copia, lineas)
```

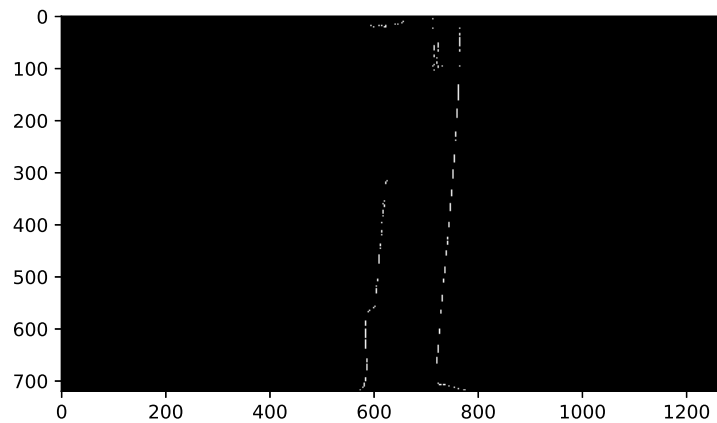
Código 5.12: Código para la búsqueda de líneas dentro de la imagen *Canny*.

```
1 def mostrar_lineas(imagen, lineas):
2     linea_imagen=np.zeros_like(imagen)
3     if lineas is not None:
4         for linea in lineas:
5             x1, y1, x2, y2=linea.reshape(4)
6             cv2.line(linea_imagen, (x1, y1), (x2,y2), (255, 255, 255), 1)
7     return linea_imagen
```

Código 5.13: Función *mostrar\_lineas*.



(a) Imagen Original.



(b) Bordes encontrados por el algoritmo de Canny de la imagen original.

Figura 5.2: Imágenes de prueba obtenidas.

El resultado de aplicar la búsqueda de líneas es lo que se ve en la figura. 5.3.

Llegando a este punto aún no se ha encontrado la distancia entre los píxeles de forma que entregue información de tamaño y antes de eso ya se pueden visualizar los resultados de todo el procesamiento que se ha realizado. En primer lugar como se ve en el código 5.14, existe una función llamada *cv2.cvtColor* que se encarga en este caso de transformar la copia de la imagen original de un formato RGB (Red, Green, Blue es un modelo de color basado en la síntesis aditiva, con el que es posible representar un color mediante la mezcla por adición de los tres colores de luz primarios) a BGR (que es lo mismo que RGB solo que invirtiendo el orden de lectura de los colores a Blue, Green,

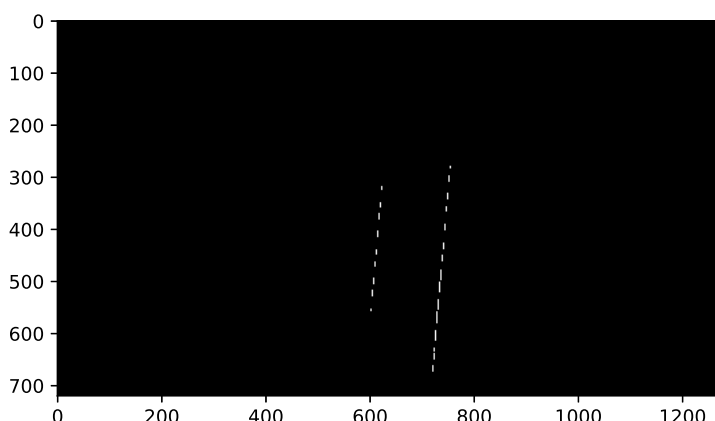


Figura 5.3: Líneas detectadas gracias a las funciones en el Código 5.12.

Red), que es el formato con el que trabaja la librería *matplotlib.pyplot* para graficar imágenes de forma que sea muy fácil su manipulación para hacer zoom y selección de zonas de interés. Cabe mencionar que RGB y BGR son espacios de colores que son utilizados para la lectura o escritura de estas, ya que cada imagen cuenta con muchos píxeles y cada uno cuenta con un vector que indica el color del píxel, este vector se puede escribir en RGB, BGR y otros. Sabiendo esto se puede explicar que RGB es Red Green Blue que es el orden en el cual se lee o crea el vector de cada píxel y BGR es Blue Green Red. Debido a lo anterior explicado se debe hacer la transformación para una correcta visualización fiel a los colores originales. Posteriormente, luego de hacer la transformación se hace la superposición de imágenes, en este caso de las líneas sobre la copia de la imagen original. El resultado se puede ver en la figura 5.4.

```

1 copia_BGR=cv2.cvtColor(copia, cv2.COLOR_RGB2BGR)
2
3 superposicion=cv2.addWeighted(copia_BGR, 0.8, lineas_imagen, 1, 1)

```

Código 5.14: Realización de la superposición y cambio de color.

A partir de acá, se definiría como una segunda parte del procesamiento de imagen, la extracción de datos, como lo es la distancia entre píxeles que describen los bordes del objeto de estudio. Desde este punto se trabaja con la imagen que se crearon mediante el código 5.12.

El código 5.15, se encarga de encontrar un par de píxeles blancos por cada fila de la imagen, esto se logra realizando un barrido por la imagen a la cual se le extraen las dimensiones. Este par de píxeles por fila encontrado se guardan por separado en dos distintos arreglos y así trabajarlos y analizarlos más cómodamente. El resultado de este procedimiento son 4 arreglos; dos arreglos que simbolizan las columnas y las otras 2 sus

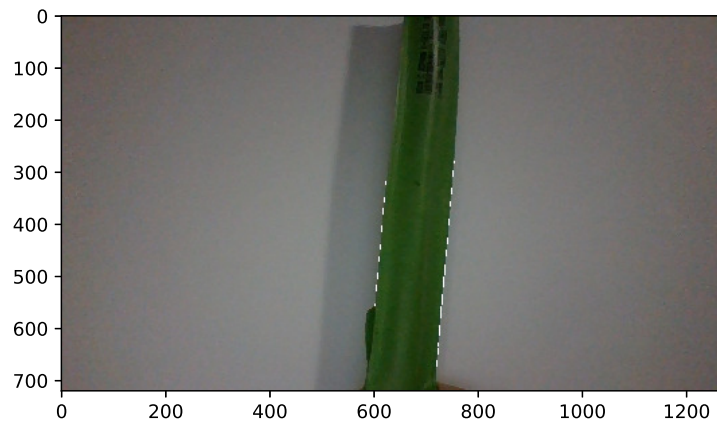


Figura 5.4: Resultado de la superposición de imágenes.

respectivas filas, en sí cada columna y fila indica una coordenada de dónde se sacó ese píxel. El resultado de esto en el ejemplo estudiado anteriormente es de 304 elementos dentro de cada uno de los cuatro arreglos.

```

1 alto=np.size(copia_linea , 0)
2 ancho=np.size(copia_linea , 1)
3
4 gris_linea=cv2.cvtColor(copia_linea , cv2.COLOR_RGB2GRAY)
5
6 for i in range(alto):
7     for j in range(ancho):
8         if gris_linea[i][j] == 255:
9             cant_por_fila=cant_por_fila+1
10            if cant_por_fila == 1:
11                col_izq.insert(1, j)
12                linea1.insert(1, i)
13                l=l+1
14            elif cant_por_fila==2:
15                linea.insert(k, i)
16                col_der.insert(k, j)
17                k=k+1
18            cant_por_fila=0

```

Código 5.15: Búsqueda de píxeles blancos.

Al tener la cantidad y ubicación de cada píxel se debe tener la precaución de que todos sean datos representativos y válidos para los estudios, por esto se hace un filtro en el cual se eliminan los pares de píxeles que sean vecinos dentro de una fila. El código 5.16 explica mediante un algoritmo lo anterior descrito y el resultado aplicando este algoritmo en el ejemplo de imagen tomada anteriormente se tiene que las dimensiones

de los arreglos es de 238, queriendo decir esto que se han eliminado 66 elementos de cada arreglo.

```

1 for i in range(len(linea)):
2     for j in range(len(linea1)):
3         if linea[i] == linea1[j]:
4             linea1_coincidencia.insert(m, linea1[j])
5             col1_coincidencia.insert(m, col_izq[j])
6             m=m+1
7     valor=col_der[i]-col1_coincidencia[i]
8     if valor!=1:
9         columna2.append(col_der[i])
10        columna1.append(col1_coincidencia[i])
11        fila1.append(linea1_coincidencia[i])
12        fila2.append(linea[i])
13        sumader_prom=col_der[i]+sumader_prom
14        sumaizq_prom=col1_coincidencia[i]+sumaizq_prom

```

Código 5.16: Eliminación de píxeles vecinos dentro de una fila de la imagen.

En el código 5.16, también se ha incluido el cálculo del promedio en cuanto a la posición del píxel. Para esto se obtuvo que el promedio de las líneas ubicadas a la izquierda del objeto de estudio dentro de la imagen es de 612.5 y a la derecha un promedio de 740,1974789915967 esto determinado por el código 5.17. Finalmente, se puede realizar el cálculo de distancia aproximada restando ambos promedios y obteniendo un ancho o diámetro del objeto en la imagen de 127.69747899159665 píxeles.

```

1 prom_izq=sumaizq_prom/(len(columna1))
2 prom_der=sumader_prom/(len(columna2))
3 distancia_pixel=prom_der-prom_izq

```

Código 5.17: Resultado promedio de la posición de cada línea vertical en la imagen.

Al tener el valor promedio de distancia entre los bordes laterales de una imagen se debe realizar alguna comprobación de que estos datos son representativos dentro de las pruebas. Para esto se hace uso de la desviación estándar, que busca eliminar los datos atípicos dentro de los bordes tomados por el algoritmo.

Para esto se hace uso de la ecuación 5.1.

$$DE = \sqrt{\frac{\sum |x - \mu|^2}{N}} \quad (5.1)$$

La ecuación 5.1 se aplica dentro del algoritmo como una función, recibiendo como argumento los datos de una columna o bordes detectado y el promedio de estos datos obtenidos, como se puede visualizar en el código 5.18.

```

1 desviacion_izq=desviacion(columna1, prom_izq)
2
3 desviacion_der=desviacion(columna2, prom_der)

```



Código 5.18: Aplicación de función desviación.

```

1 def desviacion(columna, prom):
2     suma_des=0
3     for i in range(len(columna)):
4         suma_des=suma_des+abs(columna[i]-prom)**2
5     desv=suma_des/len(columna)
6     desviacion_final=desv**0.5
7     return desviacion_final

```

Código 5.19: Función desviación.

Ahora para estandarizar el valor de las muestras se usa el valor Z que mide la diferencia entre un estadístico observado y su parámetro hipotético de población en unidades de la desviación estándar. El nivel de confianza utilizado para realizar la comparación si los valores son atípicos o no, es cuando los valores están dentro o fuera del rango  $-1,65$  a  $1,65$ . El algoritmo de aplicación se puede ver en el Código 5.20

```

1 def filtro(columna, columna1, fila, fila1, prom, desviacion):
2     col11=[]
3     fila11=[]
4     col22=[]
5     fila22=[]
6     for i in range(len(columna)):
7         aux=(columna[i]-prom)/desviacion
8         if aux<1.65 and aux>-1.65:
9             col11.append(columna[i])
10            fila11.append(fila[i])
11            col22.append(columna1[i])
12            fila22.append(fila1[i])
13     return col11, fila11, col22, fila22
14
15 fil_col11, fil_fila11, fil_col21, fil_fila21=filtro(columna1, columna2,
16            fila1, fila2, prom_izq, desviacion_izq)
17
18 fil_col21, fil_fila21, fil_col11, fil_fila11=filtro(columna2, columna1,
19            fila2, fila1, prom_der, desviacion_der)
20
21 "Calculo de promedio final"
22 suma_izq=0
23 suma_der=0
24 for i in range(len(fil_col11)):
25     suma_izq=suma_izq+fil_col11[i]
26     suma_der=suma_der+fil_col21[i]
27 prom_izq=suma_izq/len(fil_col11)
28 prom_der=suma_der/len(fil_col21)
29 distancia_pixel=prom_der-prom_izq

```

Código 5.20: Función para eliminación de datos atípicos.

Utilizando los arreglos que resultan del Código 5.20, se recalcula el promedio de los valores de los píxeles que representan los bordes del objeto de estudio, para luego realizar la resta de estos valores obtenidos, teniendo como resultado final el valor del

ancho del objeto en píxeles (*distancia\_pixel*).

Finalmente, a partir de pruebas realizadas con un objeto de medidas conocidas y distancias conocidas, se extrae una función la cuál se puede visualizar en la figura 5.5, que describe el tamaño en milímetros representado por un píxel dentro de una imagen a diversas distancias. En donde la línea y puntos verdes representan los valores obtenidos con la recolección de datos y la línea punteada de color rojo es la curva que describe la función polinómica aproximada.

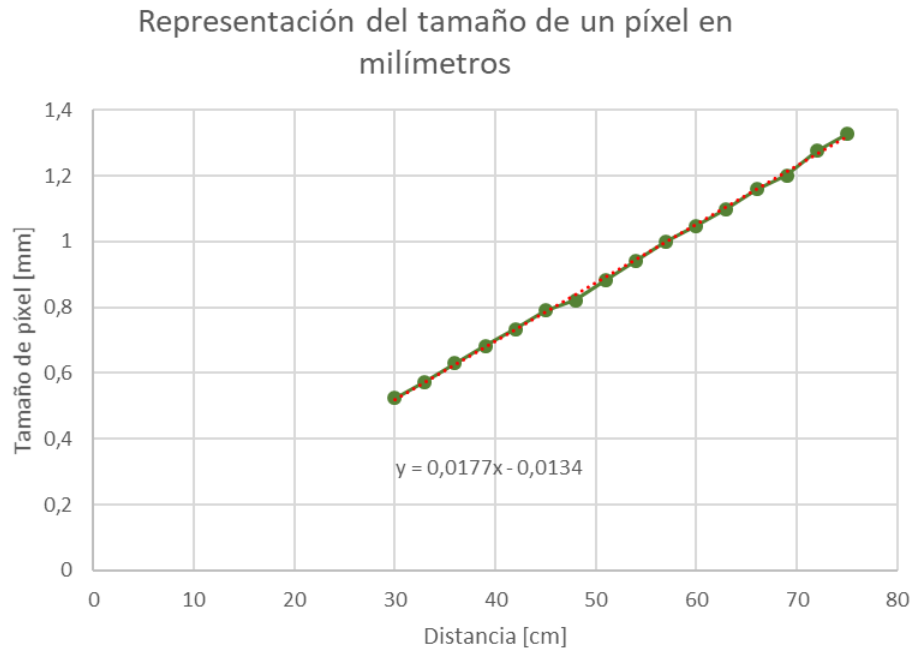


Figura 5.5: Gráfico para la aproximación de una curva y su función.

Entonces, con la función obtenida a partir de la curva aproximada es utilizada en el Código 5.21, para entregar el valor del diámetro del objeto en milímetros luego de haber entregado una distancia a la que se encuentra el objeto de forma manual, pero que es calculada y medida mediante visión estéreo.

```

1 distancia_stereo=float(input("Introduce la distancia en [cm] con 2
2   decimales: "))
3 distancia_stereo=distancia_stereo*10
4 tamaño_pixel=0.0012*distancia_stereo+0.0229
5 diametro=tamaño_pixel*distancia_pixel
6 print("El diametro del tallo es: \n", diametro)

```

Código 5.21: Transformación de medición de unidad de píxel a milímetros.

### 5.3. Implementación de sistema de prueba para visión estereoscópica

Momentáneamente, se dispone de 3 programas con extensión *.py*, que quiere decir que son 3 códigos por separado que en su conjunto de forma manual llegan a entregar un resultado final, que en este caso es el diámetro de un objeto.

En primer lugar, se debe armar el sistema de sujeción creado para las cámaras. Ver figura 5.6.



Figura 5.6: Montaje de cámaras en sistema de sujeción.

Luego, se calibran las cámaras obteniendo imágenes de un tablero de ajedrez, en este caso se capturan 30 imágenes para poder extraer y ajustar las matrices intrínsecas de las cámaras para la captura de imagen. Ver figura 5.7.

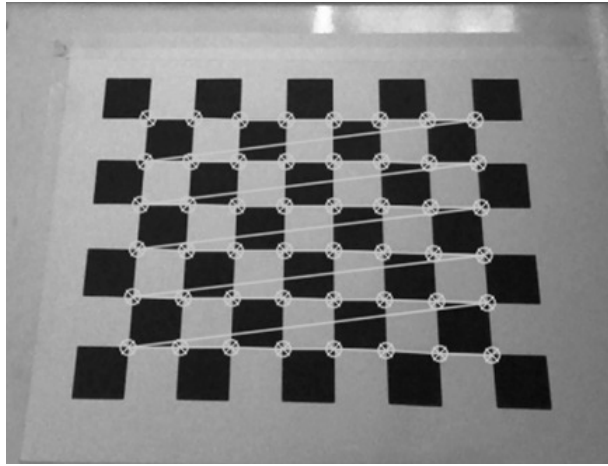


Figura 5.7: Imagen para la calibración de las cámaras.

Después, se monta un escenario de pruebas para extraer mediciones como se visualiza en la figura 5.8, cabe mencionar que para este caso en especial por las condiciones del lugar de pruebas (dormitorio), hay buena iluminación y no hay muchos objetos que alteren las mediciones, para información del lector se da a conocer que el diámetro del objeto de estudio es de  $16[mm]$ , medido con un calibre o pie de metro.

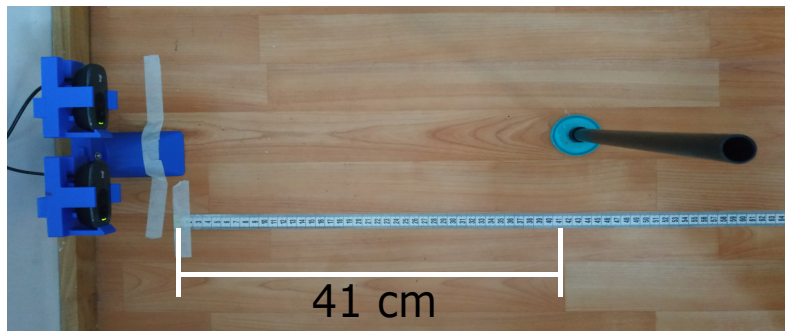


Figura 5.8: Disposición de materiales en el espacio físico de pruebas.

Se ejecuta el código de obtención de profundidad en el cual se hace doble click de forma manual en la ventana emergente generada marcando donde está el objeto de interés de estudio. En el caso de la figura 5.9 se simboliza el punto marcado con un punto rojo.

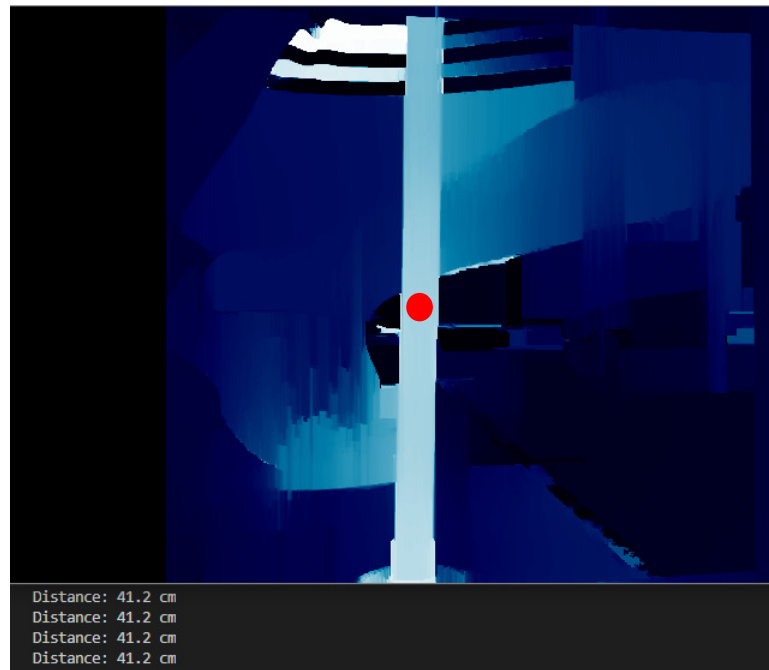


Figura 5.9: Mapa de profundidad que entrega la distancia de las cámaras al objeto de estudio.

Finalmente, se utiliza el código de detección de bordes y diámetro y se obtiene lo que se puede visualizar en la figura 5.10, que muestra efectivamente que el diámetro del objeto de estudio es de  $16[mm]$ .

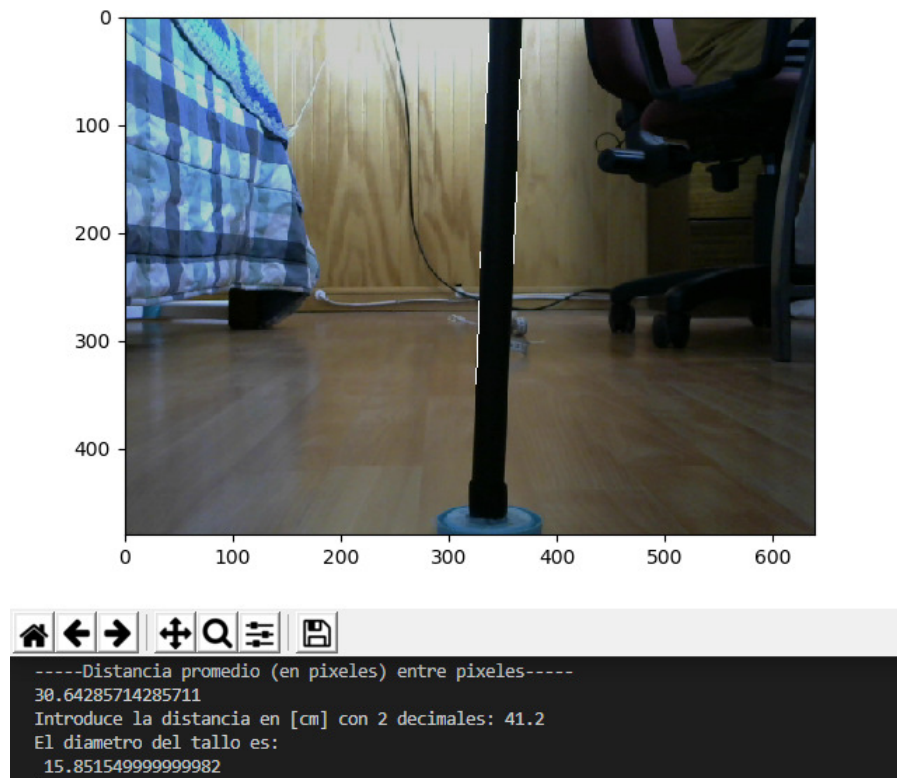


Figura 5.10: Resultado final, listo para la clasificación.

## 5.4. Resultados finales aplicados en Raspberry Pi 4

Una vez aplicados los cambios a los algoritmos y puestos a prueba en una *raspberrypi 4*, se tiene una situación de estudio similar a la expuesta en la sección anterior. En la figura 5.11 se ve la disposición de las cámaras, simulando estar sobre una plataforma en altura, como si estuviese montada sobre el robot móvil diseñado. El objeto de análisis es nuevamente una vara de color negro de diámetro 16 milímetros.



Figura 5.11: Situación de estudio.

En la figura 5.12 se muestra el resultado obtenido por la detección de objetivos, cabe mencionar que para efectos de mostrar evidencias sobre esta operación es que se extrae esta imagen, en el caso práctico se configura de forma predeterminada que no se muestre ningún resultado gráfico.

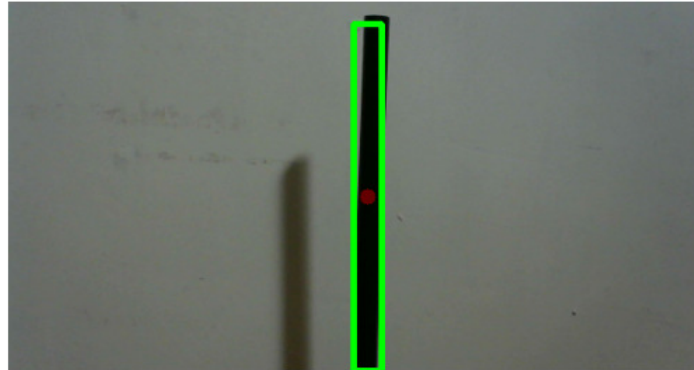


Figura 5.12: Resultado de la detección de objetivo y ubicación aproximada de su centro.

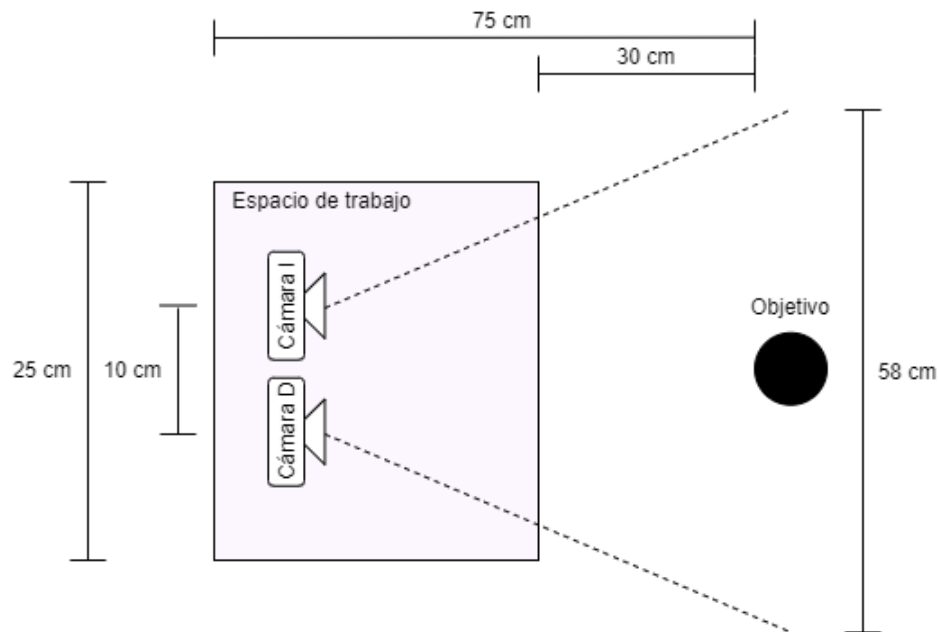


Figura 5.13: Espacio de trabajo del sistema estéreo para correctas mediciones.

Para evidenciar resultados, se acudió a mostrar imágenes nuevamente, teniendo en cuenta que el código final no incluye dicha acción. En las figuras 5.14 y 5.15 se muestra el resultado final y mediante un análisis de un grupo de resultados se obtuvo que el error de medición es de aproximadamente  $\pm 500 \mu m$ , considerando que el espacio de trabajo en donde puede situarse el par estereoscópico respecto al objetivo es el visualizado en la figura 5.13, que describe una superficie de  $45cm \times 25cm$ .



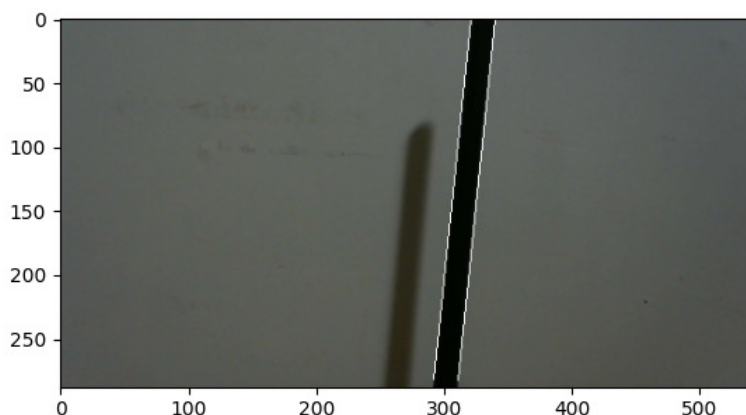


Figura 5.14: Resultado de aplicación de algoritmos en Raspberry Pi 4.

```

El centro es: [311, 157]
Distancia: 47.9 cm
La distancia es: 47.94818970513058
-----Distancia promedio (en pixeles) entre pixeles-----
19.0512820512821
-----Tamaño del pixel-----
0.8398384925975753
El diametro del tallo es:
16.423785580285752

```

Figura 5.15: Mensajes finales del resultado de aplicación de algoritmos en Raspberry Pi 4.

Los programas utilizados para trabajar dentro de la *raspberry pi 4*, se pueden revisar en detalle dentro del anexo E.

## 5.5. Aplicación de movimiento a robot móvil

El movimiento del robot se realiza mediante el uso de los motores DC seleccionados, estos se activan a través de del controlador de motores que a su vez recibe las señales de control desde el *Arduino UNO*. como se explica en la sección 4.3 mediante el uso de diagramas de conexión. Las órdenes las envía el ordenador de placa reducida al detectar un objetivo (detener) y cuando termina de analizar el objetivo (avanzar). Para visualizar cómo resultaría de forma práctica la ejecución de ese algoritmo de hace uso del *software Proteus* en su versión 8, el cual tiene la funcionalidad de realizar simulaciones de circuitos y añadir librerías con elementos que de serie no existen en el programa.

La simulación que se visualiza en las figuras 5.16, 5.17 y 5.18 muestra el resultado de las diversas respuestas del circuito ante las distintos estímulos de los sensores reflectivos infrarrojos que son representados por potenciómetros simbolizados con *SI* (sensor izquierdo), *SC* (sensor central) y *SD* (sensor derecho). Además, para dar la orden entregada por la *Raspberry Pi 4* en los programas de medición de calibre, se incorpora una terminal serial que simula ser la *Raspberry* comunicándose de forma serial con el *Arduino UNO*, con la diferencia de que en el caso real se hace a través de cable USB y



no mediante cables conectados a los pines de ambas placas.

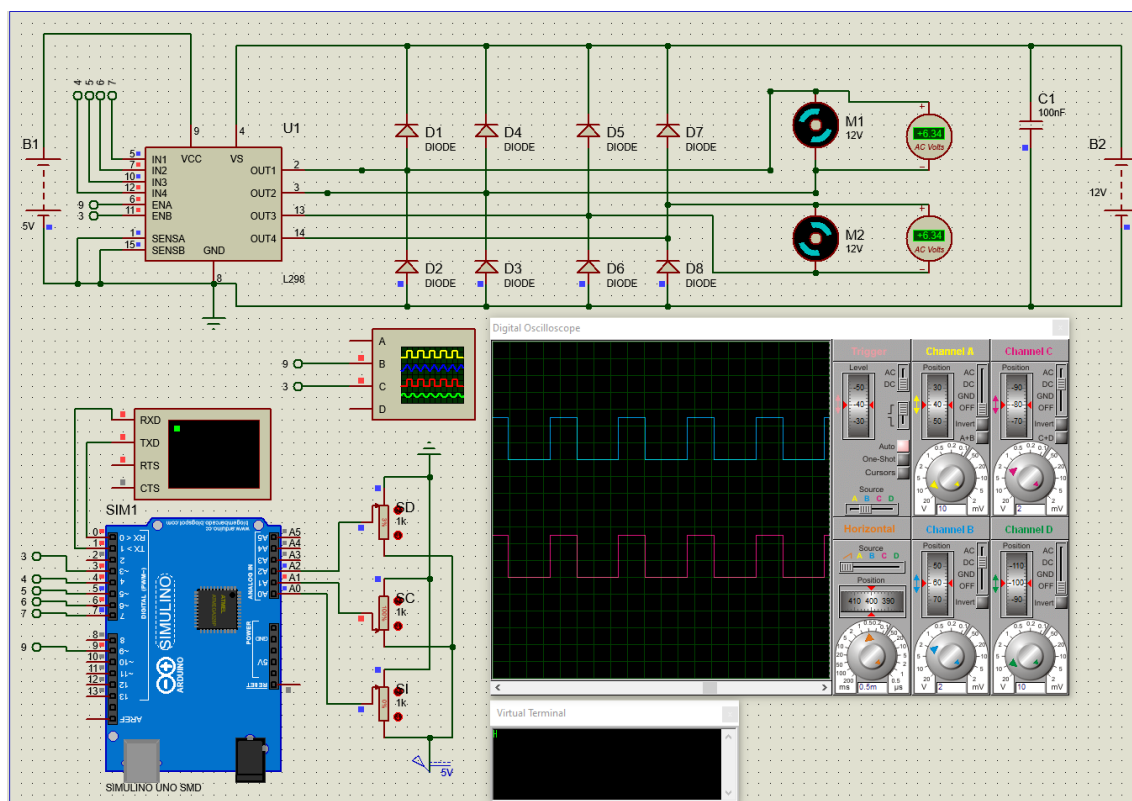


Figura 5.16: Simulación de comportamiento seguimiento de línea. Recorrido en línea recta.

En la figura 5.16 se comprueba que la señal de control *PWM* para ambos motores es de la misma magnitud, de igual forma el resultado del voltaje de los motores, manteniendo una misma velocidad en ambas ruedas logrando un desplazamiento en línea recta hacia adelante.

En la figura 5.17 se comprueba que la señal de control *PWM* permite que el motor derecho (M2) de la simulación permanezca con una mayor cantidad de tiempo activado que el motor izquierdo (M1), y como se explica anteriormente esto provoca el viraje hacia la izquierda. Por ende, el resultado del voltaje obtenido en los motores indica que el motor M2 girará a una mayor velocidad que el motor M1. En la figura 5.18 se ve el caso contrario, es decir, el viraje a la derecha. Pudiéndose comprobar lo mismo que para el giro a la izquierda.

Los resultados de esta simulación confirmaron el buen funcionamiento del sistema seguidor de línea, logrando ver el giro de motores según el caso lo requería; cuando el sensor infrarrojo izquierdo detecta que está sobre la línea negra entrega una señal en estado bajo haciendo que a través de la programación los motores ajusten sus velocidades de tal forma que el motor derecho gire más rápido que el izquierdo para lograr un leve viraje hacia la izquierda. De la misma forma se trabaja cuando se activa el sensor infrarrojo derecho. Pero, cuando se activa el sensor central, ambas ruedas giran a la misma velocidad consiguiendo avanzar en línea recta hasta que uno de los sensores laterales se active.

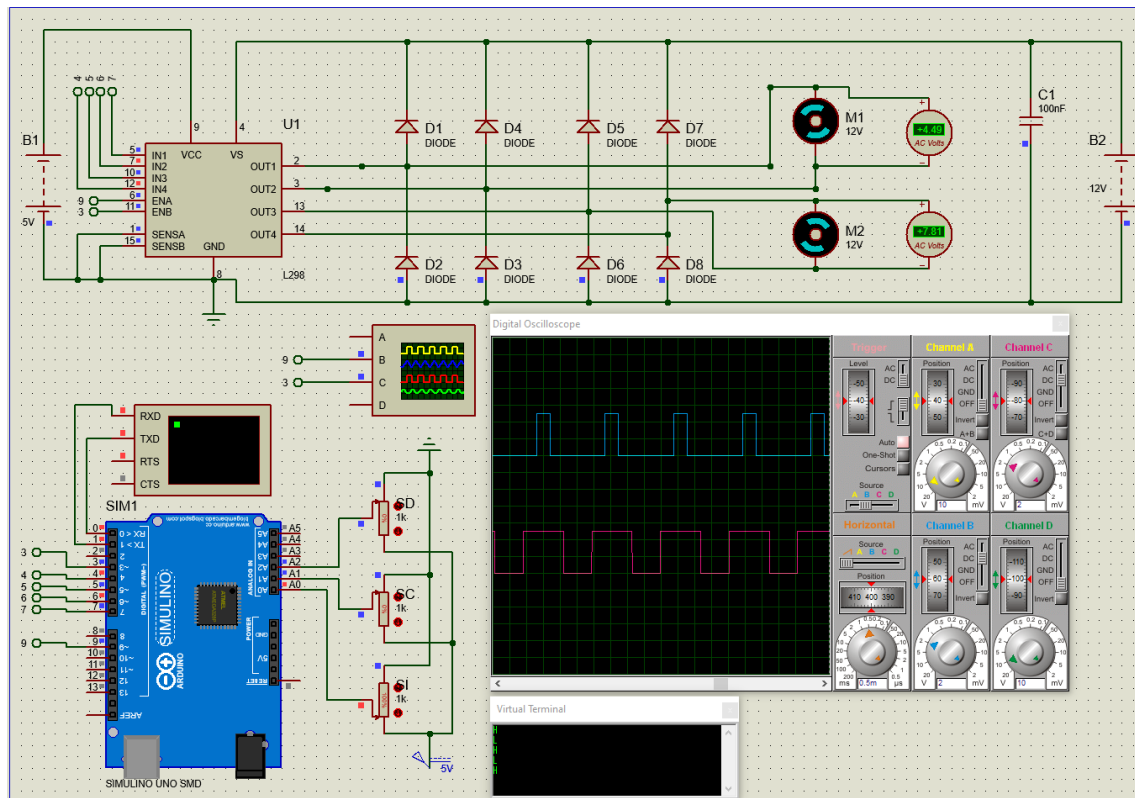


Figura 5.17: Simulación de comportamiento seguimiento de línea. Giro a la izquierda.

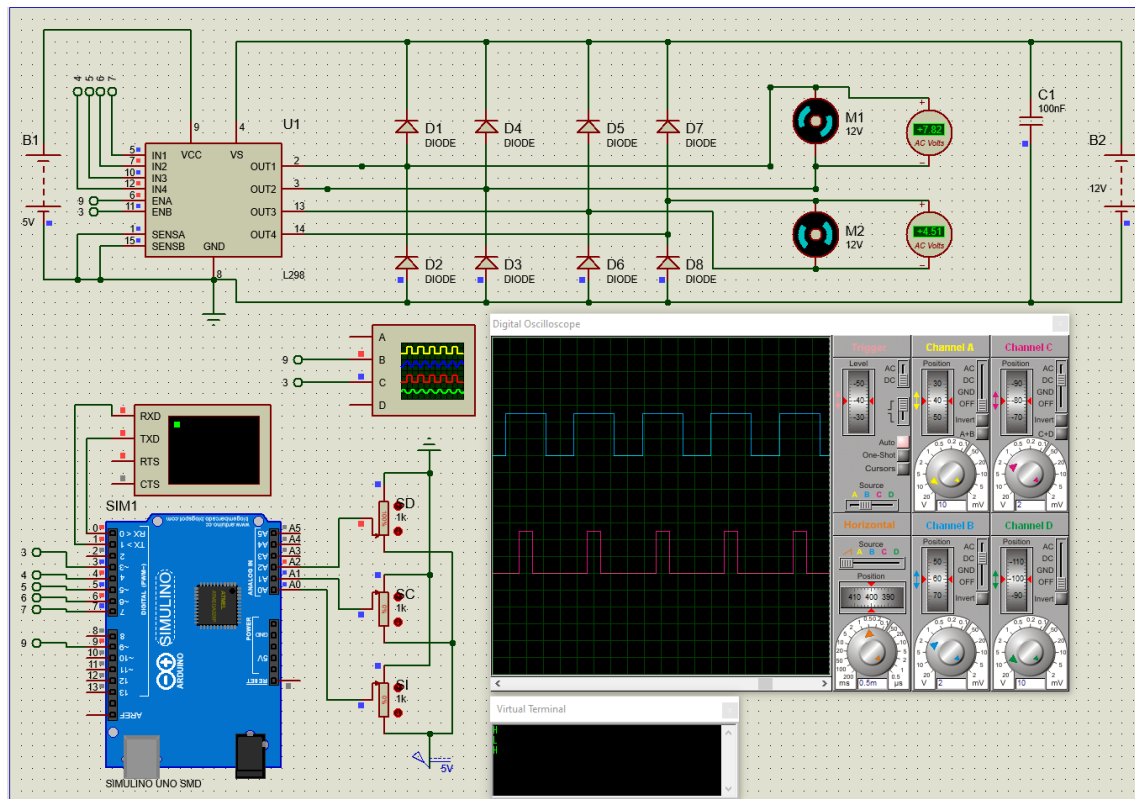


Figura 5.18: Simulación de comportamiento seguimiento de línea. Giro a la derecha.

## 6. Conclusiones

### 6.1. Sumario

- Se realizó una revisión bibliográfica sobre la visión estereoscópica y robótica móvil y sus aplicaciones en la investigación, industria y en el área de la agricultura que competía a este proyecto.
- Se revisaron alternativas de lenguajes de programación y métodos de recopilación de información tridimensional de imágenes 2D. Concluyendo que la mejor alternativa es el uso de *Python* en conjunto de la librería *OpenCV* por la cantidad de documentación disponible para su uso y guías de ejemplos de procesamiento digital de imágenes.
- Se considera el diseño de un sistema estereoscópico para toma de muestras y comprobación de viabilidad del proyecto.
- Se hizo la creación de un sistema de soporte para cámaras web consideradas para la configuración estereoscópica. Estas piezas fueron impresas en 3D dentro del laboratorio de robótica aérea.
- Se realizaron trabajos primero con imágenes sólo captadas por una cámara con el fin de extraer las características deseadas dentro de la imagen.
- Una vez tratada la imagen individual se logró obtener en unidad de píxel el diámetro de los objetivos estudiados.
- Posteriormente, se procedió con la aplicación de la visión estereoscópica con el fin de encontrar la distancia entre el sistema observador y el objetivo en cm. Esto fue realizado con algoritmos implementados rigiéndose por la técnica *SGBM* para el tratamiento de la disparidad en las imágenes.
- Una vez realizado el cálculo de distancia y de diámetro (este último en unidad de píxel) se crea una función que relaciona estos parámetros y se establece para cada caso el tamaño que representa cada píxel dentro de la imagen a la distancia que está la planta.
- Con las pruebas realizadas exitosas, se diseña un robot móvil con el objetivo de transportar el sistema estereoscópico. Se realizan primeros planos del robot y se ensamblan teniendo en consideración los requerimientos existentes para la creación de este vehículo.
- Se decide por diseñar la configuración tipo diferencial para la categoría de robots móviles terrestres, debido a la simplicidad en el control de movimiento y su maniobrabilidad.
- Con el diseño mecánico del robot creado, se diseña el apartado electrónico involucrado para ejecutar las acciones de desplazamiento en un espacio siguiendo una línea. Esto haciendo uso de comunicación serial para la interacción entre los microcontroladores y también entre el *arduino UNO* y la pantalla de notificación

de calibre. También, se realizó un estudio sobre consumos y alimentación de los circuitos.

- Se crearon los programas de control de movimiento del robot en conjunto de la función de clasificación de calibre anexada a los códigos de medición de calibre.
- Finalmente, se muestran los resultados experimentales obtenidos por la medición de calibre y la simulación del movimiento del robot.

## 6.2. Conclusiones

1. Es posible implementar un sistema de visión estereoscópica para emular un prototipo para la medición de calibre con el uso de cámaras de acceso comercial. Esto considerando condiciones de laboratorio con ambiente controlado.
2. Realizando depuración de los códigos y adaptándolos al entorno de desarrollo de la *Raspberry Pi 4*, se demuestra que es posible ejecutar los programas de medición estereoscópica en ordenadores de placa reducida.
3. La implementación de los algoritmos de medición de calibre en ordenadores de placa reducida reduce la capacidad de procesar imágenes de mayor calidad teniendo que escalar la resolución de las imágenes de  $640p \times 480p$  a  $544p \times 288p$ , esto debido a los inconvenientes presentados en los tiempos de ejecución y errores por falta de poder de cómputo.
4. El error asociado a la medición de diámetro es de  $\pm 500\mu m$  asociado a la detección de profundidad más la medición de diámetro en unidad de píxel.
5. Una limitación del sistema es la consideración de que si el robot se ubica muy alejado de los objetivos se corre el riesgo de perder consistencia en los resultados obtenidos por los algoritmos. Es por esto que se propuso el área de trabajo de  $25cm \times 45cm$  frente al objetivo a una distancia de  $30cm$  mínimo para asegurar una buena medición considerando una buena detección de bordes del objetivo en la etapa de procesamiento digital de imágenes.
6. El transporte y recorrido del robot queda delimitado por el seguimiento de una ruta preestablecida y demarcada en el terreno, en este caso una línea negra.
7. Si bien no es posible concluir que puede ser aplicado a un vivero, si es un buen comienzo para poder iniciar una línea investigativa en este aspecto.

### 6.3. Plan de cierre

Al finalizar este proyecto se espera que las herramientas utilizadas como las cámaras queden a disposición de la Escuela de Ingeniería Civil Mecatrónica, en donde se le podrá dar uso como escuela para nuevos proyectos y laboratorios.

El sistema de sujeción creado en material PLA de impresión 3D, de momento no se puede reciclar de forma fácil ni ecológica. Se dice que el PLA es biodegradable, pero no es tan cierto, ya que solo bajo ciertas condiciones se degrada en el tiempo. Lamentablemente, estas condiciones no se especifican en ningún lugar. Por lo que, al igual que con las cámaras web, el sistema de sujeción quedará a disposición de la Escuela.

### 6.4. Trabajos futuros

En primer lugar, se debería implementar el sistema robótico completo para realizar pruebas y depuraciones necesarias para un funcionamiento completo en condiciones de laboratorio e intentar implementar en ambientes no controlados en donde se puedan extrapolar resultados y verificar si es necesario seguir depurando aún más los códigos para obtener buenos resultados en ambos escenarios.

Las condiciones de laboratorio que se mencionan en las conclusiones es el punto que se debería atacar próximamente para asegurar el funcionamiento del sistema de medición en ambientes reales con variaciones en la luminosidad y terrenos en donde no se pueda implementar el uso de seguimiento de línea o ruta demarcada en el piso.

Por lo anterior, es posible considerar un tópico de investigación de este proyecto el uso de algoritmos de control de robots que transformen al vehículo en cuestión en un robot completamente autónomo considerando que se tiene que adaptar a los distintos escenarios manteniendo una buena distancia entre este y los objetivos de estudio.

Se sugiere en un futuro verificar la posibilidad de montar el sistema de visión estéreo para la navegación y medición de calibre, pudiendo realizar las dos tareas a la vez, considerando quizás una unidad de cómputo más potente.

## Referencias

- [1] P. Caro Molina, “MAGNITUD Y CARACTERÍSTICAS DE LA PARTICIPACIÓN LABORAL EN EL EMPLEO TEMPORAL AGRÍCOLA EN CHILE,” *Si Somos Americanos*, vol. 12, pp. 19 – 49, 12 2012.
- [2] J. chun Zhao, J. feng Zhang, Y. Feng, and J. xin Guo, “The study and application of the iot technology in agriculture,” in *2010 3rd International Conference on Computer Science and Information Technology*, vol. 2, pp. 462–465, 2010.
- [3] M. Jiménez, A. Ponce, M. Hurtado, and F. Navarro, “Terminología utilizada para definir el tipo de planta.” Accedido en 08-Septiembre-2019.
- [4] M. A. Mora Villate, J. D. Bernal Méndez, J. D. Bernal Méndez, J. E. Paneso Echeverry, and J. E. Paneso Echeverry, “Anatomía quirúrgica del ojo: Revisión anatómica del ojo humano y comparación con el ojo porcino,” *Morfología*, vol. 8, pp. 21 – 44, sep. 2016.
- [5] R. Muñoz Manso, “Sistema de visión artificial para la detección y lectura de matrículas,” Master’s thesis, UNIVERSIDAD DE VALLADOLID, 2014.
- [6] “Facultad de ingeniería - universidad nacional de san juan - vision estereoscópica.” Accedido en 29-Abril-2019. Apuntes de clases.
- [7] F. Zhao and Z. Jiang, “A new algorithm for three-dimensional construction based on the robot binocular stereo vision system,” in *2012 4th International Conference on Intelligent Human-Machine Systems and Cybernetics*, vol. 2, pp. 302–305, 2012.
- [8] F. Lecumberry, “Cálculo de disparidad en imágenes estéreo, una comparación,” in *III Workshop de Computación Gráfica, Imágenes y Visualización*, 2005.
- [9] H. Hirschmuller, “Stereo processing by semiglobal matching and mutual information,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 328–341, 2008.
- [10] A. Ollero, *Robótica manipuladores y robots móviles*. marcombo boixareu editores, 2001.
- [11] S. Goldberg, M. Maimone, and L. Matthies, “Stereo vision and rover navigation software for planetary exploration,” in *Proceedings, IEEE Aerospace Conference*, vol. 5, pp. 5–5, 2002.
- [12] D. Carabias, R. García, and J. Rodríguez, “sistema de visión estereoscópica para navegación autónoma de vehículos no tripulados,” Master’s thesis, Universidad Complutense de Madrid Facultad de Informática, 2014.
- [13] K. Huh, J. Park, J. Hwang, and D. Hong, “A stereo vision-based obstacle detection system in vehicles,” *Optics and Lasers in Engineering*, vol. 46, no. 2, pp. 168–178, 2008.



- [14] S. Jin, J. Cho, X. D. Pham, K. M. Lee, S.-K. Park, M. Kim, and J. W. Jeon, "Fpga design and implementation of a real-time stereo vision system," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 1, pp. 15–26, 2010.
- [15] H. Schreier, D. Garcia, and M. Sutton, "Advances in light microscope stereo vision," *Experimental Mechanics*, vol. 44, no. 1, pp. 278–288, 2004.
- [16] G. Viera-Maza, "Procesamiento de imágenes usando opencv aplicado en raspberry pi para la clasificación del cacao," Master's thesis, Universidad de Piura, 2017.
- [17] L. Pencue-Fierro and J. León Téllez, "Detección y clasificación de defectos en frutas mediante el procesamiento digital de imágenes," vol. 35, 01 2003.
- [18] OpenCV, "Opencv tutorial c++." Accedido en 25-Abril-2019 a url<https://www.opencv-srf.com/p/introduction.html>.
- [19] OpenCV, "Installation in windows." accedido en 10-Mayo-2019.
- [20] Numerentur.org, "Núcleo (kernel) de las svm." accedido en 31-October-2020.
- [21] J. Valverde-Rebaza, "Detección de bordes mediante el algoritmo de canny," 10 2007.
- [22] A. de Ingeniería de Sistemas y Automática, "Segmentación transformada de hough." Curso 2005/2006.
- [23] I. ESTEVEZ, "Estimación de un mapa disperso de profundidad usando visión estéreo," Master's thesis, PONTIFICIA UNIVERSIDAD JAVERIANA, 2010.
- [24] U. Mujica, "Reconstrucción densa de modelos tridimensionales utilizando visión artificial," Master's thesis, Universidad del País Vasco, 2010.
- [25] M. Kokshoorn and R. Green, "Low cost real-time stereo vision system," Master's thesis, University of Canterbury.
- [26] M. Montalvo, "Técnicas de visión estereoscópica para determinar la estructura tridimensional de la escena," Master's thesis, Universidad Complutense de Madrid, 2010.
- [27] F. Lanzaro and M. Fuentes, "Calibración y posicionamiento 3d." accedido en 01- Noviembre-2020.
- [28] C. Ricolfe and A. Sánchez, "Procedimiento completo para el calibrado de cámaras utilizando una plantilla plana," *Revista Iberoamericana de Automática e Informática Industrial*, vol. 5, pp. 93–101, Enero 2008.
- [29] Z. Zhang, "A flexible new technique for camera calibration," tech. rep., Microsoft Corporation, 1998.
- [30] OpenCV, "Camera calibration and 3d reconstruction." accedido en 03- Noviembre-2020.

- [31] MathWorks, “Stereo disparity using semi-global block matching.” accedido en 03-  
Noviembre-2020.
- [32] OpenCV, “Transformaciones morfológicas.” accedido en 03-  
Noviembre-2020.
- [33] Autodesk, “Materiales pmma.” accedido en 22-  
Mayo-2021.
- [34] Y. Ishida, H. Asama, S. Tomita, K. Ozaki, A. Matsumoto, and I. Endo, “Functional  
complement by cooperation of multiple autonomous robots,” in *Proceedings of the  
1994 IEEE International Conference on Robotics and Automation*, pp. 2476–2481  
vol.3, 1994.
- [35] M. Serna, A. Sánchez, M. Estrada, M. Bernabé, and E. Pérez, “Navigation of  
mobile robots using extended bugs algorithms,” in *Research in Computing Science*,  
vol. 148, pp. 159–171 vol.3, 2019.
- [36] J. Arango, O. Arcila, and J. Bañón, “Sobre los algoritmos de planificación del  
movimiento de robots,” *Ingeniería y Competitividad*, vol. 2, 10 2011.
- [37] S. O. Robots, “Rmf calculator.” accedido en 05-  
Octubre-2020.
- [38] Vishay Semiconductor GmbH, *Reflective Optical Sensor with Transistor Output*,  
83760 ed., 07 2000.
- [39] X. Yang, H. Jiang, and Z. Deng, “Design of a battery management system based on  
matrix switching network,” in *2015 IEEE International Conference on Information  
and Automation*, pp. 138–141, 2015.
- [40] LUMEX, *LCM-S01602DTR/M*, 0 ed., 11 2002.
- [41] RS, “Display monocromo lcd lumex, alfanumérico, reflectante, 2 filas x 16 carac-  
teres, paralelo.” accedido en 21-  
Marzo-2021.
- [42] Texas Instruments, *REMOTE 8-BIT I/O EXPANDER FOR I2C BUS*, 655303 ed.,  
07 2001.
- [43] C. electrónica, “Módulo serial i2c para display lcd 16 x 2 pcf8574a.” accedido en  
21-  
Marzo-2021.
- [44] J. Guerra, “Comunicación i2c con arduino lo mejor de dos mundos.” accedido en  
22-  
Marzo-2021.

## 7. Anexos

### A. Especificaciones de ordenador en primera iteración de creación de algoritmos

Tabla 7.1: Especificaciones técnicas computador de pruebas.

| Componente               | Modelo                                                                           |
|--------------------------|----------------------------------------------------------------------------------|
| <i>Procesador</i>        | Ryzen 5 1600, 3.2 GHz, 6 núcleos, 12 hilos                                       |
| <i>Tarjeta de video</i>  | GTX 1060 6 GB                                                                    |
| <i>RAM</i>               | 16 GB 2400 MHz                                                                   |
| <i>Almacenamiento</i>    | - Samsung Evo 970 NVMe M.2 250 GB<br>- WD green SSD 120 GB<br>- Toshiba HDD 1 TB |
| <i>Sistema operativo</i> | Windows 10                                                                       |

## B. Especificaciones técnicas de la Raspberry Pi 4.

- \* Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- \* 4GB LPDDR4-3200 SDRAM
- \* 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- \* Extended 40-pin GPIO header
- \* Full-size HDMI
- \* **2 puertos USB 3.0; 2 puertos USB 2.0**
- \* 2 puertos micro HDMI
- \* 1 puerto Micro SD para la carga del sistema operativo y almacén de información
- \* Alimentación DC 5V/2.5A
- \* 5V DC vía USB-C

## C. Planos de robot

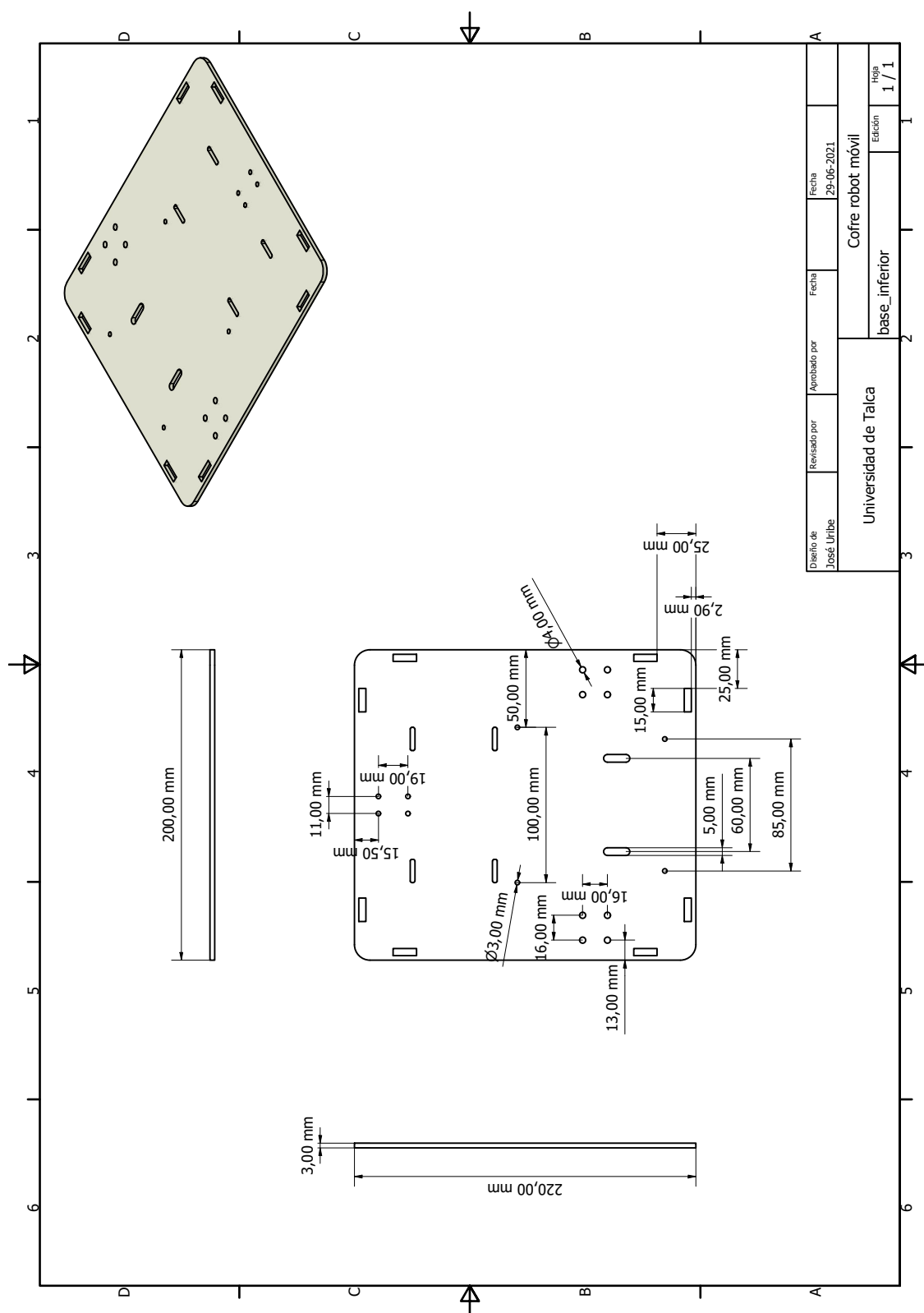


Figura 7.1: Placa inferior robot móvil.

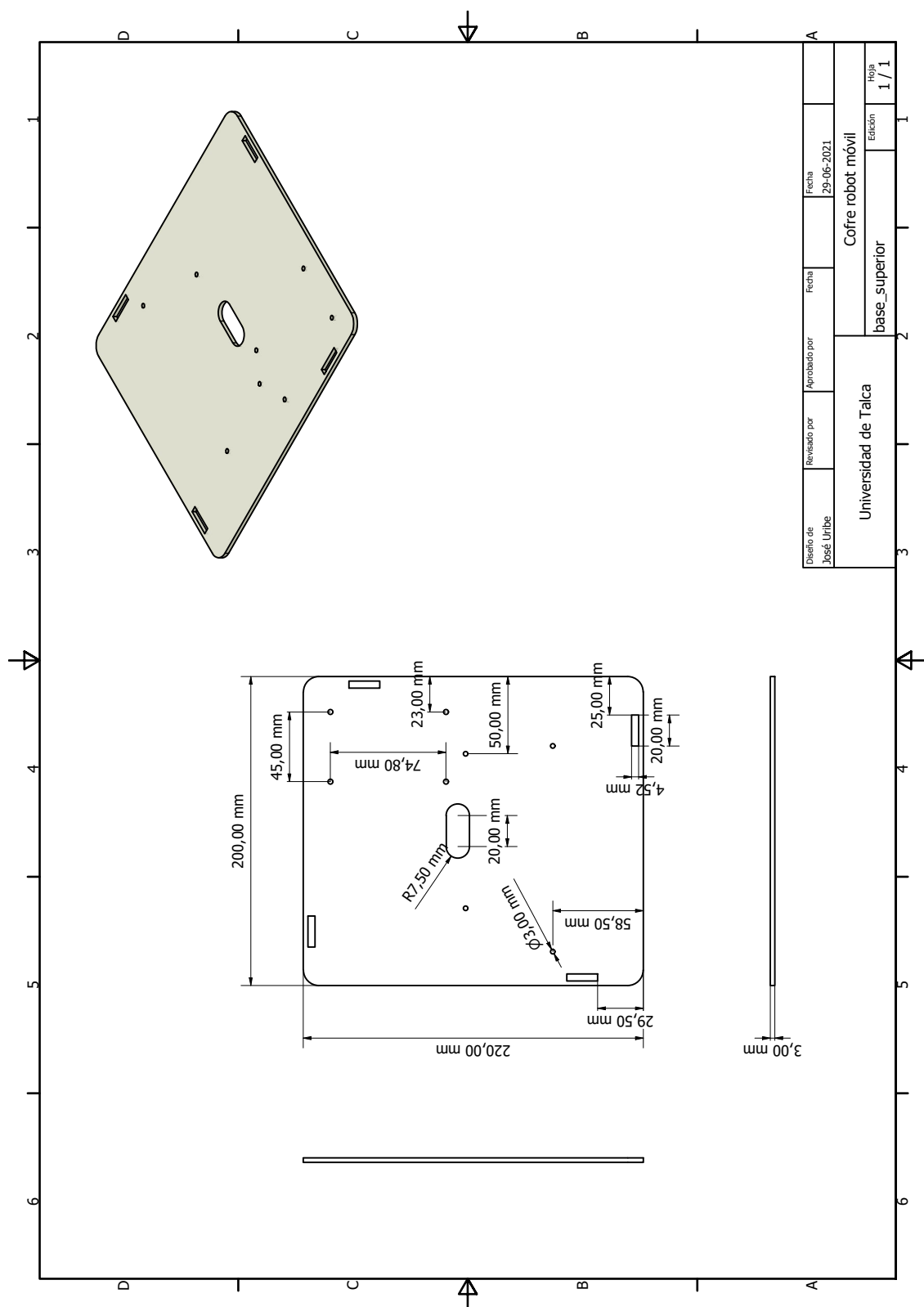


Figura 7.2: Placa superior robot móvil.

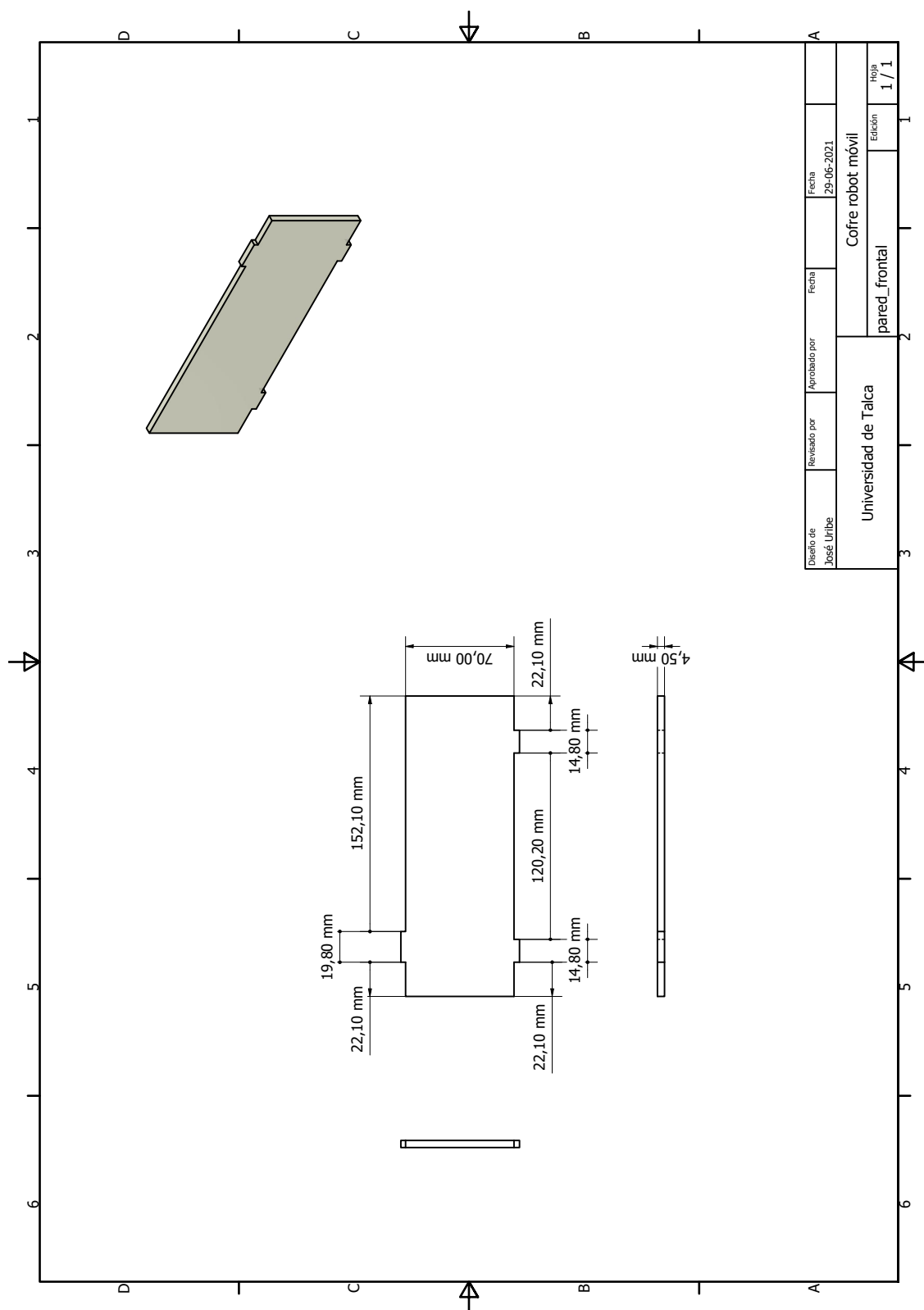


Figura 7.3: Placa frontal robot móvil.

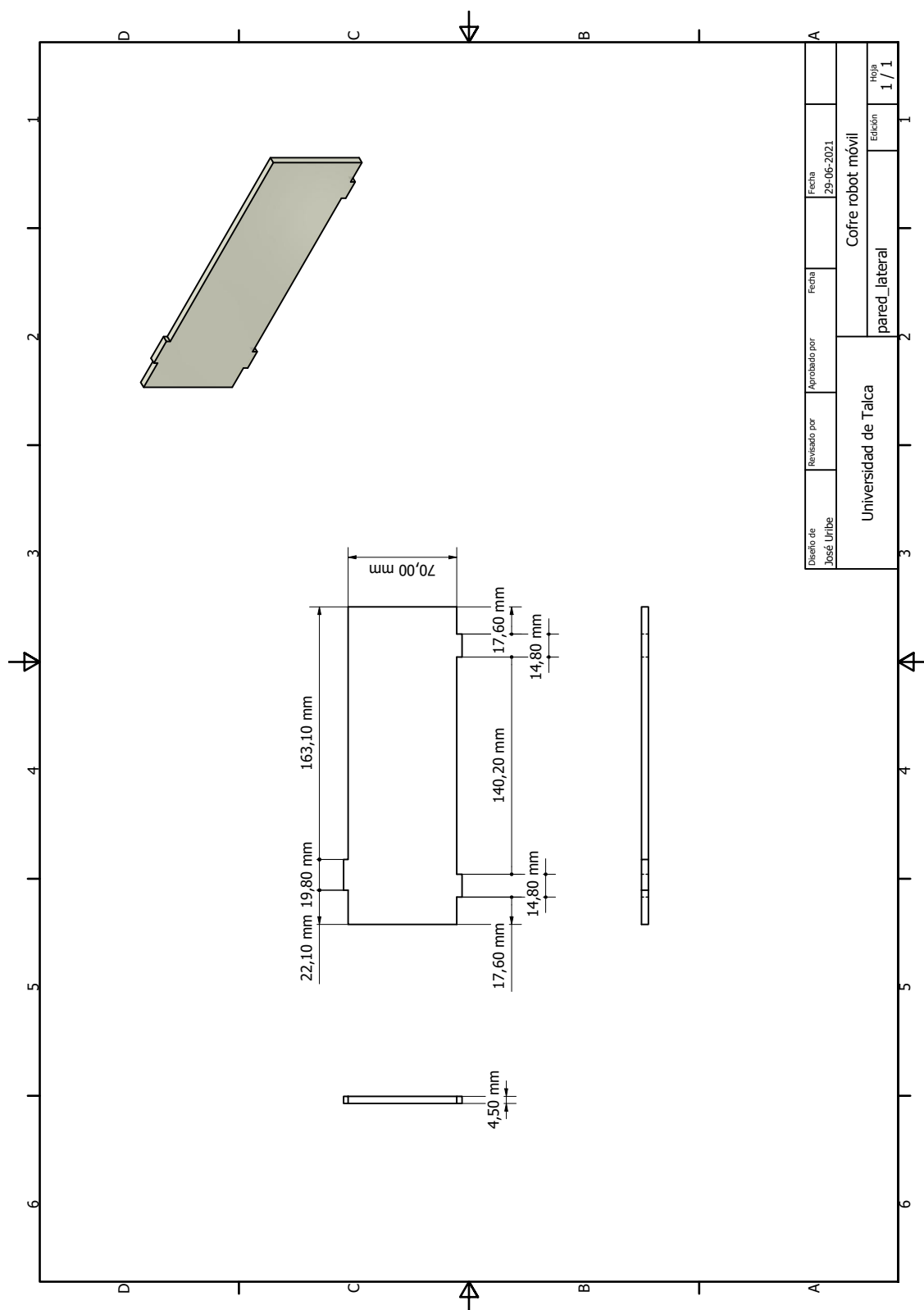


Figura 7.4: Placa lateral robot móvil.



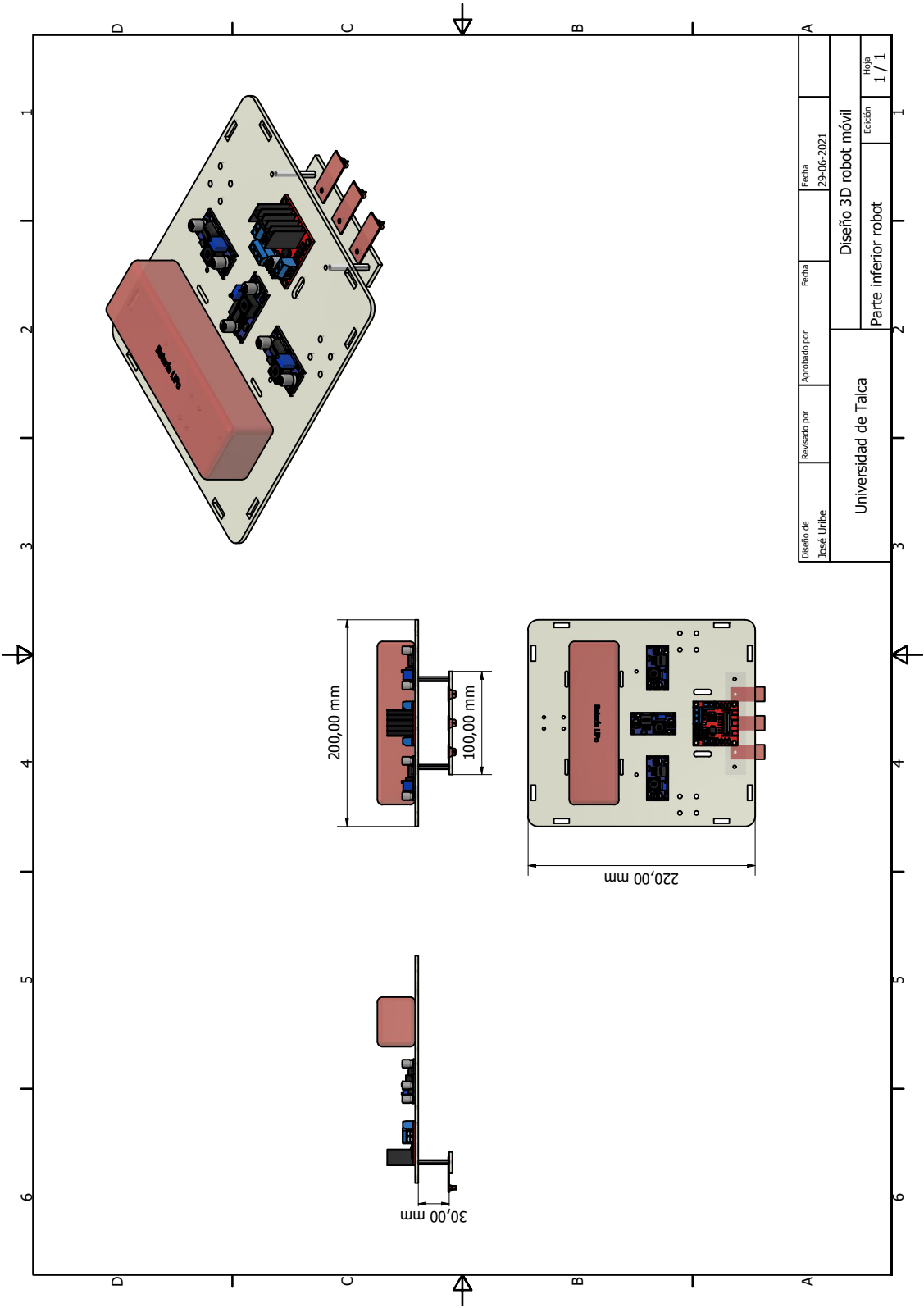


Figura 7.5: Distribución de componentes en la placa inferior.

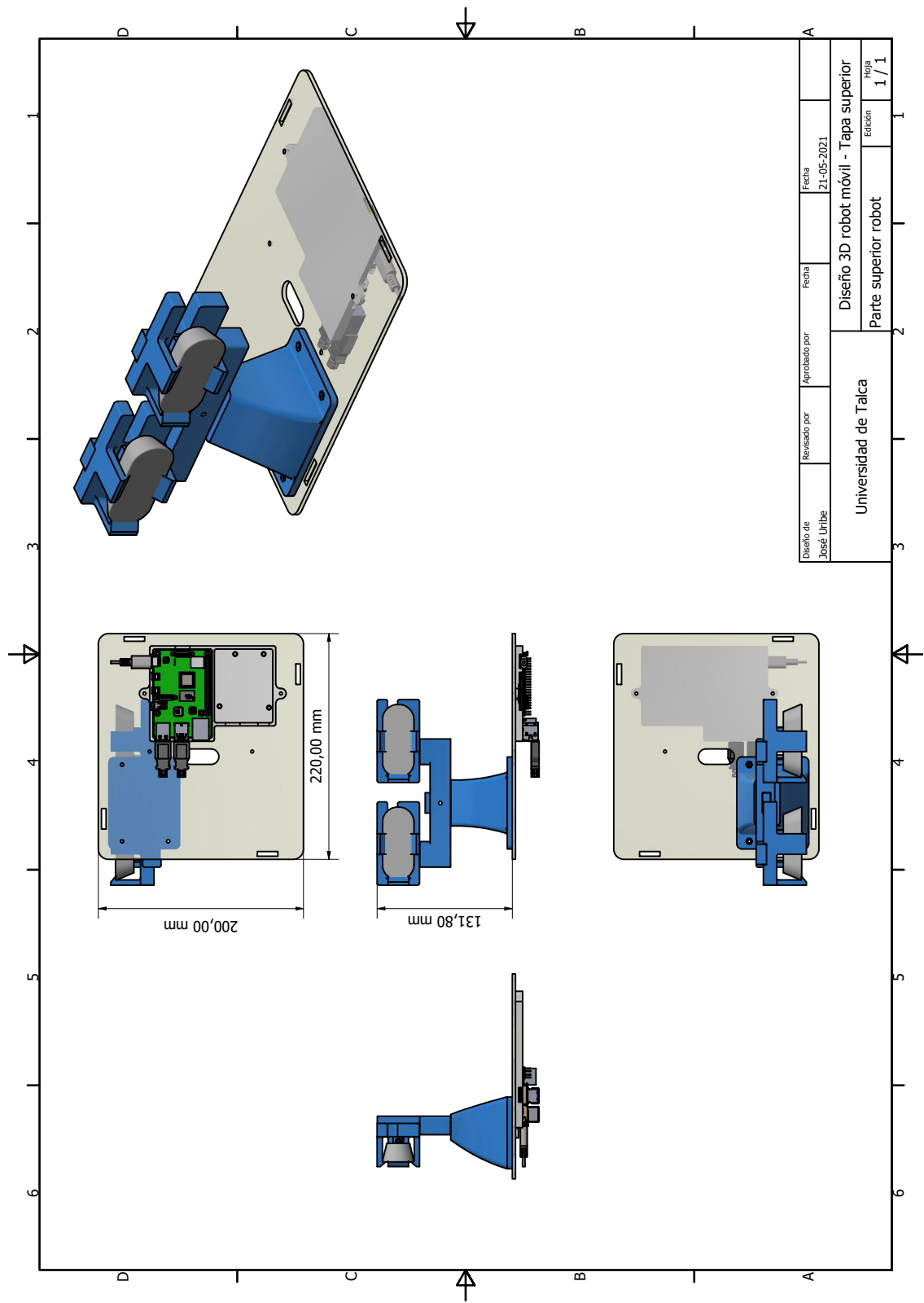


Figura 7.6: Distribución de componentes en la placa superior.

## D. Planos de sistema de sujeción

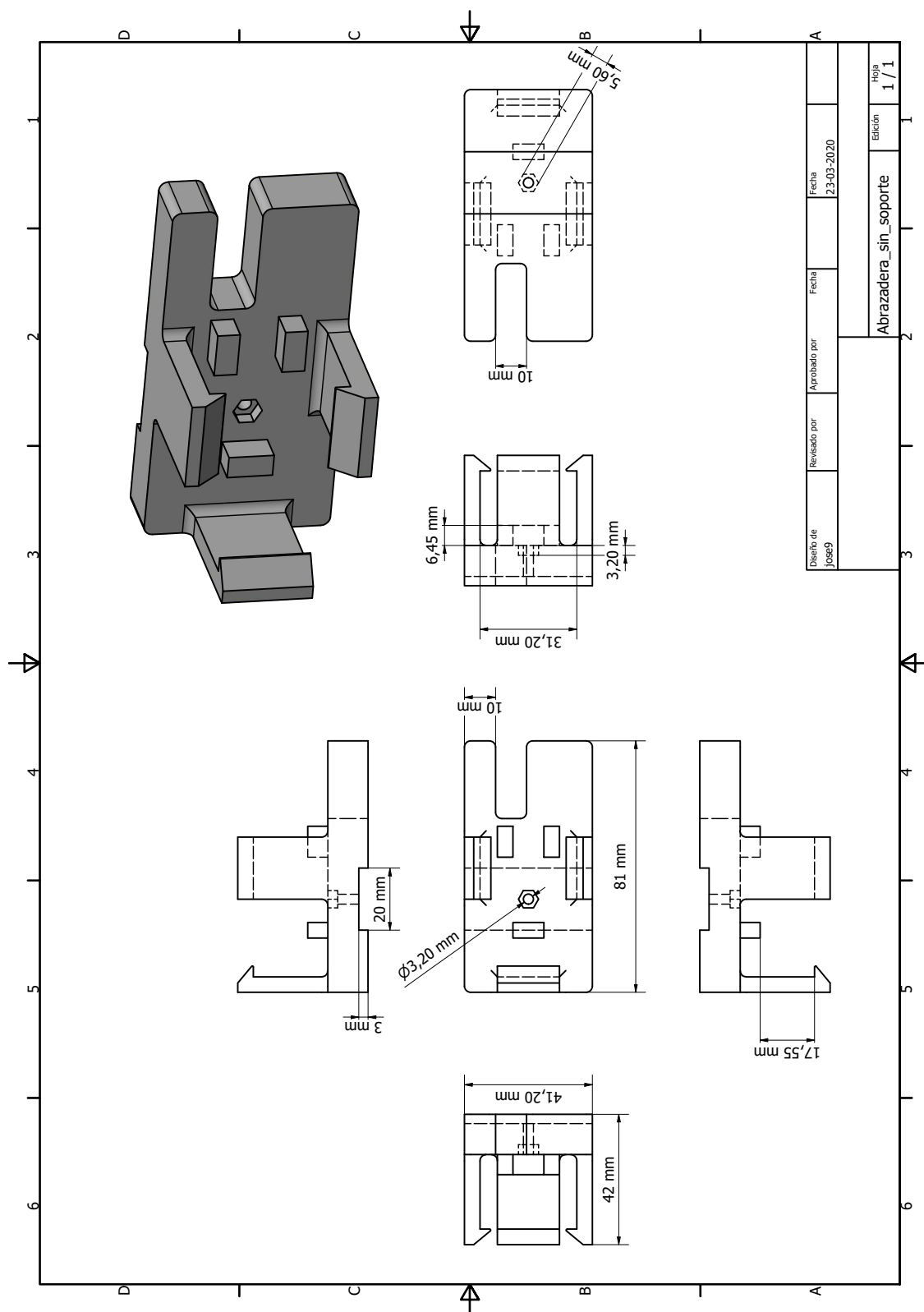


Figura 7.7: Pieza de agarre para cámara web.

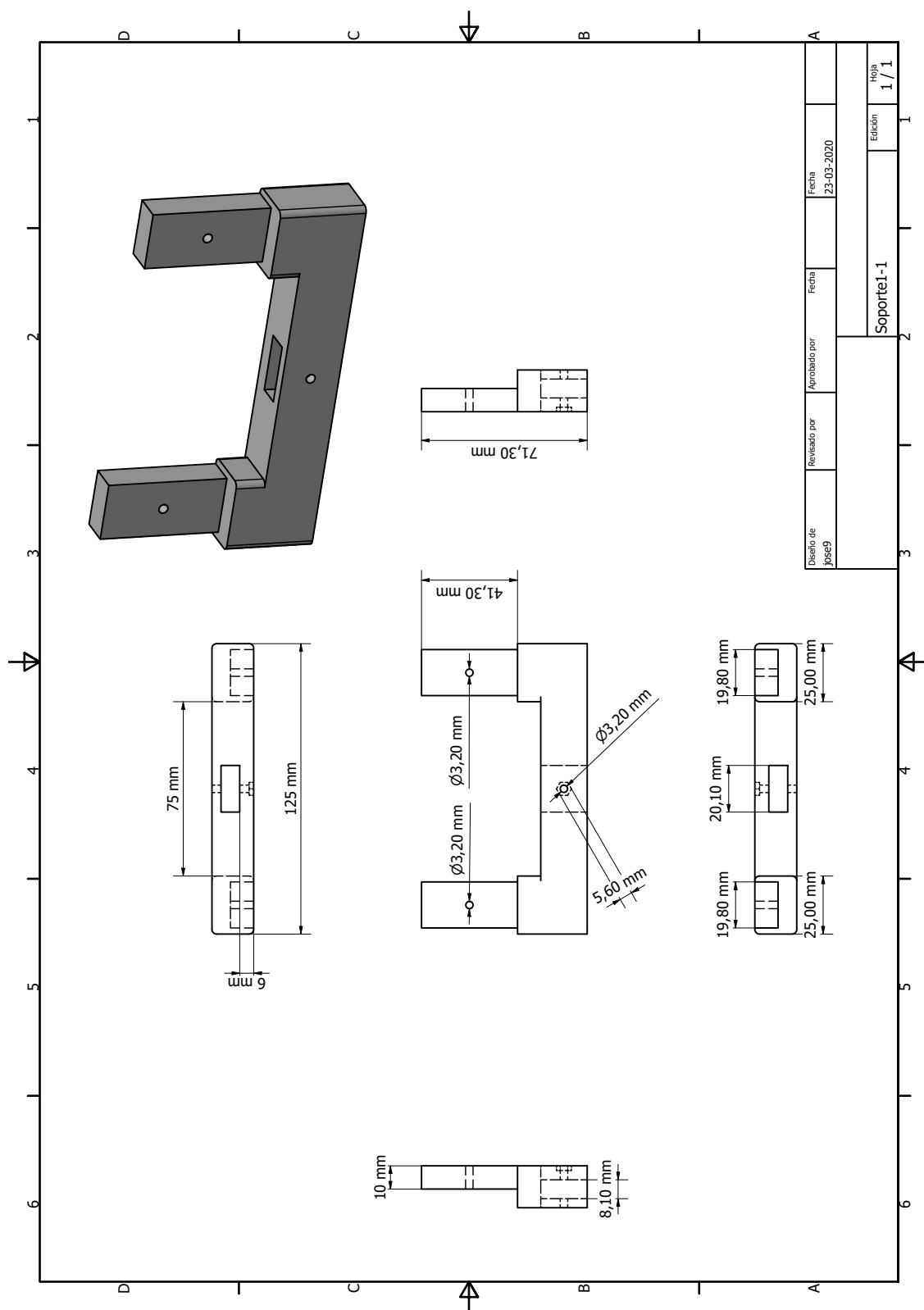


Figura 7.8: Pieza de anclaje de dos cámaras.

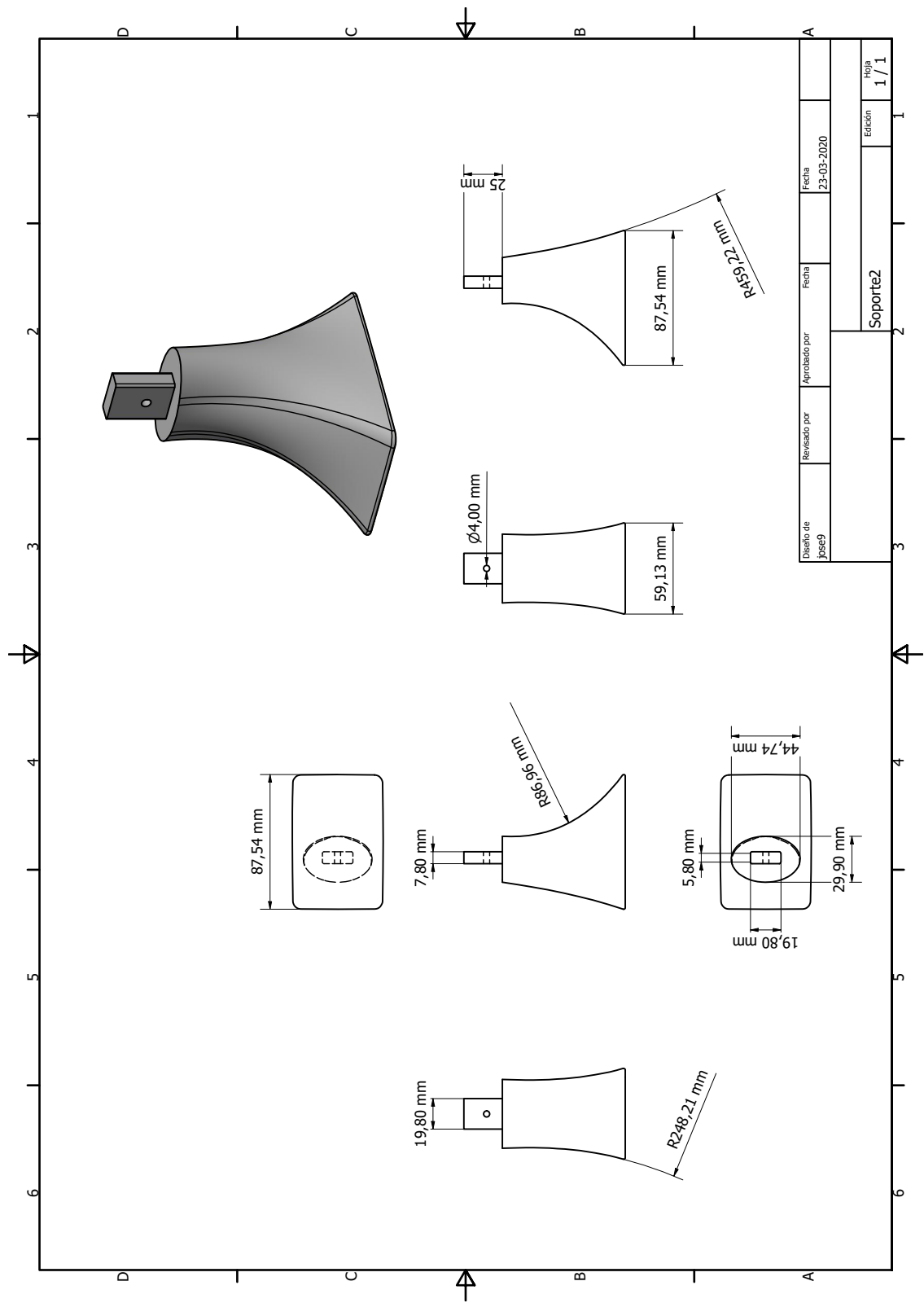


Figura 7.9: Pieza base, para la estabilización y postura en superficies.

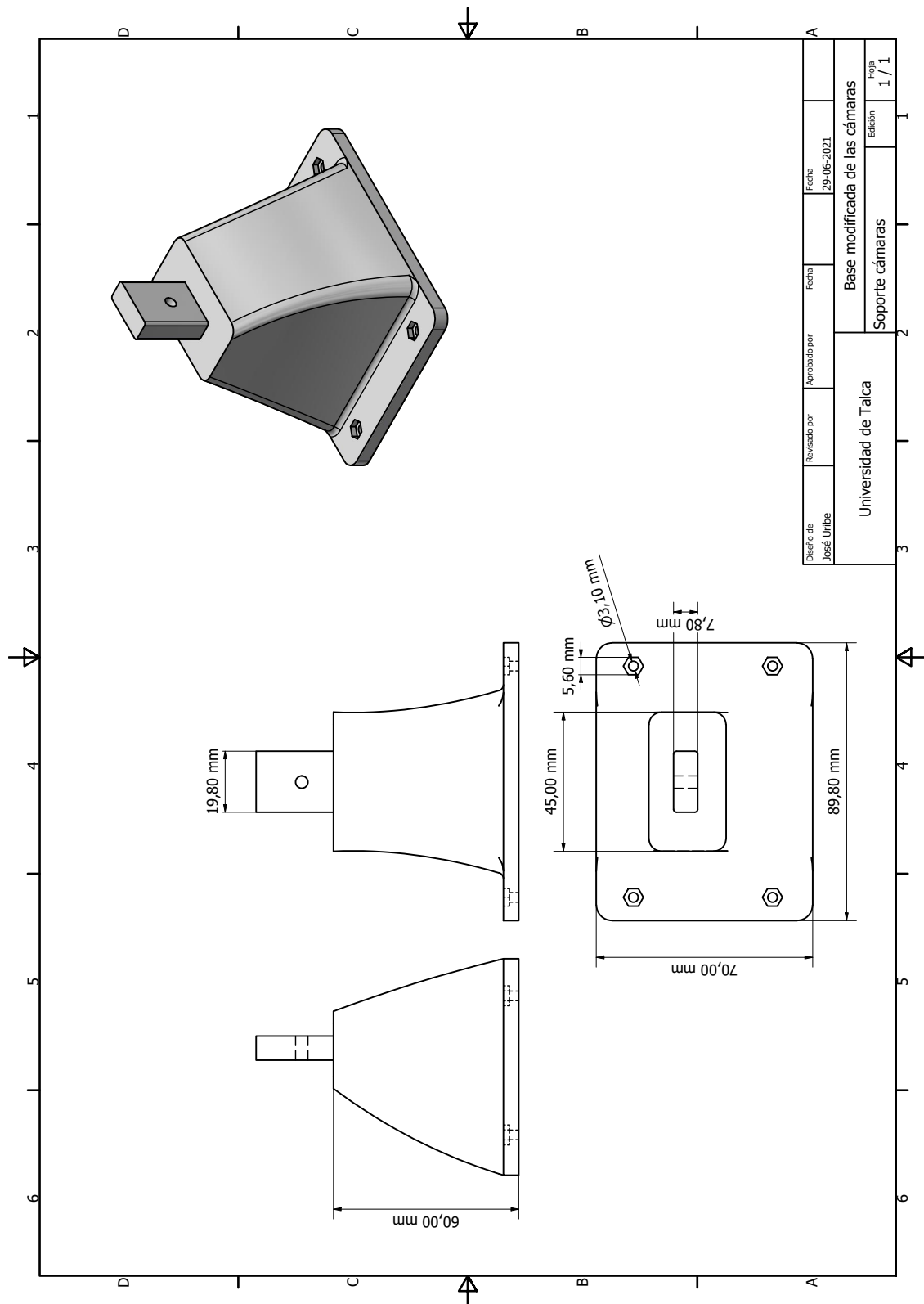


Figura 7.10: Pieza base modificada para montaje en robot móvil.

## E. Código depurado para uso en *Raspberry Pi 4*

### E.a. Código principal

```

1
2 import cv2
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from deteccion import detect
6 from distancia import distancia_func
7 from calibre import calculo_calibre
8 from calibracion import calibracion
9
10 detencion=0
11
12 Left_stereo_map, Right_stereo_map, stereo, stereoR, wls_filter=
    calibracion()
13
14 while (True):
15     centro, detencion, img_izq, img_der, centro_img=detect()
16     cv2.imshow('objetivo',centro_img)
17     print('El centro es: ', centro)
18
19     # cv2.imshow('camaraI',img_izq)
20     #
21     # cv2.imshow('camaraD',img_der)
22
23     if detencion==0:
24         distancia, detencion=distancia_func(centro, img_izq, img_der,
25         Left_stereo_map, Right_stereo_map, stereo, stereoR, wls_filter)
26         print('La distancia es: ', distancia)
27
28     if detencion==0:
29         diametro=calculo_calibre(distancia, img_izq)
30         #print('El diametro es: ', diametro)
31
32     print("Instruccion de proseguir al robot")
33
34     if cv2.waitKey(1) & 0xFF == ord(' ') or detencion==1:
35         break

```

Código 7.1: Código principal de ejecución.

## E.b. Calibración de cámaras

```

1  import numpy as np
2  import cv2
3  from openpyxl import Workbook
4  from sklearn.preprocessing import normalize
5  import matplotlib.pyplot as plt
6
7
8
9  def calibracion():
10
11     kernel= np.ones((3,3),np.uint8)
12
13
14     COUNTERDIST=30
15
16     criteria =(cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30,
17 0.001)
18     criteria_stereo= (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER,
19 30, 0.001)
20
21     objp = np.zeros((8*6,3), np.float32)
22     objp[:, :2] = np.mgrid[0:8,0:6].T.reshape(-1,2)
23
24     objpoints= []
25     imgpointsR= []
26     imgpointsL= []
27
28     print('Comenzando calibracion de las camaras')
29
30     for i in range(0,30):
31         t= str(i)
32         ChessImaR= cv2.imread('Calibracion/chessboard-R'+t+'.png',0)
33         ChessImaL= cv2.imread('Calibracion/chessboard-L'+t+'.png',0)
34         retR, cornersR = cv2.findChessboardCorners(ChessImaR,(8,6),None)
35         retL, cornersL = cv2.findChessboardCorners(ChessImaL,(8,6),None)
36         if (True == retR) & (True == retL):
37             objpoints.append(objp)
38             cv2.cornerSubPix(ChessImaR,cornersR,(11,11),(-1,-1),criteria)
39             cv2.cornerSubPix(ChessImaL,cornersL,(11,11),(-1,-1),criteria)
40             imgpointsR.append(cornersR)
41             imgpointsL.append(cornersL)
42
43     retR, mtxR, distR, rvecsR, tvecsR = cv2.calibrateCamera(objpoints,
44 imgpointsR, ChessImaR.shape[::-1],None,None)
45     hR,wR= ChessImaR.shape[:2]
46     OmtxR, roiR= cv2.getOptimalNewCameraMatrix(mtxR,distR,(wR,hR),1,(wR,
47 hR))
48
49     retL, mtxL, distL, rvecsL, tvecsL = cv2.calibrateCamera(objpoints,
50 imgpointsL, ChessImaL.shape[::-1],None,None)
51     hL,wL= ChessImaL.shape[:2]
52     OmtxL, roiL= cv2.getOptimalNewCameraMatrix(mtxL,distL,(wL,hL),1,(wL,
53 hL))

```



```

49     print('Cameras listas para usar')
50
51     flags = 0
52     flags |= cv2.CALIB_FIX_INTRINSIC
53
54     retS, MLS, dLS, MRS, dRS, R, T, E, F= cv2.stereoCalibrate(objpoints,
imgpointsL, imgpointsR, mtxL, distL, mtxR, distR, ChessImaR.shape[::-1],
criteria_stereo, flags)
55
56     rectify_scale= 0
57     RL, RR, PL, PR, Q, roiL, roiR= cv2.stereoRectify(MLS, dLS, MRS, dRS,
ChessImaR.shape[::-1], R, T, rectify_scale, (0,0))
58     Left_Stereo_Map= cv2.initUndistortRectifyMap(MLS, dLS, RL, PL,
ChessImaR.shape[::-1], cv2.CV_16SC2)
59     Right_Stereo_Map= cv2.initUndistortRectifyMap(MRS, dRS, RR, PR,
ChessImaR.shape[::-1], cv2.CV_16SC2)
60
61     window_size = 3
62     min_disp = 2
63     num_disp = 130-min_disp
64     stereo = cv2.StereoSGBM_create(minDisparity = min_disp,
65         numDisparities = num_disp,
66         blockSize = window_size,
67         uniquenessRatio = 10,
68         speckleWindowSize = 100,
69         speckleRange = 32,
70         disp12MaxDiff = 5,
71         P1 = 8*3*window_size**2,
72         P2 = 32*3*window_size**2)
73
74     stereoR=cv2.ximgproc.createRightMatcher(stereo)
75
76     lmbda = 80000
77     sigma = 1.8
78     visual_multiplier = 1.0
79
80     wls_filter = cv2.ximgproc.createDisparityWLSFilter(matcher_left=
stereo)
81     wls_filter.setLambda(lmbda)
82     wls_filter.setSigmaColor(sigma)
83
84     return Left_Stereo_Map, Right_Stereo_Map, stereo, stereoR, wls_filter

```

Código 7.2: Calibración de cámaras.

## E.c. Detección de objetivo

```

1  import cv2
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5
6  def detect():
7      detencion=0
8      camR= cv2.VideoCapture(0)
9      camR.set(cv2.CAP_PROP_FRAME_WIDTH, 544)
10     camR.set(cv2.CAP_PROP_FRAME_HEIGHT, 288)
11
12     camL= cv2.VideoCapture(2)
13     camL.set(cv2.CAP_PROP_FRAME_WIDTH, 544)
14     camL.set(cv2.CAP_PROP_FRAME_HEIGHT, 288)
15
16
17     kernel = np.ones((5,5),np.uint8)
18     limites=[240, 435, 120, 240] #360p 480x360 (funciona)
19
20     aux=0
21     cuenta=0
22     cuenta_medicion=0
23     encuentra=0
24
25     while (aux==0):
26         retL, frameL = camL.read()
27         retR, frameR = camR.read()
28         copia=np.copy(frameL)
29         rangomax = np.array([10,10,10])
30         rangomin = np.array([0,0,0])
31         mascara = cv2.inRange(copia, rangomin, rangomax)
32         opening = cv2.morphologyEx(mascara, cv2.MORPH_OPEN, kernel)
33         x,y,w,h = cv2.boundingRect(opening)
34         cv2.rectangle(copia,(x,y),(x+w,y+h),(0,255,0),4)
35         cv2.circle(copia,(x+int(w/2),y+int(h/2)),6,(0,0,100),-1)
36         centro=[]
37         centro=[x+int(w/2),y+int(h/2)]
38
39         if centro[0] > limites[0] and centro[0] < limites[1] and centro
40         [1] > limites[2] and centro[1] < limites[3] and encuentra==0:
41             cuenta=cuenta+1
42             if cuenta==25:
43                 print("STOP")
44                 encuentra=1
45                 cuenta_medicion=0
46             elif centro[0] > limites[0] and centro[0] < limites[1] and centro
47             [1] > limites[2] and centro[1] < limites[3] and encuentra==1:
48                 cuenta_medicion=cuenta_medicion+1
49                 if cuenta_medicion==50:
50                     img_izq=frameL
51                     img_der=frameR
52                     aux=1
53             else:
54                 cuenta=0

```

```

53         print('no detecta')
54
55     #cv2.imshow('Izq', frameL)
56
57     if cv2.waitKey(1) & 0xFF == ord(' '):
58         detencion=1
59         break
60
61     camL.release()
62     camR.release()
63     cv2.destroyAllWindows()
64     #plt.imshow(frame)
65     #plt.show()
66     return centro, detencion, img_izq, img_der, copia

```

Código 7.3: Calibración de cámaras.

## E.d. Cálculo de distancia

```

1
2 import numpy as np
3 import cv2
4 from openpyxl import Workbook # Used for writing data into an Excel file
5 from sklearn.preprocessing import normalize
6 import matplotlib.pyplot as plt
7 detencion=0
8
9 def coords(centro , disp):
10     average=0
11     for u in range (-1,2):
12         for v in range (-1,2):
13             average += disp[centro[1]+u,centro[0]+v]
14     average=average/9
15     Distance=33.766*average**(4)-117.2*average**(3)+165.7*average**(2)
16     -137.87*average+85.475
17     Distance_1= np.around(Distance , decimals=1)
18     print( 'Distancia: ' + str(Distance_1)+' cm')
19     return Distance
20
21
22 def distancia_func(punto , img_L , img_R , Left_Stereo_Map , Right_Stereo_Map
23 , stereo , stereoR , wls_filter):
24     detencion=0
25
26     kernel= np.ones((3,3),np.uint8)
27
28     aux_distancia=0
29     detencion=0
30
31     frameR=img_R
32     frameL=img_L
33
34     Left_nice= cv2.remap(frameL,Left_Stereo_Map[0],Left_Stereo_Map[1],
35     cv2.INTER_LANCZOS4, cv2.BORDER_CONSTANT, 0) # Rectify the image using
36     the kalibration parameters founds during the initialisation
37     Right_nice= cv2.remap(frameR,Right_Stereo_Map[0],Right_Stereo_Map[1],
38     cv2.INTER_LANCZOS4, cv2.BORDER_CONSTANT, 0)
39
40     grayR= cv2.cvtColor(Right_nice ,cv2.COLOR_BGR2GRAY)
41     grayL= cv2.cvtColor(Left_nice ,cv2.COLOR_BGR2GRAY)
42
43     disp= stereo.compute(grayL ,grayR)
44     dispL= disp
45     dispR= stereoR.compute(grayR ,grayL)
46     dispL= np.int16(dispL)
47     dispR= np.int16(dispR)
48
49     filteredImg= wls_filter.filter(dispL ,grayL ,None ,dispR)
50     filteredImg = cv2.normalize(src=filteredImg , dst=filteredImg , beta=0,
51     alpha=255, norm_type=cv2.NORM_MINMAX)
52     filteredImg = np.uint8(filteredImg)

```

```

49 min_disp = 2
50 num_disp = 130-min_disp
51 disp= ((disp.astype(np.float32)/ 16)-min_disp)/num_disp
52
53 closing= cv2.morphologyEx(disp ,cv2.MORPH_CLOSE, kernel)
54
55 dispC= (closing-closing.min())*255
56 dispC= dispC.astype(np.uint8)
57 disp_Color= cv2.applyColorMap(dispC ,cv2.COLORMAP_OCEAN)
58 filt_Color= cv2.applyColorMap(filteredImg ,cv2.COLORMAP_OCEAN)
59
60
61 aux_distancia=coords(punto, disp)
62
63 return aux_distancia ,detencion

```

Código 7.4: Calibración de cámaras.

## E.e. Cálculo de diámetro

```

1
2 import cv2
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 def canny(imagen):
7     gris=cv2.cvtColor(imagen, cv2.COLOR_RGB2GRAY)
8     blur=cv2.GaussianBlur(gris, (5,5), 0)
9     canny=cv2.Canny(blur, 50, 150)
10    return canny
11
12 def mostrar_lineas(imagen, lineas):
13     linea_imagen=np.zeros_like(imagen)
14     if lineas is not None:
15         for linea in lineas:
16             x1, y1, x2, y2=linea.reshape(4)
17             cv2.line(linea_imagen, (x1, y1), (x2,y2), (255, 255, 255), 1)
18     return linea_imagen
19
20 def desviacion(columna, prom):
21     suma_des=0
22     for i in range(len(columna)):
23         suma_des=suma_des+abs(columna[i]-prom)**2
24
25     if len(columna)==0:
26         desv=0
27     else:
28         desv=suma_des/len(columna)
29     desviacion_final=desv**0.5
30     return desviacion_final
31
32 def filtro(columna, columna1, fila, fila1, prom, desviacion):
33     col11=[]
34     fila11=[]
35     col22=[]
36     fila22=[]
37     for i in range(len(columna)):
38         if desviacion==0:
39             aux=0
40         else:
41             aux=(columna[i]-prom)/desviacion
42         if aux<1.65 and aux>-1.65:
43             col11.append(columna[i])
44             fila11.append(fila[i])
45             col22.append(columna1[i])
46             fila22.append(fila1[i])
47     return col11, fila11, col22, fila22
48
49 def calculo_calibre(medicion_zdist, img_izq):
50
51     copia=np.copy(img_izq)
52
53     canny_imagen=canny(copia)
54

```

```

55 | lineas=cv2.HoughLinesP(canny_imagen, 1, np.pi/180, 115, np.array([]),
56 | minLineLength=100, maxLineGap=10)
57 | lineas_imagen=mostrar_lineas(copia, lineas)
58 |
59 | copia_linea=np.copy(lineas_imagen)
60 |
61 | copia_BGR=cv2.cvtColor(copia, cv2.COLOR_RGB2BGR)
62 |
63 | superposicion=cv2.addWeighted(copia_BGR, 0.8, lineas_imagen, 1, 1)
64 |
65 | cant_por_fila=0
66 | dist=[]
67 | linea=[]
68 | linea1=[]
69 | k=0
70 | l=0
71 | col_der=[]
72 | col_izq=[]
73 |
74 | alto=np.size(copia_linea, 0)
75 | ancho=np.size(copia_linea, 1)
76 |
77 | gris_linea=cv2.cvtColor(copia_linea, cv2.COLOR_RGB2GRAY)
78 |
79 | for i in range(alto):
80 |     for j in range(ancho):
81 |         if gris_linea[i][j] == 255:
82 |             cant_por_fila=cant_por_fila+1
83 |             if cant_por_fila == 1:
84 |                 col_izq.insert(l, j)
85 |                 linea1.insert(l, i)
86 |                 l=l+1
87 |             elif cant_por_fila==2:
88 |                 linea.insert(k, i)
89 |                 col_der.insert(k, j)
90 |                 k=k+1
91 |             cant_por_fila=0
92 |
93 | linea1_coincidencia=[]
94 | col1_coincidencia=[]
95 | distancia=[]
96 | m=0
97 |
98 | col1=0
99 | col2=0
100 | sumader_prom=0
101 | sumaizq_prom=0
102 | columna1=[]
103 | columna2=[]
104 | fila1=[]
105 | fila2=[]
106 |
107 | for i in range(len(linea)):
108 |     for j in range(len(linea1)):
109 |         if linea[i] == linea1[j]:
110 |             linea1_coincidencia.insert(m, linea1[j])

```

```

110         col1_coincidencia.insert(m, col_izq[j])
111         m=m+1
112     valor=col_der[i]-col1_coincidencia[i]
113     if valor!=1:
114         columna2.append(col_der[i])
115         columna1.append(col1_coincidencia[i])
116         fila1.append(linea1_coincidencia[i])
117         fila2.append(linea[i])
118         sumader_prom=col_der[i]+sumader_prom
119         sumaizq_prom=col1_coincidencia[i]+sumaizq_prom
120
121     if len(columna1)==0:
122         prom_izq=0
123     else:
124         prom_izq=sumaizq_prom/(len(columna1))
125
126     if len(columna2)==0:
127         prom_der=0
128     else:
129         prom_der=sumader_prom/(len(columna2))
130     distancia_pixell=prom_der-prom_izq
131
132
133     desviacion_izq=desviacion(columna1, prom_izq)
134
135     desviacion_der=desviacion(columna2, prom_der)
136
137     fil_fila11=[]
138     fil_fila21=[]
139     fil_col11=[]
140     fil_col21=[]
141
142     fil_col11, fil_fila11, fil_col21, fil_fila21=filtro(columna1,
143     columna2, fila1, fila2, prom_izq, desviacion_izq)
144
145     fil_col21, fil_fila21, fil_col11, fil_fila11=filtro(columna2,
146     columna1, fila2, fila1, prom_der, desviacion_der)
147
148     suma_izq=0
149     suma_der=0
150     for i in range(len(fil_col11)):
151         suma_izq=suma_izq+fil_col11[i]
152         suma_der=suma_der+fil_col21[i]
153     if len(fil_col11)==0:
154         prom_izq=0
155     else:
156         prom_izq=suma_izq/len(fil_col11)
157     if len(fil_col21)==0:
158         prom_der=0
159     else:
160         prom_der=suma_der/len(fil_col21)
161     distancia_pixel=prom_der-prom_izq
162
163     print("———Distancia promedio (en pixeles) entre pixeles———")
164     print(distancia_pixel)

```



```

164     if distancia_pixel==0:
165         aux_pixel=0
166     else:
167         aux_pixel=16/distancia_pixel
168
169     print("———Tamaño del pixel———")
170     print(aux_pixel)
171
172     tamaño_pixel=0.0177*medicion_zdist+0.0134
173     diametro=tamaño_pixel*distancia_pixel
174     print("El diametro del tallo es: \n", diametro)
175
176     cv2.destroyAllWindows
177     plt.imshow(superposicion)
178     plt.show()
179     return diametro

```

Código 7.5: Calibración de cámaras.

### E.f. Programa Arduino UNO seguidor de línea

```

1
2 #define led 3
3 #define sensorI A0
4 #define sensorC A1
5 #define sensorD A2
6 #define ENB 3
7 #define IN4 4
8 #define IN3 5
9 #define IN2 6
10 #define IN1 7
11 #define ENA 9
12
13 //const int umbral=50;
14 const int umbral=125;
15 const int vel_min=50; //velocidad motor
16 const int vel_max=150; //velocidad motor
17 const int vel_media=100; //velocidad motor
18
19 int si;
20 int sc;
21 int sd;
22 int out;
23
24 void setup() {
25     Serial.begin(9600);
26
27     pinMode(led,OUTPUT);
28     pinMode(ENA,OUTPUT);
29     pinMode(ENB,OUTPUT);
30     pinMode(IN1,OUTPUT);
31     pinMode(IN2,OUTPUT);
32     pinMode(IN3,OUTPUT);
33     pinMode(IN4,OUTPUT);
34

```

```

35  detencion();
36  digitalWrite(IN1, LOW);
37  digitalWrite(IN2, HIGH);
38  digitalWrite(IN3, LOW);
39  digitalWrite(IN4, HIGH);
40  }
41
42  void loop() {
43
44      if (Serial.available()) { //Si está disponible
45          char c = Serial.read(); //Guardamos la lectura en una variable char
46          if (c == 'H') { //Si es una 'H', comenzamos a seguir la linea
47              si=lectura_a(sensorI);
48              sc=lectura_a(sensorC);
49              sd=lectura_a(sensorD);
50
51              seguidor(si,sc,sd);
52          } else if (c == 'L') { //Si es una 'L', dejamos de seguir la linea
53              detencion();
54          }
55      }
56  }
57
58  int lectura_a(int pin){
59      out=analogRead(pin);
60      out=out/4;
61      return out;
62  }
63
64  void seguidor(int si,int sc,int sd){
65      if(si>umbral && sc<umbral && sd<umbral){
66          girarIzq();
67      } else if(si<umbral && sc>umbral && sd<umbral){
68          recto();
69      } else if(si<umbral && sc<umbral && sd>umbral){
70          girarDer();
71      } else if(si<umbral && sc<umbral && sd<umbral){
72          detencion();
73      } else if(si>umbral && sc>umbral && sd>umbral){
74          detencion();
75      }
76  }
77
78  void girarIzq(){
79      analogWrite(ENA,vel_min);
80      analogWrite(ENB,vel_max);
81  }
82
83  void girarDer(){
84      analogWrite(ENB,vel_min);
85      analogWrite(ENA,vel_max);
86  }
87
88  void recto(){
89      analogWrite(ENB,vel_media);
90      analogWrite(ENA,vel_media);

```

```
91     }  
92  
93     void detencion() {  
94         digitalWrite(ENA, 0);  
95         digitalWrite(ENB, 0);  
96     }
```

Código 7.6: Seguidor de línea.

