

**UNIVERSIDAD
NACIONAL DE CÓRDOBA**



**FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y
NATURALES**

INGENIERIA EN COMPUTACIÓN

Proyecto Integrador

“Telemetría mediante el reconocimiento del plano.”

Autor: Vidable, Nicolás Javier

Email: njvidable@gmail.com Celular: 0351156756818

Matrícula: 34456612

Directores:

Ingeniero Eschoyez, Maximiliano A.

Geólogo Cebollada y Verdaguer, Marcelo

Año: 2016

Índice

Índice	3
Índice de figuras	7
Índice de tablas.....	9
Resumen	11
Capítulo 1: Introducción.....	13
1.1. Introducción.....	13
1.2. Motivación.....	13
1.3. Objetivos	13
1.4. Requerimientos.....	13
1.5. Lineamientos del prototipo.....	13
1.6. Análisis de riesgos.....	14
Capítulo 2: Marco teórico.....	15
2.1. Introducción.....	15
2.2. Telemetría.....	15
2.3. Telémetro láser	15
2.4. Telemetría mediante visión artificial	15
2.5. Algunos conceptos sobre imágenes digitales	15
2.5.1. Imágenes digitales	15
2.5.2. Resolución	15
2.5.3. Profundidad de bits.....	16
2.5.4. Escala de grises.....	16
2.5.5. RGB.....	17
2.5.6. HSV	17
2.6. Visión artificial	18
2.7. OpenCV	18
2.8. OpenCV-Python	18
2.9. Distorsión en las imágenes, calibración de cámaras	19
Capítulo 3: Desarrollo.....	21
3.1. Introducción.....	21
3.2. Iteración I: Construcción del prototipo y análisis del fenómeno	21
3.2.1 Prototipo N°1.....	21
3.2.2. Principio de funcionamiento	23
3.2.2.1. ¿Por qué son necesarios tres punteros láser?	24
3.2.2.2. Deformación del triángulo.....	24
3.2.3. Análisis de la deformación del triángulo	26
3.3. Iteración II: Reconocimiento de los puntos láser.....	27
3.3.1. Reconocimiento mediante el modelo de color HSV	27
3.3.1.1. Pruebas	27
3.3.1.2. Análisis sobre el reconocimiento de los puntos láser utilizando el espacio de color HSV	32
3.3.2. Reconocimiento mediante Thresholding	33
3.3.2.1. Thresholding de umbral fijo	33
Binario.....	33
Binario invertido.....	34
Truncado.....	34
A cero	35
A cero invertido	35
3.3.2.2. Thresholding adaptativo	36
Thresholding adaptativo, Mean	36
Thresholding adaptativo, Gaussian.....	36
3.3.2.3. Análisis del reconocimiento de los puntos láser mediante Thresholding	37
3.4. Iteración III: Suavizado de imágenes	38
3.4.1. Tipos de filtros.....	39
3.4.1.1. Averaging (promedio)	39
3.4.1.2. Gaussian filtering (gaussiano)	39
3.4.1.3. Median filtering (mediana)	39
3.4.1.4. Bilateral filtering (bilateral).....	40
3.4.2. Precauciones y análisis	40

3.5. Iteración IV: Determinar la posición de los puntos láser en la imagen	41
3.5.1. Método I: “Parasyte”	41
3.5.2. Método II: Hough Circle Transform.....	42
3.5.3. Método III: Mediante la detección de contornos	44
3.5.3.1. Contornos en OpenCV.....	44
Encontrar los contornos para determinar la posición de los puntos láser	45
3.6. Iteración V: Encontrar las figuras que forman el triángulo	48
3.6.1. Imagen como plano en 2D.....	48
3.6.2. Método “Trinity Hunter”.....	49
3.6.2.1. Diagrama de actividad del algoritmo.....	51
3.6.2.2. Condiciones definitivas del algoritmo	51
3.6.3. Método “Trinity Hunter Rev”.....	52
3.7. Iteración VI: Análisis del Triángulo	53
3.7.1. Cálculo de la inclinación mediante la ecuación del plano	53
3.8. Iteración VII: Determinar las dimensiones reales de objetos desde una imagen	57
3.8.1. Mediciones para planos sin inclinación	58
3.8.1.1. Mediciones de largo y ancho (rectilíneas)	58
3.8.1.2. Mediciones de área y perímetro.....	58
3.8.2. Transformación de perspectiva.....	60
3.9. Función para el cálculo de distancia.....	61
3.9.1. Función de correspondencia	61
3.9.2. Nueva función de correspondencia.....	63
3.9.3. 3Mpx	64
3.10. Iteración VIII: Pruebas de sensibilidad.....	66
3.10.1. Prototipo N°2.....	66
3.10.2. Pruebas	66
3.10.2.1. Prueba N°1	66
3.10.2.2. Prueba N°2	67
3.10.3. Análisis de las pruebas de sensibilidad.....	68
3.11. Iteración IX: Software del dispositivo	68
3.11.1. Pantalla inicial RAFI (Range Finder Telémetro)	69
3.11.1.1. Selección de una carpeta (“Browse”)	70
3.11.1.2. Ajustes de parámetros, sensibilidad.....	72
3.11.1.3. Ejecución y presentación de resultados	72
3.11.2. PET (Perspective Transform Transformación de perspectiva)	73
3.11.3. OM (Object Measurement Medición de objetos).....	76
3.11.3.1. Líneas	77
3.11.3.2. Contornos	78
Capítulo 4: Diagrama completo del proceso	81
4.1. Introducción.....	81
4.2. Pipeline de la calibración de la cámara.....	81
4.3. Pipeline asociado a RAFI(Range Finder)	82
4.3.1. Rectificado.....	82
4.3.2. Umbralización	82
4.3.3. Filtrado de figuras.....	83
4.3.4. Detección del triángulo.....	83
4.3.5. Refinamiento de la posición de las figuras	83
4.3.6. Análisis del triángulo.....	84
4.3.7. Salida/Output	84
4.4. Pipeline asociado a PET (Perspective Transform)	84
4.4.1. Confección de la “Cruz”	85
4.4.2. Detección de los puntos amarillos	85
4.4.3. Salida/Output (PET)	85
4.5. Pipeline asociado a OM (Object Measurement)	85
4.6. Pipeline integral.....	86
Capítulo 5: Conclusiones y trabajos futuros	87
5.1. Introducción.....	87
5.2. En cuanto a los objetivos	87
5.3. En cuanto a los requerimientos	88
5.4. En cuanto a las limitaciones	89

5.5. En general.....	89
5.6. Trabajos futuros	92
Anexo A	93
Estructura de directorios.....	93
Referencias	95

Índice de figuras

Figura 1.1. Disposición de los punteros láser	14
Figura 2.1. Matriz mxn de una imagen digital.....	15
Figura 2.2. Diferentes resoluciones para una misma imagen	16
Figura 2.3. Valor asociado a cada píxel en Gray scale	17
Figura 2.4. Espacio de color RGB diagrama (izq). Ejemplo de porcentajes de cada canal por pixel (der)	17
Figura 2.5. Modelo de color HSV	18
Figura 2.6. Deformación debido a la lente de cámaras fisheye	19
Figura 2.7. Imágenes capturadas con fisheye rectificadas	20
Figura 3.1. Puntero láser.....	21
Figura 3.2. Vista lateral del soporte para los punteros láser (izq). Vista frontal del soporte para los punteros láser (der).....	22
Figura 3.3. Cámara digital utilizada	22
Figura 3.4. Plaqueta reguladora de tensión.....	23
Figura 3.5. Proyección de los puntos láser	23
Figura 3.6. Proyecciones de los puntos láser superpuestas.....	23
Figura 3.7. Análisis de proyecciones.....	24
Figura 3.8. Modelo con solo un puntero láser	24
Figura 3.9. Inclinación vertical en el plano de proyección	25
Figura 3.10. Deformación del triángulo debido a la inclinación vertical	25
Figura 3.11. Deformación del triángulo debido a la inclinación vertical (análisis).....	25
Figura 3.12. Deformación del triángulo debido a la inclinación vertical (análisis) 2	25
Figura 3.13. Inclinación vertical en el plano de proyección	26
Figura 3.14. Deformación del triángulo debido a la inclinación vertical (análisis)	26
Figura 3.15. Deformación del triángulo debido a la inclinación vertical (análisis) 2	26
Figura 3.16. Proyección de puntero láser de color verde en BGR.....	27
Figura 3.17. Proyección de puntero láser de color verde en HSV	28
Figura 3.18. Máscara resultante para el filtrado del color verde en la imagen	28
Figura 3.19. Resultado de aplicar la máscara a la figura 2.1	29
Figura 3.20. Prueba n°2: Proyección de puntero láser rojo de 5mW (RGB).....	29
Figura 3.21. Proyección de puntero láser rojo 5mW (HSV)	30
Figura 3.22. Mascaras para filtrado del color rojo. Con rango de matiz de 0 a 10 (izq). Rango de 170 a 179 (der).....	30
Figura 3.23. Suma de máscaras para el filtrado del color rojo	30
Figura 3.24. Resultado al aplicar máscara de filtrado.....	30
Figura 3.25. Imagen original de la prueba n° 3 (izq). Y su correspondencia en HSV (der)	31
Figura 3.26. Mascaras para filtrado del color rojo. Con rango de matiz de 0 a 10 (izq) y de 170 a 179 (der).....	31
Figura 3.27. Máscara resultante (izq). Resultado de aplicarla sobre la imagen original (der).....	31
Figura 3.28. Prueba n°4: imagen original (izq). Resultado al aplicar máscara de filtrado (der).....	31
Figura 3.29. Prueba n°5: imagen original (izq), conversión a HSV(centro) y resultado al filtrar el rojo (der).....	32
Figura 3.30. Thresholding: imagen en RGB	33
Figura 3.31. Thresholding: imagen en escala de grises	33
Figura 3.32. Thresholding: Thresholding binario.....	34
Figura 3.33. Thresholding: Thresholding binario invertido.....	34
Figura 3.34. Thresholding: Thresholding binario truncado	35
Figura 3.35. Thresholding: Thresholding a cero (to zero)	35
Figura 3.36. Thresholding: Thresholding a cero invertido	36
Figura 3.37. Thresholding adaptativo: Mean + binario	36
Figura 3.38. Thresholding adaptativo: Mean + binario invertido	36
Figura 3.39. Thresholding adaptativo: Gaussian + binario.....	37
Figura 3.40. Thresholding adaptativo: Gaussian + binario invertido	37
Figura 3.41. Proyección del puntero láser	38
Figura 3.42. Proyección del puntero luego de aplicar thresholding.....	38
Figura 3.43. Proyección del puntero thresholding y luego averaging	40
Figura 3.44. Proyección del puntero con averaging y luego thresholding.....	40
Figura 3.45. Mecánica del método Parasyte de cuatro direcciones	41

Figura 3.46. Mecánica del método Parasyte de cuatro direcciones para figuras irregulares	42
Figura 3.47. Mecánica del método Parasyte de ocho direcciones para figuras irregulares.....	42
Figura 3.48. Logo de OpenCV para testing de Hough Circle Transform.....	43
Figura 3.49. Logo de OpenCV para testing de Hough Circle Transform (resultado).....	44
Figura 3.50. Figura para testing de la función findContours de OpenCV	45
Figura 3.51. Repercusiones sobre la imagen de entrada para la función findContours	46
Figura 3.52. Contorno detectado junto con el círculo envolvente	46
Figura 3.53. Parasyte para ajustar el centro de una figura	47
Figura 3.54. Parasyte para ajustar el centro de una figura n°2	47
Figura 3.55. Figura con orificio.....	48
Figura 3.56. Figura con orificio, contornos detectados	48
Figura 3.57. Imagen como plano xy	49
Figura 3.58. Proyección del triángulo en cualquier zona de la imagen	49
Figura 3.59. Proyecciones superpuestas con el centro de la imagen como centro del triángulo	50
Figura 3.60. Interpretaciones del triángulo según los puntos encontrados. Aplicando las nuevas condiciones para el punto superior (izq). Aplicando las nuevas condiciones para el punto superior y el inferior izquierdo (der)	50
Figura 3.61. Diagrama de actividad del algoritmo Trinity Hunter	51
Figura 3.62. Triángulo equilátero circunscrito y ángulos interiores	52
Figura 3.63. Medida de arista a partir del radio de un triángulo.....	53
Figura 3.64. Triángulos equiláteros para cada uno de los radios de un triángulo escaleno	54
Figura 3.65. Vectores definidos por tres puntos en el mismo plano.....	54
Figura 3.66. Vector normal al plano	55
Figura 3.67. Deformación de dimensiones de acuerdo a la perspectiva	58
Figura 3.68. Deformación de dimensiones de acuerdo a la perspectiva n°2	58
Figura 3.69. Contorno cerrado definido por un conjunto de puntos	59
Figura 3.70. Contorno cerrado definido por un conjunto de puntos fraccionado marcando un punto interior	59
Figura 3.71. Efecto de la transformación de perspectiva de OpenCV	61
Figura 3.72. Gráfica de la tabla 2.1	62
Figura 3.73. Aproximación lineal para los puntos de la tabla 2.1	62
Figura 3.74. Aproximación lineal para los puntos de la tabla 2.1. n°2	63
Figura 3.75. Gráfica de la nueva función de correspondencia.....	64
Figura 3.76. Gráfica de función de correspondencia para 1Mpx y 3Mpx	65
Figura 3.77. Molde triangular con los punteros montados	66
Figura 3.78. Imágenes del equipo montado y funcionando para las pruebas de sensibilidad.....	66
Figura 3.79. Pantalla inicial de la interfaz de usuario.....	69
Figura 3.80. Figura candidata a ser la proyección de un punto láser, imagen n°1.....	71
Figura 3.81. Figura candidata a ser la proyección de un punto láser, imagen n°2.....	71
Figura 3.82. Figura candidata resultante de la superposición.....	71
Figura 3.83. Vista de la interfaz de usuario una vez ejecutada la detección del triángulo láser	72
Figura 3.84. Presentación del plano detectado en gráfica 3D	73
Figura 3.85. Resultados de distancia, inclinación y dimensiones del triángulo detectado	74
Figura 3.86. Vista de la pestaña de transformación de perspectiva	74
Figura 3.87. Resultados de aplicar la transformación de perspectiva	75
Figura 3.88. Vista de la pestaña de medición de objetos	77
Figura 3.89. Mediciones rectilíneas en la pestaña de medición.....	77
Figura 3.90. Resultados de mediciones rectilíneas en la pestaña de medición	78
Figura 3.91. Mediciones de contornos cerrados en la pestaña de medición	78
Figura 3.92. Resultados de mediciones de contornos cerrados en la pestaña de medición	79
Figura 4.1. Pipeline de la calibración de la cámara	81
Figura 4.2. Pipeline de la detección del triángulo, cálculo de distancia, inclinación y factor de conversión	82
Figura 4.3. Pipeline de la transformación de perspectiva	85
Figura 4.4. Pipeline de la medición de objetos	86
Figura 4.6. Pipeline integrador	86
Figura A.1. Estructura de directorios.....	93
Figura A.2. Imagen de Error.....	93

Índice de Tablas

Tabla 1.1. Análisis de riesgos.....	14
Tabla 2.1. Profundidades de bits de las imágenes digitales.....	16
Tabla 3.1. Correspondencia entre arista en pixeles y distancia en metros.....	61
Tabla 3.2. Comparación entre los valores experimentales de la tabla 3.1 con los generados mediante la nueva función de correspondencia	63
Tabla 3.3. Comparación de valores experimentales con los generados mediante la función de correspondencia para una resolución de 3Mpx	64
Tabla 3.4. Datos de la prueba de sensibilidad N°1	66
Tabla 3.5. Datos de la prueba de sensibilidad N°2	67

Resumen

En el siguiente proyecto se describe un sistema de software y hardware para realizar telemetría mediante imágenes. Haciendo uso de tres punteros láser formando un triángulo se estudia, en imágenes capturadas, la deformación que sufre la proyección del mismo debido a la distancia e inclinación. El sistema desarrollado permite obtener la distancia y las características del plano de proyección, brindando la posibilidad de medir objetos a partir de imágenes.

A lo largo del desarrollo se han realizado pruebas con el correspondiente análisis de resultados, con el fin de elegir los métodos adecuados para cumplir con los objetivos de cada etapa del procesamiento de las imágenes. Luego, para la tarea de transportar lo captado en imágenes a la realidad, se recurre al álgebra lineal y al estudio de la equivalencia entre distancia en unidades de longitud y de pixeles.

Los parámetros para el procesamiento de las imágenes se ajustan mediante una interfaz de usuario, desarrollada para facilitar la operación del software. Esta también permite realizar mediciones sobre la imagen misma.

El desarrollo resulta en una serie de etapas que tienen un orden específico. Entonces, por último se presenta el diagrama explicativo, en forma de pipeline, del proceso y las conclusiones.

Capítulo 1: Introducción

1.1. *Introducción*

En este capítulo se describe la motivación del desarrollo. Se fija el objetivo primario y también objetivos secundarios. Se establecen los requerimientos. Y por último se hace un análisis de los riesgos del proyecto.

1.2. *Motivación*

La motivación nace del estudio del entorno, el querer recaudar información de lo que nos rodea. Donde medir distancia a la que se encuentra algún objetivo es una de las bases. Si bien puede usarse un telémetro para ello, con este desarrollo se da un paso más. Construyendo un sistema que hace las veces de telémetro, pero agregando funcionalidad con el fin de apreciar otras magnitudes.

1.3. *Objetivos*

- Primario:
 - Sistema de telemetría basado en imágenes digitales y láser.
- Secundarios:
 - Analizar las limitaciones del hardware y software.
 - Investigar las alternativas que ofrece OpenCV para trabajar con visión artificial para cada etapa del procesamiento digital de imágenes.
 - Investigar sobre la deformación que provoca la lente de la cámara en las imágenes.
 - Desarrollar un sistema físico que integre la cámara con los punteros láser.
 - Desarrollar software de telemetría.

1.4. *Requerimientos*

El sistema debe cumplir los requerimientos a continuación expresados:

- Medir distancia
- Determinar las características del plano de proyección.
- Mediciones de objetos no solo longitudinales, sino también de área y perímetro.
- Mediciones no solo para planos perpendiculares a los haces de luz de los punteros, sino también para planos inclinados.
- Un grado de robustez que permita encontrar los puntos en la imagen aún en ambientes iluminados.

1.5. *Lineamientos del prototipo*

Para medir distancia y ángulo tres punteros son necesarios, y se colocan formando un triángulo alrededor de la cámara. La posición de la lente de la cámara debe coincidir con el centro del triángulo equilátero, para conocer la posición del centro del mismo en las imágenes de manera inmediata como muestra la imagen 1.1.

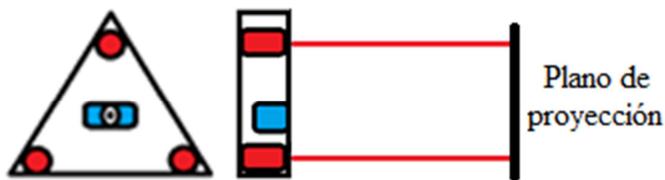


Figura 1.1. Disposición de los punteros láser.

Los haces de luz deben ser paralelos, e idealmente proyectar un triángulo equilátero para cualquier plano de proyección perpendicular a los mismos, sin importar la distancia.

1.6. Análisis de riesgos

Mencionados los objetivos y requerimientos a continuación se detallan los riesgos que se desprenden del software a confeccionar y del hardware a utilizar.

Riesgo	Gravedad	Probabilidad	Medidas de Contingencia
No lograr el paralelismo de los haces de luz	Alta	Alta	Generar las proyecciones en base al comportamiento del fenómeno
Resolución de la cámara insuficiente	Media	Media	Aumentar la resolución utilizando otra cámara
Rendimiento debido al uso de Python	Alta	Media	Utilizar un lenguaje de tipado estático (C por ej.)
Dificultad para encontrar las funcionalidades adecuadas en OpenCV	Alta	Baja	Migrar al uso de otras librerías o confeccionar una propia
No poder detectar las proyecciones en ambientes iluminados	Alta	Alta	Hacer uso filtros de teniendo en cuenta varias características (forma, posición, etc.)

Tabla 1.1. Análisis de riesgos.

Capítulo 2: Marco teórico

2.1. Introducción

En este capítulo se describen los conceptos que constituyen la base del desarrollo.

2.2. Telemetría

La palabra telemetría procede de las palabras griegas tele, que quiere decir a distancia y metrón medida^[1].

2.3. Telémetro láser

Utiliza un rayo láser para determinar la distancia entre el telémetro y un objeto. Funciona según el principio de tiempo de vuelo enviando un pulso láser hacia el objeto, midiendo el tiempo que le toma al pulso rebotar en el objeto y retornar al emisor. Dado que se trabaja con la velocidad de la luz la técnica no es apropiada para mediciones submilimétricas de alta precisión^[2].

2.4. Telemetría mediante visión artificial

Es el objetivo de este proyecto que combina los dos conceptos anteriores con la visión artificial, permitiendo realizar mediciones de objetos a distancia. En este caso utilizando tres punteros láser y mediante la captura y análisis de imágenes digitales de las proyecciones de los mismos.

2.5. Algunos conceptos sobre imágenes digitales

2.5.1. Imágenes digitales

Las imágenes digitales son una representación bidimensional de una imagen a partir de una matriz numérica. Cada elemento en ésta matriz mapea con un pixel de una imagen que es la mínima unidad de información. Específicamente cada elemento es el color de cada pixel y la posición en la matriz es la posición que ocupa en la imagen^[3].

$$\begin{bmatrix} p_{00} & p_{01} & p_{02} & \cdots & p_{0n} \\ p_{10} & p_{11} & p_{12} & \cdots & p_{1n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{m0} & p_{m1} & p_{m2} & \cdots & p_{mn} \end{bmatrix}$$

Figura 2.1. Matriz mxn de una imagen digital.

2.5.2. Resolución

Mientras más grande sea la matriz, más pixeles, más detalle puede observarse en la imagen. Entonces se llama resolución de una imagen digital a la cantidad de pixeles, y

se expresa con dos números enteros donde uno es la cantidad de columnas y el otro la cantidad de filas de la matriz, ancho y alto de la imagen respectivamente. A continuación se muestran ejemplos de distintas resoluciones^[4]:

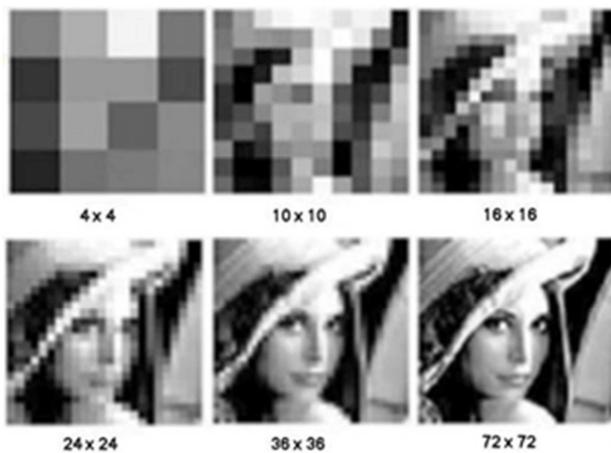


Figura 2.2. Diferentes resoluciones para una misma imagen.

2.5.3. Profundidad de bits

Se denomina profundidad de bits, de color o de pixel, de una imagen digital a la cantidad de bits de información por pixel. Con las combinaciones posibles de estos bits se definen los colores que pueden tomar los pixeles. Según la cantidad de bits por pixel o la forma de interpretarlos podemos definir distintos modos de imágenes digitales. En la siguiente tabla se muestran algunos ejemplos^[5]:

Nombre	Bits por pixel	Fórmula	Nº Colores
B/N	1	2^1	2
Pantalla Windows	4	2^4	16
Escala de Grises	8	2^8	256
Color indexado	8	2^8	256
Color Alta Densidad	16	2^{16}	65 mil
Color Verdadero RGB	24	2^{24}	16,8 millones
Color CMYK alta calidad	32	2^{32}	68,7 millones

Tabla 2.1. Profundidades de bits de las imágenes digitales.

De la tabla 1.1 podemos destacar tres modos:

- Escala de grises:
- Color verdadero RGB
- HSV

2.5.4. Escala de grises

Como indica la tabla 1.1 para cada elemento de la matriz destina un byte por lo que define 256 colores posibles para los pixeles. A continuación tenemos para los valores numéricos del 0 al 255 el color asociado^[6]:

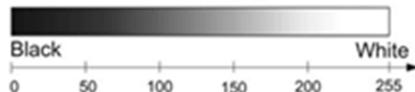


Figura 2.3. Valor asociado a cada pixel en Gray scale.

Podría pensarse que se pierde información al trabajar con una imagen solo en escala de grises. Pero la realidad es que para varios métodos en el procesamiento digital de imágenes es requisito indispensable.

2.5.5. RGB

El modelo RGB (del inglés *Red, Green, Blue*), ocupa 24 bits para definir colores. Se divide en tres canales: canal rojo (*red*), verde (*green*) y azul (*blue*). Al destinar 8 bits por canal la cantidad de colores posibles es $256 \times 256 \times 256$ (16,8 millones)^[7].

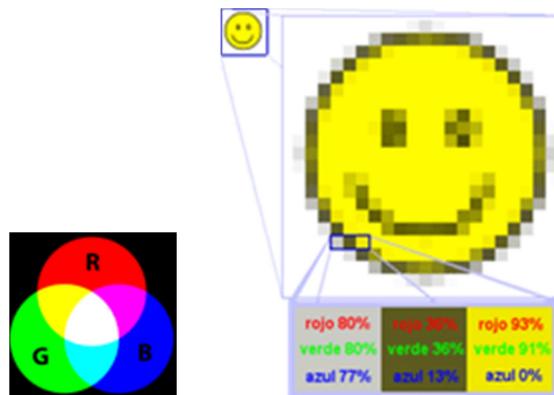


Figura 2.4. Espacio de color RGB diagrama (izq). Ejemplo de porcentajes de cada canal por pixel (der).

También nos podemos encontrar con el espacio de color BGR que es exactamente lo mismo solo que el canal azul y el rojo se intercambian.

2.5.6. HSV

El modelo HSV (del inglés *Hue, Saturation, Value*), al igual que RGB define los colores con tres canales de 8 bits cada uno. *Hue* representa el color, *Saturation* expresa la “pureza” del color y *Value* la luminosidad. La cualidad de poder reconocer un color mediante *Hue*, es decir, independientemente de la cantidad de luz hace a este modelo muy útil para ciertas tareas del procesamiento digital de imágenes, como ser el reconocimiento de objetos.

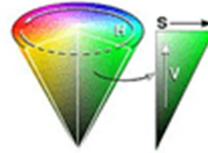


Figura 2.5. Modelo de color HSV.

Los nombres correspondientes en castellano para *Hue*, *Saturation* y *Value* son Matiz, Saturación y Valor.

A matiz le corresponden valores de 0 a 359° , mientras que saturación y valor se expresan en porcentajes (de 0 a 100). Si la saturación es del 100% el color es puro, y al disminuir este se verá decolorado, con una tonalidad grisácea. Como se ve en la figura anterior, valor es la mezcla con blanco o negro del color^[8].

2.6. Visión artificial^[9]

Visión artificial o también llamada visión por computador. Su propósito es programar un computador para que “entienda” una escena o las características de una imagen. Podemos mencionar algunos de sus objetivos, por ejemplo:

- Detección, segmentación, localización y reconocimiento de objetos en imágenes (por ejemplo, cara humanas).
- Seguimiento de objetos en una secuencia de imágenes.
- Mapeo de una escena para generar un modelo tridimensional de la misma.
- Búsqueda de imágenes digitales por su contenido.

2.7. OpenCV^[10]

Es una biblioteca libre de visión artificial desarrollada por Intel. Licencia BSD, es una licencia de software libre flexible respecto a la distribución de modo que el software puede ser redistribuido como software libre o propietario.

OpenCV posee extensa documentación que puede encontrarse en la web e integrarlo a Python resulta sencillo. La versión utilizada para este desarrollo es la 2.4.10. Es de vital importancia conocer la versión de OpenCV con la que se trabaja debido a los cambios que se realizan de una versión a otra. Sobre todo en los valores de retorno de las funciones.

2.8. OpenCV-Python

Python es un lenguaje de propósitos generales muy popular debido a su simplicidad y la legibilidad del código. Comparado con otros lenguajes como C/C++ es lento por el control que realiza sobre los argumentos de las funciones en tiempo de ejecución.

Para mantener un rendimiento aceptable al trabajar con OpenCV se debe tener en cuenta lo siguiente^[11]:

- Evitar los bucles, especialmente los anidados.
- No realizar copias de arreglos a menos que sea estrictamente necesario ya que es una operación muy costosa.
- Vectorizar los algoritmos y código, lo máximo posible ya que OpenCV y Numpy (módulo de Python) están optimizados para operar sobre vectores.
- Explotar la coherencia de cache.

Si bien Python es un lenguaje interpretado, lo que trae aparejado que sea más lento que otros lenguajes como C o C++, fue elegido dado que la comunidad de desarrolladores es muy activa, y para un mismo período de tiempo se puede llegar a ser más productivo que al hacer uso de otros lenguajes. Si surgen problemas de rendimiento conforme al desarrollo avanza se analizarán alternativas.

Nota: Un arreglo es un conjunto de datos o una estructura de datos homogéneos que se encuentran ubicados de forma consecutiva en memoria.

2.9. Distorsión en las imágenes, calibración de cámaras

Afortunadamente es un tema ya conocido y solucionado. Es decir, existe la manera de calcular la deformación que produce la lente de la cámara, obteniendo sus parámetros intrínsecos y extrínsecos a partir de la calibración. Así como la manera de rectificar las imágenes a partir de los parámetros.

Existe una tesis desarrollada por Melisa Elizabeth Del Valle Torres con el título de “Sistema de Visión Estereoscópica para el Control de la Movilidad en Espacios con Obstáculos”. Que hace un extenso análisis de cómo están constituidas y cómo funcionan las cámaras digitales, la deformación en las imágenes capturadas, la obtención de los parámetros extrínsecos e intrínsecos y la rectificación por medio de los mismos.

A grandes rasgos las lentes de las cámaras alteran como es lo capturado en imágenes en la realidad, haciendo que las líneas rectas tiendan a curvarse. Para saber en qué medida se producen dichas alteraciones, se hacen capturas de un tablero de ajedrez a diferentes distancias de la cámara y en distintas posiciones. Las esquinas entre los casilleros negros están en línea recta, por lo que el algoritmo de calibración localiza las esquinas y calcula qué tanto se deforman las líneas^[12].

De la calibración de una cámara obtenemos como resultado valores que indican que tan buena fue la calibración y los parámetros para el rectificado.

Existen lentes que si bien provocan distorsión en las imágenes son utilizadas porque captan una vista más amplia. Es el caso de las cámaras conocidas como *fisheye* que capturan imágenes como la siguiente:



Figura 2.6. Deformación debido a la lente de cámaras fisheye.

Y mediante el rectificado se obtienen resultados como estos:

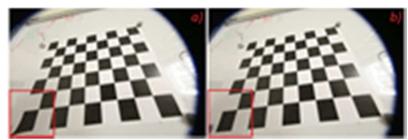


Figura 2.7. Imágenes capturadas con fisheye rectificadas.

Capítulo 3: Desarrollo

3.1. Introducción

Como punto de partida se contaba con la información de que para dos punteros láser de haces de luz paralelos, la distancia entre las proyecciones disminuye a medida que aumenta la distancia al plano donde se proyectan, permitiendo realizar mediciones de distancia. Pero cómo reaccionan las proyecciones ante la inclinación del plano era desconocido. Por lo tanto para conocer el comportamiento del fenómeno desde el comienzo es necesario contar con un prototipo.

A lo largo de este capítulo donde figure código se utilizará #n para referirse a la línea n del mismo.

3.2. Iteración I: Construcción del prototipo y análisis del fenómeno.

3.2.1. Prototipo N°1

Los punteros láser utilizados:



Figura 3.1. Puntero láser.

Características:

- Color: rojo
- Potencia: 5mW
- Longitud de onda 650nm
- Alimentación: 3V
- Dimensiones: 3.5 x 1.1cm
- Vida útil: 5000-8000 horas
- Consumo de 40mA
- Haz de luz invisible
- Foco regulable: permite achicar o agrandar las dimensiones del punto proyectado.

Como base para el montado, es utilizando una cupla reducción de PVC de 110x63 con los punteros fijados en la parte externa y la cámara en el centro:

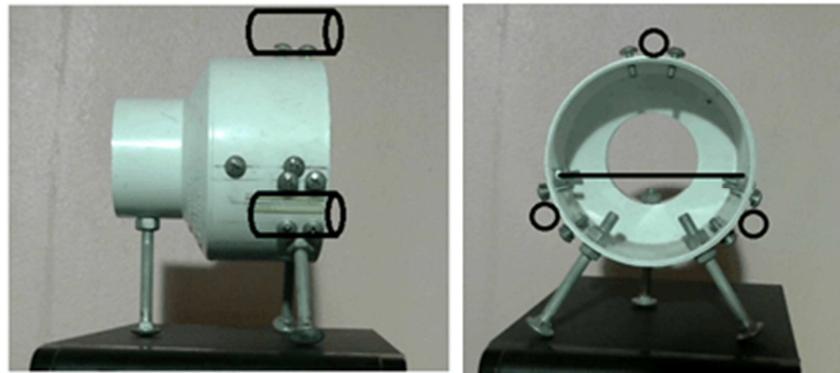


Figura 3.2. Vista lateral del soporte para los punteros láser (izq). Vista frontal del soporte para los punteros láser (der).

El paralelismo perfecto entre los haces de luz es imposible, por ende también lo será la proyección de un triángulo equilátero.

Los punteros son de foco ajustable, cambiando la distancia entre el diodo emisor y la lente. Para esto la lente se encuentra contenida en una rosca, mientras más se ajuste la rosca más cerca del diodo estará la lente. La misma rosca es el problema, no es perfecta, por lo tanto no queda alineada con el diodo emisor y además al ser ajustada o desajustada cambia la dirección del haz.

Para distancias relativamente pequeñas entre la cámara el plano de proyección (no más de 1,4m), la desviación de los haces debido a la falta de paralelismo no es severa. Pero conseguir que el triángulo sea perfectamente equilátero sigue siendo imposible, por lo que el método para la detección de las proyecciones no debe ser muy estricto en cuanto a la posición de las mismas.

La cámara digital utilizada es la siguiente:



Figura 3.3. Cámara digital utilizada.

C270 de Logitech, resoluciones de interés: 1Mpx (intermedia), 3Mpx (máxima). Para realizar capturas de 3Mpx es imprescindible la utilización del software que provee Logitech.

Para alimentar los punteros, es utilizado un transformador de 12V 1A, conectado a una plaqueta.

La plaqueta tiene el circuito necesario para bajar el voltaje de 12V a 3V. Utilizando un regulador LM317, dos trimpots horizontales, dos diodos, cuatro conectores de barril, dos capacitores electrolíticos y uno de cerámica:

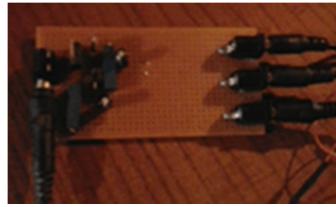


Figura 3.4. Plaqueta reguladora de tensión.

El esquemático del circuito se encuentra disponible en la hoja de datos del LM317.

3.2.2. Principio de funcionamiento

Analizando las imágenes proporcionadas por el prototipo N°1 fue posible conseguir información acerca del fenómeno:

Supongamos un fondo blanco ubicado frente a la cámara, al prender los punteros y capturar una imagen se observará algo como lo que muestra la figura 2.2.



Figura 3.5. Proyección de los puntos láser.

Si aumentamos la distancia entre la cámara con los punteros y el fondo, mientras mayor es la distancia, las aristas del triángulo serán más pequeñas. Superponiendo tres capturas veremos algo así:



Figura 3.6. Proyecciones de los puntos láser superpuestas.

Como puede apreciarse también el área de los puntos disminuye a la distancia.

Supongamos que al fabricar el dispositivo las distancias entre los punteros (o aristas) son de 14cm, por lo tanto las aristas de los tres triángulos de la imagen anterior son todas proyecciones del emisor de 14cm de arista sin importar su tamaño que tengan en las imágenes. En las imágenes se miden las distancias en pixeles por lo tanto:

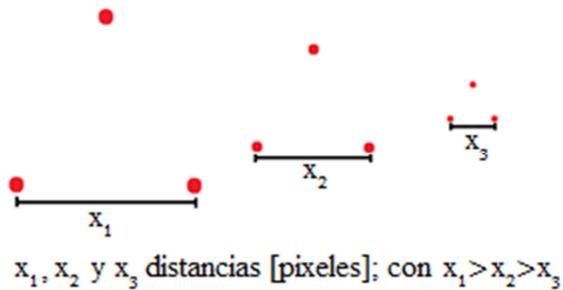


Figura 3.7. Análisis de proyecciones.

La variación de la distancia entre el emisor y el plano donde se proyectan los puntos provoca que cambie la medida en pixeles de la arista. Tomando varias lecturas, se puede confeccionar una función que permita conocer la distancia al objetivo en base a la cantidad de pixeles que posee la arista de la proyección.

3.2.2.1. ¿Por qué son necesarios tres punteros láser?

- Con solo un puntero láser se puede medir distancia mediante el “tiempo de vuelo” (Telémetro láser).
- Con dos se pueden colocar en paralelo y utilizar la misma mecánica y relación que en figura anterior. De igual manera puede ser realizado con un solo puntero y la cámara:

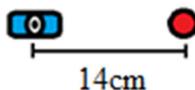


Figura 3.8. Modelo con solo un puntero láser.

- Si bien en los casos anteriores es posible medir la distancia al plano u objeto donde se proyectan los puntos láser, es imposible detectar la inclinación del plano por falta de información. Pero al usar tres punteros es posible obtener las coordenadas de tres puntos pertenecientes al plano de proyección (el método será explicado más adelante). Por ahora solo describimos el comportamiento de la proyección de los puntos al variar la inclinación.

3.2.2.2. Deformación del triángulo

Como muestra la imagen con los tres triángulos, mientras más lejos se encuentre el plano donde se proyectan los puntos del emisor más cercanos al centro de la imagen estarán. Y de manera contraria se alejan del centro al disminuir la distancia de separación. Para verlo con más claridad a continuación se exponen ejemplos con gráficos:

- 1) Inclinación vertical:

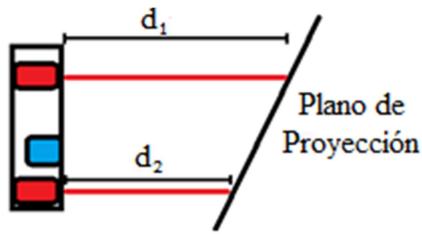


Figura 3.9. Inclinación vertical en el plano de proyección.

En la figura podemos ver que la distancia d_1 es mayor que la distancia d_2 , por lo tanto la cámara capta el fenómeno de la siguiente manera:



Figura 3.10. Deformación del triángulo debido a la inclinación vertical.

Es marcada una cruz para tener referencia del centro del plano. El cambio en el área de los puntos puede notarse, y se acentúa cuando la diferencia entre d_1 y d_2 es mayor. El triángulo ya no es equilátero, se deforma a isósceles ya que la distancia al objetivo para los dos punteros láser inferiores es la misma, d_2 .

La siguiente figura muestra en verde los puntos originales, es decir si el plano fuese perfectamente perpendicular a los haces de luz, y las flechas azules marcan el sentido del desplazamiento que sufren las proyecciones debido a la inclinación del plano.

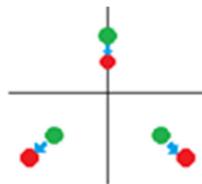


Figura 3.11. Deformación del triángulo debido a la inclinación vertical (análisis).

En el caso de que la inclinación también hubiese sido vertical, pero en el sentido opuesto:

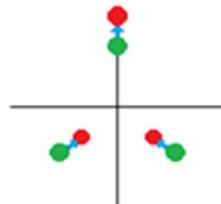


Figura 3.12. Deformación del triángulo debido a la inclinación vertical (análisis) 2.

Nuevamente es isósceles pero con una base más angosta.

2) Inclinación horizontal:

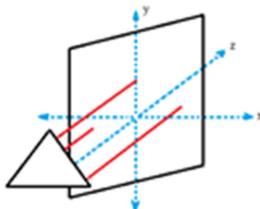


Figura 3.13. Inclinación vertical en el plano de proyección.

Visión de la cámara:

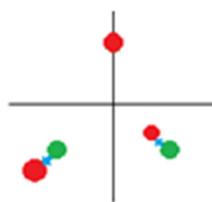


Figura 3.14. Deformación del triángulo debido a la inclinación vertical (análisis).

Si la variación es solo el ángulo de inclinación manteniendo la distancia del centro del emisor al objetivo, es decir, que la coordenada en x del centro de la lente de la cámara en el emisor coincide con la del eje de rotación del plano. En esta oportunidad el punto superior mantiene su posición.

En el sentido opuesto:



Figura 3.15. Deformación del triángulo debido a la inclinación vertical (análisis) 2.

3) Inclinación vertical y horizontal:

Si se mantiene la distancia y se inclina el plano de proyección tanto verticalmente como horizontalmente, los tres puntos sufren desplazamiento. El puntero láser que se encuentre más cerca del plano proyectará el punto más alejado del centro, y el que se encuentre más lejos el más cercano al centro del plano.

3.2.3. Análisis de la deformación del triángulo

La dirección en que se mueven siempre es la misma, es la dirección de las rectas que pasan por el centro de triángulo y los vértices formando ángulos de 120° entre ellas.

Pero el sentido en el que se desplazan los puntos depende del sentido de la inclinación, y cuánto se desplazan depende de la distancia.

Lo ideal es poder calcular la distancia al plano de cada uno de los punteros láser. Para ello será necesaria una función de correspondencia entre arista del triángulo en pixeles y la distancia al plano en metros. Por simplicidad puede ser lineal, o sea de la forma:

$$y = f(x) = ax + b$$

Con pixeles en abscisas y distancia en ordenadas. A más pixeles menos distancia por lo que a será negativo y b positivo. La distancia máxima al objetivo permitida será b cuando $x = 0$ y la distancia mínima delimitada por el valor de x tal que $ax + b = 0$.

3.3. Iteración II: Reconocimiento de los puntos láser

La primera tarea a realizar es detectar los puntos láser en una imagen, más bien llevar a cabo un filtrado para que solo la información relevante permanezca en las imágenes. Para ello solo basta con proporcionarle a un puntero una fuente de poder, y capturar imágenes para su análisis.

3.3.1. Reconocimiento mediante el modelo de color HSV

Al trabajar con rayos láser de colores la primera idea que se nos viene a la mente es la de filtrar el color del láser en la imagen.

HSV divide la información de cada pixel en tres valores (*Hue*, *Saturation* y *Value* o Matiz, Saturación y Valor respectivamente). OpenCV define el matiz para los distintos colores de 0 a 179, saturación y valor ambos de 0 a 255. Para filtrar un color en específico debemos conocer el matiz del mismo. A continuación tenemos los 3 colores principales con el rango de matiz asociado^[13]:

- Azul: 110 - 120
- Verde: 50 - 70
- Rojo: 0 – 10 y 170 – 179

3.3.1.1. Pruebas

Para las pruebas solo es necesario un puntero láser para estudiar las características de la proyección que produce. Son examinadas proyecciones para distintos colores de luz, potencias, inclinación del plano de proyección e iluminación en el ambiente.

- Prueba N°1: La siguiente imagen es de un puntero láser verde de 100mW:

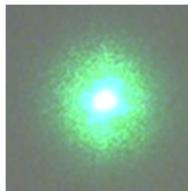


Figura 3.16. Proyección de puntero láser de color verde en BGR.

Código para filtrar el color verde en una imagen:

```

[1] import numpy as np
[2] import cv2
[3] img = cv2.imread("greendot.jpg")
[4] hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
[5] lower = np.array([50,50,50])
[6] upper = np.array([70,255,255])
[7] mask = cv2.inRange(hsv, lower, upper)
[8] result = cv2.bitwise_and(img, img, mask=mask)
[9] cv2.imshow("original", img)
[10] cv2.imshow("hsv", hsv)
[11] cv2.imshow("mascara", mask)
[12] cv2.imshow("resultado", result)
[13] cv2.waitKey()
[14] cv2.destroyAllWindows()

```

#1 y #2 importan los módulos necesarios.

#3 almacena la matriz de la imagen en memoria leyéndola desde la ubicación en donde se encuentra.

#4 cambia el espacio de color de BGR a HSV y guarda la nueva imagen en hsv. Vista de la imagen en HSV:

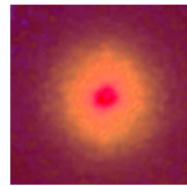


Figura 3.17. Proyección de puntero láser de color verde en HSV.

#5 define el valor inferior del rango para el color verde.

#6 define el valor superior del rango para el color verde.

#7 crea una máscara, es decir una imagen en blanco y negro. Donde solo serán blancos aquellos pixeles cuyo color esté dentro del rango.



Figura 3.18. Máscara resultante para el filtrado del color verde en la imagen.

#8 realiza la operación lógica AND entre la imagen original en formato BGR y la máscara. Resultado:

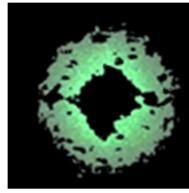


Figura 3.19. Resultado de aplicar la máscara a la figura 3.16.

#9 - #14 muestran las imágenes en ventanas y al presionar alguna tecla termina la ejecución cerrándose todas ellas.

Como puede verse en la imagen resultante, el centro del punto láser no es verde. En el centro es tanta la intensidad luz, tan brillante, que la cámara lo registra como luz blanca.

- Prueba N°2: láser rojo de 5mW de potencia:



Figura 3.20. Prueba n°2: Proyección de puntero láser rojo de 5mW (RGB).

En esta prueba el plano de color oscuro donde se proyecta el punto láser es perpendicular al haz de luz, y la luz en el ambiente es de intensidad media. Ahora filtramos el color usando el código anterior, pero cambiando los valores upper y lower correspondientes para el rojo, y agregando algunas líneas más dado que el rango de matiz está fraccionado en dos:

```
[1] import numpy as np
[2] import cv2
[3] img = cv2.imread("reddot.jpg")
[4] hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
[5] lower = np.array([0,50,50])
[6] upper = np.array([10,255,255])
[7] mask1 = cv2.inRange(hsv, lower, upper)
[8] lower = np.array([170,50,50])
[9] upper = np.array([179,255,255])
[10] mask2 = cv2.inRange(hsv, lower, upper)
[11] mask = cv2.bitwise_or(mask1, mask2)
[12] result = cv2.bitwise_and(img, img, mask=mask)
[13] cv2.imshow("original", img)
[14] cv2.imshow("hsv", hsv)
[15] cv2.imshow("mascara1", mask1)
[16] cv2.imshow("mascara2", mask2)
[17] cv2.imshow("mascara", mask)
[18] cv2.imshow("resultado", result)
[19] cv2.waitKey()
[20] cv2.destroyAllWindows()
```

#4 Imagen en HSV:

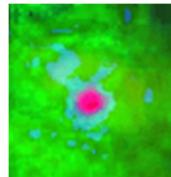


Figura 3.21. Proyección de puntero láser rojo 5mW (HSV).

#7 y #10 crean dos máscaras una para cada rango de matiz:



0-10



170-179

Figura 3.22. Máscaras para filtrado del color rojo. Con rango de matiz de 0 a 10 (izq). Rango de 170 a 179 (der).

#11 Se “suman” las dos máscaras utilizando la operación lógica OR:



Figura 3.23. Suma de máscaras para el filtrado del color rojo.

#12 Resultado:



Figura 3.24. Resultado al aplicar máscara de filtrado.

Puede verse claramente donde se encuentra el centro del punto láser, pero el rojo filtrado no está uniformemente distribuido alrededor del mismo.

- Prueba N°3: en la imagen el plano no es perpendicular al haz de luz y en las mismas condiciones de luz:



Figura 3.25. Imagen original de la prueba n° 3 (izq). Y su correspondencia en HSV (der).



Figura 3.26. Máscaras para filtrado del color rojo. Con rango de matiz de 0 a 10 (izq) y de 170 a 179 (der).



Figura 3.27 Máscara resultante (izq). Resultado de aplicarla sobre la imagen original (der).

Puede apreciarse que disminuye el rojo filtrado, tanto que la máscara correspondiente al rango 0-10 no detecta ningún pixel rojo.

- Prueba N°4: ambiente oscuro, plano perpendicular al haz de luz:



Figura 3.28. Prueba n°4: imagen original (izq). Resultado al aplicar máscara de filtrado (der).

- Prueba N°5: ambiente iluminado, el plano de proyección de color claro y perpendicular al haz:

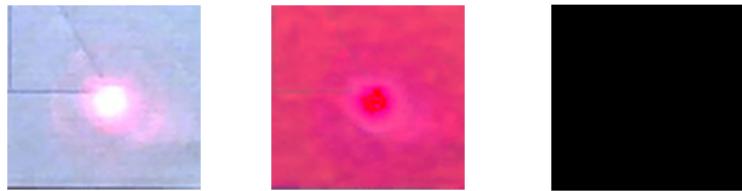


Figura 3.29. Prueba n° 5: Imagen original (izq), conversión a HSV (centro) y resultado al filtrar el rojo (der).

No es detectado ningún pixel con su color en los rangos de color rojo utilizados.

3.3.1.2. Análisis sobre el reconocimiento de los puntos láser utilizando el espacio de color HSV

Podemos concluir a partir de las pruebas realizadas que el filtrado de un color en específico, que obtener una figura que permita determinar el centro del punto láser es posible pero depende de varios factores:

- La luz en el ambiente influye en el resultado y en mayor medida si el láser es de baja potencia. Un ambiente iluminado con 2 lámparas de bajo consumo de 20W ya es suficiente para ocultar todo rastro del color rojo de un láser de 5mW.
- Influye el ángulo que forma el haz de luz con el plano donde se proyecta el punto láser, ya que si este último es perpendicular al haz refleja más luz hacia la cámara.
- El color del plano: si es de color oscuro refleja mucho menos.
- Las formas o figuras detectadas, las que aparecen en las imágenes resultantes, muchas veces son demasiado irregulares. Y en ocasiones fragmentadas en varias partes por lo que determinar el centro del punto láser se hace difícil, se pierde precisión o de pronto es imposible.
- El color rojo de los puntos láser puede que no sea lo único rojo que esté presente en la imagen, por lo que debería corregirse el valor upper y lower, específicamente el valor (el tercer elemento) para filtrar aquellos pixeles de mayor intensidad, o idear algún algoritmo para encontrar los tres puntos. Por otro lado, cambiar dichos valores de pronto altera la figura resultante haciéndola inservible para los objetivos a cumplir.

Al utilizar éste método, la detección de los puntos láser, es muy dependiente de las condiciones en las que es tomada la imagen. Lo ideal es que funcione en ambientes de poca a moderada intensidad de luz, así poder identificar los objetos en la imagen y poder realizar las mediciones.

Por todo lo expuesto sobre el reconocimiento de los puntos láser mediante HSV, el método es descartado y se analizan otras alternativas.

3.3.2. Reconocimiento mediante Thresholding^[14]

Gracias a las pruebas de filtrado utilizando el modelo HSV, sabemos que la cámara ve las zonas de mucha intensidad lumínica como si fuese luz blanca, sin importar si el color del láser es rojo, verde, etc. A diferencia del método anterior, Thresholding trabaja con las imágenes en escala de grises ya que filtra en base a la intensidad.

La traducción de thresholding es umbralización. Aplicado al procesamiento digital de imágenes, significa marcar el umbral entre lo que se quiere hacer resaltar en la imagen y lo que será ignorado. En las imágenes en escala de grises el valor asociado a cada pixel va de 0 a 255, entonces para realizar el thresholding se establece un valor umbral dentro de dicho rango. A continuación se muestran varios modos o estilos de thresholding:



Figura 3.30. Thresholding: imagen en RGB.



Figura 3.31. Thresholding: imagen en escala de grises.

3.3.2.1. Thresholding de umbral fijo

Binario

La siguiente fórmula extraída de la documentación de OpenCV describe cómo trabaja:

$$dst(x, y) = \begin{cases} \text{maxval} & \text{if } \text{src}(x, y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$$

`src(x,y)`: se refiere al valor del pixel de la posición (x,y) de la imagen original.

`thresh`: es el valor umbral, elegido por el desarrollador.

`if src(x,y) > thresh`: si se cumple la condición se asigna al pixel de la posición (x,y) en la imagen resultante (`dst(x,y)`) el valor `maxval`. De lo contrario asigna el valor 0.

maxval: valor seleccionado por el programador para los pixeles que superan el umbral de intensidad.

En resumen el thresholding binario produce una imagen de salida asignando el valor maxval a los pixeles cuya intensidad es mayor que el valor umbral, y a los que se encuentran por debajo de éste les asigna el color negro.

Resultado:



Figura 3.32. Thresholding: Thresholding binario.

Binario Invertido

Exactamente la misma mecánica que el thresholding binario, solo que invierte los valores que asigna a la imagen de salida.

$$dst(x, y) = \begin{cases} 0 & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{maxval} & \text{otherwise} \end{cases}$$



Figura 3.33. Thresholding: Thresholding binario invertido.

Truncado

Aquellos pixeles que estén por debajo del valor umbral no presentan cambio alguno. El valor umbral pasa a ser el máximo de intensidad permitido, para cualquier pixel que esté por encima le será asignado el valor umbral.

$$dst(x, y) = \begin{cases} \text{threshold} & \text{if } src(x, y) > \text{thresh} \\ src(x, y) & \text{otherwise} \end{cases}$$

Resultado:



Figura 3.34. Thresholding: Thresholding binario truncado.

A cero



Figura 3.35. Thresholding: Thresholding a cero (to zero).

Donde los pixeles que sobrepasen el valor umbral permanecerán en las mismas condiciones, y los que estén por debajo, su intensidad será reemplazada por cero.

$$dst(x, y) = \begin{cases} src(x, y) & \text{if } src(x, y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$$

A cero invertido

La intensidad será puesta a cero para todos aquellos pixeles que sobrepasen el umbral, y los demás no sufrirán cambio alguno.

$$dst(x, y) = \begin{cases} 0 & \text{if } src(x, y) > \text{thresh} \\ src(x, y) & \text{otherwise} \end{cases}$$

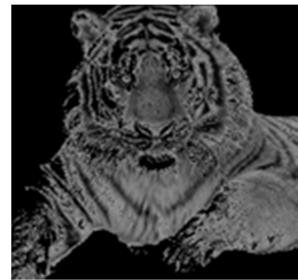


Figura 3.36. *Thresholding: Thresholding a cero invertido.*

3.3.2.2. *Thresholding adaptativo*^[14]

Existen otros tipos de thresholding que se adaptan a las distintas condiciones de luz que pueden existir en una imagen, lo logran mediante la variación del valor umbral. Con estos métodos solo se puede utilizar el thresholding binario y binario invertido.

Thresholding adaptativo, Mean

El nombre deriva del método que utiliza para el cálculo del valor umbral para cada pixel, la media de los valores de intensidad de los pixeles vecinos.



Figura 3.37. *Thresholding adaptativo: Mean + binario.*



Figura 3.38. *Thresholding adaptativo: Mean + binario invertido.*

Thresholding adaptativo, Gaussian

Utiliza la suma ponderada en lugar de la media:



Figura 3.39. Thresholding adaptativo: Gaussian + binario.



Figura 3.40. Thresholding adaptativo: Gaussian + binario invertido.

Entre los parámetros que posee la función `adaptiveThreshold`, es proporcionado el tamaño de la ventana de pixeles que tiene en cuenta para el cálculo del valor umbral, y una constante que resta al valor umbral. El parámetro que define la ventana de pixeles debe ser un número impar mayor que uno, mientras mayor es el número más pixeles vecinos tendrá en cuenta.

3.3.2.3. Análisis del reconocimiento de los puntos láser mediante Thresholding

El objetivo es generar una máscara para los puntos láser proyectados, entonces se debe elegir el tipo de thresholding más adecuado:

- Thresholding adaptativo: Al ver con atención los dos tipos de thresholding adaptativos, en las imágenes resultantes claramente se puede apreciar el tigre. En lugar de obtener una máscara, más que nada resaltan los contornos presentes en la imagen. No cumplen con lo requerido por lo que quedan descartados.
- Truncado: Lo que no satisface la condición lo deja tal cual está en la imagen original, no lo convierte a negro; no servirá como máscara; queda descartado.
- Binario invertido y a cero invertido: convierten a negro los pixeles que sobrepasan el nivel de intensidad, es lo contrario de lo que se necesita. También se descartan.

- Binario: es perfecto para generar una máscara ya que convierte en negro todo lo que no cumple la condición y lo que la cumple a un valor a elección.
- A cero: Es útil para generar una máscara al igual que el binario, con la diferencia que los pixeles que cumplen la condición conservarán el valor de intensidad original.

Idealmente los únicos pixeles con intensidad distinta de cero que sobrevivan al filtro, deben ser los que pertenecen a las figuras creadas por las proyecciones de los tres punteros. Pero en ocasiones, en las pruebas realizadas, cerca de ellas pueden apreciarse pequeñas “manchas”, o bien podría llamarse ruido. Producto de la superficie donde se proyectan los puntos, o de pronto la luz roja alcanza un valor de intensidad que supera al umbral, por lo que será detectado en el thresholding de la imagen. Demostración con una imagen:

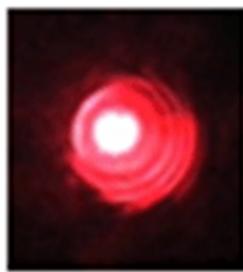


Figura 3.41. Proyección del puntero láser.

Aplicando thresholding con un valor umbral de 200 tenemos:



Figura 3.42. Proyección del puntero luego de aplicar thresholding.

En la zona cercana al punto láser se ven imperfecciones al crear la máscara. Para solucionarlo se probaron varias funciones de filtrado para suavizar imágenes.

3.4. Iteración III: Suavizado de imágenes^[15]

Al igual que las señales a las imágenes se les puede aplicar filtros pasa altos, filtros pasa bajos, etc. OpenCV provee varias funciones que trabajan como filtros pasa bajos para reducir el ruido en las imágenes, tales como:

- Averaging

- Gaussian filtering
- Median filtering
- Bilateral filtering

Las funciones de filtrado, al igual que las de thresholding adaptativo, determinan los cambios a realizar sobre cada pixel mediante su información y la de sus vecinos. Mediante la convolución de la matriz de la imagen con otra matriz cuyo tamaño se especifica como parámetro.

Sea una matriz K de 3x3 como la siguiente:

$$K = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

La matriz K, conocido también como kernel del filtro, se va desplazando por la matriz de la imagen como si fuese una ventana tomando de a 9 pixeles. Con el elemento central toma el pixel que será modificado, y con los demás toma 8 pixeles vecinos para realizar el cálculo del valor que asigna. Lo que diferencia a cada uno de los métodos de filtrado, es como utiliza la información de los pixeles que toma la matriz K, junto con la cantidad de filas, de columnas, los elementos de K y el escalar por el cual se multiplica a K.

3.4.1 Tipos de filtros

3.4.1.1. Averaging (promedio)

Calcula el promedio de los valores de los pixeles que toma la ventana, y reemplaza el valor del pixel central con el promedio calculado. La función de OpenCV es cv2.blur() que requiere dos parámetros: uno es la imagen y el segundo es el tamaño del kernel, que se especifica con una tupla (m,n) por lo que la matriz del kernel será del tamaño mxn.

3.4.1.2. Gaussian filtering (gaussiano)

Útil para filtrar el ruido gaussiano. cv2.GaussianBlur, a diferencia del filtro anterior los elementos de la matriz K no serán todos iguales. Se debe especificar en los parámetros de la función la cantidad de filas y de columnas con una tupla, y la desviación estándar en x e y. Al especificar sólo la desviación en x, en y será del mismo valor. Con cero como tercer parámetro, las desviaciones serán calculadas a partir del tamaño del kernel.

3.4.1.3 Median filtering (mediana)

Con la mediana de los valores de los pixeles que toma la ventana, reemplaza el valor del pixel central. Lo característico es que siempre asigna un valor que verdaderamente se encuentra en la imagen.

Para la función cv2.medianBlur el tamaño del kernel se especifica con un solo valor dado que soporta solo matrices cuadradas.

3.4.1.4. Bilateral filtering (bilateral)

Los tres métodos anteriormente nombrados pueden desdibujar los límites entre las figuras u objetos en una imagen, este en cambio los preserva.

Utiliza dos filtros gaussianos: uno de ellos trabaja en el dominio espacial, tomando ventanas de pixeles y el otro analiza los valores de intensidad. Por lo tanto el primero asegura que solo los pixeles que son vecinos sean considerados para el filtrado, y el segundo los que poseen un valor de intensidad similar al pixel central.

Parámetros de la función bilateralFilter:

- #1 es la imagen de entrada.
- #2 tamaño de la ventana de pixeles.
- #3 sigma/desviación estándar en el espacio de color.
- #4 sigma/desviación estándar en el espacio de coordenadas.

3.4.2. Precauciones y análisis

Con estas dos iteraciones, a las imágenes de entrada se les aplican dos operaciones. Si se realiza el thresholding primero, al realizar el suavizado la ventana de pixeles no tomará los pixeles con intensidad moderada, los pertenecientes a la luz roja de los puntos láser, ya que fueron removidos por no superar el valor umbral. Entonces las figuras resultantes del thresholding estarán rodeadas directamente por pixeles de color negro, en consecuencia la ventana al recorrer los bordes tomará muchos pixeles vecinos con valor cero de intensidad. Además, los efectos del filtrado averaging de desdibujar los límites entre las figuras, y asignarle al pixel central de la ventana valores que de pronto no están presentes en la imagen. Resulta en una imagen donde se puede ver como desde el negro pasa por grises hasta llegar al centro de las figuras y existe un pequeño rastro de las imperfecciones:



Figura 2.43. Proyección del puntero thresholding y luego averaging.

En cambio sí primero realizamos el suavizado de la imagen el resultado es otro:



Figura 2.44. Proyección del puntero con averaging y luego thresholding

Realizadas las pruebas, aplicando primero el suavizado para todos los filtros son eliminadas las imperfecciones. También se puede aumentar el valor umbral, para obtener resultados similares. Pero el aumento del valor umbral hace que el área de las

figuras sea menor, porque la intensidad de los píxeles decrece en dirección a los límites de las mismas. Debe encontrarse el punto de equilibrio según lo que se quiera calcular a partir de las figuras detectadas.

La decisión está entre un suavizado mediante Bilateral filtering o Median filtering, ya que tienen cuidado al momento de asignar el valor al pixel central. Por simplicidad es seleccionado Median filtering.

El siguiente paso es detectar el centro de los puntos láser en la imagen. Dado que se busca el centro de las figuras, podemos usar un valor umbral de 220 para el thresholding. Si bien reducirá el área de las mismas, en el centro es donde la intensidad es máxima, por lo que no habrá ningún problema. Y como serán ignoradas las intensidades de 0 a 220 la detección soportará que el ambiente esté iluminado.

3.5. Iteración IV: Determinar la posición de los puntos láser en la imagen

3.5.1. Método I: “Parasyte”

La primera idea que se podría tener es recorrer la matriz de píxeles de la imagen, buscando un pixel cuya intensidad no sea cero, y de alguna manera determinar el centro de la figura a la que pertenece. Al no encontrar un método que detecte las figuras a la vez que ubica el centro de acuerdo a sus características fue desarrollado el Parasyte.

Durante el recorrido de la imagen, al encontrarse un pixel perteneciente al perímetro, sus coordenadas serán el punto de inicio del método. Para el cálculo y sentido del desplazamiento, para llegar al centro, se considera la intensidad de cuatro píxeles adyacentes al inicial formando una cruz con él. El pixel de inicio será coloreado en rojo, en verde píxeles que pertenecen a la figura, y en amarillo, son coloreados píxeles que no pertenecen a la figura.



Figura 3.45. Mecánica del método Parasyte de cuatro direcciones.

Para el cálculo del desplazamiento, si uno o más píxeles en cualquiera de las cuatro direcciones pertenece a la figura, el centro de la cruz se moverá un pixel en esa dirección, de lo contrario no habrá desplazamiento. Si direcciones opuestas indican desplazamiento, si bien éste no es nulo el desplazamiento resultante sí que lo es.

Los pasos son tres:

- Cálculo del desplazamiento.
- Aplicar el desplazamiento al pixel central.
- Incremento del tamaño de la cruz en un pixel.

El proceso termina una vez que los cuatro pixeles de la cruz se encuentran fuera de la figura.

Como vemos en la figura 2.50, se llega al centro perfectamente. Pero mucho tiene que ver la forma de la figura, que para la cantidad de pixeles con la que está formada, se podría considerar un círculo perfecto. Si utilizamos el mismo método para una figura irregular como la siguiente:



Figura 3.46. Mecánica del método Parasyte de cuatro direcciones para figuras irregulares.

Para este tipo de figuras, el objetivo es encontrar el “centro de masa” de la misma. Ya que en esa zona de la figura es la que tiene más posibilidad de encontrarse el centro del punto láser.

El resultado, en este caso, está lejos de ser ideal. Los pixeles de la cruz quedan fuera de la figura antes de que el central llegue al centro de masa.

Para mejorar el método son incluidas las diagonales, entonces serán ocho las direcciones en las que puede haber desplazamiento.



Figura 3.47. Mecánica del método Parasyte de ocho direcciones para figuras irregulares.

Al tener mayor cantidad de pixeles dentro de la figura, en cada iteración se desplaza más rápido hacia el centro de masa de la figura.

Recorrer la imagen fila por fila, o columna por columna realizando este procedimiento para cada uno de los pixeles blancos encontrados es computacionalmente costoso. Este método nos podría ser útil para realizar ajustes en la posición de las figuras candidatas a ser los puntos láser. Por lo pronto buscamos otras alternativas para determinar la posición de las figuras.

3.5.2. Método II: Hough Circle Transform^[16]

Es una función de OpenCV que permite encontrar los círculos en una imagen, devolviendo las coordenadas de los centros y radios de los mismos. Es común uso en los proyectos donde es necesario la detección de punteros láser. En la documentación podemos apreciar que recibe numerosos parámetros y algunos de ellos bastante específicos, por ejemplo: radio máximo, radio mínimo y distancia mínima entre los centros de los círculos que puedan existir en la imagen. Es de común uso en la detección de punteros láser

Si tenemos la siguiente imagen (“circles3.jpg”):



Figura 3.48. Logo de OpenCV para testing de Hough Circle Transform .

```
[1] import cv2
[2] import numpy as np

[3] img = cv2.imread('circles3.jpg',0)
[4] img = cv2.medianBlur(img,5)
[5] cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)

[6] circles=cv2.HoughCircles(img,cv2.cv.CV_HOUGH_GRADIENT,1,\n    20,param1=50,param2=30,minRadius=0,maxRadius=0)

[7] circles = np.uint16(np.around(circles))
[8] for i in circles[0,:]:
    # draw the outer circle
[9]     cv2.circle(cimg,(i[0],i[1]),i[2],(0,255,0),2)
    # draw the center of the circle
[10]    cv2.circle(cimg,(i[0],i[1]),2,(0,0,255),3)

[11] cv2.imshow('detected circles',cimg)
[12] cv2.waitKey(0)
[13] cv2.destroyAllWindows()
```

#5 crea una imagen igual a la almacenada en la variable img en escala de grises con el espacio de color BGR, de esta manera se podrá graficar en ella con colores los círculos detectados.

#6 cv2.HoughCircles es la función que detecta los círculos.

#7 redondea los valores que describen los círculos (centro y radio), ya que para marcarlos en una imagen deben ser valores enteros.

#9 marca la circunferencia asociada a cada círculo.

#10 marca el centro de cada círculo.

Como resultado obtenemos:



Figura 3.49. Logo de OpenCV para testing de Hough Circle Transform (resultado).

Podemos apreciar que considera círculos formas que no los son y en algunos casos hasta comete errores, esto se debe a la sensibilidad de los parámetros. Sumado a todo esto las condiciones de luz empeoran la situación.

La opción de utilizar esta función también es descartada dado las complicaciones y la necesidad de un método más robusto.

3.5.3. Método III: Mediante la detección de contornos

3.5.3.1. Contornos en OpenCV

OpenCV provee numerosas funciones para trabajar los contornos, la primera que será descripta es la función cv2.findContours. Esta función, como su nombre lo indica, encuentra los contornos en una imagen y los devuelve en una lista. Antes de utilizarla se debe realizar sobre la imagen alguna operación que posibilite un buen contraste. Entonces, en primer lugar, se realiza un thresholding de la imagen y luego se aplica la función.

La función devuelve una lista que contiene todos los contornos (los puntos que los forman), expresados en arreglos. También devuelve un arreglo que contiene tantos arreglos como contornos encontrados, los cuales expresan la jerarquía de cada uno^[17].

Requiere de 3 parámetros:

- La imagen para procesar.
- El modo para expresar la jerarquía entre los contornos.
- El modo de almacenar en memoria los puntos que describen el contorno.

La jerarquía establece un orden para los contornos y brinda algunas características de ellos, esto es, si un contorno envuelve a otro o si es envuelto por uno. Basta con saber que el arreglo con la jerarquía de cada contorno consta de 4 elementos: los dos primeros denotan el orden, y los dos restantes, si contienen un contorno en su interior o son contenidos. El arreglo de jerarquía estaría constituido de la siguiente manera^[18]:

[Contorno próximo, Contorno anterior, Primer hijo, Contorno padre]

Si a un contorno de ID 40 el contorno de ID 47 lo envuelve, entonces 47 es el padre de 40. Por lo que el último elemento del arreglo de jerarquía del contorno 40 será 47. De manera general: si existe algún contorno que cumpla alguno de los cuatro vínculos su ID estará en el arreglo de jerarquía en la posición que corresponde, si no existe para alguno en esa posición tendremos -1.

El tercer parámetro indica si almacenar todos los puntos del contorno en memoria o solo los necesarios. Por ejemplo si se trata de un rectángulo perfecto, no tiene sentido almacenar más puntos además de los vértices del mismo.

Encontrar los contornos para determinar la posición de los puntos láser

La función descripta en el punto anterior “findContours” da a conocer los puntos que forman cada contorno. Realizará una detección más general de las figuras, por lo que es necesario idear un algoritmo que haga las veces de filtro.

Encontrados los contornos para cada contorno podemos usar la función cv2.minEnclosingCircle, que retorna el círculo que encierra a un contorno que se pasa como parámetro. Con la función obtenemos las coordenadas del centro y el radio. Ejemplo para una figura cualquiera:

Partiendo de la siguiente imagen, que bien puede ser el resultado de aplicar un thresholding binario, veremos el resultado de la utilización de ambas funciones:



Figura 3.50. Figura para testing de la función findContours de OpenCV.

Código:

```
[1] import cv2
[2] import numpy as np
[3] img = cv2.imread('c.jpg',0)
[4] img = cv2.threshold(img, 220, 255, cv2.THRESH_BINARY)[1]
[5] cimg = np.zeros((img.shape[0],img.shape[1], 3), np.uint8)
[6] c, _= cv2.findContours(img, cv2.RETR_TREE,\n    cv2.CHAIN_APPROX_SIMPLE)
[7] circle = cv2.minEnclosingCircle(c[0])
[8] cv2.drawContours(cimg, c, -1, (0,255,255), 1)
[9] cv2.circle(cimg, (int(circles[0][0]), int(circles[0][1])),\n    int(circles[1]), (0, 255, 0), 1)
[10] cv2.imshow('Input image', img)
[11] cv2.imshow('Detected contours',cimg)
[12] cv2.waitKey(0)
[13] cv2.destroyAllWindows()
```

#5 crea una imagen con todos sus pixeles con valor cero (color negro) con la misma resolución que la imagen de entrada pero con 24 bits por pixel para que esté en el espacio de color BGR.

#6 almacena en la variable “c” los contornos encontrados en la imagen. Donde se encuentra “_”, de colocar una variable allí, se le asignará la jerarquía de los contornos.

#7 Dado que en la imagen solo tenemos un contorno, la función cv2.minEnclosingCircle recibe como parámetro el único elemento en la lista de contornos encontrados.

#8 dibuja los contornos encontrados en la imagen

#9 dibuja la circunferencia del círculo envolvente. Con el último parámetro se especifica el grosor, y si este es negativo se rellena la circunferencia dibujando el círculo.

Los parámetros del círculo deben ser valores enteros por lo que requieren de la transformación int().

Una imagen que se guarda desde algún editor en ocasiones es suavizada alterando el valor de los pixeles, en mayor medida en los bordes de las figuras presentes. Por este motivo, para evitar obtener un resultado erróneo con la función cv2.findContours, primero se realiza el thresholding binario para el máximo contraste posible, y luego se aplica la función. Por más de que la imagen de entrada ya esté en blanco y negro.

Además, al ejecutarse #6, se altera la imagen de entrada. Si es necesaria más adelante en el código, se tendrá que realizar una copia previa o leer nuevamente.

Imagen de entrada alterada:



Figura 3.51. Repercusiones sobre la imagen de entrada para la función findContours.

Resultado de dibujar el contorno junto con el círculo envolvente:



Figura 3.52. Contorno detectado junto con el círculo envolvente.

Al obtener la posición de los contornos con este método, puede que la forma de la figura haga que esta no coincida con la posición en la que estaría el centro de masa. Reutilizando el método I (Parasyte), pero en esta oportunidad iniciando desde el centro del círculo, se puede realizar una corrección del centro del contorno según la forma que tiene. Lo vemos en la siguiente figura, donde el círculo envolvente se marca en violeta, y se conservan las referencias (pixeles en verde y amarillo) en cuanto a las marcas del método Parasyte.



Figura 3.53. Parasyte para ajustar el centro de una figura.

Si el punto ya se encuentra en una posición aceptable tiende a conservarla:

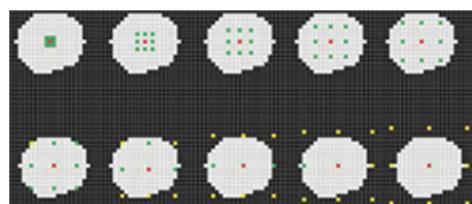


Figura 3.54. Parasyte para ajustar el centro de una figura n°2.

El método “Parasyte” desplaza el centro de acuerdo a la forma de la figura. Pero si para obtener las figuras se utiliza el estilo de thresholding TOZERO (la intensidad varía en la figura), más útil es un desplazamiento que trabaje con la intensidad. Posee la misma mecánica pero desplaza el centro en dirección a donde haya más intensidad, y además reduce los pixeles a tener en cuenta para el desplazamiento. Usa cuatro ya que si cuenta con 8, el salto de dos pixeles o más puede encerrarlo en un bucle infinito.

Cuando se realice el análisis de la imagen obtenida por la cámara, según la cantidad de luz en el ambiente, podremos tener muchas figuras después de aplicar el thresholding. Las figuras de los puntos láser tendrán una tendencia a ser formas circulares, por lo tanto podremos aplicar un filtro en cuanto a la forma. Si el contorno de una figura tiende a ser circular, el área del mismo será muy parecida al área del círculo que lo contiene, o por lo menos será un valor más cercano que el que puedan llegar a tener las figuras que sean irregulares.

El discriminante, para el filtrado, será el cociente entre el área del contorno y el área del círculo que lo envuelve. Si dicho valor supera, por ejemplo, los 0,6 el contorno es candidato a ser el contorno de una figura creada por los puntos láser.

A este filtro, ya que se analizan todos los contornos encontrados en la imagen, sumaremos otra condición debido a la siguiente situación:



Figura 3.55. Figura con orificio.

Contornos encontrados:



Figura 3.56. Figura con orificio, contornos detectados.

Para una misma figura encuentra dos contornos. Las figuras pertenecientes a los puntos láser por ningún motivo tendrán un contorno hijo, o un contorno padre, por lo tanto los dos últimos elementos del arreglo de jerarquía deben ser igual a -1. De esta manera se hace más robusto el filtro.

3.6. Iteración V: Encontrar las figuras que forman el triángulo

Por más que se establezca un valor de thresholding alto y el análisis de contornos, más figuras además de las de los puntos láser pueden no ser filtradas.

Del paso anterior conocemos las posiciones de las figuras candidatas a ser las de los puntos láser, entonces lo siguiente es confeccionar un algoritmo que nos informe cuales son las posiciones de los puntos del triángulo.

El algoritmo debe poder localizar un triángulo de cualquier tamaño, ya que este será más chico a mayor distancia entre el emisor y el plano de proyección, y más grande a menor distancia.

3.6.1. Imagen como plano en 2D

No está de más mencionar que una imagen puede ser pensada como un cuadrante del eje de coordenadas en R2. Precisamente como el primer cuadrante pero con el eje de las ordenadas invertido. Las dimensiones del cuadrante concuerdan con la resolución de la imagen y los puntos son discretos como los pixeles de una imagen.

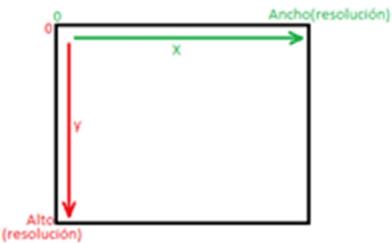


Figura 3.57. Imagen como plano xy

Para el caso de los pares ordenados se expresan con la coordenada horizontal primero seguida de la vertical. Las funciones de OpenCV que requieren la posición de un pixel o devuelven una, siguen el mismo orden. Pero si se desea marcar un pixel, recortar una porción de la imagen, o alguna otra operación que involucra directamente acceder a la matriz de la imagen, se invierten las coordenadas. La primer coordenada indica la fila de la matriz por lo que es la altura y la segunda indica la columna que mapea con la coordenada horizontal.

Entonces queda establecido que cuando se hable de coordenadas serán expresadas como x e y , coordenada horizontal y coordenada vertical respectivamente. Con x que aumenta de izquierda a derecha, e y que lo hace de arriba hacia abajo.

3.6.2. Método “Trinity Hunter”

Trinity Hunter fue el método desarrollado para resolver la detección del triángulo, ante la falta de un método que tenga en cuenta características tan específicas.

El algoritmo debe tener un punto de partida, este es encontrar un primer punto y a partir de él los demás.

De manera general, para la detección de los punteros formando un triángulo en cualquier parte de la imagen, es la proyección del punto inferior derecho la que siempre se encuentra a mayor distancia del vértice superior izquierdo de la imagen.

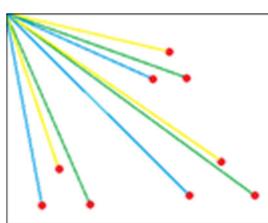


Figura 3.58. Proyección del triángulo en cualquier zona de la imagen.

Por más que el triángulo se deforme siempre será la distancia marcada en verde mayor a la azul y amarilla.

Para encontrar los puntos restantes, los candidatos deben cumplir con ciertas condiciones:

- Los puntos inferiores tendrán poca diferencia respecto a la coordenada vertical, se puede establecer un rango o tolerancia para la misma.
- El punto superior horizontalmente siempre se encuentra entre los puntos inferiores.
- El punto superior siempre es superior en y a los otros dos.

Para detectar algo en específico en una imagen, agregar condiciones a cumplir acelera la convergencia. Si la cámara es colocada en el centro de los punteros láser, como se estableció en los lineamientos del prototipo, es posible sumar más condiciones a cumplir.

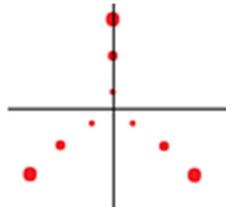


Figura 3.59. Proyecciones superpuestas con el centro de la imagen como centro del triángulo.

Punto inferior izquierdo, nuevas condiciones:

- Siempre se encuentra por debajo de la línea horizontal que pasa por el centro.
- Siempre se encuentra a la izquierda de la línea vertical que pasa por el centro.

Punto superior, nuevas condiciones:

- Siempre coincide verticalmente con el centro, es decir tienen la misma coordenada horizontal.
- Siempre se encuentra por encima de la línea horizontal que pasa por el centro.

Debe seguir cumpliéndose la tercera de las primeras condiciones (“el punto superior siempre es superior en y a los otros dos”), dado que puede presentarse la siguiente situación:

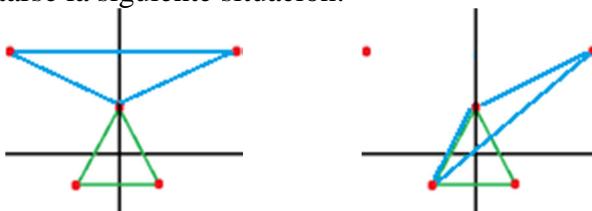


Figura 3.60. Interpretaciones del triángulo según los puntos encontrados. Aplicando las nuevas condiciones para el punto superior (izq). Aplicando las nuevas condiciones para el punto superior y el inferior izquierdo (der).

En ambos casos el triángulo en azul es incorrecto pero cumple con las condiciones propuestas. Conservando la tercera de las anteriores, se llega triángulo verde que es el correcto (Fig. 3.60 (izq)). Para la figura 3.60 (der) se requiere un replanteo de la condición del punto inicial.

Puede darse el caso de que el punto inferior derecho cumpla también las condiciones para ser el izquierdo, por lo tanto es necesario verificar que no sea el mismo punto. Reemplazando la condición de distancia máxima para el punto de partida, por una que exija que el punto se encuentre en el cuadrante inferior derecho, se limita el área de búsqueda a ese cuadrante y se evita la verificación.

El algoritmo recibe un arreglo de 3 o más elementos con los puntos candidatos detectados, y devuelve los tres que forman el triángulo alrededor del centro. Si encuentra el triángulo, si no se encuentra ningún punto en el cuadrante inferior derecho, o si la longitud del arreglo de entrada es menor a 3 (al inicio o en alguna iteración) termina el proceso.

3.6.2.1. Diagrama de actividad del algoritmo

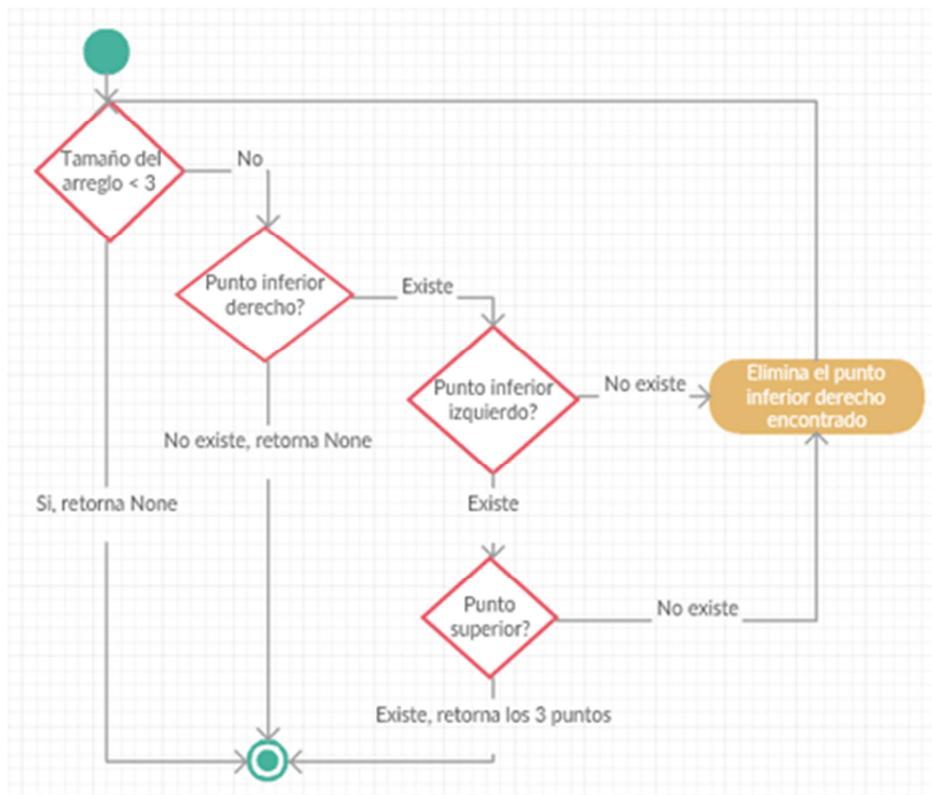


Figura 3.61. Diagrama de actividad del algoritmo Trinity Hunter.

3.6.2.2. Condiciones definitivas del algoritmo

Condiciones del punto inferior derecho:

- Debe pertenecer al cuadrante inferior derecho de la imagen.

Condiciones del punto inferior izquierdo:

- Debe estar en el cuadrante inferior izquierdo de la imagen.
- La diferencia entre su coordenada vertical y la del punto inferior derecho debe ser menor o igual a una tolerancia máxima configurable.

Condiciones del punto superior:

- La diferencia en entre la coordenada horizontal con respecto a la del centro debe ser menor o igual a la tolerancia máxima (permite un poco de libertad).
- Debe pertenecer a la parte superior de la imagen, o sea la coordenada vertical debe ser menor a la del centro.
- La coordenada horizontal debe ser mayor a la del punto inferior izquierdo y menor que la del inferior derecho.

Nota: La última condición a simple vista parece no ser necesaria, pero mientras más distante se encuentra el plano de proyección menor será la distancia entre los puntos inferiores del triángulo, por lo que en algún momento dicha distancia puede ser menor a la máxima tolerancia (si no se ajusta).

3.6.3. Método “Trinity Hunter Rev”

Solo hace un mejor uso de la tolerancia para la detección de los puntos inferiores del triángulo. En el caso anterior es la diferencia máxima que puede existir entre las coordenadas verticales de los puntos inferiores, u horizontales entre el centro y el punto superior, pero al estar el plano de proyección inclinado, la deformación del triángulo hace que se incrementen las diferencias. Si es muy grande la diferencia la tolerancia debe ser más grande también, por lo que se pierde robustez; los puntos tendrían más libertad en cuanto a la posición y figuras incorrectas podrían ser tomadas en cuenta (falso positivo).

Es posible calcular en la imagen que pixeles están sobre las rectas que forman el centro con los vértices del triángulo, que son las direcciones de desplazamiento de cada punto. Valiéndonos los ángulos centrales de 120° que posee el triángulo equilátero:

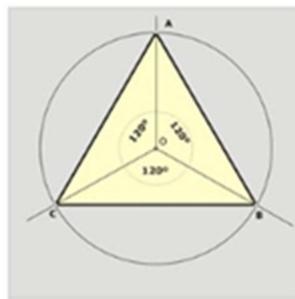


Figura 3.62. Triángulo equilátero circunscrito y ángulos interiores.

Se han de construir dos rectas: una desde el centro y pasa por el punto C, y la otra desde el centro pasando por el punto B. Luego la tolerancia será utilizada como la distancia máxima a la que puede estar cada punto de su dirección, o recta de desplazamiento.

Para el punto superior no es necesaria la construcción de una recta, basta con comprobar que la diferencia entre su coordenada horizontal y la del centro sea menor o igual a la tolerancia.

La ventaja es que es más acorde al modo en que se deforma el triángulo.

La desventaja es que requiere más precisión de construcción del dispositivo en cuanto al paralelismo de los haces de luz si se quiere usar una tolerancia más baja. La tolerancia como mínimo puede ser igual a 1 dado que las posiciones de los píxeles son discretas.

Este algoritmo logra aumentar la robustez del proceso de encontrar el triángulo. Lo siguiente es el análisis del mismo para determinar la distancia al objetivo y la inclinación.

3.7. Iteración VI: Análisis del Triángulo

En marco teórico quedó establecido el método para calcular la distancia entre la cámara y el plano donde se proyectan los puntos láser. La medición es incorrecta si no se tienen en cuenta las características del plano, la longitud de la arista se incrementa al haber inclinación. Ahora... ¿Cómo determinar las características del plano?

3.7.1. Calculo de la inclinación mediante la ecuación del plano

Con tres puntos es posible determinar la ecuación del plano que los contiene. De los tres puntos, de más está decir que son los puntos láser, las posiciones que ocupan en la imagen serán las primeras dos coordenadas y la tercera coordenada es la que tiene que ser calculada.

Para el cálculo utilizaremos las distancias en píxeles desde el centro del triángulo a los vértices. Para un triángulo equilátero son todas del mismo valor, que es el radio de la circunferencia que pasa por los tres vértices y encierra al triángulo.

Podemos determinar el radio fácilmente, es la distancia desde el centro de la imagen a un vértice, y luego calcular el valor de la arista de un triángulo equilátero para ese radio.

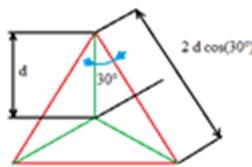


Figura 3.63. Medida de arista a partir del radio de un triángulo.

Calculamos el valor de la arista correspondiente para cada uno de los radios, ya que el triángulo podría deformarse a isósceles o de pronto escaleno. En la siguiente imagen el triángulo en rojo es el original (escaleno), y vemos los triángulos equiláteros que corresponden a los radios según su tamaño:



Figura 3.64. Triángulos equiláteros para cada uno de los radios de un triángulo escaleno.

Cada triángulo equilátero es del mismo color que el radio que le corresponde. Puede que no sean perfectamente equiláteros en el dibujo, pero la idea es mostrar que solo con conocer un radio se puede llegar al triángulo equilátero.

Con el valor de cada arista y la fórmula:

$$\text{distancia} = \alpha \text{ arista} + b$$

Podemos calcular a qué distancia de la cámara se encuentra cada punto del triángulo. Para obtener los tres puntos tenemos dos opciones:

- 1) Tomar las coordenadas x, y de los tres puntos detectados en la imagen y agregar como tercer coordenada las distancias calculadas a partir de las aristas^[19]:

$$p_0 = (x_0, y_0, z_0)$$

$$p_1 = (x_1, y_1, z_1)$$

$$p_2 = (x_2, y_2, z_2)$$

Luego seguir el procedimiento normal para determinar las características de un plano, calculando primero los siguientes vectores:

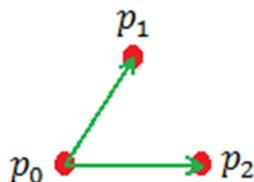


Figura 3.65. Vectores definidos por tres puntos en el mismo plano.

$$\overrightarrow{p_0p_1} = (x_1 - x_0, y_1 - y_0, z_1 - z_0)$$

$$\overrightarrow{p_0p_2} = (x_2 - x_0, y_2 - y_0, z_2 - z_0)$$

Con ellos calculamos el vector perpendicular al plano mediante el producto vectorial:

$$\vec{n} = \overrightarrow{p_0p_1} \times \overrightarrow{p_0p_2} = (n_0, n_1, n_2)$$

Vale la pena mostrar la función que realiza este último cálculo:

```

def get_nv(p1p2v, p1p3v):
    return [float(p1p2v[1] * p1p3v[2] - p1p2v[2] * p1p3v[1]), float(p1p2v[2] * p1p3v[0] - p1p2v[0] * p1p3v[2]), float(p1p2v[0] * p1p3v[1] - p1p2v[1] * p1p3v[0])]

```

Es muy importante el uso de `float(...)` para cada valor del vector, para que los valores que tengan decimales no sean tomados como enteros, siendo los más importantes los que tienen valor absoluto entre cero y uno. Estos últimos pueden transformarse en ceros anulando completamente el vector normal al plano.

Con un punto que pertenece al plano y el vector perpendicular al mismo podemos calcular la coordenada z para cualquier par x, y .

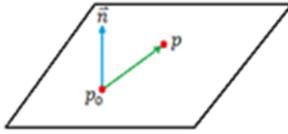


Figura 3.66. Vector normal al plano.

Tomamos un punto p cualquiera y p_0 , el producto escalar entre dos vectores ortogonales es cero, por lo que solo hay que determinar el valor de z de la siguiente ecuación:

$$\overrightarrow{p_0p} = (x - x_0, y - y_0, z - z_0)$$

$$\overrightarrow{p_0p} \cdot \vec{n} = 0$$

$$(x - x_0, y - y_0, z - z_0) \cdot (n_0, n_1, n_2) = 0$$

$$(x - x_0)n_0 + (y - y_0)n_1 + (z - z_0)n_2 = 0$$

$$(z - z_0)n_2 = -[(x - x_0)n_0 + (y - y_0)n_1]$$

$$z = \frac{-[(x - x_0)n_0 + (y - y_0)n_1]}{n_2} + z_0$$

$$z = \frac{(x_0 - x)n_0 + (y_0 - y)n_1}{n_2} + z_0$$

Ahora podemos generar tantos puntos del plano como queramos, pero este plano posee las primeras dos coordenadas como posición de pixel y la tercera en metros, por lo tanto si se calculan los ángulos de inclinación del plano no serían los verdaderos. El plano si nos será útil para calcular a qué distancia de la cámara se encuentra punto del centro del plano, y con esta distancia calcularemos cuantos pixeles tiene la arista del triángulo equilátero para esa distancia. Entonces...

Distancia en metros entre el centro de la imagen y la cámara:

$$centro\ de\ la\ imagen = (c_x, c_y)$$

$$c_z = \frac{(x_0 - c_x)n_0 + (y_0 - c_y)n_1}{n_2} + z_0$$

Arista en pixeles según la distancia al plano:

$$arista = \frac{distancia - b}{\alpha}$$

Se calcula el z del centro, ya que si se trata de un plano inclinado la verdadera distancia es entre el punto en el plano que coincide con el centro de la imagen y la cámara.

La separación (arista) que existen entre los punteros láser del aparato físico es conocida, podemos realizar una conversión para determinar a cuántos metros equivale la distancia entre cualquier par de puntos sobre el plano. Para ello con la arista obtenida hacemos el cálculo del siguiente factor de conversión:

$$\varepsilon = \frac{arista\ [metros]}{arista\ [pixeles]}$$

No solo podemos utilizarlo para calcular dimensiones en el plano, sino también para hacer compatible la tercera coordenada de cada punto con las dos primeras:

$$distancia\ en\ metros = distancia\ en\ pixeles * \varepsilon$$

2) La otra alternativa consiste en:

- Tomar las distancias en metros de los tres puntos detectados y calcular el promedio, que es una buena aproximación de la distancia a la que se encuentra el centro del triángulo.
- Con la distancia obtenida utilizar la fórmula lineal para obtener la medida de arista.
- Con la arista obtener el factor de conversión.
- Transformar las primeras dos coordenadas de los puntos de pixeles a metros utilizando el factor de conversión.

Lo que sigue vale para las dos alternativas:

Con los tres puntos con todas sus coordenadas en metros calculamos (nuevamente en el caso de la primera opción) el vector perpendicular al plano, y un punto perteneciente

(bien puede ser uno de los que ya tenemos). Con estos dos elementos podemos obtener puntos del plano y los ángulos de inclinación del plano.

Para expresar la inclinación del plano serán calculados dos ángulos:

- El ángulo vertical (*V angle*): es el ángulo que forma el vector normal al plano \vec{n} con el plano *xz*.
- El ángulo horizontal (*H angle*): es el ángulo que forma \overrightarrow{nxz} , que es la proyección de \vec{n} en el plano *xz*, con el plano *zy*.

La fórmula se obtiene de la utilizada para el cálculo del producto punto^[19]:

$$\vec{u} \cdot \vec{v} = \|\vec{u}\| \|\vec{v}\| \cos \beta$$

Luego:

$$\beta = \cos^{-1} \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \|\vec{v}\|}$$

Entonces:

$$V \text{ angle} = \cos^{-1} \frac{\vec{n} \cdot \overrightarrow{nxz}}{\|\vec{n}\| \|\overrightarrow{nxz}\|}$$

Con $\overrightarrow{nxz} = (n_0, 0, n_2)$

$$H \text{ angle} = \cos^{-1} \frac{\overrightarrow{nxz} \cdot \overrightarrow{nz}}{\|\overrightarrow{nxz}\| \|\overrightarrow{nz}\|}$$

Con $\overrightarrow{nz} = (0, 0, n_2)$

Habiendo construido el plano tenemos la información necesaria para realizar mediciones de objetos.

3.8. Iteración VII: Determinar las dimensiones reales de objetos desde una imagen

Un problema que se presenta en las imágenes a la hora de hacer mediciones es la perspectiva. Una caja rectangular en perspectiva con una inclinación horizontal de 45° se ve de la siguiente manera:

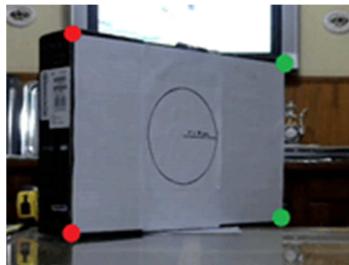


Figura 3.67. Deformación de dimensiones de acuerdo a la perspectiva.

La perspectiva ocasiona que la distancia entre los puntos rojos sea mayor que la que existe entre los puntos verdes, por más que en realidad sean exactamente iguales. Entonces, no podemos asumir que las coordenadas de los puntos marcados en la imagen son las mismas que las de los vértices de la caja en el plano detectado. Para esta inclinación los puntos sobre el plano, aproximadamente y en amarillo, serían:



Figura 3.68. Deformación de dimensiones de acuerdo a la perspectiva n°2.

3.8.1. *Mediciones para planos sin inclinación*

Comenzamos ideando la manera de resolver el caso más simple, esto es para imágenes donde no se detecta inclinación.

Las mediciones serán válidas para aquellos objetos que se encuentren a la distancia detectada, es decir solo podremos medir el objeto sobre el cual se proyectan los puntos láser.

3.8.1.1. *Mediciones de largo y ancho (rectilíneas)*

Son muy simples de realizar basta con tomar las coordenadas x, y de dos puntos en la imagen, calcular la distancia entre ellos (en pixeles), y aplicar el factor de conversión a la distancia para obtener la medida en metros.

3.8.1.2. *Mediciones de área y perímetro*

Son necesarios 3 o más puntos de la imagen para formar un contorno cerrado. Para calcular el perímetro, se toman los puntos de a pares y se suman las medidas de longitud de las rectas que forman.

Para medir área, supongamos el siguiente contorno construido a partir de 6 puntos:

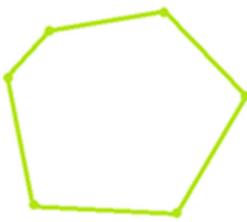


Figura 3.69. Contorno cerrado definido por un conjunto de puntos.

Se calculan las coordenadas (x, y) de un punto interior del contorno, donde x será el promedio de todas las coordenadas horizontales e y el de las verticales de todos los puntos del contorno. El resultado será un punto interior, en el centro o cercano, para subdividir el contorno en triángulos:



Figura 3.70. Contorno cerrado definido por un conjunto de puntos fraccionado marcando un punto interior.

Se toma triángulo por triángulo, se calcula la longitud de las tres rectas que componen cada uno, y se hace la conversión de la longitud con el factor de conversión. Mediante la fórmula de Herón se obtiene el área de cada triángulo y la suma de todas ellas será el área del contorno.

Fórmula de Herón ^[19]

El área de un triángulo de lados a, b y c es:

$$S = \sqrt{\rho(\rho - a)(\rho - b)(\rho - c)}$$

Siendo ρ el semiperímetro del triángulo, es decir:

$$\rho = \frac{a + b + c}{2}$$

Para el cálculo del perímetro basta con tomar la recta de cada triángulo que forma parte del contorno original y sumarlas.

OpenCV posee funciones para calcular área y perímetro de contornos, estas son:

- Área: cv2.contourArea(contour)
- Perímetro: cv2.arcLength(contour, True)

El parámetro “contour” son los puntos que componen el contorno, para obtenerlos tenemos dos alternativas:

1. Almacenar los puntos marcados en la estructura de datos que OpenCV interpreta como contorno.
2. Que se genere a partir de filtrar el color con el cual está dibujado el contorno, y luego detectarlo con la función “cv2.findContours”.

Si bien ambas alternativas no calculan el área y perímetro de manera perfecta, la sumatoria de triángulos tiene menos error, por lo que es elegida como método para obtener área y perímetro.

3.8.2. Transformación de perspectiva^[20]

Si el plano presenta inclinación se debe realizar una transformación previa a las mediciones, esta es una transformación de perspectiva. El motivo es que el factor de conversión es calculado para el plano detectado. Para hacer mediciones en 2D se debe revertir la inclinación del plano donde se proyectan los puntos manipulando la imagen.

La transformación de perspectiva que provee OpenCV requiere que se seleccionen cuatro puntos de la imagen en coordenadas x, y , así como las nuevas coordenadas que ocuparan los puntos.

```
[1] img = cv2.imread('sudokusmall.png')
[2] rows,cols,ch = img.shape

[3] pts1 = np.float32([[56,65],[368,52],[28,387],[389,390]])
[4] pts2 = np.float32([[0,0],[300,0],[0,300],[300,300]])

[5] M = cv2.getPerspectiveTransform(pts1,pts2)

[6] dst = cv2.warpPerspective(img,M,(300,300))

[7] plt.subplot(121),plt.imshow(img),plt.title('Input')
[8] plt.subplot(122),plt.imshow(dst),plt.title('Output')
[9] plt.show()
```

#2 Obtiene el número de filas de la matriz de la imagen en la variable “rows”, el número de columnas en “cols”, la cantidad de bytes de información por pixel en “ch”.

#3 pts1 almacena los puntos originales, los que se marcan primero.

#4 pts2 guarda las coordenadas nuevas de los puntos.

#5 calcula la matriz para la transformación de perspectiva.

#6 aplica la transformación y recorta la imagen.

#7-9 muestra la imagen de entrada y el resultado.

En lugar de (300,300) en la línea #6 podría colocarse en su lugar (cols,rows) que es la resolución de la imagen para no realizar recorte alguno.

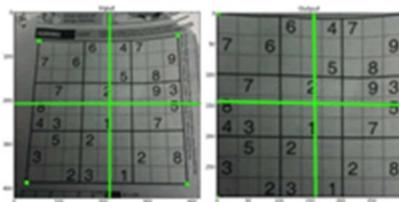


Figura 3.71. Efecto de la transformación de perspectiva de OpenCV.

El efecto de la transformación hace que la deformación debido a la perspectiva del cuadrado exterior del sudoku se revierta, al seleccionar puntos que forman un cuadrado de 300 píxeles de lado.

Se debe tener cuidado con la selección de las nuevas coordenadas, dado que provocan expansión o achicamiento de las imágenes.

3.9. Función para el cálculo de distancia

3.9.1. Función de correspondencia

El prototipo N°1 no cuenta haces perfectamente paralelos. Como alternativa se realizan capturas de los vértices de un triángulo dibujado de 11cm de arista en una superficie plana. Se toman muestras a diferentes distancias de la cámara. De las imágenes obtenemos la medida en píxeles de la arista según la distancia. Con la cámara configurada para tomar imágenes de 1Mpx, resulta la siguiente tabla:

Arista [Pixel]	Distancia [Metros]
51	3,03
55	2,83
59	2,63
64	2,43
70	2,23
77	2,03
85	1,83
95	1,63
109	1,43
127	1,23
151	1,03
188	0,83
248	0,63
361	0,43
648	0,23

Tabla 3.1. Correspondencia entre arista en pixeles y distancia en metros.

La variación en distancia es constante, no así la de arista. Para apreciar mejor el comportamiento se presenta la gráfica asociada a la tabla (arista [pixeles] en abscisas y distancia [metros] en ordenadas):

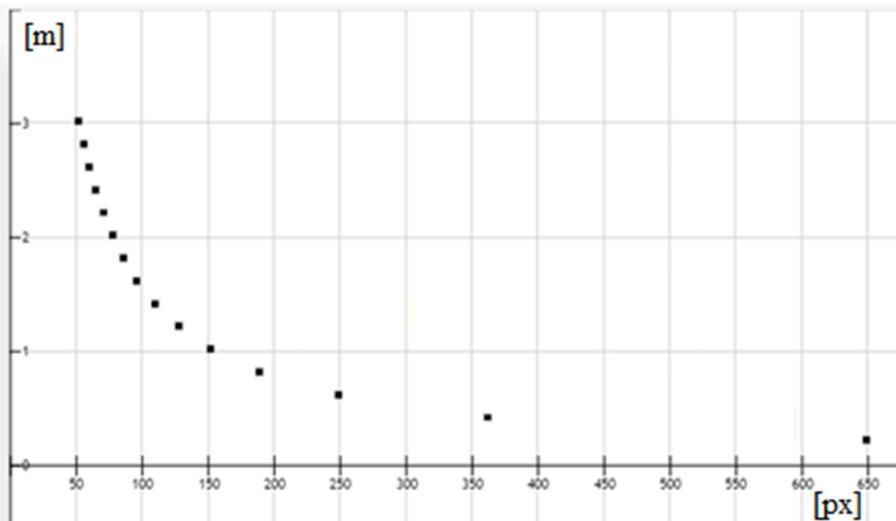


Figura 3.72. Gráfica de la tabla 2.1.

De la gráfica podemos sacar algunas conclusiones:

- 1) A menor distancia más grande es la variación de pixeles, lo que nos dice que el dispositivo podrá detectar con mayor sensibilidad la distancia. Dicha zona de medición es de 0 a aproximadamente 1,5 metros.
- 2) A medida que la distancia aumenta, a ritmo constante, el valor de arista disminuye rápidamente para luego volverse asintótico al eje de ordenadas, por lo que en algún momento a distintos valores de distancia les corresponderá el mismo valor de arista (por ser arista un valor entero positivo), es decir se pierde sensibilidad.
- 3) Al querer aproximar mediante una función lineal, para cubrir la mayor cantidad de puntos tenemos:

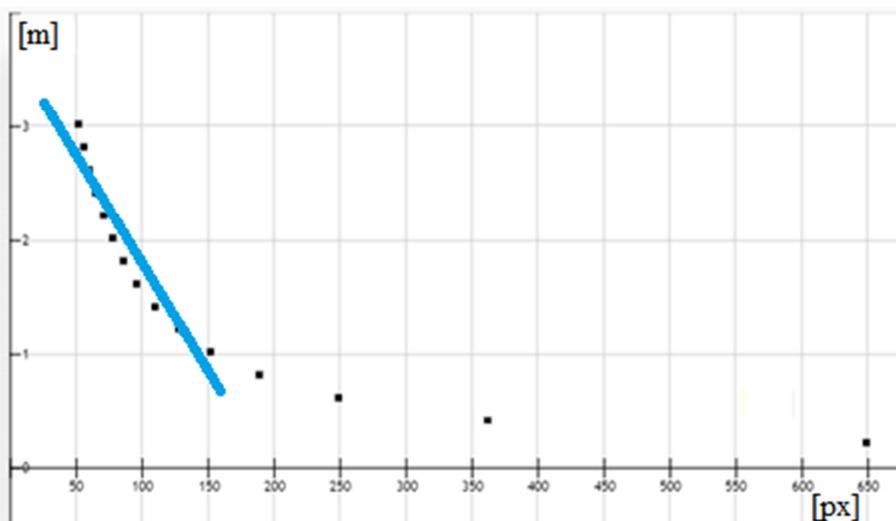


Figura 3.73. Aproximación lineal para los puntos de la tabla 2.1.

O bien:

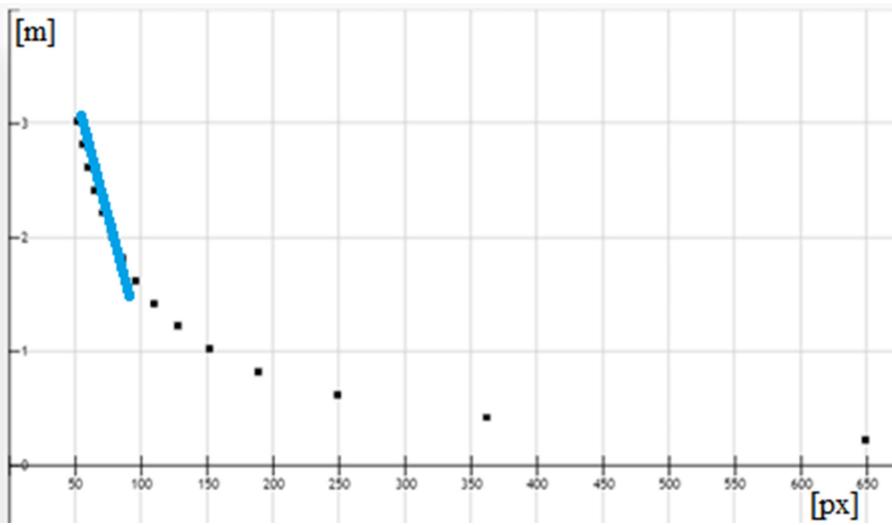


Figura 3.74. Aproximación lineal para los puntos de la tabla 2.1. n°2.

En ambos casos la aproximación es mala, porque solo es válida en un intervalo. Además de introducir error a las mediciones al no ser perfecta. Si la aproximación se hace en la zona de alta sensibilidad, es decir para $150 \leq \text{arista} \leq 650$, la distancia máxima es muy baja, e introduce error como las demás. Como conclusión lo ideal sería encontrar una nueva función que permita hacer la conversión de arista a distancia y viceversa.

3.9.2. Nueva función de correspondencia

Al observar la forma que describen los puntos podemos decir que tiende a ser parecida a la función $f(x) = 1/x$. Variando la constante superior y corroborando con la herramienta que provee Excel, que permite determinar funciones partiendo de pares (x, y) , ingresando los valores de la tabla Arista vs Distancia obtenemos la siguiente función:

$$y = f(x) = \frac{156,25}{x}$$

Tabla comparativa:

Arista [Pixelles] (x)	Distancia [Metros]	Aproximación (f(x))	Error [Metros]
51	3,03	3,06	0,03
55	2,83	2,84	0,01
59	2,63	2,65	0,02
64	2,43	2,44	0,01
70	2,23	2,23	0,00
77	2,03	2,03	0,00
85	1,83	1,84	0,01
95	1,63	1,64	0,01

109	1,43	1,43	0,00
127	1,23	1,23	0,00
151	1,03	1,03	0,00
188	0,83	0,83	0,00
248	0,63	0,63	0,00
361	0,43	0,43	0,00
648	0,23	0,24	0,01

Tabla 3.2. Comparación entre los valores experimentales de la tabla 3.1 con los generados mediante la nueva función de correspondencia.

La aproximación con esta función muy buena, dado que si produce error, este no excede 3cm. Solo para dos casos sobrepasa el error de 1cm.

Las capturas para las medidas de aristas fueron realizadas de 23cm hasta 3,03 metros, pero de seguir aumentando la distancia seguirán estando sobre la curva.

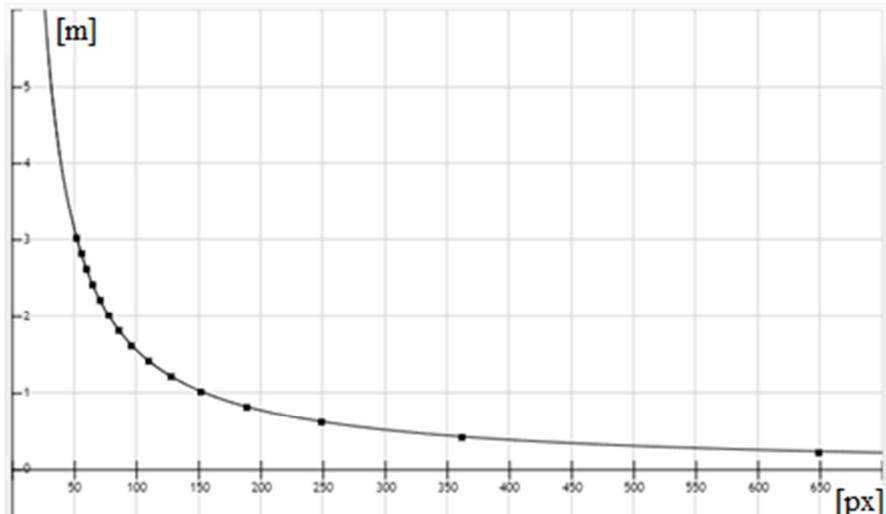


Figura 3.75. Gráfica de la nueva función de correspondencia.

La gráfica más extensa de la función pone más en evidencia la pérdida de sensibilidad al aumentar la distancia.

3.9.3. 3Mpx

Hasta ahora todo lo descripto es resultado de pruebas realizadas con imágenes de 1Mpx. Si construimos una tabla a partir de imágenes capturadas con 3Mpx de resolución, es decir 2048x1536, obtenemos la siguiente tabla:

Arista [Pixel]	Distancia [Metros]	Aproximación [Metros]	Error [Metros]
64	3,83	3,91	0,08
69	3,63	3,62	0,01
73	3,43	3,42	0,01
78	3,23	3,21	0,02

84	3,03	2,98	0,05
88	2,83	2,84	0,01
96	2,63	2,60	0,03
103	2,43	2,43	0,00
113	2,23	2,21	0,02
123	2,03	2,03	0,00
137	1,83	1,82	0,01
153	1,63	1,63	0,00
175	1,43	1,43	0,00
202	1,23	1,24	0,01
242	1,03	1,03	0,00
300	0,83	0,83	0,00
394	0,63	0,63	0,00
577	0,43	0,43	0,00
1071	0,23	0,23	0,00

Tabla 3.3. Comparación de valores experimentales con los generados mediante la función de correspondencia para una resolución de 3Mpx.

Responde a una función de aproximación similar:

$$y = f(x) = \frac{250}{x}$$

Y es la que se aleja más de los ejes en la siguiente gráfica mientras que la otra es la anterior (para 1Mpx).

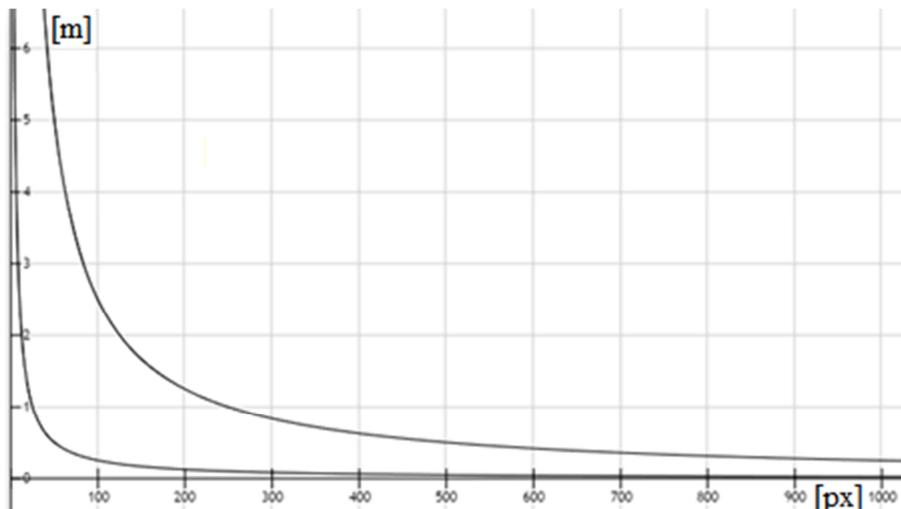


Figura 3.76. Gráfica de la función de correspondencia para 1Mpx y 3Mpx.

Para 3Mpx de resolución la curva se hace asintótica para una distancia mayor, por lo que la zona de sensibilidad se expande. Además para 3,83m el valor de arista es 64 que es mayor que 51 (valor asociado a 3,03m para 1Mpx), que significa que tiene más sensibilidad aunque la distancia sea mayor.

3.10. Iteración VIII: Pruebas de sensibilidad

3.10.1. Prototipo N°2

Es tanta la inestabilidad de los punteros que es necesario un nuevo montaje, uno que haga que los haces mantengan su dirección el tiempo suficiente para tomar las imágenes de muestra.

Nuevo montaje: los punteros y la cámara se encuentran montados entre dos placas de fibrofacil:



Figura 3.77. Molde triangular con los punteros montados.

Al estar montados los punteros, la distancia que existe desde el centro de un puntero al de cualquiera de los otros dos es de 11cm.

Si bien los haces de luz de los punteros no son perfectamente paralelos para distancias relativamente pequeñas podemos realizar pruebas de sensibilidad.

3.10.2. Pruebas

Con las siguientes pruebas se busca poner en evidencia la existencia de la zona de sensibilidad, variando la distancia al plano de proyección y la inclinación del mismo. En las tablas se encuentra: los ángulos de inclinación y las coordenadas de los puntos del triángulo detectados, junto con los desplazamientos en x y en y de los puntos debido a la inclinación.

3.10.2.1. Prueba N°1

Distancia al plano: 65,5cm.

	Inicial	Variación 1	Variación 2	Variación 3
Ángulo vertical	0°	0°	0°	0°
Ángulo horizontal	0°	10°	40°	60°
Coordenadas del punto inferior izquierdo	(557, 553)	(556, 553)	(552, 556)	(543, 561)
Coordenadas del punto superior	(662, 362)	(663, 363)	(663, 364)	(662, 365)
Coordenadas del punto inferior	(804, 554)	(802, 554)	(794, 548)	(784, 542)

derecho				
Desplazamiento del punto inferior izquierdo en <i>x</i>	-	1px a la izquierda	5px a la izquierda	14px a la izquierda
Desplazamiento del punto inferior izquierdo en <i>y</i>	-	-	3px hacia abajo	8px hacia abajo
Desplazamiento del punto superior en <i>x</i>	-	1px a la derecha	1px a la derecha	-
Desplazamiento del punto superior en <i>y</i>	-	1px hacia abajo	2px hacia abajo	3px hacia abajo
Desplazamiento del punto inferior derecho en <i>x</i>	-	2px a la izquierda	10px a la izquierda	20px a la izquierda
Desplazamiento del punto inferior derecho en <i>y</i>	-	1px hacia arriba	6px hacia arriba	12px hacia arriba

Tabla 3.4. Datos de la prueba de sensibilidad N°1.

3.10.2.2. Prueba N°2

Distancia al plano: 1,40m.

	Inicial	Variación 1	Variación 2
Ángulo vertical	0°	0°	0°
Ángulo horizontal	0°	40°	60°
Coordenadas del punto inferior izquierdo	(620, 516)	(619, 517)	(618, 518)
Coordenadas del punto superior	(662, 434)	(662, 434)	(662, 433)
Coordenadas del punto inferior derecho	(742, 518)	(739, 516)	(736, 514)
Desplazamiento del punto inferior izquierdo en <i>x</i>	-	1px a la izquierda	2px a la izquierda
Desplazamiento del punto inferior izquierdo en <i>y</i>	-	1px hacia abajo	2px hacia abajo
Desplazamiento del punto superior en <i>x</i>	-	-	-
Desplazamiento del punto superior en <i>y</i>	-	-	1px hacia arriba
Desplazamiento del punto inferior	-	3px a la izquierda	6px a la izquierda

derecho en x			
Desplazamiento del punto inferior derecho en y	-	2px hacia arriba	4px hacia arriba

Tabla 3.5. Datos de la prueba de sensibilidad N°2.



Figura 3.78. Imágenes del equipo montado y funcionando para las pruebas de sensibilidad.

3.10.3. Análisis de las pruebas de sensibilidad

En la prueba N°2 para 1,40m, con 40° de inclinación, los desplazamientos de los puntos son similares a los que suceden en la prueba N°1 para 65,5cm con 10° de inclinación, demostrando la pérdida de sensibilidad, ya que cada pixel desplazado representa cada vez una variación mayor de ángulo a medida que aumenta la distancia.

Con la información obtenida de las pruebas, es posible la confección de imágenes para el *testing* del software para el dispositivo. A continuación se enumeran los pasos a seguir:

- 1) Se mide la distancia entre el dispositivo y el plano.
- 2) Se toma una imagen de la proyección de los puntos láser en plano sin inclinación.
- 3) Se altera la inclinación del plano sin que cambie la distancia al mismo. Y se realizan dos capturas: una con las proyecciones de los puntos láser, y otra sin ellas.
- 4) Se analiza el desplazamiento que sufren las proyecciones debido a la inclinación.
- 5) Se calculan las coordenadas que tendrían, en la imagen, los puntos láser formando un triángulo equilátero para la distancia medida en el punto 1.
- 6) Se alteran las coordenadas según el desplazamiento registrado en el punto 4.
- 7) En la imagen sin proyecciones obtenida en el punto 3, son agregados círculos de color blanco en las coordenadas alteradas (punto 6), emulando las proyecciones.

3.11. Iteración IX: Software del dispositivo

El software posee una interfaz de usuario que permite ajustar los parámetros para procesar las imágenes capturadas, y realizar las mediciones de longitud, área y perímetro. También, con los resultados que provee, permite la calibración y análisis de sensibilidad del equipo una vez construido.

Confeccionada utilizando PyQt4 (versión 4.10.4), que provee una herramienta de diseño facilitando la ubicación de los elementos y establecer las interacciones entre los mismos. Luego del armado en la herramienta, mediante la línea de comandos se debe acceder a la carpeta donde fue guardado el archivo .ui, e ingresar el siguiente comando (suponiendo que fue guardado con el nombre “interfaz”):

```
pyuic4 interfaz.ui -o interfaz.py
```

De este modo se genera el archivo .py en el mismo directorio, cuyo contenido será la declaración de los elementos, la colocación en sus respectivas ubicaciones (layout), y la conexión de señales con las respuestas a las mismas. Claro que debe agregarse todo lo que el diseñador crea necesario como incluir módulos, más señales y por supuesto la lógica del proceso.

3.11.1. Pantalla inicial RAFI (Range Finder / Telémetro)

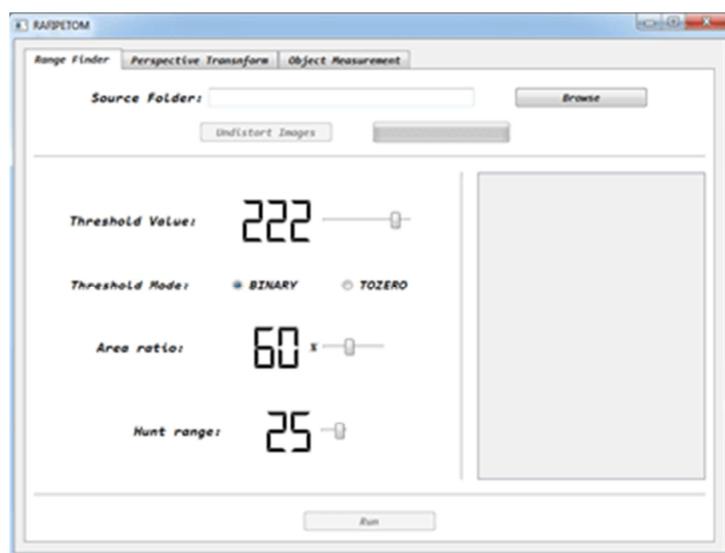


Figura 3.79. Pantalla inicial de la interfaz de usuario.

Al iniciar solo el botón “Browse” se encuentra habilitado, el cual permite seleccionar la carpeta que contenga las imágenes a ser procesadas. Una vez se selecciona la carpeta verifica que todos los archivos en la carpeta sean imágenes de extensión jpeg.

Si la carpeta contiene todas imágenes .jpg se habilitan los botones “Undistort Images” y “Run”:

- Undistort Images: Si se usa por primera vez cada vez que se use el programa, carga los parámetros para la rectificación de imágenes obtenidos de la calibración de la cámara. Luego rectifica las imágenes, lo que las modifica de manera permanente.
Si en la carpeta las imágenes no poseen la misma resolución con la que se realizó la calibración de la cámara, informa con el mensaje de error: “Resolution of the images must match with the calibration resolution”. Deben coincidir dado que las imágenes deben ser de la misma cámara que fue calibrada.
- Run: No es necesario rectificar las imágenes previamente para accionar este botón. “Run” realiza la búsqueda de los puntos láser en las imágenes, y el análisis del triángulo para determinar la distancia a la que se encuentra el plano de proyección, y los ángulos de inclinación del mismo haciendo uso de los cuatro parámetros ajustables:
 - o Threshold Value (Valor umbral): Establece el valor umbral. Para la el thresholding, 222 es el valor recomendado para realizar la primera ejecución, ya que generalmente los valores de intensidad de los puntos láser se encuentran por encima de este valor.
 - o Thresholding Mode (Modo/Estilo de umbralización): Si es seleccionado “BINARY”, el ajuste para encontrar el centro del punto se realiza de acuerdo a la forma, de lo contrario, seleccionado “TOZERO”, se realiza buscando la mayor intensidad.
 - o Area Ratio (Relación de área): Establece el valor que debe superar el cociente entre área de la figura y área del círculo envolvente, de superarlo es candidata a ser la proyección de un puntero láser. El valor recomendado es 60%, entonces el cociente calculado a partir de las figuras debe superar 0,6.
 - o Hunt Range (Rango de caza): Establece la tolerancia del método Trinity hunter rev. Si el paralelismo de los punteros láser es excelente, sin problemas podría ser puesto a uno. Pero al brindar tolerancia da la posibilidad de calibrar el paralelismo y hacer pruebas de las demás etapas sin contar con un dispositivo perfecto.

Luego presenta los resultados en el área que se encuentra a la derecha. Los datos que presentan son:

- o Medidas de las aristas.
- o Medidas de las tres rectas que forma el centro con los vértices.
- o El centro de la imagen.
- o La distancia al plano de proyección.
- o El ángulo vertical y el ángulo horizontal.

3.11.1. Selección de una carpeta (“Browse”)

¿Por qué se selecciona una carpeta? ¿No basta con solo una imagen?

Brinda la posibilidad de trabajar con solo una o más imágenes de entrada. El uso de todas las imágenes de la carpeta se da sólo en la etapa de thresholding. Los pasos son los siguientes:

- Crea una imagen de igual resolución a las imágenes dentro de la carpeta con todos sus pixeles en negro.
- Toma una imagen de la carpeta seleccionada, realiza el thresholding. Luego la operación lógica OR de esta imagen con la imagen en negro creada en el primer paso. Es la misma operación que se realizó en la “suma” de las máscaras utilizadas en la sección de HSV.
- Repite el segundo paso con las demás imágenes en la carpeta.

Para verlo con más claridad, supongamos que la una de las imágenes dentro de la carpeta luego de aplicar thresholding es la siguiente:



Figura 3.80. Figura candidata a ser la proyección de un punto láser, imagen n°1.

La de una segunda imagen:



Figura 3.81. Figura candidata a ser la proyección de un punto láser, imagen n°2.

Al aplicar la operación OR se obtiene algo así:



Figura 3.82. Figura candidata resultante de la superposición.

Para los punteros láser es beneficioso, ya que en ocasiones la cámara no suele captarlos de igual manera de una imagen a otra. De este modo las figuras tendrán una forma más circular, posibilitando el uso de una relación de área más alta. Muchas cámaras hoy en día cuentan con la opción de tomar ráfagas de imágenes.

Al seleccionar una carpeta pueden producirse errores, los cuales se dan conocer con una ventana con el mensaje, y son los siguientes:

- “Insert folder path”: Cuando en lugar de seleccionar una carpeta se cancela en la ventana de búsqueda haciendo click en el botón cancelar.
- “Empty folder”: La carpeta seleccionada está vacía.

- “At least one file is not a .jpg image”: Indica que en la carpeta existe al menos un archivo que no es una imagen de extensión .jpg.

3.11.1.2. Ajuste de parámetros, sensibilidad

Los más sensible, y cuyo ajuste significa encontrar los puntos láser o no encontrarlos, son el valor umbral y la relación de área.

Es alta la probabilidad que los puntos láser sean lo de más alta intensidad de luz en la imagen, entonces el valor umbral debe fijarse lo más alto posible, por lo que se recomienda probar desde los valores más altos y de obtener resultados negativos ir bajándolo. De esta manera también resulta menos cantidad de figuras candidatas y acelera la convergencia de la búsqueda. Lo mismo para el valor que establece la relación de área.

3.11.1.3. Ejecución y presentación de resultados

Basta con hacer click en el botón “Run” para que se procesen las imágenes dentro de la carpeta seleccionada, y en el área a la derecha de los parámetros se vean los resultados. Además en una ventana adicional se grafica el plano detectado en 3D.

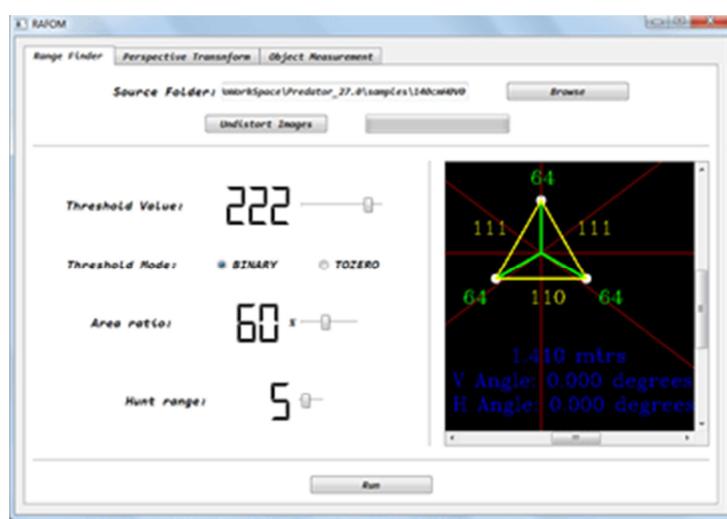


Figura 3.83. Vista de la interfaz de usuario una vez ejecutada la detección del triángulo láser.

Son detectados los tres puntos láser formando el triángulo, y en imagen se indican los valores en pixeles de las aristas en amarillo y de los radios en verde. Las cuatro líneas de color rojo marcan el centro de la imagen con su intersección. En azul, primero tenemos la distancia a la que se encuentra el plano, luego el ángulo vertical y por último el ángulo horizontal.

La ventana adicional que aparece es la siguiente:

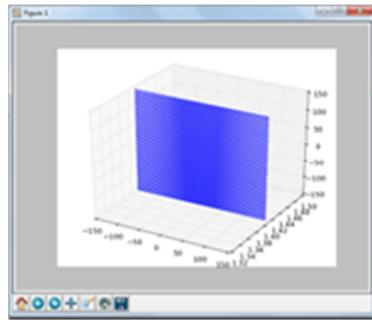


Figura 3.84. Presentación del plano detectado en gráfica 3D.

Plano resultante con ángulos vertical y horizontal iguales a cero.

Al ejecutar no siempre se llega a ver la información resultante, dado que puede que ocurra algún error durante el proceso. A continuación son enumerados los mensajes de error y los motivos:

- “Different resolutions @ Source folder”: El mensaje informa que en la carpeta seleccionada existen imágenes de distintas resoluciones; deben ser todas de la misma (puede que no se haya realizado el rectificado previo, entonces ya que en la etapa de thresholding se utilizan todas las imágenes disponibles también puede realizarse el control).
- “Can’t find any dot”: El mensaje informa que no ha sido posible encontrar ni una sola figura que pueda ser la proyección de un láser. Puede suceder cuando efectivamente no existe ninguna en las imágenes, o los dos parámetros más sensibles poseen un valor demasiado exigente. Como ser el utilizar una relación de área de 100%, que ninguna figura por más de sobrevivir al thresholding pueda cumplir, a menos que se trate de un láser de calidad y condiciones de luz óptimas.
- “Triangle not found”: Son detectadas figuras candidatas, pero no están formando un triángulo. Para este caso se recomienda variar los parámetros para hacerlos más permisivos.

3.11.2. PET (Perspective Transform / Transformación de Perspectiva)

Antes de la detección del triángulo que realiza el RAFI, la pestaña se encuentra deshabilitada. Sea el siguiente el triángulo detectado:



Figura 3.85. Resultados de distancia, inclinación y dimensiones del triángulo detectado.

Entonces se encuentra habilitada la transformación de perspectiva:

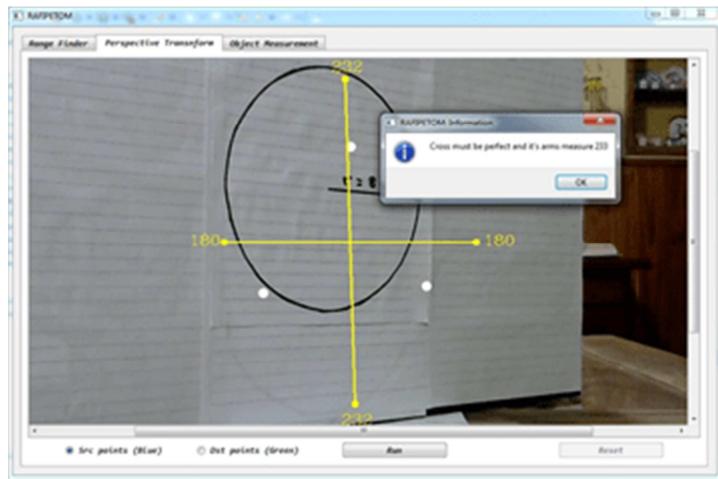


Figura 3.86. Vista de la pestaña de transformación de perspectiva.

La ventana emergente indica medida de longitud que deben tener las líneas en amarillo, que van desde el centro a los puntos amarillos. Esto fue ideado para brindar al usuario información, para que pueda saber cuándo terminó de realizar la transformación de perspectiva. Como vemos los valores de longitud se encuentran próximos a los puntos, y este caso la longitudes deben ser iguales a 233 pixeles.

Las líneas verticales son de 232 pixeles dado que la inclinación vertical es de $2,89^\circ$. El usuario deberá manipular la perspectiva de la imagen hasta lograr, o lo más cercano posible, que las cuatro líneas midan 233 pixeles y formen cuatro ángulos rectos.

Para cambiar la perspectiva debe marcar cuatro puntos, y luego cuatro más que representan a donde los primeros cuatro serán desplazados. En la parte inferior izquierda tenemos dos botones “Src points (Blue)” y “Dst points (Green)”: al estar seleccionado Src points, que serán los puntos de origen u originales, al hacer click en algún lugar de la imagen se dibujará un punto azul en ese mismo lugar. Si es Dst points el que está seleccionado, los puntos que se dibujan en la imagen son de color verde, y serán donde se ubicaran los originales luego de ser desplazados. Se admiten cuatro puntos azules y cuatro verdes, que son los ocho puntos necesarios para la transformación de perspectiva. Los puntos se pueden borrar haciendo click sobre ellos (para borrar los azules, Src points debe estar seleccionado).

Es de vital importancia que todo aquello que se quiera medir, y los puntos extremos de la cruz, estén contenidos en el cuadrilátero que definen los puntos azules.

Error posible en la etapa de marcado: Al tener marcados cuatro azules y se intenta marcar otro... una ventana informa que cuatro son el máximo, lo mismo se aplica a los puntos verdes.

Luego de marcar los ocho puntos se puede ejecutar la transformación accionando el botón “Run”:

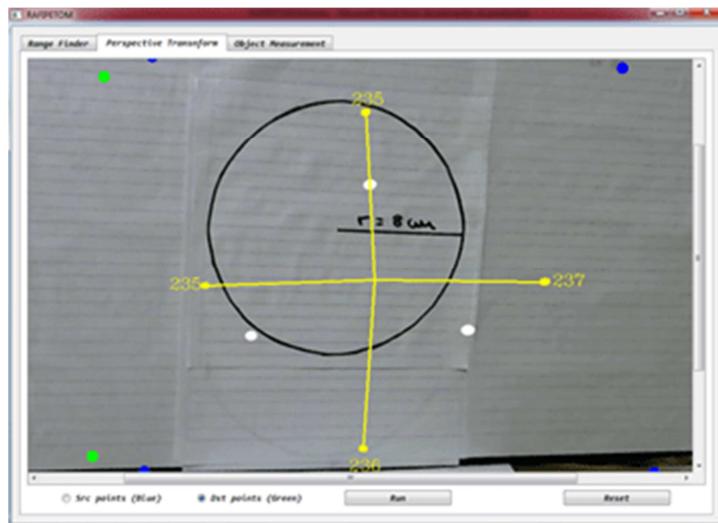


Figura 3.87. Resultados de aplicar la transformación de perspectiva.

Error posible al ejecutar: Si se ejecuta sin haber marcado los ocho puntos, se informa al usuario de qué color se requiere que sean marcados más puntos.

No es necesario marcar todos los puntos nuevamente si no se obtienen las rectas de longitud 233, se puede borrar y marcar nuevamente los puntos (uno, dos, tres o todos) que están introduciendo error a la transformación y ejecutar otra vez.

El botón “Reset” hace que la imagen retorne a la perspectiva original y borra todos los puntos.

La longitud de las líneas iniciales de la cruz está dada por la inclinación del plano detectado. Son calculadas de la siguiente manera:

Primero tomamos cuatro los vectores:

$$\overrightarrow{v_1} = (1,0,0)$$

$$\overrightarrow{v_2} = (0,1,0)$$

$$\overrightarrow{v_3} = (-1,0,0)$$

$$\vec{v_4} = (0, -1, 0)$$

Para lograr que los vectores reflejen la inclinación debemos aplicarles dos rotaciones, una con el ángulo horizontal, y otra con el ángulo vertical. Necesitamos elegir un vector \vec{u} como eje de rotación, que debe cumplir la siguiente condición:

$$u_x^2 + u_y^2 + u_z^2 = 1,$$

Para ser utilizado en la matriz de rotación R :

$$R = \begin{bmatrix} \cos\theta + u_x^2(1 - \cos\theta) & u_xu_y(1 - \cos\theta) - u_z\sin\theta & u_xu_z(1 - \cos\theta) + u_y\sin\theta \\ u_yu_x(1 - \cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1 - \cos\theta) & u_yu_z(1 - \cos\theta) - u_x\sin\theta \\ u_zu_x(1 - \cos\theta) - u_y\sin\theta & u_zu_y(1 - \cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1 - \cos\theta) \end{bmatrix}$$

El signo del ángulo determina el sentido de rotación, dado que no se le ha asignado a los dos ángulos detectados un signo, obtenemos la información a partir del vector normal al plano por medio de la primera coordenada para el ángulo horizontal, y para el vertical la segunda.

Primero se realiza la rotación utilizando el ángulo horizontal, por lo tanto el vector elegido para ser el eje de rotación es el vector:

$$\vec{u_h} = (0, 1, 0)$$

Se rotan los cuatro vectores multiplicando cada uno por la matriz. Y para la rotación vertical utilizamos como nuevo eje el vector $\vec{v_1}$ ya afectado por la primera rotación.

Por último las líneas amarillas que se dibujan en la imagen, son el resultado de multiplicar cada vector por la medida de arista del triángulo equilátero asociada a la distancia detectada, proyectados en el plano xy . El motivo de la multiplicación por la medida de arista es que las líneas sean de menor longitud a medida que aumenta la distancia, para darles una longitud relacionada con la distancia.

3.11.3. OM (*Object Measurement / Medición de Objetos*)

Los elementos de la pestaña al igual que los de PET se habilitan una vez detectado y examinado el triángulo. Cuando RAFI termina su labor, es cargada la imagen original en la pestaña OM. Se pueden realizar mediciones, pero solo serán válidos si el plano detectado no presenta inclinación alguna. Si se detecta inclinación, al realizar una transformación de perspectiva la imagen resultante es cargada en OM, donde de haber sido anulada, las mediciones son válidas. Como se ve en la siguiente imagen:

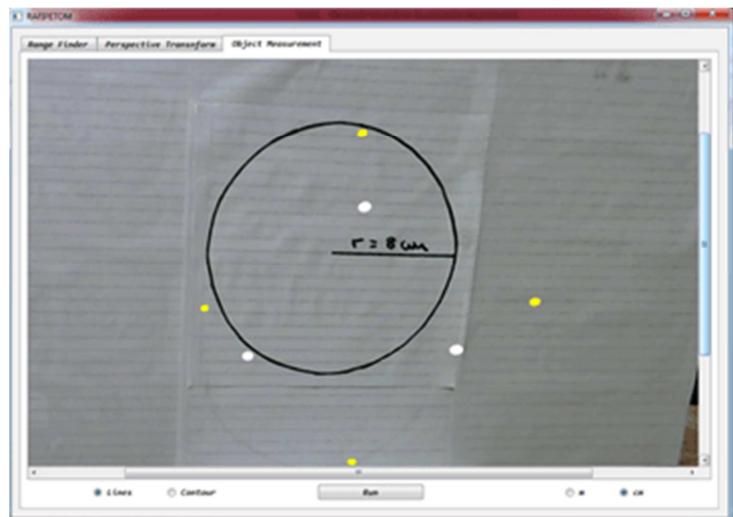


Figura 3.88. Vista de la pestaña de medición de objetos.

En la zona inferior izquierda nos brinda la opción si medir líneas rectas (Lines) o un contorno cerrado (Contour). Debemos marcar puntos, y al igual que los de PET, se borran al volver a clickear sobre ellos, o bien pueden borrarse todos al cambiar de opción.

En la zona inferior derecha podemos elegir la unidad en la que serán expresadas las mediciones.

3.11.3.1. Líneas

Un primer click en la imagen marca un punto, un segundo click en otro lugar marca otro punto, y se dibuja una línea que los une. En la siguiente imagen, el objetivo es medir el radio de la circunferencia dibujada en la superficie:

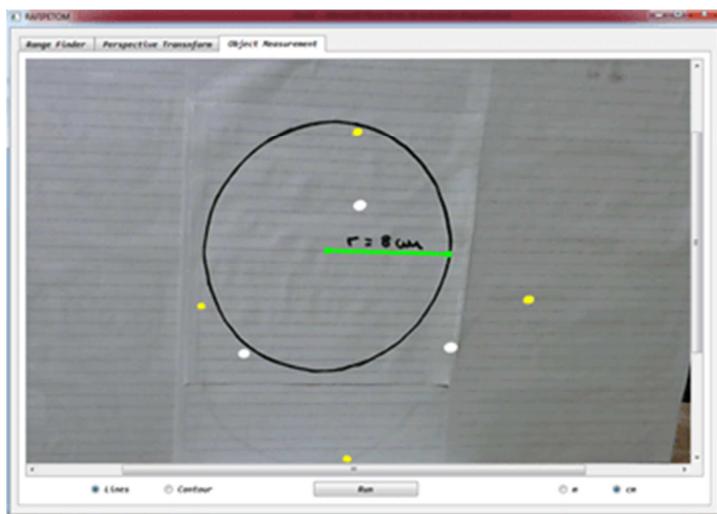


Figura 3.89. Mediciones rectilíneas en la pestaña de medición.

Haciendo click en el botón “Run” calcula las medidas:

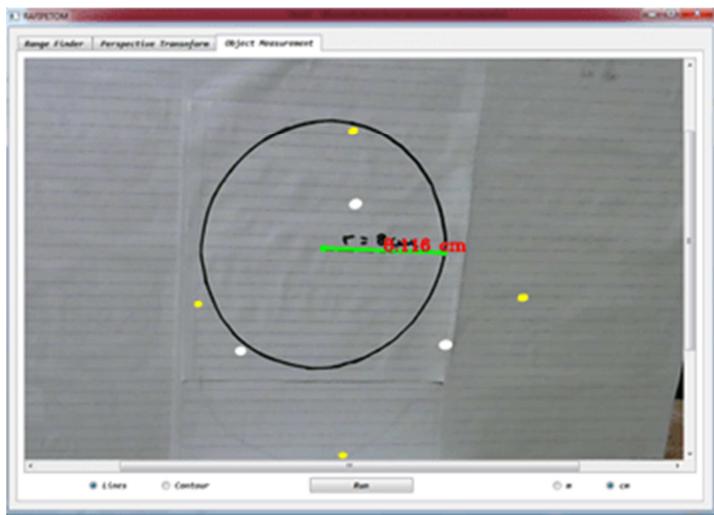


Figura 3.90. Resultados de mediciones rectilíneas en la pestaña de medición.

El radio de la circunferencia dibujada es de 8cm. En la medición tenemos un error de 0,116cm

Nota 11.3.1.1: Cambiando la unidad de medición se actualizan en la imagen.

Al hacer click sobre punto ya marcado, este se borra, y también el que forma la línea con él.

Error posible:

Si se ejecuta la medición con un número impar de puntos marcados, anula la ejecución, e informa que debe ser par la cantidad de puntos para poder definir las rectas.

3.11.3.2. Contornos

Deben marcarse puntos en la imagen para formar un contorno. Sea $[p_0, p_1, \dots, p_i, p_{i+1}, \dots, p_n]$ con $0 \leq i \leq n$, la lista de puntos marcados en la imagen, el punto p_i con el punto p_{i+1} forman una recta, p_{i+1} con p_{i+2} , y p_n con p_0 cerrando el contorno.

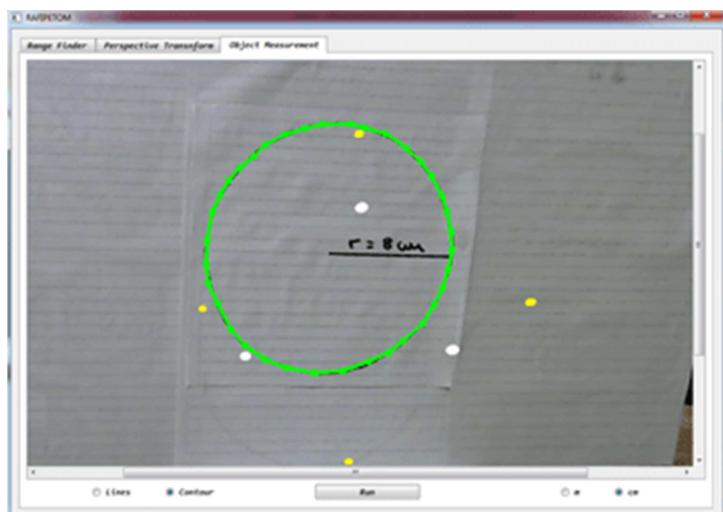


Figura 3.91. Mediciones de contornos cerrados en la pestaña de medición.

La circunferencia dibujada en el plano de proyección tiene:

- Área: 201cm²
- Perímetro: 50,24cm

Habiendo ejecutado:

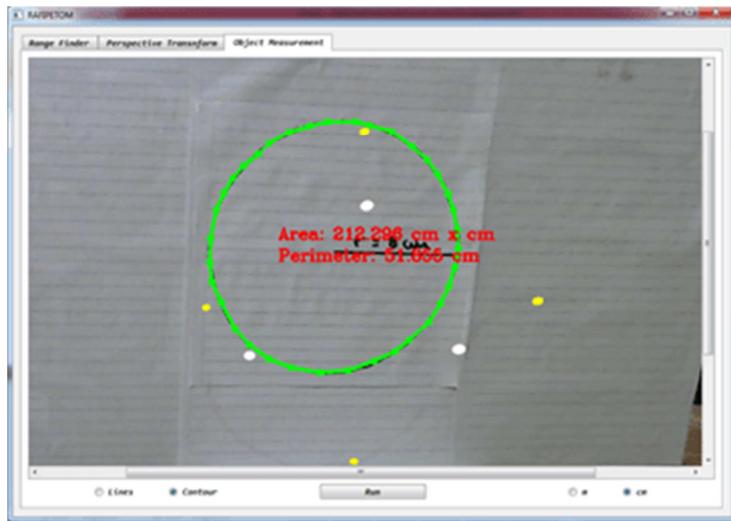


Figura 3.92. Resultados de mediciones de contornos cerrados en la pestaña de medición.

Para el cálculo del área, el radio con error al cuadrado, y un marcado no muy prolífico de los puntos, hacen que el error total sea de 11.3cm². En cuanto al perímetro, cerca de 2cm de error.

Error Posible:

El marcar menos de 3 puntos se considera un error, que anula la medición de área y perímetro del contorno, ya que no es apreciable el área del contorno.

Capítulo 4: Diagrama completo del proceso

4.1. Introducción

El software del prototipo es un conjunto de funciones. Solo existen dos clases, que no son solo funciones debido a que además de los parámetros que reciben para realizar su tarea, requieren de otros que se pasan por el constructor. Por lo tanto la confección de diagramas de clases no proporcionaría información. También es el caso de los diagramas de secuencia, ya que no existe interacción o un ida y vuelta entre objetos.

Son procesos donde hay claramente una entrada y una salida, las imágenes de entrada son procesadas para obtener información. Para verlo con más claridad, si bien las operaciones descriptas ya se encuentran en orden, a continuación tenemos cuatro diagramas en forma de pipeline que esquematizan las operaciones que se realizan en el proceso, sumando explicaciones y detalles no mencionados aún.

4.2. Pipeline de la calibración de la cámara:

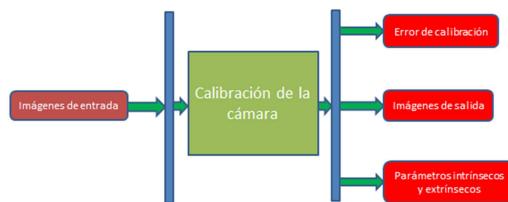


Figura 4.1. Pipeline de la calibración de la cámara.

Las imágenes de entrada son capturas de un tablero de ajedrez a diferentes distancias y posiciones.

La etapa intermedia “Calibración de cámara” realiza los siguientes subprocesos o tareas^[21]:

- 1) Antes de ser procesada cada imagen, es convertida a escala de grises.
- 2) Son detectadas las esquinas de los casilleros, registrando las coordenadas en pixeles como puntos en 2D; calcula y almacena los puntos que ocupan en el mundo real (3D).
- 3) Haciendo uso de los puntos en 2D y 3D calcula los parámetros de la cámara.
- 4) Con los parámetros realiza la proyección de los puntos en 3D sobre el plano de la imagen, compara con los puntos en 2D para el cálculo del error.

Como resultado del pipe de calibración obtenemos:

- El error de calibración, que si se encuentra entre 0,1 y 1,0 se considera una buena calibración.
- Imágenes, que son las mismas que ingresan al pipe solo que los puntos (las esquinas de los casilleros) están marcados con colores.
- Los parámetros extrínsecos e intrínsecos de la cámara.

4.3. Pipeline asociado a RAFI (Range Finder)

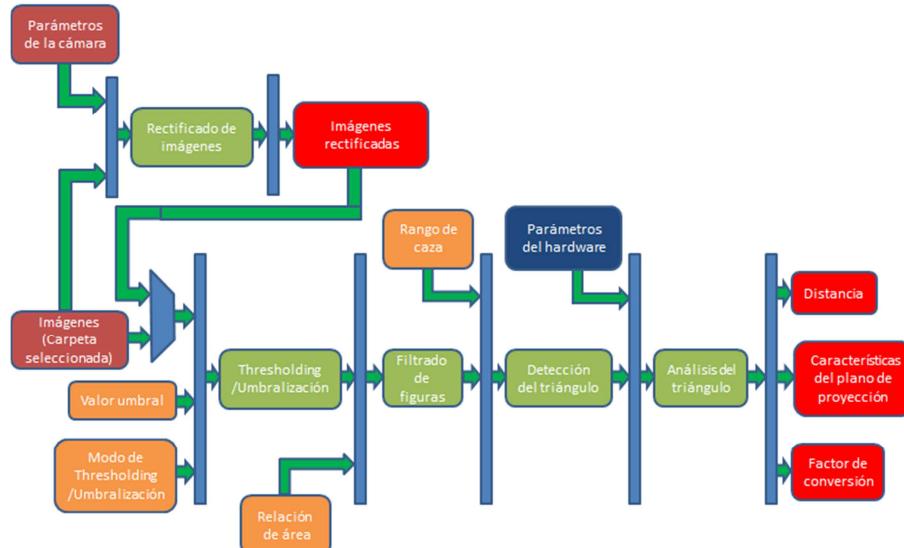


Figura 4.2. Pipeline de la detección del triángulo, cálculo de distancia, inclinación y factor de conversión.

4.3.1. Rectificado

Las imágenes que ingresan al pipe, pueden hacerlo de manera directa o pasar antes por un proceso de rectificado.

El proceso de rectificado hace uso de los parámetros obtenidos a partir de la calibración de la cámara. Genera una matriz que permite eliminar la distorsión provocada por la lente. El proceso no elimina la distorsión en toda la imagen; las regiones cercanas a los límites deben ser descartados. Por suerte la misma función que genera la matriz mencionada, también genera las coordenadas que posibilitan recortar la imagen y quedarnos con la parte válida. En el caso de RAFI la zona valida es marcada con un rectángulo rojo.

4.3.2. Umbralización

Junto con las imágenes rectificadas o no, para el proceso de umbralización son necesarios dos parámetros que se definen en la interfaz de usuario (Valor umbral y Modo de umbralización). Luego suceden los siguientes subprocesos:

- 1) Las imágenes son cargadas en escala de grises.

- 2) Se les aplica un suavizado, “medianBlur” con un tamaño de kernel o ventana igual a nueve, con el fin de redondear más los límites de las figuras.
- 3) Todas las imágenes son “sumadas” con la operación OR.

El modo de umbralización: BINARY o TOZERO, su explicación quedará pendiente hasta la etapa análisis del triángulo. Por lo pronto hasta este punto podemos decir que ambas son buenas para obtener las figuras de alta intensidad. Pero TOZERO permite con una ejecución saber si es posible aumentar el valor umbral, ya que podemos apreciar en la imagen los distintos valores de intensidad superiores al umbral. Con BINARY vemos todas las figuras que superan el valor umbral, pero la intensidad de las figuras llevada al máximo.

4.3.3. Filtrado de figuras

La etapa “Filtrado de figuras” requiere el parámetro relación de área para filtrar, de una lista de figuras detectadas aquellas que posean una forma más circular.

Filtrado:

Tenemos la lista figuras, precisamente de contornos, una segunda lista con la jerarquía de cada contorno, y una tercera donde se encuentran los círculos que contienen a las figuras expresados como $((x, y), radio)$. Lo importante es el par (x, y) ya que es donde suponemos ubicada la figura. Las figuras son filtradas según si la relación de área, entre el área del contorno de la figura y el área del círculo que la contiene, supera la configurada en la interfaz, sumado a que los dos últimos valores del arreglo de jerarquía sean -1. Para aquellas figuras o contornos que cumplan la condición, serán guardados los círculos contenidos en una nueva lista.

De la imagen, las figuras que no satisfacen la condición son eliminadas. Dibujando en una imagen completamente en negro los círculos contenidos de aquellas que satisfacen la condición, y luego se aplica la operación lógica AND entre esta imagen creada (una máscara), y aquella donde están todas las figuras (imagen resultante del subproceso umbralización).

4.3.4. Detección del triángulo

Solo hace uso del algoritmo “Trinity Hunter Rev”. Empleando el parámetro de la interfaz “Rango de caza”/“Hunt range” como tolerancia para filtrar, de un listado de posiciones, las figuras que están formando un triángulo.

4.3.5. Refinamiento de la posición de las figuras

Se habla de refinamiento de posición, al desplazamiento de las coordenadas en 2D donde se encuentra una figura según las características de la misma.

No aparece en el gráfico del pipe, dado que el centro del círculo que contiene a cada figura es una buena aproximación del “centro de masa” de la misma.

Para el refinamiento se usa el método “Parasyte”, que desplaza las coordenadas según la forma de la figura o la intensidad de los pixeles cercanos según el modo de umbralización seleccionado:

BINARY -> Análisis de Forma

TOZERO -> Análisis de intensidad

Para distancias de proyección (distancia entre el emisor y el plano u objeto donde se proyectan los puntos láser) menores a un 1m, el refinamiento de los puntos no genera grandes cambios, dado que nos encontramos en la zona donde es mayor la sensibilidad. Por cada pixel que se desplaza un punto, es menos significativo en cuanto a distancia a medida que la distancia de proyección disminuye.

Además, que los punteros láser sean de buena calidad y proyecten un punto perfectamente circular o casi perfecto, disminuye la probabilidad de la necesidad de refinamiento.

4.3.6. Análisis del triángulo

Es utilizado el método del promedio para el cálculo de la distancia al plano de proyección. Por ser más simple, y demostrar medio milímetro menos de error.

Requiere los parámetros del hardware:

- 1) La medida en metros de la arista del triángulo que forman los tres punteros láser en el emisor, en el hardware.
- 2) La constante, que es el numerador de la función hiperbólica para la función de correspondencia.

4.3.7. Salida/Output (RAFI)

- 1) La distancia al plano de proyección.
- 2) La inclinación del plano expresada en dos ángulos.
- 3) Factor de conversión.

4.4. Pipeline asociado a PET (Perspective Transform)

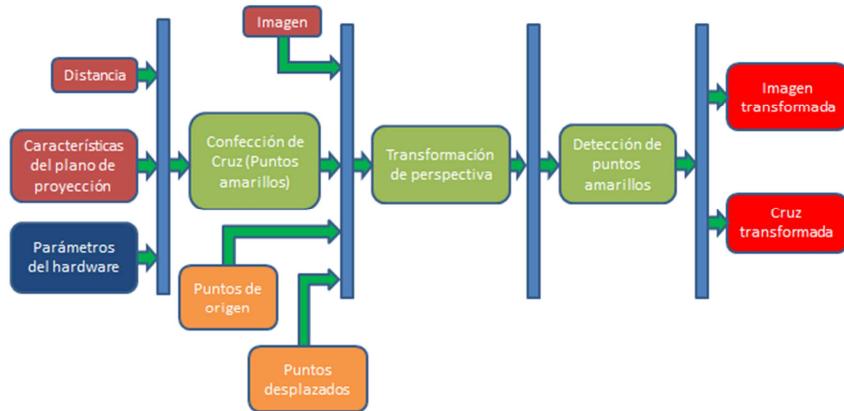


Figura 4.3. Pipeline de la transformación de perspectiva.

4.4.1. Confección de la “Cruz”

La distancia, las características del plano, y la función de correspondencia cuya constante es un parámetro que proporciona el hardware, son necesarias para la confección de la cruz.

4.4.2. Detección de los puntos amarillos

Una vez alterada la perspectiva de la imagen la cruz cambia. Para la detección y análisis de la nueva cruz se siguen los siguientes pasos:

- 1) Se convierte la imagen al espacio de color HSV.
- 2) Se aplica un filtro, para que quede en la imagen solo el color amarillo puro con el que se han dibujados los puntos de la cruz.
- 3) Se aplica un suavizado medianBlur para evitar que queden pixeles solitarios, que pueden ser erróneamente considerados como puntos de la cruz.
- 4) Se detectan los puntos, utilizando la función “findContours”.
- 5) Se calcula la distancia de cada uno de los cuatro puntos al centro.
- 6) Se dibujan las rectas en la imagen y se informa la longitud de cada recta.

4.4.3. Salida/Output (PET)

Es la imagen con la perspectiva alterada con la cruz y la información asociada a ella.

4.5. Pipeline asociado a OM (Object Measurement)

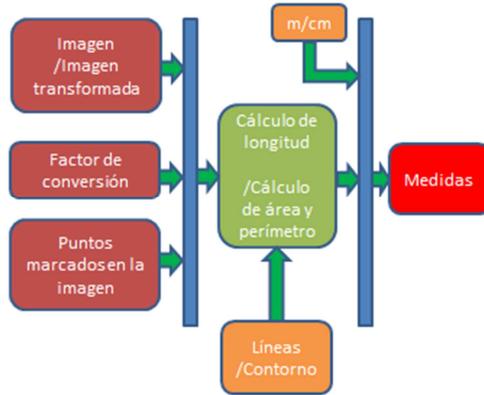


Figura 4.4. Pipeline de la medición de objetos.

No son necesarios más detalles, ya que el siguiente paso sería explicar el código que realiza la labor línea por línea. Pero si se puede decir que requiere el factor de conversión, y la imagen transformada o sin alterar según la inclinación detectada.

En la etapa intermedia, se interpretan los puntos como líneas o un solo contorno según la opción seleccionada.

4.6. Pipeline integral

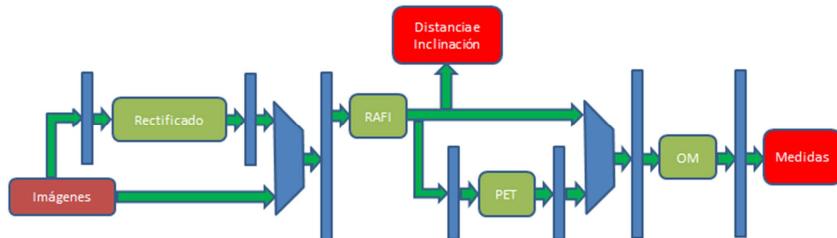


Figura 4.6. Pipeline integrador.

No figura el proceso de calibración, dado que la calibración para una cámara se realiza solo una vez.

No figuran todos los elementos de entrada necesarios para los procesos, porque en el pipeline detallado de cada proceso están a la vista.

La idea del pipeline integral es mostrar el orden los pipelines detallados, y que es opcional usar el rectificado, así como la transformación de perspectiva según la inclinación detectada.

Capítulo 5: Conclusiones y trabajos futuros

5.1 Introducción

A continuación se encuentran las conclusiones separadas respecto a: los objetivos propuestos, los requerimientos establecidos, y de una manera general.

5.2. En cuanto a los objetivos

El Objetivo principal que fue establecido es la construcción de un sistema de telemetría basado en imágenes digitales y láser, entonces podemos decir que:

- En cuanto al hardware, habiendo construido prototipos, se requieren punteros láser de mejor calidad, con los que se pueda conseguir tres haces de luz paralelos y una buena proyección de los puntos (lo más circular posible). Por lo menos hasta la distancia máxima que se necesita medir. No solo la calidad de los punteros influye en el paralelismo, sino también el material con el que se construye el soporte para los mismos, debido al desgaste con el tiempo.

La otra parte clave del hardware es la cámara, que define la función de correspondencia (punto 3.9). Esta función define el tamaño de la zona de sensibilidad, que influye directamente en el comportamiento y los resultados que podemos obtener con el software.

El hardware de los prototipos está lejos de ser ideal, posibilitó el estudio del comportamiento del fenómeno, y la realización de pruebas.

- En cuanto al software, que acompaña los prototipos, con la librería de visión artificial OpenCV participando en cada uno de los procesos, con las funciones que provee, sumado al álgebra multidimensional, fue posible cumplir con las funcionalidades propuestas como objetivos. Debido a la variedad de funciones y a la documentación disponible, fue posible encontrar la adecuada para cumplir cada tarea.
- Respecto a la deformación que provoca la lente de la cámara sobre las imágenes, investigando sobre la calibración de una cámara a partir del reconocimiento de patrones en imágenes, fue posible obtener los parámetros intrínsecos y extrínsecos de nuestra cámara. El error de reproyección resultante de la calibración es 0.2929, que se encuentra

dentro del rango [0,1; 1,0], nos habla de una buena calibración, con una media de error 0.03931. Entonces, está a disposición del usuario el rectificado de imágenes.

5.3. En cuanto a los requerimientos

- La distancia máxima que puede detectar el dispositivo depende de dos factores:
 - La función de correspondencia: la hipérbola crece al crecer la constante en el numerador. La constante crece mientras más alta sea la resolución de la cámara, expandiendo la zona por debajo de la curva y en consecuencia aumenta la zona más sensible de la función.
Sería ideal conseguir mejorar la sensibilidad y al mismo tiempo aumentar el alcance incrementando la resolución, pero si aumenta el alcance, lo hace de manera poco significativa. Sean c1 y c2 cámaras, c2 con mayor resolución: ambas cámaras capturan la misma vista desde la misma posición, ambas imágenes mostrarán lo mismo solo que c2 con mayor nivel de detalle. Si en una imagen capturada por c1 se ven las proyecciones de los puntos casi en contacto, con c2 pasa lo mismo. Pero puede ocurrir que la mejor definición distinga una pequeña separación existente que c1 no puede llegar a ver.
 - El aumento del alcance se logra aumentando la distancia entre los punteros. Es decir, hacer que formen un triángulo equilátero con un valor de arista mayor, pero al hacerlo el tamaño del dispositivo crece.
- Los resultados al obtener las características del plano son satisfactorios, siempre y cuando la distancia a la que se encuentra el plano de proyección esté en la zona de mayor sensibilidad. Fuera de la zona, disminuyendo de a uno los píxeles, en la medida de la arista del triángulo, el salto en cuanto a distancia cada vez es más alto, por ende lo mismo sucede para los ángulos. Esto causa error en el valor tanto de ángulos como de distancia detectada.
El límite que se le puede dar, a la distancia máxima de acuerdo a la zona de sensibilidad, es cuando la variación en un pixel involucra una variación en distancia mayor la mínima unidad que se quiere apreciar (1mm, 1cm, 1m).
- Las mediciones longitudinales, de área y perímetro son sencillas, no existe problema alguno mientras el plano no presente inclinación. Para unidades de medición como metros y centímetros los resultados son satisfactorios.
- Para realizar mediciones habiendo detectado inclinación en el plano, es necesario anular la inclinación realizando la transformación de perspectiva. Para hacerlo y obtener el mejor resultado, la inclinación debe tener el menor error posible, por lo que nuevamente es necesario que la distancia al centro del plano sea menor a la límite (dentro de la zona sensible).

Pero la transformación de perspectiva (PET) resulta poco intuitiva para el usuario. Además, si bien la cruz brinda la información que permite saber cuándo fue ajustada la perspectiva, es difícil llegar a ello. Y es incierto que tan certera es la información que provee, o si solo sirve como una aproximación hasta realizar pruebas con un hardware de mayor precisión.

- Para encontrar los puntos no solo se realiza un filtrado de intensidad, sino que se analizan la forma de las figuras detectadas, y la disposición en la imagen, por lo que el método software permite encontrar las proyecciones de los punteros láser aun en ambiente con considerable iluminación.

5.4. En cuanto a las limitaciones

- Para una adecuada sensibilidad, que nos permita obtener buenos resultados hasta la distancia máxima que el dispositivo debe detectar, se debe aumentar la resolución. Aumentar la resolución trae aparejada la manipulación y alojamiento en memoria de matrices de mayor tamaño, dilatando así el tiempo de respuesta, perjudicando el rendimiento. No solo perjudica el rendimiento, sino también el mostrar las imágenes en pantalla. Por más de que las ventanas cuenten con barras de desplazamiento, llega un punto en que es molesto para resoluciones altas, sobre todo en la pestaña OM de la interfaz de usuario. Debería ser agregado el uso de factores de zoom, teniendo en cuenta el impacto que tiene sobre las coordenadas en la imagen.

El cambiar a un lenguaje donde los tipos de datos son definidos en tiempo de compilación, es una alternativa para la mejora del rendimiento.

A pesar de todos los procesos encadenados que deben atravesar las imágenes, el tiempo de respuesta es aceptable, dado que se está trabajando con imágenes de 1Mpx.

- La limitación más clara es la distancia máxima detectable, relacionada directamente con el tamaño del dispositivo.

Entonces podemos decir que se trata de buscar un punto intermedio que satisfaga las necesidades de alcance, rendimiento, y resultados en cuanto al error aceptable.

5.5. En general

Lo que empezó como el diseño y construcción de un dispositivo terminó como desarrollo de prototipos, análisis del comportamiento del fenómeno (deformación del triángulo equilátero debido a la inclinación), de alternativas para cumplir con las tareas en cada etapa del procesamiento de las imágenes, limitaciones de un método para realizar telemetría mediante visión artificial.

5.5.1. Comparación con otros métodos

Existen métodos alternativos que también hacen uso de sensores o cámaras como el telémetro láser mencionado en marco teórico, que mediante el tiempo de vuelo mide distancia, y otro método conocido como mapa de disparidad. A continuación se comparan con el método de este proyecto (“triángulo láser”), en cuanto a ventajas y desventajas entre ellos.

- Telémetro láser mediante tiempo de vuelo:
 - Ventajas:
 - En cuanto al alcance es superior ya que depende solo de poder detectar el momento en que el rayo de láser retorna.
 - Al trabajar con la velocidad de la luz puede realizar un mapeo en tres dimensiones del entorno, debido a que el rayo es devuelto en muy poco tiempo. Entonces no requiere de una transformación de perspectiva para realizar mediciones de objetos.
 - Extenso campo de visión, existen dispositivos con barrido angular de 180° por ejemplo.
 - Desventajas:
 - La rapidez con la que el sensor o cámara detecta cuando retorna el rayo de luz define la sensibilidad en cuanto a medir la distancia. No es apropiado para distancias submilimétricas. El tema en cuestión es si pueden apreciar unidades menores milímetros, es decir poder identificar cambios en la distancia por ejemplo de 5,4305m a 5,4300m (medio milímetro). Que utilizando una cámara de alta resolución, el triángulo láser puede conseguirlo, pero por la zona de sensibilidad esta capacidad disminuye a la distancia. Trabajando con la velocidad de la luz tal precisión resulta difícil de conseguir.
 - Si se implementa con una cámara, requiere de una que sea de alta velocidad.
 - Suelen ser dispositivos aparatosos.

- Mapa de disparidad:

Son necesarias dos cámaras separadas una distancia imitando a los ojos humanos. Con respecto a los objetivos del proyecto con el procesamiento de las imágenes, mediante un algoritmo es posible mapear las coordenadas de la imagen (2D) con las del mundo real (3D), para obtener distancia, inclinación, mediciones de longitud, área y perímetro de objetos sin necesidad de alterar la perspectiva.

La función que utiliza el mapa de disparidad para obtener la distancia a la que se encuentra un punto es la siguiente:

$$z = \frac{f b}{d}$$

Donde d es la disparidad para ese punto, f la distancia focal de la cámara utilizada (las dos cámaras son idénticas) y b la separación entre las cámaras. Disparidad es la diferencia horizontal en pixeles para cada punto/pixel entre las dos imágenes, debido a que las cámaras se encuentran en el mismo plano y misma posición vertical, pero separadas la distancia b ^[12].

La función es de la forma $\frac{cte}{x}$ al igual que las funciones de correspondencia encontradas. Para comprobar si son las mismas tomamos f , que es el elemento a_{11} y también el a_{22} de la matriz asociada a la cámara utilizada, que resulta de la calibración de la misma. Con la constante del numerador de la función de correspondencia para 1Mpx hacemos:

$$\frac{156,25}{f} = \frac{156,25[px\ m]}{1400[px]} = 0,11160[m]$$

Se trata de la misma función, ya que 0,11m es la medida de las aristas del triángulo que forman los punteros láser montados. La distancia focal f se expresa en pixeles por lo tanto la constante en $px\ m$. Entonces podemos expresar las siguientes conclusiones, ventajas y desventajas:

- Conclusiones:
 - En ambos casos existe la zona de mayor sensibilidad.
 - Entonces para el triángulo láser se puede obtener la función de correspondencia a partir de la calibración de la cámara con la que se trabaje.
- Ventajas:
 - El mapa de disparidad puede calcular la distancia para toda la imagen, ya que puede obtener la disparidad para cualquier punto de la misma. Y el cálculo del plano tomando tres puntos. En cambio el triángulo láser solo puede obtener la distancia y el plano para el centro de la imagen.
 - Se puede realizar mediciones de objetos sin necesidad de una transformación de perspectiva.
 - No requiere la utilización de punteros láser.
- Desventajas:
 - Frente a las técnicas que utilizan láser presenta la desventaja de no poder obtener información de imágenes capturadas en la oscuridad.
 - Para aumentar el alcance se debe aumentar la separación entre las cámaras, así como con la técnica desarrollada con el triángulo láser, se debe aumentar la medida de arista del mismo. Lo que trae aparejado problemas para el desempeño del mapa de disparidad al diferenciarse cada vez más las imágenes que toman

las cámaras. Y en ambos casos provoca un aumento del tamaño del dispositivo.

La principal ventaja que tienen las dos técnicas respecto a la utilizada en el proyecto es que no requieren de la transformación de perspectiva.

5.6. Trabajos futuros

Como trabajo futuro podemos plantear lo siguiente:

- Construcción de un dispositivo con punteros láser de mejor calidad, cámara con mayor resolución y un soporte duradero de modo de garantizar el paralelismo de los haces de luz.
- Una vez establecido el alcance máximo que se le quiere dar al dispositivo, hacer pruebas de rendimiento para una resolución de la cámara que brinde una zona de sensibilidad lo suficientemente extensa para cubrir ese alcance.
- Un estudio del error en las mediciones de distancia, inclinación, longitud, área y perímetro.

Anexo A

Estructura de directorios:

```
 Predator Release
  +-- calibration
    +-- input_images
    +-- output_images
    +-- undistortion_remapping_result
    +-- camera_calibration.py
  +-- camera_data
    +-- camera_parameters.txt
  +-- json_trade
    +-- json_functions.py
  +-- rafipetom
    +-- brightest_dots_finder.py
    +-- circle_functions.py
    +-- distance_functions.py
    +-- error_img.png
    +-- measure_functions.py
    +-- paint_functions.py
    +-- parasite.py
    +-- r3_functions.py
    +-- run_rf.py
    +-- triangle_analyzer.py
    +-- trinity_hunter_rev.py
    +-- trinity_hunter.py
    +-- ui.py
```

Figura A.1. Estructura de directorios

En camera_parameters.txt son escritos los parámetros resultantes de la calibración.

error.img es la siguiente imagen (pero en 640x503):



Figura A.2. Imagen de Error.

Referencias

- [1] <https://es.wikipedia.org/wiki/Telemetr%C3%A9dica>
- [2] <http://wiki.robotica.webs.upv.es/wiki-de-robotica/sensores/sensores-proximidad/sensor-laser/>
- [3] https://es.wikipedia.org/wiki/Imagen_digital
- [4] https://es.wikipedia.org/wiki/Resoluci%C3%B3n_de_imagen
- [5] https://es.wikipedia.org/wiki/Profundidad_de_color
- [6] <https://en.wikipedia.org/wiki/Grayscale>
- [7] https://en.wikipedia.org/wiki/RGB_color_model
- [8] https://es.wikipedia.org/wiki/Modelo_de_color_HSV
- [9] https://es.wikipedia.org/wiki/Visi%C3%B3n_artificial
- [10] <https://es.wikipedia.org/wiki/OpenCV>
- [11] https://opencv-python-tutorial.readthedocs.io/en/latest/py_tutorials/py_core/py_optimization/py_optimization.html
- [12] Melisa Elizabeth Del Valle Torres (2015). Sistema de Visión Estereoscópica para el Control de la Movilidad en Espacios con Obstáculos (Tesis de grado). Universidad Nacional de Córdoba, Facultad de Ciencias Exactas, Físicas y Naturales, Córdoba.
- [13] https://opencv-python-tutorial.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_colorspaces/py_colorspaces.html#converting-colorspaces
- [14] https://opencv-python-tutorial.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html
- [15] https://opencv-python-tutorial.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html
- [16] https://opencv-python-tutorial.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghcircles/py_houghcircles.html
- [17] https://opencv-python-tutorial.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contours_begin/py_contours_begin.html
- [18] https://opencv-python-tutorial.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contours_hierarchy/py_contours_hierarchy.html
- [19] Elizabeth Vera de Payer (2005). Álgebra Lineal. Manual de Cátedra - Editorial Universitas
- [20] https://es.wikipedia.org/wiki/F%C3%B3rmula_de_Her%C3%B3n
- [21] https://opencv-python-tutorial.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html
- [22] https://opencv-python-tutorial.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html#calibration