RECONSTRUCCIÓN 3D DE OBJETOS SUMERGIDOS EN AGUA

ANEXOS

EDILSON QUIÑONEZ JIMÉNEZ
HÉCTOR DANIEL VICTORIA PIZO

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
DEPARTAMENTO DE ELECTRÓNICA, INSTRUMENTACIÓN Y CONTROL
INGENIERÍA EN AUTOMÁTICA INDUSTRIAL
POPAYÁN, FEBRERO DE 2013

RECONSTRUCCIÓN 3D DE OBJETOS SUMERGIDOS EN AGUA

ANEXOS

DOCUMENTO FINAL DE TRABAJO DE GRADO PARA OPTAR AL TÍTULO DE INGENIERO EN AUTOMÁTICA INDUSTRIAL

EDILSON QUIÑONEZ JIMENEZ
HÉCTOR DANIEL VICTORIA PIZO

DIRECTOR: ING. LEYDY VIVIANA MUÑOZ ROBLES

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
DEPARTAMENTO DE ELECTRÓNICA, INSTRUMENTACIÓN Y CONTROL
INGENIERÍA EN AUTOMÁTICA INDUSTRIAL
POPAYÁN, FEBRERO DE 2013

Tabla de contenido

Lista de Figuras

Lista de Tablas

Anexo A: Resultados obtenidos en la etapa de calibración de la cámara

Terminado el proceso de calibración, ya se cuenta con los valores de los parámetros intrínsecos y extrínsecos de la cámara. Sin embargo estos valores son muy sensibles a variaciones si se cambia la posición del patrón de calibración. De acuerdo a esto se hace necesario realizar repetidas veces el proceso de calibración y posteriormente comparar los resultados obtenidos. En este trabajo se estimaron 10 veces los parámetros de la cámara. Con el conjunto de datos obtenidos se realiza la validación de estos y se determina el mejor valor posible para trabajar con el. Los datos se muestran en la tabla 1.

Tabla 1. Parámetros de la cámara calculados para diferentes capturas del patrón de calibración.

Número de Capturas	$\boldsymbol{f_x = (f * k_u)}$	$\boldsymbol{f_y = (f * k_v)}$	$\boldsymbol{u_0}$	$\boldsymbol{v_0}$
20	713,89	703,42	321,26	243,32
20	683,15	689,10	227,59	255,77
20	665,21	670,10	339,73	289,24
20	683,15	689,10	287,59	255,77
18	669,97	701,59	334,06	230,28
18	678,15	685,11	317,59	243,77
18	708,10	711,05	318,02	255,27
14	680,53	670,52	306,23	266,33
14	614,35	427,34	349,14	271,77
14	576,38	489,56	375,36	285,81

De los resultados obtenidos se puede apreciar que existen variaciones de los mismos. Para 14 capturas se tienen valores muy variantes, esto permite concluir que para la cámara utilizada es necesario tomar más de 14 capturas. Como se dijo anteriormente no se tienen valores del fabricante para los parámetros, por lo tanto se debe tomar cada resultado obtenido y evaluarlo con un método que permita determinar si dichos valores están bien estimados.

Los valores estimados por el algoritmo de calibración corresponden al factor f_x y f_y Como se puede ver, en ambos factores se repite el parámetro f el cual representa la distancia focal. Este parámetro es multiplicado por el tamaño pixel/milímetro horizontal y vertical respectivamente (k_u, k_v). Cuando el factor de escala es igual a

la unidad $(k_u, k_v = 1)$ se tiene que el factor f_x es igual al factor f_y con lo que se concluye que el pixel es cuadrado. En la práctica esto no sucede debido a que es casi imposible garantizar tal precisión en la construcción de las cámaras. Explicado lo anterior lo adecuado es escoger los parámetros estimados f_x, f_y que menos se diferencien en valor. De los datos registrados en la tabla 1 se puede apreciar que los mas adecuados son $f_x = 708{,}10$ pixeles, $f_y = 711{,}05$ pixeles. Con estos valores aun no se puede calcular el valor de la distancia focal f. Se necesita conocer el tamaño horizontal y vertical en milímetros del sensor de imagen CMOS de la cámara. Para las cámaras web y en especial para la que se está trabajando los sensores de imagen están estandarizados a un tamaño de ¼ de pulgada. Con este valor se puede verificar la relación en milímetros de acuerdo a la tabla 2.

Tabla 2. Tamaños del sensor de imagen

Tipo	1/8"	1/6"	1/4"	1/3.6"	1/3.2"	1/3"	1/2.7"	1/2.5"	1/2.3"	1/2"	1/1.8"	1/1.7"	1/1.6"	2/3"
Diagonal (mm)	2.00	3.00	4.00	5.00	5.68	6.00	6.72	7.18	7.7	8.00	8.93	9.50	10.07	11.0
Ancho (mm)	1.60	2.40	3.20	4.00	4.54	4.80	5.37	5.76	6.16	6.40	7.18	7.60	8.08	8.80
Alto (mm)	1.20	1.80	2.40	3.00	3.42	3.60	4.04	4.29	4.62	4.80	5.32	5.70	6.01	6.60
Área (mm^2)	1.92	4.32	7.68	12.0	15.5	17.3	21.7	24.7	28.5	30.7	38.2	43.3	48.56	58.1

Fuente: http://es.wikipedia.org/wiki/Formato_del_sensor_de_imagen

La tabla 2 registra los valores típicos en milímetros de los tamaños de los sensores de imagen en diferentes valores de pulgadas. Con esto se sabe que para el sensor de la cámara utilizada los tamaños son: 3.2 de ancho y 2,4 de alto. Además la cámara trabaja a una resolución de 640x480 pixeles. Entonces se puede obtener la relación milímetro/pixel como se muestra en la ecuación 1.

$$3{,}2/640 = 0{,}005\ mm/pixel \qquad 2{,}4/480 = 0{,}005\ mm/pixel \quad (1)$$

El número de pixeles por milímetro se calcula en la ecuación 2.

$$1/5\mu m = 200\ píxeles/mm \quad (2)$$

Con este dato se calcula la distancia focal f como se muestra en las ecuaciones 3, 4, 5.

$$f = fx/(200\ píxeles/mm)\ y\ f = fy/(200\ píxeles/mm) \quad (3)$$

$$f = 708{,}10/(200\ píxeles/mm)\ y\ f = 711{,}05/(200\ píxeles/mm) \quad (4)$$

$$f = 3{,}54\ píxeles\ y\ f = 3{,}56\ píxeles \quad (5)$$

En cámaras web tanto de gama baja como de gama alta la longitud de la distancia focal se ha estandarizado a 3,7 milímetros. En los resultados que se han obtenido en el cálculo de la distancia focal se puede notar que los valores son muy cercanos al valor real definido por los fabricantes. Por lo tanto se asume que el valor de la cámara utilizada en este proyecto es de 3,6 milímetros.

Anexo B: Algoritmos de Renderizado

B.1 Tipos De Archivos Utilizados Por Las Herramientas De Renderizado

Cuando se obtiene la información de un punto en el espacio, es posible no solo obtener información correspondiente a la posición, sino además obtener información de la textura y el color. Sin embargo el uso o no de esta información dependerá de la aplicación.

Existen unos formatos definidos para almacenar estos datos en forma de arreglos de tamaño NxM, donde N es la cantidad de puntos que tiene la nube (o pixeles) y M la cantidad de atributos por cada punto (ubicación en el espacio, textura y/o color). Es importante utilizar los formatos, debido a que será por medio de estos que se obtenga y extraiga las nubes de puntos.

Archivos VRML: Es un estándar ISO publicado en 1997, sus siglas significan “Virtual Reality Modeling Languaje”. Un archivo VRML describe una escena como un grafo, entre sus nodos se encuentran los “PointSet” y los “IndexedFaceSet”. Un nodo “PointSet” almacena las coordenadas en tres dimensiones (X, Y, Z) de cada punto y el color de cada punto y un nodo “IndexedFaceSet” puede almacenar

mallas poligonales con colores, vectores normales y coordenadas de texturas [1].

Archivos PLY: PLY es un formato de archivo informático conocido como el formato Polígono o el formato de triángulo Stanford, este último nombre debido a que en la Universidad de Stanford utilizó el formato PLY para almacenar los datos resultantes de un escaneo 3D de la escultura Michelangelo, en una resolución extremadamente alta. Permite almacenar color y transparencia, normales, y coordenadas de textura [1].

B.2 Algoritmos De Renderizado

Según [2] los algoritmos de reconstrucción se clasifican en cinco categorías descritas a continuación:

De acuerdo al tipo de dato de entrada:

✓ Nubes de punto no organizadas: Los algoritmos trabajan con datos no organizados, es decir no se tiene información de la posición espacial de los puntos, ni se realiza ningún tipo de asunciones con respecto a la geometría del objeto. estos algoritmos suelen fallar. si los puntos no son uniformemente distribuidos.

✓ Nubes de punto estructuradas: Estos algoritmos pueden tomar en cuenta la información adicional que se les suministre como por ejemplo color y textura y lograr una mejor estimación de la superficie original sin inferir cualidades de la misma.

De acuerdo a su división espacial:

✓ Orientados a superficies: Estos algoritmos no distinguen entre superficies cerradas o abiertas. La mayoría de los algoritmos existentes son de éste tipo.

✓ Orientados a volumen: Funcionan en particular con superficies cerradas y generalmente son basados en la triangulación Delaunay.

De acuerdo al tipo de representación de la superficie:

✓ Representación paramétrica: Estos métodos representan la superficie como un conjunto paramétrico de “partes” de superficies, descritos por ecuaciones paramétricas. Posteriormente, estas partes son unidas para formar una superficie

continua. Ejemplos de estos algoritmos son el B - spline, Curvas Bezier.

✓ Representación implícita: Estos métodos consisten en utilizar una función implícita para todos los puntos, donde para algunos el resultado obtenido será un valor especificado (usualmente cero).

✓ Representación simple: En esta representación la superficie es una colección en de entidades simples como puntos, lados y triángulos Ejemplos Alpha Shapes, Crusts Algorithm.

De acuerdo a la manera de aproximación o interpolación:

✓ Superficies aproximadas: No siempre contienen todos los puntos originales, pero tienen puntos muy cercanos a ellos. Pueden utilizar una función de distancia (como la distancia más corta al punto en espacio de la superficie generada) para estimar la malla (mesh) correcta.

✓ Superficies interpoladas: Son utilizadas cuando se requieren modelos precisos. Se requieren la obtención de más de una nube de puntos a fin de interpolarlas y presentar los resultados totales.

De acuerdo a las diferentes asunciones que realizan los algoritmos:

✓ Asumen tipo topológico ajustado: Usualmente asumen que el tipo topológico de la superficie es conocido a priori (plano, cilindro o esfera).

✓ Explotan estructura o información de orientación: Explotan la estructura de los datos para la reconstrucción de superficie o el conocimiento de orientación de la superficie otorgado por los datos.

B.3 Pasos Para Construir Una Superficie Poligonal A Partir De Una Nube De Puntos

✓ **Operaciones de Pre-procesamiento:**

Las operaciones de editado constituyen una herramienta importante antes de generar una superficie triangular. Las operaciones generalmente consisten el remover los pixeles que se introdujeron de manera errónea en la nube de puntos debido a brillos ocasionados por el láser sobre otra superficie diferente a la que se

desea reconstruir. Existen diferentes métodos para eliminar estos pixeles.

a) *Muestreo de datos:* Basado en la curvatura de los puntos o aplicación uniforme. Este método se utiliza para reducir la redundancia de datos de entrada (downsampling), generalmente se utiliza cuando se toman varias muestras de una imagen, es decir la imagen es escaneada varias veces.

b) *Rellenado de huecos:* Busca introducir nuevos pixeles en los espacios de la nubes de datos sin información, estos nuevos pixeles pueden ser agregados de forma manual o automática mediante herramientas software. Existen diferentes métodos para rellenado, sin embargo todos se utilizan la curvatura y densidad de puntos circundantes para reconstruir los nuevos pixeles.

- ✓ **Generación de superficie poligonal**

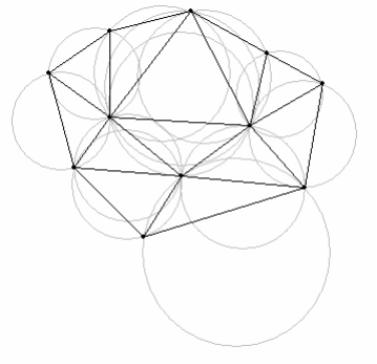
Esta constituye la etapa más importante de la reconstrucción y busca convertir un conjunto de datos (pixeles) provenientes de la nube de puntos, en un modelo poligonal conocido como maya, basándose en diferentes métodos que buscan la conformación de triángulos o cuadriláteros en dos dimensiones y tetraedros en tres dimensiones, generando vértices, aristas y caras.

La triangulación puede ser realizada en 2D o en 3D, de acuerdo con la geometría de los datos de entrada.

a) *Triangulación 2D:* El dominio de los datos de entrada es el plano, y se busca generar triángulos que se crucen en los bordes compartidos y vértices. El método más conocido es la triangulación de Delaunay, consistente en una red de triángulos que se caracterizan por formar triángulos lo más equiláteros posible, es decir, maximiza el mínimo ángulo de los triángulos, de forma que los puntos mas próximos entre si estén conectados por una arista, en la que los triángulos resultantes sean lo más regulares posibles [3].

La triangulación Delaunay es una estructura geométrica que ha tomado gran popularidad en la generación de mallas. Consiste en un conjunto de triángulos cuyos vértices no se interceptan con ningún otro; es decir, que para cada triángulo debe existir un circumcirculo que no contenga ningún otro vértice [4]. En la figura 1 se observa un ejemplo de triangulación de Delaunay.

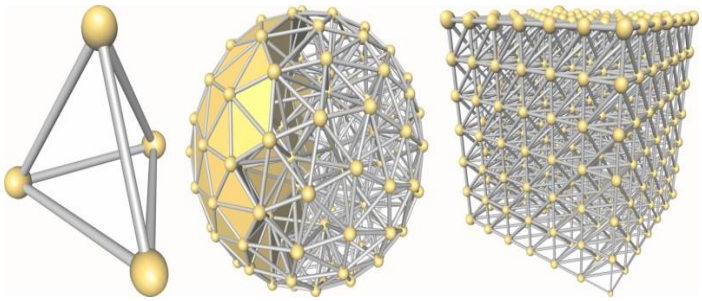
Figura 1. Ejemplo de la triangulación de Delaunay



Fuente: [3]

b) *Triangulación 3D:* La triangulación en 3D se llama tetraedralización o tetraedración. Una tetraedralización es una partición del dominio de entrada en un conjunto de tetraedros que se encuentran sólo en caras compartidas (vértices, aristas o triángulos). Los resultados tetraedralizados son mucho más complicados que los de una triangulación 2D como se observa en la figura 2. Los tipos de dominios de entrada podrían ser poliedro simples (esfera), poliedros no-simples (torus) o nubes de puntos.

Figura 2. Triangulación 3D



Fuente: [2]

- ✓ **Operaciones de post-procesamiento**

Las mayas obtenidas generalmente necesitan algunas correcciones, bien sea mayas aisladas del objeto reconstruido, o mayas unidas al objeto y que deben ser eliminadas porque no forman parte del objeto como tal, algunas de las tareas post-procesamiento pueden ser:

a) Inserción de triángulos: Es posible rellenar los agujeros con estructuras poligonales basadas en el área adyacente o triángulos vecinos.

b) Edición de polígonos: Es posible reducir el número de polígonos obtenidos al realizar la maya de la nube de puntos, esto permite reducir el tiempo de procesamiento y la velocidad de edición de la imagen. También es posible suavizar la imagen mediante la eliminación de picos.

B.4 Herramientas Software Que Utilizan Nubes De Puntos Para Construir Superficies

Meshlab*:* Es un software de código libre que procesa mallas 3D, su desarrollo empezó en el 2005 y su función principal es el procesamiento de datos que vienen de un escaneo 3D. Esta herramienta se utiliza para el procesamiento y edición de nubes de puntos en dos y tres dimensiones. Permite girar la imagen a fin de ver la profundidad de los puntos y la posición en el espacio.

El sistema se basa principalmente en la biblioteca VCG desarrollada en el Laboratorio de Computación Visual de ISTI – CNR en Italia, y está disponible para Windows, MacOSX y Linux.

Geomagic: Permite reconstruir una infinita variedad de formas a partir de datos de un escaneo 3D. Está diseñado para manejar la mayoría de la ingeniería inversa. Transformando datos de escaneo 3D en mallas poligonales. Es un complemento perfecto para CAD. Disponible únicamente en Windows.

Otros Softwares: Rapidform, Polyworks, existe un plug-in para AutoCad denominado Point Cloud.

Anexo C: Criterios de Selección de la Técnica de Reconstrucción 3D

El objetivo del análisis DOFA es obtener información clara, específica y necesaria para tomar la decisión en cuanto a la escogencia de la técnica que se utilizará en el sistema de reconstrucción de vistas isométricas de objetos sumergidos en agua. Esta herramienta permite clasificar la información en cuatro partes las cuales son: Debilidades, Oportunidades, Fortalezas y Amenazas, que en este caso presentan las técnicas mencionadas en el capítulo 1 (sección 1.1).

Las Debilidades hacen referencia a desventajas de las técnicas, vulnerabilidades en su funcionamiento e implementación, los altos costos de implementación. Las Oportunidades hacen referencia a los desarrollos tecnológicos, a los mercados objetivos, a las investigaciones realizadas. Las Fortalezas hacen referencia a las ventajas con respecto a las otras técnicas, las capacidades en su funcionamiento, los bajos costos de implementación. Las Amenazas hacen referencia a nuevas técnicas y al desarrollo tecnológico de éstas, influencias del sitio donde funcionarán las técnicas, problemas en el funcionamiento de las mismas. Cave aclarar que estos cuatro puntos de vista se enmarcaron dentro del contexto de los criterios anteriormente definidos.

A continuación se registran en las tablas 3 y 4 las características de las técnicas con contacto y sin contacto.

Tabla 3. Técnicas de reconstrucción 3D con contacto

TEMA DE ANÁLISIS: TÉCNICAS DE RECONSTRUCCIÓN 3D CON CONTACTO	
DEBILIDADES	**OPORTUNIDADES**
Alto costo de los sistemas que implementan las técnicas. Como ejemplo se pueden citar los brazos articulados. Alto grado de complejidad en la implementación de las técnicas. Se emplean en pocas áreas de investigación. Según la literatura consultada, hasta el momento no existen trabajos desarrollados	Exististe un fuerte trabajo de investigación.

en el medio acuático. No es fácil la aplicación en el campo académico. No se obtienen resultados rápidamente. No puede trabajar a grandes distancias. Manipula el objeto que se está reconstruyendo y puede llegar a dañarlo.	
FORTALEZAS Alto grado de precisión. Capturan gran cantidad de datos. Puede reconstruir objetos huecos.	**AMENAZAS** A lo lago de los años son menos las áreas de investigación y desarrollo que implementan estás técnicas, inclusive se puede pensar que tiende a desaparecer. Otras técnicas económicas, eficientes y tecnológicamente desarrolladas.

Tabla 4. Técnicas de reconstrucción 3D sin contacto

TEMA DE ANÁLISIS: TÉCNICAS DE RECONSTRUCCIÓN 3D SIN CONTACTO	
DEBILIDADES Alto grado de complejidad en el funcionamiento de algunas técnicas. Algunas técnicas no son fácilmente implementadas en el campo académico. Algunas técnicas no funcionan a grandes distancias. Son medianamente sensibles a niveles de incertidumbre en los resultados obtenidos.	**OPORTUNIDADES** Actualmente son implementadas en todos los campos de investigación y desarrollo. Están en continuo desarrollo tecnológico. Existe instrumentación de fácil acceso para implementar algunas de las técnicas. Existen técnicas que se pueden implementar con sistemas muy económicos.
FORTALEZAS	**AMENAZAS**

Menos costo en los sistemas que implementan estas técnicas. Alto grado de precisión. Existen trabajos desarrollados en el medio acuático implementando algunas de estas técnicas. Capturan gran cantidad de datos. Puede reconstruir objetos huecos. Algunas técnicas son aplicables en el campo académico. Gran velocidad en la adquisición de datos. Algunas técnicas funcionan a grandes distancias. No manipula el objeto que se está reconstruyendo.	Algunas de estas técnicas han sido remplazadas por otras pertenecientes a este mismo grupo.

Del resultado del anterior análisis se puede concluir que la técnica de reconstrucción 3D mas adecuada para implementar en este trabajo pertenece al grupo de técnicas sin contacto. Seguidamente se especifican en las tablas 5 y 6 las técnicas que pertenecen a este grupo.

Tabla 5. Técnicas de reconstrucción 3D Ópticas

TEMA DE ANÁLISIS: TÉCNICAS DE RECONSTRUCCIÓN 3D ÓPTICAS	
DEBILIDADES	**OPORTUNIDADES**
Existen pocos trabajos desarrollados en el medio acuático. Son sensibles a incertidumbres cuando el	Están en continuo desarrollo tecnológico. Actualmente son aplicadas en muchos campos que trabajan con realidad virtual.

objeto es hueco. No funcionan a grandes distancias. Su buen funcionamiento depende de la iluminación de la escena en la que se está trabajando. Presenta sensibilidad al ruido e información errónea.	Se consigue gran cantidad de instrumentos y/o sistemas económicos y de fácil acceso.
FORTALEZAS Bajo costo en comparación a las técnicas no ópticas. Alto grado de precisión. Capturan gran cantidad de datos. Gran velocidad en la adquisición de datos. Funcionan a distancias cortas. Se pueden implementar en el campo académico.	**AMENAZAS**

Tabla 6. Técnicas de reconstrucción 3D no Ópticas

TEMA DE ANÁLISIS: TÉCNICAS DE RECONSTRUCCIÓN 3D NO ÓPTICAS	
DEBILIDADES Alto costo en la implementación. Como ejemplo de sistemas y/o instrumentos que funcionan con estas técnicas se mencionan los sonares, radares y ecosondas. Alto grado de complejidad en la construcción de sistemas que implementen estas técnicas. Estas técnicas no son fácilmente implementadas en el campo académico. Presenta sensibilidad al ruido e información errónea.	**OPORTUNIDADES** Están en continuo desarrollo tecnológico. Están empezando a incursionar en otros campos de ingeniería. Hoy en día se consiguen instrumentos más económicos pero con menos prestaciones.
FORTALEZAS Alto grado de precisión. Existen trabajos desarrollados en el medio acuático. Capturan gran cantidad de datos. Pueden reconstruir objetos huecos. Gran velocidad en la adquisición de datos. Funcionan a grandes distancias. No son afectadas por la iluminación de la escena.	**AMENAZAS** Algunas de estas técnicas han sido remplazadas por otras pertenecientes a este mismo grupo. Actualmente son implementadas en pocos campos, siendo el campo marítimo su bandera.

Con la información registrada en las tablas 5 y 6 se puede decir que los dos grupos de técnicas analizadas presentan similitud en cuanto a fortalezas, sin embargo basándose en los criterios de selección definidos se puede afirmar que el grupo más adecuado para trabajar es el grupo de técnicas ópticas. Seguidamente

en las tablas 7 y 8 se registra la información que se obtuvo del análisis de las técnicas pertenecientes a este grupo.

Tabla 7. Técnicas de reconstrucción: Triangulación láser, Franjas

TEMA DE ANÁLISIS: TÉCNICAS DE RECONSTRUCCIÓN TRIANGULACIÓN LÁSER Y FRANJAS	
DEBILIDADES Existen pocos trabajos desarrollados en el medio acuático. Son sensibles a incertidumbres cuando el objeto es hueco. Presentan problemas de reconstrucción cuando el objeto tiene oclusiones. Su buen funcionamiento depende de la iluminación de la escena en la que se está trabajando, sin embargo no es tan crítico como lo es con las técnicas pasivas. Presenta sensibilidad al ruido e información errónea.	**OPORTUNIDADES** Están en continuo desarrollo tecnológico. Actualmente son aplicadas en muchos campos que trabajan con realidad virtual. Se consigue gran cantidad de instrumentos y/o sistemas económicos y de fácil acceso.
FORTALEZAS Bajo coste. Alto grado de precisión. Funcionan a mayores distancias que las técnicas pasivas. Fácil implementación. Funciona mejor cuando la escena tiene poca iluminación.	**AMENAZAS**

Capturan gran cantidad de datos. Gran velocidad en la adquisición de datos. Funcionan a distancias cortas. Se pueden implementar en el campo académico.	

Tabla 8. Técnicas de reconstrucción: Visión estereoscópica

TEMA DE ANÁLISIS: TÉCNICA DE RECONSTRUCCIÓN VISIÓN ESTEREOSCÓPICA	
DEBILIDADES Existen pocos trabajos desarrollados en el medio acuático. Son sensibles a incertidumbres cuando el objeto es hueco. No funcionan a grandes distancias. Su buen funcionamiento depende de la iluminación de la escena en la que se está trabajando. Presenta sensibilidad al ruido e información errónea.	**OPORTUNIDADES** Están en continuo desarrollo tecnológico. Actualmente son aplicadas en muchos campos que trabajan con realidad virtual. Se consigue gran cantidad de instrumentos y/o sistemas económicos y de fácil acceso.
FORTALEZAS Bajo coste. Alto grado de precisión. Captura gran cantidad de datos. Gran velocidad en la adquisición de datos. Funcionan a distancias cortas.	**AMENAZAS**

Se pueden implementar en el campo académico.	

De la información plasmada en las tablas 7 y 8 se puede notar que ambas técnicas analizadas presentan similares características. Aun así, la triangulación láser se acopla mejor a los criterios definidos.

Es necesario comentar que en la actualidad existen gran número de trabajos de investigación desarrollados en los que han combinado la visión estereoscópica con la triangulación láser. Con dicha combinación se obtienen resultados más óptimos debido a que se emplean ambos sistemas de reconstrucción y uno sirve como método de evaluación para verificar resultados del segundo, y viceversa. La contraparte de esto es que aumentan los costos de implementación del sistema conjunto. Por esta razón se decidió trabajar con la técnica triangulación láser. Se escogió esta técnica debido a que funciona bien en escenarios con poca iluminación. Este criterio es fundamental en este trabajo debido a que los resultados que se obtengan en el mismo, son una base para un futuro trabajo de maestría donde se trabajarán aguas turbias.

Anexo D: Algoritmos Desarrollados

D.1 Criterios De Selección De Las Herramientas Software

Inicialmente se hizo una valoración de los lenguajes de programación existentes, para determinar cual de estos aporta mayores prestaciones. Para esto se definieron los siguientes criterios de evaluación y selección de los lenguajes.

- Velocidad de cálculo y de procesamiento.
- Perteneciente a la familia de software libre.
- Rápido aprendizaje de su sintaxis.
- Inclusión de librerías de visión artificial.
- Multiplataforma.
- Bien desarrollado e utilizado en la industria.

Basándose en estos criterios se escogieron en primera medida los lenguajes: Java, C++, Python, los cuales satisfacen la mayoría de criterios.

Java es un lenguaje de programación multiplataforma, de código abierto, muy fuerte en la industria y de fácil aprendizaje. Sin embargo para que funcione en cualquier sistema operativo necesita de su máquina virtual, que es la que se encarga de interpretar los scripts desarrollados en dicho lenguaje y convertirlos a lenguaje máquina. Este proceso requiere de un tiempo de ejecución adicional, que sumado con el tiempo de procesamiento de los scripts disminuyen la velocidad de cálculo.

Por su parte, C++ es el lenguaje de programación más eficiente que actualmente existe. En contraparte, la curva de aprendizaje de éste es lenta. Además de esto, el lenguaje no controla el problema de desbordamiento de buffer, dejando este trabajo al programador. Esto se traduce en que si no se controla periódicamente el acceso a memoria y la cantidad de datos que se copian en un área determinada de la misma, puede ocurrir que se sobre pase el tamaño de almacenamiento de esa área y la información que no se pudo guardar automáticamente se guarda en áreas adyacentes a la primera, sobre escribiendo la información que ya estaba guardada en estas últimas.

Por último, Python es un lenguaje de programación multiplataforma, de código abierto, muy fuerte en la industria y de fácil aprendizaje. Es actualmente el segundo lenguaje de programación más rápido en tiempos de procesamientos y cálculos, controla el desbordamiento de *buffer* y tiene un fuerte soporte en la librería de visión artificial OpenCV. Adicional a lo anterior, se quiso trabajar con un sistema operativo de código abierto, que éste se acerque a un sistema de tiempo real y que el lenguaje a utilizar fuera fácil de instalar. Entre el sistema operativo Windows y el sistema operativo Ubuntu se escogió el último, este sistema operativo trae preinstalado el lenguaje de programación Python y la instalación de los componentes adicionales, como la librería OpenCv, es muy sencilla.

Con todo lo anterior se optó por seleccionar el lenguaje de programación Python. En las secciones siguientes se detallan todos los algoritmos desarrollados en este lenguaje de programación y las funciones implementadas. Los algoritmos se exponen de acuerdo a al orden de los procesos desarrollados en este trabajo.

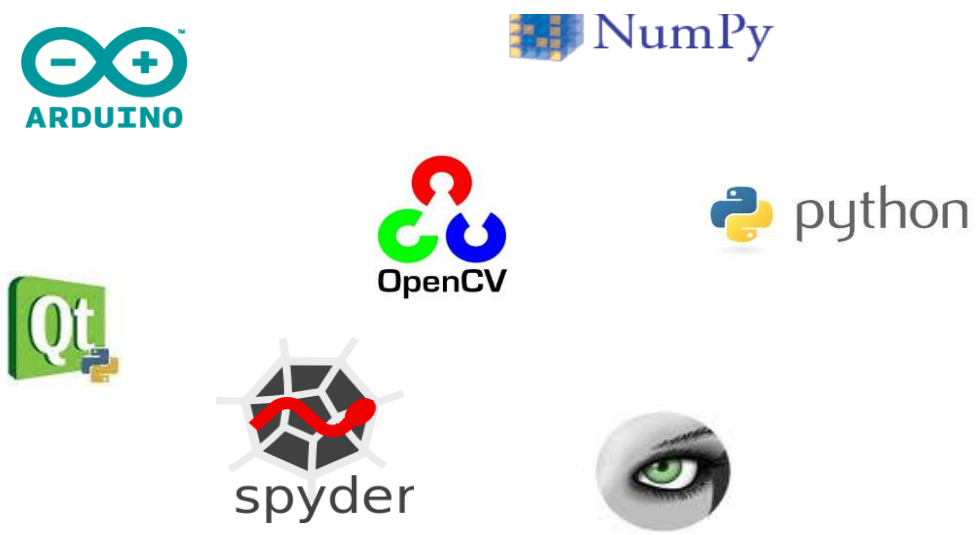
En segundo lugar se escogió la herramienta PyQt4 para el desarrollo de las interfaces gráficas, Spyder como IDE para el desarrollo del código.

Por último se escogió la herramienta MeshLab para realizar el renderizado de las nubes de puntos. Los criterios de selección de esta herramienta se describen en la tabla 9. En la figura 3 se presentan los logotipos de estas herramientas.

Tabla 9. Criterios de selección de la herramienta para el renderizado

CARACTERÍSTICA	SOPORTADA	
	MeshLab	Blender
Selección interactiva y supresión de una porción de malla para eliminar ruidos ocasionados por el láser.	X	X
Representar de manera gráfica la nube de puntos, mediante la importación de un archivo que contiene las coordenadas tridimensionales del objeto.	X	X
Eliminación de vértices duplicados.	X	
Eliminación de pequeños componentes aislados.	X	X
Construcción de superficies a partir puntos.	X	X
Medidas lineales entre los puntos de la malla	X	
Historial de todas las acciones de limpieza y editado realizadas., se puede volver a reproducir en diferentes mallas o guardarla con fines de archivo.	X	

Figura 3. Logos de las herramientas implementadas



D.2 Descripción de Las Interfaces de Usuario desarrolladas

Para la calibración de la cámara se construyó una interfaz de usuario, la cual se muestra en la figura 4. Esta interfaz recibe los parámetros descritos a continuación:

- Variables del sistema:

✓ Parámetro de la cámara: este parámetro es utilizado por la librería de OpenCV y puede tomar los siguientes valores 0, 1. 2, etc.: En caso de tener múltiples cámaras instaladas en el equipo, este parámetro nos permite cambiar de cámara, por ejemplo: 0 para la cámara web del portátil, 1 para la cámara web externa, 2 para una segunda cámara externa, etc., -1 cuando se cuenta con una sola cámara.

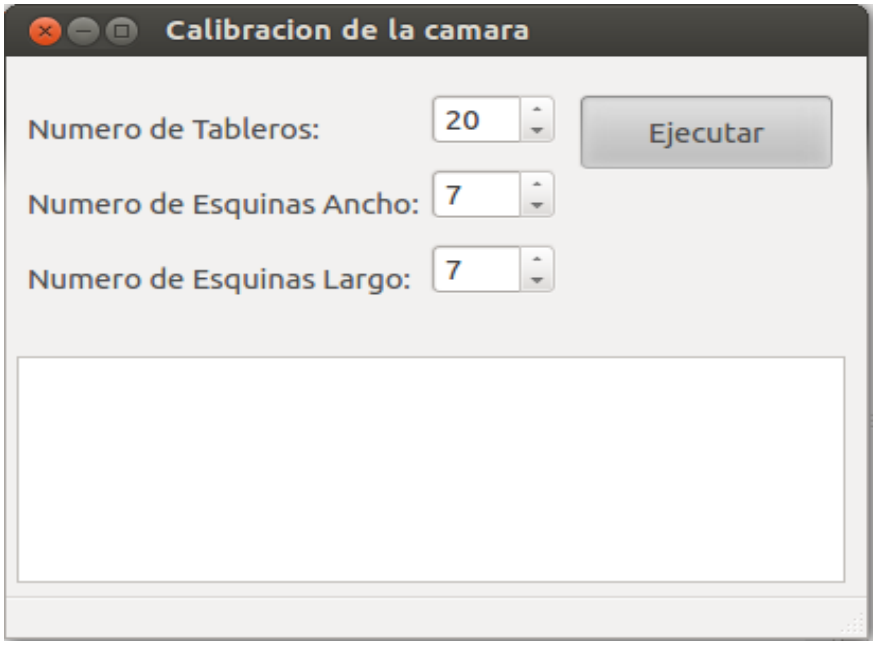
- Variables del algoritmo definidos en la interfaz de usuario:

✓ Número de tableros: Define la cantidad de imágenes o *frames* que la cámara va a tomar del tablero de ajedrez para ser usados en el proceso de calibración. Se recomienda un valor superior a 15 *frames*.

✓ Numero de Esquinas Ancho: Pueden utilizarse tableros de ajedrez no cuadrados. El número de esquinas está definido por el número de cuadrados de la primera fila-1, esto debido a que se cuentan son las esquinas internas del tablero.

✓ Numero de Esquinas Alto: El número de esquinas está definido por el número de cuadrados de la primera columna-1, esto debido a que se cuentan son las esquinas internas del tablero.

Figura 4. . Interfaz de Usuario utilizada para calibrar la cámara



Al dar click en el botón ejecutar de la interfaz de usuario se inicia un bucle de captura de video donde el algoritmo queda a la espera de que la cámara pueda ver el patrón de calibración y así detecte las esquinas internas del tablero de ajedrez que corresponden a los puntos del objeto. En la figura 17 se muestran 9 de los 20 *frames* que el programa tomó, donde la cámara detectó las esquinas internas y donde el sistema representó esta detección dibujando puntos y líneas de diferentes colores.

Para la captura de imágenes y cálculo de las coordenadas tridimensionales se desarrolló una interfaz de usuario compuesta por dos pestañas.

En la primera pestaña se elaboró la interfaz de usuario que se muestra en la figura 5. En esta pestaña se puede establecer la distancia que el sistema cámara – láser debe recorrer para escanear el objeto. También existe un Checkbox que permite escoger si se guardan o no las imágenes tomadas por la cámara.

En la pestaña “cálculo de coordenadas” (ver figura 6) se establecen los parámetros que se necesitan para calcular las coordenadas espaciales. Estos parámetros son: ángulo, corrimiento, distancia focal, umbral inferior y umbral

superior. A continuación se detallan las variables que el sistema necesita para la obtención de los valores de las coordenadas espaciales del objeto a reconstruir.

- Ángulo: Ángulo al que se encuentra el plano de luz que genera el láser con respecto a una línea horizontal que corta el lente de la cámara.
- Corrimiento: Es el cociente entre la longitud del objeto en centímetros, y el número de capturas (imágenes) tomadas por la cámara.
- Distancia cámara láser: Distancia entre el eje óptico de la cámara y el punto de giro del láser.
- Distancia Focal: Es un parámetro intrínseco de la cámara el cual se ha estimado previamente y evaluado su valor. El valor estimado de este parámetro de la cámara utilizada en este trabajo es de 3.6 milímetros.
- Umbral superior: Está definido entre 0 y 255, donde 0 corresponde a negro y 255 a blanco en una escala de grises. Esto permite filtrar las zonas más blancas, que son aquellas proyectadas por el láser.
- Umbral inferior: Está definido entre 0 y 255, donde 0 corresponde a negro y 255 a blanco en una escala de grises. Esto permite filtrar las zonas más blancas, que son aquellas proyectadas por el láser. Por ejemplo si se desea que pasen solo las tonalidades más blancas de la imagen se configura un umbral inferior de 230 y uno superior de 255.

Al dar click en el botón comenzar de la interfaz de usuario, comienzan los procesos descritos en este proyecto. Al culminar la ejecución de todos los algoritmos enlazados a la interfaz de usuario se obtienen las coordenadas espaciales buscadas. Estas coordenadas se calculan para cada imagen esqueletizada. La información que contiene cada imagen es el *perfil* del objeto en ese instante de captura. Por cada *perfil* se obtienen N puntos con coordenadas tridimensionales (X, Y, Z) y se cuenta con M *perfiles* del objeto a reconstruir, esto se traduce en que al finalizar todos los procesos se obtiene una nube de puntos con coordenadas espaciales del objeto.

Figura 5. Interfaz de usuario para captura de imágenes

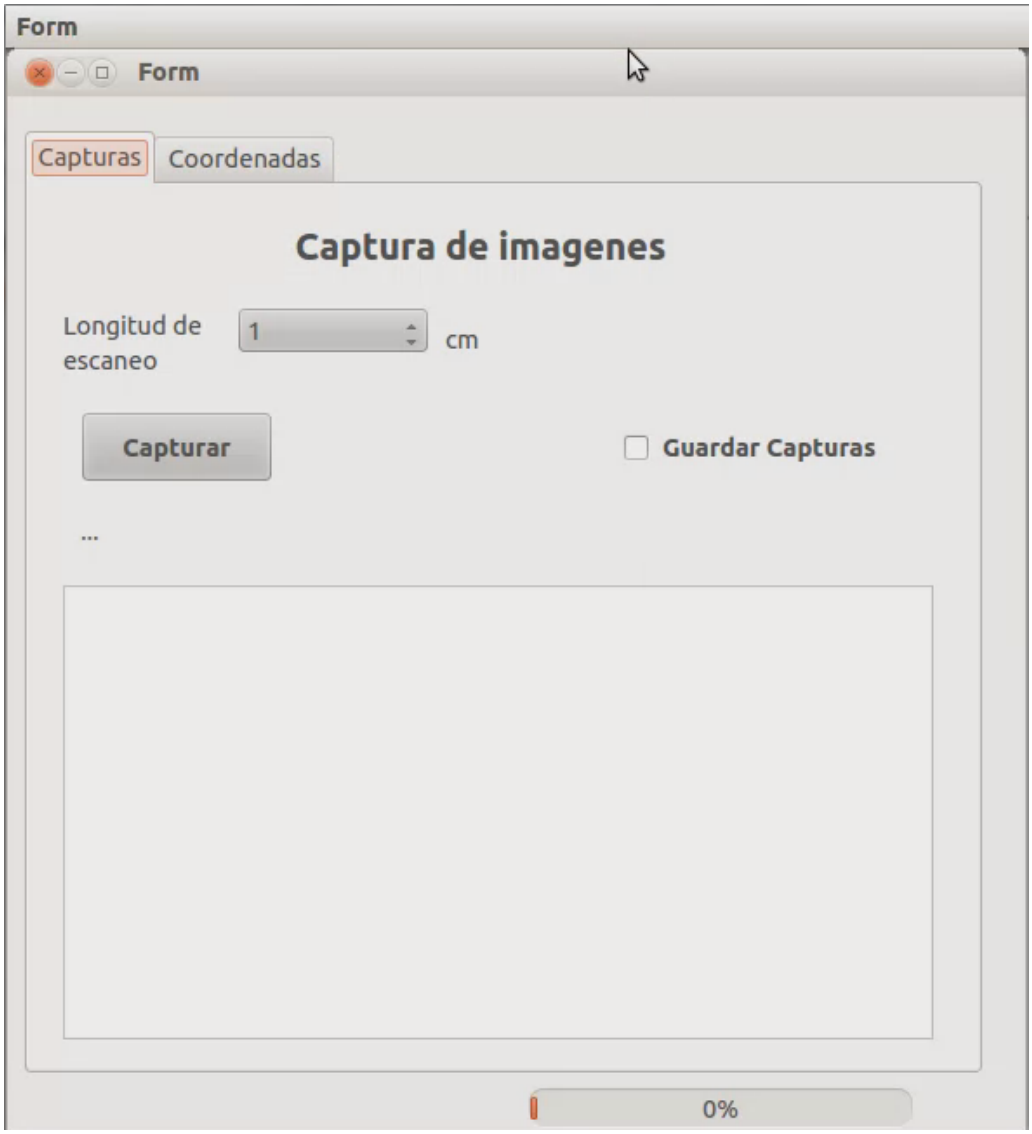
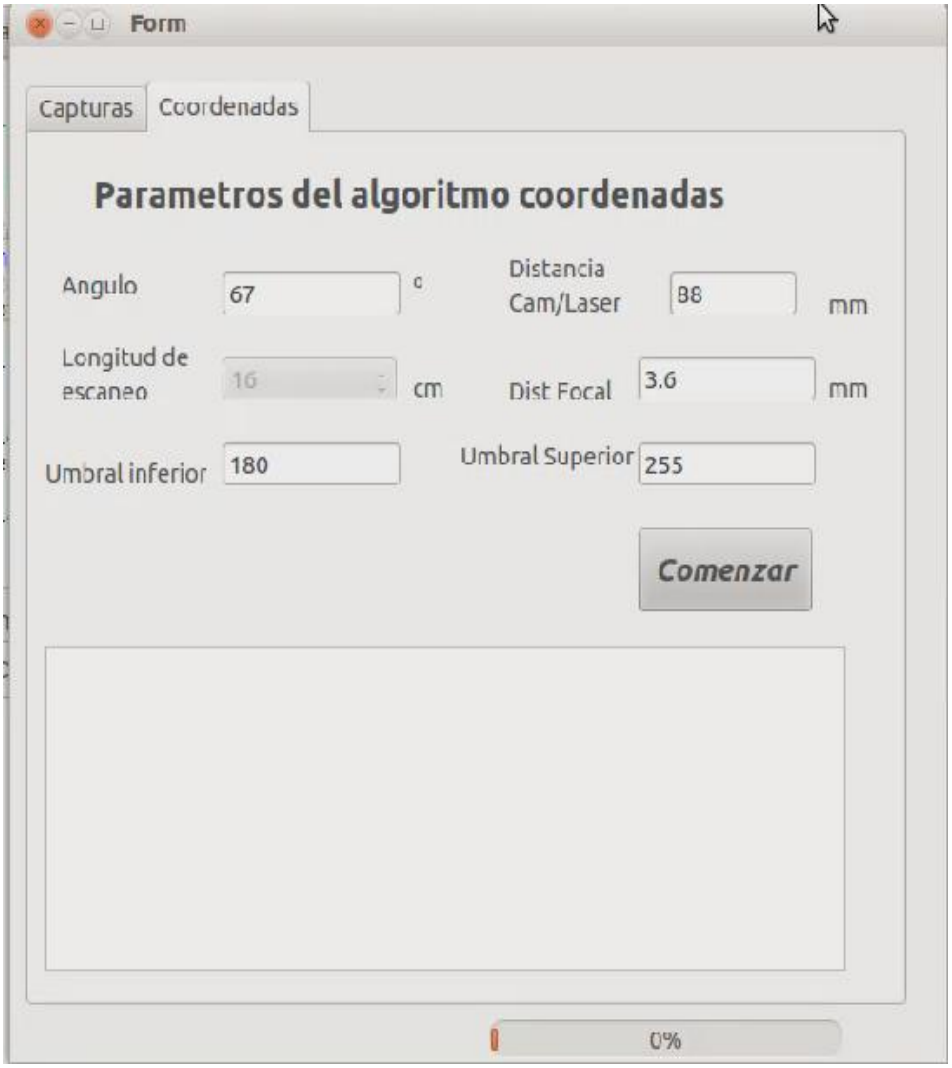


Figura 6. Interfaz de usuario para el cálculo de coordenadas espaciales



D.3 Descripción De Las Funciones Implementadas En Los Algoritmos

❖ Algoritmos De Calibración De La Cámara

Para la calibración de la cámara se desarrollaron dos algoritmos, uno para estimar los parámetros intrínsecos de la cámara y otro para rectificar las imágenes que tienen algún tipo de distorsión, tangencial y/o radial. Se recuerda que la librería de visión artificial utilizada es OpenCV. Esta librería ya cuenta con funciones

desarrolladas para realizar la estimación de los parámetros y la rectificación de las imágenes. Las funciones que se utilizaron son:

- Para la estimación de los parámetros de la Cámara.

✓ FindChessboardCorners: Esta función encuentra las esquinas internas (puntos) del tablero de ajedrez (patrón de calibración). Recibe como parámetros: la imagen del tablero de ajedrez capturada por la cámara, el tamaño de la imagen y el número de esquinas internas del tablero de ajedrez. Devuelve un vector con los valores de las coordenadas de las esquinas detectadas transformadas a unidades de pixeles.

✓ DrawChessboardCorners: Esta función dibuja en pantalla círculos de colores que encierran las esquinas internas del tablero de ajedrez y líneas de colores que unen estas esquinas. Esto permite verificar de manera gráfica si dichas coordenadas han sido correctamente detectadas. Recibe como parámetros: la imagen del tablero de ajedrez capturada por la cámara, el tamaño de la imagen, el vector de coordenadas obtenido de la función FindChessboardCorners. Devuelve un valor booleano, uno si no se han producido errores en la detección de las esquinas, 0 en cualquier otro caso.

✓ CalibrateCamera2: Esta función realiza la estimación de los parámetros de la cámara. Recibe como parámetros: la matriz de los valores de las coordenadas espaciales del tablero de ajedrez definidas por el programador, el vector de puntos arrojado por la función FindChessboardCorners y la cantidad de fotogramas capturados por la cámara. Retorna una matriz con los parámetros intrínsecos de la cámara, un vector con los coeficientes de distorsión radial y/o tangencial, un vector con los valores de las coordenadas de posición del objeto respecto a la cámara y una matriz de orientación del objeto respecto a la cámara.

- Para la rectificación de las imágenes.

✓ InitUndistortMap: Esta función realiza la rectificación de los parámetros intrínsecos estimados por la función CalibrateCamera2: Recibe como parámetros: la matriz que contiene los parámetros intrínsecos estimados, el vector que contiene los coeficientes de distorsión. Devuelve un mapa de puntos rectificados en la posición horizontal y un mapa de puntos rectificados en la posición vertical.

- ✓ Remap: Esta función realiza una transformación geométrica a la imagen distorsionada. Es la que se encarga de rectificar la imagen. Recibe como parámetros: una copia de la imagen distorsionada, la captura actual de la imagen y los dos mapas que devuelve la función InitUndistortMap.

Los algoritmos funcionan en el siguiente orden:

- Se definen todas las variables que los algoritmos necesitan tales como: variables de almacenamiento de información, contadores, parámetros de las funciones.
- Se utiliza la función FindChessboardCorners.
- Se utiliza la función DrawChessboardCorners.
- Se utiliza la función CalibrateCamera2.
- Se utiliza la función InitUndistortMap.
- Se utiliza la función Remap.

El resultado de la implementación de estos algoritmos es la buena estimación de los parámetros de la cámara y la rectificación de las imágenes que presentan problemas de distorsión.

❖ Algoritmo De Procesamiento De Imágenes

Para el procesamiento de las imágenes se desarrolló un algoritmo que contiene las funciones necesarias que garantizan la información adecuada para el cálculo de las coordenadas espaciales del objeto a reconstruir. Algunas de estas funciones ya están desarrolladas en la librería OpenCV, las que no están desarrolladas se crearon. Las operaciones que se realizaron con este algoritmo son: cálculo del histograma, umbralización, segmentación y esqueletización. Las funciones que se utilizaron son:

- Definidas en OpenCV

✓ CalcHist: Esta función calcula el número de veces que se repite un valor dentro de un rango establecido. Recibe como parámetros: la imagen a la que se le realiza la operación, uno de los canales RGB, una máscara que permite inicializar los valores de la función, el tamaño de la imagen, el rango en el cual se quiere obtener valores y el tamaño del vector que guarda los valores (este tamaño es equivalente a la diferencia del rango).

✓ Threshold: Esta función realiza la umbralización de la imagen que se está trabajando. Recibe como parámetros: la imagen que se está trabajando en uno de los canales RGB y el rango de umbralización.

✓ MorphologyEx: Esta función realiza el proceso de clasificación. Más específicamente se encarga de eliminar los falsos positivos y falsos negativos de la imagen. Los falsos positivos son píxeles blancos que no hacen parte de la información de la imagen, por lo tanto la función los elimina (los vuelve de color negro). Los falsos negativos son píxeles negros que hacen parte de la información de la imagen, por lo tanto la función los adiciona (los vuelve de color blanco). Recibe como parámetros: la imagen a procesar, el tipo de método que se va a realizar, en este caso se realiza una dilatación y luego una erosión. (para detalles de estos métodos consultar la sección 1.4.3 del capítulo 1).

- Creadas en este trabajo

✓ Eskel: Esta función realiza el esqueletizado de las imágenes que han sufrido las operaciones anteriores. El algoritmo que soporta esta técnica es el de Zhang Suen. Recibe como parámetros: la imagen después de aplicarle la función MorphologyEx. Devuelve la imagen esqueletizada.

El algoritmo funciona en el siguiente orden:

- Se definen todas las variables que el algoritmo necesita tales como: variables de almacenamiento de información, contadores, parámetros de las funciones.

- Se utiliza la función CalcHist.

- Se utiliza la función Threshold.

- Se utiliza la función MorphologyEx.

- Se utiliza la función Eskel.

El resultado que arroja la finalización de este algoritmo es la obtención de los *perfiles* de todas las capturas con un grosor de un pixel. Esto permite tener un solo punto para cada posición de los *perfiles* evitando tener redundancia de información en la misma posición.

❖ Algoritmo Para La Obtención De Coordenadas 3D

Estos algoritmos son los que permiten obtener las coordenadas espaciales del objeto a reconstruir, partiendo de las coordenadas de los N puntos de los M *perfiles*. Todas las funciones dentro de estos algoritmos fueron creadas por los desarrolladores de este trabajo. A continuación se detalla cada una de ellas.

- CoordenadasUV: Esta función es la que realiza el cálculo de las coordenadas de los N puntos para los M *perfiles*. Para calcular estas coordenadas, la función en primer lugar, recorre el fotograma (imagen) evaluado, en este recorrido verifica la posición actual y evalúa si esa posición tiene un valor de 255 que corresponde a un pixel blanco, si se cumple la condición se calcula la distancia a la que se encuentra esa posición del centro de la imagen, que idealmente debe coincidir con el punto principal, que es el parámetro intrínseco de la cámara estimado. Por el contrario si no se cumple la condición se sigue recorriendo la imagen. Recibe como parámetros: la imagen esqueletizada. Devuelve un vector con ordenadas de la imagen en unidades de píxeles.

- CoordenadasEspaciales: Esta función realiza el cálculo de las coordenadas espaciales (3D) del objeto a reconstruir. En esta función se emplea la técnica de triangulación laser (ver sección 1.1) para obtener el valor de cada coordenada (X, Y, Z). Recibe como parámetros: el vector de coordenadas de la imagen. Devuelve un archivo con los N x M puntos calculados.

Al aplicar este algoritmo se obtiene una nube de puntos con los valores de las coordenadas (X, Y, Z), las cuales son suministradas a un sistema de renderización para reconstruir el objeto trabajado.

D.4 Algoritmos Implementados

D.4.1 Interfaz de usuario para la calibración de la cámara

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file
'CalibracionDesigner.ui'
#
# Created: Fri Oct  5 18:15:48 2012
#      by: Edilson Quiñonez, Daniel Victoria
#
# WARNING! All changes made in this file will be lost!

from PyQt4 import QtCore, QtGui

try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    _fromUtf8 = lambda s: s

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName(_fromUtf8("MainWindow"))
        MainWindow.resize(406, 310)
        self.centralwidget = QtGui.QWidget(MainWindow)
        self.centralwidget.setObjectName(_fromUtf8("centralwidget"))
        self.btnEjecutar = QtGui.QPushButton(self.centralwidget)
        self.btnEjecutar.setGeometry(QtCore.QRect(270, 20, 121, 41))
        self.btnEjecutar.setMouseTracking(False)
        icon = QtGui.QIcon()

icon.addPixmap(QtGui.QPixmap(_fromUtf8(":/Icon/set_security_question.p
ng")), QtGui.QIcon.Normal, QtGui.QIcon.Off)
        self.btnEjecutar.setIcon(icon)
        self.btnEjecutar.setIconSize(QtCore.QSize(32, 32))
        self.btnEjecutar.setCheckable(True)
        self.btnEjecutar.setChecked(True)
        self.btnEjecutar.setAutoRepeat(False)
        self.btnEjecutar.setObjectName(_fromUtf8("btnEjecutar"))
        self.label = QtGui.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(10, 30, 141, 17))
        self.label.setObjectName(_fromUtf8("label"))
        self.label_3 = QtGui.QLabel(self.centralwidget)
        self.label_3.setGeometry(QtCore.QRect(10, 70, 191, 17))
        self.label_3.setObjectName(_fromUtf8("label_3"))
        self.label_4 = QtGui.QLabel(self.centralwidget)
        self.label_4.setGeometry(QtCore.QRect(10, 110, 191, 17))
        self.label_4.setObjectName(_fromUtf8("label_4"))
```

```
        self.textSalida = QtGui.QTextBrowser(self.centralwidget)
        self.textSalida.setGeometry(QtCore.QRect(5, 160, 391, 121))
        self.textSalida.setObjectName(_fromUtf8("textSalida"))
        self.txtNumTableros = QtGui.QSpinBox(self.centralwidget)
        self.txtNumTableros.setGeometry(QtCore.QRect(200, 20, 60, 27))
        self.txtNumTableros.setProperty("value", 0)
        self.txtNumTableros.setObjectName(_fromUtf8("txtNumTableros"))
        self.txtNumEsquinasAncho = QtGui.QSpinBox(self.centralwidget)
        self.txtNumEsquinasAncho.setGeometry(QtCore.QRect(200, 60, 60,
27))
        self.txtNumEsquinasAncho.setProperty("value", 0)

self.txtNumEsquinasAncho.setObjectName(_fromUtf8("txtNumEsquinasAncho"
))
        self.txtNumEsquinasLargo = QtGui.QSpinBox(self.centralwidget)
        self.txtNumEsquinasLargo.setGeometry(QtCore.QRect(200, 100,
60, 27))
        self.txtNumEsquinasLargo.setProperty("value", 0)

self.txtNumEsquinasLargo.setObjectName(_fromUtf8("txtNumEsquinasLargo"
))
        MainWindow.setCentralWidget(self.centralwidget)
        self.statusbar = QtGui.QStatusBar(MainWindow)
        self.statusbar.setObjectName(_fromUtf8("statusbar"))
        MainWindow.setStatusBar(self.statusbar)
        self.actionEjecutar = QtGui.QAction(MainWindow)
        self.actionEjecutar.setCheckable(True)
        self.actionEjecutar.setChecked(True)
        self.actionEjecutar.setObjectName(_fromUtf8("actionEjecutar"))

        self.retranslateUi(MainWindow)
        QtCore.QObject.connect(self.btnEjecutar,
QtCore.SIGNAL(_fromUtf8("clicked()")), MainWindow.ejecutarCalibracion)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):

MainWindow.setWindowTitle(QtGui.QApplication.translate("MainWindow",
"Calibracion de la camara", None, QtGui.QApplication.UnicodeUTF8))

self.btnEjecutar.setText(QtGui.QApplication.translate("MainWindow",
"Ejecutar", None, QtGui.QApplication.UnicodeUTF8))
        self.label.setText(QtGui.QApplication.translate("MainWindow",
"Numero de Tableros:", None, QtGui.QApplication.UnicodeUTF8))

self.label_3.setText(QtGui.QApplication.translate("MainWindow",
"Numero de Esquinas Ancho:", None, QtGui.QApplication.UnicodeUTF8))

self.label_4.setText(QtGui.QApplication.translate("MainWindow",
"Numero de Esquinas Largo:", None, QtGui.QApplication.UnicodeUTF8))
```

```
self.actionEjecutar.setText(QtGui.QApplication.translate("MainWindow",
"Ejecutar", None, QtGui.QApplication.UnicodeUTF8))

self.actionEjecutar.setToolTip(QtGui.QApplication.translate("MainWindo
w", "EjecutarCalibracion", None, QtGui.QApplication.UnicodeUTF8))

#import iconos_rc
```

D.4.2 Algoritmo de calibración de la cámara

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
Created: Fri Oct  5 18:15:48 2012
#      by: Edilson Quiñonez, Daniel Victoria

import cv,time,cv2
from GUIcalibracion import *
import sys

class myForm(QtGui.QMainWindow):
    """clase para el formulario de calibracion"""

    def __init__(self, parent=None):
        QtGui.QWidget.__init__(self,parent)
        self.ui=Ui_MainWindow()
        self.ui.setupUi(self)

        self.ui.txtNumEsquinasAncho.setValue(7)
        self.ui.txtNumEsquinasLargo.setValue(7)
        self.ui.txtNumTableros.setValue(20)
#        self.ui.txtTiempoFrame.setValue(2)


    def ejecutarCalibracion(self):
        """Funcion de calibraciÃłn de la cÃĄmara"""
        #---------------------------------------------------------------
---------------------#

#########################################################################
##############
        #DefiniciÃłn de variables
        mensaje=""
        board_w=self.ui.txtNumEsquinasAncho.value()
#numero de esquinas horisontales
        board_h=self.ui.txtNumEsquinasLargo.value()
#numero de esquinas verticales
        n_boards=self.ui.txtNumTableros.value()
#Cantidad de tableros
        board_n=board_w*board_h          #Total de esquinas
        board_sz=(board_w,board_h)          #tamaÃąo del tablero
```

```
#########################################################################
############
        #-----------------------------------------------------------
---------------------#

        mensaje+="Programa en ejecucion...\n\n"
        self.ui.textSalida.setText(mensaje)

        #-----------------------------------------------------------
---------------------#

#########################################################################
##############
        #AsignaciÃłn de Memoria para los Arrays donde se guardarÃĄn
los
        #datos calculados posteriormente: matriz de parÃĄmetros
intrÃ-nsecos de
        #la cÃĄmara, coeficientes de distorciÃłn, etc.
        image_points=cv.CreateMat(n_boards*board_n,2,cv.CV_32FC1)
#Matriz de 49*2 tipo Fltante que guarda los puntos de la imagen
        object_points=cv.CreateMat(n_boards*board_n,3,cv.CV_32FC1)
#Matriz de 49*3 tipo Flotante que guarda los puntos del objeto
        point_counts=cv.CreateMat(n_boards,1,cv.CV_32SC1)
#Matriz de n*1 tipo Flotante, donde n son la antidad de fotogramas
tomados
        intrinsic_matrix=cv.CreateMat(3,3,cv.CV_32FC1)
#Matriz de 3*3 tipo Fltante que guarda los parametros intrinsecos
        distortion_coefficient=cv.CreateMat(5,1,cv.CV_32FC1)
#Matriz de 5*1 tipo Fltante vector que guarda los coeficientes de
distoricion

#########################################################################
##############
        #-----------------------------------------------------------
---------------------#

        #-----------------------------------------------------------
---------------------#

#########################################################################
##############
        #SecciÃłn donde se capturan los frames del video, Se calculan
las cooredenadas de las
        #esquinas del tablero de ajedres por cada fotograma tomado y
se guardan dichas coordenadas en las
        #respectivas matricez definidas anteriormente.
        i=0             #Variable definida como contador para recorrer
las matrices image_points2 y object_points2
        z=0          #Variable definida como contador para imprimir
el nÃşmero de fotogramas
```

```
        successes=0    #Variable definida como contador para revasar
la cantidad de fotogramas que se van a capturar
        capture=cv.CaptureFromCAM(-1)     #Inicializa la captura de
video por medio de la cĂĄmara


                                #Crea una Imagen del frame capturado

        #Capturando el nÚmero requerido de vistas
        #Se realiza un while para obtener las coordenadas de las
esquinas de todas las vistas tomadas
        tecla=1;
        while(successes<n_boards and tecla!=27):
#Se capturan n_boards vistas (fotos) del video
            found=0
            image=cv.QueryFrame(capture)
            gray_image=cv.CreateImage(cv.GetSize(image),8,1)
              #se crea una matriz con tamaÑo del frame y profundidad
8 que es la cantidad de bits que se necesitan para representar el
color de un pixel (2expn)
            cv.CvtColor(image,gray_image,cv.CV_BGR2GRAY)       #Se
transforma a escala de grises la imagen obtenida del frame

            #Se obtienen las esquinas del tablero de ajedres pasando
la imagen en escala de grises, el tamano del tablero y unos filtros
            #Retorna un uno si encontrÃł todas las esquinas y el
vector con las coordenadas de las esquinas


(found,corners)=cv.FindChessboardCorners(gray_image,board_sz,cv.CV_CAL
IB_CB_ADAPTIVE_THRESH + cv.CV_CALIB_CB_FILTER_QUADS +
cv2.CALIB_CB_FAST_CHECK)

            #Calcula de manera mas precisa la coordenadas de las
esquinas del tablero de ajedres utilizando el metodo del gradiente
            #La funcion termina cuando alcance 30 iteracciones o
cuando la precision sea de 0.1
            cv.SaveImage("imagen.png {0}".format(z+1),image)



            #Dibuja las esquicas del tablero si found es igual a 1
            if found==1:

corners=cv.FindCornerSubPix(gray_image,corners,(11,11),(-1,-
1),(cv.CV_TERMCRIT_EPS + cv.CV_TERMCRIT_ITER,30,0.1))
                print "Fotograma encontrado numero {0}".format(z+1)
                mensaje+="Fotograma encontrado numero "+str(z+1)+ "\n"
                self.ui.textSalida.setText(mensaje)

                cv.DrawChessboardCorners(image,board_sz,corners,found)
#Dibuja las esquinas del tablero de ajedres
```

```
                corner_count=len(corners)    #se asigna la lingitud de
el array corners que tiene 7x7 esquinas
                z=z+1                        #Se incrementa el
contador de los fotogramas tomados

            #Si el total de esquinas son 49 se obtuvo una buena imagen
entonces los valores de las esquinas se guardan
            #en las matrices image_points y object_points...
            if len(corners)==board_n:
#igual a 49
                step=successes*board_n
#step tendra valores que son multiplos de 49. Va a guardar multiplos
de 49
                k=step
#por cada incremento de successes se incrementara en multiplos de 49
                for j in range(board_n):                             #Se
recorren longitudes de tamaÃąo 49
                    cv.Set2D(image_points,k,0,corners[j][0])       #Se
asignan los valores de la primera columna del array que contiene las
coordenadas de las esquinas
                    cv.Set2D(image_points,k,1,corners[j][1])       #Se
asignan los valores de la segunda columna del array que contiene las
coordenadas de las esquinas

cv.Set2D(object_points,k,0,float(j)/float(board_w))  #Se llena la
primera columna de la matriz objec_points con los cocientes de la
division j/7

cv.Set2D(object_points,k,1,float(j)%float(board_w))  #Se llena la
segunda columna de la matriz objec_points con los residuos de la
division j/7
                    cv.Set2D(object_points,k,2,0.0)
#Se llena la tercera columna de la matriz objec_points con ceros
                    k=k+1

                cv.Set2D(point_counts,successes,0,board_n)         #Se
llena este vector de tamaÃąo 10*1 de valores 49
                successes=successes+1                              #Se
incrementa hasta que sea menor a la cantidad de frames tomados
                time.sleep(3)                                      #Se
esperan dos segundos
                print "----------------------------------------------
-- \n"
                mensaje+="-----------------------------------------
----- \n\n"
                self.ui.textSalida.setText(mensaje)
            cv.ShowImage("Frame de Prueba",image)
#Se muestra el frame capturado
            tecla=cv.WaitKey(15)                                  #Se
retarda cada 0.1 segundos
```

```
#####################################################################
##############
        #-----------------------------------------------------------
--------------------#

        #-----------------------------------------------------------
--------------------#

#####################################################################
##############
        #Se destruye la ventana
        print "Chequeando si todo esta bien ,Todas las matrices han
sido parametrizadas"
        mensaje+="Chequeando si todo esta bien ,Todas las matrices han
sido parametrizadas \n\n"
        self.ui.textSalida.setText(mensaje)
        cv.DestroyWindow("Frame de Prueba")

#####################################################################
##############
        #-----------------------------------------------------------
--------------------#

        #-----------------------------------------------------------
--------------------#

#####################################################################
##############
        #Se crean copias de las matrices object_points, image_points y
points_counts
        object_points2=cv.CreateMat(successes*board_n,3,cv.CV_32FC1)
#Se crea una copia de la matriz object_points
        image_points2=cv.CreateMat(successes*board_n,2,cv.CV_32FC1)
#Se crea una copia de la matriz image_points
        point_counts2=cv.CreateMat(successes,1,cv.CV_32SC1)
#Se crea una copia de la matriz points_counts

#####################################################################
##############
        #-----------------------------------------------------------
--------------------#

        #-----------------------------------------------------------
--------------------#

#####################################################################
##############
        #Se transfieren los valores de las anteriores matrices a las
nuevas
        for i in range(successes*board_n):
#Se recorre la longitud successes*board_n
```

```
            cv.Set2D(image_points2,i,0,cv.Get2D(image_points,i,0))
#Se asignan los valores a la matriz image_points2
            cv.Set2D(image_points2,i,1,cv.Get2D(image_points,i,1))
#Se asignan los valores a la matriz image_points2
            cv.Set2D(object_points2,i,0,cv.Get2D(object_points,i,0))
#Se asignan los valores a la matriz object_points2
            cv.Set2D(object_points2,i,1,cv.Get2D(object_points,i,1))
#Se asignan los valores a la matriz object_points2
            cv.Set2D(object_points2,i,2,cv.Get2D(object_points,i,2))
#Se asignan los valores a la matriz object_points2
        for i in range(successes):
            cv.Set2D(point_counts2,i,0,cv.Get2D(point_counts,i,0))
#Se asignan los valores a la matriz point_counts2

########################################################################
##############
        #---------------------------------------------------------------
--------------------#

        #---------------------------------------------------------------
--------------------#

########################################################################
##############
        #Se manipula la matriz de parametros intrinsecos para que se
pueda guardar la distancia focal
        cv.Set2D(intrinsic_matrix,0,0,1.0)  #Se asigna 1 a la posicion
1,1 y 2,2 de la matriz 3x3, esto para garantizar
        cv.Set2D(intrinsic_matrix,1,1,1.0)  #que se puedan guardar los
valores de la distanci focal en estas dos posiciones

########################################################################
##############
        #---------------------------------------------------------------
--------------------#

        #---------------------------------------------------------------
--------------------#

########################################################################
##############
        #Se definen las matrices que guardarĂĄn los parametros
extrinsecos de la camara
        rvector = cv.CreateMat(n_boards, 3, cv.CV_64FC1)  #Se define
la matriz de rotacion
        tvector = cv.CreateMat(n_boards, 3, cv.CV_64FC1)  #Se define
la mtriz de traslacion

########################################################################
##############
        #---------------------------------------------------------------
--------------------#
```

```
        #-------------------------------------------------------------
--------------------#

##########################################################################
##############
        print "Verificando la calibraciÃłn de la cÃĄmara............."
        mensaje+="Verificando la calibraciÃłn de la
cÃĄmara.............. \n\n"
        self.ui.textSalida.setText(mensaje)
        # Se calculan los parametros intrÃ-nsecos y los coeficientes
de distorcion...

cv.CalibrateCamera2(object_points2,image_points2,point_counts2,cv.GetS
ize(image),intrinsic_matrix,distortion_coefficient,rvector,tvector,0)
        print " VerificaciÃłn de la calibraciÃłn de la
cÃĄmara...................Correcta "
        mensaje+="Verificacion de la calibracion de la
camara...................Correcta \n\n"
        self.ui.textSalida.setText(mensaje)

##########################################################################
##############
        #-------------------------------------------------------------
--------------------#

        #-------------------------------------------------------------
--------------------#

##########################################################################
##############
        #Se guardan los resultados: "ParÃĄmetros intrÃ-nsecos y
extrÃ-nsecos" en archivos con extensiÃłn xml
        cv.Save("Intrinsics.xml",intrinsic_matrix)
        cv.Save("Distortion.xml",distortion_coefficient)
        cv.Save("MzRotacion.xml",rvector)
        cv.Save("VtTraslacion.xml",tvector)


##########################################################################
##############
        #-------------------------------------------------------------
--------------------#

        #-------------------------------------------------------------
--------------------#

##########################################################################
##############
        # Se leen los datos de los archivos con extensiÃłn xml
        intrinsic = cv.Load("Intrinsics.xml")
        distortion = cv.Load("Distortion.xml")
```

```
        print " Todos los parametros de distorcion han sido
parametrizados"
        mensaje+="Todos los parametros de distorcion han sido
parametrizados \n"
        self.ui.textSalida.setText(mensaje)

########################################################################
##############
        #----------------------------------------------------------------
--------------------#

        #----------------------------------------------------------------
--------------------#

########################################################################
##############
        #Se crean las matrices de mapeo de los parametros intrinsecos
de calibracion...
        mapx = cv.CreateImage( cv.GetSize(image), cv.IPL_DEPTH_32F, 1
);
        mapy = cv.CreateImage( cv.GetSize(image), cv.IPL_DEPTH_32F, 1
);
        #Calcula una nueva imagen con los parametros intrÃ-nsecos
rectificados y sin distorsiÃłn
        cv.InitUndistortMap(intrinsic,distortion,mapx,mapy)
        cv.NamedWindow( "Sin distorcion" )
        print "Todo el mapeo completado"
        mensaje+="Todo el mapeo completado \n\n"
        self.ui.textSalida.setText(mensaje)
        time.sleep(2)         #Se espera 8 segundos

########################################################################
##############
        #----------------------------------------------------------------
--------------------#

        #----------------------------------------------------------------
--------------------#

########################################################################
##############
        #En esta secciÃłn se captura un nuevo frame
        print "La camara esta encendida"
        mensaje+="La camara esta encendida \n"
        self.ui.textSalida.setText(mensaje)

        tiempo =1
        while(tiempo):
            image=cv.QueryFrame(capture)          #Crea una imagen del
frame capturado
            t = cv.CloneImage(image);             #Se clona el frame
capturado
```

```
            cv.ShowImage( "Calibration", image )  #Se muestra la
ventana de CalibraciÃłn
            cv.Remap( t, image, mapx, mapy )      #Se aplica una
transformaciÃłn geomÃŠtrica genÃŠrica a la imagen
            cv.ShowImage("Sin distorcion", image)      #Se muestra la
ventana Sin distorcion
            c = cv.WaitKey(33)
            if(c == 27):                          #Presionar la tecla
"p" para detener el tiempo
                #cv.WaitKey(2000)
                tiempo=0                  #Esperar 2000 milisegundos
        #    elif c== 27 or c == 10:               #Presionar la tecla
esc para salir
        #        break

        print "TODO ESTÃ HECHO"
        mensaje+="TODO ESTÃ HECHO \n"
        self.ui.textSalida.setText(mensaje)


        cv.DestroyWindow("Sin distorcion")
        cv.DestroyWindow("Calibration")

########################################################################
##############
        #----------------------------------------------------------
--------------------#


#Procedimiento principal

app= QtGui.QApplication(sys.argv)
myapp= myForm()
myapp.show()
sys.exit(app.exec_())
```

D.4.3 Interfaz de usuario para captura de imágenes y cálculo de coordenadas espaciales

```
# -*- coding: utf-8 -*-
"""
Created on Fri Oct  5 20:01:17 2012

@author: Edilson Quiñonez, Daniel Victoria

"""
#from PyQt4 import QtCore, QtGui
from GUI3D import *
#from frmCaptura import *
```

```
import frmCaptura
import time
import serial
import cv
import sys
import esqueleto_Holt
import Coordenadas_Imagen
import glob
from ModuloClasificacion import Clasificar



class myForm(QtGui.QMainWindow):
    def __init__(self, parent=None):
        QtGui.QWidget.__init__(self,parent)
        self.ui=Ui_menu()
        self.ui.setupUi(self)
#         self.ui.txtDistancia.setText("83")
#         self.ui.txtUmbralInf.setText("160")
#         self.ui.txtUmbralSup.setText("255")
#         self.ui.txtFoco.setText("3.6")


    def comenzar(self):

        self.ui.progressBar.setValue(0)
        umbralInferior=int(self.ui.txtUmbralInf.text())
        umbralSuperior=int(self.ui.txtUmbralSup.text())
        distancia=float(self.ui.txtDistanciaCamLaser.text())
        angulo=float(self.ui.txtAngulo.text())

        foco=float(self.ui.txtFoco.text())

        #algoritmo de clasificacion
        self.ui.txtTerminal1.setText("clasificando imagenes...")
        Clasificar(umbralInferior,umbralSuperior)
        self.ui.txtTerminal1.setText("imganes clasificadas")


        #Algoritmo de esqueletizado
        lista=glob.glob("imagenes/*.png")
        numeroImagenes=len(lista)

corrimiento=(float(self.ui.cbxDistEscaneo1.currentText())*10)/numeroIm
agenes

        print corrimiento
        self.ui.progressBar.setValue(60)
        self.ui.lbProgresBar.setText("Esqueletizando...
"+str(numeroImagenes) +" Imagenes")
        esqueleto_Holt.start(umbralInferior,umbralSuperior)
        self.ui.progressBar.setValue(85)
```

```
        #Algoritmo de calculo de Coordenadas XYZ
        self.ui.lbProgresBar.setText("Calculo de coordenadas
espaciales...")
        Coordenadas_Imagen.start(distancia,angulo,foco,corrimiento)
        self.ui.progressBar.setValue(100)
        self.ui.lbProgresBar.setText("Operacion terminada con exito")


    def capturas(self):
        try:
            arduino = serial.Serial('/dev/ttyACM0', 9600)

        except:
            print 'Error, puerto no encontrado... \n vuelva a conectar
el cable de datos'
            pass


        print "inicializando..."
        time.sleep(2) # waiting the initialization...

        pasosMotor=str(int(self.ui.cbxDistEscaneo1.currentText())*48)
        print pasosMotor
        arduino.write(pasosMotor+'\n') # turns LED ON
        print "termino de escribir"

        cv.NamedWindow("captura",cv.CV_WINDOW_FULLSCREEN)
        capture=cv.CaptureFromCAM(1)

        #time.sleep(1) # waits for 1 second

        print "iniciando..."

        estado= self.ui.checkBox.checkState()
        self.ui.checkBox.enabledChange=False
        z=0
        salir ='inicio'
        while(salir != 'end'):


            imagen=cv.QueryFrame(capture)

            if estado ==2:
                cv.SaveImage("imagenes/imagen_%03d"%z+".png",imagen)
                z+=1

            try:
                salir=arduino.readline()
                arduino.timeout=.01;
                salir=salir.split('\r')[0]
```

```
            except:
                salir='end'
                pass

            if salir:
                print '---------'
                print "lectura: ",salir


            cv.ShowImage("captura",imagen)
            k=cv.WaitKey(5)



        self.ui.checkBox.enabledChange=True
        time.sleep(1) # waits for 1 s   :height: 100px
        print "Terminado"
        arduino.close() #say goodbye to Arduino
        cv.DestroyWindow("captura")

    def otro(self,valor):
        self.ui.progressBar.setValue(valor)

    def imprimir(self):
        print "jajajajaj asi es la vida"

    def cambioCbxLongitud(self):

self.ui.cbxDistEscaneo2.setItemText(0,self.ui.cbxDistEscaneo1.currentT
ext())

        pasosMotor=(int(self.ui.cbxDistEscaneo1.currentText())*48)
        print "pasosMotor: ", pasosMotor
        self.ui.txtTerminal1.setText("pasos motor = "+str(pasosMotor))


if __name__ == "__main__":
    app= QtGui.QApplication(sys.argv)
    myapp= myForm()
    myapp.show()
    sys.exit(app.exec_())
```

D.4.4 Cálculo del histograma

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 28 22:04:33 2012
@author: Edilson Quiñonez, Daniel Victoria
"""

import cv2
import numpy as np
import BarChar


imagenColor=cv2.imread("imagenes/bafle17.png")
imagenGray=cv2.imread("tarro_101.png",0)
canal = np.zeros(imagenGray.shape,np.uint8)
imagen=cv2.split(imagenColor,canal)

print np.size(imagen[0])
hist=cv2.calcHist([imagen[0]],channels=[0],mask=np.ones(imagen[0].shap
e,dtype=np.uint8),histSize=[217],ranges=[40,257])

lista=[]

cont=40
for a in hist:

        if cont%50==0:
            lista.append([str(cont),a[0]])
        else:
            lista.append(["",a[0]])

        cont+=1


BarChar.dibujaBarras(lista)
#imagen=cv2.imread("chart.png")
cv2.imshow("rojo",imagen[0])


cv2.waitKey(0)
cv2.destroyAllWindows()
```

D.4.5 Módulo de segmentación

```
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 15 01:03:57 2013
```

```
@author: Edilson Quiñonez, Daniel Victoria
"""
from multiprocessing import Pool
import Etiquetado
import glob,cv2
import numpy as np

def Clasificar(umbralInf=100,umbralSup=255):
    """CLASIFICA LAS IMAGENES,ELIMINANDO EL RUIDO"""

    numProcesos=0
    lista = glob.glob("imagenes/*.png")
    lista.sort()

    pool = Pool(processes=len(lista))
    element= cv2.getStructuringElement(cv2.MORPH_CROSS, (3,3))

    for foto in lista:

        imagenColor=cv2.imread(foto)
        imagenGray=cv2.imread(foto,0)
        canal = np.zeros(imagenGray.shape,np.uint8)
        #obtiene los canales de la imagen
        imagen=cv2.split(imagenColor,canal)


        ret,imagen = cv2.threshold(imagen[0],umbralInf,umbralSup,0)

cv2.imwrite("umbralizadas/umbral%03d"%numProcesos+".png",imagen)

        imagen = cv2.morphologyEx(imagen,cv2.MORPH_DILATE,element)
        imagen = cv2.morphologyEx(imagen,cv2.MORPH_ERODE,element)




pool.apply_async(Etiquetado.clasificacion,(imagen,numProcesos))
        numProcesos+=1

    pool.close()
    pool.join()
    pool.terminate()
```

D.4.6 Algoritmo de segmentación

```
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 12 21:19:08 2013

@author: Edilson Quiñonez, Daniel Victoria
```

```
"""
"""
Scrit creado para eliminar el ruido de las imagenes.
"""

#import cv2
import numpy as np,cv2
def clasificacion(imagen,numeroPrecesos):

    #definicion de variables
    posicionFilas=[]
    posicionColumnas=[]
    numeroPixeles =0
    maxI = 0
    maxJ = 0
    minI = 0
    minJ = 0

    posiciones=np.where(imagen==255)
    posiciones=np.transpose(posiciones)

    #se obtiene el anchoy alto de la imagen
    filas,columnas=np.shape(imagen)
    for posicion in posiciones:
            i=posicion[0]
            j=posicion[1]
            if i <= filas-4 and j<=columnas-2:
                zona =imagen[i:i+4,j:j+2]
                numeroPixeles=np.count_nonzero(zona)
                if numeroPixeles >=8:
                    posicionFilas.append(i)
                    posicionColumnas.append(j)
#                else:
#
cv2.imwrite("clasificadas/clasificada%03d"%numeroPrecesos+".png",image
n)
    if len(posicionFilas)!=0 and len(posicionColumnas)!=0:

            #se calculan las posiciones maximas y minimas
            maxI = np.max(posicionFilas)
            maxJ = np.max(posicionColumnas)
            minI = np.min(posicionFilas)
            minJ = np.min(posicionColumnas)

#    #se vuelven a calcular las posiciones garantizando
#    #que se cumpla el criterio especificado
#    while(minI < 20):
#            posicionFilas.remove(minI)
#            minI=np.min(posicionFilas)
#
#    while(minJ < 400):
#            posicionColumnas.remove(minJ)
```

```
#            minJ=np.min(posicionColumnas)
#
#    while(maxI > 400):
#            posicionFilas.remove(maxI)
#            maxI=np.max(posicionFilas)
#
#    while(maxJ > 500):
#            posicionColumnas.remove(maxJ)
#            maxJ=np.max(posicionColumnas)

        #Elimina el ruido de la imagen...
    nposiciones=np.where(imagen!=0)
    nposiciones=np.transpose(nposiciones)

    for nposicion in nposiciones:
            x=nposicion[0]
            y=nposicion[1]
            if (x < minI-5 or x > maxI+5) or (y<minJ-5 or y > maxJ+5):
                imagen[x,y]=0


cv2.imwrite("clasificadas/clasificada%03d"%numeroPrecesos+".png",image
n)
```

D.4.7 Módulo de esqueletización

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 18 14:40:38 2012

@author: Edilson Quiñonez, Daniel Victoria
"""
import numpy as np
import cv2


def eskel2(imagen,ciclo):
    """adelgaza la imagen"""



    esqueleto=imagen.copy()
    esqueleto2=imagen.copy()
    final=True
    iteraciones=0

zonaSecuencia=[[0,1],[0,2],[1,2],[2,2],[2,1],[2,0],[1,0],[0,0],[0,1]]
    while final==True:
        final=False
        iteraciones=iteraciones+1
        posicion=np.where(imagen==255)
```

```
        posicion=np.transpose(posicion)
        rowMax,columMax=np.shape(imagen)

        #PRIMER RECORRIDO
        for k in posicion:
           i=k[0]
           j=k[1]

           if i != 0 and i !=rowMax-1 and j != 0 and j!=columMax-1:
               zona=imagen[i-1:i+2,j-1:j+2]
#               print i,j

               N=(zona[0][1]==0)           #Norte
               E=(zona[1][2]==0)           #Este
               S=(zona[2][1]==0)           #Sur
               if N==True or E==True or S==True:
                   O=(zona[1][0]==0)           #Oeste
                   if O==True or S==True or E==True:

                           numVecinos=(np.sum(zona)/255)-1
                               #si el numero de vecinos esta entre 2
y 6
                           if 2<=numVecinos and numVecinos<=6:

                               valorPixelAnterior=255
                               #contVecinos=0
                               contContinuidad=0
                               valorActual=0

                               #si la conectividad es 1
                               for p in zonaSecuencia:
                                   valorActual=zona[p[0]][p[1]]
                                   if valorActual ==255:
                                       #contVecinos=contVecinos+1
                                       if valorPixelAnterior==0:

contContinuidad=contContinuidad+1
                                   valorPixelAnterior=valorActual

        #                           if(zona[0][1]==255):
        #                               contVecinos=contVecinos-1

                               if contContinuidad==1:
                                   esqueleto[i][j]=0
                                   esqueleto2[i][j]=150
                                   final=True



        #SEGUNDO RECORRIDO
        imagen=esqueleto.copy()
```

```
        posicion=np.where(imagen==255)
        posicion=np.transpose(posicion)
        for k in posicion:
           i=k[0]
           j=k[1]

           if i != 0 and i !=rowMax-1 and j != 0 and j!=columMax-1:
               zona=imagen[i-1:i+2,j-1:j+2]
#               print zona

               N=(zona[0][1]==0)           #Norte
               E=(zona[1][2]==0)           #Este
               O=(zona[1][0]==0)           #Oeste

               if N==True or E==True or O==True:
                   S=(zona[2][1]==0)           #Sur
                   if O==True or S==True or N==True:
                            numVecinos=(np.sum(zona)/255)-1
                                #si el numero de vecinos esta entre 2
y 6
                            if 2<=numVecinos and numVecinos<=6:

                                valorPixelAnterior=255
                                #contVecinos=0
                                contContinuidad=0
                                valorActual=0

                                #si la conectividad es 1
                                for p in zonaSecuencia:
                                    valorActual=zona[p[0]][p[1]]
                                    if valorActual ==255:
                                        #contVecinos=contVecinos+1
                                        if valorPixelAnterior==0:

contContinuidad=contContinuidad+1
                                    valorPixelAnterior=valorActual

        #                           if(zona[0][1]==255):
        #                               contVecinos=contVecinos-1

                                if contContinuidad==1:
                                    esqueleto[i][j]=0
                                    esqueleto2[i][j]=150
                                    final=True




        imagen=esqueleto.copy()

#    print "consulta: "+str(ciclo)
```

```
    cv2.imwrite("esqueletos/esqueleto_%03d"%ciclo+".png",esqueleto)
    imagen=esqueleto2.copy()
    return imagen
```

D.4.8 Algoritmo de esqueletización

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 14 09:23:09 2012

@author: Edilson Quiñonez, Daniel Victoria
"""

import cv2
import numpy as np
import PosicionImagen
from multiprocessing import *
import getopt
import glob
from frmLienzo import *
import sys
from PyQt4 import *

def start(umbralInf=180,umbralSup=255):
    """esqueletiza las imagenes del directorio /imagenes y
      las guarda en el directorio /esqueletizadas, con un umbral
      por defecto umbralInf=180 y umbralSup=255"""
#     __init__(self)

    lista=glob.glob("clasificadas/*.png")

    print str(len(lista))
    numProcesos=0
    pool=Pool(processes=len(lista))
    lista.sort()

#    element= cv2.getStructuringElement(cv2.MORPH_CROSS, (3,3))



    for foto in lista:


#        imagenColor=cv2.imread(foto)
#        imagenGray=cv2.imread(foto,0)
#        canal = np.zeros(imagenGray.shape,np.uint8)
#        #obtiene los canales de la imagen
```

```
#         imagen=cv2.split(imagenColor,canal)
#
#
#         ret,imagen[0] =
cv2.threshold(imagen[0],umbralInf,umbralSup,0)
#
cv2.imwrite("umbralizadas/umbral%03d"%numProcesos+".png",imagen[0])
        #close = cv2.morphologyEx(imagen[0],cv2.MORPH_CLOSE,element)

        imagen=cv2.imread(foto,0)
        pool.apply_async(PosicionImagen.eskel2,(imagen,numProcesos))
        numProcesos+=1


    pool.close()
    pool.join()
    pool.terminate()

    print "esqueletizacion Terminada con exito"

    #cv2.waitKey(0)
    cv2.destroyAllWindows()


#def __init__(self, parent=None):
#         QtGui.QWidget.__init__(self,parent)
#         self.ui=Ui_menu()
#
#         self.ui.setupUi(self)
```

D.4.9 Algoritmo para el cálculo de coordenadas espaciales

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 21 11:30:03 2012

@author: Edilson Quiñonez, Daniel Victoria
"""
import numpy as np
import math as operacion
#f=3.7
#b=88
#alfa=180-85.5

def
coordenadasEspaciales(imagen,nombreArchivo,corrimiento,b,alfa,f,desfas
e):
    """calcula las coordenadas espaciales"""
```

```
    steep = 0.1
    array=coordenadasUV(imagen)

    resultado=[]
    ctg=1/operacion.tan((alfa*operacion.pi)/180)

    for a in array:
        m=b/(f*ctg-a[0])
        x=-(a[0]*m)-corrimiento
        y=a[1]*m
        z=f*m
        resultado.append([x,y,z])

    taminio=len(resultado)
#    print "Este es el tamanio",taminio
#    adicion = int(desfase/steep)
#    print "Miremos a ver que tal",adicion
#    print "Esta es la adicion",adicion
#    taminio = round(taminio*adicion)
#    print "Este es el nuevo tamanio",taminio
    archi=open('coordenadas_xyz/datos_%03d'%nombreArchivo+'.ply','w')
    archi.close

    with open('coordenadas_xyz/datos_%03d'%nombreArchivo+'.ply',"a")
as archivo:

        #archi=open('datos.txt','a')
        archivo.write("ply"+"\n"+"format ascii 1.0"+"\n"+"element
vertex "+str(taminio)+"\n"+"property float x"+"\n"+"property float
y"+"\n"+"property float z"+"\n"+"end_header"+"\n")

        for txt in resultado:
            archivo.write(str(txt[0])+" "+ str(txt[1])+" "+
str(txt[2])+"\n")
#            xValue = txt[0]
#            contador=desfase
#            auxiliar=0
#            yValue=txt[1]
#            yValueMax=yValue+0.8
#            yValueMin=yValue-0.8
#            while contador > steep:
#                auxiliar +=1
#
#                if(auxiliar % 2 ==0):
#
#                    archivo.write(str(xValue)+" "+ str(yValueMax)+"
"+ str(txt[2])+"\n")
#                else: archivo.write(str(xValue)+" "+ str(yValueMin)+"
"+ str(txt[2])+"\n")
#                xValue = xValue - 0.3
#                contador = contador -steep
```

```
#           1 print "Cantidad de veces que entra", auxiliar
#            for str(txt[0] in range()
#    print "Se escribio"+ 'datos'+str(nombreArchivo)+'.ply\n'


def coordenadasUV(imagen):
    "coordenadasUV-----\n"
    #se obtiene el tamaño de la imagen
    alto,ancho =imagen.shape


    #se define el centro de la imagen
    centro_x=ancho/2
    centro_y=alto/2
    coordenadas_xy=[]

    #se guadan las coordenadas xy
    array=np.where(imagen==255)
    array=np.transpose(array)


    for k in array:
        x=k[1]
        y=k[0]
#        centros reales de la camara
#        centro_x=310
#        centro_y=253
        coordenada_x=-(x-centro_x)/213.3115
        coordenada_y=-(y-centro_y)/194.6088
        coordenadas_xy.append([coordenada_x,coordenada_y])

#    coordenadas_xy.sort
    return coordenadas_xy
```

D.4.10 Algoritmo de control del motor paso a paso por medio de tarjeta Arduino Uno.

```
/*
Control de velocidad y sentido de Motor paso a paso

El control del motor paso a paso se hace con el fin de manipular el
barrido de un laser
que se encuentra acoplada a una banda que va a ser controlar por dicho
motor.

Este programa activa el MotorPP ...
```

```
Creado 7 Agosto 2012
@author: Edilson Quiñonez, Daniel Victoria
UNIVERSIDAD DEL CAUCA
*/

#include <Stepper.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

const int pasosPorRevolucion = 200;  // numero de pasos por revolución
int EstadoBotonSentido=0; //estado del botonSentido
const int LedBarrido=3; // sentido izquierdo
const int LedInicio=4; // sentido derecha
const int FinCarrera=2; //switch fin de carrera

long int valor;
boolean stringComplete = false;  // whether the string is complete



// Inicializacion, pines de salida del 10 al 13:
Stepper myStepper(pasosPorRevolucion, 10,12,11,13);

int stepCount=0;  // numero de pasos que el motor a tomado

void setup() {

  pinMode(LedBarrido, OUTPUT);     // initialize the LedIzquierda pin
as an output:
  pinMode(LedInicio, OUTPUT);
  pinMode(FinCarrera,INPUT);


    // initialize serial:
  Serial.begin(9600);

  motorInicial();      //fija cam-laser en su posicion inicial
  digitalWrite(LedBarrido,HIGH);

}


//--------------------------------------//
//  PROCEDIMIENTO PRINCIPAL             //
//--------------------------------------//

void loop() {


  digitalWrite(LedInicio,LOW);
```

```
  //verificacion si existen datos en el bufer puerto serie
  if(Serial.available()>0){



     //lectura de datos
     serialEvent();

     // if (stringComplete==true) {
     delay(1000);
     Serial.println("start");
     delay(1);

     stepCount=0;
     while(stepCount < valor)
     {
         // read the sensor value:
        int sensorReading = analogRead(A0);
        // map it to a range from 0 to 100:
        int motorSpeed = map(sensorReading, 0, 1023, 0, 100);



        if (motorSpeed > 0) {
              digitalWrite(LedBarrido,HIGH);
              myStepper.setSpeed(motorSpeed);


              myStepper.step(1);     //gira hacia la derecha un paso
              stepCount=stepCount+1;

        }

  }
    //digitalWrite(LedBarrido,LOW);
    Serial.println("end");
    delay(3);
    motorInicial();
    valor=0;
    stringComplete = false;
    Serial.flush();
    //Serial.println("end2");
    //delay(3)
    //Serial.println("end3");

  }



}
```

```
//---------------------------------------------------------------
//metodo que ecurre cuando llegan datos por
//el puerto serial

void serialEvent() {
  int i=0;
  char inputDatos[10] = "";
  while (Serial.available()) {

    // get the new byte:
    char inChar = (char)Serial.read();
    delay(2);



    if (inChar != '\n') {
       inputDatos[i] = inChar;
       i++;

    }
    else{
      stringComplete = true;
      valor=atol(inputDatos);
   }
  }
}

//**************************************************************
//metodo para inicializar la posicion del motor pp

void motorInicial()
{

  myStepper.setSpeed(100);
  while(digitalRead(FinCarrera)==0) //mientras que cam-laser no este
en la posicion inicial
  {
      myStepper.step(-1);
      digitalWrite(LedInicio,HIGH);

  }

}
```

Anexo E: Descripción del funcionamiento del algoritmo de Zhang Suen

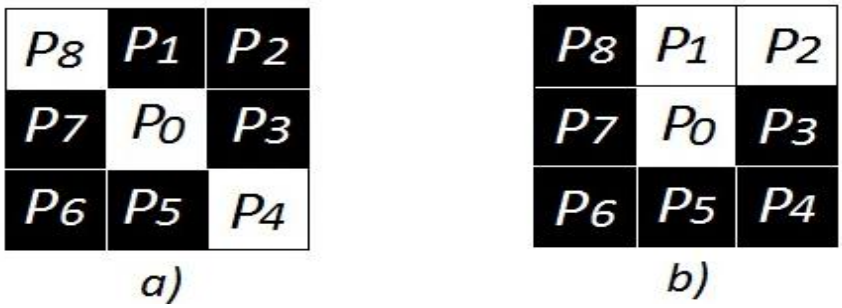
El algoritmo de Zhang Suen usa el concepto de los ocho vecinos. La máscara que utiliza es una matriz 3x3 donde sus celdas se encuentran enumeradas como se muestra en la tabla 10.

Tabla 10. Enumeración de celdas establecidas en el algoritmo de Zhang Suen

$\boldsymbol{P_8}$	$\boldsymbol{P_1}$	$\boldsymbol{P_2}$
$\boldsymbol{P_7}$	$\boldsymbol{P_0}$	$\boldsymbol{P_3}$
$\boldsymbol{P_6}$	$\boldsymbol{P_5}$	$\boldsymbol{P_4}$

Las condiciones y reglas que se mencionan para evaluar si un pixel debe ser borrado, requiere de la definición de dos funciones. Una función $A(p)$ que cuenta la cantidad de veces que se repite el patrón 1 - 0 (blanco - negro), si se recorre la matriz desde P_1 hasta volver a P_1 $(P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8)$, donde P representa el pixel evaluado. La segunda función $B(p)$ que cuenta el número de vecinos (tiene que ser uno (pixel blanco) para que se cumpla) que se encuentran alrededor del pixel evaluado. En la figura 7 se describen las funciones.

Figura 7. Zona 3x3. $\boldsymbol{a)}$ $\boldsymbol{A(p) = 2}$, $\boldsymbol{B(p) = 2}$, $\boldsymbol{b)}$ $\boldsymbol{A(p) = 1}$, $\boldsymbol{B(p) = 2}$.



Luego de que las funciones han sido determinadas, se debe pasar la máscara a través de todos los pixeles de la imagen con el fin de evaluar si éste debe ser eliminado (cambiar el pixel al color del fondo, en este caso de blanco a negro). Para que un pixel sea eliminado, éste debe cumplir con las condiciones de la tabla 9.

Tabla 11. Condiciones que se deben cumplir para que un pixel sea eliminado

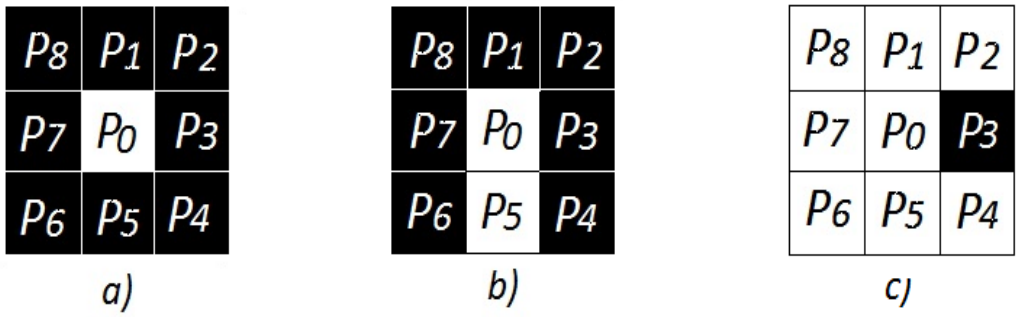
Para la primera sub iteración	***Para la segunda sub iteración***
$\mathbf{1)} \quad \mathbf{2} \leq \boldsymbol{B}(\boldsymbol{p_0}) \leq \mathbf{6}$	$\mathbf{5)} \quad \mathbf{2} \leq \boldsymbol{B}(\boldsymbol{p_0}) \leq \mathbf{6}$
$\mathbf{2)} \quad \boldsymbol{A}(\boldsymbol{p_0}) = \mathbf{1}$	$\mathbf{6)} \quad \boldsymbol{A}(\boldsymbol{p_0}) = \mathbf{1}$
$\mathbf{3)} \quad \boldsymbol{p_1} * \boldsymbol{p_3} * \boldsymbol{p_5} = \mathbf{0}$	$\mathbf{7)} \quad \boldsymbol{p_1} * \boldsymbol{p_3} * \boldsymbol{p_7} = \mathbf{0}$
$\mathbf{4)} \quad \boldsymbol{p_3} * \boldsymbol{p_5} * \boldsymbol{p_7} = \mathbf{0}$	$\mathbf{8)} \quad \boldsymbol{p_1} * \boldsymbol{p_5} * \boldsymbol{p_7} = \mathbf{0}$

Para una mejor compresión del algoritmo es necesario estudiar cada una de las condiciones mencionadas.

$\boldsymbol{Condición\ 1)}\quad \boldsymbol{2 \leq B(p_0) \leq 6}$

Esta condición evalúa que el número de vecinos del pixel esté entre 2 y 6. Esto garantiza que ningún pixel punto final sea eliminado, o sea ningún pixel que tiene como vecino un elemento del objeto de la imagen. Además de esto asegura que el pixel a ser eliminado debe ser parte del borde. La figura 8 da más claridad a lo dicho.

Figura 8. Zona 3x3. $\boldsymbol{a)}$ $\boldsymbol{B(p_0) = 0}$,) $\boldsymbol{b)}$ $\boldsymbol{B(p_0) = 1}$, c) $\boldsymbol{B(p_0) = 7}$

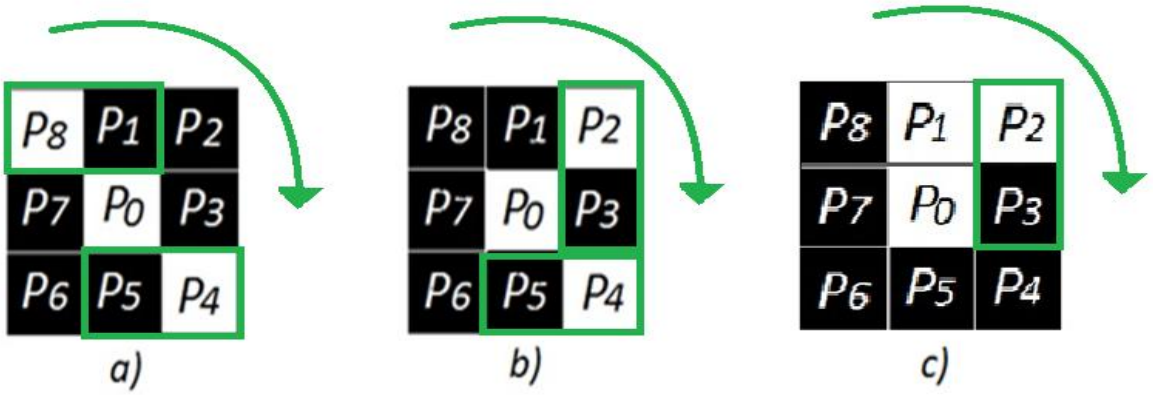


En la figura 8 (a) se observa que cuando $B(\boldsymbol{p_0}) = \boldsymbol{0}$, el pixel es un punto aislado que no hace parte del objeto a esqueletizar, por lo tanto es una clase de ruido, pero el algoritmo no se encarga de eliminarlo. En la figura 8 (b) $B(\boldsymbol{p_0}) = 1$, entonces $P_{\boldsymbol{0}}$ es un píxel punto final el cual hace parte del esqueleto, por lo tanto tampoco debe ser eliminado. En la figura 8 (c) se ve que como $B(\boldsymbol{p_0}) = 7$, $P_{\boldsymbol{0}}$ no es un pixel de borde, por lo tanto no debe ser eliminado.

$\boldsymbol{Condición\ 2)}\quad \boldsymbol{A(p_0) = 1}$

Esta condición evalúa la conectividad. La figura 9 muestra tres ejemplos donde en los dos primeros $A(p_0) > 1$ y en el tercero $A(p_0) = 1$. En los dos primeros ejemplos no se puede eliminar el pixel P_0porque si se hace el objeto queda desconectado. En el tercer ejemplo si se puede eliminar el pixel P_0.

Figura 9. . Zona 3x3. $\boldsymbol{a})\ \boldsymbol{A(p_0) = 2}$, $\boldsymbol{b})\ \boldsymbol{A(p_0) = 2}$, $\boldsymbol{c})\ \boldsymbol{A(p_0) = 1}$



Condición 3, 4, 5, 6:

$\boldsymbol{Condiciónes\ 3, 4, 5, 6}$

Las condiciones 3 y 4 evalúan si el pixel está en el borde Sur o borde Este y si pertenece a la esquita Noroeste, mientras que las condiciones 5 y 6 evalúan si el pixel se encuentra en el borde Norte u Oeste y si pertenece a la esquina Sureste, validando así que estos puntos no pertenecerán al esqueleto (ver tabla 12).

Tabla 12. Configuración de las posiciones del pixel evaluado

Norte				
Oeste	$\boldsymbol{P_8}$	$\boldsymbol{P_1}$	$\boldsymbol{P_2}$	Este
	$\boldsymbol{P_7}$	$\boldsymbol{P_0}$	$\boldsymbol{P_3}$	
	$\boldsymbol{P_6}$	$\boldsymbol{P_5}$	$\boldsymbol{P_4}$	
sur				

Anexo F: Validación del Sistema Hardware/Software implementado

La validación experimental del sistema hardware/software implementado es necesario para comprobar que el valor de las coordenadas calculadas por la técnica de reconstrucción utilizada es muy cercano al valor de las coordenadas reales de los puntos analizados.

Este proceso de validación se dividió en dos etapas: la primera consistió básicamente en comparar los valores obtenidos por el algoritmo implementado con los valores de referencia, es decir con los valores reales. Para esto se elaboró un sistema de referencia 3D con coordenadas previamente conocidas, se proyectó un punto sobre un plano paralelo al sistema cámara laser, donde además de conocer el ángulo de inclinación del punto de luz láser con respecto a la cámara, también se conoce la posición espacial donde se refleja dicho punto (coordenadas X, Y, Z reales) (ver figura 10). Una vez hecho lo anterior se procedió a realizar el cálculo de las coordenadas (X, Y, Z) mediante el algoritmo implementado y posteriormente se compararon los resultados obtenidos con los valores reales, para medir los errores presentes y validar el sistema. La segunda etapa de validación se efectuó en el agua, para comprobar que el efecto de observar un objeto sumergido en agua, desde el aire a través de un material plano y transparente como el vidrio, (agua en un acuario) consiste en la visualización de una imagen aparente respecto a la posición original del objeto, sin embargo su forma y tamaño no se ven alterados, razón por la cual al reconstruir el objeto aparente (posición e imagen vista por la cámara) conservará las mismas características que el objeto real. Para llevar a cabo esta prueba se proyectaron dos puntos en diferentes posiciones donde se conocía la distancia entre ellos, luego mediante la técnica triangulación láser se hallaron las posiciones espaciales de dichos puntos y posteriormente el cálculo de la distancia entre ellos (ver figura 11). Al final se comparan los resultados obtenidos con los esperados para su posterior análisis. Lo anterior se repitió con diferentes posiciones de los puntos proyectados.

F.1 Validación primera etapa: Cálculo de coordenadas espaciales

Esta validación es importante en el desarrollo del presente trabajo, ya que es necesario comprobar si los valores calculados corresponden o se asemejan a los que representa la realidad. Para ello se hicieron muchas tomas de puntos proyectados por el láser en el espacio y se midió el ángulo de este instrumento. Todo bajo un sistema de referencia 3D conocido, por lo cual se tienen los valores

de las coordenadas reales, las cuales son la base para determinar si los valores de las coordenadas calculadas por el algoritmo de reconstrucción se asemejan a éstas.

El cálculo del error obtenido por cada punto se determinó midiendo la distancia entre las coordenadas X_C, Y_C, Z_C calculadas y las coordenadas X_R, Y_R, Z_R reales. Lo anterior mediante la siguiente expresión matemática (ver (1)):

$$E = \sqrt{(X_R - X_C)^2 - (Y_R - Y_C)^2 - (Z_R - Z_C)^2} \quad (1)$$

Los resultados obtenidos se muestran en las tablas 13, 14, 15, 16, 17, 18 y 19, 20, y 21 para diferentes valores de Z.

Tabla 13. Resultados para una distancia Z_R de 20 cm.

X_R	Y_R	Z_R		X_C	Y_C	Z_C	Error
COORDENADAS (mm)			**ÁNGULOS (grados)**	**COORDENADAS (mm)**			**DISTANCIA (mm)**
-16	30	200	60	-14,1	29,5	198,5	2,4
23	53	200	65	22,3	51,3	198,6	2,3
23	-20	200	70	22,7	-17,7	199,0	2,6
17	2	200	68	16,2	1,2	198,0	2,3
-38	-26	200	56	-37,0	-25,3	197,5	2,8
26	-35	200	70	22,2	-33,3	197,2	5,0
63	-8	200	80	60,1	-7,5	198,2	3,5
65	28	200	80	63,3	27,8	198,1	2,6
25	35	200	71	23,8	37,0	198,0	3,1
50	45	200	75	48,6	43,0	199,2	2,6
Promedio del Error							2,9

Tabla 14. Resultados para una distancia Z_R de 25 cm

X_R	Y_R	Z_R		X_C	Y_C	Z_C	Error
COORDENADAS (mm)			**ÁNGULOS (grados)**	**COORDENADAS (mm)**			**DISTANCIA (mm)**
-22	53	250	67	-19,6	51,8	252,4	3,6
21	7	250	76	20,1	7,7	252,5	2,8
-5	8	250	70	-4,3	6,8	252,5	2,9
-49	42	250	61	-47,4	42,4	247,7	2,8
-51	43	250	60	-50,6	41,4	252,7	3,1
10	-52	250	74	11,0	-53,2	251,5	2,2
-3	38	250	67	-4,2	37,0	252,5	3,0
-13	2	250	68	-10,9	2,6	251,9	2,8
21	-26	250	77	21,3	-28,8	252,1	3,5
30	-37	250	78	28,9	-39,0	252,2	3,2
Promedio del Error							3,0

Tabla 15. Resultados para una distancia Z_R de 30 cm.

X_R	Y_R	Z_R		X_C	Y_C	Z_C	Error
COORDENADAS (mm)			**ÁNGULOS (grados)**	**COORDENADAS (mm)**			**DISTANCIA (mm)**
17	-41	300	78	16,6	-43,4	301,9	3,0
8	42	300	76	6,7	43,9	302,1	3,1
-73	36	300	64	-71,1	37,6	301,1	2,7
-73	12	300	64	-70,5	13,5	302,6	3,9
-75	7	300	62	-75,6	5,2	299,6	1,9
-87	7	300	60	-88,0	8,1	297,6	2,8
16	39	300	76	13,4	38,2	299,4	2,8
8	20	300	74	6,5	19,9	297,2	3,2
-25	20	300	69	-23,4	19,4	299,5	1,8
-26	40	300	69	-21,7	37,3	299,8	5,1
Promedio del Error							3,0

Tabla 16. Resultados para una distancia Z_R de 35 cm.

X_R	Y_R	Z_R		X_C	Y_C	Z_C	Error
COORDENADAS (mm)			**ÁNGULOS (grados)**	**COORDENADAS (mm)**			**DISTANCIA (mm)**
-13	45	350	74	-11,75	43,96	348,86	2,0
-95	48	350	60,5	-94	47	348	2,4
-115	77	350	62	-112,08	75,45	348,87	3,5
-65	45	350	66	-60,18	44,18	349,24	4,9
-68	58	350	66,5	-66,53	56,07	352,39	3,4
31	50	350	81,5	28,71	51,47	351,72	3,2
31	30	350	81,5	28,71	32,47	351,72	3,8
-83	68	350	65	-79,21	63,14	348,71	6,3
-59	63	350	68	-58,32	61,04	348,81	2,4
-48	-43	350	70	-48,32	-43,96	353,05	3,2
Promedio del Error							3,5

Tabla 17. Resultados para una distancia $\boldsymbol{Z_R}$ de 40 cm

X_R	Y_R	Z_R		X_C	Y_C	Z_C	Error
COORDENADAS (mm)			**ÁNGULOS (grados)**	**COORDENADAS (mm)**			**DISTANCIA (mm)**
-45	23	400	72	-44,2	21,6	399,4	1,7
-65	20	400	70	-63,2	22,6	398,6	3,4
-87	28	400	67	-89,3	25,9	397,5	4,0
-97	-20	400	66	-94,3	-17,2	397,5	4,6
-65	-17	400	70	-62,7	-16,1	396,5	4,3
-75	15	400	68	-72,9	13,1	399,2	2,9
-60	15	400	71	-59,0	13,7	403,5	3,9
-58	85	400	72	-55,3	88,1	402,7	4,9
-30	67	400	74	-27,5	65,3	399,3	3,1
-28	56	400	74	-27,4	55,7	398,6	1,5
Promedio del Error							3,4

Tabla 18. Resultados para una distancia Z_R de 45 cm**.**

X_R	Y_R	Z_R		X_C	Y_C	Z_C	Error
COORDENADAS (mm)			**ÁNGULOS (grados)**	**COORDENADAS (mm)**			**DISTANCIA (mm)**
-83	48	450	70	-80,3	46,0	448,9	3,5
-53	89	450	71	-55,9	87,3	449,2	3,5
-29	80	450	76	-28,1	79,5	452,0	2,3
-23	59	450	76	-21,1	57,3	450,6	2,6
-17	53	450	75	-18,0	50,0	449,0	3,3
-127	62	450	64	-128,7	63,7	452,0	3,1
-132	78	450	64	-131,0	77,9	453,0	3,2
-118	77	450	65	-117,0	75,9	451,1	1,8
-78	82	450	70	-77,8	83,4	451,9	2,3
-15	27	450	72	-14,8	26,5	453,3	3,3
Promedio del Error							2,9

Tabla 19. Resultados para una distancia Z_R de 50 cm.

X_R	Y_R	Z_R		X_C	Y_C	Z_C	Error
COORDENADAS (mm)			**ÁNGULOS (grados)**	**COORDENADAS (mm)**			**DISTANCIA (mm)**
-100	32	500	70	-101,4	32,5	498,4	2,2
-65	31	500	75	-67,0	31,1	501,3	2,4
-65	31	500	73	-64,6	30,1	502,3	2,5
-63	22	500	74	-61,6	22,6	500,6	1,7
-51	118	500	71	-48,5	117,0	500,0	2,7
-107	80	500	69	-105,7	78,6	501,4	2,4
-122	30	500	68	-120,0	31,0	499,0	2,4
-36	-8	500	77	-33,7	-7,8	502,1	3,1
-44	58	500	76	-40,0	58,8	501,8	4,5
-98	52	500	70	-94,7	51,0	498,5	3,8
Promedio del Error							2,8

Tabla 20. Resultados para una distancia Z_R de 54 cm.

X_R	Y_R	Z_R		X_C	Y_C	Z_C	Error
COORDENADAS (mm)			**ÁNGULOS (grados)**	**COORDENADAS (mm)**			**DISTANCIA (mm)**
-63	80	540	75	-61,5	79,1	540,9	2,0
-120	70	540	72	-120,7	71,3	540,9	1,7
-14	-5	540	81	-15,7	-6,3	542,7	3,4
-33	55	540	80	-35,1	55,8	543,0	3,8
-68	-10	540	75	-67,7	-8,3	543,8	4,2
-45	120	540	78	-43,2	120,3	540,1	1,9
-50	80	540	77	-54,2	78,8	542,8	5,1
-43	35	540	80	-45,5	40,9	541,4	6,6
-68	40	540	75	-66,8	39,0	541,9	2,4
-115	40	540	70	-113,5	38,6	540,4	2,1
Promedio del Error							3,3

Tabla 21. Resultados para una distancia $\boldsymbol{Z_R}$ de 60 cm

X_R	Y_R	Z_R		X_C	Y_C	Z_C	Error
COORDENADAS (mm)			**ÁNGULOS (grados)**	**COORDENADAS (mm)**			**DISTANCIA (mm)**
5	40	600	82	6,0	44,0	597,0	5,1
10	10	600	83	12,0	15,0	595,0	7,3
14	5	600	84	16,0	8,0	595,0	6,2
-10	70	600	81	-11,0	73,0	597,0	4,4
20	-10	600	85	22,0	-13,0	593,0	7,9
21	80	600	85	26,0	84,0	594,0	8,8
30	50	600	86	37,0	57,0	592,0	12,7
13	25	600	84	15,7	29,0	596,0	6,3
28	30	600	86	32,0	35,0	589,0	12,7
22	-50	600	85	24,0	-55,0	593,0	8,8
Promedio del Error							8,0

De los datos registrados en las tablas anteriores se puede constatar que los resultados obtenidos en esta primera etapa de validación arrojan un error que oscila entre 3 mm para ángulos menores o iguales a 80 grados, por el contrario, para ángulos mayores a este valor el sistema se vuelve sensible al error y no se puede garantizar unos buenos cálculos de coordenadas espaciales, ya que el error es cercano a un centímetro. Este efecto ocurre debido a que estos ángulos están muy cerca a los 90 grados lo cual implica que el cálculo de las tangentes implícitas en las fórmulas de la técnica implementada en el algoritmo tienda a infinito.

Por lo tanto se recomienda que el sistema cámara/láser esté ubicado a una distancia del objeto a reconstruir menor o igual a 30 cm.

F.2 Validación segunda etapa: Experimento en el agua

Esta validación consistió básicamente en proyectar un punto con el láser en dos posiciones espaciales en el agua donde es conocido de antemano la distancia entre los puntos, luego se procede a determinar mediante la técnica de reconstrucción las coordenadas espaciales de dichos puntos, una vez medido el ángulo del láser. Finalmente se calcula la distancia entre los puntos utilizando la ecuación (1) y se comparan con el valor de referencia. Lo anterior se repite para varios datos que se encuentran en diferentes posiciones. Los datos obtenidos de esta validación se registran en las tablas 22, 23 y 24.

Tabla 22. Resultados para una distancia de 5 cm entre dos puntos

DISTANCIA DE 5 CENTÍMETROS ENTRE DOS PUNTOS PROYECTADOS SOBRE UN PLANO QUE SE ENCUENTRA A UNA DISTANCIA X DEL SISTEMA CÁMARA - LASER								
ÁNGULOS (grados)	**COORDENADAS (mm)**			**ÁNGULOS (grados)**	**COORDENADAS (mm)**			**DISTANCIA (mm)**
69,0	-22,5	42,3	315,0	79,0	24,0	50,3	330,0	51,3
69,0	-41,2	51,9	310,0	76,0	9,1	48,8	319,4	53,2
71,0	-17,0	46,6	318,9	80,0	30,6	49,8	335,6	53,4
71,0	-20,5	48,6	319,3	80,0	29,1	51,5	323,6	48
71,0	-23,6	50,5	324,2	79,0	25,5	50,0	335,4	52,1
72,0	-22,6	60,0	345,0	78,0	26,0	45,0	330,6	52,9

69,0	-40,6	38,4	357,3	73,0	-10,5	53,3	322,1	48,6
66,0	-72,1	51,9	359,6	70,0	-30,0	48,6	324,1	55,2
64,0	-60,8	26,0	305,0	69,5	-38,3	55,2	337,7	48
66,0	-50,1	48,7	310,1	73,5	-14,4	52,8	345,7	52,3
PROMEDIO DE LA DISTANCIA ENTRE DOS PUNTOS								52

Tabla 23. Resultados para una distancia de 6 cm entre dos puntos.

DISTANCIA DE 6 CENTÍMETROS ENTRE DOS PUNTOS PROYECTADOS SOBRE UN PLANO QUE SE ENCUENTRA A UNA DISTANCIA X DEL SISTEMA CÁMARA - LASER								
ÁNGULOS (grados)	COORDENADAS (mm)			ÁNGULOS (grados)	COORDENADAS (mm)			DISTANCIA (mm)
61,5	-65,6	31,5	254,6	66,5	-38,2	73,3	290,2	60,4
61,5	-79,6	34,8	272,2	66,0	-50,1	75,7	310,1	58,1
65,5	-81,6	34,5	305,8	67,5	-33,7	72,5	293,7	60,4
64,0	-49,6	53,9	282,1	70,0	-33,5	75,7	333,7	58,3
58,0	-66,2	31,4	246,8	66,0	-38,5	74,2	284,1	58,2
58,5	-75,3	30,8	266,5	65,5	-38,2	75,3	277,0	58,9
58,0	-69,7	28,5	242,6	65,5	-33,7	68,6	267,2	59,2
59,0	-71,6	28,8	265,6	66,0	-35,4	76,5	277,1	57
55,5	-82,3	28,3	247,7	63,5	-51,3	73,8	279,4	57,5
59,0	-75,5	24,5	272,0	65,5	-34,7	71,6	269,3	59,4
59,5	-42,8	40,9	222,0	60,0	-45,0	-20,1	230,4	57,6
62,0	-68,7	45,4	294,7	62,0	-71,5	-9,9	299,9	55,6
62,0	-50,0	-13,3	259,6	62,0	-48,5	43,2	256,6	56,7
57,0	-71,8	53,6	246,0	57,0	-74,7	-9,7	250,6	57,5

67,5	-57,7	40,6	351,8	68,0	-59,4	-19,3	364,8	58
PROMEDIO DE LA DISTANCIA ENTRE DOS PUNTOS								58,2

Tabla 24. Resultados para una distancia de 7 cm entre dos puntos.

DISTANCIA DE 7 CENTÍMETROS ENTRE DOS PUNTOS PROYECTADOS SOBRE UN PLANO QUE SE ENCUENTRA A UNA DISTANCIA X DEL SISTEMA CÁMARA - LASER								
ÁNGULOS (grados)	COORDENADAS (mm)			ÁNGULOS (grados)	COORDENADAS (mm)			DISTANCIA (mm)
60,0	-77,8	32,0	287,1	68,5	-48,2	34,5	345,7	65,7
57,5	-86,9	30,2	274,5	68,5	-46,1	33,5	340,4	68,5
57,5	-83,4	28,8	269,0	69,0	-33,9	31,7	317,6	69,4
60,5	-68,1	28,6	295,6	69,0	-15,4	28,1	255,3	66,3
61,0	-65,1	28,4	296,3	70,0	-9,8	29,2	246,4	67,2
53,0	-94,7	28,4	242,5	65,0	-32,2	28,7	257,8	64,4
55,0	-86,8	28,5	249,6	67,5	-20,5	30,3	261,8	67,5
55,5	-82,3	27,2	247,7	69,0	-17,1	30,9	273,8	70,3
56,0	-79,0	27,2	247,6	68,5	-13,4	28,3	257,4	66,3
57,0	-72,9	29,0	247,8	70,5	-7,4	30,0	269,3	69,0
58,5	-88,5	27,6	278,8	69,0	-30,0	30,7	315,3	69,1
60,0	-94,9	33,0	316,7	70,0	-24,1	30,8	307,9	71,4
60,0	-85,9	32,7	301,2	71,5	-21,3	34,0	326,6	69,4
59,0	-82,4	31,6	307,5	72,0	-16,8	33,6	322,5	67,3
62,5	-72,1	31,6	307,5	74,0	-6,4	33,4	329,3	69,2
PROMEDIO DE LA DISTANCIA ENTRE DOS PUNTOS								68,1

Posteriormente se calculó el promedio del error de los datos registrados. Este promedio aparece en la tabla 25.

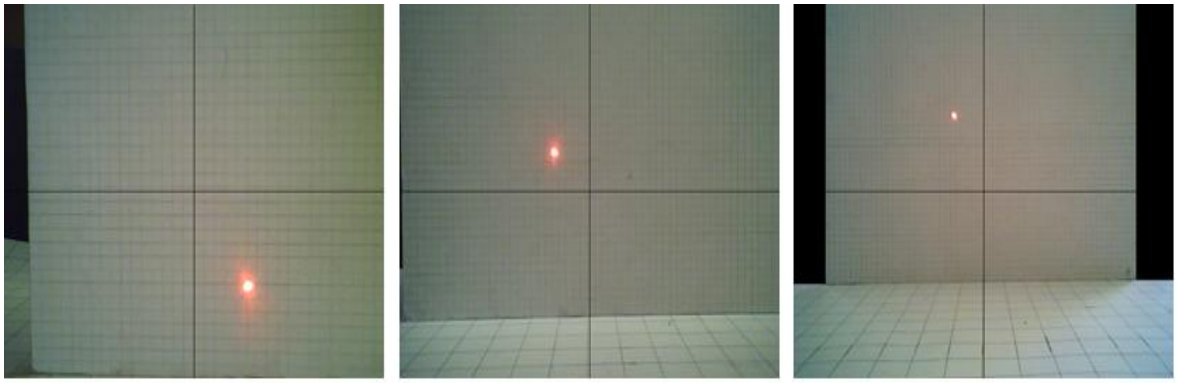
Tabla 25. Promedio de los errores

DISTANCIA REAL	DISTANCIA CALCULADA	ERROR
50	52	2
60	58,2	1,8
70	68,1	1,9
PROMEDIO DEL ERROR		1,9

Los resultados obtenidos en esta segunda etapa de validación evidencian que las distancias obtenidas por el sistema entre los puntos analizados son muy cercanas a las distancias reales definidas previamente, con un error que oscila entre dos milímetros. Con esto se ratifica que el fenómeno de refracción de la luz que ocurre entre diferentes medios (aire – vidrio - agua) trae como consecuencia que la cámara vea los objetos en una posición aparente, la cual es diferente a la posición real a la que se encuentra el objeto sumergido en el agua. No obstante la forma de este objeto es invariante.

Por lo tanto al realizar el proceso de reconstrucción 3D de la vista aparente de este objeto, se obtiene los mismos resultados que si se hiciera la reconstrucción 3D de la vista real del mismo.

Figura 10. Planos de referencia. $\boldsymbol{a}) Z = 20,\ \boldsymbol{b}) Z = 35,\ \boldsymbol{c}) Z = 40$



a) b) c)

Figura 11. Planos de referencia en el agua. $\boldsymbol{a})$ $\boldsymbol{Patrón\ diagonal},\ \boldsymbol{b})$ $\boldsymbol{Patrón\ horizontal}$ $\boldsymbol{c})$ $\boldsymbol{Patrón\ vertical}$

a) b) c)

REFERENCIAS

[1] C. Eitzinger, “Report on interfaces to relevant engineering systems and standards” [Online]. Seventh Framework Programme, Febrero 2011. Disponible en: http://darwin-project.eu/files/2010/07/D81_Report_on_interfaces1.pdf. Consultado: Agosto 2012

[2] H. López, A. Gómez, “Superficies a partir de nubes de puntos” [Online]. Computación Gráfica II, Universidad Simón Bolívar, Noviembre 2011. Disponible en: http://ldc.usb.ve/~alacruz/cursos/ci5321/SurfacePointClouds/Presentaci%C3%B3n%20Te%C3%B3rica/Informe.pdf. Consultado: Agosto 2012.

[3] I. Gómez, “Algoritmo para la construcción de grandes mallas mediante la triangulación de Delaunay”. Trabajo fin de carrera, Facultad de Informática, Universidad Politécnica de Madrid, España, Marzo 2011.

[4] G. Sánchez, S. Mateus, J. Branch, “Construcción de Mallas Triangulares no Estructuradas Aplicado al Ajuste de Superficies de Objetos Tridimensionales”. Revista Avances en Sistemas e Informática, Escuela de Sistemas, Universidad Nacional de Colombia, Sede Medellín, Vol. 1, No. 2, pp. 33-39, Diciembre, 2004.