INSTITUTO POLITECNICO NACIONAL

Centro de Investigación en Ciencia Aplicada y Tecnología Avanzada

Posgrado en Tecnología Avanzada

ESTIMACIÓN DEL VOLUMEN DE OBJETOS POR MEDIO DE RECONSTRUCCIÓN 3-DIMENSIONAL

TESIS

QUE PARA OBTENER EL GRADO DE
MAESTRO EN TECNOLOGÍA AVANZADA

PRESENTA

Gabriel González Flores

Director de Tesis

Dr. Eduardo Castillo Castañeda

SIP-14

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de Querétaro, Qro siendo las 12:00 horas del día 30 del mes de mayo del 2011 se reunieron los miembros de la Comisión Revisora de Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación de CICATA – Unidad Qro para examinar la tesis titulada:

" ESTIMACIÓN DEL VOLUMEN DE OBJETOS POR MEDIO DE RECONSTRUCCIÓN 3-DIMENSIONAL "

Presentada por el alumno:

GONZÁLEZ	**FLORES**	**GABRIEL**
Apellido paterno	Apellido materno	Nombre(s)

Con registro:

A	0	9	0	0	7	6

aspirante de:

Maestría en Tecnología Avanzada

Después de intercambiar opiniones, los miembros de la Comisión manifestaron ***APROBAR LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Director de tesis

Dr. Eduardo Castillo Castañeda

Dr. José Joel González Barbosa

Dr. Francisco Javier Ornelas Rodríguez

Dr. Eduardo Morales Sánchez

Dr. Jorge Adalberto Huerta Ruelas

PRESIDENTE DEL COLEGIO DE PROFESORES

Dr. Jorge Adalberto Huerta Ruelas

INSTITUTO POLITECNICO NACIONAL
CENTRO DE INVESTIGACIÓN EN CIENCIA APLICADA Y TECNOLOGÍA AVANZADA
UNIDAD QUERÉTARO
DIRECCIÓN

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de Querétaro, Qro el día 30 del mes de mayo del año 2011, el que suscribe Gabriel González Flores alumno del Programa de Maestría en Tecnología Avanzada con número de registro A090076, adscrito a CICATA – Unidad Querétaro, IPN, manifiesta que es autor intelectual del presente trabajo de Tesis bajo la dirección de Eduardo Castillo Castañeda y cede los derechos del trabajo intitulado **Estimación del volumen de objetos por medio de reconstrucción 3-dimensional**, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección CICATA – Unidad Querétaro, IPN, Cerro Blanco #141, Colinas del Cimatario, C.P. 76090, Querétaro, Qro., México. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Gabriel González Flores

Estimación del volumen de objetos por medio de Reconstrucción 3-Dimensional

Gabriel González Flores

Centro de Investigación en Ciencia Aplicada y Tecnología Avanzada

Instituto Politécnico Nacional

Querétaro, México.

Agradecimientos

A Dios, por darme la vida, salud, y por permitirme la culminación de una meta más en mi vida.

A mis padres y mis hermanos, por que han infundido en mí el deseo de superación y sobre todo por el gran apoyo que he recibido de ellos a lo largo de mi vida.

A los profesores del CICATA, que dedicaron su tiempo para transmitirme parte de sus conocimientos y me ayudaron a adquirir las herramientas necesarias para la culminación de este trabajo.

A todos mis compañeros del CICATA, que de manera directa ó indirecta colaboraron en mi formación personal y profesional.

Al CONACyT por el apoyo económico durante el desarrollo de este proyecto.

Contenido

Índice de figuras

Índice de tablas

Resumen

En este trabajo se muestra el desarrollo de un prototipo mecatrónico novedoso capaz de estimar el volumen de objetos sólidos de forma irregular, que hace uso de técnicas de visión por computadora. Dicho prototipo presenta un sistema de visión por computadora enfocado a la realización de mediciones de alta precisión.

El prototipo presenta ventajas sobre las técnicas convencionales de estimación del volumen de objetos (por cortes, por principio de Arquímedes, por medición con contacto, entre otros), ya que se obtiene el volumen del objeto mediante una técnica de medición no destructiva y sin contacto.

La obtención del volumen de objetos irregulares, está basada en la reconstrucción tridimensional mediante visión por computadora. La metodología con la cual se desarrolló dicho prototipo se compone principalmente de dos partes. En la primera, se realiza el procesamiento de la imagen, cuyo objetivo es determinar las siluetas del objeto de sus diferentes vistas, esto implica la calibración de la cámara. La segunda parte tiene como objetivo crear el modelo 3D del objeto a partir de nube de puntos que se genera con las siluetas obtenidas. Una vez obtenida la nube de puntos total, se genera un mallado 3D aplicando la triangulación de Delaunay. Considerando la calibración de la cámara y el número total de puntos de la nube, se puede determinar el volumen del objeto.

Finalmente, para implementar esta metodología se construyó un dispositivo mecánico con tres grados de movilidad, que permite, el montaje del sistema de adquisición de la imagen, y el giro del objeto sobre su eje. El algoritmo para el procesamiento de imágenes y la reconstrucción tridimensional se llevó a cabo utilizando un lenguaje de alto nivel. Los resultados obtenidos del proceso de reconstrucción son modelados en un ambiente virtual GEOMAGIC para obtener una visualización más realista del objeto.

Abstract

This work presents the development of an innovative mechatronic prototype able to estimate the volume of irregular objects, which uses computer vision techniques. The prototype presents a computer vision system focused on the realization of high precision measurements.

The prototype has advantages over conventional techniques for estimating the volume of objects (for slides, Archimedes' principle, by contact techniques, etc.), since the volume of the object is obtained from a nondestructive measurement technique without contact.

Volume measurement of irregular objects is based on three-dimensional reconstruction by computer vision. The methodology used to developed this prototype is mainly composed of two parts. The first carries out the image processing, which is to determine the silhouettes of the object of their different views, this implies the calibration of the camera. The second part is to create the 3D model of the object from point cloud generated with the silhouettes obtained. Once obtained the total point cloud, it creates a 3D mesh using Delaunay triangulation. Given the camera calibration and the total number of points in the cloud, one can determine the volume of the object.

Finally, to implement this methodology we built a mechanical device with three degrees of mobility, which allows the system installation image acquisition, and the rotation of the object around its axis. The algorithm for image processing and three-dimensional reconstruction was carried out using a high-level language. The results of the reconstruction process are displayed using a virtual environment GEOMAGIC to get a more realistic view of the object.

Glosario

Magnitud: propiedad de un fenómeno, de un cuerpo o de una sustancia que se puede expresar mediante un número y una referencia.

Sistema Internacional de Magnitudes: sistema de magnitudes con base en las siete magnitudes de base: longitud, masa, tiempo, corriente eléctrica, temperatura termodinámica, cantidad de sustancia e intensidad luminosa.

Medición: proceso que consiste en obtener experimentalmente uno o varios valores que puedan atribuirse razonablemente a una magnitud.

Metrología: ciencia de la medición y sus aplicaciones. NOTA. La metrología incluye todos los aspectos teóricos y prácticos de las mediciones, cualquiera que sea la incertidumbre de la medida y del campo de aplicación.

Mensurando: magnitud propuesta para medirse.

Método de medición: descripción genérica de una organización lógica de operaciones realizadas en una medición.

Procedimiento de medición: descripción detallada de una medición de acuerdo a uno o más principios de medida y a un método de medida dado, con base en un modelo de medida y que incluye los cálculos para obtener un resultado de medición.

Resultado de medición: conjunto de valores de una magnitud atribuidos a un mensurando acompañados de cualquier otra información relevante disponible.

Valor medido: valor de una magnitud que representa un resultado de medición.

Exactitud: proximidad de concordancia entre un valor medido de la magnitud y un valor verdadero del mensurando.

Precisión: proximidad de concordancia entre indicaciones o valores medidos obtenidos por mediciones repetidas de un mismo objeto, o de objetos similares, bajo condiciones especificadas.

Error: diferencia entre un valor medido de una magnitud y un valor de referencia.

Resolución: la más pequeña variación de la magnitud medida que produce una variación perceptible de la indicación correspondiente.

Patrón de medida: realización de la definición de una magnitud dada, con un valor de la magnitud determinado y la incertidumbre de medida respectiva, usada como referencia.

Volumen: El volumen es una magnitud escalar definida como el espacio ocupado por un cuerpo. Es una función derivada ya que se halla multiplicando las tres dimensiones.

Reconstrucción tridimensional: es el proceso mediante el cual, los objetos reales son reproducidos en la memoria de una computadora, manteniendo sus características físicas de dimensión, volumen y forma.

Silueta: es una vista de cierto objeto o escena que consiste en el esquema y un interior sin rasgos distintivos, con la silueta generalmente siendo negra. Forma que presenta a la vista la masa de un objeto más oscuro que el fondo sobre el cual se proyecta.

Calibración de la cámara: implica el diseño de un modelo matemático para estimar los parámetros extrínsecos e intrínsecos de la cámara dadas imágenes de un patrón de calibración, y su precisión depende de que tan eficazmente ubiquemos los puntos en el espacio y los puntos de referencia de la cámara.

Parámetros intrínsecos: estos caracterizan las propiedades ópticas, geométricas y digitales de la visión de la cámara que son necesarias para unir las coordenadas en píxeles de un punto imagen con las correspondientes coordenadas en el marco de la cámara.

Parámetros extrínsecos: estos identifican la orientación y posición de la cámara con respecto a las coordenadas del mundo real.

Pixel: es la menor unidad homogénea en color que forma parte de una imagen digital, ya sea esta una fotografía, un fotograma de vídeo o un gráfico.

Voxel: es la unidad cúbica que compone un objeto tridimensional. Constituye la unidad mínima procesable de una matriz tridimensional y es, por tanto, el equivalente del píxel en un objeto 2D.

Homografía: toda transformación proyectiva que determina una correspondencia entre dos figuras geométricas planas, de forma que a cada uno de los puntos y las rectas de una de ellas le corresponden, respectivamente, un punto y una recta de la otra.

Nomenclatura

2D	: bidimensional
3D	: tridimensional
Kg	: unidad de masa
m^3	: unidad de volumen, metros cúbicos
ρ_f	: es la densidad del fluido
V	: el volumen del cuerpo sumergido
g	: la aceleración de la gravedad
E	: empuje hidrostático
L	: unidad de volumen (litros)
R	: matriz de rotación
T	: vector de traslación
A	: matriz de los parámetros intrínsecos de la cámara
cc	: centro del eje óptico o centro de la imagen
f , fc	: distancia focal
(u_0, v_0)	: coordenadas del centro de la imagen
α , β	: se derivan del largo focal y tamaño del píxel
γ , alpha_c	: asimetría del pixel
Kc	: coeficientes de distorsión
$[X , Y , Z]^t$	: coordenadas del mundo real
$[u , v]^t$	: coordenadas de una imagen dado en pixeles
(sy_i, sz_i)	: coordenadas del i-ésimo pixel de una silueta
S_i	: nube de puntos en 2D de la i-ésima silueta
d_{eg-cs}	: distancia entre el eje de giro de la base y el centro de la silueta
d_{eg-co}	: distancia entre el eje de giro de la base y el centro del objeto
R_L^S	: matriz de rotación de la webcam superior con respecto a la webcam lateral
T_L^S	: es el vector de traslación de la webcam superior con respecto a la webcam lateral
X_S	: vector de coordenadas $[X , Y , Z]^t$ del mundo real con respecto al plano de la imagen de la webcam superior
X_T	: vector de coordenadas $[X , Y , Z]^t$ del mundo real con respecto al plano de la imagen de la webcam lateral
r	: radio de la esfera en pixeles
d	: diámetro de la esfera en pixeles
vol_{esf}	: volumen de la esfera en vóxeles
VSVS	: volumen de la esfera sin aplicar la vista superior

VCVS : volumen de la esfera aplicando la vista superior

EVSVS : error del volumen estimado de la esfera sin aplicar la vista superior

EVCVS : error del volumen estimado de la esfera aplicando la vista superior

Ac : área del circulo con radio r

Err_{dig}: Error de digitalización

Capítulo 1. Introducción general

1.1 Introducción

La visión por computadora es una división de la inteligencia artificial, que tiene como propósito programar una computadora para que "entienda" una escena o las características de una imagen. Actualmente se desarrollan trabajos de investigación para dar solución a problemas en la industria, aunque en ocasiones no es la mejor solución a un problema, ya que el problema es tan complejo que la solución humana es lo mejor. Pero a veces, las soluciones humanas tienden a ser inexactas o subjetivas y en ocasiones lentas y presentan una ausencia de rigor así como una pobre percepción.

Dentro de las aplicaciones en la industria de la visión por computadora se tienen dos casos. El primer caso engloba todas aquellas aplicaciones en las que el único sensor presente es el de visión. Mientras que el segundo caso se refiere a los sistemas multisensoriales tal como los equipos de navegación en robótica donde la visión constituye una capacidad sensorial más, para la percepción del entorno que rodea al robot.

Las aplicaciones de la visión por computadora son muy variadas, e incluyen aspectos como el reconocimiento de caracteres, reconstrucción tridimensional, automatización de manufactura, detección de blancos, análisis de imágenes biomédicas, inspección automática o medición remota. Actualmente se cuenta con una variedad de empresas que se dedican a implementar sistemas de visión artificial con un fin industrial, como se pueden mostrar en sus respectivas páginas web (JasVisio, 2010), (VIEWTECH, 2010), (NAVITAR MACHINE, 2010).

Por tal motivo, la inspección se refiere a la verificación de si un objeto cumple con determinados criterios. Lo cual implica comparar el objeto con algún objeto modelo que describe las características relevantes del objeto. Existiendo tolerancias definidas dentro de las cuales las medidas realizadas pueden considerarse como aceptables.

En un sistema de inspección y control de calidad, se encuentra una sección dedicada a la inspección geométrica tridimensional, que está enfocada procesos industriales de producción, en donde se requiere que los productos manufacturados sean inspeccionados para asegurar que algunas medidas de calidad y fiabilidad se cumplen.

Por lo tanto para resolver el problema de la inspección geométrica tridimensional, se hace uso de la reconstrucción tridimensional (aplicación de la visión por computadora), que es el proceso mediante el cual objetos son reproducidos en la memoria de una computadora, manteniendo sus características físicas (dimensiones, volumen, forma). Actualmente se han desarrollado una diversidad de prototipos que determinan la reconstrucción 3D de un objeto.

En este trabajo de tesis se realiza la inspección de objetos sólidos con la finalidad de determinar el volumen del objeto en análisis. Para realizar esta tarea se estudian deferentes métodos para determinar dicha característica, entre esos métodos se encuentran mediciones por contacto (principio de Arquímedes, por cortes) y sin contacto (ópticos).

1.2 Objetivos

1.2.1 Objetivo general

Diseñar y construir un sistema mecatrónico que mida el volumen de objetos usando la reconstrucción tridimensional de una secuencia de imágenes.

1.2.2 Objetivos específicos

- Definir el sistema de estimación del volumen de objetos a partir de una reconstrucción tridimensional del objeto.
- Diseñar y construir la plataforma de la base experimental donde va colocado el objeto.
- Implementar el algoritmo de reconstrucción tridimensional para estimar el volumen del objeto.
- Validar el sistema estimando el volumen de objetos con piezas de volumen conocido.

1.3 Justificación

Las inspecciones realizadas por los seres humanos, a menudo no pueden cumplir con los requisitos de la industria moderna respecto a la velocidad de producción, calidad de producto y costes de producción. Los humanos se cansan, cometen errores y los criterios que se aplican durante las inspecciones son inevitablemente subjetivos. En algunos casos, no es humanamente posible llevar a cabo las tareas de inspección debido a las condiciones ambientales. Las cámaras y los sistemas que componen un sistema de visión artificial, por el contrario, llevan a cabo las

mediciones con una precisión constante y a un ritmo que es establecido por el propio proceso de producción.

1.4 Alcance

El alcance para este trabajo de tesis, es obtener el volumen de objetos irregulares a partir de la reconstrucción de tridimensional de múltiples vistas. La estimación del volumen será aproximadamente de 6,000 cm^3 con un margen de error menor al 3% del volumen real. El alcance queda sujeto a las características físicas de la cámara y del objeto en estudio.

1.5 Estructura de la tesis

En el **Capítulo 1** se presenta una breve introducción a la importancia que tienen los sistemas de visión por computadora en la industria, en específico en la inspección de objetos, por otro lado se definen los objetivos y se justifica el desarrollo del trabajo de tesis presentado. Mientras que en el **Capítulo 2** se presenta una breve introducción sobre cómo se determina el volumen de objetos sólidos irregulares, así como la descripción de describen los sistemas de visión por computadora, haciendo énfasis en los métodos de calibración de una cámara, y por último se presentan las técnicas más comunes o trabajos desarrollados sobre reconstrucción tridimensional de objetos.

En el **Capítulo 3** encontramos conceptos y modelos matemáticos que nos ayudarán a desarrollar el objetivo de la tesis, dentro de este capítulo podemos encontrar conceptos de adquisición y procesamiento de imágenes con una cámara, la calibración de dicha cámara para obtener sus parámetros intrínsecos y extrínsecos, la distorsión radial y tangencial, es decir el modelo matemático de proyección de la cámara. Por otro lado, se describe el modelo matemático de la ley de control para el posicionamiento del servo, y a su vez se da una breve descripción del Software y librerías necesarias para la implementación de una Interfaz Gráfica de usuario.

El diseño y construcción de la base experimental, así como el diseño de la interfaz gráfica, y el algoritmo de reconstrucción tridimensional, se presentan en el **Capítulo 4**. Se mencionan algunas especificaciones técnicas del prototipo: características técnicas del diseño mecánico, capacidad de carga, detalles de la interfaz gráfica, entre otras. También en ese capítulo se desarrolla paso a paso el algoritmo propuesto para determinar el volumen de los objetos, y se explica por medio de un diagrama de flujo las fases que determinan la metodología propuesta.

El **Capítulo 5** consiste en la validación del sistema, en donde describe las diferentes pruebas que se usaron para evaluar la operación del prototipo propuesto. La interfaz gráfica se realizó en Visual C++ con ayuda de librerías de OpenCV, dicha interfaz genera un archivo de texto que contiene una nube de puntos del objeto reconstruido y el valor del volumen calculado. Las nubes de puntos son representadas con el software Geomagic para poder darle textura al objeto reconstruido.

Finalmente en el **Capítulo 6**, se reporta una serie de conclusiones acerca del funcionamiento de prototipo y se hacen recomendaciones y propuestas para trabajos futuros.

Capítulo 2. Estado del arte

En este capítulo se presenta una breve introducción sobre cómo se determina el volumen de objetos sólidos irregulares, así como una descripción de los sistemas de visión por computadora, haciendo énfasis en los métodos de calibración de una cámara, y por último se da una breve historia de las técnicas o trabajos desarrollados sobre reconstrucción tridimensional de objetos.

2.1 Técnicas para la estimación del volumen de objetos sólidos

Para comenzar con el estudio de las técnicas para determinar el volumen de objetos sólidos, primero definimos lo que significa dicha unidad de medida, según el Sistema Internacional de Unidades (BIPM, 2006), el volumen corresponde al lugar que ocupa un cuerpo en el espacio, se simboliza con la letra V y generalmente esta propiedad se asocia con el tamaño, sus unidades es tan dadas por el metro cúbico (m^3), ó litro. Es una unidad derivada de la longitud (metro).

De esta forma, para realizar la estimación del volumen de objetos, se tienen dos casos, cuando el objeto tiene geometría regular y cuando es de forma irregular. En caso de tener geometría regular (posee dimensiones bien definidas) se pueden aplicar las fórmulas muy bien conocidas para el cálculo de volumen, como lo son para el cubo, esfera, prisma, pirámide, entre otras formas.

En el caso de tener objetos con forma irregular, la obtención del volumen del objeto ya no es tan sencilla, pero ya existe una solución aunque con limitantes. Tal es el caso del principio de Arquímedes (Vitruvius,2010), (HyperPhysics, 2010) y (Carroll, 2010), en el cual plantea que un cuerpo total o parcialmente sumergido en un fluido en reposo, será empujado con una fuerza vertical ascendente igual al peso del volumen de fluido desplazado por dicho cuerpo. Esta fuerza recibe el nombre de empuje hidrostático o de Arquímedes, y se mide en newtons. El principio de Arquímedes se formula así

$$E = mg = \rho_f g V \qquad Ec.(2.1)$$

Donde ρ_f es la densidad del fluido, V el volumen del cuerpo sumergido y g la aceleración de la gravedad, de este modo, el empuje depende de la densidad del fluido, del volumen del cuerpo y de la gravedad existente en ese lugar. En el ejemplo de la Figura 2.1 el volumen del objeto es $2cm^3$.

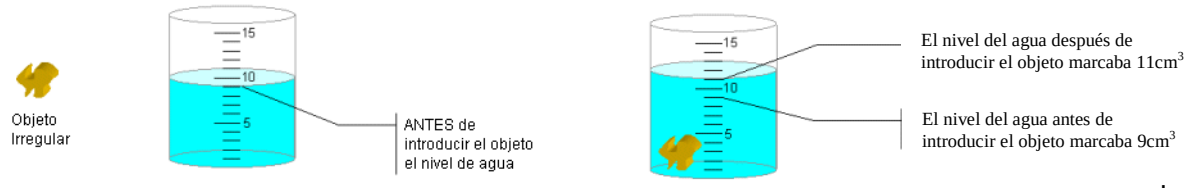


Figura 2.1. Volumen del objeto de acuerdo al nivel del agua desplazado

Mientras que por otro lado, se encuentra el Principio de Cavalieri (Harris y Stöcker, 1998), que dice: *Si dos cuerpos tienen la misma altura y al cortarlos por planos paralelos a las bases obtenemos figuras con la misma área, los cuerpos tienen el miso volumen*. Es decir, el principio de Cavalieri permite calcular el volumen de cualquier objeto de forma irregular, seccionada de manera sistemática, si se conocen los valores del área que ocupa cada corte y el grosor de ellos. El volumen total está dado por la sumatoria del volumen de cada uno de los segmentos del objeto.

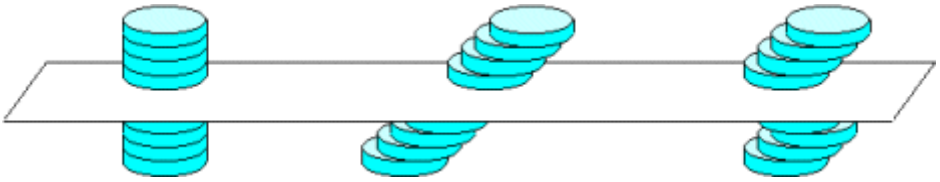


Figura 2.2. Principio de Cavalieri

Las áreas de las fichas que tocan la cinta son iguales para las tres pilas y si pasas la cinta a cualquier otra altura, las áreas de las fichas siguen siendo iguales. El Principio de Cavalieri asegura que si esto ocurre para cualquier altura entonces las tres pilas tienen el mismo volumen.

Por otro lado en México, el Centro Nacional de Metrología – CENAM (CENAM, 2011) es el laboratorio nacional de referencia en materia de mediciones. Ahí se establecen las unidades de medición con las más altas cualidades metrológicas posibles de acuerdo a sus capacidades científicas y técnicas. En el departamento de Metrología de Masa y Densidad, cuenta con el Sistema de Pesada Hidrostática (Becerra y Centeno, 2003), (CENAM – Patrón Nacional de Densidad, 2011), que está diseñado para realizar mediciones de cualquier líquido y en sólidos cuya masa sumergida no rebase los 900 g debido al alcance de medición de la balanza comparadora de masa. Se pueden realizar mediciones de densidad trazables al Patrón Nacional de Densidad en sólidos desde 2 200 kg/m^3 hasta 21 000 kg/m^3 (Ver Figura 2.3). Considerando que la densidad es el cociente de la masa con respecto al volumen de los cuerpos, es posible determinar de esta manera el volumen del cuerpo. De acuerdo con el sistema Internacional de Unidades (SI) las unidades de la densidad es el kilogramo por metro cúbico (kg/m^3).

Figura 2.3. Sistema de pesada hidrostática del Patrón Nacional de Densidad

Dentro del mismo departamento, en el Laboratorio de Densidad de sólidos (CENAM - Metrología de Masa y Densidad, 2011), cuentan con otro sistema que determina la densidad de los sólidos. En este laboratorio se realiza la determinación del volumen de sólidos cuyos valores en masa están en el intervalo de 1 g a 50 kg. Para sólidos de una masa de 100 g a 1 kg se puede alcanzar una incertidumbre relativa del orden de 2×10^{-5} (con un nivel de confianza del 95%) y que tengan una densidad desde 2,5 g/cm^3 a 21,0 g/cm^3 (Ver Figura 2.4).

Figura 2.4. Determinación de la densidad de un sólido. Laboratorio de metrología mecánica - CENAM

La determinación de volumen se hace mediante un método basado en el principio de Arquímedes, el cual consiste en medir alternadamente el empuje que sufre un sólido en aire y en un líquido en donde las densidades del aire y del líquido son conocidas.

Otra técnica para medir el volumen de un cuerpo es a través de una Máquina de Medición por Coordenadas, la cual se encuentra en el área de metrología dimensional del CENAM, que es

fundamental para la producción en serie y la intercambiabilidad de partes. Con tal propósito la División de Metrología Dimensional (CENAM - Metrología Dimensional, 2011) tiene a su cargo los patrones nacionales de longitud y ángulo plano. La unidad de longitud se obtiene mediante la calibración interferométrica (Bueno, 2010) de bloques patrón de alto grado de exactitud. Estos, a su vez, calibran otros de menor exactitud, estableciéndose la cadena de trazabilidad que llega hasta las mediciones de los instrumentos de uso industrial común. En esta división se encuentra el Laboratorio de Máquinas de Medición por Coordenadas, con la cual se realizan mediciones y caracterización de objetos con geometrías complejas y objetos con múltiples sistemas de referencia.

El laboratorio cuenta con dos máquinas de medición por coordenadas de alta exactitud (Ver Figura 2.5). La primera de ellas posee un volumen de medición de 1200mm x 710mm x 550mm con una incertidumbre volumétrica estimada de (0,8 + 1,25 L) mm, y la segunda máquina posee un volumen de medición de 1400mm x 900mm x 700mm, con incertidumbres volumétricas estimadas de (1 + 4L) mm, en ambos casos con [L] = metros (ver Apéndice A).

Las primeras máquinas de coordenadas en realidad fueron las máquinas de trazos, que son instrumentos con tres ejes mutuamente perpendiculares a fin de alcanzar coordenadas volumétricas en un sistema cartesiano para localizar un punto en el espacio sobre una pieza de tres dimensiones. Se conoce que a finales del año 1962, la firma italiana DEA construyó la primera máquina de medición cerca de Turín, Italia.

Figura 2.5. Laboratorio de Máquinas de Medición por Coordenadas, CENAM

Posteriormente en 1973 la compañía Carl Zeiss creó una máquina, equipada con un palpador, una computadora y un control numérico. Desde entonces han surgido muchas marcas y modelos de máquinas de coordenadas, que se distinguen entre sí por sus materiales de fabricación utilizados, software utilizado, versatilidad, alcances de medición, etc.

Actualmente se encuentra los brazos articulados, por ejemplo los FARO (FARO, 2010), son brazos que los puedes llevar a medir a la pieza de interés, buscando una buena zona de agarre para posteriormente medir, estos brazos no son muy precisos pero puede ser la única opción de medir en una situación especial (Ver Figura 2.6).

Figura 2.6. Brazo Articulado FARO

Por otro lado, se tienen los sistemas de inspección mediante visión por computadora, que suelen comprobar la conformidad de una pieza con ciertos requisitos, tales como las dimensiones, números de serie, la presencia de componentes, etc.

2.2 Sistemas de visión por computadora

La visión por computadora aplicada a la industrial abarca la informática, la óptica, la ingeniería mecánica y la automatización industrial. A diferencia de la visión por computadora académica, que se centra principalmente en máquinas basadas en el procesamiento de imágenes, las aplicaciones de visión por computadora industrial integran sistemas de captura de imágenes digitales, dispositivos de entrada/salida y redes de computadoras para el control de equipos destinados a la fabricación tales como brazos robóticos.

2.2.1 Mediciones con sistemas de visión por computadora

Entre las innumerables aplicaciones en el área de visión por computadora se encuentra la de realizar mediciones de longitudes, distancias, volúmenes y otras del mismo tipo, cuyas características principales son la automatización y la ausencia de contacto entre lo que se mide y

el instrumento de medición. Dado que una imagen es una representación de dos dimensiones (2D) del mundo real en tres dimensiones (3D), se pueden inferir muchas de las características de este último a partir del análisis de las imágenes. Por ejemplo, para determinar la distancia entre dos puntos (González y Castillo, 2010), primero se determinan las coordenadas en píxeles de los puntos correspondientes en la imagen, posteriormente hace uso de los parámetros de la cámara (intrínsecos y extrinsecos), así como de la calibración de cámara. Esto es posible bajo la condición de que los puntos en análisis se encuentren sobre un mismo plano, paralelo al plano XY del marco de referencia de la cámara, es decir, que tengan la misa coordenada en el eje Z.

Un sistema de visión por computadora está compuesto de los siguientes elementos:

- Una cámara analógica o digital (monocromática o a color).
- Un procesador (suele ser una computadora).
- Una tarjeta digitalizadora para la adquisición de imágenes.
- Software para el procesamiento de las imágenes, extracción de características, calibración, medición, etc.
- Un sistema de iluminación para facilitar la extracción de características de la imagen.

2.2.2 Métodos de calibración de cámaras

Dentro del área de visión por computadora un aspecto importante a considerar es la calibración de la cámara, dicho proceso consiste en determinar los parámetros intrínsecos (distancia focal, asimetría, factor de factor de distorsión y centro óptico del plano imagen). Existen diversos métodos que básicamente se diferencian en la forma de adquirir desde las imágenes los parámetros intrínsecos

La necesidad de extraer información métrica a partir de imágenes 2D del mundo real, nos lleva a realizar el proceso de la calibración de la cámara. Aunque no es obligatoria la calibración de la cámara, ya que existen técnicas para inferir esta información desde imágenes captadas por cámaras no calibradas, a través de procesos iterativos de auto-calibración o el convencional proceso de calibración de los 3 puntos, el proceso de calibración de cámara abre la posibilidad de realizar aplicaciones efectivas en visión, tal como realidad aumentada, reconocimiento, seguimiento y reconstrucción 3D, los cuales se basan en el conocimiento de calibración y pose de la cámara (Trucco, 1998).

Por lo tanto, el objetivo principal de la calibración de cámara describir un modelo de proyección que relacione los sistemas de coordenadas (plano imagen – mundo real) que permiten obtener los parámetros de la cámara. Es decir, determinar la geometría y características internas de la cámara,

parámetros intrínsecos (tamaño horizontal y vertical de los píxeles o aspecto proporcional, coordenadas de proyección del centro óptico, largo focal), y los parámetros extrínsecos (rotación y translación), que representan la localización y orientación (pose) de la cámara relativa a una imagen en un sistema de coordenadas. Estos parámetros normalmente son calculados desde un patrón de calibración (Ver Figura 2.7) que contiene rasgos fácilmente detectables de manera precisa en la imagen capturada (Tuceryan, 1995).

Para localizar un objeto en el mundo real, se establece un marco de referencia (llamado marco de referencia del mundo real). Mientras que un objeto en una imagen es localizado en términos de coordenadas de píxeles, los cuales están en el marco de referencia de la imagen. Para obtener una relación entre ambos marcos de referencia, es necesario establecer ecuaciones que unan el marco de referencia del mundo con el marco de referencia de la imagen, con el fin de establecer la relación entre los puntos en coordenadas en el espacio 3D y los puntos en coordenadas de imagen 2D. Para lograr esta relación se establece un marco de referencia intermedio, llamado marco de referencia de la cámara. Por lo tanto, se deben encontrar las ecuaciones que unan el marco de referencia de la cámara con el marco de referencia de la imagen, y las ecuaciones que unan el marco de referencia del mundo real con el marco de referencia de la cámara. Al resolver el sistema generado se obtiene la relación buscada, lo cual es equivalente a encontrar las características de la cámara.

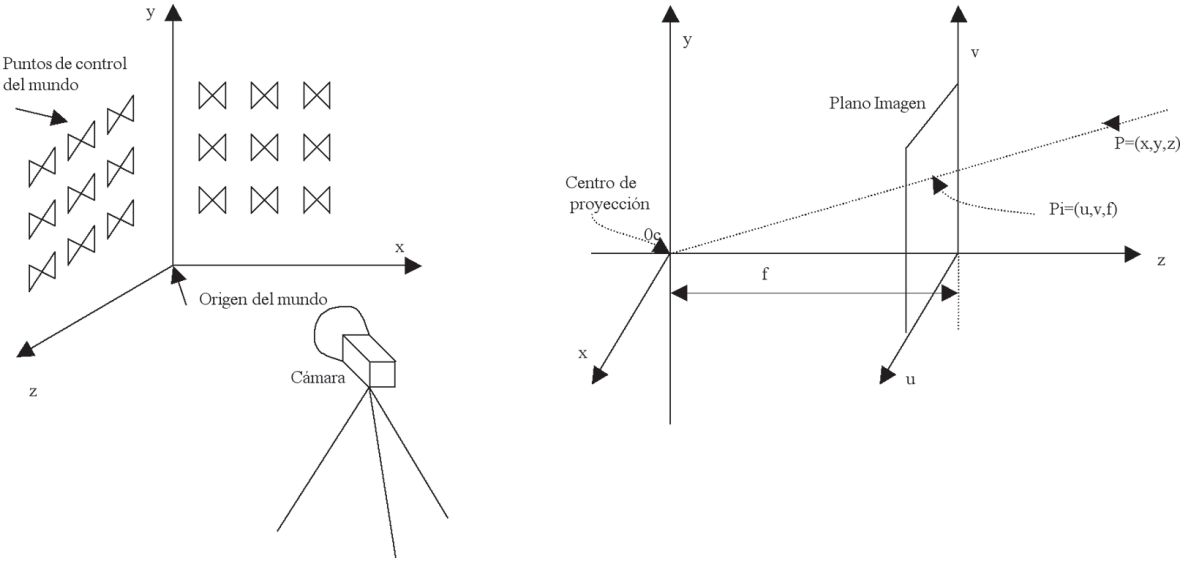


Figura 2.7. Grilla padrón de Calibración y sistema de referencia (M. Tuceryan, 1995)

La calibración de cámara se puede clasificar en dos categorías: Calibración fotogramétrica y auto-calibración (Zhang, 1999):

- Calibración fotogramétrica. Consiste en el cálculo de los parámetros intrínsecos y extrínsecos de la cámara a partir de un conjunto de puntos de control, conocidas las coordenadas 3D de esos puntos y midiendo las correspondientes coordenadas 2D en una imagen obtenida con dicha cámara. Esta calibración utiliza un objeto 3D (o patrones) de referencia cuya geometría es conocida a su perfección. Los patrones de calibración normalmente están posicionados en dos o tres planos ortogonales entre ellos. En algunos casos, basta con un único plano, cuya traslación es perfectamente conocida (Zhang, 1999).
- Auto-calibración. Este método requiere de varias imágenes de una misma escena, por lo tanto se propone el movimiento de la cámara observando una escena estática y mediante la correspondencia entre puntos de diferentes imágenes se pueden encontrar los parámetros de un modelo que determine dicha correspondencia. La rigidez de la escena impone en general restricciones sobre los parámetros de cámara. Tres imágenes tomadas por una misma cámara con parámetros intrínsecos fijos son suficientes para obtener tanto los parámetros extrínsecos como intrínsecos.

Dos métodos comunes de calibración, en la categoría de calibración fotogramétrica, son “Calibración directa de parámetros” y “Recuperación desde la homografía o matriz de proyección”. A través de la “Calibración directa de parámetros” se obtienen los parámetros extrínsecos e intrínsecos directamente, es decir, requieren de patrones de calibración consistentes, en al menos dos planos ortogonales, y a través de la “Recuperación desde la matriz de proyección” se obtienen los parámetros a partir de los valores de la homografía o matriz de proyección. Los métodos de calibración directa requieren de patrones de calibración consistentes, en al menos dos planos ortogonales. Sin embargo, la calibración mediante homografía o matriz de proyección, una variante recientemente propuesta, consiste en utilizar patrones que descansan en un único plano.

Tommaselli y Tozzi (Tommaselli y Tozzi, 1999), abordaron el problema de calibración dinámica de una cámara considerando un objeto en movimiento constante sobre líneas rectas como referencias con la cámara fija. El modelo se basa en la equivalencia del vector normal al plano de interpretación en el marco de la imagen y el vector normal al plano de interpretación rotado en el marco del objeto. Para resolver la calibración dinámica de una cámara se aplica un proceso iterativo basado en el filtro de Kalman usando los parámetros de orientación de la cámara secuencialmente estimados para retroalimentar el proceso de extracción de características de la imagen. La solución propuesta se implementó usando datos reales de un cubo moviéndose. Las restricciones que deben resaltarse son que únicamente se considera movimiento lineal constante y sobre una superficie plana. Dentro de las pruebas, los parámetros de traslación no son muy confiables y los parámetros intrínsecos deben considerarse más precisos y con cámaras de mejor

resolución. De igual forma no se aplicaron técnicas de procesamiento de imagen muy efectivas para poder alcanzar resultados más deseables.

Peter Eisert (Eisert, 2002) propone una nueva técnica para la determinación de los parámetros extrínsecos e intrínsecos de la cámara basada en un modelo que consiste en absorber información característica de una imagen que vinculamos con uno o varios objetos de prueba. Después obtiene su representación 3D, a la vez que compara en todo momento posibles proyecciones en imagen de los puntos del modelo con las imágenes originales capturados por la cámara. El modelo prevé un proceso iterativo para disminuir el error entre imágenes originales y de proyección del modelo. Se plantean dos posibles estrategias, la primera hacer sólo la estimación de los parámetros extrínsecos y la segunda de los dos tipos de parámetros. Así, el patrón de calibración utilizado no se restringe a mostrar simples características discretas detectables fácilmente, y si pueden contemplarse objetos con un amplio rango de frecuencias espaciales.

El Método de Faugeras (Faugeras, 1992), es un proceso de calibración que consiste en estimar la matriz de proyección P, y posteriormente se estiman los parámetros intrínsecos y extrínsecos de la cámara a partir de la matriz de proyección obtenida anteriormente. La calibración se logra descomponiendo en valores singulares la matriz que forma el sistema de ecuaciones que proyecta un punto del espacio en el plano imagen de la cámara. Obteniéndose una matriz de proyección a partir de la cual se extraen todos los parámetros, tanto los extrínsecos (traslación y rotación del patrón respecto a la cámara) como los intrínsecos (centro del eje óptico, factores y factores de escalado). Dicho método, no toma en cuenta la distorsión introducida por las lentes. Para realizar el método de Faugeras únicamente es necesaria una imagen.

(Zhang, 1999) propone una técnica de calibración utilizando una plantilla plana (con forma de tablero de ajedrez) como la de la Figura 2.8 colocada en diferentes poses.

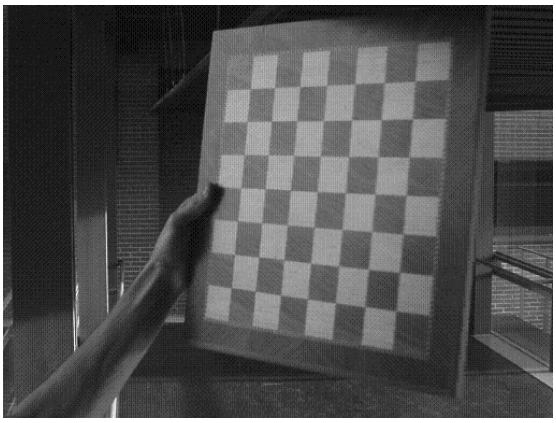


Figura 2.8. Plantilla plana de calibración

Es una técnica flexible ya que no requiere una preparación exhaustiva de la escena. Teniendo como ventaja sobre los demás métodos de calibración la fácil obtención de los parámetros de la cámara a partir de una plantilla plana sin necesidad de conocer la posición de algún punto de la escena. Esto se debe a la selección de las coordenadas del mundo de tal forma que dos de sus ejes coinciden con los lados de la plantilla, y el tercero (Z) es perpendicular a esta. Dicho método requiere de al menos tres imágenes con la plantilla tomada en distintas orientaciones. Este número de imágenes puede ser inferior si se fijan los valores de algunos parámetros intrínsecos. Por ejemplo, si no se calcula la ortogonalidad del plano imagen, sólo son necesarias dos imágenes.

El método de Heikkila (Heikkila y Silven, 1997) no necesita conocer la posición absoluta de los puntos en referencia para realizar el proceso de calibración. De igual manera que los métodos de calibración explicados anteriormente, obtiene los parámetros intrínsecos y extrínsecos del modelo, pero toma en cuenta las distorsiones radial y tangencial de la óptica, convirtiéndose en un método no lineal. Por tal motivo, primero realiza una aproximación inicial de la matriz de transformación basándose en las características de la óptica proporcionadas por el fabricante.

En cambio (Tuceryan, 1995) y (Trucco, 1998) se basan en un modelo conocido 3D y su correspondencia en el plano imagen 2D, diferenciándose en la consideración de los parámetros intrínsecos que deben ser determinados o de influencia en la matriz de cámara final.

2.3 Métodos para la reconstrucción tridimensional

La reconstrucción en tres dimensiones (3D) es el proceso mediante el cual, los objetos reales son representados digitalmente en la memoria de una computadora, manteniendo sus características físicas de dimensión, volumen y forma. El problema puede enfocarse como la relación entre una imagen en un sistema de coordenadas 2D, y un objeto en el mundo real representado en un sistema de coordenadas en 3D.

La obtención de información 3D de una escena se puede realizar mediante técnicas muy diversas. Estas técnicas se dividen, según (Rocchini, 2001), en dos grandes grupos: adquisición con contacto y adquisición sin contacto como se muestra en la Figura 2.9. Las técnicas de adquisición con contacto, se dividen a su vez en técnicas destructivas como el laminado, y en técnicas no destructivas, como los brazos articulados o máquinas de coordenadas. Por su parte, las técnicas sin contacto se subdividen en técnicas transmisivas y técnicas reflectivas.

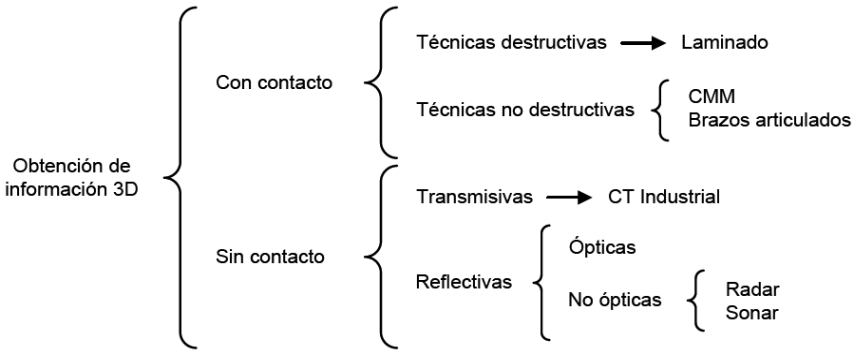
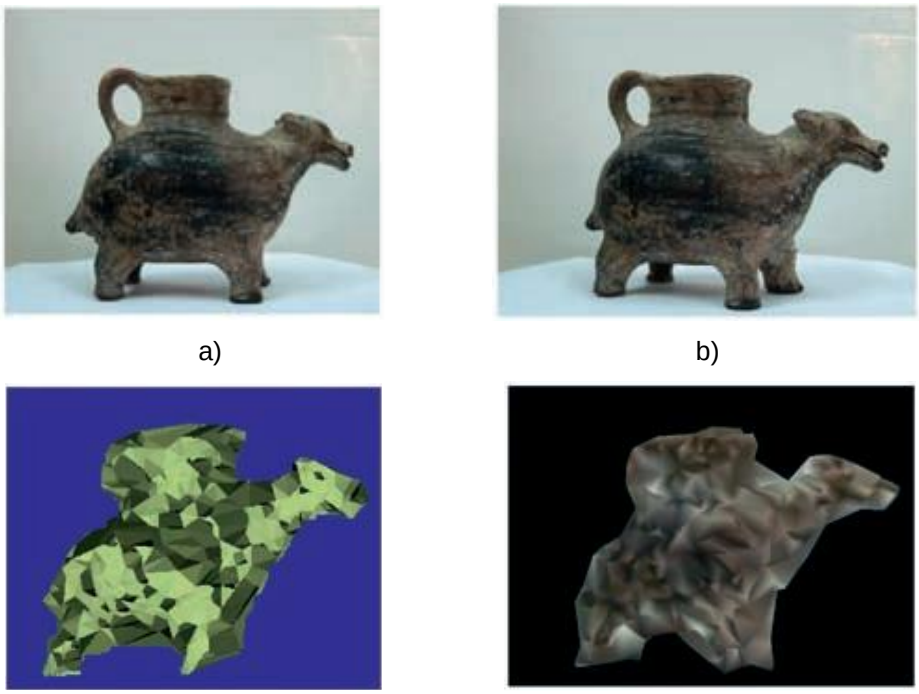


Figura 2.9. Clasificación de técnicas de adquisición de información 3D de una escena

Dentro de las técnicas reflectivas se agrupan, por un lado, las técnicas no ópticas (radar y sonar) y por otro lado, a las técnicas ópticas. Por lo tanto, las técnicas ópticas de visión artificial son un subconjunto dentro de las técnicas reflectivas que se pueden utilizar para obtener información tridimensional de una escena, y a su vez se dividen en activas y pasivas (Besl, 1988). En la Figura 2.10 se muestra la clasificación de las técnicas ópticas.

Figura 2.10. Clasificación de las técnicas ópticas para la obtención de información 3D de una escena

En la actualidad, han sido muchos los trabajos de reconstrucción 3D de objetos a partir de múltiples vistas (Baker, 1997), (Martin, 1983), (Pastén, 2007), (Schmitt, 2002), (Niem, 1994), cada uno con distintas técnicas pero con el mismo fin. Baker (Baker, 1997) utilizó las siluetas de un objeto sobre una mesa giratoria para construir un modelo 3D del objeto. (Martin, 1983) realizan proyecciones ortográficas por cada una de las siluetas del objeto (“Volume segment”). (Pastén, 2007) realizan la reconstrucción 3D a partir de un conjunto de vistas, a la cual se le aplica la triangulación de Delaunay para generar la malla del objeto para poder visualizarla en un ambiente virtual denominado VRML (Ver Figura 2.11).



a) b)

c) d)

Figura 2.11. Reconstrucción de objeto perro nativo, a) Vista izquierda Objeto, b) Vista derecha objeto, c) Reconstrucción objeto y d) Reconstrucción objeto aplicando mapeamiento

En (Schmitt, 2001), se presenta un sistema de reconstrucción 3D combinando dos procedimientos, primero utilizando la técnica de reconstrucción mediante la silueta del objeto para obtener un modelo 3D inicial, y posteriormente utilizan una técnica de múltiples correspondencias para esculpir el modelo inicial (Ver Figura 2.12).

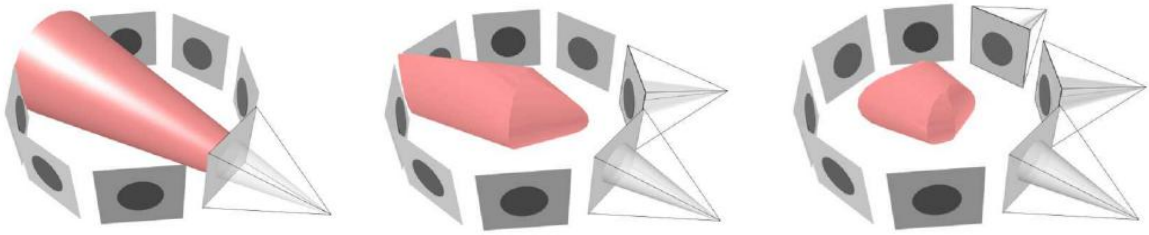


Figura 2.12, Reconstrucción 3D mediante proyecciones cónicas de las siluetas

Y (Niem, 1994) desarrolla un algoritmo robusto y rápido para la construcción de un modelo 3D de algún objeto usando imágenes desde múltiples vistas (Ver Figura 2.13).

a) b) c)

Figura 2.13. Reconstrucción 3D de Niem , a) Base experimental, b) Objeto real, c) Objeto reconstruido

Park y Subbarao (Park & Subbarao, 2002) propusieron una nueva técnica de Reconstrucción 3D, la cual se basa en obtener dos mallados del objeto en diferentes poses. Posteriormente, se hace la intersección de los dos mallados para obtener el mallado completo del objeto, como se muestra en Figura 2.14.

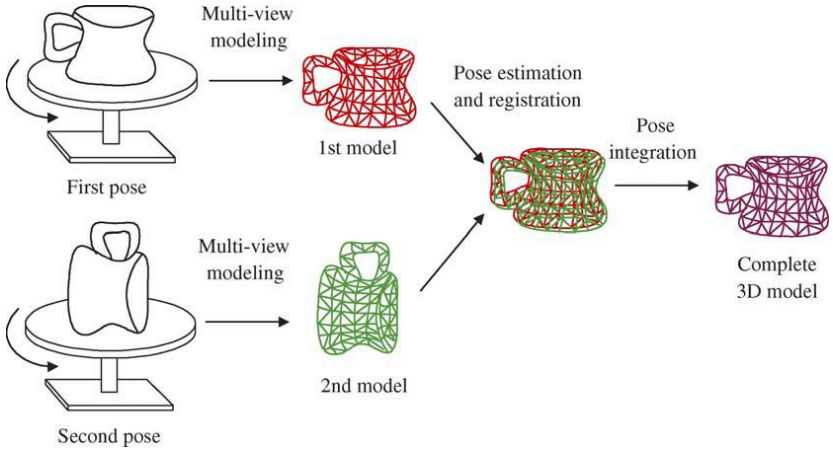


Figura 2.14. Reconstrucción 3D por el método de Park-Subbarao

Kuzu y Sinram (Kuzu y Sinram, 2003) proponen primero calibrar la cámara para obtener la rotación entre el plano de la imagen y la cámara, posteriormente realizan la reconstrucción tridimensional mediante el procesamiento de sus múltiples siluetas (Ver Figura 2.15).

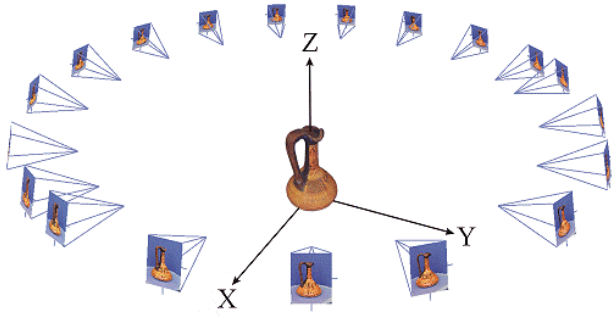


Figura 2.15. Reconstrucción tridimensional del objeto mediante el procesamiento de sus siluetas

Por último, Tadeo G. y Gallegos F. (Tadeo y Gallegos, 2008) propusieron la reconstrucción tridimensional de imágenes de tomografías mediante secciones o cortes, como lo descrito con el principio de Cavalieri, en el cual van empalmando cada una de las tomografías hasta generar el modelo tridimensional (Ver Figura 2.16 y Figura 2.17)

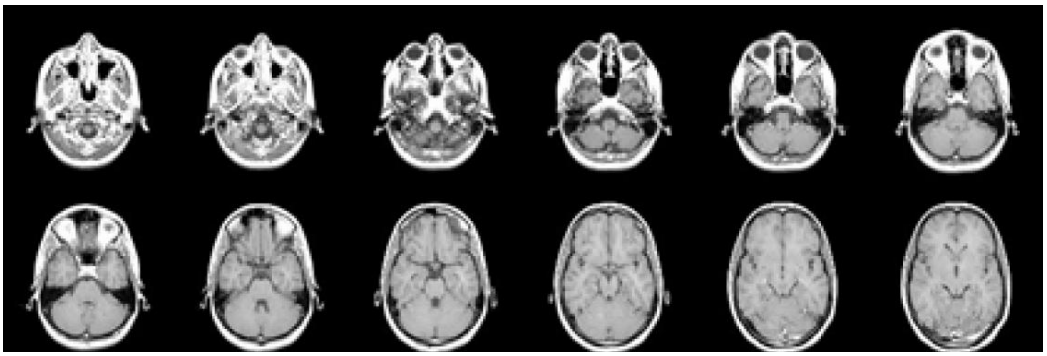


Figura 2.16. Muestra de imágenes de Tomografía

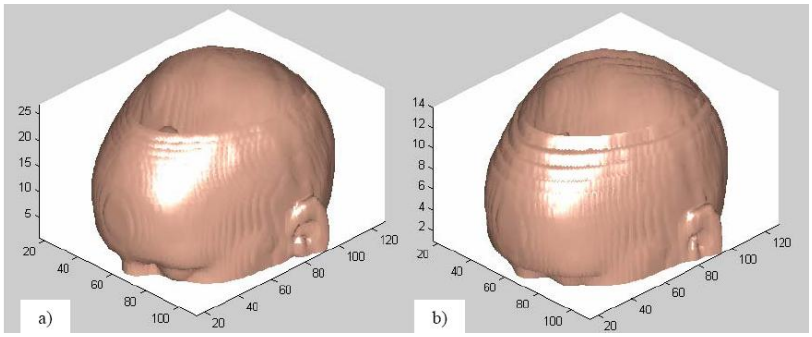


Figura 2.17. Reconstrucción con a) 30 rebanadas y, b) 14 rebanadas

Como se puede observar, cada uno de los métodos de reconstrucción 3D mencionados anteriormente, tienen como característica principal la obtención de la silueta del objeto en diferentes posiciones, posteriormente se realiza una proyección para generar un arreglo matricial por cada silueta, y por último se realiza la intersección de todas las proyecciones para generar el arreglo o la nube de puntos de la reconstrucción 3D del objeto. La nube de puntos resultante del proceso de reconstrucción 3D es triangulado para crear el modelo 3D del objeto. Para ello se utiliza el método triangulación de Delaunay (Delaunay, 1934) el cual determina la malla que será visualizada en un ambiente 3D.

Capítulo 3. Marco teórico

En este capítulo se muestran los conceptos y herramientas necesarios para la realización del proyecto. Por lo tanto encontraremos una breve descripción de los sistemas de visión por computadora aplicados en la reconstrucción 3D, por otro lado se describe el modelo matemático de la cámara, así como la descripción del Toolbox de Matlab para la calibración de las cámaras por medio de Zhang, en ese mismo sentido, se dá una breve descripción de OpenCV que es un conjunto de librerías para la adquisición y procesamiento de imágenes.

Por otro lado, se presentan los conceptos básicos para el control de posicionamiento angular de la plataforma giratoria, se da una breve explicación del control PID para un motor, se describe su modelo matemático y se explica los pasos a seguir para realizar la sintonización del control.

3.1 Visión por computadora en la reconstrucción 3D

Un sistema de visión por computador para la reconstrucción de la forma tridimensional de una escena está compuesto por un conjunto de elementos. La distribución de esos elementos en el espacio, así como su distancia relativa con referencia a la escena, se denomina geometría del sistema de visión. Según las características de los elementos que forman el sistema de visión se establecerá una serie de restricciones que debe cumplir su geometría, definiendo un conjunto de configuraciones posibles. La elección de la geometría a implantar entre una de las posibles configuraciones válidas se realiza en función de la naturaleza del problema y del entorno de instalación. En la siguiente sub-secciones se describen los elementos principales que componen un sistema de visión por computadora.

3.1.1 Fuente de iluminación

La iluminación es la característica que utilizan las técnicas de visión por computadora para extraer información de los objetos de una escena. La iluminación de una escena se puede realizar con luz natural o mediante utilización de lámparas.

Las tecnologías de iluminación (Ver Figura 3.1) más comúnmente utilizadas son las lámparas halógenas, lámparas fluorescentes, diodos leds y láser. La luz láser posee ciertas particularidades, tales como coherencia espacial y temporal, que la diferencian claramente del reto de tecnologías de iluminación (Bachs y Cuesta, 1988). Los aspectos a considerar para escoger el sistema de

iluminación se pueden agrupar en: color y composición espectral, intensidad de la radiación luminosa, regulación de intensidad, velocidad de respuesta, costo (adquisición y mantenimiento), tipo de superficies de los objetos, tipo de características a resaltar, entre otros.

a) b) c)

Figura 3.1. Fuentes de iluminación a) Iluminación por LED de alta potencia, b) Sistemas para iluminar áreas, c) Luz fluorescente en forma de anillo

Para la adecuada selección de la técnica de iluminación a utilizar, se toma en cuenta la Tabla 3.1 elaborada por Giraldo, Arroyave y Montilla (Giraldo, Arroyave y Montilla, 2008):

Tipo de Iluminación	Ventajas	Desventajas
Posterior	Fácil de implementar, bajo costo, crea silueta de la pieza, imágenes de muy alto contraste.	Se debe tener espacio detrás del objeto.
Frontal	Fácil de implementar, revela algunos detalles superficiales.	Puede crear sombras indeseadas, la iluminación no es uniforme.
Lateral	Fácil de implementar, revela algunos detalles superficiales.	Puede crear sombras indeseadas, la iluminación no es uniforme.
Por campo oscuro	Ilumina defectos superficiales, provee imágenes de alto contraste en algunas aplicaciones.	No ilumina de forma plana, difumina las superficies.
Coaxial	Elimina sombras, iluminación uniforme en todo el campo de visión.	Difícil de implementar, reflexión intensa en superficies brillantes.
Difusa continua	Elimina sombras, elimina el deslumbramiento.	Debe rodear el objeto, la fuente de iluminación es grande, puede ser costosa.

Tabla 3.1. Clasificación de las técnicas de iluminación.

3.1.2 Cámara

La cámara de un sistema de visión por computadora es el dispositivo que recibe la luz reflejada por la escena y la utiliza para generar imágenes. El elemento más importante de la cámara es su sensor, que está formado por una capa de material fotosensible que transforma la luz en señales eléctricas. El sensor puede estar construido en base a una tecnología analógica (cámara de tubo) o en base a una tecnología digital (cámaras de estado sólido). La tecnología digital más utilizada actualmente para la construcción de sensores de imagen es CCD.

Además de la tecnología de fabricación, un aspecto importante del sensor de una cámara digital e su factor de forma. Los dos tipos básicos son lineales y matriciales. Las cámaras lineales obtienen mucho menos información de la escena pero poseen una tasa de transferencia mucho más elevada que las capturas matriciales.

3.1.3 Procesador de datos

La computadora que es la encargada de realizar el procesamiento de las imágenes necesita un dispositivo hardware para capturar la imágenes que le envía la cámara. Además de tareas de captura, este dispositivo puede realizar algunas tareas de procesamiento de las imágenes, liberando de esta tarea al procesador principal de la computadora. La tarjeta procesadora y la cámara se comunican mediante un protocolo de transferencia de datos que define los parámetros de la señal de video a transmitir, el formato de codificación de las imágenes y la sincronización. Todos los parámetros son función de los estándares de video utilizados en cada caso: NTSC y RS-170 en Estados Unidos, PAL y CCIR en Europa, y Camera Link, entre otros.

Dos de los lenguajes de programación más utilizados para el procesamiento de imágenes son C++ y MATLAB. C++ tiene la ventaja de que los programas son lo suficientemente rápidos y éste es uno de los principales requisitos en muchas aplicaciones de este tipo. La ventaja del uso de MATLAB radica en que es un lenguaje orientado al manejo de matrices, lo que facilita el procesamiento de imágenes dado que, para la computadora, ambas son equivalentes. MATLAB incluye además de *toolbox* para el procesamiento de imágenes con funciones para realizar transformaciones espaciales, filtraje de imágenes, operaciones morfológicas, análisis y mejora de imágenes, operaciones con imágenes a color y algunas otras (Matlab, 2004).

3.2 Procesamiento de imágenes

En este apartado se presentan algunos algoritmos para la extracción de características de una imagen. Por un lado, la extracción de las siluetas de un objeto en una imagen se realiza a partir de una imagen binaria. Estas deben obtenerse por técnicas de segmentación, una de ellas es mediante el algoritmo de Otsu, que más adelante se describe. Por otro lado, para la extracción del contorno de la silueta, se describe el algoritmo de Canny.

3.2.1 Algoritmo de Otsu

El termino segmentación está orientado a dos tareas, a detección de regiones y detección de bordes. Se entiende como "región" al área de la imagen en la que sus píxeles poseen propiedades similares de intensidad o de color; y como "bordes" a las líneas que separan dos regiones.

El método de Otsu, llamado así en honor a Nobuyuki Otsu, se basa en técnicas estadísticas, para determinar el valor del umbral. Utiliza la variancia, que es una medida de la dispersión de valores, es decir mide la dispersión de los niveles de gris. La importancia del método radica en que es automático, es decir, no necesita supervisión humana ni información previa de la imagen antes de su procesamiento. Se emplea cuando la escena se caracteriza por un fondo uniforme y por objetos parecidos.

En particular, el método de Otsu, que es el objetivo de esta sección, elige el umbral óptimo maximizando la varianza entre clases (between-class variance) mediante una búsqueda exhaustiva.

Se considera a una imagen como una función bidimensional de la intensidad del nivel de gris, que contiene N píxeles cuyos niveles de gris se encuentran entre 1 y L. La cantidad de píxeles con nivel de gris i se denota como f_i, y la probabilidad de ocurrencia del nivel de gris i en la imagen está dada por

$$p_i = \frac{f_i}{N} \qquad Ec.(3.1)$$

Los píxeles son divididos en dos clases: C_1, con niveles de gris [1,..., t]; y C_2, con niveles de gris [t+1,..., L]. Entonces, la distribución de probabilidad de los niveles de gris para las dos clases son:

$$C_1: \frac{p_1}{w_1(t)}, \dots, \frac{p_t}{w_1(t)} \qquad Ec.(3.2)$$

$$C_2: \frac{p_{t+1}}{w_2(t)}, \frac{p_{t+2}}{w_2(t)} \dots, \frac{p_L}{w_2(t)} \qquad Ec.(3.3)$$

Donde

$$w_1(t) = \sum_{i=1}^{t} p_i \qquad w_2(t) = \sum_{i=t+1}^{L} p_i \qquad Ec.(3.4)$$

También, la media para la clase C_1 y la clase C_2 es

$$\mu_1(t) = \sum_{i=1}^{t} \frac{i*p_i}{w_1(t)} \qquad \mu_2(t) = \sum_{i=t+1}^{L} \frac{i*p_i}{w_2(t)} \qquad Ec.(3.5)$$

Sea μ_T la intensidad media de toda la imagen. Es fácil demostrar que

$$w_1\mu_1 + w_2\mu_2 = \mu_T \qquad w_1 + w_2 = 1 \qquad Ec.(3.6)$$

Usando análisis discriminante, Otsu definió la variancia entre clases de una imagen umbralizada como

$$\sigma_B^2 = w_1(\mu_1 - \mu_T)^2 + w_2(\mu_2 - \mu_T)^2 \qquad Ec.(3.7)$$

Para una umbralización de dos niveles, Otsu verificó que el umbral óptimo t* se elige de manera que σ_B^2 sea máxima; esto es

$$t^* = Max\{\sigma_B^2(t)\} \qquad 1 \le t \le L \qquad Ec.(3.8)$$

3.2.2 Algoritmo de Canny

Este algoritmo está considerado como uno de los mejores métodos de detección de contornos mediante el empleo de máscaras de convolución, basado en la primera derivada. Básicamente se basa en tres criterios:

- Criterio de detección: evitar la eliminación de bordes importantes y no suministrar falsos bordes.
- Criterio de localización: la distancia entre la posición real y la localizada del borde se debe minimizar.
- Criterio de respuesta: que integre las respuestas múltiples correspondientes a un único borde.

Esta técnica consta de tres etapas principales: filtrado, decisión inicial, e histéresis. La primera etapa consiste en un filtrado de convolución de la primera derivada de una función gaussiana normalizada discreta sobre la imagen, realizada en dos direcciones: horizontal y vertical. La segunda etapa realiza una primera toma de decisiones sobre los posibles bordes de la imagen, mediante el cálculo de los picos de las imágenes obtenidas en la primera etapa. La última etapa de procesamiento realiza una optimización de la decisión llevada a cabo en la etapa anterior, mediante la aplicación de una función de histéresis.

Este método es uno de los más robustos contra el ruido (filtro óptimo), en comparación con los métodos de Roberts, Sobel o Prewitt.

En resumen, el algoritmo de Canny consiste en tres grandes pasos:

- Obtención del gradiente: en este paso se calcula la magnitud y orientación del vector gradiente en cada píxel.
- Supresión no máxima: en este paso se logra el adelgazamiento del ancho de los bordes, obtenidos con el gradiente, hasta lograr bordes de un píxel de ancho.
- Histéresis de umbral: en este paso se aplica una función de histéresis basada en dos umbrales; con este proceso se pretende reducir la posibilidad de aparición de contornos falsos.

3.3 Modelo matemático de una cámara (Pinhole)

En esta sección se describe el modelo matemático que definen la transformación que aplica la óptica de un sistema de visión a la luz que recibe: modelo del agujero o modelo pinhole. Además, se describe cómo distorsionan las ópticas los rayos de luz provenientes de la escena.

3.3.1 Coordenadas homogéneas

Las coordenadas homogéneas de un punto tridimensional con coordenadas cartesianas *(X,Y,Z)* se definen como el punto tetra-dimensional $(kX,kY,kZ,k)^T$, donde *k* es una constante que define el escalado (habitualmente se suele utilizar k=1). Por tanto, un punto P del espacio, representado mediante coordenadas homogéneas, se expresa en forma vectorial como

$$P = \begin{bmatrix} kX \\ kY \\ kZ \\ k \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \qquad Ec.(3.9)$$

3.3.1.1 Traslación

La traslación de un punto $P=[X\ Y\ Z]^T$ expresada mediante coordenadas homogéneas se puede observar en la siguiente ecuación, donde $=[X_0\ Y_0\ Z_0]^T$ es el vector de traslación

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & X_0 \\ 0 & 1 & 0 & Y_0 \\ 0 & 0 & 1 & Z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \qquad Ec.(3.10)$$

3.3.1.2 Escalado

El escalado de un punto del espacio $P=[X\ Y\ Z]^T$ con los factores S_X, S_Y y S_Z en los ejes cartesianos X, Y y Z, respectivamente, expresado en coordenadas homogéneas se puede observar en la siguiente ecuación:

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_X & 0 & 0 & 0 \\ 0 & S_Y & 0 & 0 \\ 0 & 0 & S_Z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \qquad Ec.(3.11)$$

3.3.1.3 Rotación

La rotación de un punto del espacio $P=[X\ Y\ Z]^T$ respecto a cada uno de los ejes cartesianos, de forma que α es el ángulo de rotación en el eje X, β el ángulo de rotación alrededor del eje Y y γ el ángulo en el eje Z, expresada en coordenadas homogéneas viene dada por las matrices R_α,R_β y R_γ para los ejes X, Y y Z, respectivamente.

$$R_\alpha = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & sin(\alpha) & 0 \\ 0 & -sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_\beta = \begin{bmatrix} \cos(\beta) & 0 & sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad Ec.(3.12)$$

$$R_\gamma = \begin{bmatrix} \cos(\gamma) & sin(\gamma) & 0 & 0 \\ -sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.3.2 Modelo matemático Pinhole

El modelo pinhole es el modelo de transformación utilizado en visión por computadora para eliminar las deformaciones en imágenes causadas por distorsiones no lineales de la óptica. En este modelo se considera la óptica del sistema formada por una única lente representada por un punto infinitesimal, denominado foco, a través del cual pasan los rayos de luz procedentes de la escena hacia el sensor de la cámara. La Figura 3.2 muestra una representación del modelo pinhole.

Dado que el foco en este modelo representa un agujero de diámetro infinitesimal, sólo uno de los rayos de luz de todos los que proceden del mismo punto de la escena llega al sensor de la cámara, es decir, sobre cada elemento del sensor sólo incide un único rayo de luz. Por tanto, todos los

puntos de la escena estarán perfectamente enfocados en la imagen. Este modelo matemático equivale a una transformación perspectiva y el único parámetro del modelo es la distancia focal de la cámara.

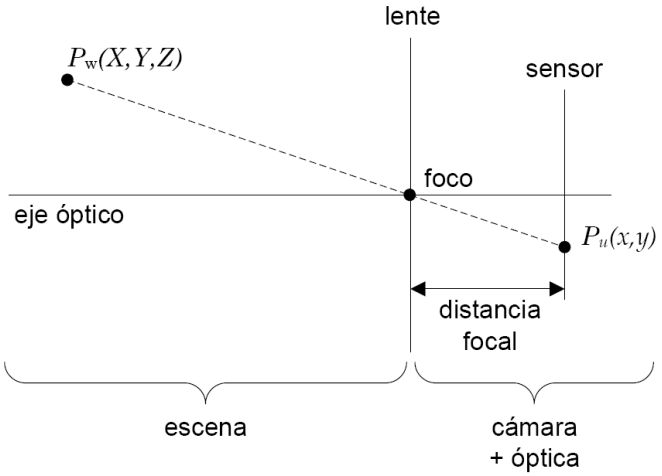


Figura 3.2. Modelo Pinhole

La proyección de la luz reflejada por los objetos de la escena en el plano de la imagen se considera realizada, independientemente de la óptica utilizada en el sistema de visión, mediante una transformación perspectiva. Esta transformación define, geométricamente, la formación de la imagen a partir de la proyección de un espacio tridimensional a otro bidimensional.

El modelo de cámara pinhole describe las relaciones entre un punto 3D, coordenadas globales $P_w=[X,Y,Z,1]^T$, y su proyección en el plano imagen $P_u = [u,v,1]^T$ como:

$$sP_u = A[RT]P_w$$

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A[RT]\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \qquad Ec.(3.13)$$

Donde *s* es un factor de escala, *R* es la rotación existente entre los marcos de referencia, *T* es el vector de traslación, *A* es la matriz de los parámetros intrínsecos de la cámara, la cual está en función de las coordenadas del centro de la imagen (u_0, v_0), α y β los cuales se derivan del largo focal y tamaño del píxel y γ la asimetría. Así la matriz de cámara es de la forma:

$$A = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \qquad Ec.(3.14)$$

3.4 Toolbox de Matlab para la calibración de las cámaras (Zhang)

Como se describe en la sección 2.2, una de las partes que conforman un sistema de visión por computadora, es una cámara digital o analógica. En donde para aplicaciones con visión por computadora se debe considerar el proceso denominado calibración de cámara, el cual consiste en determinar los parámetros internos y extrínsecos de la cámara. Uno de los métodos de calibración más común, y que viene implementado como un Toolbox del software de Matlab, es el algoritmo de Zhang (Camera Calibration Toolbox for Matlab, 2010).

Por lo tanto, la calibración es el proceso de estimar los parámetros extrínsecos e intrínsecos de las cámaras. Para utilizar la herramienta del Toolbox es necesario utilizar un patrón de calibración como el que se observa en la Figura 2.8 para obtener imágenes del patrón desde las diferentes posiciones. El patrón está formado por cuadrados negros y blancos con un tamaño conocido. El objetivo de los cuadros son blancos y negros es detectar las esquinas fácilmente y obtener un entramado de puntos tridimensionales conocidos y su proyección en las diferentes posiciones. La herramienta detecta las proyecciones de los puntos 3D en la imagen 2D y resuelve el sistema de ecuaciones correspondiente para hallar los parámetros extrínsecos e intrínsecos. Es necesario indicar el tamaño del cuadrado, y el origen de coordenadas del mundo fijado en una esquina del patrón y los límites del patrón siempre en el mismo orden para todas las posiciones del patrón. Esto es así, para fijar un mismo sistema de coordenadas del mundo para todas las posiciones del patrón. Este algoritmo establece que son necesarias un mínimo de dos posiciones distintas del plano de calibración para la calibración.

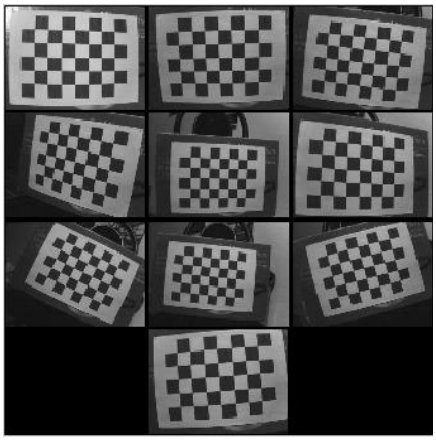


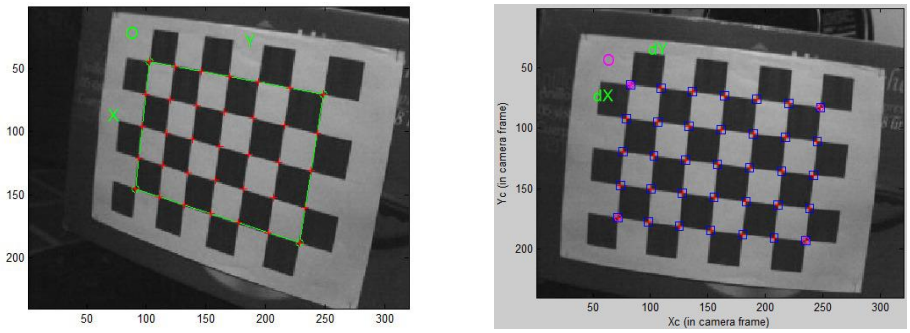
Figura 3.3. Patrón de calibración en 10 poses diferentes.

La calibración se realiza sobre un plano de puntos (patrón de calibración), y aunque no es necesario conocer el desplazamiento que se realiza entre tomas de imágenes, para que el método funcione correctamente se necesitan al menos tres imágenes con el patrón tomado en distintas

orientaciones. Este número de imágenes puede ser inferior si se fijan los valores de algunos parámetros intrínsecos.

Para el proceso de calibración se utiliza una técnica basada en homografías que hace uso de un patrón de calibración, en este caso un tablero de ajedrez (Ver Figura 3.3), del cual se conoce sus dimensiones.

Con el objetivo de hallar las esquinas de cada uno de los patrones se procede utilizar el algoritmo de Harris que básicamente es un detector de esquinas, posteriormente se realiza una localización a nivel sub-pixel de los puntos encontrados por Harris, la Figura 3.4 muestra la detección de esquinas y puntos en un patrón.



a) b)

Figura 3.4. a) Detección de las esquinas del patrón, b) Detección de los puntos internos del patrón

Después del proceso de obtención de las esquinas y puntos de cada uno de los patrones se procede a realizar la calibración con cada uno de los puntos obtenidos iterándolos en cada patrón subsecuente.

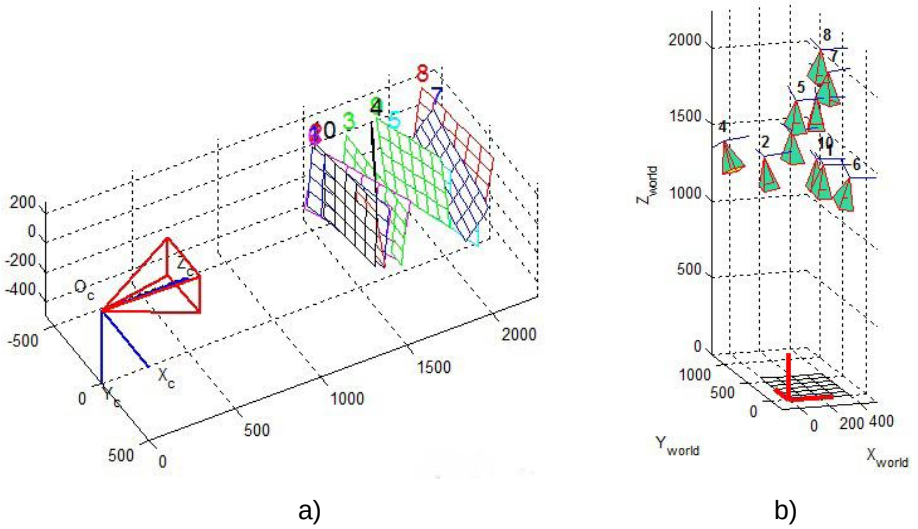
El resultado que entrega el Toolbox son los parámetros la cámara. Los parámetros intrínsecos son:

- Centro del eje óptico ($cc = [u_0, v_0]$): también llamado punto principal. Define el punto donde el eje óptico (zc) atraviesa el plano imagen. Las coordenadas de este punto vienen dadas en píxeles. Se almacena en un vector 2x1 llamado ***"cc"***.
- Distancia focal (f): distancia entre el centro óptico y el centro del plano imagen, viene dada en pixeles. Se almacena en un vector llamado "***fc"***.
- Coeficiente de asimetría: es el ángulo definido por los ejes X y Y del pixel, actualmente las cámaras tienen pixeles cuadrados, dando como valor igual a 90°. Se almacena en una variable escalara llamada ***"alpha_c"***.
- Coeficientes de distorsión: aquí vienen incluidos los coeficientes de distorsión radial y tangencial. Se almacena en un vector 5x1 llamado ***"kc"***.

Mientras que los parámetros extrínsecos están dados por:

- Un vector de rotación de dimensiones 3x1, que mediante el *"algoritmo de rodrigues"* (Taubin y Savarese, 2001), se convierte a una matriz de rotación de 3x3.
- Un vector de traslación (3x1) de que va del marco de referencia del origen del patrón de calibración a la cámara.

En la Figura 3.5 se puede apreciar la relación de los parámetros extrínsecos de la cámara, para cada patrón presentado se conoce su posición y orientación al eje principal, esto nos sirve para lograr una relación entre todos los puntos de los tableros y obtener un modelo de corrección para cualquier punto presentado a futuro.



a) b)

Figura 3.5. a) Vista de las posiciones de los patrones de calibración, b) Vista del patrón de calibración en referencia a la cámara

3.5 Librería de OpenCV

OpenCV (Open source Computer Vision library) es una librería abierta desarrollado por Intel. Esta librería proporciona un alto nivel funciones para el procesado de imágenes. Estas librerías permiten a los programadores crear aplicaciones poderosas en el dominio de la visión digital. Permite realizar: Operaciones básicas, Procesado de imágenes y análisis, Análisis estructural, Análisis de movimiento, Reconocimiento del modelo, Reconstrucción 3d y calibración de la cámara, Interfaz gráfica y adquisición, entre otras.

Es compatible con Intel Image Processing Library (IPL) que implementa algunas operaciones en imágenes digitales. A parte de realizar tareas como binarización, filtrado, estadísticas de la imagen,

cuenta con una librería que implementa algoritmos para las técnicas de la calibración (Calibración de la Cámara), detección de rasgos, para rastrear (Flujo Óptico), análisis de la forma (Geometría, Contorno que Procesa), análisis del movimiento (Plantillas del Movimiento, Estimadores), reconstrucción 3D (Transformación de vistas), segmentación de objetos y reconocimiento (Histograma, etc.).

Para la creación y manejo de imágenes utiliza una estructura denominada como *"Iplimage"*. Esta estructura tiene varios parámetros a configurar, entre esos parámetros se encuentran *"width" q*ue es la anchura del *"Iplimage"*, el *"height"* es la altura, *"depth"* es la profundidad en bits de la imagen y *"nChannels"* el número de canales (uno por cada nivel de gris de las imágenes y tres para las imágenes coloridas).

El software de OpenCV tiene las siguientes convenciones:

- Los identificadores constantes están en mayúsculas; por ejemplo, CV_SEQ_KIND_GRAPH.
- Todos los nombres de las funciones usadas para el procesado de imagen tienen el prefijo del *cv*, por ejemplo: *"cvCreateImage"*, *"cvSobel"*, *"cvAdd".*
- Todas las funciones externas de OpenCV empiezan con el prefijo *cv*, y todas las estructuras con prefijo *Cv*.
- Cada nueva parte de una función empieza con un carácter en mayúscula, por ejemplo: *cvContourTree*.
- Los nombres de funciones en OpenCV tienen el siguiente formato: cv [action] [target] [mod] () donde:
    - Action: la acción indica la funcionalidad del centro, por ejemplo, -Set-, -Create-, -Convert-.
    - Target: indica el área donde el procesado de la imagen está siendo establecido, por ejemplo : -Find Contours, -ApproxPoly.

      En la mayoría de los casos, target consiste en dos o más palabras , por ejemplo: MatchContourTree.

      Algunos nombres de funciones consiste en una acción u objetivo solamente, por ejemplo: las funciones, cvUnDistort o cvAcc respectivamente.
    - mod: es un campo opcional; indica una modificación de la funcionalidad de la función. Por ejemplo: en la función cvFindExtrinsicCameraParams_64d, _64d, indica que es una función constante particular de 64d valores.

En el Apéndice B se muestran algunas de las funciones principales de OpenCV.

3.6 Control del servomotor

Para tener un sistema mecánico, en este caso una base giratoria acoplada a un servomotor en la cual se tengan movimientos de posición precisos, es necesario implementar un sistema de control. Por tal motivo, en esta sección se describe la función de transferencia de un servomotor y se especifican los índices de desempeño para determinar que el sistema sea estable. Más adelante se explica qué y cómo actúa un control PID en un sistema, y por último, se describen los pasos para realizar un ajuste manual de las constantes kp, kd, y ki del controlador PID.

3.6.1 Función de transferencia de un motor de CD

En la industria se usan muchos tipos de motores de cd. Los motores de cd que se usan en los sistemas de seguimiento se denominan servomotores. En los servomotores de cd, la inercia del rotor se ha hecho muy pequeño, por lo que existen en el mercado con razones muy altas entre el par y la inercia. Algunos servomotores de cd tienen constantes de tiempo muy pequeñas. Los servomotores de cd con rangos de corriente muy pequeños se usan en instrumentos y equipo relacionados con computadoras, tales como unidades de disco, unidades de cinta, impresoras y procesadoras de texto. Los servomotores de cd, con razones de corriente mediana y grande, se usan en sistemas robóticos, en máquinas de fresado controladas numéricamente, etc.

A continuación en la Figura 3.6 se muestra un diagrama eléctrico de un servomotor:

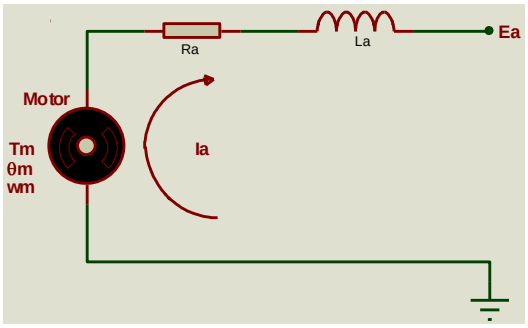


Figura 3.6. Diagrama eléctrico de un servomotor

R_a = Resistencia de armadura

L_a = Inductancia de armadura

I_a = Corriente de armadura

E_a = Voltaje de alimentación

θ_m = Posición angular del eje del motor

w_m = velocidad angular del eje de salida

G = coeficiente del flujo electico motriz

B = coeficiente de fricción

J = coeficiente de inercia

k_T = constante de par del motor

De acuerdo a la relación entre las ecuaciones del circuito eléctrico y el sistema mecánico del servomotor, se obtiene la función de transferencia mostrada en la siguiente ecuación:

$$\frac{\theta_m(s)}{E_a(s)} = \frac{k_T}{L_aJs^3 + (BL_a + R_aJ)s^2 + (BR_a + k_TG)s} \qquad Ec.(3.15)$$

Dicha ecuación se puede simplificar, debido a que la inductancia de la armadura (L_a) es muy pequeña, es decir:

Cuando $L_a \to 0$

$$\frac{\boldsymbol{\theta_m(s)}}{\boldsymbol{E_a(s)}} = \frac{\boldsymbol{k_m}}{\boldsymbol{s(T_ms + 1)}} \qquad Ec.(3.16)$$

$$donde: \quad T_m = \frac{JR_a}{BR_a + k_TG} \qquad k_m = \frac{k_T}{BR_a + k_TG}$$

3.6.1.1 Índices de desempeño para sistemas de segundo orden

La forma general de la función de transferencia de un sistema de segundo orden es:

$$\frac{Y(s)}{U(s)} = \frac{w_n^2}{s^2 + 2\tau w_n s + w_n^2} \qquad Ec.(3.17)$$

Donde: $w_n = frecuencia\ natural$

$\tau = coeficiente\ de\ amortiguamiento$

$s = variable\ compleja$

Aplicando la transformada inversa de Laplace a la forma general de segundo orden, se tiene:

$$y(t) = 1 - e^{-\tau w_n t}\left(\cos w_d t + \frac{\tau}{\sqrt{1-\tau^2}}\sin w_d t\right) \qquad Ec.(3.18)$$

$$w_d = w_n\sqrt{1-\tau^2} \qquad Ec.(3.19)$$

Por lo tanto, los índice de desempeño se pueden obtener a partir de $y(t)$ ó de $Y(s)$.

El tiempo de respuesta: $t_r = \frac{1}{w_d}\left(\pi - \tan^{-1}\left(\frac{\sqrt{1-\tau^2}}{\tau}\right)\right)$ $Ec.(3.20)$

Máximo sobre-impulso: $M_p = e^{-\left(\frac{\tau}{\sqrt{1-\tau^2}}\right)\pi}$ $Ec.(3.21)$

Tiempo de sobre-impulso: $t_{mp} = \frac{\pi}{w_d}$ $Ec.(3.22)$

Tiempo de establecimiento: $t_s = \frac{4}{\tau w_n}$ $Ec.(3.23)$

Error en estado estacionario: $y(\infty) = \lim_{s\to\infty} sY(s)$ $Ec.(3.24)$

3.6.1.2 Análisis de estabilidad

Esta es una característica dinámica que se pretende encontrar en el control de un sistema. En el marco del control automático se utiliza el término estabilidad en el sentido BIBO (Bounded Input, Bounded Output). Es decir un sistema es estable si todas las señales de salida están limitadas. Una señal está limitada si su amplitud no se incrementa hacia ±∞ cuando el tiempo pasa.

La estabilidad es una característica necesaria pero no suficiente para asegurar que el sistema sea estable. Hay que tener en cuenta también los valores numéricos de los índices de desempeño para asegurar el buen funcionamiento de un sistema.

La información esencial para verificar la estabilidad de un sistema se encuentra en su función de transferencia:

$$F(s) = \frac{Y(s)}{U(s)} = \frac{b_0 s^n + b_1 s^{n-1} + \cdots + b_n}{s^n + a_1 s^{n-1} + \cdots + a_n} \qquad Ec.(3.25)$$

Particularmente, en el denominador, llamada también ecuación característica

$$s^n + a_1 s^{n-1} + \cdots + a_n = 0 \qquad Ec.(3.26)$$

Un sistema es estable (en el sentido BIBO) si todas sus raíces del polinomio característico, es decir sus polos, tienen la parte real negativa. Si una o varias raíces tienen parte real positiva o cero, el sistema es inestable.

Bajo este criterio, el sistema (ver Figura 3.7) representado por la función de transferencia del motor en lazo abierto es inestable, ya que tiene un polo igual a cero.

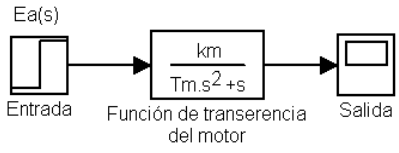


Figura 3.7. Diagrama a bloques de la función de transferencia del motor en lazo abierto

$$\frac{\theta_m(s)}{E_a(s)} = \frac{k_m}{s(T_m s + 1)} \qquad Ec.(3.27)$$

$$s(T_m s + 1) = 0 \ ; \qquad s_1 = 0 \qquad s_2 = -\frac{1}{T_m}$$

Por otro lado, si se cierra el lazo de control (como el que se muestra en la Figura 3.8), el sistema se vuelve estable,

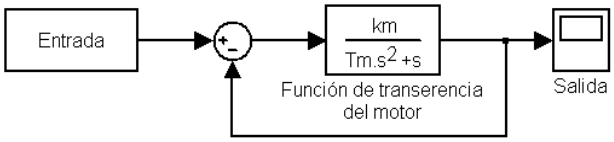


Figura 3.8. Diagrama a bloques de la función de transferencia del motor en lazo cerrado

Entrada: $\theta_r(s)$ Salida: $\theta(s)$

Dando como resultado la función de transferencia:

$$\frac{\theta(s)}{\theta_r(s)} = \frac{\frac{km}{Tm}}{s^2 + \frac{1}{Tm}s + \frac{km}{Tm}} \qquad Ec.(3.28)$$

Calculando los polos *P* de la función de transferencia,

$$P = -\frac{1}{2Tm} \pm \frac{\sqrt{1 - 4T_m k_m}}{2Tm} \qquad Ec.(3.29)$$

Como $T_m > 0 \;\; y \;\; k_m > 0$, entonces *P* siempre es negativo, por lo que el sistema del motor es estable en lazo cerrado.

3.6.2 Control PID

Un controlador PID (ver Figura 3.9) corrige el error entre un valor medido y el valor que se quiere obtener calculándolo y luego sacando una acción correctora que puede ajustar al proceso acorde. El algoritmo de cálculo del control PID se da en tres parámetros distintos: el proporcional, el integral, y el derivativo. El valor Proporcional determina la reacción del error actual. El Integral genera una corrección proporcional a la integral del error, esto nos asegura que aplicando un esfuerzo de control suficiente, el error de seguimiento se reduce a cero. El Derivativo determina la reacción del tiempo en el que el error se produce. La suma de estas tres acciones es usada para ajustar al proceso vía un elemento de control como la posición de una válvula de control o la energía suministrada a un calentador, por ejemplo. Ajustando estas tres variables en el algoritmo de control del PID, el controlador puede proveer un control diseñado para lo que requiera el proceso a realizar. La respuesta del controlador puede ser descrita en términos de respuesta del control ante un error, el grado el cual el controlador llega al "set point", y el grado de oscilación del

sistema. El uso del PID para control no garantiza control óptimo del sistema o la estabilidad del mismo. Algunas aplicaciones pueden solo requerir de uno o dos modos de los que provee este sistema de control. Un controlador PID puede ser llamado también PI, PD, P o I en la ausencia de las acciones de control respectivas. Los controladores PI son particularmente comunes, ya que la acción derivativa es muy sensible al ruido, y la ausencia del proceso integral puede evitar que se alcance al valor deseado debido a la acción de control.

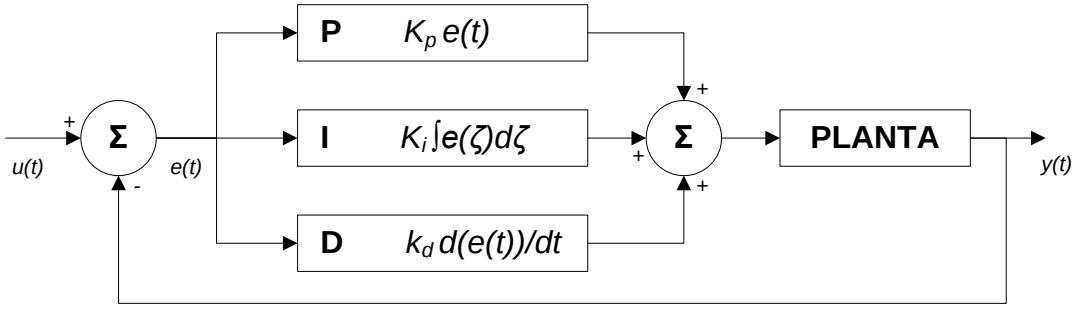


Figura 3.9. Diagrama a bloques del control PID en un sistema

3.6.2.1 Acción de control proporcional

La parte proporcional (ver Figura 3.10) es el producto entre la señal de error y la constante proporcional, para hacer que el error en estado estacionario sea casi nulo, en la mayoría de los casos, estos valores solo serán óptimos en una determinada porción del rango total de control. La parte proporcional no considera el tiempo, por lo tanto, la mejor manera de solucionar el error permanente y hacer que el sistema contenga alguna componente que tenga en cuenta la variación respecto al tiempo, es incluyendo y configurando las acciones integral y derivativa.

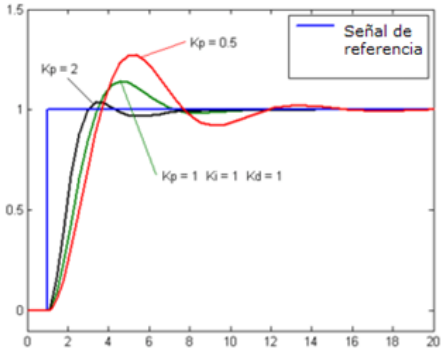


Figura 3.10. Curva de respuesta de la acción de control proporcional (P) en un sistema

3.6.2.2 Acción de control integral

La acción control Integral (ver Figura 3.11) disminuye y elimina el error en estado estacionario, provocado por el modo proporcional. Actúa cuando hay una desviación entre la variable y el punto de consigna, integrando esta desviación en el tiempo y sumándola a la acción proporcional.

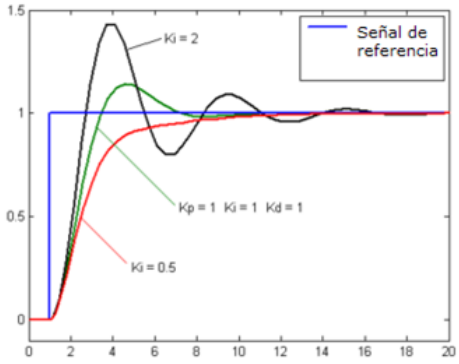


Figura 3.11. Curva de respuesta de la acción de control integral (I) en un sistema

3.6.2.3 Acción de control derivativo

La acción derivativa (ver Figura 3.12) se visualiza cuando hay un cambio en el valor absoluto del error; (si el error es constante, solamente actúan los modos proporcional e integral). El error es la desviación existente entre el punto de medida y el valor consigna. La acción derivativa es mantiene el error al mínimo corrigiéndolo proporcionalmente con la misma velocidad que se produce; de esta manera evita que el error se incremente.

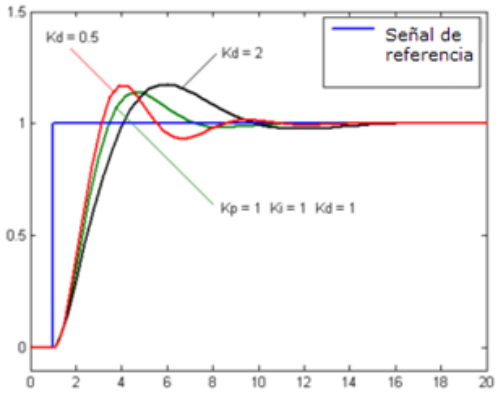


Figura 3.12. Curva de respuesta de la acción de control derivativo (D) en un sistema

3.6.3 Ajuste del control PID

El objetivo de los ajustes de los parámetros PID es lograr que el bucle de control corrija eficazmente y en el mínimo tiempo los efectos de las perturbaciones; se tiene que lograr la mínima integral de error. Si los parámetros del controlador PID (la ganancia del proporcional, integral y

derivativo) se eligen incorrectamente, el proceso a controlar puede ser inestable, por ejemplo, que la salida de este varíe, con o sin oscilación, y está limitada solo por saturación o rotura mecánica. Ajustar un lazo de control significa ajustar los parámetros del sistema de control a los valores óptimos para la respuesta del sistema de control deseada. El comportamiento óptimo ante un cambio del proceso o cambio del "setpoint" varía dependiendo de la aplicación. Generalmente, se requiere estabilidad ante la respuesta dada por el controlador, y este no debe oscilar ante ninguna combinación de las condiciones del proceso y cambio de "setpoints". Algunos procesos tienen un grado de no-linealidad y algunos parámetros que funcionan bien en condiciones de carga máxima no funcionan cuando el proceso está en estado de "sin carga". Hay varios métodos para ajustar un lazo de PID. El método más efectivo generalmente requiere del desarrollo de alguna forma del modelo del proceso, luego elegir P, I y D basándose en los parámetros del modelo dinámico. Los métodos de ajuste manual pueden ser muy ineficientes. La elección de un método dependerá de si el lazo puede ser "desconectado" para ajustarlo, y del tiempo de respuesta del sistema. Si el sistema puede desconectarse, el mejor método de ajuste a menudo es el de ajustar la entrada, midiendo la salida en función del tiempo, y usando esta respuesta para determinar los parámetros de control. Ahora describimos como realizar un ajuste manual.

Si el sistema debe mantenerse online, un método de ajuste consiste en establecer primero los valores de I y D a cero. A continuación, incremente P hasta que la salida del lazo oscile. Luego establezca P a aproximadamente la mitad del valor configurado previamente. Después incremente I hasta que el proceso se ajuste en el tiempo requerido (aunque subir mucho I puede causar inestabilidad). Finalmente, incremente D, si se necesita, hasta que el lazo sea lo suficientemente rápido para alcanzar su referencia tras una variación brusca de la carga.

Un lazo de PID muy rápido alcanza su set-point de manera veloz. Algunos sistemas no son capaces de aceptar este disparo brusco; en estos casos se requiere de otro lazo con un P menor a la mitad del P del sistema de control anterior.

Capítulo 4. Descripción de la base experimental

En este capítulo determinaremos el diseño mecánico para obtener la mayor cantidad de información de las vistas de un objeto. Por lo tanto, al comenzar con el diseño mecánico, se definen características importantes como dimensiones de los elementos y mecanismos, así como el tipo de material a utilizar. Además, para automatizar el sistema mecánico, se da una breve descripción de los componentes electrónicos adecuados para realizar la conexión entre ellos, y así controlar el sistema mecánico desde la computadora.

Por otro lado, se describen las técnicas de adquisición y procesamiento de imágenes para realizar la reconstrucción tridimensional del objeto, dicha reconstrucción se representa mediante una nube de puntos. A partir de la nube de puntos y con una previa calibración de la cámara se determina el volumen del objeto.

Por último, se describe la interfaz gráfica entre el usuario y el prototipo, desarrollada con Visual C++. ´Con dicha interfaz se automatiza y sincroniza el sistema mecánico con el sistema de adquisición de imágenes. Posteriormente se determina la reconstrucción tridimensional del objeto, dando como resultado final el valor de la estimación del volumen del objeto.

4.1 Diseño del sistema mecánico

En la sección 2.3 se describen algunas técnicas utilizadas para la reconstrucción tridimensional, dentro de estas técnicas, se tiene como principal objetivo la adquisición y el procesamiento de las imágenes en diferentes poses del objeto. Por lo tanto, para llevar a cabo la adquisición de las imágenes en diferentes poses del objeto es necesaria la construcción de un sistema mecánico que nos permita su desplazamiento o rotación.

Se tienen principalmente dos formas para diseñar y construir el sistema mecánico de acuerdo a los requerimientos necesarios para llevar a cabo la adquisición de imágenes. Una es dejando fijo el objeto y moviendo la cámara de video (o el objeto a reconstruir) en forma lineal, lo poca información de las poses del objeto, solo una vista es analizada. La segunda forma es girando o rotando la cámara de video (o el objeto a reconstruir), de este modo se obtienen poses del objeto desde 0° a 360°, teniendo más información para la reconstrucción tridimensional. Por lo tanto, se eligió la segunda opción, diseñando (con la ayuda del software SolidWorks) el sistema mecánico en donde va colocado el objeto (ver Figura 4.1).

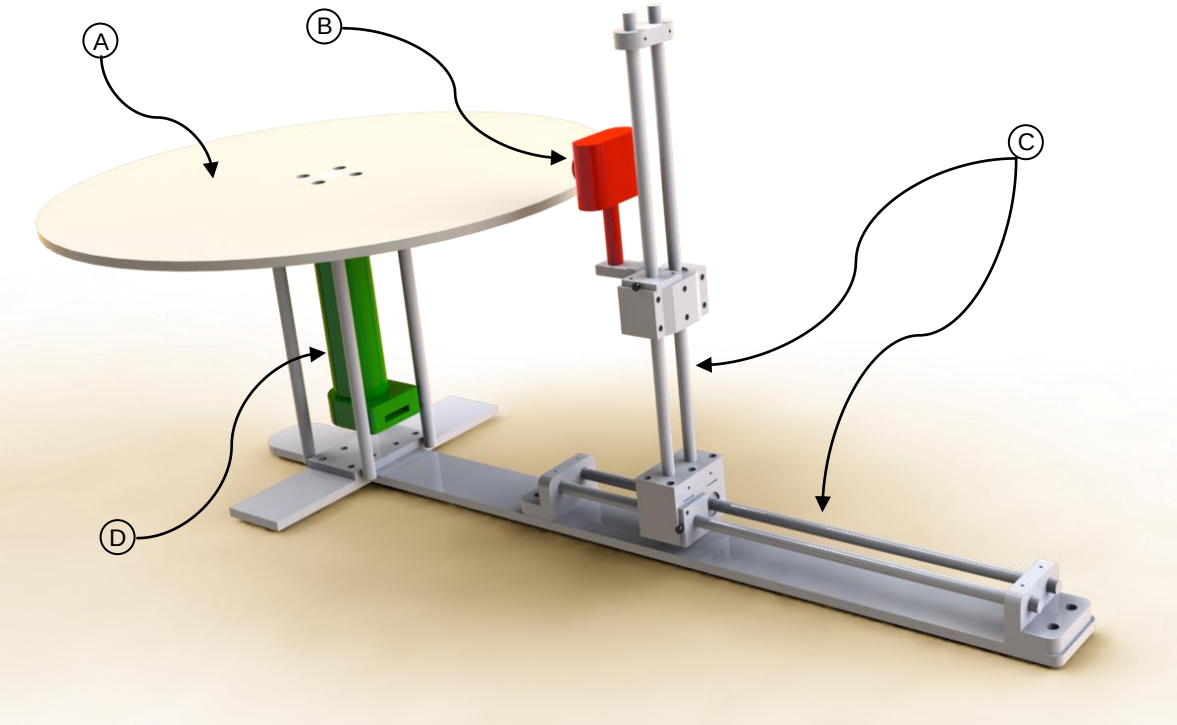


Figura 4.1. Sistema mecánico para la adquisición de imágenes

ELEMENTOS DEL SISTEMA MECÁNICO PARA LA ADQUISICIÓN DE IMÁGENES	
A	Base giratoria para rotar el objeto
B	Webcam USB Microsoft
C	Rieles lineales para el ajuste de la Webcam
D	Servomotor con encoder

Tabla 4.1. Elementos del sistema mecánico para la adquisición de imágenes

Dicho sistema mecánico (ver Tabla 4.1) tiene tres grados de movilidad, dos de ellos se consiguen gracias a rieles lineales (***C***), mientras que el tercero es rotacional (***A***). Sobre los dos rieles mecánicos se monta el dispositivo de adquisición de imágenes, en este caso una Webcam Usb Microsoft (***B***) como se muestra en la Figura 4.2. La finalidad de los dos rieles es ajustar la posición de la Webcam con respecto al objeto a reconstruir. Por último se tiene acoplado un servomotor con encoder (***D***) sobre la base giratoria.

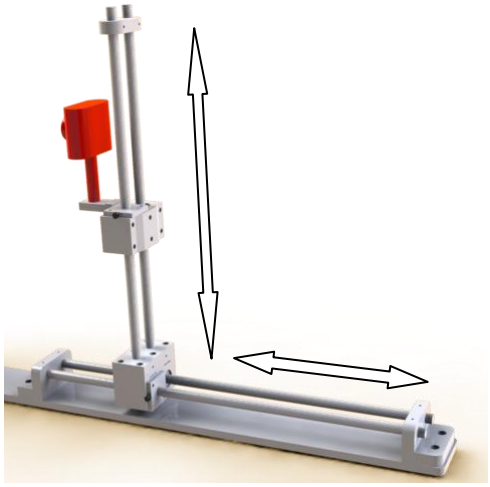


Figura 4.2. Rieles lineales para el ajuste de posición de la Webcam

Cada riel consta de dos flechas de acero inoxidable de 6mm de diámetro y 30cm de largo. Sobre dichos rieles se deslizan dos baleros lineales acoplados a una chumacera doble (ver Figura 4.3).

Figura 4.3. Chumacera doble

Por otro lado, se tiene el acoplamiento entre el servomotor marca MAXON modelo 281272 y la base giratoria. El cual está conformado por una base para fijar el motor, y un cople que transfiere el movimiento del eje del servomotor a la base giratoria, como se muestra en la Figura 4.4.

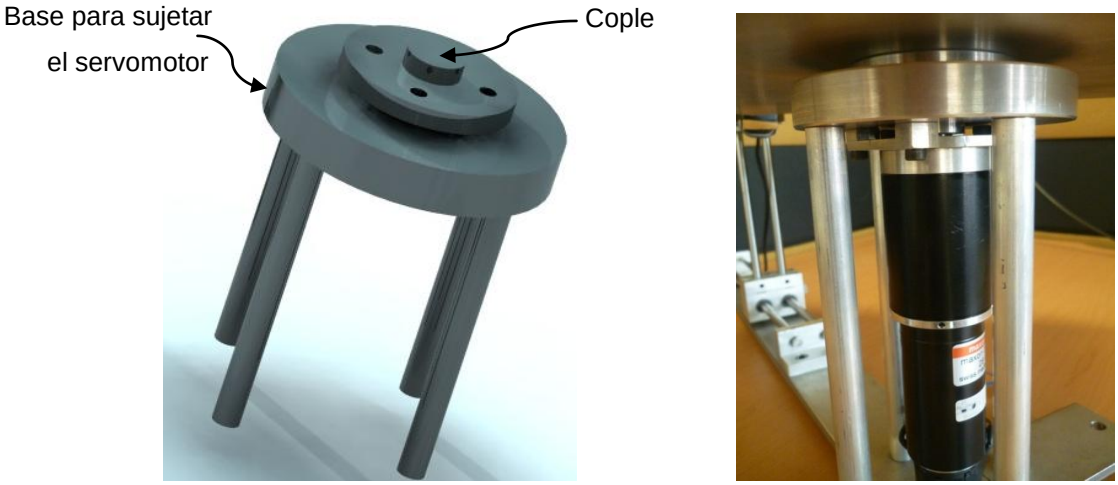


Figura 4.4. Acoplamiento entre el servomotor y la base giratoria

Finalmente, se construyó una estructura de Perfil Tubular Rectangular (PTR) cerrada con placas de acrílico blanco, con la finalidad de proyectar luz blanca por la parte de atrás de la estructura (según la Tabla 3.1 de las técnicas de iluminación) para generar imágenes de alto contraste y se facilite el procesamiento de información, como se muestra en la Figura 4.5.

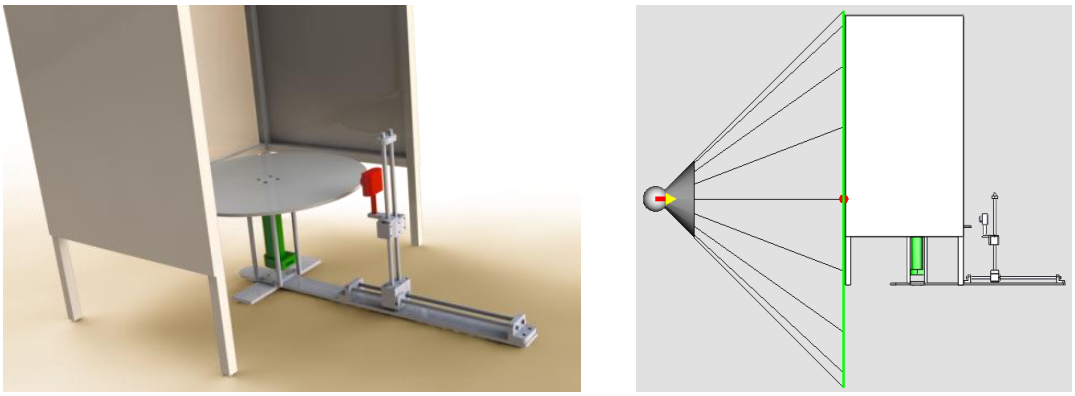


Figura 4.5. Estructura de PTR para aplicar la técnica de iluminación posterior

4.2 Implementación electrónica para la automatización del sistema mecánico

El sistema de control para la automatización del sistema mecánico, está conformado de componentes electrónicos que nos permiten recibir y enviar señales digitales de control desde la computadora a través del puerto USB. Al mismo tiempo, recibe y envía las señales que llegan desde los encoders y las señales de potencia hacia los servomotores.

4.2.1 Tarjetas PIC-SERVO SC y SSA-485

El sistema de control consta de una tarjeta PIC-SERVO SC y una tarjeta SSA-485. La tarjeta SSA-485 es un convertidor de protocolo de comunicación USB/RS232 a RS485 que permite realizar la comunicación con la tarjeta PIC-SERVO SC a través del puerto USB de la PC.

La tarjeta PIC-SERVO SC es un sistema completo de control en lazo cerrado con las siguientes características:

- Proporciona control en lazo cerrado de motores de CD con encoders incrementales, incluyendo perfiles trapezoidales de velocidad y movimientos multi-ejes coordinados.
- Posee amplificadores LMD18200 con la capacidad de manejar 3A continuamente y 6A pico hasta 48 VCD.
- Sensado de corriente, limitación activa de corriente, y protección contra sobre-voltaje.

- Provee señales PWM y DIR para usarlas con amplificadores externos, en caso de requerir mayor potencia de salida.
- La interface RS485 permite hasta 32 PIC-SERVO SC a partir de un solo puerto USB.
- Ofrece dos entradas para interruptores de fin de carrera.
- Incluye software de prueba del tipo 32 bit Windows DL y código fuente para lenguaje C y BASIC.

En la Figura 4.6, se muestra el diagrama a bloques de la tarjeta de control PIC-SERVO SC:

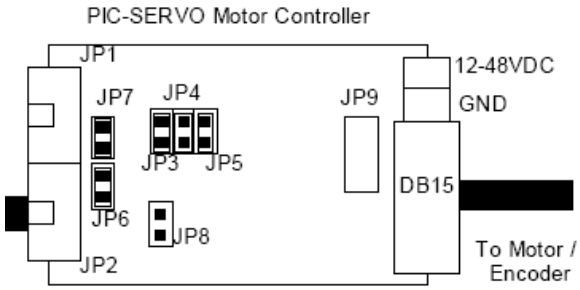


Figura 4.6. Diagrama a bloques de la PIC - SERVO SC

El conector JP2 se conecta a la salida de la tarjeta SSA-485, que viene siendo el convertidor de USB a RS485. Por otro lado, el conector DB15 tiene la función de conectar el servomotor y con el encoder. En la Tabla 4.2 se describen cada uno de los pines del conector DB15:

Pin	Definición
1	Salida positiva al motor (M+)
2	Salida positiva al motor (M+)
3	Led indicador de alimentación
4	Limit Switch (LM1)
5	Canal A del encoder
6	Canal B del encoder
7	Limit Switch (LM2)
8	Canal Index del encoder
9	Salida negativa al motor (M-)
10	Salida negativa al motor (M-)
11	GND
12	GND
13	GND del encoder
14	Alimentacion del encoder, 5V
15	GND

Tabla 4.2. Pines del conector DB15

De acuerdo con las hojas características de la PIC – SERVO SC, es necesario alimentarla con un voltaje entre 12 – 48 VCD, esto va dependiendo del servomotor a utilizar, ya que este voltaje alimenta la etapa de potencia que suministra el voltaje y corriente del servomotor. De acuerdo a las pruebas realizadas, con una fuente de voltaje de 12 VCD, es suficiente para mover el mecanismo diseñado.

En la Figura 4.7, se muestra un diagrama a bloques de la forma en cómo conectar la tarjeta del convertidor y la tarjeta de control del servomotor.

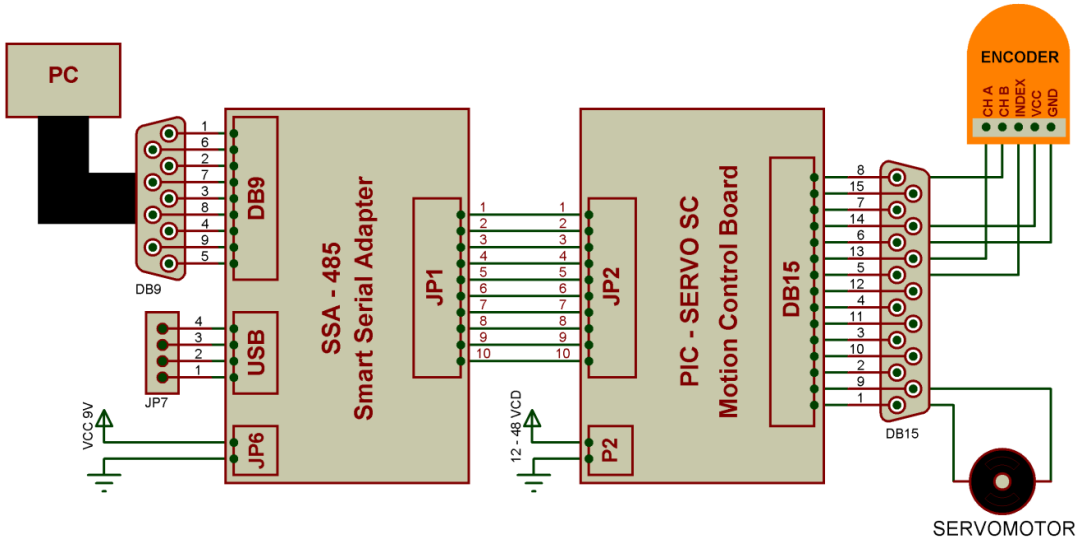


Figura 4.7. Diagrama a bloques de conexión entre la tarjeta de control, servo, encoder y computadora

Información adicional se puede consultar en las siguientes fuentes:

- JEFFREY KERR, Sitio Web: www.jrkerr.com
- Microchip, PIC18F2331, Sitio Web: www.microchip.com
- National Semiconductor, LMD18200, Sitio Web: www.national.com
- Microchip, PIC18Fxxxx Sitio Web: www.microchip.com
- FTDI Chip y drivers para FTB232BM USB, Sitio Web: www.ftdichip.com

4.2.2 Encoder HEDS 5540C11

El encoder es un transductor rotativo que transforma un movimiento angular en una serie de impulsos digitales, estos impulsos generados son utilizados para el control de desplazamiento (tipo angular o lineal). El encoder que se utilizó es de la marca Agilent Technologies modelo HEDS 5540C11 (ver Figura 4.8), es de tres canales (chanel A, chanel B e Index), tiene una resolución de 100 cuentas o impulsos por revolución, mientras que el servomotor MAXON 281272 tiene acoplado un reductor 74:1.

Figura 4.8. Encoder HEDS 5540C11

El encoder, es un dispositivo que codifica información del desplazamiento y su dirección, normalmente el mínimo desplazamiento es decodificado a partir de un ciclo completo de la señal A o B. Se basa en la rotación de un disco graduado con un retículo radial formado por espacios opacos, alternados con espacios transparentes. Un sistema óptico de emisor receptor infrarrojo detecta el cambio en la superficie del disco, generando dos señales en cuadratura (defasadas 90°), las señales se identifican como A y B.

Para obtener el número total de cuentas de encoder acoplado al servomotor con reductor, se multiplica la resolución del encoder, por el valor del reductor por 4 (debido a que se tienen dos señales en cuadratura). $Ec.(4.1)$

$$cuentas\ de\ encoder\ por\ revolución\ de\ la\ base\ giratoria\ =$$

$$(cuentas\ por\ revolución\ del\ encoder)\ *(valor\ del\ reductor)*(4) = 100\ x\ 74\ x\ 4 = 29400$$

Por lo tanto, para girar la base a 1°, se deben de contar aproximadamente 82 cuentas.

4.2.3 Fuente de alimentación

En electrónica, una fuente de alimentación es un dispositivo que convierte el voltaje de corriente alterna de la red de suministro, en uno o varios voltajes de corriente directa, que alimentan los distintos circuitos del aparato electrónico al que se conecta (computadora, televisión, impresora, router, etc.). De manera general, una fuente de alimentación está compuesta por las siguientes partes:

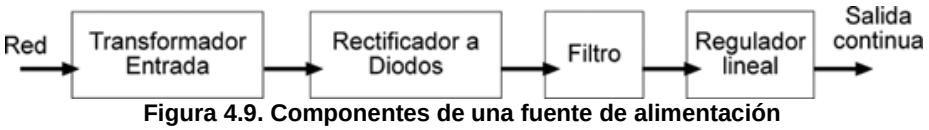


Figura 4.9. Componentes de una fuente de alimentación

De esta manera, de acuerdo con las especificaciones de la tarjeta de control PIC–SERVO SC y del convertidor RS232 a RS485, deben de ser alimentadas con voltaje de corriente directa. Por tal motivo, para la tarjeta del convertidor se conecta a una fuente de voltaje de 12VCD a 20W, ya que

no demanda mucha corriente. Mientras que para la PIC–SERVO SC se alimenta con otra fuente de voltaje de 12VCD, pero ésta a 40W, que equivale a suministrar una corriente aproximadamente de 3A (ver Figura 4.10).

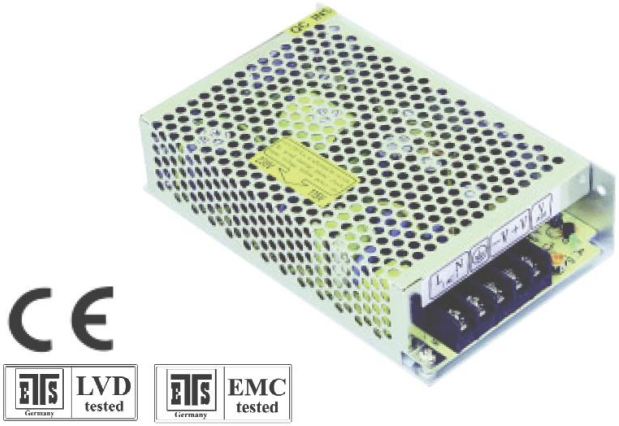


Figura 4.10. Fuente de alimentación

La potencia de la fuente de alimentación que se conecta a la PIC–SERVO SC es de primordial importancia, ya que está en función de la corriente que demande el servomotor.

4.3 Algoritmo para la estimación del volumen del objeto

La estrategia para obtener el volumen del objeto consiste en desarrollar un algoritmo que realice en primer lugar, la reconstrucción tridimensional del objeto, y posteriormente estime su volumen. Por lo tanto, con la base giratoria se adquieren N imágenes de las siluetas del objeto, con incrementos de giro constante, como se muestra en la Figura 4.11.

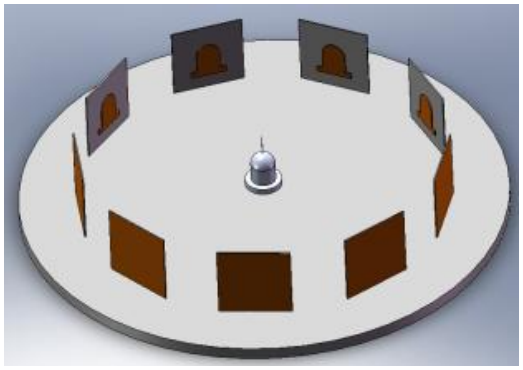


Figura 4.11. Silueta del objeto en diferentes posiciones

Posteriormente, se crean nubes de puntos del sub-volumen formado por la proyección de cada una de las siluetas. De este modo, para obtener la reconstrucción 3D del objeto basta con realizar la

intersección de todos los sub-volúmenes de las siluetas proyectadas. En la Figura 4.12 se muestra la intersección entre dos sub-volúmenes de dos siluetas y las proyecciones de los sub-volúmenes de todas las siluetas.

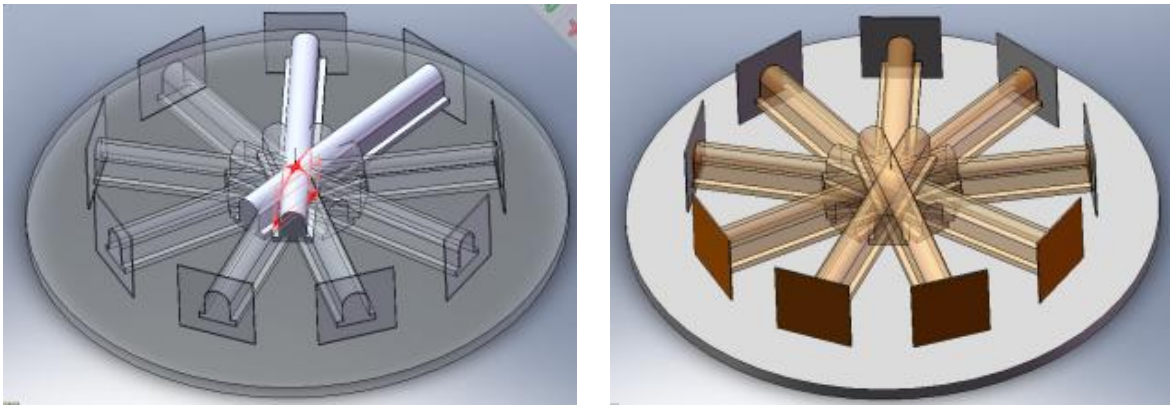


Figura 4.12. a) Proyección volumétrica de dos vistas, b) Proyección volumétrica de todas las vistas,

Para la adquisición de imágenes se utiliza una WebCam Microsoft, la cual nos proporciona imágenes con una resolución de 352x288 pixeles

En primer lugar, se realiza la extracción de las siluetas del objeto, mediante la segmentación de la imagen, es decir, se aplica un umbral a la imagen para separar la silueta del fondo. Dicho umbral se obtiene mediante el algoritmo de Otsu, el cual elige el umbral óptimo maximizando la varianza entre clases (between-class variance) mediante una búsqueda exhaustiva, como se muestra en la Figura 4.13.

Figura 4.13. Silueta del objeto aplicando el método de Otsu

Después de obtener la siluetas del objeto, se extraen las coordenadas de los pixeles *(sy,sz)* donde se encuentra la silueta, y se almacenan en matrices S_i, donde *i = 1, 2, …, N*, *N* corresponde al número de imágenes adquiridas.

$$Si=[sy_i \; sz_i] \; , \; sx_i=0 \qquad Ec.(4.2)$$

Posteriormente, con las matrices S_i se obtienen las nubes de puntos en 2D (ver Figura 4.14), correspondientes a las siluetas.

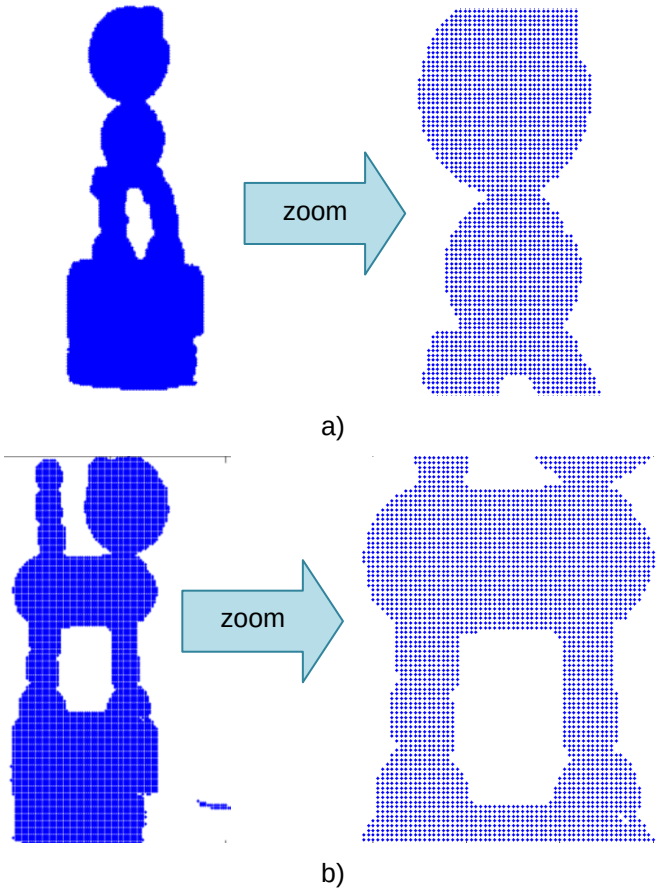


Figura 4.14. Nube de puntos en 2D de las siluetas a) a 0° de giro, b) a 90° de giro

Una vez que se tienen las nubes de puntos S_i, se obtienen las coordenadas tridimensionales (como se muestra en Figura 4.15) de la silueta de acuerdo al ángulo θ_i con el que fue obtenida la imagen respectiva, es decir, $P_i = (xp_i, \quad yp_i, \quad zp_i)$

$$\begin{bmatrix} xp_i \\ yp_i \\ zp_i \end{bmatrix} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 \\ \sin\theta_i & \cos\theta_i & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} sx_i \\ sy_i \\ sz_i \end{bmatrix} \qquad Ec.(4.3)$$

Con $\theta_i = \theta_{i-1} + \alpha$, donde α igual a un incremento constante.

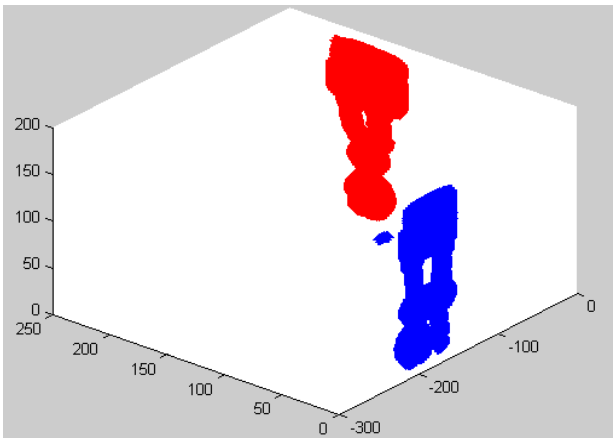


Figura 4.15. Rotación de dos nubes de puntos correspondientes a las siluetas del objeto a 0° (rojo) y a 90° (azul) respectivamente

Las coordenadas de los puntos que pertenecen al volumen parcial del objeto $V_i = (xv_{i,k},\ yv_{i,k},\ zv_{i,k})$, son generados por *M* traslaciones T_k, donde $k = 1,\ldots,M$ (*M* depende del tamaño de la imagen), que son normales al plano de proyección *Pi*, es decir:

$$\begin{bmatrix} xv_{i,k} \\ yv_{i,k} \\ zv_{i,k} \end{bmatrix} = \begin{bmatrix} xp_i \\ yp_i \\ zp_i \end{bmatrix} + \begin{bmatrix} k\lambda \sin(\theta_i + \pi/4) \\ k\lambda \cos(\theta_i + \pi/4) \\ 0 \end{bmatrix} \qquad Ec.\,(4.4)$$

donde λ es un incremento constante que sirve para garantizar la densidad de la malla del volumen generado por la silueta (ver Figura 4.16).

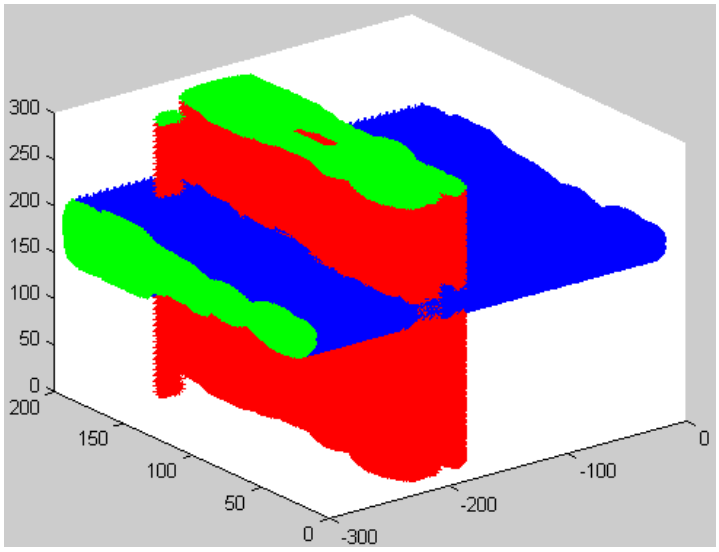


Figura 4.16. Proyección volumétrica de las siluetas a 0° (azul) y 90° (rojo) respectivamente

Por lo tanto, el volumen total generado está dado por la intersección de los N volúmenes parciales V_i, es decir:

$$V_T = \bigcap_{i=1,\dots,N} V_i \qquad Ec.(4.5)$$

A cada elemento de la nube de puntos de V_T, se le llama vóxel (ver Figura 4.17), que es la unidad cúbica que compone un objeto tridimensional y constituye la unidad mínima procesable de una matriz tridimensional y es, por tanto, el equivalente del píxel en un objeto 2D.

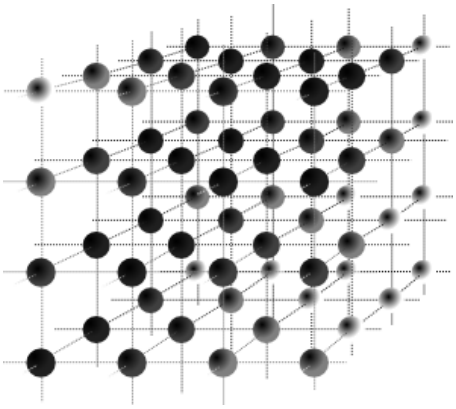


Figura 4.17. Representación tridimensional de vóxeles

Una vez que se tiene la nube de puntos del objeto reconstruido, se continúa con la visualización del objeto, es decir, con la generación de una malla a partir de los puntos. Para llevar a cabo la visualización del objeto, es necesario realizar la triangulación de Delaunay, que es el proceso mediante el cual se generan triángulos entre los puntos para generar un mallado del objeto 3D como se muestra en la Figura 4.18.

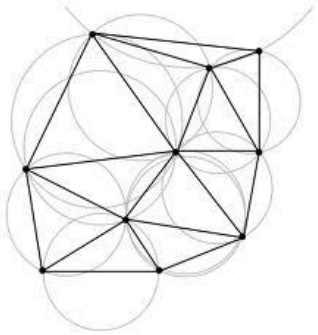


Figura 4.18. Triangulación de Delaunay para generación del mallado de una nube de puntos

Debido a que la reconstrucción 3D genera una nube de puntos que van desde la superficie hasta la parte interna del objeto, la triangulación de Delaunay puede ser muy lenta e inexacta por el procesamiento de datos. Por tal motivo, se implementa un algoritmo para eliminar los puntos internos de la nube y dejar sólo los puntos de la parte superficial del objeto.

Este algoritmo está basado en dividir la nube de puntos en cortes o capas, como se muestra en la Figura 2.2. Como la nube de puntos es un arreglo matricial tridimensional, estas divisiones se obtienen tomando capas de igual altura (un pixel).

Posteriormente, a partir de una capa, se obtiene una silueta de una sección del objeto, como si fuera su sección transversal. A dicha silueta se le aplica el algoritmo de Canny, que consiste en determinar únicamente el contorno de la silueta.

El conjunto de todas las coordenadas de los contornos representa la nube de puntos sólo de la parte superficial del objeto reconstruido. De esta manera ahora se puede aplicar algún método de triangulación, con la ventaja de que su procesamiento será más rápido.

Por lo tanto, teniendo la nube de puntos del volumen total del objeto reconstruido V_T y con los parámetros de la Webcam que nos entregue el Toolbox de calibración de MATLAB, se encuentra una equivalencia para ver cuánto equivale un vóxel a unidades de volumen (mm^3, cm^3, m^3). Por lo tanto, para estimar el volumen del objeto, solo es necesario saber la cantidad de vóxeles que tiene la nube de puntos y calcular su equivalencia a unidades de volumen.

4.4 Desarrollo de la interfaz gráfica

Esta es desarrollada en lenguaje C++, debido a que el procesamiento de imágenes es más rápido ya que la programación es de lenguaje de alto nivel, y por otro lado, debido a que la tarjetas de control PIC - SERVO SC cuenta con librerías desarrolladas en Visual C++, Borland C ó Visual Basic. En la Figura 4.19, se muestra el entorno del software que utilizaremos para la implementación de la interfaz gráfica (Visual Studio),

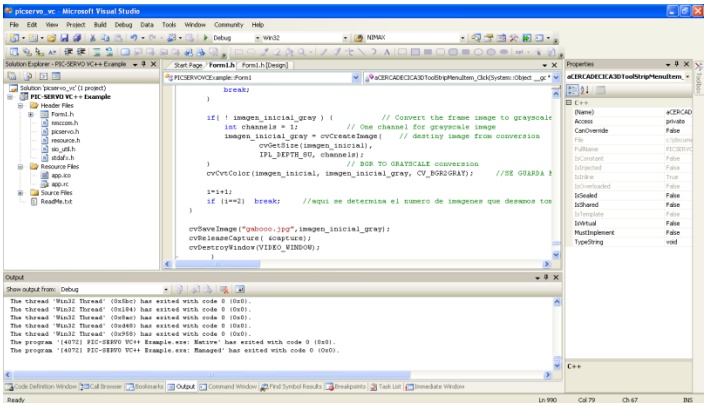


Figura 4.19. Ventana de Visual C++

La interfaz grafica se realizó siguiendo los pasos del diagrama a bloques mostrado en la Figura 4.22. El diseño de la interfaz gráfica se muestra en la Figura 4.20, consta de 3 menús: INICIO, ADQUISICIÓN DE IMÁGENES y AYUDA.

Figura 4.20. Interfaz de usuario, a) Menú INICIO, b) Menú ADQUISICIÓN DE IMAGEN, c) Menú AYUDA

En el Menú INICIO (ver Figura 4.21), lo que encontraremos es la configuración para habilitar la tarjeta de control PIC SERVO SC, en primer lugar se selecciona el puerto [COM1, COM15] para comunicar la computadora con la tarjeta. Posteriormente, se Inicializa la conexión entre la tarjeta y la computadora, manda un mensaje dependiendo del estado en que se encuentre, puede ser MÓDULO ENCONTRADO ó MÓDULO NO ENCONTRADO.

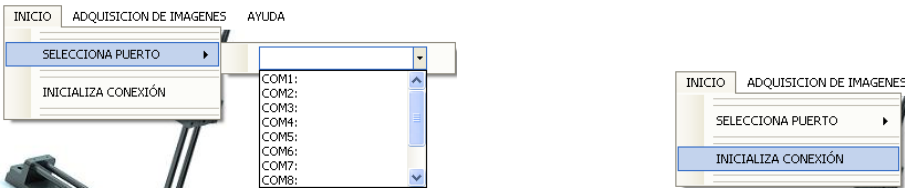


Figura 4.21. Menú INICIO, a) Selección del puerto, b) Inicializa conexión

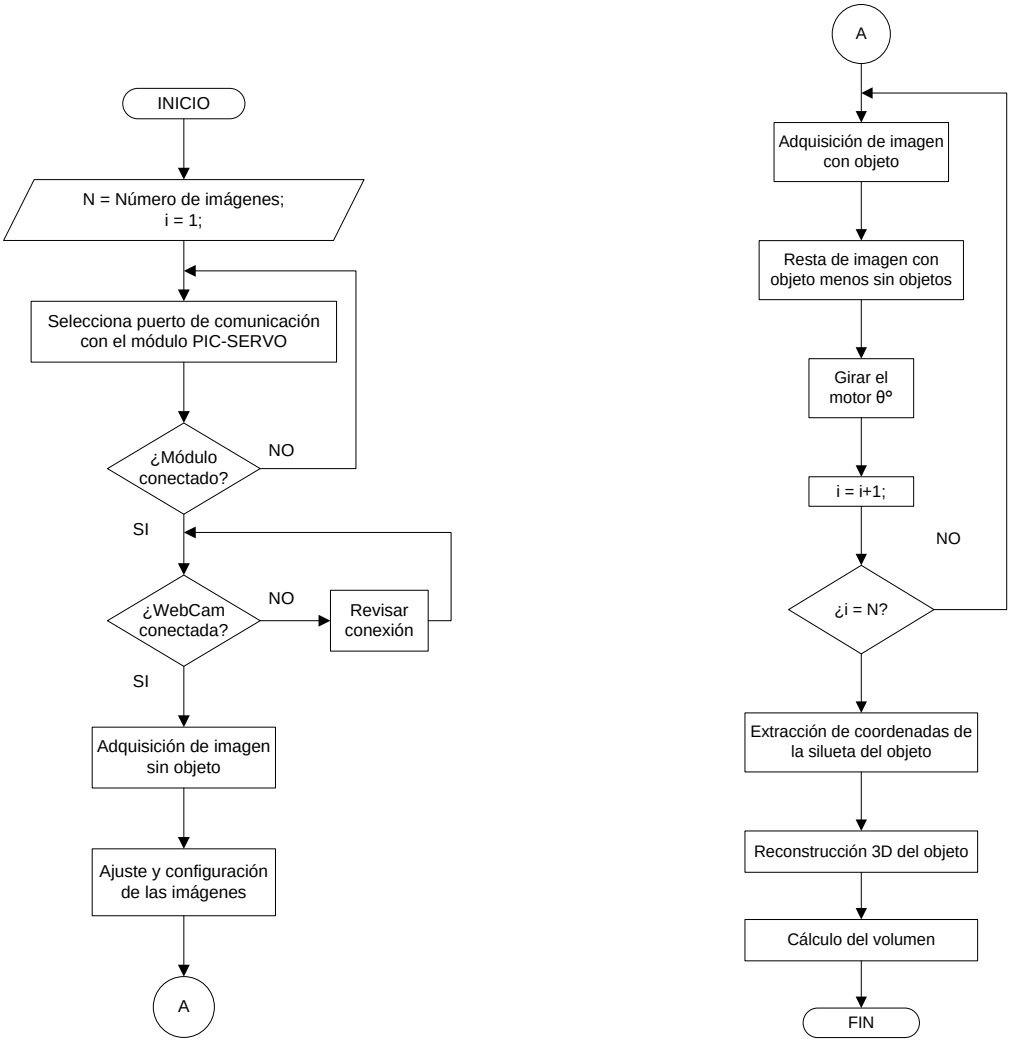


Figura 4.22. Diagrama a bloques del diseño de la Interfaz gráfica

El Menú ADQUISICIÓN DE IMÁGENES (ver Figura 4.23), cuenta con los sub-menús: visualización sin objeto, ajuste de video, adquisición de imágenes, extracción de coordenadas, reconstrucción 3D, y por último cálculo del volumen.

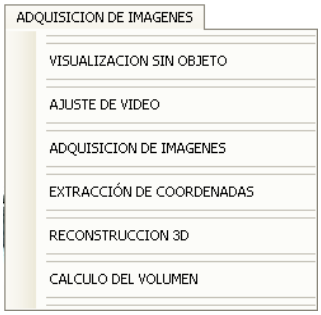


Figura 4.23. Menú de ADQUISICIÓN DE IMÁGENES

Entre los dos primeros sub-menús (Visualización sin objeto – Ajuste de video) lo que se realiza es primero adquirir una imagen sin el objeto montado sobre la base giratoria, posteriormente se adquiere de igual manera una imagen pero ahora con el objeto montado. Teniendo las dos imágenes, se obtiene el valor absoluto de la resta entre ambas imágenes.

$$RESTA_i = \left|(\text{Imagen con objeto})_i - (\text{Imagen sin objeto})\right|, \qquad i = 1,\ldots,N \qquad Ec.\,(4.6)$$

Donde N es el número de imágenes a adquirir

El resultado de esta operación (Ver Figura 4.24) es tener en la imagen únicamente el objeto montado sobre la plataforma.

Figura 4.24. Visualización del objeto a reconstruir

De acuerdo con la imagen mostrada en la Figura 4.24, sigue el proceso de umbralización para poder determinar la silueta del objeto, la umbralización es una técnica de segmentación ampliamente utilizada en las aplicaciones industriales. Se emplea cuando hay una clara diferencia entre los objetos a extraer respecto del fondo de la escena. Los principios que rigen son la similitud entre los píxeles pertenecientes a un objeto y sus diferencias respecto al resto. Por tanto, la escena debe caracterizarse por un fondo uniforme y por objetos parecidos.

Por lo tanto, el sub-menú Adquisición de imágenes consiste en obtener las imágenes umbralizadas del objeto en cada una de las posiciones de giro de la base. Es decir, al presionar este botón, se activa la parte del movimiento del motor, la adquisición y pre-procesamiento de las imágenes de manera sincronizada. Con esto se obtiene en cada una de las imágenes la silueta del objeto en diferentes posiciones de giro. También en este apartado se definen el número de imágenes a adquirir (N imágenes).

En el sub-menú Extracción de Coordenadas, por cada una de las imágenes umbralizadas (con la silueta del objeto en diferente posición), se obtienen las coordenadas de los pixeles donde existe silueta. Por cada silueta, se crea un archivo de texto con las coordenadas de la silueta.

En el sub-menú Reconstrucción 3D, se desarrolló un algoritmo para poder obtener la reconstrucción tridimensional del objeto.

Para el desarrollo de la interfaz en Visual C++, como se observó anteriormente es necesario realizar adquisición y procesamiento de imágenes (adquisición de imágenes con cámara web, resta de imágenes, binarizar una imagen, entre otras operaciones), para realizar este tipo de operaciones y aplicaciones se utilizó OpenCV (Open source Computer Vision library) que es una librería abierta desarrollada por Intel. Esta librería proporciona un alto nivel de funciones para el procesado de imágenes. Estas librerías permiten a los programadores crear aplicaciones poderosas en el dominio de la visión digital. OpenCv permite realizar Operaciones básicas, Procesado de imágenes y análisis, Análisis estructural, Análisis de movimiento, Reconocimiento del modelo, Reconstrucción 3d y calibración de la cámara, Interfaz gráfica y adquisición.

En el Apéndice C, se muestra el código del algoritmo para la reconstrucción tridimensional, y a su vez la interfaz de usuario, todo esto desarrollado en Visual C++ y con librerías de OpenCV.

Capítulo 5. Validación del sistema

En este capítulo se describen los experimentos llevados a cabo para validar algunas de las características del sistema diseñado. En la primera parte se demuestra la efectividad del algoritmo de reconstrucción tridimensional al colocar el objeto fuera del eje de giro de la base experimental. En la segunda parte se realiza una comparación entre dos técnicas diferentes para la reconstrucción 3D del objeto (utilizando solo una cámara y utilizando dos cámaras). Mientras que en la tercera parte, se determina el volumen de los objetos mediante la reconstrucción 3D del objeto.

5.1 Algoritmo de reconstrucción 3D.

Como se ha descrito en el capítulo 4, el algoritmo desarrollado en este trabajo está basado en colocar el objeto sobre una base giratoria sin importar que el centro del objeto coincida con el centro de giro de la base. Por tal motivo en esta sección se verifica que al algoritmo de reconstrucción 3D implementado no le afecta la posición del objeto sobre la base giratoria.

En la parte superior de la Figura 5.1 se muestra la vista superior de un objeto colocado sobre una base giratoria. Dicho objeto tiene su centro de masa alejado a 30 unidades del centro de giro de la base. Mientras que en la parte inferior de la Figura 5.1 corresponde a la proyección del objeto sobre el plano imagen, en este caso corresponde a la silueta del objeto. En este ejemplo se propone como objeto a reconstruir un cilindro de 20 unidades de diámetro.

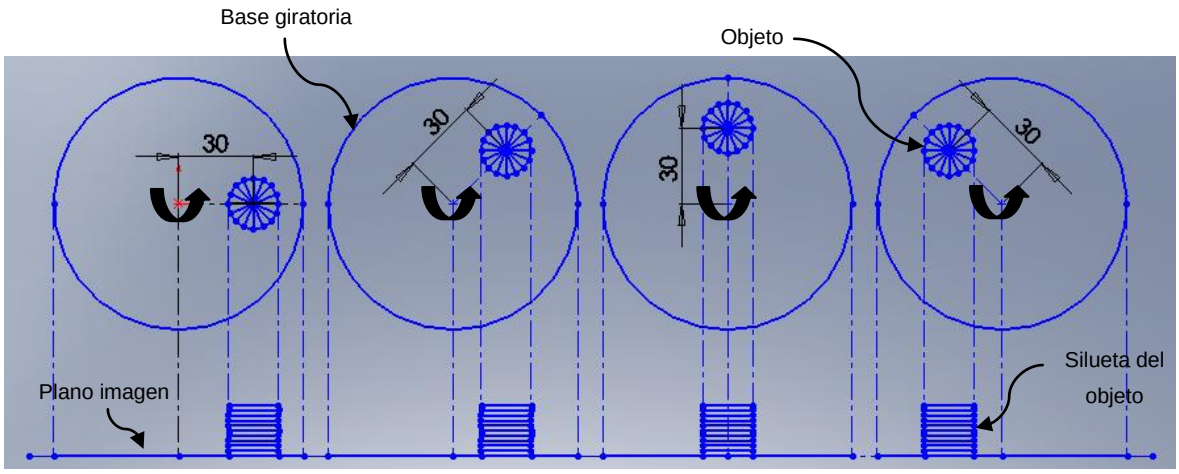


Figura 5.1. Posición del objeto sobre la base giratoria y proyección sobre el plano imagen

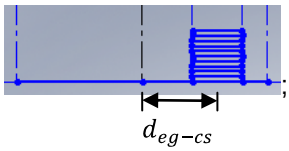
En este caso, y de acuerdo a lo mostrado en la Figura 5.1 solo se obtienen 4 siluetas del objeto repartidas en 180°, es decir a 0°, a 45°, a 90° y a 135°. Para determinar a qué distancia se encuentran el eje de giro y el centro de la silueta del objeto proyectada en el plano imagen, utilizamos la siguiente fórmula:

$$d_{eg-cs} = d_{eg-co} * \cos(\theta) \qquad Ec.(5.1)$$

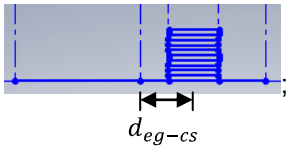
Donde: d_{eg-cs}: distancia entre el eje de giro de la base y el centro de la silueta

d_{eg-co}: distancia entre el eje de giro de la base y el centro del objeto

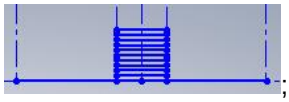
Por lo tanto, en este ejemplo, $d_{eg-co} = 30\ unidades$, y $\theta = 0°, 45°, 90°, 145°$.

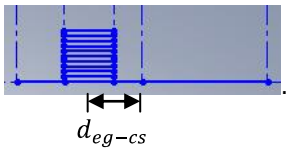
Para $\theta = 0°$, $d_{eg-cs} = 30\ unidades$, 

;

Para $\theta = 45°$, $d_{eg-cs} = 21.21\ unidades$, 

;

Para $\theta = 90°$, $d_{eg-cs} = 0\ unidades$, 
;

Para $\theta = 135°$, $d_{eg-cs} = -21.21\ unidades$, 

.

Conociendo la posición de las siluetas sobre cada uno de los planos de la imagen, y de acuerdo al algoritmo de reconstrucción 3D propuesto en el Capitulo 4, el paso a seguir consiste en girar cada uno de los planos imagen (el ángulo de giro de los planos corresponde al dado por la base), y proyectar la silueta, como se muestra en la Figura 5.2.

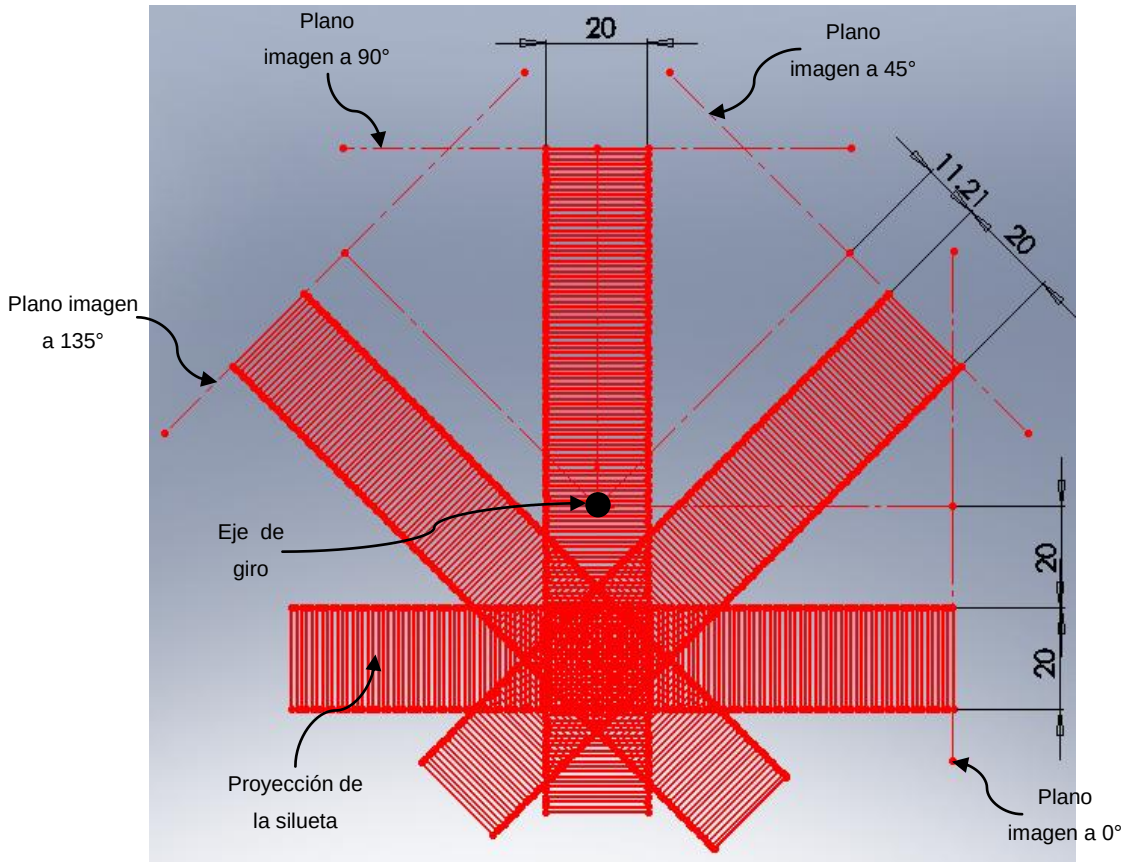


Figura 5.2. Vista superior de las proyecciones de las siluetas a 0°, 45°, 90° y 135°

De esta manera, la intersección de las 4 siluetas (ver Figura 5.3), da como resultado la reconstrucción 3D del objeto, quedando con esto comprobado que no importa el lugar donde se coloca el objeto sobre la base giratoria.

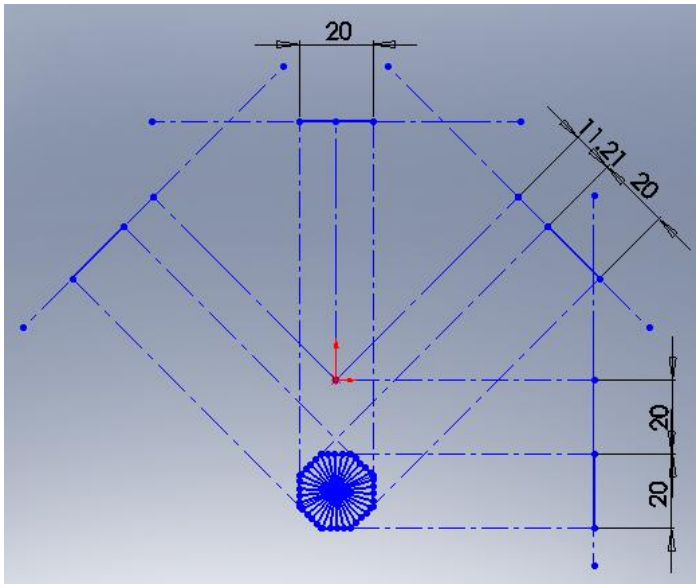


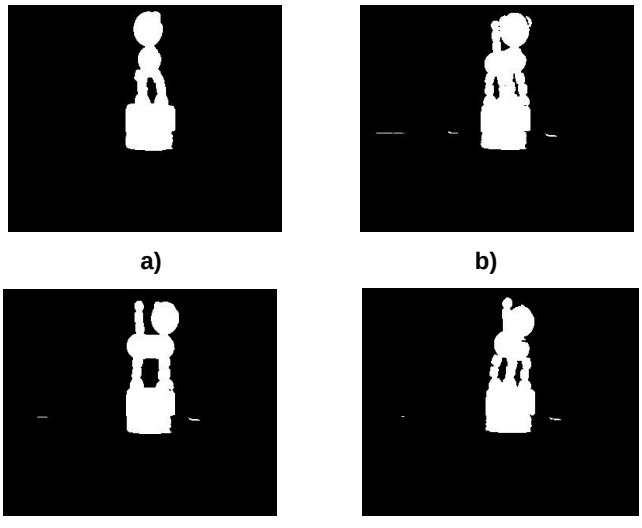
Figura 5.3. Vista superior de la intersección de las proyecciones

5.2 Reconstrucción tridimensional de un objeto

Para validar el sistema se propone reconstruir el objeto de la imagen que se muestra en la Figura 5.4.

Figura 5.4. Juguete en forma de trigre

Con la interfaz de usuario desarrollada, se realiza de manera sincronizada la obtención de imágenes y el movimiento de giro de la base, con lo cual se obtienen un total de 18 imágenes, con resolución de 352x288 pixeles. El ángulo de giro entre las imágenes es de 10° y es constante para todos los incrementos de giro de la base. En la Figura 5.5, se muestran algunas de las siluetas del objeto.

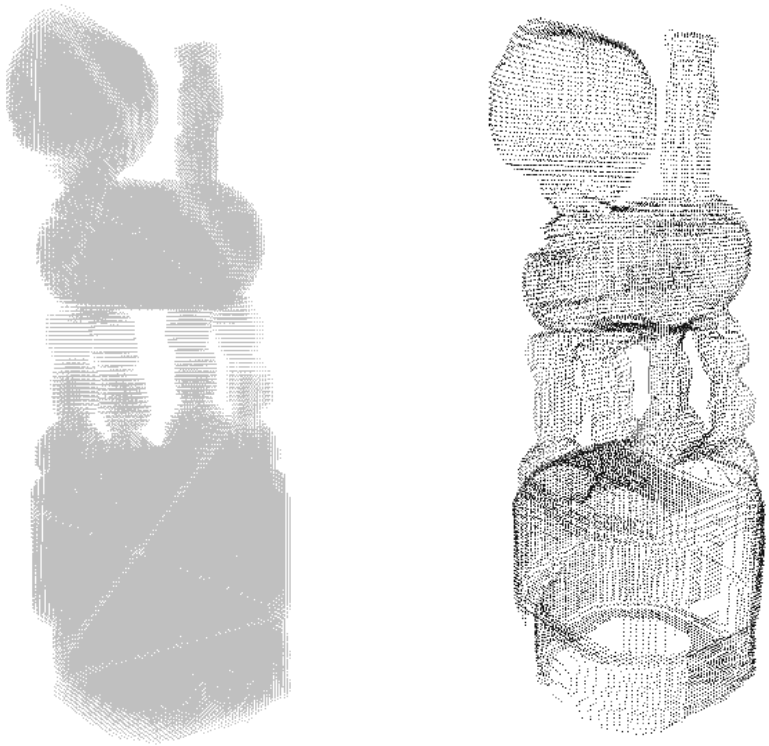


a) **b)**

c) **d)**

Figura 5.5. Silueta del objeto a a) 0°, b) 50°, c) 100°, d) 150°

Al aplicar el algoritmo desarrollado en este trabajo, se obtiene la nube de puntos del objeto reconstruido como se muestra en la Figura 5.6. La nube de puntos del inciso a) corresponde a todos los puntos encontrados en la reconstrucción 3D, mientras que la nube de puntos del inciso b) corresponde únicamente a los puntos de la superficie del juguete.



a) **b)**

Figura 5.6. Nube de puntos del juguete reconstruido, superficie del objeto

Por último, para la visualización final de la nube de puntos del objeto se utilizó Geomagic Studio 12, el cual nos ayuda a generar la triangulación de la nube de puntos de la reconstrucción 3D (ver Figura 5.7).

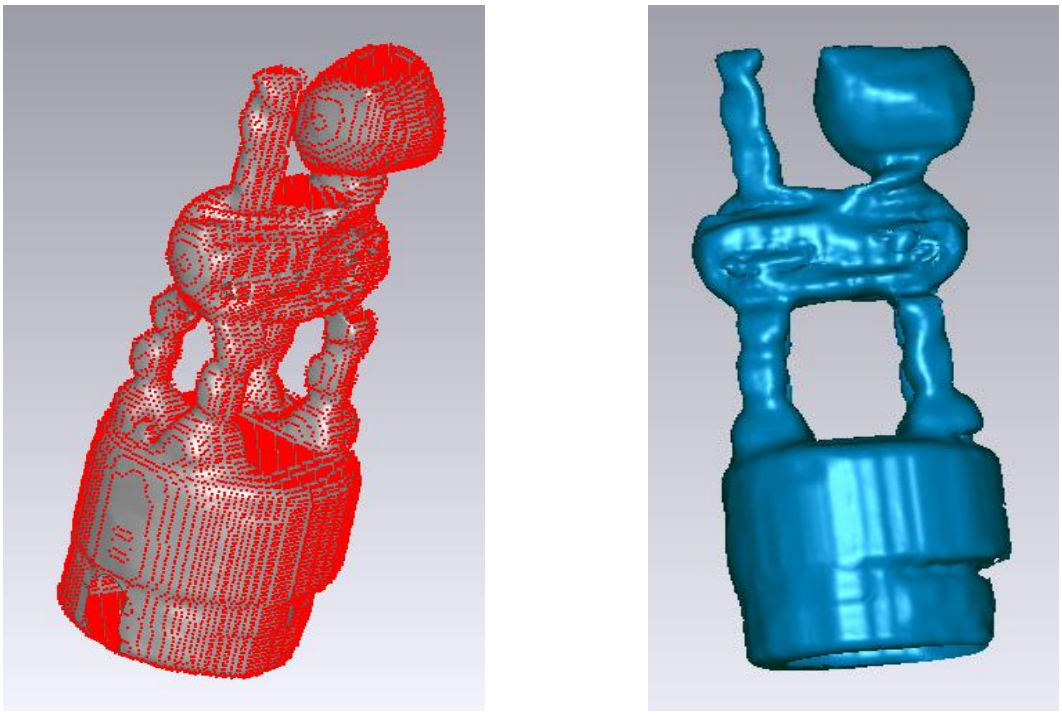
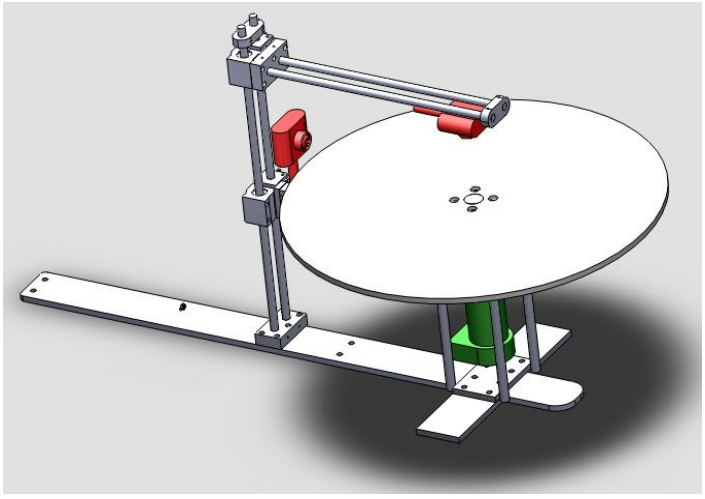


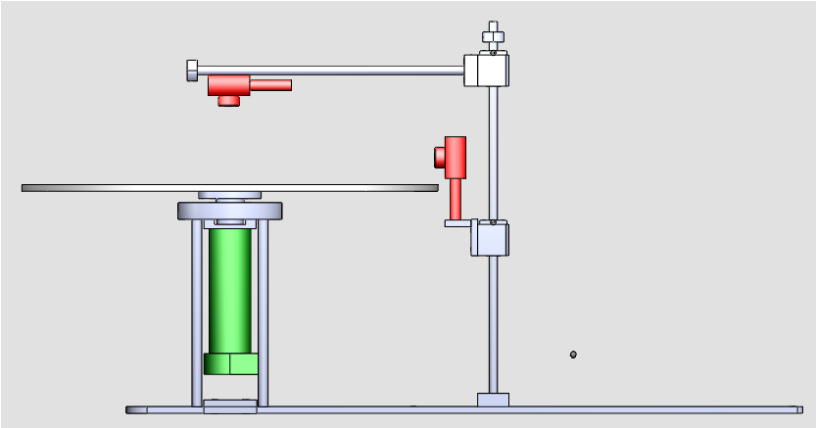
Figura 5.7. Visualización del objeto reconstruido

5.3 Reconstrucción tridimensional (una cámara VS dos cámaras)

Hasta esta sección se ha descrito el sistema de adquisición de imágenes utilizando una sola webcam para adquirir las siluetas del objeto, dicha webcam es colocada a un costado de la base giratoria, como se mostró en la Figura 4.1. En este apartado, se propone agregar una segunda webcam colocándola en la parte superior del objeto (los ejes de referencia de ambas cámaras son ortogonales) como se muestra el arreglo mecánico en la Figura 5.8, con la finalidad de reducir el error de aproximación de la reconstrucción 3D.



a)



b)

Figura 5.8. Ensamble mecánico para el montaje de dos cámaras, a) vista isométrica y, b) vista lateral del sistema

Con el objetivo de determinar la posición y orientación relativa entre ambas webcam's colocadas en el sistema mecánico, es necesario realizar la calibración de dichas mediante un Toolbox de Matlab desarrollado por Zhang descrito en la sección 3.4.

En dicho Toolbox, primero se realiza la calibración de ambas webcam's de manera independiente, para conocer sus parámetros extrínsecos e intrínsecos de forma individual. Por lo tanto, se cuenta con un patrón de calibración con forma de tablero de ajedrez de 7x7 cuadros, como se muestra en la Figura 5.9, donde sus dimensiones de cada cuadro del tablero son de 4mm por lado.

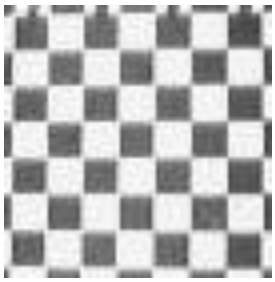


Figura 5.9. Patrón de calibración de las webcam's

Posteriormente, se adquieren imágenes con ambas webcam's en diferentes poses del tablero y de manera simultánea, teniendo como resultado una serie de imágenes como se muestra en la Figura 5.10, y de esta manera, con las diferentes poses del tablero, se lleva a cabo la calibración de las webcam's de manera independiente, obteniendo sus parámetros intrínsecos y extrínsecos (ver Tabla 5.1).

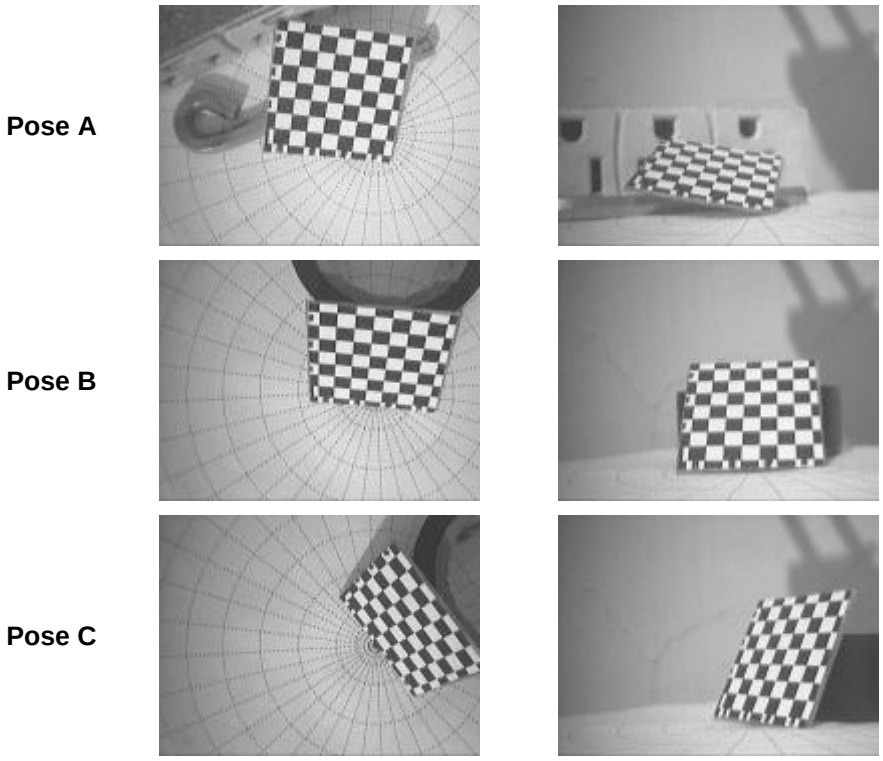


Figura 5.10. Diferentes poses del patrón de calibración visto desde ambas webcam's

Parámetros de la Webcam	Lateral	Superior
Distancia focal	[235.8378 253.7699] pixeles	[269.5716 272.7317] pixeles
Centro de eje óptico	[89.8633 92.8980] pixeles	[110.8849 81.1522] pixeles
Coeficiente de asimetría	0.0000	0.0000
Coeficientes de distorsión radial y tangencial	[0.4347 -5.5727 -0.0172 0.0164 0.0000]	[0.1282 2.5148 0.0265 0.0046 0.0000]
Tamaño de imagen	nx = 160 pixeles ny = 120 pixeles	nx = 160 pixeles ny = 120 pixeles

Tabla 5.1. Parámetros intrínsecos y extrínsecos de la webcam lateral y superior

Por lo tanto, teniendo los parámetros de las webcam's, nuevamente se recurre al uso del Toolbox de Zhang, para realizar la calibración Stereo de ambas, que consiste en obtener una orientación y una posición relativa de una webcam con respecto de la otra. En este caso, se obtiene la matriz de rotación y el vector de traslación de la webcam superior con respecto a la webcam lateral.

$$R_L^S = \begin{bmatrix} 0.9985 & -0.0460 & 0.0264 \\ 0.0286 & 0.0471 & -0.9984 \\ 0.0447 & 0.9978 & 0.0483 \end{bmatrix} \cong \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$T_L^S = \begin{bmatrix} -2.8132 \\ 118.4033 \\ 122.6731 \end{bmatrix}$$

Donde R_L^S es la matriz de rotación de la webcam superior con respecto a la webcam lateral

T_L^S es el vector de traslación de la webcam superior con respecto a la webcam lateral

De acuerdo con la aproximación de la matriz R_L^S, nos indica que los marcos de referencia de ambas webcam's se encuentran colocados ortogonalmente, y desplazados sus orígenes por un vector de traslación T_L^S, como se muestra en la Figura 5.11; por otro lado, en la misma figura se muestran cada una de las poses del patrón de calibración, que en este caso fueron 7.

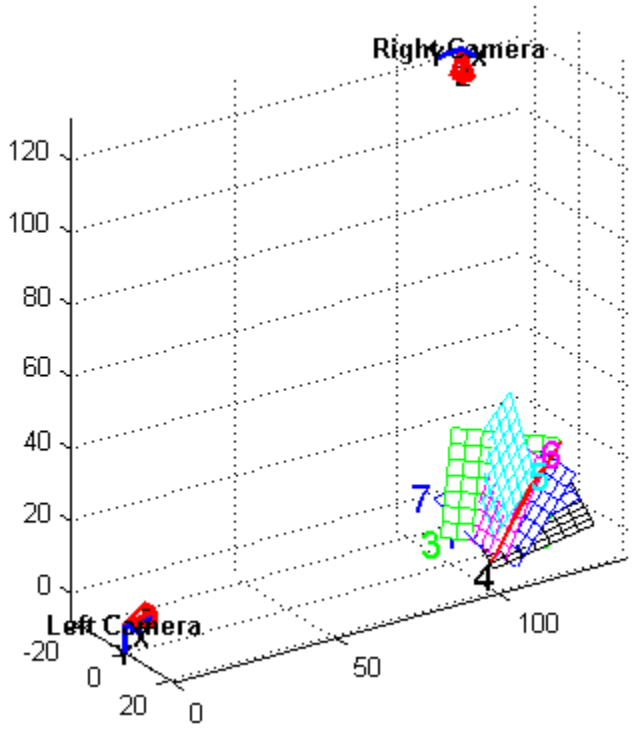


Figura 5.11. Ubicación de las poses del patrón de calibración y de las webcam's superior y lateral

Donde: Left Camera: corresponde a la posición de la webcam lateral

Right Camera: corresponde a la posición de la webcam superior

Las unidades de los tres ejes de coordenadas esta dado en milímetros [mm]

La relación matemática entre ambas webcam's se encuentra descrita mediante el siguiente modelo matemático:

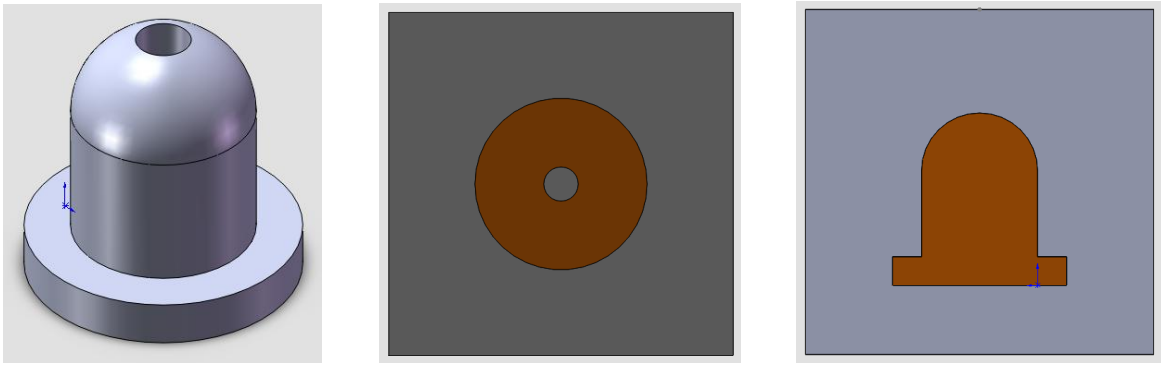
$$X_S = R_L^S X_L + T_L^S \qquad Ec.(5.2)$$

Donde, X_S: Vector de coordenadas [X , Y , Z]^t del mundo real con respecto al plano de la imagen de la webcam superior.

X_T: Vector de coordenadas [X , Y , Z]^t del mundo real con respecto al plano de la imagen de la webcam lateral.

Con dicha relación matemática, se puede transformar el marco de referencia de la webcam superior al marco de referencia de la webcam lateral.

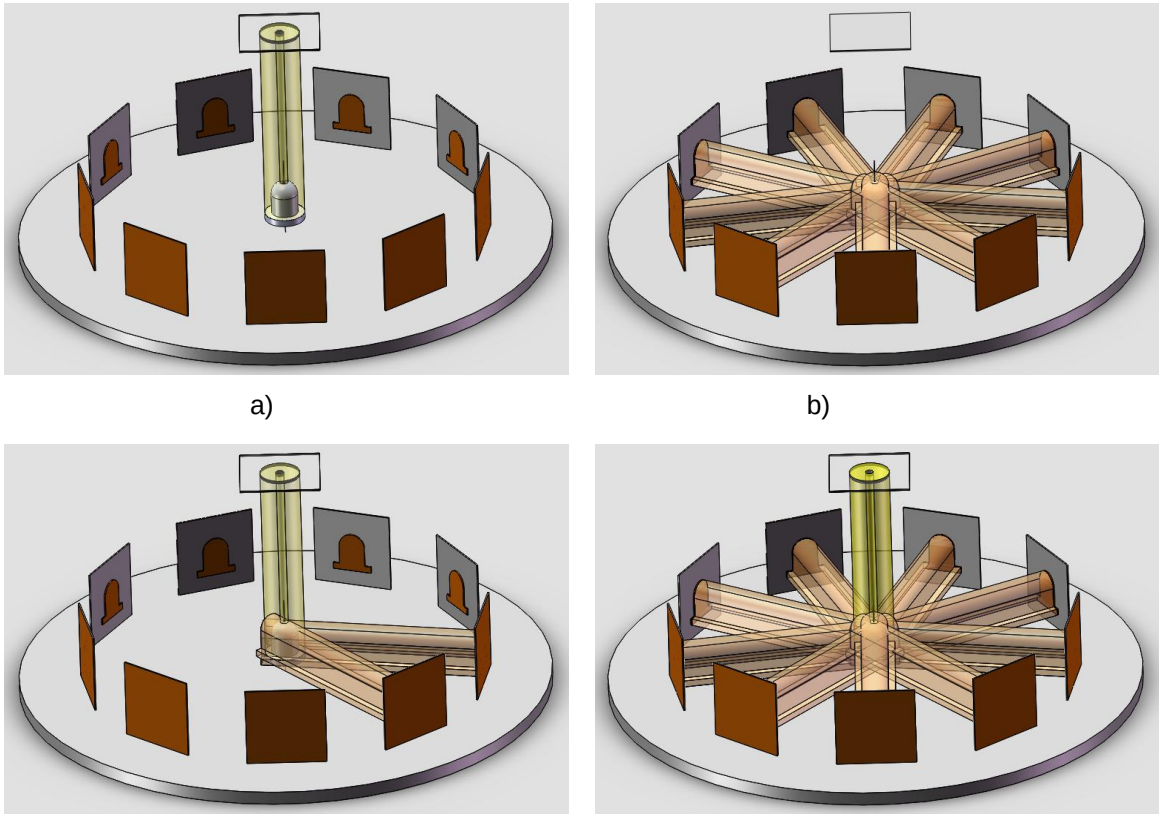
Por otro lado, el algoritmo de reconstrucción 3D usando dos webcam's está basado en el algoritmo descrito en la sección 4.3, pero ahora agregando una silueta más al procesamiento de imágenes (silueta de la vista superior). En la Figura 5.12 se muestra en a) el objeto deseado a reconstruir, en b) la silueta de la vista superior, en c) la silueta de la vista lateral (como el objeto es simétrico, se mantiene la misma silueta en cualquier posición).



a) b) c)

Figura 5.12. Vistas del objeto a reconstruir tridimensionalmente, a) Vista isométrica, b) Vista superior y c) Vista lateral, del objeto

Posteriormente, se crean nubes de puntos del sub-volumen formado por la proyección de cada una de las siluetas. De este modo, para obtener la reconstrucción 3D del objeto basta con realizar la intersección de todos los sub-volúmenes de las siluetas proyectadas, tanto laterales como la superior. En la se muestra en el inciso a) la proyección del sub-volumen de la silueta superior, en b) las proyecciones de los sub-volúmenes de todas las siluetas laterales, en c) la proyección del sub-volumen de la silueta superior más dos sub-volúmenes de siluetas laterales, y en d) la proyección de todos los sub-volúmenes involucrados en la reconstrucción tridimensional.



a) b)

c) d)

Figura 5.13. Visualización de la intersección de las proyecciones de los sub-volúmenes de a) la silueta superior, b) las siluetas laterales, c) la silueta superior más dos de siluetas laterales, y c) todas las siluetas involucradas

Una prueba muy sencilla para realizar la comparación entre la reconstrucción 3D con una webcam y con dos webcam's, es reconstruyendo una esfera, ya que tanto las siluetas adquiridas por la webcam lateral, son iguales a la silueta adquirida por la webcam superior, ya que desde cualquier perspectiva que se visualice a una esfera, siempre se conserva su silueta en forma de circunferencia.

Por lo tanto para este experimento se reconstruyen esferas de tres diámetros diferentes, la primera con un diámetro de 41 pixeles, la segunda con un diámetro de 61 pixeles y por último, la tercera con un diámetro de 81 pixeles.

Para calcular el volumen de una esfera, se tiene la siguiente ecuación:

$$Vol_{esf} = \frac{4}{3}\pi r^3 = \frac{\pi d^3}{6} \qquad Ec.(5.3)$$

Donde: r : radio de la esfera en pixeles

d : diámetro de la esfera en pixeles

vol_{esf} : volumen de la esfera en vóxeles

Para la esfera de 41 pixeles de diámetro tiene un volumen de 36086.951 vóxeles, para la esfera de 61 pixeles de diámetro el volumen es de 118846.974 vóxeles, y para la esfera de 81 pixeles de diámetro su volumen es de 278261.857 vóxeles.

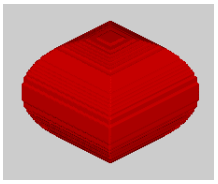
En cada una de las tres esferas a reconstruir, se aplicó el algoritmo de reconstrucción 3D con diferente número de siluetas a procesar (ver Tabla 5.2).

Num. de imágenes	Diametro = 41 pixeles		Diametro = 61 pixeles		Diametro = 81 pixeles	
	VSVS (vóxeles)	VCVS (vóxeles)	VSVS (vóxeles)	VCVS (vóxeles)	VSVS (vóxeles)	VCVS (vóxeles)
2	47070	41239	153161	134363	357354	313517
3	41301	39326	133811	128103	311530	298705
6	38355	37588	124269	122562	289097	286079
9	37888	37233	122635	121353	285474	283321
12	37701	37095	122237	120987	284401	282412
15	37740	37109	122111	120908	284030	282123
18	37698	37070	121759	120668	283367	281607
20	37749	37108	121747	120656	283345	281617
30	37602	37010	121763	120628	283350	281570
36	37616	37024	121687	120596	283139	281431

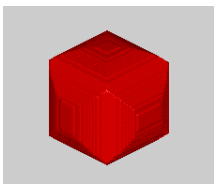
Tabla 5.2. Estimación del volumen de una esfera mediante la reconstrucción 3D

VSVS: Volumen de la esfera sin aplicar la vista superior,
VCVS: volumen de la esfera aplicando la vista superior

En la Figura 5.14 se muestra la nube de puntos con textura de la reconstrucción de la esfera de 81 pixeles de diámetro, en el inciso a) se encuentra la reconstrucción con sólo dos siluetas y sin aplicar la vista superior, en el inciso b) se encuentra igual la reconstrucción con sólo dos siluetas, pero ahora sí, considerando la vista superior. El inciso c) y d) se encuentra la reconstrucción aplicando solo 3 siluetas, para el primero sin aplicar la vista superior, mientas que para el segundo se le aplica la vista superior. Y por último, para el inciso e) se encuentra la reconstrucción de la esfera aplicando 18 imágenes considerando la visa superior.



a)



b)

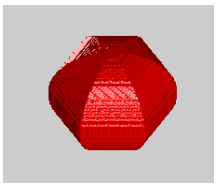
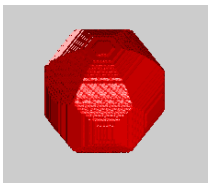
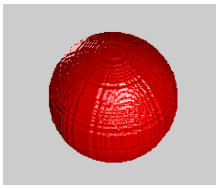

c)

d)

e)

Figura 5.14. Reconstrucción 3D de la esfera de 81 pixeles de diámetro, con diferente numero de imágenes a procesar

Con los datos de la Tabla 5.2 y el valor real del volumen de las esferas se puede calcular el error de la aproximación que se tiene en la reconstrucción 3D, como se muestra en la Tabla 5.3.

Núm. de imágenes	Diámetro = 41 pixeles		Diámetro = 61 pixeles		Diámetro = 81 pixeles	
	EVSVS (%)	EVCVS (%)	EVSVS (%)	EVCVS (%)	EVSVS (%)	EVCVS (%)
2	30.435	14.282	28.872	13.055	28.423	12.670
3	14.449	8.981	12.591	7.788	11.956	7.347
6	6.285	4.165	4.562	3.126	3.894	2.810
9	4.991	3.181	3.187	2.109	2.592	1.818
12	4.472	2.799	2.852	1.800	2.206	1.492
15	4.580	2.838	2.746	1.734	2.073	1.388
18	**4.464**	**2.729**	**2.450**	**1.532**	**1.835**	**1.203**
20	4.606	2.835	2.440	1.522	1.827	1.206
30	4.198	2.563	2.454	1.498	1.828	1.189
36	4.237	2.602	2.389	1.472	1.753	1.139

Tabla 5.3. Porcentaje de error calculado en la reconstrucción 3D

EVSVS: Error del volumen estimado de la esfera sin aplicar la vista superior,
EVCVS: Error del volumen estimado de la esfera aplicando la vista superior

Por lo tanto de acuerdo a lo valores obtenidos en la Tabla 5.3 puede observar que mientras más pixeles tenga el diámetro de la esfera, la estimación del volumen se aproxima al valor real. Por otro lado, exactitud de la estimación del volumen está en función del número de siluetas a procesar. Por último, la reconstrucción 3D es más exacta cuando se tiene una segunda webcam, ya que se tienen más información. Este análisis de los resultados obtenidos se puede observar en la grafica de la Figura 5.15.

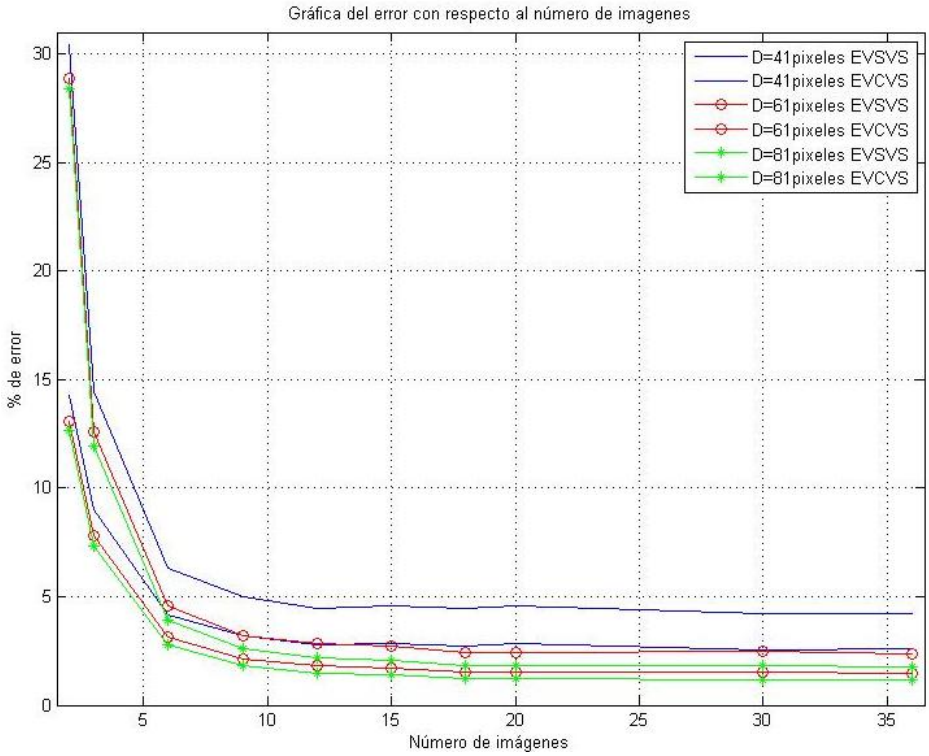
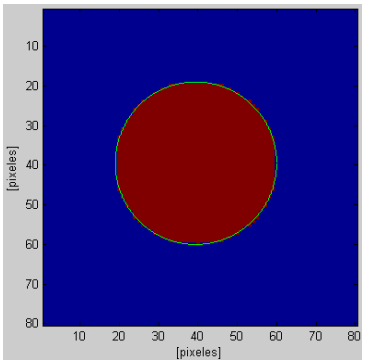
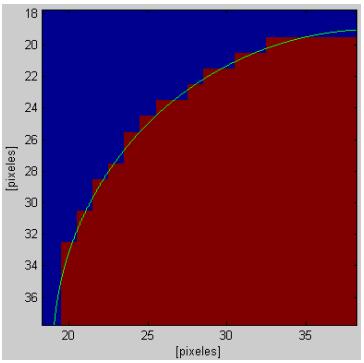


Figura 5.15. Curvas que representan el porcentaje de error en la reconstrucción 3D de acuerdo al número de imágenes y el diámetro de la esfera

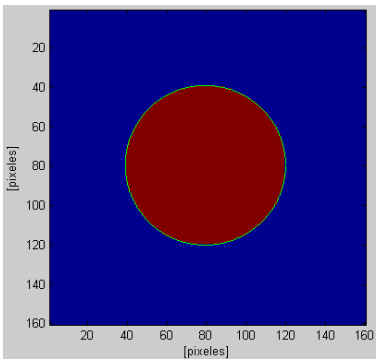
Dicho error aumenta o disminuye en función de cómo se digitalice la silueta del círculo. En la Figura 5.16 se muestran dos círculos, el circulo del inciso a) tiene un diámetro de 41 pixeles, mientras que el círculo del inciso c) tiene un diámetro de 81 pixeles. Por otro lado, en los incisos b) y d) se muestra un zoom de ambos círculos, notándose cómo es que al digitalizar la silueta, se pierde o se gana información. Por tal motivo, desde que obtenemos la silueta estamos agregando un error de digitalización a la reconstrucción 3D de la esfera.

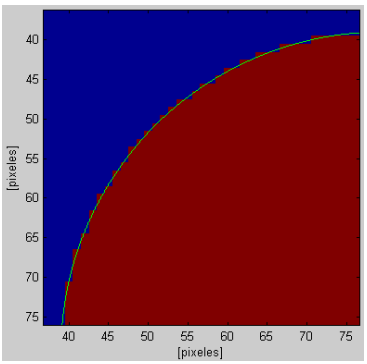


a)



b)



c)


d)

Figura 5.16. Digitalización de un círculo a) diámetro de 41 pixeles, c) diámetro de 81 pixeles

Dicho error de digitalización es contable, ya que de manera ideal, el área del círculo debe ser igual al número de pixeles contados en la silueta. Por lo tanto,

$$Ac = \frac{\pi(diametro\ círculo)^2}{4} \qquad Ec.(5.4)$$

Para el círculo de diámetro de 41 pixeles, $Ac = 1320.254\ pixeles^2$
Para el círculo de diámetro de 81 pixeles, $Ac = 5152.997\ pixeles^2$

Mientras que:

- el número de pixeles contados por la silueta del círculo del círculo de 41 pixeles es de 1325
- el número de pixeles contados por la silueta del círculo del círculo de 81 pixeles es de 5161

Por lo tanto, el error de digitalización para ambos círculos es:

Circulo de 41 pixeles de diámetro $err_{dig} = 0.358\%$

Circulo de 81 pixeles de diámetro $err_{dig} = 0.155\%$

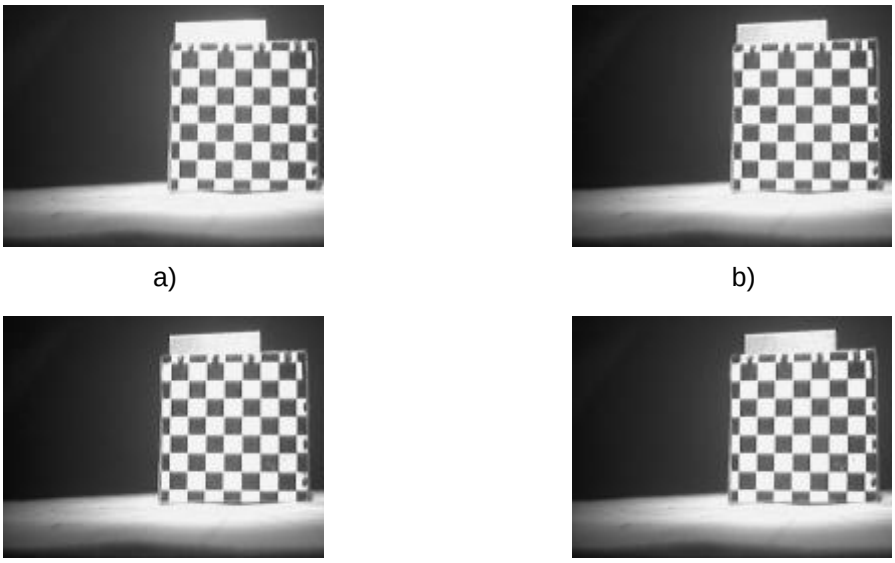
Por lo tanto, mientras más pixeles se tengan disminuye el error de digitalización, y por consecuencia disminuye el error en la reconstrucción 3D.

5.4 Estimación del volumen de objetos sólidos

Después de haber evaluado el algoritmo de reconstrucción 3D con diferentes pruebas, continuamos con la estimación del volumen de objetos sólidos. Esto se logra mediante un patrón de longitud (que nos permita definir la relación entre pixel-milímetro) y la nube de puntos de la reconstrucción 3D del objeto. El patrón de longitud a utilizar tiene la forma de tablero de ajedrez de

7x7 cuadros (ver Figura 5.9) donde las dimensiones de cada cuadro del tablero es de 4mm por lado.

Por lo tanto para obtener la relación pixel-milímetro, se tomaron 4 imágenes del tablero de calibración, dentro de cada imagen se obtienen 20 mediciones de la distancia que ocupan 4 cuadros del tablero (de manera horizontal o vertical), posteriormente, se realizan 10 mediciones de la distancia en diagonal a 45° de tres cuadros.



a) b)

Figura 5.17. Cuatro poses diferentes del tablero de calibración

En la Tabla 5.4 y la Tabla 5.5 se muestran las diferentes mediciones realizadas para determinar la cantidad de pixeles que se tienen en una distancia equivalente a 3 cuadros del tablero (vertical, horizontal y en diagonal a 45°), es decir, en 12mm de manera horizontal ó vertical, ó en 16.97mm en diagonal a 45°.

Distancia entre 3 cuadros (Vertical / Horizontal) [pixeles]				
Muestra	**Pose 1**	**Pose 2**	**Pose 3**	**Pose 4**
1	26.036	26.115	27.231	25.855
2	26.986	27.031	26.796	25.911
3	26.142	25.707	26.810	25.983
4	26.949	27.085	26.448	27.419
5	27.756	27.807	27.722	27.450
6	27.644	26.390	27.752	28.311
7	26.734	27.142	26.576	27.482
8	27.055	26.213	27.845	26.183
9	26.819	27.339	25.607	26.082
10	25.525	26.648	27.092	26.494

11	28.399	27.850	27.249	28.176
12	28.518	27.685	27.341	26.502
13	27.598	26.409	27.485	28.332
14	28.307	27.394	26.866	26.632
15	25.685	28.511	27.661	26.495
16	25.795	27.874	26.340	27.502
17	26.279	26.443	27.121	26.616
18	27.263	26.879	25.627	28.443
19	26.540	25.738	28.302	27.971
20	25.647	26.481	25.988	26.969

Tabla 5.4. Distancia entre tres cuadros (Vertical / Horizontal)

Distancia entre 3 cuadros en diagonal a 45° [pixeles]				
Muestra	**Pose 1**	**Pose 2**	**Pose 3**	**Pose 4**
1	36.820	40.162	39.761	39.386
2	38.163	40.330	38.227	36.324
3	36.971	39.030	36.355	37.348
4	38.112	39.432	38.305	38.741
5	39.960	36.324	39.325	40.321
6	39.095	36.480	37.322	39.419
7	37.807	37.164	38.384	37.396
8	38.261	39.970	37.070	38.013
9	37.928	37.534	38.663	36.400
10	36.098	36.271	37.686	37.450

Tabla 5.5. Distancia entre tres cuadros en diagonal a 45°

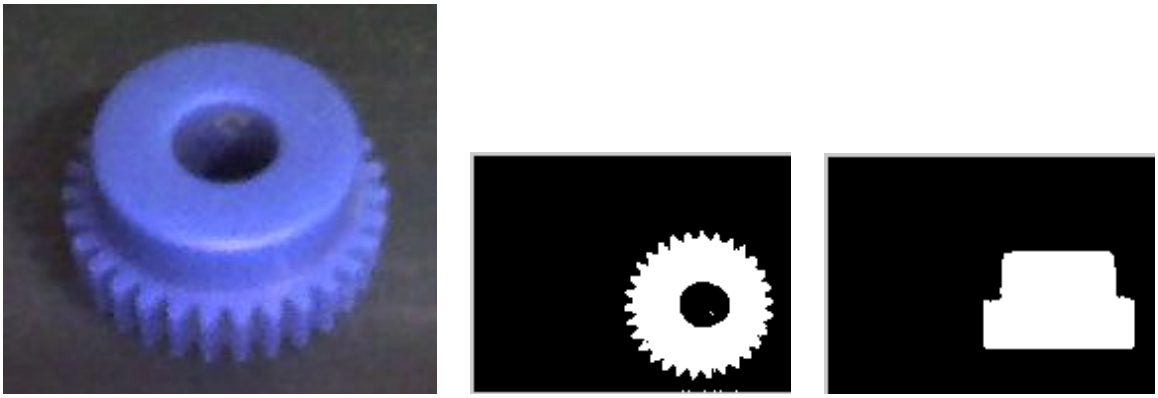
Por lo tanto, de acuerdo a las mediciones realizadas en las dos tablas anteriores, podemos obtener una aproximación a la relación pixel-milímetro, y por consecuencia, a la relación vóxel-milímetro cúbico. Dando como resultado:

1 pixel = 0.445mm

1 vóxel = 0.088mm^3

Por lo tanto para determinar el volumen de un objeto, es suficiente con realizar el conteo de los puntos de la nube de puntos (vóxeles) y multiplicarlos por la relación vóxel-mm^3.

Tal es el caso para el objeto mostrado en la Figura 5.18. a) Engrane de Nylamid. El objeto es un engrane de nylamid. El volumen aproximado con instrumentos de medición manuales (Vernier) y ecuaciones matemáticas que determinan el volumen de prismas o cilindros, nos indica que es igual a 11380.419mm^3.



a) b) c)

Figura 5.18. a) Engrane de Nylamid, b) silueta de la vista superior del engrane, c) silueta de la vista lateral del engrane

Mientras que el volumen obtenido con la reconstrucción 3D sin la vista superior (ver Figura 5.19) es de 14285.505mm^3, ya que se obtuvieron 162566 voxeles.

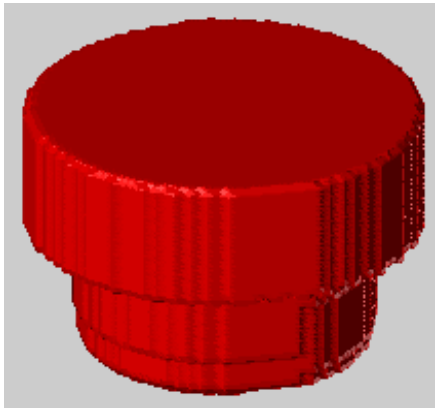


Figura 5.19. Reconstrucción 3D sin vista superior

Por otro lado, el volumen obtenido con la reconstrucción 3D con la vista superior (ver Figura 5.20) es de 11495.470mm3, ya que se obtuvieron 130816 voxeles.

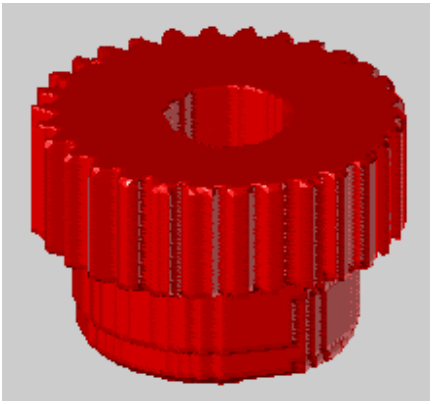


Figura 5.20. Reconstrucción 3D con vista superior

Por lo tanto, el error de aproximación del volumen de dicho objeto es de:

Reconstrucción 3D con vista superior:	1.011%
Reconstrucción 3D sin vista superior:	25.527%

Por lo tanto se puede observar que el volumen del objeto se estima de mejor manera aumentando el número de visualizaciones del objeto, es decir en este caso, se mejora en gran medida el error al aumentarle una vista más al algoritmo de reconstrucción 3D.

Capítulo 6. Conclusiones

6.1 Conclusiones

Mientras mayor sea el número de imágenes tomadas al objeto, mayor información se obtiene sobre el objeto y mejora la reconstrucción 3D. La reconstrucción 3D depende en gran medida de este factor.

Otro parámetro a considerar es la resolución de la cámara, ya que a mayor resolución, se tiene mayor información de la imagen del objeto, por lo tanto se tiene mayor precisión en la reconstrucción 3D.

Una desventaja de este método es que no puede reconstruir objetos que tengan concavidades. Una alternativa para solucionar este problema es agregarle a este sistema un láser para determinar profundidades en el objeto.

Al agregar una vista a la reconstrucción 3D por la parte superior, nos ayuda a reducir el error de aproximación del volumen del objeto cuando se tienen orificios pasados sobre la pieza.

No importa si colocamos el objeto sobre el eje de la base de giro, el algoritmo es capaz de reconstruirlo, aunque el problema que se presenta es sobre la óptica de la lente.

Se ha presentado una metodología para la reconstrucción 3D de objetos bastante precisa y de muy bajo costo. Se ha demostrado que mientras mayor sea el número de imágenes adquiridas la reconstrucción es más exacta, sin embargo con más allá de 20 imágenes no se obtiene una mejora apreciable. La integración de una cámara ortogonal mejora considerablemente la reconstrucción ya que permite la reconstrucción de cavidades ocultas con una sola cámara.

Otra ventaja de esta técnica, es que el objeto a reconstruir puede ser colocado en cualquier parte de la mesa giratoria. Este punto es muy importante, ya que cuando se coloca el objeto sobre la base giratoria no garantizamos alinear el centro del objeto con el centro del eje de giro. Por tal motivo, con esta técnica eliminamos dicho problema.

La técnica presentada permite también estimar el volumen de objetos. Esto se logra mediante el conteo de los pixeles (vóxeles) de la nube de puntos total y multiplicarlos por la relación vóxel/mm^3.

Bibliografía

[Bachs y Cuesta, 1988], L. Bachs y J. Cuesta. *"Aplicaciones industriales del láser", 1988.*

[Baker,1997], H. Baker, "Three-dimensional modeling". In Proc. Of 5th International Joint Conference on Artificial Intelligence, 1997.

[Becerra y Centeno, 2003], Luis O. Becerra y Luz Maria Centeno, "Mexican Density Standard: data treatment in Hydrostatic weighing". y Dubrovnik, Croatia : XVII IMEKO World Congress Metrology in the 3rd Millennium, 2003.

[Besl, 1988], P. J. Besl, " Optical range imaging sensors". Machine vision and applications, 1988 – Springer.

[BIPM, 2006], BIPM – Bureau International des Poids et Mesures. "The International System of Units (SI)". Organisation Intergouvernementale de la Convention du Mètre, 2006.

[Bueno, 2010], Rafael Muñoz Bueno, Curso académico: "*Introducción a la metrología".* Laboratorio de Metrología y Metrotecnia, 2010.

[Camera Calibration Toolbox for Matlab, 2010], *Camera Calibration Toolbox for Matlab*. Sitio Web: http://www.vision.caltech.edu/bouguetj/calib_doc/

[Caroll, 2010], Bradley W. Carroll, "Archimedes' Principle". Weber State University, 2010. Sitio Web: http://www.physics.weber.edu/carroll/Archimedes/principle.htm.

[CENAM, 2011], CENAM.*"Centro Nacional de Metroogía", 2011*. Sitio Web: http://www.cenam.mx/

[CENAM - Metrología Dimensional, 2011], CENAM. *"Metrología Dimensional"*, Centro Nacional de Metrologìa, Querétaro, México, 2011, Sitio Web: http://www.cenam.mx/Dimensional/

[CENAM - Metrología de Masa y Densidad, 2011], CENAM. *"Metrología de Masa y Densidad"*, Centro Nacional de Metrologìa, Querétaro, México, 2011, Sitio Web: http://www.cenam.mx/myd/

[CENAM – Patrón Nacional de Densidad, 2011], "Patrón Nacional de Densidad CNM-PNM-26", Centro Nacional de Metrologìa, Querétaro, México, 2011.

[Delaunay, 1934], B. Delaunay, “Sur la sphere vide”. Bulletin de L´Académie des Sciencs de L´URSS, 1934.

[Eisert, 2002], Peter Eisert, "Model-based camera calibration using analysis by synthesis techniques", Vision Modeling and Visualization, 2002.

[FARO, 2010], FARO, *"Brazos Articulados de Medición",* 2010, Sitio Web http://www.cam2.it/default_es.aspx?ct=sp

[Faugeras, 1992], O. D. Faugeras, Maybank, QT Loung, . (1992). "Camera self-calibration: Theory and experiments", Computer Vision - ECCV'92, Springer, 1992.

[Giraldo, Arroyave y Montilla, 2008], C. E. Giraldo, L. Arroyave y M. Montilla. “Dimensionamiento de piezas en un sistema de visión aplicado a una celda de manufactura”. Pereira, Colombia: Scientia Et Technica, 2008, Vol. XIV.

[González y Castillo, 2010], G. Gonzalez y E. Castillo, "Inspección de las dimensiones de objetos mediante procesamiento de imágenes", Encuentro Nacional de Investigadores en Ingeniería Eléctrica, ENINVIE 2010, Zacatecas, 2010.

[Harris y Söcker, 1998], John W. Harris y Horst Stöcker, "Handbook of mathematics and computational science". Birkhäuser: Springer, 1998. ISBN: 0-387-94746_9.

[Heikkila y Silven, 1997], Heikkila, J., y Silven, O., "A four-step camera calibration procedure with implicit image correction", Computer Vision and Pattern Recognition, 1997., IEEE Computer Society Conference on Proceedings.

[HyperPhysics, 2010], HyperPhysics, *"Buoyanc"*, Georgia State University, 2010. Sitio Web: http://hyperphysics.phy-astr.gsu.edu/Hbase/pbuoy.html

[JasVisio, 2010], JasVisio, 2010, Sitio Web: http://www.jasvisio.com/index-es.html

[Kuzu y Sinram, 2003], Y. Kuzu y O. Sinram. "Volumetric reconstruction of cultural heritage artifacts". "volumetric reconstruction of cultural heritage artifacts". Photogrammetry and Cartography Technical University of Berlin, CIPA International Symposium, 2003.

[NAVITAR MACHINE, 2010], NAVITAR MACHINE, 2010, Sitio Web: http://www.machinevision.navitar.com/index.cfm

[Niem, 1994], W. Niem, "Robust and fast modelling of 3D natural objects from multiple views". In Image and Video Processing II, Proc. of SPIE, 1994.

[Martin, 1983], Martin, W. N., & K.Aggarwal, J., "Volumetric description of objects from multiple views". Martin, W. N. y K.Aggarwal, J. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI : s.n., 1983.

[Matlab, 2004], *"The MathWorks. Image Processing Toolbox for use with MATLAB. User´s Guide". Version 5, 2004.*

[Park y Subbarao, 2002], Park, S. Y., & M.Subbarao. "Automatic 3D model reconstruction based on novel pose estimation and integration techniques". New York : Image and vision computing, 2002.

[Pasten, 2007], N. G. Pastén, D. A. Pizarro, C. L. Tozzi, "Reconstrucción de objeto 3D a partir de imágenes calibradas". Revista chilena de ingeniería, 2007, Vol. 15.

[Rocchini, 2001], Rocchini, C., Cignoni, P., Montani, C., & Scopigno, P. P. (2001). "A low cost 3D scanner based on structured light". Eurographics: Guest Editors, 2001, Vol. 20.

[Schmitt, 2002], Esteban, C. H., & Schmitt, F. "Multi-Stereo 3D Object Reconstruction". 3D Data Processing Visualization and Transmission, 2002. First International Symposium on Proceedings.

[Tadeo y Gallegos, 2008], R. Tadeo G. y Francisco Gallegos F., *"Reconstrucción de objetos tridimensionales a partir de información bidimensional".* México, DF, 2008.

[Taubin y Savarese, 2001], Taubin, G., & Savarese, S. (2001). “Rotations”, CalTech, 2001.

[Tommaselli y Tozzi, 1999], Maria, A., Tommaselli, G., & Tozzi, C. L. (1999). “Line based camera calibration in machine vision dynamic applications”. SBA Controle & Automacao, Vol. 10, 1999.

[Trucco, 1998], Trucco, Alessandro Verri, *“Introductory Techniques for 3-D Computer Vision”,* Prentice Hall, 1998, ISBN: 0132611082

[Tuceryan, 1995], M. Tuceryan, D. Greer, R. Whitaker, D. Breen, C. Crapton and K. Ahlers. "Calibration Requirements and Procedures for a Monitor-Based Augmented Reality System". IEEE Trans. On Visualization and Computer Graphics, 1995, Vol. 1.

[VIEWTECH, 2010], VIEWTECH, 2010, Sitio Web: http://www.viewtech.it/home.swf

[Vitruvius, 2010], Vitruvius, *"De Architectura, Book IX, paragraphs 9–12"*. (University of Chicago), 2010. Sitio Web: http://penelope.uchicago.edu/Thayer/E/Roman/Texts/Vitruvius/9*.html

[Zhang, 1999], Zhengyou Zhang, “A Flexible New Technique for Camera Calibration”, Microsoft Research, One Microsoft Way, Redmond, WA 98052-6399, USA, 1999.

Apéndice A. Incertidumbre volumétrica de una MMC

La incertidumbre de medición

La incertidumbre de medición (IM) es el error máximo que puede cometer una MMC durante la medición de una longitud conocida y de la manera establecida por un estándar internacional. Los estándares ampliamente reconocidos en la actualidad para la certificación de la IM de una MMC son:

- La VDI, para Europa y sus áreas de influencia
- La B89, para los Estados Unidos y sus áreas de influencia
- La JIS, para algunas áreas de Asia

Recientemente ha sido aprobado un estándar ISO que, por el momento, no está siendo muy empleado debido principalmente al mucho tiempo que requieren los ensayos que establece.

La IM es el parámetro más significativo, pues contiene todos los posibles componentes de error (errores geométricos de la estructura mecánica, errores de los sensores, etc).

Por lo general, la IM se expresa en términos de +/- 2, obtenida según la siguiente fórmula:

a + **b** L/1.000 (en µm), donde:

a (µm) es la constante de error declarada por el fabricante para una MMC determinada

b (µm) es la variable de error en función de la longitud del bloque patrón, declarada por el fabricante para una MMC determinada

L (mm) es la longitud del bloque patrón

La fórmula indicada puede ser expresada en forma gráfica, tal como se muestra en la figura. Hay que destacar que el error no tiende (obviamente) al infinito, pero se convierte en asintótico para una longitud especificada por el fabricante.

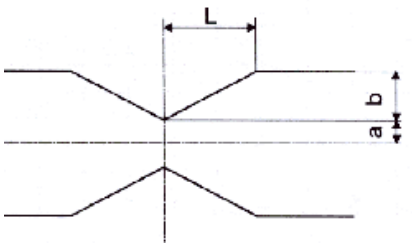


Figura A. 1. Incertidumbre de medición

Como ha sido indicado, la incertidumbre de medida está estrechamente relacionada con las condiciones térmicas del entorno. Por tanto, el fabricante está obligado a especificar bajo qué condiciones de operación ha obtenido la IM declarada, por ejemplo de la siguiente manera:

- Temperatura ambiente en el lugar de la instalación: +20º
- Gradiente térmico espacial: 1º C/m

- Tiempo máximo de gradiente térmico: 0.5 oC/h y 2oC/24h

Resumiendo: la incertidumbre volumétrica de medida (IVM), corresponde a la diferencia entre la longitud del bloque patrón, orientado en el espacio, y su correspondiente valor medido por la MMC.

Apéndice B. Funciones principales de la librería OpenCV

OpenCV (Open source Computer Vision library) es una librería abierta desarrollado por Intel. Esta librería proporciona un alto nivel funciones para el procesado de imágenes. Estas librerías permiten a los programadores para crear aplicaciones poderosas en el dominio de la visión digital. OpenCV ofrece muchos tipos de datos de alto-nivel como juegos, árboles, gráficos, matrices, etc. OpenCV es opensource para poder funcionar en muchas plataformas.

OpenCv permite:

Operaciones básicas,	Procesado de imágenes y análisis,
Análisis estructural,	Análisis de movimiento,
Reconocimiento del modelo,	Reconstrucción 3d y calibración de la cámara,
Interfaz gráfica y adquisición,	Etc.

Hay varios tipos de datos fundamentales en OpenCV, y varios tipos de datos auxiliares para hacer OpenCV más simple y uniforme.

Los tipos de los datos fundamentales incluyen tipos de vectores como: IplImage (imagen de IPL), CvMat (matriz), growable collections: CvSeq (deque), CvSet, CvGraph y tipos mixtos: CvHistogram (histograma multi-dimensional).

Los tipos de datos auxiliares incluyen: CvPoint (2d punto), CvSize (anchura y altura), CvTermCriteria (criterio de la terminación para los procesos iterativos), IplConvKernel (núcleo de la circunvolución), CvMoments (momentos espaciales), etc.

cvLoadImage: carga una imagen desde archivo

IplImage* cvLoadImage(const char* filename, int iscolor CV_DEFAULT(1));

- filename : Nombre de la imagen a cargar
- iscolor if menor que 0, la imagen cargada tendrá 3 canales ;
  if = 0, tendrá 1 canal;
  if mayor que 0, se cargará tal y como es (con el número de canales que tiene inicialmente).

La función cvLoadImage carga una imagen del archivo especificado y devuelve un puntero a la imagen. Los formatos que son soportados son los siguientes :BMP, JPEG, JPG, JPE, DIB, PNG, PBM, PPM, PGM, SR, RAS, TIFF, TIF.

Si la ruta especificada no está completa, el archivo se busca en el directorio actual y en los directorios especificados por cvAddSearhPath.

cvSaveImage: Guarda una imagen en un fichero

int cvSaveImage(const char* filename, const CvArr* image);

- filename : Nombre del archivo.
- Image: Imagen que va a ser guardada

La función cvSaveImage guarda la imagen en el archivo especificado.

cvShowImage: Muestra la imagen en la ventana especificada

void cvShowImage(const char* name, const CvArr* image);

- name : Nombre de la ventana
- image : Imagen a ser mostrada

La función cvShowImage muestra la imagen en la ventana especificada. Si la ventana ha sido creada con el flag CV_WINDOW_AUTOSIZE, entonces la imagen se mostrará con su tamaño original, en caso contrario la imagen se escalará con la ventana.

CloneImage: Crea una copia de la imagen

IplImage* cvCloneImage(const IplImage* image);

- image : Imagen original.

La función cvCloneImage cara una copia de la imagen que le pasamos como parámetro.

cvCanny : Aplica el algoritmo Canny para la detección de bordes en la imagen

void cvCanny(const CvArr* img, CvArr* edges, double threshold1, double threshold2, int apertureSize=3);

- img : Imagen de entrada.
- edges : Imagen para almacenar los bordes encontrados
- threshold1 : El primer umbral.
- threshold2 : El Segundo umbral.
- apertureSize : El mismo que para el operador Sobel

La función cvCanny encuentra los bordes de la imagen original y marca en la imagen de salida los bordes encontrados usando el algoritmo Canny. El umbral más pequeño es utilizado para conectar bordes, y el umbral más grande es utilizado para encontrar segmentos iniciales de bordes fuertes.

Apéndice C. Código fuente de la interfaz gráfica

El siguiente código se desarrollo en Visual C++ con ayuda de las librerías de OpenCV.

```
#pragma once
namespace PICSERVOVCExample
{
	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;

	//For using DLL's add this namespace (Needed for DllImportAttribute)
	using namespace System::Runtime::InteropServices;

	#include <windows.h>
	#include "sio_util.h"
	#include "nmccom.h"
	#include "picservo.h"
	// Inicia Para OpenCV
	#include "stdafx.h"                                          //Libraries
	#include <cv.h>
	#include <cxcore.h>
	#include <highgui.h>
	#include <stdio.h>
	#include <string.h>

	#include "conio.h"
	#include "dos.h"
	#include "math.h"
	#include "stdlib.h"

	#define VIDEO_WINDOW   "Webcam"                      //Definitions
	#define USEC_PER_SEC 1000000L
	#define PI 3.141592
	#define NXYZ 290
	#define NIMAX 180


	// ESTRUCTURA PARA LA NUBE DE PUNTOS 3D
	struct cubo{
		int pix[NXYZ][NXYZ][NXYZ];
		struct cubo *sig;
	};

	typedef struct cubo *volumen;

	volumen crea(volumen _v)
	{
		int i,j,k;

		_v = (volumen)malloc(sizeof(struct cubo));
		for(i=0;i<NXYZ;i++)
			for(j=0;j<NXYZ;j++)
				for(k=0;k<NXYZ;k++)
					_v->pix[i][j][k] = 0;
					_v->sig = NULL;
	return _v;
	}

	int kp1;
	int kd1;
	int ki1;
	int kp01;
	int kd01;
	int ki01;
```

```
        int inicioOK =0;
        int nummod =0;
        int numdesp = 0;
        int numimagenes;
        char a[20];
        char* b="hola";
        int i=1;
        int trackbarVal=500;
        int maxVal=1000;
        int pos=500;
        int thresdese;
        IplImage* imagen_inicial_gray = 0;

        public __gc class Form1 : public System::Windows::Forms::Form
        {
        public:
                Form1(void)
                {
                        InitializeComponent();
                }

        protected:
                void Dispose(Boolean disposing)
                {
                        if (disposing && components)
                        {
                                components->Dispose();
                        }
                        __super::Dispose(disposing);
                }


        private: System::Windows::Forms::Button *  ExitButton;
        private: System::Windows::Forms::Button*  adqima;
        private: System::Windows::Forms::NumericUpDown*  numima;
        private: System::Windows::Forms::Label*  label1;
        private: System::Windows::Forms::PictureBox*  pictureBox1;
        private: System::Windows::Forms::TextBox*  textBox1;
        private: System::Windows::Forms::MenuStrip*  menuStrip1;
        private: System::Windows::Forms::ToolStripMenuItem*  iNICIOToolStripMenuItem;
        private:                                        System::Windows::Forms::ToolStripMenuItem*
sELECCIONAPUERTOToolStripMenuItem;
        private:                                        System::Windows::Forms::ToolStripMenuItem*
iNICIALIZACONEXIÓNToolStripMenuItem;
        private: System::Windows::Forms::ToolStripComboBox*  toolStripComboBox1;
        private:                                        System::Windows::Forms::ToolStripMenuItem*
aDQUISICIONDEIMAGENESToolStripMenuItem;
        private: System::Windows::Forms::ToolStripMenuItem*  visualizacion;
        private:                                        System::Windows::Forms::ToolStripMenuItem*
aDQUISICIONIMAGENToolStripMenuItem;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator1;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator4;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator2;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator3;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator5;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator6;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator7;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator8;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator9;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator10;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator11;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator12;
        private: System::Windows::Forms::ProgressBar*  barra;
        private: System::Windows::Forms::ToolStripMenuItem*  obte_coor;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator13;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator14;
        private: System::Windows::Forms::ToolStripMenuItem*  toolStripMenuItem1;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator16;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator15;
        private: System::Windows::Forms::ToolStripMenuItem*  toolStripMenuItem2;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator17;
```

```
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator18;
        private: System::Windows::Forms::ToolStripMenuItem*  toolStripMenuItem3;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator19;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator20;
        private: System::Windows::Forms::ToolStripMenuItem*  toolStripMenuItem4;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator21;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator22;
        private:                                    System::Windows::Forms::ToolStripMenuItem*
aCERCADECICA3DToolStripMenuItem;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator23;
        private: System::Windows::Forms::ToolStripSeparator*  toolStripSeparator24;
        private:


                System::ComponentModel::Container * components;

                void InitializeComponent(void)
                {
                        System::ComponentModel::ComponentResourceManager*   resources  =  (new
System::ComponentModel::ComponentResourceManager(__typeof(Form1)));
                        this->ExitButton = (new System::Windows::Forms::Button());
                        this->adqima = (new System::Windows::Forms::Button());
                        this->numima = (new System::Windows::Forms::NumericUpDown());
                        this->label1 = (new System::Windows::Forms::Label());
                        this->pictureBox1 = (new System::Windows::Forms::PictureBox());
                        this->textBox1 = (new System::Windows::Forms::TextBox());
                        this->menuStrip1 = (new System::Windows::Forms::MenuStrip());
                        this->iNICIOToolStripMenuItem     =     (new     System::Windows::Forms
::ToolStripMenuItem());
                        this->toolStripSeparator1      =      (new      System::Windows::Forms
::ToolStripSeparator());
                        this->toolStripSeparator4      =      (new      System::Windows::Forms
::ToolStripSeparator());
                        this->sELECCIONAPUERTOToolStripMenuItem = (new System::Windows::Forms
::ToolStripMenuItem());
                        this->toolStripComboBox1      =      (new      System::Windows::Forms
::ToolStripComboBox());
                        this->toolStripSeparator2      =      (new      System::Windows::Forms
::ToolStripSeparator());
                        this->toolStripSeparator3      =      (new      System::Windows::Forms
::ToolStripSeparator());
                        this->iNICIALIZACONEXIÓNToolStripMenuItem   =   (new   System::Windows
::Forms::ToolStripMenuItem());
                        this->toolStripSeparator5      =      (new      System::Windows::Forms
::ToolStripSeparator());
                        this->toolStripSeparator6      =      (new      System::Windows::Forms
::ToolStripSeparator());
                        this->aDQUISICIONDEIMAGENESToolStripMenuItem  =  (new  System::Windows
::Forms::ToolStripMenuItem());
                        this->toolStripSeparator16      =      (new      System::Windows::Forms
::ToolStripSeparator());
                        this->toolStripSeparator15      =      (new      System::Windows::Forms
::ToolStripSeparator());
                        this->toolStripMenuItem1      =      (new      System::Windows::Forms
::ToolStripMenuItem());
                        this->toolStripSeparator7      =      (new      System::Windows::Forms
::ToolStripSeparator());
                        this->toolStripSeparator8      =      (new      System::Windows::Forms
::ToolStripSeparator());
                        this->visualizacion        =        (new        System::Windows::Forms
::ToolStripMenuItem());
                        this->toolStripSeparator9      =      (new      System::Windows::Forms
::ToolStripSeparator());
                        this->toolStripSeparator10      =      (new      System::Windows::Forms
::ToolStripSeparator());
                        this->aDQUISICIONIMAGENToolStripMenuItem   =   (new   System::Windows
::Forms::ToolStripMenuItem());
                        this->toolStripSeparator11      =      (new      System::Windows::Forms
::ToolStripSeparator());
                        this->toolStripSeparator12      =      (new      System::Windows::Forms
::ToolStripSeparator());
```

```
                    this->obte_coor = (new System::Windows::Forms::ToolStripMenuItem());
                    this->toolStripSeparator13      =      (new      System::Windows::Forms
::ToolStripSeparator());
                    this->toolStripSeparator14      =      (new      System::Windows::Forms
::ToolStripSeparator());
                    this->toolStripMenuItem2       =       (new      System::Windows::Forms
::ToolStripMenuItem());
                    this->toolStripSeparator17      =      (new      System::Windows::Forms
::ToolStripSeparator());
                    this->toolStripSeparator18      =      (new      System::Windows::Forms
::ToolStripSeparator());
                    this->toolStripMenuItem3       =       (new      System::Windows::Forms
::ToolStripMenuItem());
                    this->toolStripSeparator19      =      (new      System::Windows::Forms
::ToolStripSeparator());
                    this->toolStripSeparator20      =      (new      System::Windows::Forms
::ToolStripSeparator());
                    this->toolStripMenuItem4       =       (new      System::Windows::Forms
::ToolStripMenuItem());
                    this->toolStripSeparator21      =      (new      System::Windows::Forms
::ToolStripSeparator());
                    this->toolStripSeparator22      =      (new      System::Windows::Forms
::ToolStripSeparator());
                    this->aCERCADECICA3DToolStripMenuItem  =  (new  System::Windows::Forms
::ToolStripMenuItem());
                    this->toolStripSeparator23      =      (new      System::Windows::Forms
::ToolStripSeparator());
                    this->toolStripSeparator24      =      (new      System::Windows::Forms
::ToolStripSeparator());
                    this->barra = (new System::Windows::Forms::ProgressBar());
                    (__try_cast<System::ComponentModel::ISupportInitialize*        >(this-
>numima))->BeginInit();
                    (__try_cast<System::ComponentModel::ISupportInitialize*        >(this-
>pictureBox1))->BeginInit();
                    this->menuStrip1->SuspendLayout();
                    this->SuspendLayout();
                    //
                    // ExitButton
                    //
                    this->ExitButton->Image      =      (__try_cast<System::Drawing::Image*
>(resources->GetObject(S"ExitButton.Image")));
                    this->ExitButton->ImageAlign                                          =
System::Drawing::ContentAlignment::MiddleRight;
                    this->ExitButton->Location = System::Drawing::Point(412, 426);
                    this->ExitButton->Name = S"ExitButton";
                    this->ExitButton->Size = System::Drawing::Size(89, 39);
                    this->ExitButton->TabIndex = 5;
                    this->ExitButton->Text = S"SALIR";
                    this->ExitButton->TextAlign                                           =
System::Drawing::ContentAlignment::MiddleLeft;
                    this->ExitButton->Click     +=     new     System::EventHandler(this,
&Form1::ExitButton_Click);
                    //
                    // adqima
                    //
                    this->adqima->Image        =        (__try_cast<System::Drawing::Image*
>(resources->GetObject(S"adqima.Image")));
                    this->adqima->ImageAlign                                              =
System::Drawing::ContentAlignment::MiddleRight;
                    this->adqima->Location = System::Drawing::Point(154, 34);
                    this->adqima->Name = S"adqima";
                    this->adqima->Size = System::Drawing::Size(184, 40);
                    this->adqima->TabIndex = 6;
                    this->adqima->Text = S"ADQUISICIÓN \r\nDE IMÁGENES";
                    this->adqima->Visible = false;
                    this->adqima->Click      +=      new      System::EventHandler(this,
&Form1::button1_Click);
                    //
                    // numima
                    //
                    this->numima->Location = System::Drawing::Point(33, 65);
```

```
                        System::Int32 __mcTemp__1[] = new System::Int32[4];
                        __mcTemp__1[0] = 300;
                        __mcTemp__1[1] = 0;
                        __mcTemp__1[2] = 0;
                        __mcTemp__1[3] = 0;
                        this->numima->Maximum = System::Decimal(__mcTemp__1);
                        System::Int32 __mcTemp__2[] = new System::Int32[4];
                        __mcTemp__2[0] = 5;
                        __mcTemp__2[1] = 0;
                        __mcTemp__2[2] = 0;
                        __mcTemp__2[3] = 0;
                        this->numima->Minimum = System::Decimal(__mcTemp__2);
                        this->numima->Name = S"numima";
                        this->numima->Size = System::Drawing::Size(77, 20);
                        this->numima->TabIndex = 7;
                        System::Int32 __mcTemp__3[] = new System::Int32[4];
                        __mcTemp__3[0] = 5;
                        __mcTemp__3[1] = 0;
                        __mcTemp__3[2] = 0;
                        __mcTemp__3[3] = 0;
                        this->numima->Value = System::Decimal(__mcTemp__3);
                        this->numima->Visible = false;
                        //
                        // label1
                        //
                        this->label1->AutoSize = true;
                        this->label1->Location = System::Drawing::Point(9, 36);
                        this->label1->Name = S"label1";
                        this->label1->Size = System::Drawing::Size(132, 26);
                        this->label1->TabIndex = 9;
                        this->label1->Text = S"NÚMERO DE IMAGENES\r\n           A ADQUIRIR";
                        this->label1->Visible = false;
                        //
                        // pictureBox1
                        //
                        this->pictureBox1->Image     =     (__try_cast<System::Drawing::Image*
>(resources->GetObject(S"pictureBox1.Image")));
                        this->pictureBox1->Location = System::Drawing::Point(2, 424);
                        this->pictureBox1->Name = S"pictureBox1";
                        this->pictureBox1->Size = System::Drawing::Size(408, 45);
                        this->pictureBox1->TabIndex = 10;
                        this->pictureBox1->TabStop = false;
                        //
                        // textBox1
                        //
                        this->textBox1->Location = System::Drawing::Point(351, 59);
                        this->textBox1->Multiline = true;
                        this->textBox1->Name = S"textBox1";
                        this->textBox1->Size = System::Drawing::Size(145, 26);
                        this->textBox1->TabIndex = 13;
                        this->textBox1->Visible = false;
                        //
                        // menuStrip1
                        //
                        this->menuStrip1->BackColor                                          =
System::Drawing::SystemColors::ButtonHighlight;
                        System::Windows::Forms::ToolStripItem*     __mcTemp__4[]     =     new
System::Windows::Forms::ToolStripItem*[3];
                        __mcTemp__4[0] = this->iNICIOToolStripMenuItem;
                        __mcTemp__4[1] = this->aDQUISICIONDEIMAGENESToolStripMenuItem;
                        __mcTemp__4[2] = this->toolStripMenuItem4;
                        this->menuStrip1->Items->AddRange(__mcTemp__4);
                        this->menuStrip1->Location = System::Drawing::Point(0, 0);
                        this->menuStrip1->Name = S"menuStrip1";
                        this->menuStrip1->Size = System::Drawing::Size(508, 24);
                        this->menuStrip1->TabIndex = 15;
                        this->menuStrip1->Text = S"menuStrip1";
                        //
                        // iNICIOToolStripMenuItem
                        //
```

```
                    System::Windows::Forms::ToolStripItem*     __mcTemp__5[]     =     new
System::Windows::Forms::ToolStripItem*[8];
                    __mcTemp__5[0] = this->toolStripSeparator1;
                    __mcTemp__5[1] = this->toolStripSeparator4;
                    __mcTemp__5[2] = this->sELECCIONAPUERTOToolStripMenuItem;
                    __mcTemp__5[3] = this->toolStripSeparator2;
                    __mcTemp__5[4] = this->toolStripSeparator3;
                    __mcTemp__5[5] = this->iNICIALIZACONEXIÓNToolStripMenuItem;
                    __mcTemp__5[6] = this->toolStripSeparator5;
                    __mcTemp__5[7] = this->toolStripSeparator6;
                    this->iNICIOToolStripMenuItem->DropDownItems->AddRange(__mcTemp__5);
                    this->iNICIOToolStripMenuItem->Name = S"iNICIOToolStripMenuItem";
                    this->iNICIOToolStripMenuItem->Size = System::Drawing::Size(53, 20);
                    this->iNICIOToolStripMenuItem->Text = S"INICIO";
                    //
                    // toolStripSeparator1
                    //
                    this->toolStripSeparator1->Name = S"toolStripSeparator1";
                    this->toolStripSeparator1->Size = System::Drawing::Size(193, 6);
                    //
                    // toolStripSeparator4
                    //
                    this->toolStripSeparator4->Name = S"toolStripSeparator4";
                    this->toolStripSeparator4->Size = System::Drawing::Size(193, 6);
                    //
                    // sELECCIONAPUERTOToolStripMenuItem
                    //
                    System::Windows::Forms::ToolStripItem*     __mcTemp__6[]     =     new
System::Windows::Forms::ToolStripItem*[1];
                    __mcTemp__6[0] = this->toolStripComboBox1;
                    this->sELECCIONAPUERTOToolStripMenuItem->DropDownItems-
>AddRange(__mcTemp__6);
                    this->sELECCIONAPUERTOToolStripMenuItem->Name                          =
S"sELECCIONAPUERTOToolStripMenuItem";
                    this->sELECCIONAPUERTOToolStripMenuItem->Size                          =
System::Drawing::Size(196, 22);
                    this->sELECCIONAPUERTOToolStripMenuItem->Text = S"SELECCIONA PUERTO";
                    //
                    // toolStripComboBox1
                    //
                    System::Object* __mcTemp__7[] = new System::Object*[11];
                    __mcTemp__7[0] = S"COM1:";
                    __mcTemp__7[1] = S"COM2:";
                    __mcTemp__7[2] = S"COM3:";
                    __mcTemp__7[3] = S"COM4:";
                    __mcTemp__7[4] = S"COM5:";
                    __mcTemp__7[5] = S"COM6:";
                    __mcTemp__7[6] = S"COM7:";
                    __mcTemp__7[7] = S"COM8:";
                    __mcTemp__7[8] = S"COM9:";
                    __mcTemp__7[9] = S"COM10:";
                    __mcTemp__7[10] = S"COM11:";
                    this->toolStripComboBox1->Items->AddRange(__mcTemp__7);
                    this->toolStripComboBox1->Name = S"toolStripComboBox1";
                    this->toolStripComboBox1->Size = System::Drawing::Size(152, 21);
                    this->toolStripComboBox1->Click   +=   new   System::EventHandler(this,
&Form1::toolStripComboBox1_Click);
                    //
                    // toolStripSeparator2
                    //
                    this->toolStripSeparator2->Name = S"toolStripSeparator2";
                    this->toolStripSeparator2->Size = System::Drawing::Size(193, 6);
                    //
                    // toolStripSeparator3
                    //
                    this->toolStripSeparator3->Name = S"toolStripSeparator3";
                    this->toolStripSeparator3->Size = System::Drawing::Size(193, 6);
                    //
                    // iNICIALIZACONEXIÓNToolStripMenuItem
                    //
```

```
                    this->iNICIALIZACONEXIÓNToolStripMenuItem->Name                    =
S"iNICIALIZACONEXIÓNToolStripMenuItem";
                    this->iNICIALIZACONEXIÓNToolStripMenuItem->Size                    =
System::Drawing::Size(196, 22);
                    this->iNICIALIZACONEXIÓNToolStripMenuItem->Text     =     S"INICIALIZA
CONEXIÓN";
                    this->iNICIALIZACONEXIÓNToolStripMenuItem->Click        +=        new
System::EventHandler(this, &Form1::iNICIALIZACONEXIÓNToolStripMenuItem_Click);
                    //
                    // toolStripSeparator5
                    //
                    this->toolStripSeparator5->Name = S"toolStripSeparator5";
                    this->toolStripSeparator5->Size = System::Drawing::Size(193, 6);
                    //
                    // toolStripSeparator6
                    //
                    this->toolStripSeparator6->Name = S"toolStripSeparator6";
                    this->toolStripSeparator6->Size = System::Drawing::Size(193, 6);
                    //
                    // aDQUISICIONDEIMAGENESToolStripMenuItem
                    //
                    System::Windows::Forms::ToolStripItem*     __mcTemp__8[]     =     new
System::Windows::Forms::ToolStripItem*[20];
                    __mcTemp__8[0] = this->toolStripSeparator16;
                    __mcTemp__8[1] = this->toolStripSeparator15;
                    __mcTemp__8[2] = this->toolStripMenuItem1;
                    __mcTemp__8[3] = this->toolStripSeparator7;
                    __mcTemp__8[4] = this->toolStripSeparator8;
                    __mcTemp__8[5] = this->visualizacion;
                    __mcTemp__8[6] = this->toolStripSeparator9;
                    __mcTemp__8[7] = this->toolStripSeparator10;
                    __mcTemp__8[8] = this->aDQUISICIONIMAGENToolStripMenuItem;
                    __mcTemp__8[9] = this->toolStripSeparator11;
                    __mcTemp__8[10] = this->toolStripSeparator12;
                    __mcTemp__8[11] = this->obte_coor;
                    __mcTemp__8[12] = this->toolStripSeparator13;
                    __mcTemp__8[13] = this->toolStripSeparator14;
                    __mcTemp__8[14] = this->toolStripMenuItem2;
                    __mcTemp__8[15] = this->toolStripSeparator17;
                    __mcTemp__8[16] = this->toolStripSeparator18;
                    __mcTemp__8[17] = this->toolStripMenuItem3;
                    __mcTemp__8[18] = this->toolStripSeparator19;
                    __mcTemp__8[19] = this->toolStripSeparator20;
                    this->aDQUISICIONDEIMAGENESToolStripMenuItem->DropDownItems-
>AddRange(__mcTemp__8);
                    this->aDQUISICIONDEIMAGENESToolStripMenuItem->Name                  =
S"aDQUISICIONDEIMAGENESToolStripMenuItem";
                    this->aDQUISICIONDEIMAGENESToolStripMenuItem->Size                  =
System::Drawing::Size(158, 20);
                    this->aDQUISICIONDEIMAGENESToolStripMenuItem->Text = S"ADQUISICION DE
IMAGENES";
                    //
                    // toolStripSeparator16
                    //
                    this->toolStripSeparator16->Name = S"toolStripSeparator16";
                    this->toolStripSeparator16->Size = System::Drawing::Size(243, 6);
                    //
                    // toolStripSeparator15
                    //
                    this->toolStripSeparator15->Name = S"toolStripSeparator15";
                    this->toolStripSeparator15->Size = System::Drawing::Size(243, 6);
                    //
                    // toolStripMenuItem1
                    //
                    this->toolStripMenuItem1->Name = S"toolStripMenuItem1";
                    this->toolStripMenuItem1->Size = System::Drawing::Size(246, 22);
                    this->toolStripMenuItem1->Text = S"VISUALIZACION SIN OBJETO";
                    this->toolStripMenuItem1->Click   +=   new   System::EventHandler(this,
&Form1::toolStripMenuItem1_Click);
                    //
                    // toolStripSeparator7
```

```
                    //
                    this->toolStripSeparator7->Name = S"toolStripSeparator7";
                    this->toolStripSeparator7->Size = System::Drawing::Size(243, 6);
                    //
                    // toolStripSeparator8
                    //
                    this->toolStripSeparator8->Name = S"toolStripSeparator8";
                    this->toolStripSeparator8->Size = System::Drawing::Size(243, 6);
                    //
                    // visualizacion
                    //
                    this->visualizacion->Name = S"visualizacion";
                    this->visualizacion->Size = System::Drawing::Size(246, 22);
                    this->visualizacion->Text = S"AJUSTE DE VIDEO";
                    this->visualizacion->Click    +=    new    System::EventHandler(this,
&Form1::nÚMERODEIMÁGENESToolStripMenuItem_Click);
                    //
                    // toolStripSeparator9
                    //
                    this->toolStripSeparator9->Name = S"toolStripSeparator9";
                    this->toolStripSeparator9->Size = System::Drawing::Size(243, 6);
                    //
                    // toolStripSeparator10
                    //
                    this->toolStripSeparator10->Name = S"toolStripSeparator10";
                    this->toolStripSeparator10->Size = System::Drawing::Size(243, 6);
                    //
                    // aDQUISICIONIMAGENToolStripMenuItem
                    //
                    this->aDQUISICIONIMAGENToolStripMenuItem->Name                      =
S"aDQUISICIONIMAGENToolStripMenuItem";
                    this->aDQUISICIONIMAGENToolStripMenuItem->Size                      =
System::Drawing::Size(246, 22);
                    this->aDQUISICIONIMAGENToolStripMenuItem->Text   =   S"ADQUISICION   DE
IMAGENES";
                    this->aDQUISICIONIMAGENToolStripMenuItem->Click         +=         new
System::EventHandler(this, &Form1::aDQUISICIONIMAGENToolStripMenuItem_Click);
                    //
                    // toolStripSeparator11
                    //
                    this->toolStripSeparator11->Name = S"toolStripSeparator11";
                    this->toolStripSeparator11->Size = System::Drawing::Size(243, 6);
                    //
                    // toolStripSeparator12
                    //
                    this->toolStripSeparator12->Name = S"toolStripSeparator12";
                    this->toolStripSeparator12->Size = System::Drawing::Size(243, 6);
                    //
                    // obte_coor
                    //
                    this->obte_coor->Name = S"obte_coor";
                    this->obte_coor->Size = System::Drawing::Size(246, 22);
                    this->obte_coor->Text = S"EXTRACCIÓN DE COORDENADAS";
                    this->obte_coor->Click     +=     new     System::EventHandler(this,
&Form1::obte_coor_Click);
                    //
                    // toolStripSeparator13
                    //
                    this->toolStripSeparator13->Name = S"toolStripSeparator13";
                    this->toolStripSeparator13->Size = System::Drawing::Size(243, 6);
                    //
                    // toolStripSeparator14
                    //
                    this->toolStripSeparator14->Name = S"toolStripSeparator14";
                    this->toolStripSeparator14->Size = System::Drawing::Size(243, 6);
                    //
                    // toolStripMenuItem2
                    //
                    this->toolStripMenuItem2->Name = S"toolStripMenuItem2";
                    this->toolStripMenuItem2->Size = System::Drawing::Size(246, 22);
                    this->toolStripMenuItem2->Text = S"RECONSTRUCCION 3D";
```

```
                    this->toolStripMenuItem2->Click  +=  new  System::EventHandler(this,
&Form1::toolStripMenuItem2_Click);
                    //
                    // toolStripSeparator17
                    //
                    this->toolStripSeparator17->Name = S"toolStripSeparator17";
                    this->toolStripSeparator17->Size = System::Drawing::Size(243, 6);
                    //
                    // toolStripSeparator18
                    //
                    this->toolStripSeparator18->Name = S"toolStripSeparator18";
                    this->toolStripSeparator18->Size = System::Drawing::Size(243, 6);
                    //
                    // toolStripMenuItem3
                    //
                    this->toolStripMenuItem3->Name = S"toolStripMenuItem3";
                    this->toolStripMenuItem3->Size = System::Drawing::Size(246, 22);
                    this->toolStripMenuItem3->Text = S"CALCULO DEL VOLUMEN";
                    //
                    // toolStripSeparator19
                    //
                    this->toolStripSeparator19->Name = S"toolStripSeparator19";
                    this->toolStripSeparator19->Size = System::Drawing::Size(243, 6);
                    //
                    // toolStripSeparator20
                    //
                    this->toolStripSeparator20->Name = S"toolStripSeparator20";
                    this->toolStripSeparator20->Size = System::Drawing::Size(243, 6);
                    //
                    // toolStripMenuItem4
                    //
                    System::Windows::Forms::ToolStripItem*     __mcTemp__9[]     =     new
System::Windows::Forms::ToolStripItem*[5];
                    __mcTemp__9[0] = this->toolStripSeparator21;
                    __mcTemp__9[1] = this->toolStripSeparator22;
                    __mcTemp__9[2] = this->aCERCADECICA3DToolStripMenuItem;
                    __mcTemp__9[3] = this->toolStripSeparator23;
                    __mcTemp__9[4] = this->toolStripSeparator24;
                    this->toolStripMenuItem4->DropDownItems->AddRange(__mcTemp__9);
                    this->toolStripMenuItem4->Name = S"toolStripMenuItem4";
                    this->toolStripMenuItem4->Size = System::Drawing::Size(53, 20);
                    this->toolStripMenuItem4->Text = S"AYUDA";
                    //
                    // toolStripSeparator21
                    //
                    this->toolStripSeparator21->Name = S"toolStripSeparator21";
                    this->toolStripSeparator21->Size = System::Drawing::Size(180, 6);
                    //
                    // toolStripSeparator22
                    //
                    this->toolStripSeparator22->Name = S"toolStripSeparator22";
                    this->toolStripSeparator22->Size = System::Drawing::Size(180, 6);
                    //
                    // aCERCADECICA3DToolStripMenuItem
                    //
                    this->aCERCADECICA3DToolStripMenuItem->Name                          =
S"aCERCADECICA3DToolStripMenuItem";
                    this->aCERCADECICA3DToolStripMenuItem->Size                          =
System::Drawing::Size(183, 22);
                    this->aCERCADECICA3DToolStripMenuItem->Text = S"ACERCA DE CICA3D";
                    this->aCERCADECICA3DToolStripMenuItem->Click          +=          new
System::EventHandler(this, &Form1::aCERCADECICA3DToolStripMenuItem_Click);
                    //
                    // toolStripSeparator23
                    //
                    this->toolStripSeparator23->Name = S"toolStripSeparator23";
                    this->toolStripSeparator23->Size = System::Drawing::Size(180, 6);
                    //
                    // toolStripSeparator24
                    //
                    this->toolStripSeparator24->Name = S"toolStripSeparator24";
```

```
                        this->toolStripSeparator24->Size = System::Drawing::Size(180, 6);
                        //
                        // barra
                        //
                        this->barra->Location = System::Drawing::Point(346, 34);
                        this->barra->Maximum = 1000;
                        this->barra->Name = S"barra";
                        this->barra->Size = System::Drawing::Size(159, 22);
                        this->barra->Style                                                    =
System::Windows::Forms::ProgressBarStyle::Continuous;
                        this->barra->TabIndex = 16;
                        this->barra->Visible = false;
                        //
                        // Form1
                        //
                        this->AutoScaleBaseSize = System::Drawing::Size(5, 13);
                        this->BackgroundImage      =      (__try_cast<System::Drawing::Image*
>(resources->GetObject(S"$this.BackgroundImage")));
                        this->ClientSize = System::Drawing::Size(508, 468);
                        this->Controls->Add(this->barra);
                        this->Controls->Add(this->textBox1);
                        this->Controls->Add(this->pictureBox1);
                        this->Controls->Add(this->label1);
                        this->Controls->Add(this->numima);
                        this->Controls->Add(this->adqima);
                        this->Controls->Add(this->ExitButton);
                        this->Controls->Add(this->menuStrip1);
                        this->Icon   =   (__try_cast<System::Drawing::Icon*      >(resources-
>GetObject(S"$this.Icon")));
                        this->MainMenuStrip = this->menuStrip1;
                        this->Name = S"Form1";
                        this->StartPosition                                                   =
System::Windows::Forms::FormStartPosition::CenterScreen;
                        this->Text = S"CICA3D";
                        this->Load += new System::EventHandler(this, &Form1::Form1_Load);
                        this->Closing += new System::ComponentModel::CancelEventHandler(this,
&Form1::Form1_Closing);
                        (__try_cast<System::ComponentModel::ISupportInitialize*        >(this-
>numima))->EndInit();
                        (__try_cast<System::ComponentModel::ISupportInitialize*        >(this-
>pictureBox1))->EndInit();
                        this->menuStrip1->ResumeLayout(false);
                        this->menuStrip1->PerformLayout();
                        this->ResumeLayout(false);
                        this->PerformLayout();

                }

        private: System::Void ExitButton_Click(System::Object *  sender, System::EventArgs *
e)
                         {
                                Close();
                         }

        private:      System::Void      Form1_Closing(System::Object      *            sender,
System::ComponentModel::CancelEventArgs *  e)
                         {
                                if (nummod == 0) return;
                                NmcShutdown();

                         }

private: System::Void button1_Click(System::Object*  sender, System::EventArgs*  e) {
                         i=1;
                         int x,bmax,bmin;
                         bmax=0;
                         bmin=0;
        textBox1->Text="_";
        numimagenes=int(numima->Value);

        CvCapture* capture = 0;                              // Variable to capture
```

```
        IplImage* curr_frame = 0;                   // current video frame
        IplImage* gray_frame = 0;                   // grayscale version of current frame
        IplImage* nueva_im = 0;
        int w, h;

                                                                                    // Capture from a
webcam
        cvSetCaptureProperty( capture, CV_CAP_PROP_FRAME_WIDTH,  1600 );  //1280         960
//1600 x 1200
        cvSetCaptureProperty( capture, CV_CAP_PROP_FRAME_HEIGHT, 1200 );  //capture        =
cvCaptureFromCAM(CV_CAP_ANY);
        capture = cvCreateCameraCapture(0);


        if ( !capture) {                                    // No camera error handling
                textBox1->Text="No hay cámara";
        }
        for (;;){
                cvSetCaptureProperty(  capture,  CV_CAP_PROP_FRAME_WIDTH,   1600 );//1280  960
//1600 x 1200
            cvSetCaptureProperty( capture, CV_CAP_PROP_FRAME_HEIGHT, 1200 );      //
                curr_frame = cvQueryFrame(capture);
                if ( !curr_frame) {                         // No frame error handling
                        //fprintf(stderr, "ERROR: frame is null... Exiting\n");
                        textBox1->Text="No hay imagen";
                        break;
                }                                                                   // DO NOT RELEASE
THE FRAME!

                w = curr_frame->width;              // Get frame size
                h = curr_frame->height;

                if( ! gray_frame ) {                // Convert the frame image to grayscale
                        int channels = 1;                   //  One  channel  for  grayscale
image
                        gray_frame = cvCreateImage(   // destiny image from conversion
                                        cvGetSize(curr_frame),
                                        IPL_DEPTH_8U, channels);
                }                                                                   //      BGR      TO
GRAYSCALE conversion

                cvCvtColor(curr_frame, gray_frame, CV_BGR2GRAY);

                IplImage* resta;
                resta=cvCreateImage(cvSize(w,h),IPL_DEPTH_8U,1);
                cvAbsDiff(gray_frame,imagen_inicial_gray, resta );

                cvThreshold(resta,resta,10,255,CV_THRESH_OTSU);
                IplImage* nueva_im=0;
                nueva_im=cvCreateImage(cvSize(w,h),IPL_DEPTH_8U,1);
                int ii,jj;
                CvScalar s;

                for (ii=0;ii<=h-1;ii=ii+1){
                        for (jj=0;jj<=w-1;jj=jj+1){
                        s=cvGet2D(resta,ii,jj);
                        cvSet2D(nueva_im,h-1-ii,jj,s);
                        }
                }

                sprintf_s(a,"%s%d",b,i);
                strcat_s(a,".jpg");
                //cvSaveImage(a,gray_frame);
                cvSaveImage(a,nueva_im);
                //strcpy_s(a, " ");

                Sleep(2000);   //Tiempo de la pausa entre cada fot

                //CONTRL DEL MOTORLLLLDAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
                                        byte statbyte;
```

```
                              ServoLoadTraj(1,              //addr = 0
                                                      LOAD_POS  |  LOAD_VEL  |  LOAD_ACC  |
ENABLE_SERVO | START_NOW,
                                                    i*29400/(2*numimagenes),          //pos     =
2000
                                                    50000,      //vel = 100,000
                                                    100,               //acc = 100
                                                    0                  //pwm = 0
                                                    );

                              do
                                {
                                NmcNoOp(1);  //poll controller to get current status data
                                statbyte = NmcGetStat(1);
                                }
                              while ( !(statbyte & MOVE_DONE) );  //wait for MOVE_DONE bit
to go HIGH


                              bmin=1000*(i-1)/numimagenes;
                              bmax=1000*i/numimagenes;
                              for(x=bmin;x<=bmax;x=x+1) {
                                     Sleep(1);
                                     barra->Value=int(x);
                              }
                              Sleep(1000);
               i=i+1;   // Incremeta el indice del nombre de la foto o imagen adquirida
               //imamos->Image=gray_frame;
               if (i==numimagenes+1)  break;      //aqui se determina el numero de imagenes
que desamos tomar
        }
        // Release the capture device housekeeping
        cvReleaseCapture( &capture);
        cvDestroyWindow(VIDEO_WINDOW);


        textBox1->Text="imagenes tomadas";
        //textBox1->Text="imagenes tomadas";

                }


private: System::Void Form1_Load(System::Object*  sender, System::EventArgs*  e) {
                }



private: System::Void toolStripComboBox1_Click(System::Object*   sender, System::EventArgs*
e) {
                }
private:  System::Void  iNICIALIZACONEXIÓNToolStripMenuItem_Click(System::Object*    sender,
System::EventArgs*  e) {
                           int c;
                              char puerto[20];
                              char* comm="COM";
                              char datastr[80];
                              //c = int ( combo->SelectedIndex)+1;
                              c = int ( toolStripComboBox1->SelectedIndex)+1;
                              sprintf_s(puerto,"%s%d",comm,c);
                              strcat_s(puerto,":");

                              nummod = NmcInit(puerto, 19200);  //Controllers on COM1, use
19200 baud

        //Returns the number of modules found
                              if (nummod==0)
                                     {
                                     SimpleMsgBox("PIC-SERVO Módulo no encontrado!!!");
                                     //Close();
                                     }
                              else
```

```
                                {
                                if (NmcGetModType(1) == SERVOMODTYPE)  //Determine the type
for module 1
                                   {
                                   sprintf(datastr,"PIC-SERVO Controlador Encontado");
                                   SimpleMsgBox(datastr);
                                      kp01=100;
                                      kd01=1000;
                                      ki01=10;

                                      //            setGains(  1,kp01,kd01,ki01);//setGains(
int Motor,Kp,Kd,Ki)

                                   //             motEnableDis(1,1);
                                   ServoSetGain(1,     //axis = 1
                                                  kp01,  //Kp = 100
                                                kd01, //Kd = 1000
                                                   ki01,            //Ki = 0
                                                      0,            //IL = 0
                                                      255,   //OL = 255
                                                      0,            //CL = 0
                                                      4000,  //EL = 4000
                                                      1,            //SR = 1
                                                      1             //DC = 0
                                                      );

                                   ServoStopMotor(1,  AMP_ENABLE  |  MOTOR_OFF);   //enable
amp
                                   ServoStopMotor(1, AMP_ENABLE | STOP_ABRUPT);  //stop at
current pos.
                                   ServoResetPos(1);                              //reset
the posiiton counter to 0


                                   iNICIOToolStripMenuItem->Enabled = false;
                                   //aDQUISICIONDEIMAGENESToolStripMenuItem->Enabled     =
true;


                                   }
                                else
                                  {
                                  SimpleMsgBox("Módulo no encontrado");
                                  //Close();
                                  }
                                }
                }
private:   System::Void   nÚMERODEIMÁGENESToolStripMenuItem_Click(System::Object*     sender,
System::EventArgs*  e) {

                i=1;


        CvCapture* capture = 0;                          // Variable to capture
        IplImage* curr_frame = 0;                  // current video frame
        IplImage* gray_frame = 0;                  // grayscale version of current frame

        int w, h;                                                  // video frame size

                                                                   // Capture from a
webcam
        //capture = cvCaptureFromCAM(CV_CAP_ANY);
        capture = cvCreateCameraCapture(0);
        cvSetCaptureProperty( capture, CV_CAP_PROP_FRAME_WIDTH,  1280 );//1280 960    //1600
x 1200
        cvSetCaptureProperty( capture, CV_CAP_PROP_FRAME_HEIGHT, 960 );
        cvNamedWindow(VIDEO_WINDOW, 0);


        if ( !capture) {                                 // No camera error handling
                textBox1->Text="No hay cámara";
```

```
        }
        for (;;){

                curr_frame = cvQueryFrame(capture);
                if ( !curr_frame) {                          // No frame error handling
                        textBox1->Text="No hay imagen";
                        break;
                }

                w = curr_frame->width;
                h = curr_frame->height;

                if( ! gray_frame ) {                 // Convert the frame image to grayscale
                        int channels = 1;                    //  One  chafffnnel  for  grayscale
image
                        gray_frame = cvCreateImage(   // destiny image from conversion
                                        cvGetSize(curr_frame),
                                        IPL_DEPTH_8U, channels);
                }                                                                //      BGR      TO
GRAYSCALE conversion
                cvCvtColor(curr_frame, gray_frame, CV_BGR2GRAY);


                IplImage* resta;
                resta=cvCreateImage(cvSize(w,h),IPL_DEPTH_8U,1);
                cvAbsDiff(gray_frame,imagen_inicial_gray, resta );
                cvSaveImage("resta.jpg",resta);

                cvThreshold(resta,resta,0,255,CV_THRESH_OTSU);

                IplImage* nueva_im=0;
                nueva_im=cvCreateImage(cvSize(w,h),IPL_DEPTH_8U,1);
                int ii,jj;
                CvScalar s;

                for (ii=0;ii<=h-1;ii=ii+1){
                        for (jj=0;jj<=w-1;jj=jj+1){
                        s=cvGet2D(resta,ii,jj);
                        cvSet2D(nueva_im,h-1-ii,jj,s);
                        }
                }

                cvShowImage(VIDEO_WINDOW, nueva_im);

                if ( (cvWaitKey(10) & 255) == 27)  break;      //aqui se determina el numero
de imagenes que desamos tomar
        }
        cvReleaseCapture( &capture);
        cvDestroyWindow(VIDEO_WINDOW);

        }

 void trackbarHandler(int pos)
  {
    printf("Trackbar position: %d\n",pos);
  }

private:  System::Void  aDQUISICIONIMAGENToolStripMenuItem_Click(System::Object*    sender,
System::EventArgs*  e) {
                                    label1->Visible = true;
                                numima->Visible = true;
                                adqima->Visible = true;
                                textBox1->Visible = true;
                                barra->Visible = true;
                 }
private:         System::Void         umbdes_Scroll(System::Object*                   sender,
System::Windows::Forms::ScrollEventArgs*  e) {
                 }
private: System::Void obte_coor_Click(System::Object*  sender, System::EventArgs*  e) {

                        char aa[20];
```

```
                        char* ruta="hola";
                        char bb[20];
                        char* ruta2="uno";
                        int j,k;
                        CvScalar s;
                        IplImage* imagen=NULL;

                        for(i=1;i<=numimagenes;i=i+1){
                               sprintf_s(aa,"%s%d",ruta,i);
                               strcat_s(aa,".jpg");
                               imagen=cvLoadImage(aa,-1);
                               int height = imagen-> height;
                               int width = imagen-> width;

                               FILE *stream;
                               sprintf_s(bb,"%s%d",ruta2,i);
                               strcat_s(bb,".txt");
                               stream = fopen(bb,"w+");

                               for(j=0;j<=height-1;j=j+1){
                                      for(k=0;k<=width-1;k=k+1){
                                             s=cvGet2D(imagen,j,k);

                                             if (s.val[0]>=200){
                                                    fprintf(stream, "%d  %d \n", j , k);
                                             }
                                      }
                               }
                               fclose(stream);
                               cvReleaseImage(&imagen);
                        }
                        textBox1->Text="Coordenadas obtenidas";
                 }


private: System::Void toolStripMenuItem1_Click(System::Object*   sender, System::EventArgs*
e) {
                                      i=0;
       IplImage* imagen_inicial = 0;
       CvCapture* capture = 0;                              // Variable to capture

       capture = cvCreateCameraCapture(0);
       cvSetCaptureProperty( capture, CV_CAP_PROP_FRAME_WIDTH,  1280 );//1280 960    //1600
x 1200
       cvSetCaptureProperty( capture, CV_CAP_PROP_FRAME_HEIGHT, 960 );

       if ( !capture) {                                     // No camera error handling
              textBox1->Text="No hay cámara";
       }
       for (;;){
              imagen_inicial = cvQueryFrame(capture);
              if ( !imagen_inicial) {                              //    No    frame    error
handling
                     textBox1->Text="No hay imagen";
                     break;
              }


              if( ! imagen_inicial_gray ) {                //  Convert  the  frame  image  to
grayscale
                     int channels = 1;                     //  One  channel  for  grayscale
image
                     imagen_inicial_gray = cvCreateImage( // destiny image from conversion
                                   cvGetSize(imagen_inicial),
                                   IPL_DEPTH_8U, channels);
              }                                                          //      BGR      TO
GRAYSCALE conversion
              cvCvtColor(imagen_inicial, imagen_inicial_gray, CV_BGR2GRAY);     //SE GUARDA
EN GRAY_...
```

```
            i=i+1;
            if (i==2)  break;      //aqui se determina el numero de imagenes que desamos
tomar
       }

       cvSaveImage("gabooo.jpg",imagen_inicial_gray);
       cvReleaseCapture( &capture);
       cvDestroyWindow(VIDEO_WINDOW);
             }
private:   System::Void   aCERCADECICA3DToolStripMenuItem_Click(System::Object*      sender,
System::EventArgs*  e) {

                           char datastr[350];
                           sprintf(datastr,"                              CICA3D versión 1.0
\n\n              Desarrollado por Gabriel González Flores   \n\nSupervisado por E.C.C. -
J.J.G.B. - J.A.H.R. - F.J.O.R \n\n                              CICATA Unidad Querétaro\n\n
INSTITUTO POLITÉCNICO NACIONAL \n\n           - - La Técnica al Servicio de la Patria - -");


                           SimpleMsgBox(datastr);

             }
private: System::Void toolStripMenuItem2_Click(System::Object*   sender, System::EventArgs*
e) {
       double ct[NIMAX+1], st[NIMAX+1], ctn[NIMAX+1], stn[NIMAX+1], teta[NIMAX+1];
       int x1[NXYZ*NXYZ], y1[NXYZ*NXYZ], z1[NXYZ*NXYZ];
       char nom[100],nomsal[100],path[100]="C:\\uno\\";
       char raiz[10], num[3];


       FILE *fp, *fq;
       volumen vol;
       int Z, X1, Y1, nim;
       int i, ii, j, c, vt, nl,luz, N;
       double k, xn, yn, X, Y;

       int yaxe, inca;

       vol=crea(vol);
       printf(" HOLA !! \n");
       printf("\nNumero de imagenes: ");//19
       scanf("%d",&nim);
       printf("\nPosicion del eje (pixeles): ");//40
       scanf("%d",&yaxe);
       printf("\nIncremento angular (grados): ");//10
       scanf("%d",&inca);
       printf("\nNombre raiz de archivos de coordenadas: ");//soldado
       scanf("%s",&raiz);

       // Creacion del nombre del archivo de salida:
       strcpy(nomsal,path); strcat(nomsal,raiz); strcat(nomsal,"\\CXY");
       /*strcat(nomsal,raiz);*/ strcat(nomsal,"vol.txt");
       printf("\nNombre de archivo de salida: %s \n",nomsal);
       if((fq=fopen(nomsal,"w"))==NULL) {
                     printf("\nNo se puede abrir el archivo resultado CXYvol.txt \n");
                     exit(0);
              }
       for (i=1; i<=nim; i++){
              teta[i]=(PI*inca*(i-1))/180.0;
              ct[i]=cos(teta[i]);
              st[i]=sin(teta[i]);
              ctn[i]=cos(teta[i]+PI/2.0);
              stn[i]=sin(teta[i]+PI/2.0);
              printf("\nTeta %d: %f \n",i,teta[i]);
       }
       nl=0;
       ii=0;
       c=0;
       vt=0;

       for (i=1;i<=nim;i++){
```

```
		printf("\n\nPROCESANDO IMAGEN ... %d \n",i);

		strcpy(nom,path); strcat(nom,raiz); strcat(nom,"\\");strcat(nom,raiz);
		itoa(i,num,10); strcat(nom,num); strcat(nom,".txt");
		printf("\nNombre de archivo %d: %s \n",i,nom);
		if((fp=fopen(nom,"r"))==NULL) {
			printf("\nNo se puede abrir el archivo %d \n",i);
			exit(0);
		}
		while(!feof(fp)) {
			nl++;
			ii++;
			//fscanf(fp,"%d\t%d\n",&y1[nl],&z1[nl]);
			fscanf(fp,"%d\t%d\n",&z1[nl],&y1[nl]);
			y1[nl]=y1[nl]-yaxe;
			z1[nl]=z1[nl]+2;
			x1[nl]=0;
		}
		printf("\nNumero de elementos, archivo %d: %d \n",i,nl);
		fclose(fp);
		N=nl;
		nl=0;
		k=-NXYZ/2.0;
		do {
			xn=-stn[i]*k;
			yn=ctn[i]*k;
			for (j=1; j<=N; j++){
				X=ct[i]*x1[j]-st[i]*y1[j];
				Y=st[i]*x1[j]+ct[i]*y1[j];
				Z=z1[j];
				X1 = floor(xn+X+yaxe+0.5);
				Y1 = floor(yn+Y+yaxe+0.5);
				if ((X1 < NXYZ) && (X1> 0)){
					if ((Y1 < NXYZ) && (Y1 > 0)){
						if (vol->pix[Y1][X1][Z]==(i-1)){
							vol->pix[Y1][X1][Z]=i;
								if (i==nim){
									vt=vt+1;

	fprintf(fq,"%d\t%d\t%d\n",X1,Y1,Z);
								} // end if i=nim
						} // end de if vol
					} // end de abs Y
				} // end de abs X
			}
			//k=k+0.03;
			k=k+0.2;
			//k=k+.6;
		} while (k<=NXYZ/2.0); //numero de barridos
	} // end For numero de imagenes
	printf("\n\n VOLUMEN TOTAL [voxeles]: %d \n",vt);
	printf("\nADIOS !!\n!");
	fclose(fq);

	 }
};
}
```