UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSÍ

Facultad de Ingeniería

Centro de Investigación y Estudios de Posgrado

"RECONOCIMIENTO DE OBJETOS POR VISIÓN ARTIFICIAL USANDO MODELOS SÓLIDOS CAD"

TESIS

que para obtener el grado de

Maestro en Ingeniería Mecánica

presenta:

ING. OMAR DANIEL RODRÍGUEZ GUTIÉRREZ

Asesor:

DR. DIRK F. DE LANGE

San Luis Potosí, S.L.P Febrero 2012

UNIVERSIDAD AUTÓNOMA
DE SAN LUIS POTOSÍ

20 de octubre de 2011

ING. OMAR DANIEL RODRÍGUEZ GUTIÉRREZ
P R E S E N T E. –

En atención a su solicitud de Temario, presentada por el **Dr. Dirk Frederik de Lange** Asesor de la Tesis que desarrollará Usted, con el objeto de obtener el Grado de ***Maestría en Ingeniería Mecánica***. Me es grato comunicarle que en la Sesión de Consejo Técnico Consultivo celebrada el día 20 de octubre del presente año, fue aprobado el Temario propuesto:

TEMARIO:

"RECONOCIMIENTO DE OBJETOS POR VISIÓN ARTIFICIAL USANDO MODELOS SÓLIDOS CAD"

INTRODUCCIÓN.
I. ESTADO DEL ARTE DE VISIÓN ARTIFICIAL.
II. METODOLOGÍA BASADA EN MODELOS PARA IDENTIFICACIÓN DE OBJETOS.
III. IDENTIFICACIÓN DE CARACTERÍSTICAS DE OBJETOS EN 2D.
IV. RECONOCIMIENTO Y UBICACIÓN DE OBJETOS EN 3D.
V. DESARROLLO DEL MÉTODO DE CALIBRACIÓN DEL SISTEMA.
VI. EXPERIMENTOS Y DISCUSIÓN DE RESULTADOS CON OBJETOS 3D.
CONCLUSIONES.
BIBLIOGRAFÍA.
APÉNDICE.

"MODOS ET CUNCTARUM RERUM MENSURAS AUDEBO"

A T E N T A M E N T E

ING. ARMANDO VIRAMONTES ALDANA
DIRECTOR

UNIVERSIDAD AUTONOMA
DE SAN LUIS POTOSI
FACULTAD DE INGENIERIA
DIRECCION

**FACULTAD
DE INGENIERÍA**
Av. Manuel Nava 8
Zona Universitaria • CP 78290
San Luis Potosí, S.L.P.
tel. (444) 826 2330 al39
fax (444) 826 2336
www.uaslp.mx

**bcb

"2010, Bicentenario del Natalicio del Lic. Ponciano Arriaga Leija, Padre de la Constitución de 1857"

Resumen

En los procesos de visión artificial para manufactura automática, ensamble, inspección de productos, medición y reconocimiento en sistemas de vigilancia, es importante detectar e identificar objetos en particular, así como algunos de sus parámetros específicos, tales como posición, orientación, tamaño y configuración.

En este trabajo se desarrolla y se pone en práctica una metodología distinta a los enfoques convencionales, en los cuales frecuentemente se utiliza un scanner láser o luz estructurada con una o dos cámaras para la obtención de datos en 2D, para después, aplicando el principio de triangulación, realizar una reconstrucción en 3D del objeto deseado, determinando de esta forma los parámetros geométricos que permitan reconocer al objeto, así como su posición y orientación. Otros de los enfoques convencionales estudian los contornos de las imágenes adquiridas y determinan las características del objeto, aplicando la teoría de los momentos o la transformada de Hough.

Esta investigación se enfoca en la identificación de objetos en 3D, que forman parte de un conjunto de objetos conocidos con anterioridad y de los cuales se proporciona el modelo CAD. Además de la identificación del objeto, los parámetros específicos a obtener son: la posición, orientación y configuración. La metodología consiste en la generación de una imagen teórica a partir del modelo conocido del objeto que se desea identificar. Entonces se realiza una comparación con la imagen real del objeto tomada con una cámara y finalmente mediante un método de optimización se encontrarán los valores de los parámetros que se busca determinar.

El método se desarrolla principalmente con dos casos de estudio: la detección de características de objetos en una imagen 2D y la identificación de objetos en 3D. En el caso de los objetos en 2D se ilustra el método con dos ejemplos de problemas simples. En el primer ejemplo se busca encontrar la posición y tamaño de un objeto arbitrario (en este caso de forma cuadrada) en una imagen, mientras que en el segundo ejemplo se busca la detección de la posición, inclinación y longitud de una línea recta. Con los resultados se pudo corroborar, que con el uso adecuado de un optimizador se puede identificar los parámetros en la imagen, ofreciendo un alto grado de precisión con una cantidad razonable de esfuerzo computacional. Para el caso de identificación de objetos y sus parámetros relevantes en 3D se muestra que el método puede identificar correctamente cualquier objeto, del cual se tenga su respectivo modelo CAD disponible.

Palabras clave: Visión artificial, reconocimiento de objetos, visualización de modelos CAD, optimización, simplex, calibración, cámara.

Dedicatoria

Dedicado a mis padres con mucho amor y cariño, por todo el respeto y la inmensa admiración que tengo por ellos.

Agradecimientos

A mis padres, por su ejemplo, sus consejos y por su apoyo en los momentos más difíciles.

A mi asesor de tesis por transmitirme sus conocimientos y experiencia, por sus consejos y observaciones, por toda su ayuda y paciencia, que más allá de ser un gran profesional es una excelente persona.

A mi hermano Eliot, que siempre tiene la disposición para explicarme y ayudarme cuando necesito de su ayuda, por ser un gran hermano con Oscar y conmigo.

A mi novia Karen, por su apoyo y comprensión en las dificultades, por toda la ayuda incondicional, por su compañía y por su amor que me motivó durante la realización de este proyecto.

A mi hermano Oscar, mi hermanita Adris, mi tía Adriana, mi abuelo Ignacio, a toda mi familia por su preocupación y por siempre estar al pendiente de mi, por haberme brindado momentos alegres e inolvidables que me ayudaron a continuar con esmero y dedicación.

A mis compañeros quienes durante la maestría fueron un excelente grupo y grandes amigos siempre con la disposición de enseñar y colaborar cuando se les necesitaba, en especial a Guillermo Rodríguez, Raúl Alonso, Víctor Juárez, Verónica Méndez, José Jaramillo, Ernesto Temblador, Eder Govea, Germánico Gonzales y Raúl Chávez.

A todos mis profesores quienes fueron formadores de habilidades y capacidades durante la maestría en el Centro de Investigación y Estudios de Posgrado.

Al Concejo Nacional de Ciencia y Tecnología por haberme becado durante mis estudios.

Índice

INTRODUCCIÓN

Motivación

En muchas ocasiones cuando un producto llega al proceso de manufactura ya se dispone de un dibujo del producto u objeto 3D definido anteriormente en el proceso de diseño. Para llevar una pieza, objeto o material a la producción, uno de los principales problemas usando procesos automatizados de manipulación, es reconocer el producto y la posición en que se está transportando, para de esta manera poderlo sujetar, mover y acomodar de forma rápida sencilla y precisa en el momento que llegue a la siguiente estación. Ejemplo típico reportado es el caso de una ensambladora de piezas de plomería (Wilson). Debido a la variedad de piezas que la empresa produce la producción en masa requiere muchos tipos diferentes de máquinas de montaje, cada uno de los cuales son alimentados manualmente por operarios calificados, la empresa ha optado por automatizar esta operación utilizando un sistema de visión artificial con haz de laser tipo línea. Ahora las piezas de hasta 10 tipos diferentes son puestas al azar y en cualquier posición en una cinta transportadora llegando a los 5 m/min. En la estación de trabajo un sistema automático debe de reconocer las piezas y recogerlos con precisión adecuada para procesos de ensamblaje posteriores. Surge la necesidad de resolver este tipo de problemas para agilizar los procedimientos de cualquier línea de producción con características similares.

Numerosos trabajos se han realizado con el propósito de identificar objetos en imágenes, pues este es un problema presente en aplicaciones de ingeniería, industriales y de la vida cotidiana (Xie, Xie, Cheng, Wong, & Haemmerle, 2008).

En el caso de las aplicaciones industriales, por mencionar algunas, destacan la inspección de productos manufacturados y la automatización de procesos que mediante el reconocimiento y localización de piezas permiten a un robot identificar la localización exacta de diversos objetos depositados en una área de trabajo para así realizar una determinada tarea (Xie, Xie, Cheng, Wong, & Haemmerle, 2008) (Tafur Sotelo & Sobrado Malpartida, 2007).

Las aplicaciones más comunes en la vida cotidiana son los sistemas de monitoreo automático basados en cámara para la detección y reconocimiento de múltiples personas o rostros humanos en una secuencia de video (Tao, Tan, & Lu, 2007), que bien pueden ser utilizados en sistemas de vigilancia(Xie, Xie, Cheng, Wong, & Haemmerle, 2008). Además este tipo de sistemas de monitoreo pueden proveer un proceso de rastreo robusto que es apto para reconocer eventos complejos en interiores o en exteriores que involucren a muchos actores interactuando entre sí(Fusier, y otros, 2007).

Antecedentes

Todos los sistemas de visión por cámara utilizan procedimientos de segmentación de imagen para la detección de características del objeto deseado, las cuales son entidades geométricas como líneas de bordes, regiones homogéneas, instancias abstractas como histogramas de color o atributos de

textura, entonces aprovechando esta información y utilizando una metodología adecuada se puede identificar al objeto y determinar sus dimensiones, posición y orientación en 2D o 3D(Bauckhage, Braunt, & Sagerert, 2004).

Existen una gran cantidad de metodologías convencionales para realizar la identificación de objetos de las cuales destacan los métodos basados en gráficos, en características, en apariencia, en vistas de los modelos en 3D, en filtros como la transformada de Hough y transformaciones como lo es la teoría de los momentos.

Trabajo Propuesto

El tipo de problemas de identificación resultan ser muy difíciles para casos generales, por lo que usualmente se debe de hacer ciertas asunciones sobre los objetos, parámetros o propiedades que se está tratando de detectar. En este trabajo se pretende estudiar las imágenes provenientes de una cámara desde otro punto de vista, en el que se asume desde el inicio tener información sobre los objetos, la configuración, o los parámetros que se está buscando, aprovechando este conocimiento para mejorar la precisión y robustez del reconocimiento. En lugar de reconstruir objetos en el mundo real con base en la imagen, se genera más bien la imagen que se espera encontrar, dada la información que se conoce de la realidad en este caso el modelo CAD. El problema es que falta el conocimiento de ciertos parámetros, que justamente son los que se deben de determinar.

Para esto se propone usar un algoritmo de optimización, como lo ha sido utilizado exitosamente en otras aplicaciones (Maslov & Gertner, 2005). Al resolver el problema de optimización se puede encontrar los parámetros requeridos, considerando que la imagen generada con estos parámetros es lo más parecida a la imagen real de la observación. Este método puede tener éxito bajo la condición de que existe un modelo que puede generar o construir una imagen teórica del objeto y su configuración, dicha imagen debe tener características similares a las imágenes reales obtenidas, para ello se realiza un procedimiento de calibración utilizando la misma metodología pero con parámetros distintos de entrada.

Organización de la Tesis

En el presente documento en el capítulo 1 se presenta el estado del arte de los sistemas de reconocimiento de objetos mediante visión artificial, en seguida en el capítulo 2 se detalla la metodología propuesta y se explica el proceso de optimización que se utilizó. El capítulo 3 permite comprender fácilmente la metodología mediante el caso de estudio para la identificación de características de objetos en 2D donde se detectara desde la posición y tamaño de un objeto arbitrario hasta posición y orientación. Luego, el capítulo 4 presenta el caso de estudio principal para la identificación y reconocimiento de objetos en 3D, detallando en capitulo 5 la calibración del sistema y en el capítulo 6 se presentan los experimentos y resultados obtenidos y su interpretación desde el punto de vista de los experimentos realizados. Después de este capítulo siguen las conclusiones del proyecto donde se discute las dificultades que se presentaron en la realización del algoritmo en su conjunto, la comprobación de la eficiencia del método propuesto y la confiabilidad en los resultados. Algunos detalles relevantes para el proyecto y su seguimiento en el futuro se adjuntan en los apéndices que dan al lector una mejor comprensión de todo el desarrollo.

CAPÍTULO 1

Estado del Arte de Visión Artificial

1.1. Motivación y Definición

En el mundo real los objetos de diferentes geometrías existen en 2D y 3D, por eso es necesario que un sistema de reconocimiento de objetos maneje ambos tipos. El principal reto involucrado en el reconocimiento en 3D es la gran cantidad de información con la que se tiene que lidiar. En muchos casos el reconocimiento en 2D solo tiene que lidiar con translación y rotación del objeto, sin embargo en los sistemas de reconocimiento en 3D hay un número infinito de posibles puntos de vista, además de las variaciones que surgen de articulaciones, oclusión, cambios de iluminación, etc. haciendo difícil la comparación de la información del objeto obtenida con una base de datos (Cyr & Kimia, 2004) (Xie, Xie, Cheng, Wong, & Haemmerle, 2008). Los sistemas de visión pueden clasificarse en sistemas activos y pasivos (Xie, Xie, Cheng, Wong, & Haemmerle, 2008).

1.1.1. Los Sistemas de Visión Activa

Los sistemas de visión activa se refieren al caso en que una cámara y un proyector se utilizan. El proyector se utiliza para proyectar un patrón de luz sobre el objeto, mientras que la cámara adquiere una imagen del objeto con el patrón proyectado. Gracias a esto es posible obtener información en 3D de la superficie del objeto al conocer cuál es el patrón de luz proyectado sobre la superficie y se analiza la forma del patrón de luz en la imagen adquirida. Una desventaja de este enfoque es la necesidad de hardware adicional, que no siempre es deseable. En muchas aplicaciones de la vida real, es importante reducir la producción y gastos de inspección. La introducción de hardware adicional aumenta los costos (Xie, Xie, Cheng, Wong, & Haemmerle, 2008).

1.1.2. Los Sistemas de Visión Pasiva

Los sistemas de visión pasivos se refieren al caso cuando se utilizan una o más cámaras para obtener imágenes del objeto. Estos tipos de sistemas difieren de los sistemas de visión activa, ya que es posible obtener todas las imágenes necesarias del objeto por el uso de una sola cámara. Dichos sistemas utilizan algoritmos de inteligencia artificial y son entrenados tomando fotografías con la misma cámara desde diferentes puntos de vista de los objetos a identificar, algunos otros conforman una base de datos tomando una serie de fotografías de los objetos deseados, manteniendo la cámara fija pero moviendo a los objetos en diferentes posiciones. Este tipo de enfoques a menudo no obtienen la misma cantidad de información del objeto como los sistemas de visión activa esto se debe a la diferencia en hardware utilizado. Algoritmos inteligentes a menudo son requeridos para disminuir esta diferencia (Xie, Xie, Cheng, Wong, & Haemmerle, 2008).

Muchos de estos sistemas de visión tienen una alta tasa de éxito para la identificación de la mayoría de los objetos con alta precisión mediante la utilización de técnicas de sensores de planificación

(sensor planning techniques), en conjunto con un algoritmo complejo de reconocimiento de objetos. Las técnicas de sensores de planificación consisten en: dada información sobre el medio ambiente, así como información sobre la tarea que el sistema de visión debe llevar a cabo, desarrollar estrategias para determinar de forma automática los valores de parámetros del sensor que realiza esta tarea con un cierto grado de satisfacción. Con estas estrategias, los valores de los parámetros del sensor se pueden seleccionar y se pueden cambiar con el fin de desempeñar eficazmente la tarea deseada. Cuando la planificación del sensor de visión se centra en la tarea de detección de características de un objeto ó en la reconstrucción de escenas, los parámetros de dicha tarea, pueden ser la orientación y posición de la cámara. (Tarabanis, Allen, & Tsai, 2002).

1.2. Aplicaciones

Un sistema de identificación de objetos tiene una gran cantidad de aplicaciones las cuales incluyen manufactura automática, inspección de productos, contabilidad y medición, cirugía medica, modelado de objetos físicos en 3D en ingeniería inversa, reconocimiento de rostros humanos en sistemas de vigilancia, sistemas de monitoreo automático, sistemas que evitan colisiones y mapeo de navegación en ambientes de interiores y exteriores (Xie, Xie, Cheng, Wong, & Haemmerle, 2008) (Tao, Tan, & Lu, 2007).

El desarrollo de los sistemas de reconocimiento de objetos en 2D y 3D en robots industriales ha ganado un particular interés en la industria recientemente. Desde que el mundo pasa al siglo de la automatización, tediosas, repetitivas y peligrosas tareas de ensamblado no son favorecidas por los humanos. Un consistente trabajo de calidad es difícil de alcanzar por trabajadores humanos. Para que las manufactureras sobrevivan en un mercado competitivo, necesitan proveer grandes variaciones de tipos de productos. Sin el empleo de sistemas de visión, la orientación de los objetos, sus dimensiones, forma, patrones y otros detalles, los cuales son importantes en tares de ensamblado serian difíciles de determinar por las máquinas. Las formas comunes de obtener esta información incluyen medición del peso y de las dimensiones utilizando una variedad de sensores. Estos métodos son muy limitados y pueden restringir el trabajo a aquellas máquinas con trabajos repetitivos que requieren poca inteligencia o juicio. Teniendo un sistema de visión puede habilitar las capacidades de trabajo de las máquinas. Las aplicaciones comunes de visión de máquina en manufactura yacen en el campo de control de calidad y detección de errores. A pesar de que la visión ha llegado a incursionar en ensamblado robótico, su uso en tareas más complejas está aun en la infancia (Bauckhage, Braunt, & Sagerert, 2004) (Xie, Xie, Cheng, Wong, & Haemmerle, 2008).

1.3. Dificultades de los Sistemas de Reconocimiento por Visión

El reconocimiento de objetos es un tema importante en la visión por computadora en el que varios enfoques han sido desarrollados. A pesar de mucho esfuerzo, el reconocimiento robusto y automático de objetos en las imágenes por computadoras sigue siendo una tarea muy difícil a pesar de la facilidad con que los seres humanos y animales realizan esta tarea (Oberhoff & Kolesnik, 2008) (Bourbakis, Yuan, & Kakumanu, 2008). Numerosas cuestiones técnicas requieren una mayor investigación, especialmente para el reconocimiento de objetos en 3D los principales desafíos no solo son las variaciones en la dirección de la vista y el ángulo, sino además cambios de iluminación, entornos complejos de baja visibilidad o excesiva intensidad, escenas desordenadas, oclusiones y

los objetos que no son perfectamente rígidos pueden aparecer en nuevas configuraciones que no coinciden con ninguna de las geometrías conocidas (Hu & Su, 2008) (Seokho, Caldas, & Dae Young, 2009) (Glover, Rus, & Roy, 2008).

Algunos sistemas de visión no tienen en cuenta la posibilidad de que una imagen no puede tener su correspondiente modelo en la base de datos entonces el mejor resultado siempre se utiliza para determinar la combinación correcta, y por lo tanto es propenso a falsas comparaciones si el objeto en una imagen no está presente en la base de datos (Xie, Xie, Cheng, Wong, & Haemmerle, 2008). Aplicaciones desafiantes como el entendimiento de procesos visuales o aprendizaje por demostraciones son tópicos del desarrollo robótico que no se encuentran aun en la práctica. Esta ausencia de soluciones industriales indica que el reconocimiento de ensamblado visual es complicado y complejo (Bauckhage, Braunt, & Sagerert, 2004).

1.4. Métodos Convencionales

Existen una gran cantidad de metodologías convencionales para realizar la identificación de objetos de las cuales se mencionan las más importantes a continuación.

1.4.1. Métodos Basados en Grafos ó Gráficas

Un "grafo" (graph) $G = (V, E)$ en su forma básica se compone de vértices y ejes. Donde V es el conjunto de vértices también llamados nodos o puntos y E es el conjunto de aristas. En el contexto de visión por computadora, los vértices de los grafos usualmente representan entidades u objetos esperados en una imagen, a veces pueden contener solamente el conjunto de características que proveen una descripción local de la apariencia de este objeto, mientras que las aristas representan conocidas o importantes relaciones entre ellos, en ocasiones se representan como vectores de desplazamiento que conectan a nodos vecinos (Bauckhage, Braunt, & Sagerert, 2004) (Eckes, Triesch, & von der Malsburg, 2006).

Los grafos (graphs) y la relación de grafos (graph matching) son poderosos mecanismos para la representación de conocimiento, patrones de reconocimiento y aprendizaje de máquina. Especialmente en visión por computadora su aplicación es amplia. Los grafos pueden caracterizar relaciones a través de características de imágenes como puntos o regiones pero también pueden representar el conocimiento de objetos simbólico. Por lo tanto la relación con grafos puede completar tareas de reconocimiento en diferentes niveles de abstracción (Bauckhage, Braunt, & Sagerert, 2004).

Los mismos autores presentan un sistema para el monitoreo de ensamblado visual que integra estrategias de segmentación para el reconocimiento de objetos y automáticamente genera y aprende modelos de grafos para reconocer ensambles complejos de los mismos objetos. Acorde con estas consideraciones se selecciona una combinación de características para cada modelo, a cada vértice se le atribuye con su tipo de característica y color del objeto del que pertenece (Bauckhage, Braunt, & Sagerert, 2004).

En otro trabajo experimental se utilizan grafos con una topología simple de cuadricula (grid-like). La distancia entre la dirección de los nodos en x y y son 5 pixeles y cada nodo se conecta a sus vecinos más cercanos. Comparan los modelos de los objetos con diferentes regiones en la imagen,

al encontrar nodos candidatos busca en las vecindades y marca dicha región como ocupada con el respectivo objeto identificado (Eckes, Triesch, & von der Malsburg, 2006).

En un trabajo de investigación se presentó un marco flexible para el reconocimiento de objetos en 3D en el cual se usa un enfoque basado en similitud de aspecto de grafo (similarity-based aspect-graph) utilizando una base de datos de vistas en 2D de los modelos. Este enfoque se evalúa en varios problemas de reconocimiento de objetos, incluyendo el reconocimiento de objetos en 3D, reconocimiento de la postura humana y el reconocimiento de una escena. Características de forma y color se utilizan en diferentes aplicaciones con el marco propuesto y las tres mejores tazas de comparación demuestran la eficacia del método (Hu & Su, 2008).

En otro texto se realizó una metodología para el reconocimiento de objetos en 3D mediante la síntesis de grafos bajo ciertos criterios de vecindad de vistas en 2D. En particular, la metodología utiliza wavelets para reorganizar la forma de la vista percibida en 2D de un objeto. Los grafos para la representación de forma, color y ubicación de cada región del objeto de la imagen se obtiene por la síntesis de las regiones segmentadas que componen el objeto. La representación gráfica del objeto extraído se compara con un conjunto de modelos almacenados en una base de datos. La metodología reconoce objetos existentes en la base de datos y tiene la capacidad de "aprender" patrones nuevos. Para cada modelo almacenado en la base de datos sólo hay seis puntos de vista a partir de los cuales pueden ser generados todos los puntos de vista intermedios mediante una síntesis adecuada (Bourbakis, Yuan, & Kakumanu, 2008).

El punto de partida de este método para comparar la curva de una región del objeto es que la región modelo y la región candidata están representados por las curvas $f(t)$ y $g(t)$. Para que coincida una región, se tiene que encontrar la transformación geométrica que mejor mapea la curva $g(t)$ en $f(t)$. La transformación geométrica básica se compone de traslación (T), rotación (M) y de escala (S) (Bourbakis, Yuan, & Kakumanu, 2008). La transformación entre dos curvas cerradas, $f(t)$ y $g(t)$ se define en la Ecuación 1.1 y la Ecuación 1.2:

$$g(t) = S \cdot R \cdot f(t) + T \tag{1.1}$$

$$\begin{bmatrix} x'(t) \\ y'(t) \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & s_x \\ 0 & 1 & s_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(\theta) & -sen(\theta) & 0 \\ sen(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x(t) \\ y(t) \\ 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1.2}$$

1.4.2. Métodos Basados en Características

El reconocimiento “basado en características” (feature-based recognition), como su nombre indica, utiliza características extraídas a partir del objeto de reconocimiento y que permiten el mapeo de la imagen en un espacio de características de una dimensión menor que la original malla de píxeles. Un número de diferentes tipos de características se pueden utilizar para el reconocimiento, y pueden ser categorizados bajo características de puntos, características de línea, y características de región, los ejemplos más comunes de características son el color, la forma y la textura de los objetos. Una ventaja de los enfoques basados en características es su capacidad para reconocer objetos en la presencia de cambios en la iluminación, traslación, rotación y escala (Xie, Xie, Cheng, Wong, & Haemmerle, 2008) (Westphal & Wärtz, 2009) (Seokho, Caldas, & Dae Young, 2009).

En otro trabajo se propone un método que se basa principalmente en la forma que los objetos reflejan de su segmentación. La forma de cada segmento se describe tanto por sus secciones de contorno significativas como por su región. El método entonces examina múltiples hipótesis de segmentaciones sobrepuestas, usando su forma y color en un intento de encontrar un resultado coherente, como lo es una colección de segmentos que consistentemente votan por un objeto en una sola localización de la imagen (Gorelick & Basri, 2009).

La información disponible en la imagen, en forma de regiones o segmentos coherentes y sus contornos de limite, proveen una poderosa característica que puede ser usada para la detección y segmentación de objetos bajo una variedad de condiciones de luz y en la presencia de deformaciones considerables (Gorelick & Basri, 2009).

En otro trabajo se presenta un sistema de reconocimiento de objetos basado en una combinación de características y patrones. La parte basada en características, es una red de información aportada por cada característica de la decisión en cuestión. Para el procesamiento de objetos arbitrarios, cuenta con gráficos pequeños regulares cuyos nodos son atribuidos con amplitudes de Gabor, denominados gráficos de parquet. El método logra altos índices de reconocimiento en la identificación y posición. A diferencia de muchos otros métodos, también puede hacer frente a variaciones de fondo, varios objetos, y oclusiones parciales (Westphal & Wärtz, 2009). En este trabajo presentan una red neuronal para la preselección de las imágenes modelo y se diseña para que las neuronas selectivas tiendan a responder a particulares vistas de los objetos evaluando las características de referencia.

Las características de Gabor describen la textura local en una imagen. Son las respuestas complejas de un conjunto de filtros de Gabor aplicado a una imagen en una posición de interés. Las respuestas de este complejo conjunto de filtros de Gabor constituyen vectores de números complejos denominados jets los cuales pueden ser comparados con funciones de similitud, las cuales se implementan como el producto escalar normalizado entre las amplitudes de dos vectores (Westphal & Wärtz, 2009).

1.4.3. Métodos Basados en Apariencia

Los métodos “Basado en la apariencia” (appearance-based) se basan en la similitud de la intensidad proyectada de la imagen a lo largo de vistas vecinas. Estos métodos suelen utilizar una forma de análisis de componentes principales (PCA) sobre la imagen, la cual es considerada como un vector de grandes dimensiones (highdimensional vector). Con este análisis se determina la dirección principal de las variaciones así que sólo un subconjunto de la información se mantendrá como se propone en los eigenmodelos. Se ha utilizado el análisis de componentes principales (PCA) en imágenes de objetos conocidos, usando solo el color y la información de la textura.

En concreto, se construye un vector para cada canal de color normalizado (rojo, verde, y azul) de las imágenes generadas de los modelos conocidos, cada imagen se genera con incrementos de rotación con respecto a la vertical (dependiendo del problema) de 7,5 grados. Se analizan los componentes principales en estos vectores y se proyecta cada uno de estos vectores en el eigenespacio, que es significativamente de menor dimensión. Cada objeto por lo tanto tiene su propio valor en el eigenespacio y esto se hace para cada objeto en una colección de objetos conocidos. Para llevar a cabo el reconocimiento PCA, se realiza en los tres canales de color de la imagen de un objeto

desconocido y el resultado se proyecta en el eigenespacio de los objetos conocidos. El objeto se reconoce si la totalidad de sus tres vectores están cerca del valor formado por un objeto.

Los métodos basados en apariencia, pueden ser sensibles a cambios no integrados en el conjunto de entrenamiento como cambios en la iluminación, la rotación de objetos, deformaciones, cambios de vista y la oclusión, afectando los resultados del reconocimiento de objetos (Hu & Su, 2008) (Cyr & Kimia, 2004).

1.4.4. Método de la Teoría de los Momentos

La teoría de los momentos proporciona una alternativa para la representación de formas de objetos. Si tenemos un objeto en una región que viene dado por los puntos donde $f(x,y) > 0$, definimos el momento de orden p,q como en la Ecuación 1.3:

$$m(p,q) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^p y^q f(x,y)dxdy \tag{1.3}$$

$$f(x,y) \leftrightarrow \{m_{p,q}\},\ p,q = 0,1 \dots \infty \tag{1.4}$$

En este sentido, dada una función acotada $f(x,y)$, existe un conjunto de momentos generales con el cual se puede reconstruir una función $f(x,y)$ única, simbólicamente.

Para un imagen de *N x N* es preciso pasar de una integral doble a una doble sumatoria. Los momentos generales discretos de la función $I_0(x,y)$ se calculan por la Ecuación 1.5:

$$m_{p,q} = \sum_{y=0}^{N-1}\sum_{x=0}^{N-1} x^p y^q I_0(x,y) \tag{1.5}$$

Donde:

$$I_0(x,y) = 1 \quad \text{(objeto)}$$
$$I_0(x,y) = 0 \quad \text{(fondo)}$$

Los momentos generales se pueden hacer invariantes a traslaciones en el plano si se hace referencia al centro de gravedad $(\overline{x},\overline{y})$, obteniéndose los llamados momentos centrales en la Ecuación 1.6:

$$u_{pq} = \sum_{y=0}^{N-1}\sum_{x=0}^{N-1} (x-\bar{x})^p (y-\bar{y})^q I_0(x,y) \tag{1.6}$$

Donde:

$$\overline{x} = \frac{m_{10}}{m_{00}} \quad y \quad \overline{y} = \frac{m_{01}}{m_{00}}$$

Puesto que el centro de masas $(\overline{x}, \overline{y})$ ocupa siempre la misma posición relativa respecto a todos los puntos del objeto, los momentos centrales no varían ante traslaciones de los objetos (Tafur Sotelo & Sobrado Malpartida, 2007) (Kang, Lim Kah, & Yao, 2009).

En un proyecto de investigación se implementó un sistema de visión de máquina, el cual a partir de una imagen digitalizada determina las coordenadas de posición y orientación en el plano imagen y posteriormente las traslada a las coordenadas del espacio de trabajo de un robot. Esta etapa permite obtener un modelo (descriptor) que representa las características del objeto, que son relevantes para los objetivos específicos del Sistema de Visión Artificial. Los contornos son la información utilizada en las imágenes y en los modelos de los objetos (Tafur Sotelo & Sobrado Malpartida, 2007).

Para la fase de Reconocimiento también se utiliza una Red Neuronal Backpropagation, que se entrena con un conjunto de vectores característicos de la base de datos que contiene modelos de los objetos en diferentes posiciones y orientaciones (Tafur Sotelo & Sobrado Malpartida, 2007).

En otro proyecto, se realizó un sistema apto para detectar, rastrear y reconocer personas en movimiento sin ninguna intervención humana. El sistema consiste de dos módulos principales primero un modulo detector de personas con base en las regiones con valores de color diferentes de un modelo de fondo estadístico, entonces se analizan los momentos y se desarrolla una imagen binaria para confinar cada persona detectada dentro de un rectángulo de límites mínimos (bounding box), finalmente cada persona en movimiento se representa con un conjunto de atributos de identidad que incluyen una etiqueta, tamaño y centroide (Tao, Tan, & Lu, 2007).

Obtener la forma o la silueta del cuerpo humano es computacionalmente complejo debido a la naturaleza no rígida del cuerpo humano. Además el análisis requiere un robusto rastreo de objetos de colores también necesita de alta precisión en la segmentación del primer plano, cualquier error debido a efectos de sombra o oclusiones degradaran el desarrollo del sistema ampliamente (Tao, Tan, & Lu, 2007).

1.4.5. Métodos Basados en Vistas

El enfoque basado en vistas (view-based) forma parte de la intuición de la percepción humana, en la que una persona memoriza un objeto con varios puntos de vista sin necesidad de un modelo exhaustivo de un objeto en 3D resultando en una reducción significante en la dimensión al comparar imágenes en 2D en lugar de objetos en 3D (Hu & Su, 2008). La caracterización de un objeto en 3D con una densa colección de muestras de puntos de vista independientes lo describe a detalle, sin embargo, este método aumenta de manera significativa el tiempo de cálculo debido al expansivo espacio de búsqueda. Por lo tanto, varios métodos han sido desarrollados para extraer un conjunto mínimo de puntos de vista del objeto (Hu & Su, 2008) (Cyr & Kimia, 2004).

El reconocimiento de objetos en tres dimensiones es mucho más complejo en comparación con los casos en 2D. Esto se debe a que un objeto en 3D puede tener muchas caras diferentes y la apariencia del objeto puede ser muy diferente de un punto de vista a otro. Por otra parte, un sistema de reconocimiento de objetos a menudo necesita identificar más de un objeto. Si la cámara se fija sólo en un lugar para tomar las imágenes, puede no tener suficiente información para identificar a un objeto si se parece a más de un modelo en la base de datos, también puede reportar el nombre

equivocado del modelo cuando existe oclusión, por lo tanto, tomar varias imágenes desde distintos ángulos es un enfoque común de los sistemas de reconocimiento de objetos en 3D (Xie, Xie, Cheng, Wong, & Haemmerle, 2008).

Considere el caso donde hay varios objetos en 3D para ser reconocidos y no hay ninguna restricción sobre el número de posiciones que cada objeto puede tener. Un enfoque para la solución de este problema es obtener y entrenar a varios puntos de vista de cada objeto utilizando la cámara y almacenar los resultados en el sistema. En tiempo de ejecución, si una de las imágenes en la base de datos es similar al objeto a ser reconocido, el sistema informara el nombre del modelo que a esta imagen pertenece (Xie, Xie, Cheng, Wong, & Haemmerle, 2008).

A veces, puede haber dos o más imágenes de los diferentes modelos que se parezcan a los puntos de vista del objeto debido a la similitud de los modelos. En este caso, el sistema tendrá que tomar otra imagen de una nueva posición y repetir el proceso otra vez hasta que sólo quede una imagen en la base de datos que coincida con la imagen capturada durante el tiempo de ejecución (Xie, Xie, Cheng, Wong, & Haemmerle, 2008).

Si sólo hay un modelo en la base de datos, este algoritmo puede funcionar razonablemente rápido. Sin embargo, si hay más modelos a ser reconocidos o si los modelos tienen formas más complejas, habrá muchas más imágenes en la base de datos (Xie, Xie, Cheng, Wong, & Haemmerle, 2008).

Investigadores en el campo de visión desarrollaron una aplicación de visión para el procesamiento de imágenes y reconocimiento de objetos en 2D y 3D utilizando una cámara CCD y los paquetes de software VisionPro y LabVIEW. Se pueden reconocer tres objetos en 2D y cinco o menos objetos en 3D dependiendo del número de puntos de vista permitidos para cada uno de los objetos. Las imágenes de los objetos a ser reconocidos por el sistema tienen que ser preadquiridas, entrenadas y almacenadas por el sistema de visión, para que después puedan ser comparadas con las imágenes tomadas durante el tiempo de ejecución. Una comunicación entre el sistema de visión por computadora y el controlador del robot KUKA permite al sistema de visión tomar imágenes de un objeto desde diferentes posiciones y realizar el proceso de reconocimiento de objetos (Xie, Xie, Cheng, Wong, & Haemmerle, 2008).

El sistema resultó ser apto para reconocer cualquier objeto en 2D en un tiempo de 6 segundos y de 12 a 20 segundos para los objetos en 3D, esta diferencia de tiempo se debe a que el sistema es afectado por el numero de imágenes almacenadas en la base de datos, la complejidad del objeto y la presencia de objetos similares en la base de datos. A pesar de la complejidad del reconocimiento de objetos, tanto la precisión global como el rango de éxito del sistema están cerca del 100% (Xie, Xie, Cheng, Wong, & Haemmerle, 2008).

En otro trabajo se propone un método basado en vistas (view-based) y basado en características (feature-based) para el reconocimiento de objetos en 3D a través de imágenes en 2D de un conjunto de modelos conocidos (principalmente rostros). Emplean una estructura de aspecto gráfico (graph aspect), donde los aspectos gráficos se forman usando una noción de similitud de forma entre diversas vistas. Una vez que la base de datos de los objetos en 3D está organizada como un conjunto de aspectos gráficos para cada objeto, se comparan con los objetos desconocidos y los

resultados se ordenan por similitud. La comparación se realiza basada en curvas y en gráficos y almacenan las tres mejores comparaciones.

Las características de las vistas que se usan para decidir similitud se basan en imágenes estéreo fotométricas, tales como movimiento de cara, relaciones de cara, forma de cara, relaciones de ejes, un histograma de las orientaciones de la superficie y distribución de las características de la superficie (tales como planar, cilíndrica, elíptica, o hiperbólica). El éxito de este enfoque como otros sistemas basados en características (feature-based) es dominado por el desarrollo de las características seleccionadas. Si son seleccionadas características inestables entre las vistas se pueden causar problemas significativos (Cyr & Kimia, 2004).

1.4.6. Método Basado en Modelos

Muchos investigadores han estudiado la identificación y el seguimiento de objetos basado en modelos 3D (model-based), ya que es capaz de representar los objetos reales de mejor manera que los enfoques en 2D, esto debido a que utiliza normalmente un conocimiento a priori de los modelos geométricos de los objetos que describen la forma y la apariencia de estos. En lugar de comparar directamente dos características extraídas de imágenes secuenciales, este enfoque compara una característica extraída en cada imagen con modelos predeterminados. Esta capacidad de los métodos de identificación en 3D ha mejorado la precisión y solidez de la función de correspondencia que desempeña un papel clave para el correcto seguimiento. Las principales preocupaciones en este tipo de reconocimiento de objetos son la extracción de las características más relevantes de una imagen y una comparación de precisión de estas características con un modelo en 3D. Algunos investigadores estudiaron diferentes tipos de autos con los parámetros de un modelo de coche en 3D, con escenas grabadas de tráfico para el seguimiento de vehículos en movimiento de forma automática. También se ha investigado el seguimiento del movimiento humano como el de una mano articulada entre otros (Seokho, Caldas, & Dae Young, 2009).

La identificación de objetos basado en modelos 3D por lo general construye plantillas del modelo renderizado que contienen información de la posición. Se estiman primero los parámetros de posición de las características extraídas de la imagen, y los demás parámetros se obtienen comparando las características de la imagen con los modelos (Seokho, Caldas, & Dae Young, 2009). Para comparar dos imágenes diferentes, usualmente los algoritmos primero colocan la imagen obtenida del objeto sobre la imagen del modelo con los centroides alineados. Si las dos imágenes son exactamente las mismas, la distancia de cada punto entre ambas imágenes se convierte en cero. Por lo tanto, esta distancia puede determinar el grado de similitud entre la imagen escaneada y el modelo (Seokho, Caldas, & Dae Young, 2009).

Los objetivos principales del trabajo de investigación de (Fusier, y otros, 2007), fueron los de proveer un proceso de rastreo que pueda reconocer eventos complejos que involucren a muchos actores interactuando entre sí en una escena grande por medio de una red de cámaras (Fusier, y otros, 2007).

El rastreo de la escena (Scene Tracking) consiste primero en detectar objetos e identificarlos con base en su movimiento, de esta forma se les categoriza en personas, aviones, autos y camiones, enseguida se hace el cálculo de una representación coherente de la escena en 3D y se determina la posición de los objetos móviles en un sistema de coordenadas globales (Fusier, y otros, 2007).

En el siguiente escenario de dicha clasificación se reconocen las categorías individuales de los vehículos, los cuales no pueden ser determinados en el rastreo de la escena, por lo tanto se utilizan modelos de los vehículos en 3D. El modelo se ajusta a un punto en particular de las coordenadas mundiales y se proyecta en los objetos detectados en la imagen de la escena utilizando la normalized cross-correlation, (que es similar en naturaleza a la convolución de dos funciones). Para encontrar el mejor ajuste del modelo se usa el algoritmo de optimización llamado Simplex-Method, asumiendo que el modelo 3D se mueve únicamente en el plano del suelo (Ground-Plane) la búsqueda se restringe en las coordenadas x, y y un angulo θ, la posición inicial del modelo 3D se estima del centroide del objeto proyectado en el plano del suelo y se mantiene su dirección de movimiento, mientras que el ángulo θ se restringe a +-15°. El algoritmo de ajuste del modelo en 3D resultó ser computacionalmente intensivo y no se puede correr en tiempo real. Esto problema se soluciono corriendo este algoritmo aparte y actualizando la clasificación del objeto cuando estuviese disponible (Fusier, y otros, 2007).

En otra investigación se considera que una de las posibles soluciones en monitoreo de tiempo real en zonas de trabajo es la identificación y seguimiento de objetos mediante el sensor de high-frame-rate, el cual es un sistema de imagen óptico que ofrece datos de alta resolución de imágenes en 3D con nubes de densos puntos en tiempo real. El sensor de rango puede determinar no sólo la intensidad local de la escena, sino también el mapa de la distancia completa. En dicho trabajo se presenta una metodología para la rápida identificación y seguimiento de objetos. Utiliza algoritmos para el modelado espacial con los datos adquiridos para posteriormente realizar la comparación de imágenes de una base de datos de los modelos espaciales obtenidos en análisis anteriores (Seokho, Caldas, & Dae Young, 2009).

1.5. Conclusiones del Estado del Arte

Lograr que un sistema de visión artificial realice una ubicación y un reconocimiento robusto de objetos en 3D es una tarea difícil de realizar, principalmente por las dificultades que deben vencer estos sistemas. De entre dichas dificultades destacan las variaciones en el punto de vista de los objetos, cambios de iluminación, entornos complejos, baja visibilidad, excesiva intensidad de luz, oclusiones, objetos no rígidos y falsas comparaciones. Los sistemas de visión disminuyen esta problemática, utilizando algoritmos de inteligencia artificial o combinando más de una metodología con el fin de obtener mejores resultados dependiendo del enfoque de cada investigador. Los métodos de visión artificial más utilizados para el reconocimiento de objetos, presentan ventajas y desventajas como se puede ver en la Tabla 1.1.

La teoría de los momentos permite representar las formas de los objetos utilizando los contornos, la representación se hace de manera precisa y puede trasladarse, rotarse o escalarse, para identificar de manera correcta a los objetos, desafortunadamente cualquier error debido a efectos de sombra u oclusiones degradaran el desarrollo del sistema ampliamente. Además este enfoque aunque es rápido, funciona solamente en 2D por lo que para un sistema de visión en 3D, se tendría que realizar un entrenamiento de los diferentes puntos de vista de los objetos desde la posición en que se encuentre la cámara y adecuar el algoritmo para compensar los efectos de la perspectiva.

Tabla 1.1 Ventajas y desventajas de los sistemas de visión artificial.

Nombre del Método de Visión Artificial	Ventajas	Desventajas
Métodos Basados en Grafos ó Gráficas	Son poderosos mecanismos para la representación de conocimiento, patrones de reconocimiento y aprendizaje de máquina.	Pueden estar limitados a un número máximo de nodos.
	Pueden caracterizar relaciones a través de características como puntos o regiones pero también objetos.	
Métodos Basados en Características	Permiten el mapeo de la imagen en un espacio de características de una dimensión menor que la original malla de píxeles.	Si son seleccionadas características inestables, se pueden causar problemas significativos en el reconocimiento.
	Un número de diferentes tipos de características se pueden utilizar para el reconocimiento (puntos, línea, región, etc.).	
	Tienen la capacidad para reconocer objetos en la presencia de cambios en la iluminación, traslación, rotación y escala.	
Métodos Basados en Apariencia	Presentan niveles altos de éxito en el reconocimiento de objetos.	Pueden ser sensibles a cambios no integrados en el conjunto de entrenamiento como cambios en la iluminación, la rotación de objetos, deformaciones, cambios de vista y la oclusión, afectando los resultados del reconocimiento de objetos
Método de la Teoría de los Momentos	Los momentos generales se pueden hacer invariantes a traslaciones, rotaciones y escalas.	Cualquier error debido a efectos de sombra o oclusiones degradaran el desarrollo del sistema ampliamente.
Métodos Basados en Vistas	Extraen un conjunto mínimo de puntos de vista del objeto.	Si la cámara se fija sólo en un lugar para tomar las imágenes, puede no tener suficiente información para identificar a un objeto si se parece a más de un modelo en la base de datos.
	No tienen la necesidad de usar un modelo exhaustivo de un objeto en 3D.	Pueden reportar el nombre equivocado del modelo cuando existe oclusión.
Métodos Basados en Modelos	Representan los objetos reales de mejor manera que los enfoques en 2D.	Pueden ser computacionalmente intensivos.
	Mejoran la precisión y solidez de la función de correspondencia que desempeña un papel clave para el correcto reconocimiento y la ubicación.	

Los métodos basados en apariencia, comparan la similitud de la intensidad proyectada de una imagen a través de una base de datos de imágenes de puntos de vista vecinos de los diferentes objetos que se desea identificar, lo que significa, que para cada objeto, se debe tener una colección de imágenes las cuales se generan rotando los objetos con pequeños incrementos con respecto a la vertical. La ventaja principal es que presenta niveles altos de éxito en el reconocimiento de objetos. Sin embargo los métodos basados en apariencia, pueden ser sensibles a cambios no integrados en el

conjunto de entrenamiento como cambios en la iluminación, la rotación de objetos, deformaciones, cambios de vista y la oclusión, afectando los resultados del reconocimiento de objetos.

Los métodos basados en vistas, memorizan un objeto con varios puntos de vista, este tipo de enfoques puede confundirse con los métodos basados en apariencia, pero la principal diferencia, es que los primeros extraen un conjunto mínimo de puntos de vista del objeto. Entonces los métodos basados en vistas, no tienen la necesidad de una base de datos exhaustiva de imágenes de vistas vecinas, ni de un modelo CAD de los objetos. La desventaja de este enfoque, es que si la cámara se fija sólo en un lugar para tomar las imágenes, puede no tener suficiente información para identificar a un objeto si el punto de vista se parece a más de un modelo en la base de datos. También puede reportar el nombre equivocado del modelo cuando existe oclusión.

El método basado en características como su nombre lo indica, utilizan características extraídas a partir del objeto de reconocimiento. Por lo tanto presenta diversas ventajas, como permitir el mapeo de la imagen en un espacio de características de una dimensión menor que la original malla de píxeles y reconocer objetos bajo la presencia de cambios en la iluminación, traslación, rotación y escala. Además se puede utilizar un número de diferentes tipos de características para el reconocimiento, como características de puntos, características de línea, y características de región. La desventaja de estos enfoques, es que si son seleccionadas características inestables, se pueden causar problemas significativos en el reconocimiento.

Los métodos basados en grafos representan objetos esperados en una imagen e importantes relaciones entre ellos, a veces pueden contener solamente el conjunto de características que proveen una descripción local de la apariencia de este objeto. Tienen la ventaja de ser un enfoque adecuado para reconocimiento y aprendizaje de máquina. La única desventaja de los grafos que se encontró en la literatura, es que pueden estar limitados a un número máximo de nodos, lo que significa, que no se podrán ingresar grafos con un número mayor a este límite o que si el límite es demasiado grande, entonces se esté desperdiciando memoria cuando los grafos sean muy pequeños. Es común que esta metodología así como la metodología basada en características se combinen con otros enfoques para ampliar las capacidades de los sistemas de visión artificial.

Finalmente el método basado en modelos, realiza la identificación de objetos utilizando sus respectivos modelos CAD por lo general construye plantillas del modelo renderizado de acuerdo a ciertos parámetros y compara las características de la imagen con dichas plantillas. Entre las ventajas que ofrece esta metodología es que permite representar los objetos reales de mejor manera que los enfoques en 2D. Esta capacidad le permite mejorar la precisión y solidez de la función de correspondencia que desempeña un papel clave para un correcto reconocimiento y una buena ubicación de los objetos. La desventaja de este enfoque es el ajuste del modelo en 3D con las imágenes capturadas de los objetos, ya que puede llegar a ser computacionalmente intensivo.

CAPÍTULO 2

Metodología Basada en Modelos para Identificación de Objetos

2.1. Introducción de la Metodología Propuesta

La metodología propuesta en este trabajo surge por la necesidad de identificar objetos mediante un método de visión artificial, la principal característica que se busca es la de analizar la forma de estos objetos desde el punto de vista de su intensidad de luz proyectada en una imagen real. Entonces aquellos pixeles de la imagen real, que se encuentren iluminados con determinado valor serán aquellos que conformen a los objetos buscados. Otra característica importante necesaria para el desarrollo de la investigación es la de poder determinar la ubicación de cualquier forma con una precisión sub-pixel que no se limite a la resolución de las imágenes capturadas. La metodología debe partir de ciertas suposiciones antes de comenzar el procedimiento de identificación. Como todos los sistemas de visión artificial se diseñan para determinadas tareas, es así que se debe tener una base de datos de las figuras que se esperan en las imágenes reales capturadas con una cámara, además de las dimensiones del problema, es decir el área de trabajo de donde se realizaran las tomas de la cámara.

Es una tarea difícil lograr que un sistema de visión artificial reconozca de forma automática un objeto que se encuentra en una imagen previamente segmentada. Una forma de lograr esto es realizando una comparación con otra imagen, es decir una imagen teórica que contenga un objeto que pueda coincidir en posición forma y dimensiones. Esta imagen teórica puede ser controlada de forma automática mediante un proceso de optimización que permita encontrar las variables que provean una mejor coincidencia con la imagen real.

Para resolver la problemática de la intensidad de luz de los objetos, es importante considerar que existe la posibilidad de un valor constante para todos los pixeles de las formas esperadas, por ello es importante dar ya sea a la imagen teórica o a la imagen real una distribución de intensidad de luz, que provea al objeto de la imagen una mayor intensidad en el centro y que disminuya conforme se acerca a los bordes.

La adquisición de características de figuras en dos dimensiones es un importante primer paso para plantear las estrategias que se seguirán en casos más complicados como lo es el reconocimiento y ubicación de objetos en 3D. Utilizando la misma metodología se puede comparar el objeto de una imagen real, con una imagen teórica de un modelo CAD. Los sistemas de visión basados en modelos como se vio en el capitulo anterior no necesitan una gran base de datos con las diferentes vistas de cada objeto, puesto que son capaces de representar los objetos reales en la posición que se les indique. Para encontrar el mejor ajuste del modelo se puede seguir utilizando un algoritmo de optimización como para los casos de identificación de características en 2D.

2.2. Estructura de la Metodología

Ahora se explicará la metodología con ayuda del diagrama de flujo de la Figura 2.1.

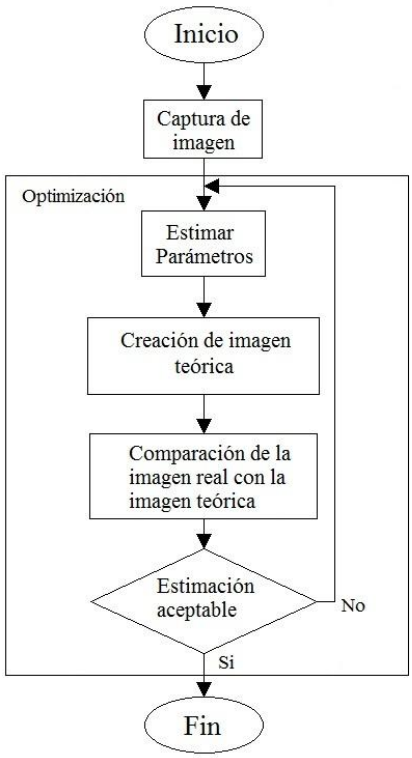


Figura 2.1 Diagrama de flujo de la metodología propuesta.

- **Captura de imagen:** Se obtiene una imagen real por medio de una cámara que está observando el objeto o la forma que se requiere identificar. Esta etapa puede variar dependiendo del procedimiento que se vaya a utilizar o de las condiciones del problema.

- **Proceso de optimización:** Una vez que se cuenta con la imagen real, se pasa a la siguiente etapa donde inicia un proceso de optimización:

    - **Estimar parámetros:** En cada ciclo iterativo del proceso de optimización se hace un estimado de los parámetros de interés en la imagen capturada. En primer instante esto será un estimado arbitrario con base en el rango de valores que se espera u otra información que podría mejorar el valor, reconociendo que el proceso de optimización converge rápidamente cuando el estimado inicial está más cercano al valor final. En subsecuentes ciclos, la nueva estimación está predicha con base en los resultados evaluados de los ciclos anteriores.

    - **Creación de imagen teórica:** Con los valores estimados o "supuestos" de los parámetros a buscar, se creará una imagen teórica, la cual tendrá características similares a la imagen real.

- **Comparar la imagen real con la imagen teórica generada:** Contando con ambas imágenes se determinará la "calidad de la correspondencia" entre las dos de acuerdo al método explicado posteriormente.

- **Estimación aceptable:** Mientras que la comparación no sea satisfactoria o no se ha alcanzado un máximo local, es decir que no se ha alcanzado una tolerancia de comparación previamente establecida (Lakhany & Mausser, 2000), entonces se repite el ciclo con otra estimación a partir de los resultados de las evaluaciones anteriores.

En lo siguiente se detallará mas sobre los pasos más importantes mencionados anteriormente.

2.3. Generador de Imagen Teórica

La imagen teórica generada debe de parecer en sus características a las imágenes reales obtenidas, en términos de la presencia de zonas iluminadas (siluetas) o zonas oscuras (fondo), para que en la comparación de imágenes se obtenga que las zonas de alta (o baja) intensidad correspondan en posición con la imagen teórica y la imagen real.

Para poder generar la imagen se debe de contar con un modelo capaz de construir la imagen teórica como función de los parámetros requeridos, de tal forma que cada combinación de parámetros generará una imagen diferente. Solo cuando la influencia de cada uno de los parámetros a la imagen es independiente y detectable en la imagen resultante, se puede esperar que el optimizador pueda converger a un conjunto único de parámetros que representa adecuadamente los parámetros (físicos) requeridos.

Una característica importante para que funcione el proceso de optimización es que ya sea la imagen teórica o la imagen real, tengan transiciones suaves de intensidad en los bordes de las zonas iluminadas. Esto permite una ampliación del rango en el que existe un "traslape parcial" entre los objetos de la imagen teórica (normalmente zonas de luz) y sus correspondientes zonas de luz en la imagen real capturada, lo cual garantiza que el optimizador detecte el gradiente y la dirección en la que debe de buscar el óptimo. Este principio se hace más claro en los casos de estudio a presentar en capítulo 3.

2.4. Comparación de Imágenes

Una vez generada una imagen teórica se debe determinar su "calidad" comparándolo con la imagen real como fue capturada. Para este fin se hace uso de una metodología similar a lo usado en algebra vectorial. Para comparar la dirección de dos vectores se puede calcular su producto interno, lo que da el valor máximo cuando los dos vectores están orientados de forma paralela, y reduce a valores menores entre más grande sea el ángulo entre los dos vectores. Cuando ambos vectores están normalizados, el resultado de dicho producto es igual a $\cos\theta$, donde θ es el ángulo entre ambos vectores, por lo tanto la escala de comparación será en el rango de -1 a 1, dando 1 únicamente cuando los dos vectores normalizados se encuentren paralelos.

De manera similar, se puede definir la calidad *S* de correspondencia entre dos imágenes R_{ij} y T_{ij} con dimensiones iguales de $n \times m$, por medio del cálculo de su producto interno, definido como:

$$S = R_{ij} \cdot T_{ij} = \sum_{i=1}^{n} \sum_{j=1}^{m} R_{ij} T_{ij} \tag{2.1}$$

Para obtener la calidad *S* en una escala absoluta sirve normalizar cada imagen, por lo que se divide cada imagen $I_{ij}\, I$ entre la norma $|I| = \sqrt{I \cdot I}$ resultando en una imagen normalizada $\hat{I}_{ij}\, I$ de acuerdo a:

$$\hat{I}_{ij} = \frac{I_{ij}}{\sqrt{I_{kl} \cdot I_{kl}}} \tag{2.2}$$

Asumiendo que las imágenes cuentan únicamente con valores positivos en cada pixel, se puede concluir que el resultado del producto interno de la Ecuación 2.1solo puede ser positivo, y tiene como máximo el valor de *S=1* justo cuando las dos imágenes normalizadas son totalmente idénticas.

2.5. Optimizador Downhill Simplex Method

Las consideraciones de minimización multidimensional son, encontrar el mínimo de una función de más de una variable independiente.

El método simplex (the downhill simplex method) es un procedimiento de minimización multidimensional que fue presentado por Nelder y Mead en 1965. El método sólo requiere evaluaciones de funciones sin derivadas. Es un método de búsqueda directa y de uso frecuente en la práctica debido a una convergencia de optimización bastante buena, y un número relativamente pequeño de evaluaciones de función por iteración, que se traduce en un lapso corto de tiempo de duración en comparación con otros métodos de búsqueda directa. En general, el método ha ganado mucha popularidad entre los profesionales, especialmente en el campo de la economía, química, ingeniería y medicina (Lagarias, Reeds, Wright, & Wright, 1998) (Adler & Friedman, 2000) (Press, Teukolsky, Vetterling, & Flannery, 1992) (Kwang-Ok, Chang-Hwan, & Hyun-Kyo, 2008) (Maslov & Gertner, 2005).

Considerando la minimización de un problema con N variables, esto se puede considerar como la búsqueda al punto mínimo en un espacio n-dimensional. Un simplex es la figura geométrica básica dentro de este espacio de *N* dimensiones, y consiste de la figura formada por $N + 1$ puntos (o vértices) y todos sus respectivos segmentos de línea interconectados. Por ejemplo en dos dimensiones, un simplex es un triángulo y en tres dimensiones es un tetraedro, no necesariamente el tetraedro regular. Un simplex es no-degenerativo, es decir, que encierra un volumen finito de *N* dimensiones internas. Si cualquier punto de un simplex no-degenerativo es tomado como el origen, entonces los otros *N* puntos definen las direcciones del conjunto de vectores que forman una base

para describir todo el espacio vectorial de N dimensiones. Para esto es necesario que el conjunto de vectores formados por el conjunto de vértices sean "independientes".

El método simplex debe ser iniciado no sólo con un único punto, sino con $N + 1$ puntos, para definir un simplex inicial. Si se selecciona uno de estos puntos (no importa cual) como el punto de partida inicial P_0, entonces se pueden tomar los demás N puntos como

$$P_i = P_0 + \lambda e_i, \ (i = 1, \dots, N) \tag{2.3}$$

donde e_i son N vectores unitarios y donde λ es una constante, o bien, puede tener diferentes λ_i para cada vector unitario. Los valores de λ_i determinan el volumen del simplex, y se deben de elegir con base en el rango de variación dentro de lo cual se espera encontrar el mínimo. Los puntos P_i con $(i = 0, \dots, N)$ son los $N + 1$ vértices que forman el simplex. En cada punto P_i se evalúa la función a optimizar, dando los valores correspondientes F_i para $(i = 0, \dots, N)$, que se utilizará para determinar la dirección en la cual se va a buscar el mínimo.

Ahora el método simplex toma una serie de pasos, los cuales consisten en solo reemplazar el vértice P_h donde en la cual la función F_h es más grande (el peor punto) por un vértice al lado opuesto al simplex esperando que se encuentra un valor F_n más bajo. Estos pasos se llaman reflexiones, y se construyen de tal forma que se conserva en este paso el volumen del simplex (de ahí que es no-degenerativo). Cuando puede hacerlo, el método expande al simplex en una u otra dirección con pasos más grandes, permitiendo que crezca su volumen, para que en reflexiones posteriores se desplace con pasos más grandes en la dirección en la que se ha expandido el simplex. Por otro parte, cuando se llega a un "valle", el método se contrae ahí en una o todas las direcciones, tirando de sí mismo en torno a su punto más bajo P_l (el mejor punto). El procedimiento se desarrolla a partir de varias iteraciones. Por el movimiento del objeto simplex a través del espacio, este método de optimización normalmente es llamado *amiba,* por la analogía con el simplex que está continuamente cambiando su forma. El método del simplex puede ser utilizado en cualquier problema de minimización multidimensional. Por lo general se puede identificar un "ciclo" o "paso" del algoritmo. El criterio de terminación del algoritmo puede ser una parte delicada, pero en general se considera que la ubicación del mínimo se ha aproximada satisfactoriamente cuando la diferencia en la función objetivo en el mejor y el peor punto, o sea $F_h - F_l$ sea menor en magnitud que alguna tolerancia f_{tol} o que más bien el volumen del simplex se ha reducido a cierta magnitud pequeña. Tenga en cuenta también que cualquiera de los criterios anteriores puede ser engañado por un paso que, por una razón u otra, no llega a ninguna parte. Por lo tanto, es con frecuencia una buena idea reiniciar la rutina de minimización multidimensional en un punto donde se afirma haber encontrado un mínimo. Para este reinicio, debe actualizar las cantidades auxiliares de entrada con los resultados previamente obtenidos. El reinicio nunca debe ser muy caro; dado que el algoritmo ya ha convergido hasta este punto (Kwang-Ok, Chang-Hwan, & Hyun-Kyo, 2008) (Maslov & Gertner, 2005) (Press, Teukolsky, Vetterling, & Flannery, 1992).

Los ingredientes del proceso de sustitución del simplex consisten en cuatro operaciones básicas: la reflexión, la expansión, contracción y multicontracción como lo puede observar en la Figura 2.2. Este proceso consiste de un ciclo iterativo que realiza los siguientes pasos (Kwang-Ok, Chang-Hwan, & Hyun-Kyo, 2008).

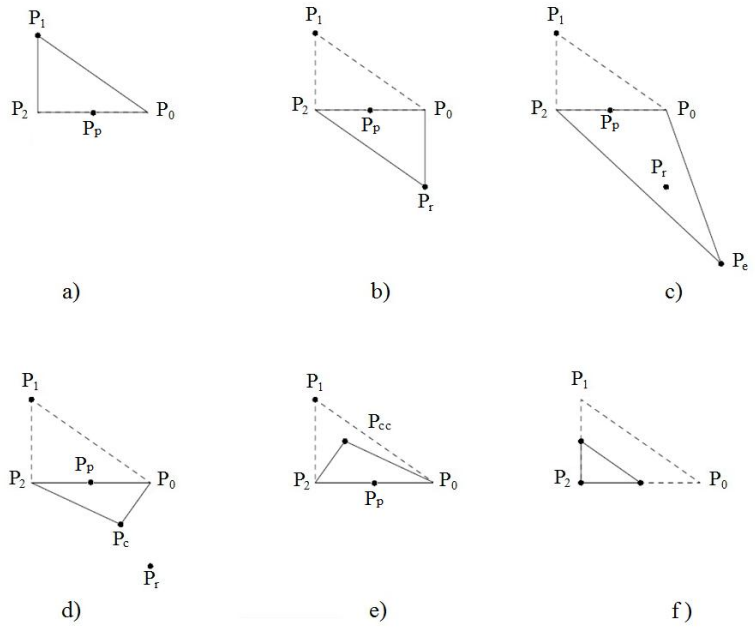


Figura 2.2 Cuatro operaciones básicas del downhill simplex method (tetraedro en el espacio 3D): (a) Simplex original. (b) La reflexión. (c) La expansión. (d) Contracción externa. (e) Contracción interna. (f) Multicontracción.

1. Todos los $(N+1)$ vértices del simplex se clasifican en el orden descendente de sus respectivos valores de función, de modo que $F_h > F_{sh} > \cdots > F_{sl} > F_l$. El punto P_h tiene el más alto valor de F_h y es considerado como el peor, el P_{sh} es el siguiente valor más alto con F_{sh} por lo tanto es el segundo peor y el punto P_l es el valor más bajo considerado como el mejor, de acuerdo con el planteamiento del problema de minimización.

Enseguida, en los pasos 2 a 5 se busca sucesivamente de reemplazar el peor punto P_h, por un punto mejor.

2. Al principio de los procesos de sustitución, P_h se mueve a través de un punto promedio de los otros puntos llamado P_p definido como en la Ecuación 2.4 y entonces se genera un punto de reflexión P_r. El punto de reflexión P_r se calcula como en la Ecuación 2.5:

$$\mathrm{P_p} = \frac{\left(\sum_{i=0}^{N} \mathrm{P_i} - \mathrm{P_h}\right)}{\mathrm{N}} \tag{2.4}$$

$$\mathrm{P_r} = \mathrm{P_p} + \alpha\,(\mathrm{P_h} - \mathrm{P_p}) \tag{2.5}$$

donde el coeficiente de reflexión $\alpha = -1$. La función de valor de F_r se evalúa en el punto de P_r, si F_r se encuentra entre el valor de F_l y F_h significa que el simplex va en la dirección correcta, por lo tanto se sustituye P_h por P_r y se reinicia el proceso de lo contrario continúan los siguientes pasos.

3. Si $F_r < F_l$ (es decir, P_r es mejor que el valor más bajo), significa que la reflexión ha generado un nuevo mínimo, entonces se calcula un punto de expansión P_e que se evalúa como:

$$P_e = P_p + \gamma * (P_r - P_p) \tag{2.6}$$

donde el coeficiente de expansión es $\gamma = 2$, si $F_e < F_l$ nuevamente esto indica que el simplex sigue desplazándose en la dirección correcta, se remplaza P_h por P_e y se reinicia el proceso, pero sí $F_e > F_l$ entonces se tiene una fallida expansión y se remplaza P_h por P_r antes de reiniciar.

4. Si se encuentra que en la reflexión $F_r \geq F_{sh}$ (es decir, F_r es peor que el segundo valor más alto), se realiza una contracción.
    a. Contracción externa. Si $F_{sh} \leq F_r < F_h$ entonces ya se remplazó P_h por P_r y ahora lo que se hará es buscar un mejor punto que no sea P_r ni el viejo P_h:

$$P_c = P_p + \beta * (P_r - P_p) \tag{2.7}$$

    donde el coeficiente de contracción es $\beta = 1/2$. El valor de la función F_c se evalúa en el punto de P_c y si es menor a F_h entonces se remplaza P_h por P_c.

    b. Contracción interna. Si $F_r \geq F_h$ calcula:

$$P_{cc} = P_p + \beta * (P_h - P_p) \tag{2.8}$$

    El valor de la función F_{cc} se evalúa en el punto de P_{cc}.

5. Si $F_c > F_h$ (es decir, F_c es peor que el valor más alto), entonces se tiene una contracción fallida y aunque esto es muy raro de suceder, significa que uno de los puntos está muy lejos de el punto mínimo a diferencia de los otros puntos, la contracción solo causará que dicho punto se aleje más del mínimo y contracciones subsecuentes serán inútiles por lo tanto en estos casos se propone que el simplex se contraiga alrededor del vértice P_l con el valor más bajo de la función, es decir, sus vértices P_i se volverán a calcular de la siguiente manera:

$$P_i = P_l + \delta * (P_i - P_l) \quad \text{para} \quad (i = 0, \dots, N \mid i \neq l) \tag{2.9}$$

donde el coeficiente de contracción $\delta = 1/2$.

A través de estas operaciones, el simplex se puede mejorar y llegar cada vez más cerca a un punto óptimo de forma secuencial. El proceso finaliza cuando el volumen del simplex se hace más

pequeño que un valor predeterminado (Maslov & Gertner, 2005) (Kwang-Ok, Chang-Hwan, & Hyun-Kyo, 2008) (Lagarias, Reeds, Wright, & Wright, 1998). El algoritmo del optimizador, realizado en C++, se puede observar en el apéndice B.

En problemas de optimización con una sola variable independiente el método del simplex no es muy eficiente en términos de número de evaluaciones de función que requiere. El método de Powell es casi con toda seguridad más rápido. Sin embargo, el método simplex con frecuencia puede ser el mejor método a utilizar si el objetivo es trabajar con rapidez para un problema cuya carga computacional es pequeña (Press, Teukolsky, Vetterling, & Flannery, 1992).

Cabe mencionar que al trabajar con problemas de optimización de más de una variable independiente no sucede lo mismo. Como fue el caso en que al minimizar la diferencia entre las características de la silueta de un modelo en 3D de una mano y los parámetros extraídos de imágenes de una mano real. Se utilizaron tres métodos de optimización clásica. El de Levenberg-Marquardt que es un método que utiliza derivadas parciales de la función de error, el Downhill Simplex Method y el Método de Powell que repite la minimización de una dimensión en la dirección de la función de error. Al comparar todas las combinaciones de estos métodos de minimización y funciones de error, el método de Levenberg-Marquardt no se pudo implementar debido al cálculo complicado de la derivada. La optimización con el método de Powell sobre la superficie de la silueta da resultados de minimización muy cercanos a los del método del simplex, pero requiere un costo mayor de evaluaciones de función. Esto es porque el método de Powell usa un procedimiento de minimización de una dimensión (Ouhaddi & Horain, 1999).

Además tres diferentes técnicas de optimización multidimensional fueron comparadas en una serie de casos de prueba de diseño. El Downhill Simplex Method, el método llamado Simulated Annealing que cuenta con una búsqueda al azar en el espacio y en ocasiones acepta pasos de subida en su intento de encontrar el mínimo global y el Differential Evolution que simula la selección natural en la búsqueda del mejor conjunto de variables. Con los resultados de comparación se observa que si bien el método más preciso es el Differential Evolution es también el que realiza la mayor cantidad de evaluaciones de función, los otros dos métodos muestran valores muy parecidos en sus resultados pero es el Downhill Simplex Méthod el que realiza menos evaluaciones de funciones, por lo que para problemas con N variables independientes que requieran de alta velocidad de optimización con resultados aceptables de precisión el Downhill Simplex Method parece ser la mejor opción (Rogalsky, Derksen, & Kocabiyik, 1999).

Ya se ha hablado que el Downhill Simplex Method es rápido para problemas con más de una variables independientes, una de las dificultades encontradas por Nelder y Mead es que el tamaño y la orientación del simplex inicial tuvieron un efecto sobre la velocidad de convergencia. Un problema general que se encuentra en todos los métodos de minimización es el de la convergencia falsa en un punto distinto que el mínimo, es decir que no puede garantizar que se encuentre un mínimo global en un problema de optimización de N dimensiones. Lo cual puede ser corregido al reiniciar el proceso de optimización con un simplex que tenga que someterse a un cambio drástico en su tamaño y forma, sin embargo se debe tener cuidado puesto que refinar el criterio de convergencia a menudo implica evaluaciones innecesarias, y en otros casos más extremos podría ser ineficaz (Rogalsky, Derksen, & Kocabiyik, 1999) (Ouhaddi & Horain, 1999) (Nelder & Mead, 1965).

CAPÍTULO 3

Identificación de Características de Objetos en 2D

Para comprobar la funcionalidad de la metodología se aplica el método a dos casos de estudio simplificados en los cuales se limite al reconocimiento de características directamente en imágenes, evitando la complejidad de la transformación del mundo 3D a una imagen en 2D. En lugar de aplicar el método directamente a imágenes reales, se utiliza el método en estos casos de estudios para detectar parámetros típicos en una imagen creada de manera sencilla y controlable, por lo que se puede saber con anterioridad la respuesta exacta para cada uno de los parámetros a determinar.

En el primer caso de estudio se define la tarea de la identificación de la posición y estimación del tamaño de un objeto arbitrario que se encuentra en la imagen real. En el segundo caso se propone un reto ligeramente más difícil, buscando la determinación de la posición, el ángulo y la longitud de una línea recta. El desarrollo del código fue realizado en la plataforma de Matlab puesto que es fácil de programar y ofrece muchas herramientas para un ambiente en 2D como son imágenes y graficas.

3.1 Caso 1: Reconocimiento de Posición y Tamaño de un Objeto Arbitrario

La determinación de la posición de un objeto arbitrario y su tamaño puede ser un problema bastante frecuente en muchas situaciones. Aunque existen métodos mucho más eficientes y rápidos para determinar los parámetros requeridos para este tipo de problema, como son la determinación del primer momento de área iluminada (posición) y el cálculo de la raíz del área iluminada (tamaño característica del objeto), sirve usar este caso simple para analizar el comportamiento y funcionalidad del método.

Para tener referencia de la posición exacta del centro del objeto, y poder definir el objeto de manera simple, se decide que el objeto a buscar será definido como un cuadrado simple.

Las variables que se están buscando son 3, siendo las coordenadas del centro del objeto arbitrario x_p y y_p, y la dimensión L típica del objeto, por lo que se tiene un espacio de parámetros de 3 dimensiones, y el simplex tendría 4 vértices y tiene la forma tetraédrica.

En primer lugar para garantizar que se conoce el valor exacto de las variables del problema, se crea la imagen real de dimensión de n por n pixeles, cada uno de estos pixeles tendrá una intensidad igual a cero (negro), excepto para los pixeles que comprenden el objeto cuadrado, los cuales tendrán

el valor de uno (blanco), este objeto cuadrado mide por lado 10 pixeles, resultando en la imagen mostrada en la Figura 3.1.

Para la imagen teórica se considera que la figura con la que se va a buscar el objeto arbitrario, no debe de tener orientación, por lo que se debe de definir una forma rotacional simétrica. Por otra parte, como se había comentado anteriormente, es importante que la imagen teórica tenga bordes graduales, que se extienden a larga distancia, para que siempre se garantice que existe un traslape parcial entre la zona con luz del objeto y de la figura con la que se busca. Esto garantiza que la calidad *S* no se reduce a cero y que se obtiene un gradiente entre las evaluaciones de *S* en cada uno de los 4 vértices del simplex.

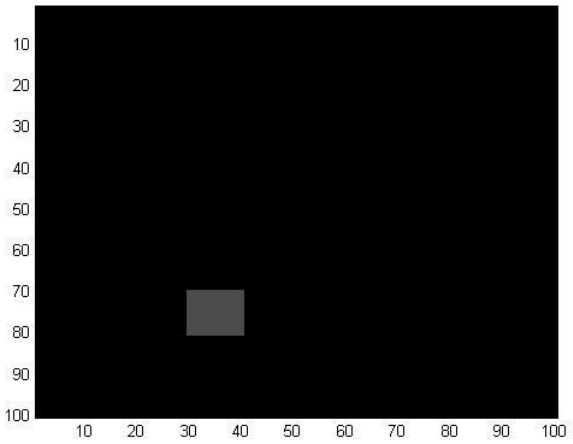


Figura 3.1 Imagen real de un objeto arbitrario procesada.

Una figura que satisface estos requisitos es una distribución Gaussiana, con su centro en las coordenadas x_c y y_c, y un radio R_c por lo que la imagen teórica *I* se define en este caso de acuerdo a las siguientes ecuaciones:

$$\mathrm{X_{ij}} = \begin{bmatrix} 1 \\ 2 \\ \vdots \\ \mathrm{n} \end{bmatrix} * [1, 1, \ldots, 1] \tag{3.1}$$

$$\mathrm{Y} = \mathrm{X^{T}} \tag{3.2}$$

$$\mathrm{R_{ij}} = \sqrt{\left(\mathrm{X_{ij}} - \mathrm{x_c}\right)^2 + \left(\mathrm{Y_{ij}} - \mathrm{y_c}\right)^2} \tag{3.3}$$

$$\mathrm{I_{ij}} = \exp\left(-\mathrm{f}\left(\frac{\mathrm{R_{ij}}}{\mathrm{R_c}}\right)^{\mathrm{E}}\right) \tag{3.4}$$

en donde *X*, *Y* y *R* solo son matrices de apoyo para definir la matriz o la imagen *I*, el coeficiente *E* es un factor de potencia que se aplica en el argumento de la función exponencial, de tal forma que para *E=2* da la distribución propiamente Gaussiana, mientras que se usa el valor de *E=8* para obtener una distribución más "marcada" con flancos más abruptos. Finalmente, el coeficiente f define el valor de intensidad que tenga la distribución Gaussiana a distancia R_c desde el centro. Para nuestro caso se busca que la intensidad sea 0.5, por lo que se define $f = |\log(0.5)|$.

El sistema optimizador determinará en qué punto de traslape existe mayor coincidencia comparando las dos imágenes. En la Figura 3.3 se presenta las dos imágenes juntas para que se pueda apreciar el traslape entre ellos.

Para hacer el proceso iterativo de optimización de manera eficiente, se realiza en múltiples etapas.

Primera etapa:

En la primera etapa se reduce el problema a solo 2 parámetros que se busca optimizar. Solo se buscará la posición aproximada del centro, dejando el tamaño en un valor fijo. En este paso, se define el simplex (un triangulo en este caso de 2 parámetros) de un tamaño relativamente grande, para encontrar rápidamente la zona en que se encuentra el objeto, dando una estimación de su posición suficientemente cercana como se puede ver en la Figura 3.2.

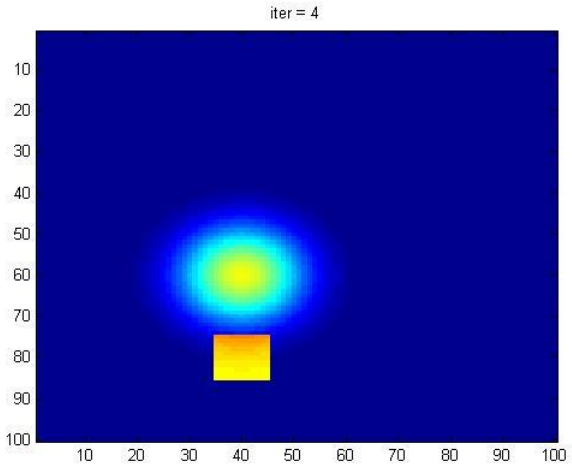
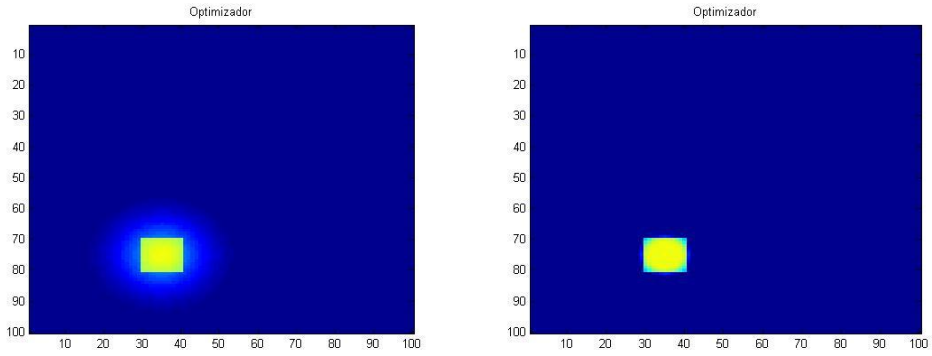


Figura 3.2 El objeto es buscado por el optimizador.

Segunda etapa:

Una vez que la figura del buscador ya se encuentra cercana, se reinicia el proceso, ahora buscando tanto la posición como el tamaño, con un simplex medianamente grande, esperando que todavía sirva iterar con grandes pasos para encontrar un primer estimado del tamaño y una refinación de la posición como se aprecia en la Figura 3.3.



(a) (b)

Figura 3.3 (a) La posición del objeto es localizada (b) La dimensión aproximada del objeto es localizada.

Tercera etapa:

Finalmente, después de un número contado de iteraciones, se asume que ya se ha acercado al punto óptimo, por lo que se reinicia el proceso de optimización, pero ahora con un simplex fino, para obtener la resolución adecuada.

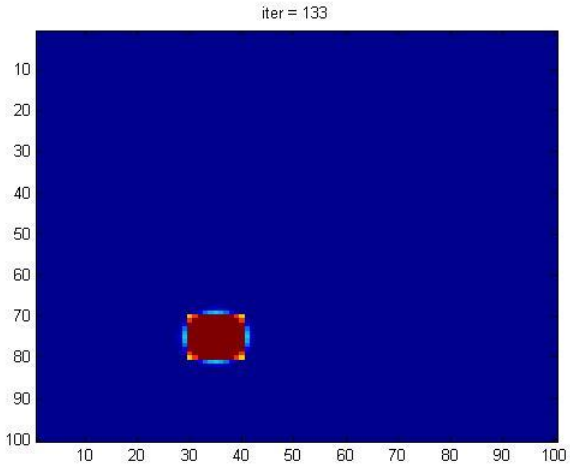


Figura 3.4 El objeto y su dimensión son localizados.

En la Figura 3.4 el óptimo ha sido localizado por el código y como se puede observar ahora la distribución Gaussiana traslapa precisamente al cuadrado, y tiene un radio un poco más grande que el tamaño del lado del cuadrado, pero menor a la diagonal del mismo, de tal forma que sus áreas son comparables en magnitud.

3.1.1 Resultados y Discusión Caso 1

En la identificación demostrada en la Figura 3.5 se observa una gran precisión en la determinación de la posición y tamaño del objeto buscado.

Tabla 3.1 *Resultados para la identificación de un cuadrado.*

Caso de estudio para un cuadrado				
Variables	**1er Etapa**	**2da Etapa**	**3er Etapa**	**Valor Real**
xc inicial	50	74.6004	74.5849	
yc inicial	50	34.4144	34.4916	
Rc inicial	10	10	5.5426	
λ xc	10	0.1	0.1	
λ yc	10	0.1	0.1	
λ Rc	NA	4	0.1	
xc final	74.6004	74.5849	74.4992	74.5
yc final	34.4144	34.4916	34.4993	34.5
Rc final	NA	5.5426	5.5123	5
#iter	33	74	133	0
Duración (s)	5.61	6.20	6.89	
Error	0.4061	0.0580	0.0576	

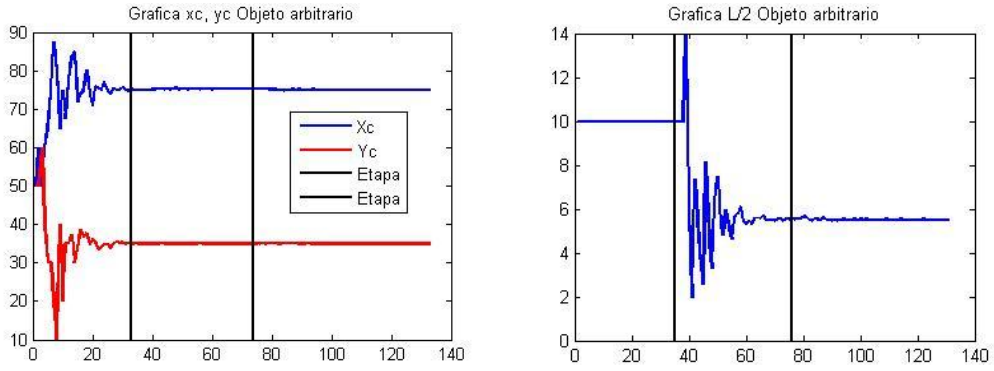
En la Tabla 3.1 se indica que para el caso del cuadrado de lado $L = 10$, los valores de la posición obtenida difieren de la real por 0.0008 en x_c y por 0.0007 en y_c. En cada etapa en la última fila de la tabla se indica el error de la función de comparación de la imagen teórica con la real, dicho valor es igual a 1 cuando las imágenes no se sobreponen y 0 cuando ambas imágenes son iguales y están sobrepuestas una con la otra, claramente se observa que el error de comparación disminuye conforme se avanza en las diferentes etapas. Para el caso del radio R_c el error absoluto es mayor por 0.5123, pero en este caso se debe de mencionar que el tamaño no está bien definido por la distribución Gaussiana, debido a que esta al tener un área circular busca aproximar el área de un cuadrado como se puede ver en la Ecuación 3.5 y por lo tanto el radio real se define como en la Ecuación 3.6 dando como resultado R_c = 5.6419.

$$\pi \cdot R_c{}^2 = L^2 \tag{3.5}$$

$$R_c = L \cdot \sqrt{\frac{1}{\pi}} \tag{3.6}$$

Cabe notar que el optimizador realizó 133 iteraciones en total, con una duración de 18.70s, las cuales se realizaron en tres etapas, donde en la tabla se puede apreciar los resultados y la evolución de las variables en cada una de las etapas. Las tres filas de la tabla con valores de λ para cada parámetro representan las dimensiones iniciales del simplex, como definido anteriormente en ecuación (3), e indican el tamaño de los primeros desplazamientos del simplex, que después cambia por operaciones de contracción y expansión.

En la Figura 3.5 se demuestra la convergencia de las variables como función de iteración, indicando con línea vertical el inicio de una nueva etapa.



(a) (b)

Figura 3.5 (a) Gráfica de la posición x_c y y_c de un objeto arbitrario. (b) Gráfica del radio R_c correspondiente a *L/2* para el cuadrado.

3.2 Caso 2: Reconocimiento de las Características de una Línea

En el segundo caso, se busca determinar un problema con 4 variables: Las variables de interés en este caso son la posición del centro de la línea x_c y y_c, el ángulo de giro θ de la línea con respecto a la horizontal o a la vertical según sea el caso, y la longitud L de la línea. Por lo tanto en esta ocasión las variables del problema requieren un simplex en un espacio de dimensión 4.

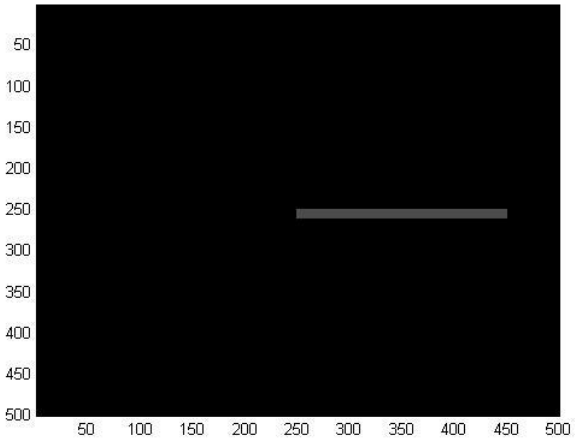


Figura 3.6 Imagen real de una línea procesada**.**

Nuevamente el primer paso será crear la imagen real en una matriz de pixeles con una intensidad de color igual a cero e igual a uno para aquellos pixeles que forman la línea de interés, igual como se demuestra en la Figura 3.6. Para poder tener la certeza de que la longitud, ángulo y posición encontrados por el sistema de minimización son los resultados esperados, se decide trabajar con una línea exactamente horizontal o vertical, aunque esto no implica ninguna limitación de la metodología.

Igual que en el caso anterior, se decide realizar el proceso en cuatro etapas de minimización.

Primera Etapa:

En primer instante se trata de encontrar la posición aproximada de la línea, básicamente reduciendo el problema al caso anterior. La posición global de la línea se busca utilizando la distribución de intensidad Gaussiana, dejando el radio en un valor fijo y suficientemente grande para encontrar la línea, véase la Figura 3.7.

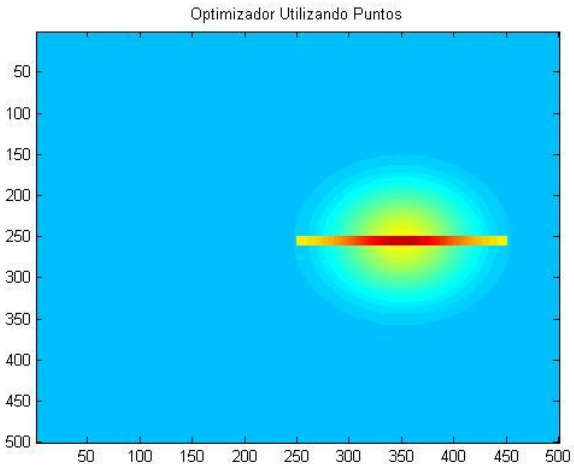


Figura 3.7 Optimizador buscando la línea.

Segunda Etapa:

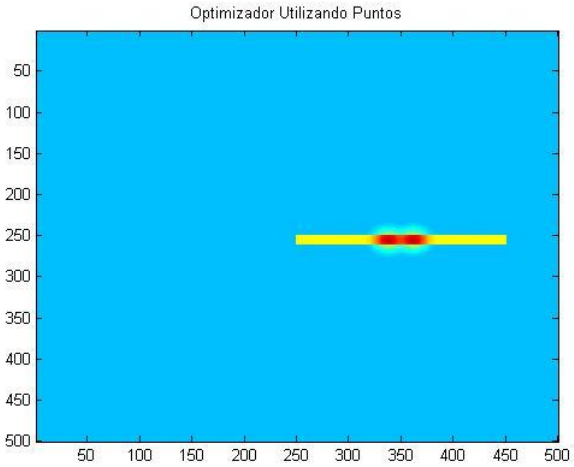


Figura 3.8 Optimizador buscando el ángulo de la línea con 2 puntos.

Una vez localizada la posición en el paso anterior, se reiniciara el sistema de optimización, pero en esta ocasión los valores iniciales de las variables x_c y y_c se toman de la minimización anterior, y se trata de determinar el primer estimado del ángulo de la línea. Para este paso, la imagen teórica estará formada por dos puntos Gaussianos posicionados a cierta distancia fija desde el centro x_c y y_c y ambos en una línea con el ángulo de giro indicado por el parámetro θ como se puede apreciar en

la Figura 3.8. En esta etapa se está primordialmente variando el valor del ángulo dejando la posición del centro prácticamente fija.

Tercera Etapa:

De las etapas anteriores ya se cuenta con una posición de la línea y de un ángulo muy aproximado de la misma, así que en la penúltima etapa se debe de determinar la longitud de la línea, asimismo como mejorar el valor final de los otros tres parámetros. Para esta etapa, la imagen teórica se crea como una serie de puntos Gaussianos, distribuidos sobre una sección de línea definido por los 4 parámetros (x_c, y_c, θ, L), como muestra la Figura 3.9.

Nuevamente se debe reiniciar el optimizador e inicializar las variables con los valores ya conocidos de las etapas anteriores.

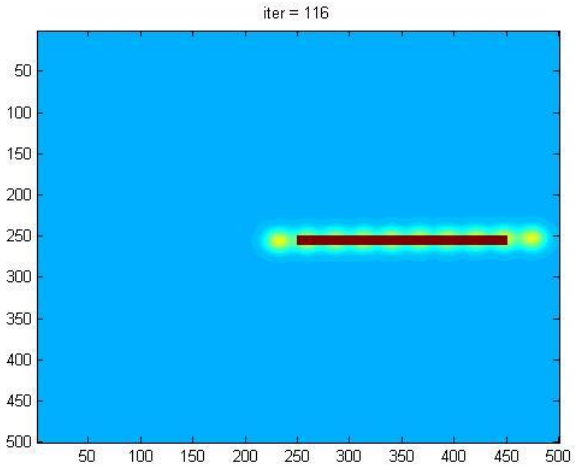


Figura 3.9 Optimizador ubicando la longitud de la línea con varios puntos.

Cuarta Etapa:

Ahora que los cuatro parámetros buscados están muy cerca de su valor exacto, en esta última etapa se vuelve a optimizar a partir de las últimas variables encontradas, pero en esta ocasión el simplex del optimizador se restringe con el objeto de obtener una mayor precisión.

3.2.1 Resultados y Discusión Caso 2

En las primeras dos etapas el proceso de optimización rápidamente converge y determina de manera adecuada las estimaciones de los parámetros requeridos. Sin embargo, la tercera y cuarta etapa presentaban deficiencias en la convergencia de la longitud de la línea, pues en ocasiones el algoritmo no tenía una referencia para ajustarse correctamente.

Se consideró que la determinación de la longitud no tiene un gradiente importante que ayude a converger rápidamente al valor correcto. En particular, el optimizador considera la solución satisfecha cuando la longitud es menor a la longitud de la línea real.

Para apoyar en el proceso de convergencia se decidió agregar dos puntos más a los extremos pero de valores negativos, véase la Figura 3.10. Al momento de comparar estas zonas con intensidad baja (o inclusivo negativa) con la intensidad de la imagen de referencia (la imagen real) el optimizador

buscará que estas zonas correspondan a zonas oscuras, forzando a que la longitud de la línea en la imagen teórica crezca hasta que los puntos negativos se ubiquen fuera de la línea buscada, pero manteniendo los puntos positivos (puntos de luz) traslapando con la línea.

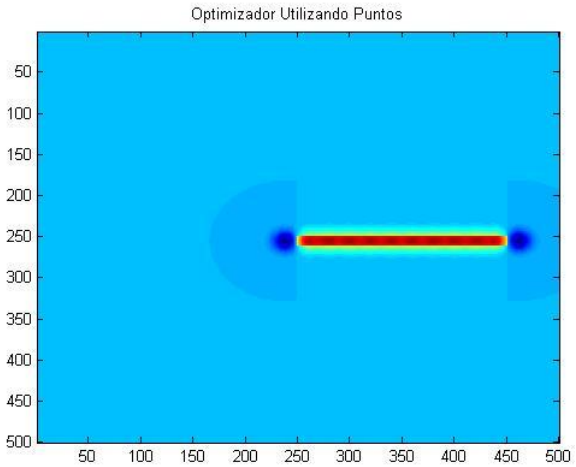


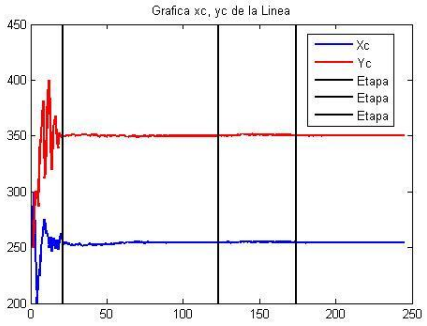
Figura 3.10 Optimizador ubicando la longitud de la línea con varios puntos y dos puntos negativos a los extremos.

Realizando esta modificación al generador de imágenes permitió una buena convergencia del valor de la longitud de la línea, y conjuntamente el optimizador en esta última etapa termina por precisar los valores de la posición y ángulo.

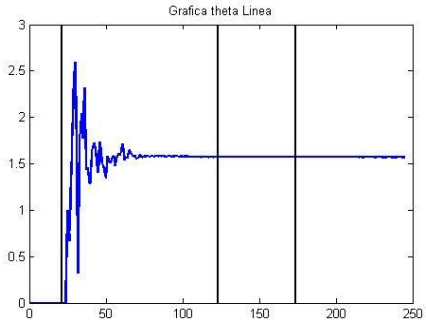
En la Tabla 3.2 se puede ver los resultados obtenidos en un caso particular de convergencia para el caso de la detección de una línea. Como se puede observar, al igual que en el caso 1 se tienen resultados muy cercanos a los valores reales: en el caso de la posición x_c y y_c (eje de simetría), los resultados terminaron con un error absoluto de 0.0149 y 0.0046 respectivamente, el ángulo medido con respecto a la vertical marca un error absoluto de 0.0011 radianes, y finalmente la longitud difiere del valor real por 0.0052. El procedimiento se realizó en 4 etapas y termino en las 245 iteraciones con una duración de 75.38 segundos. La ultima fila de la tabla nuevamente representa el error de comparación entre la imagen real y la imagen teórica, se debe mencionar que el error en este caso es significativo, puesto que varios puntos gaussianos son solo una aproximación de la forma de una línea, entonces el error muetra toda el área de no correspondencia entre los puntos y la línea.

Tabla 3.2 *Resultados para la identificación de una línea.*

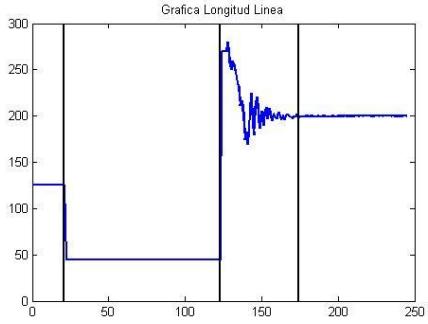
Caso de estudio para una línea					
Variables	**1er Etapa**	**2da Etapa**	**3er Etapa**	**4ta Etapa**	**Real**
xc inicial	250	253.210	254.5123	254.5750	
yc inicial	250	349.6399	349.9900	350.2699	
θ inicial	0	0	1.5721	1.5757	
L inicial	100	25	25	199.4274	
λ *xc*	50	1	0.5	0.5	
λ *yc*	50	1	0.5	0.5	
λ θ	NA	1	0.001	0.001	
λ L	NA	NA	10	0.5	
xc final	253.2104	254.5123	254.5750	254.4851	254.5
yc final	349.6399	349.9900	350.2699	350.5046	350.5
θ final	Fijo	1.5721	1.5757	1.5719	$\pi/2$
L final	Fijo	Fijo	199.4274	199.9948	200
# iter	21	123	174	245	
Duración (s)	7.31	9.26	27.73	31.08	
Error	0.6761	0.6928	0.3403	0.3351	



(a)



(b)



(c)

Figura 3.11 (a) Grafica de la posición xc y yc de una línea. (b) Grafica de $\boldsymbol{\theta}$ de una línea respecto a la vertical. (c) Grafica de la longitud de una línea.

La precisión y el numero de iteraciones depende en gran parte de los valores iniciales que se le dan a las variables en cada etapa, así como de la correcta determinación de las dimensiones del simplex (o sea el tamaño de los pasos a tomar en la dirección de cada variable) al momento de reiniciar el proceso de optimización para la siguiente etapa. En la Figura 3.11 se demuestra las graficas de convergencia de las variables para el caso analizado. Como se puede ver en las etapas 2 y 3, es probable que el número de iteraciones pueda ser reducido en estos pasos intermedios, considerando que en estos pasos solamente satisface obtener una estimación adecuada de los parámetros de ángulo y longitud que se está buscando en los correspondientes pasos. La refinación al valor exacto se obtiene en el paso final. Sin embargo, dentro del objetivo de las pruebas, la optimización de los parámetros para cada paso no se ha considerado necesario, enfocándose a la obtención robusta del valor optimo final.

3.3 Discusión

Una cuestión importante para aplicaciones futuras es la eficiencia y rapidez de la metodología. Para los casos estudiados, existen métodos simples y rápidos para determinar los parámetros requeridos.

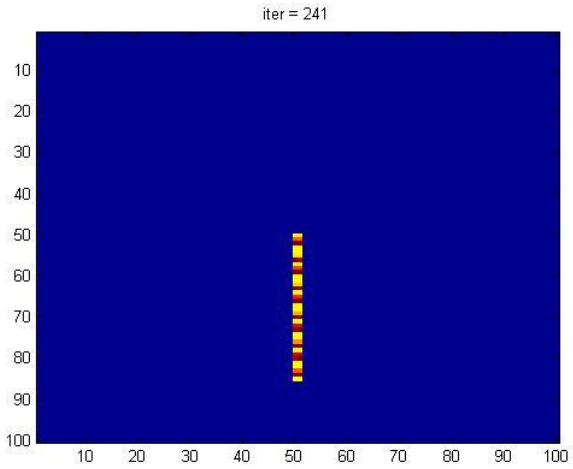


Figura 3.12 Conjunto de puntos de 4 pixeles sobre la línea de interés.

Por otra parte, es importante hacer el generador de imágenes más eficiente. En caso de representar líneas o superficies como conjuntos de puntos Gaussianos, es importante notar que cada punto Gaussiano implica el cálculo de la función exponente para cada pixel, para después sumando todos los pixeles a la imagen final. Una propuesta de mejora es, que en lugar de calcular y agregar toda la distribución Gaussiana para cada punto, se agregua a la imagen toda la energía de cada punto distribuida en un cuadro de 2×2 pixeles. La razón de distribuir la intensidad sobre un cuadro de 4 pixeles tiene su origen en la idea de ubicar el "centro de energía" de la iluminación precisamente en la posición, por lo que se mantiene la información con resolución a nivel sub-pixel. Para esto se define la intensidad en cada uno de los pixeles de forma:

$$I(i,j) = \begin{bmatrix} I_{i,j+1} & I_{i+1,j+1} \\ I_{i,j} & I_{i+1,j} \end{bmatrix} = \begin{bmatrix} f_y \\ 1-f_y \end{bmatrix} [(1-f_x) \quad f_x] \tag{3.7}$$

con *i* y *j* los números naturales justo abajo del valor de *x* y *y* respectivamente, o sea *i=floor(x)* y f_x y f_y el restante fraccional de acuerdo a $f_x = x - i$, o sea $f_x = x - floor(x)$ dando como resultado una matriz cuya suma es justamente igual a 1 y que tiene el centro de intensidad justo en el punto *(x,y)*. El resultado de una imagen generada por este método se puede observar en la Figura 3.12.

El gradiente de intensidad alrededor de cada uno de estos puntos concentrados, similar como se obtuvo con la distribución Gaussiana se puede crear posteriormente aplicando un filtro de convolución a la imagen, para "suavizar" la imagen, causando un efecto de difusión de intensidad a sus alrededores, resultando en la Figura 3.13.

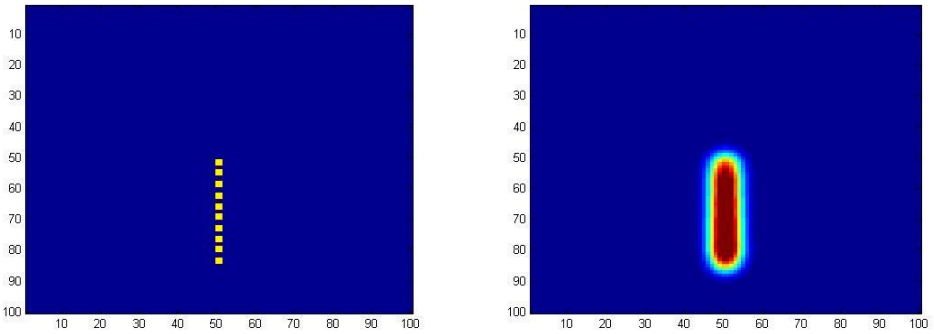


Figura 3.13 (a) Imagen del conjunto de puntos. (b) Imagen una vez aplicado el filtro gaussiano sobre los puntos.

3.4 Conclusiones

La metodología propuesta mostró buenos resultados para los casos en 2D. La precisión que proporciona resultará ser útil para aplicaciones en 3D. El método deja lugar a la experimentación puesto que la selección de un simplex adecuado puede no ser tan obvia y además el método de optimización es considerablemente más rápido cuando se hace la adecuada selección de los valores iniciales. En los ejemplos presentados, la distribución en etapas y los parámetros en cada reinicio, sobretodo el tamaño del simplex no es optimo, pero funciona para los ejemplos de objetos o secciones de línea que se ha tratado de reconocer. Por último el método de optimización ha demostrado ser bueno en términos de robustez de convergencia.

CAPÍTULO 4

Reconocimiento y Ubicación de Objetos en 3D

4.1. Desarrollo del Ambiente Físico, Generación de la Imagen Real

La generación de la imagen real consiste en obtener la silueta del objeto que se desea identificar mediante una cámara digital. Una vez tomada la fotografía, procesar la imagen y obtener la silueta del objeto no es algo trivial, se han realizado numerosas investigaciones con este propósito debido al impacto que tiene en muchas aplicaciones importantes como son el rastreo de objetos, generación de estructuras en 3D a partir de formas en 2D y la clasificación de objetos basados en su forma (Thakoor & Gao, 2008). Uno de los problemas más difíciles en esta área es el de lograr un sistema que de soluciones confiables, eficientes y automáticas que permitan obtener una silueta bien definida como en la Figura 4.1 (Veit, Cao, & Bouthemy, 2006).

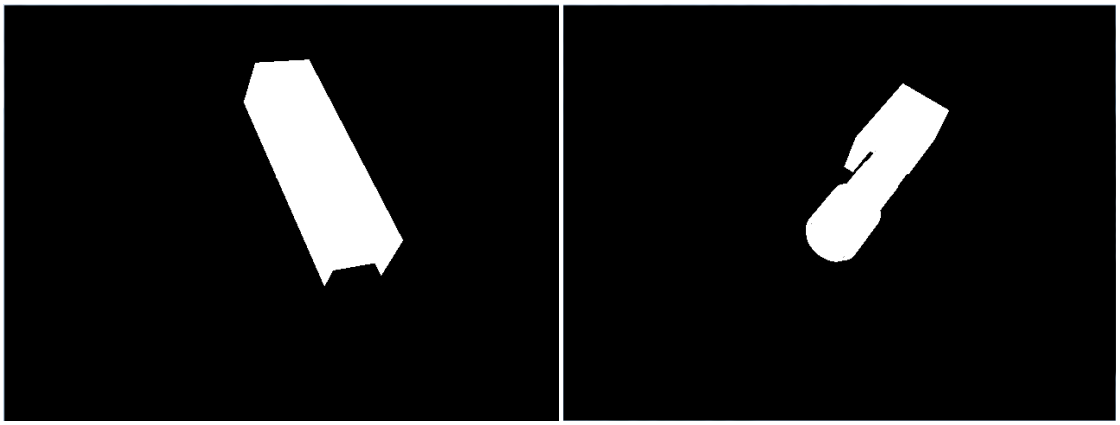


Figura 4.1 Silueta ideal de un objeto real capturado con una cámara.

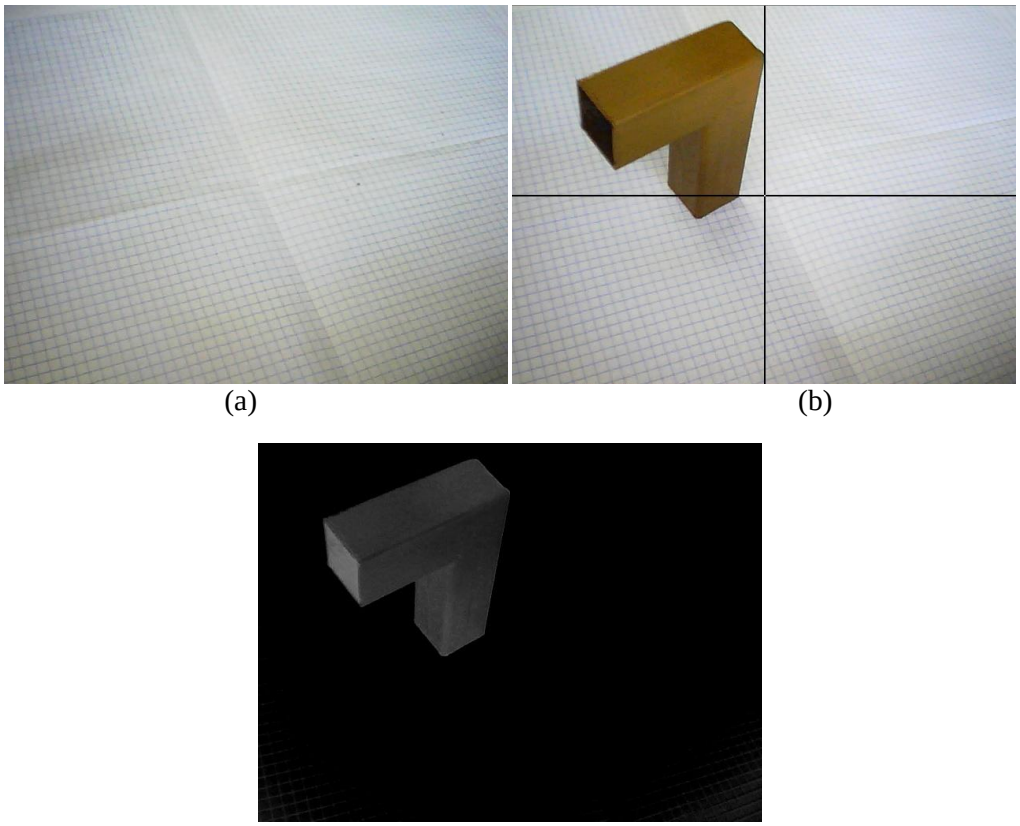
4.1.1. Detección de Movimiento

El objetivo de la técnica de detección de movimiento (diferencia de imágenes) es decidir si una parte determinada de una imagen pertenece al objeto en movimiento (foreground) o al fondo estático de la misma (background) (Veit, Cao, & Bouthemy, 2006). Para ello el registro de una imagen que sirva de referencia es vital para el análisis del resto de las imágenes; esta referencia corresponde con el fondo de la imagen que se pretende estudiar y representa el entorno sin objetos adicionales, la finalidad es establecer un filtro gracias al cual se pueden obtener imágenes que contengan únicamente aquellos objetos que se encuentren en movimiento en la escena de la cámara. Esta detección es posible mediante la resta de píxel por píxel de la imagen que se desea analizar I_{ij} con respecto a la imagen que se ha tomado como referencia F_{ij} y se descartan aquellos pixeles que no han cambiado, es decir, aquellos cuya resta es igual a cero o no mayor de un umbral determinado u (en caso de que hayan pequeñas variaciones en la iluminación), gracias a este proceso, se consigue una imagen O_{ij} como se

explica en la Ecuación 4.1 o de la forma binaria (unos y ceros) en la Ecuación 4.2, donde únicamente se mantienen los píxeles correspondientes con los objetos que hayan incurrido en la escena. Se ilustra un ejemplo de esta técnica en la Figura 4.2 (Chan, 2008) (Hsu, Hsiao, Chang, & Chen, 2009).

$$O_{ij} = \begin{cases} |I_{ij} - F_{ij}|, & |I_{ij} - F_{ij}| > u \\ 0, & |I_{ij} - R_{ij}| \leq u \end{cases} \tag{4.1}$$

$$O_{ij} = \begin{cases} 1, & |I_{ij} - F_{ij}| > u \\ 0, & |I_{ij} - F_{ij}| \leq u \end{cases} \tag{4.2}$$



(a) (b)

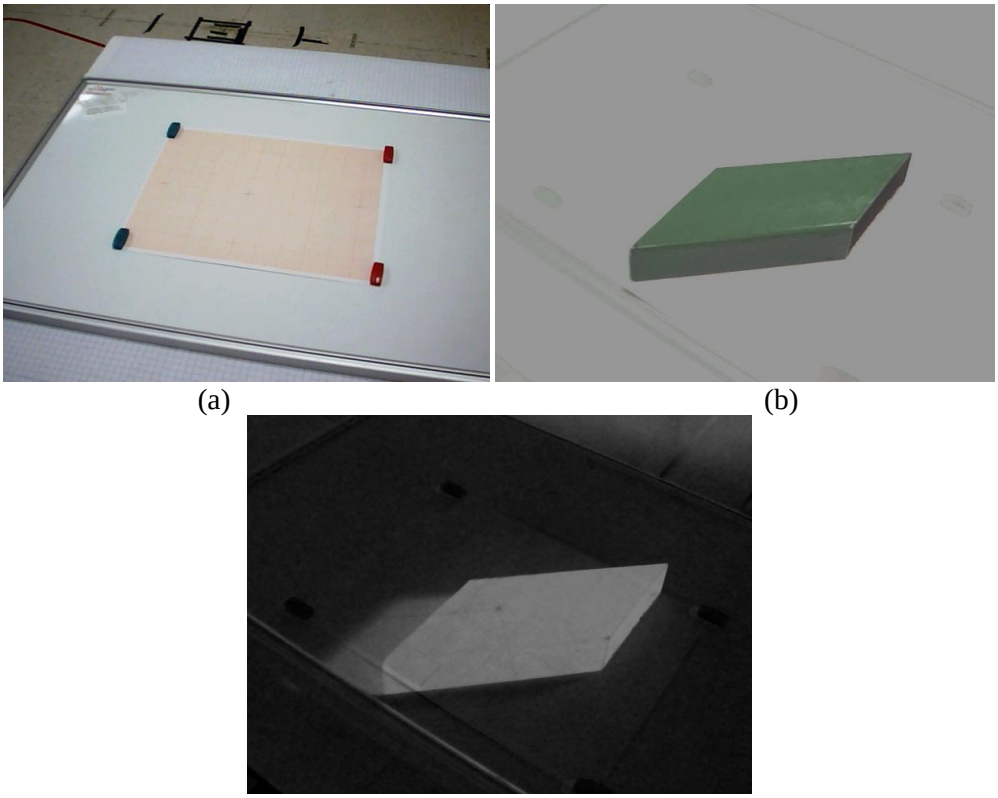
(c)

Figura 4.2 Etapas del método de detección de movimiento. (a) Imagen del fondo estático. (b) Imagen de la escena con el objeto en movimiento. (c) Silueta del Objeto en movimiento.

4.1.2. Ruido

Cuando se dispone de una imagen procesada por determinada técnica de segmentación de imágenes como por ejemplo la detección de movimiento, falla cuando no hay suficiente información para distinguir entre el fondo y el objeto debido a que ambos contienen una distribución de colores similar, cuando la sombra de los objetos es muy contrastante ó cuando hay cambios bruscos de luz por lo tanto se dice que la escena de la cámara es complicada (Kim, Im, & Paik, 2009)(Tao, Tan, & Lu, 2007). Entonces es común que esté presente cierta cantidad de ruido dependiendo de las condiciones de la escena o la cámara, suponga que se tiene una imagen procesada por la técnica

antes mencionada entonces los pixeles que corresponden al objeto en movimiento tienen un valor que puede ir de 0 – 255 para variables de tipo caracter (char) o de 0 – 124 para variables enteras (integer) que corresponderán a los denominados pixeles blancos, entonces a los demás pixeles se les llamara pixeles negros y se les asigna un valor de cero. A simple vista se puede observar si existe ruido en la imagen puesto que quedan manchas remanentes en lugares apartados del objeto en movimiento como en la Figura 4.3.



(a) (b)

(c)

Figura 4.3 (a) Imagen del fondo. (b) Imagen del objeto deseado. (c) Imagen segmentada con la presencia de ruido.

Debido a esto existen diversas técnicas con el propósito de obtener la silueta del objeto en movimiento con un sistema automático y robusto, entre ellas la generación de fondo (background generation) que cuando la escena de la cámara es dinámica ignora los movimientos distractores como pueden ser las ramas de un árbol que se mueven con el viento, las olas del mar, cambios suaves de luz y ruido de la cámara, esto se hace gracias a múltiples imágenes del fondo y algunos algoritmos ó técnicas de inteligencia artificial, sin embargo estas técnicas no pueden manejar exitosamente las sombras de los objetos (Kim, Im, & Paik, 2009).

Para suprimir la sombra de un objeto detectado por movimiento o para identificar su silueta cuando el fondo tiene características similares se utiliza la crominancia (la diferencia entre un color y una referencia de la misma intensidad de color o brillo) de una imagen, se basa en la habilidad de representar colores en términos de un sistema coordenado de 3D que tiene saturación independiente del brillo (IHLS del inglés Improved, Hue, Luminance, Saturation), pero como ya se mencionaba si

no hay suficiente información en la imagen el resultado no será exitoso (Kampel, Wildenauer, Blauensteiner, & Hanbury, 2007).

Para separar los componentes cromáticos del brillo, los canales RGB pueden ser normalizados usando: $L = R + G + B,\ donde\ r = \frac{R}{L}, g = \frac{G}{L}, b = \frac{B}{L},\quad r = g = b = 0\ si\ L = 0$ entonces por definición r , g y b suman 1 en cada pixel. No obstante se sabe que la normalización RGB sufre de un problema de ruido a bajas intensidades que genera componentes cromáticos inestables (Kampel, Wildenauer, Blauensteiner, & Hanbury, 2007).

Lo que normalmente se hace para eliminar el ruido en una imagen segmentada es etiquetar cada conjunto de pixeles blancos asignándoles un numero único, la mayoría de estos conjuntos en la imagen usualmente son eliminados examinando su tamaño ya que frecuentemente son mucho más pequeños que el objeto deseado, o también se puede recurrir a la posición ya que las esquinas de las imágenes son normalmente una fuente de ruido no deseado (Entzinger, 2005) (Tafur Sotelo & Sobrado Malpartida, 2007).

Otro procedimiento muy común para eliminar el ruido en las imágenes procesadas es la operación de erosión la cual específicamente convierte a los pixeles blancos que tengan vecinos negros en pixeles negros. El procedimiento de erosión rompe 'puentes' entre objetos blancos y remueve picos de las imágenes, la contraparte de la erosión es la dilatación, la cual agrega pixeles blancos a las orillas de los objetos y puede ser usado para llenar pequeños huecos en las imágenes (Entzinger, 2005) (Tafur Sotelo & Sobrado Malpartida, 2007).

4.1.3. Ambiente Controlado

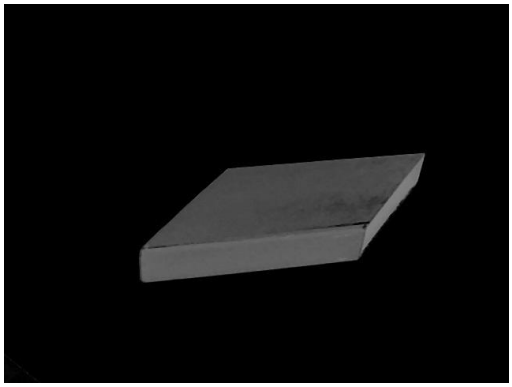


Figura 4.4 Imagen de la silueta de un objeto segmentada por detección de movimiento bajo un ambiente controlado.

Si se trabaja en un ambiente controlado para el proceso de captura de un sistema de reconocimiento visual automático se pueden evitar los problemas debido a las sombras, brillos en los objetos y fondos dinámicos lo que disminuye considerablemente la cantidad de ruido como se puede apreciar en la Figura 4.4. Se deben cuidar tres aspectos importantes, necesarios para obtener una imagen de calidad que no altere las condiciones reales agregando errores considerables al sistema. Uno de ellos es la iluminación, la cual debe ser homogénea sin llegar a saturar o crear brillos en los objetos además el realce que se consigue de la escena simplifica el análisis digital de la imagen, los mejores resultados se obtienen con un halógeno a una distancia adecuada donde la intensidad luminosa sea

menor. Otro son los sistemas ópticos, ya que sufren de distorsiones geométricas que alteran las proporciones reales de las dimensiones de los objetos. Aunque existen múltiples algoritmos para obtener los parámetros de distorsión del lente, es importante para aplicaciones de medición contar con una lente de baja distorsión con el fin de reducir el tiempo de procesamiento y aumentar la exactitud del sistema, además las características de la cámara deben ser seleccionadas de acuerdo con la aplicación, como la capacidad para manipular los parámetros internos, la resolución y velocidad de captura. Finalmente para obtener un alto contraste se debe considerar un fondo adecuado dependiendo del material de los objetos (Zambrano Rey, Parra Rodríguez, Manrique Torres, & Bustacara Medina, 2007) (Collado) (Tafur Sotelo & Sobrado Malpartida, 2007).

4.1.4. Desarrollo de la Generación de la Imagen Real

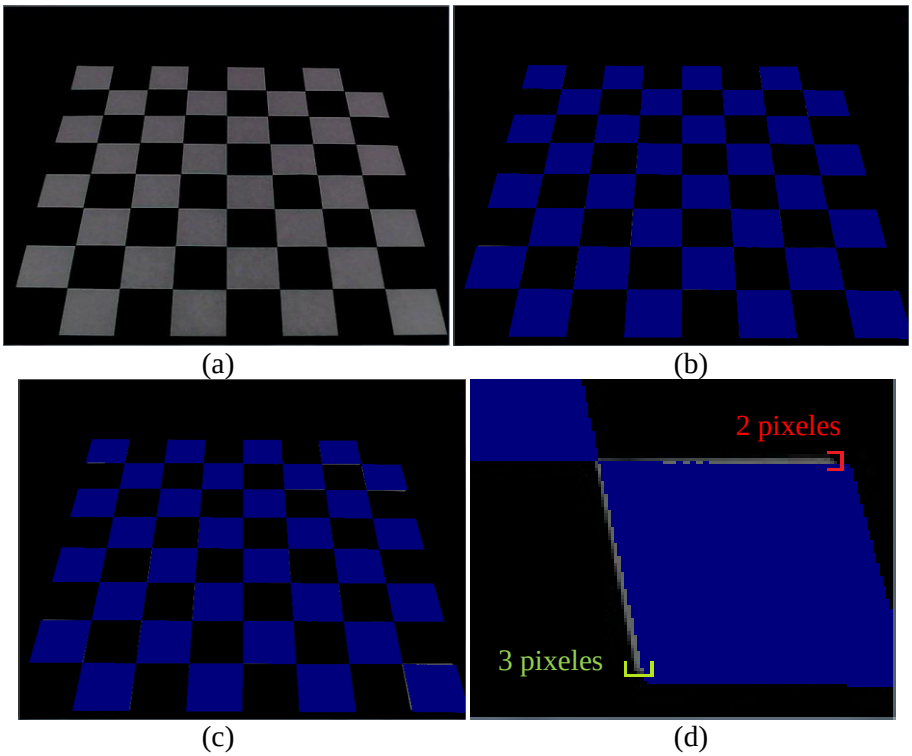


Figura 4.5 (a) Tablero de ajedrez con distorsiones radiales. (b) Tablero de ajedrez sin distorsiones radiales. (c) Tableros sobrepuestos. (d) Acercamiento de las distorsiones radiales.

Se optó por implementar la técnica de detección de movimiento puesto que es un método práctico y robusto, además se usará un ambiente controlado en cuestión de iluminación y fondo que evite la falta de información que como ya se ha visto impide la clara detección de la silueta. De acuerdo a la metodología propuesta pequeñas cantidades de ruido en la imagen no son factores que causen un error significativo a los resultados de la identificación o la localización de los objetos. Además en este proyecto se propone aplicar un filtro de suavizado sobre la imagen real quitando picos y fluctuaciones locales, para que la intensidad de luz sea más suave en los extremos de la silueta del objeto deseado, buscando que se facilite la convergencia de las variables independientes durante el proceso de optimización y evite caer en mínimos locales.

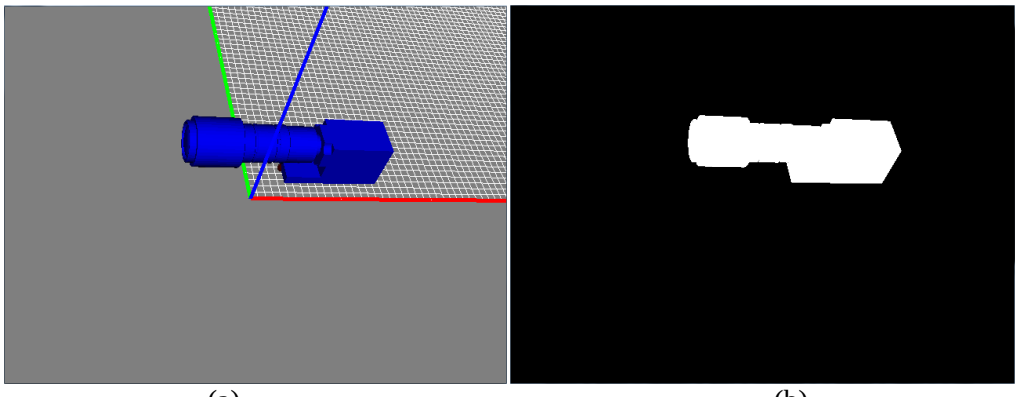
Es importante cuantificar las distorsiones radiales de la cámara puesto que es un factor que se encuentra presente y dada la metodología propuesta es evidente que tendrá repercusiones en cierto grado, en la Figura 4.5 se puede observar un tablero de ajedrez capturado por la cámara que se utilizó en este proyecto, y también un tablero de color azul que fue creado en las librerías de VTK el cual no tiene ninguna distorsión de cámara. Ambos tableros se encuentran en la misma posición y al sobreponerlos se pueden apreciar las distorsiones radiales causadas por la lente de la cámara.

Los pixeles de diferencia entre una imagen y la otra nos pueden dar una idea de la distorsión que tiene la lente, son dos pixeles que de la dimensión de la ventana de 480 corresponden a un 0.41% en y y tres pixeles en x que corresponden a un 0.46% del total de 640.

4.2. Generador de Imagen Teórica

Como se ha mencionado antes, el método propuesto solo funciona con el conocimiento de ciertas características acerca de los objetos que se examinan, es por eso que se contará con el modelo CAD de los objetos que se requiere identificar. Dichos modelos están en formato STL (Stereolithography) que es uno de los formatos más comunes, por lo que es sencillo importar cualquier objeto en 3D de otro formato a este. Las características de los modelos CAD deben especificarse en un archivo de texto llamado "ModelosCAD.txt" de la forma que se explica en el apéndice E. Cada modelo CAD posee características dependiendo de cómo fue elaborado tales como unidades de medida, punto de origen y orientación, es importante mencionar que el algoritmo funciona con modelos elaborados en unidades de milímetros aunque al momento de inicializar un modelo en el archivo de texto todas las características de posición se escriben en centímetros y aquellas características relacionadas con ángulos de rotación se escriben en grados. Las características de posición y orientación ayudan al algoritmo para arrojar un resultado congruente con respecto a dicho modelo.

Para generar la imagen teórica se emplean las librerías de Visual Tool Kit las cuales permiten cargar objetos en 3D de formato STL. Al generar la imagen teórica se deben conocer no solo los modelos CAD sino también las coordenadas mundiales de todo el sistema, como son las coordenadas de origen del plano donde se espera que pase el objeto y las coordenadas de la cámara.

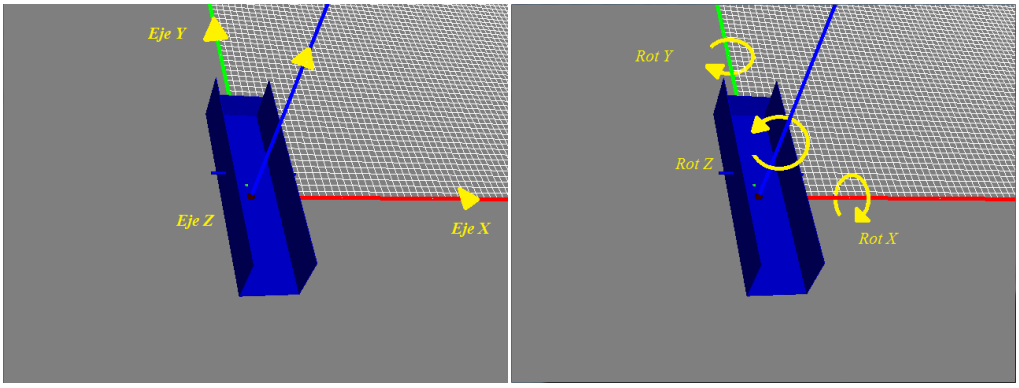


(a) (b)

Figura 4.6 Generación de la imagen teórica. (a) Modelo CAD renderizado ubicado acorde a una suposición. (b) Silueta del modelo CAD.

Lo que se hace es reproducir de manera virtual el ambiente físico que se compone del plano de trabajo y los objetos que se desea identificar. Dichos objetos se renderizan de acuerdo a las suposiciones indicadas por el algoritmo de identificación de objetos y por el optimizador, de tal forma que cada modelo CAD se coloca en cierta posición y orientación con respecto al plano de trabajo, previamente se establece una alta intensidad de iluminación con fondo negro y se selecciona color blanco al modelo CAD, entonces se captura una imagen del ambiente virtual renderizado lo anterior se ilustra en la Figura 4.6.

4.3. Desarrollo del Reconocimiento y Ubicación de Objetos en 3D

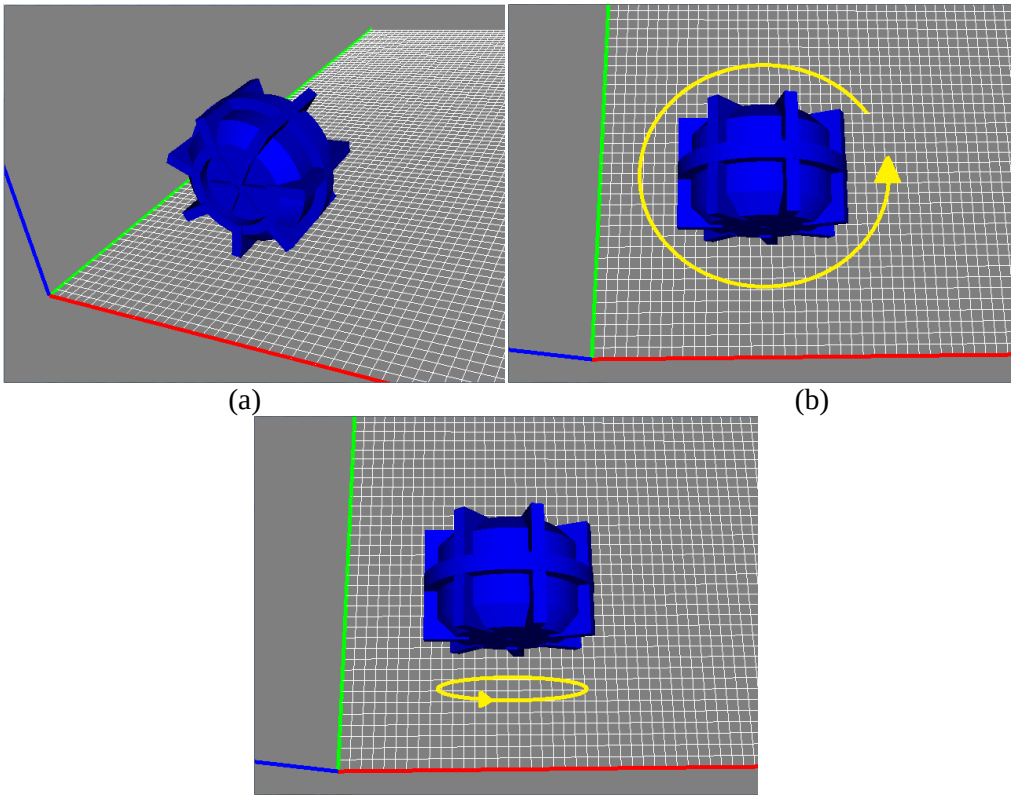


(a) (b)

Figura 4.7 (a) Variables de translación. (b) Variables de rotación.

Se debe tener en cuenta que la identificación de un objeto en 3D, así como su posición y orientación es un problema que consiste de 7 variables independientes que se pueden apreciar en la Figura 4.7 y las cuales son: las coordenadas de posición x,y y z, los ángulos de rotación en el eje x,y y z, y por último el nombre o numero del objeto. Tantas variables independientes de forma simultánea para un proceso de optimización puede tomar demasiado tiempo hasta converger en un valor, además el sistema seguramente caerá en mínimos locales no deseados que comprometen la efectividad y precisión.

Para reducir la cantidad de variables independientes es útil limitar la búsqueda a un plano de trabajo en donde se pueda asegurar que los objetos esperados no sufrirán alteraciones en la posición z, por lo tanto dicha superficie se convierte en un plano de referencia donde se fijan las coordenadas mundiales x,y y z, entonces los objetos se encontraran en estado estable sobre alguna de sus vistas es decir en determinada configuración. Aprovechando esta limitación física las variables independientes que quedan por encontrar son solamente las coordenadas del plano de trabajo x, y, un ángulo de rotación θ en el eje z y la configuración del objeto. Esto funciona para la mayoría de los objetos aunque algunos con forma cilíndrica, o irregular presentan una rotación adicional $\emptyset$ en los ejes x ó y como se ve en la Figura 4.8.

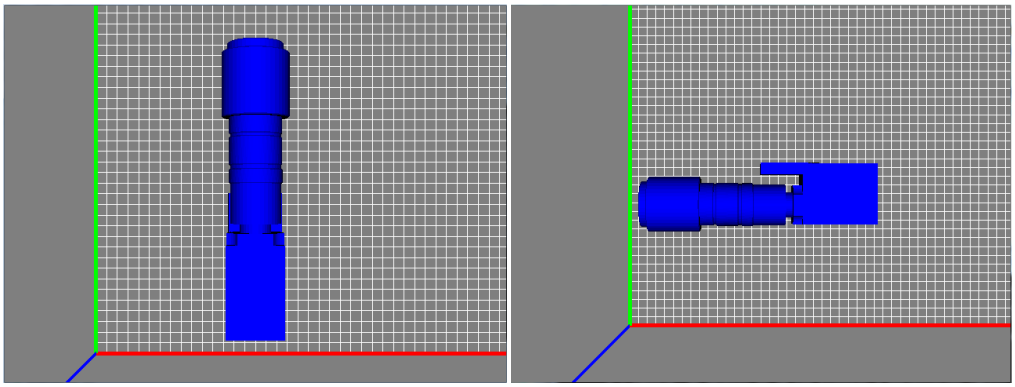


(a) (b)

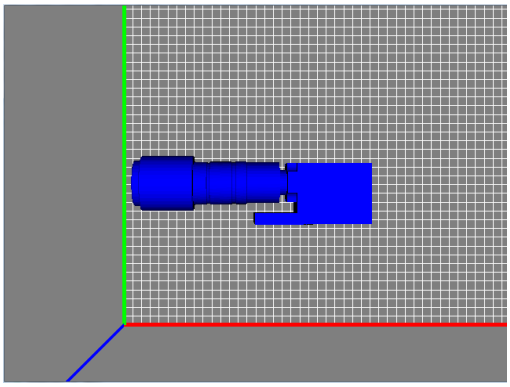
(c)

Figura 4.8 (a) Modelo CAD con forma cilíndrica. (b) Rotación $\boldsymbol{\theta}$ en el eje $\boldsymbol{z}$. (c) Rotación ∅ en otro eje diferente de $\boldsymbol{z}$.

En realidad podríamos obtener tantas vistas de un objeto como quisiéramos. Depende de la posición del objeto con respecto a los planos de proyección, o dicho de otro modo: depende de desde dónde lo miremos. Sin embargo, en la práctica siempre se supone que el objeto está situado de manera que la mayor parte de sus caras (o las más importantes) sean paralelas o perpendiculares a los planos de proyección, porque de esta manera son más sencillas sus proyecciones (o vistas). Partiendo de este supuesto, podemos definir hasta 6 vistas de un objeto. Pero para el caso de las configuraciones si una vista es igual a otra ya no se considera, además existen vistas en las cuales al apoyar el objeto sobre el área de trabajo cambia por la geometría del mismo hasta quedar en equilibrio estable. Un cuerpo en equilibrio estable, si no se le perturba, no sufre aceleración de traslación o de rotación, porque la suma de todas las fuerzas o la suma de todos los momentos que actúan sobre él son cero, es decir cuando el centro de gravedad cae sobre la base de soporte. Entonces para facilitar el proceso de optimización, en el archivo de texto "ModelosCAD.txt" se realiza una configuración de las posibles posiciones de cada objeto sobre un plano. De esta forma se especifica si el modelo CAD debe rotar en algún ángulo o trasladarse sobre algún eje para representar de la manera correcta al objeto físico y los efectos resultantes de apoyar este objeto sobre una de sus vistas. Por lo tanto cada objeto tendrá diferentes configuraciones dependiendo de su geometría, si una configuración es igual a otra por la simetría del objeto, entonces se puede omitir. Un ejemplo de las configuraciones de un objeto se muestra en la Figura 4.9.



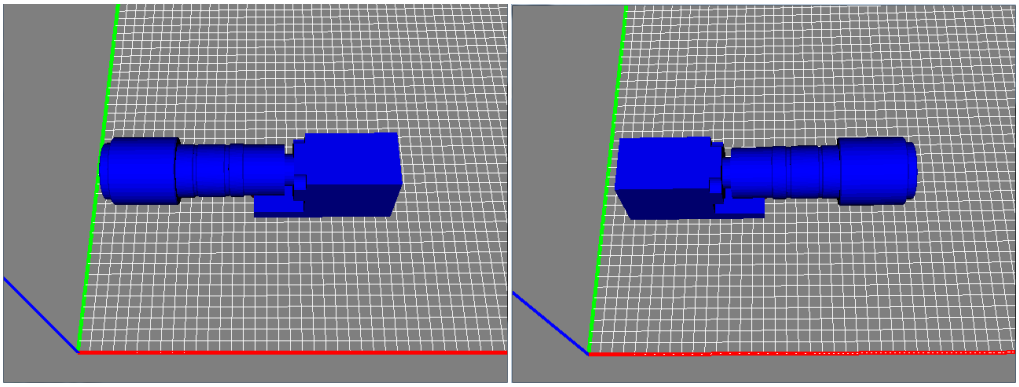
(a) (b)



(c)

Figura 4.9 Diferentes configuraciones de un modelo CAD. (a) Configuración superior. (b) Configuración lateral derecha. (c) Configuración lateral izquierda.

Aun configurando cada objeto el proceso de optimización puede caer en un mínimo local por lo que para evitar esto lo que se hace es iniciar el optimizador con cierta diferencia de grados para cada configuración dependiendo de su geometría, véase la Figura 4.10.



(a) (b)

Figura 4.10 Posiciones de una configuración propensas a caer en mínimo local. (a) Configuración con ángulo inicial $\boldsymbol{\theta}$ de 0°. (b) Configuración con ángulo inicial $\boldsymbol{\theta}$ de 180°.

Una vez que se tienen y se han configurado correctamente los modelos CAD de los objetos que son introducidos previamente por el usuario y que se conoce la posición de la cámara y el plano de

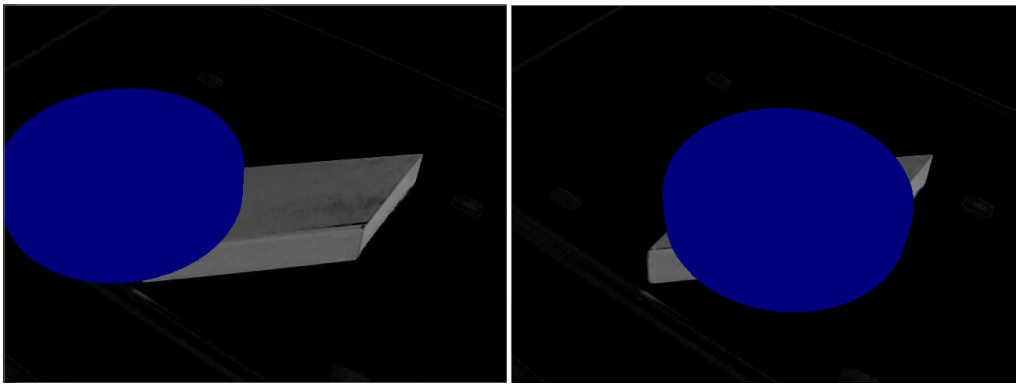
trabajo, entonces se construye el ambiente virtual con la ayuda de las librerías de VTK, que es el que generara las imágenes teóricas necesarias.

Cada imagen teórica generada se normaliza y se compara con la imagen real realizando el producto punto entre las dos obteniendo un factor de correspondencia. Si bien la silueta de la imagen teórica no tiene una distribución gaussiana como se hizo en el capitulo anterior, la diferencia más importante entre una comparación y otra radica principalmente en la forma de las siluetas y en un filtro que suaviza las transiciones entre los pixeles con intensidad de luz y aquellos sin intensidad de luz, es importante mencionar que para agilizar el procedimiento este filtro solo se aplicará a la imagen real. Para visualizar la comparación de las imágenes en cada iteración se usaron las librerías de OpenCV.

La estructura del sistema consiste de una primera etapa de optimización para obtener una posición aproximada del objeto y una segunda etapa que consiste de un algoritmo para el reconocimiento del objeto y su ubicación.

Primera Etapa:

En la primera etapa se utiliza una figura cilíndrica o circular para formar una imagen teórica. Esta figura tiene una dimensión que concuerda con el tamaño de la escena y los objetos a identificar, utilizando esta metodología la figura se traslada sobre el plano de trabajo en x y y hasta que encuentra una estimación de la posición del objeto en 3D como se puede ver en la Figura 4.11, en seguida comienza el proceso de identificación y localización de objetos en 3D.

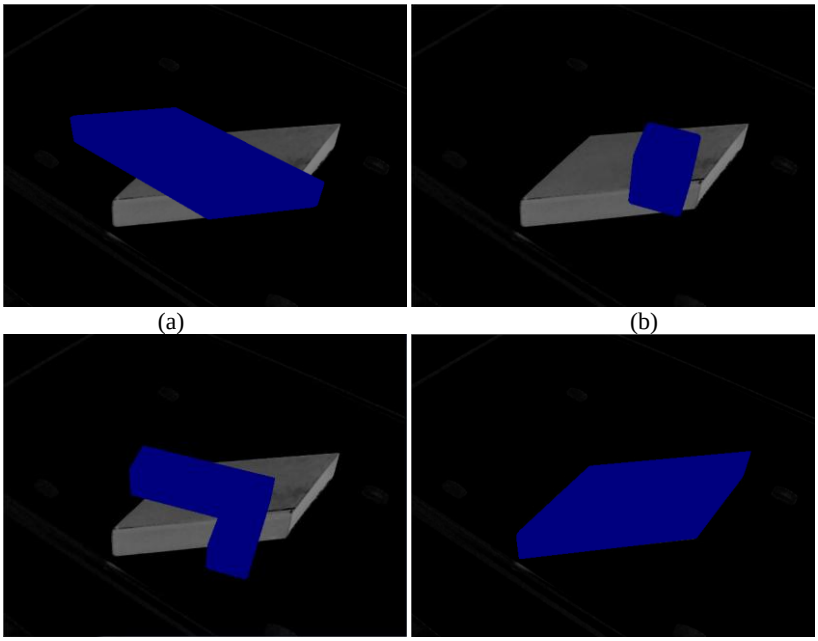


(a) (b)

Figura 4.11 (a) Modelo CAD cilíndrico utilizado para identificar la posición del objeto deseado. (b) Posición del objeto de la imagen real ubicado por el cilindro.

Segunda Etapa (Algoritmo para el Reconocimiento y Ubicación):

Para entender el proceso de reconocimiento de objetos está el módulo de la Figura 4.13 en donde las variables desconocidas son el objeto deseado, su configuración, posición y orientación. Básicamente lo que se hace es un proceso de optimización para todas las configuraciones existentes, y bajo la suposición de que se tiene la posición aproximada, de los objetos. Se tienen H modelos CAD configurados, de los cuales cada uno presenta un numero G de configuraciones, por ultimo cada configuración tiene F numero de ángulos de rotación inicial.



(a) (b)

(c) (d)

Figura 4.12 (a) Imagen teórica del primer modelo CAD comparado con el objeto de la imagen real. (b) Imagen teórica del segundo modelo CAD comparado con el objeto de la imagen real. (c) Imagen teórica del tercer modelo CAD comparado con el objeto de la imagen real. (d) Objeto ubicado, utilizando la mejor comparación de entre todas las configuraciones de los modelos CAD.

El algoritmo que se muestra en el apéndice F, consiste principalmente de 3 niveles de búsqueda, por lo tanto se tienen tres ciclos anidados en los cuales el primero recorre los H modelos CAD como se muestra en la Figura 4.12, el segundo ciclo cubre las G configuraciones del modelo que se está utilizando y el último ciclo varía dependiendo de la geometría de la configuración actual donde el ángulo θ inicial antes de optimizar puede ser cada 180° hasta llegar a 360 o puede ir cada 90°, pero es dependiendo de el valor que se establezca previamente en la configuración del modelo, entonces se inicia el proceso de optimización y se guarda el error resultante, una vez que se ha terminado de optimizar todas las posibles configuraciones de todos los objetos, se encuentra el menor error, el cual permitirá conocer el objeto de la imagen real y su localización con respecto a las coordenadas mundiales.

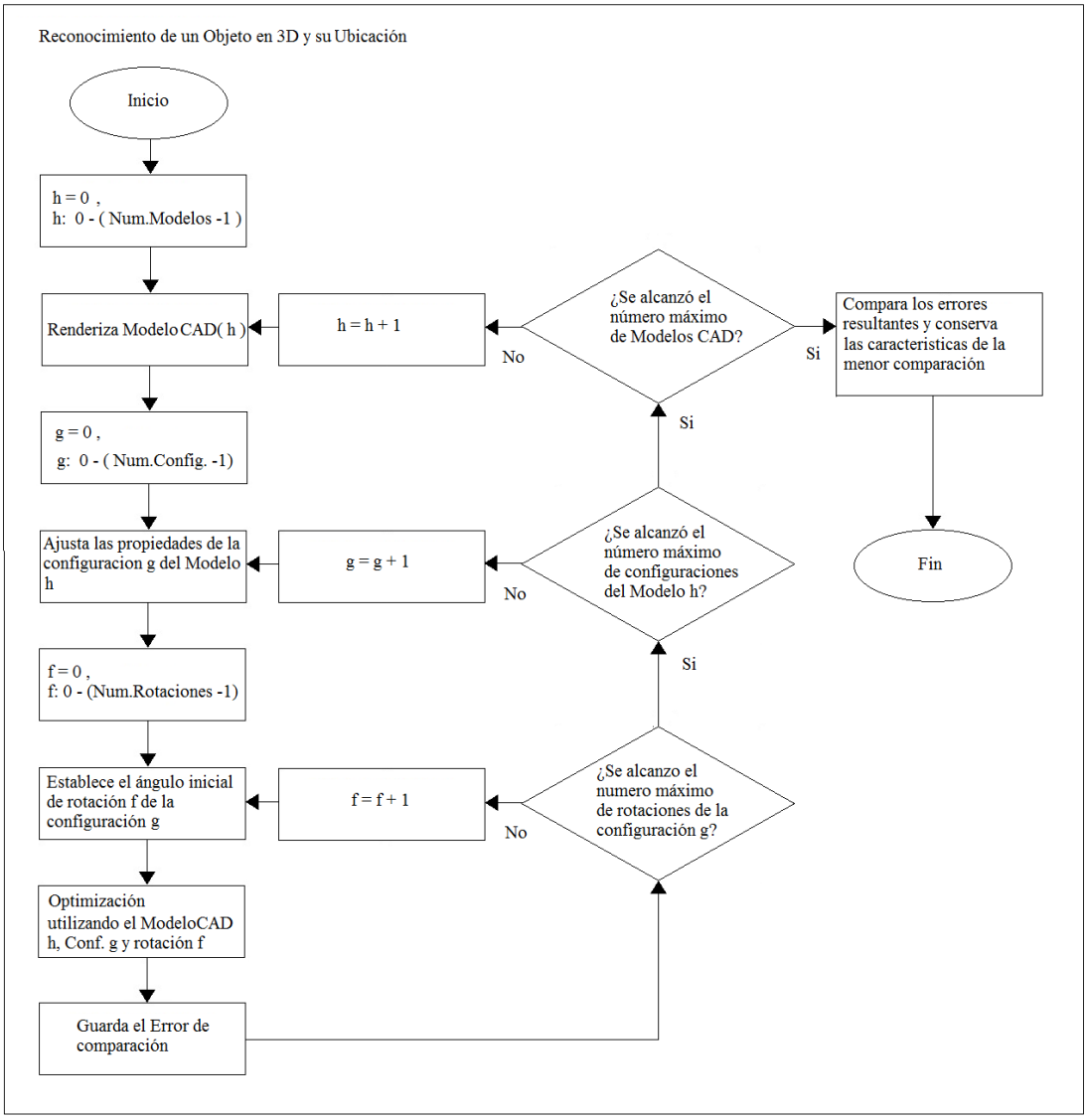


Figura 4.13 Módulo del reconocimiento de un objeto en 3D y su ubicación.

CAPÍTULO 5

Desarrollo del Método de Calibración del Sistema

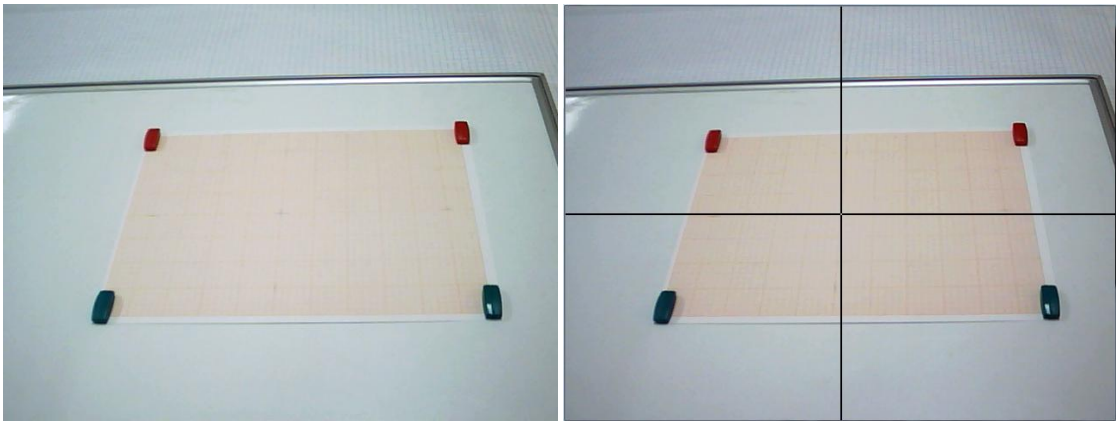
El procedimiento de calibración consiste en encontrar los valores exactos de las variables involucradas con la cámara, como son: su posición exacta, su orientación y el zoom. Asimismo, la calibración también define el sistema de coordenadas de referencia en el que se ubica el plano de referencia sobre el cual se espera que se encuentren los objetos. La calibración es muy importante puesto que nos permite crear el ambiente virtual del sistema, de la forma más cercana a la realidad, con ello se obtienen resultados más precisos del reconocimiento y la ubicación de los objetos en 3D. Para realizar el procedimiento de calibración se utiliza la misma metodología presentada hasta el momento, en la cual, durante un proceso de optimización, se compara una imagen teórica del ambiente virtual, con una imagen real de un objeto de calibración que se encuentra en la escena. La diferencia importante entre el proceso de reconocimiento de objetos discutido en el capitulo anterior y el proceso de la calibración es que en el primero, se suponen conocidos los parámetros del sistema, y se busca encontrar características del objeto, mientras que en el proceso de calibración se suponen conocidos los datos relevantes del objeto del que se toma la imagen real, mientras que se busca encontrar los parámetros del sistema. Para iniciar el proceso de calibración se supone que ya se tiene estimaciones iniciales basada en mediciones aproximadas de la posición y orientación de la cámara, para poder generar la primera imagen teórica con suficiente grado de comparación.

5.1. Desarrollo del Ambiente Físico

A grandes rasgos para desarrollar el ambiente físico de calibración lo que se necesita es establecer un origen de coordenadas mundiales, un punto de centro focal de la cámara, una orientación de los ejes de coordenadas mundiales y además objetos para calibración que deben ser colocados en determinada posición y orientación, finalmente se genera una imagen real para que el algoritmo identifique las variables independientes del problema. Todas estas características en el algoritmo están diseñadas para que el usuario las ingrese de forma práctica y sencilla en un archivo de texto llamado “ConfCalibracion.txt”, como se muestra en el apéndice H.

Lo primero que se hace es seleccionar un plano de trabajo, que es donde se espera que sean colocados los objetos para el reconocimiento y la ubicación, entonces se decide de forma arbitraria el origen de las coordenadas mundiales $(x0, y0, z0)$, es decir el punto $(0,0,0)$ dependiendo de las condiciones del problema. Después se debe contar con una cámara que observe dicho plano de trabajo. Esta cámara preferentemente debe tener una Interfaz de conexión a la computadora y utilizando las librerías de OpenCV, se despliega en pantalla lo que la cámara observa. Esto permite que la cámara se pueda dirigir de acuerdo a las necesidades del problema que se tenga, es decir las dimensiones de los objetos que se desea identificar y el plano de trabajo . El programa no solo

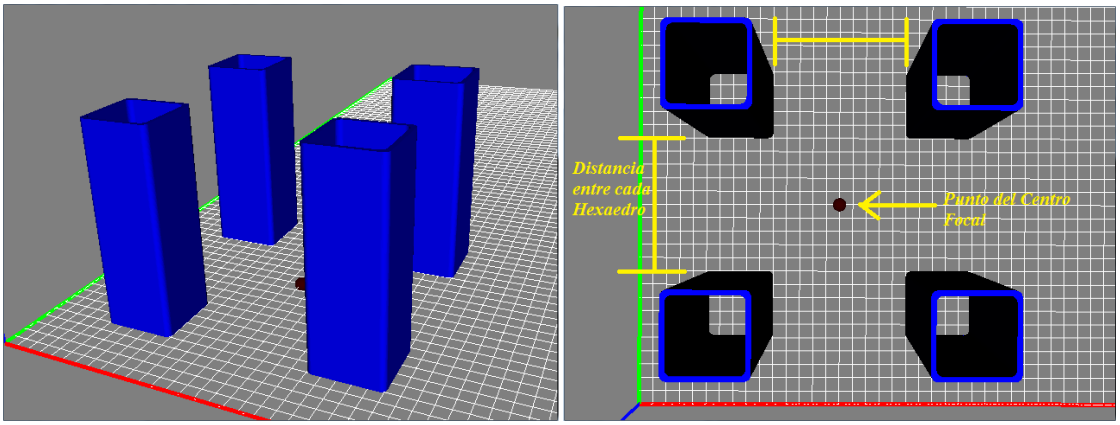
permite visualizar en pantalla lo que la cámara observa sino que también muestra una marca en el centro de la pantalla. El punto sobre el plano de trabajo que corresponde al centro de la pantalla es conocido como el punto del centro focal (xf, yf, zf). La marca en el centro de la pantalla, facilita la dirección de la cámara hacia el punto del centro focal establecido para la calibración como se puede ver en la Figura 5.1. El punto del centro focal (xf, yf, zf) es una de las variables importantes, que deben ingresarse en el archivo de texto para calibración, se escriben sus coordenadas con respecto al sistema de coordenadas de referencia (x, y, z). No es necesario que el centro de la cámara, no se encuentre exactamente sobre el punto del centro focal que se ha establecido. El proceso de calibración encontrará el verdadero punto del centro focal. Sin embargo es importante ingresar en el archivo de texto para calibración, el valor de una variable de cierta tolerancia llamada `F_tol`. Esta variable evita que el sistema caiga en un mínimo local no deseado, lo que hace es impedir que el optimizador tome valores para las coordenadas del punto, que vayan más allá de dicha tolerancia. Sin embargo mientras más se acerque la marca de la pantalla al supuesto centro focal mejores serán los resultados de calibración, esto se pudo observar en pruebas con un ambiente ideal y controlado generado en VTK.



(a) (b)

Figura 5.1 (a) Imagen del plano de trabajo. (b) Imagen de la marca visual del centro de la cámara dirigida a una posición aproximada del punto del centro focal (xf, yf, zf).

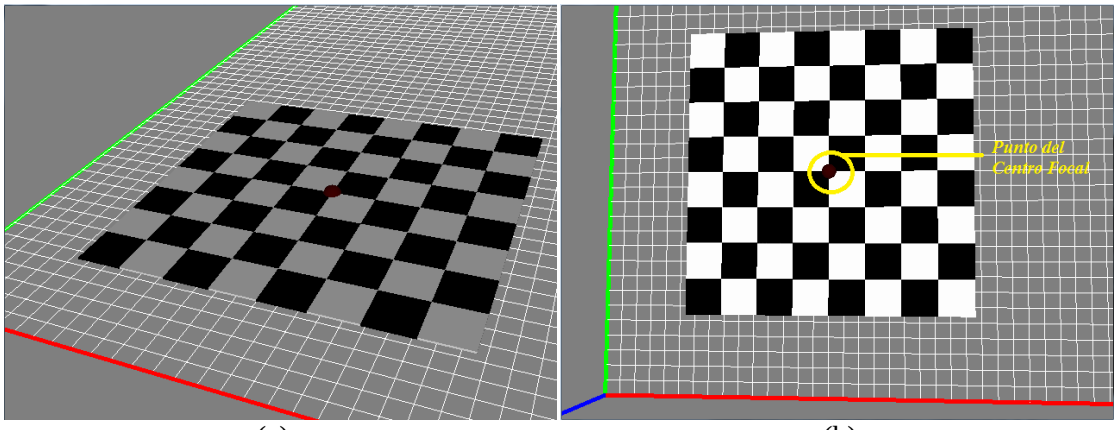
Una vez que se establece el punto del centro focal, lo ideal para este proceso de calibración, es continuar con una medición aproximada de la posición de la cámara (xc, yc, zc), con respecto al origen de coordenadas mundiales y se ingresa la posición de la cámara en el archivo de texto para calibración, aunque podemos colocar la cámara en cualquier posición a simple vista o con un instrumento de medición, realmente no se puede conocer con exactitud la posición del origen de las coordenadas de la cámara para el modelo de pinhole, por eso se dice que la medición que se hace de la posición de la cámara es aproximada, entonces dependiendo de las dimensiones físicas de la cámara puede haber un grado maximo de error el cual se representa con la variable `C_tol`. Al momento que se establecen las coordenadas aproximadas de la cámara, implícitamente el algoritmo tiene suficiente información para determinar un aproximado de la orientación de los ejes de coordenadas mundiales y recrear el ambiente virtual de forma cercana a la realidad.



(a) (b)

Figura 5.2 (a) Hexaedros de calibración ubicados con la orientación establecida por el usuario. (b) Distancia de los hexaedros con respecto al punto del centro focal establecido por el usuario.

Los objetos de calibración que se van a utilizar, se seleccionaron realizando diversas pruebas en un ambiente ideal y controlado generado en VTK, los mejores resultados para una primera y segunda etapa de calibración son 4 hexaedros separados a la misma distancia del punto del centro focal como se ilustra en la Figura 5.2. Para una tercera y cuarta etapa de calibración lo más conveniente es utilizar un tablero de ajedrez que permita obtener resultados más precisos, esto por la cantidad de información que los cuadros del tablero ofrecen al realizar la comparación de imágenes. La orientación con la que se coloquen los objetos de calibración con respecto a los ejes de coordenadas mundiales establecidos por el usuario, será la orientación final que el algoritmo establezca para dichos ejes coordenados.



(a) (b)

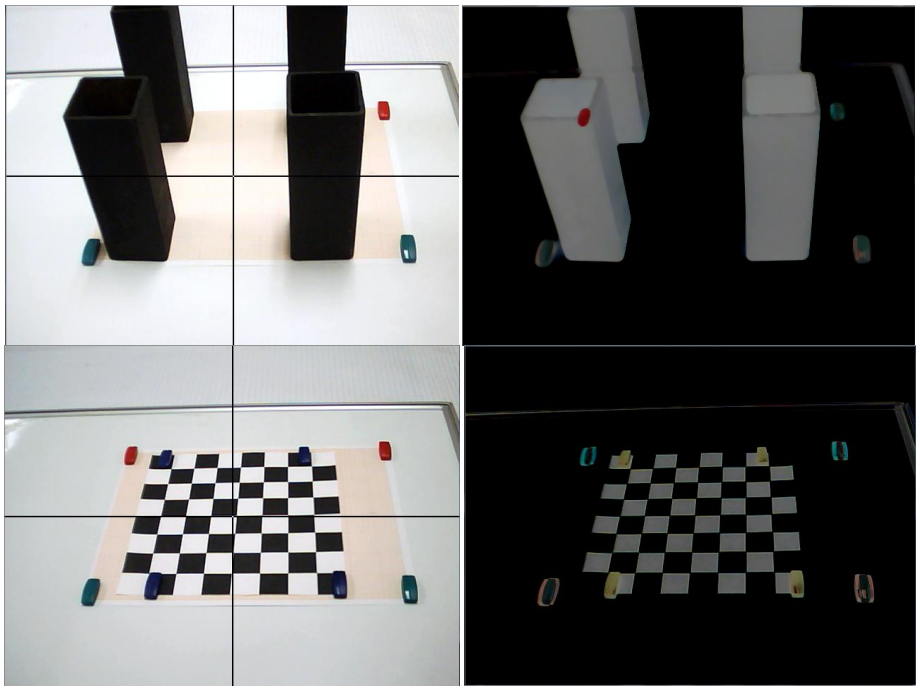
Figura 5.3 (a) Tablero para calibración con la orientación establecida por el usuario. (b) Centro del tablero ubicado sobre el punto del centro focal establecido por el usuario.

Los 4 hexaedros deben ser de dimensiones iguales, cada uno debe tener un ancho igual para todos sus lados y el único valor diferente es la altura. Los valores que se ingresan en el archivo de texto para calibración son el ancho y la altura de los pilares, así como la distancia de los pilares con respecto al punto del centro focal. En caso de que no se cuente con estos 4 objetos de calibración pero desea realizar una aproximación burda de la primera y la segunda etapa de calibración, para así

poder continuar con la siguiente etapa, puede seleccionar la distancia que hay entre los hexaedros igual a cero y esto generara un bloque de lado igual a 2 veces el ancho de cada uno de estos hexaedros y una altura deseada, es algo más fácil de conseguir puesto que se puede utilizar incluso una caja para calibrar en la primera y segunda etapa.

Para el tablero de ajedrez, las condiciones son que tenga el mismo número de cuadros en cada lado, que el numero de cuadros sea par y que el centro del tablero este sobre el punto del centro focal. Los valores que se ingresan en el archivo para calibración son el número de cuadros por lado, el ancho de cada cuadro y el grueso que presente el tablero como se puede ver en la Figura 5.3. Si se desea, el algoritmo puede realizar esta segunda etapa de calibración con el mismo tablero pero a dos diferentes alturas para no perder detalle de la perspectiva de la imagen, entonces se ingresa también en el archivo la altura a la que se presentará la segunda imagen del tablero de ajedrez.

5.1.1. Generación de la Imagen Real de Calibración



(a) (b)

Figura 5.4 (a) Objetos de calibración colocados en la posición y orientación deseada. (b) Imágenes reales de los objetos de calibración.

Para generar la imagen real, se utiliza el método de detección de movimiento mencionado en el capítulo anterior. Se toman las fotografías una vez que se cuenta con los objetos de calibración y se establecen todos los parámetros antes mencionados. Lo primero que se hace es capturar una fotografía de la escena sin objetos es decir una fotografía de referencia. Después se toma una fotografía de los 4 hexaedros para las primeras dos etapas de calibración y otra del tablero de ajedrez para las siguientes dos etapas como se muestra en la Figura 5.4. En el caso que se decida utilizar el tablero a dos diferentes alturas, se capturan dos imágenes una del tablero de ajedrez preferentemente sobre el plano de trabajo y la otra del tablero de ajedrez a determinada altura.

5.2. Generador de Imagen Teórica

La imagen teórica se desarrolla en un ambiente virtual en donde se han ingresado la posición de la cámara, el punto del centro focal y las dimensiones de los objetos de calibración. Al principio la imagen teórica se realiza con el objeto de calibración de la primera etapa y de acuerdo a los valores establecidos en el archivo de texto para calibración, pero en adelante la suposición de dichos valores dependerá del proceso de optimización y del error resultante entre cada comparación de la imagen real con la imagen teórica. El algoritmo dicha comparación para el proceso de calibración se muestra en el apéndice G.

Al generar la imagen teórica, los parámetros variables son la posición de la cámara (xc, yc, zc) y el punto del centro focal (xf, yf, zf) , todos con respecto de los ejes de coordenadas mundiales (x, y, z), el factor Zoom (k) y la orientación de la cámara, específicamente el ángulo de rotación (`roll`), aparentemente se tienen 8 variables desconocidas, pero debido a que el plano donde se está trabajando es conocido el punto del centro focal se limita únicamente a sus coordenadas xf y yf, entonces se reduce el problema a 7 variables independientes, al momento de dar un valor aproximado de la posición de la cámara y del punto del centro focal, se disminuye en gran medida la posibilidad de caer en mínimos locales no deseados.

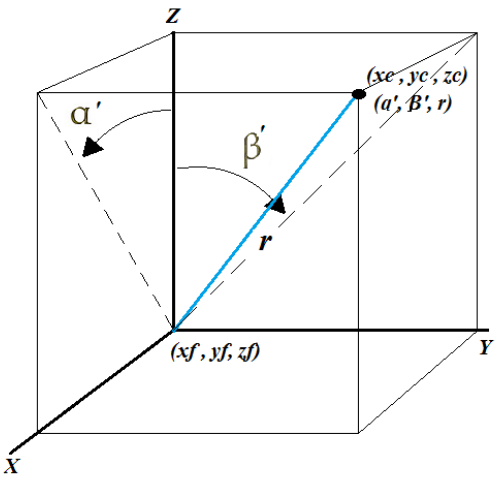


Figura 5.5 Coordenadas de la posición de cámara con base a $\boldsymbol{\alpha'}$, $\boldsymbol{\beta'}$ y $\boldsymbol{r}$.

Para que el proceso de optimización se realice de una forma más eficiente se busca evitar que dos variables causen el mismo efecto en la imagen teórica, por lo tanto se usaran variables que tengan cierta dependencia unas de otras, supóngase que las coordenadas de la posición de la cámara (xc, yc, zc)se representan con respecto a dos ángulos α', β' y una distancia r que va desde el punto del centro focal (xf, yf, zf) a la posición de la cámara, como se muestra en la Figura 5.5.

Por lo tanto las variables independientes que corresponden a la posición de la cámara (α, β, r), se definen como se muestra en la Ecuación 5.1, la Ecuación 5.2 y la Ecuación 5.3.

$$\alpha = \tan(\alpha') = (xc - xf)/(zc - zf) \tag{5.1}$$

$$\beta = \tan(\beta') = (yc - yf)/(zc - zf) \tag{5.2}$$

$$r = (\, zc - zf \,)\sqrt{1 + \alpha^2 + \beta^2} \tag{5.3}$$

Además la variable del $zoom = k$ se considerará dependiente de la distancia r de tal forma que queda como en la Ecuación 5.4.

$$m = k/r \tag{5.4}$$

Si el punto del centro focal (xf, yf, zf) seleccionado es igual a $(0,0,0)$ entonces las ecuaciones anteriores se reducen a la forma como se ve en la Ecuación 5.5, Ecuación 5.6 y en la Ecuación 5.7.

$$\alpha = \tan(\alpha') = xc/zc \tag{5.5}$$

$$\beta = \tan(\beta') = yc/zc \tag{5.6}$$

$$r = (\, zc \,)\sqrt{1 + \alpha^2 + \beta^2} \tag{5.7}$$

Entonces las variables que se utilizan para generar la imagen teórica son $(xc, yc, zc, xf, yf, k, roll)$, pero los parámetros a optimizar son $(\alpha, \beta, r, xf, yf, m, roll)$. Por ello dentro del proceso de optimización para crear la imagen teórica se necesita convertir los parámetros del optimizador a los parámetros que manejan las librerías de VTK, como se muestra en las siguientes ecuaciones.

$$zc = (\, r \,/\, \sqrt{1 + \alpha^2 + \beta^2} \,) + zf \tag{5.8}$$

$$xc = \alpha\,(zc - zf) + xf \tag{5.9}$$

$$yc = \beta(zc - zf) + yf \tag{5.10}$$

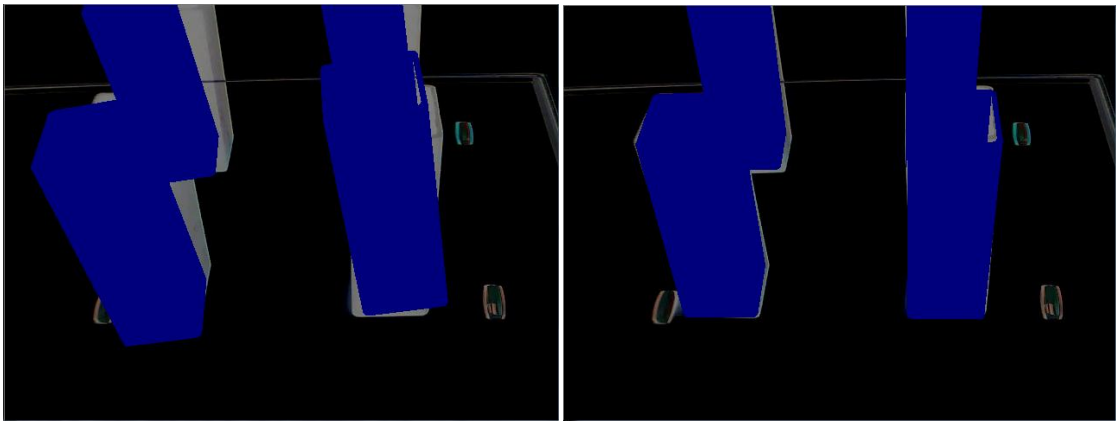
$$k = m \cdot r \tag{5.11}$$

5.3. Desarrollo del Procedimiento de Calibración

Cuando el algoritmo comienza el proceso de calibración, entonces ya se conocen las dimensiones de los objetos de calibración y su posición, además se tiene la posición aproximada de la cámara y del punto del centro focal. Posteriormente el algoritmo comienza a optimizar las 7 variables independientes generando con cada iteración una imagen teórica en donde los objetos y los ejes de coordenadas mundiales se mantienen fijos y solo se mueven las características concernientes a la cámara. Como ya se mencionaba el procedimiento se realiza en 4 etapas: las primeras dos utilizan 4 hexaedros para dar una primera aproximación de las variables independientes y la tercera y cuarta utilizan un tablero de ajedrez para darle precisión a la calibración.

Primera Etapa:

La primera etapa que se muestra en la Figura 5.6 sirve para dar una primera aproximación de las variables independientes, es importante que los 4 hexaedros ocupen un espacio considerable de la imagen para que provean suficiente información de comparación. Los hexaedros son figuras que al obtener sus siluetas nos dan formas sencillas y adecuadas para el optimizador puesto que no cae en mínimos locales con facilidad. En esta primera etapa el optimizador da pasos relativamente grandes para encontrar los mejores valores, el tamaño de los pasos está definido como una fracción de las tolerancias ingresadas en el archivo para calibración. Además sirve optimizar únicamente con 5 variables independientes que son la posición de la cámara, el zoom y el ángulo de roll, dejando de lado el punto del centro focal, en las pruebas previas se ha observado que esto evita que la calibración caiga en mínimos locales y permite acercar a las variables a sus verdaderos valores.



(a) (b)

Figura 5.6 Comparación entre la Imagen teórica y la imagen real. (a) Comparación al inicio de la 1ra etapa. (b) Comparación al final de la 1ra etapa.

Segunda Etapa:

En la segunda etapa de calibración se sigue utilizando la misma imagen real de los 4 hexaedros pero en esta ocasión el optimizador da pasos más pequeños para buscar con el simplex las variables independientes. En esta etapa ya se tiene una aproximación más cercana de la posición de la cámara del zoom y del roll como se puede ver en la Figura 5.7, entonces se optimiza con las 7 variables independientes es decir ahora se integra también la posición del punto del centro focal. Al finalizar esta segunda etapa, se obtienen valores muy cercanos a los reales, si se decide terminar el proceso de calibración en este momento ya se puede empezar a reconocer objetos y ubicarlos, aunque la precisión puede no ser la mejor debido a que puede haber un error al colocar 4 hexaedros en posiciones diferentes con respecto al punto del centro focal aademas que estos 4 objetos pueden no ser exactamente iguales .

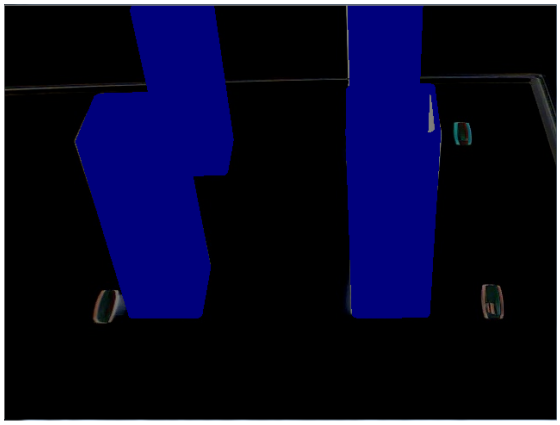
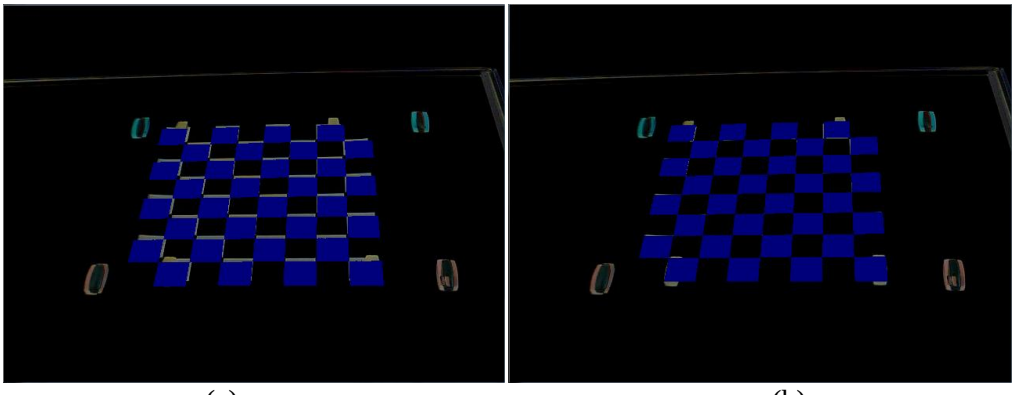


Figura 5.7 Comparación entre la imagen teórica y la imagen real al final de la 2da etapa.

Tercera Etapa:

Los resultados obtenidos de la segunda etapa de optimización se convierten en la primera suposición de esta nueva etapa de calibración, en la cual se realiza la búsqueda de una mejor precisión utilizando un tablero de ajedrez, con el propósito de que la correcta comparación de las imágenes radique principalmente en las transiciones entre los cuadros blancos y negros como se aprecia en la Figura 5.8. Es importante mencionar que no se recomienda utilizar el tablero de ajedrez hasta no tener los resultados de la segunda etapa, puesto que el optimizador puede caer en un mínimo local del cual no va a poder salir. Un ejemplo de un mínimo local será cuando la imagen teórica se resulta de tal forma que solo un número significante de sus cuadros coinciden con los cuadros de la imagen real, pero desplazado por un múltiplo de la dimensión de un cuadro. El hecho de que no todos los cuadros del tablero coincidan, no da suficiente influencia para poder reconocer que se trate de un mínimo local, dado que un desplazamiento del traslape de uno de los cuadros al cuadro "correcto", causará que el error en la comparación se aumente (o sea que la calidad de comparación disminuye). Sin embargo, gracias a la segunda etapa se puede suponer que la imagen real del tablero de ajedrez queda suficientemente bien alineada con la imagen teórica para garantizar que el optimizador no se desvíe a un mínimo local no deseado, sino que al contrario da una mejor aproximación de las variables de la cámara. En esta etapa el optimizador da saltos pequeños puesto que solo busca un mejor acercamiento de todas las variables independientes.



(a) (b)

Figura 5.8 Comparación entre la Imagen teórica y la imagen real. (a) Comparación al inicio de la 3ra etapa. (b) Comparación al final de la 3ra etapa.

Cuarta Etapa:

En esta última etapa de calibración se sigue utilizando la misma imagen real del tablero de ajedrez, vea la Figura 5.9, pero se optimiza utilizando saltos muy pequeños para las variables, únicamente para dar más precisión al sistema.

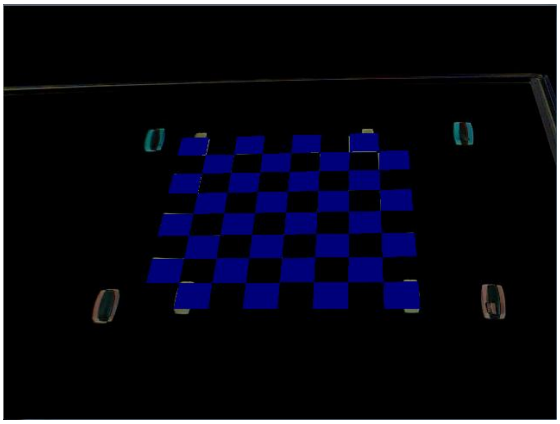
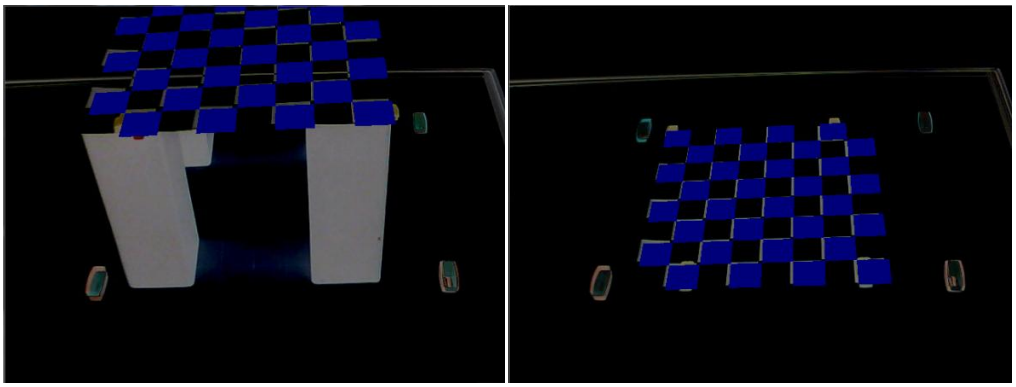


Figura 5.9 Comparación entre la imagen teórica y la imagen real al final de la 4ta etapa.

Tercera y Cuarta Etapas Alternativas:

Después de la segunda etapa de calibración el programa permite optar por seguir calibrando con dos imágenes reales de forma simultánea como se muestra en la Figura 5.10, esto consiste en capturar dos imágenes reales del mismo tablero de ajedrez pero colocado a diferentes alturas, la primer imagen se captura colocando el tablero a cierta altura sobre el punto del centro focal, y la segunda imagen es preferentemente colocando el tablero al nivel del plano de trabajo. Con dos imágenes a diferentes alturas del tablero permite mejorar la detección de la influencia de la perspectiva de la escena que se está calibrando. En la tercera etapa se optimiza con saltos pequeños en las variables, y en la cuarta etapa con saltos un poco más pequeños, únicamente para dar precisión al proceso. Se debe tener cuidado al utilizar las dos imágenes simultáneas, ya que si la posición a la que se coloque el tablero no es paralela una con otra, esto puede comprometer la precisión en la calibración.



(a) (b)

Figura 5.10 (a) Comparación entre la imagen teórica y la imagen real del tablero a una altura igual a la de los pilares. (b) Comparación de la imagen teórica y la imagen real del tablero sobre el plano de trabajo.

CAPÍTULO 6

Experimentos y Discusión de Resultados con Objetos 3D

6.1. Plataforma para los Pruebas de Calibración y Reconocimiento

El programa de visión artificial donde se desarrollaron las pruebas para la calibración y el reconocimiento de los objetos, fue elaborado en el lenguaje de programación de C++, bajo el compilador de Visual Studio, con el apoyo de las librerías de Visual Toolkit (VTK) y de OpenCV. El programa se implementó en Windows 7, utilizando un procesador AMD Turion(tm) 2.20 GHz, 2.0 GB RAM. Al iniciar el programa se abre un menú como se muestra en la Figura 6.1, cada letra del teclado iniciara el procedimiento que se indica, estos procedimientos se dividen en tres secciones principales que son:

- **Procedimientos con imágenes artificiales:** Esta parte está diseñada principalmente para hacer pruebas con imágenes reales que no son capturadas con una cámara, sino creadas en un ambiente virtual, de tal forma que se puedan comparar los resultados con los parámetros utilizados para elaborar la imagen real.
- **Procedimientos con imágenes reales:** En esta parte cada procedimiento utiliza imágenes reales que sean capturadas con cámara. Al seleccionar cualquiera de las letras de esta sección, automáticamente se abrirá una pantalla de lo que ve la cámara y en el instante que se presione Q ó q sobre dicha pantalla, se generará la imagen real. Cuando se realice la calibración del sistema y mientras la cámara se mantenga en una posición fija, se podrá realizar el reconocimiento y ubicación de objetos una y otra vez sin necesidad de volver a calibrar, incluso si se cierra la aplicación. Es importante destacar que antes de seleccionar cualquiera de los procedimientos de esta sección se debe capturar una imagen del fondo estático del área de trabajo con la letra Y ó y, la cual abrirá una pantalla mostrando lo que ve la cámara y guardara la imagen del fondo en el instante que se presione Q ó q.
- **Pruebas con cámara:** Los procedimientos de esta sección, permiten capturar una imagen del fondo estático del área de trabajo con la tecla Y ó y. Una vez que se cuenta con la imagen del fondo, también permite capturar una imagen de cualquier objeto que se coloque en el área de trabajo con la tecla X ó x, generando una imagen real, que permita al usuario visualizar la calidad de las imágenes reales que está generando. Además con la tecla Z ó z, permite visualizar cualquier modelo CAD en la configuración y con el ángulo de rotación que el usuario establezca, con el fin de determinar si se ha configurado correctamente dicho modelo CAD.

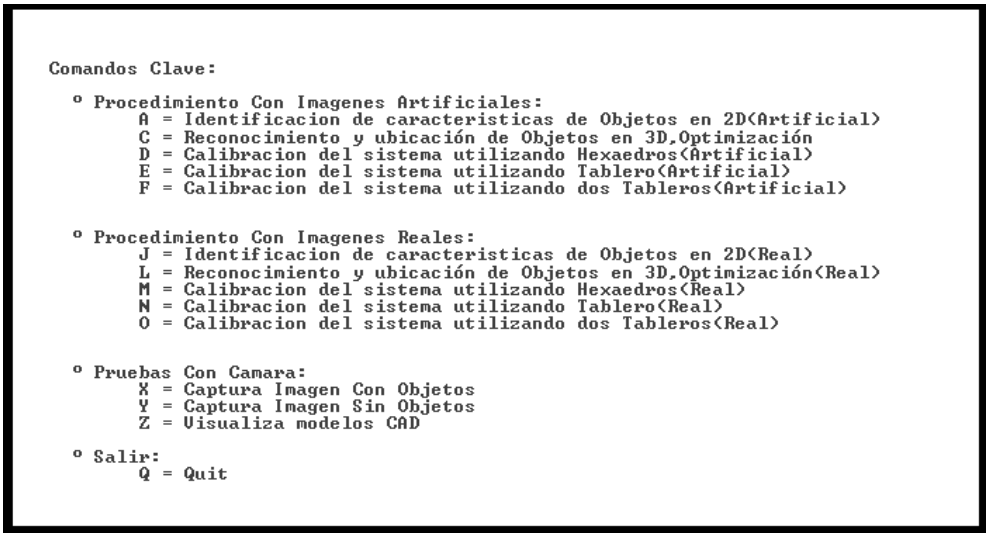


Figura 6.1 Menú de la plataforma del sistema de visión artificial.

6.2. Pruebas Virtuales en la Calibración del Sistema

Como se mencionó en el capítulo anterior la calibración del sistema se realiza con un proceso de optimización, el cual utiliza una figura geométrica denominada simplex para generar las imágenes teóricas. Las constantes λ_i representan el volumen inicial del simplex, es decir la dimensión dentro de la cual se espera que se encuentre rápidamente un mínimo. En la Tabla 6.1 se muestran las constantes λ_i para cada etapa de calibración, las cuales se determinan en relación de los parámetros ingresados por el usuario en el archivo de texto "ConfCalibracion.txt".

Tabla 6.1 *Volumen inicial del simplex para calibración en sus cuatro etapas.*

Volumen del Simplex	1er Etapa	2da Etapa	3ra Etapa	4ta Etapa
$\lambda\ \alpha$	C_Tol/2	C_Tol/6	C_Tol/4	C_Tol/6
$\lambda\ \beta$	C_Tol/2	C_Tol/6	C_Tol/4	C_Tol/6
$\lambda\ r$	C_Tol/2	C_Tol/6	C_Tol/4	C_Tol/6
λ *roll*	60	0.5	1	0.5
λ *zoom*	m/5	m/10	m/10	m/15
λ *xfoc*	Na	F_Tol/6	F_Tol/4	F_Tol/6
λ *yfoc*	Na	F_Tol/6	F_Tol/4	F_Tol/6

Para comprobar que el proceso de calibración funciona correctamente se realizaron diversas pruebas virtuales, en las cuales no solo se generan las imágenes teóricas en un ambiente virtual por medio de las librerías de VTK sino que también se generan las imágenes reales de manera virtual usando los mismos algoritmos. En estas pruebas se conocen todas las variables independientes de la cámara, por lo que se puede comparar el resultado de calibración contra las variables esperadas. Además, la ventaja de estas pruebas es que son bajo condiciones ideales, es decir que no hay errores debido a las distorsiones radiales de la cámara, errores de medición, o ruido en las imágenes capturadas, por lo tanto los resultados quedan libres de cualquiera de estos factores y permite hacer una observación

crítica del procedimiento de calibración. En la Tabla 6.2, se pueden observar 3 de las pruebas de calibración realizadas puramente en el ambiente virtual. En dicha tabla se presentan los valores reales que se asignaron a la posición de la cámara en cada una de estas 3 pruebas y también los valores obtenidos para las variables independientes. En los tres experimentos se alcanza de manera muy cercana la posición del centro focal (xf, yf, zf) donde los resultados se separan de sus valores reales en fracciones de milímetros. El ángulo de $roll$ es otro de los parámetros que no presenta mayores dificultades, pues como se observa la diferencia no llega ni a medio grado. De igual forma el $zoom$ termina con una pequeña diferencia del valor real, pero desafortunadamente para los parámetros de la posición de cámara (xc, yc, zc) y la distancia r la diferencia es más significativa de hasta 1.5 cm, lo que quizá pueda indicar que el algoritmo haya caído en un mínimo local en donde la comparación de la imagen real con la teórica es suficientemente buena como para poder salir de ese mínimo y buscar una mejor alternativa.

Tabla 6.2 *Valores reales contra valores obtenidos del procedimiento de calibración virtual.*

	Valores Reales de la posición de cámara 1	Result. De la Etapa 4	Diferencia de Valores Reales con valores Obtenidos	Valores Reales de la posición de cámara 2	Result. De la Etapa 4	Diferencia de Valores Reales con valores Obtenidos	Valores Reales de la posición de cámara 3	Result. De la Etapa 4	Diferencia de Valores Reales con valores Obtenidos
Número de Iteraciones		408			411			402	
Posición de Cámara X en cm	0	0.6625	-0.6625	35	36.5491	-1.5491	10	9.8012	0.1988
Posición de Cámara Y en cm	-40	-38.6757	-1.3243	35	36.2782	-1.2782	37	36.0015	0.9985
Posición de Cámara Z en cm	40	39.0897	0.9103	30	31.1741	-1.1741	40	39.2973	0.7027
Posición de Centro Focal X en cm	0.3	0.3089	-0.0089	0.5	0.4471	0.0529	0.2	0.1974	0.0026
Posición de Centro Focal Y en cm	0.5	0.4572	0.0428	0.5	0.4828	0.0172	0.3	0.3008	-0.0008
Posición de Centro Focal Z en cm	0	0	0	0	0	0	0	0	0
Angulo de Roll en grados	12	11.6988	0.3012	15	15.2987	-0.2987	0	0.1873	-0.1873
Zoom	0.9	0.8777	0.0223	0.9	0.9385	-0.0385	1.2	1.175	0.025
α	0	0.0169	-0.0169	1.16666	1.1724	-0.00573	0.25	0.2494	0.0006
β	-1	-0.9894	-0.0106	1.16666	1.1637	0.00296	0.925	0.9161	0.0089
Distancia r en cm	56.5685	54.9932	1.5753	57.8791	60.1978	-2.3186	55.3985	54.189	1.2095
m	0.01590	0.016	-9E-05	0.015549	0.0156	-5E-05	0.021661	0.0217	-3.E-05
Duración con gráficos en seg.		91.16			98.31			96.89	
Duración sin gráficos en seg.		46.55			43.31			45.16	

Ahora se calcula el error relativo de los parámetros de cámara de la calibración en el ambiente virtual, para 5 calibraciones en diferentes posiciones como se muestra en la Tabla 6.3, además en la última columna de dicha tabla se calcula el error promedio en las diferentes variables independientes. El error relativo da la posición de cámara (xc, yc, zc) y de la posición del punto del centro focal (xf, yf, zf) fue realizado con respecto a r, que es la distancia de la cámara con el punto del centro focal. Con base en las diferencias de la tabla Tabla 6.2 los errores relativos no sorprenden puesto que son mayores para la posición de cámara y muy inferiores en los otros parámetros. En las últimas dos filas de la misma tabla se muestra el tiempo que duró el procedimiento de calibración mostrando y sin mostrar en pantalla los gráficos de la comparación de las imágenes durante cada iteración. Es importante mencionar que la duración de la calibración varía dependiendo del número de iteraciones que el algoritmo requiera para llegar a un valor óptimo.

Tabla 6.3 *Error relativo de los resultados de calibración virtual de diferentes posiciones.*

	Error relativo de la posición de cámara 1	Error relativo de la posición de cámara 2	Error relativo de la posición de cámara 3	Error relativo de la posición de cámara 4	Error relativo de la posición de cámara 5	Error Promedio
Posición de Cámara X	1.17%	2.68%	0.81%	0.13%	0.36%	1.03%
Posición de Cámara Y	2.34%	2.21%	2.58%	2.27%	1.80%	2.24%
Posición de Cámara Z	1.61%	2.03%	0.51%	3.56%	1.27%	1.79%
Posición de Centro Focal X	0.02%	0.09%	0.01%	0.09%	0.004%	0.04%
Posición de Centro Focal Y	0.08%	0.03%	0.40%	0.13%	0.001%	0.13%
Posición de Centro Focal Z	0%	0%	0%	0%	0%	0%
Angulo de Roll	0.08%	0.08%	0.12%	0.12%	0.05%	0.09%

6.3. Pruebas Reales en la Calibración del Sistema

Una vez que las pruebas virtuales resultan exitosas, se continúa con las pruebas en un ambiente real donde se espera tener los mismos resultados, lo que se hará es realizar el proceso de calibración una serie de diez veces aplicando un filtro de suavizado Gaussiano sobre la imagen, y diez veces sin aplicar ningún filtro. Entonces se realizarán las 10 calibraciones con la misma imagen real de los objetos de calibración, pero cambiando la estimación inicial de la posición y orientación de la cámara.

Tabla 6.4 *Varianza entre los resultados de las etapas de calibración con y sin filtro de suavizado.*

	Valores estimados por el usuario de la posición de cámara	Varianza de resultados de la 2da etapa de calibración con filtro	Varianza de resultados de la 2da etapa de calibración sin filtro	Varianza de resultados de la 4ta etapa de calibración con filtro	Varianza de resultados de la 4ta etapa de calibración sin filtro
Posición de Cámara X	-5	0.0471	0.0570	0.3442	0.2627
Posición de Cámara Y	-50	0.3416	0.4802	2.6583	1.2563
Posición de Cámara Z	40	0.3693	0.5970	1.7546	0.9334
Posición de Centro Focal X	0	0.00085	0.00080	0.00020	0.00011
Posición de Centro Focal Y	0	0.0120	0.0133	9.68E-05	6.33E-05
Posición de Centro Focal Z	0	0	0	0	0
Angulo de Roll	0	0.1211	0.2188	0.3891	0.2865
Zoom	1	0.00015	0.00031	0.00106	0.00049
α	-0.125	3.80E-05	4.87E-05	0.00020	0.00015
β	-1.25	0.00022	0.00021	7.64E-06	8.68E-05
Distancia r	64.2262	0.5689	0.9367	4.4147	2.1559
m	0.01556997	2.33E-09	0.00239	1E-09	4.44E-09

Una vez que se realizan dichas calibraciones se compararan los resultados para poder determinar cuánto varía la calibración con la misma muestra y si existe una gran diferencia dependiendo de si se suaviza o no la imagen real. En la Tabla 6.4 se muestran los valores estimados por el usuario de la posición y orientación de cámara con respecto al origen de coordenadas mundiales y también se observa la varianza de los resultados obtenidos después de las diferentes calibraciones en la segunda etapa y en la cuarta etapa.

En la Tabla 6.4 se aprecia que la varianza de los resultados en la segunda etapa de calibración aumenta para la posición de cámara (xc, yc, zc) cuando no se aplica el filtro de suavizado y para los parámetros del punto de centro focal (xf, yf, zf), el ángulo de $roll$ y el $zoom$ a pesar de tener una varianza más pequeña también aumenta hasta el doble para los casos sin filtro. Sin embargo la varianza de los resultados en la cuarta etapa aumenta drásticamente y sucede el efecto contrario que para la segunda, ya que ahora la varianza utilizando el filtro de suavizado es mayor que sin el filtro. Esto indica que suavizar la silueta de un objeto con tanto volumen como los pilares de calibración mejora la estabilidad de la convergencia en la calibración, sin embargo causa más variaciones suavizar la silueta de un objeto como el tablero de ajedrez donde las esquinas representan información importante para comparar la imagen real con las imágenes teóricas.

En la Tabla 6.5 se muestra el número de iteraciones promedio realizadas por la calibración con y sin filtro de suavizado, se puede observar por el número de iteraciones, que sin un filtro de suavizado, le cuesta ligeramente más trabajo a la metodología el llegar a un resultado óptimo. Con respecto al promedio de los demás parámetros los resultados son muy parecidos, la diferencia entre utilizar o no el filtro está en orden de aproximadamente (+1mm, -1mm).

Tabla 6.5 *Promedio entre los resultados de las etapas de calibración con y sin filtro de suavizado.*

	Valores estimados de la posición de cámara	Promedio de resultados de la 2da etapa de calibración con filtro	Promedio de resultados de la 2da etapa de calibración sin filtro	Promedio de resultados de la 4ta etapa de calibración con filtro	Promedio de resultados de la 4ta etapa de calibración sin filtro
Número de iteraciones		396	420	388	400
Posición de Cámara X en cm	-5	-6.5992	-6.5201	-7.1983	-7.2815
Posición de Cámara Y en cm	-50	-49.7247	-49.7027	-50.7910	-50.6705
Posición de Cámara Z en cm	40	39.9092	39.8735	41.6829	41.5439
Posición de Centro Focal X en cm	0	0.0318	0.0317	-0.0523	-0.0545
Posición de Centro Focal Y en cm	0	0.0827	0.1316	-0.3570	-0.3560
Posición de Centro Focal Z en cm	0	0	0	0	0
Angulo de Roll en grados	0	15.4502	15.4401	16.0399	16.1438
Zoom	1	1.0021	0.9993	1.0244	1.0214
α	-0.125	-0.1653	-0.1636	-0.1727	-0.1753
β	-1.25	-1.2460	-1.2467	-1.2184	-1.2197
Distancia r en cm	64.2262	64.1016	64.0542	66.1008	65.9294
m	0.015569	0.0156	0.0311	0.0155	0.0155
Duración con gráficos en seg.		46.83	48.29	45.17	49.69
Duración sin gráficos en seg.		21.93	22.06	22.63	22.77

Aprovechando los resultados obtenidos de la serie de calibraciones de cámara, lo que se busca ahora es encontrar si existe mucha diferencia entre los resultados de la segunda etapa que es donde se utilizan los 4 hexaedros con la cuarta etapa de calibración que es la última etapa y que utiliza un tablero de ajedrez. Además se desea corroborar que existe una mejoría significativa en la calibración del sistema de la tercera a la cuarta etapa, esto para poder concluir si es necesario realizar la cuarta etapa o el resultado de la tercera es suficiente para terminar con la calibración. Se elaboró la Tabla 6.6 en donde se realizó la varianza entre los resultados de calibración de las diferentes etapas utilizando un filtro de suavizado en la imagen y sin utilizarlo.

Tabla 6.6 *Varianza de la diferencia entre las etapas de calibración.*

	Valores estimados de la posición de camara	Varianza de la diferencia de las variables entre la 4ta y la 2da etapa con filtro	Varianza de la diferencia de las variables entre la 4ta y la 3ra etapa con filtro	Varianza de la diferencia de las variables entre la 4ta y la 2da etapa sin filtro	Varianza de la diferencia de las variables entre la 4ta y la 3ra etapa sin filtro
Posición de Cámara X	-5	0.34721	0.12080	0.17062	0.11047
Posición de Cámara Y	-50	2.37595	0.42122	0.42376	0.11827
Posición de Cámara Z	40	1.22955	0.39227	0.48326	0.03802
Posición de Centro Focal X	0	0.00103	0.00045	0.00055	0.00029
Posición de Centro Focal Y	0	0.01328	4.99E-05	0.01241	0.00016
Posición de Centro Focal Z	0	0	0	0	0
Angulo de Roll	0	0.52100	0.24309	0.23012	0.09358
Zoom	1	0.00079	0.00014	0.00023	5.15E-05
α	-0.125	0.00024	0.00011	9.80E-05	6.20E-05
β	-1.25	0.00021	1.02E-05	0.00013	7.01E-05
Distancia r	64.2262	3.43224	0.72995	0.83878	0.12751
m	0.01556	4E-09	1.11E-09	0.00239	4E-09

En estas pruebas se puede ver que la varianza entre la cuarta y la tercera etapa es realmente pequeña al compararla con la varianza que existe entre la segunda y la cuarta etapas, esto sucede utilizando el filtro de suavizado y también sin utilizarlo, por lo que la cuarta etapa de calibración, se puede realizar solo si el usuario no está satisfecho con el resultado de la tercera etapa. Como ya se mencionaba, de la segunda etapa de calibración a la cuarta la varianza se encuentra más marcada en especial para los parámetros de la posición de cámara (xc, yc, zc).

6.4. Pruebas en el Reconocimiento y Ubicación de los Objetos en un Ambiente Real

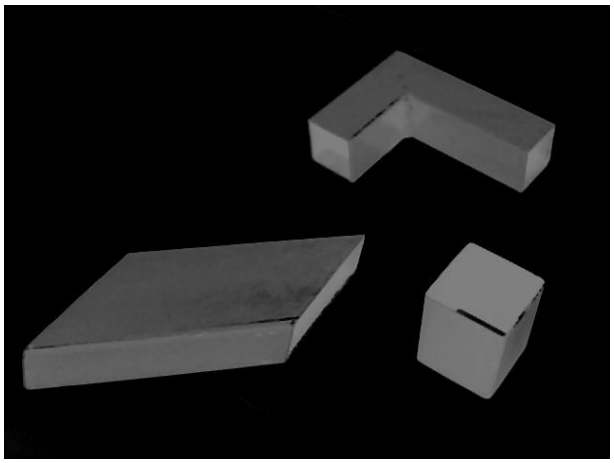


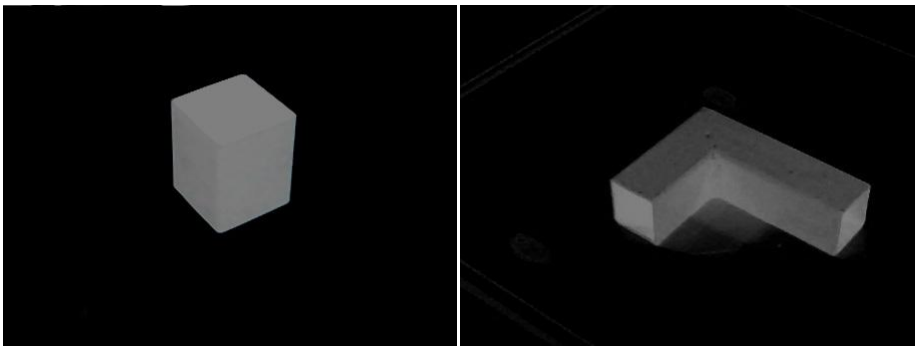
Figura 6.2 Imágenes reales de los objetos a identificar, en la parte superior "L", en el extremo izquierdo "D" y en la esquina inferior derecha "605".

Lo más importante en las pruebas realizadas en esta sección es la de determinar, la efectividad de la metodología para reconocer un objeto, su configuración y ubicación correcta de entre distintos modelos como se muestra en la Figura 6.2.

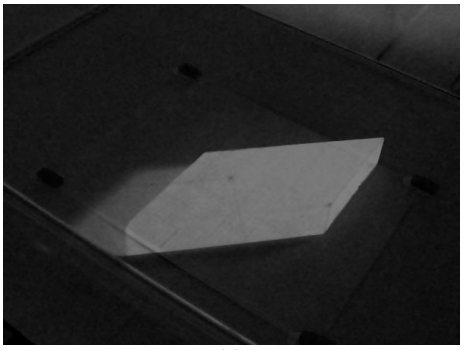
Tabla 6.7 *Volumen inicial del simplex para el reconocimiento y ubicación de objetos en sus dos etapas principales.*

Volumen del Simplex	1ra etapa	2da Etapa
$\lambda\ x$	dimx/3	dimx/20
$\lambda\ y$	dimy/3	dimy/20
$\lambda\ \theta$	Na	50°
$\lambda\ \phi$	Na	ϕ
λ radio cilindro	dimy/6	Na
λ altura cilindro	dimy/20	Na

Para el procedimiento de reconocimiento y ubicación de los objetos se utilizó un volumen para el simplex que se genere de forma automática con las constantes de λ_i que se muestran en la Tabla 6.7, dichas constantes dependen de las dimensiones del área de trabajo como se muestra en la Figura 6.4, En la primera etapa como se mencionó en el capítulo anterior, se utiliza una figura cilíndrica, es por eso que se introduce un radio y una altura para dicha figura geométrica. En la segunda etapa se realiza un ciclo que utiliza el mismo volumen inicial para el simplex en cada configuración de los diferentes objetos. Si alguna de las configuraciones presenta un ángulo de rotación adicional ϕ entonces se utilizará el valor ingresado por el usuario para $\lambda\ \phi$ en el archivo de texto "ModelosCAD.txt".



(a) (b)



(c)

Figura 6.3 (a) Imagen real capturada bajo buenas condiciones de luz. (b) Imagen real capturada bajo regulares condiciones de luz. (c) Imagen real capturada bajo malas condiciones de luz.

Las pruebas de reconocimiento y ubicación consistieron en capturar una serie de 10 fotografías por cada objeto colocado en determinada posición y orientación conocida, además de capturar las imágenes bajo diferentes condiciones de luz como buena, mala y regular, también se realizaron las pruebas aplicando o no un filtro de suavizado sobre la imagen. Con buena iluminación se capturaron las imágenes bajo tres fuentes de iluminación que permitían obtener una imagen real con muy bajas cantidades de ruido, la iluminación regular es con una sola fuente de luz en el área de trabajo y la mala iluminación consiste de una fuente de luz alejada del área de trabajo en la Figura 6.3 se puede observar algunas imágenes reales con distintas condiciones de luz, se debe considerar que si existe influencia de iluminación debida al sol en determinada hora del día, puede afectar las condiciones de iluminación mala y regular disminuyendo la cantidad de ruido esperada.

La Tabla 6.8 se elaboró utilizando un filtro de suavizado y se muestra la posición en la que fueron colocados los diferentes objetos, la configuración y la iluminación utilizada, además se muestra la varianza de los resultados obtenidos por el algoritmo con una serie de las 10 capturas, así como el promedio de los resultados y el error final promedio.

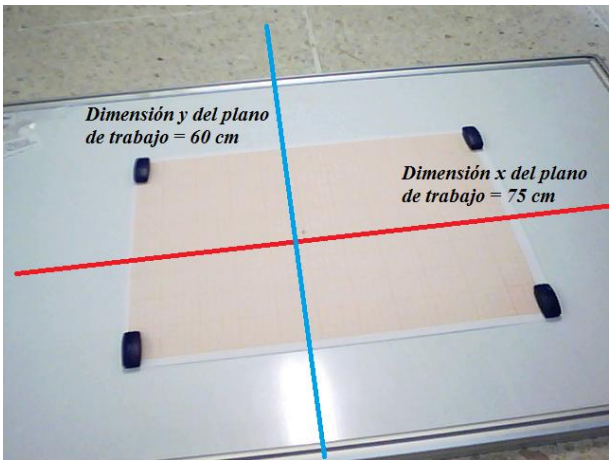


Figura 6.4 Dimensiones del área del plano de trabajo capturada por la cámara, dimx y dimy.

Cabe mencionar que se mide el error relativo de la posición con base en las dimensiones del plano de trabajo observado por la cámara, estos valores se toman como la referencia para estimar el error en los resultados de ubicación del algoritmo, que para la coordenada en x es de 75cm, para la coordenada y es de 60cm y para el ángulo θ es de 360° véase la Figura 6.4.

El error promedio del ángulo θ es bastante grande para lo que se esperaba del objeto "605" en esa posición, este efecto puede haberse causado por las distorsiones radiales, además empeora un poco cuando las condiciones de luz disminuyen, pero este efecto es obvio puesto que la información de la imagen real fue alterada por la falta de iluminación. Para el objeto "L" el error promedio más grande se presentó en la posición *y* del objeto con alrededor de 3 mm de diferencia con el valor real. Finalmente el objeto "D" fue el mejor ubicado por el algoritmo incluso bajo pésimas condiciones de iluminación.

Tabla 6.8 Resultados del reconocimiento y ubicación de objetos en 3D utilizando un filtro de suavizado.

Tabla utilizando filtro de suavizado en la imagen								
POSICIÓN DE COLOCACIÓN	Nombre de objeto	605	605	605	L	L	D	D
	Configuración de objeto	0	0	0	1	1	0	0
	Rotación de objeto	0	0	0	0	0	0	0
	x	-10	-10	-10	10	10	-10	-10
	y	5	5	5	-5	-5	-10	-10
	Ángulo θ	0	0	0	0	0	45	45
VARIANZA DE LOS RESULTADOS DE LAS VARIABLES	x	0.000619	0.00071	0.00118	0.00090	0.00060	0.00026	0.00127
	y	0.001418	0.00313	0.00246	0.00279	0.000748	0.00233	0.01180
	Ángulo θ	0.092559	0.11175	0.09124	0.09389	0.0183	0.04171	0.06483
PROMEDIO DE LOS RESULTADOS DE LAS VARIABLES OBTENIDAS	x prom	-9.96785	-9.93236	-9.9556	9.97473	9.9382	-9.9404	-9.9215
	y prom	5.06081	5.06823	5.06705	-4.73461	-4.6965	-10.018	-9.8864
	Ángulo θ	2.44434	2.87937	3.29868	0.57067	0.6185	45.164	44.6851
ERROR PROMEDIO DE LOS RESULTADOS DE LAS VARIABLES OBTENIDAS	x	0.04%	0.09%	0.06%	0.03%	0.08%	0.08%	0.10%
	y	0.10%	0.11%	0.11%	0.44%	0.51%	0.03%	0.19%
	Ángulo θ	0.68%	0.80%	0.92%	0.16%	0.17%	0.05%	0.09%
	Condiciones de luz	Buena	Regular	Mala	Buena	Regular	Buena	Mala
DURACIÓN CON GRAFICOS	Duración prom. seg.	66.301	66.958	67.205	69.848	68.37	63.494	64.995
DURACIÓN SIN GRAFICOS	Duración prom. seg.	28.38	30.26	29.93	31.48	30.38	29.40	28.01

Ahora en la Tabla 6.9 se realiza el mismo experimento de la Tabla 6.8 pero sin utilizar un filtro de suavizado sobre la imagen, nuevamente se presenta la varianza de los resultados, el promedio de las variables independientes y el error final promedio. Los resultados en la Tabla 6.9 no varían mucho con respecto a los obtenidos en la Tabla 6.8.

Tabla 6.9 Resultados del reconocimiento y ubicación de objetos en 3D sin un filtro de suavizado.

Tabla sin filtro de suavizado en la imagen								
POSICIÓN DE COLOCACIÓN	Nombre de objeto	605	605	605	L	L	D	D
	Configuración de objeto	0	0	0	1	1	0	0
	Rotación de objeto	0	0	0	0	0	0	0
	x	-10	-10	-10	10	10	-10	-10
	y	5	5	5	-5	-5	-10	-10
	Ángulo θ	0	0	0	0	0	45	45
VARIANZA DE LOS RESULTADOS DE LAS VARIABLES	x	0.000363	0.00111	0.001202	0.00058	0.000490	0.0003	0.00189
	y	0.001411	0.004084	0.00194	0.002228	0.000765	0.00296	0.01164
	Ángulo θ	0.13413	0.1356	0.1271	0.07618	0.01507	0.04668	0.05274
PROMEDIO DE LOS RESULTADOS DE LAS VARIABLES OBTENIDAS	x prom	-9.9203	-9.89372	-9.91523	9.9638	9.92852	-9.9420	-9.92142
	y prom	5.042	5.068	5.057	-4.7924	-4.762	-10.019	-9.90316
	Ángulo θ	2.795	3.462	3.852	180.321	180.34	45.148	44.718
ERROR PROMEDIO DE LOS RESULTADOS DE LAS VARIABLES OBTENIDAS	x	0.11%	0.14%	0.11%	0.05%	0.10%	0.08%	0.10%
	y	0.07%	0.11%	0.10%	0.35%	0.40%	0.03%	0.16%
	Ángulo θ	0.78%	0.96%	1.07%	0.09%	0.10%	0.04%	0.08%
	Condiciones de luz	Buena	Regular	Mala	Buena	Regular	Buena	Mala
DURACIÓN CON GRAFICOS	Duración prom. seg.	64.67	64.985	66.365	69.983	68.919	63.919	63.527
DURACIÓN SIN GRAFICOS	Duración prom. seg.	29.81	29.87	30.047	31.01	30.95	28.77	29.51

El algoritmo puede caer en un mínimo local no deseado cuando un modelo CAD no se configura adecuadamente. Se trabajó con diversos modelos en diferentes configuraciones y bajo diferentes

ángulos de rotación con respecto a la vertical y todos los casos el algoritmo es capaz de reconocer adecuadamente el objeto del cual se trataba. Sin embargo de 200 pruebas el algoritmo cayó 4 ocasiones en un mínimo local no deseado, esto sucedió con uno de los tubulares en una posición en particular donde el algoritmo equivocó la configuración correcta del objeto pero no a causa de la metodología, sino que fue a debido a una necesidad de mas iteraciones en el proceso de optimización, ya que cada configuración de los objetos tiene un número máximo de iteraciones, pero aumentar este número de iteraciones haría que el proceso en general tomara más tiempo del que toma hasta este momento.

6.5. Pruebas en el Reconocimiento de Variaciones en la Dimensión de Objetos

El algoritmo ha demostrado ser capaz de reconocer y ubicar correctamente diferentes objetos de una base de datos de sus respectivos modelos CAD y bajo diferentes condiciones de luz. En esta parte es importante poner a prueba el algoritmo para reconocer objetos de la misma forma pero de longitudes diferentes. Se utilizaron grupos de tubulares de 51mm de ancho con diferencia en su longitud de 3mm, 2mm y 1mm como se puede ver en la Figura 6.5.

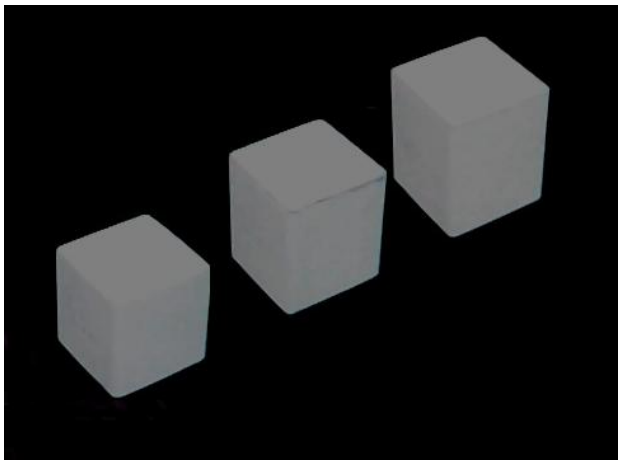


Figura 6.5 Siluetas de tres tubulares del mismo ancho pero de diferente longitud, se les asigna un nombre de acuerdo a su longitud en mm: “51”, “54” y “572” respectivamente.

Lo que se hace es crear una base de datos con los diferentes grupos de modelos y entonces capturar una imagen de uno de los objetos esperados y dejar que el algoritmo indiqué el nombre del objeto del que se trata. En la Tabla 6.10 se utilizo una base de datos de tres modelos CAD de tubulares de longitudes de 51mm, 54mm y 57.2mm, todos los objetos que fueron capturados por la cámara se colocaron en la misma posición y orientación para que no se alteraran los resultados obtenidos. De igual manera se realizó el mismo experimento en la Tabla 6.11 pero los objetos a identificar se colocaron en diferente posición y orientación con el propósito de que los resultados fueran similares sin importar la posición y orientación en que se encuentren los objetos. Tanto en la Tabla 6.10 como en la Tabla 6.11 se observa la posición y la colocación de los objetos así como el objeto, la configuración y la posición encontrada por el algoritmo, también se agregó la varianza y el promedio de la posición x y y encontradas y el ángulo θ.

Tabla 6.10 Identificación y reconocimiento de un grupo de objetos con diferencia de 3mm en una de sus dimensiones.

Identificación de objetos con 3 mm de diferencia en sus dimensiones 51mm, 54mm, 57.2mm				
Posición de Colocación	Nombre de Objeto	51	54	572
	Configuración de Objeto	0	0	0
	Rotación de Objeto	0	0	0
	Posición X	0.1	0.1	0.1
	Posición Y	0.1	0.1	0.1
	Ángulo Θ	0	0	0
Posición del Origen del Objeto Obtenida	Nombre de Objeto Identificado	51	54	572
	Configuración de Objeto Identificado	0	0	0
	Posición X	0.1666	0.1505	0.1802
	Posición Y	-0.0007	0.0012	0.0209
	Ángulo Θ	-0.7249	-0.0576	-0.8674
Varianzas y Promedios				
	Varianza		Promedio	
Posición X	0.000221043		-0.065766667	
Posición Y	0.000143043		0.092866667	
Ángulo Θ	0.186895263		0.549966667	

Tabla 6.11 Identificación y reconocimiento de un grupo de objetos con diferencia de 3mm en una de sus dimensiones.

Identificación de objetos con 3 mm de diferencia en sus dimensiones 51mm, 54mm, 57.2mm				
Posición de Colocación	Nombre de Objeto	51	54	572
	Configuración de Objeto	1	1	1
	Rotación de Objeto	1	1	1
	Posición X	0.1	0.1	0.1
	Posición Y	0	0	0
	Ángulo Θ	90	90	90
Posición del Origen del Objeto Obtenida	Nombre de Objeto Identificado	51	54	572
	Configuración de Objeto Identificado	1	1	1
	Posición X	0.1642	0.1638	0.2222
	Posición Y	-0.1382	-0.1177	-0.1332
	Ángulo Θ	90.1204	89.5107	89.513
Varianzas y Promedios				
	Varianza		Promedio	
Posición X	0.00112912		0.1834	
Posición Y	0.00011425		-0.1297	
Ángulo Θ	0.12344569		89.7147	

En las pruebas anteriores el algoritmo no tuvo ningún problema en reconocer correctamente a los objetos utilizados. Entonces ahora se utiliza la misma dinámica para los siguientes dos experimentos en este caso la diferencia es que el margen de la distancia entre las longitudes de los modelos es menor con un valor de 2mm. Los modelos que se utilizaron ahora son tres tubulares de 51mm de ancho y de longitudes de 56mm, 58mm y 60.5mm. En la Tabla 6.12 y la Tabla 6.13 se puede ver la posición y orientación de colocación de los tres diferentes objetos así como los resultados encontrados por el algoritmo en el reconocimiento y ubicación. También se agregó la varianza y el valor promedio de los resultados.

Tabla 6.12 Identificación y reconocimiento de un grupo de objetos con diferencia de 2mm en una de sus dimensiones.

Identificación de objetos con 2 mm de diferencia en sus dimensiones 56mm, 58mm, 60.5mm				
Posición de Colocación	Nombre de Objeto	56	58	605
	Configuración de Objeto	0	0	0
	Rotación de Objeto	0	0	0
	Posición X	0.1	0.1	0.1
	Posición Y	0.1	0.1	0.1
	Ángulo Θ	0	0	0
Posición del Origen del Objeto Obtenida	Nombre de Objeto Identificado	56	58	605
	Configuración de Objeto Identificado	0	0	0
	Posición X	0.1629	0.2076	0.2094
	Posición Y	-0.027	0.0001	-0.014
	Ángulo Θ	-0.1152	0.0535	-1.1352
Varianzas y Promedios				
	Varianza		**Promedio**	
Posición X	0.00069393		0.1933	
Posición Y	0.000183703		-0.013633333	
Ángulo Θ	0.413644563		-0.398966667	

Tabla 6.13 Identificación y reconocimiento de un grupo de objetos con diferencia de 2mm en una de sus dimensiones.

Identificación de objetos con 2 mm de diferencia en sus dimensiones 56mm, 58mm, 60.5mm				
Posición de Colocación	Nombre de Objeto	56	58	605
	Configuración de Objeto	1	1	1
	Rotación de Objeto	1	1	1
	Posición X	0.1	0.1	0.1
	Posición Y	0	0	0
	Ángulo Θ	90	90	90
Posición del Origen del Objeto Obtenida	Nombre de Objeto Identificado	56	58	605
	Configuración de Objeto Identificado	1	1	1
	Posición X	0.1315	0.2419	0.2468
	Posición Y	-0.1249	-0.1359	-0.1384
	Ángulo Θ	88.8927	89.4967	89.6036
Varianzas y Promedios				
	Varianza		**Promedio**	
Posición X	0.004251043		0.206733333	
Posición Y	5.15833E-05		-0.133066667	
Ángulo Θ	0.14693707		89.331	

Mientras que el algoritmo siga reconociendo correctamente a los objetos con esas variación entre la longitud de los tubulares vale la pena seguir disminuyendo gradualmente esta diferencia hasta encontrar el punto en que no distinga bien entre dos objetos de distinto tamaño. Por lo tanto en la Tabla 6.14 y la Tabla 6.15 se muestran los resultados de los últimos dos experimentos de este tipo en los cuales la base de datos de los modelos CAD estuvo formada por tres tubulares de 51mm de ancho y de longitudes de 56mm, 57.2mm y 58mm, en donde la diferencia es de 1.2mm y de 0.8mm.

Tabla 6.14 Identificación y reconocimiento de un grupo de objetos con diferencia de 1mm en una de sus dimensiones.

Identificación de objetos con 1.2 mm de diferencia en sus dimensiones 56mm, 57.2mm, 58mm				
Posición de Colocación	Nombre de Objeto	56	572	58
	Configuración de Objeto	0	0	0
	Rotación de Objeto	0	0	0
	Posición X	0.1	0.1	0.1
	Posición Y	0.1	0.1	0.1
	Ángulo Θ	0	0	0
Posición del Origen del Objeto Obtenida	Nombre de Objeto Identificado	56	572	58
	Configuración de Objeto Identificado	0	0	0
	Posición X	0.1629	0.1786	0.2093
	Posición Y	-0.027	0.0176	-0.0003
	Ángulo Θ	-0.1152	-0.8616	0.032
Varianzas y Promedios				
	Varianza		Promedio	
Posición X	0.00055699		0.1836	
Posición Y	0.000503743		-0.003233333	
Ángulo Θ	0.229550293		-0.314933333	

En la Tabla 6.15 se marca con color rojo el error cometido por el algoritmo al determinar equivocadamente el modelo CAD del objeto de la imagen real. Con un margen de diferencia de 1.2mm y de 0.8mm el algoritmo no fue capaz de reconocer correctamente a estos objetos y en esta posición. Hay que recordar que están en juego diversos factores que causan esta limitación como lo son las distorsiones radiales de la cámara, la precisión en el proceso de calibración, la distancia de la cámara con el plano de trabajo, la resolución y la nitidez de la cámara.

Tabla 6.15 Identificación y reconocimiento de un grupo de objetos con diferencia de 1mm en una de sus dimensiones.

Identificación de objetos con 1 mm de diferencia en sus dimensiones 56mm, 57.2mm, 58mm				
Posición de Colocación	Nombre de Objeto	56	572	58
	Configuración de Objeto	1	1	1
	Rotación de Objeto	1	1	1
	Posición X	0.1	0.1	0.1
	Posición Y	0	0	0
	Ángulo Θ	90	90	90
Posición del Origen del Objeto Obtenida	Nombre de Objeto Identificado	56	56	572
	Configuración de Objeto Identificado	1	1	1
	Posición X	0.1315	0.221	0.2746
	Posición Y	-0.1249	-0.0431	-0.1336
	Ángulo Θ	88.8927	89.7748	89.6529
Varianzas y Promedios				
	Varianza		Promedio	
Posición X	0.005226803		0.209033333	
Posición Y	0.002492863		-0.100533333	
Ángulo Θ	0.228477343		89.44013333	

Debido a que la duración de cada procedimiento ya sea calibración o reconocimiento de algún objeto varía mucho en el número de iteraciones que el algoritmo utiliza, se encontró que el tiempo de duración promedio de cada iteración del algoritmo es igual a 0.112678 segundos, este tiempo ocurre cuando se imprime en pantalla las imágenes que se están comparando de forma que el usuario tenga la oportunidad de visualizar lo que está ocurriendo durante la ejecución del programa. Si el usuario no desea imprimir en pantalla las imágenes entonces el tiempo de duración promedio

por cada iteración es igual a 0.042890 segundos que corresponde a un 38.06% del tiempo cuando se visualiza en pantalla el proceso de reconocimiento y ubicación de los objetos.

CONCLUSIONES

Se ha estudiado el reconocimiento de objetos por visión utilizando el reconocimiento por modelos. Esta metodología permite el reconocimiento y la ubicación de objetos de los cuales se tenga su modelo CAD.

Cuando dichos modelos CAD se configuran correctamente entonces el algoritmo arroja un resultado correcto en el reconocimiento y de una precisión de porciento para la localización.

Cuando la metodología se ejecuta sobre la misma imagen real dos veces y con la misma suposición de los parámetros iniciales entonces arroja exactamente el mismo resultado en todas las variables independientes.

La generación de la imagen real por medio de detección de movimiento es una forma bastante robusta y practica que permite trabajar adecuadamente incluso bajo condiciones malas de iluminación, permite segmentar las imágenes de los objetos en movimiento de manera clara y precisa. Uno de los principales problemas de la técnica de detección de movimiento es la aparición de sombras de los objetos cuando no se tienen las mejores condiciones de luz, aunque por lo general una sombra tendrá niveles bajos de intensidad de luz al momento de generar la imagen real, y dada la metodología estas sombras no significan un factor de gran importancia para reconocer y ubicar correctamente a los objetos de interés. El segundo de los problemas de esta técnica, es cuando los objetos son de un color igual o parecido al fondo. Cuando esto sucede la silueta del objeto en la imagen real no aparece completamente, faltando información importante para un correcto reconocimiento. Se tiene la ventaja que para la generación de imágenes reales se puede utilizar cualquier cámara con interfaz a la computadora para realizar la captura de las imágenes que permitan detectar los objetos a reconocer. El algoritmo puede manejar cualquier resolución de cámara pero evidentemente a mayor resolución requiere de un mayor costo computacional y por lo tanto un mayor tiempo de ejecución. En las pruebas realizadas se pudo obtener un correcto reconocimiento de los objetos y una cercana ubicación incluso bajo malas condiciones de iluminación, con la presencia de sombras y ruido, lo que significa un resultado muy favorable utilizando la metodología propuesta.

El algoritmo puede caer en un mínimo local no deseado cuando un modelo CAD no se configura adecuadamente. Debido a que el algoritmo permite mostrar en pantalla la serie de iteraciones realizadas, el usuario puede corregir algún error cometido en la configuración de algún modelo. Se trabajó con diversos modelos en diferentes configuraciones y bajo diferentes ángulos de rotación con respecto a la vertical y en casi todos los casos el algoritmo es capaz de reconocer adecuadamente el objeto del cual se trataba, sin embargo las ocasiones en que el algoritmo cayó en un mínimo local no deseado, sucedió con uno de los tubulares en una posición en particular donde el algoritmo equivocó la configuración correcta del objeto pero no a causa de la metodología, sino que fue a debido a una necesidad de mas iteraciones en el proceso de optimización, pero aumentar este número de iteraciones haría que el proceso en general tomara más tiempo del que toma hasta

este momento. Otra de las ocasiones en que el algoritmo puede caer en un mínimo local es cuando la diferencia entre las dimensiones de dos objetos es muy pequeña con respecto a la posición de la cámara.

La metodología propuesta aporta bases fuertes para sistemas de visión artificial que trabajen con una sola cámara y sean capaces de reconocer y ubicar objetos en 3D. La metodología puede trabajar favorablemente bajo malas condiciones de iluminación, con la presencia de sombras y ofrece una buena precisión.

El desarrollo del procedimiento de calibración causó diversas dificultades, una de ellas fue el de encontrar los objetos de calibración adecuados que permitieran optimizar las variables independientes de la mejor manera y con un resultado adecuado y bastante cercano al valor real de la posición de cámara, se intento con diversos objetos pero no mostraban los resultados esperados, finalmente después de diversas pruebas se pudo observar que mientras mayor era el área de la imagen ocupada por el objeto de calibración se obtenía una mejor aproximación de la posición correcta de la cámara y además que si dicho objeto de calibración tiene una altura suficiente que permita mejorar la detección de la influencia de su perspectiva esto mejora aun más el resultado de los parámetros buscados.

En primera instancia se había descartado el tablero de ajedrez por caer fácilmente en mínimos locales cuando se utilizaba como primera etapa de calibración, pero después de utilizar los 4 hexaedros con una posición de cámara bastante cerca de su valor real, el tablero permite dar una mejor precisión sin riesgo de caer en un mínimo local.

El sistema de calibración es parte fundamental de este sistema de reconocimiento de objetos, la precisión con que se realice la calibración afectará de forma importante a la ubicación de los objetos en 3D.

Trabajo a Futuro

En el estudio se ha detectado que las distorsiones radiales son pequeñas, pero son existentes. Esto puede causar errores en la ubicación y dimensionamiento de piezas. Como trabajo a futuro, se propone eliminar los errores causados por las distorsiones radiales de la cámara tomando la distorsión radial en cuenta en el generador de imágenes dentro de la misma metodología propuesta, lo cual se espera que mejore el proceso de calibración y de ubicación de objetos.

En el algoritmo se implementó la posibilidad de realizar el procedimiento de calibración utilizando dos imágenes del tablero de ajedrez a diferentes alturas, esto se realizó con el propósito de mejorar el procedimiento de calibración y obtener una mayor precisión, pero la dificultad se presentó al intentar tener dos tableros de ajedrez alineados a diferentes alturas, un error en la colocación causaría que el procedimiento de calibración empeorara en lugar de mejorar. Se puede seguir desarrollando esta idea, dando la posibilidad de que el tablero que se coloque a una mayor altura no tenga que estar alineado ni de forma paralela al tablero inferior, dejando que el algoritmo determine la rotación y la correcta posición del tablero superior al mismo tiempo que identifica los parámetros de la calibración.

Las pruebas presentadas solo tuvieron un objeto en cada imagen, pero en la práctica puede ser necesario reconocer y ubicar objetos parcialmente ocluidos, y de más de un solo objeto en la imagen real.

Para configurar los modelos CAD de manera más sencilla y automática, es posible agregar al programa un motor físico (physics engine), que permita simular interacciones físicas de modelos en 3D, como por ejemplo la dinámica del cuerpo rígido, entonces solo sería necesario colocar cierta vista de un determinado modelo CAD sobre un plano de trabajo y permitir que el algoritmo arroje el valor correcto de la posición para esa configuración.

Para mejorar la aplicabilidad del método en situaciones reales e industriales, es necesario reducir el tiempo de reconocimiento y ubicación de los objetos. El método se presta para el uso de procesamiento en paralelo, se puede esperar que existan oportunidades para la optimización del código del algoritmo y se puede considerar la reducción de las dimensiones de la imagen real y la imagen teórica utilizando una "bounding box" (un recuadro que abarca los límites de la silueta identificada). Además, es probable que otros algoritmos de optimización puedan converger más rápido, aunque este método presenta la ventaja de su robustez.

En este momento el método identifica la correspondencia entre imágenes únicamente en términos de la intensidad de los pixeles en cada imagen, sin usar la información de colores. Se sugiere investigar e implementar el uso de dicha información en la imagen, lo cual evitara problemas que pueden causar las sombras en caso de iluminación unilateral.

El método de la identificación de objetos y su configuración no es óptimo, y se ha detectado que en algunos casos resultó en una identificación incorrecta, por haber truncado el número de iteraciones en cada configuración. Este procedimiento puede ser mejorado, y se sugiere además la integración de procedimientos que permitan un reconocimiento y ubicación de los objetos más rápido como es el tamaño y el color de los mismos.

Realizar la calibración utilizando dos tableros de ajedrez de forma simultánea, lo cual ya está implementado, pero se está buscando la manera de colocar los tableros en el ambiente real de forma paralela y precisa sin comprometer la precisión del sistema.

Otro método que se ha utilizado con éxito para una ubicación más precisa de objetos es el uso de luz estructurada proyectada sobre los objetos. Este principio puede ser utilizado dentro del método estudiado en esta tesis, al desarrollar un generador de imágenes que correspondan a la proyección de luz estructurada sobre los objetos sólidos CAD que se busca ubicar.

Bibliografía

ActiveState - The Dynamic Language. (s.f.). *Tcl Developer Xchange*. Recuperado el Junio de 2010, de http://www.tcl.tk

Adler, V., & Friedman, E. G. (2000). Uniform Repeater Insertion in RC Trees. *IEEE Transactions on circuits and systems: Fundamental Theory And Applications* , Vol. 47, No. 10.

Bauckhage, C., Braunt, E., & Sagerert, G. (2004). From Image Features to Symbols and Vice Versa Using Graphs to Loop Data and Model Driven Processing in Visual Assembly Recognition. *International Journal of Pattern Recognition & Artificial Intelligence* , 18(3), 497-517.

Bourbakis, N. N., Yuan, P. P., & Kakumanu, P. P. (2008). A Graph Based Object Description and Recognition Methodology. *International Journal on Artificial Intelligence Tools* , 17(6), 1161-1194.

Chan, K. L. (2008). Video-Based Gait Analysis by Silhoette Chamfer Distance and Kalman Filter. *International Journal of Image & Graphics* , 8(3), 383-418.

Collado, R. (s.f.). *Rubén Collado*. Recuperado el abril de 2011, de http://www.rubencollado.net/v2/

Cyr, C. M., & Kimia, B. B. (2004). A Similarity-Based Aspect-Graph Approach to 3D Object Recognition. *International Journal of Computer Vision* , 57(1), 5-22.

Eckes, C., Triesch, J., & von der Malsburg, C. (2006). Analysis of Cluttered Scenes Using an Elastic Matching Approach for Stereo Images. *Neural Computation* , 18(6), 1441-1471.

Entzinger, J. O. (2005). A Flexible Seam Detection Technique for Robotic Laser Welding. *Tesis de Maestria* . Twente: University of Twente.

Fusier, F., Valentin, V., Brémond, F., Thonnat, M., Borg, M., Thirde, D., y otros. (2007). Video Understanding for Complex Activity Recognition. *Machine Vision and Applications* , 18:167–188 DOI 10.1007/s00138-006-0054-y.

Glover, J., Rus, D., & Roy, N. (2008). Probabilistic Models of Object Geometry for Grasp Planning. *Proceedings of Robotics: Science and Systems (RSS 2008)* . Zurich, Switzerland.

Gorelick, L., & Basri, R. (2009). Shape Based Detection and Top-Down Delineation Using Image Segments. *International Journal of Computer Vision* , 83(3), 211-232. doi:10.1007/s11263-009-0216-2.

Hsu, W. L., Hsiao, C. C., Chang, Y. L., & Chen, T. L. (2009). Vision-Based Monitoring Method Using Gray Relational Analysis. *IET Computer Vision* , 3(3), 103-111. doi:10.1049/iet-cvi.2007.0045.

Hu, J., & Su, T. (2008). Flexible 3D Object Recocnition Framework Using 2D Views Via a Similarity-Based Aspect-Gaph Approach. *International Journal of Pattern Recognition & Artificial Intelligence* , 22(6), 1141-1169.

Igual, R., & Medrano, C. (s.f.). *Tutorial de OpenCV*. Obtenido de http://docencia-eupt.unizar.es/ctmedra/tutorial_opencv.pdf

Kampel, M., Wildenauer, H., Blauensteiner, P., & Hanbury, A. (2007). Improved Motion Segmentation Based on Shadow Detection. *Electronic Letters on Computer Vision and Image Analysis* , 6(3):1-12.

Kang, L., Lim Kah, B., & Yao, J. (2009). Image Recognition of Occluded Objects Based on Improved Curve Moment Invariants. *Journal of Digital Information Management* , 7(3),152-158.

Kim, T. K., Im, J. H., & Paik, J. K. (2009). Video Object Segmentation and its Salient Motion Detection Using Adaptive Background Generation. *Electronics Letters* , Vol. 45 No. 11.

Kitware, Inc. (s.f.). *CMAKE*. Recuperado el Junio de 2010, de http://www.cmake.org/

Kitware, Inc. (s.f.). *VTK - The Visualization Toolkit*. Recuperado el Junio de 2010, de http://www.vtk.org/

Kwang-Ok, A., Chang-Hwan, I., & Hyun-Kyo, J. (2008). Magnetoencephalography Source Localization Using Improved Simplex Method. *Inverse Problems in Science & Engineering* , 16 (4), 499-510. doi:10.1080/17415970701661412.

Lagarias, J. C., Reeds, J. A., Wright, M. H., & Wright, P. E. (1998). Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions. *SIAM Journal of Optimization* , Vol. 9, No. 1, pp. 112-147.

Lakhany, A., & Mausser, H. (2000). Estimating the Parameters of the Generalized Lambda Distribution. *Algo Research Quarterly* , vol. 3, no. 3, pp. 47-58.

Maslov, I., & Gertner, I. (2005). Reducing the Computational Cost of Local Search in the Hybrid Evolutionary Algorithm with Application to Electronic Imaging. *Engineering Optimization* , 37 (1),103-119. doi:10.1080/0305215041233129838.

Nelder, J. A., & Mead, R. (1965). A Simplex Method for Function Minimization. *Computer Journal, vol. 7* , pp. 308-313.

Oberhoff, D., & Kolesnik, M. (2008). Neural Object Recognition by Hierarchical Appearance Learning. *International Journal of Pattern Recognition & Artificial Intelligence* , 22(5), 883-897.

OpenCV. (s.f.). *OpenCVWiki*. Recuperado el Junio de 2010, de http://opencv.willowgarage.com/wiki/Welcome

Ouhaddi, H., & Horain, P. (1999). 3D Hand Gesture Tracking by Model Registration. En *Proc. IWSNHC3DI'99* (págs. 70-73).

Press, W., Teukolsky, S., Vetterling, W., & Flannery, B. (1992). *Numerical Recipes in C, 2nd ed.* Cambridge: Cambridge Univ. Press.

Rogalsky, T., Derksen, R. W., & Kocabiyik, S. (1999). Differential Evolution in Aerodynamic Optimization. En *Proceedings of the 46th Annual Conference of the Canadian Aeronautics and Space Institute* (págs. 26 - 36).

Schroeder, W. J., Avila, L. S., Martin, K. M., Hoffman, W. A., & Law, C. C. (2001). *The Visualization Toolkit User's Guide.* Kitware.

Schroeder, W. J., Martin, K., & Lorensen, W. E. (1997). *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics.* Prentice Hall.

Seokho, C., Caldas, C. H., & Dae Young, K. (2009). A Methodology for Object Identification and Tracking in Construction Based on Spatial Modeling and Image Matching Techniques. *Computer-Aided Civil & Infrastructure Engineering* , 24(3), 199-211. doi:10.1111.

Tafur Sotelo, J. C., & Sobrado Malpartida, E. (2007). Sistema de Visión Artificial para el Reconocimiento y Manipulación de Objetos Utilizando un Brazo Robot. *8º Congreso Iberoamericano de Ingeniería Mecánica.*

Tao, J., Tan, Y., & Lu, W. (2007). Robust Color Object Traking with Application to People Monitoring. *International Journal of Image & Graphics* , 7(2), 227-254.

Tarabanis, K., Allen, P., & Tsai, R. (2002). A survey of sensor planning in computer vision. *Robotics and Automation, IEEE Transactions on* , 86 - 104.

Thakoor, N., & Gao, J. X. (2008). Automatic Video Object Extraction with Camera in Motion. *International Journal of Image & Graphics* , 8(4), 573-600.

Veit, T., Cao, F., & Bouthemy, P. (2006). An a contrario Decision Framework for Region-Based Motion Detection. *International Journal of Computer Vision* , 68(2), 163-178. doi:10.1007/s11263-006-6661-2.

Westphal, G., & Wärtz, R. P. (2009). Combining Feature- and Correspondence-Based Methods for Visual Object Recognition. *Neural Computation* , 21(7), 1952-1989.

Wilson, A. (s.f.). *Vision Systems Design*. Recuperado el Octubre de 2010, de http://www.vision-systems.com/articles/2010/10/plumbing-parts-assembly.html

Xie, S. S., Xie, S. S., Cheng, D. D., Wong, S. S., & Haemmerle, E. E. (2008). Three-Dimensional Object Recognition System for Enhancing the Intelligence of a KUKA Robot. *International Journal of Advanced Manufacturing Technology* , 38(7/8), 822-839. doi:10.1007/s00170.

Zambrano Rey, M. G., Parra Rodríguez, A. C., Manrique Torres, R. M., & Bustacara Medina, J. C. (2007). Estación de Control de Calidad por Visión Artificial Para un Centro de Manufactura Integrada por Computador (CIM). (Spanish). *Ingeniería y Universidad* , 11(1), 33-55.

APÉNDICE A

Instalación de Visual Toolkit

A1. Visual Toolkit

El paquete de herramientas de visualización (VTK) es un código libre, un software disponible de forma gratuita para gráficos en 3D, procesamiento de imágenes y visualización. VTK consiste en una librería de clases para C++ y para varias interfaz incluyendo Tcl / Tk, Java y Python. El equipo de Kitware creó y continúa ampliando el conjunto de herramientas, ofrece soporte profesional y servicios de consultoría para VTK. El paquete VTK incluye una amplia variedad de algoritmos de visualización incluyendo: escalares, vectores, tensores, textura, métodos volumétricos, y técnicas avanzadas de modelado, tales como: modelado implícito, reducción de polígonos, suavizado de malla, corte, contorno, y la triangulación de Delaunay. VTK dispone de un marco amplio de visualización de información, tiene un conjunto de widgets de interacción 3D (STL, OBJ y PLY, cuyos formatos son de los más utilizados por plataformas que diseñan modelos CAD), admite el procesamiento paralelo, y se integra con varias bases de datos de herramientas GUI como Qt y Tk. VTK es multiplataforma y funciona en Linux, Windows, Mac y Unix (Kitware, Inc).

Encontrar información acerca de cómo trabajar en VTK puede ser un poco complicado cuando se empieza a conocer estas librerías, el mejor lugar donde se pueden observar ejemplos, resolver dudas y discutir problemas con otros programadores es precisamente en la página oficial y en los libros: The Visualization Toolkit User's Guide y The Visualization Toolkit (Kitware, Inc)(Schroeder, Martin, & Lorensen, The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, 1997)(Schroeder, Avila, Martin, Hoffman, & Law, 2001).

A2. Instalar Visual Toolkit

Como ya se comentaba no es sencillo encontrar información acerca de las librerías de Visual Tool Kit, la instalación de las mismas tampoco es cosa trivial, en especial para aquellos cuyo fuerte no es la instalación de software, con el propósito de facilitar este primer paso en el uso de estas librerías se detallará a continuación paso a paso como se instalaron las librerías para el compilador Visual Studio 2008, Visual Studio 2010 y para las versiones Express de las mismas estas últimas son gratuitas para estudiantes. Las versiones mencionadas a continuación pueden no funcionar dependiendo de la versión del compilador, librerías y/o Windows con que se cuente, en caso de que no funcione descargue otra versión distinta e intente nuevamente la instalación, si las versiones de VTK, CMAKE o TCL mencionadas ya no están disponibles, descargue las nuevas versiones de las mismas eso debería funcionar.

Paso 1
Descargue la versión de TCL (8.5) de la página oficial http://www.tcl.tk/ e instálelo.

Tcl (Tool Command Language) es un lenguaje de programación dinámica muy potente, pero fácil de aprender, ideal para una amplia gama de aplicaciones, incluyendo aplicaciones web y de escritorio, redes, administración de pruebas, y muchos más. De código abierto y favorable a los negocios, Tcl es un lenguaje maduro aún en evolución, que es realmente multiplataforma, fácil de implementar y altamente extensible.

Tk es un conjunto de herramientas de interfaz de usuario gráfica que lleva desarrollar aplicaciones de escritorio a un nivel más alto que los enfoques convencionales. Tk es la interfaz gráfica de usuario estándar no sólo para Tcl, pero para muchos otros lenguajes dinámicos, y pueden producir aplicaciones ricas, nativas que se ejecutan sin cambios a través de Windows, Mac OS X, Linux y más (ActiveState - The Dynamic Language).

Paso 2
Descargue la versión de CMAKE (2.8.2) de la página oficial http://www.cmake.org/ e instálelo.

CMake es un sistema de fuente abierto para "compilar" código para múltiples plataformas (open source multi-platform build system). CMake es una familia de herramientas diseñadas para crear, probar y empaquetar software. CMake se usa para controlar el proceso de compilación del software usando una plataforma sencilla y archivos de configuración independientes del compilador. CMake genera archivos "makefile" nativos y espacios de trabajo que se pueden utilizar en el entorno del compilador de su elección (Kitware, Inc).

Paso 3
Descargue e instale el Visual Studio 2008 Express de la página oficial http://www.microsoft.com/express, e instálelo.

Paso 4
Descargue la versión de VTK (5.6.1) de la página oficial http://www.vtk.org/ e instálela, este paso debe generar la carpeta C:\Program Files\VTK 5.6\bin, después de instalar VTK no quiere decir que ya podemos utilizar las librerías para el compilador de nuestra preferencia, para ello siga con los siguientes pasos.

Paso 5
Ahora vaya a la carpeta donde desea instalar VTK para el compilador de Visual Studio en este caso será en la unidad C:, y ahí haga una carpeta que puede nombrar como usted lo desee, aquí la llamaremos VTKCMAKE de manera que quedará C:\VTKCMAKE.

Paso 6
Ahora vaya a la carpeta de descargas y copie la carpeta que descargó Vtk-5.6.1, pegue esta carpeta en C:\VTKCMAKE, en seguida quite del nombre de la carpeta que acaba de pegar todos los puntos y guiones de manera que quede C:\VTKCMAKE\Vtk561.

Paso 7

Dentro de la carpeta C:\VTKCMAKE genere dos carpetas nuevas que pueden ser llamadas como desee aquí se nombraran C:\VTKCMAKE\Vtkbin y C:\VTKCMAKE\Vtkinstal.

Paso 8

Inicie la aplicación CMAKE que ya fue previamente instalada.

a) Vaya a File y seleccione Delete Cache.
b) En Browse Source seleccione la dirección C:\VTKCMAKE\Vtk561.
c) En Browse Build seleccione la dirección C:\VTKCMAKE\Vtkbin.
d) En la parte inferior seleccione Configure.
e) Cuando aparesca la pantalla con los diferentes compiladores seleccione Visual 2008.
f) Seleccione Finish.
g) En donde dice CMake Install_Prefix cambie la dirección por C:\VTKCMAKE\Vtkinstal.
h) En los recuadros seleccione BuildSharedLibs, BuildExamples y Vtk_Use_Paralel.
i) De clic en el botón Configurar.
j) De clic en el botón Generar.

Paso 9

Llegado este punto si CMAKE marca algún error puede ser porque la versión de VTK que descargo no es compatible con la versión del CMAKE o con el Visual Studio en caso de que las versiones con las que cuenta difieran de las mencionadas en este texto, de ser así repita los pasos 1-8 cambiando la versión de VTK o del CMAKE hasta que obtenga resultados positivos.

Paso 10

Vaya a la carpeta C:\VTKCMAKE\Vtkbin y de doble clic al proyecto de Visual Studio 2008 que se generó.

Paso 11

Ahora en el compilador de Visual Studio seleccione el proyecto llamado ALL_BUILD con el botón derecho del mouse seleccione la opción Build, esto puede durar alrededor de 30 min, cuando termine el compilador todos los archivos deben haberse compilado exitosamente de no ser así las razones y la solución se explican en el Paso 9.

Paso 12

Una vez que el compilador ha terminado, ahí mismo vaya al proyecto llamado Install con el botón derecho del mouse seleccione la opción Build este paso puede durar alrededor de 1 hora, cuando termine el compilador todos los archivos deben haberse compilado exitosamente de no ser así las razones y la solución se explican en el Paso 9.

Paso 13

Ahora en su equipo necesita ir a Panel de Control, seleccione Sistema y Seguridad, entonces de clic en Sistema, también puede llegar a esta ventana desde Inicio y de clic con el botón derecho del mouse sobre Equipo ó Mi PC, una vez que ha llegado aquí siga los siguientes pasos.

a) Seleccione Configuración Avanzada del Sistema.
b) De clic sobre el botón que dice Variables de Entorno.

c) En el recuadro que dice Variables del Sistema de doble clic en la variable llamada Path.
d) En donde dice Valor de la Variable es importante que no borre nada de lo que ya está escrito ahí, agregue solo hasta el final y separe con ";" ";C:\VTKCMAKE\vtkinstal\bin;C:\Program Files\VTK 5.6\bin".
e) Seleccione Aceptar para todas las ventanas.

Paso 14

Para corroborar que se ha instalado correctamente las librerías de VTK vaya a la carpeta C:\VTKCMAKE\ Vtk561\Examples\Rendering\Tcl y abra el archivo Cylinder si funciona bien entonces ha instalado todo exitosamente.

Paso 15

Se puede comenzar a ver ejemplos o a programar si se va a la carpeta C:\VTKCMAKE\vtkbin\Examples y le da doble clic al proyecto de Visual Studio que ahí se generó, Ahora en Visual Studio seleccione con el botón derecho del mouse alguno de los proyectos e indique la opción Set As StartUp Proyect, Ya puede seleccionar entonces Start Debugging.

A3. Generar un Proyecto de VTK una Vez que ya se Instaló

En los siguientes pasos se explicará cómo crear un proyecto en Visual Studio 2008 después de que ha instalado correctamente las librerías de VTK.

Paso 1

Abra el compilador de Visual Studio 2008 y genere un nuevo proyecto:

a) Vaya a File.
b) Seleccione New Proyect.
c) En donde diga Proyect types seleccione Win32
d) En donde diga Templates seleccione Win32 Console Aplication.
e) En Name indique el nombre del proyecto.
f) En la ventana emergente de clic en Finish.

Paso 2

Ahora agregue las librerías a su nuevo proyecto:

a) Vaya a Project.
b) Seleccione NameProyect Properties.
c) En la ventana emergente seleccione Configuration Properties.
d) Seleccione C/C++.
e) Ahí seleccione General.
f) En Additional Include Directories agregue la dirección C:\VTKCMAKE\vtkinstal\include\vtk-5.6.
g) Seleccione Linker.
h) Ahí seleccione General.
i) En Additional Library Directories agregue la dirección C:\VTKCMAKE\vtkinstal\lib\vtk-5.6.

j) Ahora en Linker seleccione Input.
k) En Additional Dependencies agregue: vtkRendering.lib, vtkIO.lib, vtkGraphics.lib, vtkImaging.lib, vtkFiltering.lib, vtkCommon.lib, vtksys.lib.

Paso 3

Ahora ya se pueden probar en el proyecto los ejemplos que están en la carpeta C:\VTKCMAKE\vtkbin\Examples o de la página http://www.vtk.org/Wiki/VTK/Examples/Cxx.

APÉNDICE B

Optimizador

Para el proceso de optimización con el Downhill Simplex Method, se muestra a continuación el código utilizado, el cual está basado en el algoritmo amiba de Numerical Recipes(Press, Teukolsky, Vetterling, & Flannery, 1992).

El algoritmo principal para el proceso de optimización es Optimize, en el cual se manda llamar Optitry que encuentra y evalúa un nuevo punto de prueba, tanto Optimize como Optitry mandan llamar a Optifun el cual es la función de error que es donde se realiza la comparación entre la imagen real y la imagen teórica

B1. Optimize

```
//=============================================================================================
// OPTIMIZE
// Optimize: un programa que optimiza problemas multidimensionales,
//           usando el Downhill Simplex Method [Nelder and Mead]
//           (tambien vea Amoebe program en "Numerical Recipes");
// Usa llamadas a : Optitry; Optifun;
//
// Recibe:  n = numero de variables a optimizar
//                    nmax = numero maximo de iteraciones
//                    ftol = tolerancia de la funcion
//     xinit = valores iniciales de las variables [x1, ... , xn]   (1 fila , n columnas)
//                    eps = valores de las variaciones en el simplex [eps1, ... , epsn]
//
// Regresa: El numero de iteraciones o evaluaciones, neval (integer)
//                    Actualiza el valor de xfin = [1, ... ,n] y del apuntador *yfin
//=============================================================================================
int Optimize(float **xinit, int n, float **eps, float ftol, int nmax, float **xfin, float
*yfin)
{

int np,result,i,j,neval;
int ihi,inhi,ilo;
float auxhi,rtol,Ytry,Ytrytmp,Ysave;
float **P=NULL;
float **P_Opti=NULL;
float **psum;
float **Y;

//---------------------------------------------------------------------------------------
// Inicia el problema de Optimización: inicialización
// crea n+1 puntos iniciales P(i)
// np=n+1;
//---------------------------------------------------------------------------------------
np=n+1;
neval=np;


//---------------------------------------------------------------------------------------
// Matrices a Utilizar
//---------------------------------------------------------------------------------------
```

```
P=matrix(1,(n+1),1,(n));              // Matriz de Puntos del Simplex
P_Opti=matrix(1,1,1,(n));             // Fila de P, punto que se envía a Optifun
psum=matrix(1,1,1,n);                 // Matriz de la suma de las filas de P
Y=matrix(1,np,1,1);                   // Vector que contiene los errores despues de evaluar


for(i=1;i<=n+1;i++)
{
        for(j=1;j<=n;j++)
        {
                P[i][j]=xinit[1][j];

                if((i-1)==j)
                        P[i][j]=P[i][j]+eps[1][j];
        }
}

//-------------------------------------------------------------------------------------------
//psum=get_psum(P,n);         Se suman las columnas de P (n=2)
//for i=1:n+1
//   psum=psum+P(i,:);
//end;
//-------------------------------------------------------------------------------------------

for(i=1;i<=n;i++)
{
        for(j=1;j<=n;j++)
        {
                if(i==1)
                        psum[1][j]=P[i][j];
                psum[1][j]=psum[1][j]+P[i+1][j];
        }
}



//-------------------------------------------------------------------------------------------
// Inicializa Y
//
// for i=1:np                  // (np=n+1)
//    y(i)=optifun(P(i,:),n); // Optifun Regressa un error en cada Y, que corresponde a
// end;
//
// evalua la funcion en cada punto del simplex
//-------------------------------------------------------------------------------------------

for(i=1;i<=np;i++)
{
        for(j=1;j<=n;j++)
        {
                P_Opti[1][j]=P[i][j];
        }
        Y[i][1]=Optifun(P_Opti,n);
}


//-------------------------------------------------------------------------------------------
// Ciclo while
//-------------------------------------------------------------------------------------------

while(1)
{
        //-----------------------------------------------------------------------------
        // Inicializa las ubicaciones: lowest, highest y next highest de los puntos de P
        // Equivalentes al valor de Y mas bajo y los dos mas altos
        // ilo=1;    ihi=1;    inhi=2;
        // auxhi = una variable auxiliar para invertir los valores de otras dos (swap)
        //
        // if y(1)<y(2) [ihi,inhi]=swap(ihi,inhi); end;
        //-----------------------------------------------------------------------------
        ilo=1;
```

```
ihi=1;
inhi=2;

if(Y[1][1]<Y[2][1]) // swap
{
        auxhi=ihi;
        ihi=inhi;
        inhi=auxhi;
}
//------------------------------------------------------------------------------
//  Proceso Para encontrar la Y mayor, menor y antemayor
//------------------------------------------------------------------------------
for(i=1;i<=np;i++)
{
        if(Y[i][1]<=Y[ilo][1])
                ilo=i;
        if(Y[i][1]>Y[ihi][1])
        {
                inhi=ihi;
                ihi=i;
        }
        else
        {
                if(Y[i][1]>Y[inhi][1] && i!=ihi)
                        inhi=i;
        }
}

//------------------------------------------------------------------------------
// Se calcula el valor de rtol
// rtol=2*abs(y(ihi)-y(ilo))/(abs(y(ihi))+abs(y(ilo)));
//------------------------------------------------------------------------------
rtol=2*abs(Y[ihi][1]-Y[ilo][1])/(abs(Y[ihi][1])+abs(Y[ilo][1]));
//------------------------------------------------------------------------------
// Determina si se ha alcanzado el objetivo, comparando el error con la tolerancia
// establecida desde el programa principal
//------------------------------------------------------------------------------
if(rtol<ftol)
{
        auxhi=Y[1][1];
        Y[1][1]=Y[ilo][1];
        Y[ilo][1]=auxhi;

        for(j=1;j<=n;j++)
        {
                auxhi=P[1][j];
                P[1][j]=P[ilo][j];
                P[ilo][j]=auxhi;
        }
        printf("\nEnd of optimization after %d function evaluations",neval);
        printf("\nResulting %f",Y[1][1]);


        *yfin=Y[1][1];
        for(j=1;j<=n;j++)
        {
                xfin[1][j]=P[1][j];
        }
        result=1;
        return(neval); // Termina el programa y regresa el numero de evaluaciones
        break;
        printf("\nError Ooops...program continues after termination");
}



//------------------------------------------------------------------------------
// Si el numero de evaluaciones neval supera a nmax el proceso termina.
//------------------------------------------------------------------------------
if(neval>nmax)
{
```

```
                *yfin=Y[ilo][1];
                for(j=1;j<=n;j++)
                {
                        xfin[1][j]=P[ilo][j];
                }
                result=-1;
                printf("\nToo many iteractions: %d",nmax);
                return(neval);
                break;
                printf("\nError Ooops...program continues after termination");
        }
        //-------------------------------------------------------------------------------
        // Se realiza la reflexion del punto mas alto es decir el peor punto.
        //-------------------------------------------------------------------------------
        neval=neval+2;
        Ytry=Optitry(P,Y,psum,n,ihi,-1.0);
        //-------------------------------------------------------------------------------
        // Se realiza una expansión.
        //-------------------------------------------------------------------------------
        if(Ytry<=Y[ilo][1])
        {
                Ytrytmp=Ytry;
                Ytry=Optitry(P,Y,psum,n,ihi,2.0);
        }
        else if(Ytry >= Y[inhi][1])
        {
                //-------------------------------------------------------------------------------
                // Se realiza una contracción.
                //-------------------------------------------------------------------------------
                Ysave=Y[ihi][1];
                Ytry=Optitry(P,Y,psum,n,ihi,0.5);
                //-------------------------------------------------------------------------------
                // Se realiza la multicontracción alrededor del punto mas bajo.
                //-------------------------------------------------------------------------------
                if(Ytry>=Ysave)
                {
                        for(i=1;i<=np;i++)
                        {
                                if(i!=ilo)
                                {
                                        for(j=1;j<=n;j++)
                                        {
                                                P[i][j]=0.5*(P[i][j]+P[ilo][j]);
                                                P_Opti[1][j]=P[i][j];
                                        }
                                        Y[i][1]=Optifun(P_Opti,n);
                                }
                        }
                        neval=neval+n;
                        for(i=1;i<=n;i++)
                        {
                                for(j=1;j<=n;j++)
                                {
                                        if(i==1)
                                                psum[1][j]=P[i][j];
                                        psum[1][j]=psum[1][j]+P[i+1][j];
                                }
                        }
                }
        }
        else
        {
                neval=neval-1;
        }
} // fin ciclo while

} // fin Optimize
```

B2. Optitry

```
//=============================================================================================
// OPTITRY
//
// Optitry: parte de "Optimize",encuentra y evalua un nuevo punto de prueba, llama a Optifun
//
// Recibe:  n = numero de variables
//                    ihi = ihighest, posicion del valor mas alto de la ultima evaluacion
//                    fac = factor
// P = matriz de n+1 filas y n columnas, contiene informacion de las variables
// Y = matriz de n filas y 1 columna, que contiene las evaluaciones anteriores de Optifun
// psum = matriz de 1 fila y n columnas [1, ... , n]
//
// Regresa: Ytry, y actualiza los valores de las matrices P, Y y psum
//=============================================================================================
float Optitry(float **P, float **Y, float **psum, int n, int ihi, float fac)
{
        int i,j;
        float fac1,fac2;
        float **Ptry=NULL;
        float Ytry;
        i=0;
        j=0;

        Ptry=matrix(1,1,1,n);
        fac1=(1.0-fac)/n;
        fac2=fac1-fac;
        //-------------------------------------------------------------------------
        // Genera el punto de reflexion, expansión o contracción según sea el caso.
        //-------------------------------------------------------------------------
        for(j=1;j<=n;j++)
        {
                Ptry[1][j]=psum[1][j]*fac1-P[ihi][j]*fac2;
        }
        //-------------------------------------------------------------------------
        // Evalua la funcion en el punto de reflexion, expansión o contracción.
        //-------------------------------------------------------------------------
        Ytry=Optifun(Ptry,n);

        //-------------------------------------------------------------------------
        // Si la función es menor que la función evaluada en el peor punto,
        // entonces substituye el peor punto por el nuevo punto generado
        // y actualiza el valor de la suma de los N+1 puntos: psum
        //-------------------------------------------------------------------------
        if(Ytry<Y[ihi][1])
        {
                Y[ihi][1]=Ytry;
                for(j=1;j<=n;j++)
                {
                        psum[1][j]=psum[1][j]+Ptry[1][j]-P[ihi][j];
                        P[ihi][j]=Ptry[1][j];
                }
        }

        return(Ytry);

}
```

B3. Optifun

```
//=============================================================================================
// Ing. Omar Daniel Rodríguez Gutiérrez
// UASLP CIEP
//
// OPTIFUN
// Optifun: Una funcion a ser minimizada por "Optimize"
// llamada por "Optitry" y "Optimize"
//
// Variables Globales: Problem, que determina el case a utilizar en optifun
//             iter, numero de iteraciones
//             config, intensidad del imagecirc E, y/ó configuracion de Optifun
//             Opti_Image, estructura que contiene a la imagen Real Ydata y un Testim
//
// Recibe:  n = numero de variables
//                    X = una fila de la matriz P con n columnas, X = [1, ... , n]
//                    Ydata = Matriz donde se aloja la imagen que se esta optimizando
//                    problem = Caso de optimizacion que se esta utilizando
//                    config =
// Regresa: un error despues de cada evaluacion.
//=============================================================================================
float Optifun(float **X, int n)
{
        //-------------------------------------------------------------------------------------
        // Variables Globales: Problem config iter Ydata testim
        //-------------------------------------------------------------------------------------
        int i,j,k,h,g,f;
        float radius,ThetZ,sumsum=0.0;
        float error=0.0;
        float p0[3]={0,0,0};
        float LocThetX,LocThetY,LocThetZ;
        float FocPosX,FocPosY,FocPosZ;
        i=0;
        j=0;
        k=0;

        //-------------------------------------------------------------------------------------
        // Problem=0
        // Programa que identifica un objeto arbitrario en 2D mediante un
        // circulo gaussiano.
        //
        // Problem=1
        // Programa que identifica un objeto 3D en una imagen utilizando una
        // silueta y un procedimiento de escaneo.
        //
        // Problem=2
        // Programa que identifica un objeto 3D en una imagen utilizando una
        // silueta y un procedimiento completo de optimizacion.
        //
        // Problem=3
        // Programa que realiza la calibración de camara para la identificación
        // de objetos en 3D.
        //
        // Problem=4
        // Programa que realiza la calibración de camara para la identificación
        // de objetos en 3D Utilizando un Chessboard (Tablero de Ajedrez).
        //
        // Problem=5
        // Problem=6
        // Programa que realiza la calibración de camara para la identificación
        // de objetos en 3D Utilizando un Doble Chessboard (Tablero de Ajedrez).
        //-------------------------------------------------------------------------------------
         switch(Problem)
         {

        //=====================================================================
        // Identifica Un Objeto en 3D Utilizando una silueta y un Proceso
        // de Optimización
        //=====================================================================
```

```
        case 2:
        //-----------------------------------------------------
        //    iter=iter+1;
        //-----------------------------------------------------
        Opti_Image[Problem].iter=Opti_Image[Problem].iter+1;

        //-----------------------------------------------------
        // si config==2
        // Entonces el proceso maneja solo 2 variables
        // independientes (translación en X y Y)
        // y el resto quedan fijas
        //-----------------------------------------------------
        if (Opti_Image[Problem].config==2)
        {
               ThetZ=0;

               p0[0]=X[1][1];
               p0[1]=X[1][2];

               LocThetX=0;
               LocThetY=0;
               LocThetZ=0;
        }
        else if (Opti_Image[Problem].config==4)
        {
        //-----------------------------------------------------
        // si config==4
        // Entonces el proceso maneja 3 variables
        // independientes (translación en X, Y y un angulo de
        // rotación en Z) y el resto quedan fijas
        //-----------------------------------------------------

        //-----------------------------------------------------
        // Considerando el Centro de gravedad para la rotacion
        //-----------------------------------------------------
        h=BestComp[0].NumObject;
        g=BestComp[0].NumConf;
        f=BestComp[0].NumRot;
        ThetZ=X[1][3];

       p0[0] = X[1][1] - (Object[h].Configuracion[g].Hipot) * cos( 3.14159265 * ( ThetZ +
(Object[h].Configuracion[g].Fhi) ) / 180 );

       p0[1] = X[1][2] - (Object[h].Configuracion[g].Hipot) * sin( 3.14159265 * ( ThetZ +
(Object[h].Configuracion[g].Fhi) ) / 180 );

       LocThetX=0;
       LocThetY=0;
       LocThetZ=0;

       }
       else if (Opti_Image[Problem].config==8)
       {
       //-----------------------------------------------------
       // si config==4
       // Entonces el proceso maneja 6 variables
       // independientes (translación en X, Y, un angulo de
       // rotación en Z y cualquier angulo de rotacion adicional)
       //-----------------------------------------------------
       //-----------------------------------------------
       // Considerando el Centro de gravedad para la rotacion
       //-----------------------------------------------
       h=BestComp[0].NumObject;
       g=BestComp[0].NumConf;
       f=BestComp[0].NumRot;
       ThetZ=X[1][3];

       p0[0] = X[1][1] - (Object[h].Configuracion[g].Hipot) * cos( 3.14159265 * ( ThetZ +
(Object[h].Configuracion[g].Fhi) ) / 180 );
```

```
	p0[1] = X[1][2] - (Object[h].Configuracion[g].Hipot) * sin( 3.14159265 * ( ThetZ +
(Object[h].Configuracion[g].Fhi) ) / 180 );

	LocThetX=X[1][4];
	LocThetY=X[1][5];
	LocThetZ=X[1][6];
	}

	//-------------------------------------------------------
	// Manda Llamar la Funcion ImageVTKTeoricaSilueta
	// y se aloja la imagen teorica en Testim2
	//-------------------------------------------------------
	ImageVTKTeoricaSilueta(p0[0],p0[1],ThetZ,LocThetX,LocThetY,LocThetZ,Opti_Image[Probl
em].Testim2);

	//-------------------------------------------------------
	// Se Normaliza el objeto gaussiano que esta en la matriz Testim2
	//
	// Testim2=Testim2/sqrt(sum(sum(Testim2)));
	//-------------------------------------------------------
	NormalizaMatriz(Opti_Image[Problem].Testim2,&(Opti_Image[Problem].sumsum2));

	//-------------------------------------------------------
	// Una vez Normalizado el objeto gaussiano, se multiplica por Ydata normalizada
	// para obtener el error (se alojara en Testim3), como se muestra en el siguiente
	// ejemplo de Matlab:
	//
	// error=1-sum(sum(Testim2.*Testim1));
	//-------------------------------------------------------
	sumsum                                                                          =
ProductoPuntoMatriz(Opti_Image[Problem].Testim1,Opti_Image[Problem].Testim2,Opti_Image[Probl
em].Testim3);

	error=1-sumsum;

	//-----------------------------------------------------------
	// Una vez Que se ha creado La matriz se pasan los valores al apuntador de OpenCV
	// solo si desea visualizarlo en pantalla.
	// Si desea determinado color seleccione en el ciclo for
	// for(j=0; (Azul)      for(j=1;  (Verde)    for(j=2;  (Rojo)
	// y una conbinacion de Verde y Rojo = Amarillo
	//-----------------------------------------------------------
	for(i=0;i<dimy;i++)
	{
		for(j=0;j<dimx_3;j=j+3)
		{

			Opti_Image[Problem].TestimOpencv->imageData[              j+(i*dimx_3)
]=Opti_Image[Problem].Ydata->imageData[  j+(i*dimx_3)  ];

			Opti_Image[Problem].TestimOpencv->imageData[            j+1+(i*dimx_3)
]=Opti_Image[Problem].Ydata->imageData[  j+1+(i*dimx_3)  ];

			Opti_Image[Problem].TestimOpencv->imageData[            j+2+(i*dimx_3)
]=Opti_Image[Problem].Ydata->imageData[  j+2+(i*dimx_3)  ];

		if(15*Opti_Image[Problem].Testim2[i][k]>0)
			{
			Opti_Image[Problem].TestimOpencv->imageData[              j+(i*dimx_3)
]=Opti_Image[Problem].Testim2[i][k]*Opti_Image[Problem].sumsum2;

			Opti_Image[Problem].TestimOpencv->imageData[  j+1+(i*dimx_3)  ]=0;

			Opti_Image[Problem].TestimOpencv->imageData[  j+2+(i*dimx_3)  ]=0;

		if(15*Opti_Image[Problem].Testim2[i][k]>127)
			Opti_Image[Problem].TestimOpencv->imageData[  j+(i*dimx_3)  ]=127;
			}

		k=k+1;
		}
```

```
        k=0;
        }

        //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        printf("\nOPTIFUN Object 3D");
        printf("\n  Iter       Xc       Yc       ThetZ      error");
        printf("\n               %d               %.4f               %.4f               %.4f
%.4f",Opti_Image[Problem].iter,X[1][1],X[1][2],ThetZ,error);
        //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        //-----------------------------------------------------
        // Muestra las imagenes
        //-----------------------------------------------------
        cvShowImage( "TestimOpenCV Real+Teorica", Opti_Image[Problem].TestimOpencv );
        //-----------------------------------------------------
        // Espera la tecla Esc o Q para salir cuando se presione
        //-----------------------------------------------------
        cvWaitKey(5);

        break;

} // Fin Case

return(error);

} // Fin Optifun
```

APÉNDICE C

Generación de la Imagen Real

Para realizar la captura de las imágenes necesarias y algunos procesamientos de imagen, se utilizan las librerías de OpenCV versión 2.1.

OpenCV (Open Source Computer Vision) es una librería de funciones de programación para la visión por computadora en tiempo real. Está liberado bajo la licencia BSD, es gratuito para uso académico y comercial. Funciona en C++, C, Python y próximamente en las interfaces de Java que se ejecutan en Windows, Linux, Mac y Android. La biblioteca cuenta con más de 2.500 algoritmos optimizados(OpenCV).

Antes de seguir adelante es importante mencionar que el pixel es la unidad básica de una imagen por lo tanto es primordial saber acceder a ellos, ayuda también a comprender mejor los algoritmos de captura y procesamiento de las imágenes (Igual & Medrano).

Para obtener los datos de píxeles de una imagen, transmitida como BGR (Blue, Green, Red), es muy importante darse cuenta de que la memoria de la matriz se organiza desde la parte superior de la ventana hacia la parte inferior. El origen de la pantalla se encuentra en la esquina superior izquierda. El eje Y aumenta a medida que se desciende en la pantalla. Por lo que el almacenamiento de píxeles es de izquierda a derecha y de arriba hacia abajo, de la forma en que se ilustra en la Figura C.1.

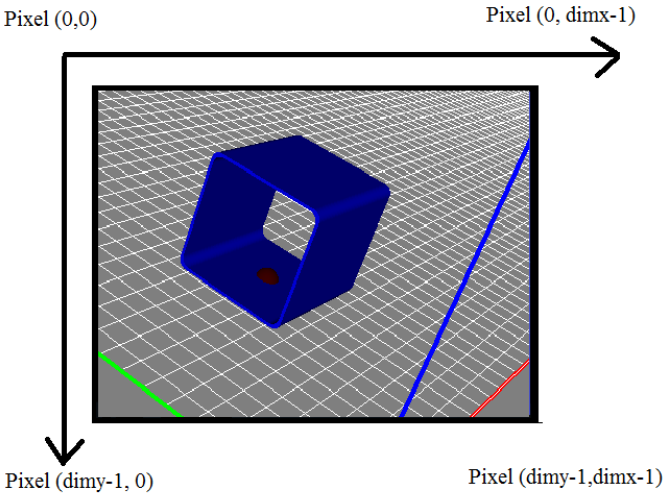


Figura C.1 Acomodo de los pixeles en una imagen.

Primero se crea una estructura de imagen como en la Ecuación C.1:

$$IplImage * img = cvCreateImage\left(cvSize(dimx, dimy), IPL_{DEPTH_{8U}}, canal\right) \qquad \text{(C.1)}$$

Donde img es el apuntador de la estructura de la imagen, dimx es el ancho de la imagen, dimy representa la altura de la imagen y canal es la cantidad de canales que tiene la imagen. De tal forma que si es una imagen en escala de grises entonces solo tiene un canal, pero si es una imagen RGB tendrá entonces tres canales, la forma en que se encuentran acomodados los pixeles y sus respectivos canales de color en el apuntador de la imagen es como se muestra en la Figura C.2.

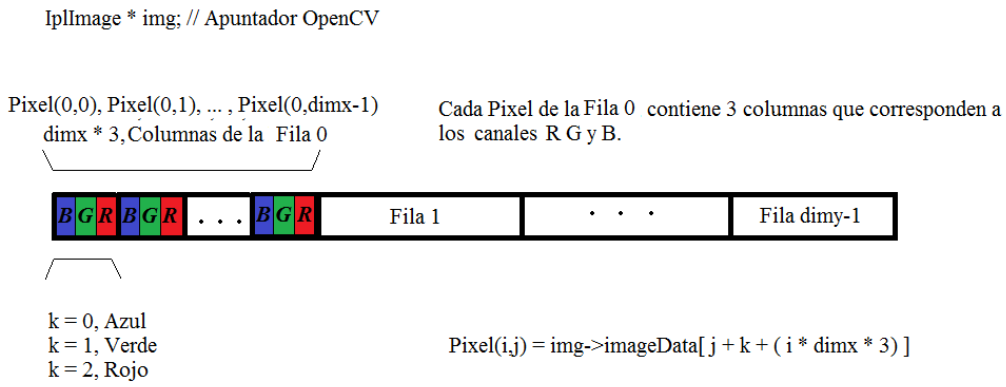


Figura C.2 Acomodo de los pixeles en un apuntador de OpenCV.

Entonces para acceder de la forma más eficiente al apuntador de los pixeles de la estructura de la imagen es como en la Ecuación C.2:

$$Pixel(i,j) = img \rightarrow imageData[\, j + k + (i * dimx * 3)] \qquad \text{(C.2)}$$

En este caso se busca el $Pixel(i,j)$ correspondiente a la imagen, cuando $k = 0$ entonces es el canal azul del $Pixel(i,j)$, si $k = 1$ es el canal verde y si $k = 2$ es el canal rojo.

Para el procedimiento de captura de imagen, lo primero es inicializar la cámara, entonces se captura una imagen y se decide si es la que pertenece al fondo estático o a la imagen con objetos, una vez que se cuenta con las dos imágenes se realiza la resta entre las dos y se obtiene la silueta del objeto deseado, cuando no se cuenta con las condiciones adecuadas de luz o con una cámara de buena calidad entonces ayuda a eliminar el ruido cuando se aplica un filtro basado en un histograma, a la imagen resultante de la resta. Incluso se puede utilizar la función de OpenCV llamada cvSmooth, la cual suaviza la imagen eliminando los pixeles de color que se encuentran en grupos pequeños aislados, aunque si se desea una precisión milimétrica el suavizado de la imagen en los bordes de la silueta puede no ser una propuesta adecuada.

C1. Librerías de OpenCV y Variables Globales de la Captura de Imagen

```
//================================================================================================
// Librerias OpenCV
//================================================================================================
#include <cv.h>
#include <cxcore.h>
#include "highgui.h"

//================================================================================================
// #define
//================================================================================================

#define NumberCam 0           // La camara que se esta utilizando .- 0,1,2,3...
#define NumCanales 3          // Canales Red Green Blue

//------------------------------------------------------------------------------------------------
// struct Histogramas
// Estructura del histograma que se utiliza cuando se procesa
// la imagen capturada con cámara.
//------------------------------------------------------------------------------------------------
struct Histogramas
{
        int MayorInd[5];      // Orden de Mejores Indices de repeticion por imagen
        int MayorJer[5];      // Orden de Mejores Posiciones (de las Divisiones) de 127 - 0
        int Hist[11];         // Indice de repeticion de pixel por imagen de cada division

}Hist[NumCanales],HistMov[NumCanales];

//================================================================================================
// Variables Globales
//================================================================================================

int key;        // Captura una letra del teclado
int Problem;    //  Determina la aplicacion que se esta utilizando (identificación,
                calibración etc.)
int dimx=640;   // Pixeles del eje x de la imagen (Se actualiza automaticamente dependiendo
                de la camara)
int dimy=480;   // Pixeles del eje y de la imagen (Se actualiza automaticamente dependiendo
                de la camara)
int dimn;       // Eje de la imagen mas largo en pixeles
int dimx_3;     // dimx * 3
```

C2. CamCapture

```
//=============================================================================================
// Ing. Omar Daniel Rodríguez Gutiérrez
// UASLP CIEP
//
// CamCapture
// Funcion que Captura imagenes desde una WebCam
//
// La Variable NumCam, indica la camara que se va a utilizar ( 0, 1, 2, ..., n)
// La Variable NumberFrame, indica si va a capturar la imagen del background
// ( 0 ), o alguna subsecuente (1, 2, ..., n) cabe mencionar que si la imagen ( 0 )
// no existe el programa no continuará.
//
// Regresa un 1 Si no se puede inicializar la camara o si se pierde
// la conexion con la misma durante el procedimiento
// Regresa un 0 Cuando hay comunicacion con camara
// Regresa un 2 cuando se quiere tomar una foto sin antes haber tomado
// o tener en archivo la foto numero 0 : Result = CamCapture( conf, 0 );
//=============================================================================================
int CamCapture(int NumCam, int NumberFrame)
{
	int key=0,i,j,k=0,l=0,m,dimxhalf,dimyhalf,dimxhalf_3;
	float Normal;
	CvCapture *capture = NULL;
	IplImage  *frame = NULL;
	IplImage  *framemov = NULL;
	IplImage  *frameHistSmooth = NULL;
	IplImage  *frametarget = NULL;
	IplImage  *frameNoProcess = NULL;
	IplImage  *frameSmooth = NULL;

	// Variables del Zoom
	int deltaZoom=1, angleZoom=0;
	float mZoom[6], CounterZoom=0;
	double factorZoom;
	CvMat MZoom;
	IplImage  *srcZoom=NULL;

	//----------------------------------------
	// Inicializa la Cámara
	// El indice indica la cámara que se está utilizando 0,1,...
	//----------------------------------------
	capture = cvCaptureFromCAM( NumCam );

	//----------------------------------------
	// Siempre revisa que haya conexión
	//----------------------------------------
	if ( !capture )
	{
		printf( "\nCannot open initialize webcam!" );
		return(1);
	}

	//----------------------------------------
	// Crea una ventana para mostrar lo que ve la cámara
	//----------------------------------------
	cvNamedWindow( "CamView", CV_WINDOW_AUTOSIZE );

	//-------------------------------------------------------------------------------------
	// Obtenemos Las dimensiones de la imagen
	// Dimensiones que seran utilizadas en adelante
	// Puesto que dimx y dimy son variables globales
	//-------------------------------------------------------------------------------------
	frametarget = cvQueryFrame( capture );
	dimx=frametarget->width;
	dimy=frametarget->height;
	dimx_3=dimx*3;

	//-------------------------------------------------------------------------------------
```

```
// Si la Imagen no es Cuadrada, tomamos el lado mas largo
// y lo asignamos a dimn
//-----------------------------------------------------------------------------------
if(dimx > dimy)
	dimn = dimx;
else
	dimn = dimy;

dimxhalf = dimx/2;
dimyhalf = dimy/2;
dimxhalf_3=dimxhalf*3;


//---------------------------------------
// Captura La imagen de la Camara
//---------------------------------------
while( key != 'q' )
{
	// Copia la informacion de la camara al apuntador frame
	frametarget = cvQueryFrame( capture );

	// Siempre revisa que haya conexion con la camara
	if( !frametarget )
	{
		printf( "\nPerdida de Conexión con Webcam!" );
		return(1);
	}

	// Cierra la ventana si el usuario presiona 'q'
	// Operador de OpenCV que espera y detecta cualquier tecla
	key = cvWaitKey( 1 );


	//---------------------------------------
	// Aumenta un Zoom a la imagen para poder
	// observar con mas detalle la posicion
	// del Origen de las Coordenadas Mundiales (FocalPoint)
	//---------------------------------------
	if( key == '+')
	{
		CounterZoom = CounterZoom + 1;

		if(CounterZoom > 0)
		{
			factorZoom = 1.0 / CounterZoom;
		}
		else if(CounterZoom < 0)
		{
			factorZoom = (-1.0)*CounterZoom;
		}
		else
		{
			factorZoom = 1;
		}
	}
	//---------------------------------------
	// Disminuye un Zoom a la imagen para poder
	// observar con mas detalle la posicion
	// del Origen de las Coordenadas Mundiales (FocalPoint)
	//---------------------------------------
	if( key == '-')
	{
		CounterZoom = CounterZoom - 1;

		if(CounterZoom > 0)
		{
			factorZoom = 1.0 / CounterZoom;
		}
		else if(CounterZoom < 0)
		{
			factorZoom = (-1.0)*CounterZoom;
```

```
                }
                else
                {
                        factorZoom = 1;
                }

        }
        //----------------------------------------
        // Aplica el Zoom a la imagen
        //----------------------------------------
        if(  key != 'q' && key != 'Q' && CounterZoom != 0)
        {
                srcZoom = NULL;
                srcZoom = cvCloneImage( frametarget );
                MZoom = cvMat( 2, 3, CV_32F, mZoom );
                mZoom[0] = (float)(factorZoom*cos(-angleZoom*2*CV_PI/180.));
                mZoom[1] = (float)(factorZoom*sin(-angleZoom*2*CV_PI/180.));
                mZoom[2] = dimx*0.5f;
                mZoom[3] = -mZoom[1];
                mZoom[4] = mZoom[0];
                mZoom[5] = dimy*0.5f;
                cvGetQuadrangleSubPix( srcZoom, frametarget, &MZoom);
                cvReleaseImage(&srcZoom);

        } // fin del if


        //----------------------------------------
        // Dibuja una Cruz En la parte central
        // de la imagen para ajustar el Focal Point
        // Cuando se Presiona 'q' deja de dibujar
        //----------------------------------------
        if( key != 'q' && key != 'Q' )
        {
                k=0;
                // Dibuja la Cruz en la parte central
                for(i=0;i<dimy;i++)
                {
                        for(j=0;j<dimx_3;j=j+3)
                        {
                                // Dibuja la linea en Y
                                if( i == dimyhalf || i == dimyhalf+1 )
                                {
                                        if( j != dimxhalf_3 && j != (dimxhalf_3)+3 )
                                        {
                                        frametarget->imageData[  j+0+(i*dimx_3)  ]=0;
                                        frametarget->imageData[  j+1+(i*dimx_3)  ]=0;
                                        frametarget->imageData[  j+2+(i*dimx_3)  ]=0;
                                        }
                                }
                                // Dibuja la linea en X
                                if( j == dimxhalf_3 || j == (dimxhalf_3)+3 )
                                {
                                        if( i != dimyhalf && i != dimyhalf+1 )
                                        {
                                        frametarget->imageData[  j+0+(i*dimx_3)  ]=0;
                                        frametarget->imageData[  j+1+(i*dimx_3)  ]=0;
                                        frametarget->imageData[  j+2+(i*dimx_3)  ]=0;
                                        }
                                }
                        }
                        k=k+1;
                }
        } // fin del if



        // Despliega la imagen actual
        cvShowImage( "CamView", frametarget );

} // Fin del While
```

```
// Destruye La Ventana
cvDestroyWindow( "CamView" );

//----------------------------------------
// Genera La Imagen Original cuando
// NumberFrame == 0, es la imagen
// que no contiene ningun Objeto a Identificar
//----------------------------------------
if( NumberFrame == 0)
{
        // Para guardar la imagen RGB tomada se crea memoria en otro apuntador
        frame=cvCloneImage(frametarget);

        cvSaveImage("ImagenDetecMovimiento.jpg",frame );
        cvReleaseCapture( &capture );
        cvReleaseImage(&frame);

        return(0);
}

//----------------------------------------------------------------
// Para guardar la imagen RGB creamos memoria en otro apuntador
//----------------------------------------------------------------
frame=cvCloneImage(frametarget);
//----------------------------------------------------------------
// Se Lee la Imagen Que no contiene ningun Objeto
//----------------------------------------------------------------
framemov=cvLoadImage("ImagenDetecMovimiento.jpg",1);

if( !framemov )
{
        printf( "\nNo se encontro: ImagenDetecMovimiento.jpg\n" );
        printf( "Para generar haga: Result = CamCapture( NumCam, 0 );\n" );
        return(2);
}

//----------------------------------------------------------------
// Inicializa y crea las variables para los Histogramas
//----------------------------------------------------------------
int NumDiv = 10;
int Fraccion;
int Valor;
int DivActual;
int ValorMax = 127;  // Valor Entero Maximo para un pixel tipo char de OpenCV
int ValorMin = 0;

for(i=0;i<3;i++)
{
        for(j=0;j<5;j++)
        {
                Hist[i].MayorInd[j]=0;
                Hist[i].MayorJer[j]=0;

                HistMov[i].MayorInd[j]=0;
                HistMov[i].MayorJer[j]=0;
        }
}
for(i=0;i<3;i++)
{
        for(j=0;j<=NumDiv;j++)
        {
                Hist[i].Hist[j]=0;

                HistMov[i].Hist[j]=0;
        }
}

// Al ser todos los valores enteros el resultado
// de todas las divisiones sera = floor(x/y)
Fraccion = ValorMax / NumDiv ;
//-------------------------------------------------------------------------------
```

```
        // 1.-Se lee la matriz de OpenCV: frame para crear el Histograma
        //
        // HistMov[m].Hist[Div] ,  Donde Div Representa el numero de barras del histograma
        // HistMov[m].Hist[ 0 - 11] ,
        // Va desdela menor intensidad 0 hasta la maxima 11*Fraccion = 127
        // HistMov[m].Hist[Div] = # de pixeles que estan dentro de este intervalo
        //
        // Se trabajara por ahora solo en 3 Canales R G B.
        // j+0 (Azul)      j+1  (Verde)    j+2  (Rojo)
        // y una conbinacion de Verde y Rojo = Amarillo
        //-------------------------------------------------------------------------------
        k=0;
        for(i=0;i<dimy;i++)
        {
                for(j=0;j<dimx_3;j=j+3)
                {
                        //-----------------------------------------------------------------
                        // En OpenCV La max intensidad se representa con -1
                        // Para fines Practicos se le asignara el valor Entero Max
                        // correspondiente
                        //-----------------------------------------------------------------
                        if(frame->imageData[  j+(i*dimx_3)  ] < 0 )
                                frame->imageData[  j+(i*dimx_3)  ]=127;

                        if(frame->imageData[  j+1+(i*dimx_3)  ] < 0 )
                                frame->imageData[  j+1+(i*dimx_3)  ]=127;

                        if(frame->imageData[  j+2+(i*dimx_3)  ] < 0 )
                                frame->imageData[  j+2+(i*dimx_3)  ]=127;

                        //-----------------------------------------------------------------
                        // Detección de Movimiento
                        //-----------------------------------------------------------------
                        // En OpenCV La max intensidad se representa con -1
                        // Para fines Practicos se le asignara el valor Entero Max
                        // correspondiente
                        //-----------------------------------------------------------------
                        if(framemov->imageData[  j+(i*dimx_3)  ] < 0 )
                                framemov->imageData[  j+(i*dimx_3)  ]=127;

                        if(framemov->imageData[  j+1+(i*dimx_3)  ] < 0 )
                                framemov->imageData[  j+1+(i*dimx_3)  ]=127;

                        if(framemov->imageData[  j+2+(i*dimx_3)  ] < 0 )
                                framemov->imageData[  j+2+(i*dimx_3)  ]=127;

                        // Resta de imagenes
                        framemov->imageData[   j+(i*dimx_3)   ] =  abs(  framemov->imageData[
j+(i*dimx_3)  ] - frame->imageData[  j+(i*dimx_3)  ] );
                        framemov->imageData[   j+1+(i*dimx_3)   ] =  abs(  framemov->imageData[
j+1+(i*dimx_3)  ] - frame->imageData[  j+1+(i*dimx_3)  ] );
                        framemov->imageData[   j+2+(i*dimx_3)   ] =  abs(  framemov->imageData[
j+2+(i*dimx_3)  ] - frame->imageData[  j+2+(i*dimx_3)  ] );

                        // Maxma intensidad  de color hacia el canal Azul
if(  framemov->imageData[  j+1+(i*dimx_3)  ] > framemov->imageData[  j+(i*dimx_3)  ] )
                framemov->imageData[  j+(i*dimx_3) ] = framemov->imageData[  j+1+(i*dimx_3)];
if(  framemov->imageData[  j+2+(i*dimx_3)  ] > framemov->imageData[  j+(i*dimx_3)  ] )
        framemov->imageData[  j+(i*dimx_3) ] = framemov->imageData[  j+2+(i*dimx_3)];

                        // m Representa cada canal de color
                        for(m=0;m<3;m++)
                        {
                                Valor = framemov->imageData[  j+m+(i*dimx_3)  ];
                                DivActual = Valor / Fraccion;
                                HistMov[m].Hist[DivActual] = HistMov[m].Hist[DivActual] + 1;
                        }
                k=k+1;
                }
        k=0;
        }
```

```
//-------------------------------------------------------------
// Inicializa Variable para eliminar el ruido con Histograma
// se crea memoria en otro apuntador
//-------------------------------------------------------------
frameHistSmooth=cvCloneImage(framemov);
//-------------------------------------------------------------
// Inicializa Variable para visualizar la resta de imagenes
// se crea memoria en otro apuntador
//-------------------------------------------------------------
frameNoProcess=cvCloneImage(framemov);
//-------------------------------------------------------------
// Inicializa Variable para eliminar el ruido con suavisado
// se crea memoria en otro apuntador
//-------------------------------------------------------------
frameSmooth=cvCloneImage(framemov);

//-------------------------------------------------------------------------------------
// Histogramas
// Ordena las barras Para conocer cual es la que tiene mayor incidencia
// en cada canal de color
//
// m Representa cada canal de color
// HistMov[m].Hist[Div] ,  Donde Div Representa el numero de barras del histograma
// HistMov[m].Hist[ 0 - 11] ,
// Va desdela menor intensidad 0 hasta la maxima 11*Fraccion = 127
// HistMov[m].Hist[Div] = # de pixeles que estan dentro de este intervalo
//
// HistMov[m].MayorInd[#], Guarda El valor de una barra del histograma y
// # va de 0 - 4: donde 0 es la barra mas grande de todas
//
// Nota para entender mejor lo que esta sucediendo, Estudielo para un solo color
//-------------------------------------------------------------------------------------
for(i=0;i<=NumDiv;i++)
{
		//-----------------------------------------------------------------
		// Deteccion de Movimiento
		//-----------------------------------------------------------------
		// m Representa cada canal de color
		for(m=0;m<3;m++)
		{
			if(HistMov[m].Hist[i] >= HistMov[m].MayorInd[0] )
			{
				HistMov[m].MayorInd[4] = HistMov[m].MayorInd[3];
				HistMov[m].MayorInd[3] = HistMov[m].MayorInd[2];
				HistMov[m].MayorInd[2] = HistMov[m].MayorInd[1];
				HistMov[m].MayorInd[1] = HistMov[m].MayorInd[0];
				HistMov[m].MayorInd[0] = HistMov[m].Hist[i];
			}
			else if(HistMov[m].Hist[i] > HistMov[m].MayorInd[1] )
			{
				HistMov[m].MayorInd[4] = HistMov[m].MayorInd[3];
				HistMov[m].MayorInd[3] = HistMov[m].MayorInd[2];
				HistMov[m].MayorInd[2] = HistMov[m].MayorInd[1];
				HistMov[m].MayorInd[1] = HistMov[m].Hist[i];
			}
			else if(HistMov[m].Hist[i] > HistMov[m].MayorInd[2] )
			{
				HistMov[m].MayorInd[4] = HistMov[m].MayorInd[3];
				HistMov[m].MayorInd[3] = HistMov[m].MayorInd[2];
				HistMov[m].MayorInd[2] = HistMov[m].Hist[i];
			}
			else if(HistMov[m].Hist[i] > HistMov[m].MayorInd[3] )
			{
				HistMov[m].MayorInd[4] = HistMov[m].MayorInd[3];
				HistMov[m].MayorInd[3] = HistMov[m].Hist[i];
			}
			else if(HistMov[m].Hist[i] > HistMov[m].MayorInd[4] )
			{
				HistMov[m].MayorInd[4] = HistMov[m].Hist[i];
			}
```

```
                }

}
//-------------------------------------------------------------------------------------
// Ordena las barras de mayor incidencia por jerarquias
// Es decir las que tengan mas intensidad sera la 0 y las
// de menor intensidad sera la 4
//
// m Representa cada canal de color
// HistMov[m].Hist[Div] ,  Donde Div Representa el numero de barras del histograma
// HistMov[m].Hist[ 0 - 11] ,
// Va desdela menor intensidad 0 hasta la maxima 11*Fraccion = 127
// HistMov[m].Hist[Div] = # de pixeles que estan dentro de este intervalo
//
// HistMov[m].MayorJer[#], Guarda El valor de una de las 5 barras mas grandes del
// histograma pero los acomoda en grado de intensidad donde la maxima intensidad
// = 127 y # va de 0 - 4: donde 0 es la barra de mas intensidad
//-------------------------------------------------------------------------------------
for(i=0;i<=NumDiv;i++)
{
        for(j=0;j<5;j++)
        {
                //-----------------------------------------------------------------
                // Deteccion de Movimiento
                //-----------------------------------------------------------------
                // m Representa cada canal de color
                for(m=0;m<3;m++)
                {
                // Primero preguta si la barra pertenece a las de mayor incidencia
                        if( HistMov[m].Hist[i] == HistMov[m].MayorInd[j])
                        {

                                if( i >= HistMov[m].MayorJer[0])
                                {
                                HistMov[m].MayorJer[4] = HistMov[m].MayorJer[3];
                                HistMov[m].MayorJer[3] = HistMov[m].MayorJer[2];
                                HistMov[m].MayorJer[2] = HistMov[m].MayorJer[1];
                                HistMov[m].MayorJer[1] = HistMov[m].MayorJer[0];
                                HistMov[m].MayorJer[0] = i;
                                }
                                else if( i > HistMov[m].MayorJer[1])
                                {
                                HistMov[m].MayorJer[4] = HistMov[m].MayorJer[3];
                                HistMov[m].MayorJer[3] = HistMov[m].MayorJer[2];
                                HistMov[m].MayorJer[2] = HistMov[m].MayorJer[1];
                                HistMov[m].MayorJer[1] = i;
                                }
                                else if( i > HistMov[m].MayorJer[2])
                                {
                                HistMov[m].MayorJer[4] = HistMov[m].MayorJer[3];
                                HistMov[m].MayorJer[3] = HistMov[m].MayorJer[2];
                                HistMov[m].MayorJer[2] = i;
                                }
                                else if( i > HistMov[m].MayorJer[3])
                                {
                                HistMov[m].MayorJer[4] = HistMov[m].MayorJer[3];
                                HistMov[m].MayorJer[3] = i;
                                }
                                else if( i > HistMov[m].MayorJer[4])
                                {
                                        HistMov[m].MayorJer[4] = i;
                                }
                        }
                }
        }
}
//-------------------------------------------------------------------------------------
// Se rellena la imagen con base en los resultados del histograma.
//
// j+0 (Azul)      j+1  (Verde)    j+2  (Rojo)
// y una conbinacion de Verde y Rojo = Amarillo
```

```
//-------------------------------------------------------------------------------------
k=0;
for(i=0;i<dimy;i++)
{
        for(j=0;j<dimx_3;j=j+3)
        {
        //-----------------------------------------------------------------
        // Deteccion de Movimiento
        //-----------------------------------------------------------------
        // Blue Color
        if(framemov->imageData[  j+(i*dimx_3)  ] >= HistMov[0].MayorJer[0]*Fraccion)
        {
                framemov->imageData[  j+(i*dimx_3)  ]=127.0;
        }
else if(framemov->imageData[  j+(i*dimx_3)  ] >= HistMov[0].MayorJer[1]*Fraccion)
        {
                framemov->imageData[  j+(i*dimx_3)  ]=100.0;
        }
else if(framemov->imageData[  j+(i*dimx_3)  ] >= HistMov[0].MayorJer[2]*Fraccion)
        {
                framemov->imageData[  j+(i*dimx_3)  ]=80.0;
        }
else if(framemov->imageData[  j+(i*dimx_3)  ] >= HistMov[0].MayorJer[3]*Fraccion)
        {
                framemov->imageData[  j+(i*dimx_3)  ]=60.0;
        }
else if(framemov->imageData[  j+(i*dimx_3)  ] >= HistMov[0].MayorJer[4]*Fraccion)
        {
                framemov->imageData[  j+(i*dimx_3)  ]=0.0;
        }
        else
        {
                framemov->imageData[  j+(i*dimx_3)  ]=0.0;
        }

        // Green Color + Blue Color
if(framemov->imageData[  j+1+(i*dimx_3)  ] >= HistMov[1].MayorJer[0]*Fraccion)
        {
                framemov->imageData[  j+1+(i*dimx_3)  ]=127.0;
                framemov->imageData[  j+0+(i*dimx_3)  ]=127.0;
        }
else if(framemov->imageData[  j+1+(i*dimx_3)  ] >= HistMov[1].MayorJer[1]*Fraccion)
        {
                framemov->imageData[  j+1+(i*dimx_3)  ]=100.0;
                framemov->imageData[  j+0+(i*dimx_3)  ]=100.0;
        }
else if(framemov->imageData[  j+1+(i*dimx_3)  ] >= HistMov[1].MayorJer[2]*Fraccion)
        {
                framemov->imageData[  j+1+(i*dimx_3)  ]=80.0;
                framemov->imageData[  j+0+(i*dimx_3)  ]=80.0;
        }
else if(framemov->imageData[  j+1+(i*dimx_3)  ] >= HistMov[1].MayorJer[3]*Fraccion)
        {
                framemov->imageData[  j+1+(i*dimx_3)  ]=60.0;
                framemov->imageData[  j+0+(i*dimx_3)  ]=60.0;
        }
else if(frame->imageData[  j+1+(i*dimx_3)  ] >= HistMov[1].MayorJer[4]*Fraccion)
        {
                framemov->imageData[  j+1+(i*dimx_3)  ]=0.0;
                framemov->imageData[  j+0+(i*dimx_3)  ]=0.0;
        }
else
        {
                framemov->imageData[  j+1+(i*dimx_3)  ]=0.0;
        }

        // Red Color + Blue Color
if(framemov->imageData[  j+2+(i*dimx_3)  ] >= HistMov[2].MayorJer[0]*Fraccion)
        {
                framemov->imageData[  j+2+(i*dimx_3)  ]=127.0;
                framemov->imageData[  j+0+(i*dimx_3)  ]=127.0;
```

```
            }
    else if(framemov->imageData[  j+2+(i*dimx_3)  ] >= HistMov[2].MayorJer[1]*Fraccion)
            {
                    framemov->imageData[  j+2+(i*dimx_3)  ]=100.0;
                    framemov->imageData[  j+0+(i*dimx_3)  ]=100.0;
            }
    else if(framemov->imageData[  j+2+(i*dimx_3)  ] >= HistMov[2].MayorJer[2]*Fraccion)
            {
                    framemov->imageData[  j+2+(i*dimx_3)  ]=80.0;
                    framemov->imageData[  j+0+(i*dimx_3)  ]=80.0;
            }
    else if(framemov->imageData[  j+2+(i*dimx_3)  ] >= HistMov[2].MayorJer[3]*Fraccion)
            {
                    framemov->imageData[  j+2+(i*dimx_3)  ]=60.0;
                    framemov->imageData[  j+0+(i*dimx_3)  ]=60.0;
            }
    else if(framemov->imageData[  j+2+(i*dimx_3)  ] >= HistMov[2].MayorJer[4]*Fraccion)
            {
                    framemov->imageData[  j+2+(i*dimx_3)  ]=0.0;
                    framemov->imageData[  j+0+(i*dimx_3)  ]=0.0;
            }
            else
            {
                    framemov->imageData[  j+2+(i*dimx_3)  ]=0.0;
            }

            k=k+1;
            }
    k=0;
    }

    //-----------------------------------------------------------------
    // Suavisa la imagen Procesada con Histograma
    //-----------------------------------------------------------------
    cvSmooth( framemov, frameHistSmooth , CV_GAUSSIAN , 5, 0, 0, 0 );
    //-----------------------------------------------------------------
    // Suavisa la imagen de la Resta que no fue Procesada
    //-----------------------------------------------------------------
    cvSmooth( frameNoProcess, frameSmooth , CV_GAUSSIAN , 5, 0, 0, 0 );
    //-----------------------------------------------------------------
    // Guarda la Imagen acorde ala aplicacion
    //-----------------------------------------------------------------
switch(Problem)
{
           //-------------------------------------------------------------------------------
           // Identifica Objetos 2D con un Circulo Gaussiano
           //-------------------------------------------------------------------------------
    case 0:
                    cvSaveImage("Block01.jpg", frameSmooth);
        break;
           //-------------------------------------------------------------------------------
           // Identifica Un Objeto en 3D Utilizando una silueta y un Scann
           //-------------------------------------------------------------------------------
           case 1:
                    cvSaveImage("ImaRealArtSilueta.jpg", frameSmooth);
                    break;
           //-------------------------------------------------------------------------------
           // Identifica Un Objeto en 3D Utilizando una silueta y un Proceso
           // completo de Optimización
           //-------------------------------------------------------------------------------
           case 2:
                    cvSaveImage("ImaRealArtSilueta.jpg", frameSmooth);
                    break;
           //-------------------------------------------------------------------------------
           // Realiza un Proceso de Calibracion de Camara con Optimizacion y Pilares
           //-------------------------------------------------------------------------------
           case 3:
                    cvSaveImage("ImaCalibracionPilar.jpg", frameSmooth);
                    break;
           //-------------------------------------------------------------------------------
           // Realiza un Proceso de Calibracion de Camara con Optimizacion y ChessBoard
```

```
                //-------------------------------------------------------------------------
                case 4:
                        cvSaveImage("ImaCalibracionChess00.jpg", frameSmooth);
                        break;
                //-------------------------------------------------------------------------
                // Realiza un Proceso de Calibracion de Camara con Optimizacion y Dos
                // ChessBoards
                //
                // ChessBoard A una altura de la variable ChessAlto
                //-------------------------------------------------------------------------
                case 5:
                        cvSaveImage("ImaCalibracionChess01.jpg", frameSmooth);
                        break;
                //-------------------------------------------------------------------------
                // Realiza un Proceso de Calibracion de Camara con Optimizacion Y Dos
                // ChessBoards
                //
                // ChessBoard A la altura del plano del trabajo
                //-------------------------------------------------------------------------
                case 6:
                        cvSaveImage("ImaCalibracionChess00.jpg", frameSmooth);
                        break;
        }   // end switch(Problem)

        //---------------------------------------------------------------------
        // Crea Las Ventanas y les asigna un Nombre
        //---------------------------------------------------------------------
        cvNamedWindow( "Imagen Capturada con Camara RGB", CV_WINDOW_AUTOSIZE );
        cvNamedWindow( "Imagen Mov Procesada con Histograma", CV_WINDOW_AUTOSIZE );
        cvNamedWindow("Imagen Mov Procesada con Histograma y cvSmooth",CV_WINDOW_AUTOSIZE );
        cvNamedWindow( "Imagen Mov Procesada con cvSmooth", CV_WINDOW_AUTOSIZE );
        cvNamedWindow( "Imagen Mov Sin Procesamiento", CV_WINDOW_AUTOSIZE );
        //---------------------------------------------------------------------
        // Muestra las imagenes
        //---------------------------------------------------------------------
        cvShowImage( "Imagen Capturada con Camara RGB", frame );
        cvShowImage( "Imagen Mov Procesada con Histograma", framemov );
        cvShowImage( "Imagen Mov Procesada con Histograma y cvSmooth", frameHistSmooth);
        cvShowImage( "Imagen Mov Procesada con cvSmooth", frameSmooth);
        cvShowImage( "Imagen Mov Sin Procesamiento", frameNoProcess);
        //---------------------------------------------------------------------
        // Espera la tecla Esc o Q para salir cuando se presione
        //---------------------------------------------------------------------
        cvWaitKey(0);
        //---------------------------------------------------------------------
        // Para destruir las ventanas
        //---------------------------------------------------------------------
        cvDestroyWindow( "Imagen Capturada con Camara RGB");
        cvDestroyWindow( "Imagen Mov Procesada con Histograma");
        cvDestroyWindow( "Imagen Mov Procesada con Histograma y cvSmooth");
        cvDestroyWindow( "Imagen Mov Procesada con cvSmooth");
        cvDestroyWindow( "Imagen Mov Sin Procesamiento");
        //---------------------------------------------------------------------
        // free memory
        //---------------------------------------------------------------------
        cvReleaseCapture( &capture );
        //cvReleaseImage( &frametarget ); // Comparte el mismo apuntador que capture
        cvReleaseImage( &frame );
        cvReleaseImage( &framemov );
        cvReleaseImage( &frameHistSmooth );
        cvReleaseImage( &frameSmooth );
        cvReleaseImage( &frameNoProcess );
        return(0);

} // fin de CamCapture
```

APÉNDICE D

Generación de la Imagen Teórica

Para generar la imagen teórica, primero está el módulo llamado GeneradorDeImagenTeoricaSilueta como se ve en la Figura D.1, con el que las librerías de VTK podrán renderizar un ambiente virtual en 3D donde se desarrollará la generación de las imágenes utilizando los modelos CAD disponibles.

Después dentro de la clase MouseInteractorStyle01 en la cual se realiza todo el procedimiento de interacción dentro del ambiente renderizado, se genera una imagen teórica por cada iteración de acuerdo a la suposición de parámetros calculados por el optimizador esto se realiza en la función de error llamada Optifun la cual a su vez llama a ImageVTKTeoricaSilueta. Finalmente la función Screenshot es la encargada de transmitir la información de los pixeles del renderizador, a un apuntador desde la cual se puedan procesar.

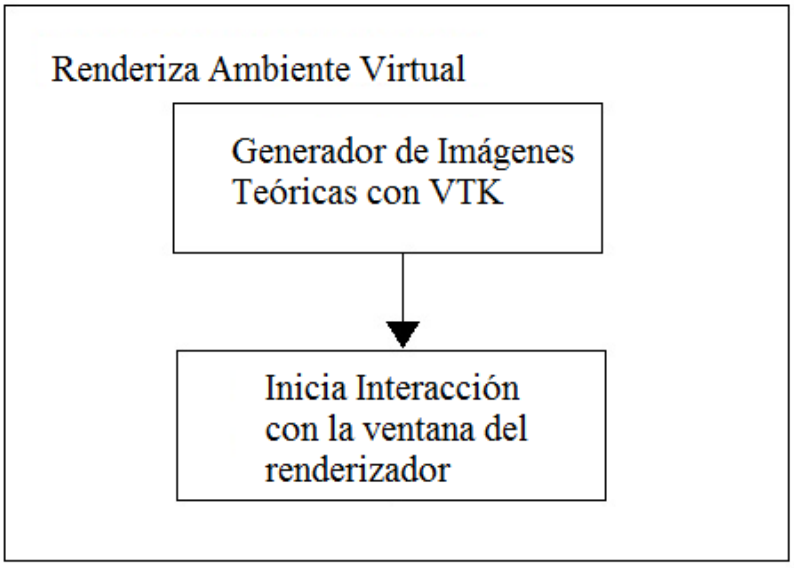


Figura D.1 Módulo de renderizado del ambiente virtual.

D1. Renderiza el Ambiente Virtual

```
//=============================================================================================
// Ing. Omar Daniel Rodríguez Gutiérrez
// UASLP CIEP
//
// Generador de Imagen Teórica
//
// Recibe las variables globales (que son previamente leidas desde
// un archivo de texto) :
//
// W_C_CamFocResult :  Posición del punto focal despues de la calibración
// W_C_CamPosResult :  Posición de la cámara despues de la calibración
// W_C_RollResult     :  Angulo de giro de la cámara despues de la calibración
// W_C_ZoomResult     :  Zoom de la cámara despues de la calibración
//=============================================================================================
void GeneradorDeImagenTeoricaSilueta(void)
{


//---------------------------------------------------------------------------------------------
// Crea la estructura gráfica. Se renderiza dentro de la ventana del render
// El "render window interactor" captura los eventos del mouse
// y desarrolla la apropiada manipulacion de los actores o la cámara
// dependiendo de la naturaleza de los eventos.
//---------------------------------------------------------------------------------------------

  //Visualizacion de la ventana
  vtkSmartPointer<vtkRenderer> ren1 =  vtkSmartPointer<vtkRenderer>::New();
  vtkSmartPointer<vtkRenderWindow> renWin =  vtkSmartPointer<vtkRenderWindow>::New();
  renWin->AddRenderer(ren1);
  vtkSmartPointer<vtkRenderWindowInteractor>                    iren                    =
vtkSmartPointer<vtkRenderWindowInteractor>::New();
  iren->SetRenderWindow(renWin);

//---------------------------------------------------------------------------------------------
// Selecciona el estilo a utilizar durante la interacción.
// Nombre de la clase de la interaccion: MouseInteractorStyle01
//---------------------------------------------------------------------------------------------

  vtkSmartPointer<MouseInteractorStyle01>                    style                    =
vtkSmartPointer<MouseInteractorStyle01>::New();
  style->SetDefaultRenderer(ren1);
  iren->SetInteractorStyle(style);

//---------------------------------------------------------------------------------------------
// Agrega los actores al renderer, Selecciona el color del fondo
// y el tamaño de la ventana.
//---------------------------------------------------------------------------------------------

  ren1->SetBackground(0,0,0);
  renWin->SetSize(dimx, dimy);

//---------------------------------------------------------------------------------------------
// Selecciona las propiedades de la iluminación
//---------------------------------------------------------------------------------------------


  vtkSmartPointer<vtkLight> light = vtkSmartPointer<vtkLight>::New();
  light->SetFocalPoint(W_C_CamFocResult);
  light->SetPosition(W_C_CamPosResult);
  light->SetLightTypeToHeadlight(); // Esta Fuente de luz Sigue la posicion de la cámara
  light->SetIntensity(5);
  ren1->AddLight(light);

  vtkSmartPointer<vtkLight> light01 = vtkSmartPointer<vtkLight>::New();
  light01->SetFocalPoint(W_C_CamFocResult);
  light01->SetPosition(W_C_CamPosResult[0]  +  LightSep,  W_C_CamPosResult[1]  +  LightSep,
W_C_CamPosResult[2] );
  light01->SetIntensity(5);
```

```
  ren1->AddLight(light01);

  vtkSmartPointer<vtkLight> light02 = vtkSmartPointer<vtkLight>::New();
  light02->SetFocalPoint(W_C_CamFocResult);
  light02->SetPosition(  W_C_CamPosResult[0]  -  LightSep,  W_C_CamPosResult[1]  -  LightSep,
W_C_CamPosResult[2] );
  light02->SetIntensity(5);
  ren1->AddLight(light02);



//-----------------------------------------------------------------------------------------
// Selecciona la posición y las propiedades de la cámara
// así como la posicion del "Punto Focal" que es el punto
// a donde esta dirigido el centro de la cámara
//-----------------------------------------------------------------------------------------

  ren1->GetActiveCamera()->SetFocalPoint(W_C_CamFocResult);
  ren1->GetActiveCamera()->SetPosition(W_C_CamPosResult);
  ren1->GetActiveCamera()->SetViewUp(0,0,1);
  ren1->GetActiveCamera()->SetRoll(W_C_RollResult);
  ren1->GetActiveCamera()->Zoom(W_C_ZoomResult);

//-----------------------------------------------------------------------------------------
// Inicia el ciclo de los eventos e invoca un render inicial.
//-----------------------------------------------------------------------------------------

  std::cout << "Presione el boton derecho del Mouse sobre la pantalla para iniciar" <<
std::endl;
  std::cout << "Presione 'Q' o 'q' en la Pantalla para Finalizar" << std::endl;

  iren->Initialize();
  renWin->Render();
  iren->Start();

  // Cierra la ventana despues de que se ha presionado 'Q' o 'q' (quit)
  iren->GetRenderWindow()->Finalize();

} // fin Generador imagen teórica
```

D2. Estructura de la Clase para la Interacción con el Ambiente Virtual

```
//=========================================================================================
//                          Clase Para La Interaccion con el Mouse
//                                    MouseInteractorStyle01
//=========================================================================================

// Catch mouse events
class MouseInteractorStyle01 : public vtkInteractorStyleTrackballCamera
{
  public:
  static MouseInteractorStyle01* New();

  //-------------------------------------------------------------------------------------
  // OnRightButtonDown():
  // Funcion que se inicia al presionar el boton derecho del mouse, puesto que la clase
  // MouseInteractorStyle01 hereda las prop. de la clase vtkInteractorStyleTrackballCamera
  //
  // En esta funcion se interactua con el render para iniciar la funcion FindObject
  //-------------------------------------------------------------------------------------
    virtual void OnRightButtonDown()
    {

        // A la VAriable global dentro de la clase:  RenWin , se le asigna el renderwindow
        // actual para poderlo utilizar desde cualquier funcion dentro de la clase
```

```
            this->RenWin = this->Interactor->GetRenderWindow();

                                    // Inicia La Medicion del Tiempo
                                    start = clock();

        // Dependiendo de la desicion tomada en el "main" se realizara el
        // proceso de interaccion correspondiente en el FindObject
                FindObject();

                                    // Termina la medicion del tiempo
                                finish = clock();
                                 duration = (double)(finish - start) / CLOCKS_PER_SEC;
                                 printf( "\nFindObject duro = %f seconds\n\n", duration );

        // Forward events:
        // Permite a la funcion OnRightButtonDown continuar con las
        // actividades para las que fue diseñada en VTK
    vtkInteractorStyleTrackballCamera::OnRightButtonDown();

    }// Virtual Void

//****************************************************************************************

        // Variables Globales Unicamente dentro de esta Clase
        vtkSTLReader * planeSource;
        vtkRenderWindow *RenWin;
        vtkSmartPointer<vtkActor> actorR;
        vtkSmartPointer<vtkPolyDataMapper> mapperR;

};
vtkStandardNewMacro(MouseInteractorStyle01);
```

D3. Generador de una Imagen Teórica

```
//=============================================================================================
// Ing. Omar Daniel Rodríguez Gutiérrez
// UASLP CIEP
//
// ImageVTKTeoricaSilueta: Funcion que crea una imagen Teorica
//
// Valores de Entrada: Coordenadas de la posicion del objeto ( X , Y )
// Dimensiones de la imagen ( dimx , dimy )
// ThetZ = Angulo de la figura
//
// Matriz donde se desea alojar la silueta:
// Md[][] es una matriz de ( dimx , dimy )
//
// En este código se Asume que esta funcion esta dentro de la interaccion de
// VTK MouseInteractorStyle01 ó que es llamada por una funcion dentro de la
// misma interaccion, Es decir que VTK ya ha renderizado una ventana y se trabaja
// con dicha ventana.
//=============================================================================================
void ImageVTKTeoricaSilueta(float X, float Y, float ThetZ, float LocThetX, float LocThetY,
float LocThetZ, float **Md)
{
        // Declaracion de Variables
        int i,j,k;
        double p0[3]={0,0,0};

        // Iniciamos Las Variables
        k=0;

        //-----------------------------------------------------------------------------
        // En esta Parte se Asume que esta funcion esta dentro de la interaccion de
        // VTK MouseInteractorStyle01 ó que es llamada por una funcion dentro de la
        // misma interaccion. Es decir que se trabaja con una ventana de VTK Renderizada
        // por lo que las dimensiones de dicha ventana deben coincidir con las  de la
        // imagen Real
        //-----------------------------------------------------------------------------
```

```
        // Vector con la posicion x, y, z. debe ser double para VTK
        double AuxZ;

        // Se obtiene la Posicion Z del objeto puesto que no es una
        // variable que se esté optimizando
        this->RenWin->GetRenderers()->GetFirstRenderer()->GetActors()->GetLastActor()-
>GetPosition(p0);

        AuxZ=p0[2];

        //---------------------------------------------------------------------
        // Se Manda llamar la funcion OnImageVTKSilueta
        //---------------------------------------------------------------------

        OnImageVTKSilueta((double)X,(double)Y,AuxZ,(double)ThetZ,(double)LocThetX,(double)Lo
cThetY,(double)LocThetZ);

//-------------------------------------------------------------------------------------
// Se Reyenan las matrices de OpenCV y nrutil.
//
// Cuando el valor es menor a cero se tiene color blanco, en caso contrario sera color
// negro. Para Realizar el proceso a cada pixel blanco se le dara valor de 127 y a los
// pixeles negros de 0.0
//
// Se trabajara por ahora solo en el canal Azul por lo que en los otros canales el valor
// sera de 0.0 para todos los pixeles.
// for(j=0; (Azul)      for(j=j+1;  (Verde)    for(j=j+2;  (Rojo)
// y una conbinacion de Verde y Rojo = Amarillo
//-------------------------------------------------------------------------------------
        for(i=0;i<dimy;i++)
        {
                for(j=0;j<dimx_3;j=j+3)
                {
                        if(Opti_Image[Problem].Testim02->imageData[  j+(i*dimx_3)  ] < 0.0)
                        {
                                Md[i][k]=127.0;
                        }
                        if(Opti_Image[Problem].Testim02->imageData[  j+(i*dimx_3)  ] >= 0)
                        {
                                Md[i][k]=0.0;
                        }

                k=k+1;
                }
        k=0;
        }

}
```

D4. Orden para la Generación de la Imagen Teórica

```
//=====================================================================================
// Ing. Omar Daniel Rodríguez Gutiérrez
// UASLP CIEP
//
// OnImageVTKSilueta  :
// Funcion que ordena generar un screenshot del Render con los valores de posicion del
// modelo requeridos, para despues ser procesada acorde a las necesidades del codigo
//=====================================================================================
void OnImageVTKSilueta(double xpixel, double ypixel, double zpixel, double ThetZ, double
LocThetX, double LocThetY, double LocThetZ)
{

        // Si se solicita alguna rotación diferente de cero en alguno de los angulos de los
        // ejes de coordenadas globales del modelo CAD, entonces se realiza.
        if(LocThetX != 0)
```

```
		this->RenWin->GetRenderers()->GetFirstRenderer()->GetActors()->GetLastActor()->RotateWXYZ(LocThetX,1,0,0);
	if(LocThetY != 0)
		this->RenWin->GetRenderers()->GetFirstRenderer()->GetActors()->GetLastActor()->RotateWXYZ(LocThetY,0,1,0);
	if(LocThetZ != 0)
		this->RenWin->GetRenderers()->GetFirstRenderer()->GetActors()->GetLastActor()->RotateWXYZ(LocThetZ,0,0,1);

	// Se aplican las Propiedades de rotación y de traslacion al modelo CAD
	this->RenWin->GetRenderers()->GetFirstRenderer()->GetActors()->GetLastActor()->RotateWXYZ(ThetZ,0,0,1);
	this->RenWin->GetRenderers()->GetFirstRenderer()->GetActors()->GetLastActor()->SetPosition(xpixel,ypixel,zpixel);

	//-------------------------------------------------------------------------------
	// Screenshot VTK  :
	// Se captura la imagen
	//-------------------------------------------------------------------------------
		ScreenShotVTK(Opti_Image[Problem].Testim02);

	  // Devido a que las rotaciones son acumulativas, se regresa toda rotacion
	  // que se haga dentro de esta funcion
	  this->RenWin->GetRenderers()->GetFirstRenderer()->GetActors()->GetLastActor()->RotateWXYZ(-ThetZ,0,0,1);

	if(LocThetX != 0)
		this->RenWin->GetRenderers()->GetFirstRenderer()->GetActors()->GetLastActor()->RotateWXYZ(-LocThetX,1,0,0);
	if(LocThetY != 0)
		this->RenWin->GetRenderers()->GetFirstRenderer()->GetActors()->GetLastActor()->RotateWXYZ(-LocThetY,0,1,0);
	if(LocThetZ != 0)
		this->RenWin->GetRenderers()->GetFirstRenderer()->GetActors()->GetLastActor()->RotateWXYZ(-LocThetZ,0,0,1);


	} // Fin OnImageVTKSilueta
```

D5. Screenshot

```
//================================================================================
// Ing. Omar Daniel Rodríguez Gutiérrez
// UASLP CIEP
//
// Funcion que toma un screenshot de el Render de VTK
// this->RenWin : es la variable global dela clase MouseInteractorStyle
// de forma que esta funcion asi como esta solo funciona dentro de la misma clase
//
// Guarda la imagen en el apuntador de opencv :
// Opti_Image[Problem].Testim02
//================================================================================void
ScreenShotVTK(IplImage *Imagen)
{
	int i=0,j=0,k=0;
	//-------------------------------------------------------------------------------
	// Screenshot VTK  :
	// Se Adquiere la Imagen desde el Renderizador
	//-------------------------------------------------------------------------------
	//-------------------------------------------------------------------------------
	// this->RenWin->Render();
	// Vuelve a dibujar en la pantalla de VTK pero hace el procedimiento
	// mas lento
	//-------------------------------------------------------------------------------
	//this->RenWin->Render();
	//-------------------------------------------------------------------------------
	// this->RenWin->GetRenderers()->GetFirstRenderer()->Render();
	// Actualiza los valores de los pixeles en el render actual de VTK
```

```
        // Deja de dibujar en pantalla pero el procedimiento es mas rapido
        //-------------------------------------------------------------------------------
        this->RenWin->GetRenderers()->GetFirstRenderer()->Render();

        //-------------------------------------------------------------------------------
        // Genera y Envia a Null el Apuntador de la matriz de pixeles ImagenVTK
        //-------------------------------------------------------------------------------
        unsigned char *ImagenVTK = NULL;

        //-------------------------------------------------------------------------------
        // this->RenWin->GetRenderers()->GetFirstRenderer()->Render();
        // Actualiza los valores de los pixeles en el render actual de VTK
        // Aunque no redibuja en pantalla pero hace el procedimiento mas rápido
        //-------------------------------------------------------------------------------
        // GetPixelData
        // Descripcion:
        // Optiene la información de los pixeles de una imagen, transmitida como RGBRGB.
        // Es responsabilidad de quien lo llama liberar la memoria del arreglo
        // resultante. Es importante darse cuenta que la memoria en este arreglo
        // está organizada de abajo de la ventana hacia arriba. El origen de la
        // pantalla está en la esquina inferior izquierda. El eje y se incrementa
        // conforme se sube en la pantalla. Así que el acomodo de los pixeles es de
        // izquierda a derecha y de abajo a arriba
        //-------------------------------------------------------------------------------
        ImagenVTK = ( this->RenWin->GetPixelData(0,0,dimx-1,dimy-1,0) );
        //-------------------------------------------------------------------------------
        // Screenshot VTK  :
        // Se Adquiere la Imagen desde el Renderizador
        // El apuntador de OpenCV es BGR y este es RGB
        //-------------------------------------------------------------------------------
        k=0;
        for(i=0;i<dimy;i++)
        {
                for(j=0;j<dimx_3;j=j+3)
                {

        Imagen->imageData[  j+(i*dimx_3)  ]=(char)ImagenVTK[  j+2+((dimy-1-i)*dimx_3)  ];
        Imagen->imageData[  j+1+(i*dimx_3)  ]=(char)ImagenVTK[  j+1+((dimy-1-i)*dimx_3)  ];
        Imagen->imageData[  j+2+(i*dimx_3)  ]=(char)ImagenVTK[  j+((dimy-1-i)*dimx_3)  ];

                k=k+1;
                }
        k=0;
        }

        //-------------------------------------------------------------------------------
        // Libera el espacio de memoria en el apuntador creado por
        // this->RenWin->GetPixelData(0,0,dimx-1,dimy-1,0) y que fue almacenado
        // en ImagenVTK
        //-------------------------------------------------------------------------------
        if (ImagenVTK!=NULL)
        {
                free(ImagenVTK);
        }
        ImagenVTK = NULL;

} // Fin Screenshot
```

APÉNDICE E

Configuración de Modelos CAD

Para configurar los modelos CAD se utiliza un archivo llamado "ModelosCAD.txt" en él se agregan las variables necesarias para que el algoritmo lea los modelos CAD disponibles y las características que permitan configurarlos adecuadamente. Dichas características fueron diseñadas para que sean fácilmente ingresadas por el usuario.

El archivo de texto "ModelosCAD.txt" contiene las siguientes líneas, por default todo aquello que este escrito después de // es ignorado por el programa y para leer cualquier valor debe seguir de un signo "=" y un espacio, es importante que se siga al pie de la letra el orden que se indica a continuación puesto que el programa por ahora solo lee los valores que se ingresan en este mismo orden:

```
//================================================================================
//                          Modelos CAD:
//
// Para cada modelo debe existir su archivo .stl
// El archivo debe estar en la carpeta del Proyecto
// Las unidades del modelo CAD deben estar en centímetros
//
// El programa solo lee los valores numéricos después de: " = y un espacio "
// Todo lo que siga de " / " se ignora
//================================================================================
```

Lo primero que debe especificarse en el archivo es el número de objetos que se desea identificar y los cuales comprenderán la base de datos de los modelos CAD.

```
//================================================================================
// Numero de Objetos
// Si no actualiza este marcador No se leerán los
// Modelos CAD que desee agregar
//================================================================================
NumeroObjetos = 4
```

En seguida, para cada modelo el orden es el mismo, se ingresa el número de caracteres del nombre del archivo del modelo CAD en formato STL, luego se escribe el nombre del archivo junto con su extensión y el número de configuraciones deseadas para este modelo, como mínimo se debe inicializar una configuración para su respectivo objeto.

```
//================================================================================
// Modelo  0  :  "camara.stl"
//================================================================================
        NumeroCaracteres_NombreSTL = 10
        NombreSTL = camara.stl
        NumeroConfiguraciones = 3
```

Una vez que ya se inicializó un Modelo CAD se puede comenzar a introducir los valores para cada una de sus configuraciones:

```
//----------------------
// Configuración 0
//----------------------
```

AnguloRotacionScanZ: es el ángulo en grados de la diferencia entre cada rotación de θ en coordenadas mundiales para el método de Scaner (No se hace alusión de este método en el proyecto de tesis debido a que consiste en una metodología convencional) no debe ser igual a 0°, use 360° si no desea que el objeto rote en esta configuración.

```
AnguloRotacionScanZ = 10
```

AnguloRotacionOptimizacionZ: es el ángulo de diferencia en grados de θ por medio del cual se inicializa el optimizador para esta configuración, no debe ser igual a 0°, use 360° si no desea que el objeto se inicialice en otro ángulo diferente de 0° para esta configuración.

```
AnguloRotacionOptimizacionZ = 180
```

Al momento de leer un modelo CAD en STL con las librerías de VTK la posición y la orientación con las que fue diseñado el modelo se conservan. Entonces para cada configuración se desea que el programa acomode al modelo CAD en una posición y orientación adecuada sobre el plano de trabajo. Para ello se espera que el usuario introduzca los ángulos de rotación pertinentes para acomodar al modelo en la configuración deseada y también los desplazamientos del objeto en coordenadas mundiales. Dichos desplazamientos van en unidades de cm, y las rotaciones van en unidades de grados.

```
DesplazamientoPosicionX = 0
DesplazamientoPosicionY = 0
DesplazamientoPosicionZ = 0

RotacionInicialThetaX = 0
RotacionInicialThetaY = 0
RotacionInicialThetaZ = 0
```

Es importante también mencionar que el programa realiza las rotaciones de cada configuración con respecto a su centro de gravedad, para obtener mejores resultados y agilizar el procedimiento de optimización. Por lo tanto para cada configuración el usuario debe ingresar la distancia del centro de gravedad del modelo CAD visto desde arriba del plano formado por los ejes x y y, hacia el origen de las coordenadas mundiales del sistema, esta distancia entre estos dos puntos la llamaremos Hipotenusa, la cual está compuesta de un cateto opuesto (cateto en el eje y) y un cateto adyacente (cateto en el eje x), por supuesto con unidades en cm.

```
CatetoY = 12.4
CatetoX = 2.5
```

ConfiguracionCritica es una variable que quiere decir que el modelo CAD para esta configuración está de Pie (si es el caso ingrese un 1), o que el modelo se encuentra Acostado (si es el caso ingrese un 0). Cuando la metodología se encuentra comparando un objeto que podría decirse que está

acostado es decir que una de sus caras más largas esta contra el plano de trabajo, entonces al momento de cambiar de configuración y realizar la comparación de la imagen con otra configuración en la cual ahora el lado con menor área esta contra el plano de trabajo, esto puede causar que el procedimiento arroje un resultado equivocado al momento de optimizar sin saber que dicho modelo tiene un cambio tan brusco de una a otra configuración. Se utiliza el sentido común para declarar esta variable de preferencia solo cuando la silueta del objeto cambia significativamente de una configuración a otra, de lo contrario use el valor de 0.

```
ConfiguracionCritica = 0
```

Hay objetos que cuando se presentan en cierta configuración pueden presentar más de una sola rotación. En el caso de que la Configuración merezca un giro adicional Seleccione el ángulo en grados en que se desee que deba girar el objeto. Ingrese 0° en caso de no necesitarse una rotación adicional.

```
RotacionLocalThetaX = 0
RotacionLocalThetaY = 0
RotacionLocalThetaZ = 0
```

En adelante se puede seguir introduciendo las variables para las diferentes configuraciones como se muestra a continuación o se puede introducir tantos objetos se requieran.

```
//----------------------
// Configuración 1
//----------------------
AnguloRotacionScanZ = 30
AnguloRotacionOptimizacionZ = 180
DesplazamientoPosicionX = 0
DesplazamientoPosicionY = 0
DesplazamientoPosicionZ = 0
RotacionInicialThetaX = 1.4995
RotacionInicialThetaY = -90.0
RotacionInicialThetaZ = 0.0
CatetoY = 12.4
CatetoX = -3.3
ConfiguracionCritica = 0
RotacionLocalThetaX = 0
RotacionLocalThetaY = 0
RotacionLocalThetaZ = 0
```

APÉNDICE F

Reconocimiento y Ubicación de los Modelos CAD

F1. Estructura Para los Modelos CAD

Para comprender mejor el algoritmo de reconocimiento y ubicación de los modelos CAD, se muestran a continuación las estructuras en donde se almacenan las variables de cada modelo, sus configuraciones y el ángulo de rotación inicial θ. Cabe mencionar que el programa puede trabajar con dos distintas metodologías por lo que algunas de las variables pertenecen a una metodología convencional denominada en el programa como "Scan" y otras pertenecen a la metodología propuesta en este proyecto denominada en el programa como "Optimización".

```
//---------------------------------------------------------------------------------------
// Struct Rotaciones
// Estructura Para Cada Rotación en CoordenadasMundiales (ThetaZ)
// de Cada Configuración de Cada ModeloCAD (objeto)
//---------------------------------------------------------------------------------------
struct Rotaciones
{
float **Silueta;// Matriz Para Alojar la silueta del objeto en este angulo de rotacion
int CGx;       // Coordenada x del centro de gravedad de la silueta en pixeles
int CGy;       // Coordenada y del centro de gravedad de la silueta en pixeles
int LongLSilueta;// distancia del lado mas largo donde se comprende la silueta en pixeles
float ResultComparation;// Resultado de comparar una silueta Teorica con la silueta Real
double  ScanThetZ;// Angulo ThetaZ en CoordenadasMundiales de una Configuracion de un
ModeloCAD
double  OptiThetZ;// Angulo ThetaZ en CoordenadasMundiales de una Configuracion de un
ModeloCAD
};

//---------------------------------------------------------------------------------------
// Struct Configuraciones
// Estructura Para Cada Configuracion Que Pueda Tener Un ModeloCAD
//---------------------------------------------------------------------------------------
struct Configuraciones
{
Rotaciones *Rotacion; // Arreglo para las diferentes Rotaciones en CoordenadasMundiales de
ThetaZ
int NumRotScan;       //  Numero  de  Rotaciones  en  CoordenadasMundiales  Max  para  esta
Configuracion ( 1,2,...) (para el metodo de Scan)
int NumRotOpti;       //  Numero  de  Rotaciones  en  CoordenadasMundiales  Max  para  esta
Configuracion ( 1,2,...) (para el metodo de Optimizacion)
int ConfCritica;      // Si la config es critica quiere decir que la rotacion es mas rapida
que la translacion (0 = No, 1 = Si)
double AnRotScanZ;    // Angulo de inicio(grados) entre cada Rotacion del angulo ThetaZ en
Coord.Mundiales,(para el metodo de Scan)
```

```
double AnRotOptimZ;   // Angulo de inicio(grados) entre cada Rotacion del angulo ThetaZ en
Coord.Mundiales,(para el metodo de Optimizacion)
double ThetaX;        // Angulo en Coord.Mundiales Fijo para esta configuracion (grados)
double ThetaY;        // Angulo en Coord.Mundiales Fijo para esta configuracion (grados)
double ThetaZ;        // Angulo en Coord.Mundiales Fijo para esta configuracion (grados)
double PosX;          // Translacion en Coord.Mundiales Fijo para esta configuracion (cm)
double PosY;          // Translacion en Coord.Mundiales Fijo para esta configuracion (cm)
double PosZ;          // Translacion en Coord.Mundiales Fijo para esta configuracion (cm)
double CateX;         // CatetoAdyacente o CatetoX de la Hipotenusa (Hipot)
double CateY;         // CatetoOpuesto o CatetoY de la Hipotenusa (Hipot)
double Hipot;         //  Distancia del Origen de CoordMundiales Centro de Gravedad del
Modelo CAD Visto desde Arriba para esta Conf. (cm)
double Fhi;           // Angulo en Grados para trasladar el Origen al Centro de Gravedad
Visto desde Arriba para esta Conf. (grados)
double LocalThetaX;   // Angulo en Coord.Locales Variable para esta configuracion si es = 0
entonces no se usa (grados)
double LocalThetaY;   // Angulo en Coord.Locales Variable para esta configuracion si es = 0
entonces no se usa (grados)
double LocalThetaZ;   // Angulo en Coord.Locales Variable para esta configuracion si es = 0
entonces no se usa (grados)
};

//-------------------------------------------------------------------------------------
// Struct Objects
// Estructura Para los diferentes ModelosCAD (objetos)
//-------------------------------------------------------------------------------------
struct Objects
{
Configuraciones  *Configuracion;// Arreglo para las configuraciones de los diferentes
ModelosCAD (objetos)
char *NameSTL;        // Nombre del Archivo STL contenedor del objeto ( Modelo.stl ))
int NumConf;          // Numero de configuraciones para este Objeto
}*Object;

//-------------------------------------------------------------------------------------
// Struct BestComparations
// Estructura que se utiliza para almacenar las mejores comparaciones
// Entre una imagen real y la imagen teórica
//-------------------------------------------------------------------------------------
struct BestComparations
{
float ValorComp;
int NumObject;
int NumConf;
int NumRot;
float xfin;
float yfin;
float thetZfin;
}BestComp[5];
```

F2. OptimizaConfiguraciones

El siguiente algoritmo es el encargado de realizar el reconocimiento y la ubicación de los modelos CAD, consiste de tres ciclos principales el primero se encarga de leer y renderizar cada uno de los modelos, el segundo aplica las características al modelo para llevarlo a la posición de cada una de sus configuraciones, y por último el tercer ciclo se encarga de posicionar cada configuración en sus respectivos ángulos de rotación θ inicial. A continuación cada paso del algoritmo se explica a detalle.

```
//=====================================================================================
// Ing. Omar Daniel Rodríguez Gutiérrez
// UASLP CIEP
//
// OptimizaConfiguraciones():
// Funcion que realiza la identificación de objetos, usando un ciclo que
```

```
// consiste en generar una imagen teorica para cada modelo CAD disponible
// y cada una de sus respectivas configuraciones
//=============================================================================================
void OptimizaConfiguraciones(double pinit0, double pinit1 , double pinit2)
{

	//---------------------------------------------------------------------------------------
	// Crea las variables y las inicializa.
	//---------------------------------------------------------------------------------------
	// Declaracion de Variables
	int i,j,h,f,g,ConfCritica=0;
	double p0[3] = {0.0 , 0.0 , 0.0}, p1[3] = {0.0 , 0.0 , 0.0};
	double ThetaZ;
	//---------------------------------------------------------------------------
	// Crea las variables para Optimizar y las inicializa.
	//---------------------------------------------------------------------------
	int n,nmax,nvarmax=6,neval;
	float ftol;
	float yfin=0;
	float **xinit=NULL;
	float **eps=NULL;
	float **xfin=NULL;

	xinit=matrix(1,(1),1,(nvarmax));
	eps=matrix(1,(1),1,(nvarmax));
	xfin=matrix(1,(1),1,(nvarmax));

	for(j=1;j<=nvarmax;j++)
	{
		xinit[1][j]=0.0;	// Valores iniciales de cada etapa de optimizacion
		eps[1][j]=0.0;		// Valores de cada salto del simplex
		xfin[1][j]=0.0;		// Resultado de cada etapa de optimizacion
	}
	xfin[1][1]=pinit0;
	xfin[1][2]=pinit1;

	for(i=0;i<5;i++)
	{
		BestComp[i].ValorComp=1;
		BestComp[i].NumObject=1;
		BestComp[i].NumConf=1;
		BestComp[i].NumRot=1;
		BestComp[i].xfin=1;
		BestComp[i].yfin=1;
		BestComp[i].thetZfin;
	}
	//---------------------------------------------------------------------
	// Crea La Ventana y le asigna un Nombre
	//---------------------------------------------------------------------
	cvNamedWindow( "TestimOpenCV Real+Teorica", CV_WINDOW_AUTOSIZE );
	//---------------------------------------------------------------------------------------
	// Inicializa los Objetos
	//---------------------------------------------------------------------------------------
	//---------------------------------------------------------------------------------------
	// Inicializador de Parametros de los Objetos y Generador de sus respectivos Actores
	//---------------------------------------------------------------------------------------
	for(h=0;h<NumObjetos;h++)
	{
		//-----------------------------------------------------------------------
		// Lee un archivo en STL
		//-----------------------------------------------------------------------
		 this->planeSource = vtkSTLReader::New();
		 this->planeSource->SetFileName (Object[h].NameSTL);

		//Se Crea El Mapper con Plano Filtrado
		 this->mapperR = vtkSmartPointer<vtkPolyDataMapper>::New();
		 this->mapperR->SetInputConnection(this->planeSource->GetOutputPort());

		//Se crea el actor con el Mapper
		  vtkSmartPointer<vtkActor> actorR = vtkSmartPointer<vtkActor>::New();
		  actorR->SetMapper(this->mapperR);
```

```
                actorR->SetPosition(p0);
                actorR->RotateWXYZ(0.0,1,0,0);
                actorR->RotateWXYZ(0.0,0,1,0);
                actorR->RotateWXYZ(0.0,0,0,1);
                actorR->GetProperty()->SetColor(1,1,1);
              //Los objetos se escalan con un factor de 1/10 para que sus unidades
              //que son mm se conviertan a cm.
                actorR->SetScale(0.1);
              //-------------------------------------------------------------------------
              // Agrega el actor al Render
              //-------------------------------------------------------------------------
              this->RenWin->GetRenderers()->GetFirstRenderer()->AddActor(actorR);
              //-----------------------------------------------------------------
              // Se declaran las variables para cada configuracion
              //-----------------------------------------------------------------
              for(g=0;g<Object[h].NumConf;g++)
              {
              // Notese que para todas las Conf. se rota primero el angulo en el eje Y
antes que X
                    this->RenWin->GetRenderers()->GetFirstRenderer()->GetActors()-
>GetLastActor()->RotateWXYZ(Object[h].Configuracion[g].ThetaY,0,1,0);
                    this->RenWin->GetRenderers()->GetFirstRenderer()->GetActors()-
>GetLastActor()->RotateWXYZ(Object[h].Configuracion[g].ThetaX,1,0,0);
                    this->RenWin->GetRenderers()->GetFirstRenderer()->GetActors()-
>GetLastActor()->RotateWXYZ(Object[h].Configuracion[g].ThetaZ,0,0,1);

                    //-----------------------------------------------------------------
                    // Hasta aqui es la inicializacion para el objeto
                    // Ahora Comienza la Optimización
                    //-----------------------------------------------------------------
                    for(f=0;f<Object[h].Configuracion[g].NumRotOpti;f++)
                    {
                          //-----------------------------------------------------------
                          // Para el optimizador (optifun) se deben usar estas variables
                          //  Que conservan informacion del objeto que se está
                          // optimizando
                          //-----------------------------------------------------------
                          BestComp[0].NumObject=h;
                          BestComp[0].NumConf=g;
                          BestComp[0].NumRot=f;
                          //-----------------------------------------------------------
                          // Determina el Angulo ThetaZ
                          //-----------------------------------------------------------
                          ThetaZ  = Object[h].Configuracion[g].Rotacion[f].OptiThetZ;
                          //-----------------------------------------------------------
                          // Se utiliza la posicion estimada
                          // Se Actualiza la posicion del Objeto, puesto que
                          // inicialmente
                          // se encuentra cerca del origen.
                          //-----------------------------------------------------------
                          p0[0] = Object[h].Configuracion[g].PosX + xfin[1][1];
                          p0[1] = Object[h].Configuracion[g].PosY + xfin[1][2];
                          p0[2] = Object[h].Configuracion[g].PosZ + pinit2;
                          this->RenWin->GetRenderers()->GetFirstRenderer()->GetActors()-
>GetLastActor()->SetPosition(p0);

                          //-----------------------------------------------------------
                          // Call: Configuración Crítica e Inicio
                          // Ajusta unicamente el desplazamiento , el giro se hace
                          // hasta que la posicion se ha aproximado, con el proposito
                          // de que los objetos que giran mas rapido de lo que se
                          // dezplazan
                          // no den un angulo ThetaZ equivocado (configuracion critica =
                          // 1).
                          // Pero tambien se accesa a esta llamada con la 1ra
                          // configuracion
                          // del 1er objeto para asegurar una corresta posicion inicial
                          //-----------------------------------------------------------
                          if(   f==0   &&   Object[h].Configuracion[g].ConfCritica   !=
ConfCritica || ( h==0 && g==0 && f==0) )
                          {
```

```
                                        n=2;
                                        ftol=0.000001;
                                        nmax=30;
                                        xinit[1][1]=p0[0];
                                        xinit[1][2]=p0[1];
                                        eps[1][1]=5;
                                        eps[1][2]=5;
                                        Opti_Image[Problem].config=2;
                                        //-------------------------------------------------------
                                        // Optimiza
                                        //-------------------------------------------------------
                                        neval=Optimize(xinit,n,eps,ftol,nmax,xfin,&yfin);

                                        p0[0] = xfin[1][1];
                                        p0[1] = xfin[1][2];

                                        //-------------------------------------------------------
                                        // Cuando hay una diferencia de configuración crítica
                                        // de 1-0
                                        // o viceversa entonces se registra con esta variable
                                        //-------------------------------------------------------
                                        ConfCritica = Object[h].Configuracion[g].ConfCritica;
                                }
                                //-------------------------------------------------------------
                                // Call: Rotacion adicional
                                // Cuando una configuracion de alguno de los modelos CAD tiene
                                // una rotación adicional en alguna de sus configuraciones
                                // entonces
                                // el algoritmo accesa a esta llamada, que permite considerar
                                // las variables independientes extra
                                //-------------------------------------------------------------
                                if(   (   Object[h].Configuracion[g].LocalThetaX   !=   0     ||
Object[h].Configuracion[g].LocalThetaY != 0 ))
                                {
                                        n=6;
                                        ftol=0.000001;
                                        nmax=50;
                                        xinit[1][1]=p0[0];
                                        xinit[1][2]=p0[1];
                                        xinit[1][3]=ThetaZ;// Giro en el Eje Z
                                        xinit[1][4]=Object[h].Configuracion[g].LocalThetaX;
                                        xinit[1][5]=Object[h].Configuracion[g].LocalThetaY;
                                        xinit[1][6]=Object[h].Configuracion[g].LocalThetaZ;
                                        eps[1][1]=3;
                                        eps[1][2]=3;
                                        eps[1][3]=50;          // Desplazamiento del Giro Z
                                        eps[1][4]=Object[h].Configuracion[g].LocalThetaX;
                                        eps[1][5]=Object[h].Configuracion[g].LocalThetaY;
                                        eps[1][6]=Object[h].Configuracion[g].LocalThetaZ;
                                        Opti_Image[Problem].config=8;
                                        //-------------------------------------------------------
                                        // Optimiza
                                        //-------------------------------------------------
                                        neval=Optimize(xinit,n,eps,ftol,nmax,xfin,&yfin);
                                }
                                else
                                {
                                //-------------------------------------------------------------
                                // Call: Localizacion Simple
                                // Optimiza utilizando unicamente 3 variables independientes
                                // que corresponden a la posicion X,  Y y una orientación
                                // theta
                                //-------------------------------------------------------------
                                        n=3;
                                        ftol=0.000001;
                                        nmax=50;
                                        xinit[1][1]=p0[0];
                                        xinit[1][2]=p0[1];
                                        xinit[1][3]=ThetaZ;// Giro en el Eje Z
                                        eps[1][1]=3;
                                        eps[1][2]=3;
```

```
                eps[1][3]=50;  // Desplazamiento del Giro Z
                Opti_Image[Problem].config=4;
                //-----------------------------------------------------
                // Optimiza
                //-----------------------------------------------------
                neval=Optimize(xinit,n,eps,ftol,nmax,xfin,&yfin);
        }
        //-------------------------------------------------------------
        // Guarda las mejores Optimizaciones
        // BestComp[0].ValorComp = error, y se determina en optifun
        //-------------------------------------------------------------
        if(yfin < BestComp[1].ValorComp)
        {
        BestComp[3].ValorComp=BestComp[2].ValorComp;
        BestComp[3].NumObject=BestComp[2].NumObject;
        BestComp[3].NumConf=BestComp[2].NumConf;
        BestComp[3].NumRot=BestComp[2].NumRot;
        BestComp[3].xfin=BestComp[2].xfin;
        BestComp[3].yfin=BestComp[2].yfin;
        BestComp[3].thetZfin=BestComp[2].thetZfin;

        BestComp[2].ValorComp=BestComp[1].ValorComp;
        BestComp[2].NumObject=BestComp[1].NumObject;
        BestComp[2].NumConf=BestComp[1].NumConf;
        BestComp[2].NumRot=BestComp[1].NumRot;
        BestComp[2].xfin=BestComp[1].xfin;
        BestComp[2].yfin=BestComp[1].yfin;
        BestComp[2].thetZfin=BestComp[1].thetZfin;

        BestComp[1].ValorComp= yfin ;
        BestComp[1].NumObject=h;
        BestComp[1].NumConf=g;
        BestComp[1].NumRot=f;
        BestComp[1].xfin=xfin[1][1];
        BestComp[1].yfin=xfin[1][2];
        BestComp[1].thetZfin=xfin[1][3];
        }
        else if(yfin < BestComp[2].ValorComp)
        {
        BestComp[3].ValorComp=BestComp[2].ValorComp;
        BestComp[3].NumObject=BestComp[2].NumObject;
        BestComp[3].NumConf=BestComp[2].NumConf;
        BestComp[3].NumRot=BestComp[2].NumRot;
        BestComp[3].xfin=BestComp[2].xfin;
        BestComp[3].yfin=BestComp[2].yfin;
        BestComp[3].thetZfin=BestComp[2].thetZfin;

        BestComp[2].ValorComp= yfin ;
        BestComp[2].NumObject=h;
        BestComp[2].NumConf=g;
        BestComp[2].NumRot=f;
        BestComp[2].xfin=xfin[1][1];
        BestComp[2].yfin=xfin[1][2];
        BestComp[2].thetZfin=xfin[1][3];
        }
        else if(yfin < BestComp[3].ValorComp)
        {
        BestComp[3].ValorComp= yfin ;
        BestComp[3].NumObject=h;
        BestComp[3].NumConf=g;
        BestComp[3].NumRot=f;
        BestComp[3].xfin=xfin[1][1];
        BestComp[3].yfin=xfin[1][2];
        BestComp[3].thetZfin=xfin[1][3];
        }
}
//---------------------------------------------------------------------
// Restaura el Angulo ThetaX,ThetaY,ThetaZ
// Debido a que las rotaciones son acumulativas.
//---------------------------------------------------------------------
```

```
                    this->RenWin->GetRenderers()->GetFirstRenderer()->GetActors()-
>GetLastActor()->RotateWXYZ(-Object[h].Configuracion[g].ThetaZ,0,0,1);
                    this->RenWin->GetRenderers()->GetFirstRenderer()->GetActors()-
>GetLastActor()->RotateWXYZ(-Object[h].Configuracion[g].ThetaX,1,0,0);
                    this->RenWin->GetRenderers()->GetFirstRenderer()->GetActors()-
>GetLastActor()->RotateWXYZ(-Object[h].Configuracion[g].ThetaY,0,1,0);
                }
        //-------------------------------------------------------------------------
        // Remueve del Render al actor que ya fue optimizado
        //-------------------------------------------------------------------------
        this->RenWin->GetRenderers()->GetFirstRenderer()->RemoveActor(actorR);
}


//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
printf("\nComp3");
printf("\n  Obj  Conf  Rot  Comp");
printf("\n               %d                                %d                                %d
%.4f",BestComp[3].NumObject,BestComp[3].NumConf,BestComp[3].NumRot,BestComp[3].ValorComp);
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
printf("\nComp2");
printf("\n  Obj Conf  Rot  Comp");
printf("\n               %d                                %d                                %d
%.4f",BestComp[2].NumObject,BestComp[2].NumConf,BestComp[2].NumRot,BestComp[2].ValorComp);
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
printf("\nComp1 -> BestComp");
printf("\n  Obj Conf  Rot  Comp");
printf("\n               %d                                %d                                %d
%.4f",BestComp[1].NumObject,BestComp[1].NumConf,BestComp[1].NumRot,BestComp[1].ValorComp);
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


//-------------------------------------------------------------------------
// Ahora se Optimiza Con los resultados Obtenidos
//-------------------------------------------------------------------------
//-------------------------------------------------------------------------
// Para el optimizador (optifun) se deben actualizar estas variables
// Que conservan informacion del objeto que se está optimizando
//-------------------------------------------------------------------------
BestComp[0].NumObject = BestComp[1].NumObject;
BestComp[0].NumConf = BestComp[1].NumConf;
BestComp[0].NumRot = BestComp[1].NumRot;

h = BestComp[1].NumObject;
g = BestComp[1].NumConf;
f = BestComp[1].NumRot;

p0[0]=BestComp[1].xfin;
p0[1]=BestComp[1].yfin;
p0[2]=Object[h].Configuracion[g].PosZ + pinit2;

//-------------------------------------------------------------------------
// Lee un archivo en STL
//-------------------------------------------------------------------------
 this->planeSource = vtkSTLReader::New();
 this->planeSource->SetFileName (Object[h].NameSTL);

//Se Crea El Mapper con Plano Filtrado
 this->mapperR = vtkSmartPointer<vtkPolyDataMapper>::New();
 this->mapperR->SetInputConnection(this->planeSource->GetOutputPort());
//Se crea el actor con el Mapper
  vtkSmartPointer<vtkActor> actorR = vtkSmartPointer<vtkActor>::New();
  actorR->SetMapper(this->mapperR);
  actorR->SetPosition(p0);
  // Notese que se rota primero Y y luego X
  actorR->RotateWXYZ(Object[h].Configuracion[g].ThetaY,0,1,0);
  actorR->RotateWXYZ(Object[h].Configuracion[g].ThetaX,1,0,0);
  actorR->RotateWXYZ(Object[h].Configuracion[g].ThetaZ,0,0,1);
  actorR->GetProperty()->SetColor(1,1,1);
```

```
  actorR->SetScale(0.1);
//-----------------------------------------------------------------------
// Agrega el actor al Render
//-----------------------------------------------------------------------
this->RenWin->GetRenderers()->GetFirstRenderer()->AddActor(actorR);

//-----------------------------------------------------------------------
// Last Call
//-----------------------------------------------------------------------
//-----------------------------------------------------------------------
// Aditional Rot
//-----------------------------------------------------------------------
if(       (       Object[h].Configuracion[g].LocalThetaX       !=       0               ||
Object[h].Configuracion[g].LocalThetaY != 0 ))
{
       n=6;
       ftol=0.000001;
       nmax=40;
       xinit[1][1]=BestComp[1].xfin;
       xinit[1][2]=BestComp[1].yfin;
       xinit[1][3]=BestComp[1].thetZfin;// Giro en el Eje Z
       xinit[1][4]=Object[h].Configuracion[g].LocalThetaX;
       xinit[1][5]=Object[h].Configuracion[g].LocalThetaY;
       xinit[1][6]=Object[h].Configuracion[g].LocalThetaZ;
       eps[1][1]=0.5;
       eps[1][2]=0.5;
       eps[1][3]=3;                  // Desplazamiento del Giro Z
       eps[1][4]=Object[h].Configuracion[g].LocalThetaX / 5;
       eps[1][5]=Object[h].Configuracion[g].LocalThetaY / 5;
       eps[1][6]=Object[h].Configuracion[g].LocalThetaZ / 5;
       Opti_Image[Problem].config=8;

       //-----------------------------------------------------------------------
       // Optimiza
       //-----------------------------------------------------------------------
       neval=Optimize(xinit,n,eps,ftol,nmax,xfin,&yfin);
}
else
{
       n=3;
       ftol=0.000001;
       nmax=40;
       xinit[1][1]=BestComp[1].xfin;
       xinit[1][2]=BestComp[1].yfin;
       xinit[1][3]=BestComp[1].thetZfin;// Giro en el Eje Z
       eps[1][1]=0.5;
       eps[1][2]=0.5;
       eps[1][3]=3;                  // Desplazamiento del Giro Z
       Opti_Image[Problem].config=4;

       //-----------------------------------------------------------------------
       // Optimiza
       //-----------------------------------------------------------------------

       neval=Optimize(xinit,n,eps,ftol,nmax,xfin,&yfin);
}
//-----------------------------------------------------------------------
// Espera la tecla Esc o Q para salir cuando se presione
//-----------------------------------------------------------------------
cvWaitKey(0);
//-----------------------------------------------------------------------
// Para destruir las ventanas
//-----------------------------------------------------------------------
cvDestroyWindow("TestimOpenCV Real+Teorica");
//-----------------------------------------------------------------------
// Remueve al actor del Render
//-----------------------------------------------------------------------
this->RenWin->GetRenderers()->GetFirstRenderer()->RemoveActor(actorR);

} // Fin OptimizaConfiguraciones
```

APÉNDICE G

Generación de la Imagen Teórica de Calibración

El procedimiento para generar la imagen teórica del proceso de calibración es similar al que se usa para la identificación de objetos, a diferencia que ahora los objetos de calibración y su posición son valores conocidos, además se tiene una posición aproximada de la cámara y el punto del centro focal. Lo que se hace es modificar la posición de la cámara, posición del punto de centro focal, el zoom y el ángulo roll de la cámara.

La función ImageVTKCalibracion recibe las suposiciones de las variables previamente mencionadas y manda llamar la función OnImageVTKCalibracion que es la que hace las correcciones de acuerdo a las suposiciones en el ambiente virtual y captura la imagen teórica.

G1. Optifun: Función de Error Para la Calibración

```
//=============================================================================================
// OPTIFUN
// Optifun: Una funcion de error que será minimizada por "Optimize"
// llamada por "Optitry" y "Optimize"
//
// Variables Globales:Problem, que determina el case a utilizar en optifun
//                    Opti_Image[Problem], Estructura para cada proceso de optimización
//
// Recibe:  n = numero de variables
//                    X = una fila de la matriz P con n columnas, X = [1, ... , n]
//
// Regresa: un error despues de cada evaluacion de la función indicada por problem.
//=============================================================================================
float Optifun(float **X, int n)
{
        //---------------------------------------------------------------------------------
        // Variables Globales: Problem config iter Ydata testim
        //---------------------------------------------------------------------------------
        int i,j,k,h,g,f;
        float radius,ThetZ,sumsum=0.0;
        float error=0.0;
        float p0[3]={0,0,0};
        float LocThetX,LocThetY,LocThetZ;
        float FocPosX,FocPosY,FocPosZ;
        float x,y,z,zoom;
        i=0;
        j=0;
        k=0;

        //=============================================================================
        // Realiza un Proceso de Calibracion de Camara con Optimizacion y Chessboard
        //=============================================================================
        //-----------------------------------------------------
        // Se lleva la cuenta del número de iteraciones
        //-----------------------------------------------------
        Opti_Image[Problem].iter=Opti_Image[Problem].iter+1;
```

```
        //-------------------------------------------------------------------
        //    Las Variables independientes se establecen de acuerdo a la
        //     configuración que se esté utilizando
        //-------------------------------------------------------------------
        if (Opti_Image[Problem].config==2)
        {
                FocPosX=W_C_CamFocResult[0];
                FocPosY=W_C_CamFocResult[1];
                FocPosZ=W_C_CamFocResult[2];
        }
        if (Opti_Image[Problem].config==4)
        {
                FocPosX=X[1][6];
                FocPosY=X[1][7];
                FocPosZ=X[1][8];
        }
        //-------------------------------------------------------------------
        // Convierte las variables independientes de la cámara provenientes
        // del proceso de optimización,
        // a los valores que pueden ser ingresados a las funciones de VTK
        //-------------------------------------------------------------------
        z=X[1][3]/(sqrt( 1 + (X[1][1]*X[1][1]) + (X[1][2]*X[1][2]) ));
        x=X[1][1]*z;
        y=X[1][2]*z;
        zoom=X[1][3]*X[1][5];
        //-------------------------------------------------------------------------------
        // Para evitar que el proceso de Calibracion caiga en un minimo local no deseado,
        // si la suposicion del optimizador se aleja de cierta tolerancia de los valores
        // medidos de posicion de camara y posicion del punto focal,entonces dicha
        // suposicion se castiga con un valor fuera de las dimensiones del problema
        //-------------------------------------------------------------------------------
if( x > ( W_C_CamPosCalib[0] + CamPosTol ) || x < ( W_C_CamPosCalib[0] - CamPosTol ) )
        {
                error = 1;
                break;
        }
if( y > ( W_C_CamPosCalib[1] + CamPosTol ) || y < ( W_C_CamPosCalib[1] - CamPosTol ) )
        {
                error = 1;
                break;
        }
if( z > ( W_C_CamPosCalib[2] + CamPosTol ) || z < ( W_C_CamPosCalib[2] - CamPosTol ) )
        {
                error = 1;
                break;
        }
if( FocPosX > ( W_C_CamFocCalib[0] + CamFocTol ) || FocPosX < ( W_C_CamFocCalib[0] -
CamFocTol ) )
        {
                error = 1;
                break;
        }
if( FocPosY > ( W_C_CamFocCalib[1] + CamFocTol ) || FocPosY < ( W_C_CamFocCalib[1] -
CamFocTol ))
        {
                error = 1;
                break;
        }
if( FocPosZ > ( W_C_CamFocCalib[2] + CamFocTol ) || FocPosZ < ( W_C_CamFocCalib[2] -
CamFocTol ))
        {
                error = 1;
                break;
        }
        //-------------------------------------------------------------------
        // Manda Llamar la Funcion ImageVTKCalibracion
        // que aloja una imagen teórica en Opti_Image[Problem].Testim2
        //-------------------------------------------------------------------
        ImageVTKCalibracion(x,y,z,FocPosX,FocPosY,FocPosZ,X[1][4],zoom,Opti_Image[Problem].T
estim2);
        //-------------------------------------------------------------------
```

```
        // Se Normaliza la imagen teórica que esta en la matriz:
        // Opti_Image[Problem].Testim2
        //
        // En Matlab:
        // Testim2=Testim2/sqrt(sum(sum(Testim2)));
        //---------------------------------------------------------------------
        NormalizaMatriz(Opti_Image[Problem].Testim2,&(Opti_Image[Problem].sumsum2));
        //---------------------------------------------------------------------
        // Una vez Normalizada la imagen teórica, se realiza el producto
        // punto por la imagen real normalizada alojada en
        // Opti_Image[Problem].Testim1
        //
        // Ejemplo de Matlab:
        // error=1-sum(sum(Testim2.*Testim1));
        //---------------------------------------------------------------------
        sumsum =
ProductoPuntoMatriz(Opti_Image[Problem].Testim1,Opti_Image[Problem].Testim2,Opti_Image[Probl
em].Testim3);
        //---------------------------------------------------------------------
        // Con el resultado del producto punto (sumsum), que va de 0 a 1
        // se optiene el valor minimo de comparación.
        //---------------------------------------------------------------------
        error=1-sumsum;
        //---------------------------------------------------------------------
        // Si desea mostrar en pantalla la imagen teórica:
        // Si desea determinado color seleccione en el ciclo for
        // for(j=0; (Azul)      for(j=1;  (Verde)    for(j=2;  (Rojo)
        // y una conbinacion de Verde y Rojo = Amarillo
        //---------------------------------------------------------------------
        for(i=0;i<dimy;i++)
        {
                for(j=0;j<dimx_3;j=j+3)
                {
                        Opti_Image[Problem].TestimOpencv->imageData[  j+(i*dimx_3)
]=Opti_Image[Problem].Ydata->imageData[  j+(i*dimx_3)  ];
                        Opti_Image[Problem].TestimOpencv->imageData[  j+1+(i*dimx_3)
]=Opti_Image[Problem].Ydata->imageData[  j+1+(i*dimx_3)  ];
                        Opti_Image[Problem].TestimOpencv->imageData[  j+2+(i*dimx_3)
]=Opti_Image[Problem].Ydata->imageData[  j+2+(i*dimx_3)  ];

                        if(15*Opti_Image[Problem].Testim2[i][k]>0)
                        {
                                Opti_Image[Problem].TestimOpencv->imageData[  j+(i*dimx_3)
]=Opti_Image[Problem].Testim2[i][k]*Opti_Image[Problem].sumsum2;
                                Opti_Image[Problem].TestimOpencv->imageData[  j+1+(i*dimx_3)
]=0;
                                Opti_Image[Problem].TestimOpencv->imageData[  j+2+(i*dimx_3)
]=0;

                        if(15*Opti_Image[Problem].Testim2[i][k]>127)
                        Opti_Image[Problem].TestimOpencv->imageData[  j+(i*dimx_3)  ]=127;

                        }
                        k=k+1;
                }
                k=0;
        }
        //---------------------------------------------------------------------------
        // Muestra las imagenes
        //---------------------------------------------------------------------------
        cvShowImage( "TestimOpenCV Real+Teorica", Opti_Image[Problem].TestimOpencv );
        //---------------------------------------------------------------------------
        // Espera 5 milisegundos
        //---------------------------------------------------------------------------
          cvWaitKey(5);

} // End Optifun
```

G2. Generador de una Imagen Teórica Para Calibración

```
//=============================================================================================
// Ing. Omar Daniel Rodríguez Gutiérrez
// UASLP CIEP
//
// ImageVTKCalibracion: Funcion que crea una imagen teórica para Calibracion
//
// Valores de Entrada: Coordenadas de la posicion del objeto ( xpixel , ypixel )
// Dimensiones de la imagen ( dimx , dimy )
// Matriz donde se desea dibujar la figura gaussiano:
// Md[][] es una matriz de c_SizeX * c_SizeY
//
// En este código se Asume que esta funcion esta dentro de la interaccion de
// VTK MouseInteractorStyle ó que es llamada por una funcion dentro de la
// misma interaccion, Es decir que VTK ya ha renderizado una ventana y se trabaja
// con la ventana Renderizada.
//=============================================================================================
void ImageVTKCalibracion(float Xpos, float Ypos, float Zpos, float Xfoc, float Yfoc, float
Zfoc, float SetRoll, float SetZoom, float **Md)
{
        // Declaracion de Variables
        int i,j,k;
        // Iniciamos Las Variables
        k=0;
        //-------------------------------------------------------------------------------
        // En esta Parte se Asume que esta funcion esta dentro de la interaccion de
        // VTK MouseInteractorStyle ó que es llamada por una funcion dentro de la
        // misma interaccion. Es decir que se trabaja con una ventana de VTK Renderizada
        // por lo que las dimensiones de dicha ventana deben coincidir con las  de la
        // imagen Real
        //-------------------------------------------------------------------------------
        //---------------------------------------------------------------------
        // Se Manda llamar la funcion OnImageVTKCalibracion
        //---------------------------------------------------------------------
        OnImageVTKCalibracion((double)Xpos,(double)Ypos,(double)Zpos,(double)Xfoc,(double)Yf
oc,(double)Zfoc,(double)SetRoll,(double)SetZoom);

        //-------------------------------------------------------------------------------------
        // Se Reyenan las matrices de OpenCV y nrutil.
        //
        // Cuando el valor es menor a cero se tiene color blanco, en caso contrario sera
        // color
        // negro. Para Realizar el proceso a cada pixel blanco se le dara valor de 127 y a
        // los pixeles negros de 0.0
        //
        // Se trabajara por ahora solo en el canal Azul por lo que en los otros canales el
        // valor sera de 0.0 para todos los pixeles.
        // for(j=0; (Azul)      for(j=j+1;  (Verde)    for(j=j+2;  (Rojo)
        // y una conbinacion de Verde y Rojo = Amarillo
        //-------------------------------------------------------------------------------------
        for(i=0;i<dimy;i++)
        {
                for(j=0;j<dimx_3;j=j+3)
                {
                        if(Opti_Image[Problem].Testim02->imageData[  j+(i*dimx_3)  ] < 0.0)
                        {
                                Md[i][k]=124.0;
                        }
                        if(Opti_Image[Problem].Testim02->imageData[  j+(i*dimx_3)  ] >= 0)
                        {
                                Md[i][k]=0.0;
                        }
                k=k+1;
                }
        k=0;
        }

}
```

G3. Actualización de Variables para la Generación de la Imagen Teórica

```
//=============================================================================================
// Ing. Omar Daniel Rodríguez Gutiérrez
// UASLP CIEP
//
// OnImageVTKCalibracion  :
// Funcion que genera un screenshot de la imagen Teorica de calibración con los valores
// requeridos, para despues ser procesada acorde a las necesidades del codigo
//=============================================================================================
void OnImageVTKCalibracion(double Xpos, double Ypos, double Zpos, double Xfoc, double Yfoc,
double Zfoc, double SetRoll, double SetZoom)
	{

//---------------------------------------------------------------------------
// Se asignan los valores a la posicion de camara y punto del centro focal
//---------------------------------------------------------------------------
	this->RenWin->GetRenderers()->GetFirstRenderer()->GetActiveCamera()-
>SetPosition(Xpos,Ypos,Zpos);
	this->RenWin->GetRenderers()->GetFirstRenderer()->GetActiveCamera()-
>SetFocalPoint(Xfoc,Yfoc,Zfoc);
	this->RenWin->GetRenderers()->GetFirstRenderer()->GetActiveCamera()-
>SetRoll(SetRoll);
	this->RenWin->GetRenderers()->GetFirstRenderer()->GetActiveCamera()->Zoom(SetZoom);

	//-------------------------------------------------------------------------------
	// Screenshot VTK  :
	// Se captura la imagen
	//-------------------------------------------------------------------------------
		ScreenShotVTK(Opti_Image[Problem].Testim02);
//---------------------------------------------------------------------------
// El Zoom SetZoom es Acumulable, con esta linea
// el valor se reinicia permitiendo que se ingrese un nuevo valor
//---------------------------------------------------------------------------
this->RenWin->GetRenderers()->GetFirstRenderer()->GetActiveCamera()->Zoom(1/SetZoom);


} // fin OnImageVTKCalibracion
```

APÉNDICE H

Configuración del Archivo de Calibración

Para configurar las variables de calibración se utiliza un archivo llamado “ConfCalibracion.txt” en el cual se agregan los valores de los objetos de calibración, las estimaciones de la posición de la cámara y otras variables para poder realizar pruebas.

El archivo de texto “ConfCalibracion.txt” contiene las siguientes líneas, por default todo aquello que este escrito después de // es ignorado por el programa y para leer cualquier valor debe seguir de un signo “ = ” y un espacio, cuando en una misma línea hay más de un valor se separan con una sola coma “ , ”, se pone un signo “ + “ cuando los valores serán reescritos por el programa. Es importante que se siga al pie de la letra el orden que se indica a continuación puesto que el programa por ahora solo lee los valores que se ingresan en este mismo orden:

```
//================================================================================
//                          Calibración:
// No permita que un renglón supere los 200 caracteres
// El programa solo lee los valores numéricos después de: " = “ y un espacio
// Cuando hay mas de un solo valor se separan con una sola coma " , "
// Todo lo que siga de " / " se ignora
// Cuando se lee un signo " + " Quiere decir que comienza una sección de lectura y escritura
//================================================================================
```

World Coordinates Camera Real: Son variables que se utilizan únicamente para realizar pruebas virtuales de calibración y representan los valores exactos de posición de la cámara, zoom y roll, unidades en cm.

```
W_C_CamPos[3] = -15,-30,40    // Posición de cámara exacta
W_C_CamFoc[3] = 0.7,0.5,0.0   // Posición del punto del centro focal de la cámara(El tercer valor Siempre igual a cero)
W_C_Roll      = 12            // Roll Camara
W_C_Zoom      = 0.9           // Zoom Camara
```

World Coordinates Camera Calibración: Son variables que se utilizan en el generador de la imagen teórica para calibración utilizando los 4 objetos hexaedros, son valores medidos de posición de cámara zoom y roll. Estas variables también sirven para realizar pruebas virtuales de calibración puede asignar valores arbitrarios cercanos a los valores de las World Coordinates Camera Real. Sus unidades están en cm.

```
W_C_CamPosCalib[3] = 0,-47,39          // Posición de cámara
W_C_CamFocCalib[3] = 0,0,0     // Posición del punto del centro focal de la cámara(El tercer valor Siempre igual a cero)
W_C_RollCalib = 0              // Roll Camera
W_C_ZoomCalib = 1                      // Zoom de cámara: Siempre = 1 (Este valor se utiliza para la 1ra suposición)
```

De las siguientes variables las primeras dos son tolerancias al momento de medir la posición de la cámara y del punto del centro focal. Para la posición de la cámara la tolerancia depende en cierta

medida del ancho y del largo de la cámara dependiendo del espacio a través del cual no se pueda precisar con exactitud el punto de la posición de cámara. La tolerancia para el punto del centro focal depende de que tanta cercanía se puede alcanzar al momento de dirigir el centro focal de la imagen obtenida de la cámara hacia el punto en el plano el cual corresponde al origen de las coordenadas mundiales que en este proyecto se usa para dirigir el centro focal de la cámara. Las siguientes tres variables son las dimensiones de los cuatro hexaedros los cuales deben ser iguales y que en el código son llamados como "pilares" esto por su forma alargada preferentemente. PilarAncho es el ancho del pilar, PilarAlto es la altura del pilar y PilarDist es la distancia que hay del lado de un pilar al de otro y que debe ser igual para los cuatro. La última de estas variables LightSep se integro para cuando se tengan 3 fuentes emisoras de luz desde el punto de observación de la cámara de esta forma el ambiente virtual puede representarse de la forma más adecuada a las condiciones reales, por ahora no es importante esta variable puesto que no se han observado problemas con una iluminación desde arriba de los objetos, puede utilizar el valor de PilarDist para LightSep.

```
CamPosTol = 7
CamFocTol = 1.0
PilarAncho = 5.1
PilarAlto = 14.7
PilarDist = 10.0
LightSep = 14.5 // Distancia de fuentes de luz aprox = PilarDist
```

Las siguientes variables se refieren a la calibración utilizando el tablero de ajedrez, ChessNumCuadros son el número de cuadros que tiene el tablero por cada lado, de preferencia el número de cuadros debe ser par, de esta manera es más fácil y preciso centrar el tablero al momento de calibrar. ChessGrueso es el grosor que puede presentar dicho tablero en cm. Si es una hoja de papel, no utilice el valor de cero, puesto que se crea el tablero como un objeto sólido, por lo que debe de tener un espesor. Para evitar que marque error al crear la imagen, indique un valor muy pequeño por ejemplo 0.01. ChessLadoCuadro es la dimensión del lado de los cuadros del tablero. El valor de ChessAlto se utiliza solo cuando se calibra con dos imágenes del tablero de ajedrez a dos alturas diferentes, con unidades en cm. La primera altura siempre es el valor de ChessGrueso y la segunda es ChessAlto. Se debe cuidar que ChessAlto siempre sea mayor a ChessGrueso.

```
ChessNumCuadros = 8
ChessGrueso = 0.01
ChessLadoCuadro = 2.4
ChessAlto = 12
```

World Coordinates Camara Result: Son las variables de cámara resultantes de la última calibración. Los valores se actualizan en este archivo de texto cada vez que se realiza un nuevo paso de calibración. Es por eso que siguen del signo " + " para indicar que son datos que se modifican

```
+                       // No Remueva este símbolo

W_C_CamPosResult[3] = -6.871394,-40.196335,45.820652
W_C_CamFocResult[3] = -0.652012,0.991644,0.000000
W_C_RollResult = 6.851523
W_C_ZoomResult = 1.073246
```