DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE RECONSTRUCCIÓN EN 3D A PARTIR DE UNA NUBE DE PUNTOS

BOLIVAR DAVID HERRERA GUERRERO
FREDY ALEXIS DULCE MERA

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
2014

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE RECONSTRUCCIÓN EN 3D A PARTIR DE UNA NUBE DE PUNTOS

BOLIVAR DAVID HERRERA GUERRERO
FREDY ALEXIS DULCE MERA

Trabajo de grado presentado como requisito parcial para optar por el título de Ingeniero Electrónico.

DIRECTOR
ING. MG. DARÍO FERNANDO FAJARDO.

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
2014

NOTA DE RESPONSABILIDAD

“la Universidad de Nariño no se hace responsable por las opiniones o resultados en el presente trabajo y para su publicación priman las normas sobre el derecho de autor”

Acuerdo 1. Artículo 324. Octubre 11 de 1966, emanado del honorable Consejo Directivo de la Universidad de Nariño.

Nota de aceptación

_____
_____
_____
_____
_____
_____
_____

_____
Firma del presidente del jurado

_____
Firma del jurado

_____
Firma del jurado

Pasto, 3 de marzo de 2014.

AGRADECIMIENTOS

A mi familia por apoyarme de forma incondicional en todo este recorrido y por estar siempre presentes en los momentos difíciles. Al ingeniero Darío Fajardo por su apoyo y acompañamiento en el desarrollo del proyecto. Y a todas las personas que de una u otra forma contribuyeron para llevar a buen término este trabajo.

Fredy Alexis Dulce Mera

Agradezco mucho a mi papa, mi mama y mi hermano por todo su apoyo durante el transcurso de mi carrera. A mis amigos por hacer grata mi experiencia en este proceso de formación. A los docentes que compartieron sus conocimientos, y especialmente al profesor Darío Fajardo por su orientación durante el desarrollo de este proyecto.

Bolivar David Herrera Guerrero

DEDICATORIA

A mis padres y a mi hermano quienes han sido pilares fundamentales en mi crecimiento personal y profesional.

Fredy Alexis Dulce Mera

A mi familia y amigos que influyeron con sus lecciones y experiencias en formarme como una mejor persona.

Bolivar David Herrera Guerrero

RESUMEN

Diferentes dispositivos se están usando para investigación en campos como la realidad aumentada y reconstrucción 3D. Una de estas herramientas es el sensor Kinect que permite obtener información de profundidad de los objetos en el mundo real. En artículos como [11], [14], [27], [28] se encuentran trabajos desarrollados con este sensor, algunos de estos tratan la corrección de datos capturados, la detección de movimientos y figuras humanas, el reconocimiento de gestos humanos y la captura de datos de profundidad. Su asequibilidad y popularidad ha logrado motivar el desarrollo de muchos trabajos de investigación.

En este trabajo se utilizó el sensor Kinect para hacer reconstrucción 3D con el uso de Matlab^{R}. En este software se realizó todo el procesamiento necesario para que cada una de las capturas hechas se ordene de tal forma que la combinación de todas resultara en la imagen digital en tercera dimensión del objeto capturado. Para lograr esto se utilizaron algoritmos como SURF [18] e ICP [16] con los que se determinaron puntos que correspondían entre una nube de puntos y otra, y se encontraron matrices de rotación y traslación que permitían el mejor acople de las capturas. Además la integración de estos algoritmos con funciones propias de Matlab^{R} hizo posible la obtención de objetos en 3D como se presenta en los resultados.

ABSTRACT

Currently different devices are being used for researches in fields such as augmented reality and 3D reconstruction. One of them is the Kinect sensor, which provides depth information of the objects in real world. In articles such as [11], [14], [27], [28] it was found researches developed with this kind of sensors, some of these papers deal with the correcting of the captured data, motion detection and human figures, human gesture recognition and depth data acquisition. Affordability and popularity of this sensor has motivated the development of many researches.

In this work the sensor Kinect was used to make 3D reconstruction with Matlab^{R}, in this software was perform all the processing to organize each capture, so that the combination of all of them result in the three-dimensional digital image of captured object. To achieve this goal, some algorithms such as Surf [18] and ICP [16] were used to determine corresponding points between point clouds and to find the rotation and translation matrixes which allow a better fit in the captures. Besides the integration of these algorithms with build-in Matlab^{R} functions made possible the production of 3D objects as presented in the results.

CONTENIDO

LISTA DE TABLAS

Pag.

LISTA DE FIGURAS

Pag.

LISTA DE ANEXOS

Pag.

GLOSARIO

ALGORITMO: conjunto ordenado y finito de operaciones que permiten hallar la solución a un problema.

CAPTURA: Adquisición de la información actual del dispositivo sensor.

ICP: Algoritmo *Iterative-closest-point*.

IMAGEN RGB: Imagen que muestra al objeto con sus tonos reales por medio de la combinación de tonos rojos verdes y azules.

KINECT: Controlador de juego libre y entretenimiento creado por Alex Kipman, desarrollado por Microsoft para la videoconsola Xbox 360. Disponible desde junio del 2011 para PC a través de Windows 7 y Windows 8.

MATLAB^{R}: Es un lenguaje de alto nivel y ambiente interactivo para cálculos numéricos, visualización y programación.

MOTION BLUR: Es el rastro dejado por los objetos en movimiento en una fotografía o en una secuencia de imágenes

PUNTOS COINCIDENTES: Pixeles con iguales características entre un par de imágenes en 2D.

SURF: Algoritmo *Speeded-Up-Robust-Features*.

SVD-DVS: *Singular Value Decomposition* - Descomposición de valores singulares

TRIANGULACIÓN: Transformación de una serie de puntos en una red de triángulos.

USB: *Universal serial bus*.

NUBE DE PUNTOS - *Point-cloud:* Es un conjunto de puntos en un sistema de coordenadas tridimensional. Estos puntos se identifican habitualmente como coordenadas *X*, *Y*, y *Z* y son representaciones de la superficie externa de un objeto.

MARCAS REGISTRADAS

Kinect: es una marca registrada de Microsoft Corporation.

Matlab^{R}: Software matemático desarrollado por la empresa Mathworks, inc.

Open NI: Software y controladores desarrollados por la empresa PrimeSense Ltd.

NITE: Software y controladores desarrollados por la empresa PrimeSense Ltd.

Prime Sensor Software y controladores desarrollados por la empresa PrimeSense Ltd.

Arduino Es una marca registrada del equipo Arduino.

INTRODUCCIÓN

La reconstrucción tridimensional ha sido uno de los campos más estudiados en cuanto al procesamiento digital de imágenes se refiere. Adquirir información sobre el tamaño, la forma y la textura de un objeto tiene infinidad de aplicaciones en la ingeniería. Algunas de las más importantes son el reconocimiento de objetos y prototipado rápido que permite la reproducción en masa de un objeto. También tiene gran importancia en el campo de la medicina en el cual se utilizan imágenes médicas para obtener una mejor perspectiva de ciertas enfermedades o alteraciones en el cuerpo humano, con mucha aplicación en cirugías. Entre otras aplicaciones encontramos la detección de errores en procesos industriales, la construcción de espacios virtuales y la creación de bases de datos de objetos arquitectónicos o culturales.

Con la llegada del dispositivo Kinect, se han estudiado diversas formas de utilizar la información espacial brindada por el instrumento, con el fin de crear aplicaciones referentes a la reconstrucción 3D. En este campo de aplicación existen dos problemas fundamentales: El registro de la información espacial, el cual consiste en lograr que todos los datos compartan un mismo espacio de referencia y la reconstrucción tridimensional a partir de los datos registrados.

En este trabajo se presenta un método para tratar estos dos problemas, el cual permite obtener resultados rápidos y a bajo costo. El principal objetivo de esta investigación es el de adquirir información sobre objetos representativos de la cultura nariñense y obtener los modelos tridimensionales de estos cuerpos que normalmente se destruyen o desaparecen en cortos periodos de tiempo.

1. PLANTEAMIENTO DEL PROBLEMA

A. Definición del problema

En la región de Nariño se presentan muchas manifestaciones artísticas y culturales que representan la variedad religiosa, de costumbres y tradiciones de esta región destacándose el carnaval de negros y blancos, nombrado como patrimonio cultural e inmaterial de la humanidad, muchas de las cuales debido a su elaboración no perduran por largos periodos de tiempo. Se evidencia la perdida de elementos culturales como un problema que es necesario abordar con propuestas de tipo tecnológico como por ejemplo la conservación de registros digitales tridimensionales del arte efímero del carnaval a través de sistemas de escaneo y reconstrucción en 3D.

Los sistemas de escaneo 3D son tema de investigación desde hace varios años, varias técnicas y tecnologías se han desarrollado para lograr reconstrucciones tridimensionales de un objeto. Existen técnicas por contacto y otras sin contacto. Las técnicas sin contacto se dividen en activas y pasivas entre las cuales se pueden mencionar: diferencia de fase, láser, luz estructurada, silueta, estereoscopía, *Structure from motion* (SFM), etc. [25].

En el departamento de ingeniería electrónica de la universidad de Nariño se han realizado investigaciones sobre este tipo de sistemas como el visto en [2], con los cuales se han logrado buenos resultados, sin embargo aún hay mucho trabajo de investigación y de aplicación que se puede realizar por lo que se propone el desarrollo de un sistema de reconstrucción a partir de una nube de puntos.

B. Justificación

Es necesario, encontrar una forma de evitar la pérdida total de elementos culturales como las expresiones artísticas que se presentan en el carnaval de negros y blancos pues la cultura es muestra de la identidad de los pueblos, e investigar formas que permitan preservar el arte y la cultura en el departamento de Nariño. Una forma de preservar estos elementos artísticos es mediante la captura y reconstrucción tridimensional de los objetos mediante un escaneo 3D.

La reconstrucción tridimensional de un objeto tiene una gran variedad de aplicaciones en la industria, la ingeniería, el entretenimiento y el patrimonio cultural, ya que permite llevar un registro con mayor detalle al otorgado por otras representaciones como imágenes o fotos. Además al lograr una reconstrucción tridimensional de un objeto, es fácil lograr la reproducción física a escala del mismo por medio de impresiones tridimensionales, inclusive permitir a personas ajenas a esta región apreciar nuestra cultura a distancia con ayuda de herramientas informáticas.

Los objetos reconstruidos tridimensionalmente permiten analizar, examinar y modificar los objetos sin la necesidad de manipularlos físicamente, un aspecto muy importante cuando se trabaja con piezas delicadas o con gran valor cultural como las producidas en el departamento de Nariño.

2. OBJETIVOS

A. Objetivo general

Diseñar un sistema de reconstrucción tridimensional a partir de un conjunto de nube de puntos sobre un barrido de 360° de un objeto.

B. Objetivos específicos

- Analizar el estado del arte sobre la técnica nube de puntos, y encontrar el método más adecuado para desarrollar un algoritmo que permita obtener una superficie 3d.

- Adaptar un dispositivo de captura de datos de profundidad para su utilización en el sistema de reconstrucción.

- Diseñar software y hardware que permita realizar la reconstrucción tridimensional a partir de los datos captados por el sistema sobre un barrido de 360°.

- Obtener la reconstrucción tridimensional de un objeto a través del sistema adaptado.

3. MARCO TEÓRICO.

A. Técnicas de captura

Entre las técnicas de adquisición se encuentra la desarrollada por estudiantes de la Universidad de Nariño [1]. En esta técnica se utiliza una iluminación con ondas planas a un objeto y a un plano de referencia y mediante procesamiento se obtiene un patrón de Moiré. La frecuencia espacial del patrón Moiré, comparada con la de referencia permite establecer una relación entre las longitudes de ondas de los haces incidentes, por último aplicando un método de desenvolvimiento se obtiene reconstrucciones 3d de diferentes objetos de tamaño mediano y pequeño.

Existen técnicas que requieren contacto con el objeto a reconstruir. En [2] y [3] se presentan algunas de este tipo como *Contact-based probing*. Estas técnicas tienen la desventaja de que al estar en contacto con el objeto éste puede dañarse o sufrir cambios. También es posible la adquisición de datos por fotogrametría que se basa en determinar las propiedades geométricas de objetos desde imágenes fotográficas [2]. Otros métodos de adquisición de datos son mediante láser [4] y la comparación de fase.

Por otra parte, recientemente con el dispositivo Kinect [5] se ha logrado adquirir información espacial a menor costo. El principio de funcionamiento de este sensor se basa en proyectar un patrón de puntos infrarrojos mediante un proyector laser IR a 830nm y los captura mediante una cámara IR CMOS. El dispositivo es calibrado con la información de la ubicación exacta de cada punto que aparece en la proyección a una distancia determinada y sobre una superficie plana, al comparar la información recibida con la de calibración, se establece la ubicación espacial de los puntos comprobando el desplazamiento de cada punto con respecto al de calibración, el sensor tiene un rango de captura de 0.7m a 4.5m aproximadamente. El dispositivo debido a sus características permite el desarrollo de numerosas aplicaciones tales como detección de movimiento, realidad aumentada, entre otras [6].

B. Métodos de reconstrucción a partir de nube de puntos.

Mediante métodos de adquisición de datos de profundidad se producen conjuntos de datos 3D (nube de puntos), obtenidos desde la superficie tridimensional de un objeto. Estos puntos son usualmente desorganizados y para utilizarlos en aplicaciones 3D se requiere procesar una malla poligonal (usualmente triangular) que mejor se aproxime a la superficie del objeto. Esto significa asociar una estructura de conectividad con el conjunto de puntos [7].

Las técnicas de creación de mallas de superficie a partir de puntos pueden caer en determinadas categorías: *Sculpting-based* y de región creciente *(region-Growing)* [8]. La ventaja de los métodos *Sculpting-Based*, normalmente basados en la triangulación de *delaunay*, es que regularmente ofrecen garantías teóricas de la calidad de la superficie

resultante y que la superficie reconstruida converge a la superficie real a medida que la densidad de muestras aumenta.

Algunos de los métodos para la creación de una malla de superficie a partir de nube de puntos son: *Marching Cubes* [9], Método *BPA (Ball pivoting Algorithm)* [8], Reconstrucción de superficie Poisson [10] y Creación de superficie utilizando parametrización esférica [7].

Por ejemplo el método *Marching Cubes* está basado en su contraparte en 2D, marching squares. El algoritmo de marching squares intenta dibujar líneas entre valores interpolados a lo largo de los límites de un cuadrado, considerando pesos dados a las esquinas y un valor de referencia.

Cada valor de la matriz tiene diferentes pesos y el valor de referencia es constante. Para encontrar la curva cuyo valor es constante e igual al de referencia, diferentes clases de interpolación pueden ser usados, la más utilizada es la interpolación linear.

En el algoritmo Marching cubes debido a que se da lugar en un espacio 3D, se enumeran 256 casos diferentes para la representación de marching cubes, todos estos pueden ser generalizados en 15 familias mediante rotación y simetría. Para determinar cada caso real, una notación ha sido establecida, la cual apunta a referenciar cada caso con un índice creado desde la interpretación binaria de los pesos de las esquinas como se indica en la figura 1.

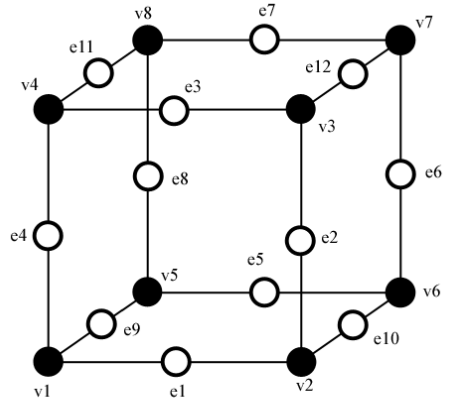


Figura 1 Índices utilizados en Marching Squares. La figura ilustra la notación utilizada mediante el método marching cubes [9].

En este sentido, algunos vertex desde 1 hasta 8 son pesados con valores de 1 hasta 128 (v1=1, v2=2, v3=3, etc.). Para cada uno de estos cubos se puede atribuir un cubo complementario. Determinar el cubo complementario consiste en revertir las normales del

cubo original, la creación de estos cubos complementarios permite darle una orientación a la superficie.

La referencia [11] presenta un método para mejorar los datos de profundidad a partir de las imágenes RGB y de profundidad del sensor, en [13] se presenta otro método para corregir los datos nulos que son generados por el ruido a través de un algoritmo de predicción llamado "2D+T". En [14] se muestra un esquema para una construcción automática y eficiente de un modelo de alta resolución y con textura más "realista" en 3D, este artículo se concentra en la construcción de un modelo facial e incluye en uno de sus pasos una segmentación del rostro para lograr un mejor modelo.

En [15] se encuentra un sistema de filtrado iterativo para suavizar las formas ruidosas de los modelos 3D a partir de mallas con triángulos, además presenta una evaluación cuantitativa para comparar diferentes niveles de filtrado y otros métodos de filtrado. Un ejemplo de esto se presenta en la figura 2, en (a) se muestra una reconstrucción a la que se ha aplicado *smoothing* y en (b) está la misma reconstrucción si ningún filtro aplicado.

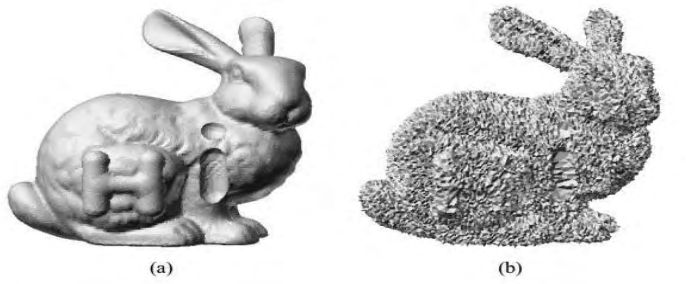


Figura 2 *Smoothing* de una superficie[15]. (a) reconstrucción suavizada. (b) datos ruidosos.

C. Iterative closest point – ICP [16]

ICP fue desarrollado por Paul Besl y Neil Mckay en 1992, en [16] se presenta el trabajo que desarrollaron con la explicación del algoritmo básico y los resultados obtenidos. Este es un algoritmo iterativo que se caracteriza por el alto coste computacional en cada iteración, lo que se traduce en velocidad de cálculo reducida.

Básicamente se tienen las nubes de puntos de 2 imágenes en 3D que se encuentran ubicados en diferentes posiciones en el espacio, y cada uno de estos puntos debe tener otro correspondiente o coincidente en la otra imagen, de modo que las 2 imágenes sean iguales o muy similares. En el caso que se trató en este proyecto, las imágenes no eran exactamente iguales entre sí, por lo que se tuvo que hacer una selección de puntos como se mostrara más adelante. El algoritmo encuentra iterativamente matrices de rotación R y de traslación T que minimizan la distancia entre puntos correspondientes en dos nubes de puntos, y se escogen las que presentaron el menor error de acople, las cuales se obtienen cuando el error dado por R y T es menor a un umbral.

Una descripción sencilla del método se indica a continuación:

Sea M y S dos nubes de puntos con NM y NS el número de puntos en cada nube respectivamente (NM=NS), cada punto de M corresponde a un punto de S. Se encuentran las matrices R de rotación y T de traslación, de tal forma que se minimice la ecuación (1):

$$f(R,T) = \frac{1}{NS}\sum_{i=1}^{Ns} \|Mi - (R * Si + T)\|^2 \quad (1)$$

Donde *f* es el error de acople entre las 2 nubes de puntos.

Los pasos básicos para el algoritmo se presentan en [17], y se describen de forma sencilla a continuación:

1. Inicialización de variables, matrices y vectores.
2. Selección de datos en un par de nubes de puntos.
3. Calculo de la matriz de rotación R y del vector de traslación T.
4. Aplicación de R y T al grupo de puntos objetivo.
5. Aplicación de criterio de parada.

Si el error de acople actual menos el error de acople de la iteración anterior es menor que un umbral, se detiene el algoritmo, de lo contrario se repiten los pasos 2 a 5.

D. SURF - Speeded Up Robust Features [18]

El algoritmo SURF es un detector y descriptor de puntos característicos locales en imágenes en escala de grises basado en SIFT [30]. El objetivo de encontrar puntos representativos entre dos imágenes de una misma escena u objeto, es localizar correspondencias que permitan identificar los cambios en traslación o rotación que ocurre entre dos imágenes. Entre las principales aplicaciones del algoritmo podemos destacar la calibración de cámaras, registro de imágenes, identificación de objetos, o en el caso tratado en este trabajo de grado, reconstrucción 3D.

Para lograr el objetivo de encontrar estas correspondencias, el algoritmo SURF utiliza 3 pasos: primero se detectan puntos de interés en ubicaciones distintivas de las imágenes (esquinas, círculos, junturas en T), cuya propiedad de mayor importancia es la repetitividad bajo diferentes condiciones visuales. El segundo paso del algoritmo es caracterizar el comportamiento de la vecindad de cada punto característico y representarlo con un vector propio de esa vecindad. Este descriptor tiene que ser distintivo y robusto. Finalmente los descriptores de cada par de imágenes son emparejados utilizando la diferencia entre los vectores por medio de la distancia Euclidiana o distancia de Mahalanobis [31].

Detector de Hessiana Rápido (Fast-Hessian Detector): El detector utilizado por el algoritmo surf está basado en la matriz Hessiana debido a que ofrece buenos resultados en cuanto a precisión y tiempo. El detector se representa matemáticamente de la siguiente forma:

Dado un punto $x = (x, y)$ en una imagen I, la matriz Hessiana $H(x, \sigma)$ en x a escala σ se define:

$$H(x, \sigma) = \begin{bmatrix} L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\ L_{yx}(x, \sigma) & L_{yy}(x, \sigma) \end{bmatrix} \quad (2)$$

Donde $L_{xx}(x, \sigma)$ es la convolución de la derivada gaussiana de segundo orden $\frac{\partial^2}{\partial x^2} g(\sigma)$ con la imagen I en el punto x, de igual forma con $L_{xy}(x, \sigma)$ y $L_{xy}(x, \sigma)$.

La matriz Gaussiana es óptima para análisis espacial de las imágenes, según estudios realizados por Koenderink [29]. Sin embargo debido a los muy buenos resultados obtenidos por aproximaciones de las derivada gaussiana, en este algoritmo se utiliza una aproximación utilizando filtros sobre áreas rectangulares como se indica en la figura 3. Estas pueden ser evaluadas rápidamente utilizando la integral de las imágenes.

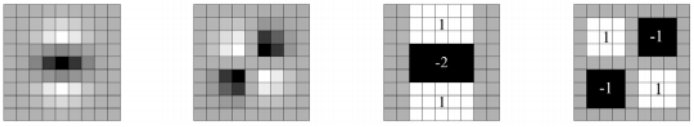


Figura 3 Comparación entre las Derivadas gaussianas de segundo orden y la aproximación realizada por el algoritmo SURF [18].

Descriptor SURF: El vector descriptor se obtiene al analizar una región cuadrada centrada alrededor del punto de interés, esta región se subdivide en regiones de 4x4 y se calculan por medio de las integrales de las imágenes la respuesta al aplicar el filtro *Haar-Wavelet* en direcciones horizontales y verticales, que se denotan con d_x y d_y respectivamente (figura 4).

Luego d_x y d_y se suman a lo largo de cada subregión y se forma un primer conjunto de entradas para el vector característico. A fin de brindar información acerca de la polaridad de los cambios de intensidad, también se extraen la suma de los valores absolutos de las respuestas $|d_x|$ y $|d_y|$, por ende cada subregión tiene un vector característico.

$$V = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$$

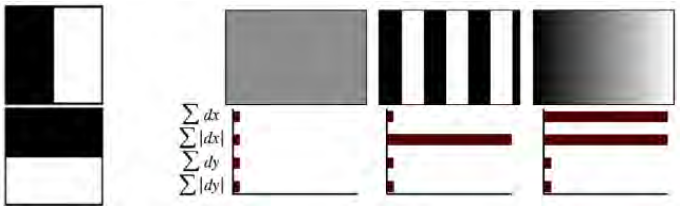


Figura 4 Izquierda: tipos de filtros *Haar-wavelet* utilizados en SURF. Derecha: comportamiento de las diferentes entradas del vector característico frente a diferentes regiones.

E. Descomposición en valores singulares

La descomposición en valores singulares (DVS) puede considerarse desde tres puntos de vista mutuamente compatibles. Por un lado, podemos verlo como un método para transformar variables correlacionadas en un conjunto de los no correlacionados que mejor exponen las diversas relaciones entre los elementos de datos originales. Al mismo tiempo, DVS es un método para identificar y ordenar las dimensiones en las que los puntos de datos muestran la mayor variación. Esto se relaciona con la tercera forma de ver DVS, y es que una vez que tengamos identificado donde está la mayor variación, es posible encontrar la mejor aproximación de los puntos de datos originales utilizando menos dimensiones. Por lo tanto, la DVS se puede ver como un método para la reducción de datos [20].

Para determinar la descomposición DVS de una matriz A es necesario encontrar los valores propios y vectores propios de A multiplicada por su matriz transpuesta A^T. El proceso matemático para encontrar las matrices se muestra a continuación como se presenta en [21].

Sea:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

Y su descomposición en valores singulares:

$$A = U\Gamma V^T$$

Donde:

$$\Gamma = \begin{bmatrix} \sigma 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma n \end{bmatrix}$$

$$\sigma_i = \sqrt{\lambda_i}$$

$$\sigma_i \; valores \; singulares \; de \; A^T A$$

$$\lambda_i \; valores \; propios \; de \; A^T A$$

Γ es la matriz que contiene los valores singulares de $A^T A$ que se calculan a partir de los valores propios. Para determinar los valores propios se resuelve (3), donde I es la matriz identidad y el número de valores propios está relacionado directamente con el grado de la matriz. Si la matriz es de grado N entonces esta tendrá N valores propios.

$$\det(A^T A - \lambda I) = 0 \tag{3}$$

Para determinar los vectores propios de $A^T A$ se resuelve (4) para el vector V por cada valor propio.

$$V \; vectores \; propios \; de \; A^T A$$
$$V = [V_1, V_2, \dots, V_n], \qquad V \in \mathbb{R}^{nx1}$$

$$(A^T A - \lambda I)V = 0 \tag{4}$$

Finalmente es necesario determinar la matriz U que se calcula a partir de la matriz A, los valores singulares y los vectores propios de $A^T A$. Ver ecuación (5).

$$U_i = \frac{AV_i}{\sigma_i} \tag{5}$$
$$i = 1, \dots., n$$

$$U = [U_1, U_2, \dots, U_n]$$

4. DESARROLLO

A. Base para el sensor

El proyecto se basó en el uso del sensor Kinect para obtener la nube de puntos, por lo que se hizo un sistema a partir de este dispositivo. Este sensor presenta los mismos problemas que una cámara fotográfica normal frente al movimiento o *motion blur*, esto puede llegar a distorsionar la calidad de los datos obtenidos. Además debido al uso del algoritmo SURF, que encuentra puntos coincidentes entre dos imágenes en escala de grises, era conveniente tener el dispositivo a una misma altura para que se encontraran más puntos compatibles.

Con el objetivo de evitar inconvenientes, para garantizar estabilidad en el momento de adquirir datos y hacer más fácil la captura de datos, se decidió ubicar el sensor sobre un trípode de 1.2m de alto. Para lograrlo fue necesario crear una base que permita soportar el dispositivo sobre el trípode, como se indica en la figura 5. Además se ubicó un nivel en la base el cual permite ajustar el ángulo del sensor con respecto a la horizontal.

El sistema funciona al mantener fijo el objeto y desplazar el sistema de captura alrededor de éste. Sin embargo, debido a los algoritmos utilizados el sistema funciona del mismo modo al mantener fija la posición del sensor y al rotar gradualmente el objeto de captura.

Figura 5 Sistema de captura.

Así mismo se instaló un dispositivo que permite realizar las capturas desde el trípode, llamado bloque de captura, mejorando la movilidad al no realizar las capturas desde el computador. Además El dispositivo brinda información sobre el número de capturas realizadas, el ángulo en el que se realizan, la existencia de capturas no válidas, y finalmente informa cuando se terminó el proceso.

Todos los desplazamientos que se realizan son respecto al objeto que se esté capturando por lo que se puede tomar a éste como el marco de referencia inercial. Pero debido al algoritmo desarrollado no es necesario realizar transformación alguna para integrar los ejes de coordenadas del objeto y el sistema de captura.

Además se determinó que la integración con sensores que proporcionaran información sobre el desplazamiento no era útil ya que estos presentaron errores bastante grandes para poder ser tenidos en cuenta. El rango de error está en el orden de los centímetros lo cual para una aplicación como esta es bastante grande y afectaría notoriamente el resultado final de la reconstrucción.

Sin embargo si fue utilizado un sensor, una brújula electrónica que se ubica en el bloque de captura. Este sensor permite saber cuál es el ángulo en que se gira el Kinect respecto a su propio eje. Ésta información es utilizada para tener mejores resultados en las capturas ya que es difícil saber cuánto se rota el sensor por simple inspección visual y así se asegura que el dispositivo capture el objeto que se desea modelar en 3D.

B. Bloque de captura

Figura 6 Bloque de captura.

El dispositivo instalado en el trípode fue realizado con los siguientes componentes:

Arduino Mega: Permite la interacción entre el computador, el sensor y el bloque de captura.

CMPS10: indica el Angulo en el que el sensor realiza una captura.

LCD 16x2: permite visualizar información acerca del proceso de captura.

El tipo de información que ofrece este dispositivo se indica en la figura 7.

Figura 7 Visualización LCD.

Información observada desde el bloque de captura. Superior izquierda: En espera. Superior derecha: indica el ángulo actual y el número de capturas realizadas. Inferior izquierda: Captura no valida, muestra el mensaje “ERROR 01 No Suficientes Coincidencias”. Inferior derecha: indica que finalizo el proceso.

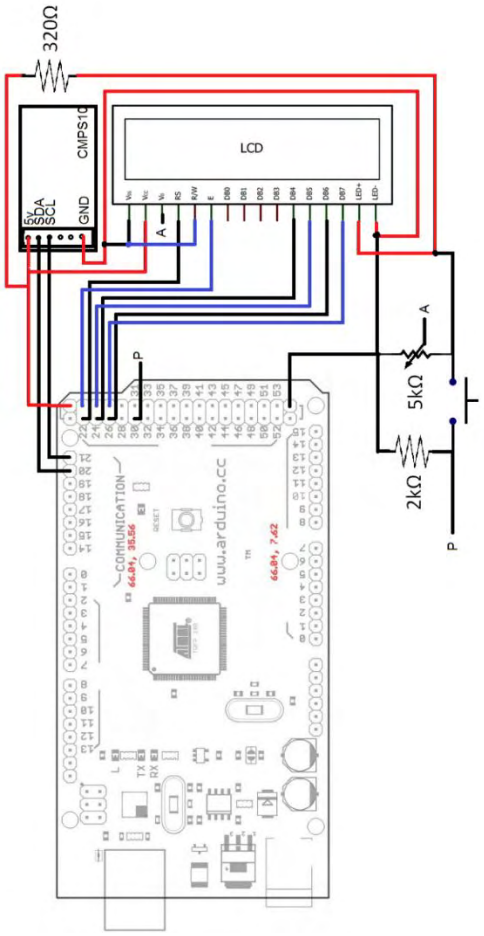


Figura 8 Esquema electrónico bloque de captura.

El esquema electrónico del bloque de captura se puede observar en la figura 8. El bloque de captura se comunica con el computador de forma serial por medio del puerto USB. De esta manera puede recibir y enviar la información necesaria para realizar las capturas desde la posición en que se encuentre el sensor. Por otro lado, el bloque de captura tiene un único botón, el cual le permite al usuario realizar una captura al presionarlo, o finalizar el proceso de capturas si se presiona durante un periodo mayor a 1.5 segundos.

5. ALGORITMO DE RECONSTRUCCIÓN

El sistema se rota en el plano *xz* para hacer un barrido de 360°. El algoritmo de reconstrucción está dividido de la siguiente forma:

1. Adquisición de datos.
2. Búsqueda de puntos coincidentes.
3. Acople de nubes de puntos.
4. Filtro.
5. Gráficas.

Para la primera parte se usó la librería mex de Matlab^{R} que permite la interacción del sensor de captura con éste programa. En la segunda parte se utilizó SURF para seleccionar puntos que coincidieran entre un par de imágenes. Estos puntos se utilizaron en la tercera parte, el acople de las nubes de puntos. Las secciones 4 y 5 son de filtrado y explicación de cómo se realizaron las gráficas. Finalmente se integró todo en una interfaz gráfica para tener un mejor manejo del programa. A continuación se detalla mejor cada parte del algoritmo.

A. Búsqueda de puntos coincidentes

Uno de los pasos del algoritmo ICP es la selección de datos entre un par de nubes de puntos a los que se les aplica el proceso de minimización y traslación. En éste sistema estos datos se escogieron con ayuda del algoritmo SURF.

Los datos almacenados en las diferentes matrices contienen la información espacial de los puntos del objeto capturado. Estos datos varían de acuerdo al perfil del objeto que se está tomando y no siempre los datos útiles están en las mismas ubicaciones ni en la misma cantidad, a pesar que las matrices resultantes en cada captura si tienen las mismas dimensiones.

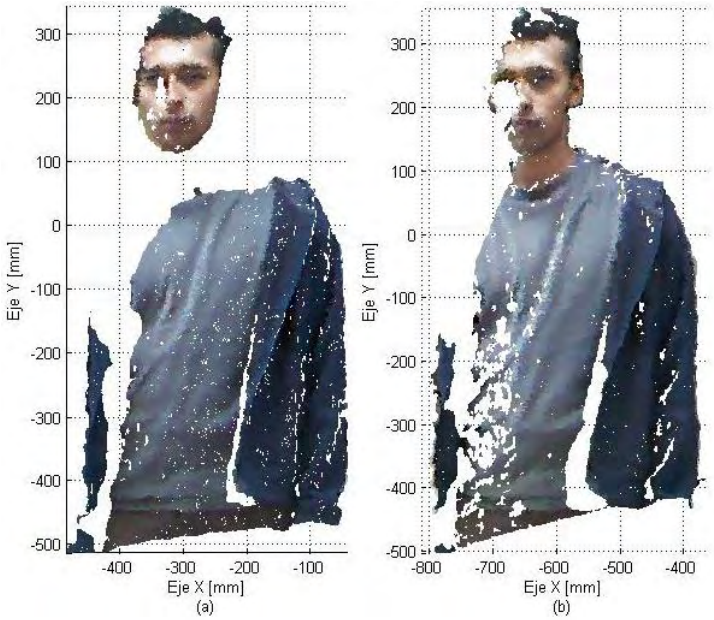


Figura 9 Capturas sobre perfil frontal con 15° de rotación entre (a) y (b).

En la figura 9 se observan 2 capturas de una vista frontal. Entre (a) y (b) se rotó el sensor 15° por lo que las nubes de puntos son diferentes. En (a) existen más zonas blancas que en (b). Por ejemplo en la zona del cuello entre 50 y 100 mm en el eje Y se puede observar que (a) está en blanco y (b) tiene el cuello presente. Debido a diferencias como está el número de puntos de una captura es diferente al de otra pero las híper-matrices que contienen los datos de cada una tienen las mismas dimensiones que son 480x640x3, esto debido a la resolución del sensor que es de 480x640. Los puntos que se obtienen sin datos son tomados como NAN (*Not A Number*) y las matrices de capturas están compuestas por datos NAN y puntos de la nube.

Para poder aplicar el algoritmo ICP es necesario que exista igual número de datos seleccionados entre un par de nubes de puntos. Sin embargo como se observa anteriormente, las nubes de puntos obtenidas con el sensor no siempre son iguales por lo que es necesario hacer una selección adecuada. Para encontrar estos sitios de coincidencia se utilizó el algoritmo SURF, este método utiliza imágenes en escala de grises por lo que se utilizaron las imágenes RGB que el sensor entrega y con la ayuda de la función *rgb2gray()* de Matlab^{R} se convirtieron en imágenes en el formato requerido. Dependiendo de las condiciones de luz que se tengan en el lugar donde se realizan las capturas cambia el contraste en las imágenes en escala de grises, por tanto el número de puntos coincidentes es diferente de un par de capturas a otra.

Con el fin de utilizar el algoritmo se encontró las coincidencias o puntos característicos correspondientes entre un par de imágenes en escala de grises. Por ejemplo, se tomaron las imágenes 1 y 2 propias a sus respectivas capturas y se encontraron puntos coincidentes entre ellas. Así mismo para las imágenes 2 y 3, 3 y 4, etc. Este proceso se realizó hasta terminar con todas las capturas realizadas.

Las posiciones de las imágenes RBG tienen correspondencia con los datos de su respectiva nube de puntos, esto quiere decir que cada pixel de las imágenes en RGB tiene asociado un dato en la nube de puntos. De esta forma es posible saber qué punto de una nube coincide con un punto en otra nube, al comparar las coincidencias en las imágenes en escala de grises. En (6) y (7) se expresa esta relación.

$$posiciones\ de\ la\ imagen\ \ RGB\ (x_i, y_j)$$
$$posiciones\ de\ los\ datos\ en\ nube\ de\ puntos\ (x'_i, y'_j)$$

$$x_i = x'_i \tag{6}$$
$$y_j = y'_j \tag{7}$$

$$PTSC_k \in \mathbb{R}^{nx2} \tag{8}$$

Donde las matrices $PTSC_k$ contienen las coordenadas de los puntos característicos de las imágenes RGB, k representa la captura correspondiente y n indica el número de coincidencias encontradas para esa captura. Para cada captura se tendrá 2 matrices PTSC asociadas excepto la primera y última imagen. Para interpretar mejor esto se puede asumir que *k* representa el número de una captura realizada sobre un objeto al cual se le hicieron *N* capturas, si *k* es 1 entonces solo se puede encontrar puntos característicos con la imagen siguiente pues no hay una captura anterior igualmente si *k=N* no habrá una captura siguiente, pero con *k≠1* y *k≠N* se pueden encontrar puntos característicos con las capturas anterior y posterior a *k*. De tal forma que se tendrán *2N-2* matrices PTSC.

Cada par (x_i, y_j) representa un pixel de la imagen RGB y cada par (x'_i, y'_j) representa la ubicación de un dato en la matriz generada por el sensor. Aunque se cumple (6) y (7) se utiliza la notación (x', y') para indicar que un conjunto de coordenadas pertenece a la imagen RGB y las otras pertenecen a la matriz con la información de la nube de puntos.

La figura 10 muestra una captura con una grilla que divide la imagen. Asumiendo que cada casilla es un pixel la marcada con el número 1 sería el primer pixel de la imagen en esa casilla se almacenaran los valores RGB que darán color a ese pixel y para la misma casilla en una matriz de datos de profundidad se almacenarán las coordenadas $(x_k, y_k, z_k,)$ de un punto de la nube. Este punto tendrá como marco de referencia la posición actual del kinect en la captura k. Si el punto no tiene datos, entonces los valores de $(x_k, y_k, z_k,)$ serán NAN. Gracias a esta correspondencia entre las imágenes RGB del sensor y la matriz de datos de profundidad, es posible saber cuáles son los puntos coincidentes en las nubes de puntos a partir de la aplicación de SURF en las imágenes RGB convertidas a blanco y negro.

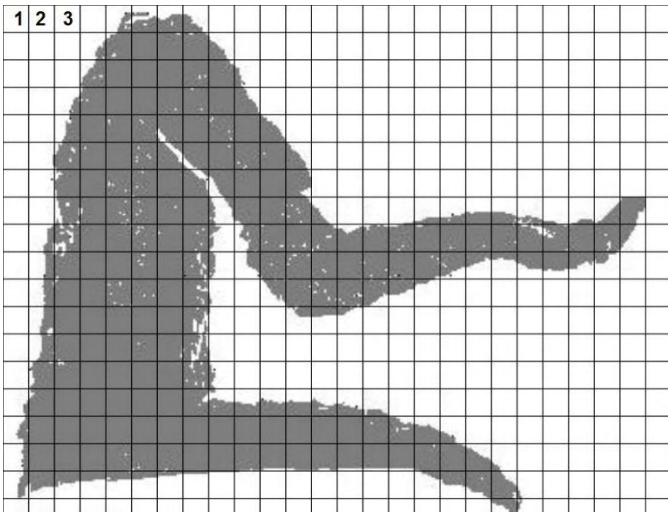


Figura 10 Pixeles de la imagen RGB

La figuras 11 y 12 muestran los puntos coincidentes entre una imagen asociada con una nube de puntos y otra imagen igualmente asociada de capturas consecutivas. Teniendo claro cuáles son los puntos coincidentes entre una captura y otra es mucho más fácil trasladar estos puntos.

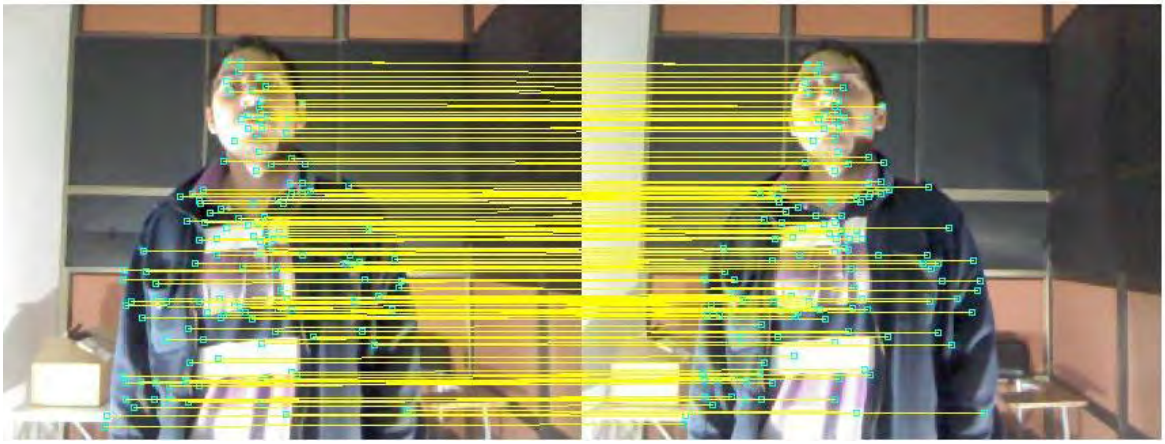


Figura 11 Coincidencias en imágenes RGB.

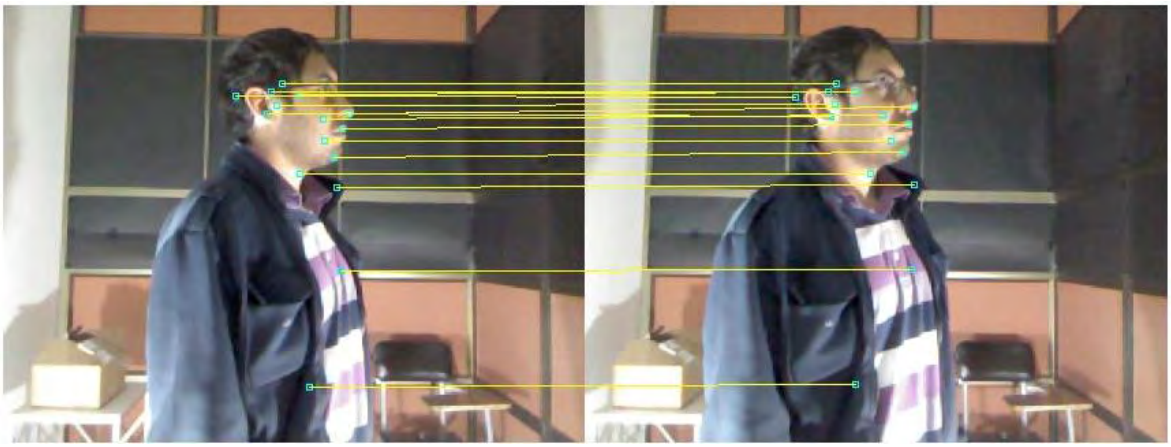


Figura 12 Coincidencias en imágenes RGB.

SURF se usó para encontrar puntos característicos de las imágenes RGB y conociendo estos puntos se determinaron sus posiciones en la matriz donde se almacenaron. Las posiciones de estos puntos característicos se comparten en la matriz de datos de profundidad. Conociendo estos puntos se puede encontrar las transformaciones R y T entre ellos.

B. Acople de nubes de puntos con el algoritmo ICP

Como se dijo en una sección anterior, ICP es un método iterativo para encontrar una matriz de rotación y un vector de traslación que permita el acople de 2 nubes de puntos para formar un solo objeto. Este método necesita 2 matrices de datos que contienen información de las coordenadas de los puntos escogidos de 2 nubes, los cuales deben tener las mismas dimensiones.

Sea L la matriz que contiene todos los datos de profundidad de la captura hecha por el sensor con dimensiones de 480x640x3 y M la matriz con los puntos característicos de la imagen RGB propia de esa captura.

$$M = [PTSC_{k,x}\,, PTSC_{k,y}\,]$$

Entonces los puntos correspondientes, almacenados en PC, en el espacio tridimensional de los datos de la matriz L en la captura k serán:

$$PC_k(x) = L(Mx\,, My, 1)$$
$$PC_k(y) = L(Mx\,, My, 2)$$
$$PC_k(z) = L(Mx\,, My, 3)$$

Donde *Mx* y *My* representan la posición en la matriz L de los puntos coincidentes, y $PC_k(x)$, $PC_k(y)$ y $PC_k(z)$, representan las coordenadas en los ejes *x*, *y*, *z* de dicha coincidencia. Estas matrices PC_k son las que se utilizan como puntos escogidos en el

algoritmo ICP y son a los que se les aplica el proceso de acople. Es decir, entre ellas se encuentran las matrices R y T.

Debido a que estos puntos ya son correspondientes no es necesario un proceso iterativo en el algoritmo de ICP, ya que teóricamente al realizar la transformación el error de acople será mínimo. Por este motivo el proceso aplicado se denominó ICP degenerado pues no se realizan iteraciones para determinar la matriz R y el vector T gracias a que se conocen cuáles son los puntos coincidentes entre un par de imágenes y por tanto entre un par de nube de puntos. Entonces los pasos que se siguieron fueron los siguientes.

1. Ingreso de puntos coincidentes.
2. Calculo de la matriz de rotación R y del vector de traslación T.
3. Aplicación de R y T al grupo de puntos objetivo.

Por otro lado, para encontrar la matriz de rotación R y el vector de traslación T no se necesita un proceso iterativo para minimizar el error entre los respectivos puntos escogidos. En la práctica se encontraron mejores resultados al aplicar una descomposición de matrices conocido como DVS (Descomposición en Valores Singulares) o SVD por sus siglas en ingles [19]. Por esta razón el proceso aplicado se nombró ICP degenerado.

En este proceso es necesario calcular el "centro geométrico" de las capturas que en el fondo es el promedio para cada coordenada, a continuación se presenta el desarrollo matemático para determinar R y T que minimicen el error cuadrático que se obtiene a partir de (1).

Sean:

$$M = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{bmatrix} \qquad M' = \begin{bmatrix} x'_1 & y'_1 & z'_1 \\ x'_2 & y'_2 & z'_2 \\ \vdots & \vdots & \vdots \\ x'_n & y'_n & z'_n \end{bmatrix}$$

Matrices que contienen las coordenadas de 2 nubes de puntos. Donde las x_i son las coordenadas para el eje "x", y_i son las coordenadas para el eje "y" y z_i son las coordenadas para el eje "z" de una de las nubes de puntos, de igual forma x'_i, y'_i, z'_i son las coordenadas para los respectivos ejes de la otra nube de puntos, con $i = 1, \dots, n$ donde *n* es la cantidad de puntos coincidentes de la nube.

$$C = \left[\frac{1}{n} \sum_{i=1}^{n} x_i \ , \quad \frac{1}{n} \sum_{i=1}^{n} y_i \ , \quad \frac{1}{n} \sum_{i=1}^{n} z_i \right]$$

$$C' = \left[\frac{1}{n} \sum_{i=1}^{n} x'_i \ , \quad \frac{1}{n} \sum_{i=1}^{n} y'_i \ , \quad \frac{1}{n} \sum_{i=1}^{n} z'_i \right]$$

Los vectores fila $C\ y\ C'$ contienen los "centros geométricos" para las coordenadas x, y, z y x', y', z'. Estos centros se obtienen haciendo la sumatoria para todas las coordenadas en cada eje y dividiendo esta suma entre en el número de datos *n*.

Llamamos M_i a una de las filas de M y de la misma forma M_i' a una de las filas de M' donde i representa el número de la fila iniciando en 1 y finalizando en n.

Se definen las nuevas matrices:

$$F_i = M_i - C$$
$$F_i, M_i, C \in \mathbb{R}^{1x3}$$
$$i = 1, \dots, n$$

$$F'_i = M'_i - C'$$
$$F'_i, M'_i, C' \in \mathbb{R}^{1x3}$$
$$i = 1, \dots, n$$

$$F = \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_n \end{bmatrix} \qquad F' = \begin{bmatrix} F'_1 \\ F'_2 \\ \vdots \\ F'_n \end{bmatrix}$$

$$F, F' \in \mathbb{R}^{nx3}$$

Donde las matrices *F* y *F'* están compuestas por los vectores fila F_i y F_i'. Las marices $F\ y\ F'$ contienen los datos con los que se encontrará la descomposición en valores singulares para lo que se define la matriz W como se indica a continuación.

$$W = F^T F' \quad \Rightarrow W \in \mathbb{R}^{3x3}$$
$$W = U\Gamma V^T \tag{9}$$
$$R = UV^T \tag{10}$$
$$T' = C - RC' \tag{11}$$

Como se indica en la ecuación (9), solo es necesario hacer la descomposición en valores singulares para poder calcular R y T' como se muestra en (10) y (11). Pero el vector de traslación que se obtiene no es el que finalmente se usará, pues debido a que se conocen los datos que son coincidentes en las 2 nubes de puntos es más fácil calcular la traslación en cada eje haciendo una resta. Pero antes es necesario aplicar la matriz de rotación que se obtuvo mediante la ecuación (12).

$$M''^T = RM'^T \tag{12}$$

$$M'' = \begin{bmatrix} x''_1 & y''_1 & z''_1 \\ x''_2 & y''_2 & z''_2 \\ \vdots & \vdots & \vdots \\ x''_n & y''_n & z''_n \end{bmatrix}$$

La nueva matriz M'' contiene los nuevos puntos, y debido a que las posiciones no se han afectado sino solamente la rotación del objeto, los puntos coincidentes siguen estando en los mismos lugares en la matriz, por lo que a partir de las matrices PTSC es posible saber la ubicación en cada nube de puntos y realizar la resta en cada eje para saber cuál es la traslación entre una nube y otra. A continuación se muestra el procedimiento.

$$xp_i = x_i - x''_i$$
$$yp_i = y_i - y''_i$$
$$zp_i = z_i - z''_i$$
$$i = 1, \dots, n$$

Se hace la resta de los puntos en cada nube para poder conocer cuál es la traslación que hay en cada eje. Después se encuentra el promedio de traslación en cada eje y así se obtiene el vector T que permitirá acoplar las 2 nubes de puntos.

$$T = \left[\frac{1}{n}\sum_{i=1}^{n} xp_i \ , \frac{1}{n}\sum_{i=1}^{n} yp_i \ , \frac{1}{n}\sum_{i=1}^{n} zp_i \right]$$

$$Mt_i = M''_i + T \tag{13}$$

Donde el subíndice *i* indica el número de la fila a la que se le resta el vector de traslación *T* el cual debe ser un vector fila para poder realizar la operación. Se podría decir que una vez aplicada la ecuación (13) se ha terminado con el proceso y se ha logrado ubicar exitosamente los puntos de una nube de forma que coincida con la otra. Sin embargo se debe recordar que los puntos M y M' son datos seleccionados de un par de capturas, por lo que se deben aplicar las matrices de rotación y traslación R y T a los datos de las capturas sucesivas como se muestra en (14).

$$Ln = RL' + T \tag{14}$$

$$Ob = [L \ , Ln, \dots . , Lf] \tag{15}$$

$$L, Ln \ \in \ \mathbb{R}^{qx3}$$

Donde *L* es la matriz "objetivo" que contiene todos los puntos entregados por el sensor, por lo que tiene mayor número de filas que M. L' es la matriz que se rotará y trasladará, y tiene iguales dimensiones que *L*. *Ln* es la matriz resultante después de aplicar la rotación y

traslación. Estas matrices entregadas por el sensor tienen un gran número de datos inutilizables que son los que aparecen como NAN y deben ser filtrados. La ecuación (15) presenta como se organizan todas las marices resultantes donde *Lf* es la matriz correspondiente a la última captura después de haber aplicado rotación y traslación y la matriz *Ob* es la que contiene todos los puntos listos para ser graficados.

Como uno de los objetivos del trabajo era hacer la reconstrucción en 360° alrededor del eje Y de un objeto, fue necesario analizar la forma como se hacía la reconstrucción. Es decir que capturas se iban a comparar entre si y cuál sería el punto de partida.

La figura 13 representa el análisis de reconstrucción. Lo que se hizo fue realizar varias capturas en un rango de 360° alrededor del eje Y. Cada captura está numerada en la figura desde la primera hasta la octava. Teniendo estos datos se aplicó el proceso de reconstrucción desde 5, relacionando las capturas en pares, por ejemplo se tomó 5 y 4 y se encontró R y T para ese par de nubes de puntos, después 4 y 3, 3 y 2, y finalmente 2 y 1. Por el lado izquierdo se tomó 6 y 7, 7 y 8. Tratando de "arrastrar" las capturas hacia el "frente" que sería la primera captura.

Para que este proceso de "arrastre" funcionara se tenía que aplicar el R y el T de 2-1 también a 3-2, 4-3 y a 5-4, El R y T de 3-2 se aplica a 4-3 y a 5-4 y se aplica el mismo procedimiento dependiendo de cuantas capturas se hayan hecho, de igual forma el R y T de 7-8 se aplica a 6-7.

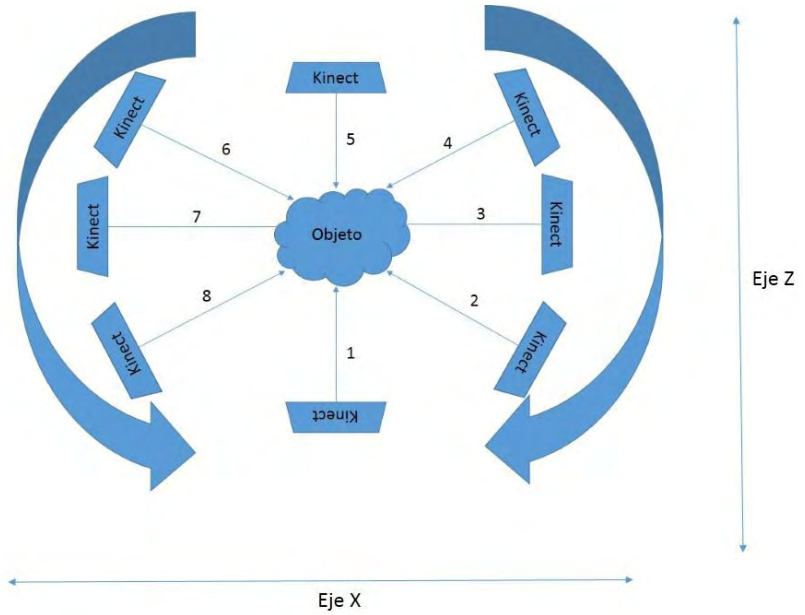


Figura 13 Capturas en 360° vista desde el plano (X,Z).

C. Filtro

Después de obtener la matriz con todos los puntos fue necesario filtrar los datos que no aportaban información útil para la reconstrucción y los datos que afectaban el resultado final de las capturas. Se eliminaron todos los datos que aparecían como NAN y los datos de los bordes de las capturas, pues estos bordes generalmente distorsionaban la reconstrucción final, ya que por las condiciones de luz estos aparecían muy oscuros y al tomarlos se obtenía un modelo final con fallas muy notorias.

Figura 14 Captura lateral.

La figura 14 muestra un ejemplo de bordes con información inválida. Al frente de la imagen en la zona de la nariz hay una especie de sombra que recubre la imagen, estos datos deben ser filtrados para que el resultado no tenga fallas o manchas que impidan su correcta visualización. Para filtrar estos puntos se realizó un algoritmo que realiza una búsqueda fila por fila de la matriz de datos buscando el máximo y el mínimo para cada fila (matriz de 640x480, resolución del sensor).

Además también fue necesario eliminar puntos repetidos que no aportaban información nueva para la reconstrucción.

Debido a que las diferentes capturas se rotan y trasladan, es lógico que algunos de los datos sobre todo los puntos coincidentes se repitan con los de otras capturas, y filtrar estos puntos hace que la matriz de reconstrucción final tenga dimensiones menores sin que se afectara el modelo reconstruido. Para esto se usó de la función *unique()* de Matlab^{R} junto con 2 parámetros llamados *'first' y 'rows'* que se ingresan en la función permitió encontrar las posiciones de los puntos que tenían un valor único, y la posición del primer dato repetido

que encontrara, borrando los demás. Es decir, si un dato estaba repetido 4 veces la función solamente toma el primero que encuentre despreciando los 3 restantes.

D. Triangulación

Para obtener una superficie a partir de una nube de puntos es necesario hacer una triangulación. Para esta parte se tomaron grupos de 3 puntos que formaran un triángulo para obtener una superficie que cubriera el espacio entre esos puntos.

Matlab^{R} cuenta con funciones de triangulación llamadas *delaunay y delaynayn*, que forman triángulos a partir de nubes de puntos. La primera toma grupos de 3 puntos alrededor de los ejes que le sean indicados y hace todos los triángulos posibles en esos ejes. La segunda, que toma grupos de 4 puntos, realiza triángulos a partir de polígonos de 4 lados, y los divide con una línea transversal con 2 esquinas de la superficie formada para todos los ejes sin importar que tan lejos o cerca puedan estar los puntos. La figura 15 explica un poco la toma de grupos de puntos para cada triangulación.

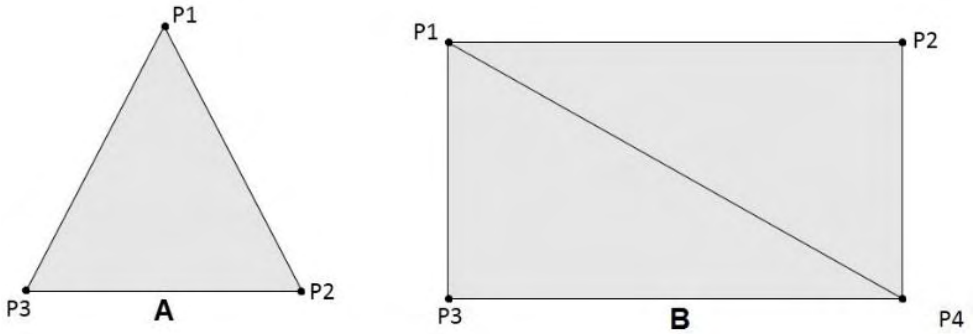


Figura 15 A) triangulación *delaunay.* B) triangulación *delaunayn*.

En este caso se utilizó la función *delaunayn*, pero como se mencionó esta función realiza triángulos en todos los datos sin tener en cuenta la distancia entre los puntos. En otras palabras hace triangulación entre todos los puntos de la nube, caso que no es deseado pues esto no permite una adecuada visualización del modelo como se puede ver en la figura 16.

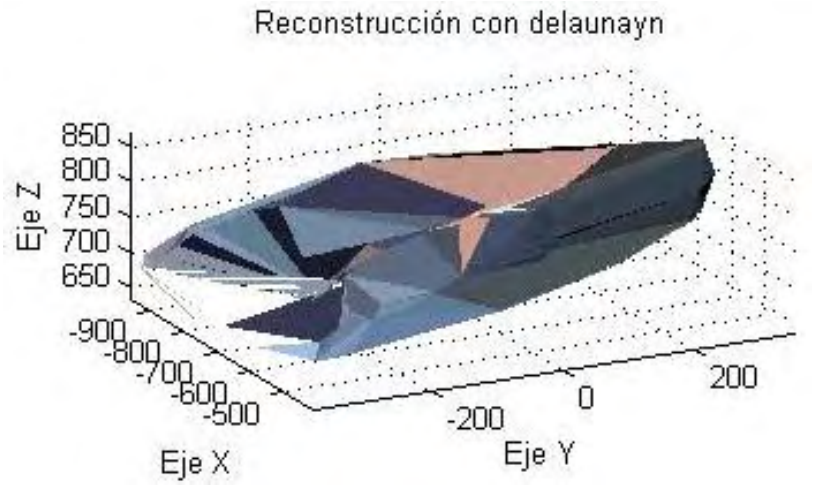


Figura 16 Resultado de reconstrucción con delaynayn.

Con el fin de corregir este problema se realizó un filtro que elimina los triángulos con lados muy grandes, y deja solamente los triángulos más pequeños de forma que el modelo se pareciera mucho más a la forma real. Este filtro se puede modificar en la interfaz gráfica que se realizó a través de un parámetro que indica que longitudes son aceptables. Para encontrar las longitudes de los triángulos se utilizó la distancia euclidiana. Ecuación (16).

$$D = \sqrt{(x_i - x_j) + (y_i - y_j) + (z_i - z_j)} \tag{16}$$

Donde j e i representan el índice del punto al que pertenecen las coordenadas (x,y,z) y D es la distancia o longitud.

E. Gráficas

Para poder graficar los datos como un sólido se necesitan la matriz de triangulación y la nube de puntos. Sin embargo como se cuenta con la imagen RGB es posible utilizar estos datos para poder graficar el modelo con los colores reales del objeto, utilizando la función *colormap*() de Matlab^{R}. Para asegurar que los colores correspondieran a cada punto con el que estaban relacionados, todos los filtros fueron aplicados a la matriz RGB y todos los datos de color se organizaron del mismo modo que los datos de profundidad, de esta forma al final se obtuvo un modelo 3D a color. La figura 17 muestra una reconstrucción a color y otra en sin datos de color en un tono gris.

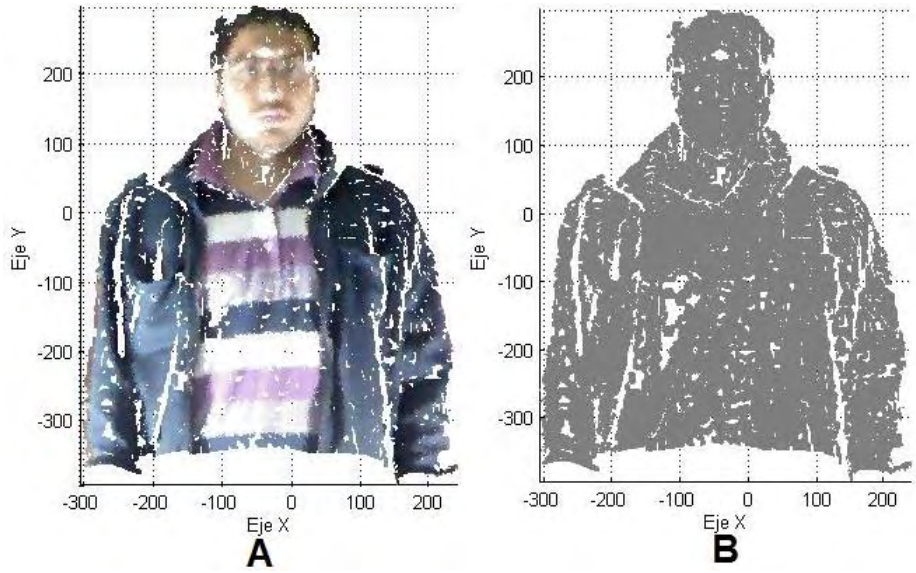


Figura 17 en A se muestra una de las capturas a color y en B los mismos datos pero esta vez en un solo color.

Es claramente visible la mejoría que hay con reconstrucciones a color, ya que en B escasamente se nota la figura y en A se puede apreciar de mejor manera el modelo en 3D.

F. Interfaz gráfica

Al iniciar la reconstrucción se presenta una interfaz que facilita el uso del sistema por cualquier persona. En esta etapa del programa se pueden definir las variables necesarias, y posteriormente al presionar el botón Comenzar, se inicia el proceso de captura y reconstrucción. Una vez iniciado el proceso, se visualiza la imagen que percibe el sensor en tiempo real, mientras se realizan las capturas desde el bloque de captura.

Figura 18 Panel de selección de variables.

En el panel indicado de selección se definen los valores de las variables necesarias para procesar la información, las cuales se describen a continuación:

***Puerto COM**:* Define el puerto USB en el cual se encuentra conectado el dispositivo Arduino, el cual se puede obtener por medio del administrador de dispositivos de Windows.

Número Mínimo de Puntos Coincidentes: Define la cantidad mínima de coincidencias entre pares de imágenes para tomar una captura como válida.

Filtro de Puntos Coincidentes: Define cuantos pixeles se encuentra distanciado un punto característico de una imagen con respecto a la segunda, para tomarlo como punto valido. A menor valor se encuentra menor cantidad de puntos coincidentes, pero presentan menores errores de coincidencia en las 2 imágenes. A mayor valor encuentra mayor cantidad de puntos coincidentes, pero aumenta la posibilidad de que sean erróneos.

***Valor Filtro Profundidad**:* Define la profundidad máxima de toma de datos del sensor.

***Valor Filtro de Datos Contorno**:* Define la distancia de filtrado desde el contorno de las capturas. Los datos que se encuentran en los límites de cada captura son los que presentan mayor cantidad de ruido.

Filtro de Valores Repetidos: Define el radio en el que solo puede existir un punto. Debido a que al unir varias capturas muchos puntos se superponen, esta variable permite eliminar

los valores repetidos permitiendo un mejor manejo de los datos. En la práctica el valor indicado inicialmente (2.5 mm) ofrece muy buenos resultados al disminuir la gran cantidad de datos repetidos en las capturas, sin afectar visiblemente la calidad de la reconstrucción final.

Filtro de Triangulación: Define el valor máximo que puede tener un lado en los triángulos producto de la triangulación.

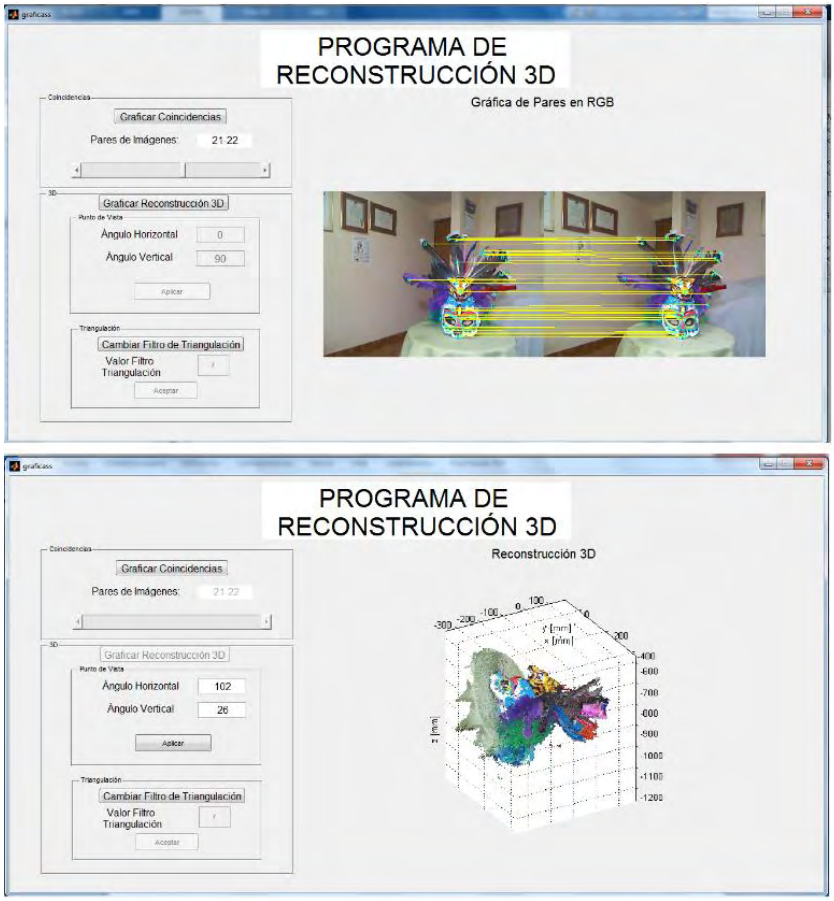


Figura 19 Ventana de visualización de resultados.

Una vez finalizado el procesamiento de los datos, se abre una nueva ventana la cual presenta los resultados del proceso como se indica en la figura 19. En esta etapa se puede observar las coincidencias entre cada par de imágenes, además de la reconstrucción 3D realizada.

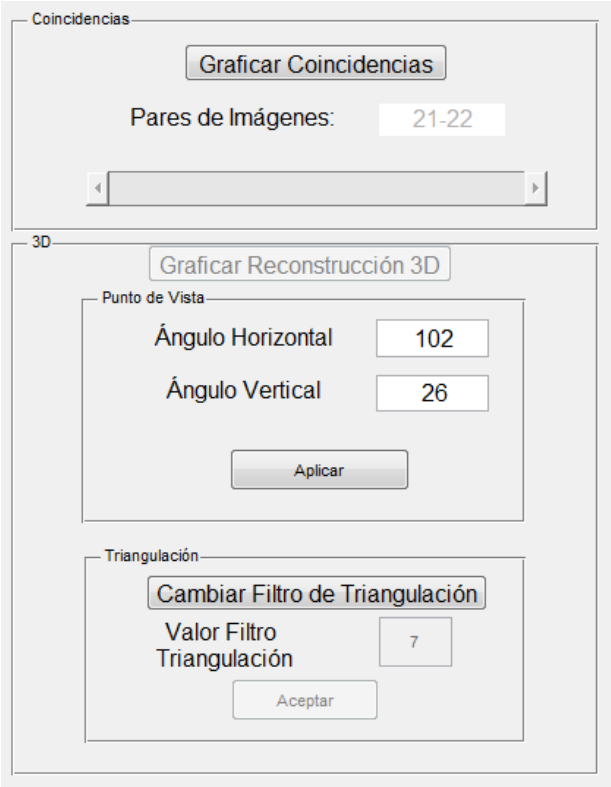


Figura 20 Paneles de coincidencia y 3D.

En la parte izquierda de la ventana se encuentran los paneles de coincidencias y panel 3D como se indica en la figura 20.

En el panel de coincidencias, se puede escoger fácilmente por medio del deslizador, los pares de imágenes que se deseen visualizar, y posteriormente se grafican automáticamente con sus respectivas coincidencias. Por otra parte, en el panel 3D se puede graficar la reconstrucción 3D que se obtuvo, escoger el punto de vista, y además permite cambiar el tamaño máximo de los triángulos producto de la triangulación de *Delaunayn.*

6. VALIDACIÓN

A continuación se procede a comparar los datos obtenidos mediante una reconstrucción con los datos reales del objeto.

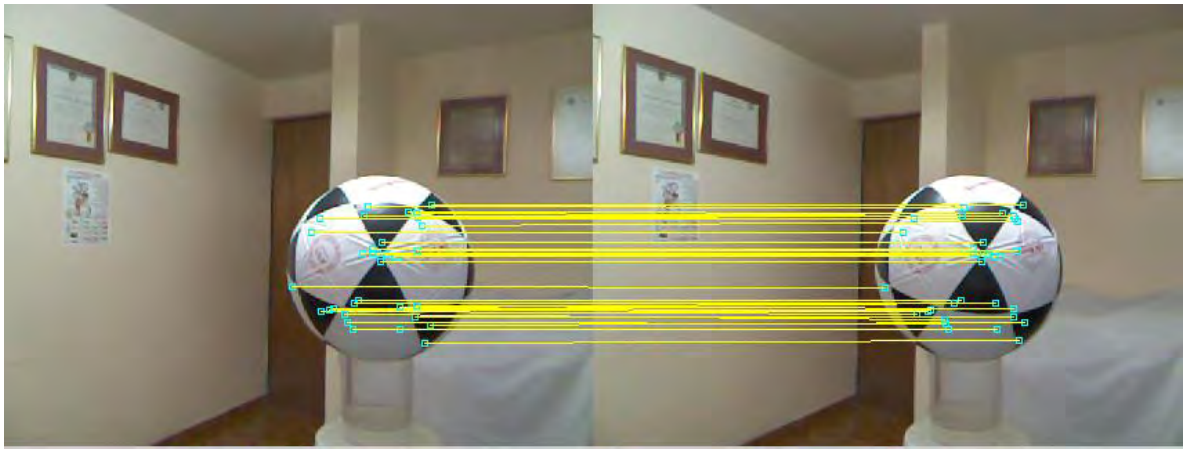


Figura 21 Gráfica de un par de imágenes del balón con sus respectivas coincidencias.

Figura 22 Medición diámetro del balón.

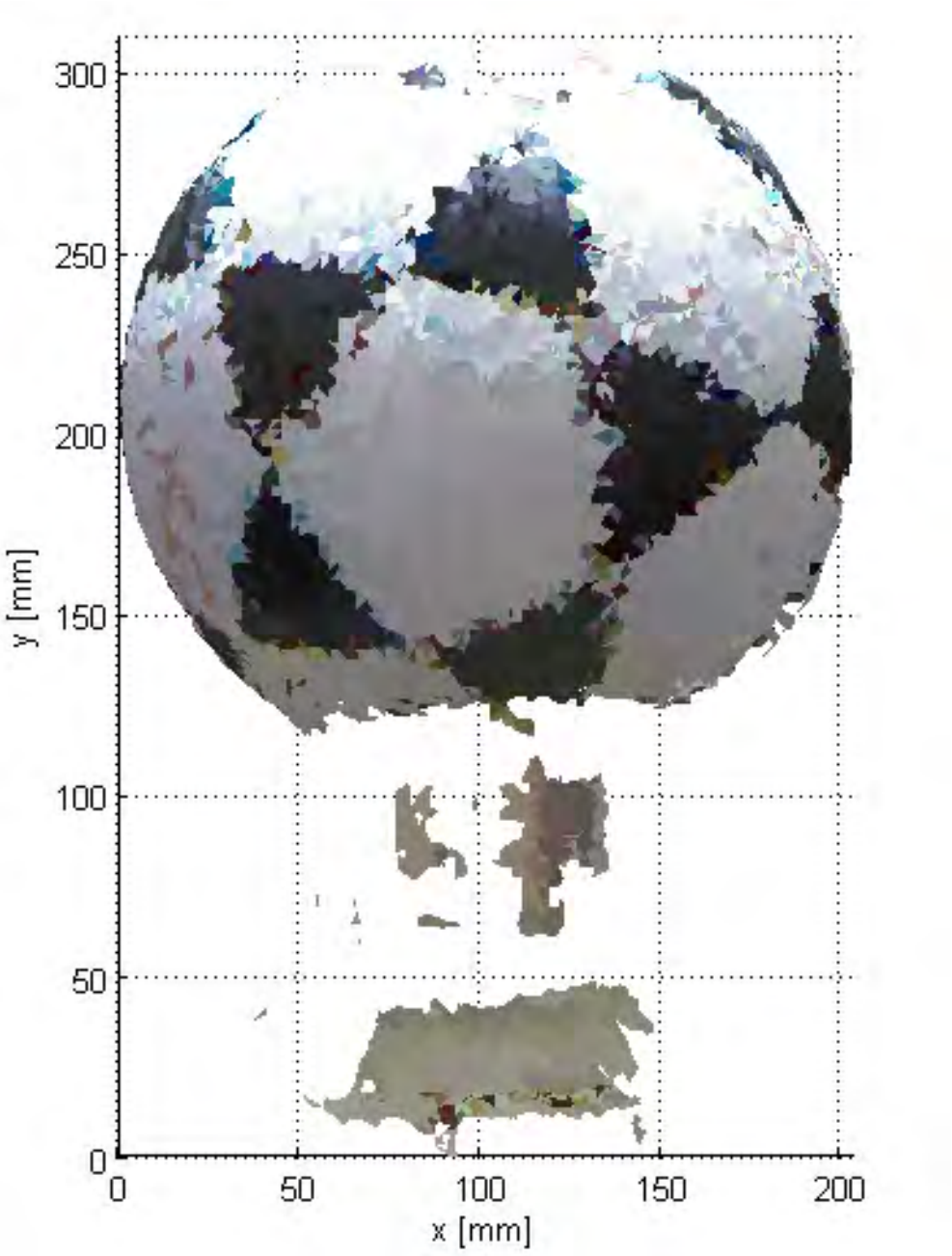


Figura 23 Reconstrucción del balón en 360°, con filtro de triangulación 5. Vista frontal.

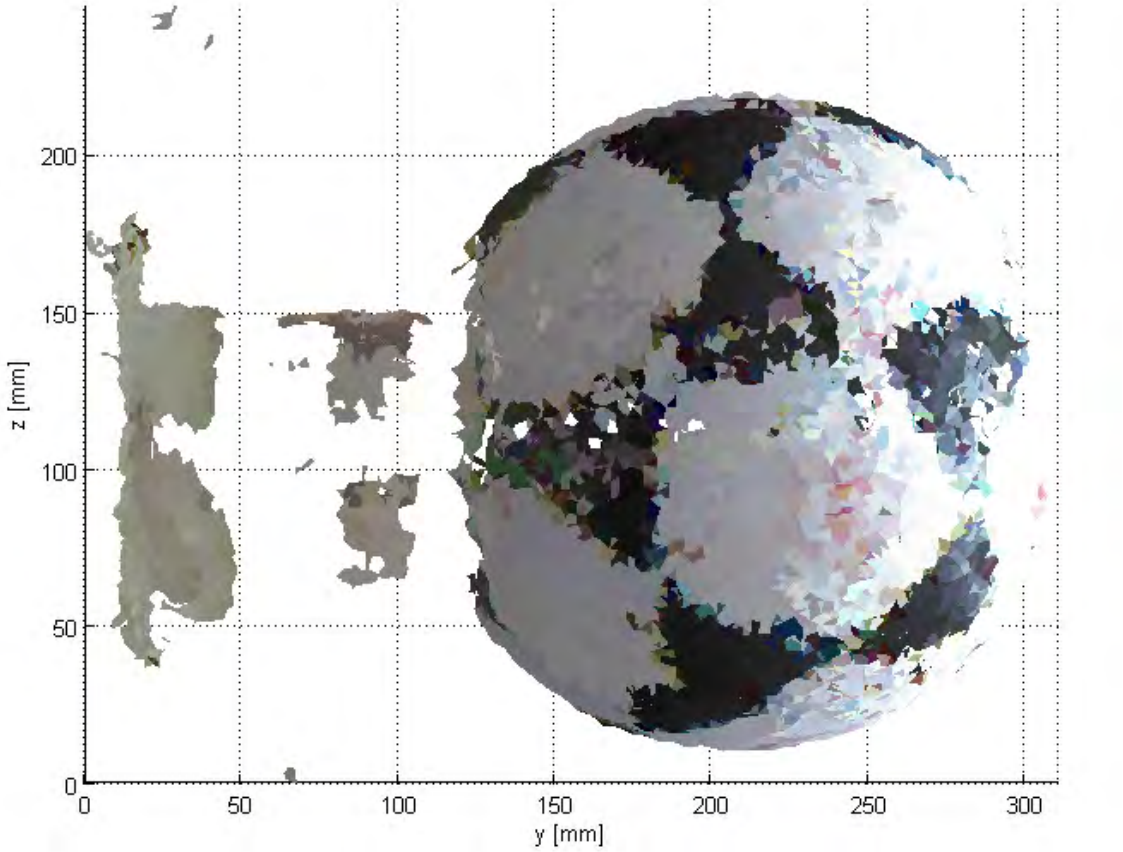


Figura 24 Reconstrucción del balón en 360°, con filtro de triangulación 5. Vista lateral.

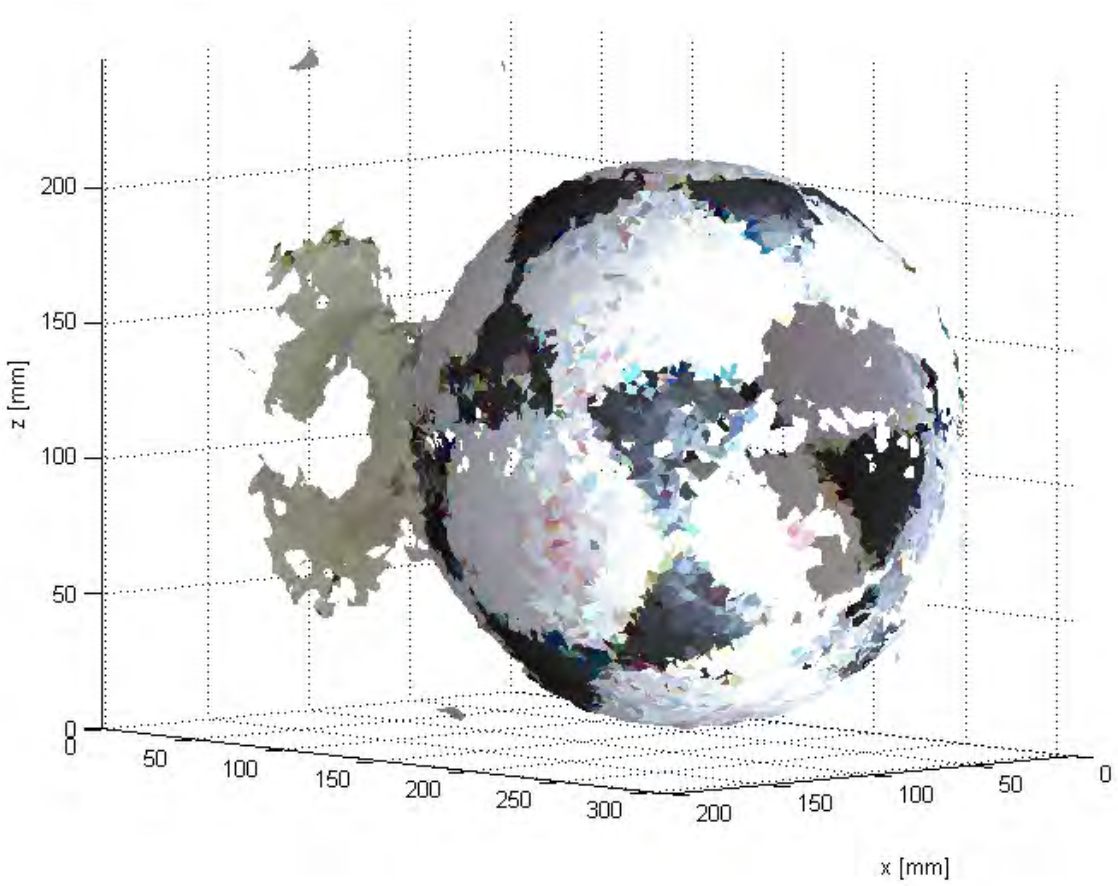


Figura 25 Reconstrucción del balón en 360°, filtro de triangulación 5.

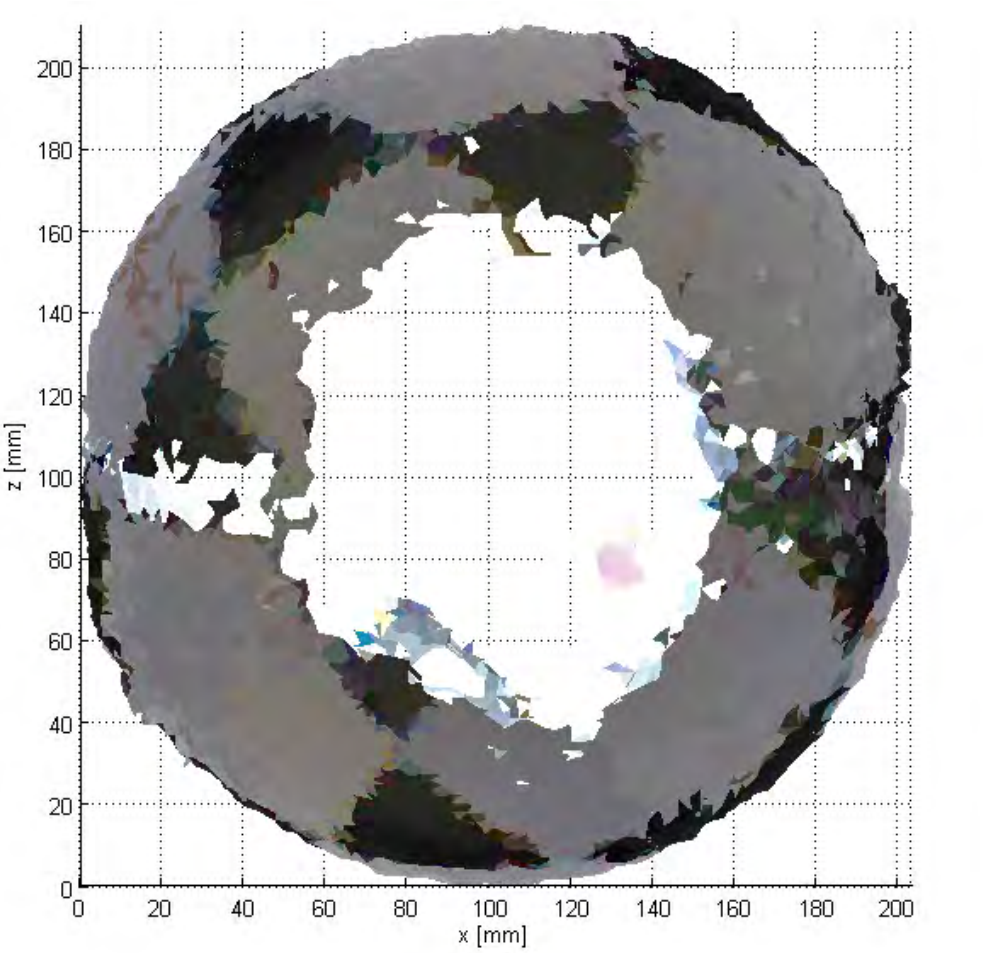


Figura 26 Reconstrucción del balón en 360°, vista superior, filtro de triangulación 5.

$$Centroide\ Balón = CB = \frac{1}{N}\left[\sum_{i=1}^{N} xi \quad , \sum_{i=1}^{N} yi \quad , \sum_{i=1}^{N} zi\right] \tag{17}$$

$$radio = \frac{1}{N}\sum_{i=1}^{N} d_E(M_i, CB) \tag{18}$$

N: número de datos en la nube de puntos M.
xi, yi, zi: coordenadas de M en la posición i.

El radio del objeto de reconstrucción se obtuvo al encontrar el centro geométrico de los puntos (17), eliminando los datos de la base del balón, y calculando la distancia promedio de todos los puntos al centroide (18). Los resultados se pueden observar en la tabla 1.

TABLA I

COMPARACIÓN DE DIÁMETROS: REAL VS RECONSTRUCCIÓN

	MEDIDO	Reconstrucción	Error relativo (%)
Diámetro	21.5 cm	20.4 cm	5.1

7. RESTRICCIONES

El método de reconstrucción tiene unas ciertas restricciones en cuanto al objeto a reconstruir, el escenario donde se realiza la captura de datos y la distancia entre el objeto y el sensor.

A. Restricciones del objeto.

El objeto a reconstruir debe estar dentro de unas dimensiones específicas, no debe superar los 1.5 metros en altura, ancho y profundidad. Además la superficie no debe ser translucida ni reflectora puesto que el método que se está utilizando se basa en la proyección de puntos infrarrojos y esto afecta la detección de estos puntos por el sensor. Éstas restricciones están dadas por la capacidad de detección del sensor.

B. Restricciones del escenario.

El escenario estará restringido básicamente por 2 puntos. Uno el tamaño del objeto a capturar de tal forma que dentro del escenario alcance éste y se tenga el espacio suficiente para desplazar el sistema de captura. Y dos que sea un ambiente sin entrada directa de luz solar.

C. Restricciones de distancia.

El fabricante del dispositivo indica que el sensor pude detectar puntos a una distancia de hasta 5 metros así que la restricción está dada básicamente por éste parámetro. Para éste trabajo se trató de no superar una distancia de 1.5 metros entre el sensor y el objeto para tener menos errores en las capturas.

8. RESULTADOS

En el desarrollo de este trabajo se alcanzó un sistema de reconstrucción 3D, de tal forma que permita capturar fácilmente los objetos físicos para observarlos en un computador. Sin embargo debido a las técnicas utilizadas las reconstrucciones de muchas capturas pueden presentar ciertos fallos debido a la acumulación de errores en la alineación de capturas.

A continuación se presentan algunas reconstrucciones realizadas con el sistema.

A. Persona

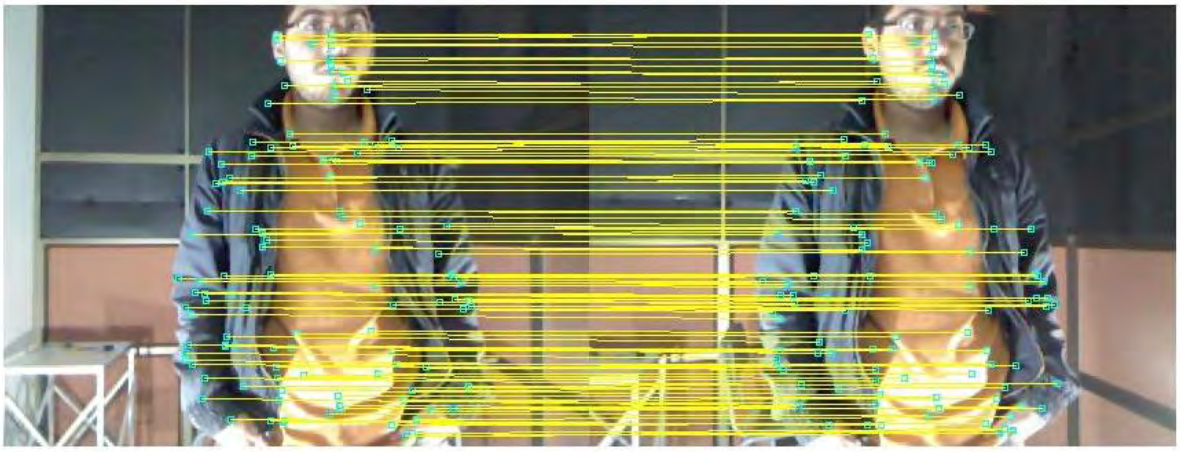


Figura 27 Grafica de un par de imágenes de la persona, y sus respectivas coincidencias en reconstrucción de 180°.

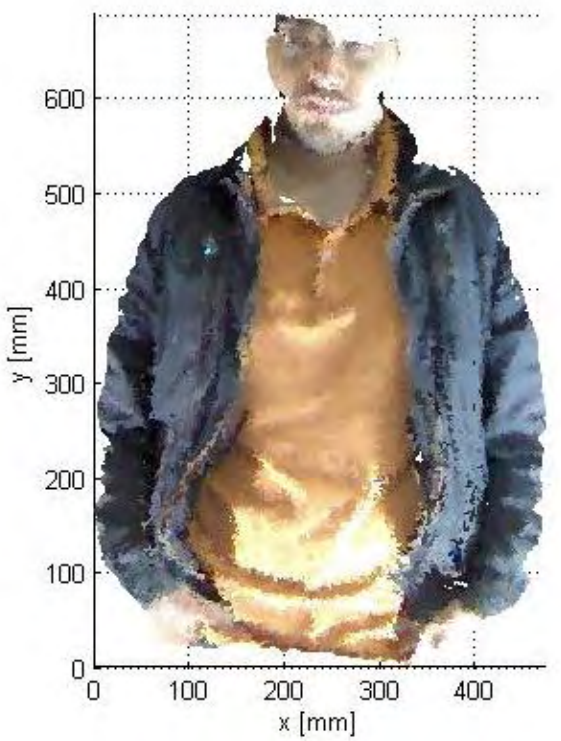


Figura 28 Reconstrucción sobre 180° de la persona, con filtro de triangulación 5. Vista frontal.

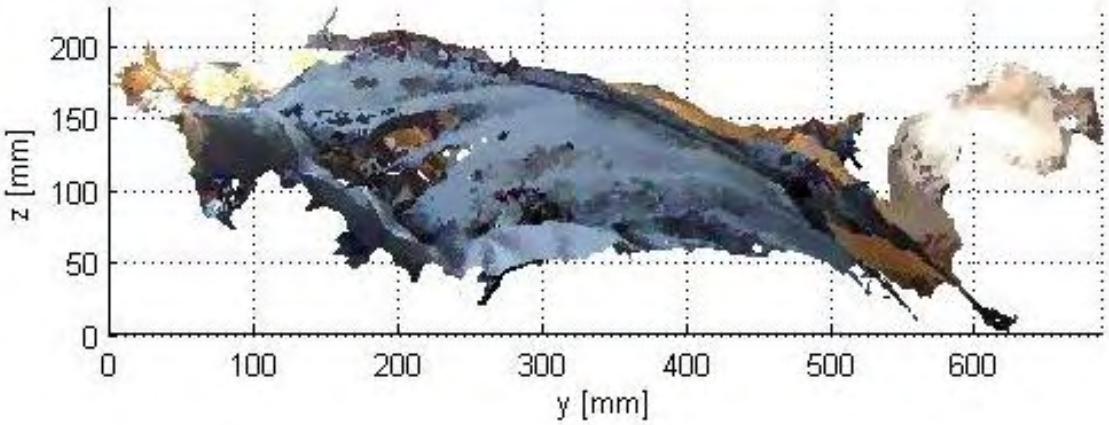


Figura 29 Reconstrucción sobre 180° de la persona, con filtro de triangulación 5. Vista lateral.

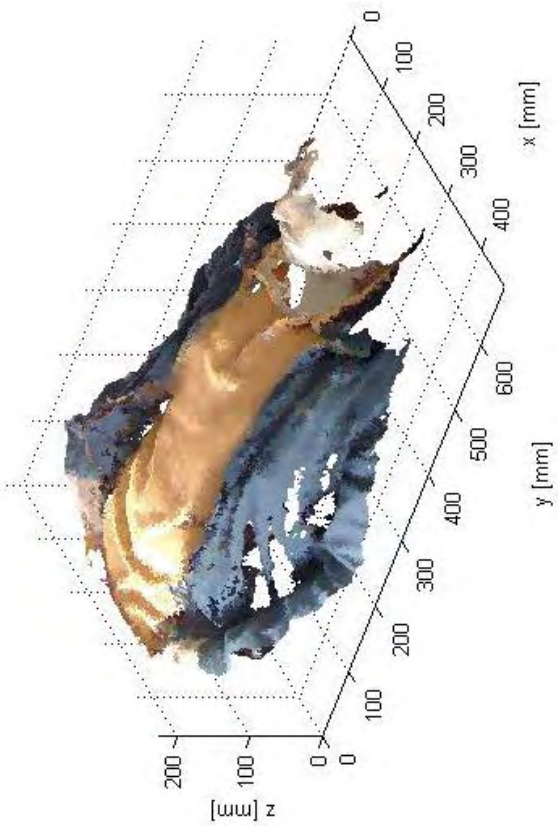


Figura 30 Reconstrucción sobre 180° de la persona, con filtro de triangulación 5. Vista superior derecha.

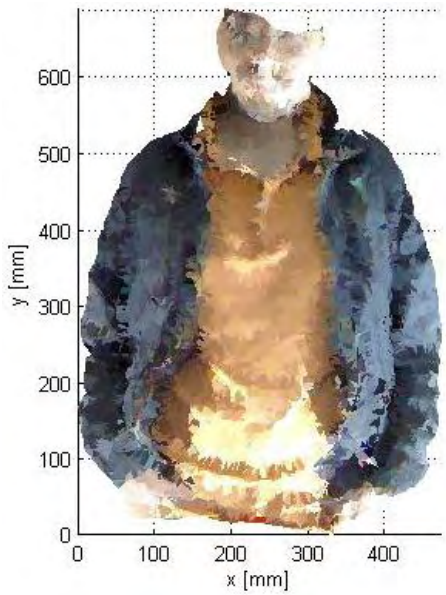


Figura 31 Reconstrucción sobre 180° de la persona, con filtro de triangulación 15. Vista frontal.

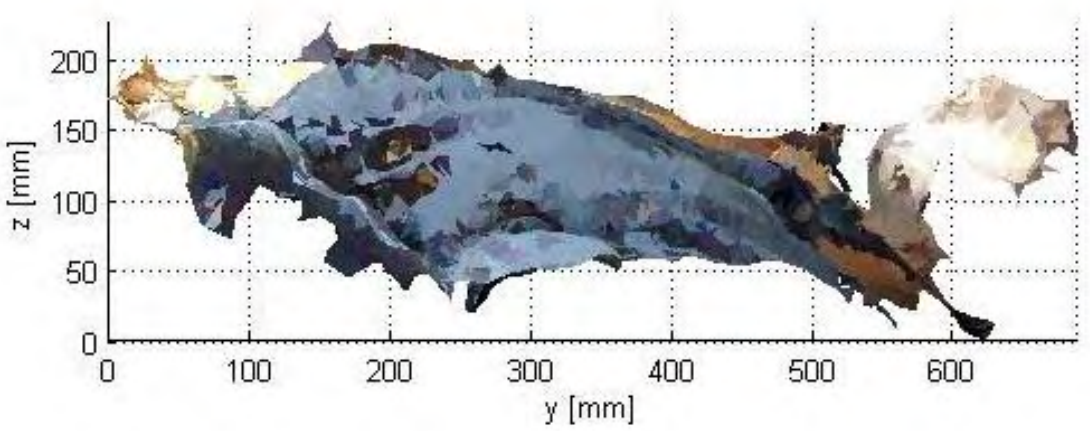


Figura 32 Reconstrucción sobre 180° de la persona, con filtro de triangulación 15. Vista lateral.

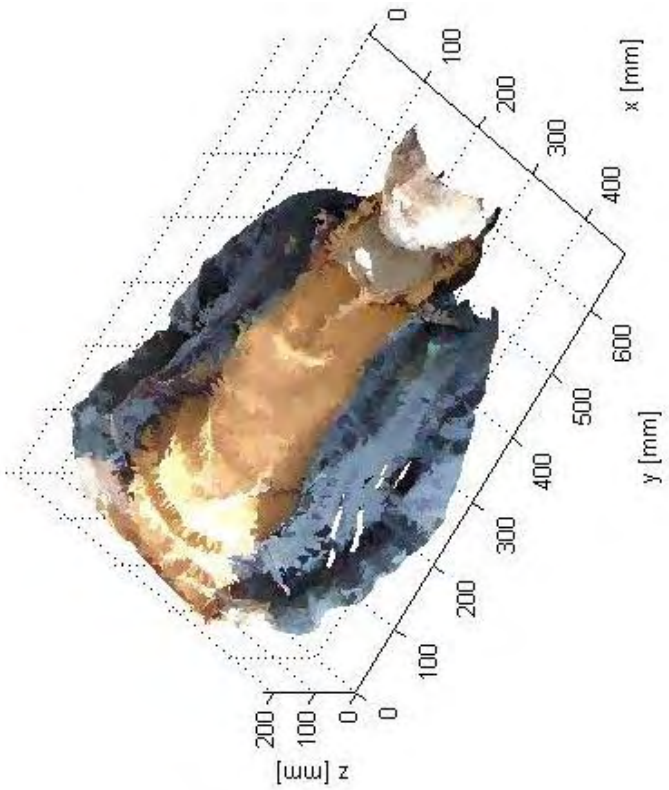


Figura 33 Reconstrucción sobre 180° de la persona, con filtro de triangulación 15. Vista superior derecha.

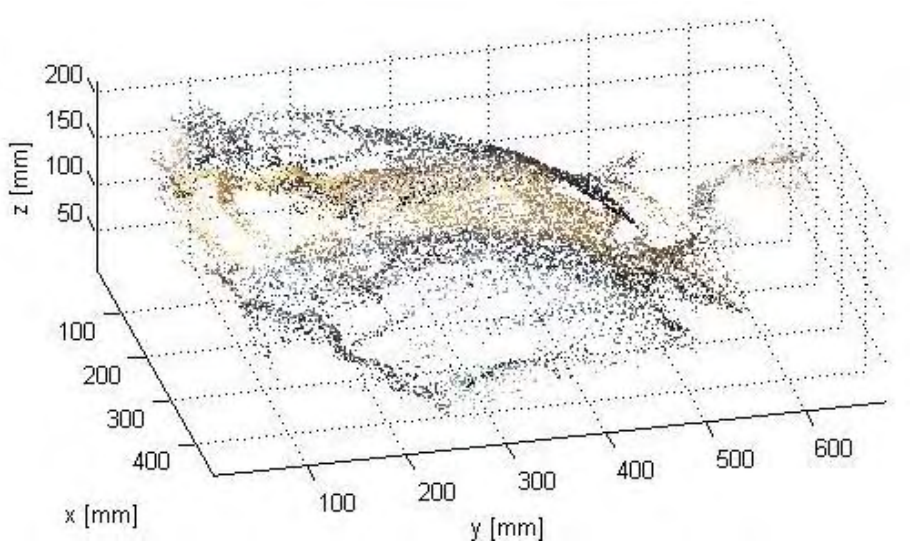


Figura 34 Reconstrucción sobre 180° de la persona, con filtro de triangulación 3.

B. Máscara de carnaval

Figura 35 Foto de artista usando la máscara, cortesía: Fredy Hidalgo.

Se realizó la reconstrucción de la máscara que se indica en la figura 35, la cual fue utilizada en los carnavales de Negros y Blancos del año 2014. Inicialmente se realizó la reconstrucción de 180° del objeto como se puede observar en las figuras 37 a 41. Posteriormente se realizó la reconstrucción tridimensional completa como se puede observar en las figuras 43 a 46.

Figura 36 Grafica de un par de imágenes de la máscara, y sus respectivas coincidencias para reconstrucción de 180°.

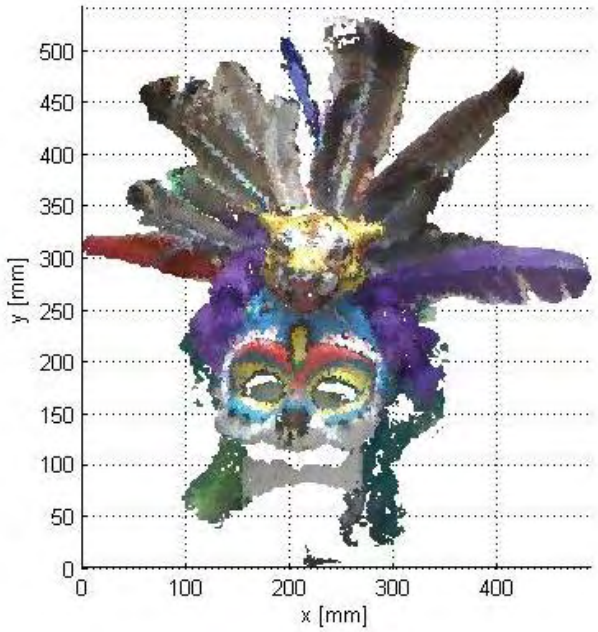


Figura 37 Reconstrucción sobre 180° de la máscara de carnaval, con filtro de triangulación 5. Vista frontal.

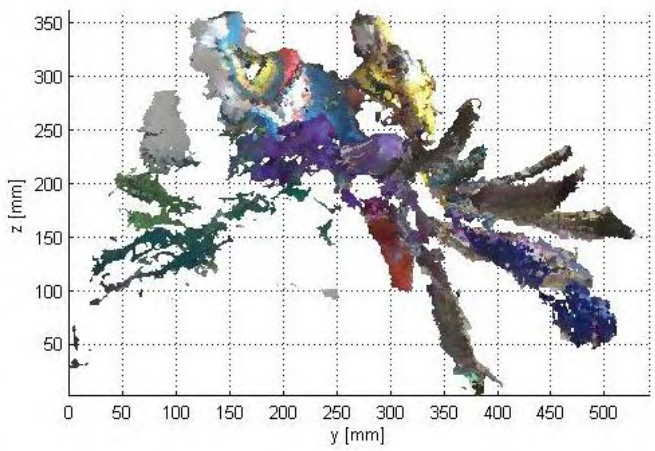


Figura 38 Reconstrucción sobre 180° de la máscara de carnaval, con filtro de triangulación 5. Vista lateral.

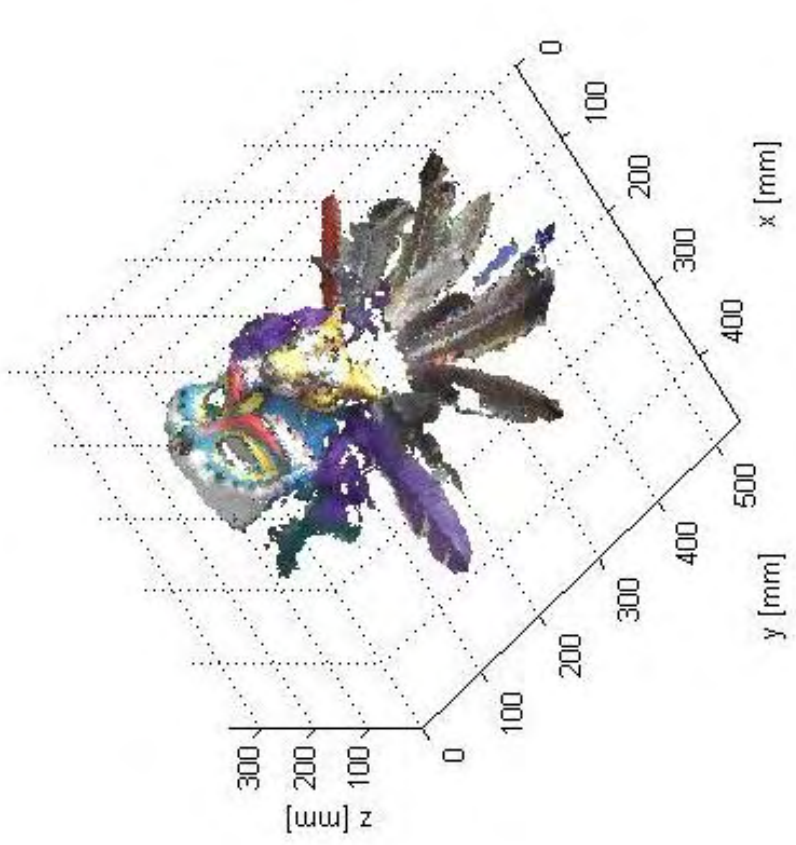


Figura 39 Reconstrucción sobre 180° de la máscara de carnaval, con filtro de triangulación 5. Vista superior derecha.

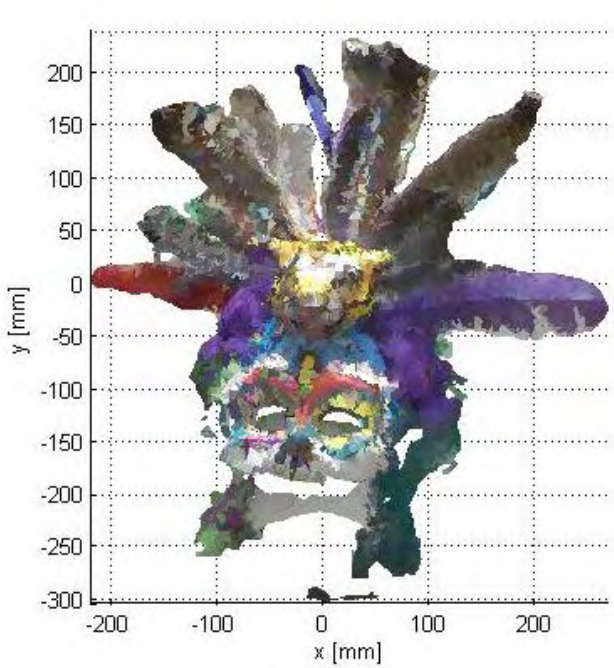


Figura 40 Reconstrucción sobre 180° de la máscara de carnaval, con filtro de triangulación 10. Vista frontal.

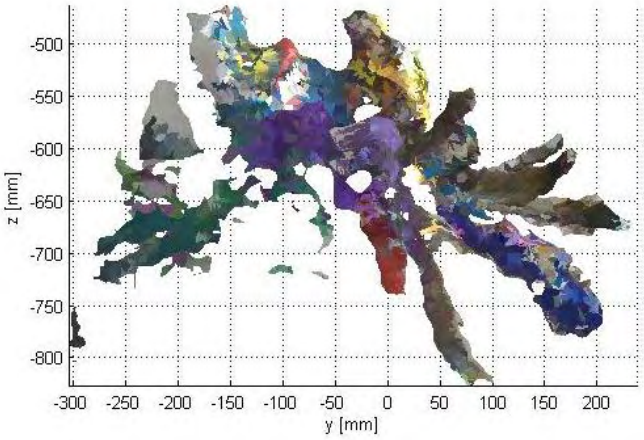


Figura 41 Reconstrucción sobre 180° de la máscara de carnaval, con filtro de triangulación 10. Vista lateral.

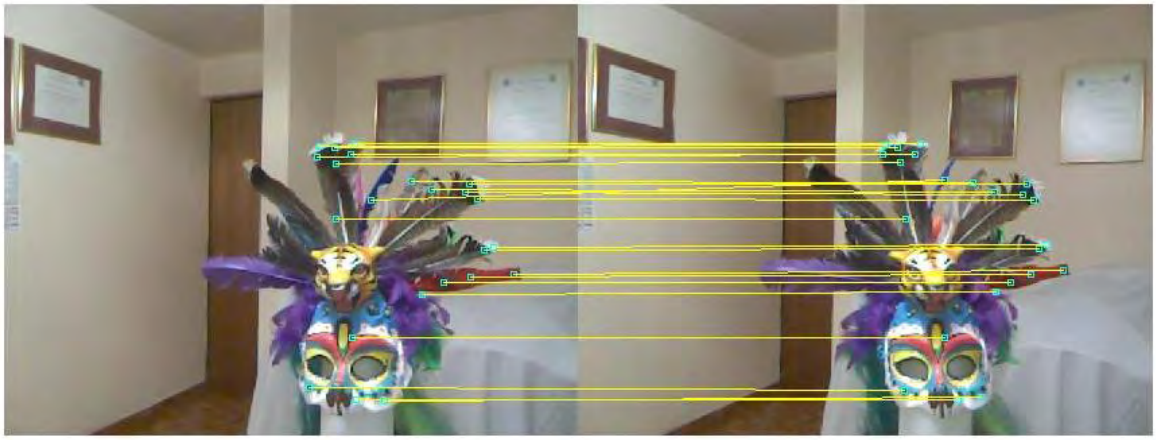


Figura 42 Grafica de un par de imágenes de la máscara, y sus respectivas coincidencias en reconstrucción de 360.

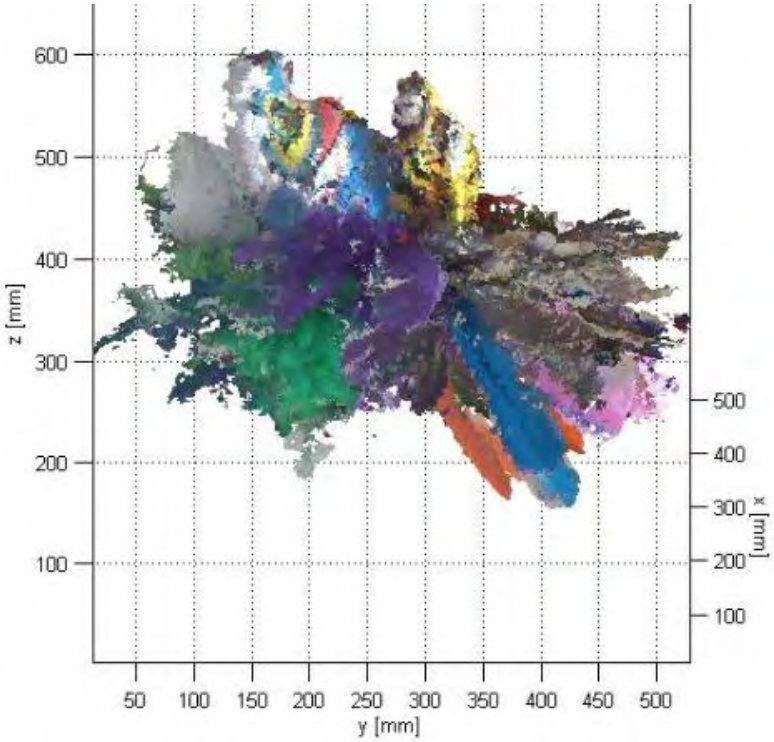


Figura 43 Reconstrucción sobre 360° de la máscara de carnaval, con filtro de triangulación 7. Vista lateral.

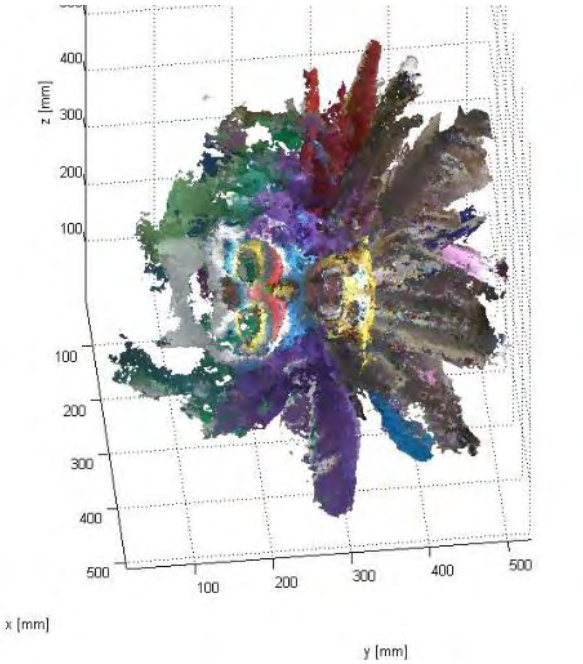


Figura 44 Reconstrucción sobre 360° de la máscara de carnaval, con filtro de triangulación 7. Vista frontal.

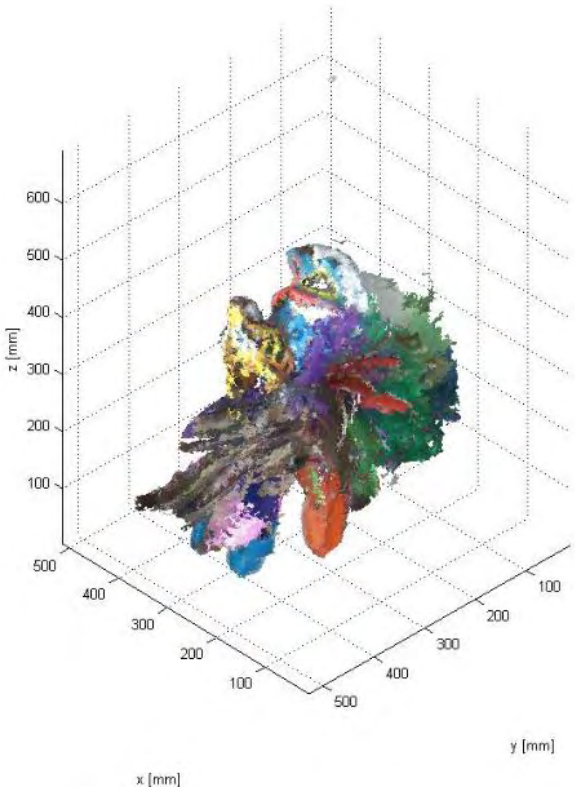


Figura 45 Reconstrucción sobre 360° de la máscara de carnaval, con filtro de triangulación 7. Vista superior izquierda.

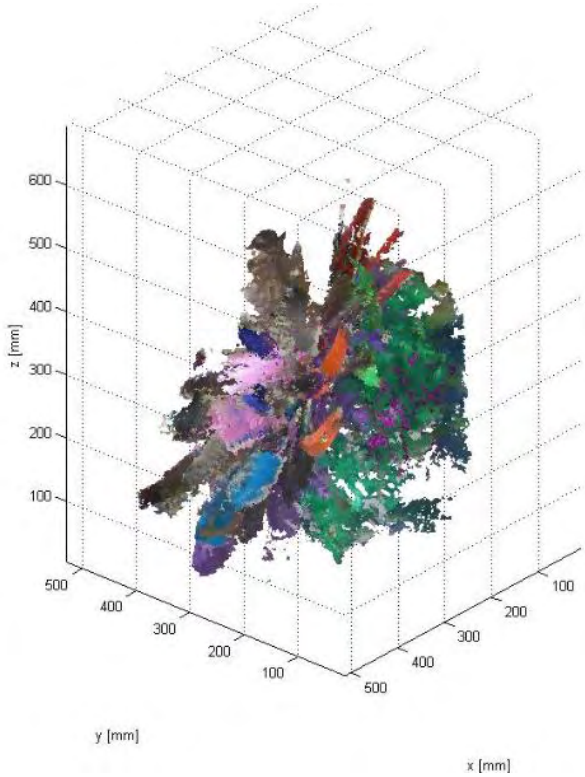


Figura 46 Reconstrucción sobre 360° de la máscara de carnaval, con filtro de triangulación 7. Vista superior.

C. Reconstrucción de Caja.

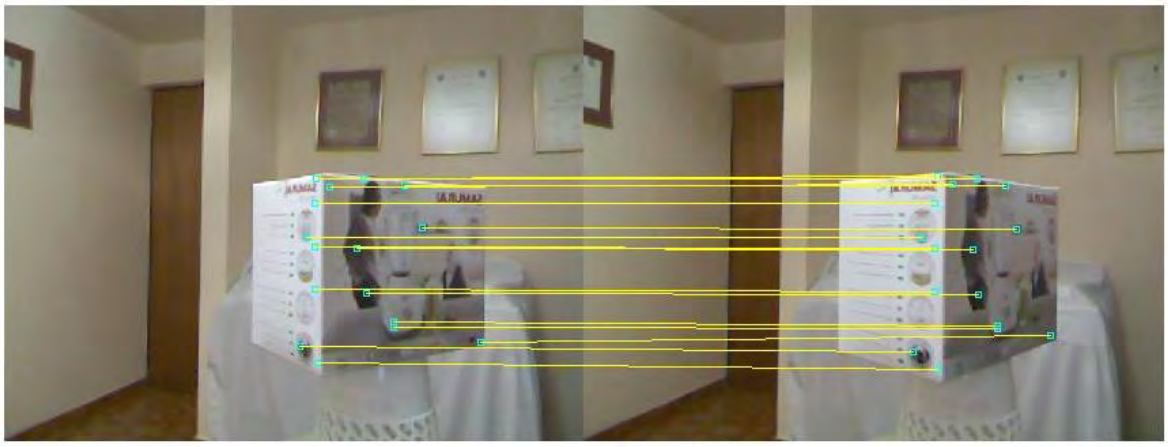


Figura 47 Gráfica de un par de imágenes de una caja y sus respectivas coincidencias.

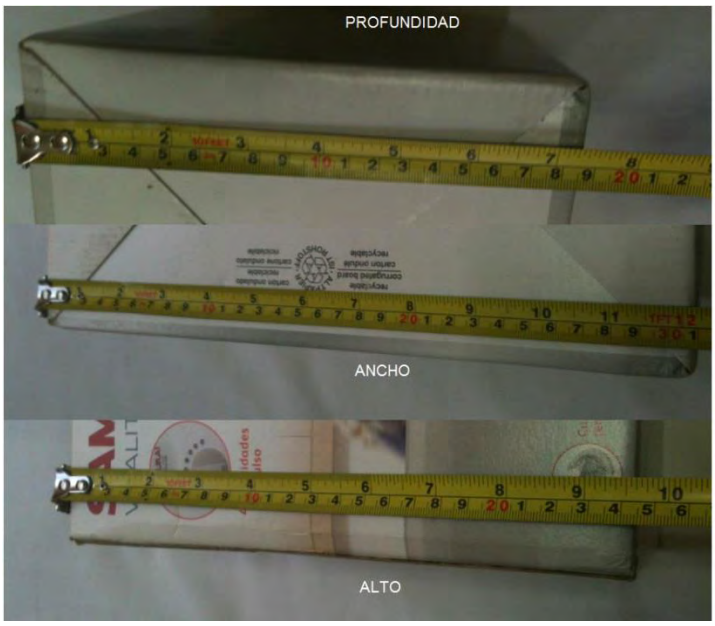


Figura 48 Medición dimensiones de la caja.

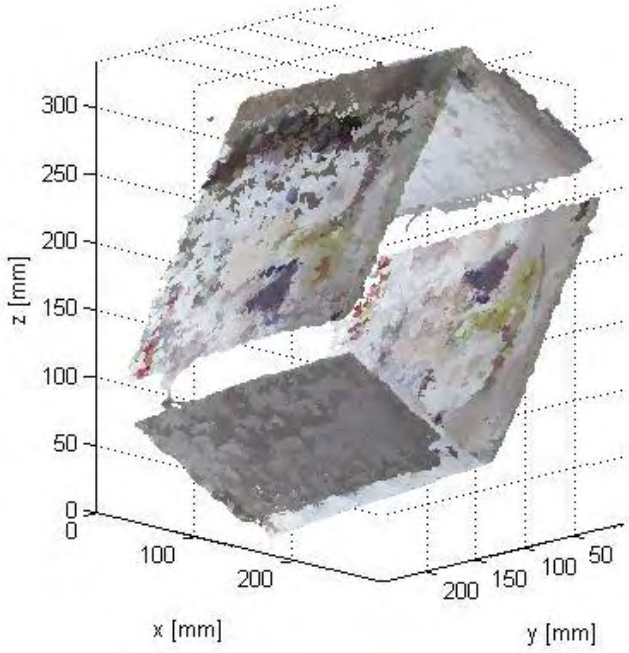


Figura 49 Reconstrucción de caja en 360°, con filtro de triangulación 7.

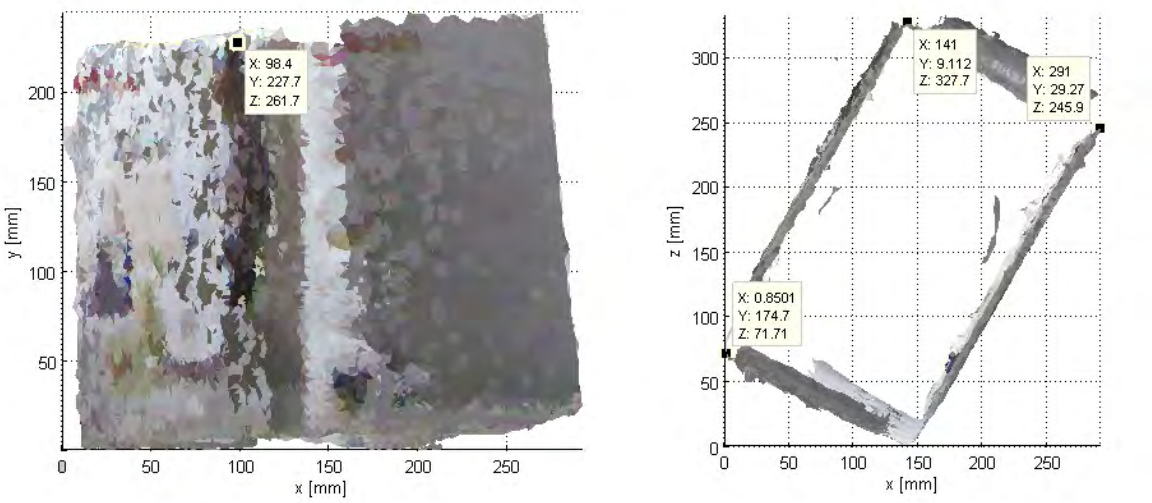


Figura 50 Mediciones de la caja reconstruida.

Para obtener las mediciones de la caja en la reconstrucción se tomaron datos aproximadamente en las esquinas del objeto y se calculó la distancia euclidiana sobre las coordenadas x, z de cada dato.

TABLA II
COMPARACIÓN DE DISTANCIAS: REAL VS RECONSTRUCCIÓN

	DISTANCIA MEDIDA	Distancia Reconstrucción	Error relativo (%)
Alto	24.5 cm	≈22.7 cm	7.3
Ancho	31cm	≈29.2 cm	5.8
Profundo	18.5 cm	≈17.1 cm	7.5

9. DISCUSIÓN

En el desarrollo de este trabajo se pudo aplicar diferentes algoritmos antes desarrollados para poder facilitar los cálculos y reducir el número de iteraciones del programa para obtener un modelo en 3D a partir de una nube de puntos.

Si no se hubiese contado con el algoritmo SURF se habría tenido total desconocimiento de cuáles eran los puntos que coincidían en una nube y otra, por lo que se hacía necesario aplicar ICP para lograr obtener las matriz R y el vector T. Sin embargo, debido a que se conocían los puntos no fue necesario hacer un proceso iterativo por lo que la aplicación de ICP no fue hecha al 100%. Se tomó como base el algoritmo pero al final solo se aplicar uno de los pasos de ICP que es el cálculo de R y T que minimizaran (1). Por lo que el poder conocer que puntos coincidían fue una gran ventaja a la hora de obtener el menor error cuadrático de (1).

Después de haber realizado varias pruebas se evidenció que dependiendo de las condiciones de luz los resultados podían variar mucho, ya que algunas de las pruebas realizadas en un laboratorio con una gran entrada de luz hizo que los datos de profundidad tuvieran muchos datos nulos que aparecían en las matrices como NAN. Esto hacia que la escasa cantidad de puntos no permitieran obtener un modelo adecuado de reconstrucción, debido a esto fue necesario adaptar una zona para evitar que las condiciones de luz afectaran de forma grave el desempeño del dispositivo.

La triangulación hecha de forma adecuada puede hacer que el modelo tenga muy buena “percepción” o que sea irreconocible. Para evitar que el modelo perdiera su forma se eliminaron triángulos que ocultaban la forma del objeto reconstruido.

10. CONCLUSIONES

El principal aporte de esta investigación es el uso del algoritmo SURF en el sistema de reconstrucción de objetos tridimensionales. Ya que permitió conocer que datos entre un par de nubes de puntos son correspondientes. Por lo que se evitó el uso de un algoritmo iterativo para encontrar R y T que minimiza el error cuadrático.

El dispositivo de captura no tiene la mejor resolución por lo que las reconstrucciones realizadas no superan en calidad a otros métodos como por ejemplo reconstrucciones con láser. Aun así los modelos en 3D obtenidos permiten identificar clara y fácilmente que objeto representa el modelo final, además la facilidad de trabajo con el sensor y flexibilidad que este tiene para ser llevado a distintos lugares, hace que esta sea una opción bastante viable a la hora de realizar reconstrucciones en diferentes lugares y no solo en un laboratorio.

No obstante es necesario tener en cuenta que se debe contar con condiciones de luz controladas, pues el sensor Kinect presenta fallas en cuanto a las capturas cuando el ambiente de trabajo presenta exposición directa a la luz solar. Así mismo se determinó que funciona óptimamente cuando se trabaja en un ambiente con luz uniforme.

11. TRABAJO FUTURO.

- La automatización del sistema podría evitar errores debidos a la intervención humana.
- Investigar otros algoritmos que detecten puntos coincidentes.
- Investigar otros algoritmos que permitan el acople de nubes de puntos.
- Investigar otras técnicas de triangulación o *meshing*.
- Utilizar nuevas y mejoradas versiones del sensor y controladores.
- Aplicar filtros de suavizado.
- Exportar los datos en distintos formatos que permitan su trabajo en otros tipos de software.

REFERENCIAS

[1] D. Risueño y J. Burbano, “Desarrollo de un sistema modelador de imágenes en 3D aplicando la técnica de Moiré”, tesis de grado Universidad de Nariño, Septiembre 2011.

[2] InnovMetric Software Inc., PolyWorks, “3D Metrology Hardware Review”, 2010, documento en línea disponible en: http://www.innovmetric.com/polyworks/3D-canners/pdf/3D_Metrology_Hardware_Review.pdf

[3] RENISHAW, “Touch-trigger probe systems”, 2003.

[4] English Heritage, “3D Laser Scanning for Heritage”, segunda edición, 2011.

[5] Greg Borestein,”Making Things See”, O’Reilly Media Inc., 2012.

[6] Leandro Cruz, Djalma Lucio, Luiz Velho,”Kinect and RGBD Images: Challenges and Applications”, IEEE computer society, 2012.

[7] M. Zwicker, C. Gostman, “Meshing Point Clouds Using Spherical Parameterization”, Massachusetts Institute of Technology, 2004.

[8] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, G. Taubin, “The Ball-Pivoting Algorithm for Surface Reconstruction”, IBM, 1999.

[9] William Lorensen, Harvey E. Cline, “MARCHING CUBES: A HIGH RESOLUTION 3D SURFACE CONSTRUCTION ALGORITHM”, Computer Graphics, volume 21, 1987.

[10] Michael Kazhdan, Matthew Bolitho, Hugues Hoppe, “Poisson Surface Reconstruction”, Eurographics Symposium on Geometry Processing, 2006.

[11] Junyi Liu; Xiaojin Gong; Jilin Liu, "Guided inpainting and filtering for Kinect depth maps," Pattern Recognition (ICPR), 2012 21st International Conference on , vol., no., pp.2055,2058, 11-15 Nov. 2012.

[12] J. Andreas Bærentzen, “ON THE IMPLEMENTATION OF FAST MARCHING METHODS FOR 3D LATTICES”, department of mathematical modelling, Technical University of Denmark, 2001.

[13] Jingjing Fu, Dan Miao, Weiren Yu, Shiqi Wang, Yan Lu, Shipeng Li, “KINECT-LIKE DEPTH COMPRESSION WITH 2D+T PREDICTION”, IEEE International Conference on Multimedia and Expo Workshops, 2012.

[14] Qi Sun; Yanlong Tang; Ping Hu; Jingliang Peng, "Kinect-based automatic 3D high-resolution face modeling," Image Analysis and Signal Processing (IASP), 2012 International Conference on , vol., no., pp.1,4, 9-11 Nov. 2012.

[15] Hirokazu Yagou, Yutaka Ohtake, Alexander G. Belyaev, “Mesh Smoothing via Mean and Median Filtering Applied to Face Normals”, Geometric Modeling and Processing, julio 2002.

[16] Paul J. Besl, Neil D. Mckay, “A Method for Registration of 3-d Shapes”, IEEE transactions on pattern analysis and machine intelligence, vol 14, 1992.

[17] Lena Maier-Hein, Alfred M. Franz, Thiago R. dos Santos, Mirko Schmidt,Markus Fangerau, Hans-Peter Meinzer, and J. Michael Fitzpatrick, "Convergent Iterative Closest-Point Algorithm to Accomodate Anisotropic and Inhomogenous Localization Error", terminar referencia.

[18] Herbert Bay, Tinne Tuytelaars, Luc Van Gool, “Speeded Up Robust Features”, Katholieke Universiteit Leuven, ETH Zurich, 2008.

[19] K. S. Arun, t. S. Huang, and s. D. Blostein,"least-squares fitting of two 3-d point sets", IEEE transactions on pattern analysis and machine intelligence, vol.5, PP-9, septiembre 1987.
[20] Kirk Baker, "Singular Value Decomposition Tutorial", marzo 29, 2005.
[21] Descomposición en valores singulares, documento en línea disponible en: http://www.mate.unlp.edu.ar/practicas/70_18_0911201012951.pdf
[22] Jan Smisek, Michal Jancosek and Tomas Pajdla, "3D with Kinect", CMP, Dept. of Cybernetics, FEE, Czech Technical University in Prague.
[23] Meng Yao-wei y Wang Shan-dong, "A New Interactive Approach of 3D Modeling Reconstruction from Contours", IEEE Computer Application and System Modeling (ICCASM), volumen 6, octubre de 2010.
[24] Marc Levoy, "The Digital Michelangelo Project", Second International Conference on 3D Digital Imaging and Modeling, octubre 1999.
[25] M. Judith Leo, D. Manimegalai, "3D Modeling of Human Faces-A Survey", Trendz in Information Sciences and Computing (TISC), diciembre 2011.
[26] Shiping Zhu, Jie Gao, "3D Modeling and Rendering Based on Uncalibrated Single View Image in Undergraduate Final Design", Computer Science and Information Processing (CSIP), agosto 2012.
[27] Draelos, M.; Deshpande, N.; Grant, E., "The Kinect up close: Adaptations for short-range imaging", Multisensor Fusion and Integration for Intelligent Systems (MFI), 2012 IEEE Conference on , vol., no., pp.251,256, 13-15 Sept. 2012.
[28] Jingjing Fu; Dan Miao; Weiren Yu; Shiqi Wang; Yan Lu; Shipeng Li, "Kinect-Like Depth Compression with 2D+T Prediction", Multimedia and Expo Workshops (ICMEW), 2012 IEEE International Conference on , vol., no., pp.599,604, 9-13 July 2012.
[29] Koenderink, J., "The structure of images". Biological Cybernetics50 pp 363 –370, 1984.
[30] D. Lowe. "Distinctive image features from scale-invariant keypoints", Computer Science Department, University of British Columbia. 2004.
[31] P.C. Mahalanobis, *On the generalised distance in statistics*, Proceedings of the National Institute of Science of India 12 (1936) 49-55.

ANEXO A: Código Arduino.

```
/* CONFIGURACIÓN DE PINES (baquelita no Arduino)
1  5v
2  Gnd
3  RS - LCD
4  Enable - LCD
5  D4 - LCD
6  D5 - LCD
7  D6 - LCD
8  D7 - LCD
9  5v - CMPS10
10 SDA - CMPS10
11 SCL - CMPS10
12,13  NO CONECTA
14  Gnd - CMPS10
15 SDA - ARDUINO
16 SCL - ARDUINO
17 SEÑAL DE PULSADOR
18,19 CONECCION PULSADOR
*/
#include <Wire.h>
#include <LiquidCrystal.h>
LiquidCrystal lcd(22, 23, 24, 25, 26, 27);//rs,e,d4,d5,d6,d7
#define ADDRESS 0x60
#define PULS 30
//variables
char jaja;
int toma=0;
int boton=LOW;
int botonp=LOW;
int flag=1;
unsigned long tiempop;
unsigned long tiempo;
int beg=0;


void setup(){
  Wire.begin();
  Serial.begin(9600);
  lcd.begin(16, 2);
  pinMode(PULS,INPUT);
}

void loop(){
  while (flag){
    lcd.setCursor(0,0);
    lcd.print("Inicializando...");
     if (Serial.available()>0){
     S11=Serial.read();
     }
     if (S11=='6'){
     flag=0;
     }
```

```
   }
  boton=digitalRead(PULS);
   if(boton!=botonp){
  botonp=boton;
   if (boton==HIGH){
  tiempop=millis();
  beg=0;
   }
   if (boton==LOW){
   tiempo=millis()-tiempop;
   beg=1;
   }
   if (beg==1){
   if (tiempo<=1500){
  Serial.println('a');
   }
   else{
   Serial.println('b');
   }
   }
  }

 if (Serial.available()>0){
S1=Serial.read();
}

  byte highByte, lowByte, fine;
  char pitch, roll;
  int bearing;

  Wire.beginTransmission(ADDRESS);
  Wire.write(2);
  Wire.endTransmission();

  Wire.requestFrom(ADDRESS, 4);
  while(Wire.available() < 4);
  highByte = Wire.read();
  lowByte = Wire.read();
  pitch = Wire.read();
  roll = Wire.read();

  bearing = ((highByte<<8)+lowByte)/10;
  fine = ((highByte<<8)+lowByte)%10;

  lcd.setCursor(0, 0);
  lcd.print("Angulo  Capturas");



  if (100<=bearing){
  lcd.setCursor(3, 1);
  }else if(10<=bearing<100){
    lcd.setCursor(3, 1);
    lcd.print(' ');
```

```
    lcd.setCursor(4, 1);
  }else{
    lcd.setCursor(3, 1);
    lcd.print("  ");
    lcd.setCursor(5, 1);
  }

  lcd.print(bearing);
  lcd.setCursor(11,1);
  lcd.print(toma);
  delay(100);

  if (S1=='3'){
     Serial.println(bearing);
  }
  if(S1=='4'){
    toma++;
  }

  if (S1=='5'){
   lcd.clear();
   lcd.setCursor(4,0);
   lcd.print("ERROR 01");
   lcd.setCursor(0,1);
   lcd.print("No Suficientes Coincidencias");
   lcd.scrollDisplayLeft();
   delay(500);
   lcd.clear();

  }
  if (S1=='1'){
    while(1){
    lcd.clear();
    lcd.setCursor(4,0);
    lcd.print("PROCESO");
    lcd.setCursor(3,1);
    lcd.print("COMPLETADO");
    delay(200);
    }
  }
}
```

ANEXO B: Código Matlab[R].

```
%% INTERFAZ GRÁFICA %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function varargout = programa(varargin)
% PROGRAMA MATLAB code for programa.fig
%      PROGRAMA, by itself, creates a new PROGRAMA or raises the existing
%      singleton*.
%
%      H = PROGRAMA returns the handle to a new PROGRAMA or the handle to
%      the existing singleton*.
%
%      PROGRAMA('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PROGRAMA.M with the given input
arguments.
%
%      PROGRAMA('Property','Value',...) creates a new PROGRAMA or raises
the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before programa_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to programa_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help programa

% Last Modified by GUIDE v2.5 11-Feb-2014 17:28:27

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @programa_OpeningFcn, ...
                   'gui_OutputFcn',  @programa_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before programa is made visible.
function programa_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to programa (see VARARGIN)

% Choose default command line output for programa
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes programa wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = programa_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in comenzar.
function comenzar_Callback(hObject, eventdata, handles)
set(hObject,'Enable','off');
set(hObject,'BackgroundColor',[1,0,0]);
set(hObject,'String','procesando');
set(handles.axes1,'Visible','on');
set(handles.text13,'Visible','on');
set(handles.edit1,'Enable','off');
set(handles.edit2,'Enable','off');
set(handles.edit3,'Enable','off');
set(handles.edit4,'Enable','off');
set(handles.edit5,'Enable','off');
set(handles.edit6,'Enable','off');
set(handles.edit7,'Enable','off');

%variables
ser=str2num(get(handles.edit1,'String'));
valorptsc=str2num(get(handles.edit2,'String'));
tol=str2num(get(handles.edit3,'String'));
prof=str2num(get(handles.edit4,'String'));
rr=str2num(get(handles.edit5,'String'));
valfil=str2num(get(handles.edit6,'String'));
```

```
r=str2num(get(handles.edit7,'String'));

% proceso
axes(handles.axes1);
[ MM,CC,Irgb,real, ptsc, depth,k,tt] =
proceso(ser,valorptsc,tol,prof,rr,valfil  );
assignin('base','MM',MM);
assignin('base','CC',CC);
assignin('base','tt',tt);
[ tria,C,M] = tri( MM,CC,r,tt );

h = datestr(clock,0);
save(['Reconstrucciones/' [h(1:11),'-',h(13:14),'-',h(16:17),'-
',h(19:20)] '.mat']);
set(handles.uipanel1,'Visible','off');

set(handles.text3,'Visible','off');
set(handles.text4,'Visible','off');
set(handles.text5,'Visible','off');
set(handles.text6,'Visible','off');
set(handles.text7,'Visible','off');
set(handles.text8,'Visible','off');
set(handles.text9,'Visible','off');
set(handles.text10,'Visible','off');

set(handles.edit1,'Visible','off');
set(handles.edit2,'Visible','off');
set(handles.edit3,'Visible','off');
set(handles.edit4,'Visible','off');
set(handles.edit5,'Visible','off');
set(handles.edit6,'Visible','off');
set(handles.edit7,'Visible','off');

assignin('base','k',k);
assignin('base','M',M);
assignin('base','C',C);
assignin('base','tria',tria);
assignin('base','Irgb',Irgb);
assignin('base','real',real);
assignin('base','ptsc',ptsc);
assignin('base','depth',depth);
run graficass.m;
close programa;
%
% hObject    handle to comenzar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
comenzar.
function comenzar_ButtonDownFcn(hObject, eventdata, handles)
```

```
% hObject    handle to comenzar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a
double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a
double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a
double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a
double


% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5 as a
double


% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6 as a
double


% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit7 as text
%        str2double(get(hObject,'String')) returns contents of edit7 as a
double


% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function tex1_CreateFcn(hObject, eventdata, handles)

% hObject    handle to tex1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

function varargout = graficass(varargin)
% GRAFICASS MATLAB code for graficass.fig
%      GRAFICASS, by itself, creates a new GRAFICASS or raises the
existing
%      singleton*.
%
%      H = GRAFICASS returns the handle to a new GRAFICASS or the handle
to
%      the existing singleton*.
%
%      GRAFICASS('CALLBACK',hObject,eventData,handles,...) calls the
local
```

```
%      function named CALLBACK in GRAFICASS.M with the given input
arguments.
%
%      GRAFICASS('Property','Value',...) creates a new GRAFICASS or
raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before graficass_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to graficass_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help graficass

% Last Modified by GUIDE v2.5 12-Feb-2014 12:03:59

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @graficass_OpeningFcn, ...
                   'gui_OutputFcn',  @graficass_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before graficass is made visible.
function graficass_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to graficass (see VARARGIN)

% Choose default command line output for graficass
handles.output = hObject;
```

```
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes graficass wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = graficass_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
set(handles.text7,'String','Gráfica de Pares en RGB');
set(handles.pushbutton2,'Enable','on');
set(handles.pushbutton3,'Enable','off');
set(handles.edit1,'Enable','off');
set(handles.edit2,'Enable','off');
set(handles.edit1,'String','0');
set(handles.edit2,'String','90');

set(handles.axes2,'Visible','off');
axes(handles.axes2);
cla;
axes(handles.axes1);
cla;
k=evalin('base','k');
i=1;
set(handles.slider1,'Enable','on');
set(handles.slider1,'Min',0);
set(handles.slider1,'Max',k-2);
set(handles.slider1,'Sliderstep',[1/(k-2),1/(k-2)]);
set(handles.slider1,'Visible','on');

set(handles.text3,'Enable','on');
Irgb=evalin('base','Irgb');
ptsc=evalin('base','ptsc');
axes(handles.axes1);
graficrgb(Irgb.(['a' num2str(i)]),Irgb.(['a' num2str(i+1)]),...
    ptsc.(['a' num2str((2*i)-1)]),ptsc.(['a' num2str(2*i)]))
set(handles.text3,'String',strcat(num2str(i),'-',num2str(i+1)))
set(handles.axes1,'Visible','on');
```

```
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of pushbutton1


% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)

i= int32(get(handles.slider1,'Value'))+1;

Irgb=evalin('base','Irgb');
ptsc=evalin('base','ptsc');
axes(handles.axes1);
graficrgb(Irgb.(['a' num2str(i)]),Irgb.(['a' num2str(i+1)]),...
    ptsc.(['a' num2str((2*i)-1)]),ptsc.(['a' num2str(2*i)]))
set(handles.text3,'String',strcat(num2str(i),'-',num2str(i+1)))


% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
slider


% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
set(handles.text7,'String','Reconstrucción 3D');
set(handles.axes1,'Visible','off');
axes(handles.axes1);
cla;
axes(handles.axes2);
cla;
set(handles.pushbutton3,'Enable','on');
```

```
set(handles.edit1,'Enable','on');
set(handles.edit2,'Enable','on');
set(handles.pushbutton2,'Enable','off');
set(handles.slider1,'Enable','off');
set(handles.axes2,'Visible','on');
set(handles.text3,'Enable','off');

M1=evalin('base','M');
C1=evalin('base','C');
tria1=evalin('base','tria');

colormap(C1)
trisurf(tria1,M1(:,1),M1(:,2),M1(:,3),1:size(C1,1),'edgecolor','none');
axis equal
view(0,90)
drawnow
set(get(handles.axes2,'XLabel'),'String','X [mm]')
set(get(handles.axes2,'YLabel'),'String','Y [mm]')
set(get(handles.axes2,'ZLabel'),'String','Z [mm]')
set(handles.axes2,'Visible','on');

 clear M1 C1 tria1

% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of pushbutton2



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a
double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a
double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
H=str2num(get(handles.edit1,'String'));
V=str2num(get(handles.edit2,'String'));
axes(handles.axes2);
view(H,V)



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a
double
```

```
% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
set(handles.pushbutton1,'Enable','off');
set(handles.pushbutton2,'Enable','off');
set(handles.pushbutton4,'Enable','off');
set(handles.pushbutton5,'Enable','on');
set(handles.edit4,'Enable','on');


% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
MM=evalin('base','MM');
CC=evalin('base','CC');
tt=evalin('base','tt');
r=str2num(get(handles.edit4,'String'));
clear M C
[ tria,C,M] = tri( MM,CC,r,tt );
assignin('base','M',M);
assignin('base','C',C);
assignin('base','tria',tria);
set(handles.pushbutton1,'Enable','on');
set(handles.pushbutton2,'Enable','on');
set(handles.pushbutton4,'Enable','on');
set(handles.pushbutton5,'Enable','off');
set(handles.edit4,'Enable','off');



% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)

% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


%%%%FUNCIONES%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ ptsc1 ] = agrupar( ptsc )
l=1;
for k=1:length(ptsc(:,1))
    ptsc1(l,:)=ptsc(k,:);
    ptsc1(l+1,:)=[ptsc(k,1)+1 ptsc(k,2)];
    ptsc1(l+2,:)=[ptsc(k,1)-1 ptsc(k,2)];
    ptsc1(l+3,:)=[ptsc(k,1) ptsc(k,2)+1];
    ptsc1(l+4,:)=[ptsc(k,1) ptsc(k,2)-1];
    l=l+5;
end
end

function [ real1 ] = applyicp( real,R)%,T)
real1=reshape(real,[307200,3])';
real1=R*real1;
real1=reshape(real1',[480,640,3]);
end

function [ captura] = applyt( captura,T )
la=isnan(captura(3,:)) | captura(3,:)==0;
captura(:,la)=nan;
TT(1,1:length(captura(1,:)))=T(1);
TT(2,1:length(captura(1,:)))=T(2);
TT(3,1:length(captura(1,:)))=T(3);
captura=captura-TT;
end


function [ bien ] = busquedaerror( ptsc,va )
if length(ptsc(:,1)) <va
   bien=0;
else
   bien=1;
end
end

function [ ptsc1,ptsc2 ] = busquedakeypoints(
real1,real2,I1rgb,I2rgb,tipo,val,tol )
pts1=[];
pts2=[];
```

```
I1 = rgb2gray(I1rgb);
I2 = rgb2gray(I2rgb);
switch tipo
    case 1
    points1 = detectSURFFeatures(I1,'MetricThreshold',val);
    points2 = detectSURFFeatures(I2,'MetricThreshold',val);
    case 2
    points1 = detectSURFFeatures(I1,'NumOctaves',val);
    points2 = detectSURFFeatures(I2,'NumOctaves',val);
    case 3
    points1 = detectSURFFeatures(I1,'NumScaleLevels',val);
    points2 = detectSURFFeatures(I2,'NumScaleLevels',val);

end
[f1, vpts1] = extractFeatures(I1, points1);
[f2, vpts2] = extractFeatures(I2, points2);
index_pairs1 = matchFeatures(f1, f2);
matched_pts1 = vpts1(index_pairs1(:, 1));
matched_pts2 = vpts2(index_pairs1(:, 2));
pos1=round(matched_pts1.Location);
pos2=round(matched_pts2.Location);
err=abs(pos1(:,2)-pos2(:,2));
a=1;
for i=1:length(err)
if err(i)<tol
pts1(a,:)=pos1(i,:);
pts2(a,:)=pos2(i,:);
a=a+1;
end
end
b=1;
if length(pts1)<3 || length(pts2)<3 || isempty(pts1) || isempty(pts2)
error1=0;
else
error1=1;
end
if error1
for i=1:length(pts1)
if (isnan(real1(pts1(i,2),pts1(i,1),3)) ||
isnan(real2(pts2(i,2),pts2(i,1),3)) ||real1(pts1(i,2),pts1(i,1),3)==0 ||
real2(pts2(i,2),pts2(i,1),3)==0)
else
ptsc1(b,:)=pts1(i,:);
ptsc2(b,:)=pts2(i,:);
b=b+1;
end
end
else
ptsc1=0;
ptsc2=0;
end
end
```

```
function [ ptsc1,ptsc2 ] = f_ptsc( ptsc1,ptsc2 )
flag=1;
k=1;
while flag
    l1=ptsc1(:,1)==ptsc1(k,1);
    l2=ptsc1(:,2)==ptsc1(k,2);
    la=l1&l2;
    if sum(la)==1
        la(k)=0;
    end
    ptsc1(la,:)=[];
    ptsc2(la,:)=[];
    if k==length(ptsc1(:,1))
        flag=0;
    else
        k=k+1;
    end
end
flag=1;
k=1;
while flag
    l1=ptsc2(:,1)==ptsc2(k,1);
    l2=ptsc2(:,2)==ptsc2(k,2);
    la=l1&l2;
    if sum(la)==1
        la(k)=0;
    end
    ptsc1(la,:)=[];
    ptsc2(la,:)=[];
    if k>=length(ptsc1(:,1))
        flag=0;
    else
        k=k+1;
    end
end
end
```

```
function [ tt ] = filt_tri( t,M,r );
tt=t;
p1=M(t(:,1),:);
p2=M(t(:,2),:);
p3=M(t(:,3),:);
p4=M(t(:,4),:);

d1=(p1-p2).^2;
d2=(p2-p3).^2;
d3=(p1-p3).^2;
d4=(p1-p4).^2;
d5=(p4-p3).^2;
d6=(p4-p2).^2;
L1=sqrt(d1(:,1)+d1(:,2)+d1(:,3));
L2=sqrt(d2(:,1)+d2(:,2)+d2(:,3));
L3=sqrt(d3(:,1)+d3(:,2)+d3(:,3));
```

```
L4=sqrt(d4(:,1)+d4(:,2)+d4(:,3));
L5=sqrt(d5(:,1)+d5(:,2)+d5(:,3));
L6=sqrt(d6(:,1)+d6(:,2)+d6(:,3));
la1=L1>r;
la2=L2>r;
la3=L3>r;
la4=L4>r;
la5=L5>r;
la6=L6>r;
la=la1 | la2 | la3 | la4 | la5 | la6;
tt(la,:)=[];
end


function [ MC,I ] = filtcontorno2( MC,I,r
tic
if length(MC(:,1))==3
    MC=MC';
end
MC=reshape(MC,[480,640,3]);
for rr=1:r
    a=1;
    for j=1:size(MC,1)
        [v i]=max(MC(j,:,1));
        x(a)=v;
        ps(a)=i;
        p(a,:)=MC(j,i,:);
        MC(j,i,1)=nan; MC(j,i,2)=nan; MC(j,i,3)=nan;
        I(j,i,1)=nan; I(j,i,2)=nan; I(j,i,3)=nan;
        [v i]=min(MC(j,:,1));
        x(a+1)=v;
        ps(a+1)=i;
        p(a+1,:)=MC(j,i,:);
        MC(j,i,1)=nan; MC(j,i,2)=nan; MC(j,i,3)=nan;
        I(j,i,1)=nan; I(j,i,2)=nan; I(j,i,3)=nan;
        a=a+2;
    end
    la=p(:,3)==0;
    p(la,:)=[];
    x(la)=[];
    ps(la)=[];
end
MC=reshape(MC,307200,3);
I=reshape(I,307200,3);
la=isnan(MC(:,3)) | MC(:,3)==0;
MC(la,:)=[];
I(la,:)=[];
fprintf('time: %4.4f s\n',toc);
end
```

```
function [ M,C] = filtrorepetidos( M,C,valfil)
MMM=M;
MMM(:,3)=round(MMM(:,3)/valfil);
MMM(:,2)=round(MMM(:,2)/valfil);
MMM(:,1)=round(MMM(:,1)/valfil);
[~, I, ~] = unique(MMM,'first','rows');
I=sort(I);
M=M(I,:);
C=C(I,:);
End

function [ ] = graficrgb( I1rgb,I2rgb,ptsc1,ptsc2 )
imshow([I1rgb,I2rgb]);
hold on;
for i = 1 : 1 : size(ptsc1,1)
    plot(ptsc1(i,1),ptsc1(i,2),'sc','MarkerSize',4);
    plot(ptsc2(i,1) + 640,ptsc2(i,2),'sc','MarkerSize',4);
    plot([ptsc1(i,1), ptsc2(i,1) + 640], ...
        [ptsc1(i,2), ptsc2(i,2)],'y','LineWidth',1);
end
hold off;
end


function [ cap,R,T] = ICP_SVD( real1,real2,ptsc1,ptsc2 )%cap,
for i=1:length(ptsc1)
icp1(:,i)=double(real1(ptsc1(i,2),ptsc1(i,1),:));
icp2(:,i)=double(real2(ptsc2(i,2),ptsc2(i,1),:));
end
la=isnan(icp1(3,:))|icp1(3,:)==0|isnan(icp2(3,:))|icp2(3,:)==0;
icp1(:,la)=[]; % Xi
icp2(:,la)=[]; % Pi
Nx=length(icp1(1,:));
ux=(sum(icp1')/Nx)';
up=(sum(icp2')/Nx)';
uxx(1,1:Nx)=ux(1);
uxx(2,1:Nx)=ux(2);
uxx(3,1:Nx)=ux(3);
upp(1,1:Nx)=up(1);
upp(2,1:Nx)=up(2);
upp(3,1:Nx)=up(3);
x=icp1-uxx;
p=icp2-upp;
W=x*p';
[U,S,V] = svd(W);
R=U*V';
T=ux-R*up;
cape=reshape(real2,[307200,3])';
cap=R*cape;
cap=reshape(cap',[480,640,3]);
cap=reshape(cap,[307200,3])';
end
```

```
function [ T ] = obtdt( real1,real2,ptsc1,ptsc2)
T=zeros(3,1);
for i=1:length(ptsc1)
icpp1(:,i)=double(real1(ptsc1(i,2),ptsc1(i,1),:));
icpp2(:,i)=double(real2(ptsc2(i,2),ptsc2(i,1),:));
end
c=1;
for i=1:length(icpp1)
    if
(isnan(icpp1(3,i))||isnan(icpp2(3,i))||icpp1(3,i)==0||icpp2(3,i)==0)
    else
     ptssc1(c,:)=ptsc1(i,:);
     ptssc2(c,:)=ptsc2(i,:);
     c=c+1;
    end
end
c=c-1;
for p=1:c
T1(:,p)=double((real2(ptssc2(p,2),ptssc2(p,1),:)-
real1(ptssc1(p,2),ptssc1(p,1),:)));
end
for p=1:c
T(:,1)=T(:,1)+T1(:,p);
end
T=T/c;
end


function [ M,C,Irgb,real, ptsc, depth,k,tt] =
proceso(ser,valorptsc,tol,prof,rr,valfil )

addpath('./Mex');
if isempty(instrfind)
else
fclose(instrfind);
delete (instrfind)
end

%% com serial
a=serial(strcat('COM',num2str(ser)));
fopen(a);
pause(2)
fprintf(a,'6');

% inicializaciones
flagg=1;
k=0;
err=0;
tipo=1;
val=1000;
%% capturas y filtros

context = mxNiCreateContext('Config/SamplesConfig.xml');
```

```
option.adjust_view_point = true;
mxNiUpdateContext(context, option);
display('En Espera')
h1 = imagesc(zeros(480,640,3,'uint8'));
while(flagg)
     flag=0;
    while(flag==0 )
      if (a.BytesAvailable >1)
          vard=fscanf(a);
      if (length(vard)>=1)
           flag=1;
      end
      end

   mxNiUpdateContext(context, option);
   [rgb, ~] = mxNiImage(context);
    set(h1,'CData',rgb);
    drawnow;

    pause(0.02)
    end

    va='a'==vard;
    vb='b'==vard;
  if( va(1))
     display(strcat('toma',num2str(k+1)))

    pause(0.02)
    fprintf(a,'3');
    while(a.BytesAvailable <3)
    pause(0.02)
    end
    angulo=fscanf(a);
    display(angulo,'Ángulo')
    mxNiUpdateContext(context, option);
   [ Ir,Pro,Da,Pr] = tomaskinect( context,option,prof );
    Irgb.(['a' num2str(k+1)])=Ir;
    depth.(['a' num2str(k+1)])=Pro;
    real.(['a' num2str(k+1)])=Da;
    projective.(['a' num2str(k+1)])=Pr;

   if k>0
        [l1,l2] = busquedakeypoints( real.(['a' num2str(k)]),...
        real.(['a' num2str(k+1)]),Irgb.(['a' num2str(k)]),...
        Irgb.(['a' num2str(k+1)]),tipo,val,tol );

        ptsc.(['a' num2str((2*k)-1)])=l1;
        ptsc.(['a' num2str(2*k)])=l2;
        clear l1 l2
        err=busquedaerror(ptsc.(['a' num2str((2*k)-1)]),valorptsc) & ...
        busquedaerror(ptsc.(['a' num2str(2*k)]),valorptsc);
    if err
         fprintf(a,'4');
```

```
          k=k+1;
          display('ok')

     else
          fprintf(a,'5');
          display('no suficientes coincidencias')
          clear Irgb.(['a' num2str(k+1)]) real.(['a' num2str(k+1)]) ...
          depth.(['a' num2str(k+1)]) projective.(['a' num2str(k+1)]) ...
          ptsc.(['a' num2str((2*k)-1)]) ptsc.(['a' num2str(2*k)])
     end
     else
         fprintf(a,'4');
         k=k+1;
         display('ok')
    end
   end
    if vb(1)
         flagg=0;
    end
end
clear Ir Pro Da Pr
mxNiDeleteContext(context);
%Encuentro de Pares
for i=1:2:2*(k-1)
    [ ptsc.(['a' num2str(i)]),ptsc.(['a' num2str(i+1)]) ] = ...
    f_ptsc( ptsc.(['a' num2str(i)]),ptsc.(['a' num2str(i+1)]));
end
for i=1:1:2*(k-1)
    if length(ptsc.(['a' num2str(i)])(:,1))<10
        [ ptsc.(['a' num2str(i)]) ] = agrupar( ptsc.(['a' num2str(i)]) );
    end
end
%%  Union de Capturas
%Rotacion
tic
capturas.(['a' num2str(round(k/2))])=...
    reshape(real.(['a' num2str(round(k/2))]),[307200,3])';
for i=round(k/2):-1:2
    [jajac,jajar,jajat]=ICP_SVD(real.(['a' num2str(i)]),...
        real.(['a' num2str(i-1)]),ptsc.(['a' num2str(2*(i-1))]),...
        ptsc.(['a' num2str((2*(i-1))-1)]));
    capturas.(['a' num2str(i-1)])=jajac;
    R.(['a' num2str(i-1)])=jajar;
    T.(['a' num2str(i-1)])=jajat;
     for j=i-1:-1:1
real.(['a' num2str(j)])=applyicp(real.(['a' num2str(j)]),jajar);
     end
  end
for i=round(k/2):k-1
    [jajac,jajar,jajat]=ICP_SVD(real.(['a' num2str(i)]),...
    real.(['a' num2str(i+1)]),ptsc.(['a' num2str((2*i)-1)]),...
    ptsc.(['a' num2str(2*i)]));
    capturas.(['a' num2str(i+1)])=jajac;
    R.(['a' num2str(i+1)])=jajar;
```

```
    T.(['a' num2str(i+1)])=jajat;
    for j=i+1:k
real.(['a' num2str(j)])=applyicp(real.(['a' num2str(j)]),jajar);
    end
end
fprintf('time: %4.4f s\n',toc);
% Traslacion
tic
for e=1:k-1
    T.(['a' num2str(e+1)])=obtdt(real.(['a' num2str(e)]),...
        real.(['a' num2str(e+1)]),ptsc.(['a' num2str((2*e)-1)]),...
        ptsc.(['a' num2str(2*e)]));

end
fprintf('time: %4.4f s\n',toc);
tic
for i=1:round(k/2)-1
    for j=i:-1:1
    capturas.(['a' num2str(j)])=applyt(capturas.(['a' num2str(j)]),...
        -1*T.(['a' num2str(i+1)]));
    end
end
for i=round(k/2)+1:k
    for j=i:k
    capturas.(['a' num2str(j)])=applyt(capturas.(['a' num2str(j)]),...
        T.(['a' num2str(i)]));
    end
end
fprintf('time: %4.4f s\n',toc);
display('okok')
%Filtro Contorno
tic
for i=1:k
    [MC,IC] = filtcontorno2(capturas.(['a' num2str(i)]),Irgb.(['a'
num2str(i)]),rr);
    cap.(['a' num2str(i)])=MC;
    Ir.(['a' num2str(i)])=IC;
end
fprintf('time: %4.4f s\n',toc);
%% visualizacion
MM=cap.a1;
RGBB=Ir.a1;
for l=2:k
   MM=[MM;cap.(['a' num2str(l)])];
   RGBB=[RGBB;Ir.(['a' num2str(l)])];
end
M=MM;
RGB=RGBB;
la=isnan(M(:,3)) | M(:,3)==0;
M(la,:)=[];
RGB(la,:)=[];

R180=[-1 0 0;0 1 0;0 0 -1];
M=(R180*M')';
```

```
C=(double(RGB)./255);

[M,C]=filtrorepetidos(double(M),C,valfil);
M=M';
TTT(1,1:length(M(1,:)))=min(M(1,:));
TTT(2,1:length(M(1,:)))=min(M(2,:));
TTT(3,1:length(M(1,:)))=min(M(3,:));
M=M-TTT;
M=M';
clear TTT
tt=delaunayn(M);
fprintf(a,'1');
fclose(a);
delete(a);
clear a;
display('Proceso Completado')
end


function [ Irgb,depth,real,projective] = tomaskinect( context,option,prof
)
mxNiUpdateContext(context, option);
[Irgb, depth] = mxNiImage(context);
real = mxNiConvertProjectiveToRealWorld(context, depth);
for h=1:480
   for s=1:640
      if (real(h,s,3)>prof)
          real(h,s,:)=[nan,nan,nan]';
      end
   end
end
projective = mxNiConvertRealWorldToProjective(context, real);
end


function [ trin,Cn,Xbn] = tri( MM,CC,r,tt )
t= filt_tri( tt,MM,r);
t1=triangulation(t,MM);
[tri,Xb]=freeBoundary(t1);
[~,IA,IB]=intersect(MM, Xb, 'rows');
Cn      = CC(IA,:);
Xbn     = Xb(IB,:);
iIB(IB) = 1:length(IB);
trin    = iIB(tri);
end
```

ANEXO C: Otras Reconstrucciones.

Reconstrucción sección sala.

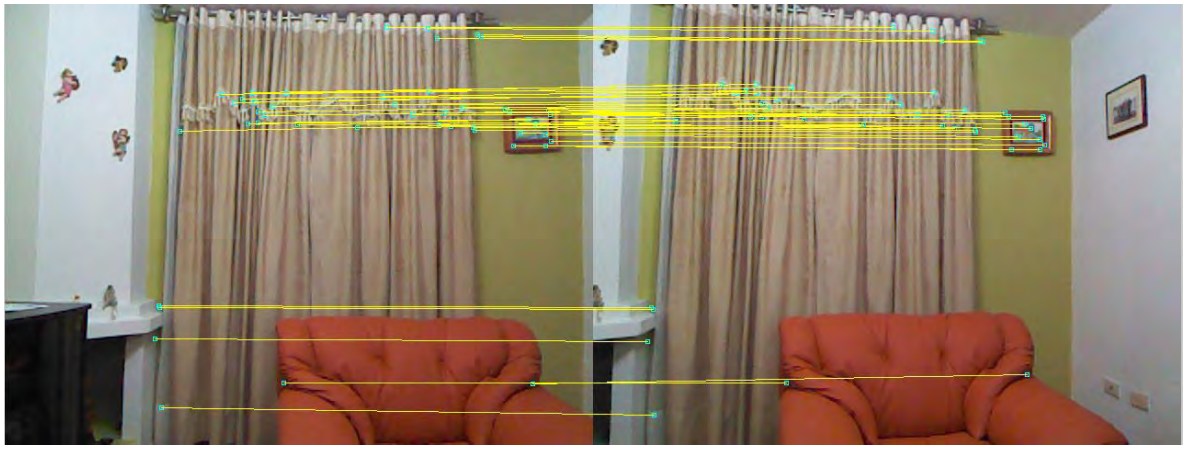


Figura 51 Gráfica par de imágenes sala.

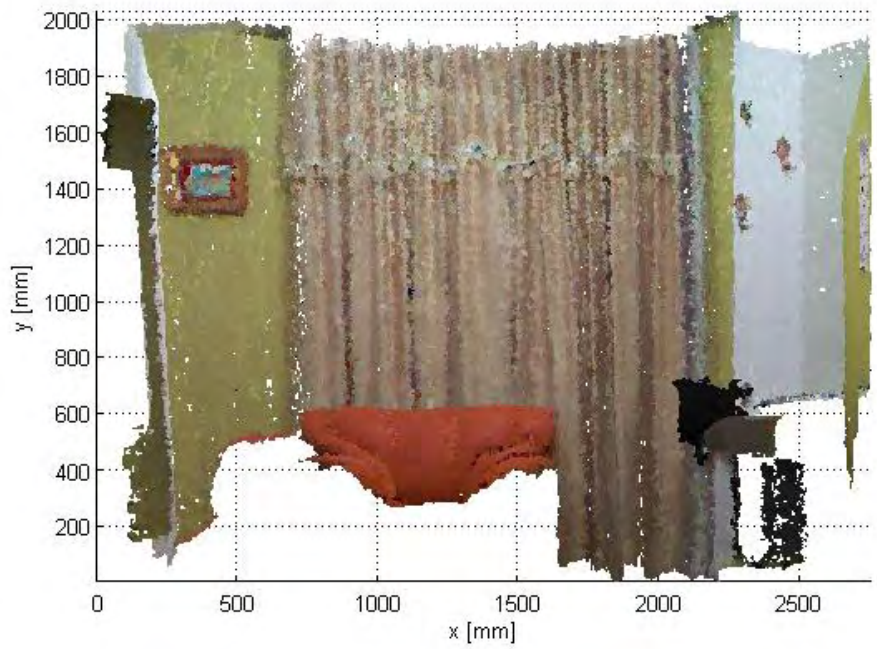


Figura 52 Reconstrucción sala. Vista frontal.

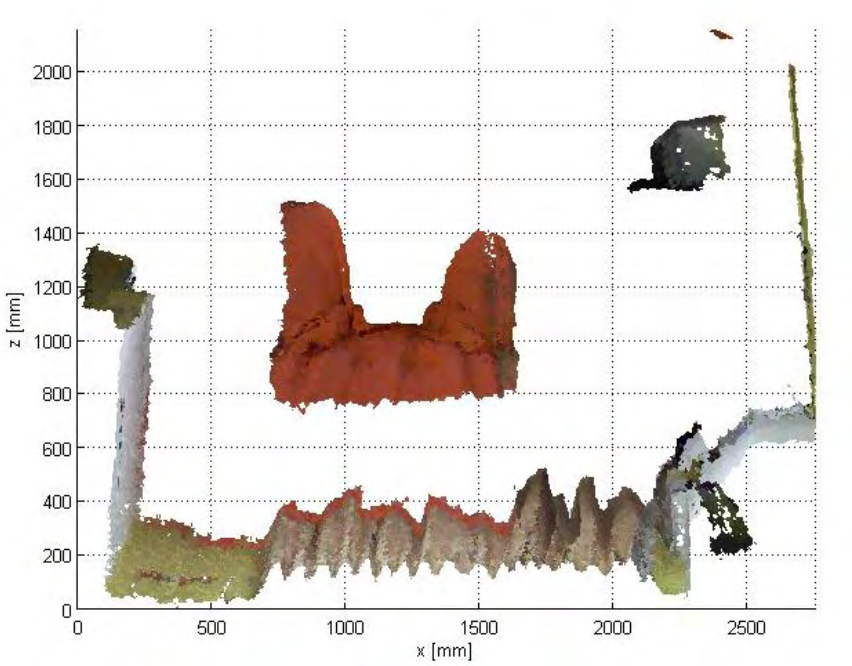


Figura 53 Reconstrucción sala. Vista superior.

Reconstrucción flores.

Figura 54 Gráfica de un par de imágenes de las flores con sus respectivas coincidencias.

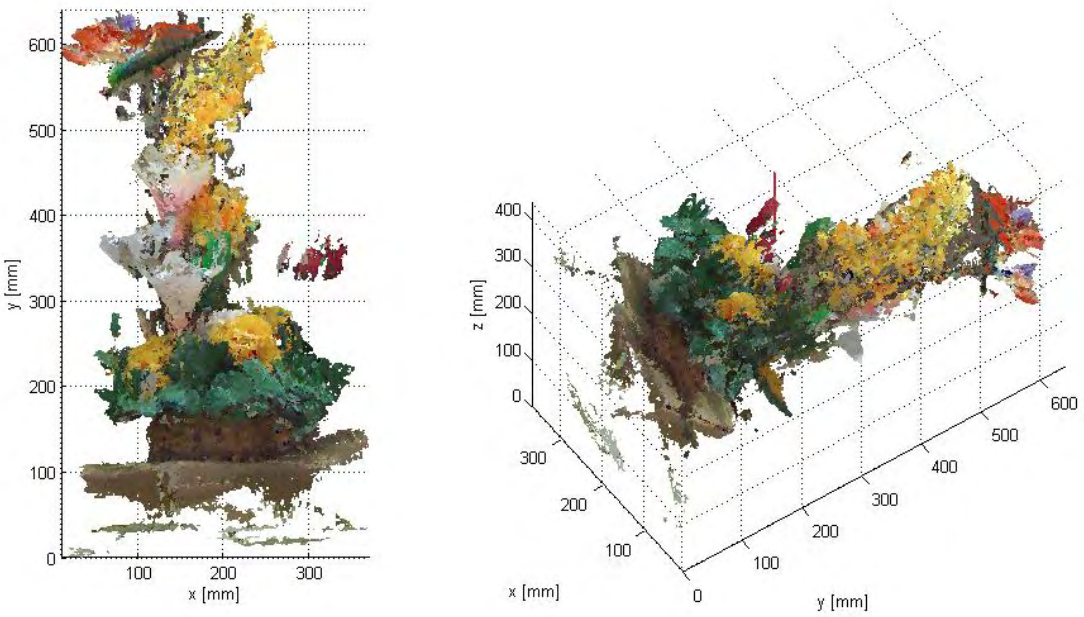
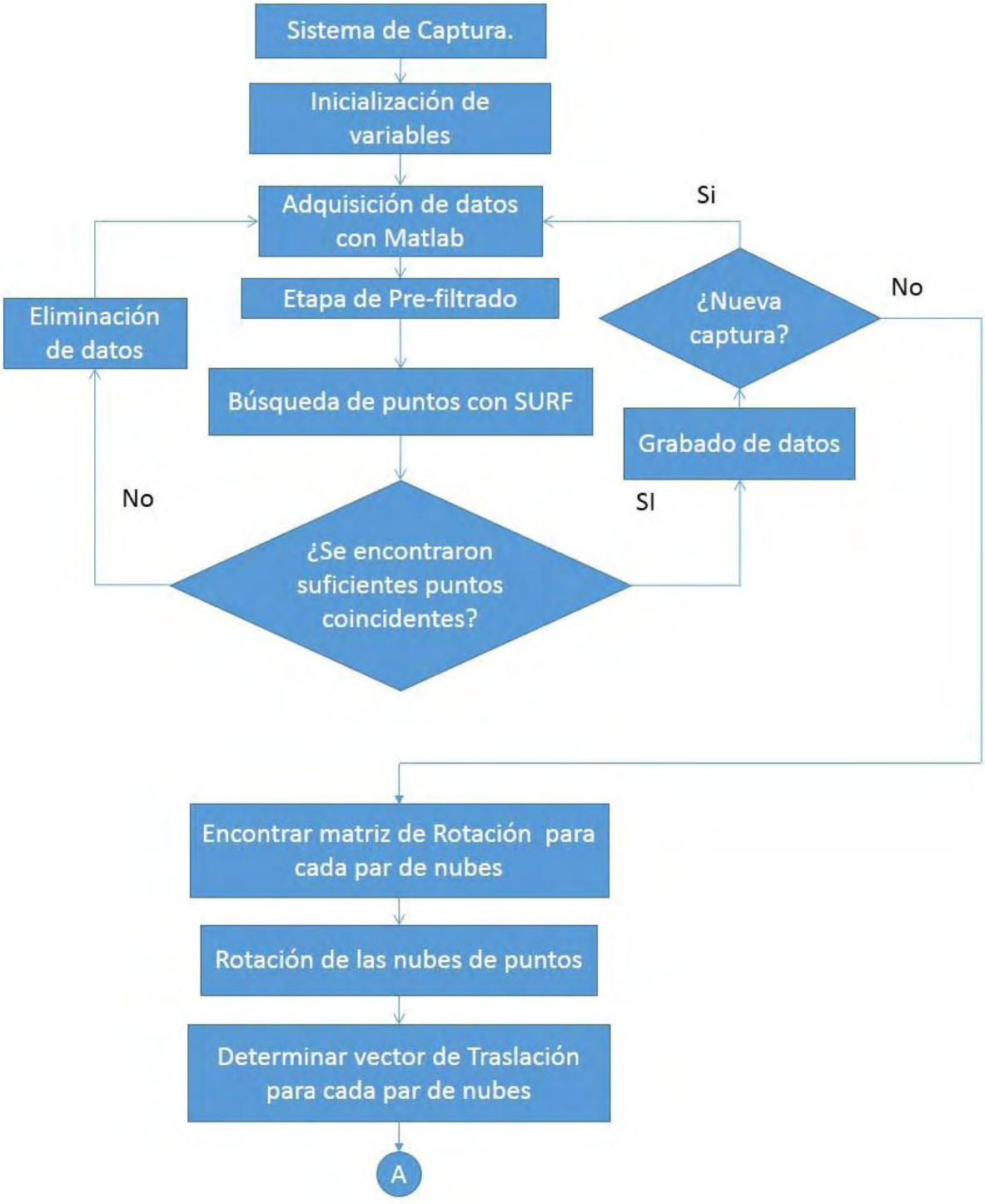


Figura 55 Reconstrucción Flores.

ANEXO D: Diagrama de Flujo.




A
Traslación de las nubes
Filtro del contorno de los datos
Agrupamiento de nubes
Filtro de Valores Repetidos
Triangulación
Guardado de datos
Visualización