INSTITUTO TECNOLÓGICO DE CHIHUAHUA
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

"MODELO TRIDIMENSIONAL DE UNA PERSONA A PARTIR DE LOS DATOS DE CUATRO SENSORES RGBD Y SU IMPRESIÓN EN 3D"

TESIS

QUE PARA OBTENER EL GRADO DE

MAESTRO EN INGENIERÍA MECATRÓNICA

PRESENTA:

ING. DANIEL RODRÍGUEZ SALGADO

DIRECTOR DE LA TESIS:
DR. ISIDRO ROBLEDO VEGA

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

CHIHUAHUA, CHIH., JUNIO DE 2019

Chihuahua, Chih. a de de .

C.
P r e s e n t e.

Por este conducto le comunico que esta División de Estudios de Posgrado e Investigación le ha concedido la autorización, para la impresión de su Tesis de Grado de Maestría, cuyo título es:

Con el siguiente contenido:

"LA TECNICA POR EL ENGRANDECIMIENTO DE MEXICO"

A t e n t a m e n t e

_____
M.F. LUIS CARDONA CHACÓN.
JEFE DIVISION DE ESTUDIOS DE POSGRADO.

Chihuahua, Chih. a 21 de junio de 2019 .

M.F. Luis Cardona Chacón.
Jefe de la División de Estudios de
Posgrado e Investigación del ITCh.

PRESENTE.

Por medio de la presente notificamos a Usted que, en cumplimiento parcial de los requerimientos para la obtención del Grado de Maestría en Ingeniería Mecatrónica, ha sido aprobado y aceptado para su impresión el documento de tesis del C. *ING. DANIEL RODRÍGUEZ SALGADO*. Cuyo título es: *"MODELO TRIDIMENSIONAL DE UNA PERSONA A PARTIR DE LOS DATOS DE CUATRO SENSORES RGBD Y SU IMPRESIÓN EN 3D"*.

Agradeciendo de antemano la atención prestada a la presente, quedamos de Usted.

_____

DR. ISIDRO ROBLEDO VEGA
Director de tesis.

_____

M.C. PEDRO RAFAEL MÁRQUEZ GUTIÉRREZ.
Miembro del comité revisor

_____

DRA. CARMEN LETICIA GARCÍA MATA.
Miembro del comité revisor

_____

M.C. ALBERTO PACHECO GÓNZALEZ.
Miembro del comité revisor

c.c.p. - Interesado.
- Comité revisor.
- Archivo.

Chihuahua, Chih., a 21 de junio de 2019.

Dra. María Elena Álvarez-Buylla
Director de CONACYT.

At'n. Luis Gil Cisneros

Dirección de Formación de Científicos y Tecnólogos.

P r e s e n t e.

Por este conducto aprovecho la ocasión para saludarlo e informarle que a la fecha he obtenido el Grado de Maestría en Ingeniería Mecatrónica en la División de Estudios de Posgrado e Investigación del Instituto Tecnológico de Chihuahua. Motivo por el cual agradezco todo el apoyo brindado por esta Institución que Usted representa, el otorgamiento de esta beca-crédito permitió dedicarme de tiempo completo a la realización de mis estudios de Posgrado y de esta manera lograr el cumplimiento del objetivo principal del convenio establecido.

Sin otro particular por el momento, me es grato quedar de Usted como su seguro servidor, no sin antes reiterar mi agradecimiento. ¡Muchas Gracias!

A t e n t a m e n t e

Daniel Rodríguez Salgado
Exbecario CONACYT No. 809644

c.c.p M.F. Luis Cardona Chacón.

Jefe de la División de Posgrado e Investigación

"No te rindas, por favor no cedas, aunque el frío queme, aunque el miedo muerda, aunque el sol se esconda y se calle el viento, aún hay fuego en tu alma, aún hay vida en tus sueños".

Mario Benedetti

AGRADECIMIENTOS Y DEDICATORIA

Nunca entendí cómo o por qué me aventuré en este proyecto, nunca entendí como las cosas salieron adelante, nunca entendí como cualquier persona puede hacer cosas extraordinarias, nunca entendí por qué en la vida se sufre. Ahora se que se llama Fé y Dios me la dio. Estas lágrimas de agradecimiento les corresponden a Él y a La Santísima Virgen María quienes nunca se olvidaron de mí.

Nunca entendí como a pesar de haberme encontrado en la peor versión, cuando no tenía nada que ofrecerle a la vida, seguías ahí; en la enfermedad, en el fracaso, en mis sueños en mis ilusiones, importándole mi ser con todo su corazón. Ahora sé que se llama amor y Dios me lo hizo notar en Margarita… mi ángel… Mi mamá.

Nunca entendí como a pesar de los golpes de la vida, la lejanía del hogar, el rechazo, las dificultades incluso la muerte de seres amados, es que se podía salir adelante con nuevo ímpetu, con el carácter fortalecido. Ahora se, que se llama madurez y lo encontré en Juan David… mi Papá, quien es mejor Ingeniero que yo.

Nunca entendí cómo es posible que, a pesar de las diferencias, de los problemas de la vida, es posible seguir unidos, bajo el mismo principio de vida que Dios y nuestros padres fomentaron en nosotros, ahora sé que se llama hermandad y es un privilegio ir creciendo junto a ustedes David y Diego.

Para Juanys y Lupe, ahora más que nunca recuerdo cuando navegaban conmigo y mis hermanos desde pequeños y ahora también forman parte de este momento. Para Josie y para Lía ejemplo de fortaleza en las dificultades. Para mi Tía Josefina a quien le apasiona la profesión docente. Para Ulises, Cynthia y Eva quienes acaban de emprender su viaje como familia.

Mi agradecimiento y dedicatoria para familia paterna y materna. En especial para mi abuelito Chon.

Al cuerpo académico, administrativo y de comité de tesis de la Maestría en Mecatrónica. Me voy con grandes enseñanzas en mi vida personal y académica, en especial al Dr. Isidro asesor y director de esta tesis. Gracias por la paciencia para conmigo.

Al Instituto Tecnológico de Chihuahua quienes me abrieron las puertas cuando en muchos otros lados se me cerraban; desde mi ingreso a la carrera, al posgrado y ahora en mi vida profesional. Al departamento de Eléctrica-Electrónica por haber confiado en mí a pesar de mi juventud e inexperiencia. En especial a la Maestra Martha, Mireya, al Inge Nicolás, Rodríguez Chico y Salvador que vieron al ingeniero que estaba escondido en mí y aún no lo podía descubrir.

A mis amigos que logré hacer en este tiempo, gracias por sus palabras y su paciencia, nunca terminaría de agradecerles a todos pero mi especial agradecimiento a mis amigos del Laboratorio de Sistemas de Visión; en especial a: Julio Omar, José Arzaga, Mariana Sáenz, Marco Flores, David Olivas. Ademas a Gabriel, Raúl Pablo, Alex, Homero, Humberto, Macías…

A mis alumnos, y estudiantes que les guste la investigación. Es maravilloso poder hacer esta labor.

Finalmente, quiero dedicarle estas palabras a Daniel; si llega el día en que Dios te conceda estar en una posición privilegiada que no se le olvide las cosas fundamentales de la vida; los valores, la educación, la empatía, la justicia y que en dónde Dios necesite tu presencia estés ahí sin importar lo demás.

RESUMEN

"MODELO TRIDIMENSIONAL DE UNA PERSONA A PARTIR DE LOS DATOS DE CUATRO SENSORES RGBD Y SU IMPRESIÓN EN 3D"

Daniel Rodríguez Salgado
Maestro en Ingeniería Mecatrónica
División de Estudios de Posgrado e Investigación del
Instituto Tecnológico de Chihuahua
Chihuahua, Chih. 2019
Director de Tesis: Dr. Isidro Robledo Vega

En este trabajo de investigación se desarrolló un sistema de escaneo, reconstrucción, modelado e impresión tridimensional de personas utilizando como medio de adquisición cuatro sensores de datos de color y profundidad denominados sensores RGBD.

La motivación que llevó a la construcción de un modelo tridimensional de personas, radica en que es posible mejorar aspectos que aquejan la calidad de vida tales como: la creación de prótesis para extremidades del cuerpo, análisis de movimiento para detección de enfermedades psicomtrices, la industria del entretenimiento como videojuegos; sistemas de simulación y la industria textil. Además, la ventaja principal es que el sistema es de bajo costo, asequible y eficiente para fines de impresión tridimensional.

La implementación del sistema de escaneo se estructuró en cuatro etapas, comenzando por la construcción del sistema de captura sincronizada con cuatro sensores RGBD. En esta etapa, se buscó establecer un área de captura que permita la adquisición del cuerpo completo de una persona. Se desarrolló el software necesario para producir las nubes de puntos tridimensionales con color para cada sensor.

La segunda etapa es la encargada de filtrar los datos capturados de cada uno de los sensores RGBD con la finalidad de evitar errores de reconstrucción debido a puntos indeseables y después lleva a cabo la fusión de datos mediante la transformación geométrica de las nubes de puntos de los cuatro sensores para generar una sola nube de puntos.

La tercera etapa aborda la construcción del modelo tridimensional. Se realiza el procesamiento de la nube de puntos fusionados por medio de algoritmos de filtrado, reconstrucción, mallado y suavizado para generar el modelo tridimensional de la persona escaneada y almacenarlo en archivos con formato de esterolitografía (STL) que puede ser procesado por impresoras 3D.

Como etapa final, se manufacturó el modelo de la persona capturada haciendo uso de la técnica de impresión 3D. Se hicieron pruebas con dos diferentes impresoras, con el fin de comparar el software de impresión y la calidad general del modelo generado.

Como resultados a destacar se obtiene un sistema de escaneo eficiente del cual se pueda generar un modelo tridimensional lo mas apegado a la persona capturada y además sea impreso tridimensionalmente. El proceso de fusión no representa dispersión significativa en sus datos lo que convierte al sistema de escaneo, como un sistema probado para futuras investigaciones.

ÍNDICE

LISTA DE FIGURAS

I. INTRODUCCIÓN

II. MARCO TEÓRICO

III. SISTEMA DE CAPTURA DE DATOS SINCRONIZADOS CON CUATRO SENSORES RGBD

IV. FUSIÓN DE DATOS TRIDIMENSIONALES DE CUATRO SENSORES RGBD

V. PROCESO DE IMPRESIÓN TRIDIMENSIONAL

ANEXO 5: MODELOS TRIDIMENSIONALES DE PERSONAS A PRUEBA

LISTA DE TABLAS

I. INTRODUCCIÓN

Los sistemas de Visión por Computadora constituyen una realidad en la que el ser humano día a día busca implementarlos para la solución de diversos problemas, como en medicina, sistemas de inspección, antropometría, manufactura, sistemas de vigilancia, sistemas de carácter militar y todos aquellos sistemas en los que se pueda emular el comportamiento de la visión humana. Cada una de estas aplicaciones pertenece a un área específica de estudio en los sistemas de visión comprendiendo temas como: el procesamiento y reconstrucción de imágenes, sistemas de video, sistemas de visión orientados a robótica, reconocimiento de patrones, modelado tridimensional, entre otros.

Particularmente como motivo de estudio, la construcción de un modelo tridimensional de una persona ofrece oportunidades en investigación y desarrollo donde el cuerpo humano es el elemento de estudio buscando soluciones que mejoren la calidad de vida de las personas, por ejemplo, la creación de prótesis para extremidades del cuerpo, análisis de movimiento para detección de enfermedades psicomotrices, la industria del entretenimiento como videojuegos, sistemas de simulación, la industria textil, la milicia y la biomédica.

Los sistemas de escaneo surgen a partir de la técnica de visión en estéreo, la cual proporciona la habilidad de inferir información de la estructura y distancia de una escena en 3D a partir de dos o más imágenes tomadas desde diferentes puntos de vista [1]. Por medio de arreglos de diodos láser que proyecta un haz de luces emitido hacia una superficie era posible que los rayos láser fueran deformados por la superficie del objeto, estas deformaciones eran procesadas por un sensor CCD para su procesamiento computacional.

La compañía Cyberware fue la primera en implementar sistemas de escaneo del cuerpo humano a mediados de los años 1980, con fines antropométricos usando la implementación anterior, añadiendo más arreglos de sensores y colocando a la persona sobre una superficie giratoria durante el escaneo, de tal suerte, que en el modelo tridimensional no existan zonas de oclusión que eviten una reconstrucción óptima [2].

Con el avance tecnológico, fueron surgiendo nuevas técnicas de escaneo como el método de Medición de Fase por Perfilometria (PMP) para adquisición de imágenes en 3D. Estas se encuentran basads en el modelo de medición de Moiré, el cuál consiste en utilizar una luz

blanca para proyectar un patrón de franjas sinusoidales sobre la superficie del objeto. El resultado de esta deformación también es tomado por una cámara con sensor CCD en conexión con una computadora para el procesamiento de los datos [2].

Son variados los tipos de sistemas de escaneo del cuerpo humano, teniendo como principal desventaja el costo y el tiempo de procesamiento de los datos. Con el surgimiento de nuevos sensores ópticos de bajo costo es posible la construcción de un sistema de escaneo. Tal es el caso de los sensores de movimiento, como el sensor Kinect desarrollado por la compañía Microsoft. Los sensores de movimiento o sensores RGBD son dispositivos capaces de proporcionar datos de color (*red, green, blue*) y de profundidad (*depth*) y son una alternativa accesible para el escaneo tridimensional de personas en movimiento.

La representación tridimensional del cuerpo humano es un campo que brinda diferentes alternativas de solución y análisis en diversas áreas del conocimiento en los que se ven involucrados directamente aspectos de la morfología del mismo. Tales disciplinas abarcan los campos de:

1. Medicina. Como en cirugías craneofaciales, donde los cirujanos que intervendrán en la operación les es de utilidad tener un panorama general de cómo se encuentra la composición craneal del individuo haciendo que la operación se vuelva más segura. La aplicación de los sistemas de visión en el modelado del cuerpo humano, permite identificar problemas que no son perceptibles de manera instantánea en la persona, como por ejemplo, ayudan a descubrir prematuramente cuando una persona sufre de síndromes o deficiencias psicomotrices y enfermedades como Parkinson. Esto a través del estudio del modelo de movimiento y la comparación con patrones estandarizados considerados como normales en un individuo.

2. Antropometría. Relacionado con el estudio de las dimensiones del cuerpo, con fines ergonómicos, estadísticos y de medicina. Por ejemplo, en las empresas se consideran las dimensiones del cuerpo humano para el diseño de lugares de trabajo, conocido como ergonomía. Además, en aplicaciones de carácter militar donde es necesario conocer de manera exacta la morfología de la persona para fines de equipamiento.

3. Industria. En la industria textil, al momento de establecer procesos de manufactura para diferentes tallas de prendas generalmente se investigan estadísticas antropométricas de peso, talla y género del individuo. Actualmente númerosas empresas recurren a tecnologías de

escaneo tridimensional para obtener estos datos con mayor precisión ahorrando tiempo de análisis estadístico.

4. Sistemas de video y reconocimiento. Bases de datos que poseen información antropométrica utilizados en hospitales y clínicas especializadas, sistemas de seguridad, sistemas de simulación para la industria textil y reconocimiento de patrones [2].

Tomando en cuenta las áreas de aplicación mencionadas, se tuvo la motivación de diseñar y construir un modelo tridimensional a escala del cuerpo humano completo a partir de los datos obtenidos de cuatro sensores RGBD y su impresión tridimensional (*3D printing*), esto llevó a implementar una metodología en esta investigación para desarrollar un sistema que consistió en cuatro etapas o fases:

1. La captura sincronizada de datos de los cuatro sensores RGBD
2. La Fusión de los datos obtenidos por los cuatro sensores
3. La construcción del modelo tridimensional del cuerpo humano
4. La conversión de los datos del modelo hacia el formato que requieren los dispositivos de impresión en 3D y su impresión.

La Figura 1.1 muestra un diagrama con las cuatro etapas del proyecto de tesis.

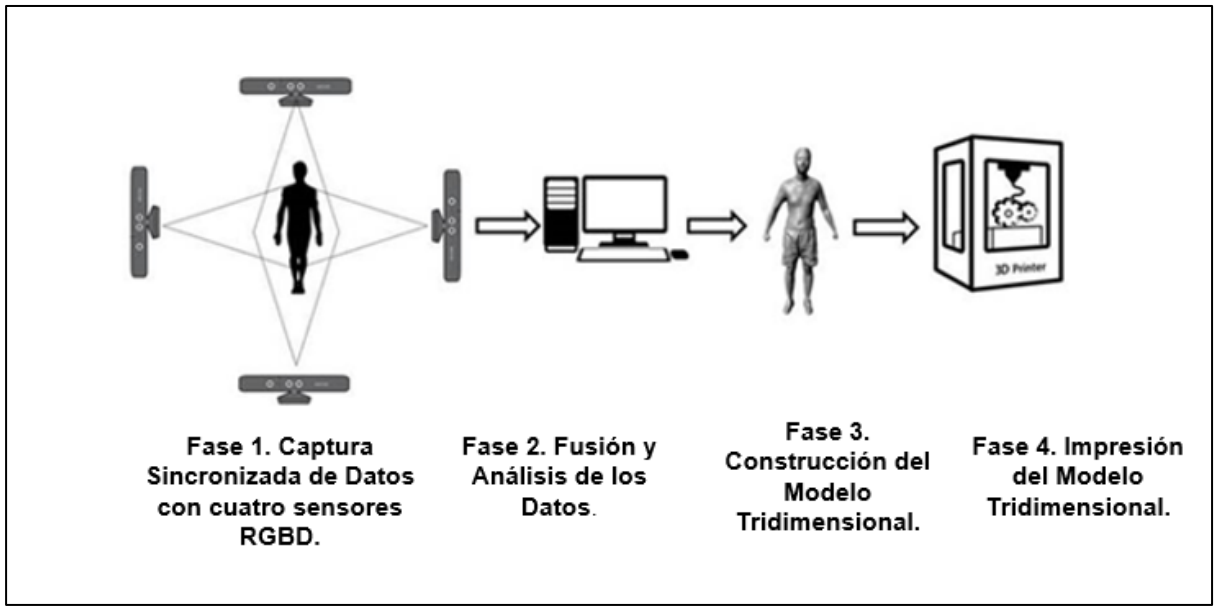


Figura 1.1. Sistema de Modelado Tridimensional del Cuerpo Humano Completo con Datos Capturados por Cuatro Sensores RGBD.

Se utilizaron datos adquiridos con cuatro sensores RGBD para realizar la reconstrucción tridimensional del cuerpo humano con el fin de probar la hipótesis de que el sistema diseñado y construido puede generar un modelo del cuerpo humano con una mayor cantidad de información obtenida desde cuatro diferentes orientaciones reduciendo de manera importante las oclusiones que generen errores en el modelo, aunado a la idea de que es posible realizar el procesamiento de los datos tridimensionales para aplicaciones innovadoras como es la impresión en 3D, así como la idea de que es posible la construcción de un sistema de escaneo asequible y eficiente para la obtención del modelo del cuerpo.

La impresión 3D comienza a surgir en 1983, cuando el investigador Chuck Null plantea métodos con esterolitografia [3]. Basado en esta técnica, Null crea patentes como el archivo STL (Stereo Litography) utilizado en la actualidad como el formato de impresión en muchas máquinas de impresión 3D. Hasta el año 2014 fue que estas tecnologías empezaron a tener un auge importante en disciplinas como la ingeniería, manufactura, prototipos de diseño mecánico y de materiales.

La tecnología de impresión en 3D ha revolucionado las técnicas de manufactura asistida por computadora y se han vuelto indispensables para el diseño y desarrollo de productos y prototipos mecánicos, ya que representa principalmente una ventaja económica menos costosa que los procesos clásicos como el maquinado CNC.

Dependiendo del tipo material y de la técnica empleada para imprimir, las impresoras 3D se clasifican en:

1. Impresoras 3D por Estereolitografía (SLA). Este tipo de impresión es una de las más importantes y de mayor disponibilidad en el mercado. Esta tecnología involucra la solidificación de un polímero líquido fotosensible, el cual se encuentra expuesto a una fuente de radiación ultravioleta que contiene la energía necesaria para que exista una reacción química en el material. Este tipo de impresora genera piezas de altísima calidad en su detallado aunque el desperdicio de material es grande. La Figura 1.2 [4] muestra la constitución de este tipo de impresoras, en donde el polímero líquido se encuentra en un recipiente y conforme una superficie elevadora va subiendo, el laser se encarga de solidificarla de acuerdo a la geometría de la pieza a imprimir.

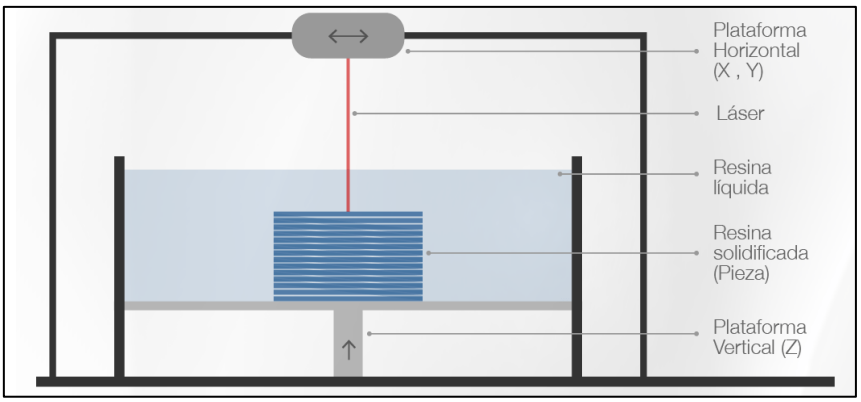


Figura 1.2. Impresión por Estereolitografía.

2. Impresoras 3D por Sinterización Selectiva con Láser (SLS). Permite el uso de materiales en polvo como la cerámica, cristal y nylon. Un haz láser es dirigido en un plano horizontal sobre una cama de material compactado que va reaccionando al entrar en contacto con el rayo láser. A esta acción se le conoce como Sinterización Selectiva [4], es decir, la solidificación se crea cuando el polvo entra en contacto con el láser. La Figura 1.3 [4] muestra la estructura de este tipo de impresión en donde después de que el laser reacciona con el polvo, la cama de material baja en el eje vertical y un rodilo pasa llenando y compactando la superficie para prepararlo para la siguiente capa.

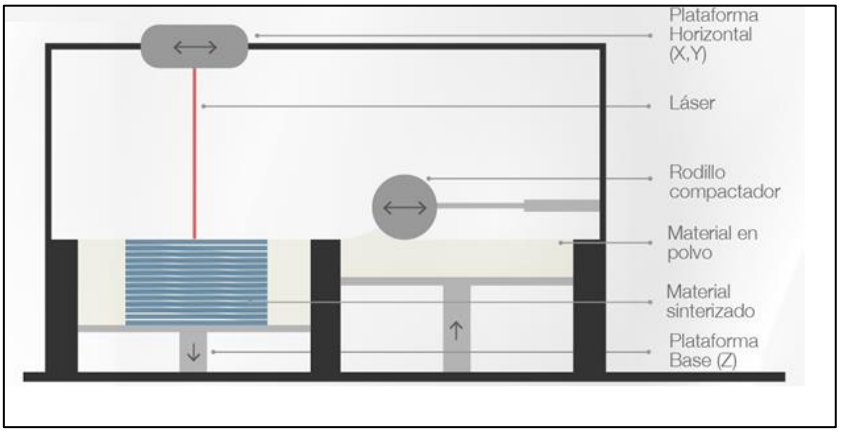


Figura 1.3. Impresión por Sinterización Selectiva.

3. Impresión por Deposición de Material Fundido (FDM). Tienen su principio de funcionamiento en máquinas de control numérico donde se procesa un modelo computarizado del objeto a reproducir. El material plástico es aplicado capa por capa hasta que el objeto

termina de imprimirse. Los principales mecanismos para una impresora genérica son: un mecanismo posicionador para cada tripleta de coordenadas (X, Y, Z) y un extrusor encargado de preparar el filamento de plástico con el que se elaborará la pieza [5]. La Figura 1.4 [4] esquematiza una impresora de esta naturaleza.

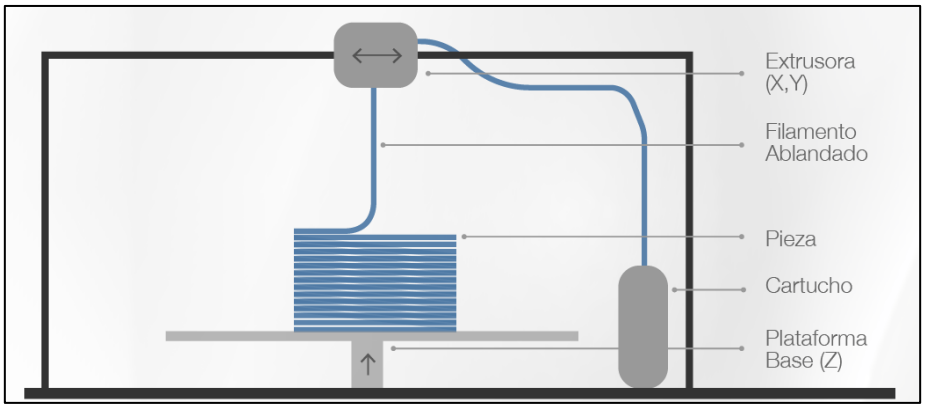


Figura 1.4. Impresión por Deposición de Material Fundido.

Para este proyecto de tesis se seleccionó la impresión en 3D del tipo FDM, específicamente una impresora de la marca Makerbot 2X Replicator como primera prueba, debido a que se encontraba disponible para trabajar con ella, además de tener la ventaja de que es accesible a todo tipo de mercado y su desempeño mecánico es eficiente y simple. Después se realizaron pruebas con una impresora de gama media de la marca Ender 3 Pro con el objetivo de comparar resultados de manufactura con la primera impresora Makerbot 2X Replicator la cual está considerada dentro de la gama alta de impresoras. En capítulos posteriores se ahondará más en el funcionamiento mecánico, eléctrico y de control de estas tecnologías, así como de los sistemas y modelos tridimensionales.

En el desarrollo de esta tesis los dos primeros capítulos abordan las diferentes investigaciones y teorías que llevaron a la concepción e implementación del sistema de escaneo y modelado tridimensional, así como las características técnicas de los elementos que lo conforman.

El capítulo tres se considera fundamental en el proyecto y explica dos situaciones: la primera aborda cómo se colocaron estratégicamente los sensores dentro de un área diseñada para obtener la mejor adquisición de datos del cuerpo, y la segunda cómo se modificó la estructura del algoritmo propuesto en [6] para que la nube de puntos de la persona incluyera la

información a color codificada en formato RGB haciendo un mapeo entre la imagen de color y la imagen de profundidad que ofrece el sensor RGBD. También describe cómo se realiza la sincronización del inicio de captura de datos con cuatro sensores RGBD utilizando *sockets* mediante el protocolo de comunicación TCP/IP.

El capítulo cuatro presenta y explica los algoritmos que filtran y fusionan las nubes de puntos tridimensionales de cada sensor RGBD considerando el área de captura correspondiente en la que se colocaron los cuatro sensores. Se creó una función la cual se encarga de realizar esta tárea cuyos parámetros de entrada son las nubes de puntos de los cuatro sensores, teniendo a la salida la nube de puntos fusionada en formato poligonal (*.ply*) y en forma de matriz con formato *.mat* de Matlab.

En el capítulo cinco se describe la construcción del modelo tridimensional de la persona a partir de los datos almacenados en el archivo con extensión *.ply* utilizando procesos de filtrado, mallado, creación de superficies y suavizado; estos últimos basados en los algoritmos de Poisson y el Pivoteo de Pelota (*Ball Pivoting*). El objetivo de este capítulo describir el modelo tridimensional de la persona y guardarlo en un archivo con el formato adecuado para su impresión en 3D. Posteriormente se realizan pruebas de impresión modificando parámetros importantes en la manufactura como es la temperatura del material, la densidad del material de impresión, el patrón de impresión, la escala de impresión, entre otras.

Además, en está tesis de tesis se presentan las características técnicas del hardware y software utilizado para la construcción del sistema de escaneo; las implicaciones que conlleva su implementación y las diferencias que existen con el equipo utilizado en investigaciones pasadas.

De esta investigación se desprenden áreas de oportunidad que pueden beneficiarse de este sistema de escaneo; investigaciones relacionadas a los sistemas biométricos como la reconstrucción tridimensional de zonas específicas del cuerpo, diseño de prótesis, análisis de movimiento, reconocimiento facial y bases de datos biométricas. La segunda área tiene que ver con la manufactura de piezas y objetos haciendo que el programa de captura en vez de segmentar a la figura humana, segmente la pieza en cuestión.

II. MARCO TEÓRICO

2.1 Fundamentos de Visión por Computadora

El objetivo de los sistemas de visión por computadora es tomar decisiones sobre objetos reales y escenarios basado en el muestreo de imágenes estudiando las siguientes técnicas:

- Sensado y formación de imágenes. Obtención de las imágenes del mundo exterior y las propiedades involucradas en la adquisición de la imagen tales como el material, la forma, la iluminación y la posición espacial del objeto en cuestión.
- Codificación de la información. Estudia cómo la información obtenida de un sensor permite identificar texturas, geometría, movimiento o identidad del o los objetos de interés en la escena.
- Representación. Estudia que representaciones de la imagen deberían ser utilizadas para almacenar la información de los objetos con sus partes y sus propiedades.
- Algoritmos. Son los métodos matemáticos para procesar la información de una imagen y de esta forma construir descripciones del mundo exterior y sus objetos [7].

Hablar de la estructura de un sistema de visión, involucra un proceso que va desde la obtención de la imagen hasta la toma de decisiones por medio del análisis de esta. Debido a que cada etapa posee sus complejidades, se vio la necesidad de dividir en ramas de estudio como lo son el procesamiento de imágenes, rama encargada del realce de atributos en la imagen además de la eliminación de factores que impiden que dichos atributos sean distinguibles como el ruido. Visión por computadora es la etapa posterior al procesamiento y la obtención de la imagen, responde al cuestionamiento del qué se hará con la imagen procesada, qué decisiones se toman y que descriptores de características es posible obtener de ellas. Tal es la cantidad de descriptores de rasgos a definir en las imágenes que surge la rama de reconocimiento de patrones que realiza esta labor para su clasificación en conjuntos de descriptores. La Figura 2.1 muestra el esquema con la relación entre las mencionadas áreas de estudio presentes en un sistema básico de visión por computadora.

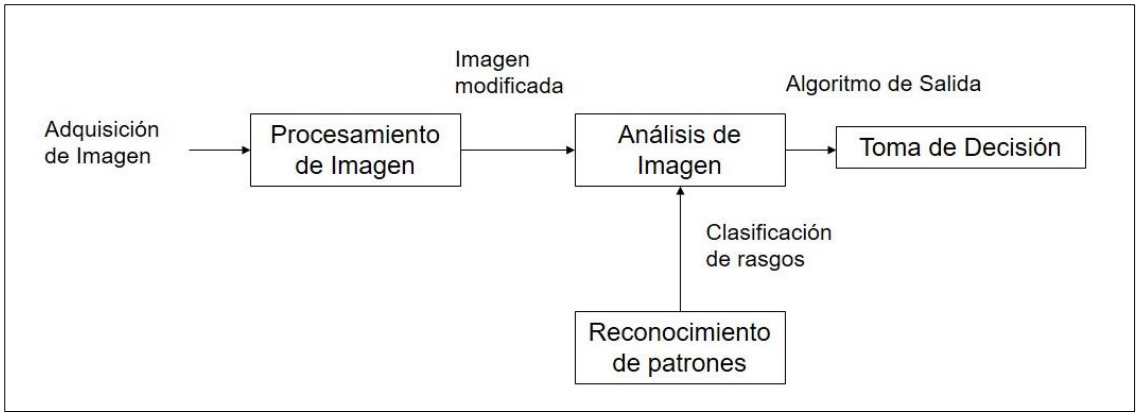


Figura 2.1. Esquema de un sistema básico de visión por computadora.

2.1.1 Formación de imágenes

El proceso de formación de imágenes en un sistema de visión trata de simular el mismo proceso de adquisición del ojo humano e inicia cuando los rayos de luz entran a través de la apertura angular (pupila) de la cámara y llegan a la pantalla o plano de la imagen [1].

La Figura 2.2 [7] muestra un modelo común en la toma de una imagen mediante una cámara. El modelo consta de una superficie iluminada por la luz natural o una bombilla emisora de luz. La superficie iluminada refleja la radiación hacia la cámara a través de sensores que captan la cantidad de energía luminosa reflejada por el objeto.

Los sensores encargados de realizar esta función son denominados dispositivos de acoplamiento por carga (CCD - *Charged Coupled Devices*) y en ellos las cargas luminosas que se acumulan en la totalidad del arreglo de fotosensores están enlazadas o acopladas de modo que pueden ser transferidas fuera del arreglo directamente (salida digital) o procesadas como una señal de video (salida analógica). Existen tres arquitecturas básicas para transferir las cargas fuera del arreglo CCD; a cuadro completo (FF – *Full Frame*), transferencia de cuadro (FT – *Frame Transfer*) y transferencia inter-línea (IT – *Interline Transfer*) [7].

La geometría de la formación de imágenes puede ser conceptualizada como la proyección de cada punto de la escena 3D a través del centro de proyección de la lente hacia el plano de la imagen. La intensidad de cada punto de la imagen está relacionada con la intensidad radiante de la superficie tridimensional, el modelo de la cámara tipo *pinhole* (orificio de alfiler)

explicado más adelante es el apropiado para representar matemáticamente reconstrucciones en 3D.

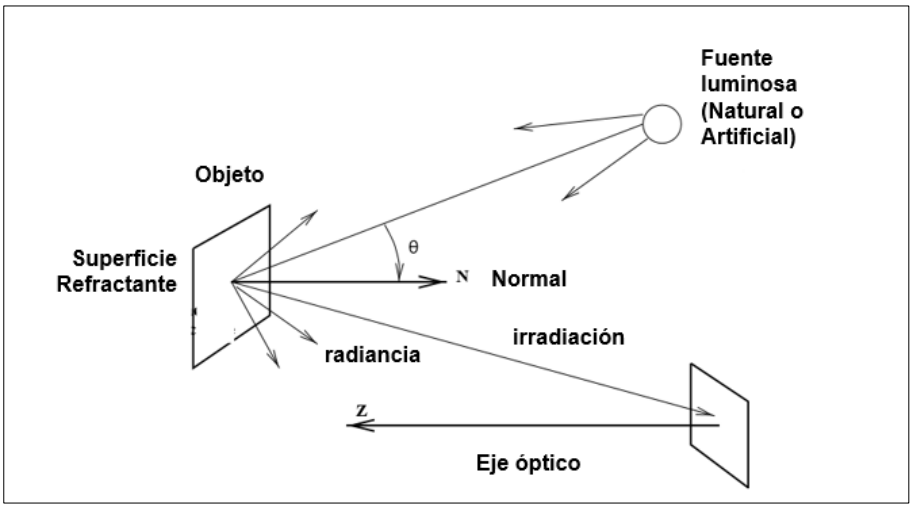


Figura 2.2. Proceso de formación de la imagen en un punto.

Para el caso de la identificación de colores en el ser humano, biológicamente la formación y percepción se deben a las diferentes células presentes en la retina del ojo humano que actúan en función de la luz a procesar, estas células son llamadas como *bastones* y *conos*. Los bastones en el ojo, son los responsables para la visualización de la luz en niveles bajos, es decir, permiten distinguir el color negro, el blanco, y los tonos de grises. Los conos, son los responsables de la visión y la función del color en niveles más altos de luz. Existen tres tipos de conos; los sensibles al color rojo que integran un 64% de todos ellos, el color verde con un 32% y el azul también con un porcentaje de 32%. Los conos están constituidos por pigmentos llamados opsinas, la cual es la unidad fundamental de los conos y son construidos por proteínas y derivados de la Vitamina A. La Figura 2.3 muestra los componentes del ojo humano que participan en la percepción del color. En las cámaras a color se crea un arreglo CCD para cada uno de estos colores triestímulo, de aquí surgen los canales RGB para la digitalización del color [8].

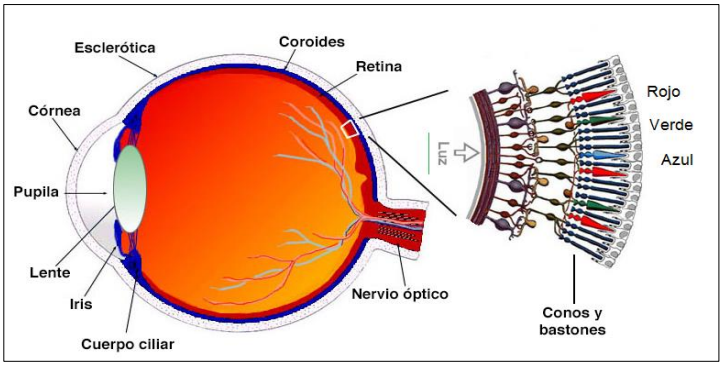


Figura 2.3. Percepción del color en el ojo humano.

2.1.2 Parámetros extrínsecos e intrínsecos de la cámara

Cuando se lleva la reconstrucción debe considerarse el marco de referencia del espacio físico o mundo real (**P**= [x, y, z]) y el marco de referencia de la cámara (**p**= [x, y]). Al proceso de correlacionar los puntos extraídos de los marcos de referencias para la reconstrucción tridimensional se le conoce como calibración.

Las coordenadas de los puntos de la imagen en el marco de referencia de la cámara son representadas en coordenadas en pixeles, que son las únicas conocidas en la imagen [1]. La Figura 2.4 muestra el modelo de perspectiva de la cámara, donde se observa la proyección del punto *P(x, y, z)* en el marco de referencia de la cámara hacia el punto *p(x,y)* en el plano de la imagen [9].

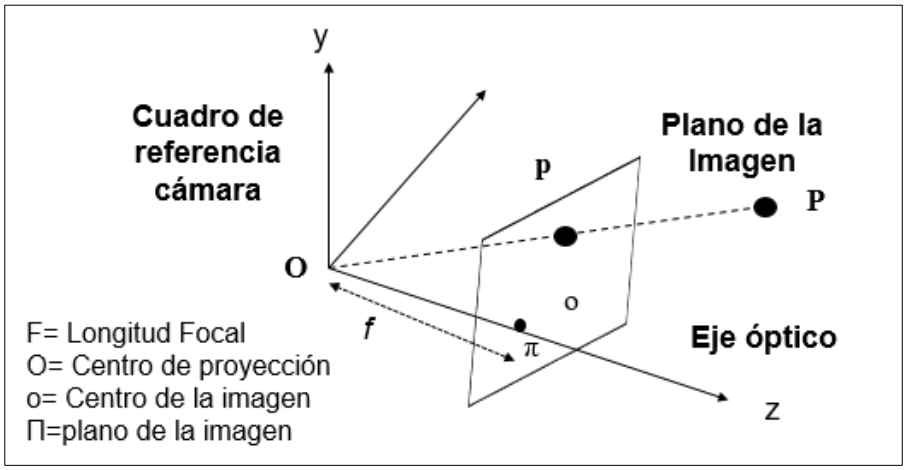


Figura 2.4. Modelo de la cámara en perspectiva.

El proceso de calibración tiene como característica adicional definir los parámetros o características que definen a la cámara y que se dividen en dos:

1. Parámetros Intrínsecos. Son los parámetros que dependen de cada cámara en particular y para una cámara tipo *Pinhole* incluye el centro de proyección que es la intersección entre el eje óptico y el plano de la imagen, dimensiones físicas de los pixeles (α y β), la longitud focal (f) y la distorsión radiométrica de la lente (k).

2. Parámetros Extrínsecos. Definen la localización del marco de referencia de la cámara con respecto al marco de referencia del mundo real. Matemáticamente se expresa como una transformación geométrica que consta de una traslación y una rotación, en ese orden (ecuación 2.1) [1] [9]. La Figura 2.5 indica la traslación y una rotación de los ejes coordenados del marco de referencia del mundo real para alinearlos con los ejes coordenados del marco de referencia de la cámara [9].

$$\boldsymbol{P} = R(\boldsymbol{p} - T) \tag{2.1}$$

Donde

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

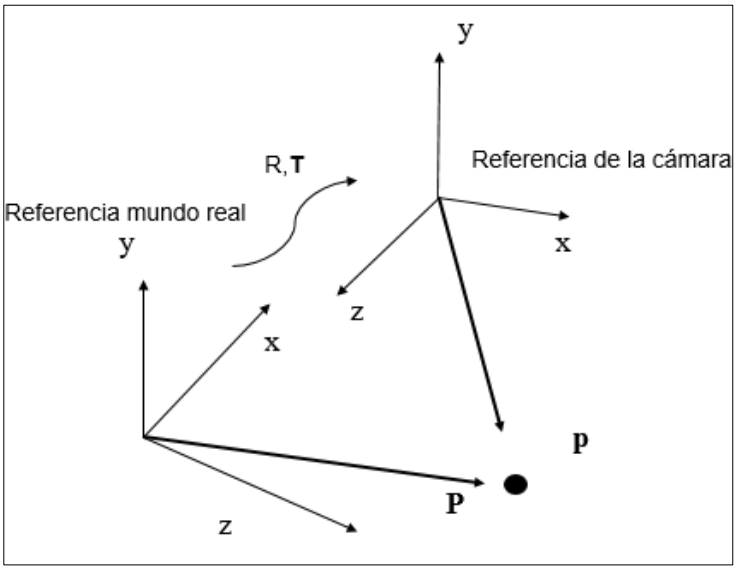


Figura 2.5. Relación entre los marcos de referencia del mundo real y de la cámara.

2.1.3 Visión en estéreo

Se define como una técnica para inferir puntos tridimensionales a partir de dos imágenes tomadas desde dos perspectivas diferentes. El proceso se encuentra fundamentado en la forma en que el ser humano percibe la distancia hacia un objeto utilizando la distancia entre las pupilas de ambos ojos, generando una medida de *disparidad* con respecto a la posición del objeto en la imagen de cada ojo. La disparidad es inversamente proporcional a la distancia que se encuentra con el objeto observado. Nótese que las proyecciones hacia ambas cámaras dependen de su campo de visión. Este debe tomarse en cuenta al momento de diseño del sistema de visión en estéreo para producir las medidas de disparidad deseadas. La Figura 2.6 muestra gráficamente el significado de la disparidad cuando se plantea un sistema en estéreo [6].

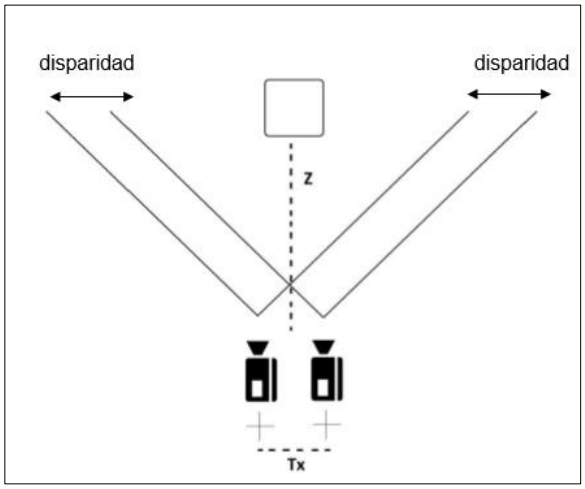


Figura 2.6. Sistema de visión en estéreo.

La construcción de un sistema de visión en estéreo enfrenta dos problemas fundamentales: la correspondencia entre imágenes y la reconstrucción tridimensional. El problema de correspondencia consiste en determinar qué puntos en la imagen izquierda corresponden a las proyecciones de los mismos puntos en 3D sobre la imagen derecha. Existen dos métodos utilizados, el método basado en correlación calcula una correspondencia para cada pixel de la imagen, y el método basado en rasgos que determina correspondencia solo sobre determinados rasgos.

En un sistema de visión en estéreo calibrado se conocen con precisión los parámetros extrínsecos e intrínsecos y es posible llevar a cabo la reconstrucción tridimensional por triangulación. Cuando solo se tienen las imágenes adquiridas desde diferentes puntos de vista se pueden calcular los parámetros del sistema si se conocen un número definido de puntos correspondientes entre las imágenes, a este proceso se le conoce como calibración.

El modelo matemático generalizado del sistema de visión en estéreo con dos cámaras tipo *pinhole* se explica en [9] para un sistema cuyos planos de imagen son perfectamente coplanares y a la línea que une sus centros de proyección se le llama *línea base*.

En la Figura 2.7 [7] se tiene que *P* es el punto de la escena del mundo real. p_l es el punto de proyección en la cámara posicionada del lado izquierdo y p_r el punto de proyección en la cámara del lado derecho. O_l y O_r son los centros de proyección izquierdo y derecho respectivamente sobre la línea base, c_i y c_r son los centros del plano de imagen, *T* es la distancia entre ambos centros de proyección o línea base, *f* es la longitud focal común, y *Z* es la distancia de *P* a la línea base.

Se forma geométricamente un triángulo (O_l, *P*, *Or*,) y otro inscrito (p_l, *P*, p_r); entonces se genera una relación con las variables de ambos triángulos para obtener la distancia Z que será la profundidad obtenida del punto en 3D hacia el espacio 2D o viceversa, como se indica en la ecuación 2.2 donde *d* es la disparidad formada por la diferencia de x_l y x_r.

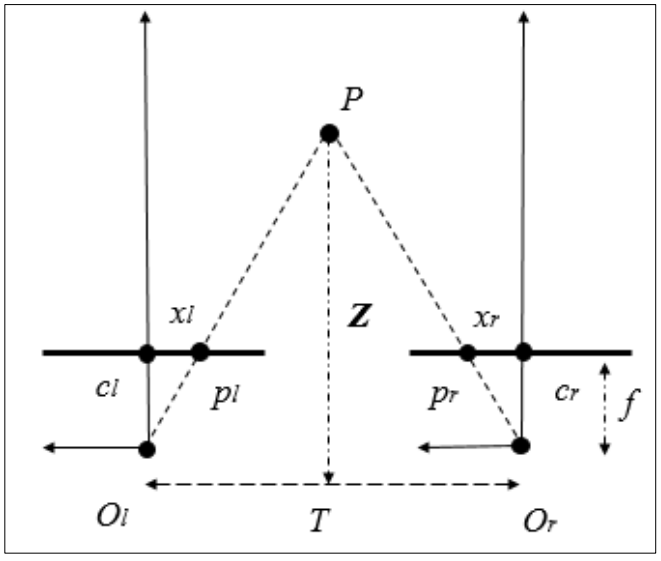


Figura 2.7. Obtención del parámetro de profundidad en una triangulación estéreo.

$$\frac{T + X_l - X_r}{Z - f} = \frac{T}{Z}$$

$$Z = f\frac{T}{d} \tag{2.2}$$

Por lo tanto, se deduce que la disparidad *d* es inversamente proporcional a la profundidad *Z*.

2.2 Sensores RGBD

Los sensores RGBD son dispositivos alternativos a los sistemas de escaneo de cuerpo completo que mejoran el proceso de reconstrucción tridimensional. Las ventajas que representa usar un sensor RGBD son significativas sobre un sistema tradicional de visión en estéreo, algunas de estas ventajas son:

- Se requiere un solo sensor para llevar a cabo el proceso de reconstrucción, a diferencia del sistema en estéreo que necesita dos cámaras.
- El hardware de un sensor RGBD está diseñado especialmente para propósitos de reconstrucción tridimensional a diferencia del sistema en estéreo cuyas cámaras, aun cuando sean idénticas, tendrán diferencias en hardware. Entonces, el sensor RGBD puede ofrecer reconstrucciones más precisas.
- El tiempo computacional de obtención de datos es más rápido debido a que algunos sensores RGBD traen integrado software de captura; a diferencia del sistema en estéreo cuyo proceso de reconstrucción de datos tridimensionales es lento y consta de más etapas, lo cual propicia existan errores de reconstrucción.
- El montaje del área de captura es menos complejo ya que el sistema de visión en estéreo requiere toda una estructura que permita que las cámaras se sitúen paralelamente a una distancia fija de separación entre ambas durante la adquisición.

Considerando lo anterior, fue que se optó por elegir el sensor Kinect de Microsoft, siendo un sensor accesible en precio y cuya plataforma en software permite manipular la adquisición de imágenes para los propósitos de esta investigación.

El sensor Kinect ofrece toda una plataforma de desarrollo de software que lo vuelve práctico para su manipulación en cuestiones relacionadas a sistemas de visión y óptica.

Además, su costo accesible, lo posiciona como la primera opción para la construcción de un sistema de captura tridimensional por encima de otros.

En este momento, existen dos versiones de sensores Kinect que comparten la misma filosofía de captura de imágenes a través del uso de una cámara a color con formato RGB y otra infrarroja. Esta combinación permite que el programador pueda detectar profundidad y las funciones en sus librerías permiten segmentar la silueta cuerpo humano entre otros objetos.

Las diferencias sustanciales entre la versión 2 de Kinect sobre la versión 1, incluyen mejoras de hardware para la adquisición de imágenes con mayor resolución, un arreglo de micrófonos con aislamiento para ruido y un mayor número de nodos o articulaciones en la detección de esqueletos. Con referencia al software, el manejo de registros para control y procesamiento de imágenes y datos de las cámaras permanece semejante a la versión 1 cambiando solamente el manejo de clases y métodos para la configuración y arranque de las cámaras. En el uso del sensor Kinect Versión 2 es necesario que la computadora de control cumpla las características de hardware y software especificados por el fabricante. La Figura 2.8 muestra la apariencia de ambos sensores.

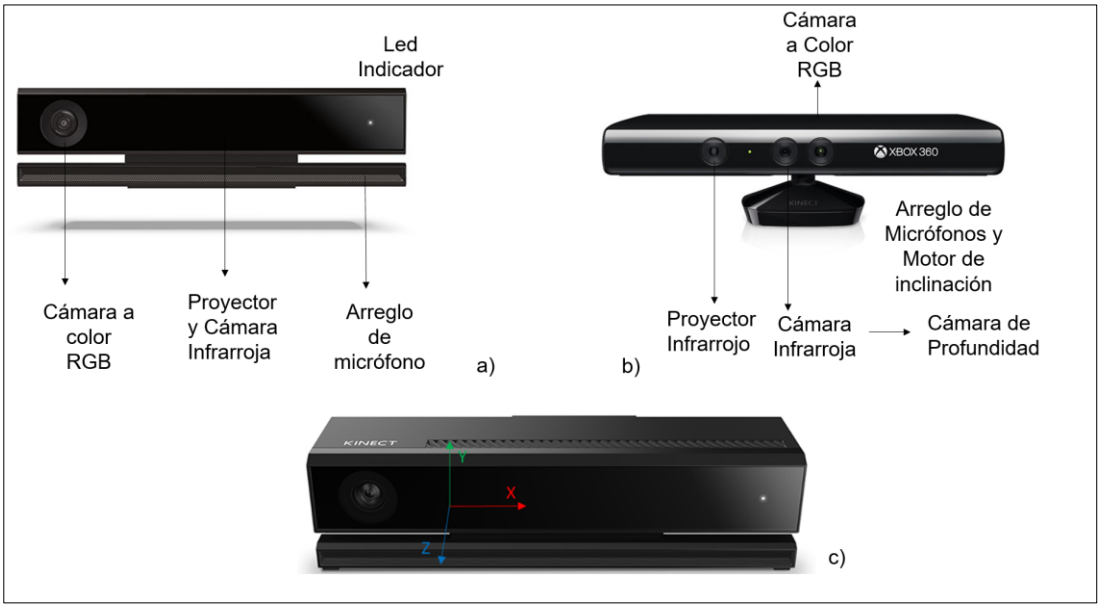


Figura 2.8. Sensor RGBD Kinect a) versión 2 y b) versión 1.

La compañía Microsoft se ha encargado del manejo de procesamiento y obtención de datos de captura para que el diseñador pueda manipular, a través de funciones, las propiedades

mencionadas en el párrafo anterior, haciendo uso del kit de desarrollo de software (SDK) del sensor Kinect para Visual Studio.

El conocimiento básico de C# y WPF en Visual Studio 2010 o posterior es indispensable para el manejo de la librería de funciones con el sensor Kinect Versión 1 o Versión 2. Estas funciones para el manejo de los sensores requieren la interfaz de programación de aplicaciones (API) del sensor Kinect, la cual contiene los manejadores del dispositivo. Ambos paquetes, SDK y API, se pueden descargar de forma gratuita desde el sitio web de Microsoft. La Tabla 2.1 muestra la comparación de las especificaciones de los sensores Kinect V1 y V2 [10].

Tabla 2.1. Comparación de las características del sensor Kinect versión 1 y versión 2.

Especificación del Sensor	**Descripción**	**Sensor Kinect V1**	**Sensor Kinect V2**
Cámara Infrarroja/Sensor de Profundidad	Resolución	320 x 240 px	512 x 424px
	Campo de Visión (hxv)	57º x 43º	70.6º x 60º
	Rango de Operación	0.4- 4.5 m	0.5-4.5 m
	Técnica de Obtención de datos	Luz Estructurada	Tiempo de vuelo (TOF)
Cámara a Color	Resolución	640 x 480 px @ 30 fps	1920 x 1080 px @30 fps
	Campo de visión (hxv)	57º x 43º	84.1º x 53.8º
Latencia Mínima		102 ms	20 ms
Conectividad	Puerto USB	2.0	3.0
Transmisión de Audio		16 KHz, 16 bit	48 kHz,16 bit

La metodología de configuración en los sensores Kinect para el acceso a las cámaras RGB e infrarroja vía software es la siguiente:

1. Activación del sensor Kinect

Es un código para indicar cuantos sensores se encuentran conectados a la PC y detectarlo para su posterior configuración. Para el caso de la versión 2, solo es posible la conexión de un sensor por equipo. El sensor conectado representa una instancia de la clase *KinectSensor*.

2. Fuentes (*Sources*)

Los sensores exponen una fuente por cada tipo de flujo de datos (*data stream*) y las fuentes consisten en indicar con cual componente del sensor se trabajará para tener acceso a los lectores (*readers*) de configuración de datos. Los flujos de datos que proporciona el sensor pueden ser el flujo de datos a color (*color stream*), el flujo de datos de profundidad (*depth stream*), el flujo de datos infrarrojos (*infrared stream*), el flujo de datos de cuerpos detectados

por el sensor (*body stream*), el flujo de datos del índice de cuerpos detectados (*body index stream*) y el flujo de datos de audio (*audio stream*).

3. Lectores (*Readers*)

Son componentes informáticos que permiten el acceso a los cuadros de imágenes ya sea mediante manejador de eventos (*Event Handler*) o sondeo computacional (*Polling*) utilizado comúnmente en otros sistemas operativos distintos de Windows [11]. Es posible configurar múltiples lectores para una sola fuente seleccionada, es decir, cada componente al que se desee acceder a sus datos, abre su propio lector y este se puede hacer la función independientemente de cuántas fuentes se estén abriendo en el mismo instante. Además, los lectores pueden ser pausados para obtener datos sin necesidad de volver a arrancar el sistema de adquisición de imágenes. Cada vez que se adquiere un cuadro de imagen se invocan los lectores correspondientes al flujo de datos específico.

4. Referencias a los cuadros de imagen (*Frame References*)

Posterior al llamado de los lectores, se adquiere una cantidad específica de cuadros por segundo (*fps – frames per second*) a través de la función miembro *AquireFrame ()*. La referencia a los cuadros de imagen fue diseñada con el propósito de minimizar el contenido de memoria y evitar retardos en la visualización de video haciendo más eficiente las características del sensor. Es posible tener acceso a los metadatos de dichos cuadros como, por ejemplo, el formato de color, el ancho y el largo de la imagen, etc. Para el manejo de las propiedades en los cuadros de imagen, se recomienda crear copias locales o acceder al buffer contenedor de ella directamente para evitar mantener el cuadro en procesamiento durante mucho tiempo.

2.2.1 Cámara a color del sensor Kinect

El sensor Kinect soporta los formatos de color RGB, YUV y Bayer creando una imagen tipo Bitmap. El formato RGB es el más utilizado, genera la imagen a color a partir de tres componentes de colores; rojo, verde y azul (*Red, Green, Blue*). Cada pixel RGB en la imagen se compone de un arreglo de cuatro bytes, ordenados como lo muestra la Figura 2.9 [11], añadiendo al final de cada trama un byte llamado Alfa, el cual da la transparencia del pixel. La

resolución de la imagen adquirida para la versión 2 es de 1920x1080 en alta resolución, pudiendo seleccionar un rango de 15 o 30 cuadros por segundo.

El segundo formato YUV toma en cuenta la luminancia (Y) para la construcción de la imagen y solo dos componentes a color, una azul (U) y otra roja (V). El formato de imagen se construye de igual manera que el RGB, teniendo el formato YUV la ventaja de que, al poseer un ancho de banda reducido para los componentes de color, es posible identificar en la secuencia de imágenes errores en la construcción de estas.

El formato Bayer hace uso de la combinación de los componentes rojo, verde y azul en una proporción de 50% para el verde, 25% para el rojo y 25% para el azul. A este patrón se le conoce como Filtro de Bayer. Este filtro tiene su fundamento en estudios acerca de la sensibilidad que tiene el ojo humano para percibir mayormente el verde y sus cambios de tonalidades [11].

Blue	Green	Red	Alpha	Blue	Green	Red	Alpha
Blue	Green	Red	Alpha	Blue	Green	Red	Alpha
Blue	Green	Red	Alpha	Blue	Green	Red	Alpha
Blue	Green	Red	Alpha	Blue	Green	Red	Alpha

Figura 2.9. Formato RGB para dos pixeles en una imagen adquirida con el sensor Kinect.

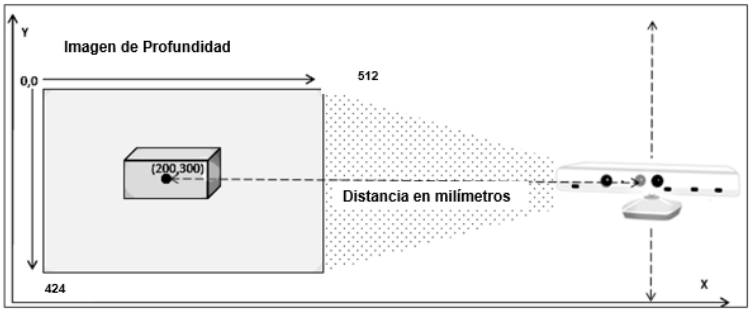
2.2.2 Cámara de profundidad y detección del cuerpo humano

La detección de la profundidad de objetos, se realiza por la integración de un emisor y una cámara infrarroja aplicando la técnica de tiempo de vuelo. La profundidad es concebida como una sucesión de imágenes de profundidad, regresando el arreglo con 16 bits en escala de grises. El pixel de profundidad en la imagen contiene la distancia entre el Kinect y los objetos enfrente de él, en milímetros. Los datos son representados basados en las coordenadas X, Y tomando como referencia el centro del sensor, como se muestra en la Figura 2.10 a). El rango de profundidad detectado por el sensor varía desde 0.5 m hasta 8 metros y la distancia se aloja en un arreglo de 16 bit, de los cuales los tres menos significativos corresponden al manejo de detección del cuerpo humano, como se muestra en la Figura 2.10 b). Entonces si se desea

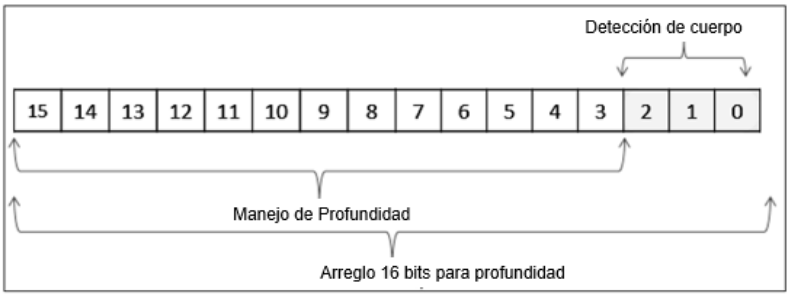
hacer uso solamente de la cámara de profundidad es necesario recorrer el registro tres bits a partir del menos significativo [11].

Con respecto a la detección del cuerpo que realiza el sensor, las características de profundidad varían de acuerdo a las características mencionadas en el párrafo anterior, por ejemplo, el rango de profundidad para la cual puede ser detectado el cuerpo va desde un rango de 0.5 metros hasta 4.5 metros. Sin embargo, los diseñadores del programa de detección del cuerpo se han encargado de coleccionar y marcar los 25 principales nodos que componen al esqueleto humano, obteniendo la posición y orientación de cada articulación del esqueleto en el espacio tridimensional. La Figura 2.10 a) esquematiza la forma en que son detectadas las profundidades de cada punto proyectado por el sensor para procesar los datos a través de un registro de 16 bits mostrado en la Figura 2.10 b) [11].

Para la configuración por software primero debe habilitarse el manejador de eventos para la cámara de profundidad, una vez activada se configura otro manejador de eventos para la detección del cuerpo humano, el cual consiste en un filtrado donde los pixeles que corresponden a la silueta humana, tendrán un valor codificado como 255 (color blanco) y los pixeles que no corresponden a dicha silueta serán asignados como un 0 (color negro).



a)



b)

Figura 2.10. Cámara de profundidad. a) Representación gráfica y b) manejo del arreglo contenedor de profundidad.

2.3 Modelado Tridimensional

En Visión por Computadora, la representación de objetos debe ser ajustada para el uso de reconocimiento, es decir, que debe existir una correspondencia potencial entre la representación y las características que son extraídas de la imagen. Muchos de los algoritmos de objetos en 3D no son suficientes para manejar una amplia variedad de características.

Los algoritmos básicos de representación de objetos en 3D se concentran en propiedades geométricas en términos de puntos, líneas y superficies. Los modelos representativos sintetizados en [7] son:

1. Modelado con mallas. Representación geométrica que describe un objeto en una serie de vértices y bordes que forman polígonos en el espacio. Una malla puede tener un número arbitrario de polígonos y puede ser una malla regular, aquella que tiene dentro de su malla polígonos del mismo tipo, o bien la malla irregular que posee polígonos de diferentes tipos. Las mallas pueden representar un objeto en diferentes niveles de resolución.

2. Modelado de vértices, bordes y superficies. Conocido en el idioma inglés como *Wire-Frame Model* y se concentra en detectar bordes, vértices y superficies del objeto en cuestión asumiendo que la superficie del objeto es plana y que el objeto solo tiene bordes rectos. Una generalización muy usada del modelo *Wire-Frame* es la representación superficie-borde-vértice, donde la representación es una estructura de datos que contienen vértices del objeto, superficies y segmentos de bordes. Cuando el objeto es poligonal, las superficies son planas y los bordes son segmentos lineales. Sin embargo, el modelo generalizado incluye bordes y superficies curvas.

3. Modelado generalizado cilíndrico. Es un volumen definido por un eje, es decir, cada volumen de características cilíndricas se encuentra rodeando un eje. Este modelo se concentra en analizar dos regiones de la imagen con geometría cilíndrica: el listón (*ribbon*) que es la proyección de la longitud del cilindro y la elipse que es la sección transversal de la geometría y que no siempre es circular. Este modelo geométrico es constantemente utilizado en la representación de personas separándolas en partes como la cabeza, brazos, torso y piernas que tienen una geometría similar a la de un cilindro.

4. Modelo de *Octrees* y Supercuadráticos. Un *octree* es una estructura de árbol comprendida en 8 aros o niveles. Cada nodo en el árbol corresponde a una región cúbica en el universo, entonces el objeto puede ser representado por un arreglo tridimensional de 2N x 2N x 2N siendo N un número entero. Los elementos del arreglo son llamados *voxels* y tienen un valor de 1 (lleno) o 0 (vacío), cuando encuentran la presencia o ausencia del objeto. Los modelos supercuadráticos se encargan de parametrizar una familia de formas a través de un modelo matricial definido por un vector S cuyos valores *x, y* y *z* son componentes especificados por funciones de los ángulos η y ω (ecuación 2.3). El modelado por supercuadráticos es utilizado comúnmente para la construcción de esferas, elipsoides, cilindros y paralelepípedos.

$$S(\eta,\omega)=\begin{bmatrix}x\\y\\z\end{bmatrix}=\begin{bmatrix}\alpha_1\cos^{\varepsilon 1}(\eta)\cos^{\varepsilon 2}(\omega)\\ \alpha_2\cos^{\varepsilon 1}(\eta)\,sen^{\varepsilon 2}(\omega)\\ \alpha 3\,sen^{\varepsilon 1}(\eta)\end{bmatrix} \tag{2.3}$$

5. Modelado físico. Usado para modelar la apariencia y el comportamiento físico del objeto en una imagen. Un término fuertemente relacionado al modelado físico es el término Modelo de Deformación Basado en la Física que analiza y modela el cambio de la forma física del objeto a consecuencia de deformaciones. Aunque es una rama nueva y avanzada en el modelado 3D, ya se han desarrollado trabajos significativos en el modelado del corazón humano al tener deformaciones por el trabajo de la sístole y la diástole [7].

La finalidad del desarrollo de diferentes geometrías para la reconstrucción tridimensional de un objeto o persona, va encaminado a apegarse a lo que se desea representar, evitando zonas de oclusión que puedan presentarse durante el proceso de escaneo. La Figura 2.11 [7] muestra las geometrías para la reconstrucción tridimensional a diferentes objetos.

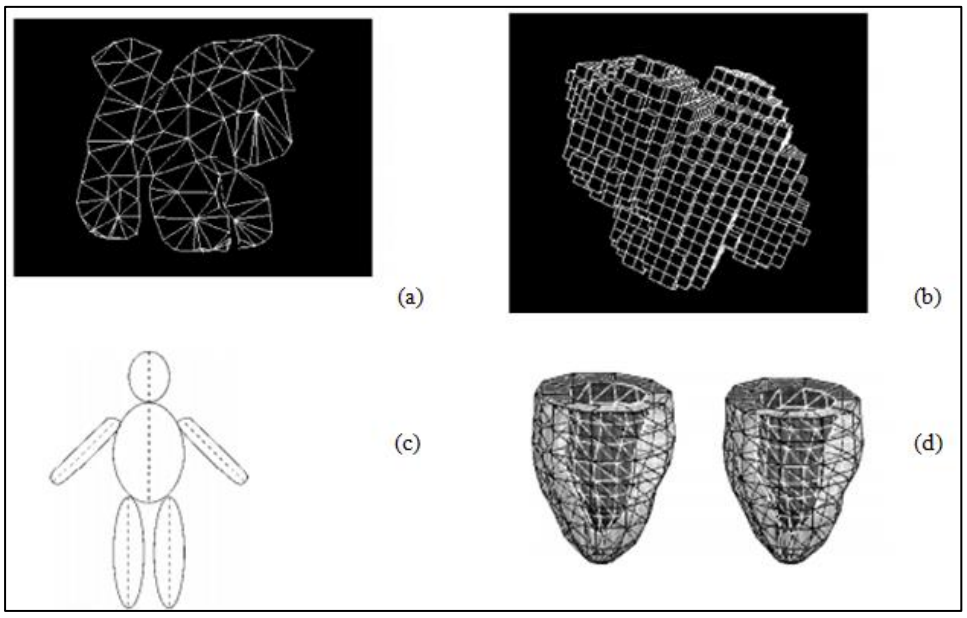


Figura 2.11. Ejemplos de los tipos de modelado en 3D. a) Modelado de un perro por mallas, b) modelado de un perro mediante la detección de superficie, bordes y vértices, c) modelado cilíndrico del cuerpo humano y d) modelado por supercuadrático del ventrículo izquierdo.

2.4 Impresión Tridimensional por Material Fundido

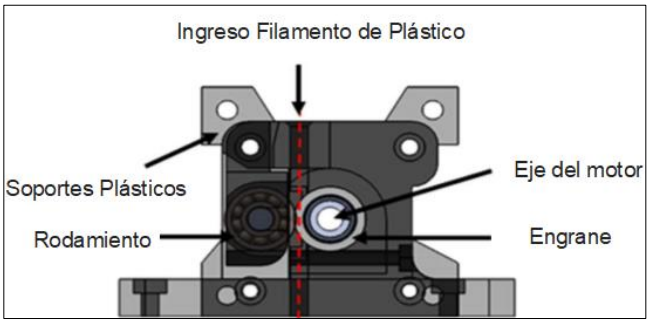
La impresión por deposición de material fundido FDM, consiste en desenrollar un filamento de plástico de una bobina y abastecer el material hacia una boquilla de extrusión, la boquilla se alimenta con el filamento que es calentado a una temperatura por debajo de fusión del material [12].

Los materiales con que se encuentra construido el filamento son el Acrilonitrilo Butadieno Estireno (ABS) y el Ácido Poliláctico (PLA). El ABS se caracteriza por ser un polímero compuesto de carbono, nitrógeno e hidrógeno lo cual le da la propiedad de alta tenacidad y dureza, el punto de fusión de este material para el proceso de extrusión es de 230º a 240º C, el PLA es un poliéster termoplástico alifático que procede de recursos renovables como pueden ser los restos de maíz, las raíces de tapioca, trozos de madera o de caña de azúcar, su punto de fusión para el proceso de extrusión es de 180º a 220º C [13] [14].

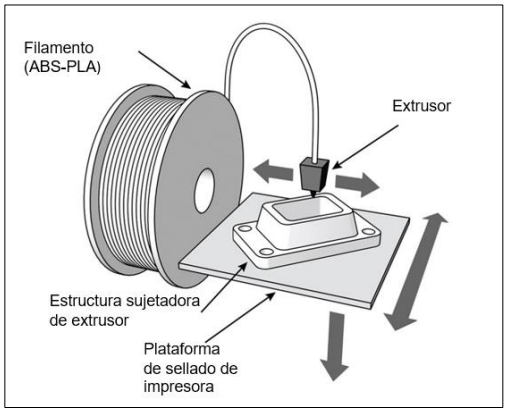
El funcionamiento de la máquina de impresión 3D tiene su principio en el robot cartesiano que puede moverse en tres direcciones lineales X, Y y Z usando motores a pasos para cada desplazamiento de los ejes. Las partes que componen cualquier impresora elemental consisten

en un mecanismo extrusor, la estructura de movimiento a lo largo de los ejes y el sistema de control.

El mecanismo extrusor mostrado en la Figura 2.12 a) [12] es el encargado de fundir el polímero para depositarlo capa sobre capa hasta lograr la impresión, se construye por un fundidor (calentador y sensor de temperatura) y un reductor (motor y sistema de transmisión). La forma de calentar el filamento es por medio de una resistencia de potencia y el sensor de temperatura será un termistor. La Figura 2.12 b) [15] muestra el montaje del filamento hacia la impresora 3D.



a)



b)

Figura 2.12. a) Composición interna de un extrusor de impresora 3D y b) montaje del extrusor a la impresora 3D.

La estructura de impresión mostrada en la Figura 2.13 [12], se construye con cuatro columnas en las cuales se encuentran apoyados los ejes X-Y. El mecanismo extrusor se mueve a lo largo y ancho de dichos ejes. El proceso de impresión comenzará al ajustar primeramente la base de deposición sobre el eje Z [12].

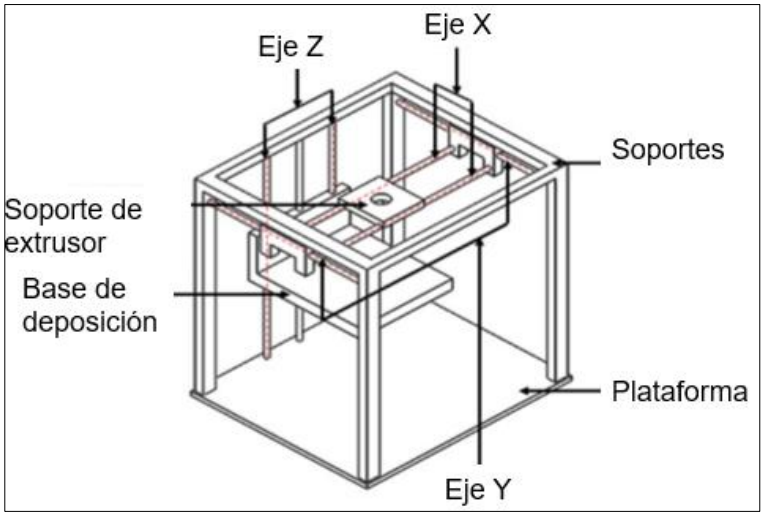


Figura 2.13. Estructura de una impresora 3D.

El sistema de control consiste de un sistema embebido capaz de traducir el formato de impresión característico a Código-G propios de un sistema de manufactura CNC y otra tarjeta de impresión embebida se encarga de controlar los aspectos mecánicos como la potencia de los servomotores, la temperatura del extrusor y los parámetros generales de los demás actuadores.

Los parámetros que definen la calidad de la pieza final, tienen que ver con el relleno de la estructura de soporte de la pieza, el número de pasadas que tendrá el acabado y la resolución del grosor del polímero fundido.

La Tabla 2.2 [16] muestra una comparativa entre los diferentes fabricantes de impresoras 3D, las características principales al momento de seleccionar alguna van desde la resolución, dimensiones físicas, la velocidad de impresión y el costo. La impresora 3D más usada para fines de prototipo y desarrollo básico de manufactura son las de la línea Makerbot, las cuales poseen software de entrenamiento y manipulación del objeto tridimensional, es de fácil acceso a su manual del usuario y actualmente se encuentra posicionada dentro del mercado [16].

Tabla 2.2. Comparativa de características de diferentes modelos de impresoras 3D.

Modelo	Dimensiones	Resolución del Extrusor	Velocidad Impresión	Material	Costo dólares
RepRap Mendel	200 × 200 × 110	0.1mm	150mm/s	3mm PLA	$830
RepRap Huxley	140 × 140 × 110	0.1mm	150mm/s	1.75mm PLA	$600
MakerBot Replicator	225 × 145 × 150	0.2mm	45mm/s	1.75mm ABS	$1,750
MakerGear Mosaic	127 × 127 × 127	0.15mm	75mm/s	1.75mm PLA	$900
Ultimaker	210 × 210 × 220	0.04mm	300mm/s	3mm PLA	$1,570
whiteAnt CNC	160 × 190 × 125	0.25mm	35mm/s	3mm ABS	NA
Aleph Objects AO-100	200 × 190 × 100	0.1mm	200mm/s	3mm PLA	$1,500
Printrbot	150 × 150 × 150	0.3mm	25mm/s	3mm ABS	$550

2.4.1 Formatos de archivo para modelado tridimensional

Existen alrededor de 140 formatos de archivo para la representación tridimensional de un objeto y cada uno se distingue por la geometría empleada para la reconstrucción de superficies [17].

Los archivos de impresión 3D se dividen en tres categorías de acuerdo al énfasis que hacen al reconstruir tridimensionalmente y son: en geometría de superficies, apariencia de texturas y colores; y en escena de la cámara.

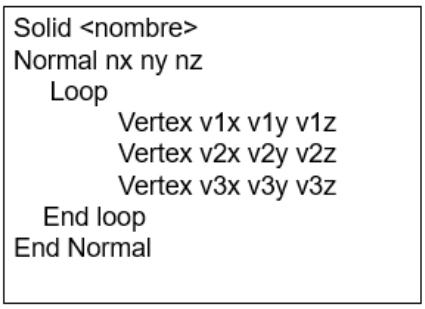
Los formatos comúnmente tratados por escáneres y sistemas de impresión 3D son el formato de esterolitografia (STL), el formato poligonal (PLY) y el tipo objeto (OBJ), ya que la reconstrucción es rápida y eficiente y sus unidades pueden ser manejadas por un entorno de programación como MATLAB o Visual Studio. A continuación, se resumen las características más importantes de estos tres formatos. (Figura 2.14).

1. El Formato STL describe la superficie mallada sin ninguna representación de color, textura y color. La superficie que genera se representa por un conjunto de triángulos, donde cada triángulo se define computacionalmente por 12 números de punto flotante. El formato STL fundamenta su estructura en la teselación (*Tesselation*) en donde la superficie tridimensional tendrá un patrón regular o no en su composición, lo que favorece a que no existan espacios o *gaps* en la pieza lo cual otros formatos como el PLY si permite. El archivo con formato STL provee dos diferentes formas de guardar la información de cada triángulo, ya sea en codificación ASCII o codificación binaria. La información que se guarda son las

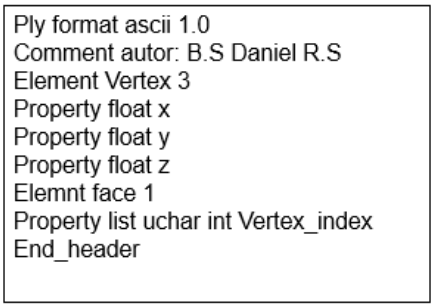
coordenadas de los vértices (*x*, *y*, *z*) y las componentes (*nx*, *ny*, *nz*) del vector normal de cada triángulo.

2. El Formato PLY fue creado para propósitos de impresión 3D, representa una nube de puntos por medio de un listado de vértices formados por las coordenadas *xyz* cuya unión formará un polígono y el plano formado entre ellos se denomina como cara. Este tipo de formato de archivo se presenta codificado de dos maneras; en ASCII si se desea contener toda la información del objeto o en binario si se desea comprimirla. La Figura 2.14 b) muestra un ejemplo del contenido del formato siendo las dos primeras líneas la cabecera, pasando después por la descripción de los vértices y finalmente por las caras. El archivo puede o no contener información de color RGB utilizando una columna por color.

3. En el Formato OBJ el mallado es sencillo y compacto para el mapeo de texturas tridimensionales. Consiste de un número de líneas que contienen una clave y varios valores. La clave en cada línea indica el tipo de información a seguir, no requiere de encabezado.



a)



b)

Clave (Key)	Descripción
#	Comentario
v	Vértice
l	Línea
fn	Cara
vt	Textura
vn	Normal
g	Grupo

c)

Figura 2.14. Formatos de archivo a) .STL, b) .PLY y c) .OBJ.

2.5 Estudio del Estado del Arte

2.5.1 Sistemas de escaneo del cuerpo humano

Los sistemas de escaneo del cuerpo humano, son dispositivos ópticos de medición tridimensional que producen una copia de la superficie tridimensional del cuerpo humano y son utilizados en aplicaciones como ergonomía, sistemas militares, medicina, industria de la manufactura, antropometría. La complejidad de estos sistemas hace que no estén disponibles para el público en general teniendo como principal desventaja el precio, además el uso de esta clase de sistemas requiere conocimientos técnicos de las áreas de visión por computadora, procesamiento de imágenes, óptica y mecánica.

La operación de un escáner 3D consiste de una o más fuentes luminosas que proyectan una línea u otro patrón sobre el cuerpo humano, las cámaras que capturan la imagen proyectada por la fuente de luz sobre el cuerpo, el software que extrae la profundidad de la superficie de las imágenes y la computadora para visualización y procesamiento de los datos extraídos [18].

Los escáneres pueden ser clasificados de acuerdo a la tecnología óptica que utilizan para el proceso y pueden dividirse en:

1. De Línea Láser. La mayoría de los escáneres del mercado proyectan una sola línea láser sobre el cuerpo. Los láseres y cámaras están montadas en un marco movible vertical llamado cabezal de escaneo. La Figura 2.15 [18] detalla el principio de esta técnica genérica usada en la reconstrucción de la piel. Básicamente, la línea se proyecta en el estómago del cuerpo. Las dos cámaras son usadas para evitar las sombras, la línea de proyección horizontal es visible en la cámara como una línea curva que representará la forma del estómago. La línea curva del sujeto se encuentra más hacía el fondo de la imagen para la cámara superior. Cuanto más cerca este un punto proyectado de la cámara, menor será su posición vertical en esta imagen. El sistema de captura escanea hasta cuatro veces con el objetivo de tener una reconstrucción óptima, claro está que la utilización de múltiples cámaras reduce efectos de sombreado y zonas de oclusión [18] [19].

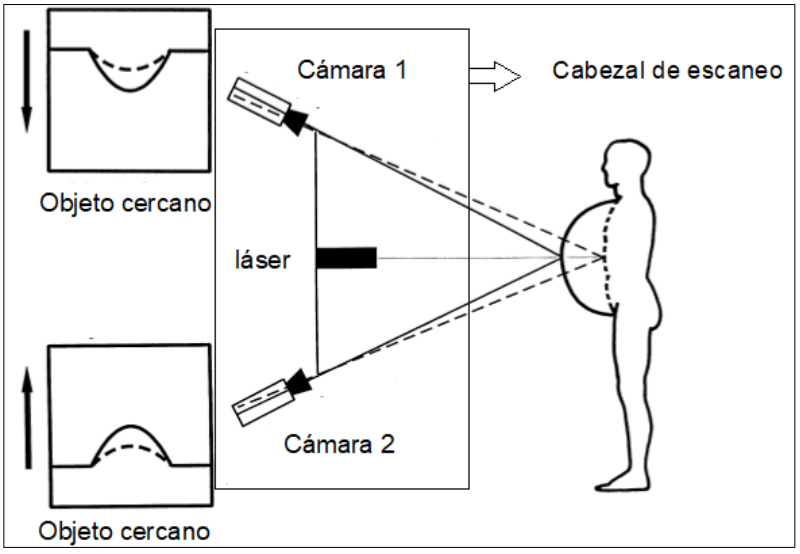


Figura 2.15. Triangulación óptica para determinar la localización de la piel.

Algunos modelos representativos son la línea Cyberware y Vitronic. Cyberware WB4 cuyo sistema se ejemplifica en la Figura 2.16 [18] a) es un modelo robusto de alta resolución y consta de cuatro cabezales con escáneres, dos motores generan el movimiento vertical de los cabezales. El escáner tiene una resolución en sus líneas de proyección horizontal y vertical de 5 y 2 milímetros respectivamente. La resolución de la línea horizontal está en función del sensor CCD mientras que la vertical por la velocidad de la cámara (50 HZ) y la velocidad de los cabezales que contienen los escáneres. El modelo Vitronic Smart de la Figura 2.16 b) captura medidas con un nivel de precisión de 1 milímetro.

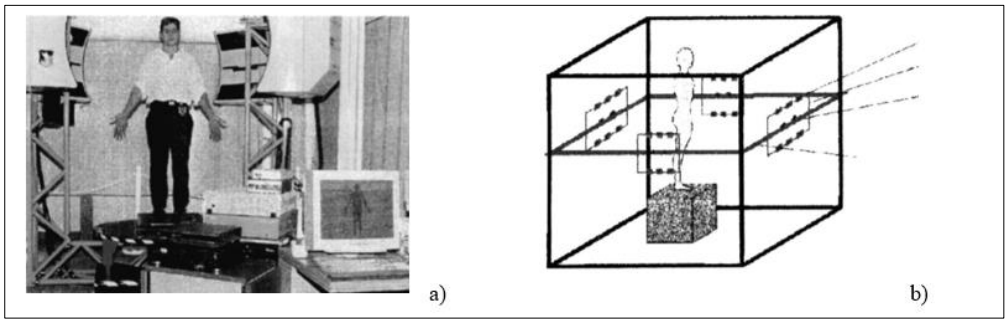


Figura 2.16. Escáneres de proyección láser. a) Cyberware y b) Vitronic.

2. De Luz Estructurada. A diferencia de los de línea láser, estos sistemas generan un patrón de luz (generalmente láser) en forma de puntos, barras, o cualquier otro patrón que el sensor de imagen pueda detectar. La ventaja principal es el tiempo de procesamiento en la reconstrucción al poder procesar hasta 30 imágenes por segundo (fps) aunque por otro lado la principal desventaja es que los patrones de luz son difíciles de mapear al espacio tridimensional, lo que requiere un mayor análisis de datos y por ende lo vuelve costoso. El sistema Hamamatsu es el más representativo de su clase y se muestra en la Figura 2.17 [20]. Funciona con un detector infrarrojo ultra sensitivo a la posición, los sensores principales constan de un arreglo de 32 diodos láser. El mecanismo de escaneo se mueve 5 mm verticalmente durante un ciclo, el cual para un escaneo total de aproximadamente 2000 mm arroja un total de 102,400 puntos muestra. Cada procesamiento de cuerpo completo ocupa una capacidad en memoria de 300kB que es guardado en un arreglo matricial de 256 x 460. Finalmente, los datos agrupados y procesados se presentan comúnmente en un eje espacial de tres dimensiones, donde el eje primario (Y) denota la altura del individuo, el eje secundario (Z) corresponde al ancho y el último eje terciario (X) a la profundidad [20].

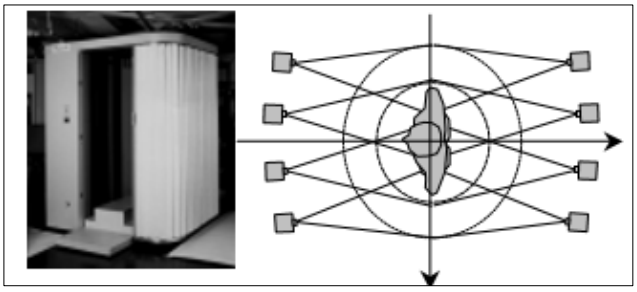


Figura 2.17. Escáner modelo Hamamatsu.

3. Sistema Multicámara. La técnica de visión en estéreo muestra ser una forma más barata de reconstrucción tridimensional, su desventaja es que no posee la exactitud que los métodos mencionados anteriormente. A los nuevos sistemas que vienen integrados con dos cámaras para la reconstrucción se les conocen como sistemas multicámara.

Este sistema tradicional consta de varias etapas para lograr una reconstrucción aceptable en términos de calidad. Los pasos inician desde el montaje de ambas cámaras, la adquisición de

una amplia secuencia de imágenes (40 a 60 imágenes) para calibrar el sistema en estéreo, el cálculo de los parámetros intrínsecos y extrínsecos cuya matriz Q definirá la distancia de profundidad y la disparidad asociada para ambas cámaras. Finalmente, para cada pixel de la imagen, se proyecta hacia el espacio tridimensional y se obtienen las coordenadas en el plano de referencia de la cámara.

Las dificultades encontradas en este tipo de sistemas es el hecho de procesar constantemente los datos obtenidos para poder definir la silueta de la persona, además de que la representación contiene zonas de oclusión.

4. Sensores RGBD. Los sensores RGBD en un inicio fueron diseñados para fines de entretenimiento en la industria de los videojuegos; el crear aplicaciones donde se pudiera detectar el movimiento del cuerpo resultó ser una iniciativa innovadora.

La Tabla 2.3 muestra una comparativa de algunos parámetros de los diferentes tipos de sistemas de esceneo de personas comentadas anteriormente [18].

Tabla 2.3. Comparativa de los diferentes sistemas de escaneo de personas.

Característica	Cyberware WB4	Hamatsu	Sensor RGBD
Principio de escaneo	Láser	Luz estructurada basada en Led	Cámara de color e infrarroja basada en tiempo de vuelo
Precio en dólares	410,000	100, 000	200
Tiempo de escaneo (segundos)	17	7	1-2
Cantidad de sensores o proyectores	4	8	Cámara a color y cámara infrarroja de profundidad
Hardware requerido	SG Índigo 2	PC	PC
Software requerido	CyScan, CyPie	Visual Basic/Visual Studio+ aplicación	Microsoft Kinect SDK versión 2.0

El impacto del surgimiento de estas tecnologías como el caso del Kinect, propició que se investigara por separado cada uno de ellos para desarrollar aplicaciones en código abierto. Microsoft decidió lanzar una plataforma de herramientas de desarrollo (*Kit Development Tool*) para utilizar el sensor como un sistema de adquisición de imágenes y modelado tridimensional.

Actualmente se generalizó la operación de cada uno de estos sistemas, cuyas características más importantes se describen en a detalle en el siguiente capítulo.

2.5.2 Técnicas de reconstrucción y superficializado tridimensional del cuerpo completo

En el proceso de construcción del modelo 3D del cuerpo humano existen inconsistencias presentadas en parte por el somatotipo y son visualizadas en la imagen como agujeros o espacio vacíos (sin datos). La Universidad de Washington [21], presenta una función ponderada de tres medidas, para corrección y reconstrucción de agujeros o espacios vacíos que son:

a) La proximidad de los vértices transformados al rango de datos.
b) Similitud entre las transformaciones de los vecinos.
c) Proximidad de los marcadores escasos que corresponden en las posiciones del modelo con la superficie analizada.

El método creado de relleno de espacios, está basado en el modelo Blanz y Vetter [22], que comienza con una muestra de 250 escaneos de diferentes tipos de cuerpos, para conseguir una función que corrija el defecto por la proximidad de los vértices transformados al rango de datos.

La obtención de los datos muestra, es a partir del proyecto CAESAR, [21] este se encarga de recolectar miles de rangos volumétricos de personas entre 16 y 65 años de edad en Estados Unidos y Europa. El proyecto consiste en definir puntos antropométricos de reconocimiento, los cuales en su mayoría son asignados en zonas donde los huesos pueden observarse o palparse a lo largo de la piel. Cada reconstrucción contiene entre 250,000 y 350,000 triángulos por vértice.

La técnica de modelado tridimensional del cuerpo humano tiene como marco de referencia, el trabajo de Marschener [23] , el cual consiste en regularizar su proceso de ajuste utilizando un término de suavizado de superficie D, pero con la diferencia de que en lugar de utilizar una superficie de suavizado, la optimización planteada minimiza la variación por sí sola, sin necesidad de construir la superficie de suavizado, entonces los orificios en la malla serán llenados en detalle desde la superficie del modelo.

Además, como referencia, también es utilizado el método de Feldmar y Ayache [24], quienes hacen coincidir puntos normales a la superficie o curvas mientras se mantiene una afinidad similar en la transformación dentro de regiones esféricas del espacio. En este caso, en vez de que el término de suavizado sea dentro del volumen esférico, será calculado directamente sobre la superficie. El algoritmo de reconstrucción encuentra una serie de transformaciones T_i que cuando se apliquen los vértices en el modelo *T*, dé como resultado una nueva superficie o superficie transformada *T´* que coincida con el punto específico de la superficie *D*, como se muestra en la Figura 2.18 [21]. Una vez elaborada la plantilla (*Template*), los datos faltantes podrán alojarse en la superficie *D* como finalización del proceso. Para que se pueda hacer coincidir *T´* con la superficie *D*, se requerirá la reducción de tres términos de error:

1. Error de datos. Se define como la suma ponderada del cuadrado de las distancias, entre la superficie del modelo transformado *D*. La minimización de este error busca que la superficie transformada $\boldsymbol{T'}$ como la superficie ideal *D* coincida a través del parámetro de control W_i.

$$E_d = \sum_{i=1}^{n} w_i \, dist^2 \, (\boldsymbol{T}_i \boldsymbol{v}_i \, D) \tag{2.4}$$

2. Error de suavizado. Compensa las diferencias entre las transformaciones adyacentes $\boldsymbol{T_i}$, es decir, ajusta cada vértice de la superficie transformada $\boldsymbol{T'}$, a su punto más cercano en la superficie ideal *D*, para evitar que existan puntos dispersos.

$$E_s = \sum_{\{i,j|\{V_i,V_j\} \in edges\,(T)\}} \left|\left|\boldsymbol{T}_i - \boldsymbol{T}_j\right|\right|^2 F \tag{2.5}$$

3. Error de marcas. Compensa la distancia entre los puntos marcados en la superficie marcada y la superficie D. Si la superficie de transformación y la superficie ideal no son cercanas, la optimización queda atrapada en mínimos locales. Por ejemplo, si el brazo izquierdo comienza a alinearse con el brazo derecho, es poco probable que el algoritmo de gradiente descendiente respalde la alineación. Entonces, para evitar los mínimos indeseables, se identifican una serie de puntos que se relacionan con la serie de puntos de la superficie tomada del modelo y se hace la relación, aquí es donde el Error de marcas (Em) minimiza la distancia entre cada posición de marcadores.

$$E_m = \sum_{i=1}^{m} \left|\left|\boldsymbol{T}_{ki}\boldsymbol{v}_{\mathbf{ki}} - \boldsymbol{m}_{\mathbf{i}}\right|\right|^2 \tag{2.6}$$

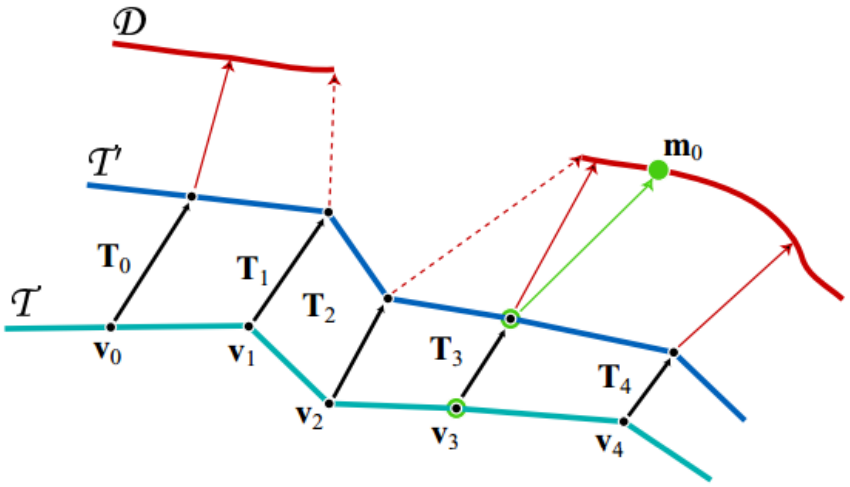


Figura 2.18. Método para minimización de errores en el modelado 3D de personas.

La función completa de optimización para el modelo del somatotipo, se completa con la suma de las tres funciones de error descritas:

$$E = \alpha E_d + \beta E_s + \gamma E_m \tag{2.7}$$

Donde α, β y γ son coeficientes de sintonización para guiar y controlar la optimización.

Un algoritmo comúnmente utilizado para reconstrucción tridimensional es la reconstrucción de superficie de Poisson [25], la cuál considera todos los puntos en una sola iteración, sin considerar la segmentación o partición de puntos, el cual tiene como principal ventaja la estabilidad de los datos considerados como ruido.

El algoritmo de Poisson consiste fundamentalmente de tres etapas para la creación de una superficie: 1. Cálculo de vectores de orientación de campo *V*, 2. Cálculo del gradiente de una función indicadora χ, la cual discretiza la superficie marcando como 1 los puntos adentro de ella y 0 los que están fuera, 3. Creación de la superficie extrayendo una superficie apropiada. La Figura 2.19 ilustra cada una de las etapas mencionadas [25].

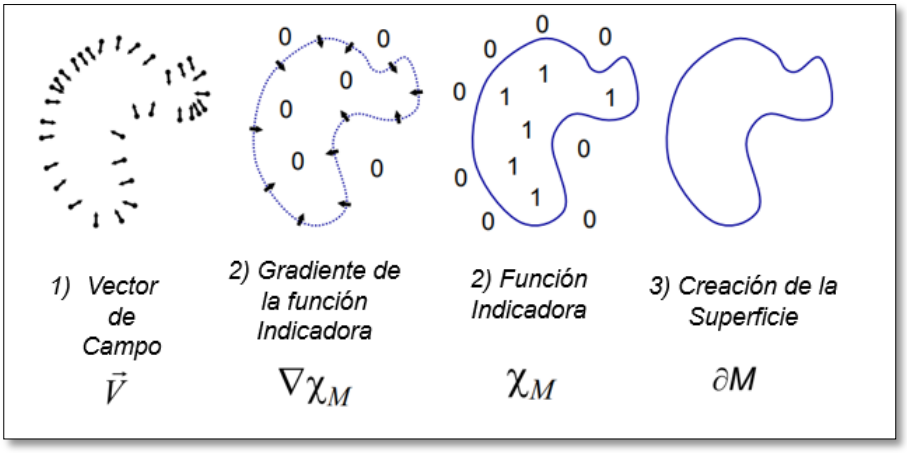


Figura 2.19. Principio de creación de superficies de Poisson.

Otro algoritmo utilizado en la creación de mallas tridimensionales es por Pivoteo de Pelota [26] (*Ball Pivoting*). El principio de este algoritmo contempla una superficie M de un objeto tridimensional y S es una muestra de M. Sobre cada muestra de S tiene que pasar una pelota o circunferencia de radio ρ que deberá empezar por hacer contacto con tres puntos de la superficie S, siendo esta la posición de pivoteo. Se pivotea alrededor de cada borde hasta que la superficie encuentre otro borde y el paso de la circunferencia a lo largo de tres puntos va marcando la superficie tridimensional reconstruida. El valor del radio ρ es fundamental para que la reconstrucción sea cubierta es su totalidad. La Figura 2.20 muestra una reconstrucción tridimensional realizada con este algoritmo usando diferentes tipos de radio ρ, en donde se aprecia que en la figura 2.20 b) el radio no alcanza a completar la superficie debido a su morfología [26].

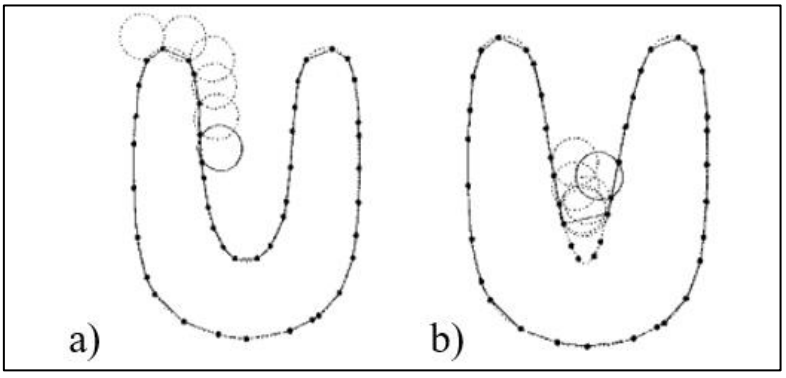


Figura 2.20. Algoritmo de Pivoteo de Pelota de radio ρ. a) Superficie cubierta totalmente y b) circunferencia no cubierta en su totalidad.

2.5.3 Sistema de escaneo con sensores Kinect y bases giratorias

Mientras se ejecuta el proceso de escaneo, la persona se coloca en una plataforma giratoria que rota a una velocidad de 360 grados en 30 segundos. La imagen a color de salida o adquirida del sistema es de resolución 1280x1024 y la imagen de profundidad de es 640 x 480 a 15 cuadros por segundo. Los ejes coordenados son automáticamente generados usando la biblioteca OpenNI [27]. La Figura 2.21 muestra el sistema de captura usado en [27] para reconstruir el cuerpo.

La representación tridimensional de un cuerpo a través de tres sensores Kinect se efectuará por medio de un algoritmo de la siguiente manera:

1. Adquisición de la imagen de cuerpo completo.
2. Registro y procesamiento de superficies no rígidas del cuerpo.
3. Alineación global de las superficies del cuerpo, cerrar superficies y ajuste de oclusiones.

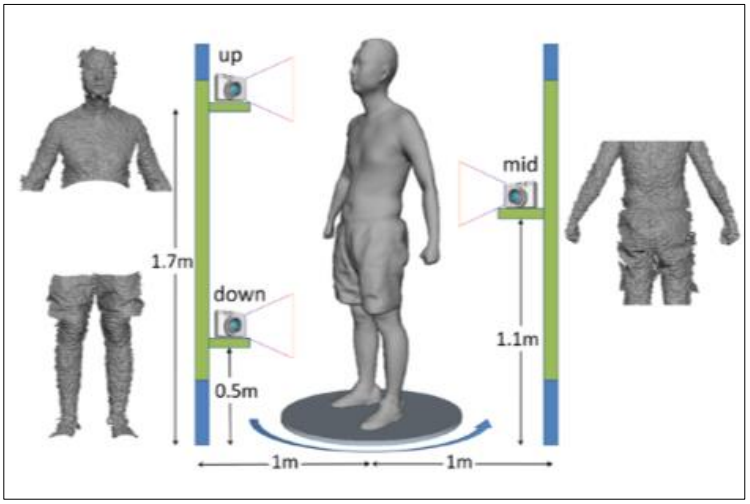


Figura 2.21. Sistema de escaneo de J. Zhou.

2.5.4 Sistema de escaneo con 2 sensores Kinect para análisis de movimiento

En la tesis desarrollada por V.H. Velasco [6], se presenta un sistema de captura sintonizada con dos sensores Kinect. Los sensores son colocados a ambos lados de la persona. La Figura 2.22 muestra la distribución de los sensores dentro del área de captura [6]. Se

construye el modelo tridimensional de la persona fusionando los datos adquiridos por ambos sensores.

Se buscó en este sistema, encontrar la distancia donde los dos sensores Kinect, tendrían un mejor campo de visión y existiera la mayor cantidad de puntos adquiridos para realizar una reconstrucción lo más aproximada a lo real. Cada sensor Kinect versión 2 utilizado tuvo su propio *Equipo de Captura*, y se buscó también que las propiedades de ambos equipos fueran iguales, en cuanto a memoria RAM y las versiones de Visual Studio y Matlab.

La forma en que se enlazan y transmiten datos los sistemas de captura, es mediante el protocolo TCP/IP contando con una computadora adicional cliente, mientras que los sistemas pertenecientes a cada uno de los sensores son los servidores. Una vez que es enviada la señal de captura por parte del cliente, como primera etapa, se sincronizan los *Equipos de Captura* a la hora marcada en el cliente, esto con la finalidad de que no se obtengan secuencias de imágenes desfasadas y cada par de imagen sea tomada al mismo instante. Después de haber logrado la sincronización de los equipos, se ejecuta el comando de capturar e iniciar la secuencia de toma de imágenes. Los 100 pares de archivos de texto que contienen de las coordenadas de posición del cuerpo humano teniendo como origen el eje coordenado del sensor, se guardan en una carpeta compartida ubicada en la raíz del disco duro de cada uno de los equipos, las cuales podrán ser trasladados al equipo donde se vaya a dar tratamiento de fusión de datos, lo que permitirá la reconstrucción del cuerpo sin haber cerrado los puntos coordenados.

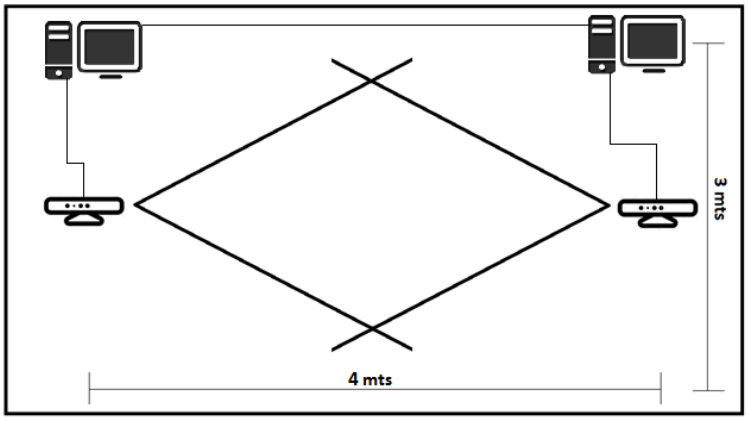
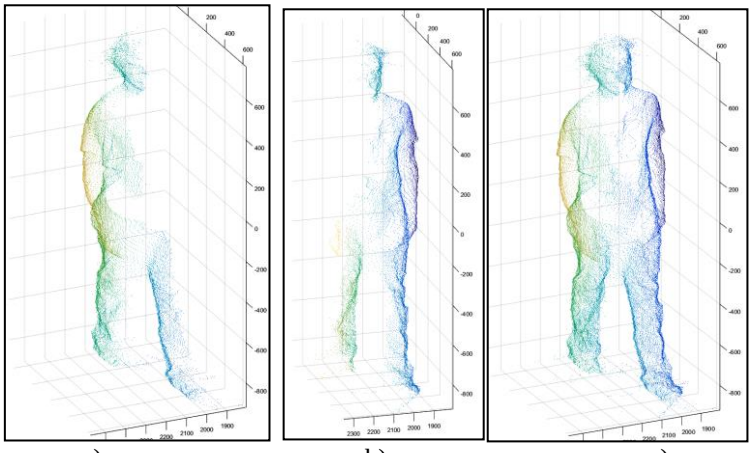


Figura 2.22. Área de captura del sistema de escaneo de V.H. Velasco.

El sistema fue construido con la intención de realizar el análisis de movimiento de personas, construyendo un modelo matemático de la forma de andar con fines de reconocimiento. Por esta razón, el sistema de captura sincronizada adquiere secuencias de 100 cuadros de imagen que contendrán la información de profundidad para cada sensor, para posteriormente fusionarlos y así obtener una nube de puntos de datos tridimensionales de las personas desde el punto de referencia de cada sensor Kinect, generando un modelo tridimensional de 360 grados, aunque con algunas oclusiones observables mostradas en la Figura 2.23 [6].

Es posible la adaptación del sistema de comunicación para cuatro sensores Kinect versión 2, en donde cada uno de ellos contiene su respectiva computadora, así como la modificación del código fuente de captura para hacer que las nubes de puntos contengan información a color que mejoren la apreciación del modelo; a pesar de que para la impresión tridimensional este parámetro no se considera relevante, para el modelo si se considerar de esta manera pues se busca la reconstrucción de un cuerpo humano con sus características en color, textura y tamaño.



a) b) c)

Figura 2.23. Datos tridimensionales del sistema con dos sensores RGBD. a) Sensor 1. b) Sensor 2 y c) Datos fusionados.

III. SISTEMA DE CAPTURA SINCRONIZADA DE DATOS TRIDIMENSIONALES CON CUATRO SENSORES RGBD

3.1 Características de los Equipos de Cómputo

Los equipos de cómputo utilizados en el sistema de escaneo son de dos tipos: El primero tipo son denominados Equipos de Captura, estos están conectados a cada sensor RGBD. El segundo tipo es denominado Equipo de Control, este es el encargado de enviar la señal de control para la sincronización en la adquisición de datos a cada Equipo de Captura. A continuación, se presenta las características de cada uno de ellos.

3.1.1 Equipos de Captura

Se usaron cuatro computadoras Intel NUC, cada una de ellas conectada a un sensor Kinect. A estas computadoras se les llama *Equipos de Captura*. Fue indispensable que los Equipos de Captura tuvieran las mismas características computacionales de hardware y software (sistema operativo y aplicaciones), para evitar problemas de compatibilidad en el desarrollo de los programas y facilitar el proceso de sincronización de inicio de captura. Las características de los Equipos de Captura se enlistan en la Tabla 3.1 y la Figura 3.1 muestra la imagen de un equipo Intel NUC.

Tabla 3.1. Características de los Equipos de Captura.

Equipos Intel NUC D54250WYK (Cuatro equipos de captura)
• Procesador Procesador Intel Core I5-5ta Generación de 64 Bits
• Intel Graphics 6000
• Puerto Intel HD Audio
• 4 Puertos USB 3.0
• Intel Dual Band Wireless-AC y Bluetooth 4.0
• Puerto RJ45 con controlador Intel Gigabit Ethernet
• Puerto mini Display Port versión 1.2
• 16 GB de memoria RAM Dual_Chanel DDR3L SODIMMs, 1.35 V
• Disco de estado sólido ADATA con capacidad de 128 GB
• Teclado Inalámbrico y touchpad de la línea Logitech
• Puerto mini HDMI 1.4ª
• Visual Studio Professional 2015 y Matlab 2017a

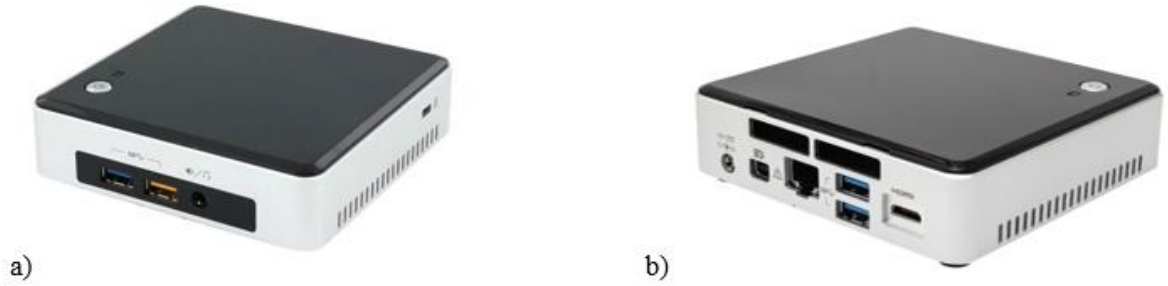


Figura 3.1. Computadora Intel NUC modelo D54250WYK a) vista de frontal y b) vista posterior.

Se homogeneizaron los entornos de desarrollo (IDE) instalados en los Equipos de Captura. Anteriormente, en el experimento presentado en [6], los programas participes en la captura y fusión de datos tridimensionales eran procesados en las versiones de Matlab 2015a y Visual Studio 2013 Express para Windows 8. Dichas aplicaciones fueron actualizadas a versiones de Matlab 2017a y Visual Studio 2015 Professional para Windows 10, esto con la finalidad de mantener actualizado el software y evitar que los códigos pierdan vigencia.

La Figura 3.2 muestra la conexión de los cuatro sensores a un solo monitor utilizando un multiplexor HDMI de cuatro entradas a una salida del monitor.

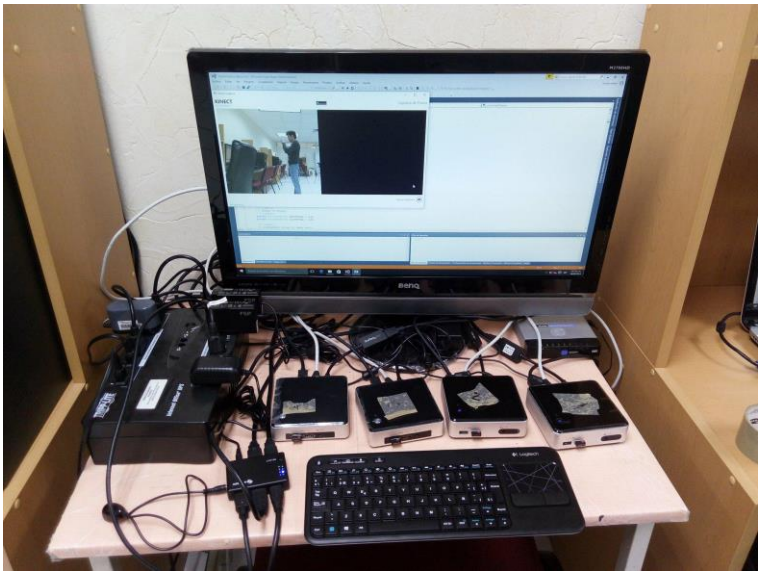


Figura 3.2. Conexión física de los cuatro Equipos de Captura.

3.1.2 Equipo de Control

Un equipo aparte, llamado *Equipo de Control*, es el encargado de mandar la señal de disparo a los cuatro Equipos de Captura para inicializar la adquisición de datos tridimensionales de manera sincronizada. Cuando se establece conexión entre el Equipo de Control y los Equipos de Captura, el Equipo de Control manda la señal de inicio de captura y obtiene los archivos de las nubes de puntos de cada uno de los cuatro Equipos de Captura al terminar la adquisición. De manera que el Equipo de Control va agregando las capturas de diferentes sujetos a una base de datos para su posterior procesamiento. Las características del Equipo de Control, se encuentran enlistadas en la Tabla 3.2 donde se destaca que el software de captura posee las mismas versiones de entornos de desarrollo (IDE) Visual Studio 2015 y Matlab 2017[a] y la Figura 3.3 muestra la imagen física del Equipo de Control.

Tabla 3.2. Características de Equipo de Control.

Equipo Hp Envy m4
• Procesador Intel Core i5-3210M a 2.5 GHz de 64 bits • Puerto Intel HD-Audio • Intel Wirelss Bluetooth • Intel Centrino Wireless N-2230 • Dos puertos USB 3.0 • Un puerto USB 2.0 • 6 GB de memoria RAM • 463 de memoria de disco duro utilizable

Figura 3.3. El Equipo de Control es una computadora HP Envy m4.

3.2 Diseño del Área de Captura

El diseño del área de captura consistió fundamentalmente en definir una zona de escaneo apropiado para generar una nube de puntos con mayor cantidad de información, además se deberá de aislar a la persona del resto de los objetos, evitando se lleguen a presentar puntos no deseados dentro de la reconstrucción tridimensional.

Se evaluaron dos diseños del área de captura con el objetivo de determinar con cuál se obtiene una mejor reconstrucción tridimensional, reduciendo oclusiones producidas cuando las extremidades bloquean la visión de otras partes del cuerpo humano. Las áreas son denominadas: 1) área con geometría rectangular y 2) área con geometría tipo cruz. Para ambas se obtendrán sus respectivos resultados estableciendo sus ventajas y desventajas.

El diseño del área de captura se encuentra directamente relacionado al campo de visión (FOV) de la cámara de profundidad del Kinect V2 cuya especificación es de 60 grados en la vertical y 70 grados en la horizontal. Además del campo de visión se debe considerar el rango de operación del sensor de profundidad para diseñar el área de captura. Para este proyecto se consideró que la profundidad óptima para la captura de un cuerpo humano usando el sensor Kinect V2 es de 2 metros. [6] [28]

3.2.1 Área de captura con geometría rectangular

La motivación de implementar el área de captura tipo rectángulo, surge a partir de ampliar el alcance al sistema implementado en [6], donde la ventaja de esta área de captura se concentró en no realizar rotaciones adicionales de los sistemas coordenados de cada uno de los sensores en el programa de fusión de datos, lo que a diferencia del área de captura con geometría tipo cruz si deberá contenerlas.

En la Figura 3.4 se muestra la distribución para el área con geometría tipo rectángulo. Se colocaron los cuatro sensores {*S1, S2, S3, S4*} de tal manera que formaron un área rectangular. En el centro del rectángulo se situó el origen {*So*}, lugar donde se ubicó a la persona para escanearla. La misma figura muestra el diseño del área de captura, en donde *d1*, *d2* y *d3* son las distancias que se usaron en el vector de traslación al momento de fusionar los datos. *d1* representa la altura al centro del sensor, es decir, la distancia desde el suelo hasta el centro del sensor, que representa una distancia negativa sobre el eje *y* del sensor. La variable *d2*

representa la distancia entre los centros de dos sensores alineados con respecto a su eje *x*; el sensor *S1* y el sensor *S2* se encuentran alineados y quedan de frente el sensor *S3* y el sensor *S4* pero con el eje *x* invertido. *d3* corresponde a la distancia del centro de cada sensor hacia enfrente sobre el eje *z* hasta el punto de referencia *So*. La distancia d3 ha sido preestablecida en dos metros, ya que es la distancia apropiada para la recolección de puntos tridimensionales de una persona con una altura máxima de 2.1 metros, sin salir del cuadro de imagen definido por el campo de visión de los sensores. [6]

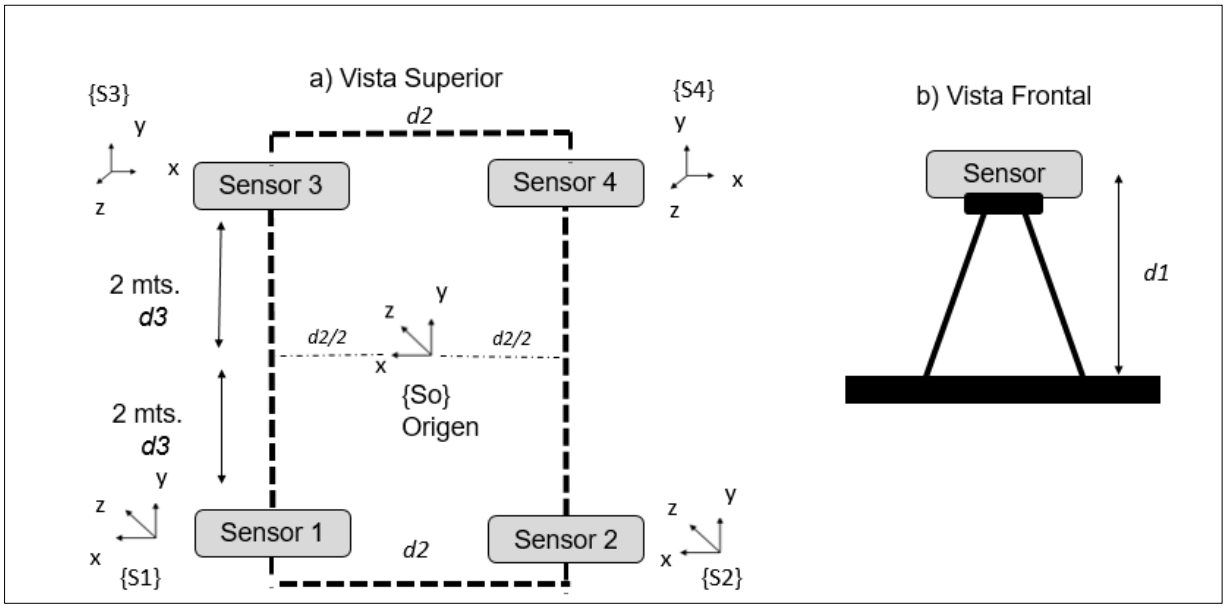


Figura 3.4. Distribución del área de captura con geometría tipo rectángulo.

En la Figura 3.5, se muestra la implementación física del área de captura en el Laboratorio de Sistemas Inteligentes y Visión por Computadora del Instituto Tecnológico de Chihuahua, donde la persona a ser escaneada se sitúa sobre la marca que señala el punto de referencia *So* de forma que quede ubicada en el centro de los cuatro sensores por un instante en lo que se obtienen los datos de color y profundidad. Esto representa una ventaja del sistema de escaneo tridimensional construido, ya que es muy corto el tiempo necesario para la captura de datos.

Figura 3.5. Implementación física del área de captura con geometría tipo rectángulo.

Se utilizó un nivel de gota para alinear los cuatro sensores con respecto a los ejes *x* y *y*. Se utilizó un flexómetro para fijar la distancia *d1*=960mm desde el suelo hasta el centro de cada sensor, que debe ser la misma para los cuatro sensores. También se fijó la distancia *d2*=1000mm entre el sensor *S1* y *S2* y también entre el sensor *S3* y *S4*.

Se ubicaron los sensores a una distancia *d3*=2000mm hacia el punto de referencia *So* sobre el eje *z*. Esta distancia debió ser calibrada de forma precisa usando la aplicación *CalibracionPlano.cs* y la estructura de calibración usada en [4] pero ahora con los cuatro sensores. Este proceso de calibración permite alinear el plano del eje *x* de los sensores con respecto al del eje coordenado de referencia con el origen en el punto *So*. Las distancias *d1*=960mm, *d2*=1000mm y *d3*= 2000 mm serán utilizados en los vectores de traslación del algoritmo de fusión de datos para esta área de captura.

El centro del sensor *S1* respecto al sensor *S3* y del sensor *S2* respecto al *S4* y debe estar alineados sobre su eje z. La Figura 3.6 muestra el uso de un nivel láser de tres ejes para asegurar que los centros de cada par de sensores (*S1*-*S3* y *S2*-*S4*) estén alineados.

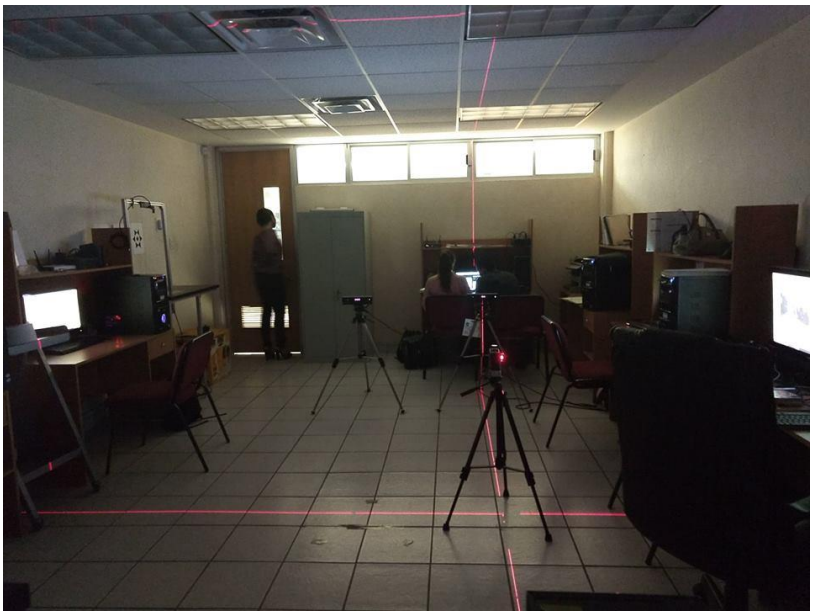


Figura 3.6. Nivel láser utilizado para alinear S1 con S3 y S2 con S4 sobre el eje Z.

Se situó a una persona en el punto de referencia *So*, el frente es capturado por los sensores *S1* y *S2*, la parte posterior la capturan los sensores *S3* y *S4* como se muestra en la Figura 3.7. Es importante corroborar que la persona se encuentre presente en los cuadros de imagen de cada sensor, de lo contrario no se recabarán los datos.

Figura 3.7. Escaneo de una persona en el área de captura tipo rectángulo.

3.2.2 Área de captura con geometría tipo cruz

Se propuso una segunda área de captura con la finalidad de compararla con la primera implementada denominada área de captura con geometría tipo cruz. El esquema de esta área mostrado en la Figura 3.8 consiste en que los cuatro sensores forman una cruz a lo largo del eje *z* de cada uno de ellos. Para este caso el sensor *S1* queda de frente al sensor *S2* y el sensor *S3* de frente al sensor *S4*. La distancia *d2* corresponde a la distancia del punto de referencia *So* hacia cada uno de los centros de los sensores, la cual es de dos metros y *d1* representa la distancia desde el suelo hasta el centro de cada sensor sobre el eje *y*.

El proceso de alineación de los sensores utiliza nivel de gota y nivel de láser para la alineación de los ejes *x* y *y* de cada sensor. Para determinar la distancia *d2* se construyó una estructura adicional que permite fijar la distancia al mismo tiempo para los dos pares de sensores (*S1*-*S2* y *S3*-*S4*). Con base en lo anterior se fijaron las distancias *d1*=960mm y *d2*=2000mm. Estos valores son usados en el vector de traslación del algoritmo de fusión de datos para esta área de captura.

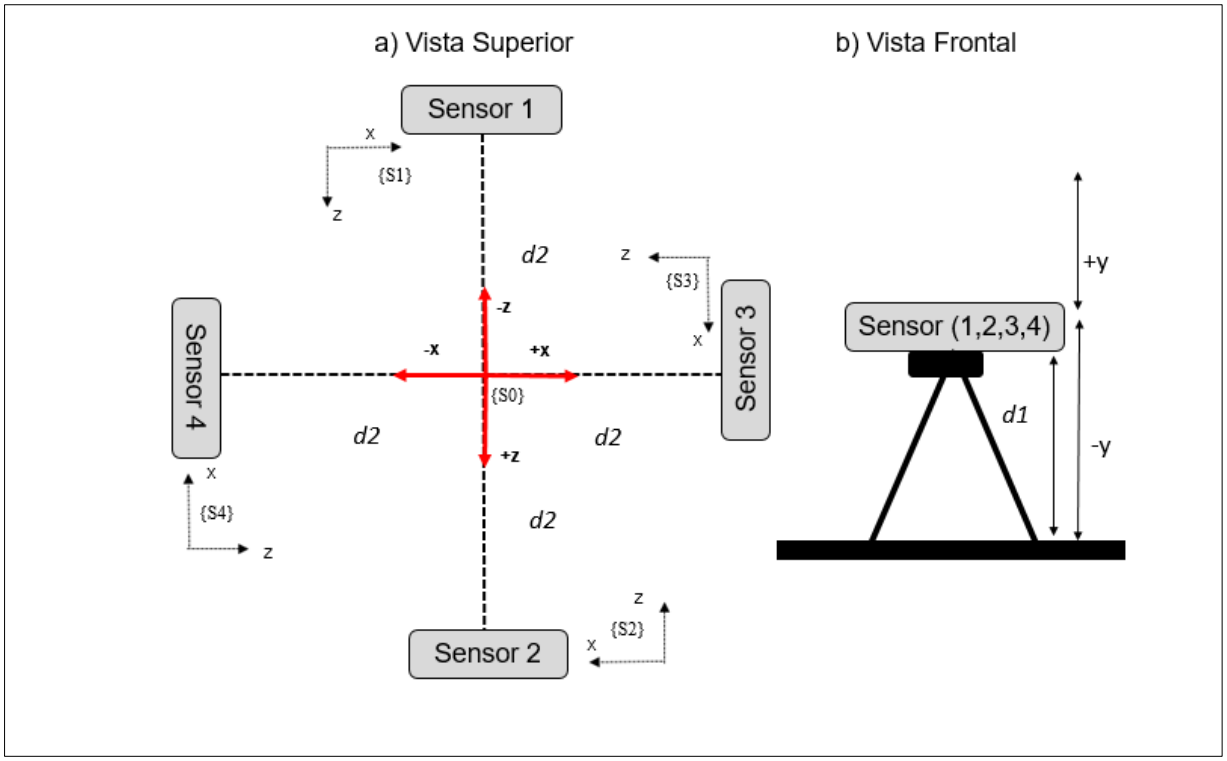


Figura 3.8. Distribución del área de captura con geometría tipo cruz.

La Figura 3.9 a) muestra la implementación y alineación de los centros de cada par de sensores en el Laboratorio de Sistemas Inteligentes y Visión por Computadora (LSIVC) usando el nivel de láser, mientras que la Figura 3.9 b) muestra la estructura que permite la calibración de la distancia *d2* del punto de referencia *So* al centro de cada sensor mediante la segmentación de los círculos azul y verde. Nótese que el láser forma la cruz que corresponde al nombre del área de captura.

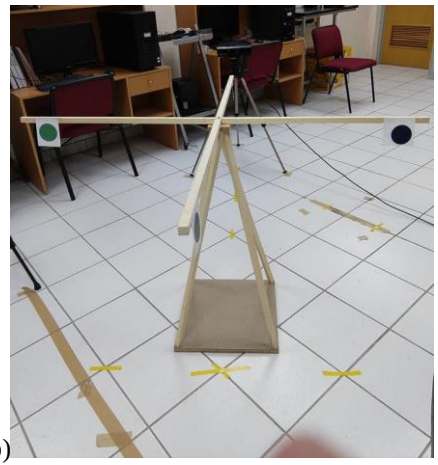
a)  b) 

Figura 3.9. a) Implementación del área de captura con geometría tipo cruz alineando los centros de los cuatro sensores con el nivel láser y b) estructura de calibración para la distancia d2.

3.3 Software de Captura de Datos Tridimensionales

Son tres aplicaciones que intervienen durante el proceso de captura de datos tridimensionales. Se desarrollaron en Lenguaje C# sobre la plataforma de desarrollo Visual Studio utilizando el kit de desarrollo de software (SDK) del sensor Kinect de Microsoft y tienen como principal antecedente lo hecho en quien obtiene las nubes de puntos de cada sensor sin la información de las componentes RGB [6]. La primera es la aplicación de calibración de sensores llamada *CalibracionPlano* cuyo código muestra en el Anexo 1. La segunda es una aplicación que se ejecuta en los Equipos de Captura llamada *KinectColorCapture*, cuyo código se muestra en el Anexo 2, encargada de capturar los datos tridimensionales con color y la tercera es una aplicación que se ejecuta en el Equipo de Control llamada *KinectSocketClient*, cuyo código se muestra en el Anexo 3, encargada de enviar la señal de disparo a los Equipos de Captura para la adquisición de datos. A

continuación se analiza cada uno de ellos en su funcionamiento y que acción ejerce un programa sobre el otro.

3.3.1 Aplicación de calibración

Antes de comenzar la adquisición de datos se debió llevar a cabo la calibración del Sistema de Captura Sincronizada de Datos Tridimensionales con Cuatro Sensores RGBD. Se lleva a cabo el proceso de calibración para alinear los planos de los ejes coordenados de los cuatro sensores con el fin de facilitar el proceso de fusión de datos. El proceso de calibración se define en función del tipo de área de captura a implementar, ya sea el área tipo rectángulo o el área tipo cruz.

La aplicación *CalibracionPlano* opera bajo el mismo proceso de calibración de planos implementado en [6] , donde se colocaron los sensores a una distancia de 2 metros del punto de referencia *So* sobre el eje *z* del marco de referencia de cada uno de los sensores.

La Figura 3.10 muestra la interfaz de usuario de la aplicación para el área de captura tipo rectángulo, donde se tiene un cuadro de imagen a color en la parte izquierda y otro cuadro de imagen a escala de grises en la parte derecha cuyos pixeles contienen información de profundidad. En la parte inferior aparece la distancia sobre el eje *z* hacia los círculos verde (G) y azul (B) montados sobre la estructura de calibración.

La aplicación de calibración consiste en segmentar dos círculos, uno de color azul y otro de color verde, montados sobre una barra en un trípode. Los círculos son visibles desde ambos lados de la estructura de calibración. El trípode es ubicado en el centro del área de captura sobre el punto de referencia *So* y la estructura de madera con los círculos es alienado sobre el eje *x*. Una vez segmentados los círculos verde y azul se toma la distancia sobre el eje z hacia un punto detectado sobre cada círculo. Esta distancia es mostrada en la interfaz de la aplicación de calibración. Los cuatro sensores ejecutan la aplicación y su ubicación es ajustada de manera que la distancia hacia los puntos de ambos círculos sea de 2000 milímetros. De esta forma se considera que el sistema se encuentra calibrado ya que los cuatro sensores quedan alineados con respecto al plano donde se ubican los círculos.

Es recomendable que para cada inicio de captura se corrobore que los equipos se encuentran calibrados, ya que cualquier movimiento de uno de ellos ocasionará un

desfasamiento en la fusión de datos, aunque esto puede corregirse vía software en el algoritmo de fusión de datos se prefiere evitarlo debido a que las correcciones vía software son tardadas.

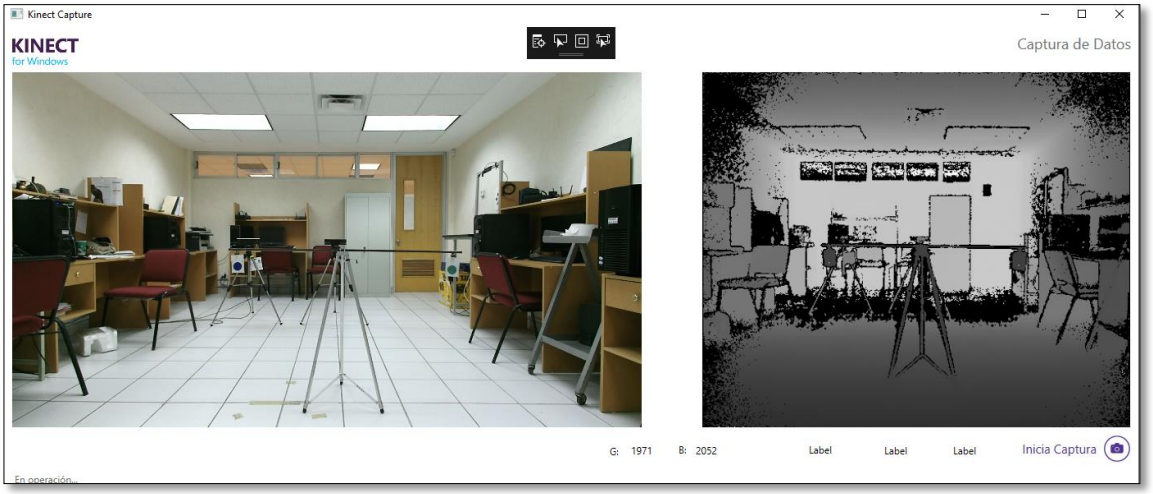


Figura 3.10. Interfaz de la aplicación de calibración de sensores con el código *CalibracionPlano*.

3.3.2 Aplicación del Equipo de Captura

La aplicación *KinectColorCapture* es ejecutada en cada Equipo de Captura, siendo la encargada de generar la nube de puntos tridimensionales en coordenadas referenciadas al marco de referencia de cada uno de los sensores. La interfaz de la aplicación, mostrada en la Figura 3.11, despliega dos cuadros de imagen; el primero es el cuadro de imagen con la información del flujo de datos a color que tiene una resolución de 1920 x 1080 pixeles en formato RGB y el otro cuadro de imagen con la información del flujo de datos de profundidad mostrada en niveles de gris con una resolución de 512 x 424.

Figura 3.11. Interfaz de la aplicación *KinectColorCapture*.

Se utiliza una función del SDK del sensor Kinect que se encarga de segmentar a la persona frente al sensor de manera que los objetos del fondo son removidos, lo cual representa una gran ventaja en el filtrado inicial y el tamaño de los archivos de datos se verá reducido significativamente en comparación a si se incluyera la información de todo el fondo.

El objetivo de la aplicación del Equipo de Captura es realizar la adquisición de datos de color y profundidad cuando reciba la señal de inicio de captura proveniente del Equipo de Control. Los datos adquiridos deben ser guardados con formato de nubes de puntos con color. Para esto se realiza un mapeo entre los pixeles de la imagen a color y los pixeles de la imagen de profundidad. El mapeo consistió en encontrar el valor de profundidad que le corresponde a cada pixel de color RGB, teniendo en cuenta que la resolución de los cuadros de imagen de color y profundidad es diferente. Para realizar el mapeo se utilizan funciones incluidas en la clase *Coordinate Mapping* del SDK del sensor Kinect, cuyo procedimiento es similar al de calibración en estéreo al proyectar un cuadro de imagen sobre otro.

La clase *Coordinate Mapping* posee funciones para el mapeo entre diferentes marcos de referencias o espacios, por ejemplo, el espacio a color representado por la imagen a color, el espacio de profundidad, representado por la imagen a profundidad y el espacio de la cámara, representado por el marco de referencia de la cámara. Para los fines de este proyecto se decidió implementar las siguientes funciones basadas en el algoritmo que se presenta en [29] para la captura de datos tridimensionales y mapeo de color a profundidad usando Kinect:

- *MapDepthFrameToCameraSpace.* Mapeo o proyección de los datos de profundidad al marco de referencia de la cámara o espacio de la cámara.
- *MapCameraPointsToColorSpace*. Mapeo o proyección de los datos del espacio de la cámara obtenidos en el punto anterior hacia el espacio de color.

Es conveniente mencionar que durante el proceso de adquisición de puntos tridimensionales, la persona deberá ser captada totalmente por ambos cuadros de imágenes, de otra forma no se podrá hacer el mapeo correspondiente y como resultado el programa se detendrá y con ello el proceso de adquisición no será satisfactorio.

La primera función genera la nube de puntos en coordenadas *xyz* y la segunda función extrae el color relacionado a la misma nube de puntos generada. La Figura 3.12 resume el proceso de obtención de datos tridimensionales a color aplicando el proceso propuesto en [29].

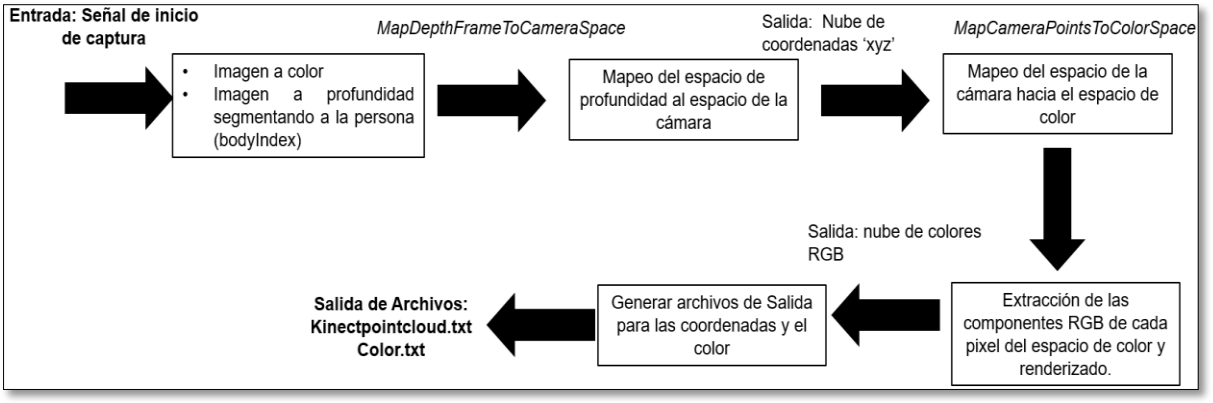


Figura 3.12. Metodología de extracción de nube de puntos en la aplicación *KinectColorCapture*.

Los archivos de salida de esta aplicación son llamados como *Kinecpointcloud.txt* y *Color.txt* que contienen la información de coordenadas y de color respectivamente. Es posible integrar los datos de profundidad con los datos de color en Matlab como se muestra en la Figura 3.13.

Una vez visualizada la nube de puntos proveniente de cada uno de los sensores, es posible trabajar ahora en la fusión de los datos aplicando transformaciones geométricas sobre las nubes de cada sensor para que tengan un origen común. Este proceso de fusión de datos se abordará en el siguiente capítulo.

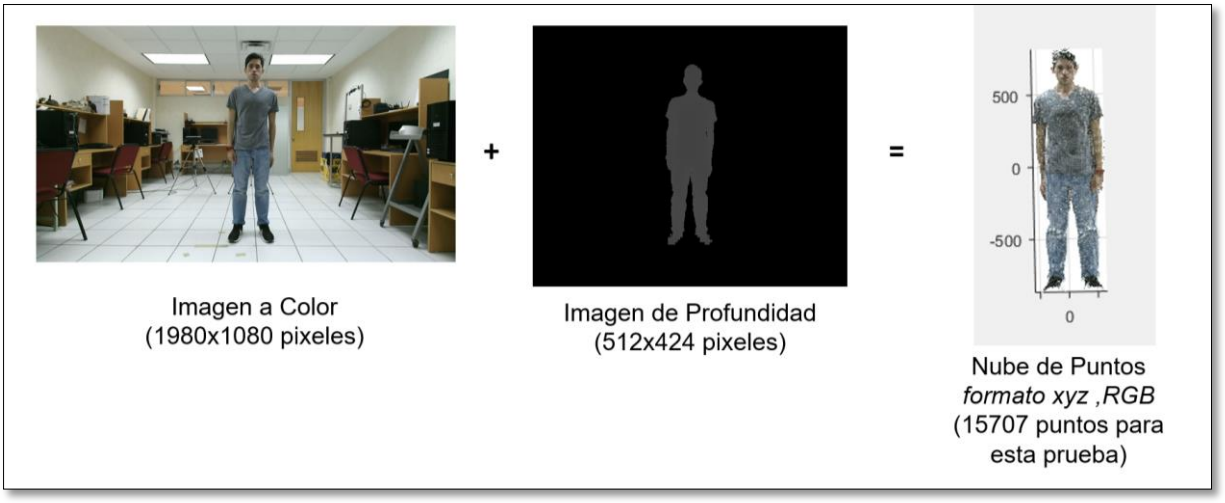


Figura 3.13. Mapeo entre la imagen con datos de color y la imagen con datos de profundidad para obtener una nube de puntos tridimensionales a color.

3.3.3 Aplicación del Equipo de Control

La aplicación *KinectSocketClient* tiene como finalidad interconectar el Equipo de Control con los Equipos de Captura y enviar comandos hacia estos últimos con el fin de controlar el proceso de captura de datos. Implementa un esquema cliente/servidor por medio *sockets* para el envío de mensajes sobre el protocolo TCP/IP. Los Equipos de Captura tendrán el rol de servidores de *sockets* y el Equipo de Control será el cliente. La Figura 3.14 muestra la conexión a través de *sockets* utilizando el protocolo TCP/IP.

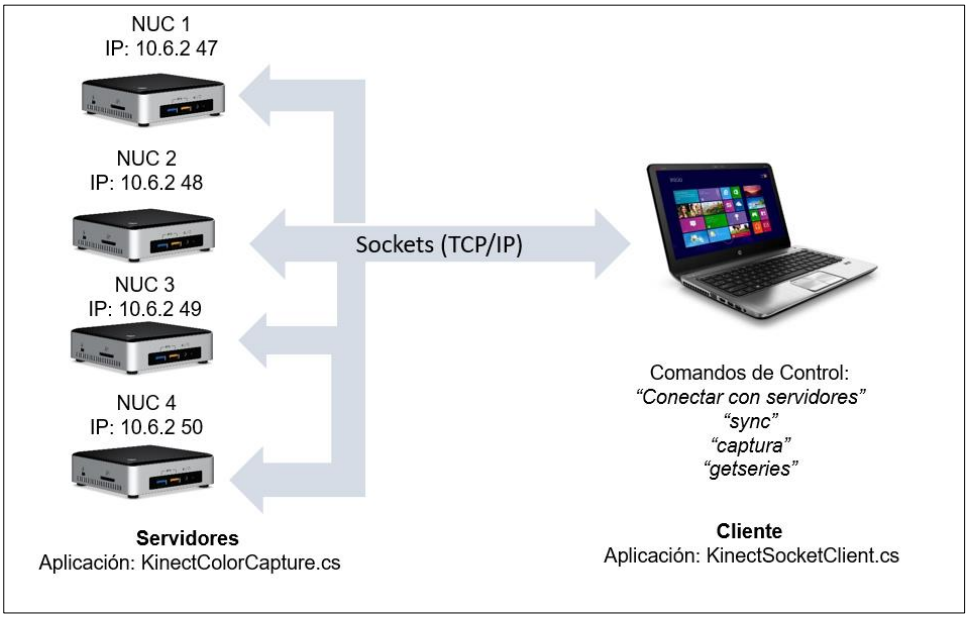


Figura 3.14. Diagrama de conexión entre servidores y clientes vía Sockets para adquisición de datos tridimensionales.

La aplicación del Equipo de Control ejecuta diferentes táreas. Primeramente, busca los servidores de *sockets* en los Equipos de Captura enviando solicitudes de conexión a sus respectivas direcciones IP y luego recibe mensajes de los servidores una vez reconocida la conexión. Entonces podrá enviar el mensaje para el inicio de captura de datos y esperar a recibir el mensaje de fin de captura para luego recolectar los datos adquiridos. La Figura 3.15 muestra la interfaz de usuario de la aplicación del Equipo de Control.

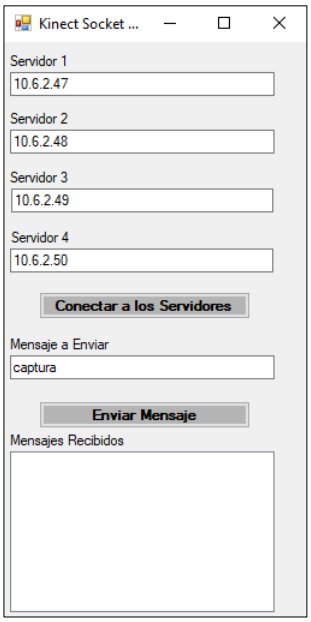


Figura 3.15. Interfaz de usuario de la aplicación del Equipo de Control.

Los Equipos de Captura solo reconocen los mensajes con comandos preestablecidos que le son enviados a través de la aplicación del Equipo de Control y en función del comando recibido será la acción a realizar. Los comandos deben escribirse en este orden para garantizar una correcta adquisición de datos:

1. Comando del botón para establecer conexión. Una vez escritas correctamente las direcciones IP de cada servidor de Sockets (Equipo de Captura), el botón Conectar a los Servidores establecerá la conexión entre el cliente (Equipo de Control) y los servidores (Equipos de Captura). Si la conexión fue exitosa se muestra en la parte inferior de la interfaz el mensaje recibido de cada Equipo Captura conectada.

2. Comando "*sync*". En el desarrollo del trabajo en [6], la sincronización de la fecha y hora de cada uno de los clientes con el servidor se ejecutaba de forma manual a través de la consola de comandos del sistema operativo ejecutando la instrucción:

> *net time \\Nombre_del_equipo_de_Control o IP /set)*

Cada cierto intervalo de tiempo (20 o 30 minutos) se vuelve a ejecutar la sincronización. Se incluyó el comando "sync" que manda llamar a la consola de comandos y ejecuta la sincronización automáticamente.

Para que el comando de sincronización funcione adecuadamente, se necesita comprobar previamente que el cliente y el servidor pertenezcan al mismo grupo de trabajo y se desactiven las directivas de *firewall* que impidan realizar la conectividad; esto en configuración de la red de los Equipos de Captura y de control.

3. Comando "*captura*". Este comando envía la señal a los Equipos de Captura para iniciar la adquisición de datos. Es posible escribir también, "captura *w* tiempo"; donde *w* es la cantidad de segundos que esperará el programa antes de enviar el comando a los Equipos de Captura. Por ejemplo, si se desea esperar a que la captura de datos inicie después de 10 segundos se debe escribir "captura w 10" y luego oprimir el botón Enviar Mensaje. Los Equipos de Captura envían mensajes de finalización de captura hacia el Equipo de Control una vez terminada la adquisición. Existirán diferentes causas de una adquisición no satisfactoria:

- No se tiene la conectividad entre el cliente y los servidores de *sockets*. Ocasionado por problemas de red, una incorrecta configuración de red o si no se ingresa correctamente la dirección IP de cada uno de los Equipos de Captura.

- Error de captura. Uno o varios de los Equipos de Captura no logró capturar información durante el proceso de adquisición generando archivos con 0Kb de almacenamiento.

- Error de versiones de los programas. Las actualizaciones automáticas que ejecuta Windows desinstalan complementos utilizados por las funciones del SDK del sensor Kinect. Una situación recurrente es una actualización que provoca se desinstale el Reproductor de Windows Media utilizado para visualización de imágenes.

4. Comando "*getseries*". Los datos almacenados en formato de nubes de puntos y de color RGB generados por cada Equipo de Captura se encuentran alojados una carpeta compartida en la raíz del disco duro local. El comando "*getseries*" se encargada de mover cada uno de los

archivos generados en el proceso de captura hacia una ruta especificada en el Equipo de Control. Para ello es importante que el Equipo de Control tenga acceso a la carpeta compartida de cada Equipo de Captura, esto es posible si se configura el acceso compartido con permisos de lectura para dicha carpeta y se habilitan los permisos de acceso por medio de la red pública.

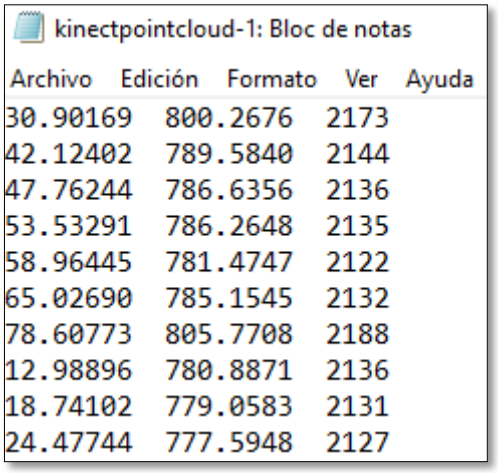
IV. FUSIÓN DE DATOS TRIDIMENSIONALES DE CUATRO SENSORES RGBD

4.1 Estructura de la Base de Datos

Una vez finalizado el proceso de captura se generó dos archivos en formato de texto plano (.txt) que contienen los datos de profundidad en coordenadas *xyz* y sus respectivos componentes de color rojo, verde y azul (RGB). Los archivos son denominados KinectPointCloud-*n*.txt y Color-*n*.txt respectivamente, donde *n* es el número de cuadro adquirido. Por lo general se capturan cinco cuadros por cada persona escaneada. Los archivos son alojados en una carpeta compartida en red en cada uno de los Equipos de Captura en su disco local C.

El tipo de datos almacenados en el archivo KinectPointCloud-n.txt son números de punto flotante de hasta siete cifras significativas. Los datos son mediciones en milímetros para cada coordenada *x, y* y *z,* en ese orden, en el marco de referencia de cada sensor. El tipo de datos almacenado en el archivo Color.txt es de tipo entero con un rango entre 0 y 255, ya que son los posibles valores que puede tomar cada componente RGB. La Figura 4.1 a) muestra una fracción de los datos de profundidad contenidos en un archivo kinectpointcloud-1.txt y la Figura 4.1 b) muestra la fracción correspondiente a los datos de color contenidos en un archivo Color-1.txt.

a)



b)

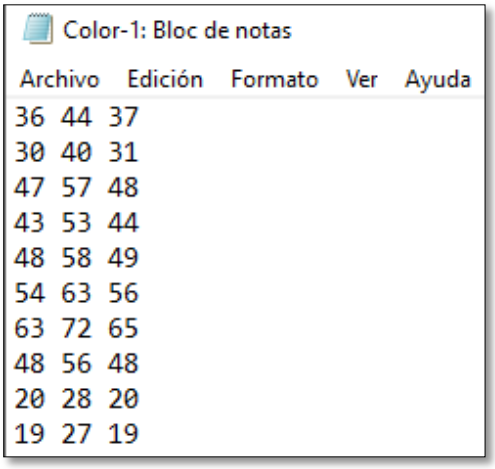


Figura 4.1. Fragmento de datos obtenidos por el sistema de captura y almacenados en: a) archivo con coordenadas xyz en milímetros y b) archivo con datos de color en formato RGB.

Al finalizar el proceso de escaneo de una persona se habrán generado varios archivos con datos de profundidad y color en cada Equipo de Captura. Una vez que el Equipo de Control recibe el mensaje de fin de captura, entonces puede transferir por medio de la red, los datos guardados en una carpeta compartida en cada Equipo de Captura y los almacena en una base de datos creada con el fin de organizar la información para su posterior procesamiento.

Respecto a la estructura de la base de datos, esta consta de tres niveles: el primer nivel hace referencia a la fecha de inicio de la sesión de captura, en el segundo nivel se crea una carpeta por cada una de las personas escaneadas, denominadas S1, S2, etc., y en el tercer nivel se crea una carpeta por cada Equipo de Captura, denominadas NUC1, NUC2, NUC3 Y NUC4, es aquí donde se guardan los archivos de datos de profundidad y de color.

Cuando se ingresa el comando *getseries* en la interfaz de usuario de la aplicación *KinectSocketClien*t, que corre en el Equipo de Control, se transfieren los archivos de las carpetas compartidas en los Equipos de Captura hacia la ubicación que les corresponde en la base de datos cuya carpeta tiene el nombre de *Muestras_fecha* en donde fecha corresponde al mes y día de la captura. Durante la etapa de fusión de datos se genera otra carpeta llamada *Fused_PC*, para almacenar la matriz de fusión de datos. La estructura de la base de datos se muestra en la Figura 4.2 y es aplicable para m número de equipos en caso de que se requiera ampliar el sistema de conectividad entre cliente y servidores.

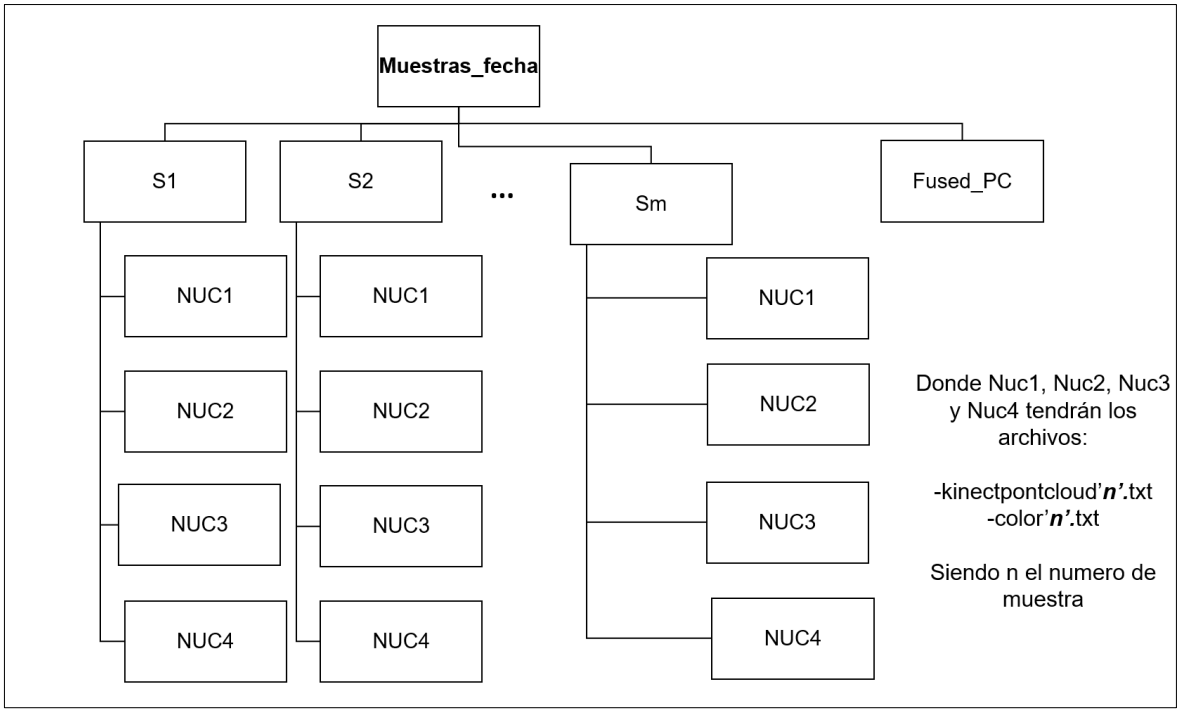


Figura 4.2. Estructura de base de archivos para cuatro sensores RGBD.

4.2 Proceso de Fusión de Datos

El proceso de fusión de datos tiene como objetivo llevar a cabo una transformación geométrica sobre los datos de profundidad capturados por cada sensor para luego fusionarlos en una sola nube de puntos con color. El programa que lleva a cabo esta tárea es llamado *data_fusion (sec_num, área),* cuyo código se muestra en el Anexo 4, en donde *sec_num* es el número de secuencia y *área* se refiere a las áreas de captura tipo rectángulo y tipo cruz. Este programa fue desarrollado en Matlab ya que cuenta con una librería de funciones para procesamiento y visualización de nubes de puntos.

- Entrada a la función: Archivo de coordenadas (.txt)+ Archivo de color (.txt)

- Salida de la función: Matriz de nube de puntos (.mat) + Archivo de modelado 3D (.ply)

El proceso de fusión de datos sigue la metodología presentada en la Figura 4.3 comenzando desde la apertura de archivos hasta la fusión final de los datos de profundidad de los cuatro sensores RGBD en una sola nube de puntos. El proceso de fusión de datos fue llevado a cabo para los datos capturados en el área de tipo rectangular y en el área con

geometría tipo cruz, cambiando únicamente la forma en que sus algoritmos realizan las transformaciones geométricas de las nubes puntos de los cuatro sensores.

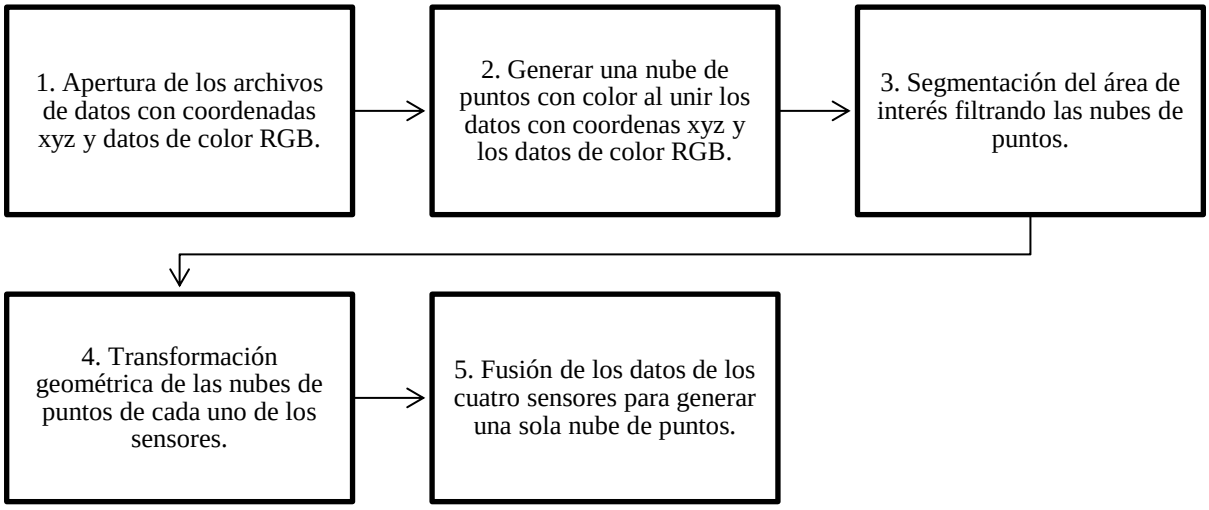


Figura 4.3. Diagrama del proceso de fusión de datos tridimensionales.

4.2.1 Apertura de archivos de datos y generación de nubes de puntos con color

La base de datos contiene los archivos de texto de coordenadas *kinectpointcloud-n.txt* y de color *Color-n.txt*, la función desarrollada en Matlab carga los datos como matrices para su procesamiento y fusión con el fin de generar una sola nube de puntos que es guardada en el archivo denominado *Fused_ptc.mat*.

La Figura 4.4 esquematiza el proceso de apertura y generación de nubes de puntos a color comenzando por abrir el archivo de coordenadas especificando la ruta del directorio raíz de la base de datos denominada *Muestras_fecha*, ubicada en el Equipo de Control. La función *load* se encarga de cargar los datos del archivo en una matriz de dimensiones de *n* x 3, donde *n* es el número de renglones que representan los puntos adquiridos durante el proceso, este oscila entre 11,000 y 20,000 puntos tridimensionales y 3 son las columnas del espacio tridimensional *xyz*. Posteriormente, se vuelve a utilizar la misma función para cargar la información de los datos de color en otra matriz. Teniendo dos matrices y usando operaciones de Matlab se unen los datos para generar otra nueva matriz de dimensiones *n* x 6, donde *n* es la cantidad de puntos totales, por lo que la matriz de coordenadas tridimensionales y la de información de

color deben tener el mismo tamaño puesto que provienen de la misma captura de imagen, de lo contrario el proceso de generación de nube de puntos no podrá realizarse. Las seis columnas que definen a la matriz llamada *fusion* contienen la coordenada en *x*, coordenada en *y*, coordenada en *z*, componente de color rojo, componente de color verde y componente de color azul, en ese orden.

La función *pointcloud* admite tres parámetros de entrada y se aplica a la matriz de dimensiones *n* x 6:

>> pointcloud(xyz,'Color',RGB)

Donde el primer parámetro son las columnas correspondientes a las coordenadas *xyz* en la matriz de *n* x 6, el segundo parámetro es la palabra reservada *'Color'*, indicando que se añade el dato de color a cada punto tridimensional. El tercer parámetro son las columnas correspondientes a las componentes de color RGB, convertido en formato byte (uint8) antes de pasarlo a la función *pointcloud*. Finalmente es posible la visualización de la nube de puntos usando la función *pcShow.*

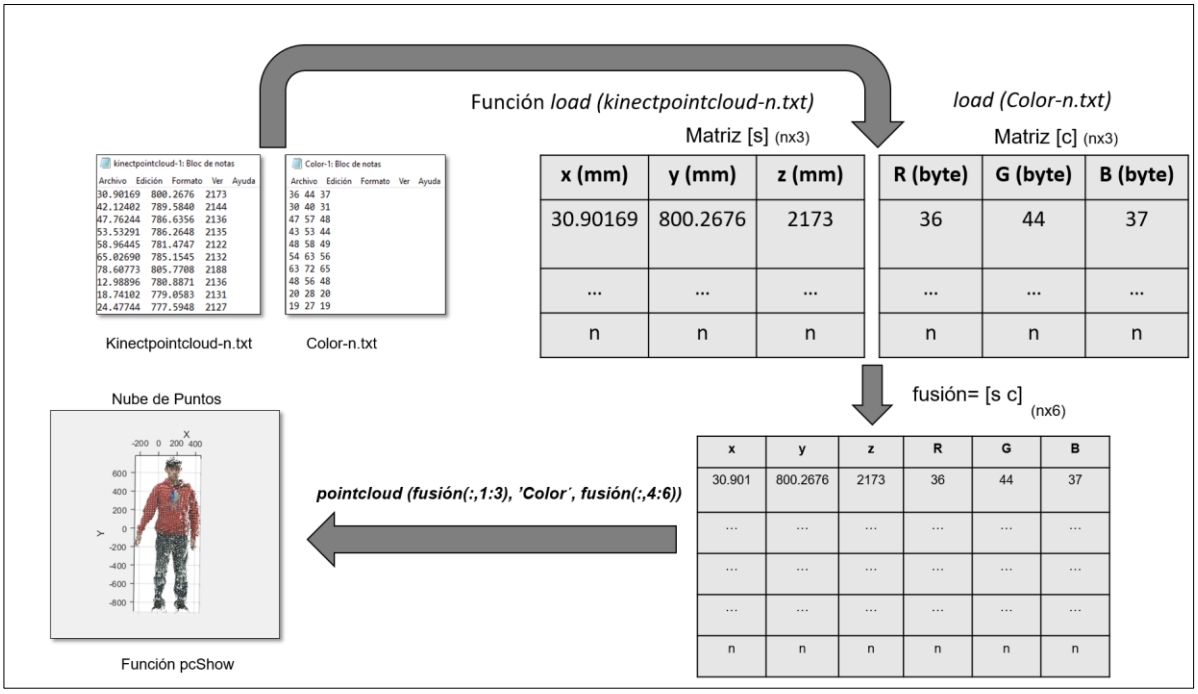


Figura 4.4. Apertura de archivos de datos y generación de nube de puntos.

4.2.2 Segmentación del área de interés en las nubes de puntos

La segmentación del área de interés queda definida como la extracción de los puntos tridimensionales del cuerpo captado por cada uno de los sensores. El área de interés tiene forma de prisma rectangular, y se acota con un valor máximo y otro mínimo para cada uno de los tres ejes y estaría ubicada con el centro de su base sobre el punto de referencia *So*. Los datos que se encuentran fuera de los límites del área de interés son considerados como ruido y, por ende, se filtran.

Las dimensiones del área de interés con respecto al centro de cada sensor son:

- Para el eje X: 1000 mm hacia la izquierda (lado negativo) y 1000 mm hacia la derecha (lado positivo).
- Para el eje Y: 900mm hacia abajo (lado negativo) para filtrar el ruido generado por el suelo y 1100mm hacia arriba (lado positivo), esto permite capturar personas de hasta 2 mts. de altura.
- Para el eje Z: Entre 1000 mm y 3000 mm hacia enfrente del sensor (lado positivo), recordar que la persona se encuentra a 2000 mm enfrente del sensor.

Este proceso de segmentación se aplica a las nubes de puntos con color capturadas por cada uno de los cuatro sensores y es independiente del tipo de área de captura.

4.2.3 Transformación geométrica de las nubes de puntos adquiridas en el área de captura tipo rectángulo

Se realizó una transformación geométrica formada por una rotación R seguida una traslación T de los datos tridimensionales en las nubes de puntos adquiridas en el área de captura tipo rectángulo. La transformación es definida con la siguiente ecuación vectorial:

$$P_T = RP_O + T\ . \tag{4.1}$$

Donde:

-P_T es el punto transformado por la rotación y traslación vectorial.

-P_o es un punto tridimensional referenciado al sistema coordenado de cada sensor con coordenadas $[Px\ Py\ Pz]^T$.

-R es la matriz de rotación en tres dimensiones, utilizada para alinear el sistema coordenado de cada uno de los sensores con el sistema coordenado de referencia con origen en *So*.

-T es el vector de traslación en tres dimensiones utilizado para ubicar el origen del sistema coordenado de cada uno de los sensores con sistema coordenado de referencia con origen en *So*.

La Figura 4.5 muestra los sistemas coordenados de cada sensor ilustrando la forma en que se realizan las rotaciones y traslaciones. Los sistemas coordenados del sensor S1 y el sensor S2 tienen la misma orientación en sus ejes coordenados. Los sistemas coordenados del sensor S3 y el sensor S4 también tienen la misma orientación en sus ejes coordenados, pero con una rotación de 180 grados sobre el eje *z* con respecto a S1 y S2.

La transformación geométrica se aplica a los datos tridimensionales de las nubes de puntos de cada uno de los cuatro sensores RGBD adquiridos en el área de captura tipo rectángulo de la siguiente manera tomando como referencia el origen So:

- *Para el sistema coordenado del sensor S1*

No sufre rotación alguna debido a que su sistema coordenado se encuentra con la misma orientación del sistema coordenado de referencia con origen en el punto *So*. En este caso, la matriz de rotación es la matriz identidad. Se realizó una traslación sobre el eje *x* por una distancia de *d2/2* milímetros y una traslación sobre el eje *z* de -2000 milímetros representada por la distancia *d3*. Lo anterior queda expresado matemáticamente como:

$$P_{T\ (Sensor\ 1)} = I.P_{O(sensor\ 1)} + [(-42 + 500), 0, -2000\ ] \quad (4.2)$$

Los -42 milímetros de la ecuación (4.2) representan la distancia del centro de la cámara de profundidad al centro del sensor. La distancia *d2/2* tiene un valor de 500 milímetros.

- *Para el sistema coordenado del sensor S2*

No sufre rotación debido a que su sistema coordenado se encuentra con la misma orientación del sistema coordenado de referencia con origen en el punto *So*, por lo tanto, la

matriz de rotación es la matriz identidad. La traslación sobre el eje *x* es de *d2/2*, el cual tiene un valor de -500 milímetros más los 42 milímetros del centro del sensor a la cámara de profundidad y la traslación sobre el eje *z* de -2000 milímetros correspondiente a *d3*. Lo anterior queda expresado matemáticamente como:

$$P_{T\,(Sensor\ 2)} = I.P_{O(sensor\ 2)} + [(+42 - \ 500), 0, -2000\ ] \tag{4.3}$$

- *Para el sistema coordenado del sensor S3*

El sistema coordenado sufrirá de una rotación en el eje *x* y en el eje *z* en un ángulo de 180º. En el caso de la traslación sobre el eje *x*, se tendrá que desplazar una distancia *d2/2* de 500 milímetros más los 42 milímetros del centro del sensor a la cámara de profundidad, y la traslación sobre el eje *z*, correspondiente a la distancia *d3*, será de 2000 milímetros. Lo anterior queda expresado matemáticamente como:

$$P_{T\,(Sensor\ 3)} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}.P_{O(sensor\ 3)} + [(42 + 500), 0, 2000\ ] \tag{4.4}$$

- *Para el sistema coordenado del sensor S4*

El sistema coordenado sufrirá de una rotación en el eje *x* y en el eje *z* en un ángulo de 180º. En el caso de la traslación sobre el eje *x*, se tendrá que desplazar una distancia *d2/2* de 500 milímetros y la traslación sobre el eje *z*, correspondiente a la distancia *d3*, será de 2000 milímetros. Lo anterior queda expresado matemáticamente como:

$$P_{T\,(Sensor\ 4)} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}.P_{O(sensor\ 4)} + [(+42 - 500), 0, +2000\ ] \tag{4.5}$$

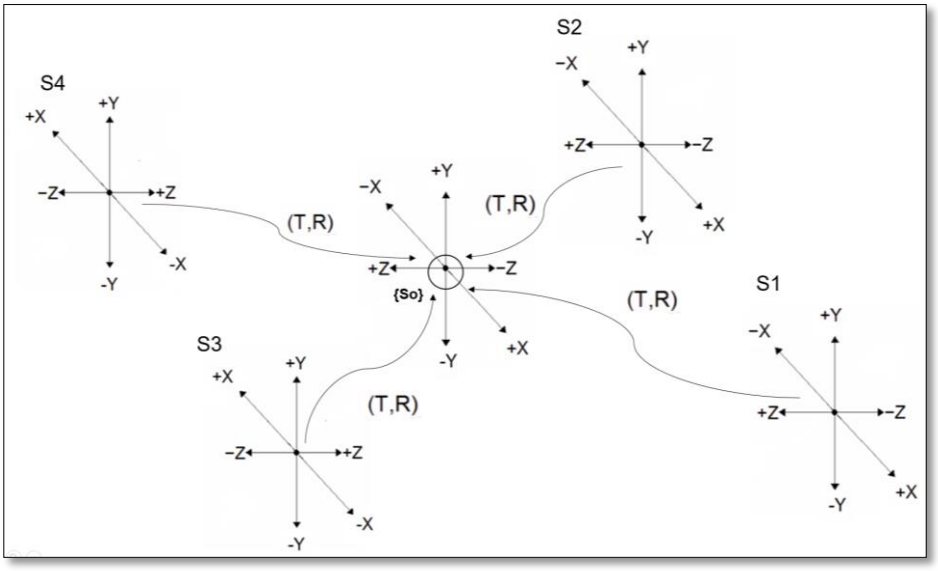


Figura 4.5. Diagrama del área de captura tipo rectángulo con los sistemas coordenados de los cuatro sensores y sus respectivas trasformaciones geométricas hacia el sistema coordenado de referencia con origen en el punto So.

Es importante resaltar que el proceso de fusión de datos tridimensionales depende de una buena calibración de los sensores en el área de captura, es decir, una mayor precisión en la alineación de los planos de los sensores permitirá que las nubes de puntos fusionadas de la persona escaneada sean unidas sin presentar deformaciones o alteraciones importantes.

Se realiza una segunda prueba de captura pidiéndole al sujeto a escanear, adopte diferentes posiciones; en las Figuras 4.6 a)- 4.6 d) se muestran las nubes de puntos antes de ser fusionadas y en la Figura 4.7 se muestra la nube de puntos resultante de la fusión de datos. Nótese que en las nubes de puntos resultantes del proceso de fusión de datos algunas partes de las zonas laterales no son captadas por ninguno de los cuatro sensores. Esto podría generar problemas en la reconstrucción tridimensional.

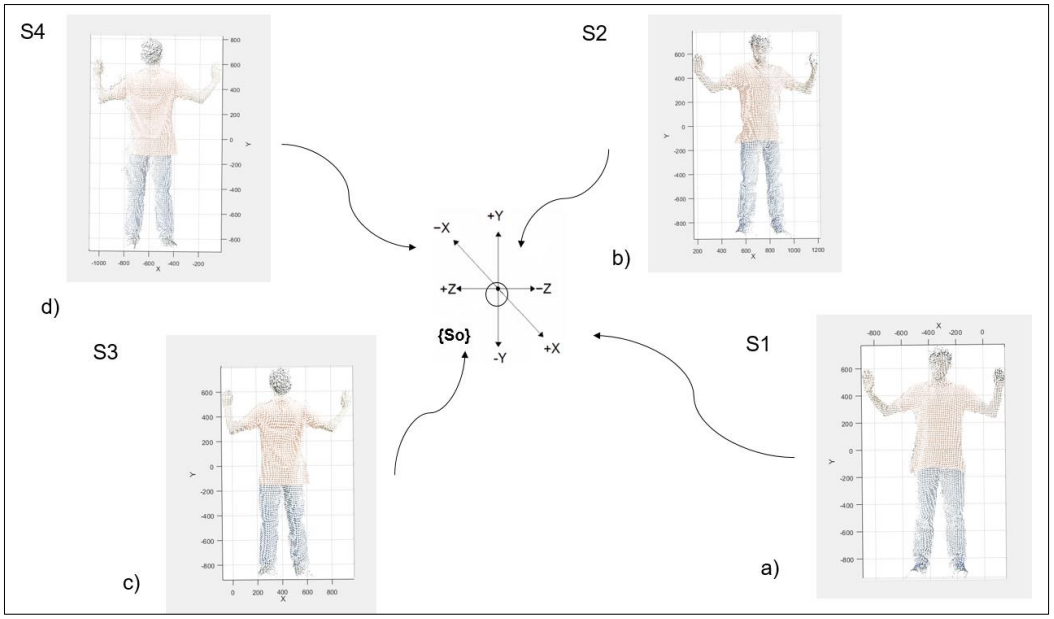


Figura 4.6. Nubes de puntos del área de captura tipo rectángulo antes de la fusión de datos. a) Sensor 1, b) sensor 2, c) sensor 3 y d) sensor 4.

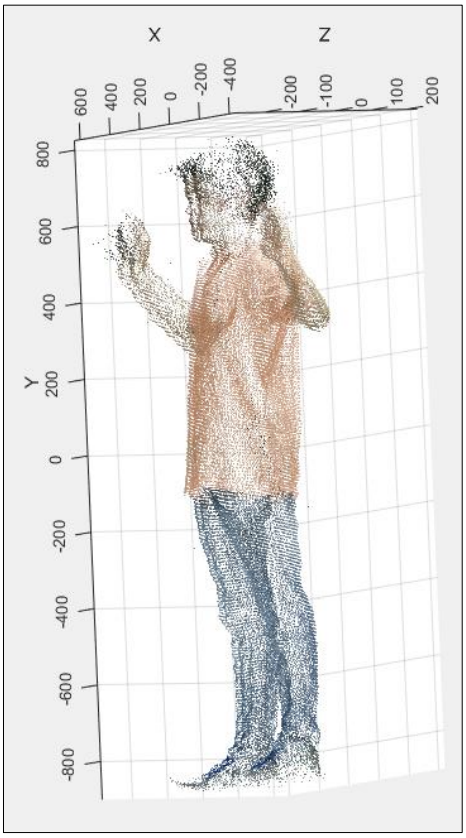


Figura 4.7. Nube de puntos resultante de la fusión de datos del área de captura tipo rectángulo.

4.2.4 Transformación geométrica de las nubes de puntos adquiridas en el área de captura tipo cruz

Se utiliza la misma función de transformación de la ecuación 4.1 para fusionar los datos de las nubes de puntos adquiridas en el área de captura tipo cruz. Se hace coincidir los sistemas coordenados de cada uno de los cuatro sensores con el sistema coordenado de referencia con origen en el punto *So* como se muestra en la Figura 4.8.

La transformación geométrica se aplica a los datos tridimensionales de las nubes de puntos de cada uno de los cuatro sensores RGBD adquiridos en el área de captura tipo cruz de la siguiente manera:

-Para el sistema coordenado del sensor S1

El sistema coordenado del sensor S1 tiene la misma orientación que el sistema coordenado de referencia con origen en el punto *So*, por lo tanto, la matriz de rotación R es la matriz identidad I. Se aplica una traslación de -42 milímetros sobre el eje *x* y una traslación sobre el eje *z* de -2000 milímetros. Entonces la función de transformación queda definida de la siguiente manera:

$$P_{T\ (Sensor\ 1)} = I.P_{O(sensor\ 1)} + [-42, 0, -2000\ ] \tag{4.6}$$

-Para sistema coordenado del sensor S2

Se realiza una rotación de 180 grados sobre el eje *x* y el eje *z* del sistema coordenado del sensor S2 para hacerlos coincidir con el sistema coordenado de referencia con origen en el punto *So*. Se aplicará una traslación de 42 milímetros sobre el eje *x* y una traslación sobre el eje *z* de 2000 milímetros. Entonces la función de transformación queda definida de la siguiente manera:

$$P_{T\ (Sensor\ 2)} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}.P_{O(sensor\ 2)} + [42\ , 0, 2000\ ] \tag{4.7}$$

- Para el sistema coordenado del sensor S3

Para alinear el sistema coordenado del sensor S3, basta con intercambiar la orientación de los ejes *x* y *z*, es decir, la dirección tomada del eje z del sensor ahora será el eje *x* y viceversa. Desde el punto de vista matricial, se intercambió la columna que contiene las coordenadas en el eje *x* por la columna que contiene las coordenadas en el eje *z*. La traslación se definió con

los ejes intercambiados, entonces la función de transformación queda definida de la siguiente manera:

$$P_{T\ (Sensor\ 3)} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} Pz \\ Py \\ Px \end{bmatrix} + [2000\ , 0, -42\ ] \tag{4.8}$$

- Para el sistema coordenado del sensor S4

Se sigue el mismo procedimiento con el sensor S4, se intercambia el eje *x* por el eje *z* y posteriormente se realiza la traslación, entonces la función de transformación queda definida de la siguiente manera:

$$P_{T\ (Sensor\ 4)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} . \begin{bmatrix} Pz \\ Py \\ Px \end{bmatrix} + [-2000\ , 0, +42\ ] \tag{4.9}$$

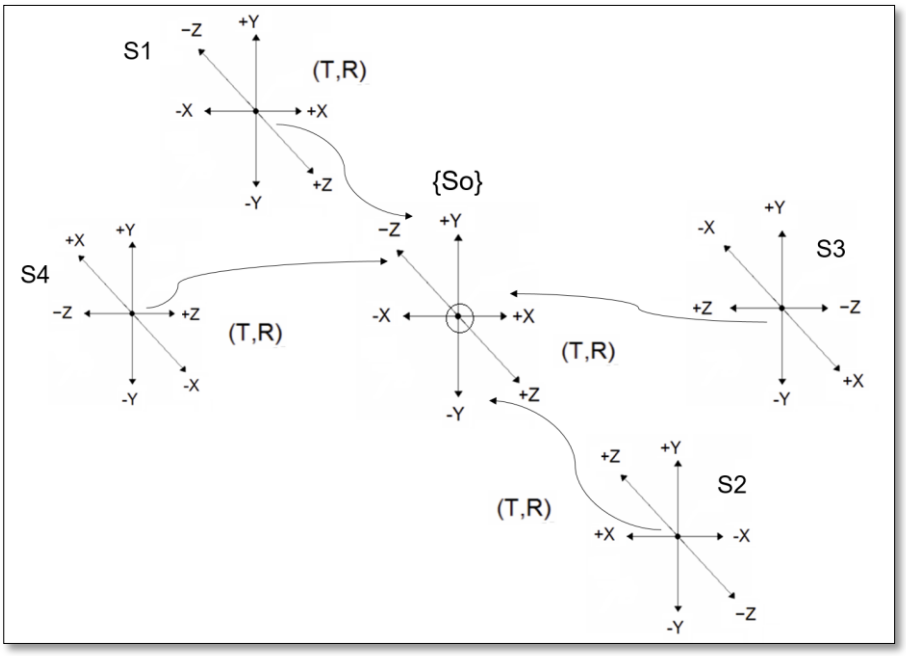


Figura 4.8. Diagrama del área de captura tipo cruz con los sistemas coordenados de los cuatro sensores y sus respectivas trasformaciones geométricas hacia el sistema coordenado de referencia con origen en el punto So.

Las Figuras 4.9 a)-d) muestran las nubes de puntos de cada sensor antes de ser fusionadas; el sujeto se posicionó dando un giro de 45 grados de tal manera que los cuatro sensores pudieran cubrir la mayor cantidad de datos en las zonas laterales.

La fusión resultante en vista isométrica puede apreciarse en la Figura 4.10 en donde se logra cubrir las zonas laterales del cuerpo, fenómeno que no sucedía con el área de captura tipo rectángulo, es decir, las zonas de oclusión laterales han podido ser cubiertas, sin embargo,

existen deficiencias en la captura de información en la parte superior de la cabeza y la detección completa de los pies debido a las variaciones en la reflexión de los rayos infrarrojos que inciden sobre el cabello y el suelo respectivamente; para ello se tendrán dos alternativas de solución: la primera es desarrollar un algoritmo destinado a la detección del cabello y eliminación de ruido que no sean afectados por la reflexión infrarroja, o bien, se cubren estos aspectos en la etapa de reconstrucción y modelado tridimensional, en donde se tendrá que hacer énfasis primeramente en tener una superficie cerrada y lo más apegada posible a la morfología de la persona escaneada.

Las Figuras 4.11 a)- 4.11 d) muestran las nubes de puntos de cada sensor que captaron una nueva persona en diferente postura y vistiendo un sombrero y lentes. Se observa que el sistema es capaz de detectar personas vistiendo lentes y sombrero. La Figura 4.12 muestra el resultado de la fusión de datos.

En el Anexo 5 se muestra una base de datos con diferentes personas escaneadas usando este tipo de área el cual resultó ser el más eficiente y en la siguiente sección se evalúa su desempeño.

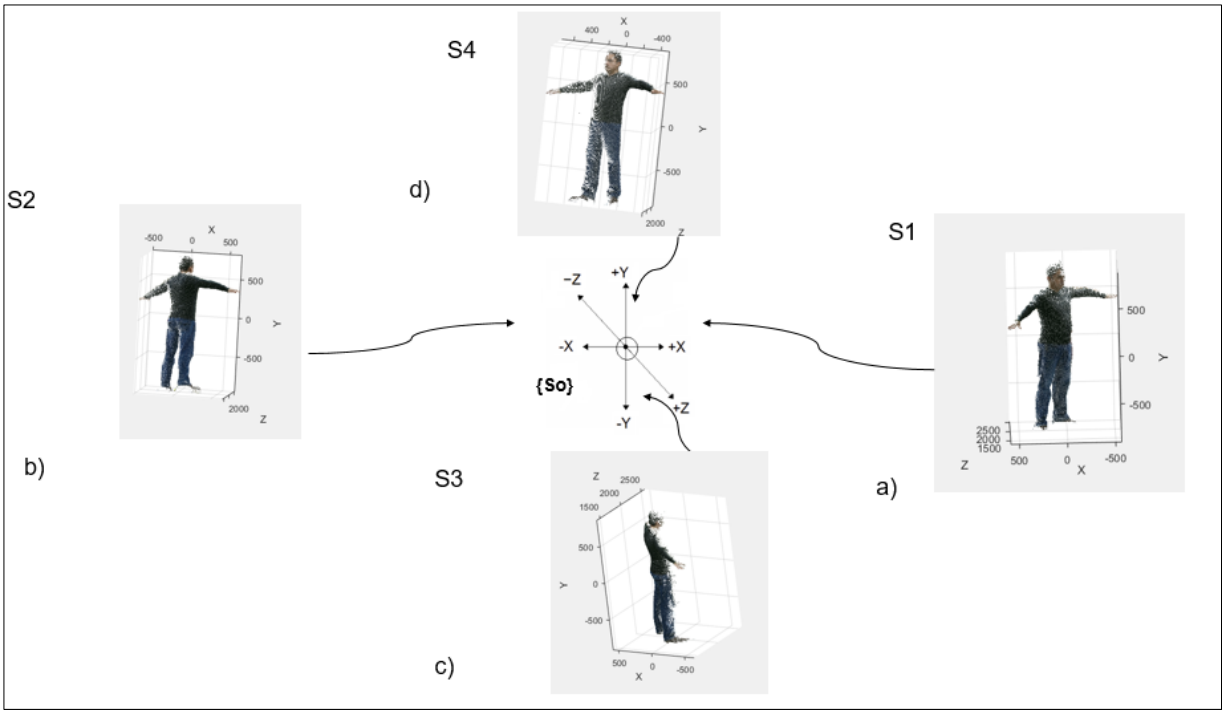


Figura 4.9. Nubes de puntos del área de captura tipo cruz antes de la fusión de datos. a) Sensor 1, b) sensor 2, c) sensor 3 y d) sensor 4.

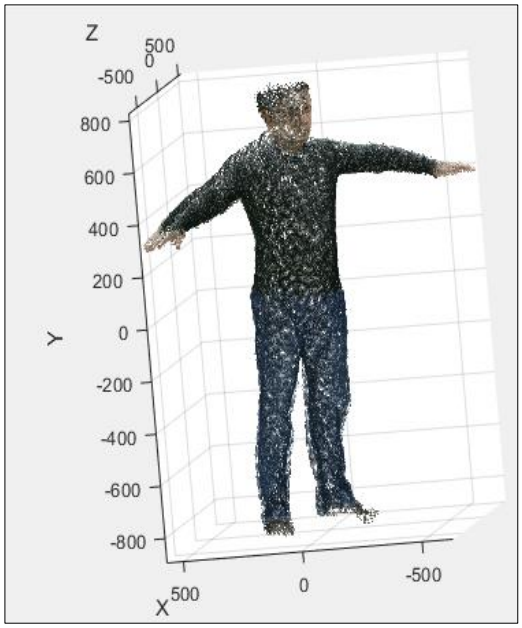


Figura 4.10. Nube de puntos resultante del proceso de fusión de datos del área de captura tipo cruz.

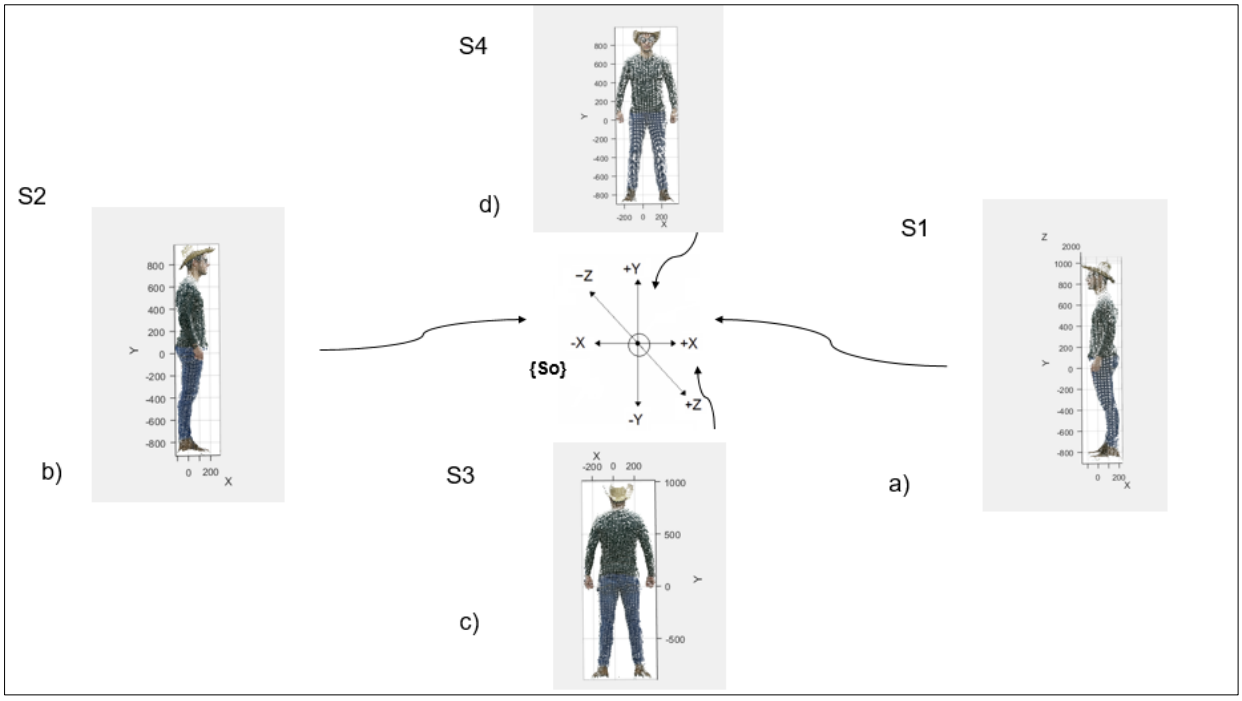


Figura 4.11. Nube de puntos de persona con sombrero y lentes antes de la fusión de datos. a) Sensor 1, b) sensor 2, c) sensor 3 y d) sensor 4

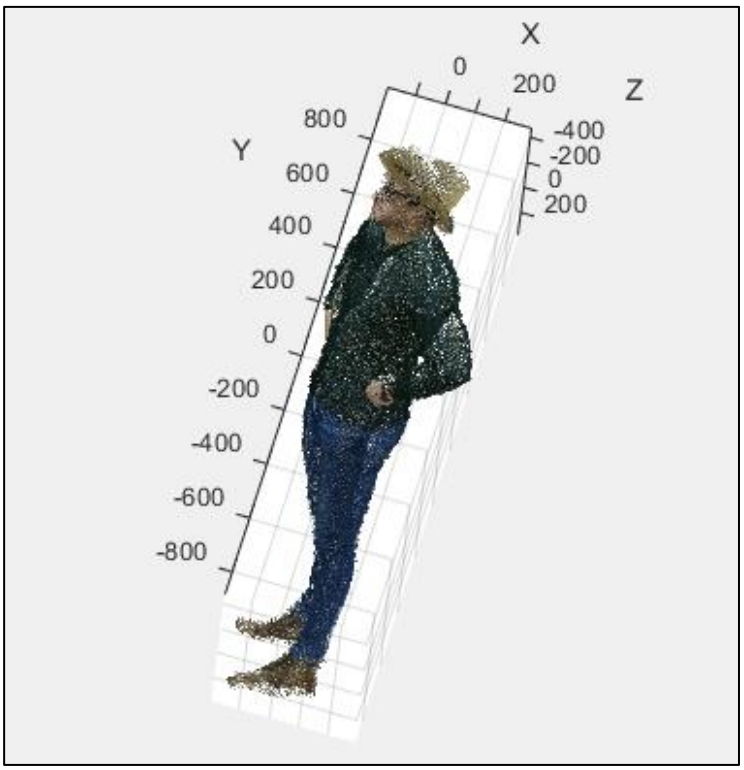


Figura 4.12. Nube de puntos resultante de la fusión de datos de una persona vistiendo sombrero y lentes obscuros.

4.3 Evaluación del Proceso de Fusión de Datos

Se llevó a cabo un experimento que permitió evaluar la eficiencia del proceso de fusión de datos. Consiste en colocar un pizarrón de color blanco en forma horizontal en el centro del área de captura cerca del suelo con la intención de que sea visto por los cuatro sensores. De esta manera se tiene una superficie plana y se conoce sus dimensiones y localización exacta con respecto a los cuatro sensores. Se modificó el código de la aplicación *KinectColorCapture* para que cada uno de los sensores captara la nube de puntos de la escena completa. La Figura 4.13 muestra la nube de puntos de la escena captada por uno de los sensores donde se puede apreciar el pizarrón (superficie plana) en la parte inferior. Se definió una región de interés en forma de cubo que contiene la superficie plana y se segmentó del resto de la escena. Una vez extraída la información de la región de interés de las nubes de puntos de cada sensor se llevó a cabo el proceso de fusión de datos aplicando la transformación geométrica a los datos de cada sensor para después fusionarlos en una sola nube de puntos. La Figura 4.14 muestra la superficie plana con los datos fusionados de los cuatro sensores.

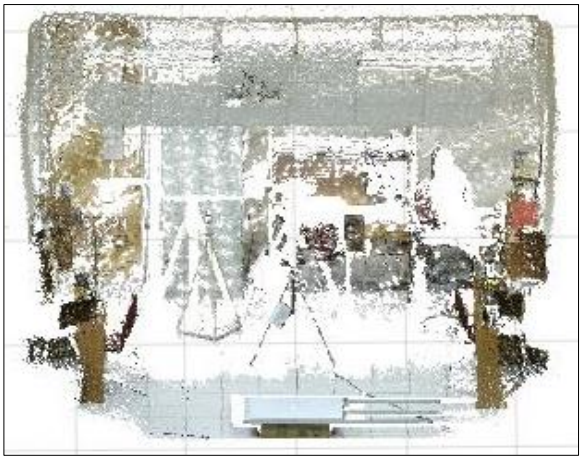


Figura 4.13. Escena capturada con el sensor 1 donde se aprecia la superficie plana.

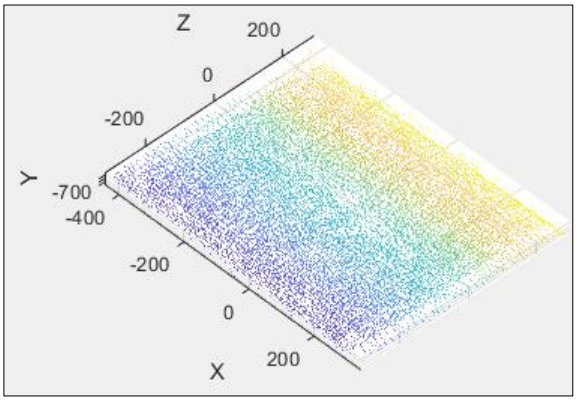


Figura 4.14. Resultado de la fusión de datos de la superficie plana segmentada.

El proceso de evaluación consistió en medir la dispersión de datos en las nubes de puntos tridimensionales de cada sensor antes y después de ser fusionadas respecto a un plano ajustado a los datos utilizando el método de regresión lineal múltiple [30]. Para el ajuste del plano se considera al eje Y como la variable dependiente en la regresión y los ejes X y Z como las variables independientes, de esta manera la ecuación del plano sería

$$y = a_0 + a_1 x + a_2 z \tag{4.10}$$

El método de regresión lineal múltiple permite calcular los coeficientes a_0, a_1 y a_2 del plano que mejor se ajusta a los datos de las nubes de puntos utilizando el criterio de minimización del residuo cuadrático. Una vez obtenido el ajuste se pueden calcular diferentes parámetros que permiten analizar la variabilidad de los datos capturados por cada sensor con respecto al plano ajustado y llevar a cabo el mismo análisis sobre los datos fusionados de los

cuatro sensores. Se espera que la variabilidad de los datos no se vea incrementada por el proceso de fusión de datos.

Los parámetros calculados para evaluar el ajuste son del Error Estándar Estimado (Sy/x,z) el cual deberá ser comparado con la desviación estándar (Sy) de los datos de las nubes de puntos. Si los parámetros cumplen con el criterio estadístico $Sy/x,z < Sy$, se considera un buen ajuste ya que el error de los datos con respecto al plano ajustado no es mayor a la desviación estándar de estos datos.

La superficie plana fue colocada de manera horizontal, de esta forma los coeficientes a_1 y a_2, correspondientes a las variables *x* y *z*, deberán ser cero y el coeficiente a_0 deberá ser igual a la distancia del centro del sensor a la superficie plana sobre el eje Y. Esta distancia fue medida físicamente dando un valor aproximado de 695 mm.

La Tabla 4.1 muestra los coeficientes calculados del plano ajustado a los datos de la superficie segmentada para cada uno de los cuatro sensores y también para del plano ajustado a los datos fusionados. Los planos ajustados a los datos coinciden, con errores mínimos, a los valores medidos físicamente de la superficie plana. En cada caso se muestra el cálculo de la desviación estándar de los datos en las nubes de puntos y del Error Estándar Estimado de estos datos con respecto al plano ajustado. Se puede observar que en todos los casos se cumple con el criterio $Sy/x,z < Sy$. Estos resultados permiten establecer que el proceso de fusión es adecuado, ya que el Error Estándar Estimado se encuentran en el orden de pocos milímetros. Este error no es significativo para la generación del modelo tridimensional.

Tabla 4.1. Resultados de la evaluación del proceso de fusión de datos.

Nube de Puntos	**Coeficientes del Polinomio de Regresión [a_0 a_1 a_2]**	**Desviación Estándar en milímetros (Sy)**	**Error Estándar Estimado en milímetros (Sy/x,z)**
Sensor 1	[-695.4932 0.0055 -0.005]	3.1250	2.7874
Sensor 2	[-696.3452 -0.0029 -0.023]	4.7122	2.3849
Sensor 3	[-695.6101 -0.0121 0.0026]	4.8616	4.2351
Sensor 4	[-695.7349 -0.0017 -0.0134]	3.5994	2.9144
Datos fusionados	[-696.3654 -0.0025 -0.0104]	4.1888	3.7507

V. PROCESO DE IMPRESIÓN TRIDIMENSIONAL DEL MODELO DE UNA PERSONA

5.1 Características del Equipo para Modelado Tridimensional

Se utilizó un equipo de cómputo de la marca Dell modelo G3. Su uso fue dedicado exclusivamente a la parte de modelado tridimensional, esto es debido a que la reconstrucción, mallado y creación de superficies de un modelo tridimensional requiere mayores recursos computacionales en comparación al de adquisición y fusión de datos. En la Tabla 5.1 se enlistan las características de dicho equipo.

Tabla 5.1. Características del equipo de cómputo usado para la reconstrucción tridimensional.

Equipo de modelado tridimensional Laptop Dell G3 15 Gaming
• Procesador Intel Core i5 8ª generación • Windows 10 Home de 64 bits • Pantalla FHD de 15,6” con retroiluminación LED y antirreflejo • Tarjeta de video NVIDIA GeForce GTZ 1050 con 4 GB de memoria gráfica • 8 GB de memoria RAM de 2.6 MHz

El software instalado en el equipo de cómputo consta de los programas: MeshLab para a la creación del modelo tridimensional en formato STL, Makerbot y Cura para la impresión 3D del modelo.

5.2 Almacenamiento de Nubes de Puntos en Archivos con Formato Poligonal (PLY)

El proceso de impresión 3D inicia almacenando la nube de puntos generada por el proceso de fusión de datos en un archivo que pueda ser manipulable para un software de diseño asistido por computadora (CAD), ya que la nube de puntos por sí sola, solo representa datos de puntos con coordenadas *xyz* sin relación alguna entre ellos.

El archivo de formato poligonal relaciona las coordenadas *xyz* como los vértices de un polígono creado uniendo dichos vértices, a esos polígonos se les denomina caras. Entonces el archivo PLY debe contener la información de los vértices y de las caras formadas. También se puede incluir información de color.

Dentro de la función desarrollada en Matlab para la fusión de datos fue posible guardar la nube de puntos en un archivo con formato PLY utilizando la función:

>>pcwrite(ptCloud, filename, 'Encoding', encodingType)

Donde el primer parámetro *ptCloud* deberá ser la matriz que contiene los datos de la nube de puntos, el segundo parámetro *filename* es el nombre del archivo donde se guardarán los datos, el tercer parámetro es la palabra reservada *'Encoding'* que define la codificación del archivo de salida y en el último parámetro *encodingType* se utiliza el tipo de codificación ASCII.

La función *pcwrite* de Matlab puede escribir archivos de nubes de puntos en formato PLY o PCD, esto dependerá de la extensión que se utilice (.ply o .pcd) en el nombre del archivo utilizado como segundo parámetro.

5.3 Reconstrucción Tridimensional con Meshlab

MeshLab es un software libre con fines académicos desarrollado por el Instituto de Ciencias y Tecnologías de la Información de Roma, destinado exclusivamente a procesos de reconstrucción, mallado, creación de superficies, edición, renderizado, reparación y filtrado de modelos tridimensionales. Se puede generar un modelo tridimensional en los formatos: PLY, STL, OBJ, VRML, COLLADA.

La ventaja de Meshlab sobre otros programas de modelado tridimensional, como Solidworks, Autodesk o Unity, es que el modelado no basa la construcción de modelos en herramientas de diseño sino en el manejo directo de algoritmos matemáticos, por lo que se requerirá conocimiento previo de cada uno de ellos y sus parámetros de funcionamiento. La Figura 5.1 muestra la interfaz de usuario de Meshlab, en la barra de herramientas se encuentra el listado de diversos algoritmos de reconstrucción y mallado entre los que destacan la creación de superficies por Poisson y el algoritmo de Pivoteo de Pelota tratados en el marco teórico.

A continuación de describe el proceso de reconstrucción tridimensional de modelo de una persona a partir de un archivo de nube de puntos con formato PLY. El objetivo es obtener un modelo cuyo formato de archivo pueda ser procesado por el software de una impresora 3D.

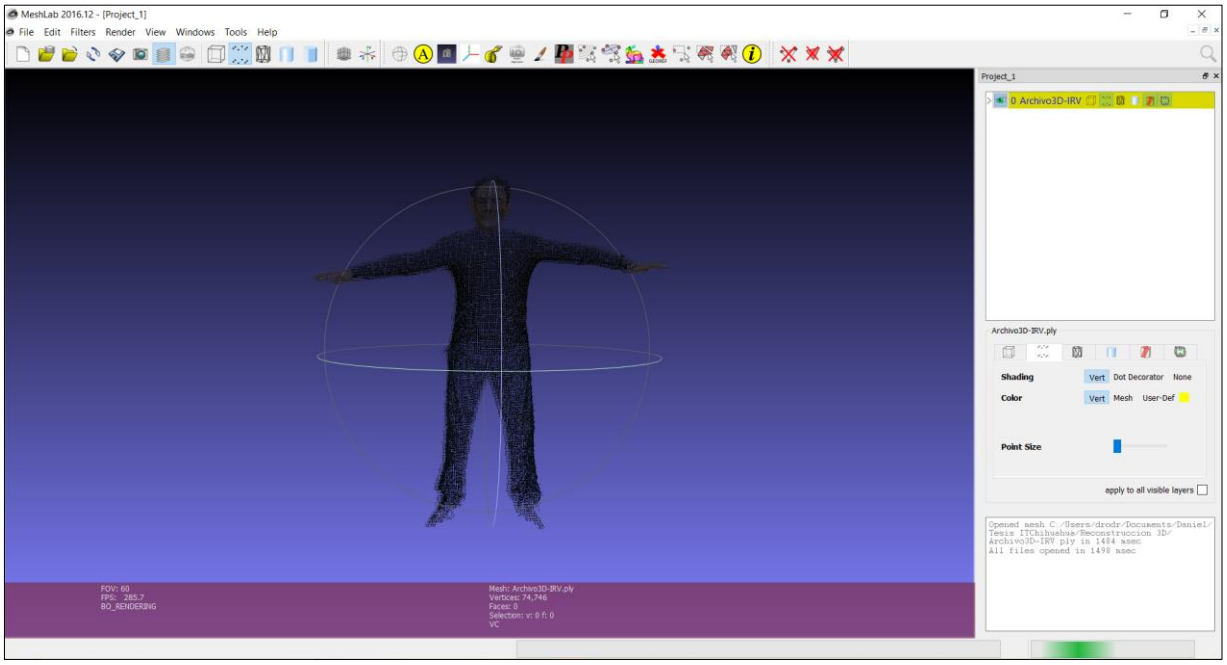


Figura 5.1. Interfaz de usuario de Meshlab.

Las etapas del proceso de reconstrucción tridimensional son las siguientes:

a. Simplificación y cálculo de vectores normales a los vértices.

A pesar de que se haya realizado un filtrado inicial de la nube de puntos durante el proceso de fusión de datos, se requiere que los puntos tridimensionales se encuentren distribuidos uniformemente para lograr la máxima eficiencia en el uso de algoritmos para la construcción de mallas y superficies.

La simplificación es una reducción de puntos que permite la obtención de los puntos más significativos para la creación de mallas y superficies. El algoritmo de reducción de puntos evita que existan datos muy cercanos entre sí, reduciendo el procesamiento computacional en etapas posteriores. Las alternativas más utilizadas para la simplificación de nube de puntos en MeshLab son el muestreo con discos de Poisson [31], el muestreo por número de puntos y el muestreo manual, donde el usuario selecciona en la interfaz los puntos a reducir aplicando operaciones de erosión y/o dilatación.

En el formato PLY se le denomina vértice a cada dato tridimensional de la nube de puntos. Entonces se realiza el cómputo de las normales de los vértices para determinar la orientación en la construcción de polígonos. Las normales se usan durante el proceso de mallado para calcular un vector normal a cada polígono construido.

b. Reconstrucción tridimensional.

Consiste en ir creando polígonos mediante la unión de los vértices. La unión de los polígonos adyacentes forma mallas y a su vez las mallas definen las superficies del modelo tridimensional.

Existen gran variedad de algoritmos de reconstrucción tridimensional los cuales son utilizados en función del tipo de geometría que presente la nube de puntos y dependen del uso que se le dé al modelo tridimensional.

Los algoritmos que fueron considerados en este proyecto de tesis son: la triangulación de Delunay, el algoritmo por pivoteo de pelota y la reconstrucción con superficies de Poisson, siendo este último el que proporcionó los mejores resultados en el cálculo del modelo tridimensional del cuerpo humano a partir de nubes de puntos adquiridas con sensores RGBD que presentan zonas de oclusión. Aunque el algoritmo es susceptible a puntos muy alejados considerados como ruido ya que tratará de unirlos a la superficie.

Los parámetros requeridos por el algoritmo de reconstrucción con superficies de Poisson son los siguientes:

-Profundidad (*Tree-Depth*). Indica la profundidad máxima del árbol que será usado para la reconstrucción tridimensional, Kazhan recomienda una profundidad de 8 niveles, llamados computacionalmente como *octrees* [25].

-Número Mínimo de Muestras. Indica el número mínimo de putos de muestra que deben caer dentro de un nodo del árbol (*octree)* y se encontrará en un rango de 1.0 a 5.0.

-*Pre-Clean*. Esta opción permite realizar un filtrado previo al inicio de la reconstrucción para remover vértices no referenciados o con vectores normales nulos.

-Confidence Flag. Manda la instrucción al reconstructor para usar el número de normales calculadas como la información básica de modelado.

c. Postprocesamiento del modelo tridimensional.

Se puede realizar algún otro tipo de filtrado, corrección de morfología en el mallado, cierre de oclusiones generados en el proceso de reconstrucción, modificación en parámetros de suavizado, compactación de vértices y caras cuando el modelo tridimensional así lo requiera.

Los procedimientos más comunes son la compactación de caras y vértices de la malla, remoción de puntos aislados, relleno de zonas de oclusión en la malla, entre otros.

d. Suavizado de la superficie.

Con los métodos de suavizado se busca que el modelo tridimensional tenga superficies uniformes. El algoritmo utilizado para el suavizado está basado en la técnica de Laplace y sus variantes, este algoritmo sustituye la posición de los vértices vecinos por un vértice central calculado por el promedio de estos en un proceso iterativo. [32].

Es conveniente mencionar que el proceso de modelado no es único y varía en función del tipo de morfología del objeto o persona, la experiencia del diseñador y hacia dónde va orientada la reconstrucción. Por esta razón, diversos trabajos manejan, inclusive, la aplicación de dos o más algoritmos de reconstrucción. Además, cuando se modifica un parámetro, se debe verificar que la capacidad computacional para realizar dicha tárea sea suficiente, de lo contrario el programa se cerrará y el modelo modificado no podrá ser guardado.

f. Exportar modelo tridimensional a un archivo con formato de esterolitografia (STL).

La finalidad de exportar el modelo tridimensional hacia un archivo tridimensional radica en el hecho de que la pieza ahora deberá ser manufacturada a través de la impresión tridimensional. Los formatos más comunes que utilizan las impresoras 3D son mencionadas en la sección 2.4 de esta tesis. Se eligió el formato STL ya que es el formato que una gran cantidad de impresoras genéricas y no genéricas manejan para imprimir.

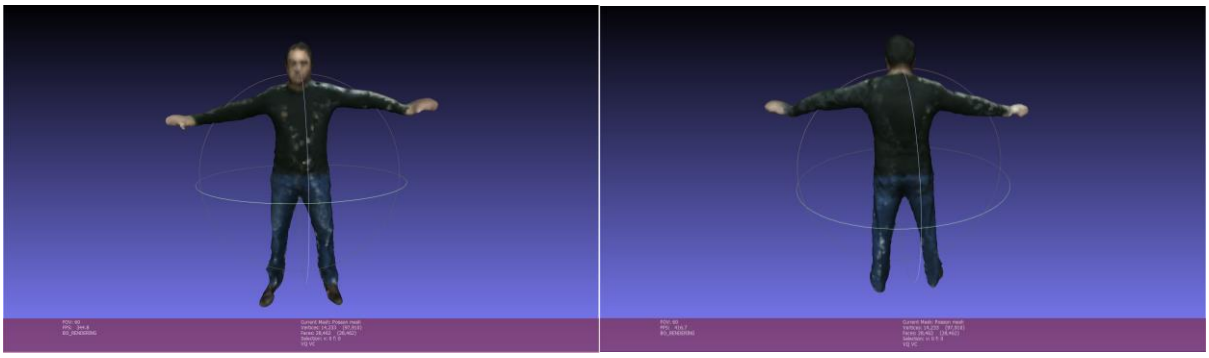
El archivo STL se genera desde Meshlab una vez que se ha terminado de editar el modelo tridimensional exportando el modelo tridimensional. Si el modelo generado tiene errores de topología no podrá ser guardado en el archivo STL y necesitará repararse los errores por medio de otros procedimientos.

A continuación, se presentan diferentes modelos tridimensionales generados a partir de la misma nube de puntos. Las Tablas 5.2, 5.3 y 5.4, muestran los tres procedimientos de reconstrucción tridimensional aplicados con diferentes algoritmos y parámetros. Las Figuras

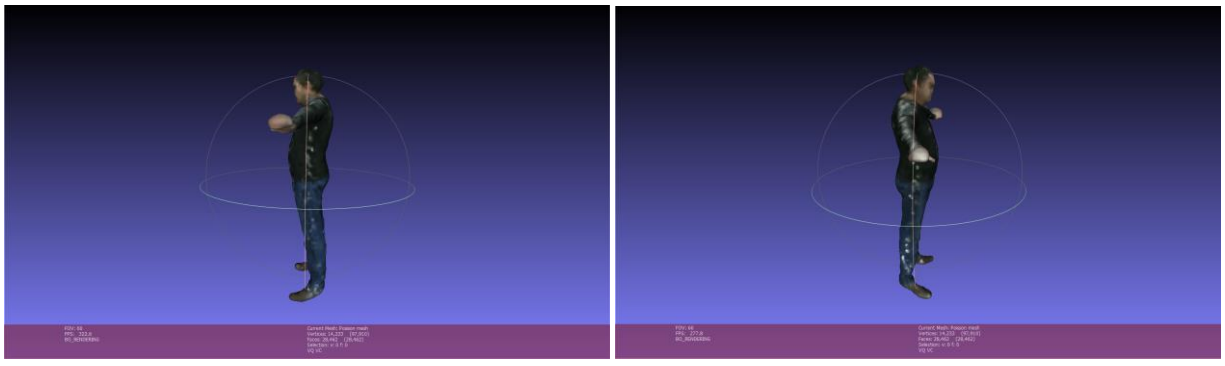
5.2, 5.3 y 5.4 muestran diferentes vistas del modelo generado en cada procedimiento. En el Anexo 5 se añaden otros modelos generados.

Tabla 5.2. Primer procedimiento de reconstrucción tridimensional.

Primer Procedimiento	
Pasos	Algoritmo/Operación
1.- Simplificación de la nube de puntos	Poisson Disk Algorithm
2.- Filtrado	Erosión+Dilatación
3.- Cálculo de vectores normales a los vértices	Sin suavizado y 10 número de vecinos
4.- Reconstrucción tridimensional	Algoritmo de superficies por Poisson con octrees
5.- Postprocesamiento	Eliminación de partes aisladas Eliminación de vértices repetidos
6.- Suavizado	HC-Laplace



a) b)

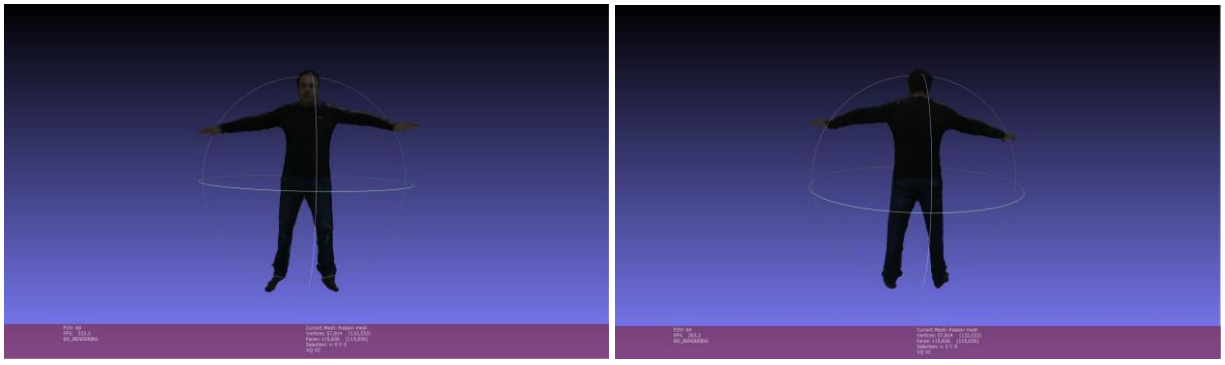


c) d)

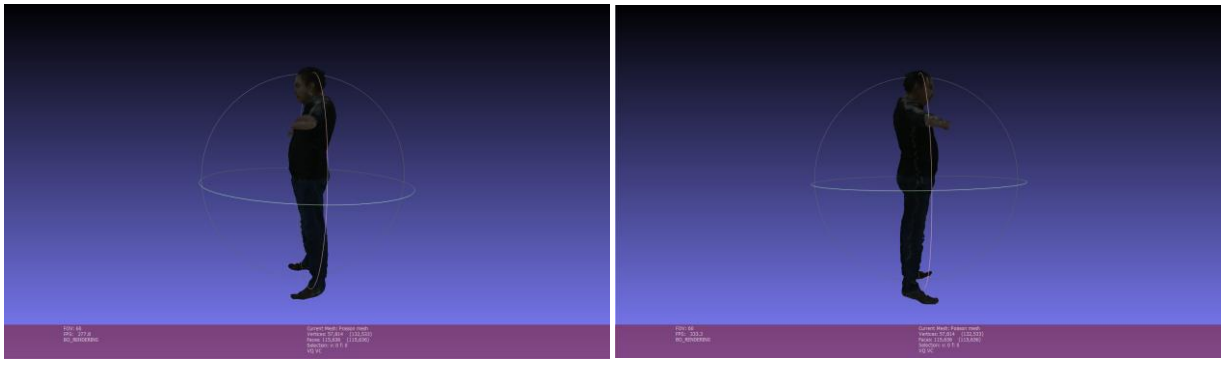
Figura 5.2. Modelo generado con el primer procedimiento de reconstrucción tridimensional. a) Vista frontal, b) vista posterior, c) y d) vistas laterales.

Tabla 5.3. Segundo procedimiento de reconstrucción tridimensional.

Segundo Procedimiento	
Pasos	Algoritmo/Operación
1.-Simplificación de Point Cloud	No aplica
2.-Filtrado	Erosión+Dilatación
3.- Cálculo de vectores normales a los vértices	Con suavizado y 11 número de vecinos
4.-Creación mallado y superficies	Algoritmo de superficies por Poisson con octrees
5.-Post Procesamiento	Eliminación de partes aisladas Compactación de vértices y caras
6.-Suavizado	HC- Laplace



a) b)

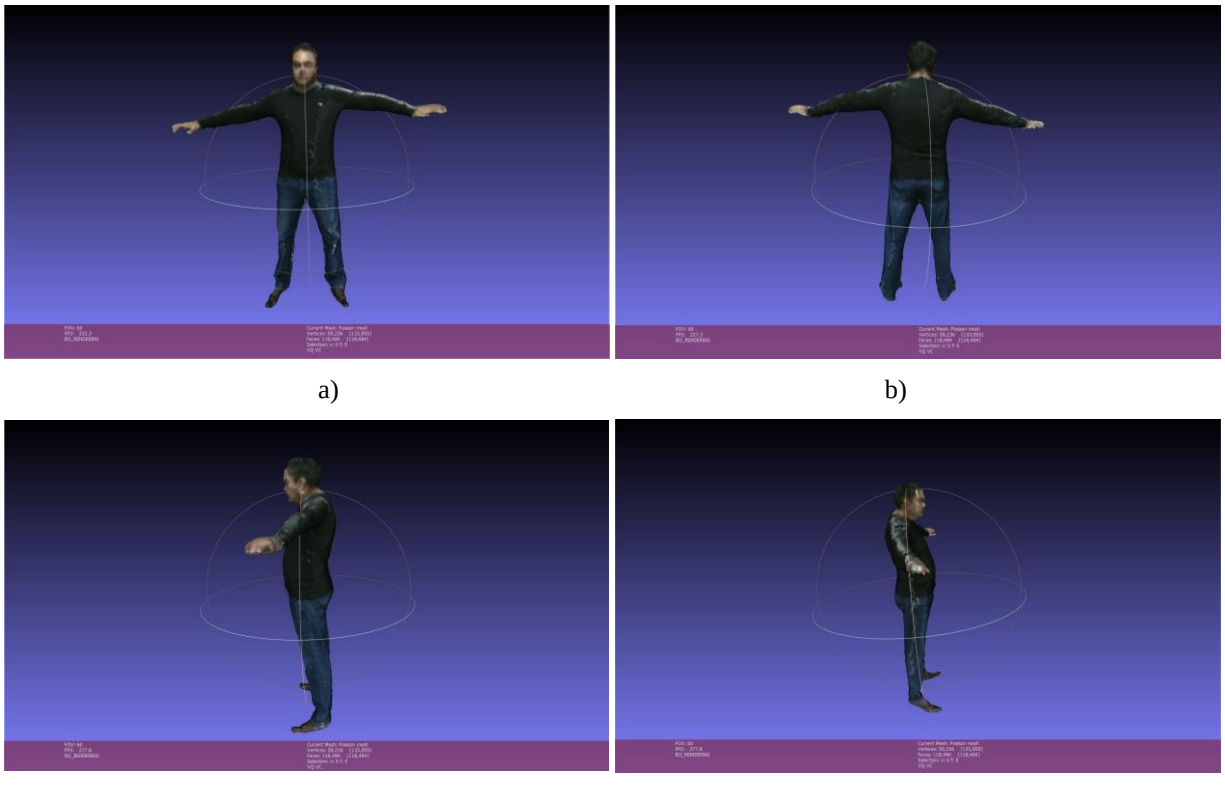


c) d)

Figura 5.3. Modelo generado con el segundo procedimiento de reconstrucción tridimensional. a) Vista frontal, b) vista posterior, c) y d) vistas laterales.

Tabla 5.4. Tercer procedimiento de reconstrucción tridimensional.

Tercer Procedimiento	
Pasos	Algoritmo/Operación
1.-Simplificación de Point Cloud	No aplica
2.-Filtrado	Erosión+Dilatación
3.- Cálculo de vectores normales a los vértices	Sin suavizado y 10 número de vecinos
4.-Creación mallado y superficies	Algoritmo de superficies por Poisson con octrees
5.-Post Procesamiento	Eliminación de partes aisladas Compactación de vértices y caras
6.-Suavizado	Suavizado de Laplace con 5 iteraciones de profundidad



a) b)

c) d)

Figura 5.4. Modelo generado con el tercer procedimiento de reconstrucción tridimensional. a) Vista frontal, b) vista posterior, c) y d) vistas laterales.

5.4 Impresión 3D de los Modelos Tridimensionales de Personas

El modelo tridimensional de la persona escaneada y exportado al formato STL adopta el nombre de *pieza* debido a que una impresora 3D no reconoce ningún patrón de objeto. El archivo con formato STL carece de información de la textura y color del modelo, solo contiene las coordenadas de los vértices.

Las condiciones para que una pieza pueda ser impresa en 3D son:

a. Las superficies de la pieza deben ser lo más uniforme posible (suavizadas).

La precisión del extrusor de la impresora no permite moldear detalladamente la geometría de la pieza u objeto, por lo que si la pieza tiene muchas uniformidades o rugosidades complicaría la manufactura generando exceso de material en algunas partes.

b. Escalamiento de la pieza.

El modelo tridimensional se realizó con base a las dimensiones originales del cuerpo de la persona y el escalamiento debe ser un ajuste para que la pieza pueda ser impresa dentro del rango de trabajo de la impresora 3D. Ya que el espacio de trabajo de una impresora 3D es limitado, se recomienda reducir la escala del modelo tridimensional de la persona de 8 al 12 % de su tamaño original.

c. Orientación de la pieza.

Este parámetro indica cómo debe colocarse la pieza en el espacio de impresión, de tal manera el extrusor requiera el mínimo número de movimientos para moldear la pieza y se eviten trayectorias innecesarias.

d. Configuración del extrusor.

Modificaciones en la temperatura de fundición, retracción y el diámetro del filamento. Por lo general, se utilizan los valores pre-establecidos en la configuración de fábrica de la impresora 3D.

e.- Relleno (infill) y estructuras de soporte.

Se define en términos de la geometría del modelo tridimensional ya que puede ser necesaria más de una pasada de material para que la pieza pueda ser terminada. Si la pieza

tiene una morfología complicada es posible imprimir estructuras de soporte activando la opción *support fill.*

5.4.1 Impresión 3D con una impresora Makerbot

Makerbot es una compañía fabricante de impresoras 3D, la cual proporciona un software de visualización y pre-procesamiento de modelos tridimensional que trabaja en conjunto con sus impresoras. Antes de imprimir el modelo tridimensional es posible definir parámetros de impresión, como el modelo de impresora a utilizar, el grosor del material, relleno de material o *infill*, distribución de la pieza a lo largo del espacio de trabajo, posición de impresión, unidades de medición, escalamiento de la pieza, temperatura de fusión del material y número de pasadas de material para la pieza. Además, el software de visualización posee un simulador y una herramienta de diagnóstico para determinar en cuánto tiempo se imprime la pieza y como el extrusor hace los recorridos de impresión. La Figura 5.5 muestra la interfaz de usuario del software Makerbot para impresión 3D, la cual acepta únicamente archivos en formato de esterolitografia (STL) y el material utilizado para todos los modelos de impresora es el ácido Poliláctico (PLA).

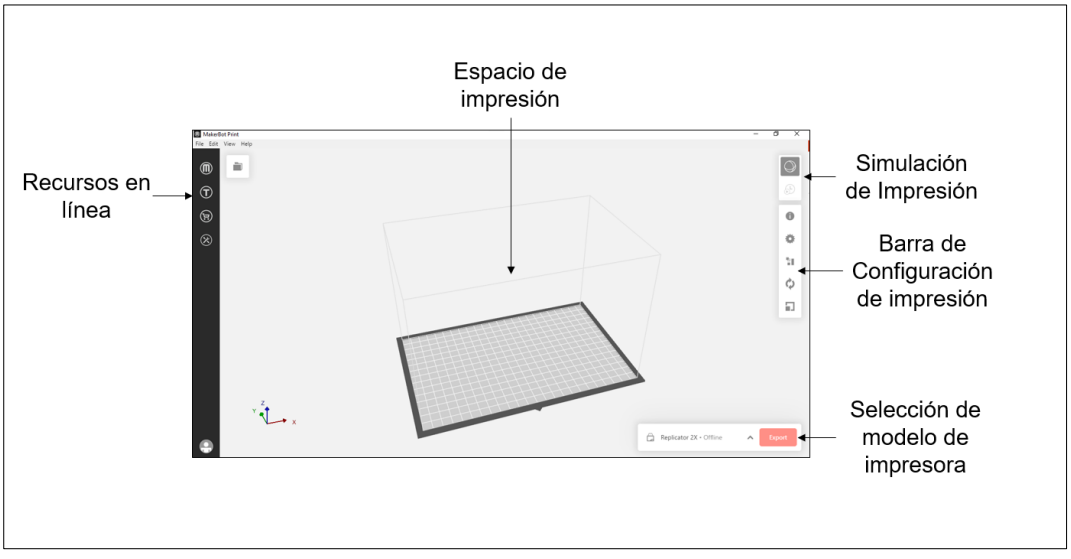


Figura 5.5. Interfaz de usuario del software Makerbot.

La Figura 5.6 muestra la impresora 3D utilizada en este proyecto, esta es una impresora Makerbot Replicator 2X con dimensiones de 49 x 52 x 53.1cms de largo, ancho y alto

respectivamente. La tecnología de impresión es por modelado por depósito de hilo fundido basado en PLA cuyo diámetro de filamento es de 1.75 milímetros de diámetro. Opera con motores de pasos de 1.8º y micropasos de 1/16".

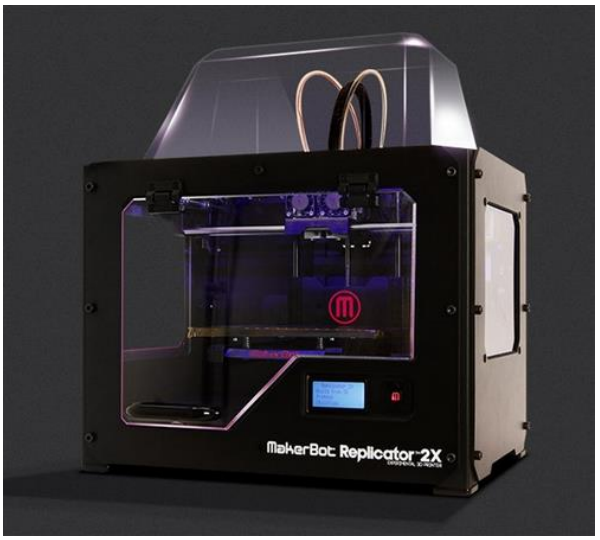


Figura 5.6. Impresora Makerbot Replicator 2X.

Se utiliza el modelo tridimensional generado con el tercer procedimiento de reconstrucción definido en la sección 5.3. Las Figura 5.7 muestra la visualización en Meshlab del modelo guardado en el archivo con formato STL y la Figura 5.8 muestra la visualización del modelo en el software de la impresora Makerbot. Se puede observar que el archivo con formato STL no tiene la información de color como se mostró en las Figuras 5.4. El tiempo estimado para imprimir el modelo de la Figura 5.8 es de 2 horas con 51 minutos utilizando 31.8 gramos de filamento PLA para un escalamiento del 8% de su tamaño real.

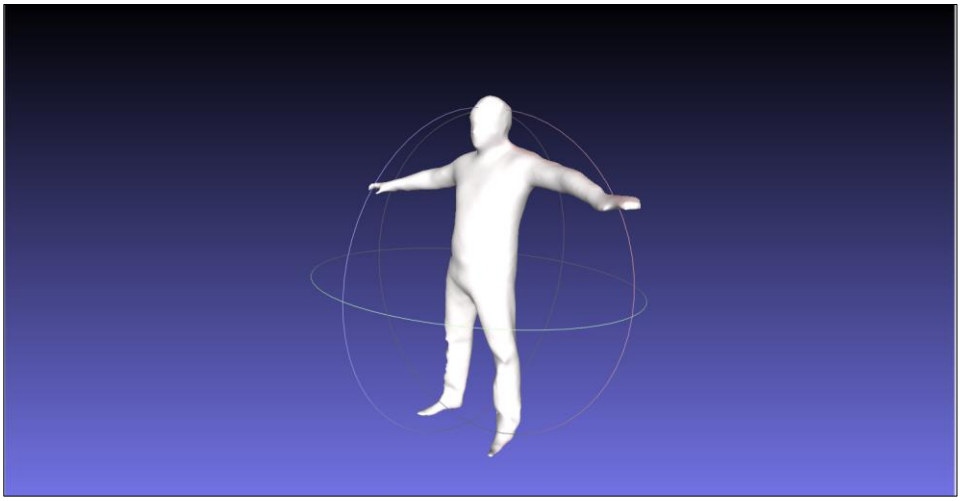


Figura 5.7. Modelo tridimensional generado por el tercer procedimiento de reconstrucción visualizado en Meshlab.

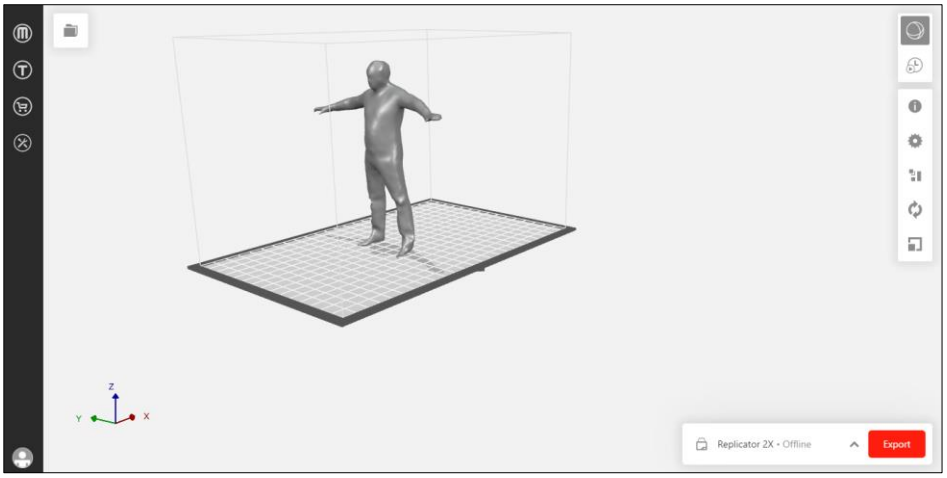


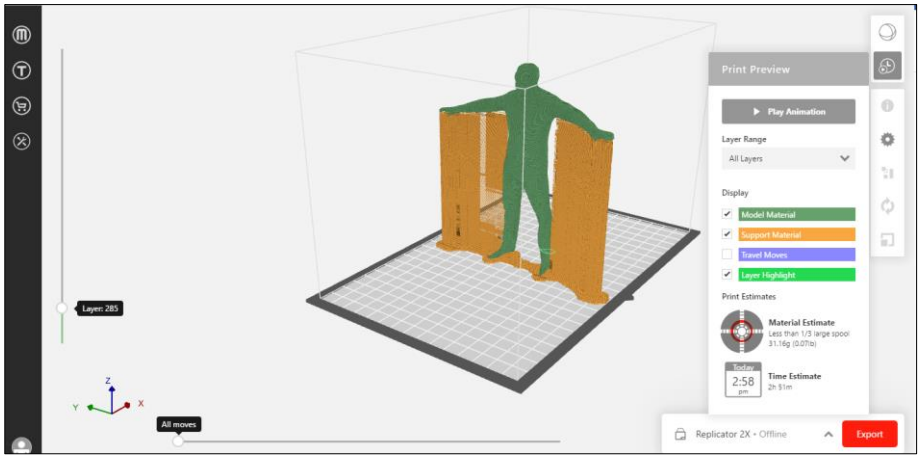
Figura 5.8. Vista del modelo escalado en el software de la impresora Makerbot.

El modelo tridimensional de la Figura 5.7 requiere del uso de una estructura de soporte ya que la posición de los brazos de la persona no permite su impresión tal como se encuentra definido. Se realizó una primera simulación de impresión, denominada configuración 1, donde el modelo tridimensional de la persona se encuentra de pie. La Tabla 5.5 muestra los parámetros definidos para la configuración 1 en el software de la impresora Makerbot. En la Figura 5.9a se puede observar el modelo tridimensional de la persona con la estructura de soporte requerida para su impresión. El modelo original se muestra en color verde y la estructura de soporte agregada por el software de la impresora Makerbot se muestra en color naranja. En la Figura 5.9b se puede observar el patrón de impresión hexagonal usado cuando

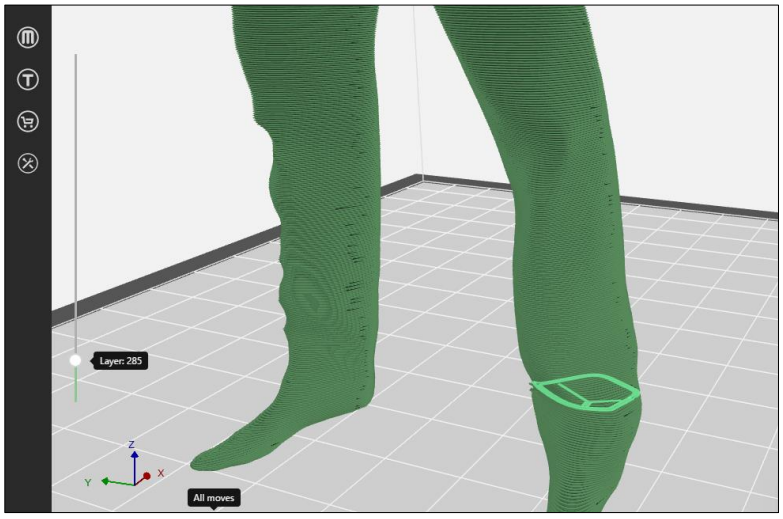
el parámetro *infill* se define en 10%. La desventaja de imprimir en estas condiciones implica mayor gasto de material utilizado en la estructura de soporte.

Tabla 5.5. Configuración 1 de la impresora 3D.

Configuración 1	
Parámetro	**Característica**
Escalamiento	8% de su tamaño original
Orientación de la Pieza	Sujeto de Pie
Extrusor	
Temperatura de extrusión	230 °C
Diámetro del filamento	1.77 mm
Velocidad de Impresión	40 mm/s
Estructuras de soporte	Aplica
Densidad de Relleno (*infill*)	10%
Patrón de Impresión	Hexagonal



a)



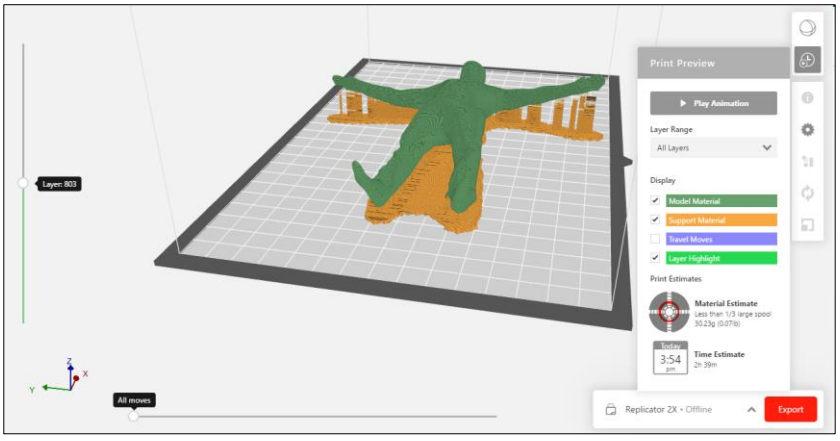
b)

Figura 5.9. Detalles de la configuración 1 de la impresora 3D. a) Vista con estructura de soporte y b) patrón de impresión hexagonal para i*nfill* de 10%.

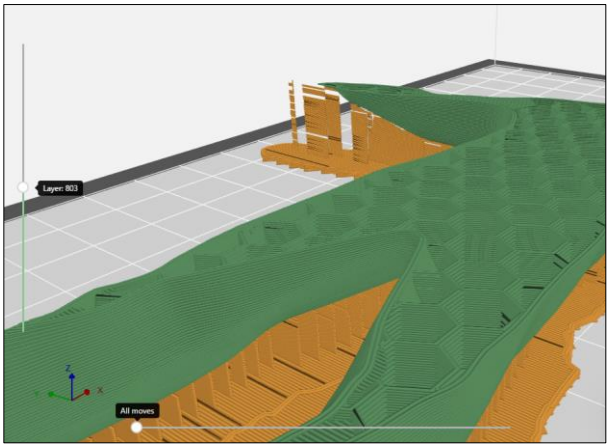
Se realizó una segunda simulación de impresión, denominada configuración 2, donde se cambió la orientación de la pieza, estando ahora el sujeto escaneado en posición horizontal y boca arriba. Se cambió el parámetro *infill* al 14% de densidad, cuyas capas serán de 0.23 mm. El patrón de impresión sigue siendo hexagonal y se utiliza una estructura de soporte. La Tabla 5.6 muestra los parámetros definidos para la configuración 2 en el software de la impresora Makerbot. La Figura 5.10 a) muestra la visualización del modelo con su estructura soporte y la Figura 5.10 b) muestra el patrón de impresión hexagonal. El tiempo estimado de impresión es de 2 horas con 39 minutos para un 8% de escalamiento de sus dimensiones originales.

Tabla 5.6. Configuración 2 de la impresora 3D.

Configuración 2	
Parámetro	**Característica**
Escalamiento	8% de su tamaño original
Orientación de la Pieza	Sujeto acostado boca arriba
Extrusor	
Temperatura de extrusión	230 °C
Diámetro del filamento	1.77 mm
Velocidad de Impresión	40 mm/s
Estructuras de soporte	Aplica
Densidad de Relleno (*infill*)	10%- 0.23 mm
Patrón de Impresión	Hexagonal



a)



b)

Figura 5.10. Detalles de la configuración 2 de la impresora 3D. a) Vista con estructura de soporte y b) patrón de impresión hexagonal para i*nfill* de 14%.

La Figura 5.11 muestra la pieza obtenida en la impresora 3D a partir del modelo tridimensional de la Figura 5.8 a). A esta pieza se le agregó una base rectangular de 4 milímetros de espesor para que pudiera ser impresa de manera vertical, también le fue retirada la estructura soporte de los brazos.

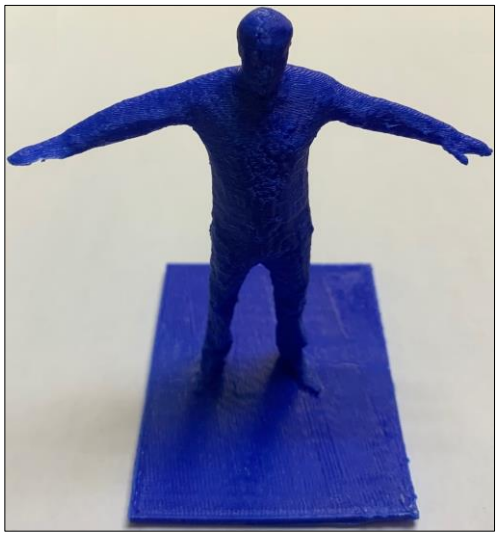


Figura 5.11. Pieza obtenida en la impresora Makerbot a partir del modelo tridimensional de una persona.

5.4.2 Impresión 3D con una impresora Ender 3 Pro

Se realizó otra prueba usando una impresora modelo Ender 3 Pro, mostrada en la Figura 5.12, cuyas características se enlistan en la Tabla 5.7. El software de simulación es diferente al de la impresora Makerbot; aunque comparten características fundamentales de impresión mencionadas en la sección 5.4. La Figuras 5.13 muestra la simulación de impresión a un 5% de la escala original con el software CURA y la Figura 5.14 muestra la pieza impresa.

Para la manufactura de la pieza debió considerarse estructuras de soporte que garanticen que la pieza sea terminada, así como la adición de una plataforma en la base del suelo pegada a los pies para que la estructura de soporte se mantenga fija durante el proceso de impresión.

Figura 5.12. Impresora Ender 3 Pro.

Tabla 5.7. Características de la impresora Ender 3 Pro.

- Dimensiones de la impresora de 440 x 440 x 465 mm
- Espacio de impresión de 220 x 200 x 250 mm
- Diámetro del extrusor de 0.4 mm pudiendo cambiarse
- Máxima velocidad de impresión de 180 mm/s
- Resolución de capa o *infill* de 0.1-0.4mm
- Tasa de cambio de temperatura de 110º C en 5 minutos
- Uso de material PLA, ABS, Madera pudiendo adaptar Láser.
- Conectividad.- Tarjeta SD, acoplamiento USB
- Pantalla de cristal líquido (LCD)
- Tecnología de material por deposición fundida.

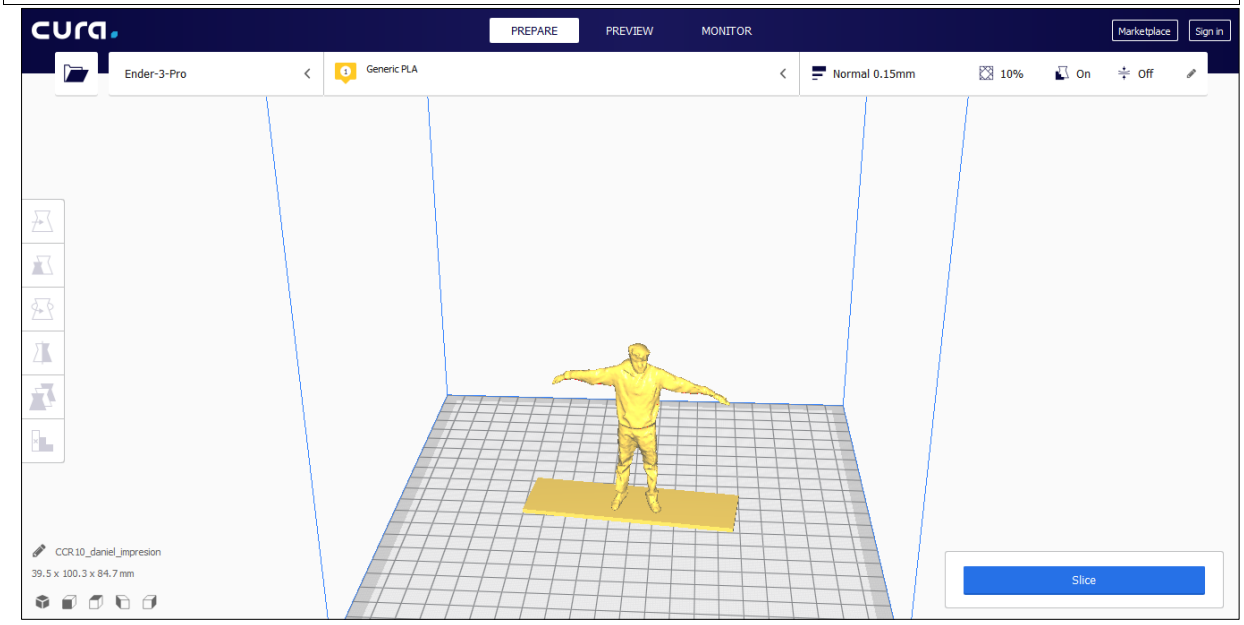


Figura 5.13. Simulación de impresión con el software CURA.

Figura 5.14. Pieza obtenida con la impresora Ender 3 Pro.

VI. CONCLUSIONES

De acuerdo a la investigación realizada para la construcción del modelo tridimensional de una persona a partir de los datos de cuatro sensores RGBD y su impresión en 3D se concluyen los siguientes aspectos:

a. De los objetivos planteados.

Se logró la construcción de un sistema de captura de datos con cuatro sensores RGBD que permite el modelado tridimensional de una persona. Los sensores utilizados son Microsoft Kinect versión 2. Se probaron dos diferentes disposiciones de los sensores dentro del área de captura; el área con geometría tipo rectángulo y el área con geometría tipo cruz, siendo este último el que mejor resultados se obtuvieron al tener menos zonas de oclusiones en la fusión de datos. Se llevó a cabo la fusión de datos tridimensionales capturados por los cuatro sensores y almacenados en forma de nube de puntos. Se llevó a cabo la reconstrucción tridimensional del modelo de la persona a partir de los datos fusionados. Este modelo fue almacenado como un archivo con formato de esterolitografia (STL) necesario para llevar a cabo su impresión en 3D.

b. Del sistema de captura de datos sincronizados con cuatro sensores RGBD.

El proceso de captura consistió en formar una red compartida entre los cuatro Equipos de Captura con un Equipo de Control. Los Equipos de Captura están conectados directamente a los sensores y el Equipo de Control les envía una señal de sincronización por medio de *sockets* para el inicio de captura y recolecta las nubes de puntos adquiridas para llevar a cabo el proceso de fusión.

Para que el sistema trabaje de forma sincronizada, se requiere que las características computacionales de los Equipos de Captura sea lo más homogénea posible en software y hardware. Se modificó el sistema de captura implementado en un proyecto previo [6] para trabajar con cuatro sensores RGBD, añadiendo la información de color RGB aparte de los datos de profundidad. Se realizó un mapeo entre los datos de profundidad y los datos de color de manera que se tiene la información de color para cada dato tridimensional incluido en la nube de puntos. Se utilizó el algoritmo propuesto en [29] para el mapeo de los datos de color y profundidad, pero se encontró que presenta una inconsistencia en la forma de extracción de las

componentes de color ya que los arreglos que guardan la información lo exponen como RGB, siendo que los arreglos debieran ser ordenados como BGR. Además, el algoritmo no contempla la segmentación del fondo de la escena y de la persona escaneada, lo que también fue contemplado en este proyecto de tesis.

Se evaluaron dos diferentes esquemas captura de datos; se les denominó como área de captura tipo rectángulo y área de captura tipo cruz por la forma en que se posicionaban los sensores. Después de realizar la alineación y calibración de los sensores se llevaron a cabo pruebas de captura y fusión de datos tridimensionales para los dos tipos de área. Se decidió utilizar el área de captura con geometría tipo cruz como la mejor alternativa ya que reduce las áreas de oclusión, aunque su proceso de calibración es más elaborado y requiere un espacio amplio para montar el equipo; por ello se recomienda que en investigaciones futuras se diseñen estructuras fijas para la colocación de los sensores que mantengan su alineación, considerando situaciones externas que puedan modificarla, facilitando el proceso de calibración.

Los datos tridimensionales capturados presentan algunas discrepancias debido a que el cabello humano genera ruido durante el proceso de adquisición ya que no refleja de manera adecuada los rayos infrarrojos.

Con base a lo anterior, es posible decir que el sistema de captura más eficiente requerirá varios sensores colocados de forma estratégica dentro del área de captura para adquirir datos completos del cuerpo humano reduciendo al mínimo las posibles oclusiones y zonas del cuerpo que no se puedan percibir.

Un área de oportunidad de este sistema se encuentra en que es posible adquirir datos tridimensionales de otros objetos diferentes al cuerpo humano. Se deberá tener en cuenta que la versión 2 de Kinect no dispondrá de mayores recursos computacionales en bibliotecas y funciones debido a que la compañía Microsoft dejó de dar soporte para dicho sensor.

c. De la fusión datos tridimensionales de cuatro sensores RGBD.

Se implementó una función denominada *data_fusion* capaz de fusionar las nubes de puntos a color provenientes de los cuatro Equipos de Captura y de acuerdo a la geometría del área de captura, con el objetivo de generar el modelo completo del cuerpo de la persona escaneada. La nube de puntos fusionada fue almacenada con formato de visualización tridimensional

poligonal (PLY) para su posterior procesamiento con el software de reconstrucción tridimensional.

Se trató a la nube de puntos de cada sensor como una matriz de (*n x 6)* en donde *n* representa el número de puntos de la nube y 6 son las columnas que representan las características de cada punto; las tres primeras columnas son las coordenadas *xyz* y las últimas tres columnas son los componentes de color RGB.

d. Del proceso de reconstrucción tridimensional.

Se llevó a cabo el proceso de reconstrucción tridimensional utilizando el software Meshlab. Se utilizó la nube de puntos en formato PLY con los datos fusionados de la persona escaneada, se realizó un filtrado previo, se calcularon los vectores normales a los vértices y se aplicó el algoritmo de reconstrucción para la generación del modelo tridimensional. Se aplicaron funciones para el suavizado de las superficies del modelo y se almacenó en un archivo con formato STL. Se utilizó este formato debido a que es aceptado por la mayoría de las impresoras 3D en el mercado.

El proceso de reconstrucción tridimensional no sigue un método específico, depende mucho de la información contenida en la nube de puntos, inclusive es posible aplicar dos o más algoritmos para perfeccionar al modelo tridimensional generado. En este proyecto se utilizó el algoritmo de reconstrucción de Poisson y el algoritmo por pivoteo de pelota siendo el primero más eficiente ya que cubre los huecos para obtener un modelo sólido listo para su impresión en 3D.

La creación del modelo tridimensional de la persona debe forzosamente apoyarse en un hardware y software apto para tratar el tiempo de procesamiento computacional de los algoritmos de reconstrucción tridimensional.

e. Del proceso de impresión tridimensional.

Se utilizó el software de la impresora 3D de la compañía Makerbot para procesar el archivo con formato STL que contiene el modelo tridimensional de la persona escaneada. Primero se debe modificar la escala ya que el modelo fue creado con las dimensiones reales de la persona y debe ajustarse a los límites del área de impresión. Se utilizó una impresora Makerbot Replicator 2X por lo que se redujo la escala con factor de aproximadamente 10% del tamaño original. Se realizó el ajuste de otros parámetros de la impresora 3D como el

material de impresión, temperatura de impresión, capas y pasadas del extrusor a lo largo de la pieza, posición, estructuras de soporte entre otros. En otra prueba se utilizó el software Cura para preparar el modelo e imprimirlo en una impresora Ender 3 Pro utilizando parámetros similares a los descritos para la impresora Makerbot.

Se detectan principalmente dos áreas de trabajo a futuro relacionados a los sistemas de escaneo del cuerpo completo de una persona, primero es la construcción de un sistema de captura con estructuras fijas para sujetar los sensores y reducir los problemas de calibración y en segundo lugar sería seguir investigando los diversos algoritmos de reconstrucción para la obtención de un mejor modelo tridimensional de la persona escaneada.

Finalmente, con el desarrollo del este proyecto de tesis permitió la incursión en diferentes áreas del conocimiento como la Computación, Electrónica, Mecánica y Manufactura incrementado de manera importante la comprensión de conceptos y técnicas necesarias para un desarrollo exitoso en estas áreas.

BIBLIOGRAFÍA

[1] I. Robledo Vega, *Visión por Computadora Parte 3 Visión en Estéreo.* Instituto Tecnológico de Chihuahua, 2017.

[2] S. Paquette, "3D scanning in apparel design and human engineering," in *IEEE Computer Graphics and Applications*, vol. 16, no. 5, pp. 11-15, Sept. 1996.

[3] 3D Systems, "3D Systems" 2019. Disponible: https://es.3dsystems.com/our-story.

[4] R. Pérez, "Tecnoimpre3D", *Tecnoimpre3D*, 10 Agosto 2015. [En línea]. Disponible: http://tecnoimpre3d.com/4-tecnologias/.

[5] C. Escobar, " Tipos de Impresoras 3D", Impresoras3D.com, 2 septiembre 2016. [En línea]. Disponible URL: https://impresoras3d.com/blogs/noticias/102883975-tipos-de-impresoras-3d. [Último acceso: 6 octubre 2017].

[6] V. Velasco, *Fusión de Datos de Dos sensores RGBD para generar un modelo de 360 grados del movimiento de una persona,* Chihuahua: División de Estudios Profesionales del Instituto Tecnológico de Chihuahua, 2016.

[7] L. Shapiro y G. Stockman, Computer Vision, New Jersey: Prentice Hall, 2001.

[8] V. C. Castillo, "Emaze", emze, 2015. El Sentido de la Vista. Disponible URL: https://www.emaze.com/@AITLORTT.

[9] E. Trucco, A. Verri Introductory Techiques for 3-D Computer Vision, EE.UU: Prentice Hall, 1998.

[10] P. Fankhauser, M. Bloesch, D. Rodriguez, R. Kaestner, M. Hutter and R. Siegwart, "Kinect v2 for mobile robot navigation: Evaluation and modeling," *2015 International Conference on Advanced Robotics (ICAR)*, Istanbul, 2015, pp. 388-394.

[11] A. Jana, Kinect For Windows SDK Programming Guide, Birmingham: Packt Publishing, 2012.

[12] F. Acuna, D. Rivas, S. Chancusi and P. Navarrete, "Design and Construction of a 3D Printer Auto Controller Wirelessly Through of Free Software," in *IEEE Latin America Transactions*, vol. 13, no. 6, pp. 1893-1898, June 2015.

[13] J. V. Molina, Caracterización de Materiales Termoplasticos de ABS y PLA semí rígido impresos en 3D con cinco mallados internos diferentes, Quito: Escuela Politécnica Nacional, 2016, p. 23.

[14] J.V. Rutkowski, B.C Levin, "Acrylonitrile-Butadiene- Styrene Copolymers and Combustion Products and their Toxicity", *Fire and Materials,* vol. 10, pp. 93-105, 1986

[15] C. Barnatt, 3D Printing, EE.UU: Createspace Independent Publishing Platform, 2016.

[16] B. Evans, Practical 3D Printers "The Science and Art of 3D Printing", EE.UU: Apress, 2012.

[17] K. McHenry, P. Bajcsy "An Overview of 3D Data Content, File Formats and Viewers", National Center for Supercomputing Applications, University of Illinois, Illinois, 2008.

[18] H. Daanen, G. J van de Water "Whole Body Scanners", *Displays,* nº 19, pp. 111-120, 1998.

[19] H. Daanen, F.B Ter Haar "3D Whole Body Scanners Revisited", *Displays,* nº 34, pp. 270-275, 2013.

[20] L. Dekker, I. Douros, B. F. Buston and P. Treleaven, "Building symbolic information for 3D human body modeling from range data," *Second International Conference on 3-D Digital Imaging and Modeling (Cat. No.PR00062),* Ottawa, Ontario, Canada, 1999, pp. 388-397

[21] B. Allen, B. Curless y Z. Popovic, "The space of human body shapes:reconstruction and parameterization from range scans" *ACM Transactions on Graphics (TOG),* vol. 22, p. 587-594, 2003.

[22] V. Blanz , T. Vetter, "Face recognition based on fitting a 3D morphable model," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 9, pp. 1063-1074, Sept. 2003

[23] S. Marschener,B. Guenter, S. Raghupathy, "Modeling and Rendering for Realistic Facial Animation" *Proceedings Rendering Techniques* , pp. 231-242, 2000.

[24] J. Feldmar, N. Ayache "Rigid and Aine Registration of Smooth Surfaces using Differential Propieties" *Proceedings of the Third European Conference,* vol. Volumen 2, pp. 397-406, 1994.

[25] M. Kazhdan, M. Bolitho, H. Hoppe "Poisson Surface Reconstruction" de *Eurographics Symposium on Geometry Processing*, Estados Unidos, 2009, pp 61-70

[26] J F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva and G. Taubin, "The Ball-Pivoting Algorithm for Surface Reconstruction," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 4, pp. 349-359, Oct.-Dec. 1999.

[27] J. Tong, J. Zhou, L. Liu, Z. Pan and H. Yan, "Scanning 3D Full Human Bodies Using Kinects," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 4, pp. 643-650, April 2012

[28] V. H. Velasco, E.G Juárez, I.R Vega, A. Pacheco , "Comparación de Mediciones de Profundidad entre un Sistema de Visión en Estéreo y un sensor RGBD", de *Congreso Internacional de Ingeniería Electrónica*, Chihuahua, pp. 187-192, 2015.

[29] J. Treven, D.M. Córdova Esparza, "Kin2. A Kinect 2 toolbox for Matlab", *Science of Computer Programming*,vol. 130, nº 30, pp. 97-106, 2016.

[30] S. Chapra, R.P. Canale, Métodos Numéricos para Ingenieros, Mc. Graw Hill, 2007.

[31] M. Corsini, P. Cignoni and R. Scopigno, "Efficient and Flexible Sampling with Blue Noise Properties of Triangular Meshes," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 6, pp. 914-924, June 2012.

[32] L. Liu, C.L Tai, Z. Ji, G. Wang, "Non-iteative Global Mesh Smoothing with Feature Preservation" *Computer-Aided Design*, vol. 39, nº 9, pp. 772-782, 2007.

[33] I. Corporation, *Nuc5i5RYK Product Brief*, Estados Unidos: Intel, 2015.

ANEXOS

Anexo 1. Código de Software para Calibración de Sensores RGBD. (C#/xaml)

```
//------------------------------------------------------------------------------------------------------------
//      Proyecto:CalibracionPlano
//      Archivo:MainWindow.xaml.cs
//      Modificado y última versión por: Ing. Rodríguez Salgado Daniel
//      Versión original: M.C. Victor Hugo Velasco, Ing. Julio Omar Vega,
//      Dr. Isidro Robledo Vega y SDK Microsoft Kinect
//      Última modificación: Enero-Junio 2019
//
//      Esta aplicación aprovecha los flujos de datos de color
//      y de profundidad que entrega el sensor
//      Kinect V2 para realizar una calibración del sistema
//      de captura de datos sincronizada.
//      Contiene funciones que detectan dos círculos de color verde
//      y azul usando la cámara a color
//      y muestra la distancia hacia estos desde ambos sensores para alinear sus planos
//
//      Entrada: Imágenes a color y de profundidad
//      Salida: Distancia a lo largo del eje z del sensor en milímetros
//              de los colores azul y verde segmentados
//------------------------------------------------------------------------------------------------------------
namespace KinectCapture
{
    //Bibliotecas
    using System;
    using System.ComponentModel;
    using System.Diagnostics;
    using System.Globalization;
    using System.IO;
    using System.Text;
    using System.Windows;
    using System.Windows.Media;
    using System.Windows.Media.Imaging;
    using Microsoft.Kinect;


    /// <summary>
    /// Lógica de interacción para MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window, INotifyPropertyChanged
    {
        /// <summary>
        /// Tamaño en bytes del pixel RGB en el mapa de bits
        /// </summary>
        private readonly int bytesPerPixel = (PixelFormats.Bgr32.BitsPerPixel + 7) / 8;

        /// <summary>
        /// Constante de conversión del rango de profundidad al rango de un byte
        /// </summary>
        private const int MapDepthToByte = 8000 / 256;
```

```
        /// <summary>
        /// Sensor Kinect Activo
        /// </summary>
        private KinectSensor kinectSensor = null;

        /// <summary>
        /// Almacenamiento temporal para el mapeo de color a profundidad
        /// </summary>
        private DepthSpacePoint[] colorMappedToDepthPoints = null;

        /// <summary>
        /// Lector múltiple para los cuadros de profundidad y color
        /// </summary>
        private MultiSourceFrameReader multiFrameSourceReader = null;

        /// <summary>
        /// Conversión de coordenadas de pixeles de color a profundidad.
        /// </summary>
        private CoordinateMapper coordinateMapper = null;

        /// <summary>
        /// Mapas de bits para mostrar en pantalla imágenes de color y profundidad
        /// </summary>
        private WriteableBitmap depthBitmap = null;
        private WriteableBitmap colorBitmap = null;

        /// <summary>
        ///  Almacenamiento temporal para los pixeles de profundidad
        /// </summary>
        private byte[] depthPixels = null;

        /// <summary>
        ///  Almacenamiento temporal para los pixeles de color
        /// </summary>
        private byte[] colorPixels = null;

        /// <summary>
        /// Almacenamiento temporal para imágenes a color y de profundidad
        /// </summary>
        private ushort[][] depthData = null;

        /// <summary>
        /// Texto de estado a mostrar a pantalla
        /// </summary>
        private string statusText = null;

        // Variables del valor de profundidad Verde (G) y Azul (B)
        private int depthIndexG = 0;
        private int depthIndexB = 0;
        private bool distancia = true;

    //Constructor de la ventana principal
        public MainWindow()
```

```
        {
            // Se inicializa el sensor para captura
            this.kinectSensor = KinectSensor.GetDefault();

              this.multiFrameSourceReader =
              this.kinectSensor.OpenMultiSourceFrameReader(FrameSourceTypes.Depth | FrameSourceTypes.Color
              );

              this.multiFrameSourceReader.MultiSourceFrameArrived += this.Reader_MultiSourceFrameArrived;

            this.coordinateMapper = this.kinectSensor.CoordinateMapper;

            //Obtiene las características del cuadro de profundidad
               FrameDescription depthFrameDescription = this.kinectSensor.DepthFrameSource.FrameDescription;

            int depthWidth = depthFrameDescription.Width;
            int depthHeight = depthFrameDescription.Height;

            // Asigna espacio en memoria para los pixeles de profundidad
            this.depthPixels = new byte[depthWidth * depthHeight];

            // Define el mapa de bits para datos de profundidad
               this.depthBitmap = new WriteableBitmap(depthWidth, depthHeight, 96.0, 96.0, PixelFormats.Gray8,
              null);

              FrameDescription colorFrameDescription = this.kinectSensor.ColorFrameSource.FrameDescription;

            int colorWidth = colorFrameDescription.Width;
            int colorHeight = colorFrameDescription.Height;

              this.colorMappedToDepthPoints = new DepthSpacePoint[colorWidth * colorHeight];

              this.colorBitmap = new WriteableBitmap(colorWidth, colorHeight, 96.0, 96.0, PixelFormats.Bgra32,
              null);

            // Asigna espacio en memoria para los pixeles de color
            this.colorPixels = new byte[colorFrameDescription.LengthInPixels*4 ];
            //Muestra el estado del sensor en la ventana
            this.kinectSensor.IsAvailableChanged += this.Sensor_IsAvailableChanged;
            this.kinectSensor.Open();

              this.StatusText = this.kinectSensor.IsAvailable ? Properties.Resources.RunningStatusText
                                            : Properties.Resources.NoSensorStatusText;

            this.DataContext = this;

            this.InitializeComponent();

        }
    /// <summary>
    /// Se define el manejador del evento de cambio en las propiedades
    /// </summary>
        public event PropertyChangedEventHandler PropertyChanged;
```

```
/// <summary>
/// Función que obtiene el mapa de bits de color para mostrar en pantalla
/// </summary>
        public ImageSource ImageSource1
        {
            get
            {
                return this.colorBitmap;
            }
        }

        /// <summary>
        ///Función que obtiene el mapa de bits de profundidad para mostrar en pantalla
        /// </summary>
        public ImageSource ImageSource2
        {
            get
            {
                return this.depthBitmap;
            }
        }


         /// <summary>
        /// Función que muestra el estado actual de la aplicación
        /// </summary>
        public string StatusText
        {
            get
            {
                return this.statusText;
            }

            set
            {
                if (this.statusText != value)
                {
                    this.statusText = value;

                    if (this.PropertyChanged != null)
                    {
                                                this.PropertyChanged(this, new PropertyChangedEventArgs("Estado de
                                Texto"));
                    }
                }
            }
        }

        /// <summary>
        /// Función que se ejecuta al cerrar la aplicación
        /// </summary>
        /// <param name="sender">object sending the event</param>
        /// <param name="e">event arguments</param>
        private void MainWindow_Closing(object sender, CancelEventArgs e)
```

```
        {
            // Cierra el lector múltiple de cuadros de color y profundidad
            if (this.multiFrameSourceReader != null)
            {

                this.multiFrameSourceReader.Dispose();
                this.multiFrameSourceReader = null;
            }

            // Cierra la conexión al sensor
            if (this.kinectSensor != null)
            {
                this.kinectSensor.Close();
                this.kinectSensor = null;
            }
        }

/// <summary>
/// Función que se ejecuta al oprimir el botón “Captura” para mostrar en pantalla
/// de forma dinámica la lectura de distancias
/// </summary>
/// <param name="sender">object sending the event</param>
/// <param name="e">event arguments</param>
        private void ScreenshotButton_Click(object sender, RoutedEventArgs e)
        {
            this.distancia = true;
        }

        /// <summary>
        ///  Función que procesa la llegada de cuadros de color y profundidad del sensor.
        /// </summary>
        /// <param name="sender">object sending the event</param>
        /// <param name="e">event arguments</param>
             private void Reader_MultiSourceFrameArrived(object sender, MultiSourceFrameArrivedEventArgs e)
        {
            //Define e inicializa las variables y objetos locales
            int depthWidth = 0;
            int depthHeight = 0;
            int colorWidth = 0;
            int colorHeight = 0;
            DepthFrame depthFrame = null;
            ColorFrame colorFrame = null;

            // Define los candados para los mapas de bits de color y profundidad
            bool isDepthBitmapLocked = false;
            bool isColorBitmapLocked = false;

            // Adquiere los cuadros de color y profundidad con el lector múltiple
            MultiSourceFrame multiSourceFrame = e.FrameReference.AcquireFrame();

            // Si los cuadros leídos están vacíos sale de la función
            if (multiSourceFrame == null)
            {
                return;
```

```
}

// Se utiliza "try" para asegurar de limpiar memoria antes de salir de la función,
// esto incluye llamar a "Dispose" para desechar objetos de cuadros que se obtengan
// y desbloquear el búfer de los mapas de bits
try
{
   // Adquiere los cuadros de profundidad y color
   depthFrame = multiSourceFrame.DepthFrameReference.AcquireFrame();
   colorFrame = multiSourceFrame.ColorFrameReference.AcquireFrame();

   // Si alguno de los cuadros está vacío sale de la función
   if ((depthFrame == null) || (colorFrame == null))
   {
      return;
   }

   // Obtiene las características del cuadro de profundidad
   FrameDescription depthFrameDescription = depthFrame.FrameDescription;
   depthWidth = depthFrameDescription.Width;
   depthHeight = depthFrameDescription.Height;

   // Bloquea el mapa de bits de profundidad para escritura
   this.depthBitmap.Lock();
   isDepthBitmapLocked = true;

   bool depthFrameProcessed = false;

   using (KinectBuffer depthBuffer = depthFrame.LockImageBuffer())
   {
      //Verifica los datos y muestra la imagen de profundidad
              if (((depthWidth * depthHeight) == (depthBuffer.Size /
            depthFrameDescription.BytesPerPixel)) &&
            (depthFrameDescription.Width == this.depthBitmap.PixelWidth) &&
            (depthFrameDescription.Height == this.depthBitmap.PixelHeight))
      {
         // Nota: Para lograr ver todo el alcance de profundidad
         // se establece "MaxDepth" al umbral máximo
         ushort maxDepth = ushort.MaxValue;

            this.ProcessDepthFrameData(depthBuffer.UnderlyingBuffer, depthBuffer.Size,
            depthFrame.DepthMinReliableDistance, maxDepth, depthFrameDescription.BytesPerPixel);
         depthFrameProcessed = true;

         // Mapea las coordenadas de los pixeles de color a los datos de profundidad
         this.coordinateMapper.MapColorFrameToDepthSpaceUsingIntPtr(
         depthBuffer.UnderlyingBuffer,
         depthBuffer.Size,
         this.colorMappedToDepthPoints);
      }
   }


     // Al terminar se desecha el cuadro de profundidad
```

```
depthFrame.Dispose();
depthFrame = null;

// Obtiene las características del cuadro de color
FrameDescription colorFrameDescription = colorFrame.FrameDescription;
colorWidth = colorFrameDescription.Width;
colorHeight = colorFrameDescription.Height;

// Bloquea el mapa de bits de color para escritura
this.colorBitmap.Lock();
isColorBitmapLocked = true;

// Verifica los datos de color y los muestra en pantalla
using (KinectBuffer colorBuffer = colorFrame.LockRawImageBuffer())
{
            if ((colorFrameDescription.Width == this.colorBitmap.PixelWidth) &&
          (colorFrameDescription.Height == this.colorBitmap.PixelHeight))
   {
          colorFrame.CopyConvertedFrameDataToArray(colorPixels, ColorImageFormat.Bgra);

       this.colorBitmap.WritePixels(
          new Int32Rect(0, 0, this.colorBitmap.PixelWidth, this.colorBitmap.PixelHeight),
       colorPixels,
       this.colorBitmap.PixelWidth * 4, 0);
   }
}
  unsafe
{
   // Cálculo de coordenadas de color a profundidad
           int colorMappedToDepthPointCount = this.colorMappedToDepthPoints.Length;

         fixed (DepthSpacePoint* colorMappedToDepthPointsPointer =
         this.colorMappedToDepthPoints)
   {
      // Maneja los datos de color como pixeles de 4 bytes
      uint* bitmapPixelsPointer = (uint*)this.colorBitmap.BackBuffer;

      // Identifica y analiza pixel por pixel en la imagen de color
          for (int colorIndex = 0; colorIndex < colorPixels.Length /4; colorIndex =    colorIndex +4)
      {
                    float colorMappedToDepthX = this.colorMappedToDepthPoints[colorIndex].X;
                    float colorMappedToDepthY = this.colorMappedToDepthPoints[colorIndex].Y;

         // Busca valores diferentes a -inf en X y Y  en los pixeles mapeados
                     if (!float.IsNegativeInfinity(colorMappedToDepthX) &&
                    !float.IsNegativeInfinity(colorMappedToDepthY))
         {
                     // Obtiene las coordenadas en X y Y en el espacio de profundidad
                       // correspondientes al pixel actual en el espacio de color
                     int depthX = (int)(colorMappedToDepthX + 0.5f);
                     int depthY = (int)(colorMappedToDepthY + 0.5f);

            // Asegura que el valor en profundidad corresponde a un punto
            // válido en el espacio de color .
```

```
                    if ((depthX >= 0) && (depthX < depthWidth) && (depthY >= 0) && (depthY <
                    depthHeight))
                {
                    //Proceso de segmentacion [R G B]
                    //Segmentando el color verde
                    if (colorPixels[colorIndex*4] <95 )
                        if (colorPixels[colorIndex*4 +1] > 95 )
                            if (colorPixels[colorIndex*4 + 2] < 55)
                            {
                                // Toma el valor de profundidad del color verde
                                this.depthIndexG = (depthY * depthWidth) + depthX;
                            }
                     // Segmentando al color azul
                            if (colorPixels[colorIndex * 4] < 86 && colorPixels[colorIndex * 4] > 80)
                                if (colorPixels[colorIndex * 4 + 1] < 35 && colorPixels[colorIndex * 4 + 1] > 27 )
                                    if (colorPixels[colorIndex * 4 + 2] < 17 && colorPixels[colorIndex * 4 + 2]
                                    >10)
                            {
                                //Toma el valor de profundidad del color azul
                                this.depthIndexB = (depthY * depthWidth) + depthX;
                            }
                }
            }
        }
    }
  }
 // Se destruye el cuadro de color
   colorFrame.Dispose();
   colorFrame = null;

    // Renderizado de Pixeles
    if (depthFrameProcessed)
    {
        this.RenderDepthPixels();
    }

}

finally
{
    // Desbloquea los búferes y desecha los cuadros no vacíos.
    if (isDepthBitmapLocked)
    {
        this.depthBitmap.Unlock();
    }

    if (isColorBitmapLocked)
    {
        this.colorBitmap.Unlock();
    }

    if (depthFrame != null)
    {
        depthFrame.Dispose();
```

```
        }

        if (colorFrame != null)
        {
            colorFrame.Dispose();
        }

    }
}


/// <summary>
/// Procesa los datos de profundidad para mostrar los puntos de referencia azul y verde
/// detectados con su valor en profundidad correspondiente.
/// </summary>
/// <param name="depthFrameData">Apuntador al cuadro de datos de profundidad</param>
/// <param name="depthFrameDataSize">Tamaño del cuadro de datos de profundidad</param>
/// <param name="minDepth">Valor de profundidad mínimo confiable</param>
/// <param name="maxDepth">Valor de profundidad máximo confiable</param>

 private unsafe void ProcessDepthFrameData(IntPtr depthFrameData, uint depthFrameDataSize, ushort
 minDepth, ushort maxDepth, uint depthBytesPerPixel)
{
    // Datos del cuadro de datos de profundidad en valores de 16 bits
    ushort* FrameData = (ushort*)depthFrameData;

    // Convierte los datos de profundidad a una representación visual
    for (int i = 0; i < (int)(depthFrameDataSize / depthBytesPerPixel); ++i)
    {
        // Obtiene la profundidad de cada pixel
        ushort depth = FrameData[i];

        // Para convertir a byte, se mapea la rofundidad al rango de byte
       this.depthPixels[i] = (byte)(depth >= minDepth && depth <= maxDepth ? (depth / MapDepthToByte) :
      0);
    }

    // Modifica los pixeles donde se encontraron los puntos de referencia azul y verde
    // para mostrar puntos blancos
    this.depthPixels[depthIndexG] = 255;
    this.depthPixels[depthIndexB] = 255;
    if (this.distancia == true)
    {
        //Muestra los valores de profundidad de ambos puntos de referencia
        LblDepthG.Content = FrameData[depthIndexG];
        LblDepthB.Content = FrameData[depthIndexB];
    }
}

/// <summary>
/// Función que pinta el mapa de bits de profundidad en pantalla.
/// </summary>
private void RenderDepthPixels()
{
```

```
            this.depthBitmap.WritePixels(
              new Int32Rect(0, 0, this.depthBitmap.PixelWidth, this.depthBitmap.PixelHeight),
             this.depthPixels,
             this.depthBitmap.PixelWidth,
               0);
        }
        /// <summary>
        /// Función que maneja el evento cuando el sensor no se encuentra disponible
        /// </summary>
        /// <param name="sender">Objeto enviado al evento</param>
        /// <param name="e">argumentos del evento</param>
              private void Sensor_IsAvailableChanged(object sender, IsAvailableChangedEventArgs e)
        {
             this.StatusText = this.kinectSensor.IsAvailable ? Properties.Resources.RunningStatusText
                                                             : Properties.Resources.SensorNotAvailableStatusText;
        }
    }
}
```

Anexo 2. Código de Software para los Equipos de Captura. Adquisición de Nube de Puntos a Color. (C#/xaml)

```
//------------------------------------------------------------------------------------------------------
//      Proyecto: KinectColorCapture
//      Archivo:MainWindow.xaml.cs
//      Modificado y última versión por: Ing. Rodríguez Salgado Daniel
//      Versión original: M.C. Velasco Víctor Hugo, Ing. Julio Omar Vega,
//      Dr. Isidro Robledo Vega, SDK Microsoft Kinect (CoordinateMapping.xaml.cs)
//      Última modificación: Enero-Junio 2019
//
//      El siguiente código se encuentra en cada uno de los Equipos de Captura
//      y se encarga de realizar la adquisición
//      de la nube de puntos a color de la persona escaneada
//      en formato (x,y,z,R,G,B).
//
//      La conexión entre los Equipos de Captura y el Equipo de Control
//      deberá pertenecer al mismo grupo hogar
//
//     Entrada: Imágenes a color y de profundidad. Lista de comandos de ejecución
//
//      Salida: knectpointcloud-n.txt (x,y,z) en el marco de referencia
//      del sensor Kinect V2
//      Color-n.txt (R,G,B)   Componentes de color R(Rojo) G( Verde) y B(Azul)
//------------------------------------------------------------------------------------------------------
namespace KinectCapture
{
   //Bibliotecas
   using System;
   using System.ComponentModel;
   using System.Diagnostics;
   using System.Globalization;
   using System.IO;
   using System.Text;
   using System.Windows;
   using System.Windows.Media;
   using System.Windows.Media.Imaging;
   using Microsoft.Kinect;
   using System.Threading;
   using System.Net;
   using System.Net.Sockets;
   using System.Collections;

   /// Lógica de interacción para MainWindow.xaml
   public partial class MainWindow : Window, INotifyPropertyChanged
   {
      /// Tamaño en bytes del pixel RGB en el mapa de bits
      private readonly int bytesPerPixel = (PixelFormats.Bgr32.BitsPerPixel + 7) / 8;

      /// Constante de conversión del rango de profundidad al rango de byte
      private const int MapDepthToByte = 8000 / 256;
```

```
        /// Sensor Kinect Activo
        private KinectSensor kinectSensor = null;

            /// Variables contadoras para el número de imágenes y banderas de término de   procesos
        private bool captureFrames = false;
        private const int FramesToStore = 5; //Número de muestras
        private int FramesCaptured = 0;
        private int size=0;

        /// Lector para los cuadros de las imágenes de profundidad y color
        /// </summary>
        private MultiSourceFrameReader multiFrameSourceReader = null;

            /// Variables de Mapeo entre el espacio de profundidad, color y marco de referencia de la cámara
        private CoordinateMapper coordinateMapper = null; // Para extracción de coordenadas
        private CameraSpacePoint[] cameraPoints; // Para mapeo al marco de referencia de la   cámara
        private ColorSpacePoint[] colorPoints; // Para el mapeo al espacio de color

/ // Mapas de bits para guardar imágenes. Se utilizaran tres imágenes; de profundidad, a color y a color como
arreglo
        private WriteableBitmap depthBitmap = null;
        private WriteableBitmap colorBitmap = null;
        private WriteableBitmap[] colorData = null;

        /// Arreglo Intermedio para guardar pixeles de profundidad
        private byte[] depthPixels = null;

            /// Arreglos intermedios para guardar datos de profundidad, de color y de nube de  puntos
        /// </summary>
        private ushort[] depthData = null;
        private byte[] colorFrameData = null;
        private float[] pointCloud = null;
        private int[] colors = null;

        // Variables para la creación de los archivos de salida
        string xValue = null;
        string yValue = null;
        string zValue = null;
        string Red = null;
        string Green = null;
        string Blue = null;
        string SavePath = null;

        /// Texto de estado para mostrar en pantalla
        private string statusText = null;

        /// Variables de propios de los Buffers de Sockets
        private byte[] m_byBuff = new byte[70];
        private Socket listener;
        private Socket Gsocket;

    public MainWindow()
        {
```

```
// Configuración de Sockets
const int nPortListen = 399;
//Dirección IP de cada Equipo de Captura:
//  NUC1: 10.6.2.47
//  NUC2: 10.6.2.48
//  NUC3: 10.6.2.49
//  NUC4: 10.6.2.50

//Asignar la IP al Equipo de Captura
IPAddress aryLocalAddr = new IPAddress(new byte[] { 10, 6, 2, 68 });

//Crea el servidor de Sockets
   this.listener = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
this.listener.Bind(new IPEndPoint(aryLocalAddr, nPortListen));
this.listener.Listen(10);
this.listener.BeginAccept(new AsyncCallback(OnConnectRequest), this.listener);

  // Se inicializa sensor para inicializacion de camara de color, profundidad, bodyIndex y pointCloud
this.kinectSensor = KinectSensor.GetDefault();
   this.multiFrameSourceReader =
  this.kinectSensor.OpenMultiSourceFrameReader(FrameSourceTypes.Depth | FrameSourceTypes.Color |
  FrameSourceTypes.BodyIndex);

  this.multiFrameSourceReader.MultiSourceFrameArrived += this.Reader_MultiSourceFrameArrived;
this.coordinateMapper = this.kinectSensor.CoordinateMapper;

// Obtiene información de características del cuadro de profundidad
  FrameDescription depthFrameDescription = this.kinectSensor.DepthFrameSource.FrameDescription;
int depthWidth = depthFrameDescription.Width;
int depthHeight = depthFrameDescription.Height;

// Asigna espacio en memoria para los pixeles de profundidad
this.depthPixels = new byte[depthWidth * depthHeight];

// Mapa de bits para datos de profundidad a mostrar en la pantalla
  this.depthBitmap = new WriteableBitmap(depthWidth, depthHeight, 96.0, 96.0, PixelFormats.Gray8,
  null);

//Obtiene información de características de los cuadros de color
  FrameDescription colorFrameDescription = this.kinectSensor.ColorFrameSource.FrameDescription;
int colorWidth = colorFrameDescription.Width;
int colorHeight = colorFrameDescription.Height;

//Mapa de bits para datos de color a mostrar en pantalla
  this.colorBitmap = new WriteableBitmap(colorWidth, colorHeight, 96.0, 96.0, PixelFormats.Bgra32,
  null);

  //Asigna espacio en memoria para los arreglos intermedios y arreglos que se encargaran del mapeo
this.pointCloud =new float[depthWidth * depthHeight * 3];
this.colorFrameData = new byte[colorWidth * colorHeight* 4];
this.colors =new int[depthWidth * depthHeight * 3];
this.colorData = new WriteableBitmap[FramesToStore];
this.depthData = new ushort[depthWidth * depthHeight];
this.cameraPoints = new CameraSpacePoint[depthWidth * depthHeight];
```

```
    this.colorPoints = new ColorSpacePoint[depthWidth * depthHeight];

    //Event Handler del Sensor. Revisa la disponibilidad del sensor y actualiza su estado
    this.kinectSensor.IsAvailableChanged += this.Sensor_IsAvailableChanged;

    this.kinectSensor.Open();

      this.StatusText = this.kinectSensor.IsAvailable ? Properties.Resources.RunningStatusText
                                        : Properties.Resources.NoSensorStatusText;

    this.DataContext = this;

    this.InitializeComponent();

  }

  /// "Callback" utilizado cuando un cliente solicita una conexión
  /// Acepta la conexión
  public void OnConnectRequest(IAsyncResult ar)
  {
     Socket lsock = (Socket)ar.AsyncState;
     NewConnection(lsock.EndAccept(ar));
  }

  /// Función que agrega una nueva conexión a la lista de clientes de sockets
  /// <param name="sockClient">Conexión a mantener</param>
  public void NewConnection(Socket sockClient)
  {
     string strmsg = String.Format("Client {0}, se ha unido", sockClient.RemoteEndPoint);
     this.StatusText = strmsg;

 // Verifica error en la conexión
     try
     {
        SendMSG("CA", sockClient);
     }
     catch (Exception ex)
     {
        strmsg = String.Format("¡Error al enviar Conexión Aceptada! {0}", ex.Message);
        this.StatusText = strmsg;
     }

     try
     {
        if (sockClient.Connected)
           SetupRecieveCallback(sockClient);
        else
        {
           strmsg = String.Format("¡ERROR: Fallo en la Conecxión!");
           this.StatusText = strmsg;
        }
     }
     catch { }
  }
```

```
///summary
/// Función de configuración para recepción de datos por medio de sockets
/// <param name="ar"></param>
public void SetupRecieveCallback(Socket lsock)
{

   try
   {
      AsyncCallback recieveData = new AsyncCallback(OnRecievedData);
                                                                  lsock.BeginReceive(m_byBuff, 0, m_byBuff.Length,
                                                        SocketFlags.None, recieveData, lsock);
   }
   catch (Exception ex)
   {
      string strmsg = String.Format("¡ERROR: Falló el inicio de recepción de datos! {0}", ex.Message);
      this.StatusText = strmsg;
   }
}

    /// Función de recepción de datos por medio de sockets
    /// <param name="ar"></param>
    public void OnRecievedData( IAsyncResult ar )
    {
    Socket lsock = (Socket)ar.AsyncState;

    int nBytesRec = 0;
    try
    {
                        nBytesRec = lsock.EndReceive(ar);
    }
    catch { }

    if (nBytesRec == 0)
    {
                        this.captureFrames = false;
                        this.MainWindow_Closing(this, null);
                        lsock.Close();
                        if (Gsocket != null)
                        {
                                if (Gsocket.Connected) { Gsocket.Close(); }
                        }
                        MessageBox.Show("¡ERROR: Conexión perdida!");
                        System.Environment.Exit(0);
    }
    byte[] byReturn = new byte[nBytesRec];
              Array.Copy(m_byBuff, byReturn, nBytesRec);

             // Si no se recibieron datos, es posible que la conexión falló
             if( m_byBuff.Length < 1 )
             {
                        lsock.Close();
                        return;
             }
```

```
            // Se procesa el mensaje recibido en el arreglo words
            // Comando para capturar imágenes
            // Lista de comandos; Si se desea mandar la señal
            // de captura con un tiempo de retardo
            // Ej. de comando: "captura w 10"; Se desea un retardo de 10 seg.
            // antes de mandar la señal de disparo
            // Si el comando es "captura" no se procederá a realizar un retardo
             //  words[0]= captura
             //  words[1]= w;
             //  words[2] [3] [4]; asigna segundos, minutos y hora

if (words[0].Equals("captura"))
{
                        if (words[1].Equals("w"))
                        {
                                hor = Convert.ToInt16(words[3]);
                                min = Convert.ToInt16(words[4]);
                                sec = Convert.ToInt16(words[5]) + Convert.ToInt16(words[2]);
                                while (sec > 59)
                                {
                                sec = sec - 60;
                                min = min + 1;
                                }
                                while (min > 59)
                                {
                                min = min - 60;
                                }
                                SendMSG("ST " + hor.ToString("00") + ":" + min.ToString("00") + ":" +
                        sec.ToString("00"), lsock);
                        }
                        // Se agrega la hora del servidor en el arreglo words aún y cuando no se
                        // utiliza el timer
                        else
                        {
                                hor = Convert.ToInt16(words[1]);
                                min = Convert.ToInt16(words[2]);
                                sec = Convert.ToInt16(words[3]) + 1;
                                //Ajuste de segundo
                                if (sec > 59) { sec = sec - 60; }
                        }
                        //Si se recibe un mensaje de inicio de captura revisa la hora de envío
                        // y espera al inicio del siguiente segundo para iniciar la adquisición
                        this.Gsocket = lsock;
                        StatusText = "La captura esta a punto de iniciar...";
                        while ((DateTime.Now.Minute < min) || (DateTime.Now.Second < sec))
                        {
                                System.Threading.Thread.Sleep(1);
                        }
                        StatusText = "Capturando!...";
                        SendMSG("SC", Gsocket);
                        ScreenshotButton_Click(this, null);
                        this.captureFrames = true;
```

```
}

        //Comando para sincronizar cada Equipo de Captura con el Equipo
        //de Control
else if (words[0].Equals("sync"))
{
                ProcessStartInfo psiProgram = new
                ProcessStartInfo("C:/Windows/SysWow64/cmd.exe");
                string comando = @"net time \\visionx /set /yes";
        psiProgram.Arguments = "/c" + comando;
                Process.Start(psiProgram);
                StatusText = "Sincronizacion con el servidor";
                SendMSG("SY", lsock);
}
//Comando para eliminar muestras capturadas e iniciar desde la toma 1
else if (words[0].Equals("clean"))
{
                StatusText = "Eliminando datos de captura...";
                CleanData();
                StatusText = "Datos de captura eliminados";
}
else
{
                StatusText = strRec;
}
SetupRecieveCallback(lsock);
}
/// Evento que manda al Equipo de Control el mensaje de conexión
public void SendMSG(string Message, Socket SSock)
{
if (SSock.Connected)
{
                string time1;
                string strMsg;

                time1 = System.DateTime.Now.ToString("hh'-'mm'-'ss.fff");
                strMsg = Message + " " + time1;

                // Convierte a byte array y lo envía.
                Byte[] bMsg =
                        System.Text.Encoding.ASCII.GetBytes(strMsg.ToCharArray());
        try
        {
                SSock.Send(bMsg, strMsg.Length, 0);
        }
        catch (Exception ex)
        {
                String strmsg = String.Format("Error enviar el mensaje {0}",
                                                                ex.Message);
                this.StatusText = strmsg;
        }
 }

/// Función que maneja el evento PropetyChanged
```

```
public event PropertyChangedEventHandler PropertyChanged;

/// Se obtine el bitmap de color para mostrar en pantalla
public ImageSource ImageSource1
{
   get
   {
      return this.colorBitmap;
   }
}
/// Se obtine el bitmap de profundidad para mostrar en pantalla
/// </summary>
public ImageSource ImageSource2
{
   get
   {
      return this.depthBitmap;
   }
}
/// Función que obtiene y asigna el texto de estado actual para mostrar en pantalla
/// </summary>
public string StatusText
{
   get
   {
      return this.statusText;
   }

   set
   {
      if (this.statusText != value)
      {
         this.statusText = value;

         if (this.PropertyChanged != null)
         {
            this.PropertyChanged(this, new PropertyChangedEventArgs("StatusText"));
         }
      }
   }
}

      /// <summary>
///   Función que ejecuta táreas de cierre de aplicación
/// </summary>
/// <param name="sender">objeto enviado en el evento</param>
/// <param name="e">argumentos del evento</param>
private void MainWindow_Closing(object sender, CancelEventArgs e)
{
   if (this.multiFrameSourceReader != null)
   {

      this.multiFrameSourceReader.Dispose();
      this.multiFrameSourceReader = null;
```

```
    }

    if (this.kinectSensor != null)
    {
        this.kinectSensor.Close();
        this.kinectSensor = null;
    }

    // Limpia
    this.listener.Close();
}

/// <summary>
/// Evento para captura manual de datos
/// </summary>
/// <param name="sender">object sending the event</param>
/// <param name="e">event arguments</param>
private void ScreenshotButton_Click(object sender, RoutedEventArgs e)
{
    this.captureFrames = true;

    // En caso de querer guardar las imágenes se alojarán como .png
    if (captureFrames == true)
    {
      número++;
      SavePath =CreateFolder();

       if (depthBitmap != null)
       {
          BitmapEncoder encoder_depth = new PngBitmapEncoder();
          encoder_depth.Frames.Add(BitmapFrame.Create(this.depthBitmap));

                string path_depth = Path.Combine("C:/Compartida/lectura"+
                Convert.ToString(número)+"/depth.png");

          try
          {
                 using (FileStream fs_depth = new FileStream(path_depth, FileMode.Create))
             {
                encoder_depth.Save(fs_depth);

             }
          }
          catch (IOException)
          {

          }
       }

       if (colorBitmap != null)
       {
          BitmapEncoder encoder_color = new PngBitmapEncoder();
          encoder_color.Frames.Add(BitmapFrame.Create(this.colorBitmap));
```

```
                string path_color = Path.Combine("C:/Compartida/lectura"+
                Convert.ToString(número)+"/color.png");

            try
            {
                using (FileStream fs = new FileStream(path_color, FileMode.Create))
                {
                    encoder_color.Save(fs);
                }
            }
            catch (IOException)
            {

            }
        }
    }
}

/// <summary>
/// Función que maneja la llegada de datos de color/profundidad y mapeo del sensor
/// </summary>
/// <param name="sender">objeto enviado en el evento</param>
/// <param name="e">argumento del evento  </param>

    private void Reader_MultiSourceFrameArrived(object sender, MultiSourceFrameArrivedEventArgs e)
    {
        int depthWidth = 0;
        int depthHeight = 0;
        int colorWidth = 0;
        int colorHeight = 0;

        DepthFrame depthFrame = null;
        ColorFrame colorFrame = null;
        BodyIndexFrame bodyIndexFrame = null;

        bool isDepthBitmapLocked = false;
        bool isColorBitmapLocked = false;

        MultiSourceFrame multiSourceFrame = e.FrameReference.AcquireFrame();

        // Sale de la función si el cuadro ha expirado
        if (multiSourceFrame == null)
        {
            return;
        }

        //  Se usa "try" para asegurar que se limpia la información tras cada captura
        // Se adquiere el cuadro de imagen en cuestión; BodyIndex, Profundidad y color
        try
        {
            depthFrame = multiSourceFrame.DepthFrameReference.AcquireFrame();
            colorFrame = multiSourceFrame.ColorFrameReference.AcquireFrame();
              bodyIndexFrame =   multiSourceFrame.BodyIndexFrameReference.AcquireFrame();
```

```
            // Si el cuadro expiró se termina el proceso
            // Se checa que los cuadros tengan información
            if ((depthFrame == null) || (colorFrame == null))
            {
               return;
            }

            // Procesamiento de datos de profundidad
            FrameDescription depthFrameDescription = depthFrame.FrameDescription;
            depthWidth = depthFrameDescription.Width;
            depthHeight = depthFrameDescription.Height;

            // Se congela el bitmap para lectura/escritura de datos
            this.depthBitmap.Lock();
            isDepthBitmapLocked = true;

            bool depthFrameProcessed = false;

            using (KinectBuffer depthBuffer = depthFrame.LockImageBuffer())
            {
               // Se verifica que las dimensiones del buffer donde se guardan los datos coincidan
                     //con las dimensiones de la imagen
                     if (((depthWidth * depthHeight) == (depthBuffer.Size / depthFrameDescription.BytesPerPixel))
                     &&
                      (depthFrameDescription.Width == this.depthBitmap.PixelWidth) &&
                     (depthFrameDescription.Height == this.depthBitmap.PixelHeight))
               {
                      //  Ajustando el rango de profundidad como el máximo valor de acuerdo a las
                     //características del sensor
                      // Aquí se asigna el umbral de profundidad máxima "MaxDepth"
                  ushort maxDepth = ushort.MaxValue;

                  // Configurando con las características anteriores el cuadro de imagen
                  this.ProcessDepthFrameData(depthBuffer.UnderlyingBuffer, depthBuffer.Size,
depthFrame.DepthMinReliableDistance, maxDepth, depthFrameDescription.BytesPerPixel);
                  depthFrameProcessed = true;
               }
            }


            // Al terminar con el cuadro de profundidad se destruye para una nueva muestra
            depthFrame.Dispose();
            depthFrame = null;

              // Procesamiento de datos de color
            FrameDescription colorFrameDescription = colorFrame.FrameDescription;
            colorWidth = colorFrameDescription.Width;
            colorHeight = colorFrameDescription.Height;

            //Bloquea el mapa de bits de datos de color para escritura
            this.colorBitmap.Lock();
            isColorBitmapLocked = true;

           using (KinectBuffer colorBuffer = colorFrame.LockRawImageBuffer())
```

```
            {
// Se verifica que las dimensiones del buffer donde se guardan los datos coincidan con las
// dimensiones de la imagen
                if ((colorFrameDescription.Width == this.colorBitmap.PixelWidth) &&
(colorFrameDescription.Height == this.colorBitmap.PixelHeight))
                {
  // Guarda dos copias de la imagen a color; la primera para desplegar en pantalla y la segunda
  // para hacer el mapeo tridimensional
                    colorFrame.CopyConvertedFrameDataToIntPtr(
                        this.colorBitmap.BackBuffer,
                         (uint)(colorFrameDescription.Width * colorFrameDescription.Height * 4),
                        ColorImageFormat.Bgra);

                    this.colorBitmap.AddDirtyRect(new Int32Rect(0, 0,
                          this.colorBitmap.PixelWidth, this.colorBitmap.PixelHeight));

                    if (this.captureFrames && colorFrame != null)
                    {
                        colorFrame.CopyConvertedFrameDataToArray(this.colorFrameData,
                                        ColorImageFormat.Bgra);
                    }
                }
            }

            // Al terminar el cuadro de color este se destruye
            colorFrame.Dispose();
            colorFrame = null;

            if (bodyIndexFrame != null)
            {

  // Aprovecha la detección de personas para filtrar el resto de los objetos
  //dentro del área de visión del sensor, en este caso el filtrado es de tipo binarizado
                using (KinectBuffer bodyIndexData = bodyIndexFrame.LockImageBuffer())
                {
                    unsafe
                    {
                        byte* bodyIndexDataPointer = (byte*)bodyIndexData.UnderlyingBuffer;

                        for (int i = 0; i < depthWidth * depthHeight; ++i)
                        {
                            if (bodyIndexDataPointer[i] != 0xff)
                            {
                                continue;
                            }
                            else
                            {
                                this.depthPixels[i] = 0;
                                this.depthData[i] = 0;
                            }
                        }
                    }
                }
            }
```

```
   // Se libera la información con BodyIndex
   if (bodyIndexFrame != null)
   {
      bodyIndexFrame.Dispose();
   }

   //Renderizado de profundidad
   if (depthFrameProcessed)
   {
      this.RenderDepthPixels();
   }


    // Aumenta la variable que cuenta el total de muestras o secuencias de imágenes
   if (this.captureFrames)
   {
      this.FramesCaptured++;
   }

}

// Se comprueba que la conexión con Sockets sigue vigente tras la captura
finally
{
   if (this.captureFrames && this.FramesCaptured >= FramesToStore)
   {
      if (Gsocket != null)
      {
         if (this.Gsocket.Connected)
         {
            try
            {
               SendMSG("EC", Gsocket);
            }
            catch (Exception ex)
            {
                                          String strmsg = String.Format("Error sending End of Capture!
                     {0}", ex.Message);
               this.StatusText = strmsg;
            }
         }
      }

      //Se cambia la bandera o señal de disparo como falsa hasta no recibir nuevamente
        ///el comando de captura
      this.captureFrames = false;
      // Se crea el directorio en la carpeta C:\\ Compartida en cada Equipo de Captura
      SavePath = CreateFolder();

      for (int i = 0; i < FramesToStore; i++)
      {
                              WriteDataToFiles(i, depthWidth,
            depthHeight,colorWidth,colorHeight,SavePath);
```

```
            }
            //Muestra en Estatus text el número de imagen capturada
            this.StatusText = Convert.ToString(this.FramesCaptured);

            // Se limpia el contador de número de imágenes
            this.FramesCaptured = 0;
        }

        // Se liberan los bitmap de profundidad y de color
        if (isDepthBitmapLocked)
        {
            this.depthBitmap.Unlock();
        }

        if (isColorBitmapLocked)
        {
            this.colorBitmap.Unlock();
        }

        if (depthFrame != null)
        {
            depthFrame.Dispose();
        }

        if (colorFrame != null)
        {
            colorFrame.Dispose();
        }

    }
}

/// <summary>
/// Función que accede directamente el buffer de datos de profundidad para crear
///el mapa de bits que se mostrará en pantalla
///  Esta función requiere compilar de modo "/unsafe" ya que se accesa directamente
///a la memoria nativa a la que hace referencia el puntero "depthFrameData"
/// </summary>
/// <param name="depthFrameData"> Puntero para los datos de profundidad>
/// <param name="depthFrameDataSize"> Tamaño del arreglo de imagen de profundidad>
/// <param name="minDepth">Minimo valor de profundidad>
/// <param name="maxDepth">Máximo valor de profundidad>

    private unsafe void ProcessDepthFrameData(IntPtr depthFrameData, uint depthFrameDataSize, ushort
    minDepth, ushort maxDepth, uint depthBytesPerPixel)
    {
        //  Datos del cuadro de profundidad en valores de 16 bits
        ushort* FrameData = (ushort*)depthFrameData;

        // Convierte el cuadro de profundidad a una representación visual
        for (int i = 0; i < (int)(depthFrameDataSize / depthBytesPerPixel); ++i)
        {
            // Obtiene la profundidad de cada pixel
            ushort depth = FrameData[i];
```

```
            // Para convertir a byte, se mapea el valor de profundidad a la escala de 0 a 255.
            // Valores fuera del rango de profundidad se le asigna un valor de 0 (negro)
            this.depthPixels[i] = (byte)(depth >= minDepth && depth <= maxDepth ? (depth / MapDepthToByte) :
0);
            if(this.captureFrames)
               this.depthData[i] = depth;
         }
      }

      /// <summary>
      /// Función que escribe los datos del cuadro de profundidad en un mapa de bits.
      /// </summary>
      private void RenderDepthPixels()
      {
         this.depthBitmap.WritePixels(
            new Int32Rect(0, 0, this.depthBitmap.PixelWidth, this.depthBitmap.PixelHeight),
            this.depthPixels,
            this.depthBitmap.PixelWidth,
            0);
      }
//Función que crea una carpeta para guardar archivos con los datos adquiridos.
//Regresa la dirección de la carpeta.
      private string CreateFolder()
      {
         int count = 1;
         // Especifica el directorio
         string número = Convert.ToString(count);
         string Fpath = Path.Combine("C:/Compartida/lectura" + número);

         // Examina si la carpeta ha sido creada
         while (Directory.Exists(Fpath))
         {
            count++;
            número = Convert.ToString(count);
            Fpath = Path.Combine("C:/Compartida/lectura" + número);
         }
         try
         {
            // En caso de no existir la carpeta, se crea.
            DirectoryInfo di = Directory.CreateDirectory(Fpath);
         }
         catch
         {

         }

            return Fpath;

      }

 /// Función que escribe los datos de los cuadros de profundidad adquiridos en archivos en
 /// modo texto con formato de nube de puntos RGBD.
```

```
public unsafe void WriteDataToFiles(int nframe, int depthcols, int depthrows,int colorWidth, int
colorHeight,string Fpath)
{
  //Especifica el nombre de los archivos de nube de puntos y el archivo a color
    string path = Path.Combine(Fpath + "/kinectpointcloud-" + Convert.ToString(nframe+1) + ".txt");

  string path_color = Path.Combine(Fpath + "/Color-" + Convert.ToString(nframe + 1) + ".txt");

   // Borrra el archivo si este existe
   if (File.Exists(path))
   {
      File.Delete(path);

   }

  // Función que crea los archivos y el mapeo entre el espacio de color de profundidad y // el espacio de
  coordenadas del sensor
 using (FileStream fs = File.Create(path))
  {
   using (StreamWriter sw_color = new StreamWriter(path_color))
   {
      using (StreamWriter sw = new StreamWriter(fs))
      {
            //Se realiza un mapeo entre los datos de profundidad hacia el marco de
            // referencia de la camara
          coordinateMapper.MapDepthFrameToCameraSpace(depthData, cameraPoints);
          // Se realiza un mapeo con los datos del marco de referencia de la camara hacia
          //su respectivo color
          coordinateMapper.MapCameraPointsToColorSpace(cameraPoints, colorPoints);

          // Se crea una variable size del tamaño de la imagen de profundidad para hacer
          //el barrido y asignar el color que le corresponde a cada punto de la nube.
       int size = depthcols * depthrows;

       // Inicia la asignación de color manejando el espacio de color,
       //  de profundidad y marco de referencia de la cámara
       // El ciclo recorrerá las dimensiones de la imagen de profundidad
       for (int i = 0; i <size; i++)
       {
          ushort depth = this.depthData[i];

          //Comprueba que el pixel de profundidad recorrido contenga información
          // o haya detectado a la persona
          if (depth != 0)
          {
             // Se crean índices para la posición x,y en el cuadro de imagen
             int cx = (int)colorPoints[i].X;
             int cy = (int)colorPoints[i].Y;

             // Las componentes a color se presentan como variables tipo byte 0-255
             byte R, G, B;

             // Se aplica un redondeo para los índices
             int colorX = (int)Math.Floor(cx + 0.5);
```

```
                int colorY = (int)Math.Floor(cy + 0.5);

                // Se realiza el mapeo corroborando que el pixel que le corresponde a la
    // imagen a color
                // La imagen a profundidad y el espacio de nube de puntos
                    if ((colorX >= 0) && (colorX < colorWidth) && (colorY >= 0) && (colorY <
                    colorHeight))
                {
                  // Extracción de coordenadas x,y,z a partir del espacio de la cámara
                  // [i] el vector x, [i+size] es vector y y [i+2*size] corresponde a z
                  // para la matriz creada

                  pointCloud[i] = cameraPoints[i].X;
                  pointCloud[i + size] = cameraPoints[i].Y;
                  pointCloud[i + 2 * size] = cameraPoints[i].Z;

                  //Conversión de los valores a tipo string para impresión en archivo
                  xValue = Convert.ToString(1000 * pointCloud[i]);
                  yValue = Convert.ToString(1000 * pointCloud[i + size]);
                  zValue = Convert.ToString(1000 * pointCloud[i + size * 2]);

                      // Extracción de componentes RGB recorriendo la imagen a color
                  // calculando el índice del arreglo que apunta a la imagen de color
                  int colorIndex = (colorX + (colorY * 1920)) * 4;
                  //Extrayendo las componentes
                  B = colorFrameData[colorIndex];
                  G = colorFrameData[colorIndex + 1];
                  R = colorFrameData[colorIndex + 2];
                  //Guardando las componentes en el arreglo colors
                  colors[i] = B;
                  colors[i + size] = G;
                  colors[i + size + size] = R;

                  //Convierte los datos para impresión en archivo de texto
                  Blue = Convert.ToString(colors[i]);
                  Green = Convert.ToString(colors[i + size]);
                  Red = Convert.ToString(colors[i + size * 2]);
                }

                sw.Write(xValue + "  " + yValue + "  " + zValue);
                sw.WriteLine();
                sw_color.Write(Red + " " + Green + " " + Blue);
                sw_color.WriteLine();

              }
            }
            sw.Close();
          }
          sw_color.Close();
        }
        fs.Close();
    }
}
```

```
 //Función que limpia los datos al recibir el comando clean
     private void CleanData()
     {
        int count = 1;
        int nfile = 1;
        string número = Convert.ToString(count);
        string Fpath = Path.Combine("C:/Compartida/lectura" + número);
          string path = Path.Combine(Fpath + "/kinectpointcloud-" + Convert.ToString(nfile) + ".txt");

 while (Directory.Exists(Fpath))
       {
           while (File.Exists(path))
           {
              File.Delete(path);
              nfile++;
                      path = Path.Combine(Fpath + "/kinectpointcloud-" + Convert.ToString(nfile) + ".txt");
           }
           Directory.Delete(Fpath);
           count++;
           número = Convert.ToString(count);
           Fpath = Path.Combine("C:/Compartida/lectura" + número);
       }

     }
   /// <summary>
   /// Función que determina la disponibilidad del sensor
   /// </summary>
   /// <param name="sender">objeto enviado al evento</param>
   /// <param name="e">argumentos del evento</param>
   private void Sensor_IsAvailableChanged(object sender, IsAvailableChangedEventArgs e)
   {
          this.StatusText = this.kinectSensor.IsAvailable ? Properties.Resources.RunningStatusText
                                                  : Properties.Resources.SensorNotAvailableStatusText;
    }
  }
}
```

Anexo 3. Código de Software para el Equipo de Control. (C#/xaml)

```
//-----------------------------------------------------------------------------------------------------------
//      Proyecto:KinectSocketClient
//      Archivo:KSCForm.cs
//      Última versión por: Ing. Rodríguez Salgado Daniel
//      Versión original: M.C. Velasco Víctor Hugo, Ing. Julio Omar Vega,
//      Dr. Isidro Robledo Vega, SDK Microsoft Kinect
//      Última modificación: Enero-Junio 2019
//
//      Esta aplicación contiene el código de la aplicación cliente
//      de Sockets con funciones que envían mensajes
//      o comandos a los Equipos de Captura o servidores para
//      el inicio de captura sincronizada.
//
//-----------------------------------------------------------------------------------------------------------

//Bibliotecas
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.IO;


// Declara el prototipo del delegado para enviar datos de regreso a la Forma
delegate void AddMessage(string sNewMessage);

//Aplicación del cliente de Sockets
namespace KinectSocketClient
{
   //Clase KSCForm que define la forma para aplicación
   public partial class KSCForm : Form
   {

      private Socket m_sock1; // Conexión al primer servidor de sockets (NUC1)
      private Socket m_sock2; // Conexión al segundo servidor de sockets (NUC2)
      private Socket m_sock3; // Conexión al segundo servidor de sockets (NUC3)
      private Socket m_sock4; // Conexión al segundo servidor de sockets (NUC4)

      // Cuatro Búferes de recepción de datos
      private byte[] m_byBuff1 = new byte[256];
      private byte[] m_byBuff2 = new byte[256];
      private byte[] m_byBuff3 = new byte[256];
      private byte[] m_byBuff4 = new byte[256];
      //Mensajes del manejador de eventos
```

```
    private event AddMessage m_AddMessage;

    // Constructor de la clase KSCForm
    public KSCForm()
    {
       InitializeComponent();
       m_AddMessage = new AddMessage(OnAddMessage);
    }

private void ConnectButton_Click(object sender, EventArgs e)
    {
       Cursor cursor = Cursor.Current;
       Cursor.Current = Cursors.WaitCursor;
       try
       {
          // Cierra el socket 1 si aún esta abierto
          if (m_sock1 != null && m_sock1.Connected)
          {
             m_sock1.Shutdown(SocketShutdown.Both);
             m_sock1.Close();
          }

          // Cierra el socket 2 si aún esta abierto
          if (m_sock2 != null && m_sock2.Connected)
          {
             m_sock2.Shutdown(SocketShutdown.Both);
             m_sock2.Close();
          }

          // Cierra el socket 3 si aún esta abierto
          if (m_sock3 != null && m_sock3.Connected)
          {
             m_sock3.Shutdown(SocketShutdown.Both);
             m_sock3.Close();
          }

          // Cierra el socket 4 si aún esta abierto
          if (m_sock4 != null && m_sock4.Connected)
          {
             m_sock4.Shutdown(SocketShutdown.Both);
             m_sock4.Close();
          }

          // Puerto de comunicacion asignado como 399
          const int nPortListen = 399;

           // Toma las direcciones IP de los servidores de Sockets de las cajas de texto
          IPAddress aryServerAddr1 = IPAddress.Parse(server1.Text);
          IPAddress aryServerAddr2 = IPAddress.Parse(server2.Text);
          IPAddress aryServerAddr3 = IPAddress.Parse(server3.Text);
          IPAddress aryServerAddr4 = IPAddress.Parse(server4.Text);

          // Se crean los sockets para conectar a los servidores
          m_sock1 = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
```

```
            m_sock2 = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
            m_sock3 = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
            m_sock4 = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

            IPEndPoint epServer1 = new IPEndPoint(aryServerAddr1, nPortListen);
            IPEndPoint epServer2 = new IPEndPoint(aryServerAddr2, nPortListen);
            IPEndPoint epServer3 = new IPEndPoint(aryServerAddr3, nPortListen);
            IPEndPoint epServer4 = new IPEndPoint(aryServerAddr4, nPortListen);

              // Realizando la conexión
            m_sock1.Blocking = false;
            m_sock2.Blocking = false;
            m_sock3.Blocking = false;
            m_sock4.Blocking = false;
            // Realiza un callback de conexión
            AsyncCallback onconnect1 = new AsyncCallback(OnConnect);
            AsyncCallback onconnect2 = new AsyncCallback(OnConnect);
            AsyncCallback onconnect3 = new AsyncCallback(OnConnect);
            AsyncCallback onconnect4 = new AsyncCallback(OnConnect);

            m_sock1.BeginConnect(epServer1, onconnect1, m_sock1);
            System.Threading.Thread.Sleep(10);
            m_sock2.BeginConnect(epServer2, onconnect2, m_sock2);
            System.Threading.Thread.Sleep(10);
            m_sock3.BeginConnect(epServer3, onconnect3, m_sock3);
            System.Threading.Thread.Sleep(10);
            m_sock4.BeginConnect(epServer4, onconnect4, m_sock4);
            System.Threading.Thread.Sleep(10);


         }

          catch (Exception ex)
         {
            string ErrMsg = "Error: " + ex.Message;
            MessageBox.Show(ErrMsg, "¡Falló la Conexión al Servidor!");

         }
         Cursor.Current = cursor;
      }

// Función que configura la recepción de mensajes al conectar los sockets
      public void OnConnect(IAsyncResult ar)
      {
         // Se pasa el socket como objeto
         Socket sock = (Socket)ar.AsyncState;

         // Corrobora si la conexión fue exitosa
         try
         {
            //sock.EndConnect( ar );
            if (sock.Connected)
               SetupRecieveCallback(sock);
            else
```

```
                AddMessage("No se puede conectar a la computadora remota. ¡Fallo de Conexión!");
            }
            catch (Exception ex)
            {
               string ErrMsg = "Error: " + ex.Message;
               MessageBox.Show(ErrMsg, "¡Error Inusual Durante la Conexión!");

               return;
            }
        }
/// <summary>
/// Función que obtiene datos y la envía al resto de las conexiónes
/// Nota: Si no se reciben datos es posible que la conexión haya fallado
/// </summary>
/// <param name="ar"></param>
        public void OnRecievedData(IAsyncResult ar)
        {
            // Se pasa al socket como objeto
            Socket sock = (Socket)ar.AsyncState;

            // Revisa si se recibieron datos
            try
            {
               int nBytesRec = sock.EndReceive(ar);
               if (nBytesRec > 0)
               {
                  //Escribe los datos recibidos en el búfer para cada servidor
                  string sRecieved = "";
                    if (sock == m_sock1)
                          {
                          sRecieved = Encoding.ASCII.GetString(m_byBuff1, 0, nBytesRec);
                          }

                  if (sock == m_sock2)
                          {
                          sRecieved = Encoding.ASCII.GetString(m_byBuff2, 0, nBytesRec);
                          }

                  if (sock == m_sock3)
                          {
                         sRecieved = Encoding.ASCII.GetString(m_byBuff3, 0, nBytesRec);
                          }

                  if (sock == m_sock4)
                          {
                          sRecieved = Encoding.ASCII.GetString(m_byBuff4, 0, nBytesRec);
                          }

                          Data_Arrival(sRecieved, sock);
                          SetupRecieveCallback(sock);

               }
               else
               {
```

```
                // Si no se recibieron datos se cierra el socket por fallo de conexión.
                    string errmsg = String.Format("Cliente {0}, disconectado", sock.RemoteEndPoint);
                AddMessage(errmsg);
                sock.Shutdown(SocketShutdown.Both);
                sock.Close();
            }
        }
        catch (Exception ex)
        {
            string ErrMsg = "Error: " + ex.Message;
            MessageBox.Show(ErrMsg, "¡Conexión perdida!");
        }

    }

    //Función que muestra los mensajes recibidos en la interfaz
    public void OnAddMessage(string sMessage)
    {
        ReceivedMessagesBox.Items.Add(sMessage);
        if (ReceivedMessagesBox.Items.Count > 9)
        {
            ReceivedMessagesBox.TopIndex = ReceivedMessagesBox.Items.Count - 1;
        }
    }

    //Función que contiene el mensaje a enviar como string
    // Se crea un Delagado para para el manejo de mensajes en el listbox
    public void AddMessage(string Message)
    {
        Invoke(m_AddMessage, new string[] { Message });

    }
/// <summary>
/// Función que configura la recpeción de datos
/// </summary>
    public void SetupRecieveCallback(Socket sock)
    {
        try
        {
            AsyncCallback recieveData = new AsyncCallback(OnRecievedData);
                if (sock == m_sock1) { sock.BeginReceive(m_byBuff1, 0, m_byBuff1.Length, SocketFlags.None,
          recieveData, sock); }
                if (sock == m_sock2) { sock.BeginReceive(m_byBuff2, 0, m_byBuff2.Length, SocketFlags.None,
          recieveData, sock); }
                if (sock == m_sock3) { sock.BeginReceive(m_byBuff3, 0, m_byBuff3.Length, SocketFlags.None,
          recieveData, sock); }
                if (sock == m_sock4) { sock.BeginReceive(m_byBuff4, 0, m_byBuff4.Length, SocketFlags.None,
          recieveData, sock); }
        }
        catch (Exception ex)
        {
            string ErrMsg = "Error: " + ex.Message;
            MessageBox.Show(ErrMsg, "¡Fallo al iniciar la rutina de recepción de mensajes!");
        }
```

```
        }
        /// <summary>
        /// Función que se ejecuta al cerrar la aplicación cerrando las conexiónes
        /// </summary>
        private void KSCForm_Closing(object sender,
                        System.ComponentModel.CancelEventArgs e)
        {
           if (m_sock1 != null && m_sock1.Connected)
           {
              m_sock1.Shutdown(SocketShutdown.Both);
              m_sock1.Close();
           }
           if (m_sock2 != null && m_sock2.Connected)
           {
              m_sock2.Shutdown(SocketShutdown.Both);
              m_sock2.Close();
           }
           if (m_sock3 != null && m_sock3.Connected)
           {
              m_sock3.Shutdown(SocketShutdown.Both);
              m_sock3.Close();
           }
           if (m_sock4 != null && m_sock4.Connected)
           {
              m_sock4.Shutdown(SocketShutdown.Both);
              m_sock4.Close();
           }
        }

/// <summary>
/// Función que se ejecuta al oprimir el botón “Enviar Mensaje”
/// </summary>
        private void SendMessageButton_Click(object sender, EventArgs e)
        {
           // Comprueba si existe aun conexión
           if ((m_sock1 == null || !m_sock1.Connected)
              || (m_sock2 == null || !m_sock2.Connected)
              || (m_sock3 == null || !m_sock3.Connected)
              || (m_sock4 == null || !m_sock4.Connected))
           {
              MessageBox.Show("Debe estar conectado para enviar un mensaje");
              return;
           }
           // Función que envia el mensaje o comandos a los Equipos de Captura
           SendMSG(SendMessageBox.Text, m_sock1);
           SendMSG(SendMessageBox.Text, m_sock2);
           SendMSG(SendMessageBox.Text, m_sock3);
           SendMSG(SendMessageBox.Text, m_sock4);

           // Función que extrae los archivos de datos de cada uno de
           // los Equipos de Captura
           string[] param = SendMessageBox.Text.Split(' ');
           CheckParams(param);
        }
```

```
/// <summary>
/// Se obtienen mensajes de que se ejecutaron las instrucciones
/// provinientes de  cada Equipo de Captura
/// </summary>
        private void Data_Arrival(string Data, Socket sock)
        {
            string[] words = Data.Split(' ');
            int NWord = 0;
            int skip = 0;

            foreach (string S in words) //Procesa cada palabra dentro de un buffer
            {
                if (skip > 0)
                {
                    skip--;
                    continue;
                }
                if (S == "") { continue; }

                if (S == "CA") // La conexión fue satisfactoria. Indica la hora
                {
                            if (sock == m_sock1) { AddMessage("Conexión aceptada por NUC1 " + words[NWord + 1]);
                            }
                            if (sock == m_sock2) { AddMessage("Conexión aceptada por NUC2 " + words[NWord + 1]); }
                            if (sock == m_sock3) { AddMessage("Conexión aceptada por NUC3 " + words[NWord + 1]); }
                            if (sock == m_sock4) { AddMessage("Conexión aceptada por NUC4 " + words[NWord + 1]); }
                    skip = 1;
                }
                else if (S == "SC") // Empieza la captura. Indica la hora
                {
                            if (sock == m_sock1) { AddMessage("Inicio de Captura del NUC1 " + words[NWord + 1]); }
                            if (sock == m_sock2) { AddMessage("Inicio de Captura del NUC2 " + words[NWord + 1]); }
                            if (sock == m_sock3) { AddMessage("Inicio de Captura del NUC3 " + words[NWord + 1]); }
                    if (sock == m_sock4) { AddMessage("Inicio de Captura del NUC4 " + words[NWord + 1]); }
                    skip = 1;
                }
                else if (S == "EC") //Hora de término de captura. Indica la hora
                {

                            if (sock == m_sock1) { AddMessage("Fin de Captura del NUC1 " + words[NWord + 1]); }
                            if (sock == m_sock2) { AddMessage("Fin de Captura del NUC2 " + words[NWord + 1]); }
                            if (sock == m_sock3) { AddMessage("Fin de Captura del NUC3 " + words[NWord + 1]); }
                            if (sock == m_sock4) { AddMessage("Fin de Captura del NUC4 " + words[NWord + 1]); }
                    skip = 1;
                }
                else if (S == "ST") //Indica la hora de disparo cuando se inicializa el temporizador
                {
                            if (sock == m_sock1) { AddMessage("NUC1 inicia a : " + words[NWord + 1] + " " +
                            words[NWord + 2]); }
                             if (sock == m_sock2) { AddMessage("NUC2 inicia a: " + words[NWord + 1] + " " +
                            words[NWord + 2]); }
                            if (sock == m_sock3) { AddMessage("NUC3 inicia a: " + words[NWord + 1] + " " +
                            words[NWord + 2]); }
```

```
                    if (sock == m_sock4) { AddMessage("NUC4 inicia a: " + words[NWord + 1] + " " +
                    words[NWord + 2]); }
                skip = 2;
            }
            else if (S == "OF") //NUC desconectado
            {

                    if (sock == m_sock1) { AddMessage("NUC1 desconectado " + words[NWord + 1]); }
                    if (sock == m_sock2) { AddMessage("NUC2 desconectado " + words[NWord + 1]); }
                    if (sock == m_sock3) { AddMessage("NUC3 desconectado " + words[NWord + 1]); }
                    if (sock == m_sock4) { AddMessage("NUC4 desconectado " + words[NWord + 1]); }
                skip = 1;
            }
            else if (S == "SY") //Hora de sincronización
            {
                if (sock == m_sock1) { AddMessage("NUC1 sync " + words[NWord + 1]); }
                if (sock == m_sock2) { AddMessage("NUC2 sync " + words[NWord + 1]); }
                if (sock == m_sock3) { AddMessage("NUC3 sync " + words[NWord + 1]); }
                if (sock == m_sock4) { AddMessage("NUC4 sync " + words[NWord + 1]); }
                skip = 1;
            }
            else
            {
                AddMessage("-> " + S);
            }

            NWord++;
        }

    }
// Se crea una carpeta en el Equipo de Control para la obtención
// de los datos tridimensionales de los cuatro Equipos de Captura
    private string CreateFolder(Socket sock)
    {
        int count = 1;
        // Ruta donde se alojaran los datos
        string Rpath =Environment.ExpandEnvironmentVariables("%userprofile%
                \\Desktop\\Kinect Data Fusion\\muestras1024\\S");
        string Fpath = Rpath + Convert.ToString(count);
        string Fdata = "";
        if (sock == m_sock1) { Fdata = "\\nuc1"; }
        if (sock == m_sock2) { Fdata = "\\nuc2"; }
        if (sock == m_sock3) { Fdata = "\\nuc3"; }
        if (sock == m_sock4) { Fdata = "\\nuc4"; }

        // Determina si existe el directorio.
        while (Directory.Exists(Rpath + Convert.ToString(count) + Fdata))
        {
            count++;
            Fpath = Rpath + Convert.ToString(count);
        }
        try
        {
            // Intenta creae el directorio
```

```
            DirectoryInfo di2 = Directory.CreateDirectory(Fpath + Fdata);

        }
        catch
        {

        }

        return Fpath + Fdata;

    }

    //Función que convierte en byte el mensaje y lo manda a través del socket
    private void SendMSG(string Message, Socket sock)
    {

        try
        {

            string sndmsg = Message + " " + DateTime.Now.ToString("hh'-'mm'-'ss'-'ff");
            Byte[] byteDateLine = Encoding.ASCII.GetBytes(sndmsg.ToCharArray());
            if (sock != null) { sock.Send(byteDateLine, byteDateLine.Length, 0); }

        }

        catch (Exception ex)
        {
            string ErrMsg = "Error: " + ex.Message;
            MessageBox.Show(ErrMsg, "¡ Falló el Envío del Mensaje!");
        }

    }

  // Función que transfiere los datos de los Equipos de Captura
//  hacia la carpeta creada en el Equipo de Control
  private void CheckParams(string[] Params)
   {
        if (Params[0] == "getseries") //Comando de transferencia
        {
            DirectoryInfo diSeriesN1 = new DirectoryInfo("//VISIONNUC1/Compartida");
            DirectoryInfo diSeriesN2 = new DirectoryInfo("//VISIONNUC2/Compartida");
            DirectoryInfo diSeriesN3 = new DirectoryInfo("//VISIONNUC3/Compartida");
            DirectoryInfo diSeriesN4 = new DirectoryInfo("//VISIONNUC4/Compartida");

            if (diSeriesN1.Exists & diSeriesN2.Exists & diSeriesN3.Exists & diSeriesN4.Exists)
            {
                AddMessage("Obteniendo datos...");
                foreach (DirectoryInfo Dir in diSeriesN1.GetDirectories("lectura*"))
                {
                    string FilePathN1 = CreateFolder(m_sock1);
                    foreach (FileInfo File in Dir.GetFiles())
                    {
                        File.MoveTo(FilePathN1 + "/" + File.Name);
                    }
```

```
                Dir.Delete();
            }


            foreach (DirectoryInfo Dir in diSeriesN2.GetDirectories("lectura*"))
            {
                string FilePathN2 = CreateFolder(m_sock2);
                foreach (FileInfo File in Dir.GetFiles())
                {
                    File.MoveTo(FilePathN2 + "/" + File.Name);
                }
                Dir.Delete();
            }

            foreach (DirectoryInfo Dir in diSeriesN3.GetDirectories("lectura*"))
            {
                string FilePathN3 = CreateFolder(m_sock3);
                foreach (FileInfo File in Dir.GetFiles())
                {
                    File.MoveTo(FilePathN3 + "/" + File.Name);
                }
                Dir.Delete();
            }

            foreach (DirectoryInfo Dir in diSeriesN4.GetDirectories("lectura*"))
            {
                string FilePathN4 = CreateFolder(m_sock4);
                foreach (FileInfo File in Dir.GetFiles())
                {
                    File.MoveTo(FilePathN4 + "/" + File.Name);
                }
                Dir.Delete();
            }

            AddMessage("Datos transferidos con exito");
        }
        else
        {
            AddMessage("No se encontro la ruta de acceso");
        }
    }
  }
}
```

Anexo 4. Código de Filtrado y Fusión de Datos Tridimensionales. (Matlab)

```
% Función data_fusion(num_sec)
% Última version: Ing. Rodríguez Salgado Daniel
% Versión anterior: M.C. Víctor Hugo Velasco, Ing. Julio Omar Vega
%
% Fecha: Enero- Julio 2019
%
% La siguiente función se encarga de realizar la fusión de los datos
% provenientes de los cuatro sensores RGBD, utilizando el área de captura
% implementado como algoritmo.
%
% Entrada: num_sec.- Número de secuencia adquirido en la captura (1, 2,3…etc.)
% Salida: Nube de fusión de datos en formato .PLY
%           Matriz con datos de fusion .mat

function data_fusion(sec_num)
%Ruta en donde se encuentran los datos capturados con los sensores
dir='C:\Users\HP\Desktop\Kinect Data Fusion\muestras1024\18-10-18\S';
% Se añaden prefijos para concatenar la ruta completa
prefix1='\nuc1\kinectpointcloud-1';
prefix2='\nuc2\kinectpointcloud-1';
prefix3='\nuc3\kinectpointcloud-1';
prefix4='\nuc4\kinectpointcloud-1';
prefix_color1='\nuc1\Color-1';
prefix_color2='\nuc2\Color-1';
prefix_color3='\nuc3\Color-1';
prefix_color4='\nuc4\Color-1';
sufix='.txt';

%Cargando a Matlab los datos del sensor 1
% Concatenación de la ruta completa, sea sec_num el número de secuencia
s1_kpc=strcat(dir,int2str(sec_num),prefix1,sufix);
%Se carga en una matriz de tamaño 6*n las coordenadas y la informacion a color
s1=load(s1_kpc);
c1_kpc=strcat(dir,int2str(sec_num),prefix_color1,sufix);
c1=load(c1_kpc);
s1=[s1 c1];
%Filtrado inicial, para remover puntos fuera de la zona de captura
j=1;
for i=1:size(s1,1)
    if s1(i,1)>-1000 && s1(i,1)<1000 && s1(i,2)>-900 && s1(i,2)<1100 && s1(i,3)>1000 && s1(i,3)<3000
       s1p(j,:)=s1(i,:);
       j=j+1;
   end
end

% Realiza la transformación geométrica para ubicar los ejes coordenados
% del sensor 1 con respecto al punto de referencia ubicado entre los
% dos sensores
s1p(:,1)=s1p(:,1)-42;
s1p(:,2)=s1p(:,2)-10;
```

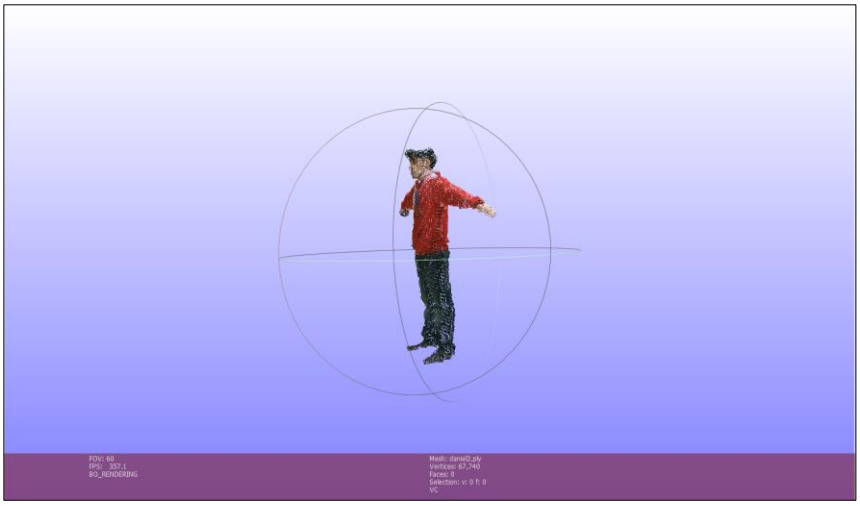
```
s1p(:,3)=s1p(:,3)-2000;

% NOTA: El código comentado permite visualizar la nube de puntos del
% sensor 1 antes de la fusión de datos
%s1_ptc = pointCloud(s1p(:,1:3),'Color',uint8(s1p(:,4:6)));
%s1_ptc_dn=pcdenoise(s1_ptc, 'NumNeighbors',10,'Threshold',8);
%pcshow(s1_ptc_dn);
%cameratoolbar('SetCoordSys','y');

% Lee la nube de puntos del Sensor 2 segmentando la región donde se
% mueve la persona
s2_kpc=strcat(dir,int2str(sec_num),prefix2,sufix);
s2=load(s2_kpc);
c2_kpc=strcat(dir,int2str(sec_num),prefix_color2,sufix);
c2=load(c2_kpc);
s2=[s2 c2];
j=1;
for i=1:size(s2,1)
    if s2(i,1)>-1000 && s2(i,1)<1000 && s2(i,2)>-900 && s2(i,2)<1100 && s2(i,3)>1000 && s2(i,3)<3000
       s2p(j,:)=s2(i,:);
       j=j+1;
   end
end

% Realiza la transformación geométrica para ubicar los ejes coordenados
% del sensor 2 con respecto al punto de referencia ubicado entre los
% dos sensores
s2p(:,1)=s2p(:,1)*(-1)+42;
s2p(:,2)=s2p(:,2)+40;
s2p(:,3)=s2p(:,3)*(-1)+2000;

% NOTA: El código comentado permite visualizar la nube de puntos del
% sensor 2 antes de la fusión de datos
%s2_ptc = pointCloud(s2p(:,1:3),'Color',uint8(s2p(:,4:6)));
%s2_ptc_dn=pcdenoise(s2_ptc, 'NumNeighbors',10,'Threshold',8);
%figure
%pcshow(s2_ptc_dn);
%cameratoolbar('SetCoordSys','y');

% Lee la nube de puntos del Sensor 3 segmentando la región donde se
% mueve la persona
s3_kpc=strcat(dir,int2str(sec_num),prefix3,sufix);
s3=load(s3_kpc);
c3_kpc=strcat(dir,int2str(sec_num),prefix_color3,sufix);
c3=load(c3_kpc);
s3=[s3 c3];
j=1;
for i=1:size(s3,1)
    if s3(i,1)>-1000 && s3(i,1)<1000 && s3(i,2)>-900 && s3(i,2)<1100 && s3(i,3)>1000 && s3(i,3)<3000
       s3p(j,:)=s3(i,:);
       j=j+1;
   end
end
```

```
% Realiza la transformación geométrica para ubicar los ejes coordenados
% del sensor 3 con respecto al punto de referencia ubicado entre los
% dos sensores
s3p=[s3p(:,3)*(-1) s3p(:,2) s3p(:,1) s3p(:,4:6)];
s3p(:,1)=s3p(:,1)+2000;
s3p(:,3)=s3p(:,3)-42;


% NOTA: El código comentado permite visualizar la nube de puntos del
% sensor 3 antes de la fusión de datos
%s3_ptc = pointCloud(s3p(:,1:3),'Color',uint8(s3p(:,4:6)));
%s3_ptc_dn=pcdenoise(s3_ptc, 'NumNeighbors',10,'Threshold',8);
%figure
%pcshow(s3_ptc_dn);
%cameratoolbar('SetCoordSys','y');

% Lee la nube de puntos del Sensor 4 segmentando la región donde se
% mueve la persona
s4_kpc=strcat(dir,int2str(sec_num),prefix4,sufix);
s4=load(s4_kpc);
c4_kpc=strcat(dir,int2str(sec_num),prefix_color4,sufix);
c4=load(c4_kpc);
s4=[s4 c4];
j=1;
for i=1:size(s4,1)
   if s4(i,1)>-1000 && s4(i,1)<1000 && s4(i,2)>-900 && s4(i,2)<1100 && s4(i,3)>1000 && s4(i,3)<3000
      s4p(j,:)=s4(i,:);
      j=j+1;
   end
end

% Realiza la transformación geométrica para ubicar los ejes coordenados
% del sensor 4 con respecto al punto de referencia ubicado entre los
% dos sensores
s4p=[s4p(:,3) s4p(:,2) s4p(:,1)*(-1) s4p(:,4:6)];
s4p(:,1)=s4p(:,1)-2000;
s4p(:,3)=s4p(:,3)+42;

% NOTA: El código comentado permite visualizar la nube de puntos del
% sensor 4 antes de la fusión de datos
%s4_ptc = pointCloud(s4p(:,1:3),'Color',uint8(s4p(:,4:6)));
%s4_ptc_dn=pcdenoise(s4_ptc, 'NumNeighbors',10,'Threshold',8);
%figure
%pcshow(s4_ptc_dn);
%cameratoolbar('SetCoordSys','y');

% Se realiza la fusión de los datos transformados de los cuatro sensores y
% se aplica un filtro a la nube de puntos fusionada para eliminar
% puntos aislados
fusion=[s1p; s2p; s3p; s4p];
%Función de nube de puntos pointcloud(xyz,'Color',RGB)
fused_ptc = pointCloud(fusion(:,1:3),'Color',uint8(fusion(:,4:6)));
fused_ptc_dn=pcdenoise(fused_ptc, 'NumNeighbors',50,'Threshold',1);
fused_ptc_dw = pcdownsample(fused_ptc_dn,'gridAverage',1);
```

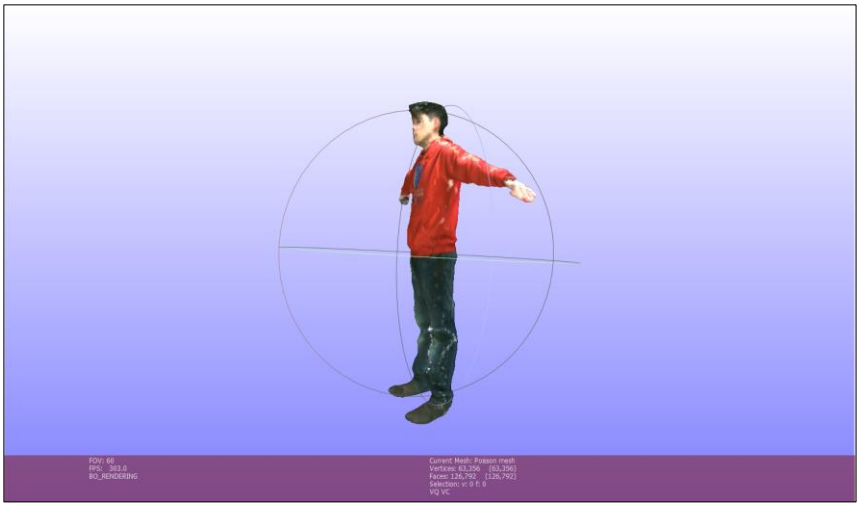
```
% NOTA: El código comentado permite visualizar la nube de puntos
% resultante despues de la fusión de datos
figure
pcshow(fused_ptc_dw);
xlabel('X');
ylabel('Y');
zlabel('Z');
cameratoolbar('SetCoordSys','y');

%Creando la ruta donde se alojarán los datos fusionados
path=strcat(dir,int2str(sec_num),'\Fused_PC\');
if exist(path,'dir')
   rmdir(path,'s');
   mkdir(path);
else
   mkdir(path);
end

%Se guarda la nube de puntos fusionada como una matriz
PCData=[fused_ptc_dw.Location double(fused_ptc_dw.Color)];
outfile_Location=strcat(path,'-PC','.mat');
save(outfile_Location,'PCData');

%Por último se genera un archivo con extensión PLY para manejo del modelo
ply_file=strcat(dir,int2str(sec_num),'\Fused_PC\Archivo3D.ply');
fused_ptc = pointCloud(PCData(:,1:3),'Color',uint8(PCData(:,4:6)));
pcwrite(fused_ptc,ply_file,'Encoding','ascii');
```

Anexo 5. Modelos Tridimensionales de Personas en la Base de Datos.

Las siguientes imágenes muestran los modelos tridimensionales de personas escaneadas bajo el área de captura tipo cruz y almacenadas en la base de datos del proyecto. Se siguió el procedimiento descrito en la sección 5.3 de esta tesis para generar los modelos tridimensionales. Estos modelos fueron construidos usando el algoritmo de Poisson variando solo en las operaciones de filtrado y de suavizado.



a)



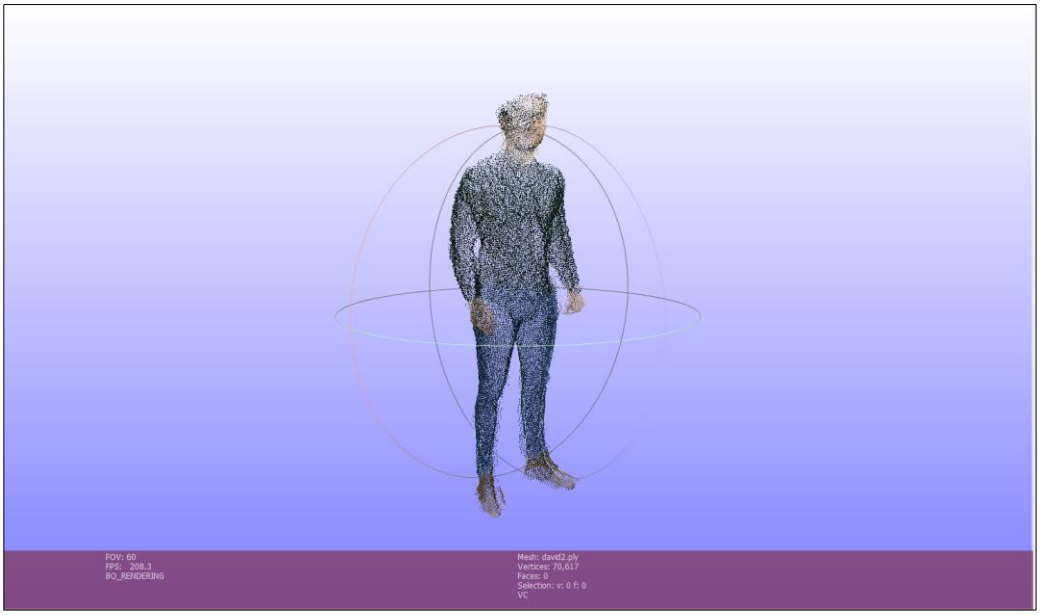
b)

Figura A.1. Modelo de persona 1. a) Nube de puntos, b) mallado y superficie.

a)

b)

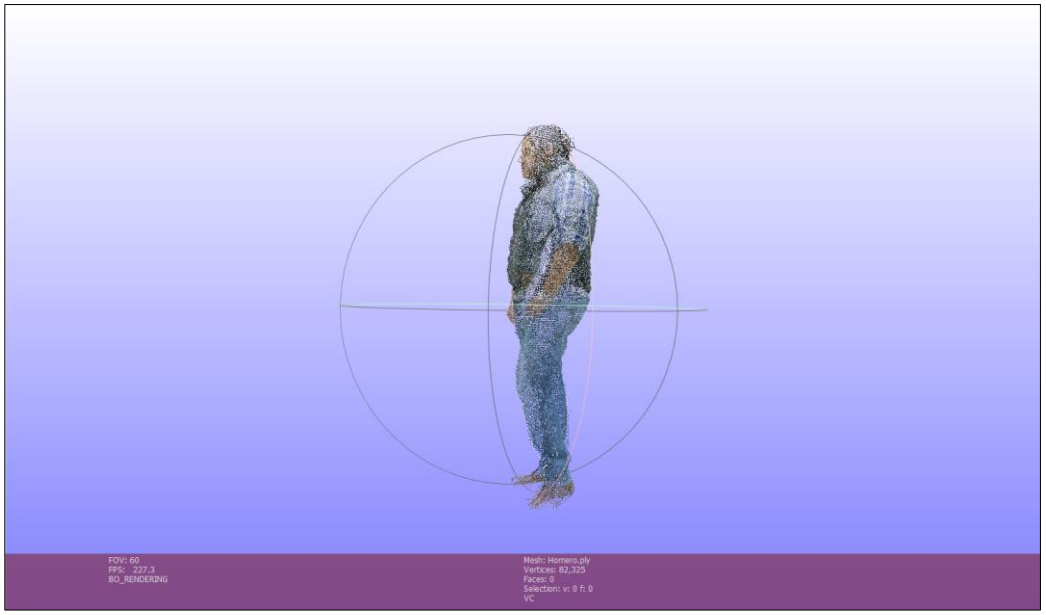
Figura A.2. Modelo de persona 2. a) Nube de puntos, b) mallado y superficie.



a)

b)

Figura A.3. Modelo de persona 3. a) Nube de puntos, b) mallado y superficie.

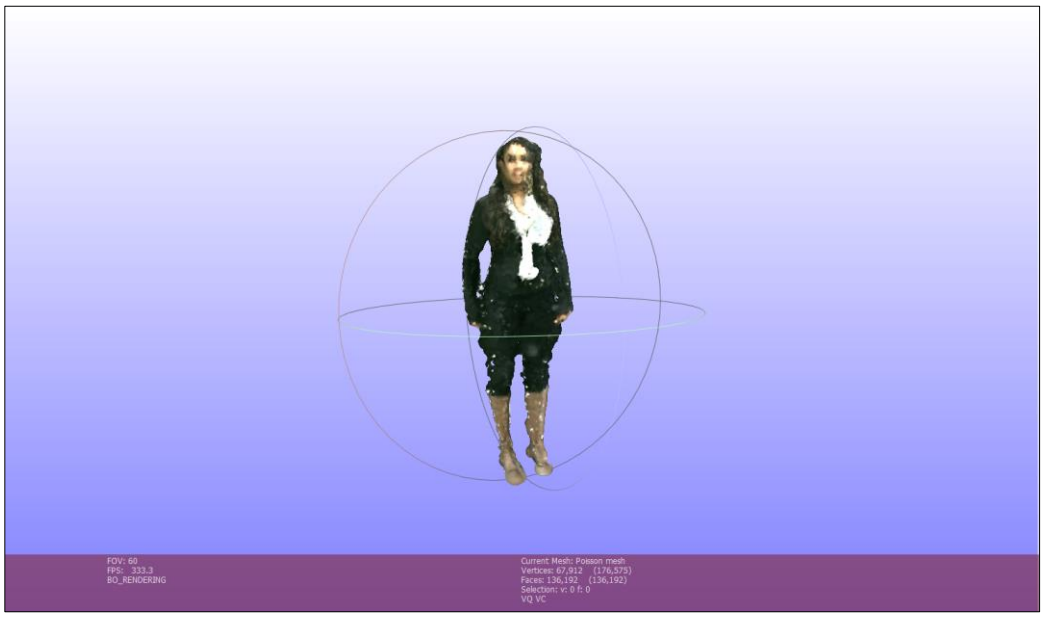


a)

b)

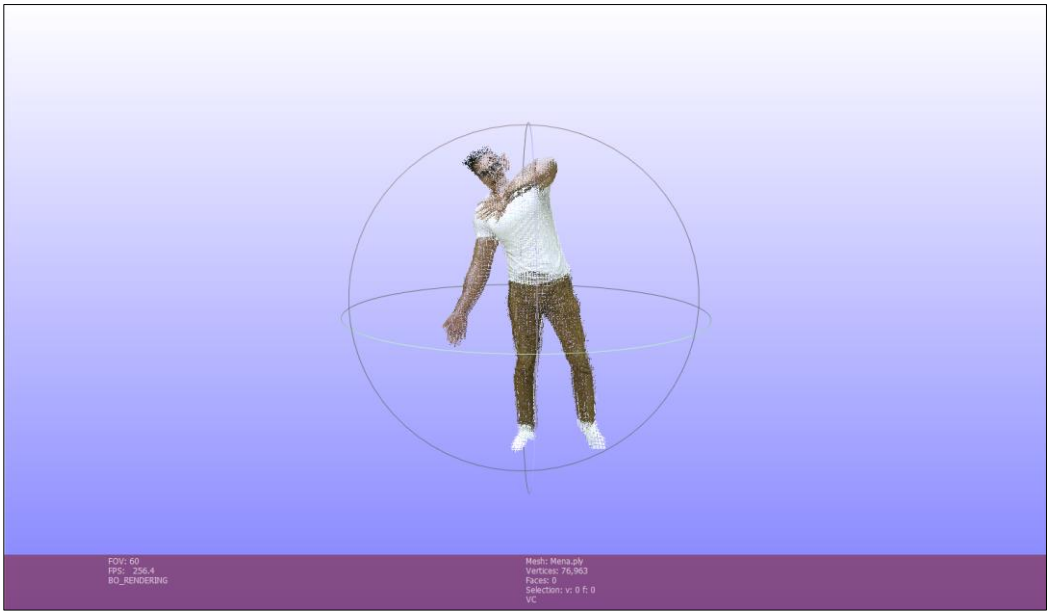
Figura A.4. Modelo de persona 4. a) Nube de puntos, b) mallado y superficie.

a)

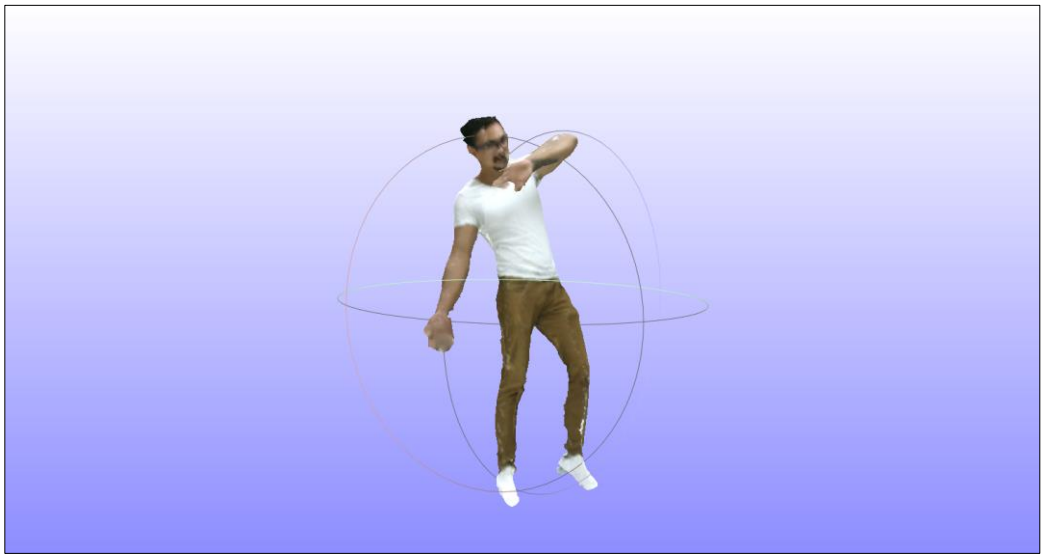


b)

Figura A.5. Modelo de persona 5. a) Nube de puntos, b) mallado y superficie.

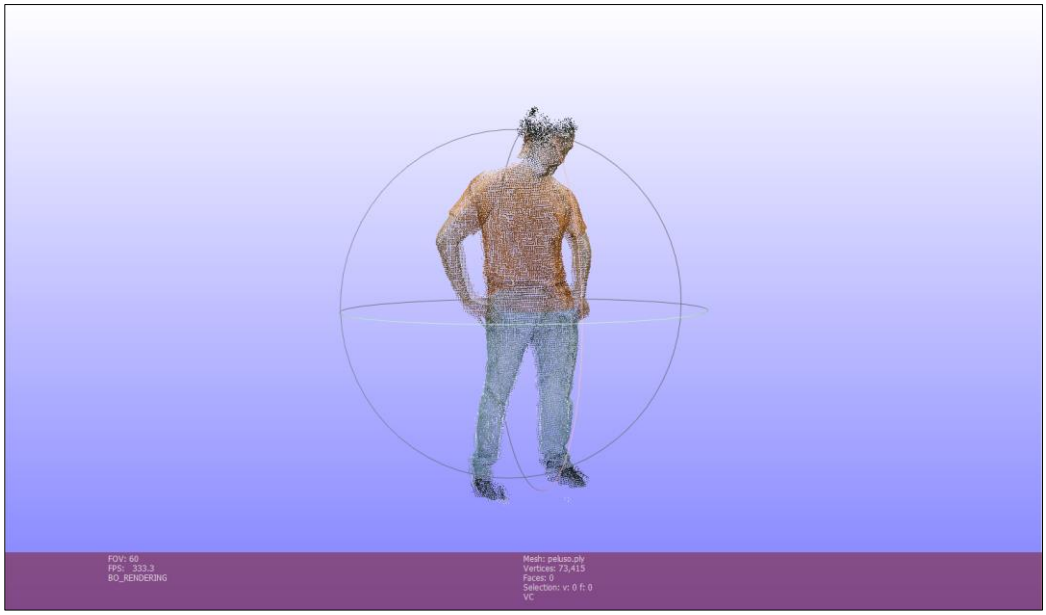


a)

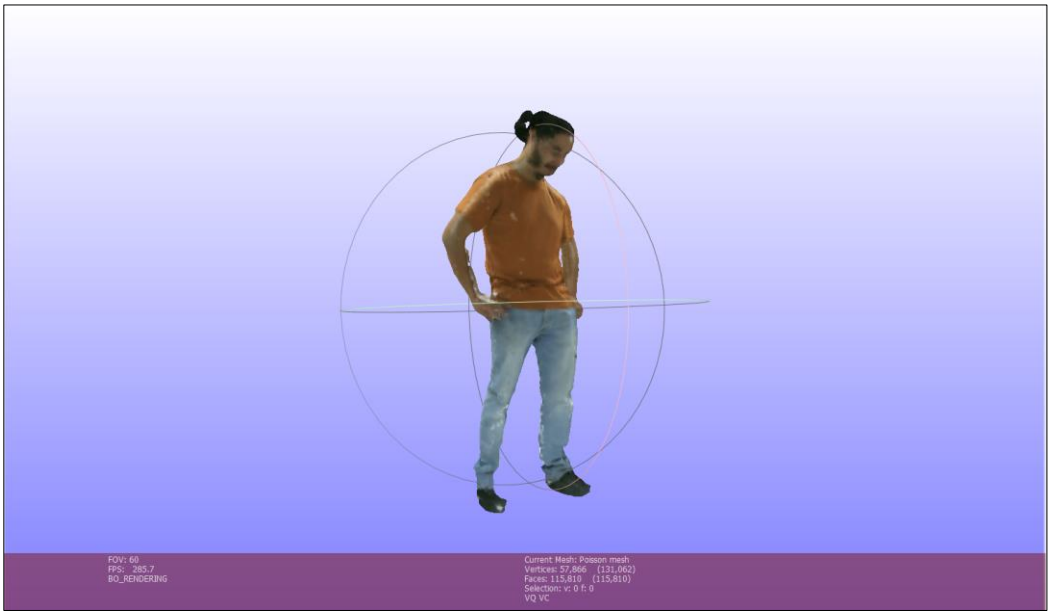


b)

Figura A.6. Modelo de persona 6. a) Nube de puntos, b) mallado y superficie.

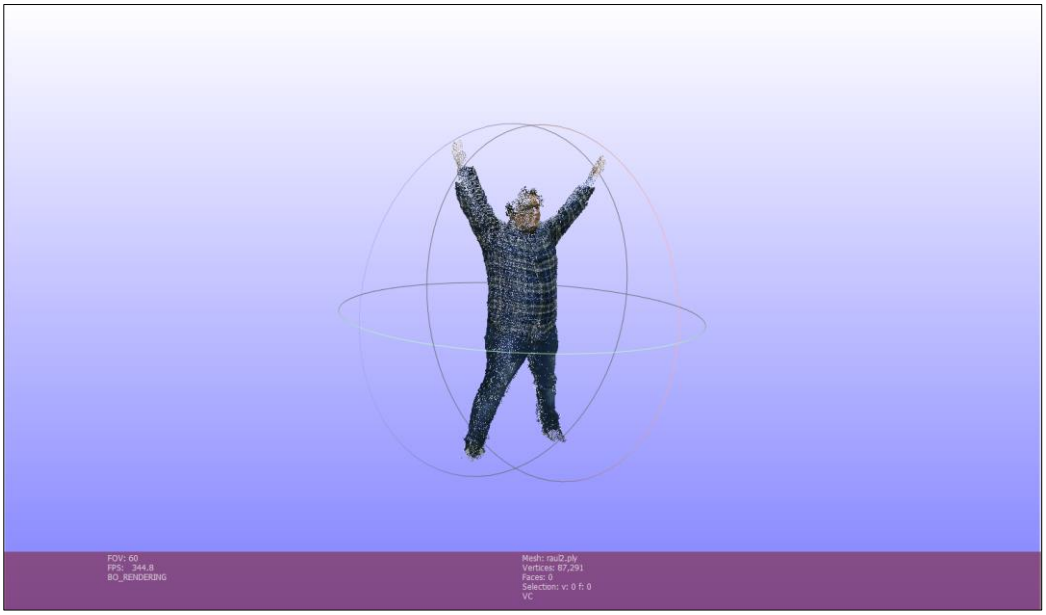


a)



b)

Figura A.7. Modelo de persona 7. a) Nube de puntos, b) mallado y superficie.



a)

b)

Figura A.8. Modelo de persona 8. a) Nube de puntos, b) mallado y superficie.