Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Escáner 3D autónomo con RaspberryPi

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Tizne Mena, Guillem

Tutor: Sánchez López, Miguel

2014 - 2015

Resumen

El proyecto realizado tiene la finalidad de encontrar una solución sencilla y barata al escaneado tridimensional. Como centro de procesamiento hemos utilizado una Raspberry Pi para el control de una línea láser y de un motor stepper, este último va unido a una plataforma giratoria donde se colocarán los objetos para ser escaneados.

Por lo que respecta a la captura de imágenes, usaremos una cámara de la marca Creative, capaz de hacer fotos en HD720p; para el movimiento de la plataforma emplearemos un servomotor con una placa de control, con compatibilidad con un paquete de librerías para su control en Java.

Para el desarrollo del proyecto, se ha optado por el lenguaje de programación Java, ya que se trata de un lenguaje muy extendido con un gran apoyo de la comunidad de desarrolladores y, además, es el principal lenguaje que se enseña en la escuela de informática. Y para la gestión de las librerías utilizaremos Maven por su facilidad y su comodidad a la hora de descargar repositorios de terceros.

Se han realizado numerosos test para determinar la iluminación, la calibración de la cámara y los pasos de motor para tomar las muestras necesarias al hacer una revolución completa, para así conseguir un escaneo satisfactorio.

Palabras clave: escáner 3d, Raspberry Pi, Java.

Abstract

The project aims to find a simple and inexpensive solution to three-dimensional scanning. As processing center we have used a Raspberry Pi to control a laser line and a stepper motor, which is attached to a rotating platform where the objects to be scanned will be placed.

Regarding the capture of images, we will use a Creative camera, able to take photos in HD720p; for the movement of the platform we will use a servomotor with a control board, with a package of libraries with support for its control in Java.

For the development of the project, we opted for the Java programming language, since it's a widespread language with great support from the developer community, and is also the main language that is taught in the school of computer science. And for the management of the libraries we use Maven thanks to its ease and convenience when it comes to download third-party repositories.

There have been made numerous test to determine the lighting, camera calibration and engine steps to take the necessary samples to make a complete revolution, to get a successful scan.

Key words: 3d scanner, Raspberry Pi, Java.

Tabla de contenidos

Tabla de contenido

1. Introducción

Durante la última década las patentes de impresión y escaneo 3D han sido liberadas, causando un abaratamiento de la producción y venta de estos productos, aun así, muchos de estos siguen teniendo precios desmesurados para su uso personal.

Un gran ejemplo de este florecimiento de la tecnología tridimensional puede ser observado en la plataforma web de financiación en masa, Kickstarter, donde se pueden encontrar proyectos muy prometedores en este campo con un gran rango de precios. Por otra parte, también grandes empresas privadas como Google o Microsoft invierten gran cantidad de recursos en investigación de estas tecnologías, como por ejemplo la famosa cámara Kinect, para la videoconsola Xbox que permitía generar nubes de puntos en tiempo real y analizarlas con su API; o el Project Tango de Google, que permite darles a los dispositivos móviles la habilidad de navegar por un mundo físico tal y como lo hacen los humanos gracias a la percepción espacial y un avanzado procesamiento de imágenes y sensores especiales.

El proyecto tiene la finalidad de encontrar una solución sencilla y barata al escaneado tridimensional. Como centro de procesamiento hemos utilizado una Raspberry Pi para el control de una línea láser y un motor stepper este último va unido a una plataforma giratoria donde se colocarán los objetos para ser escaneados.

1.1. Objetivo

El proyecto nace de la necesidad de estudiar las tecnologías en auge del escaneo y la impresión 3D.

El objetivo principal es conseguir diseñar un producto de bajo coste que sea capaz de realizar un análisis de un objeto tridimensional y, posteriormente, la producción de un archivo capaz de ser exportado a un entorno de desarrollo 3D, como Blender, para su uso.

Este producto debe constar de las siguientes partes:

- Una Raspberry Pi
- Un motor *stepper* con una plataforma giratoria
- Una línea láser

1.2. Justificación del proyecto

Las principales motivaciones que han llevado a seleccionar y realizar este proyecto son el estudio de las tecnologías 3D y desarrollo e investigación de software para la Raspberry Pi.

Tras el auge de los productos basados en tecnologías tridimensionales y su constante aparición en portales de noticias, hicieron que se centrara el foco de atención en este tema. Las amplias posibilidades y múltiples aplicaciones que la visión tridimensional y análisis de imágenes ha traído, ha hecho que se convierta en una disciplina muy atractiva y difícil de ignorar.

Arduino es una plataforma de hardware libre con un microcontrolador diseñada para facilitar el uso de la electrónica. A raíz de esto y con la popularización del “internet de las cosas” se creó la Raspberry Pi, un producto similar a Arduino pero mucho más potente, con más capacidad de procesamiento y capaz de correr un sistema operativo completo de tal forma que podría ser utilizado como un PC.

La combinación de la Raspberry Pi y las tecnologías 3D, ofrecen un gran abanico de posibilidades que han llamado a los desarrolladores a construir proyectos con ellas.

1.3. Estructura de la memoria

A continuación se explicara brevemente en que consiste cada apartado de la memoria.

2 – Tecnologías: Presentación de las diferentes tecnologías empleadas en el desarrollo del proyecto, desde las tecnologías Hardware hasta el software utilizado.

3 – Análisis: Breve análisis y explicación del método empleado para la captura y escaneo de objetos físicos empleando una cámara un láser y una plataforma giratoria con un motor de pasos.

4 – Implementación: Pasos seguidos para el montaje del proyecto, instalación del sistema operativo en la Raspberry Pi y análisis detallado del código de la aplicación y de su hilo de ejecución.

5 – Resultados: Muestra de los resultados obtenidos a partir de la ejecución del escáner de un objeto de prueba y análisis de los resultados.

6 – Conclusión: Presentación de las conclusiones obtenidas a partir de la realización del proyecto.

7 – Bibliografía: Apartado dedicado a la mención de las fuentes bibliográficas utilizadas para la producción de este proyecto.

2. Tecnologías

2.1. Hardware

2.1.1. Raspberry Pi

La Raspberry Pi es un ordenador de bajo coste del tamaño de una tarjeta que puede ser conectado a un monitor o incluso a una televisión y usa un teclado estándar y un ratón. Es capaz de hacer todo lo que esperarías que hiciera un PC de sobremesa.

Fue diseñada por la Raspberry Pi Foundation, una organización dedicada a la caridad cuyo objetivo es avanzar en la educación de adultos y niños en el campo de la informática y las ciencias relacionadas con esta.

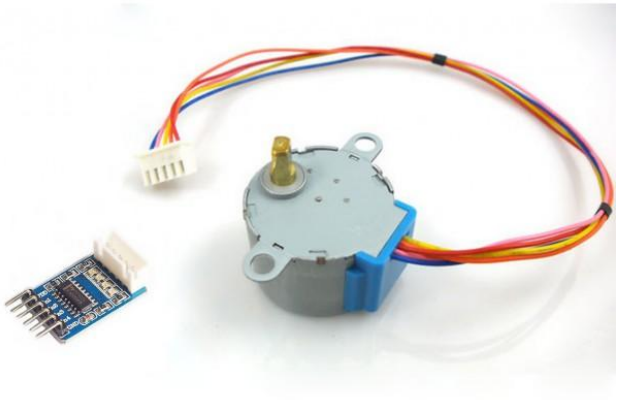
Las primeras ventas comenzaron el 29 de febrero de 2012 y en los seis meses siguientes llegarían a vender 500.000 unidades. [1]

En nuestro proyecto será el núcleo de procesamiento del escáner, que dará las órdenes al motor y a la cámara.

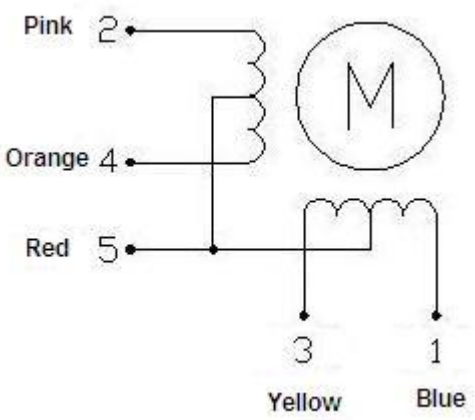
Especificaciones:

Raspberry Pi Modelo B	
SoC	Broadcom BCM2835 (CPU + GPU + DSP + SDRAM + puerto USB)
CPU	ARM 1176JZF-S a 700 MHz (familia ARM11)
Juego de instrucciones	RISC de 32 bits
GPU	Broadcom VideoCore IV, OpenGL ES 2.0, MPEG-2 y VC-1 (con licencia), 1080p30 H.264/MPEG-4 AVC
Memoria (SDRAM)	512 MiB (compartidos con la GPU)
Puertos USB 2.0	2 (vía hub USB integrado)
Entradas de video	Conector MIPI CSI que permite instalar un módulo de cámara desarrollado por la RPF
Salida de video	Conector RCA (PAL y NTSC), HDMI (rev1.3 y 1.4), Interfaz DSI para panel LCD
Salida de audio	Conector de 3.5 mm, HDMI
Almacenamiento integrado	SD / MMC / ranura para SDIO
Conectividad de Red	10/100 Ethernet (RJ-45) via hub USB
Periféricos de bajo nivel	8 x GPIO, SPI, I^2C, UART
Consumo energético	700 mA, (3.5 W)
Fuente de alimentación	5 V vía Micro USB o GPIO header
Dimensiones	85.60mm × 53.98mm (3.370 × 2.125 inch)
Sistemas operativos soportados	GNU/Linux: Debian (Raspbian), Fedora (Pidora), Arch Linux (Arch Linux ARM), Slackware Linux. RISC OS2

2.1.2. Motor 28BYJ-48

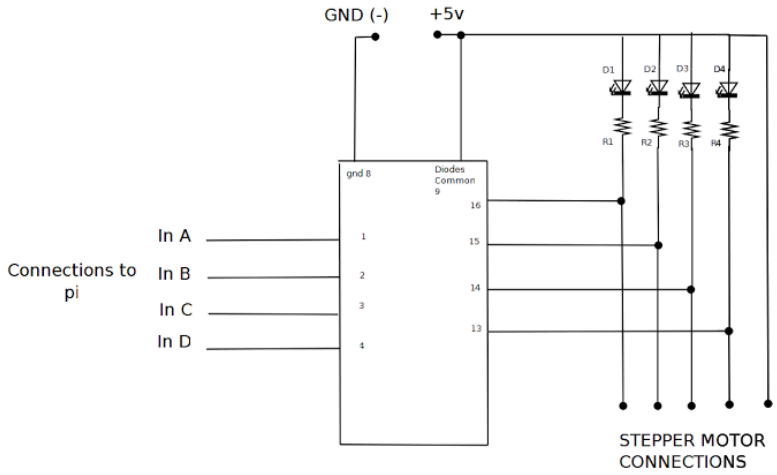


Un motor de pasos es un dispositivo electromagnético que convierte los pulsos eléctricos en movimientos mecánicos. La ventaja más significativa de un motor de pasos es la habilidad de ser precisamente controlado sin necesidad de *feedback*, este tipo de control elimina la necesidad de caros sensores tales como codificadores ópticos, el seguimiento de tu posición se mantiene contando los pulsos introducidos. [2]



En la imagen superior podemos observar las entradas del motor que controlan los electroimanes que hacen rotar el eje del motor (1-4). Y la entrada (5) que proporciona el voltaje necesario al electroimán para hacer rotar el eje del motor.

El motor es controlado por la placa ULN2003.



El motor se usará para hacer rotar la plataforma donde depositaremos los objetos para ser escaneados.

Especificaciones:

Stepper Motor 28BYJ-48 con placa ULN2003	
Voltaje	5V
Diámetro Motor	28mm
Ángulo de pasos	5,625º x 1/64
Relación de reducción	2038 aprox.
Fases	4 fases (LEDs de fase (D1, D2, D3, D4)
Tamaño de la placa	20mm x 9mm

2.1.3. Cámara Sync HD modelo VF0770

Desde la revolución de las videoconferencias por la red, Skype y otros programas similares, las webcam han sido un producto muy popular y extendido. La cámara que nosotros usaremos es la Sync HD de Creative, es una webcam USB de bajo coste capaz de grabar videos en alta resolución.

Se encargará de tomar una instantánea después de cada avance del motor para, posteriormente, ser analizada.

Especificaciones:

Cámara Sync HD modelo VF0770	
USB Support	USB 2.0
Sensor	HD 720p (1280 x 720)
Longitud del cable	1.5m
Resolución de video	1280 x 720
Resolución de imagen	3.7 megapixels
Frame Rate	30fps 720p

2.1.4. Láser 5mw Rojo

Línea láser 5mw con cuerpo de metal de 12mmx35mm con un funcionamiento de entre 3 y 6 voltios.

El objetivo del láser es marcar la silueta del objeto, para luego poder transformar los puntos más brillantes de la imagen en coordenadas xyz.

2.2. Software

2.2.1. Raspbian

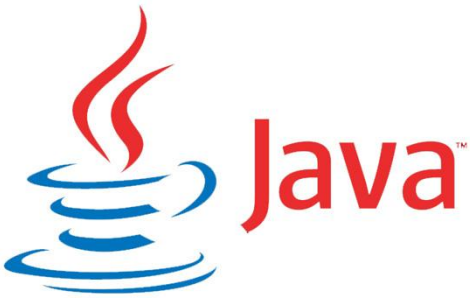
Raspbian es un sistema operativo basado en el famoso Debian y optimizado para el hardware de la Raspberry Pi, es totalmente gratuito. Raspbian aporta todo lo necesario para empezar a utilizar la Raspberry Pi, más de 35.000 paquetes y software educacional ya pre-compilado (Python, Scratch, Sonic Pi, Java, Mathematica, etc.).

Concretamente hemos usado la versión NOOBS ofrecida en el portal de la Raspberry Pi que hacen que la instalación y configuración del sistema sean fáciles e intuitivas. [1]

Nos hemos decidido por el Raspbian ya que nos aportaba todo lo que necesitábamos ya preinstalado en el sistema operativo, como servidor SSH el Java, además de estar basado en el extendido sistema de Linux haciendo que cualquier necesidad pueda ser resuelta gracias al soporte que da la comunidad y los desarrolladores al proyecto.

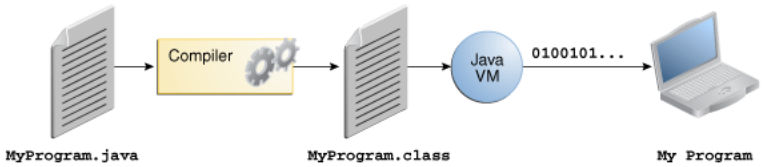
2.2.2. Java

Java es tanto un lenguaje de programación y una plataforma:



Java es un lenguaje de programación de alto nivel que es orientado a objetos, neutral a cualquier arquitectura, portable, distribuido, da soporte a aplicaciones multi-hilo y muy simple.

En java todo el código fuente es escrito en texto simple en archivos con la extensión “.java”. Estos archivos de código fuente son compilados en “.class” por el compilador “javac”. Un “.class” no contiene condigo nativo para el procesador, contiene “bytecodes” el lenguaje de Java Virtual Machine (Java VM). La herramienta para ejecutar aplicaciones Java corre una instancia de Java Virtual Machine.

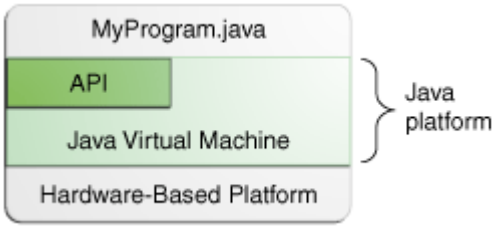


Como la Java VM está disponible en múltiples sistemas operativos, el mismo archivo ".class" es capaz de ser ejecutado en Microsoft Windows, Solaris OS, Linux, o Mac OS.

La plataforma de Java es el hardware o entorno software que permite a un programa ejecutarse. A diferencia de la mayoría de otras plataformas esta es una plataforma software que se ejecuta encima de otras basadas en hardware.

Está divida en dos componentes en la Java VM y en la Java Applicaion Programmin Interface comúnmente conocida como API. Ya se ha descrito anteriormente la Java VM, a continuación explicaremos en que consiste la API.

La API es una gran colección de componentes software que aportan varias útiles herramientas. Estas herramientas están agrupadas en librerías de la misma temática (cadenas de texto, tratamiento de archivos, funciones matemáticas, etc.), estas librerías se les conoce en Java como paquetes.



La plataforma de Java al ser totalmente independiente del hardware, su ejecución algo más lenta que el código nativo. Aunque en los avances que se realizan día a día en el compilador y la máquina virtual hacen que cada vez la diferencia de rendimiento sea más pequeña, sin amenazar la portabilidad de esta. [3]

Aunque para la Raspberry Pi el lenguaje Java no es el lenguaje más extendido de programación y no dispone de tantas librerías como otros. Hemos decidido usarlo por el reto que suponía crear un escáner 3D con esta tecnología. Otro de los motivos que llevaron a seleccionar Java fue que es el lenguaje que más soltura y experiencia tenemos ya que es el principal lenguaje de programación que se imparte en la escuela de informática.

2.2.3. Maven

Maven es una herramienta automática para la creación principalmente de proyectos Java. La palabra Maven quiere decir acumulador de conocimiento en yidis. Maven implementa dos aspectos de la creación de software:

Primero describe como se debe construir el software.

Segundo describe sus dependencias de otros proyectos y librerías.

Un archivo XML denominado como POM (project object model) describe como el software será construido y todos los módulos y librerías externas necesarios. Maven descarga dinámicamente todas las librerías descritas en el POM de los repositorios como puede ser Maven 2 Central Repository y las almacena en la carpeta de librerías de Maven dentro del proyecto.

Esto ofrece una gran facilidad a la hora de actualizar las librerías, cambiarlas y añadir nuevas. Por esta última razón decidimos usar esta tecnología en el proyecto para poder obtener todas las librerías necesarias para el control de los componentes de la Raspberry Pi como las salidas GPIO y también los drivers de la cámara.

2.2.4. Eclipse IDE

Eclipse es una plataforma de desarrollo, diseñada para ser extendida de forma indefinida a través de *plugins*. Fue concebida desde sus orígenes para convertirse en una plataforma de integración de herramientas de desarrollo. No tiene en mente un lenguaje específico, sino que es un IDE genérico, aunque goza de mucha popularidad entre la comunidad de desarrolladores del lenguaje Java.

Proporciona herramientas para la gestión de espacios de trabajo, escribir, desplegar, ejecutar y depurar aplicaciones. Según las necesidades del desarrollador hay diferentes descargas del entorno Eclipse, como Eclipse Java EE para desarrollo de aplicaciones Web en Java, Eclipse C/C++, entre otros. Todos ellos son el programa eclipse básico con *plugins* predeterminados para una tarea y lenguaje concretos.

Las principales características son:

Perspectivas, editores y vistas: en Eclipse el concepto de trabajo está basado en las perspectivas, que no es otra cosa que una preconfiguración de ventanas y editores, relacionadas entre sí, y que nos permiten trabajar en un determinado entorno de trabajo de forma óptima.

Gestión de proyectos: el desarrollo sobre Eclipse se basa en los proyectos, que son el conjunto de recursos relacionados entre sí, como puede ser el código fuente, documentación, ficheros configuración, árbol de directorios,… El IDE nos proporcionará asistentes y ayudas para la creación de proyectos. Por ejemplo, cuando creamos uno, se abre la perspectiva adecuada al tipo de proyecto que estemos creando, con la colección de vistas, editores y ventanas preconfigurada por defecto.

Depurador de código: se incluye un potente depurador, de uso fácil e intuitivo, y que visualmente nos ayuda a mejorar nuestro código. Para ello sólo debemos ejecutar el programa en modo depuración (con un simple botón). De nuevo, tenemos una perspectiva específica para la depuración de código, la

perspectiva depuración, donde se muestra de forma ordenada toda la información necesaria para realizar dicha tarea.

Extensa colección de *plugins*: están disponibles en una gran cantidad, unos publicados por Eclipse, otros por terceros. Al haber sido un estándar de facto durante tanto tiempo (no el único estándar, pero sí uno de ellos), la colección disponible es muy grande. Los hay gratuitos, de pago, bajo distintas licencias, pero casi para cualquier cosa que nos imaginemos tenemos el *plugin* adecuado.

Decidimos emplear Eclipse porque es uno de los entornos de desarrollo de software para Java más completos, que ofrece gran cantidad de facilidades y *plugins* para asistirte en la creación de aplicaciones.

2.2.5. Filezilla

FileZilla es un cliente FTP multiplataforma de código abierto y software libre, licenciado bajo la Licencia MIT. Soporta los protocolos FTP, SFTP y FTP sobre SSL/TLS (FTPS).

Inicialmente fue diseñado para funcionar en Microsoft Windows, pero actualmente es multiplataforma, estando disponible además para otros sistemas operativos, entre ellos GNU/Linux, FreeBSD y Mac OS X.

En el proyecto usamos FileZilla para el traslado de archivos de la memoria de la Raspberry Pi a la terminal donde estemos trabajando, tales como, ejecutables con el código de la aplicación, resultados de un escaneo o imágenes tomadas por la cámara en el caso de haber elegido la opción, etc.

2.2.6. PuTTY

PuTTY es un cliente de SSH y Telnet, desarrollado originalmente por Simon Tatham para el sistema operativo Windows. PuTTY es de código abierto y su código fuente está disponible, actualmente está siendo desarrollado y mantenido por un grupo de voluntarios.

A diferencia de Linux o Mac Os el sistema operativo que hemos usado para el desarrollo del proyecto no tiene cliente SSH nativo así pues usamos este cliente para poder tener el control de nuestro dispositivo Raspberry Pi.

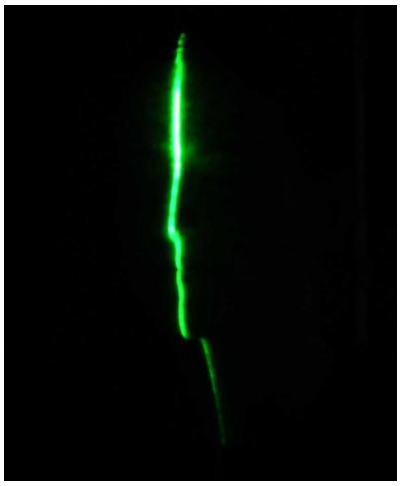
3. Análisis

Un escáner 3D es un dispositivo que analiza el mundo real y recopila información sobre su forma y apariencia. Los datos recolectados pueden ser usados posteriormente para construir un objeto digital tridimensional.

Hay múltiples formas de realizar un escaneado de un objeto cada tecnología tiene sus limitaciones, ventajas y coste. Algunas de las limitaciones más comunes son las superficies brillantes, transparentes o reflectantes que dificultan el trabajo de los dispositivos ópticos y los laser.

Para nuestro proyecto utilizaremos la tecnología de escaneado de luz estructurada con una línea laser.

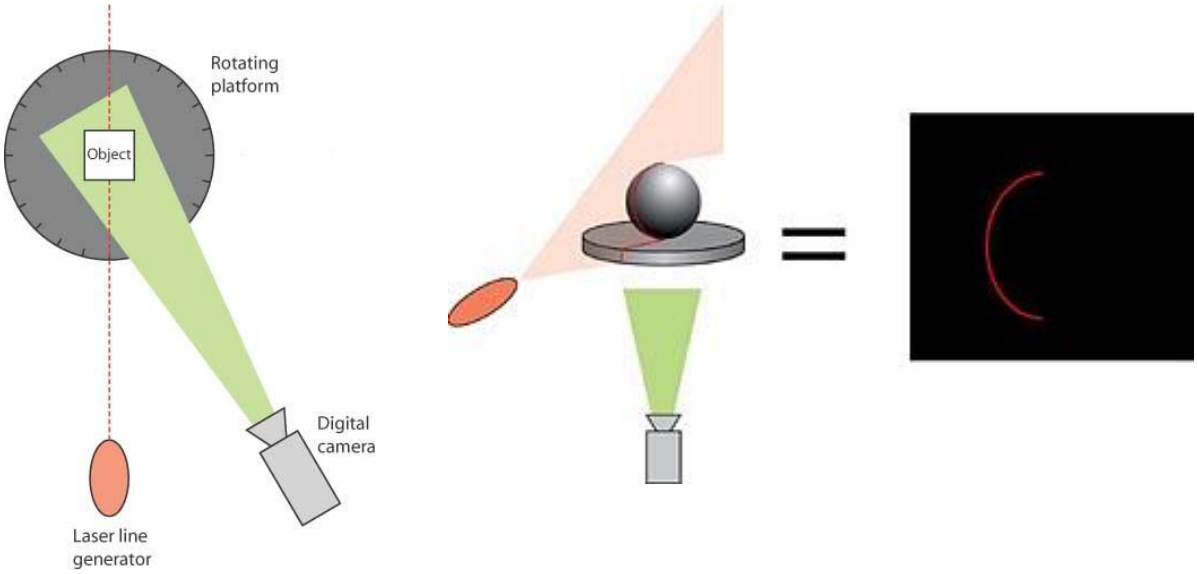
Los escáneres de luz estructurada 3D proyectan un patrón de luz sobre el objeto y analizan la deformación del patrón. El patrón es proyectado sobre el objeto usando un proyector LCD u otra fuente de luz estable como un láser.



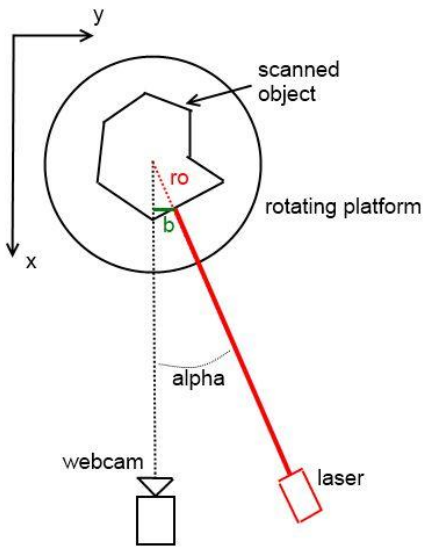
El escaneado de luz estructurada es un área de investigación muy activa en la actualidad. La ventaja del escaneado 3D de luz estructurada frente a otro tipo de escaneados es la velocidad y precisión. En vez de escanear un punto cada vez este tipo de escáner analiza una zona del objeto cada vez así capturando múltiples puntos de la muestra.

Algunos escáneres actuales son capaces de escanear objetos en movimiento con este tipo de tecnologías.

Concretamente nosotros usaremos el análisis de imagen por triangulación donde el basándonos en el ángulo de incisión del láser sobre el objeto que deseamos escanear y respecto a la posición de la cámara, dependiendo de dónde aparezca el láser dentro del campo de visión de la cámara, podemos deducir la posición del láser sobre el objeto en un sistema de coordenadas xyz. [4]



Una vez tomada la imagen del objeto analizaremos la imagen y detectaremos la silueta del láser marcada sobre el objeto, en ese momento deberemos aplicar un algoritmo para calcular las coordenadas cartesianas.



Finalmente se hará la reconstrucción del objeto mediante los puntos transformados. En este caso deberemos producir una nube de puntos, capaz de ser exportada a editores 3D para poder ser tratada y modificada. [5]

4. Implementación

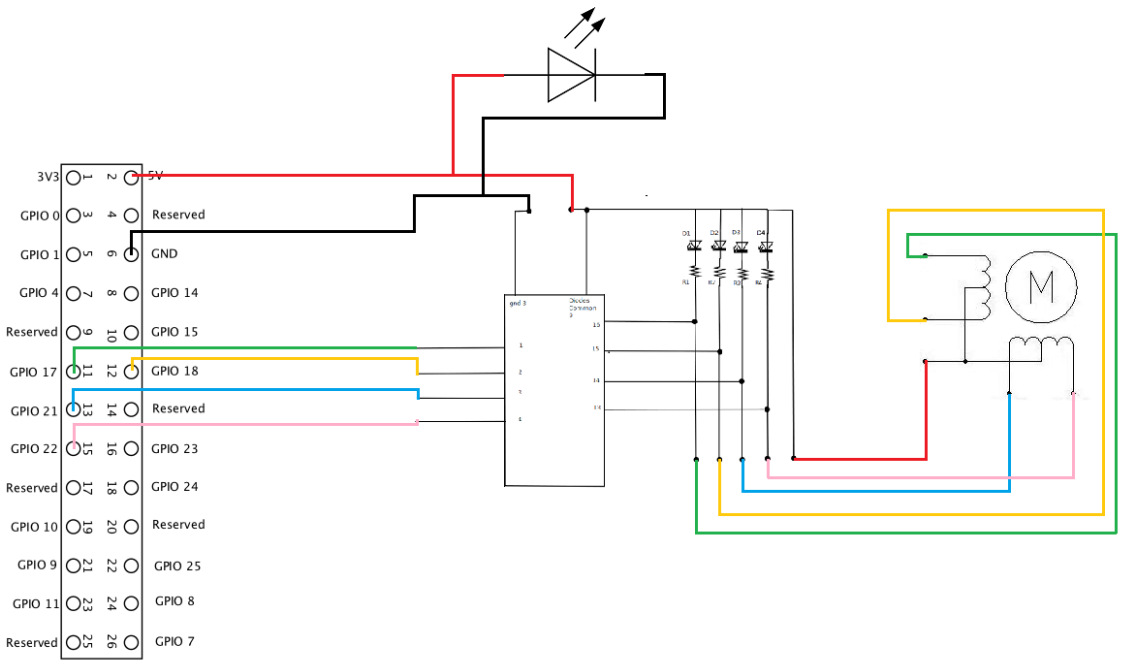
En este apartado procederemos a la explicación del montaje y desarrollo del software que hace funcionar el escáner.

4.1. Hardware

Para la parte del hardware, como hemos descrito en el apartado de tecnologías utilizaremos como centro de procesamiento una placa Raspberry Pi, como cámara, una *webcam* USB de la compañía Creative modelo VF0770, para el movimiento de la plataforma emplearemos un motor paso a paso 28BYJ-48 también utilizaremos una línea laser de 5mw color rojo. Utilizaremos una *breadboard* para hacer las conexiones sin soldaduras.

Para el montaje de la estructura, utilizaremos el juego de montaje LEGO, ya que ofrece una gran flexibilidad y robustez a la hora de hacer el montaje.

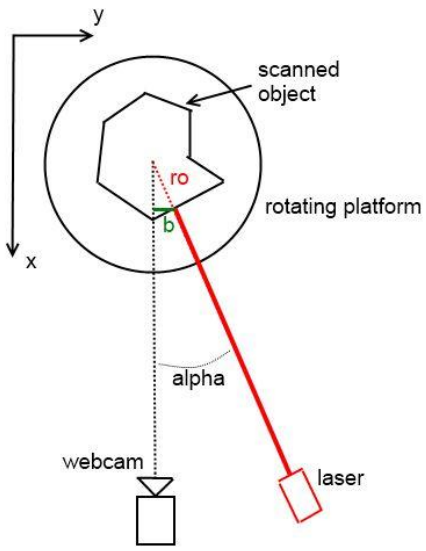
Esquema de conexiones del escáner:



Hemos conectado tal y como esta descrito en el esquema de la ilustración anterior los componentes eléctricos. Utilizando la ya mencionada *breadboard* para no tener que hacer soldaduras con los cables.

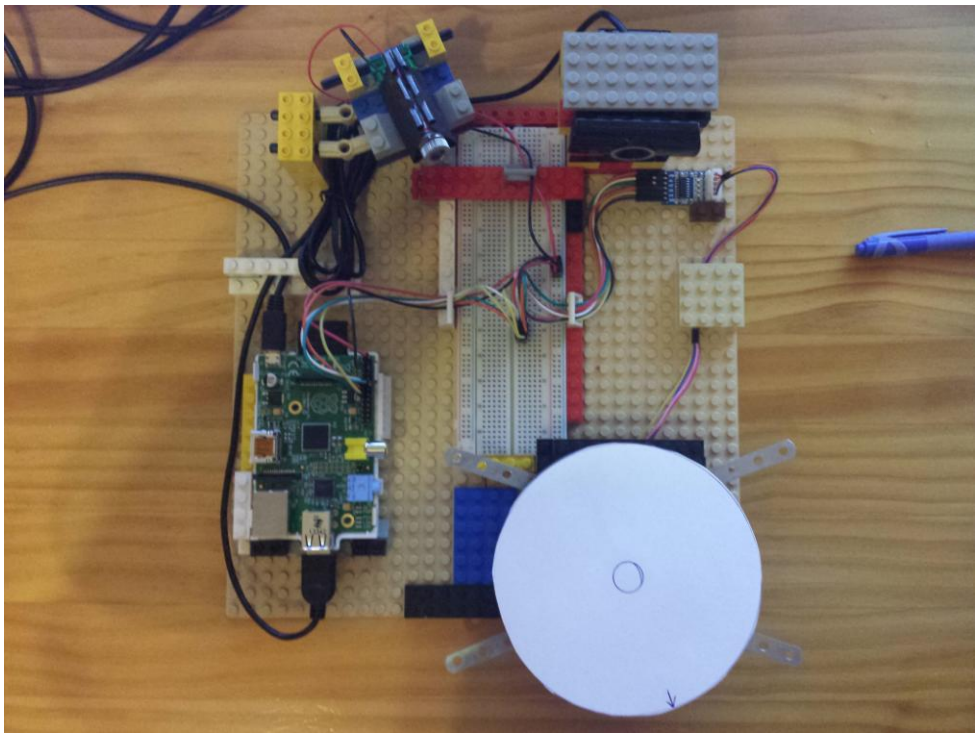
Seguidamente con las piezas de lego construiremos un soporte para el motor y con adhesivo pegaremos una plataforma circular, en nuestro caso hemos utilizado un CD, que ha sido recubierto de cartulina para que no produzca reflejos al impactar el láser.

Construiremos un soporte para la webcam perpendicular al soporte del motor y la conectaremos a una de las entradas USB de la Raspberry Pi, y finalmente crearemos otro soporte más para el láser. Se ha de tener en cuenta que el láser ha de estar apuntando al centro de rotación de la plataforma, para que los cálculos de los algoritmos de posicionamiento y detección del láser se ejecuten correctamente.



Tal y como se observa en la imagen que ya hemos presentado anteriormente. La webcam y el láser deben estar apuntando los dos al centro de la plataforma con la máxima precisión posible.

Deberemos tomar nota del ángulo alfa que se observa en la imagen para cálculos en el algoritmo. Este ángulo depende de la inclinación que le hayamos dado al laser. Se recomiendo utilizar un ángulo de unos 30º, nosotros en este proyecto hemos configurado el láser para que incida con 26º. [6]



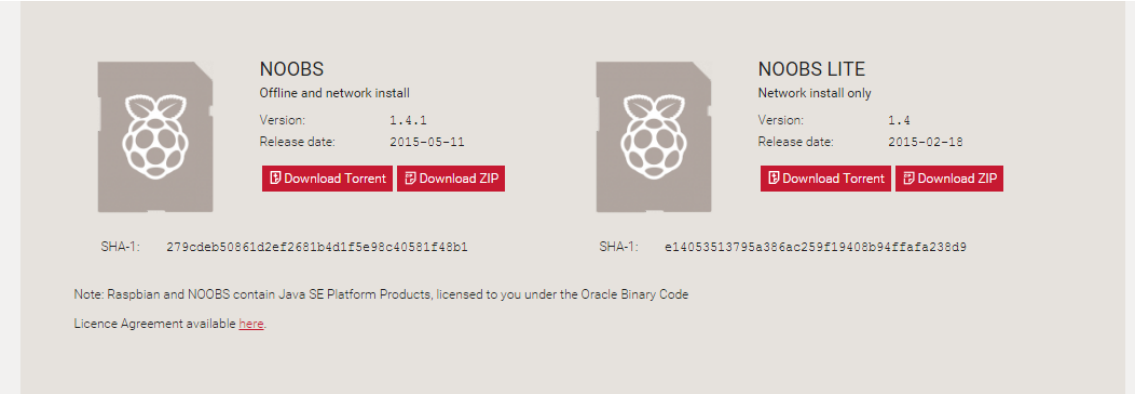
Esta ilustración muestra el resulta final del montaje de los componentes hardware se puede observar la perpendicularidad de la cámara a la plataforma y el ángulo de incisión del láser.

4.2. Software

4.2.1. Raspbian

El sistema operativo para la Raspberry Pi utilizado en este proyecto ha sido Raspbian, concretamente hemos usado la versión NOOBS ofrecida en el portal de la Raspberry Pi que hacen que la instalación y configuración del sistema sean fáciles e intuitivas.

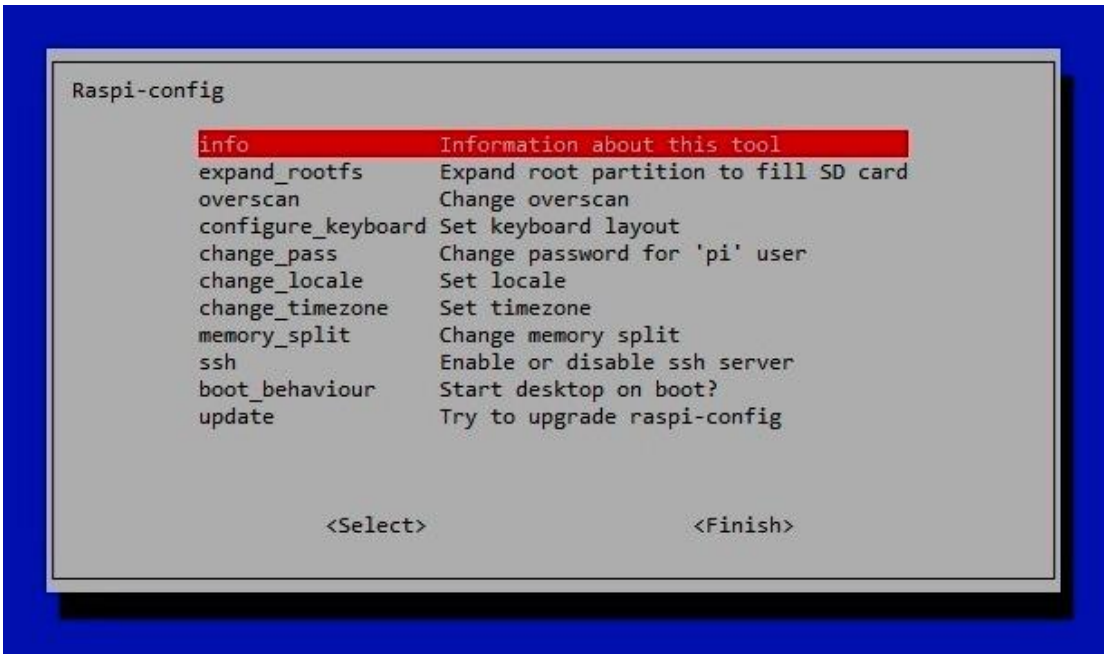
Descargaremos el archivo ".zip" de la página web. Formateamos la tarjeta SD que usaremos como memoria de nuestra Raspberry Pi. Seguidamente extraeremos el contenido de los archivos descargados dentro del directorio raíz de la tarjeta SD.



Una vez completado el proceso podemos ya introducir la tarjeta en la ranura de la Raspberry Pi.

Para el primer *boot* del sistema operativo necesitaremos conectar la Raspberry Pi a un monitor además necesitara un ratón un teclado y conexión a internet a través de un cable Ethernet.

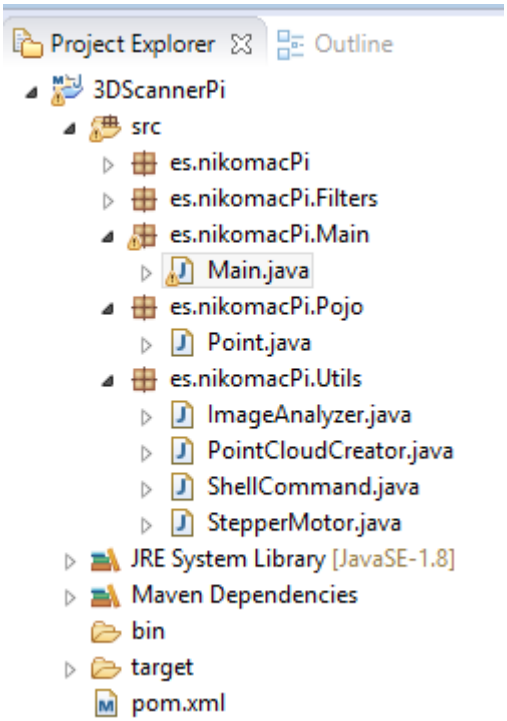
Una vez completados los pasos anteriores ya podemos conectar la Raspberry Pi, que comenzara el proceso de instalación del sistema operativo. Cuando se haya completado el proceso aparecerá el panel de configuración (raspi-config). Aquí podremos introducir todos los parámetros necesarios para el correcto funcionamiento del dispositivo, encender el servicio SSH y configurar los usuarios y contraseñas para la conexión remota con el PuTTY y FileZilla. [1]



4.2.2. Código fuente Java

Como código fuente de la aplicación hemos utilizado Java por la facilidad de compilar y exportar los ejecutables a otros dispositivos, ya que compilaremos en Windows 8.1 y ejecutaremos el código en Raspbian.

Proyecto en eclipse:



Podemos observar en la ilustración los diferentes paquetes del proyecto. En la carpeta de “src”, tenemos el código fuente de la aplicación dividido en diferentes paquetes para un mejor mantenimiento de la aplicación.

En la carpeta de “Main” se encuentra el código principal de la aplicación que ejecuta el hilo principal, en la carpeta Pojo encontramos la clase “Point.java” que describe un simple punto de dos coordenadas (x y). Y en la carpeta “Utils” tenemos todas las clases que nos sirven para analizar y encontrar la raya laser en la imagen (“ImageAnalyzer.java”), para crear una nube de puntos a partir del resultado del análisis de la imagen (“PointCloudCreator.java”) y finalmente tenemos la clase que utilizaremos para el control del motor (“StepperMotor.java”).

La clase principal que lleva el hilo de ejecución es la clase main:

```
public class Main {

	static public int WIDTH  = 960;
	static public int HEIGHT = 720;
	// 1 es la maxima precision con 2 generara la mitad de puntos...
	static public int PRECISION = 5;
	// Paradas del motor si aumentamos el numero de paradas la precision
	// de la imagen sera mayor
	// Ya que se realizaran mas instantaneas.
	static private double STOPS = 120;
	static private double ALPHA = 26;

	static {
		Webcam.setDriver(new V4l4jDriver());
	}
```

En esta parte del código se declaran las variables de configuración necesarias para el funcionamiento del escáner, declaramos por orden la anchura de la imagen y su altura, la precisión del escáner donde a mas alto sea el número menor precisión dispone el escáner, ya que solo cogeremos un pixel para analizar de cada cuantos diga la precisión, en este ejemplo uno de cada 5. La variable STOPS determina las paradas que hará el escáner y también está relacionada con la precisión, si hacemos más paradas tendrá más puntos de muestra.

```
public static void main(String[] args) throws IOException,
InterruptedException {

	PointCloudCreator pcc = new PointCloudCreator(ALPHA);
	double fi = 0;
	int step = (int) Math.round(2038/STOPS);

	Webcam webcam = Webcam.getDefault();
	System.out.println("USING WEBCAM: " + webcam.getName());
	webcam.setViewSize(new Dimension(WIDTH,HEIGHT));
	webcam.open();
```

En esta ilustración podemos observar que se inicializa la clase que genera las nubes de puntos, se declara el ángulo de rotación acumulado de la plataforma y también el paso del motor. Inicializamos la cámara y le asignamos las dimensiones declaradas anteriormente.

```
String response = "";
do{
	System.out.println("TAKING CALIBRATION PICTURE:");
	BufferedImage img = webcam.getImage();

	ArrayList<Point> briPixelList = ImageAnalyzer.getBrightestPoint(img);

	//Pintar en la imagen de test los puntos detectados del larser por la
	//camara
	for(int i=0;i<img.getHeight();i++){
		img.setRGB(img.getWidth()/2, i, (new Color(0,0,255).getRGB()));
		img.setRGB(img.getWidth()/2-1, i, (new
		Color(0,0,255).getRGB()));
	}

	for(Point point: briPixelList){
		img.setRGB((point.getX()+(img.getWidth()/2)), point.getY(), (new
		Color(0,255,0).getRGB()));
	}
	saveImage(img,"imgCalib");
	System.out.println("PICTURE Taken! name: imgCalib.png");

	System.out.println("Is it alright? YES/NO");
	Scanner in = new Scanner(System.in);
	response = in.nextLine().trim();

}while(!(response.toLowerCase().equals("yes")));

System.out.println("Save images?");
Scanner in = new Scanner(System.in);
response = in.nextLine().trim();
```

En esta muestra de código ejecutamos la calibración por si la cámara se ha desplazado y es necesario moverla. Hacemos una foto y pintamos en el centro una línea azul para comprobar si está correctamente alineada con la plataforma. Seguidamente pintamos en la foto los puntos donde se ha detectado el láser para comprobar si la iluminación de la sala está entorpeciendo el trabajo. A continuación se guarda la imagen y con el FileZilla podemos descargarla para comprobar que todo está correcto y así darle la orden de continuar el escaneo.

Finalmente se le pregunta al usuario si desea guardar cada imagen que se fuera a generar cuando se analice el objeto.

```
StepperMotor sm = new StepperMotor();
sm.initialize();
System.out.println("   MOTOR TOTAL STOPS: " + STOPS);
System.out.println("   TAKING PICTURES");
BufferedImage img = null;
for(int i=0 ; i<STOPS; i++){
	System.out.println("   IMAGE Nº: " + i + " ANGLE: " +fi);
	long startTime = System.currentTimeMillis();
	img = webcam.getImage(); // Take picture
	sm.start(step); // Move motor
	if(!response.toLowerCase().equals("no")){
		saveImage(img,"img_"+i);
		System.out.println("   Image nº "+i+" saving!  Time: " +
		((System.currentTimeMillis() - startTime)/1000f) + "s");
	}
	ArrayList<Point> briPixelList = ImageAnalyzer.getBrightestPoint(img);
	// Get points from image
	System.out.println("   Image nº "+i+" analyzing!  Time: " +
	((System.currentTimeMillis() - startTime)/1000f) + "s");
	// Transform from 2D to 3D the points
	pcc.createPointCloud(briPixelList, fi);
	System.out.println("   Image nº "+i+" point cloud!  Time: " +
	((System.currentTimeMillis() - startTime)/1000f) + "s");
	fi+=(360/STOPS);
	long finishTime = System.currentTimeMillis()-startTime;
	if(finishTime < 1000 ){
		try {
			Thread.sleep(1000-finishTime);
		}catch (InterruptedException e) {}
	}
	System.out.println("   Image nº "+i+" done!  Time: " +
	((System.currentTimeMillis() - startTime)/1000f) + "s");

}
sm.stop();
webcam.close();
System.out.println("   WRITTING RESULTS");
writeResult(pcc);
```

Esta parte del código es la parte principal de la aplicación donde ocurre realmente el proceso del escaneado.

Inicializamos el motor de pasos y entramos en el bucle que hará las paradas que le hayamos indicado en los parámetros iniciales. Capturamos el tiempo en una variable porque si el proceso de capturar y analizar la imagen es demasiado rápido al motor no le da tiempo a hacer el siguiente paso y ocurre un fallo en los cálculos. Seguidamente tomamos la imagen de la webcam y mientras hacemos los cálculos de captura del punto más brillante y luego la transformación a xyz el motor ira haciendo el movimiento. Si al final de la ejecución de los cálculos no ha pasado más de un segundo el programa esperara hasta completar un segundo que es más o menos el tiempo que le cuesta hacer el paso. Y aumentamos el ángulo de rotación fi que se usa en el cálculo xyz. [7]

Finalmente apagamos el motor, cerramos la webcam y escribimos los resultados obtenidos.

```
private static void writeResult(PointCloudCreator pcc){
	try {
		File statText = new File("resultTest.ply");
		FileOutputStream is = new FileOutputStream(statText);
		OutputStreamWriter osw = new OutputStreamWriter(is);
		BufferedWriter w = new BufferedWriter(osw);
		w.write("ply");
		w.newLine();
		w.write("format ascii 1.0");
		w.newLine();
			w.write("element vertex "+pcc.getPointCloud().size());
			w.newLine();
			w.write("property float x");
			w.newLine();
			w.write("property float y");
			w.newLine();
			w.write("property float z");
			w.newLine();
			w.write("end_header");

		for(double[] point: pcc.getPointCloud()){
		    w.newLine();
		    w.write( point[0] + " " + point[1] + " " + point[2]);
		 }

		w.close();
	  } catch (IOException e) {
		System.err.println("Problem writing to the file statsTest.txt");
	  }
}
```

Este método que se encuentra en la clase main es el encargado de escribir los resultados en un archivo de formato “Stanfor PLY” que será fácilmente exportable a cualquier editor gráfico 3D el formato de un documento PLY es el siguiente:

```
ply
format ascii 1.0
element vertex "tamaño"
property float x
property float y
property float z
end_header
"x" "y" "z"
```

Donde en “tamaño” se debe introducir el número de puntos que vamos a insertar en el documento sin las comillas, y en “x” “y” “z” introduciremos las coordenadas respectivas de cada punto, también sin comillas.

```
public static ArrayList<Point> getBrightestPoint(BufferedImage image){

	ArrayList<Point> briPixelList = new ArrayList<Point>();
	for(int y=0; y<image.getHeight(); y+=Main.PRECISION){
		float luminance = 0.25f;
		Point briPixel = new Point(0,y);
		for(int x=0; x<image.getWidth(); x++){

			int color = image.getRGB(x, y);
			// extract each color component
			int red   = (color >>> 16) & 0xFF;
			int green = (color >>>  8) & 0xFF;
			int blue  = (color >>>  0) & 0xFF;

			// calc luminance in range 0.0 to 1.0; using SRGB
			// luminance constants
			float cPixel = (red * 0.2126f + green * 0.7152f + blue *
					0.0722f) / 255;

			if (cPixel > luminance){
				luminance = cPixel;
				briPixel = new Point((x-(image.getWidth()/2)),y);
			}
		}
		briPixelList.add(briPixel);
	}
	return briPixelList;
}
```

Este método pertenece a la clase “ImageAnalicer” es el que se encarga de analizar las imágenes que se le pasan como parámetro y crear una lista de la posición de los puntos más brillantes.

Tenemos dos bucles “for” en el método el primero itera sobre el eje Y de la imagen aquí es donde el parámetro precisión entra en escena. Podemos observar que si el valor de la precisión es más alto se harán menos muestreos en la imagen ya que el avance de la variable y será más rápido. Seguidamente marcamos un baremo mínimo de luminiscencia donde si no es superado por ningún pixel de la fila Y no se introducirá ningún pixel.

En el bucle interior iteraremos sobre los pixeles de cada línea Y, obtendremos el color del pixel X lo descompondremos en rojo, verde y azul y analizaremos su luminiscencia para ver si es el pixel más brillante.

Debemos tener en cuenta que para el cálculo de la nube de puntos el origen debe ser el punto central de la imagen y no la derecha así que cuando guardamos los pixeles en la lista que devolveremos transformamos su valor para asignar el origen al centro.

```
ArrayList<double[]> pointCloud;
private static DecimalFormat df2 = new DecimalFormat(".######");

public PointCloudCreator(double alpha) {
	pointCloud = new ArrayList<double[]>();
	this.ALPHA = alpha*Math.PI/180; // Transformar a radiants
}

public void createPointCloud(ArrayList<Point> briPixelList, double fi){

	for(Point point : briPixelList){
		if(point.getX()!=0){
			pointCloud.add(CalXYZ(fi, point.getX(), Main.HEIGHT-
			point.getY()));
		}
	}
}

// alpha: angulo entre la camara y el laser
// x : desplazamiento del pixel muestreado en el vector X del centro.
// ro : distancia del pixel muestreado al centro de rotación.
// fi : rotation of the plataform.
private double[] CalXYZ(double fi, int x, int y) {
	double[] vector = new double[3];

	fi = fi*Math.PI/180; //Transformar a radiants

	double ro = x / Math.sin(ALPHA);
	vector[0] = Double.parseDouble(df2.format(ro * Math.cos(fi)));
	vector[1] = Double.parseDouble(df2.format(ro * Math.sin(fi)));
	vector[2] = Double.parseDouble(df2.format(y));

	return vector;
}
```

Esta es la clase de “PointCloudCreator” donde se transforman las coordenadas xy en xyz a través de la triangulación. Al inicializar la clase transformamos el ángulo del láser “ALPHA” de grados a radianes.

El siguiente método es invocado cada vez que la clase principal “Main” debe insertar los puntos xy en la nube de puntos que guardamos en la lista “pointCloud” de esta clase. El método a su vez invoca al método “CalXYZ” que se encarga de calcular con el ángulo “fi” (desplazamiento de la plataforma desde la posición inicial), el ángulo “ALPHA” y las coordenadas X y Y obtenidas del análisis previo de los pixeles de la imagen, el posicionamiento dentro de un sistema de coordenadas tridimensionales.

Finalmente lo inserta en la lista de la nube de puntos para su posterior escritura en un archivo.

```
public StepperMotor() {

	System.out.println("<--Pi4J--> GPIO Stepper Motor ... STARTED");

	// create gpio controller
	final GpioController gpio = GpioFactory.getInstance();

	// provision gpio pins #00 to #03 as output pins and ensure in LOW
	// state
	final GpioPinDigitalOutput[] pins = {
		gpio.provisionDigitalOutputPin(RaspiPin.GPIO_00, PinState.LOW),
		gpio.provisionDigitalOutputPin(RaspiPin.GPIO_01, PinState.LOW),
		gpio.provisionDigitalOutputPin(RaspiPin.GPIO_02, PinState.LOW),
		gpio.provisionDigitalOutputPin(RaspiPin.GPIO_03, PinState.LOW)
	};

	// this will ensure that the motor is stopped when the program
	// terminates
	gpio.setShutdownOptions(true, PinState.LOW, pins);
	// create motor component
	motor = new GpioStepperMotorComponent(pins);
	// for additional details on stepping techniques
	single_step_sequence = new byte[4];
	single_step_sequence[0] = (byte) 0b0001;
	single_step_sequence[1] = (byte) 0b0010;
	single_step_sequence[2] = (byte) 0b0100;
	single_step_sequence[3] = (byte) 0b1000;

	double_step_sequence = new byte[4];
	double_step_sequence[0] = (byte) 0b0011;
	double_step_sequence[1] = (byte) 0b0110;
	double_step_sequence[2] = (byte) 0b1100;
	double_step_sequence[3] = (byte) 0b1001;

	half_step_sequence = new byte[8];
	half_step_sequence[0] = (byte) 0b0001;
	half_step_sequence[1] = (byte) 0b0011;
	half_step_sequence[2] = (byte) 0b0010;
	half_step_sequence[3] = (byte) 0b0110;
	half_step_sequence[4] = (byte) 0b0100;
	half_step_sequence[5] = (byte) 0b1100;
	half_step_sequence[6] = (byte) 0b1000;
	half_step_sequence[7] = (byte) 0b1001;

}
```

Esta es la clase que se encarga del control del motor. Primero inicializamos las salidas de la Raspberry Pi, los GPIO, y asignamos los pines que vamos a utilizar a una lista, los inicializamos a LOW por precaución si alguno se había quedado en ejecución anteriormente. Creamos un nuevo motor y le asignamos los pines declarados, finalmente implementamos tres listas donde solo utilizaremos la segunda porque al activar los imanes de dos en dos, genera más fuerza de par que la primera y es más suave que la tercera. [8] [9]

```
public void initialize() throws InterruptedException {

	// define stepper parameters before attempting to control motor
	// anything lower than 2 ms does not work for my sample motor using
	// single step sequence
	int steps = 2038;
	motor.setStepsPerRevolution(steps);
	motor.setStepSequence(double_step_sequence);
	motor.setStepInterval(10);

	// There are 32 steps per revolution on my sample motor,
	// and inside is a ~1/64 reduction gear set.
	// Gear reduction is actually:
	// (32/9)/(22/11)x(26/9)x(31/10)=63.683950617
	// This means is that there are really 32*63.683950617 steps per
	// revolution
	// = 2037.88641975 ~ 2038 steps!

	System.out.println("   Motor SETTED "+steps+" STEPS PER REVOLUTION!");

}

public void start(int steps) {
	// motor control : STEPPING FORWARD
	System.out.println("   Motor forward " + steps + " steps");
	motor.step(steps);
}

public void stop() {
	// final stop to ensure no motor activity
	motor.stop();
	System.out.println("<--Pi4J--> GPIO Stepper Motor ... STOPPED");
}
```

Una vez declarado el motor podemos inicializarlo y asignarle los pasos necesarios para hacer una revolución, la secuencia de activación de los imanes y el intervalo entre cada paso.

Finalmente tenemos los métodos de “start” y “stop” uno para hacer avanzar el motor un número determinado de pasos y el otro para para el motor una vez terminado de utilizar.

A continuación mostraremos el documento que pertenece a Maven llamado “pom.xml”:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>3DScannerPi</groupId>
	<artifactId>3DScannerPi</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<build>
		<sourceDirectory>src</sourceDirectory>
		<plugins>
			<plugin>
				<artifactId>maven-compiler-plugin</artifactId>
				<version>3.1</version>
				<configuration>
					<source>1.7</source>
					<target>1.7</target>
				</configuration>
			</plugin>
		</plugins>
	</build>
```

En este documento se declara la versión del proyecto, y también su nombre. Más abajo se declaran los *plugins* que se usaran, en este caso solo usaremos el del compilador de Maven donde se declara la versión de Java que se utilizará.

En el documento inferior se declaran los repositorios y las dependencias que vamos a necesitar en este proyecto. Hubo que declarar dos repositorios adicionales además del que viene por defecto para poder tener acceso a todas las dependencias que necesitábamos.

La primera librería es “slf4j-simple”, se trata de una librería que es un *framework* para llevar los *logs* de algunas aplicaciones. Era necesaria para la siguiente librería, que se trata de la librería de control de la webcam, que permite hacer fotográficas, grabar videos y otras muchas opciones que no hemos llegado a utilizar.

Las tres siguientes dependencias pertenecen a los módulos de control de GPIO, son totalmente necesarias para poder acceder al hardware de bajo nivel y controlar el motor y el láser con ellas.

Finalmente la última dependencia es una librería de filtros de todo tipo para la webcam intentamos usarlos para mejorar la luminosidad y el contraste a la hora de la captura, aunque sin mucho éxito.

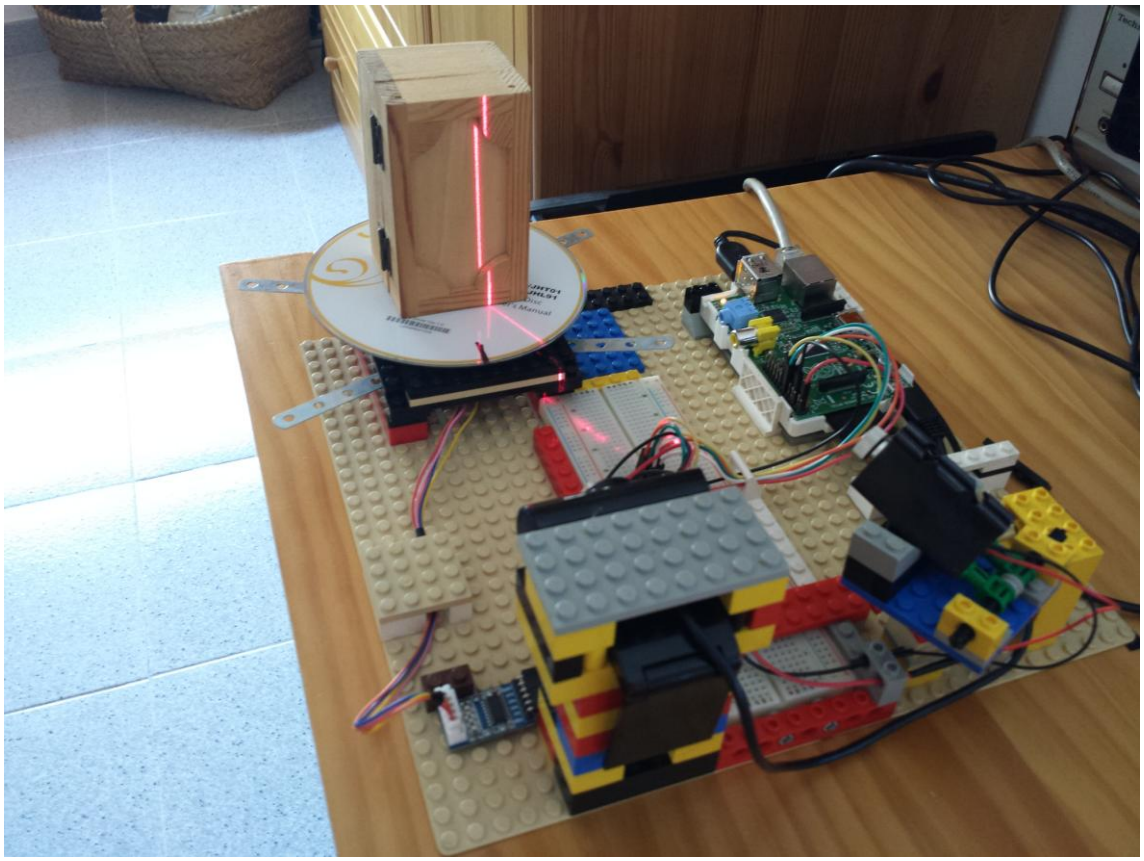
```
	<repositories>
		<repository>
			<id>sonatype</id>
			<name>Sonatype OSS Snapshots Repository</name>
			<url>http://oss.sonatype.org/content/groups/public</url>
		</repository>
		<repository>
			<id>nativelibs4java-repo</id>
			<name>NativeLibs4Java Old Snapshots Repository</name>
			<url>http://nativelibs4java.sourceforge.net/maven</url>
		</repository>
	</repositories>
	<dependencies>
		<dependency>
			<groupId>org.slf4j</groupId>
			<artifactId>slf4j-simple</artifactId>
			<version>1.6.1</version>
		</dependency>
		<dependency>
			<groupId>com.github.sarxos</groupId>
			<artifactId>webcam-capture-driver-v4l4j</artifactId>
			<version>0.3.11-SNAPSHOT</version>
		</dependency>
		<dependency>
			<groupId>com.pi4j</groupId>
			<artifactId>pi4j-core</artifactId>
			<version>1.0</version>
		</dependency>
		<dependency>
			<groupId>com.pi4j</groupId>
			<artifactId>pi4j-gpio-extension</artifactId>
			<version>1.0</version>
		</dependency>
		<dependency>
			<groupId>com.pi4j</groupId>
			<artifactId>pi4j-device</artifactId>
			<version>1.0</version>
		</dependency>
		<dependency>
			<groupId>com.jhlabs</groupId>
			<artifactId>filters</artifactId>
			<version>2.0.235-1</version>
		</dependency>
	</dependencies>
</project>
```

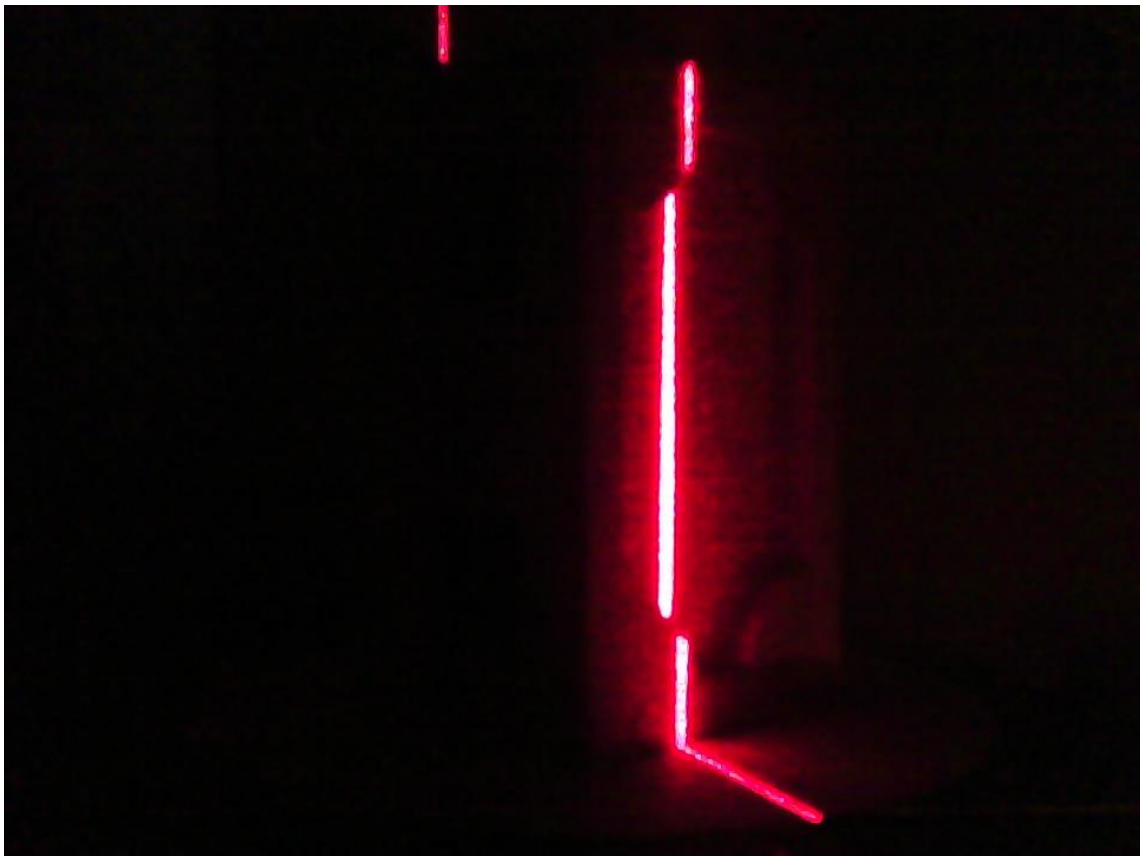
5. Resultados

Tras calibrar el escáner correctamente, el proceso de escaneado es bastante rápido en unos 3 o 4 minutos con una precisión de 4 o 5 termina. Si la precisión la dejamos a 1 a partir del paso 60 la velocidad de procesamiento de las imágenes y de la nube de puntos decae, pasa de costarle 2 segundos cada imagen a más de 10 haciendo que el proceso se ralentice bastante.

También hay que tener en cuenta que si se quiere guardar todas las imágenes que se analizan en el escaneado, guardarlas es el proceso más costoso y puede hacer que un solo escaneado cueste más de 20 minutos. Para que el análisis de la imagen sea efectivo y preciso el proceso de escaneado deberá ser efectuado en una habitación sin luz o que sea muy tenue, también se puede usar un recipiente como una caja para bloquear la luz externa.

Los resultados de escanear una caja como la que se muestra a continuación son los siguientes:

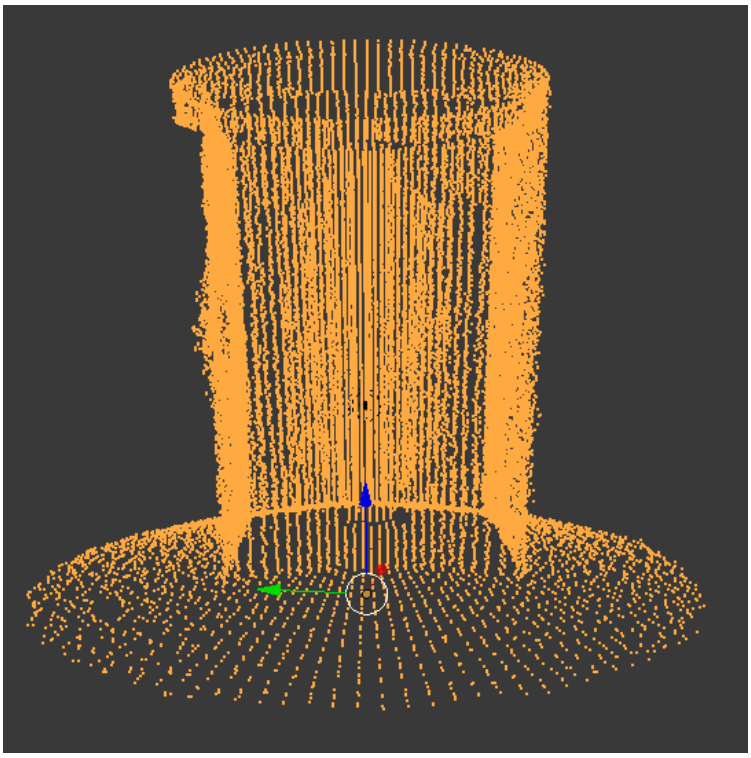


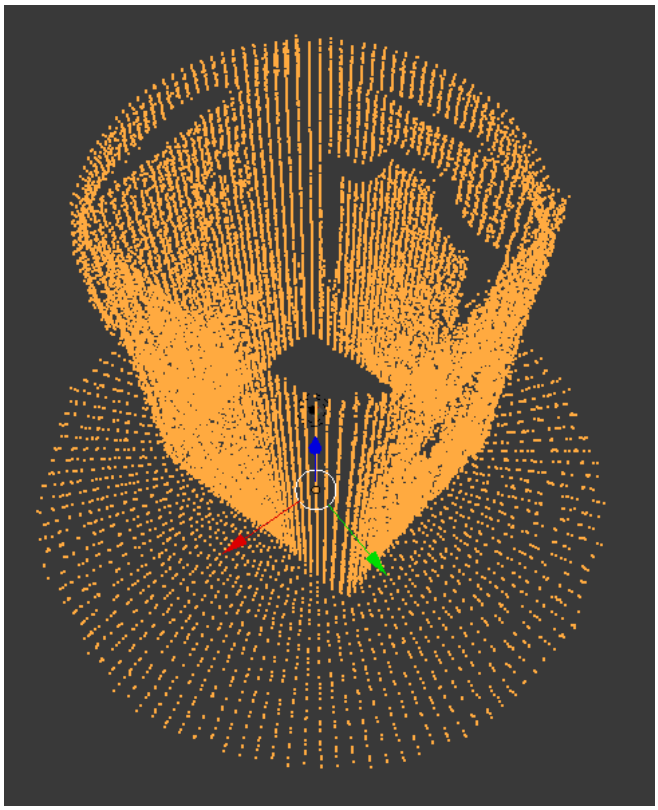


En esta imagen observamos como el láser dibuja la silueta de la caja en un ambiente sin luz.

```
ply
format ascii 1.0
element vertex 40548
property float x
property float y
property float z
end_header
-255.491268 -0.0 720.0
-255.491268 -0.0 718.0
-255.491268 -0.0 716.0
-250.928924 -0.0 714.0
```

En el archivo de formato “ply” que posteriormente importaremos a Blender 3D, podemos observar que se han generado 40548 vértices, y que la altura del primer vértice es 720 que corresponde con la altura de la imagen. [10]





Finalmente importamos el archivo y podemos observar que la caja ha sido escaneada correctamente. Pero que además también ha sido escaneada la plataforma y la parte del fondo del recipiente usado para bloquear la luz, resultando en esa corona circular encima del objeto. Se puede observar cada raya por donde ha pasado el láser desde la parte superior del objeto hasta la inferior pasando por la base.

6. Conclusión

Teniendo en cuenta los objetivos marcados para el proyecto, se ha obtenido un resultado muy satisfactorio. En los resultados hemos podido ver el proceso de escaneado hasta conseguir un objeto digital 3D con un gran parecido al real. En cualquier editor gráfico 3D podríamos importar el resultado y limpiar el ruido generado por la plataforma rotatoria y la pantalla que se usa para bloquear la luz, incluso se podría generar en MeshLab el *mesh* correspondiente al objeto.

Se ha observado que la diferente calidad de los componentes tiene un gran peso en el resultado final. Donde teniendo un láser que generara una raya más fina y marcada o una cámara de alta calidad con mejor sensor, podríamos obtener un resultado más preciso y con menos ruido, teniendo un motor más robusto también se podrían escanear objetos grandes y pesados. Incluso con dos láseres podríamos haber solucionado el problema de los ángulos muertos a la hora de escanear, donde un láser no llega a causa del ángulo de incisión y laser contrario no debería tener problemas.

A pesar de no haber podido completar el servidor web para poder controlar la aplicación sin necesidad de acceder con el PuTTY y el FileZilla no lo considero una razón de peso, ambos programas son muy fáciles de configurar y usar y resultan muy cómodos para el traslado de información y ejecución de la aplicación.

Finalmente, con este proyecto hemos conseguido aprender sobre las tecnologías 3D y aunque simplemente se haya usado un escáner con una línea laser, durante la etapa de investigación hemos podido conocer otras tecnologías como las que utilizan algunas videoconsolas o escáneres que son capaces de generar objetos 3D a partir de tan solo unas pocas fotográficas desde diferentes ángulos. Se ha podido observar también las múltiples aplicaciones que se le podrían dar a una Raspberry Pi siendo un dispositivo tan pequeño y con tanta potencia desde servidores de bajo coste hasta centros de control para drones o robots.

7. Bibliografía

[1] Raspberry Pi: https://www.raspberrypi.org/help/

[2] Youtube: https://www.youtube.com/watch?v=Dc16mKFA7Fo

[3] Documentación de Java: https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

[4] Wikipedia: https://es.wikipedia.org/wiki/Esc%C3%A1ner_3D

[5] Scan – Z: http://www.brucerayne.com/scanz_explain.html

[6] Instructables: http://www.instructables.com/id/Lets-cook-3D-scanner-based-on-Arduino-and-Proces/

[7] Webcam Sarxos: http://webcam-capture.sarxos.pl/

[8] Savage Home Automation: http://www.savagehomeautomation.com/jj-stepper

[9] EclipseSource: http://eclipsesource.com/blogs/2014/05/01/programming-the-pi-with-eclipse-and-java/

[10] Standfor PLY data format: http://paulbourke.net/dataformats/ply/